go-leetcode

发行版本 *v1.0.0*

willshang

2023 年 07 月 30 日

Contents:

CHAPTER 1

0001-0100-Easy

1.1 1. 两数之和 (3)

- 题目

```
给定一个整数数组 nums 和一个目标值 target,
请你在该数组中找出和为目标值的那 两个 整数, 并返回他们的数组下标。
你可以假设每种输入只会对应一个答案。但是, 你不能重复利用这个数组中同样的元素。
示例:给定 nums = [2, 7, 11, 15], target = 9
因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]
```

- 解答思路

```
# 暴力法: 2层循环遍历
func twoSum(nums []int, target int) []int {
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i]+nums[j] == target {
                                return []int{i, j}
                        }
                }
        }
        return []int{}
}
```

(续下页)

(接上页)

```
# 两遍哈希遍历
func twoSum(nums []int, target int) []int {
        m := make(map[int]int,len(nums))
        for k, v := range nums{
                m[v] = k
        }

        for i := 0; i < len(nums); i++{
                b := target - nums[i]
                if num, ok := m[b]; ok && num != i{
                        return []int{i,m[b]}
                }
        }
        return []int{}
}

# 一遍哈希遍历
func twoSum(nums []int, target int) []int {
        m := make(map[int]int, len(nums))
        for i, b := range nums {
                if j, ok := m[target-b]; ok {
                        return []int{j, i}
                }
                m[b] = i
        }
        return nil
}
```

1.2 7. 整数反转 (2)

- 题目

```
给出一个 32 位的有符号整数，你需要将这个整数中每位上的数字进行反转。
示例 1:输入: 123 输出: 321
示例 2:输入: -123 输出: -321
示例 3:输入: 120 输出: 21
注意:假设我们的环境只能存储得下 32 位的有符号整数，则其数值范围为 [-2^31, 2^31 - 1]。
请根据这个假设，如果反转后整数溢出那么就返回 0。
```

- 解答思路

```
// 使用符号标记，转成正数，循环得到%10的余数，再加上符号
func reverse(x int) int {
        flag := 1
        if x < 0 {
                flag = -1
                x = -1 * x
        }

        result := 0
        for x > 0 {
                temp := x % 10
                x = x / 10

                result = result*10 + temp
        }

        result = flag * result
        if result > math.MaxInt32 || result < math.MinInt32 {
                result = 0
        }
        return result
}

// 对x进行逐个%10取个位，一旦溢出，直接跳出循环
func reverse(x int) int {
        result := 0
        for x != 0 {
                temp := x % 10
                result = result*10 + temp
                if result > math.MaxInt32 || result < math.MinInt32 {
                        return 0
                }
                x = x / 10
        }
        return result
}
```

1.3 9. 回文数 (3)

- 题目

```
判断一个整数是否是回文数。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。
示例 1:        输入: 121        输出: true
示例 2:输入: -121        输出: false
解释: 从左向右读, 为 -121 。 从右向左读, 为 121- 。因此它不是一个回文数。
示例 3:输入: 10        输出: false
解释: 从右向左读, 为 01 。因此它不是一个回文数。
进阶: 你能不将整数转为字符串来解决这个问题吗？
```

- 解答思路

```go
// 数学解法，取出后半段数字进行翻转，然后判断是否相等
func isPalindrome(x int) bool {
        if x < 0 || (x%10 == 0 && x != 0) {
                return false
        }

        revertedNumber := 0
        for x > revertedNumber {
                temp := x % 10
                revertedNumber = revertedNumber*10 + temp
                x = x / 10
        }
        // for example:
        // x = 1221  => x = 12 revertedNumber = 12
        // x = 12321 => x = 12 revertedNumber = 123
        return x == revertedNumber || x == revertedNumber/10
}

// 转成字符串，依次判断
func isPalindrome(x int) bool {
        if x < 0 {
                return false
        }

        s := strconv.Itoa(x)
        for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
                if s[i] != s[j] {
                        return false
                }
        }
```

(续下页)

(接上页)

```
        return true
}

// 转成byte数组，依次判断，同2
func isPalindrome(x int) bool {
        if x < 0 {
                return false
        }
        arrs := []byte(strconv.Itoa(x))
        Len := len(arrs)
        for i := 0; i < Len/2; i++ {
                if arrs[i] != arrs[Len-i-1] {
                        return false
                }
        }
        return true
}
```

1.4 13. 罗马数字转整数 (2)

- 题目

```
罗马数字包含以下七种字符: I， V， X， L，C，D 和 M。
字符           数值
I              1
V              5
X              10
L              50
C              100
D              500
M              1000
例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。
27 写做  XXVII, 即为 XX + V + II 。
通常情况下，罗马数字中小的数字在大的数字的右边。
但也存在特例，例如 4 不写做 IIII，而是 IV。
数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。
同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况:
    I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
    X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
    C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。
给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。
示例 1:输入: "III" 输出: 3
```

(续下页)

(接上页)

```
示例 2:         输入: "IV" 输出: 4
示例 3:         输入: "IX"         输出: 9
示例 4:         输入: "LVIII"         输出: 58 解释: L = 50, V= 5, III = 3.
示例 5: 输入: "MCMXCIV"         输出: 1994 解释: M = 1000, CM = 900, XC = 90, IV = 4.
```

- 解答思路

```go
// 带标记位
func romanToInt(s string) int {
        m := map[byte]int{
                'I': 1,
                'V': 5,
                'X': 10,
                'L': 50,
                'C': 100,
                'D': 500,
                'M': 1000,
        }
        result := 0
        last := 0

        for i := len(s) - 1; i >= 0; i-- {
                current := m[s[i]]
                flag := 1
                if current < last {
                        flag = -1
                }
                result = result + flag*current
                last = current
        }
        return result
}

// 不带标记位，小于则减去2倍数
func romanToInt(s string) int {
        m := map[byte]int{
                'I': 1,
                'V': 5,
                'X': 10,
                'L': 50,
                'C': 100,
                'D': 500,
                'M': 1000,
        }
```

(续下页)

(接上页)

```
        result := 0
        last := 0

        for i := len(s) - 1; i >= 0; i-- {
                current := m[s[i]]
                if current < last {
                        result = result - current
                }else {
                        result = result + current
                }
                last = current
        }
        return result
}
```

1.5 14. 最长公共前缀 (6)

- 题目

```
编写一个函数来查找字符串数组中的最长公共前缀。
如果不存在公共前缀，返回空字符串 ""。
示例 1:输入: ["flower","flow","flight"] 输出: "fl"
示例 2:输入: ["dog","racecar","car"] 输出: ""
解释: 输入不存在公共前缀。
说明: 所有输入只包含小写字母 a-z 。
```

- 解答思路

```
// 先找最短的一个字符串，依次比较最短字符串子串是否是其他字符串子串
func longestCommonPrefix(strs []string) string {
        if len(strs) == 0{
                return ""
        }
        if len(strs) == 1{
                return strs[0]
        }

        short := strs[0]
        for _, s := range strs{
                if len(short) > len(s){
                        short = s
                }
```

(续下页)

(接上页)

```
        }

        for i := range short{
                shortest := short[:i+1]
                for _,str := range strs{
                        if strings.Index(str,shortest) != 0{
                                return short[:i]
                        }
                }
        }
        return short
}

// 暴力法:直接依次遍历
func longestCommonPrefix(strs []string) string {
        if len(strs) == 0 {
                return ""
        }
        if len(strs) == 1 {
                return strs[0]
        }

        length := 0

        for i := 0; i < len(strs[0]); i++ {
                char := strs[0][i]
                for j := 1; j < len(strs); j++ {
                        if i >= len(strs[j]) || char != strs[j][i] {
                                return strs[0][:length]
                        }
                }
                length++
        }
        return strs[0][:length]
}

// 排序后，遍历比较第一个，和最后一个字符串
func longestCommonPrefix(strs []string) string {
        if len(strs) == 0{
                return ""
        }
        if len(strs) == 1{
                return strs[0]
```

(续下页)

(接上页)

```
        }

        sort.Strings(strs)
        first := strs[0]
        last := strs[len(strs)-1]
        i := 0
        length := len(first)
        if len(last) < length{
                length = len(last)
        }
        for i < length{
                if first[i] != last[i]{
                        return first[:i]
                }
                i++
        }

        return first[:i]
}

// trie树
var trie [][]int
var index int

func longestCommonPrefix(strs []string) string {
        if len(strs) == 0 {
                return ""
        }
        if len(strs) == 1 {
                return strs[0]
        }

        trie = make([][]int, 2000)
        for k := range trie {
                value := make([]int, 26)
                trie[k] = value
        }
        insert(strs[0])

        minValue := math.MaxInt32
        for i := 1; i < len(strs); i++ {
                retValue := insert(strs[i])
                if minValue > retValue {
```

(续下页)

(接上页)

```
                        minValue = retValue
                }
        }
        return strs[0][:minValue]
}

func insert(str string) int {
        p := 0
        count := 0
        for i := 0; i < len(str); i++ {
                ch := str[i] - 'a'
                // fmt.Println(string(str[i]),p,ch,trie[p][ch])
                if value := trie[p][ch]; value == 0 {
                        index++
                        trie[p][ch] = index
                } else {
                        count++
                }
                p = trie[p][ch]
        }
        return count
}

//␣
↪水平扫描法:比较前2个字符串得到最长前缀，然后跟第3个比较得到一个新的最长前缀，继续比较，直到最后
func longestCommonPrefix(strs []string) string {
        if len(strs) == 0 {
                return ""
        }
        if len(strs) == 1 {
                return strs[0]
        }

        commonStr := common(strs[0], strs[1])
        if commonStr == "" {
                return ""
        }
        for i := 2; i < len(strs); i++ {
                if commonStr == "" {
                        return ""
                }
                commonStr = common(commonStr, strs[i])
        }
```

(续下页)

(接上页)

```
        return commonStr
}

func common(str1, str2 string) string {
        length := 0
        for i := 0; i < len(str1); i++ {
                char := str1[i]
                if i >= len(str2) || char != str2[i] {
                        return str1[:length]
                }
                length++
        }
        return str1[:length]
}

// 分治法
func longestCommonPrefix(strs []string) string {
        if len(strs) == 0 {
                return ""
        }
        if len(strs) == 1 {
                return strs[0]
        }

        return commonPrefix(strs, 0, len(strs)-1)
}

func commonPrefix(strs []string, left, right int) string {
        if left == right {
                return strs[left]
        }

        middle := (left + right) / 2
        leftStr := commonPrefix(strs, left, middle)
        rightStr := commonPrefix(strs, middle+1, right)
        return commonPrefixWord(leftStr, rightStr)
}

func commonPrefixWord(leftStr, rightStr string) string {
        if len(leftStr) > len(rightStr) {
                leftStr = leftStr[:len(rightStr)]
        }
```

(续下页)

(接上页)

```
        if len(leftStr) < 1 {
                return leftStr
        }

        for i := 0; i < len(leftStr); i++ {
                if leftStr[i] != rightStr[i] {
                        return leftStr[:i]
                }
        }
        return leftStr
}
```

1.6 20. 有效的括号 (3)

- 题目

```
给定一个只包括 '(', ')', '{', '}', '[', ']' 的字符串，判断字符串是否有效。
有效字符串需满足：
    左括号必须用相同类型的右括号闭合。
    左括号必须以正确的顺序闭合。
注意空字符串可被认为是有效字符串。
示例 1: 输入: "()" 输出: true
示例 2: 输入: "()[]{}" 输出: true
示例 3: 输入: "(]" 输出: false
示例 4: 输入: "([)]" 输出: false
示例 5: 输入: "{[]}" 输出: true
```

- 解题思路

```
// 使用栈结构实现
func isValid(s string) bool {
        st := new(stack)
        for _, char := range s {
                switch char {
                case '(', '[', '{':
                        st.push(char)
                case ')', ']', '}':
                        ret, ok := st.pop()
                        if !ok || ret != match[char] {
                                return false
                        }
                }
```

(续下页)

(接上页)

```
        }

        if len(*st) > 0 {
                return false
        }
        return true
}

var match = map[rune]rune{
        ')': '(',
        ']': '[',
        '}': '{',
}

type stack []rune

func (s *stack) push(b rune) {
        *s = append(*s, b)
}
func (s *stack) pop() (rune, bool) {
        if len(*s) > 0 {
                res := (*s)[len(*s)-1]
                *s = (*s)[:len(*s)-1]
                return res, true
        }
        return 0, false
}

// 借助数组实现栈
func isValid(s string) bool {
        if s == "" {
                return true
        }

        stack := make([]rune, len(s))
        length := 0
        var match = map[rune]rune{
                ')': '(',
                ']': '[',
                '}': '{',
        }

        for _, char := range s {
```

(续下页)

(接上页)

```
                switch char {
                case '(', '[', '{':
                        stack[length] = char
                        length++
                case ')', ']', '}':
                        if length == 0 {
                                return false
                        }
                        if stack[length-1] != match[char]{
                                return false
                        } else {
                                length--
                        }
                }
        }
        return length == 0
}

// 借助数组实现栈，使用数字表示来匹配
func isValid(s string) bool {
        if s == "" {
                return true
        }

        stack := make([]int, len(s))
        length := 0
        var match = map[rune]int{
                ')': 1,
                '(': -1,
                ']': 2,
                '[': -2,
                '}': 3,
                '{': -3,
        }

        for _, char := range s {
                switch char {
                case '(', '[', '{':
                        stack[length] = match[char]
                        length++
                case ')', ']', '}':
                        if length == 0 {
                                return false
```

(续下页)

(接上页)

```
                        }
                        if stack[length-1]+match[char] != 0 {
                                return false
                        } else {
                                length--
                        }
                }
        }
        return length == 0
}
```

1.7 21. 合并两个有序链表 (3)

- 题目

```
将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。
示例：输入：1->2->4, 1->3->4 输出：1->1->2->3->4->4
```

- 解题思路

```
// 迭代遍历
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        if l1 == nil {
                return l2
        }
        if l2 == nil {
                return l1
        }

        var head, node *ListNode
        if l1.Val < l2.Val {
                head = l1
                node = l1
                l1 = l1.Next
        } else {
                head = l2
                node = l2
                l2 = l2.Next
        }

        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
```

(续下页)

(接上页)

```
                        node.Next = l1
                        l1 = l1.Next
                } else {
                        node.Next = l2
                        l2 = l2.Next
                }
                node = node.Next
        }
        if l1 != nil {
                node.Next = l1
        }
        if l2 != nil {
                node.Next = l2
        }
        return head
}

// 递归遍历
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        if l1 == nil {
                return l2
        }
        if l2 == nil {
                return l1
        }

        if l1.Val < l2.Val{
                l1.Next = mergeTwoLists(l1.Next,l2)
                return l1
        }else {
                l2.Next = mergeTwoLists(l1,l2.Next)
                return l2
        }
}

#
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
```

(续下页)

(接上页)

```
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}
```

1.8 26. 删除排序数组中的重复项 (2)

- 题目

```
给定一个排序数组，你需要在原地删除重复出现的元素，使得每个元素只出现一次，返回移除后数组的新长度。
不要使用额外的数组空间，你必须在原地修改输入数组并在使用 O(1) 额外空间的条件下完成。
示例 1: 给定数组 nums = [1,1,2],
函数应该返回新的长度 2, 并且原数组 nums 的前两个元素被修改为 1, 2。
你不需要考虑数组中超出新长度后面的元素。
示例 2: 给定 nums = [0,0,1,1,1,2,2,3,3,4],
函数应该返回新的长度 5, 并且原数组 nums 的前五个元素被修改为 0, 1, 2, 3, 4。
你不需要考虑数组中超出新长度后面的元素。
说明: 为什么返回数值是整数，但输出的答案是数组呢?
请注意，输入数组是以“引用”方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。
你可以想象内部操作如下:
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);
// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度，它会打印出数组中该长度范围内的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

- 解题思路

```
// 双指针法
func removeDuplicates(nums []int) int {
        i , j , length := 0, 1, len(nums)
```

(续下页)

(接上页)

```
        for ; j < length; j++{
                if nums[i] == nums[j]{
                        continue
                }
                i++
                nums[i] = nums[j]
        }
        return i+1
}

// 计数法
func removeDuplicates(nums []int) int {
        count := 1
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] != nums[i+1] {
                        nums[count] = nums[i+1]
                        count++
                }
        }
        return count
}
```

1.9 27. 移除元素 (3)

- 题目

```
给定一个数组 nums 和一个值 val，你需要原地移除所有数值等于 val␣
↪的元素，返回移除后数组的新长度。
不要使用额外的数组空间，你必须在原地修改输入数组并在使用 O(1) 额外空间的条件下完成。
元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。
示例 1: 给定 nums = [3,2,2,3], val = 3,
函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。
你不需要考虑数组中超出新长度后面的元素。
示例 2: 给定 nums = [0,1,2,2,3,0,4,2], val = 2,
函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。
注意这五个元素可为任意顺序。
你不需要考虑数组中超出新长度后面的元素。
说明: 为什么返回数值是整数，但输出的答案是数组呢?
请注意，输入数组是以“引用”方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。
你可以想象内部操作如下:
// nums 是以“引用”方式传递的。也就是说，不对实参作任何拷贝
int len = removeElement(nums, val);
```

(续下页)

(接上页)

```
// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度，它会打印出数组中该长度范围内的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

- 解题思路

```
// 双指针，数字前移
func removeElement(nums []int, val int) int {
        i := 0
        for j := 0; j < len(nums); j++{
                if nums[j] != val{
                        nums[i] = nums[j]
                        i++
                }
        }
        return i
}

// 双指针，出现重复最后数字前移
func removeElement(nums []int, val int) int {
        i := 0
        n := len(nums)
        for i < n{
                if nums[i] == val{
                        nums[i] = nums[n-1]
                        n--
                }else {
                        i++
                }
        }
        return n
}

// 首位指针法
func removeElement(nums []int, val int) int {
        i, j := 0, len(nums)-1
        for {
                // 从左向右找到等于 val 的位置
                for i < len(nums) && nums[i] != val {
                        i++
                }
                // 从右向左找到不等于 val 的位置
```

(续下页)

(接上页)

```
                for j >= 0 && nums[j] == val {
                        j--
                }
                if i >= j {
                        break
                }
                // fmt.Println(i,j)
                nums[i], nums[j] = nums[j], nums[i]
        }
        return i
}
```

1.10 28. 实现 strStr()(4)

- 题目

```
实现 strStr() 函数。
给定一个 haystack 字符串和一个 needle 字符串，
在 haystack 字符串中找出 needle 字符串出现的第一个位置 (从0开始)。
如果不存在，则返回-1。
示例 1:输入: haystack = "hello", needle = "ll" 输出: 2
示例 2: 输入: haystack = "aaaaa", needle = "bba" 输出: -1
说明: 当 needle 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。
对于本题而言，当 needle 是空字符串时我们应当返回 0 。
这与C语言的 strstr() 以及 Java的 indexOf() 定义相符。
```

- 解题思路

```
// Sunday算法
func strStr(haystack string, needle string) int {
        if needle == ""{
                return 0
        }
        if len(needle) > len(haystack){
                return -1
        }
        // 计算模式串needle的偏移量
        m := make(map[int32]int)
        for k,v := range needle{
                m[v] = len(needle)-k
        }
```

(续下页)

(接上页)

```
        index := 0
        for index+len(needle) <= len(haystack){
                // 匹配字符串
                str := haystack[index:index+len(needle)]
                if str == needle{
                        return index
                }else {
                        if index + len(needle) >= len(haystack){
                                return -1
                        }
                        // 后一位字符串
                        next := haystack[index+len(needle)]
                        if nextStep,ok := m[int32(next)];ok{
                                index = index+nextStep
                        }else {
                                index = index+len(needle)+1
                        }
                }
        }
        if index + len(needle) >= len(haystack){
                return -1
        }else {
                return index
        }
}

//
func strStr(haystack string, needle string) int {
        hlen, nlen := len(haystack), len(needle)
        for i := 0; i <= hlen-nlen; i++ {
                if haystack[i:i+nlen] == needle {
                        return i
                }
        }
        return -1
}

//
func strStr(haystack string, needle string) int {
        return strings.Index(haystack, needle)
}

//
```

(续下页)

(接上页)

```
func strStr(haystack string, needle string) int {
        if len(needle) == 0 {
                return 0
        }

        next := getNext(needle)

        i := 0
        j := 0
        for i < len(haystack) && j < len(needle) {
                if j == -1 || haystack[i] == needle[j] {
                        i++
                        j++
                } else {
                        j = next[j]
                }
        }

        if j == len(needle) {
                return i - j
        }
        return -1
}

// 求next数组
func getNext(str string) []int {
        var next = make([]int, len(str))
        next[0] = -1

        i := 0
        j := -1

        for i < len(str)-1 {
                if j == -1 || str[i] == str[j] {
                        i++
                        j++
                        next[i] = j
                } else {
                        j = next[j]
                }
        }
        return next
}
```

1.11 35. 搜索插入位置 (3)

- 题目

```
给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。
如果目标值不存在于数组中，返回它将会被按顺序插入的位置。
你可以假设数组中无重复元素。
示例 1: 输入: [1,3,5,6], 5 输出: 2
示例 2: 输入: [1,3,5,6], 2 输出: 1
示例 3: 输入: [1,3,5,6], 7 输出: 4
示例 4: 输入: [1,3,5,6], 0 输出: 0
```

- 解题思路

```
// 二分查找
func searchInsert(nums []int, target int) int {
        low, high := 0, len(nums)-1
        for low <= high {
                mid := (low + high) / 2
                switch {
                case nums[mid] < target:
                        low = mid + 1
                case nums[mid] > target:
                        high = mid - 1
                default:
                        return mid
                }
        }
        return low
}

// 顺序查找
func searchInsert(nums []int, target int) int {
        i := 0
        for i < len(nums) && nums[i] < target {
                if nums[i] == target {
                        return i
                }
                i++
        }
        return i
}

// 顺序查找
```

(续下页)

(接上页)

```
func searchInsert(nums []int, target int) int {
        for i := 0; i < len(nums); i++ {
                if nums[i] >= target {
                        return i
                }
        }
        return len(nums)
}
```

1.12 38. 报数 (2)

- 题目

```
报数序列是一个整数序列，按照其中的整数的顺序进行报数，得到下一个数。其前五项如下：
1.     1
2.     11
3.     21
4.     1211
5.     111221
1 被读作  "one 1"  ("一个一") ，即 11。
11 被读作 "two 1s" ("两个一") ，即 21。
21 被读作 "one 2",  "one 1"  ("一个二" ,  "一个一") ，即 1211。
给定一个正整数 n（1 ≤ n ≤ 30），输出报数序列的第 n 项。
注意：整数顺序将表示为一个字符串。
示例 1:输入：1 输出："1"
示例 2: 输入：4 输出："1211"
```

- 解题思路

```
// 递推+双指针计数
func countAndSay(n int) string {
        strs := []byte{'1'}
        for i := 1; i < n; i++ {
                strs = say(strs)
        }
        return string(strs)
}

func say(strs []byte) []byte {
        result := make([]byte, 0, len(strs)*2)

        i, j := 0, 1
```

(续下页)

(接上页)

```
        for i < len(strs) {
                for j < len(strs) && strs[i] == strs[j] {
                        j++
                }
                // 几个几
                result = append(result, byte(j-i+'0'))
                result = append(result, strs[i])
                i = j
        }
        return result
}

// 递归+双指针计数
func countAndSay(n int) string {
        if n == 1 {
                return "1"
        }
        strs := countAndSay(n - 1)

        result := make([]byte, 0, len(strs)*2)

        i, j := 0, 1
        for i < len(strs) {
                for j < len(strs) && strs[i] == strs[j] {
                        j++
                }
                // 几个几
                result = append(result, byte(j-i+'0'))
                result = append(result, strs[i])
                i = j
        }
        return string(result)
}
```

1.13 53. 最大子序和 (5)

- 题目

```
给定一个整数数组 nums␣
↪，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。
示例：输入：[-2,1,-3,4,-1,2,1,-5,4]，输出：6 解释：连续子数组 [4,-1,2,1] 的和最大，为␣
↪6。
```

(续下页)

(接上页)

进阶：如果你已经实现复杂度为 O(n) 的解法，尝试使用更为精妙的分治法求解。

- 解题思路

```
// 贪心法
func maxSubArray(nums []int) int {
        result := nums[0]
        sum := 0
        for i := 0; i < len(nums); i++ {
                if sum > 0 {
                        sum += nums[i]
                } else {
                        sum = nums[i]
                }
                if sum > result {
                        result = sum
                }
        }
        return result
}

// 暴力法
func maxSubArray(nums []int) int {
        result := math.MinInt32

        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum += nums[j]
                        if sum > result {
                                result = sum
                        }
                }
        }
        return result
}

//
func maxSubArray(nums []int) int {
        dp := make([]int, len(nums))
        dp[0] = nums[0]
        result := nums[0]

        for i := 1; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                if dp[i-1]+nums[i] > nums[i] {
                        dp[i] = dp[i-1] + nums[i]
                } else {
                        dp[i] = nums[i]
                }

                if dp[i] > result {
                        result = dp[i]
                }
        }
        return result
}

// 动态规划
func maxSubArray(nums []int) int {
        dp := nums[0]
        result := dp

        for i := 1; i < len(nums); i++ {
                if dp+nums[i] > nums[i] {
                        dp = dp + nums[i]
                } else {
                        dp = nums[i]
                }

                if dp > result {
                        result = dp
                }
        }
        return result
}

// 分治法
func maxSubArray(nums []int) int {
        result := maxSubArr(nums, 0, len(nums)-1)
        return result
}

func maxSubArr(nums []int, left, right int) int {
        if left == right {
                return nums[left]
        }
```

(续下页)

(接上页)

```
        mid := (left + right) / 2
        leftSum := maxSubArr(nums, left, mid)         // 最大子序在左边
        rightSum := maxSubArr(nums, mid+1, right)     // 最大子序在右边
        midSum := findMaxArr(nums, left, mid, right) // 跨中心
        result := max(leftSum, rightSum)
        result = max(result, midSum)
        return result
}

func findMaxArr(nums []int, left, mid, right int) int {
        leftSum := math.MinInt32
        sum := 0
        // 从右到左
        for i := mid; i >= left; i-- {
                sum += nums[i]
                leftSum = max(leftSum, sum)
        }

        rightSum := math.MinInt32
        sum = 0
        // 从左到右
        for i := mid + 1; i <= right; i++ {
                sum += nums[i]
                rightSum = max(rightSum, sum)
        }
        return leftSum + rightSum
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

1.14 58. 最后一个单词的长度 (2)

- 题目

```
给定一个仅包含大小写字母和空格 ' ' 的字符串，返回其最后一个单词的长度。
如果不存在最后一个单词，请返回 0 。
说明：一个单词是指由字母组成，但不包含任何空格的字符串。
示例：输入："Hello World" 输出：5
```

- 解题思路

```
// 调用系统函数，切割为数组取最后一个值
func lengthOfLastWord(s string) int {
        arr := strings.Split(strings.Trim(s, " "), " ")
        return len(arr[len(arr)-1])
}

// 遍历统计
func lengthOfLastWord(s string) int {
        length := len(s)
        if length == 0 {
                return 0
        }

        result := 0
        for i := length - 1; i >= 0; i-- {
                if s[i] == ' ' {
                        if result > 0 {
                                return result
                        }
                        continue
                }
                result++
        }
        return result
}
```

1.15 66. 加一 (2)

- 题目

```
给定一个由整数组成的非空数组所表示的非负整数，在该数的基础上加一。
最高位数字存放在数组的首位， 数组中每个元素只存储单个数字。
你可以假设除了整数 0 之外，这个整数不会以零开头。
示例 1: 输入: [1,2,3] 输出: [1,2,4] 解释: 输入数组表示数字 123。
示例 2: 输入: [4,3,2,1] 输出: [4,3,2,2] 解释: 输入数组表示数字 4321。
```

- 解题思路

```go
// 模拟进位
func plusOne(digits []int) []int {
        length := len(digits)
        if length == 0 {
                return []int{1}
        }

        digits[length-1]++
        for i := length - 1; i > 0; i-- {
                if digits[i] < 10 {
                        break
                }
                digits[i] = digits[i] - 10
                digits[i-1]++
        }

        if digits[0] > 9 {
                digits[0] = digits[0] - 10
                digits = append([]int{1}, digits...)
        }

        return digits
}

// 模拟进位
func plusOne(digits []int) []int {
        for i := len(digits) - 1; i >= 0; i-- {
                if digits[i] < 9 {
                        digits[i]++
                        return digits
                } else {
                        digits[i] = 0
```

(续下页)

(接上页)

```
                }
        }
        return append([]int{1}, digits...)
}
```

1.16 67. 二进制求和 (2)

- 题目

```
给定两个二进制字符串，返回他们的和（用二进制表示）。
输入为非空字符串且只包含数字 1 和 0。
示例 1: 输入: a = "11", b = "1" 输出: "100"
示例 2:输入: a = "1010", b = "1011" 输出: "10101"
```

- 解题思路

```
// 转换成数组模拟
func addBinary(a string, b string) string {
        if len(a) < len(b) {
                a, b = b, a
        }
        length := len(a)

        A := transToInt(a, length)
        B := transToInt(b, length)

        return makeString(add(A, B))
}

func transToInt(s string, length int) []int {
        result := make([]int, length)
        ls := len(s)
        for i, b := range s {
                result[length-ls+i] = int(b - '0')
        }
        return result
}

func add(a, b []int) []int {
        length := len(a) + 1
        result := make([]int, length)
        for i := length - 1; i >= 1; i-- {
```

(续下页)

(接上页)

```
                temp := result[i] + a[i-1] + b[i-1]
                result[i] = temp % 2
                result[i-1] = temp / 2
        }
        i := 0
        for i < length-1 && result[i] == 0 {
                i++
        }
        return result[i:]
}

func makeString(nums []int) string {
        bytes := make([]byte, len(nums))
        for i := range bytes {
                bytes[i] = byte(nums[i]) + '0'
        }
        return string(bytes)
}

// 直接模拟
func addBinary(a string, b string) string {
        i := len(a) - 1
        j := len(b) - 1
        result := ""
        flag := 0
        current := 0

        for i >= 0 || j >= 0 {
                intA, intB := 0, 0
                if i >= 0 {
                        intA = int(a[i] - '0')
                }
                if j >= 0 {
                        intB = int(b[j] - '0')
                }
                current = intA + intB + flag
                flag = 0
                if current >= 2 {
                        flag = 1
                        current = current - 2
                }
                cur := strconv.Itoa(current)
                result = cur + result
```

(续下页)

(接上页)

```
                i--
                j--
        }
        if flag == 1 {
                result = "1" + result
        }
        return result
}
```

1.17 69.x 的平方根 (5)

- 题目

```
实现 int sqrt(int x) 函数。
计算并返回 x 的平方根，其中 x 是非负整数。
由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。
示例 1:输入: 4 输出: 2
示例 2:输入: 8 输出: 2
说明: 8 的平方根是 2.82842...,
     由于返回类型是整数，小数部分将被舍去。
```

- 解题思路

```
// 系统函数
func mySqrt(x int) int {
        result := int(math.Sqrt(float64(x)))
        return result
}

// 系统函数
func mySqrt(x int) int {
        result := math.Floor(math.Sqrt(float64(x)))
        return int(result)
}

// 牛顿迭代法
func mySqrt(x int) int {
        result := x
        for result*result > x {
                result = (result + x/result) / 2
        }
        return result
```

(续下页)

(接上页)

```
}

// 二分查找法
func mySqrt(x int) int {
        left := 1
        right := x
        for left <= right {
                mid := (left + right) / 2
                if mid == x/mid {
                        return mid
                } else if mid < x/mid {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        if left * left <= x{
                return left
        }else {
                return left-1
        }
}

// 暴力法:遍历
func mySqrt(x int) int {
        result := 0
        for i := 1; i <= x/i; i++ {
                if i*i == x {
                        return i
                }
                result = i
        }
        return result
}
```

1.18 70. 爬楼梯 (3)

- 题目

```
假设你正在爬楼梯。需要 n 阶你才能到达楼顶。
每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？
注意：给定 n 是一个正整数。
示例 1：  输入：  2 输出：  2 解释：  有两种方法可以爬到楼顶。
1.  1 阶 + 1 阶
2.  2 阶
示例 2：  输入：  3 输出：  3 解释：  有三种方法可以爬到楼顶。
1.  1 阶 + 1 阶 + 1 阶
2.  1 阶 + 2 阶
3.  2 阶 + 1 阶
```

- 解题思路

```
// 递归
var arr []int

func climbStairs(n int) int {
        arr = make([]int, n+1)
        return climbStart(0, n)
}

func climbStart(i, n int) int {
        if i > n {
                return 0
        }
        if i == n {
                return 1
        }
        if arr[i] > 0 {
                return arr[i]
        }
        arr[i] = climbStart(i+1, n) + climbStart(i+2, n)
        return arr[i]
}

// 动态规划
func climbStairs(n int) int {
        if n == 1 {
                return 1
        }
```

(续下页)

(接上页)

```
        if n == 2 {
                return 2
        }
        dp := make([]int, n+1)
        dp[1] = 1
        dp[2] = 2
        for i := 3; i <= n; i++ {
                dp[i] = dp[i-1] + dp[i-2]
        }
        return dp[n]
}

// 斐波那契
func climbStairs(n int) int {
        if n == 1 {
                return 1
        }
        first := 1
        second := 2
        for i := 3; i <= n; i++ {
                third := first + second
                first = second
                second = third
        }
        return second
}
```

1.19 83. 删除排序链表中的重复元素 (3)

- 题目

```
给定一个排序链表，删除所有重复的元素，使得每个元素只出现一次。
示例 1:输入: 1->1->2 输出: 1->2
示例 2:输入: 1->1->2->3->3 输出: 1->2->3
```

- 解题思路

```
// 直接法
func deleteDuplicates(head *ListNode) *ListNode {
        if head == nil {
                return nil
        }
```

(续下页)

(接上页)

```
        temp := head
        for temp.Next != nil {
                if temp.Val == temp.Next.Val {
                        temp.Next = temp.Next.Next
                } else {
                        temp = temp.Next
                }
        }
        return head
}

// 递归法
func deleteDuplicates(head *ListNode) *ListNode {
        if head == nil || head.Next == nil{
                return head
        }
        head.Next = deleteDuplicates(head.Next)
        if head.Val == head.Next.Val{
                head = head.Next
        }
        return head
}

// 双指针法
func deleteDuplicates(head *ListNode) *ListNode {
        if head == nil || head.Next == nil{
                return head
        }
        p := head
        q := head.Next
        for p.Next != nil{
                if p.Val == q.Val{
                        if q.Next == nil{
                                p.Next = nil
                        }else {
                                p.Next = q.Next
                                q = q.Next
                        }
                }else {
                        p = p.Next
                        q = q.Next
                }
        }
```

(续下页)

(接上页)

```
        return head
}
```

1.20 88. 合并两个有序数组 (3)

- 题目

```
给定两个有序整数数组 nums1 和 nums2，将 nums2 合并到 nums1 中，使得 num1␣
↪成为一个有序数组。
说明:
    初始化 nums1 和 nums2 的元素数量分别为 m 和 n。
    你可以假设 nums1 有足够的空间（空间大小大于或等于 m + n）来保存 nums2 中的元素。
示例:输入: nums1 = [1,2,3,0,0,0], m = 3 nums2 = [2,5,6],       n = 3
输出: [1,2,2,3,5,6]
```

- 解题思路

```
// 合并后排序
func merge(nums1 []int, m int, nums2 []int, n int) {
        nums1 = nums1[:m]
        nums1 = append(nums1, nums2[:n]...)
        sort.Ints(nums1)
}

// 双指针法
func merge(nums1 []int, m int, nums2 []int, n int) {
        for m > 0 && n > 0 {
                if nums1[m-1] < nums2[n-1] {
                        nums1[m+n-1] = nums2[n-1]
                        n--
                } else {
                        nums1[m+n-1] = nums1[m-1]
                        m--
                }
        }
        if m == 0 && n > 0 {
                for n > 0 {
                        nums1[n-1] = nums2[n-1]
                        n--
                }
        }
}
```

(续下页)

(接上页)

```
// 拷贝后插入
func merge(nums1 []int, m int, nums2 []int, n int) {
        temp := make([]int, m)
        copy(temp, nums1)

        if n == 0 {
                return
        }
        first, second := 0, 0
        for i := 0; i < len(nums1); i++ {
                if second >= n {
                        nums1[i] = temp[first]
                        first++
                        continue
                }
                if first >= m {
                        nums1[i] = nums2[second]
                        second++
                        continue
                }
                if temp[first] < nums2[second] {
                        nums1[i] = temp[first]
                        first++
                } else {
                        nums1[i] = nums2[second]
                        second++
                }
        }
}
```

1.21 100. 相同的树 (2)

- 题目

```
给定两个二叉树，编写一个函数来检验它们是否相同。
如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。
示例 1:
输入:       1          1
          / \        / \
         2   3      2   3
        [1,2,3],   [1,2,3]
```

(续下页)

(接上页)

```
输出: true
示例 2:
输入:      1           1
          /            \
         2              2
        [1,2],      [1,null,2]
输出: false
示例 3:
输入:       1          1
          / \         / \
         2   1       1   2
        [1,2,1],    [1,1,2]
输出: false
```

- 解题思路

```
// 递归(深度优先)
func isSameTree(p *TreeNode, q *TreeNode) bool {
        if p == nil && q == nil {
                return true
        }

        if p == nil || q == nil {
                return false
        }

        return p.Val == q.Val && isSameTree(p.Left, q.Left) &&
                isSameTree(p.Right, q.Right)
}

// 层序遍历(宽度优先)
func isSameTree(p *TreeNode, q *TreeNode) bool {
        if p == nil && q == nil {
                return true
        }
        if p == nil || q == nil {
                return false
        }

        var queueP, queueQ []*TreeNode
        if p != nil {
                queueP = append(queueP, p)
                queueQ = append(queueQ, q)
        }
```

(续下页)

(接上页)

```
        for len(queueP) > 0 && len(queueQ) > 0 {
                tempP := queueP[0]
                queueP = queueP[1:]

                tempQ := queueQ[0]
                queueQ = queueQ[1:]

                if tempP.Val != tempQ.Val {
                        return false
                }

                if (tempP.Left != nil && tempQ.Left == nil) ||
                        (tempP.Left == nil && tempQ.Left != nil) {
                        return false
                }
                if tempP.Left != nil {
                        queueP = append(queueP, tempP.Left)
                        queueQ = append(queueQ, tempQ.Left)
                }

                if (tempP.Right != nil && tempQ.Right == nil) ||
                        (tempP.Right == nil && tempQ.Right != nil) {
                        return false
                }
                if tempP.Right != nil {
                        queueP = append(queueP, tempP.Right)
                        queueQ = append(queueQ, tempQ.Right)
                }
        }
        return true
}
```

CHAPTER 2

0001-0100-Medium

2.1 2. 两数相加 (2)

- 题目

```
给出两个 非空 的链表用来表示两个非负的整数。
其中，它们各自的位数是按照 逆序 的方式存储的，并且它们的每个节点只能存储 一位 数字。
如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。
您可以假设除了数字 0 之外，这两个数都不会以 0 开头。
示例：输入：(2 -> 4 -> 3) + (5 -> 6 -> 4) 输出：7 -> 0 -> 8
原因：342 + 465 = 807
```

- 解题思路

```
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        cur := res
        carry := 0
        for l1 != nil || l2 != nil || carry > 0 {
                sum := carry
                if l1 != nil {
                        sum += l1.Val
                        l1 = l1.Next
                }
                if l2 != nil {
```

(续下页)

(接上页)

```
                        sum += l2.Val
                        l2 = l2.Next
                }
                carry = sum / 10 // 进位
                cur.Next = &ListNode{Val: sum % 10}
                cur = cur.Next
        }
        return res.Next
}

#
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        if l1 == nil && l2 == nil {
                return nil
        }
        if l1 == nil {
                return l2
        }
        if l2 == nil {
                return l1
        }
        sum := l1.Val + l2.Val
        res := &ListNode{Val: sum % 10}
        if sum >= 10 {
                l1.Next = addTwoNumbers(l1.Next, &ListNode{Val: 1})
        }
        res.Next = addTwoNumbers(l1.Next, l2.Next)
        return res
}
```

2.2 3. 无重复字符的最长子串 (4)

- 题目

```
给定一个字符串，请你找出其中不含有重复字符的 最长子串 的长度。
示例 1:输入: "abcabcbb"输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
示例 2:输入: "bbbbb" 输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
示例 3:输入: "pwwkew" 输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
```

(续下页)

(接上页)

同剑指offer面试题48.最长不含重复字符的子字符串

- 解题思路

```
func lengthOfLongestSubstring(s string) int {
        arr := [256]int{}
        for i := range arr {
                arr[i] = -1
        }
        max, j := 0, 0
        for i := 0; i < len(s); i++ {
                if arr[s[i]] >= j {
                        j = arr[s[i]] + 1
                } else if i+1-j > max {
                        max = i + 1 - j
                }
                arr[s[i]] = i
        }
        return max
}

# 2
func lengthOfLongestSubstring(s string) int {
        max, j := 0, 0
        for i := 0; i < len(s); i++ {
                index := strings.Index(s[j:i], string(s[i]))
                if index == -1 {
                        continue
                }
                if i-j > max {
                        max = i - j
                }
                j = j + index + 1
        }
        if len(s)-j > max {
                max = len(s) - j
        }
        return max
}

# 3
func lengthOfLongestSubstring(s string) int {
        m := make(map[uint8]int)
        max, j := 0, 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(s); i++ {
                if v, ok := m[s[i]]; ok && v >= j {
                        j = v + 1
                } else if i+1-j > max {
                        max = i + 1 - j
                }
                m[s[i]] = i
        }
        return max
}

# 4
func lengthOfLongestSubstring(s string) int {
        if len(s) < 1 {
                return 0
        }
        dp := make([]int, len(s))
        dp[0] = 1
        res := 1
        m := make(map[byte]int)
        m[s[0]] = 0
        for i := 1; i < len(s); i++ {
                index := -1
                if value, ok := m[s[i]]; ok {
                        index = value
                }
                if i-index > dp[i-1] {
                        dp[i] = dp[i-1] + 1
                } else {
                        dp[i] = i - index
                }
                m[s[i]] = i
                if dp[i] > res {
                        res = dp[i]
                }
        }
        return res
}

# 5
func lengthOfLongestSubstring(s string) int {
        arr := [256]int{}
        for i := range arr {
```

(续下页)

(接上页)

```
                arr[i] = -1
        }
        res, j := 0, -1
        for i := 0; i < len(s); i++ {
                if arr[s[i]] > j { // 出现重复了，更新下标
                        j = arr[s[i]]
                } else {
                        res = max(res, i-j) // 没有重复，更新长度
                }
                arr[s[i]] = i
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

2.3 5. 最长回文子串 (5)

- 题目

```
给定一个字符串 s，找到 s 中最长的回文子串。你可以假设 s 的最大长度为 1000。
示例 1：输入："babad"输出："bab" 注意："aba" 也是一个有效答案。
示例 2：输入："cbbd" 输出："bb"
```

- 解题思路

```
// dp(l,r)=dp(l+1,r-1)&&(s[l]==s[r])
// dp[l,r]：字符串s从索引l到r的子串是否是回文串
func longestPalindrome(s string) string {
        if len(s) <= 1 {
                return s
        }
        dp := make([][]bool, len(s))
        start := 0
        max := 1
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
```

(续下页)

(接上页)

```
                dp[r][r] = true
                for l := 0; l < r; l++ {
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
                        }
                        if dp[l][r] == true {
                                if r-l+1 > max {
                                        max = r - l + 1
                                        start = l
                                }
                        }
                }
        }
        return s[start : start+max]
}

# 2
func longestPalindrome(s string) string {
        if len(s) <= 1 {
                return s
        }
        start := 0
        end := 0
        for i := 0; i < len(s); i++ {
                left1, right1 := find(s, i, i)
                left2, right2 := find(s, i, i+1)
                if right1-left1 > end-start {
                        start, end = left1, right1
                }
                if right2-left2 > end-start {
                        start, end = left2, right2
                }
        }
        return s[start : end+1]
}

func find(s string, left, right int) (int, int) {
        for ; 0 <= left && right < len(s) && s[left] == s[right]; left, right = left-
↪1, right+1 {
        }
        return left + 1, right - 1
```

(续下页)

(接上页)

```
}

# 3
func longestPalindrome(s string) string {
        if len(s) <= 1 {
                return s
        }
        res := ""
        for i := 0; i < len(s); i++ {
                for j := i; j < len(s); j++ {
                        str := s[i : j+1]
                        if len(str) < len(res) && res != "" {
                                continue
                        }
                        if judge(str) == true && len(res) < len(str) {
                                res = str
                        }
                }
        }
        return res
}

func judge(s string) bool {
        for i := 0; i < len(s)/2; i++ {
                if s[i] != s[len(s)-1-i] {
                        return false
                }
        }
        return true
}

# 4
func longestPalindrome(s string) string {
        if len(s) <= 1 {
                return s
        }
        str := add(s)
        length := len(str)
        max := 1
        begin := 0
        for i := 0; i < length; i++ {
                curLength := search(str, i)
                if curLength > max {
```

(续下页)

(接上页)

```
                        max = curLength
                        begin = (i - max) / 2
                }
        }
        return s[begin : begin+max]
}

func search(s string, center int) int {
        i := center - 1
        j := center + 1
        step := 0
        for ; i >= 0 && j < len(s) && s[i] == s[j]; i, j = i-1, j+1 {
                step++
        }
        return step
}

func add(s string) string {
        var res []rune
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        return string(res)
}

#
func longestPalindrome(s string) string {
        if len(s) <= 1 {
                return s
        }
        str := add(s)
        length := len(str)
        temp := make([]int, length)
        maxRight := 0
        center := 0
        max := 1
        begin := 0
        for i := 0; i < length; i++ {
                if i < maxRight {
                        mirror := 2*center - i
                        temp[i] = min(maxRight-i, temp[mirror])
```

(续下页)

(接上页)

```
                }
                left := i - (1 + temp[i])
                right := i + (1 + temp[i])
                for left >= 0 && right < len(str) && str[left] == str[right] {
                        temp[i]++
                        left--
                        right++
                }
                if i+temp[i] > maxRight {
                        maxRight = i + temp[i]
                        center = i
                }
                if temp[i] > max {
                        max = temp[i]
                        begin = (i - max) / 2
                }
        }
        return s[begin : begin+max]
}

func add(s string) string {
        var res []rune
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        return string(res)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

2.4 6.Z 字形变换 (2)

- 题目

```
将一个给定字符串根据给定的行数，以从上往下、从左到右进行 Z 字形排列。
比如输入字符串为 "LEETCODEISHIRING" 行数为 3 时，排列如下：
L   C   I   R
E T O E S I I G
E   D   H   N
之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如："LCIRETOESIIGEDHN"。
请你实现这个将字符串进行指定行数变换的函数：
string convert(string s, int numRows);
示例 1:输入：s = "LEETCODEISHIRING", numRows = 3 输出："LCIRETOESIIGEDHN"
示例 2:输入：s = "LEETCODEISHIRING", numRows = 4 输出："LDREOEIIECIHNTSG"
解释：
L     D     R
E   O E   I I
E C   I H   N
T     S     G
```

- 解题思路

```
func convert(s string, numRows int) string {
        if numRows == 1 {
                return s
        }
        arr := []rune(s)
        total := numRows*2 - 2
        res := make([]string, numRows)
        for i := 0; i < len(arr); i++ {
                index := i % total
                if index < numRows {
                        res[index] = res[index] + string(arr[i])
                } else {
                        res[total-index] = res[total-index] + string(arr[i])
                }
        }
        return strings.Join(res, "")
}

#
func convert(s string, numRows int) string {
        if numRows == 1 {
                return s
```

(续下页)

(接上页)

```
        }
        arr := []rune(s)
        res := make([]string, numRows)
        flag := -1
        index := 0
        for i := 0; i < len(arr); i++ {
                res[index] = res[index] + string(arr[i])
                if index == 0 || index == numRows-1 {
                        flag = -flag
                }
                index = index + flag
        }
        return strings.Join(res, "")
}
```

2.5 8. 字符串转换整数 (atoi)(3)

- 题目

```
请你来实现一个 atoi 函数，使其能将字符串转换成整数。
首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。接下来的转化规则如下
 ␣
↪如果第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字字符组合起来，形成一个有符号
   假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成一个整数。
 ␣
↪该字符串在有效的整数部分之后也可能会存在多余的字符，那么这些字符可以被忽略，它们对函数不应该造成影
注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，
则你的函数不需要进行转换，即无法进行有效转换。
在任何情况下，若函数不能进行有效的转换时，请返回 0 。
提示：
    本题中的空白字符只包括空格字符 ' ' 。
    假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 [−231,  231 − 1]。
    如果数值超过这个范围，请返回  INT_MAX (231 − 1) 或 INT_MIN (−231) 。
示例 1:输入："42" 输出：42
示例 2:输入："   -42" 输出：-42
解释：第一个非空白字符为 '-', 它是一个负号。
     我们尽可能将负号与后面所有连续出现的数字组合起来，最后得到 -42 。
示例 3:输入："4193 with words" 输出：4193
解释：转换截止于数字 '3' ，因为它的下一个字符不为数字。
示例 4:输入："words and 987" 输出：0
解释：第一个非空字符是 'w', 但它不是数字或正、负号。
     因此无法执行有效的转换。
```

(续下页)

(接上页)

```
示例 5:输入: "-91283472332" 输出: -2147483648
解释: 数字 "-91283472332" 超过 32 位有符号整数范围。
     因此返回 INT_MIN (-231) 。
```

- 解题思路

```
func myAtoi(str string) int {
        i := 0
        for i < len(str) && str[i] == ' ' {
                i++
        }
        str = str[i:]
        arr := make([]byte, 0)
        isFlag := byte(' ')
        for j := 0; j < len(str); j++ {
                if str[j] >= '0' && str[j] <= '9' {
                        arr = append(arr, str[j])
                } else {
                        if len(arr) > 0 {
                                break
                        }
                        if str[j] != ' ' && str[j] != '+' && str[j] != '-' {
                                return 0
                        }
                        if isFlag != ' ' {
                                return 0
                        }
                        isFlag = str[j]
                }
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                value := int(arr[i] - '0')
                res = res*10 + value
                if isFlag == '-' {
                        if -1*res < math.MinInt32 {
                                return math.MinInt32
                        }
                } else if isFlag == ' ' || isFlag == '+' {
                        if res > math.MaxInt32 {
                                return math.MaxInt32
                        }
                }
        }
```

(续下页)

(接上页)

```
        if isFlag == '-' {
                return -1 * res
        }
        return res
}

#
func myAtoi(str string) int {
        re := regexp.MustCompile(`^[+-]?\d+`)
        arrS := re.FindAllString(strings.Trim(str, " "), -1)
        if len(arrS) == 0{
                return 0
        }
        arr := arrS[0]
        res := 0
        isFlag := byte(' ')
        if !(arr[0] >= '0' && arr[0] <= '9') {
                isFlag = arr[0]
                arr = arr[1:]
        }
        for i := 0; i < len(arr); i++ {
                value := int(arr[i] - '0')
                if isFlag == '-' {
                        if res > 214748364 || (res==214748364 && value >= 8) {
                                return math.MinInt32
                        }
                } else if isFlag == ' ' || isFlag == '+' {
                        if res > 214748364 || (res==214748364 && value >= 7) {
                                return math.MaxInt32
                        }
                }
                res = res*10 + value
        }
        if isFlag == '-' {
                return -1 * res
        }
        return res
}

#
func myAtoi(str string) int {
        str = strings.TrimSpace(str)
        result := 0
```

(续下页)

(接上页)

```
        flag := 1
        for i, v := range str {
                if v >= '0' && v <= '9' {
                        result = result*10 + int(v-'0')
                } else if v == '-' && i == 0 {
                        flag = -1
                } else if v == '+' && i == 0 {
                        flag = 1
                } else {
                        break
                }
                if result > math.MaxInt32 {
                        if flag == -1 {
                                return math.MinInt32
                        }
                        return math.MaxInt32
                }
        }
        return flag * result
}
```

2.6 11. 盛最多水的容器 (2)

- 题目

```
给你 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。
在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0)。
找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。
说明：你不能倾斜容器，且 n 的值至少为 2。
图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,
↪7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。
示例：输入：[1,8,6,2,5,4,8,3,7] 输出：49
```

- 解题思路

```
func maxArea(height []int) int {
        i := 0
        j := len(height) - 1
        res := 0
        for i < j {
                area := (j - i) * min(height[i], height[j])
                if area > res {
```

(续下页)

(接上页)

```
                        res = area
                }
                // 移动较小的指针，尝试获取更大的面积
                if height[i] > height[j] {
                        j--
                } else {
                        i++
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func maxArea(height []int) int {
        res := 0
        for i := 0; i < len(height); i++ {
                for j := i + 1; j < len(height); j++ {
                        area := (j - i) * min(height[i], height[j])
                        if area > res {
                                res = area
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

2.7 12. 整数转罗马数字 (2)

- 题目

```
罗马数字包含以下七种字符： I， V， X， L，C，D 和 M。
字符          数值
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。
27 写做  XXVII, 即为 XX + V + II 。
通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做␣
↪IIII，而是 IV。
数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。
同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：
    I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
    X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
    C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。
给定一个整数，将其转为罗马数字。输入确保在 1 到 3999 的范围内。
示例 1:输入: 3输出: "III"
示例 2:输入: 4 输出: "IV"
示例 3:输入: 9 输出: "IX"
示例 4:输入: 58 输出: "LVIII"
解释: L = 50, V = 5, III = 3.
示例 5:输入: 1994 输出: "MCMXCIV"
解释: M = 1000, CM = 900, XC = 90, IV = 4.
```

- 解题思路

```
func intToRoman(num int) string {
        m := map[int]string{
                1:    "I",
                4:    "IV",
                5:    "V",
                9:    "IX",
                10:   "X",
                40:   "XL",
                50:   "L",
                90:   "XC",
                100:  "C",
```

(续下页)

(接上页)

```
                400:  "CD",
                500:  "D",
                900:  "CM",
                1000: "M",
        }
        arr := []int{1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1}
        result := ""
        for i := 0; i < len(arr); i++ {
                if num == 0 {
                        break
                }
                value := num / arr[i]
                for j := 0; j < value; j++ {
                        result = result + m[arr[i]]
                }
                num = num - value*arr[i]
        }
        return result
}

#
func intToRoman(num int) string {
        res := ""
        arr1 := []string{"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"}
        arr2 := []string{"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"}
        arr3 := []string{"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"}
        arr4 := []string{"", "M", "MM", "MMM"}
        res = arr4[num/1000] + arr3[num%1000/100] + arr2[num%100/10] + arr1[num%10]
        return res
}
```

2.8 15. 三数之和 (2)

- 题目

```
给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b +␣
↪c = 0 ？
请你找出所有满足条件且不重复的三元组。
注意：答案中不可以包含重复的三元组。
示例：
给定数组 nums = [-1, 0, 1, 2, -1, -4]，
满足要求的三元组集合为：
```

(续下页)

(接上页)

```
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

- 解题思路

```
func threeSum(nums []int) [][]int {
        res := make([][]int, 0)
        sort.Ints(nums)
        for i := 0; i < len(nums)-1; i++ {
                target := 0 - nums[i]
                left := i + 1
                right := len(nums) - 1
                if nums[i] > 0 || nums[i]+nums[left] > 0 {
                        break
                }
                if i > 0 && nums[i] == nums[i-1] {
                        continue
                }
                for left < right {
                        if left > i+1 && nums[left] == nums[left-1] {
                                left++
                                continue
                        }
                        if right < len(nums)-2 && nums[right] == nums[right+1] {
                                right--
                                continue
                        }
                        if nums[left]+nums[right] > target {
                                right--
                        } else if nums[left]+nums[right] < target {
                                left++
                        } else {
                                res = append(res, []int{nums[i], nums[left],␣
↪nums[right]})
                                left++
                                right--
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func threeSum(nums []int) [][]int {
        res := make([][]int, 0)
        m := make(map[[2]int]int)
        p := make(map[int]int)
        sort.Ints(nums)
        for k, v := range nums {
                p[v] = k
        }
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if j != i+1 && nums[j] == nums[j-1] {
                                continue
                        }
                        sum := nums[i] + nums[j]
                        if sum > 0 {
                                break
                        }
                        if value, ok := p[-sum]; ok && value > j {
                                if _, ok2 := m[[2]int{nums[i], nums[j]}]; !ok2 {
                                        res = append(res, []int{nums[i], nums[j], 0 -␣
↪nums[i] - nums[j]})
                                        m[[2]int{nums[i], nums[j]}] = 1
                                }
                        }
                }
        }
        return res
}
```

2.9 16. 最接近的三数之和 (2)

- 题目

```
给定一个包括 n 个整数的数组 nums 和 一个目标值 target。
找出 nums 中的三个整数，使得它们的和与 target␣
↪最接近。返回这三个数的和。假定每组输入只存在唯一答案。
示例：输入：nums = [-1,2,1,-4], target = 1 输出：2
解释：与 target 最接近的和是 2 (-1 + 2 + 1 = 2) 。
提示：
    3 <= nums.length <= 10^3
    -10^3 <= nums[i] <= 10^3
```

(续下页)

(接上页)

```
    -10^4 <= target <= 10^4
```

- 解题思路

```
func threeSumClosest(nums []int, target int) int {
        sort.Ints(nums)
        res := nums[0] + nums[1] + nums[2]
        for i := 0; i < len(nums); i++ {
                left := i + 1
                right := len(nums) - 1
                for left < right {
                        sum := nums[i] + nums[left] + nums[right]
                        if sum > target {
                                right--
                        } else if sum < target {
                                left++
                        } else {
                                return target
                        }
                        if abs(sum, target) < abs(res, target) {
                                res = sum
                        }
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func threeSumClosest(nums []int, target int) int {
        res := nums[0] + nums[1] + nums[2]
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        for k := j + 1; k < len(nums); k++ {
                                sum := nums[i] + nums[j] + nums[k]
                                if abs(sum, target) < abs(res, target) {
                                        res = sum
                                }
```

(续下页)

(接上页)

```
                        }
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

2.10 17. 电话号码的字母组合 (2)

- 题目

```
给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。
给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。
示例:输入："23"输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
说明:尽管上面的答案是按字典序排列的，但是你可以任意选择答案输出的顺序。
```

- 解题思路

```
func letterCombinations(digits string) []string {
        if len(digits) == 0 {
                return nil
        }
        arr := []string{"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv",
↪"wxyz"}
        res := []string{""}
        for i := 0; i < len(digits); i++ {
                length := len(res)
                for j := 0; j < length; j++ {
                        for k := 0; k < len(arr[digits[i]-'0']); k++ {
                                res = append(res, res[j]+string(arr[digits[i]-'0
↪'][k]))
                        }
                }
                res = res[length:]
        }
        return res
```

(续下页)

(接上页)

```
}

#
var res []string
var arr = []string{"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"}

func letterCombinations(digits string) []string {
        if len(digits) == 0 {
                return nil
        }
        res = make([]string, 0)
        dfs(digits, 0, "")
        return res
}

func dfs(digits string, index int, str string) {
        if index == len(digits) {
                res = append(res, str)
                return
        }
        for i := 0; i < len(arr[digits[index]-'0']); i++ {
                dfs(digits, index+1, str+string(arr[digits[index]-'0'][i]))
        }
}
```

2.11 18. 四数之和 (3)

- 题目

```
给定一个包含 n 个整数的数组 nums 和一个目标值 target，判断 nums 中是否存在四个元素␣
↪a，b，c 和 d ，
使得 a + b + c + d 的值与 target 相等？找出所有满足条件且不重复的四元组。
注意：答案中不可以包含重复的四元组。
示例：给定数组 nums = [1, 0, -1, 0, -2, 2]，和 target = 0。
满足要求的四元组集合为：
[
  [-1,  0, 0, 1],
  [-2, -1, 1, 2],
  [-2,  0, 0, 2]
]
```

- 解题思路

```
func fourSum(nums []int, target int) [][]int {
        sort.Ints(nums)
        res := make([][]int, 0)
        for i := 0; i < len(nums); i++ {
                if i > 0 && nums[i] == nums[i-1] {
                        continue
                }
                for j := i + 1; j < len(nums); j++ {
                        if j > i+1 && nums[j] == nums[j-1] {
                                continue
                        }
                        temp := target - nums[i] - nums[j]
                        left := j + 1
                        right := len(nums) - 1
                        for left < right {
                                if left > j+1 && nums[left] == nums[left-1] {
                                        left++
                                        continue
                                }
                                if right < len(nums)-2 && nums[right] ==␣
↪nums[right+1] {
                                        right--
                                        continue
                                }
                                if nums[left]+nums[right] > temp {
                                        right--
                                } else if nums[left]+nums[right] < temp {
                                        left++
                                } else {
                                        res = append(res, []int{nums[i], nums[j],␣
↪nums[left], nums[right]})
                                        left++
                                        right--
                                }
                        }
                }
        }
        return res
}

#
func fourSum(nums []int, target int) [][]int {
        m := make(map[[3]int]int)
        p := make(map[int]int)
```

(续下页)

(接上页)

```
        sort.Ints(nums)
        for k, v := range nums {
                p[v] = k
        }
        res := make([][]int, 0)
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        for k := j + 1; k < len(nums); k++ {
                                sum := nums[i] + nums[j] + nums[k]
                                if value, ok := p[target-sum]; ok && value > k {
                                        if _, ok2 := m[[3]int{nums[i], nums[j],␣
↪nums[k]}]; !ok2 {
                                                res = append(res, []int{nums[i],␣
↪nums[j], nums[k], target - nums[i] - nums[j] - nums[k]})
                                                m[[3]int{nums[i], nums[j], nums[k]}]␣
↪= 1
                                        }
                                }
                        }
                }
        }
        return res
}

#
var res [][]int

func fourSum(nums []int, target int) [][]int {
        sort.Ints(nums)
        res = make([][]int, 0)
        dfs(nums, target, []int{}, 0)
        return res
}

func dfs(nums []int, target int, arr []int, level int) {
        if len(arr) == 4 {
                sum := 0
                for i := 0; i < len(arr); i++ {
                        sum = sum + arr[i]
                }
                if sum == target {
                        tempArr := make([]int, len(arr))
                        copy(tempArr, arr)
```

(续下页)

(接上页)

```
                        res = append(res, tempArr)
                }
                return
        }
        prev := math.MaxInt32
        for i := level; i < len(nums); i++ {
                if nums[i] != prev {
                        prev = nums[i]
                        arr = append(arr, nums[i])
                        dfs(nums, target, arr, i+1)
                        arr = arr[:len(arr)-1]
                }
        }
}
```

2.12 19. 删除链表的倒数第 N 个节点 (3)

- 题目

```
给定一个链表，删除链表的倒数第 n 个节点，并且返回链表的头结点。
示例：给定一个链表：1->2->3->4->5, 和 n = 2.
当删除了倒数第二个节点后，链表变为 1->2->3->5.
说明：给定的 n 保证是有效的。
进阶：你能尝试使用一趟扫描实现吗？
```

- 解题思路

```
func removeNthFromEnd(head *ListNode, n int) *ListNode {
        temp := &ListNode{Next: head}
        cur := temp
        total := 0
        for cur.Next != nil {
                cur = cur.Next
                total++
        }
        cur = temp
        count := 0
        for cur.Next != nil {
                if total-n == count {
                        cur.Next = cur.Next.Next
                        break
                }
```

(续下页)

(接上页)

```
                cur = cur.Next
                count++
        }
        return temp.Next
}

#
func removeNthFromEnd(head *ListNode, n int) *ListNode {
        temp := &ListNode{Next: head}
        fast, slow := temp, temp
        for i := 0; i < n; i++ {
                fast = fast.Next
        }
        for fast.Next != nil {
                fast = fast.Next
                slow = slow.Next
        }
        slow.Next = slow.Next.Next
        return temp.Next
}

#
var count int

func removeNthFromEnd(head *ListNode, n int) *ListNode {
        if head == nil {
                count = 0
                return nil
        }
        head.Next = removeNthFromEnd(head.Next, n)
        count = count + 1
        if count == n {
                return head.Next
        }
        return head
}
```

2.13 22. 括号生成 (3)

- 题目

```
数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 有效的括号组合。
示例：输入：n = 3
输出：[
       "((()))",
       "(()())",
       "(())()",
       "()(())",
       "()()()"
     ]
```

- 解题思路

```
var res []string

func generateParenthesis(n int) []string {
        res = make([]string, 0)
        dfs(0, 0, n, "")
        return res
}

func dfs(left, right, max int, str string) {
        if left == right && left == max {
                res = append(res, str)
                return
        }
        if left < max {
                dfs(left+1, right, max, str+"(")
        }
        if right < left {
                dfs(left, right+1, max, str+")")
        }
}

#
/*
dp[i]表示n=i时括号的组合
dp[i]="(" + dp[j] + ")"+dp[i-j-1] (j<i)
dp[0] = ""
*/
```

(续下页)

(接上页)

```
func generateParenthesis(n int) []string {
        dp := make([][]string, n+1)
        dp[0] = make([]string, 0)
        if n == 0 {
                return dp[0]
        }
        dp[0] = append(dp[0], "")
        for i := 1; i <= n; i++ {
                dp[i] = make([]string, 0)
                for j := 0; j < i; j++ {
                        for _, a := range dp[j] {
                                for _, b := range dp[i-j-1] {
                                        str := "(" + a + ")" + b
                                        dp[i] = append(dp[i], str)
                                }
                        }
                }
        }
        return dp[n]
}

#
type Node struct {
        str   string
        left  int
        right int
}

func generateParenthesis(n int) []string {
        res := make([]string, 0)
        if n == 0 {
                return res
        }
        queue := make([]*Node, 0)
        queue = append(queue, &Node{
                str:   "",
                left:  n,
                right: n,
        })
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.left == 0 && node.right == 0 {
```

(续下页)

(接上页)

```
                        res = append(res, node.str)
                }
                if node.left > 0 {
                        queue = append(queue, &Node{
                                str:   node.str + "(",
                                left:  node.left - 1,
                                right: node.right,
                        })
                }
                if node.right > 0 && node.left < node.right {
                        queue = append(queue, &Node{
                                str:   node.str + ")",
                                left:  node.left,
                                right: node.right - 1,
                        })
                }
        }
        return res
}
```

2.14 24. 两两交换链表中的节点 (2)

- 题目

```
给定一个链表，两两交换其中相邻的节点，并返回交换后的链表。
你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。
示例:给定 1->2->3->4, 你应该返回 2->1->4->3.
```

- 解题思路

```
func swapPairs(head *ListNode) *ListNode {
        temp := &ListNode{Next: head}
        prev := temp
        for head != nil && head.Next != nil {
                first, second := head, head.Next
                prev.Next = second
                first.Next, second.Next = second.Next, first
                prev, head = first, first.Next
        }
        return temp.Next
}
```

(续下页)

(接上页)

```
#
func swapPairs(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        first, second := head, head.Next
        first.Next, second.Next = swapPairs(second.Next), first
        return second
}
```

2.15 29. 两数相除 (2)

- 题目

```
给定两个整数，被除数 dividend 和除数 divisor。将两数相除，要求不使用乘法、除法和 mod␣
↪运算符。
返回被除数 dividend 除以除数 divisor 得到的商。
整数除法的结果应当截去 (truncate) 其小数部分，
例如：truncate(8.345) = 8 以及 truncate(-2.7335) = -2
示例 1:输入：dividend = 10, divisor = 3输出：3
解释：10/3 = truncate(3.33333..) = truncate(3) = 3
示例 2:输入：dividend = 7, divisor = -3 输出：-2
解释：7/-3 = truncate(-2.33333..) = -2
提示：
    被除数和除数均为 32 位有符号整数。
    除数不为 0。
    假设我们的环境只能存储 32 位有符号整数，其数值范围是 [-2^31,  2^31 - 1]。
    本题中，如果除法结果溢出，则返回 2^31 - 1。
```

- 解题思路

```
func divide(dividend int, divisor int) int {
        if divisor == 0 || dividend == 0 {
                return 0
        }
        if divisor == 1 {
                return dividend
        }
        flag, count := 1, 1
        if dividend < 0 {
                flag = -flag
                dividend = -dividend
```

(续下页)

(接上页)

```
        }
        if divisor < 0 {
                flag = -flag
                divisor = -divisor
        }
        a, b, c := dividend, divisor, 0
        temp := b
        for a-b >= 0 {
                for a-b >= 0 {
                        a = a - b
                        c = c + count
                        b = b + b
                        count = count + count
                }
                b = temp
                count = 1
        }
        if c > math.MaxInt32 {
                return math.MaxInt32
        }
        if flag < 0 {
                return -c
        }
        return c
}

#
func divide(dividend int, divisor int) int {
        res := dividend / divisor
        if res > math.MaxInt32 {
                return math.MaxInt32
        }
        return res
}
```

2.16 31. 下一个排列 (2)

- 题目

```
实现获取下一个排列的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。
如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。
必须原地修改，只允许使用额外常数空间。
以下是一些例子，输入位于左侧列，其相应输出位于右侧列。
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1
```

- 解题思路

```
func nextPermutation(nums []int) {
        n := len(nums)
        left := n - 2
        // 以12385764为例，从后往前找到5<7 的升序情况，目标值为左边的数5
        for left >= 0 && nums[left] >= nums[left+1] {
                left--
        }
        if left == -1 {
                sort.Ints(nums)
                return
        }
        right := n - 1
        // 从后往前，找到第一个大于目标值的数，如6>5，然后交换
        for right >= 0 && nums[right] <= nums[left] {
                right--
        }
        nums[left], nums[right] = nums[right], nums[left]
        count := 0
        // 后面是降序状态，让它变为升序
        for i := left + 1; i <= (left+1+n-1)/2; i++ {
                nums[i], nums[n-1-count] = nums[n-1-count], nums[i]
                count++
        }
}

# 2
func nextPermutation(nums []int) {
        n := len(nums)
        left := n - 2
        // 以12385764为例，从后往前找到5<7 的升序情况，目标值为左边的数5
```

(续下页)

(接上页)

```
        for left >= 0 && nums[left] >= nums[left+1] {
                left--
        }
        if left >= 0 { // 存在升序的情况
                right := n - 1
                // 从后往前，找到第一个大于目标值的数，如6>5，然后交换
                for right >= 0 && nums[right] <= nums[left] {
                        right--
                }
                nums[left], nums[right] = nums[right], nums[left]
        }
        reverse(nums, left+1, n-1)
}

func reverse(nums []int, left, right int) {
        for left < right {
                nums[left], nums[right] = nums[right], nums[left]
                left++
                right--
        }
}
```

2.17 33. 搜索旋转排序数组 (2)

- 题目

```
假设按照升序排序的数组在预先未知的某个点上进行了旋转。
( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。
搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。
你可以假设数组中不存在重复的元素。
你的算法时间复杂度必须是 O(log n) 级别。
示例 1:输入: nums = [4,5,6,7,0,1,2], target = 0 输出: 4
示例 2:输入: nums = [4,5,6,7,0,1,2], target = 3 输出: -1
```

- 解题思路

```
func search(nums []int, target int) int {
        left, right := 0, len(nums)-1
        for left <= right {
                mid := left + (right-left)/2
                if nums[mid] == target {
                        return mid
```

(续下页)

(接上页)

```
                }
                if nums[left] <= nums[mid] {
                        if nums[left] <= target && target < nums[mid] {
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                } else {
                        if nums[mid] < target && target <= nums[right] {
                                left = mid + 1
                        } else {
                                right = mid - 1
                        }
                }
        }
        return -1
}

#
func search(nums []int, target int) int {
        for i := 0; i < len(nums); i++{
                if nums[i] == target{
                        return i
                }
        }
        return -1
}
```

2.18 34. 在排序数组中查找元素的第一个和最后一个位置 (4)

- 题目

```
给定一个按照升序排列的整数数组 nums，和一个目标值␣
↪target。找出给定目标值在数组中的开始位置和结束位置。
你的算法时间复杂度必须是 O(log n) 级别。
如果数组中不存在目标值，返回 [-1, -1]。
示例 1:输入: nums = [5,7,7,8,8,10], target = 8 输出: [3,4]
示例 2:输入: nums = [5,7,7,8,8,10], target = 6 输出: [-1,-1]
```

- 解题思路

```
func searchRange(nums []int, target int) []int {
        left := 0
        right := len(nums) - 1
        for left <= right {
                if nums[left] != target {
                        if target > nums[left] {
                                left++
                        }
                }
                if nums[right] != target {
                        if target < nums[right] {
                                right--
                        }
                }
                if left < len(nums) && right >= 0 &&
                        nums[left] == nums[right] && nums[left] == target {
                        break
                }
        }
        if right < left {
                return []int{-1, -1}
        }
        return []int{left, right}
}

#
func searchRange(nums []int, target int) []int {
        left := -1
        right := -1
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        right = i
                } else if nums[i] > target {
                        break
                }
        }
        for i := len(nums) - 1; i >= 0; i-- {
                if nums[i] == target {
                        left = i
                } else if nums[i] < target {
                        break
                }
        }
        return []int{left, right}
```

(续下页)

(接上页)

```
}

# 3
func searchRange(nums []int, target int) []int {
        if len(nums) == 0 || nums[0] > target || nums[len(nums)-1] < target {
                return []int{-1, -1}
        }
        left := leftSearch(nums, target)
        right := rightSearch(nums, target)
        return []int{left, right}
}

func leftSearch(nums []int, target int) int {
        left, right := 0, len(nums)-1
        for left <= right {
                mid := left + (right-left)/2
                if target > nums[mid] {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        if left < len(nums) && nums[left] == target {
                return left
        }
        return -1
}

func rightSearch(nums []int, target int) int {
        left, right := 0, len(nums)-1
        for left <= right {
                mid := left + (right-left)/2
                if target < nums[mid] {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        if right >= 0 && nums[right] == target {
                return right
        }
        return -1
}
```

(续下页)

(接上页)

```
#
func searchRange(nums []int, target int) []int {
        left := -1
        right := -1
        for i, j := 0, len(nums)-1; i <= j; {
                mid := i + (j-i)/2
                if nums[mid] < target {
                        i = mid + 1
                } else if nums[mid] > target {
                        j = mid - 1
                } else {
                        for temp := mid; temp >= 0; temp-- {
                                if target == nums[temp] {
                                        left = temp
                                } else {
                                        break
                                }
                        }
                        for temp := mid; temp < len(nums); temp++ {
                                if target == nums[temp] {
                                        right = temp
                                } else {
                                        break
                                }
                        }
                        break
                }
        }
        return []int{left, right}
}
```

2.19 36. 有效的数独 (1)

- 题目

```
判断一个 9x9 的数独是否有效。只需要根据以下规则，验证已经填入的数字是否有效即可。
    数字 1-9 在每一行只能出现一次。
    数字 1-9 在每一列只能出现一次。
    数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。
上图是一个部分填充的有效的数独。
数独部分空格内已填入了数字，空白格用 '.' 表示。
```

(续下页)

(接上页)

```
示例 1:输入:
[
  ["5","3",".",".","7",".",".",".",".","."],
  ["6",".",".","1","9","5",".",".",".","."],
  [".","9","8",".",".",".",".","6","."],
  ["8",".",".",".","6",".",".",".",".","3"],
  ["4",".",".","8",".","3",".",".",".","1"],
  ["7",".",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".",".","2","8","."],
  [".",".",".","4","1","9",".",".",".","5"],
  [".",".",".",".",".","8",".",".","7","9"]
]
输出: true
示例 2:输入:
[
  ["8","3",".",".","7",".",".",".",".","."],
  ["6",".",".","1","9","5",".",".",".","."],
  [".","9","8",".",".",".",".","6","."],
  ["8",".",".",".","6",".",".",".",".","3"],
  ["4",".",".","8",".","3",".",".",".","1"],
  ["7",".",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".",".","2","8","."],
  [".",".",".","4","1","9",".",".",".","5"],
  [".",".",".",".",".","8",".",".","7","9"]
]
输出: false
解释: 除了第一行的第一个数字从 5 改为 8 以外, 空格内其他数字均与 示例1 相同。
     但由于位于左上角的 3x3 宫内有两个 8 存在, 因此这个数独是无效的。
说明:
    一个有效的数独(部分已被填充)不一定是可解的。
    只需要根据以上规则, 验证已经填入的数字是否有效即可。
    给定数独序列只包含数字 1-9 和字符 '.' 。
    给定数独永远是 9x9 形式的。
```

- 解题思路

```
func isValidSudoku(board [][]byte) bool {
        var row, col, arr [9][9]int
        for i := 0; i < 9; i++ {
                for j := 0; j < 9; j++ {
                        if board[i][j] != '.' {
                                num := board[i][j] - '1'
                                index := (i/3)*3 + j/3
                                if row[i][num] == 1 || col[j][num] == 1 ||␣
```

(续下页)

(接上页)

```
↪arr[index][num] == 1 {
                                        return false
                                    }
                                    row[i][num] = 1
                                    col[j][num] = 1
                                    arr[index][num] = 1
                                }
                            }
                        }
                        return true
}
```

2.20 39. 组合总和 (2)

- 题目

```
给定一个无重复元素的数组 candidates 和一个目标数 target ，
找出 candidates 中所有可以使数字和为 target 的组合。
candidates 中的数字可以无限制重复被选取。
说明：
    所有数字（包括 target）都是正整数。
    解集不能包含重复的组合。
示例 1:输入: candidates = [2,3,6,7], target = 7, 所求解集为:
[
  [7],
  [2,2,3]
]
示例 2:输入: candidates = [2,3,5], target = 8, 所求解集为:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

- 解题思路

```
var res [][]int

func combinationSum(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
```

(续下页)

(接上页)

```
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        if target < 0{
                return
        }
        for i := index; i < len(candidates); i++ {
                arr = append(arr, candidates[i])
                dfs(candidates, target-candidates[i], arr, i)
                arr = arr[:len(arr)-1]
        }
}

#
var res [][]int

func combinationSum(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(candidates); i++ {
                if target < candidates[i] {
                        return
                }
                dfs(candidates, target-candidates[i], append(arr, candidates[i]), i)
        }
}
```

2.21 40. 组合总和 II(2)

- 题目

```
给定一个数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为
↪target 的组合。
candidates 中的每个数字在每个组合中只能使用一次。
说明：
    所有数字（包括目标数）都是正整数。
    解集不能包含重复的组合。
示例 1:输入: candidates = [10,1,2,7,6,1,5], target = 8,
所求解集为:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
示例 2:输入: candidates = [2,5,2,1,2], target = 5,
所求解集为:
[
  [1,2,2],
  [5]
]
```

- 解题思路

```
var res [][]int

func combinationSum2(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        if target < 0 {
```

(续下页)

(接上页)

```
                return
        }
        for i := index; i < len(candidates); i++ {
                origin := i
                for i < len(candidates)-1 && candidates[i] == candidates[i+1] {
                        i++
                }
                arr = append(arr, candidates[i])
                dfs(candidates, target-candidates[i], arr, origin+1)
                arr = arr[:len(arr)-1]
        }
}

#
var res [][]int

func combinationSum2(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(candidates); i++ {
                if i != index && candidates[i] == candidates[i-1] {
                        continue
                }
                if target < 0 {
                        return
                }
                arr = append(arr, candidates[i])
                dfs(candidates, target-candidates[i], arr, i+1)
                arr = arr[:len(arr)-1]
        }
}
```

2.22 43. 字符串相乘 (1)

- 题目

```
给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2 的乘积，
它们的乘积也表示为字符串形式。
示例 1:输入: num1 = "2", num2 = "3"输出: "6"
示例 2:输入: num1 = "123", num2 = "456"输出: "56088"
说明:
    num1 和 num2 的长度小于110。
    num1 和 num2 只包含数字 0-9。
    num1 和 num2 均不以零开头，除非是数字 0 本身。
    不能使用任何标准库的大数类型（比如 BigInteger）或直接将输入转换为整数来处理。
```

- 解题思路

```
func multiply(num1 string, num2 string) string {
        if num1 == "0" || num2 == "0" {
                return "0"
        }
        arr := make([]int, len(num1)+len(num2))
        for i := len(num1) - 1; i >= 0; i-- {
                a := int(num1[i] - '0')
                for j := len(num2) - 1; j >= 0; j-- {
                        b := int(num2[j] - '0')
                        value := a*b + arr[i+j+1]
                        arr[i+j+1] = value % 10
                        arr[i+j] = value/10 + arr[i+j]
                }
        }
        res := ""
        for i := 0; i < len(arr); i++ {
                if i == 0 && arr[i] == 0 {
                        continue
                }
                res = res + string(arr[i]+'0')
        }
        return res
}

# 2
func multiply(num1 string, num2 string) string {
        a, b := new(big.Int), new(big.Int)
        a.SetString(num1, 10)
```

(续下页)

(接上页)

```
        b.SetString(num2, 10)
        a.Mul(a, b)
        return a.String()
}
```

2.23 46. 全排列 (3)

- 题目

```
给定一个 没有重复 数字的序列，返回其所有可能的全排列。
示例:输入: [1,2,3]输出:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

- 解题思路

```
var res [][]int

func permute(nums []int) [][]int {
        res = make([][]int, 0)
        arr := make([]int, 0)
        visited := make(map[int]bool)
        dfs(nums, 0, arr, visited)
        return res
}

func dfs(nums []int, index int, arr []int, visited map[int]bool) {
        if index == len(nums) {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == false {
                        arr = append(arr, nums[i])
```

(续下页)

(接上页)

```
                        visited[i] = true
                        dfs(nums, index+1, arr, visited)
                        arr = arr[:len(arr)-1]
                        visited[i] = false
                }
        }
}

#
func permute(nums []int) [][]int {
        if len(nums) == 1 {
                return [][]int{nums}
        }
        res := make([][]int, 0)
        for i := 0; i < len(nums); i++ {
                tempArr := make([]int, len(nums)-1)
                copy(tempArr[0:], nums[:i])
                copy(tempArr[i:], nums[i+1:])
                arr := permute(tempArr)
                for _, v := range arr {
                        res = append(res, append(v, nums[i]))
                }
        }
        return res
}

#
var res [][]int

func permute(nums []int) [][]int {
        res = make([][]int, 0)
        arr := make([]int, len(nums))
        dfs(nums, 0, arr)
        return res
}

func dfs(nums []int, index int, arr []int) {
        if index == len(nums) {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
```

(续下页)

(接上页)

```
        for i := index; i < len(nums); i++ {
                arr[index] = nums[i]
                nums[i], nums[index] = nums[index], nums[i]
                dfs(nums, index+1, arr)
                nums[i], nums[index] = nums[index], nums[i]
        }
}
```

2.24 47. 全排列 II(3)

- 题目

```
给定一个可包含重复数字的序列，返回所有不重复的全排列。
示例:输入: [1,1,2] 输出:
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]
```

- 解题思路

```
var res [][]int

func permuteUnique(nums []int) [][]int {
        res = make([][]int, 0)
        sort.Ints(nums)
        dfs(nums, 0, make([]int, len(nums)), make([]int, 0))
        return res
}

func dfs(nums []int, index int, visited []int, arr []int) {
        if len(nums) == index {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == 1 {
                        continue
                }
```

(续下页)

(接上页)

```
                // visited[i-1] == 0 或者 visited[i-1] == 1都可以
                if i > 0 && nums[i] == nums[i-1] && visited[i-1] == 0 {
                        // if i > 0 && nums[i] == nums[i-1] && visited[i-1] == 1 {
                        continue
                }
                arr = append(arr, nums[i])
                visited[i] = 1
                dfs(nums, index+1, visited, arr)
                visited[i] = 0
                arr = arr[:len(arr)-1]
        }
}

# 2
var res [][]int

func permuteUnique(nums []int) [][]int {
        res = make([][]int, 0)
        sort.Ints(nums)
        dfs(nums, 0)
        return res
}

func dfs(nums []int, index int) {
        if index == len(nums) {
                temp := make([]int, len(nums))
                copy(temp, nums)
                res = append(res, temp)
                return
        }
        m := make(map[int]int)
        for i := index; i < len(nums); i++ {
                if _, ok := m[nums[i]]; ok {
                        continue
                }
                m[nums[i]] = 1
                nums[i], nums[index] = nums[index], nums[i]
                dfs(nums, index+1)
                nums[i], nums[index] = nums[index], nums[i]
        }
}

# 3
```

(续下页)

(接上页)

```
var res [][]int

func permuteUnique(nums []int) [][]int {
        res = make([][]int, 0)
        sort.Ints(nums)
        dfs(nums, make([]int, 0))
        return res
}

func dfs(nums []int, arr []int) {
        if len(nums) == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(nums); i++ {
                if i != 0 && nums[i] == nums[i-1] {
                        continue
                }
                tempArr := make([]int, len(nums))
                copy(tempArr, nums)
                arr = append(arr, nums[i])
                dfs(append(tempArr[:i], tempArr[i+1:]...), arr)
                arr = arr[:len(arr)-1]
        }
}
```

2.25 48. 旋转图像 (3)

- 题目

```
给定一个 n × n 的二维矩阵表示一个图像。
将图像顺时针旋转 90 度。
说明：你必须在原地旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。
示例 1:给定 matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],
原地旋转输入矩阵，使其变为:
```

(续下页)

(接上页)

```
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
示例 2:给定 matrix =
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],
原地旋转输入矩阵，使其变为:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

- 解题思路

```
func rotate(matrix [][]int) {
        n := len(matrix)
        // 同行逆置
        // [[1 2 3] [4 5 6] [7 8 9]]
        // [[3 2 1] [6 5 4] [9 8 7]]
        for i := 0; i < n; i++ {
                for j := 0; j < n/2; j++ {
                        matrix[i][j], matrix[i][n-1-j] = matrix[i][n-1-j],␣
↪matrix[i][j]
                }
        }
        // 左下右上对角线对互换
        // [[3 2 1] [6 5 4] [9 8 7]]
        // [[7 4 1] [8 5 2] [9 6 3]]
        for i := 0; i < n-1; i++ {
                for j := 0; j < n-1-i; j++ {
                        matrix[i][j], matrix[n-1-j][n-1-i] = matrix[n-1-j][n-1-i],␣
↪matrix[i][j]
                }
        }
}
```

(续下页)

(接上页)

```
# 2
func rotate(matrix [][]int) {
        n := len(matrix)
        for start, end := 0, n-1; start < end; {
                for s, e := start, end; s < end; {
                        matrix[start][s], matrix[e][start], matrix[end][e],␣
↪matrix[s][end] =
                                matrix[e][start], matrix[end][e], matrix[s][end],␣
↪matrix[start][s]
                        s++
                        e--
                }
                start++
                end--
        }
}

# 3
func rotate(matrix [][]int) {
        n := len(matrix)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        arr[j][n-1-i] = matrix[i][j]
                }
        }
        copy(matrix, arr)
}
```

2.26 49. 字母异位词分组 (2)

- 题目

```
给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。
示例:输入: ["eat", "tea", "tan", "ate", "nat", "bat"]
输出:
[
  ["ate","eat","tea"],
  ["nat","tan"],
```

(续下页)

(接上页)

```
  ["bat"]
]
说明：
    所有输入均为小写字母。
    不考虑答案输出的顺序。
```

- 解题思路

```
func groupAnagrams(strs []string) [][]string {
        m := make(map[string]int)
        res := make([][]string, 0)
        for i := 0; i < len(strs); i++ {
                arr := []byte(strs[i])
                sort.Slice(arr, func(i, j int) bool {
                        return arr[i] < arr[j]
                })
                newStr := string(arr)
                if _, ok := m[newStr]; ok {
                        res[m[newStr]] = append(res[m[newStr]], strs[i])
                } else {
                        m[newStr] = len(res)
                        res = append(res, []string{strs[i]})
                }
        }
        return res
}

#
func groupAnagrams(strs []string) [][]string {
        m := make(map[[26]int]int)
        res := make([][]string, 0)
        for i := 0; i < len(strs); i++ {
                arr := [26]int{}
                for j := 0; j < len(strs[i]); j++{
                        arr[strs[i][j]-'a']++
                }
                if _, ok := m[arr]; ok {
                        res[m[arr]] = append(res[m[arr]], strs[i])
                } else {
                        m[arr] = len(res)
                        res = append(res, []string{strs[i]})
                }
        }
        return res
```

(续下页)

(接上页)

```
}
```

2.27 50.Pow(x,n)(4)

- 题目

```
实现 pow(x, n) ，即计算 x 的 n 次幂函数。
示例 1:输入: 2.00000, 10 输出: 1024.00000
示例 2:输入: 2.10000, 3 输出: 9.26100
示例 3:输入: 2.00000, -2 输出: 0.25000
解释: 2-2 = 1/22 = 1/4 = 0.25
说明:  -100.0 < x < 100.0
    n 是 32 位有符号整数，其数值范围是 [-231, 231 - 1] 。
```

- 解题思路

```
func myPow(x float64, n int) float64 {
        if n == 0 {
                return 1
        }
        if n < 0 {
                return 1 / myPow(x, -n)
        }
        if n%2 == 1 {
                return x * myPow(x, n-1)
        }
        return myPow(x*x, n/2)
}

#
func myPow(x float64, n int) float64 {
        if n < 0 {
                x = 1 / x
                n = -n
        }
        res := float64(1)
        for n > 0 {
                if n%2 == 1 {
                        res = res * x
                }
                x = x * x
                n = n / 2
```

(续下页)

(接上页)

```
        }
        return res
}

#
func myPow(x float64, n int) float64 {
        return math.Pow(x, float64(n))
}

#
func myPow(x float64, n int) float64 {
        if n == 0 {
                return 1
        }
        if n == 1 {
                return x
        }
        res := 1.0
        if n > 0 {
                res = myPow(x, n/2)
                return res * res * myPow(x, n%2)
        } else {
                res = myPow(x, -n/2)
                res = res * res * myPow(x, -n%2)
                return 1 / res
        }
}
```

2.28 54. 螺旋矩阵 (2)

- 题目

```
给定一个包含 m x n 个元素的矩阵（m 行，n␣
↪列），请按照顺时针螺旋顺序，返回矩阵中的所有元素。
示例 1:输入:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
输出: [1,2,3,6,9,8,7,4,5]
示例 2:输入:
```

(续下页)

(接上页)

```
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
输出: [1,2,3,4,8,12,11,10,9,5,6,7]
```

- 解题思路

```
var res []int

func spiralOrder(matrix [][]int) []int {
        res = make([]int, 0)
        rows := len(matrix)
        if rows == 0 {
                return res
        }
        cols := len(matrix[0])
        if cols == 0 {
                return res
        }
        start := 0
        for cols > start*2 && rows > start*2 {
                printCircle(matrix, cols, rows, start)
                start++
        }
        return res
}

func printCircle(matrix [][]int, cols, rows, start int) {
        x := cols - 1 - start
        y := rows - 1 - start
        // 左到右
        for i := start; i <= x; i++ {
                res = append(res, matrix[start][i])
        }
        // 上到下
        if start < y {
                for i := start + 1; i <= y; i++ {
                        res = append(res, matrix[i][x])
                }
        }
        // 右到左
        if start < x && start < y {
```

(续下页)

(接上页)

```
                for i := x - 1; i >= start; i-- {
                        res = append(res, matrix[y][i])
                }
        }
        // 下到上
        if start < x && start < y-1 {
                for i := y - 1; i >= start+1; i-- {
                        res = append(res, matrix[i][start])
                }
        }
}

#
func spiralOrder(matrix [][]int) []int {
        res := make([]int, 0)
        rows := len(matrix)
        if rows == 0 {
                return res
        }
        cols := len(matrix[0])
        if cols == 0 {
                return res
        }
        x1, x2, y1, y2 := 0, rows-1, 0, cols-1
        direct := 0
        for x1 <= x2 && y1 <= y2 {
                direct = (direct + 4) % 4
                if direct == 0 {
                        for i := y1; i <= y2; i++ {
                                res = append(res, matrix[x1][i])
                        }
                        x1++
                } else if direct == 1 {
                        for i := x1; i <= x2; i++ {
                                res = append(res, matrix[i][y2])
                        }
                        y2--
                } else if direct == 2 {
                        for i := y2; i >= y1; i-- {
                                res = append(res, matrix[x2][i])
                        }
                        x2--
                } else if direct == 3 {
```

(续下页)

(接上页)

```
                        for i := x2; i >= x1; i-- {
                                res = append(res, matrix[i][y1])
                        }
                        y1++
                }
                direct++
        }
        return res
}
```

2.29 55. 跳跃游戏 (4)

- 题目

```
给定一个非负整数数组，你最初位于数组的第一个位置。
数组中的每个元素代表你在该位置可以跳跃的最大长度。
判断你是否能够到达最后一个位置。
示例 1:输入: [2,3,1,1,4] 输出: true
解释: 我们可以先跳 1 步，从位置 0 到达 位置 1, 然后再从位置 1 跳 3␣
↪步到达最后一个位置。
示例 2:输入: [3,2,1,0,4] 输出: false
解释: 无论怎样，你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0 ，
所以你永远不可能到达最后一个位置。
```

- 解题思路

```
func canJump(nums []int) bool {
        j := len(nums) - 1
        for i := len(nums) - 2; i >= 0; i-- {
                if nums[i]+i >= j {
                        j = i
                }
        }
        return j <= 0
}

#
func canJump(nums []int) bool {
        if len(nums) <= 1 {
                return true
        }
        dp := make([]bool, len(nums))
```

(续下页)

(接上页)

```
        dp[0] = true
        for i := 1; i < len(nums); i++ {
                flag := false
                for j := 0; j < i; j++ {
                        if dp[j] && nums[j]+j >= i {
                                flag = true
                                break
                        }
                }
                dp[i] = flag
        }
        return dp[len(nums)-1]
}

#
func canJump(nums []int) bool {
        max := 0
        for i := 0; i < len(nums); i++ {
                if i <= max {
                        if i+nums[i] > max {
                                max = i + nums[i]
                        }
                        if max >= len(nums)-1 {
                                return true
                        }
                }
        }
        return false
}

#
func canJump(nums []int) bool {
        zero := -1
        for i := len(nums) - 2; i >= 0; i-- {
                if zero > 0 {
                        if i+nums[i] > zero {
                                zero = -1
                        }
                        continue
                }
                if nums[i] == 0 {
                        zero = i
                        continue
```

(续下页)

(接上页)

```
                }
        }
        return zero < 0
}
```

2.30 56. 合并区间 (2)

- 题目

```
给出一个区间的集合，请合并所有重叠的区间。
示例 1:输入: [[1,3],[2,6],[8,10],[15,18]] 输出: [[1,6],[8,10],[15,18]]
解释: 区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].
示例 2:输入: [[1,4],[4,5]] 输出: [[1,5]]
解释: 区间 [1,4] 和 [4,5] 可被视为重叠区间。
```

- 解题思路

```
func merge(intervals [][]int) [][]int {
        res := make([][]int, 0)
        if len(intervals) == 0 {
                return nil
        }
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][0] < intervals[j][0]
        })
        res = append(res, intervals[0])
        for i := 1; i < len(intervals); i++ {
                arr := res[len(res)-1]
                if intervals[i][0] > arr[1] {
                        res = append(res, intervals[i])
                } else if intervals[i][1] > arr[1] {
                        res[len(res)-1][1] = intervals[i][1]
                }
        }
        return res
}

#
func merge(intervals [][]int) [][]int {
        res := make([][]int, 0)
        if len(intervals) == 0 {
                return nil
```

(续下页)

(接上页)

```
        }
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][0] < intervals[j][0]
        })
        for i := 0; i < len(intervals); {
                end := intervals[i][1]
                j := i + 1
                for j < len(intervals) && intervals[j][0] <= end {
                        if intervals[j][1] > end {
                                end = intervals[j][1]
                        }
                        j++
                }
                res = append(res, []int{intervals[i][0], end})
                i = j
        }
        return res
}
```

2.31 59. 螺旋矩阵 II(2)

- 题目

```
给定一个正整数 n，生成一个包含 1 到 n2␣
↪所有元素，且元素按顺时针顺序螺旋排列的正方形矩阵。
示例:输入: 3 输出:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

- 解题思路

```
func generateMatrix(n int) [][]int {
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                res[i] = make([]int, n)
        }
        count := 1
        level := 1
        for count <= n*n {
```

(续下页)

(接上页)

```
                top, bottom, left, right := level-1, n-level, level-1, n-level
                // 左到右
                for i := left; i <= right && left <= right; i++ {
                        res[top][i] = count
                        count++
                }
                // 上到下
                for i := top + 1; i <= bottom && top <= bottom; i++ {
                        res[i][right] = count
                        count++
                }
                // 右到左
                for i := right - 1; i >= left && left <= right; i-- {
                        res[bottom][i] = count
                        count++
                }
                // 下到上
                for i := bottom - 1; i >= top+1 && top <= bottom; i-- {
                        res[i][left] = count
                        count++
                }
                level++
        }
        return res
}

#
func generateMatrix(n int) [][]int {
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                res[i] = make([]int, n)
        }
        count := 1
        top, bottom, left, right := 0, n-1, 0, n-1
        for count <= n*n {
                for i := left; i <= right; i++ {
                        res[top][i] = count
                        count++
                }
                top++
                for i := top; i <= bottom; i++ {
                        res[i][right] = count
                        count++
```

(续下页)

(接上页)

```
                }
                right--
                for i := right; i >= left; i-- {
                        res[bottom][i] = count
                        count++
                }
                bottom--
                for i := bottom; i >= top; i-- {
                        res[i][left] = count
                        count++
                }
                left++
        }
        return res
}
```

2.32 60. 第 k 个排列 (1)

- 题目

```
给出集合 [1,2,3,…,n]，其所有元素共有 n! 种排列。
按大小顺序列出所有排列情况，并一一标记，当 n = 3 时，所有排列如下：
    "123"
    "132"
    "213"
    "231"
    "312"
    "321"
给定 n 和 k，返回第 k 个排列。
说明：
    给定 n 的范围是 [1, 9]。
    给定 k 的范围是[1,  n!]。
示例 1:输入: n = 3, k = 3 输出: "213"
示例 2:输入: n = 4, k = 9 输出: "2314"
```

- 解题思路

```
func getPermutation(n int, k int) string {
        res := ""
        arr := []string{"1", "2", "3", "4", "5", "6", "7", "8", "9"}
        times := make([]int, 0)
        times = append(times, 1)
```

(续下页)

(接上页)

```
        value := 1
        for i := 1; i <= 9; i++ {
                times = append(times, value*i)
                value = value * i
        }
        k--
        for n > 0 {
                i := k / times[n-1]
                k = k % times[n-1]
                n--
                res = res + arr[i]
                arr = append(arr[:i], arr[i+1:]...)
        }
        return res
}
```

2.33 61. 旋转链表 (2)

- 题目

```
给定一个链表，旋转链表，将链表每个节点向右移动 k 个位置，其中 k 是非负数。
示例 1:
输入: 1->2->3->4->5->NULL, k = 2
输出: 4->5->1->2->3->NULL
解释:
向右旋转 1 步: 5->1->2->3->4->NULL
向右旋转 2 步: 4->5->1->2->3->NULL
示例 2:
输入: 0->1->2->NULL, k = 4
输出: 2->0->1->NULL
解释:
向右旋转 1 步: 2->0->1->NULL
向右旋转 2 步: 1->2->0->NULL
向右旋转 3 步: 0->1->2->NULL
向右旋转 4 步: 2->0->1->NULL
```

- 解题思路

```
func rotateRight(head *ListNode, k int) *ListNode {
        if head == nil || k == 0 {
                return head
        }
```

(续下页)

(接上页)

```
        temp := head
        count := 1
        for temp.Next != nil {
                temp = temp.Next
                count++
        }
        temp.Next = head
        k = k % count
        for i := 0; i < count-k; i++ {
                temp = temp.Next
        }
        head, temp.Next = temp.Next, nil
        return head
}

#
func rotateRight(head *ListNode, k int) *ListNode {
        if head == nil || k == 0 {
                return head
        }
        temp := head
        count := 0
        arr := make([]*ListNode, 0)
        for temp != nil {
                arr = append(arr, temp)
                temp = temp.Next
                count++
        }
        k = k % count
        if k == 0 {
                return head
        }
        arr[count-1].Next = head
        temp = arr[count-1-k]
        head, temp.Next = temp.Next, nil
        return head
}
```

2.34 62. 不同路径 (4)

- 题目

```
一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。
机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。
问总共有多少条不同的路径？
例如，上图是一个7 x 3 的网格。有多少可能的路径？
示例 1:输入: m = 3, n = 2 输出: 3
解释:
从左上角开始，总共有 3 条路径可以到达右下角。
1. 向右 -> 向右 -> 向下
2. 向右 -> 向下 -> 向右
3. 向下 -> 向右 -> 向右
示例 2:输入: m = 7, n = 3 输出: 28
提示：
    1 <= m, n <= 100
    题目数据保证答案小于等于 2 * 10 ^ 9
```

- 解题思路

```
// dp[i][j] = dp[i-1][j] + dp[i][j-1]
func uniquePaths(m int, n int) int {
        if m <= 0 || n <= 0 {
                return 0
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                dp[i][0] = 1
        }
        for i := 0; i < m; i++ {
                dp[0][i] = 1
        }
        for i := 1; i < n; i++ {
                for j := 1; j < m; j++ {
                        dp[i][j] = dp[i-1][j] + dp[i][j-1]
                }
        }
        return dp[n-1][m-1]
}

#
// dp[i]= dp[i-1] + dp[i]
```

(续下页)

(接上页)

```
func uniquePaths(m int, n int) int {
        if m <= 0 || n <= 0 {
                return 0
        }
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = 1
        }
        for i := 1; i < m; i++ {
                for j := 1; j < n; j++ {
                        dp[j] = dp[j] + dp[j-1]
                }
        }
        return dp[n-1]
}

# 3
func uniquePaths(m int, n int) int {
        if m == 1 || n == 1 {
                return 1
        }
        if m > n {
                m, n = n, m
        }
        a := 1
        for i := 1; i <= m-1; i++ {
                a = a * i
        }
        b := 1
        for i := n; i <= m+n-2; i++ {
                b = b * i
        }
        return b / a
}

# 4
var arr [][]int

func uniquePaths(m int, n int) int {
        arr = make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
```

(续下页)

(接上页)

```
        return dfs(m, n)
}

func dfs(m, n int) int {
        if m <= 0 || n <= 0 {
                return 0
        }
        if m == 1 || n == 1 {
                return 1
        }
        if arr[n][m] > 0 {
                return arr[n][m]
        }
        arr[n][m] = dfs(m, n-1) + dfs(m-1, n)
        return arr[n][m]
}
```

2.35 63. 不同路径 II(3)

- 题目

```
一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。
机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。
现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？
网格中的障碍物和空位置分别用 1 和 0 来表示。
说明：m 和 n 的值均不超过 100。
示例 1:
输入:
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
输出: 2
解释:
3x3 网格的正中间有一个障碍物。
从左上角到右下角一共有 2 条不同的路径：
1. 向右 -> 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右 -> 向右
```

- 解题思路

```
func uniquePathsWithObstacles(obstacleGrid [][]int) int {
        n := len(obstacleGrid)
        if n < 1 {
                return 0
        }
        m := len(obstacleGrid[0])
        if m < 1 {
                return 0
        }
        if obstacleGrid[0][0] == 1 {
                return 0
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        if i == 0 && j == 0 {
                                dp[i][j] = 1
                        } else if i == 0 && j != 0 {
                                if obstacleGrid[i][j] == 0 {
                                        dp[i][j] = dp[i][j-1]
                                }
                        } else if i != 0 && j == 0 {
                                if obstacleGrid[i][j] == 0 {
                                        dp[i][j] = dp[i-1][j]
                                }
                        } else {
                                if obstacleGrid[i][j] == 0 {
                                        dp[i][j] = dp[i-1][j] + dp[i][j-1]
                                }
                        }
                }
        }
        return dp[n-1][m-1]
}

# 2
// dp[j] = dp[j] + dp[j-1]
func uniquePathsWithObstacles(obstacleGrid [][]int) int {
        n := len(obstacleGrid)
        if n < 1 {
                return 0
        }
        m := len(obstacleGrid[0])
```

(续下页)

(接上页)

```
        if m < 1 {
                return 0
        }
        if obstacleGrid[0][0] == 1 {
                return 0
        }
        dp := make([]int, m)
        dp[0] = 1
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if obstacleGrid[i][j] == 1 {
                                dp[j] = 0
                                continue
                        }
                        if j >= 1 && obstacleGrid[i][j-1] == 0 {
                                dp[j] = dp[j] + dp[j-1]
                        }
                }
        }
        return dp[m-1]
}

# 3
func uniquePathsWithObstacles(obstacleGrid [][]int) int {
        n := len(obstacleGrid)
        if n < 1 {
                return 0
        }
        m := len(obstacleGrid[0])
        if m < 1 {
                return 0
        }
        if obstacleGrid[0][0] == 1 {
                return 0
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if obstacleGrid[i][j] == 1 {
                                obstacleGrid[i][j] = 0
                                continue
                        }
                        if i == 0 {
                                if j == 0 {
```

(续下页)

(接上页)

```
                                        obstacleGrid[i][j] = 1
                                } else {
                                        obstacleGrid[i][j] += obstacleGrid[i][j-1]
                                }
                        } else {
                                if j == 0 {
                                        obstacleGrid[i][j] += obstacleGrid[i-1][j]
                                } else {
                                        obstacleGrid[i][j] += obstacleGrid[i][j-1] +␣
↪obstacleGrid[i-1][j]
                                }
                        }
                }
        }
        return obstacleGrid[n-1][m-1]
}
```

2.36 64. 最小路径和 (4)

- 题目

```
给定一个包含非负整数的 m x n␣
↪网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。
说明：每次只能向下或者向右移动一步。
示例:
输入:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
输出: 7解释: 因为路径 1→3→1→1→1 的总和最小。
```

- 解题思路

```
func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        dp := make([][]int, n)
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        dp[0][0] = grid[0][0]
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i == 0 && j != 0 {
                                dp[i][j] = dp[i][j-1] + grid[i][j]
                        } else if i != 0 && j == 0 {
                                dp[i][j] = dp[i-1][j] + grid[i][j]
                        } else if i != 0 && j != 0 {
                                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
                        }
                }
        }
        return dp[n-1][m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i == 0 && j != 0 {
                                grid[i][j] = grid[i][j-1] + grid[i][j]
                        } else if i != 0 && j == 0 {
                                grid[i][j] = grid[i-1][j] + grid[i][j]
                        } else if i != 0 && j != 0 {
                                grid[i][j] = min(grid[i-1][j], grid[i][j-1]) +␣
↪grid[i][j]
                        }
                }
```

(续下页)

(接上页)

```
        }
        return grid[n-1][m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        dp := make([]int, m)
        dp[0] = grid[0][0]

        for i := 1; i < m; i++ {
                dp[i] = dp[i-1] + grid[0][i]
        }
        for i := 1; i < n; i++ {
                dp[0] = dp[0] + grid[i][0]
                for j := 1; j < m; j++ {
                        dp[j] = min(dp[j-1], dp[j]) + grid[i][j]
                }
        }
        return dp[m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
var arr [][]int
```

(续下页)

(接上页)

```
func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        return dfs(grid, n-1, m-1)
}

func dfs(grid [][]int, n, m int) int {
        if m == 0 && n == 0 {
                arr[0][0] = grid[0][0]
                return grid[0][0]
        }
        if n == 0 {
                return grid[0][m] + dfs(grid, 0, m-1)
        }
        if m == 0 {
                return grid[n][0] + dfs(grid, n-1, 0)
        }
        if arr[n][m] > 0 {
                return arr[n][m]
        }
        arr[n][m] = min(dfs(grid, n-1, m), dfs(grid, n, m-1)) + grid[n][m]
        return arr[n][m]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

2.37 71. 简化路径 (2)

- 题目

```
以 Unix 风格给出一个文件的绝对路径，你需要简化它。或者换句话说，将其转换为规范路径。
在 Unix 风格的文件系统中，一个点 (.) 表示当前目录本身；
此外，两个点 (..) ␣
↪表示将目录切换到上一级（指向父目录）；两者都可以是复杂相对路径的组成部分。
更多信息请参阅：Linux / Unix中的绝对路径 vs 相对路径
请注意，返回的规范路径必须始终以斜杠 / 开头，并且两个目录名之间必须只有一个斜杠 /。
最后一个目录名（如果存在）不能以 /␣
↪结尾。此外，规范路径必须是表示绝对路径的最短字符串。
示例 1: 输入："/home/" 输出："/home"
解释：注意，最后一个目录名后面没有斜杠。
示例 2: 输入："/../" 输出："/"
解释：从根目录向上一级是不可行的，因为根是你可以到达的最高级。
示例 3: 输入："/home//foo/" 输出："/home/foo"
解释：在规范路径中，多个连续斜杠需要用一个斜杠替换。
示例 4: 输入："/a/./b/../../c/" 输出："/c"
示例 5: 输入："/a/../../b/../c//.//" 输出："/c"
示例 6: 输入："/a//b////c/d//././/.." 输出："/a/b/c"
```

- 解题思路

```
func simplifyPath(path string) string {
        stack := make([]string, 0)
        arr := strings.Split(path, "/")
        for i := 0; i < len(arr); i++ {
                if arr[i] == "." || arr[i] == "" {
                        continue
                }
                if arr[i] == ".." {
                        if len(stack) > 0 {
                                stack = stack[:len(stack)-1]
                        }
                } else {
                        stack = append(stack, arr[i])
                }
        }
        return "/" + strings.Join(stack, "/")
}

#
func simplifyPath(path string) string {
```

(续下页)

(接上页)

```
        return filepath.Clean(path)
}
```

2.38 73. 矩阵置零 (4)

- 题目

```
给定一个 m x n 的矩阵，如果一个元素为 0，则将其所在行和列的所有元素都设为␣
↪0。请使用原地算法。
示例 1:
输入:
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
输出:
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
示例 2:
输入:
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
输出:
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
进阶:
    一个直接的解决方案是使用  O(mn) 的额外空间，但这并不是一个好的解决方案。
    一个简单的改进方案是使用 O(m + n) 的额外空间，但这仍然不是最好的解决方案。
    你能想出一个常数空间的解决方案吗？
```

- 解题思路

```
func setZeroes(matrix [][]int) {
        x := make(map[int]int)
        y := make(map[int]int)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                x[i] = 1
                                y[j] = 1
                        }
                }
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if x[i] == 1 || y[j] == 1 {
                                matrix[i][j] = 0
                        }
                }
        }
}

#
func setZeroes(matrix [][]int) {
        m := make(map[[2]int]bool)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == math.MinInt32 {
                                m[[2]int{i, j}] = true
                        }
                }
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                for k := 0; k < len(matrix); k++ {
                                        for l := 0; l < len(matrix[k]); l++ {
                                                if (k == i || l == j) && matrix[k][l]␣
↪!= 0 {
                                                        delete(m, [2]int{k, l})
                                                        matrix[k][l] = math.MinInt32
                                                }
                                        }
                                }
                        }
                }
```

(续下页)

(接上页)

```
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == math.MinInt32 && m[[2]int{i, j}] == false {
                                matrix[i][j] = 0
                        }
                }
        }
}

# 3
func setZeroes(matrix [][]int) {
        flag := false
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] == 0 {
                        flag = true
                }
                for j := 1; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                matrix[i][0] = 0
                                matrix[0][j] = 0
                        }
                }
        }
        for i := 1; i < len(matrix); i++ {
                for j := 1; j < len(matrix[i]); j++ {
                        if matrix[i][0] == 0 || matrix[0][j] == 0 {
                                matrix[i][j] = 0
                        }
                }
        }
        // 第一行处理
        if matrix[0][0] == 0 {
                for j := 0; j < len(matrix[0]); j++ {
                        matrix[0][j] = 0
                }
        }
        // 第一列处理
        if flag == true {
                for i := 0; i < len(matrix); i++ {
                        matrix[i][0] = 0
                }
        }
```

(续下页)

(接上页)

```
}

# 4
func setZeroes(matrix [][]int) {
        flag := false
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] == 0 {
                        flag = true
                }
                for j := 1; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                matrix[i][0] = 0
                                matrix[0][j] = 0
                        }
                }
        }
        for i := len(matrix) - 1; i >= 0; i-- {
                for j := len(matrix[i]) - 1; j >= 1; j-- {
                        if matrix[i][0] == 0 || matrix[0][j] == 0 {
                                matrix[i][j] = 0
                        }
                }
        }
        // 第一列处理
        if flag == true {
                for i := 0; i < len(matrix); i++ {
                        matrix[i][0] = 0
                }
        }
}
```

2.39 74. 搜索二维矩阵 (6)

- 题目

```
编写一个高效的算法来判断 m x n 矩阵中，是否存在一个目标值。该矩阵具有如下特性：
    每行中的整数从左到右按升序排列。
    每行的第一个整数大于前一行的最后一个整数。
示例 1:输入:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
```

(续下页)

(接上页)

```
  [23, 30, 34, 50]
]
target = 3
输出: true
示例 2:输入:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
输出: false
```

- 解题思路

```
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == target {
                                return true
                        }
                }
        }
        return false
}

# 2
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        for j := 0; j < len(matrix[i]); j++ {
                                if matrix[i][j] == target {
```

(续下页)

(接上页)

```
                                        return true
                                }
                        }
                }
        }
        return false
}

# 3
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        res := binarySearch(matrix[i], target)
                        if res == true {
                                return true
                        }
                }
        }
        return false
}

func binarySearch(arr []int, target int) bool {
        left := 0
        right := len(arr) - 1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        return true
                } else if arr[mid] > target {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return false
}
```

(续下页)

(接上页)

```
# 4
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := len(matrix) - 1
        j := 0
        for i >= 0 && j < len(matrix[0]) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        i--
                } else {
                        j++
                }
        }
        return false
}

# 5
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := 0
        j := len(matrix[0]) - 1
        for j >= 0 && i < len(matrix) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        j--
                } else {
                        i++
                }
        }
        return false
}
```

(续下页)

(接上页)

```
# 6
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                index := sort.SearchInts(matrix[i], target)
                if index < len(matrix[i]) && target == matrix[i][index] {
                        return true
                }
        }
        return false
}
```

2.40 75. 颜色分类 (3)

- 题目

```
给定一个包含红色、白色和蓝色，一共 n 个元素的数组，原地对它们进行排序，
使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。
此题中，我们使用整数 0、 1 和 2 分别表示红色、白色和蓝色。
注意:
不能使用代码库中的排序函数来解决这道题。
示例:输入: [2,0,2,1,1,0] 输出: [0,0,1,1,2,2]
进阶:
    一个直观的解决方案是使用计数排序的两趟扫描算法。
    首先，迭代计算出0、1 和 2 元素的个数，然后按照0、1、2的排序，重写当前数组。
    你能想出一个仅使用常数空间的一趟扫描算法吗？
```

- 解题思路

```
func sortColors(nums []int) {
        sort.Ints(nums)
}

# 2
func sortColors(nums []int) {
        left := 0
```

(续下页)

(接上页)

```
        right := len(nums) - 1
        for i := 0; i <= right; i++ {
                if nums[i] == 0 {
                        nums[left], nums[i] = nums[i], nums[left]
                        left++
                } else if nums[i] == 2 {
                        nums[right], nums[i] = nums[i], nums[right]
                        right--
                        i--
                }
        }
}

# 3
func sortColors(nums []int) {
        arr := make([]int, 3)
        for i := 0; i < len(nums); i++ {
                arr[nums[i]]++
        }
        count := 0
        for i := 0; i < len(arr); i++ {
                for j := 0; j < arr[i]; j++ {
                        nums[count] = i
                        count++
                }
        }
}
```

2.41 77. 组合 (5)

- 题目

```
给定两个整数 n 和 k，返回 1 ... n 中所有可能的 k 个数的组合。
示例:输入: n = 4, k = 2 输出:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

- 解题思路

```
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        nums := make([]int, 0)
        for i := 1; i <= n; i++ {
                nums = append(nums, i)
        }
        dfs(nums, 0, k)
        return res
}

func dfs(nums []int, index, k int) {
        if index == k {
                temp := make([]int, k)
                copy(temp, nums[:k])
                res = append(res, temp)
                return
        }
        for i := index; i < len(nums); i++ {
                if index == 0 || nums[i] > nums[index-1] {
                        nums[i], nums[index] = nums[index], nums[i]
                        dfs(nums, index+1, k)
                        nums[i], nums[index] = nums[index], nums[i]
                }
        }
}

# 2
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        dfs(n, k, 1, make([]int, 0))
        return res
}

func dfs(n, k, index int, arr []int) {
        if len(arr) == k {
                temp := make([]int, k)
                copy(temp, arr)
                res = append(res, temp)
                return
```

(续下页)

(接上页)

```
        }
        for i := index; i <= n; i++ {
                arr = append(arr, i)
                dfs(n, k, i+1, arr)
                arr = arr[:len(arr)-1]
        }
}

# 3
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        nums := make([]int, 0)
        for i := 1; i <= n; i++ {
                nums = append(nums, i)
        }
        dfs(nums, 0, k, make([]int, 0))
        return res
}

func dfs(nums []int, index, k int, arr []int) {
        if len(arr) == k {
                temp := make([]int, k)
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(nums); i++ {
                arr = append(arr, nums[i])
                dfs(nums, i+1, k, arr)
                arr = arr[:len(arr)-1]
        }
}

# 4
func combine(n int, k int) [][]int {
        res := make([][]int, 0)
        arr := make([]int, 0)
        for i := 1; i <= k; i++ {
                arr = append(arr, 0)
        }
        i := 0
```

(续下页)

(接上页)

```
        for i >= 0 {
                arr[i]++
                if arr[i] > n {
                        i--
                } else if i == k-1 {
                        temp := make([]int, k)
                        copy(temp, arr)
                        res = append(res, temp)
                } else {
                        i++
                        arr[i] = arr[i-1]
                }
        }
        return res
}

# 5
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        dfs(n, k, 1, make([]int, 0))
        return res
}

func dfs(n, k, index int, arr []int) {
        if index > n+1 {
                return
        }
        if len(arr) == k {
                temp := make([]int, k)
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        dfs(n, k, index+1, arr)
        arr = append(arr, index)
        dfs(n, k, index+1, arr)
}
```

2.42 78. 子集 (4)

- 题目

```
给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。
说明：解集不能包含重复的子集。
示例:输入: nums = [1,2,3] 输出:
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

- 解题思路

```
var res [][]int

func subsets(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, make([]int, 0), 0)
        return res
}

func dfs(nums []int, arr []int, level int) {
        temp := make([]int, len(arr))
        copy(temp, arr)
        res = append(res, temp)
        for i := level; i < len(nums); i++ {
                dfs(nums, append(arr, nums[i]), i+1)
        }
}

# 2
func subsets(nums []int) [][]int {
        res := make([][]int, 0)
        res = append(res, []int{})
        for i := 0; i < len(nums); i++ {
                temp := make([][]int, len(res))
                for key, value := range res {
```

(续下页)

(接上页)

```
                        value = append(value, nums[i])
                        temp[key] = append(temp[key], value...)
                }
                for _, v := range temp {
                        res = append(res, v)
                }
        }
        return res
}

# 3
func subsets(nums []int) [][]int {
        res := make([][]int, 0)
        n := len(nums)
        left := 1 << n
        right := 1 << (n + 1)
        for i := left; i < right; i++ {
                temp := make([]int, 0)
                for j := 0; j < n; j++ {
                        if i&(1<<j) != 0 {
                                temp = append(temp, nums[j])
                        }
                }
                res = append(res, temp)
        }
        return res
}

# 4
var res [][]int

func subsets(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, make([]int, 0), 0)
        return res
}

func dfs(nums []int, arr []int, level int) {
        if level >= len(nums) {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
```

(续下页)

(接上页)

```
        }
        dfs(nums, arr, level+1)
        dfs(nums, append(arr, nums[level]), level+1)
}
```

2.43 79. 单词搜索 (2)

- 题目

```
给定一个二维网格和一个单词，找出该单词是否存在于网格中。
单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格
同一个单元格内的字母不允许被重复使用。
示例:board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
给定 word = "ABCCED", 返回 true
给定 word = "SEE", 返回 true
给定 word = "ABCB", 返回 false
提示：
    board 和 word 中只包含大写和小写英文字母。
    1 <= board.length <= 200
    1 <= board[i].length <= 200
    1 <= word.length <= 10^3
```

- 解题思路

```
func exist(board [][]byte, word string) bool {
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[0]); j++ {
                        if dfs(board, i, j, word, 0) {
                                return true
                        }
                }
        }
        return false
}

func dfs(board [][]byte, i, j int, word string, level int) bool {
        if i < 0 || i >= len(board) || j < 0 || j >= len(board[0]) ||
```

(续下页)

(接上页)

```
                board[i][j] != word[level] {
                return false
        }
        if level == len(word)-1 {
                return true
        }
        temp := board[i][j]
        board[i][j] = ' '
        res := dfs(board, i+1, j, word, level+1) ||
                dfs(board, i-1, j, word, level+1) ||
                dfs(board, i, j+1, word, level+1) ||
                dfs(board, i, j-1, word, level+1)
        board[i][j] = temp
        return res
}

#
func exist(board [][]byte, word string) bool {
        visited := make([][]bool, len(board))
        for i := 0; i < len(board); i++ {
                visited[i] = make([]bool, len(board[0]))
        }
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[0]); j++ {
                        if dfs(board, i, j, word, 0, visited) {
                                return true
                        }
                }
        }
        return false
}

func dfs(board [][]byte, i, j int, word string, level int, visited [][]bool) bool {
        res := false
        if i >= 0 && i < len(board) && j >= 0 && j < len(board[0]) &&
                visited[i][j] == false && board[i][j] == word[level] {
                if level == len(word)-1 {
                        return true
                }
                visited[i][j] = true
                level = level + 1
                res = dfs(board, i+1, j, word, level, visited) ||
                        dfs(board, i-1, j, word, level, visited) ||
```

(续下页)

(接上页)

```
                        dfs(board, i, j+1, word, level, visited) ||
                        dfs(board, i, j-1, word, level, visited)
                if !res {
                        visited[i][j] = false
                        level = level - 1
                }
        }
        return res
}
```

2.44 80. 删除排序数组中的重复项 II(2)

- 题目

```
给定一个排序数组，你需要在原地删除重复出现的元素，使得每个元素最多出现两次，返回移除后数组的新长度。
不要使用额外的数组空间，你必须在原地修改输入数组并在使用 O(1) 额外空间的条件下完成。
示例 1:给定 nums = [1,1,1,2,2,3],
函数应返回新长度 length = 5, 并且原数组的前五个元素被修改为 1, 1, 2, 2, 3 。
你不需要考虑数组中超出新长度后面的元素。
示例 2:给定 nums = [0,0,1,1,1,1,2,3,3],
函数应返回新长度 length = 7, 并且原数组的前五个元素被修改为 0, 0, 1, 1, 2, 3, 3 。
你不需要考虑数组中超出新长度后面的元素。
说明:为什么返回数值是整数，但输出的答案是数组呢?
请注意，输入数组是以“引用”方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。
你可以想象内部操作如下:
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);
// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度，它会打印出数组中该长度范围内的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

- 解题思路

```
func removeDuplicates(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
                return 1
        }
```

(续下页)

(接上页)

```
        n := 2
        i := n
        for j := n; j < len(nums); j++ {
                if nums[i-n] != nums[j] {
                        nums[i] = nums[j]
                        i++
                }
        }
        return i
}

#
func removeDuplicates(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        prev := nums[0]
        count := 1
        j := 1
        for i := 1; i < len(nums); i++ {
                if nums[i] == prev {
                        count = count + 1
                } else {
                        count = 1
                        prev = nums[i]
                }
                if count <= 2 {
                        nums[j] = nums[i]
                        j++
                }
        }
        return j
}
```

2.45 81. 搜索旋转排序数组 II(2)

- 题目

```
假设按照升序排序的数组在预先未知的某个点上进行了旋转。
( 例如，数组 [0,0,1,2,2,5,6] 可能变为 [2,5,6,0,0,1,2] )。
编写一个函数来判断给定的目标值是否存在于数组中。若存在返回 true，否则返回 false。
示例 1:输入: nums = [2,5,6,0,0,1,2], target = 0 输出: true
```

(续下页)

(接上页)

```
示例 2:输入: nums = [2,5,6,0,0,1,2], target = 3 输出: false
进阶:
    这是 搜索旋转排序数组 的延伸题目，本题中的 nums  可能包含重复元素。
    这会影响到程序的时间复杂度吗？会有怎样的影响，为什么？
```

- 解题思路

```
func search(nums []int, target int) bool {
        for i := 0; i < len(nums); i++ {
                if target == nums[i] {
                        return true
                }
        }
        return false
}

#
func search(nums []int, target int) bool {
        left, right := 0, len(nums)-1
        for left <= right {
                for left < right && nums[left] == nums[left+1] {
                        left++
                }
                for left < right && nums[right] == nums[right-1] {
                        right--
                }
                mid := left + (right-left)/2
                if nums[mid] == target {
                        return true
                }
                if nums[left] <= nums[mid] {
                        if nums[left] <= target && target < nums[mid] {
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                } else {
                        if nums[mid] < target && target <= nums[right] {
                                left = mid + 1
                        } else {
                                right = mid - 1
                        }
                }
        }
```

(续下页)

(接上页)

```
        return false
}
```

2.46 82. 删除排序链表中的重复元素 II(3)

- 题目

```
给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 没有重复出现 的数字。
示例 1:输入: 1->2->3->3->4->4->5 输出: 1->2->5
示例 2:输入: 1->1->1->2->3 输出: 2->3
```

- 解题思路

```
func deleteDuplicates(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        temp := &ListNode{Next: head}
        cur := temp
        value := 0
        for cur.Next != nil && cur.Next.Next != nil {
                if cur.Next.Val == cur.Next.Next.Val {
                        value = cur.Next.Val
                        for cur.Next != nil && cur.Next.Val == value {
                                cur.Next = cur.Next.Next
                        }
                } else {
                        cur = cur.Next
                }
        }
        return temp.Next
}

#
func deleteDuplicates(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        flag := false
        for head.Next != nil && head.Val == head.Next.Val{
                head = head.Next
                flag = true
```

(续下页)

(接上页)

```
        }
        head.Next = deleteDuplicates(head.Next)
        if flag{
                return head.Next
        }
        return head
}

#
func deleteDuplicates(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        flag := false
        for head.Next != nil && head.Val == head.Next.Val{
                head = head.Next
                flag = true
        }
        head.Next = deleteDuplicates(head.Next)
        if flag{
                return head.Next
        }
        return head
}
```

2.47 86. 分隔链表 (2)

- 题目

```
给定一个链表和一个特定值 x，对链表进行分隔，使得所有小于 x 的节点都在大于或等于 x␣
↪的节点之前。
你应当保留两个分区中每个节点的初始相对位置。
示例:
输入: head = 1->4->3->2->5->2, x = 3
输出: 1->2->2->4->3->5
```

- 解题思路

```
func partition(head *ListNode, x int) *ListNode {
        first := &ListNode{}
        second := &ListNode{}
        a := first
```

(续下页)

(接上页)

```
        b := second
        for head != nil {
                if head.Val < x {
                        a.Next = head
                        a = head
                } else {
                        b.Next = head
                        b = head
                }
                head = head.Next
        }
        b.Next = nil
        a.Next = second.Next
        return first.Next
}

#
func partition(head *ListNode, x int) *ListNode {
        a := make([]*ListNode, 0)
        b := make([]*ListNode, 0)

        for head != nil {
                if head.Val < x {
                        a = append(a, head)
                } else {
                        b = append(b, head)
                }
                head = head.Next
        }
        temp := &ListNode{}
        node := temp
        for i := 0; i < len(a); i++ {
                node.Next = a[i]
                node = node.Next
        }
        for i := 0; i < len(b); i++ {
                node.Next = b[i]
                node = node.Next
        }
        node.Next = nil
        return temp.Next
}
```

2.48 89. 格雷编码 (3)

- 题目

```
格雷编码是一个二进制数字系统，在该系统中，两个连续的数值仅有一个位数的差异。
给定一个代表编码总位数的非负整数␣
↪n，打印其格雷编码序列。即使有多个不同答案，你也只需要返回其中一种。
格雷编码序列必须以 0 开头。
示例 1:输入: 2 输出: [0,1,3,2]
解释:
00 - 0
01 - 1
11 - 3
10 - 2
对于给定的 n，其格雷编码序列并不唯一。例如，[0,2,3,1] 也是一个有效的格雷编码序列。
00 - 0
10 - 2
11 - 3
01 - 1
示例 2:输入: 0 输出: [0]
解释: 我们定义格雷编码序列必须以 0 开头。
     给定编码总位数为 n 的格雷编码序列，其长度为 2n。当 n = 0 时，长度为 20 = 1。
     因此，当 n = 0 时，其格雷编码序列为 [0]。
```

- 解题思路

```
func grayCode(n int) []int {
        if n == 0 {
                return []int{0}
        }
        res := []int{0, 1}
        for i := 1; i < n; i++ {
                temp := make([]int, 0)
                value := 1 << i
                for j := len(res) - 1; j >= 0; j-- {
                        // 10 1 11
                        // 10 0 10
                        // 100 10 110
                        // 100 11 111
                        // 100 1 101
                        // 100 0 100
                        // fmt.Printf("%b %b %b\n", value, res[j], res[j]^value)

                        // temp = append(temp, res[j]|value)
```

(续下页)

(接上页)

```
                        // temp = append(temp, res[j]^value)
                        temp = append(temp, res[j]+value)
                }
                res = append(res, temp...)
        }
        return res
}

# 2
func grayCode(n int) []int {
        total := 1 << n
        res := make([]int, 0)
        for i := 0; i < total; i++ {
                res = append(res, i^(i>>1))
        }
        return res
}

# 3
func grayCode(n int) []int {
        if n == 0 {
                return []int{0}
        }
        res := []int{0, 1}
        for i := 1; i < n; i++ {
                value := 1 << i
                for j := len(res) - 1; j >= 0; j-- {
                        res= append(res, res[j]+value)
                }
        }
        return res
}
```

2.49 90. 子集 II(2)

- 题目

```
给定一个可能包含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。
说明：解集不能包含重复的子集。
示例:输入: [1,2,2] 输出:
[
  [2],
```

(续下页)

(接上页)

```
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

- 解题思路

```
var res [][]int

func subsetsWithDup(nums []int) [][]int {
        sort.Ints(nums)
        res = make([][]int, 0)
        dfs(nums, make([]int, 0), 0)
        return res
}

func dfs(nums []int, arr []int, level int) {
        temp := make([]int, len(arr))
        copy(temp, arr)
        res = append(res, temp)
        for i := level; i < len(nums); i++ {
                if i > level && nums[i] == nums[i-1] {
                        continue
                }
                arr = append(arr, nums[i])
                dfs(nums, arr, i+1)
                arr = arr[:len(arr)-1]
        }
}

# 2
var res [][]int

func subsetsWithDup(nums []int) [][]int {
        sort.Ints(nums)
        res = make([][]int, 0)
        dfs(nums, make([]int, 0))
        return res
}

func dfs(nums []int, arr []int) {
        temp := make([]int, len(arr))
```

(续下页)

(接上页)

```
        copy(temp, arr)
        res = append(res, temp)
        for i := 0; i < len(nums); i++ {
                if i > 0 && nums[i] == nums[i-1] {
                        continue
                }
                arr = append(arr, nums[i])
                dfs(nums[i+1:], arr)
                arr = arr[:len(arr)-1]
        }
}
```

2.50 91. 解码方法 (3)

- 题目

```
一条包含字母 A-Z 的消息通过以下方式进行了编码:
'A' -> 1
'B' -> 2
...
'Z' -> 26
给定一个只包含数字的非空字符串，请计算解码方法的总数。
示例 1:输入: "12" 输出: 2
解释: 它可以解码为 "AB" (1 2) 或者 "L" (12) 。
示例 2:输入: "226" 输出: 3
解释: 它可以解码为 "BZ" (2 26), "VF" (22 6), 或者 "BBF" (2 2 6) 。
```

- 解题思路

```
func numDecodings(s string) int {
        if s[0] == '0' {
                return 0
        }
        pre := 1
        cur := 1
        for i := 1; i < len(s); i++ {
                temp := cur
                if s[i] == '0' {
                        if s[i-1] == '1' || s[i-1] == '2' {
                                cur = pre
                        } else {
                                return 0
```

(续下页)

(接上页)

```
                        }
                } else if s[i-1] == '1' ||
                        (s[i-1] == '2' && s[i] >= '1' && s[i] <= '6') {
                        cur = cur + pre
                }
                pre = temp
        }
        return cur
}

# 2
func numDecodings(s string) int {
        if s[0] == '0' {
                return 0
        }
        dp := make([]int, len(s)+1)
        dp[0] = 1
        for i := 0; i < len(s); i++{
                if s[i] == '0' {
                        if i == 0 || s[i-1] == '1' || s[i-1] == '2' {
                                return 0
                        }
                        dp[i+1] = dp[i-1]
                } else{
                        if  i > 0 && (s[i-1] == '2' && s[i] >= '1' && s[i] <= '6') {
                                dp[i+1] = dp[i-1]+dp[i]
                        }else {
                                dp[i+1] = dp[i]
                        }
                }
        }
        return dp[len(s)]
}

# 3
var m map[string]int

func numDecodings(s string) int {
        m = make(map[string]int)
        return dfs(s)
}

func dfs(s string) int {
```

(续下页)

(接上页)

```
        if m[s] > 0 {
                return m[s]
        }
        if len(s) == 0 {
                return 1
        }
        if s[0] == '0' {
                return 0
        }
        if len(s) == 1 {
                return 1
        }
        if (s[0]-'0')*10+s[1]-'0' > 26 {
                return dfs(s[1:])
        }
        m[s] = dfs(s[1:]) + dfs(s[2:])
        return m[s]
}
```

2.51 92. 反转链表 II(2)

- 题目

```
反转从位置 m 到 n 的链表。请使用一趟扫描完成反转。
说明: 1 ≤ m ≤ n ≤ 链表长度。
示例:输入: 1->2->3->4->5->NULL, m = 2, n = 4 输出: 1->4->3->2->5->NULL
```

- 解题思路

```
func reverseBetween(head *ListNode, m int, n int) *ListNode {
        if m == n || head == nil {
                return head
        }
        temp := &ListNode{Next: head}
        prev := temp
        for i := 1; i < m; i++ {
                prev = prev.Next
        }
        head = prev.Next
        for i := m; i < n; i++ {
                next := head.Next
                head.Next = next.Next
```

(续下页)

(接上页)

```
                next.Next = prev.Next
                prev.Next = next
        }
        return temp.Next
}

# 2
func reverseBetween(head *ListNode, m int, n int) *ListNode {
        if m == 1 {
                return reverseN(head, n)
        }
        head.Next = reverseBetween(head.Next, m-1, n-1)
        return head
}

var next *ListNode

func reverseN(head *ListNode, n int) *ListNode {
        if n == 1 {
                next = head.Next
                return head
        }
        prev := reverseN(head.Next, n-1)
        head.Next.Next = head
        head.Next = next
        return prev
}
```

2.52 93. 复原 IP 地址 (2)

- 题目

```
给定一个只包含数字的字符串，复原它并返回所有可能的 IP 地址格式。
有效的 IP 地址正好由四个整数（每个整数位于 0 到 255 之间组成），整数之间用 '.' 分隔。
示例:输入: "25525511135" 输出: ["255.255.11.135", "255.255.111.35"]
```

- 解题思路

```
var res []string

func restoreIpAddresses(s string) []string {
        res = make([]string, 0)
```

(续下页)

(接上页)

```
        if len(s) < 4 || len(s) > 12 {
                return nil
        }
        dfs(s, make([]string, 0), 0)
        return res
}

func dfs(s string, arr []string, level int) {
        if level == 4 {
                if len(s) == 0 {
                        str := strings.Join(arr, ".")
                        res = append(res, str)
                }
                return
        }
        for i := 1; i <= 3; i++ {
                if i <= len(s) {
                        value, _ := strconv.Atoi(s[:i])
                        if value <= 255 {
                                str := s[i:]
                                dfs(str, append(arr, s[:i]), level+1)
                        }
                        if value == 0 {
                                // 避免出现001,01这种情况
                                break
                        }
                }
        }
}

# 2
func restoreIpAddresses(s string) []string {
        res := make([]string, 0)
        if len(s) < 4 || len(s) > 12 {
                return nil
        }
        for i := 1; i <= 3 && i < len(s)-2; i++ {
                for j := i + 1; j <= i+3 && j < len(s)-1; j++ {
                        for k := j + 1; k <= j+3 && k < len(s); k++ {
                                if judge(s[:i]) && judge(s[i:j]) &&
                                        judge(s[j:k]) && judge(s[k:]) {
                                        res = append(res, s[:i]+"."+s[i:j]+"."+s[j:k]+
↪"."+s[k:])
```

(续下页)

(接上页)

```
                                }
                        }
                }
        }
        return res
}

func judge(s string) bool {
        if len(s) > 1 && s[0] == '0' {
                return false
        }
        value, _ := strconv.Atoi(s)
        if value > 255 {
                return false
        }
        return true
}
```

2.53 94. 二叉树的中序遍历 (3)

- 题目

```
给定一个二叉树，返回它的中序 遍历。
示例:输入: [1,null,2,3]
   1
    \
     2
    /
   3
输出: [1,3,2]
进阶: 递归算法很简单，你可以通过迭代算法完成吗？
```

- 解题思路

```
func inorderTraversal(root *TreeNode) []int {
        if root == nil {
                return nil
        }
        left := inorderTraversal(root.Left)
        right := inorderTraversal(root.Right)
        res := left
        res = append(res, root.Val)
```

(续下页)

(接上页)

```
        res = append(res, right...)
        return res
}

# 2
func inorderTraversal(root *TreeNode) []int {
        if root == nil {
                return nil
        }
        stack := make([]*TreeNode, 0)
        res := make([]int, 0)
        for len(stack) > 0 || root != nil {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                last := len(stack) - 1
                res = append(res, stack[last].Val)
                root = stack[last].Right
                stack = stack[:last]
        }
        return res
}

# 3
var res []int

func inorderTraversal(root *TreeNode) []int {
        res = make([]int, 0)
        dfs(root)
        return res
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                res = append(res, root.Val)
                dfs(root.Right)
        }
}
```

2.54 95. 不同的二叉搜索树 II(2)

- 题目

```
给定一个整数 n，生成所有由 1 ... n 为节点所组成的 二叉搜索树 。
示例：输入：3
输出：
[
  [1,null,3,2],
  [3,2,null,1],
  [3,1,null,null,2],
  [2,1,3],
  [1,null,2,null,3]
]
解释：以上的输出对应以下 5 种不同结构的二叉搜索树：
   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
提示：
    0 <= n <= 8
```

- 解题思路

```
func generateTrees(n int) []*TreeNode {
        if n == 0 {
                return nil
        }
        return dfs(1, n)
}

func dfs(left, right int) []*TreeNode {
        if left > right {
                return []*TreeNode{nil}
        }
        if left == right {
                return []*TreeNode{
                        {Val: left},
                }
        }
        arr := make([]*TreeNode, 0)
        for i := left; i <= right; i++ {
                leftTree := dfs(left, i-1)
```

(续下页)

(接上页)

```
                rightTree := dfs(i+1, right)
                for j := 0; j < len(leftTree); j++ {
                        for k := 0; k < len(rightTree); k++ {
                                node := &TreeNode{Val: i}
                                node.Left = leftTree[j]
                                node.Right = rightTree[k]
                                arr = append(arr, node)
                        }
                }
        }
        return arr
}

# 2
func generateTrees(n int) []*TreeNode {
        if n == 0 {
                return nil
        }
        dp := make([][]*TreeNode, n+1)
        dp[1] = append(dp[1], &TreeNode{Val: 1})
        for i := 2; i <= n; i++ {
                for _, node := range dp[i-1]{
                        root := &TreeNode{Val:i}
                        root.Left = node
                        dp[i] = append(dp[i], copyTree(root))
                        root = node
                        temp := root
                        newNode := &TreeNode{Val:i}
                        for temp != nil{
                                newNode.Left = temp.Right
                                temp.Right = newNode
                                dp[i] = append(dp[i], copyTree(root))
                                temp.Right = newNode.Left
                                newNode.Left = nil
                                temp = temp.Right
                        }
                }
        }
        return dp[n]
}

func copyTree(node *TreeNode) *TreeNode {
        if node == nil {
```

(续下页)

(接上页)

```
                return nil
        }
        newNode := &TreeNode{Val: node.Val}
        newNode.Left = copyTree(node.Left)
        newNode.Right = copyTree(node.Right)
        return newNode
}
```

2.55 96. 不同的二叉搜索树 (3)

- 题目

```
给定一个整数 n, 求以 1 ... n 为节点组成的二叉搜索树有多少种?
示例:输入: 3 输出: 5
解释: 给定 n = 3, 一共有 5 种不同结构的二叉搜索树:
   1          3     3      2      1
    \        /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

- 解题思路

```
func numTrees(n int) int {
        dp := make([]int, n+1)
        dp[0] =1
        dp[1] =1
        for i := 2; i <= n;i++{
                for j := 1; j <= i; j++{
                        dp[i] = dp[i] + dp[j-1]*dp[i-j]
                }
        }
        return dp[n]
}

#
/*
C0 = 1
Cn+1 = 2(2n+1)/(n+2) * Cn
*/
func numTrees(n int) int {
        c := 1
```

(续下页)

(接上页)

```
        for i := 1; i < n;i++{
                c = c * 2 * (2*i+1)/(i+2)
        }
        return c
}

#
func numTrees(n int) int {
        c := 1
        for i := 1; i <= n;i++{
                c = c * (n+i)/i
        }
        return c/(n+1)
}
```

2.56 98. 验证二叉搜索树 (5)

- 题目

```
给定一个二叉树，判断其是否是一个有效的二叉搜索树。
假设一个二叉搜索树具有如下特征：
    节点的左子树只包含小于当前节点的数。
    节点的右子树只包含大于当前节点的数。
    所有左子树和右子树自身必须也是二叉搜索树。
示例 1:输入:
    2
   / \
  1   3
输出: true
示例 2:输入:
    5
   / \
  1   4
     / \
    3   6
输出: false
解释: 输入为: [5,1,4,null,null,3,6]。
     根节点的值为 5 ，但是其右子节点值为 4 。
```

- 解题思路

```
func isValidBST(root *TreeNode) bool {
        return dfs(root, math.MinInt64, math.MaxInt64)
}

func dfs(root *TreeNode, left, right int) bool {
        if root == nil {
                return true
        }
        if left >= root.Val || right <= root.Val {
                return false
        }
        return dfs(root.Left, left, root.Val) && dfs(root.Right, root.Val, right)
}

# 2
var res []int

func isValidBST(root *TreeNode) bool {
        res = make([]int, 0)
        dfs(root)
        for i := 0; i < len(res)-1; i++ {
                if res[i] >= res[i+1] {
                        return false
                }
        }
        return true
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                res = append(res, root.Val)
                dfs(root.Right)
        }
}

# 3
func isValidBST(root *TreeNode) bool {
        if root == nil {
                return true
        }
        stack := make([]*TreeNode, 0)
        res := make([]int, 0)
        for len(stack) > 0 || root != nil {
```

(续下页)

(接上页)

```
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                last := len(stack) - 1
                res = append(res, stack[last].Val)
                root = stack[last].Right
                stack = stack[:last]
        }
        for i := 0; i < len(res)-1; i++ {
                if res[i] >= res[i+1] {
                        return false
                }
        }
        return true
}

# 4
func isValidBST(root *TreeNode) bool {
        if root == nil {
                return true
        }
        stack := make([]*TreeNode, 0)
        pre := math.MinInt64
        for len(stack) > 0 || root != nil {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                last := len(stack) - 1
                if stack[last].Val <= pre {
                        return false
                }
                pre = stack[last].Val
                root = stack[last].Right
                stack = stack[:last]
        }
        return true
}

# 5
var pre int
```

(续下页)

(接上页)

```
func isValidBST(root *TreeNode) bool {
        pre = math.MinInt64
        return dfs(root)
}

func dfs(root *TreeNode) bool {
        if root == nil {
                return true
        }
        if dfs(root.Left) == false {
                return false
        }
        if root.Val <= pre {
                return false
        }
        pre = root.Val
        return dfs(root.Right)
}
```

CHAPTER 3

0001-1000-Hard

3.1 4. 寻找两个正序数组的中位数 (4)

- 题目

```
给定两个大小为 m 和 n 的正序（从小到大）数组 nums1 和 nums2。
请你找出这两个正序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。
你可以假设 nums1 和 nums2 不会同时为空。
示例 1:nums1 = [1, 3] nums2 = [2]
则中位数是 2.0
示例 2:nums1 = [1, 2] nums2 = [3, 4]
则中位数是 (2 + 3)/2 = 2.5
```

- 解题思路

```
func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {
        nums1 = append(nums1, nums2...)
        sort.Ints(nums1)
        if len(nums1)%2 == 1 {
                return float64(nums1[len(nums1)/2])
        }
        return float64(nums1[len(nums1)/2]+nums1[len(nums1)/2-1]) / 2
}

# 2
```

(续下页)

(接上页)

```
func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {
        total := len(nums1) + len(nums2)
        if total%2 == 1 {
                mid := total / 2
                return float64(getKth(nums1, nums2, mid+1))
        }
        mid1, mid2 := total/2-1, total/2
        return float64(getKth(nums1, nums2, mid1+1)+getKth(nums1, nums2, mid2+1)) / 2.
↪0
}

func getKth(nums1 []int, nums2 []int, k int) int {
        a, b := 0, 0
        for {
                if a == len(nums1) {
                        return nums2[b+k-1]
                }
                if b == len(nums2) {
                        return nums1[a+k-1]
                }
                if k == 1 {
                        return min(nums1[a], nums2[b])
                }
                mid := k / 2
                newA := min(a+mid, len(nums1)) - 1
                newB := min(b+mid, len(nums2)) - 1
                valueA, valueB := nums1[newA], nums2[newB]
                if valueA < valueB {
                        k = k - (newA - a + 1)
                        a = newA + 1
                } else {
                        k = k - (newB - b + 1)
                        b = newB + 1
                }
        }
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
# 3
func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {
        total := len(nums1) + len(nums2)
        a, b := total/2, (total-1)/2
        count := 0
        res := 0
        for i, j := 0, 0; i < len(nums1) || j < len(nums2); count++ {
                if i < len(nums1) && (j == len(nums2) || nums1[i] < nums2[j]) {
                        if count == a {
                                res = res + nums1[i]
                        }
                        if count == b {
                                res = res + nums1[i]
                        }
                        i++
                } else {
                        if count == a {
                                res = res + nums2[j]
                        }
                        if count == b {
                                res = res + nums2[j]
                        }
                        j++
                }
        }
        return float64(res) / 2
}

# 4
func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {
        n, m := len(nums1), len(nums2)
        if n > m {
                return findMedianSortedArrays(nums2, nums1)
        }
        left, right := 0, n
        a, b := 0, 0
        for left <= right {
                // 左半部分最大的值小于等于右半部分最小的值: max(A[i-1],B[j-1])) <=␣
↪min(A[i],B[j]))
                i := left + (right-left)/2 // i,j分别对num1,num2的划分
                j := (n+m+1)/2 - i         // i+j == (n+m+1)/2
                // 偶数求a=>max(A[i-1],B[j-1])) b=>min(A[i],B[j]))
```

(续下页)

(接上页)

```
                // 奇数求a=>max(A[i-1],B[j-1]))
                if j != 0 && i != n && nums1[i] < nums2[j-1] {
                        left = i + 1
                } else if i != 0 && j != m && nums1[i-1] > nums2[j] {
                        right = i - 1
                } else {
                        if i == 0 {
                                a = nums2[j-1]
                        } else if j == 0 {
                                a = nums1[i-1]
                        } else {
                                a = max(nums1[i-1], nums2[j-1])
                        }
                        if (n+m)%2 == 1 {
                                return float64(a)
                        }
                        if i == n {
                                b = nums2[j]
                        } else if j == m {
                                b = nums1[i]
                        } else {
                                b = min(nums1[i], nums2[j])
                        }
                        return float64(a+b) / 2.0
                }
        }
        return 0.0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

3.2 10. 正则表达式匹配 (3)

- 题目

```
给你一个字符串 s 和一个字符规律 p，请你来实现一个支持 '.' 和 '*' 的正则表达式匹配。
'.' 匹配任意单个字符
'*' 匹配零个或多个前面的那一个元素
所谓匹配，是要涵盖 整个 字符串 s的，而不是部分字符串。
说明:
    s 可能为空，且只包含从 a-z 的小写字母。
    p 可能为空，且只包含从 a-z 的小写字母，以及字符 . 和 *。
示例 1:输入:s = "aa" p = "a" 输出: false
解释: "a" 无法匹配 "aa" 整个字符串。
示例 2:输入: s = "aa" p = "a*" 输出: true
解释: 因为 '*' 代表可以匹配零个或多个前面的那一个元素，在这里前面的元素就是 'a'。
因此，字符串 "aa" 可被视为 'a' 重复了一次。
示例 3:输入: s = "ab" p = ".*" 输出: true
解释: ".*" 表示可匹配零个或多个 ('*') 任意字符 ('.') 。
示例 4:输入:s = "aab"p = "c*a*b" 输出: true
解释: 因为 '*' 表示零个或多个，这里 'c' 为 0 个，'a' 被重复一次。因此可以匹配字符串
↪"aab"。
示例 5:输入:s = "mississippi"p = "mis*is*p*." 输出: false
```

- 解题思路

```
func isMatch(s string, p string) bool {
        return dfs(s, p, 0, 0)
}

func dfs(s string, p string, i, j int) bool {
        if i >= len(s) && j >= len(p) {
                return true
        }
        if i <= len(s) && j >= len(p) {
                return false
        }
        if j+1 < len(p) && p[j+1] == '*' {
                if (i < len(s) && p[j] == s[i]) || (p[j] == '.' && i < len(s)) {
                        return dfs(s, p, i+1, j+2) ||
                                dfs(s, p, i+1, j) ||
                                dfs(s, p, i, j+2)
                } else {
                        return dfs(s, p, i, j+2)
                }
```

(续下页)

(接上页)

```
        }
        if (i < len(s) && s[i] == p[j]) || (p[j] == '.' && i < len(s)) {
                return dfs(s, p, i+1, j+1)
        }
        return false
}

# 2
func isMatch(s string, p string) bool {
        // dp[i][j]表示p[:i]能否正则匹配s[:j]
        dp := make([][]bool, len(p)+1)
        for i := 0; i < len(p)+1; i++ {
                dp[i] = make([]bool, len(s)+1)
        }
        // 1.初始化
        dp[0][0] = true
        for i := 2; i < len(p)+1; i++ {
                if i%2 == 0 && p[i-1] == '*' {
                        dp[i][0] = dp[i-2][0]
                }
        }
        // 2.dp状态转移
        for i := 1; i < len(p)+1; i++ {
                for j := 1; j < len(s)+1; j++ {
                        // 2.1 相同或者 .
                        if p[i-1] == s[j-1] || p[i-1] == '.' {
                                dp[i][j] = dp[i-1][j-1]
                        } else if p[i-1] == '*' {
                                if i > 1 {
                                        if p[i-2] == s[j-1] || p[i-2] == '.' {
                                                dp[i][j] = dp[i][j-1] || dp[i-2][j-1]
↪|| dp[i-2][j]
                                        } else {
                                                dp[i][j] = dp[i-2][j]
                                        }
                                }
                        }
                }
        }
        return dp[len(p)][len(s)]
}

# 3
```

(续下页)

(接上页)

```
func isMatch(s string, p string) bool {
        if len(s) == 0 && len(p) == 0 {
                return true
        } else if len(p) == 0 {
                return false
        }
        match := false
        // 正常匹配条件=>相等，或者 p[0]等于.就不用管s[0]
        if len(s) > 0 && (s[0] == p[0] || p[0] == '.') {
                match = true
        }
        // 匹配多个 就把 s 往后移1位，注意p不移动
        // 匹配0个 就把 p 往后移2位，相当于p的*当前作废
        if len(p) > 1 && p[1] == '*' {
                return (match && isMatch(s[1:], p)) || isMatch(s, p[2:])
        }
        // 匹配当前成功，同时往后移
        return match && isMatch(s[1:], p[1:])
}
```

3.3 23. 合并 K 个排序链表 (4)

- 题目

```
合并 k 个排序链表，返回合并后的排序链表。请分析和描述算法的复杂度。
示例:输入:
[
  1->4->5,
  1->3->4,
  2->6
]
输出: 1->1->2->3->4->4->5->6
```

- 解题思路

```
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        temp := &ListNode{}
        for i := 0; i < len(lists); i++ {
                temp.Next = mergeTwoLists(temp.Next, lists[i])
```

(续下页)

(接上页)

```
        }
        return temp.Next
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}

# 2
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        if len(lists) == 1 {
                return lists[0]
        }
        first := mergeKLists(lists[:len(lists)/2])
        second := mergeKLists(lists[len(lists)/2:])
        return mergeTwoLists(first, second)
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
```

(续下页)

(接上页)

```
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}

# 3
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        var h IntHeap
        heap.Init(&h)
        for i := 0; i < len(lists); i++ {
                if lists[i] != nil {
                        heap.Push(&h, lists[i])
                }
        }
        res := &ListNode{}
        temp := res
        for h.Len() > 0 {
                minItem := heap.Pop(&h).(*ListNode)
                temp.Next = minItem
                temp = temp.Next
                if minItem.Next != nil {
                        heap.Push(&h, minItem.Next)
                }
        }
        return res.Next
}

type IntHeap []*ListNode
```

(续下页)

(接上页)

```
func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i].Val < h[j].Val }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.(*ListNode)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 4
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        arr := make([]*ListNode, 0)
        for i := 0; i < len(lists); i++ {
                temp := lists[i]
                for temp != nil {
                        arr = append(arr, temp)
                        temp = temp.Next
                }
        }
        if len(arr) == 0 {
                return nil
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].Val < arr[j].Val
        })
        for i := 0; i < len(arr)-1; i++ {
                arr[i].Next = arr[i+1]
        }
        arr[len(arr)-1].Next = nil
        return arr[0]
}
```

3.4 25.K 个一组翻转链表 (4)

- 题目

```
给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。
k 是一个正整数，它的值小于或等于链表的长度。
如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。
示例：
给你这个链表：1->2->3->4->5
当 k = 2 时，应当返回：2->1->4->3->5
当 k = 3 时，应当返回：3->2->1->4->5
说明：
    你的算法只能使用常数的额外空间。
    你不能只是单纯的改变节点内部的值，而是需要实际进行节点交换。
```

- 解题思路

```
func reverseKGroup(head *ListNode, k int) *ListNode {
        length := getLength(head)
        if length < k || k <= 1 {
                return head
        }
        pre := &ListNode{}
        cur := head
        for i := 0; i < k; i++ {
                temp := cur
                cur = cur.Next
                temp.Next = pre
                pre = temp
        }
        head.Next = reverseKGroup(cur, k)
        return pre
}

func getLength(head *ListNode) int {
        if head == nil {
                return 0
        }
        temp := head
        res := 0
        for temp != nil {
                res++
                temp = temp.Next
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func reverseKGroup(head *ListNode, k int) *ListNode {
        res := 0
        temp := head
        for temp != nil {
                res++
                temp = temp.Next
        }
        if res < k || k <= 1 {
                return head
        }
        pre := &ListNode{}
        cur := head
        for i := 0; i < k; i++ {
                next := cur.Next
                cur.Next = pre
                pre = cur
                cur = next
        }
        head.Next = reverseKGroup(cur, k)
        return pre
}

# 3
func reverseKGroup(head *ListNode, k int) *ListNode {
        res := &ListNode{Next: head}
        prev := res
        for head != nil {
                tail := prev
                for i := 0; i < k; i++ {
                        tail = tail.Next
                        if tail == nil {
                                return res.Next
                        }
                }
                next := tail.Next
                head, tail = reverse(head, tail)
                prev.Next = head
                tail.Next = next
                prev = tail
```

(续下页)

(接上页)

```
                head = tail.Next
        }
        return res.Next
}

func reverse(head, tail *ListNode) (*ListNode, *ListNode) {
        prev := tail.Next
        temp := head
        for prev != tail {
                next := temp.Next
                temp.Next = prev
                prev = temp
                temp = next
        }
        return tail, head
}

# 4
func reverseKGroup(head *ListNode, k int) *ListNode {
        res := &ListNode{Next: head}
        prev, end := res, res
        for end.Next != nil {
                for i := 0; i < k && end != nil; i++ {
                        end = end.Next
                }
                if end == nil {
                        break
                }
                start := prev.Next          // 开始的位置
                next := end.Next            // 结束的下一个位置
                end.Next = nil              // 断开尾部连接
                prev.Next = reverse(start) // 反转后接到prev.Next
                start.Next = next           //␣
↪start的指针指向下一个开头（此时start已经是反转的最后一个节点）
                prev = start                // 已经处理后的最后一个节点
                end = prev                  // end也移动到prev
        }
        return res.Next
}

func reverse(head *ListNode) *ListNode {
        var result *ListNode
        for head != nil {
```

(续下页)

(接上页)

```
                temp := head.Next
                head.Next = result
                result = head
                head = temp
        }
        return result
}
```

3.5 30. 串联所有单词的子串 (2)

- 题目

```
给定一个字符串 s 和一些长度相同的单词 words。
找出 s 中恰好可以由 words 中所有单词串联形成的子串的起始位置。
注意子串要与 words 中的单词完全匹配，中间不能有其他字符，但不需要考虑 words␣
↪中单词串联的顺序。
示例 1: 输入: s = "barfoothefoobarman",words = ["foo","bar"] 输出: [0,9]
解释: 从索引 0 和 9 开始的子串分别是 "barfoo" 和 "foobar" 。
输出的顺序不重要, [9,0] 也是有效答案。
示例 2: 输入: s = "wordgoodgoodgoodbestword",words = ["word","good","best","word
↪"]输出: []
```

- 解题思路

```
func findSubstring(s string, words []string) []int {
        res := make([]int, 0)
        length,n := len(s),len(words)
        if length == 0 || n == 0 || len(words[0]) == 0 {
                return res
        }
        single := len(words[0])
        m := make(map[string]int)
        for i := 0; i < len(words); i++ {
                m[words[i]]++
        }
        for i := 0; i <= length-n*single; i++ {
                temp := make(map[string]int)
                for j := 0; j < n; j++ {
                        l := i + j*single
                        str := s[l : l+single]
                        if _, ok := m[str]; !ok {
                                break
```

(续下页)

(接上页)

```
                        }
                        temp[str]++
                        if temp[str] > m[str] {
                                break
                        }
                        if compare(m, temp) == true {
                                res = append(res, i)
                                break
                        }
                }
        }
        return res
}

func compare(m1, m2 map[string]int) bool {
        if len(m1) != len(m2) {
                return false
        }
        for k, v := range m1 {
                if m2[k] != v {
                        return false
                }
        }
        return true
}

# 2
func findSubstring(s string, words []string) []int {
        res := make([]int, 0)
        length := len(s)
        n := len(words)
        if length == 0 || n == 0 || len(words[0]) == 0 {
                return res
        }
        single := len(words[0])
        m := make(map[string]int)
        for i := 0; i < len(words); i++ {
                m[words[i]]++
        }
        for i := 0; i < single; i++ {
                left, right, count := i, i, 0
                temp := make(map[string]int)
                for right+single <= length {
```

(续下页)

(接上页)

```
                        str := s[right : right+single]
                        right = right + single
                        if m[str] > 0 {
                                temp[str]++
                                if temp[str] == m[str] {
                                        count++
                                }
                        }
                        if right-left == n*single {
                                if count == len(m) {
                                        res = append(res, left)
                                }
                                leftStr := s[left : left+single]
                                left = left + single
                                if m[leftStr] > 0 {
                                        if temp[leftStr] == m[leftStr] {
                                                count--
                                        }
                                        temp[leftStr]--
                                }
                        }
                }
        }
        return res
}
```

3.6 32. 最长有效括号 (4)

- 题目

```
给定一个只包含 '(' 和 ')' 的字符串，找出最长的包含有效括号的子串的长度。
示例 1:输入: "(()"输出: 2
解释: 最长有效括号子串为 "()"
示例 2:输入: ")()())" 输出: 4
解释: 最长有效括号子串为 "()()"
```

- 解题思路

```
func longestValidParentheses(s string) int {
        res := 0
        stack := make([]int, 0)
        stack = append(stack, -1)
```

(续下页)

(接上页)

```
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        stack = append(stack, i)
                } else {
                        stack = stack[:len(stack)-1] //␣
↪弹出栈顶元素表示匹配了当前右括号
                        if len(stack) == 0 {          //␣
↪没有匹配到左括号，存入最后一个没有被匹配到的右括号下标
                                stack = append(stack, i)
                        } else {
                                res = max(res, i-stack[len(stack)-1])
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestValidParentheses(s string) int {
        res := 0
        dp := make([]int, len(s))
        for i := 1; i < len(s); i++ {
                if s[i] == ')' {
                        // '()' 匹配到
                        if s[i-1] == '(' {
                                if i < 2 {
                                        dp[i] = 2
                                } else {
                                        dp[i] = dp[i-2] + 2
                                }
                        } else {
                                // '))'情况
                                if i-dp[i-1] > 0 && s[i-dp[i-1]-1] == '(' {
                                        if i-dp[i-1] < 2 {
                                                dp[i] = dp[i-1] + 2
                                        } else {
```

(续下页)

(接上页)

```
                                                dp[i] = dp[i-1] + dp[i-dp[i-1]-2] + 2
                                        }
                                }
                        }
                }
                res = max(res, dp[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func longestValidParentheses(s string) int {
        res := 0
        left, right := 0, 0
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        left++
                } else {
                        right++
                }
                if left == right {
                        res = max(res, 2*left)
                } else if right > left {
                        left, right = 0, 0
                }
        }
        left, right = 0, 0
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '(' {
                        left++
                } else {
                        right++
                }
                if left == right {
                        res = max(res, 2*left)
                } else if left > right {
```

(续下页)

(接上页)

```
                        left, right = 0, 0
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func longestValidParentheses(s string) int {
        res := 0
        for i := 0; i < len(s); i++ {
                count := 0
                for j := i; j < len(s); j++ {
                        if s[j] == '(' {
                                count++
                        } else {
                                count--
                        }
                        if count < 0 {
                                break
                        }
                        if count == 0 {
                                res = max(res, j+1-i)
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

3.7 37. 解数独 (2)

- 题目

```
编写一个程序，通过已填充的空格来解决数独问题。
一个数独的解法需遵循如下规则：
    数字 1-9 在每一行只能出现一次。
    数字 1-9 在每一列只能出现一次。
    数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。
空白格用 '.' 表示。
一个数独。
答案被标成红色。
Note:给定的数独序列只包含数字 1-9 和字符 '.' 。
    你可以假设给定的数独只有唯一解。
    给定数独永远是 9x9 形式的。
```

- 解题思路

```
var rows, cols, arrs [9][9]int

func solveSudoku(board [][]byte) {
        rows = [9][9]int{}
        cols = [9][9]int{}
        arrs = [9][9]int{}
        for i := 0; i < 9; i++ {
                for j := 0; j < 9; j++ {
                        if board[i][j] != '.' {
                                num := board[i][j] - '1'
                                index := (i/3)*3 + j/3
                                rows[i][num] = 1
                                cols[j][num] = 1
                                arrs[index][num] = 1
                        }
                }
        }
        dfs(board, 0)
}

func dfs(board [][]byte, index int) bool {
        if index == 81 {
                return true
        }
        row := index / 9
        col := index % 9
```

(续下页)

(接上页)

```
        c := (row/3)*3 + col/3
        if board[row][col] != '.' {
                return dfs(board, index+1)
        }
        for i := 0; i < 9; i++ {
                if rows[row][i] == 1 || cols[col][i] == 1 || arrs[c][i] == 1 {
                        continue
                }
                board[row][col] = byte(i + '1')
                rows[row][i], cols[col][i], arrs[c][i] = 1, 1, 1
                if dfs(board, index+1) == true {
                        return true
                }
                rows[row][i], cols[col][i], arrs[c][i] = 0, 0, 0
                board[row][col] = '.'
        }
        return false
}

# 2
func solveSudoku(board [][]byte) {
        dfs(board, 0)
}

func dfs(board [][]byte, index int) bool {
        if index == 81 {
                return true
        }
        row := index / 9
        col := index % 9
        if board[row][col] != '.' {
                return dfs(board, index+1)
        }
        for i := 0; i < 9; i++ {
                board[row][col] = byte(i + '1')
                if isValidSudoku(board) == false {
                        board[row][col] = '.'
                        continue
                }
                if dfs(board, index+1) == true {
                        return true
                }
                board[row][col] = '.'
```

(续下页)

(接上页)

```
        }
        return false
}

func isValidSudoku(board [][]byte) bool {
        var row, col, arr [9][9]int
        for i := 0; i < 9; i++ {
                for j := 0; j < 9; j++ {
                        if board[i][j] != '.' {
                                num := board[i][j] - '1'
                                index := (i/3)*3 + j/3
                                if row[i][num] == 1 || col[j][num] == 1 ||␣
↪arr[index][num] == 1 {
                                        return false
                                }
                                row[i][num] = 1
                                col[j][num] = 1
                                arr[index][num] = 1
                        }
                }
        }
        return true
}
```

3.8 41. 缺失的第一个正数 (5)

- 题目

```
给你一个未排序的整数数组，请你找出其中没有出现的最小的正整数。
示例 1:输入: [1,2,0] 输出: 3
示例 2:输入: [3,4,-1,1]输出: 2
示例 3:输入: [7,8,9,11,12]输出: 1
提示：你的算法的时间复杂度应为O(n)，并且只能使用常数级别的额外空间。
```

- 解题思路

```
func firstMissingPositive(nums []int) int {
        n := len(nums)
        for i := 0; i < n; i++ {
                // 非正数处理
                if nums[i] <= 0 {
                        nums[i] = n + 1
```

(续下页)

(接上页)

```
                }
        }
        for i := 0; i < n; i++ {
                value := abs(nums[i])
                // 标负
                if value <= n {
                        nums[value-1] = -abs(abs(nums[value-1]))
                }
        }
        for i := 0; i < n; i++ {
                if nums[i] > 0 {
                        return i + 1
                }
        }
        return n + 1
}

func abs(a int) int {
        if a >= 0 {
                return a
        }
        return -a
}

# 2
func firstMissingPositive(nums []int) int {
        n := len(nums)
        for i := 0; i < n; i++ {
                for nums[i] > 0 && nums[i] <= n && nums[nums[i]-1] != nums[i] {
                        nums[i], nums[nums[i]-1] = nums[nums[i]-1], nums[i]
                }
        }
        for i := 0; i < n; i++ {
                if nums[i] != i+1 {
                        return i + 1
                }
        }
        return n + 1
}

# 3
func firstMissingPositive(nums []int) int {
        n := len(nums)
```

(续下页)

(接上页)

```
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                m[nums[i]] = 1
        }
        for i := 1; i <= n; i++ {
                if m[i] == 0 {
                        return i
                }
        }
        return n + 1
}

# 4
func firstMissingPositive(nums []int) int {
        n := len(nums)
        for i := 1; i <= n; i++ {
                flag := false
                for j := 0; j < n; j++ {
                        if i == nums[j] {
                                flag = true
                                break
                        }
                }
                if flag == false {
                        return i
                }
        }
        return n + 1
}

# 5
func firstMissingPositive(nums []int) int {
        n := len(nums)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                if 1 <= nums[i] && nums[i] <= n {
                        arr[nums[i]] = 1
                }
        }
        for i := 1; i <= n; i++ {
                if arr[i] == 0 {
                        return i
                }
```

(续下页)

(接上页)

```
        }
        return n + 1
}
```

3.9 42. 接雨水 (4)

- 题目

```
给定 n 个非负整数表示每个宽度为 1␣
↪的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。
上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，
在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。 感谢 Marcos 贡献此图。
示例:输入: [0,1,0,2,1,0,1,3,2,1,2,1]输出: 6
```

- 解题思路

```
func trap(height []int) int {
        res := 0
        for i := 0; i < len(height); i++ {
                left, right := 0, 0
                for j := i; j >= 0; j-- {
                        left = max(left, height[j])
                }
                for j := i; j < len(height); j++ {
                        right = max(right, height[j])
                }
                // 当前坐标形成的面积=(min(左边最高，右边最高)-当前高度) * 宽度(1,
↪可省略)
                area := min(left, right) - height[i]
                res = res + area
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}

# 2
func trap(height []int) int {
        res := 0
        if len(height) == 0{
                return 0
        }
        left := make([]int, len(height))
        right := make([]int, len(height))
        left[0] = height[0]
        right[len(right)-1] = height[len(height)-1]
        for i := 1; i < len(height); i++ {
                left[i] = max(height[i], left[i-1])
        }
        for i := len(height) - 2; i >= 0; i-- {
                right[i] = max(height[i], right[i+1])
        }
        for i := 0; i < len(height); i++ {
                // 当前坐标形成的面积=(min(左边最高, 右边最高)-当前高度) * 宽度(1,
↪可省略)
                area := min(left[i], right[i]) - height[i]
                res = res + area
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
```

(续下页)

(接上页)

```
}

# 3
func trap(height []int) int {
        res := 0
        stack := make([]int, 0)
        for i := 0; i < len(height); i++ {
                for len(stack) > 0 && height[i] > height[stack[len(stack)-1]] {
                        bottom := height[stack[len(stack)-1]]
                        stack = stack[:len(stack)-1]
                        if len(stack) > 0 {
                                prev := stack[len(stack)-1]
                                // 横着的面积=长(min(height[i], height[prev])-
↪bottom)*宽(i-prev-1)
                                h := min(height[i], height[prev]) - bottom
                                w := i - prev - 1
                                area := h * w
                                res = res + area
                        }
                }
                stack = append(stack, i)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func trap(height []int) int {
        res := 0
        if len(height) == 0 {
                return 0
        }
        left := 0
        right := len(height) - 1
        leftMax := 0  // 左边的最大值
        rightMax := 0 // 右边的最大值
        for left < right {
```

(续下页)

(接上页)

```
                // 当前坐标形成的面积=(min(左边最高, 右边最高)-当前高度) * 宽度(1,
↪可省略)
                // 选择高度低的一边处理并求最大值, 说明当前侧最大值小于另一侧
                if height[left] < height[right] {
                        // 也可以写成这样
                        // leftMax = max(leftMax, height[left])
                        // res = res + leftMax - height[left]
                        if height[left] >= leftMax { // 递增无法蓄水
                                leftMax = height[left]
                        } else {
                                res = res + leftMax - height[left]
                        }
                        left++
                } else {
                        // 也可以写成这样
                        // rightMax = max(rightMax, height[right])
                        // res = res + rightMax - height[right]
                        if height[right] >= rightMax { // 递减无法蓄水
                                rightMax = height[right]
                        } else {
                                res = res + rightMax - height[right]
                        }
                        right--
                }
        }
        return res
}
```

3.10 44. 通配符匹配 (3)

- 题目

```
给定一个字符串 (s) 和一个字符模式 (p) , 实现一个支持 '?' 和 '*' 的通配符匹配。
'?' 可以匹配任何单个字符。
'*' 可以匹配任意字符串 (包括空字符串) 。
两个字符串完全匹配才算匹配成功。
说明:s 可能为空, 且只包含从 a-z 的小写字母。
    p 可能为空, 且只包含从 a-z 的小写字母, 以及字符 ? 和 *。
示例 1: 输入:s = "aa" p = "a" 输出: false
解释: "a" 无法匹配 "aa" 整个字符串。
示例 2: 输入: s = "aa" p = "*" 输出: true
解释: '*' 可以匹配任意字符串。
```

(续下页)

(接上页)

```
示例 3:输入: s = "cb" p = "?a" 输出: false
解释: '?' 可以匹配 'c', 但第二个 'a' 无法匹配 'b'。
示例 4:输入:s = "adceb" p = "*a*b" 输出: true
解释: 第一个 '*' 可以匹配空字符串, 第二个 '*' 可以匹配字符串 "dce".
示例 5:输入:s = "acdcb" p = "a*c?b" 输出: false
```

- 解题思路

```
func isMatch(s string, p string) bool {
        n, m := len(s), len(p)
        dp := make([][]bool, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]bool, m+1)
        }
        dp[0][0] = true
        for i := 1; i <= m; i++ {
                if p[i-1] == '*' { // 可以匹配任意字符串(包括空字符串)
                        dp[0][i] = true
                } else {
                        break
                }
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if p[j-1] == '*' {
                                // dp[i][j-1]=>不使用这个*, dp[i-1][j]=>使用这个*
                                dp[i][j] = dp[i][j-1] || dp[i-1][j]
                        } else if p[j-1] == '?' || s[i-1] == p[j-1] {
                                dp[i][j] = dp[i-1][j-1]
                        }
                }
        }
        return dp[n][m]
}

# 2
var dp [][]int

func isMatch(s string, p string) bool {
        n, m := len(s), len(p)
        dp = make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m+1)
        }
```

(续下页)

(接上页)

```
        return dfs(s, p, 0, 0)
}

func dfs(s, p string, i, j int) bool {
        if i == len(s) && j == len(p) {
                return true
        }
        if dp[i][j] > 0 {
                if dp[i][j] == 1 {
                        return false
                } else {
                        return true
                }
        }
        if i >= len(s) {
                return p[j] == '*' && dfs(s, p, i, j+1)
        }
        if j >= len(p) {
                return false
        }
        res := false
        if p[j] == '*' {
                res = dfs(s, p, i+1, j) || dfs(s, p, i, j+1)
        } else {
                res = (s[i] == p[j] || p[j] == '?') && dfs(s, p, i+1, j+1)
        }
        if res == true {
                dp[i][j] = 2
        } else {
                dp[i][j] = 1
        }
        return res
}

# 3
func isMatch(s string, p string) bool {
        i, j := 0, 0
        start, last := 0, 0
        for i = 0; i < len(s); {
                if j < len(p) && (s[i] == p[j] || p[j] == '?') {
                        i++
                        j++
                } else if j < len(p) && p[j] == '*' {
```

(续下页)

(接上页)

```
                        last = i  // 记录s的位置
            j++
                        start = j // 记录*的位置
                } else if start != 0 {
                        last++
                        i = last // 更新到记录位置的下一个
                        j = start
                } else {
                        return false
                }
        }
        for ; j < len(p) && p[j] == '*'; j++ {
        }
        return j == len(p)
}
```

3.11 45. 跳跃游戏 II(4)

- 题目

```
给定一个非负整数数组，你最初位于数组的第一个位置。
数组中的每个元素代表你在该位置可以跳跃的最大长度。
你的目标是使用最少的跳跃次数到达数组的最后一个位置。
示例:输入: [2,3,1,1,4] 输出: 2
解释: 跳到最后一个位置的最小跳跃数是 2。
     从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。
说明:假设你总是可以到达数组的最后一个位置。
```

- 解题思路

```
func jump(nums []int) int {
        last := len(nums) - 1
        res := 0
        for last > 0 {
                // 从前往后，找到第一个一步能走到终点的，更新终点的位置
                for i := 0; i < last; i++ {
                        if i+nums[i] >= last {
                                last = i
                                res++
                                break
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

# 2
func jump(nums []int) int {
        res := 0
        end := 0
        maxValue := 0
        for i := 0; i < len(nums)-1; i++ {
                maxValue = max(maxValue, i+nums[i])
                if i == end {
                        end = maxValue
                        res++
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func jump(nums []int) int {
        dp := make([]int, len(nums))
        dp[0] = 0
        for i := 1; i < len(nums); i++ {
                dp[i] = i
                for j := 0; j < i; j++ {
                        if nums[j]+j >= i {
                                dp[i] = min(dp[i], dp[j]+1)
                        }
                }
        }
        return dp[len(nums)-1]
}

func min(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return b
        }
        return a
}

# 4
func jump(nums []int) int {
        if len(nums) <= 1 {
                return 0
        }
        dp := make([]int, len(nums))
        for i := 1; i < len(nums); i++ {
                dp[i] = math.MaxInt32
        }
        dp[0] = 0
        for i := 0; i < len(nums)-1; i++ {
                if i+nums[i] >= len(nums)-1 {
                        return dp[i] + 1
                }
                for j := i + 1; j <= i+nums[i]; j++ {
                        if j < len(nums) {
                                dp[j] = min(dp[j], dp[i]+1)
                        } else {
                                break
                        }
                }
        }
        return dp[len(nums)-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

3.12 51.N 皇后 (3)

- 题目

```
n 皇后问题研究的是如何将 n 个皇后放置在 n×n 的棋盘上，并且使皇后彼此之间不能相互攻击。
上图为 8 皇后问题的一种解法。
给定一个整数 n，返回所有不同的 n 皇后问题的解决方案。
每一种解法包含一个明确的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.'␣
↪分别代表了皇后和空位。
示例:
输入: 4
输出: [
 [".Q..",  // 解法 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // 解法 2
  "Q...",
  "...Q",
  ".Q.."]
]
解释: 4皇后问题存在两个不同的解法。
提示:
    皇后，是国际象棋中的棋子，意味着国王的妻子。皇后只做一件事，那就是“吃子”。
  ␣
↪当她遇见可以吃的棋子时，就迅速冲上去吃掉棋子。当然，她横、竖、斜都可走一到七步，可进可退。
```

- 解题思路

```
var res [][]string

func solveNQueens(n int) [][]string {
        res = make([][]string, 0)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0)
```

(续下页)

(接上页)

```
        return res
}

func dfs(arr [][]string, row int) {
        if len(arr) == row {
                temp := make([]string, 0)
                for i := 0; i < len(arr); i++ {
                        str := ""
                        for j := 0; j < len(arr[i]); j++ {
                                str = str + arr[i][j]
                        }
                        temp = append(temp, str)
                }
                res = append(res, temp)
                return
        }
        // 每列尝试
        for col := 0; col < len(arr[0]); col++ {
                if valid(arr, row, col) == false {
                        continue
                }
                arr[row][col] = "Q"
                dfs(arr, row+1)
                arr[row][col] = "."
        }
}

func valid(arr [][]string, row, col int) bool {
        n := len(arr)
        // 当前列判断(竖着)
        for row := 0; row < n; row++ {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        // 左上角
        for row, col := row-1, col-1; row >= 0 && col >= 0; row, col = row-1, col-1 {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        // 右上角
        for row, col := row-1, col+1; row >= 0 && col < n; row, col = row-1, col+1 {
```

(续下页)

(接上页)

```
                if arr[row][col] == "Q" {
                        return false
                }
        }
        return true
}

# 2
var res [][]string
var rows, left, right []bool

func solveNQueens(n int) [][]string {
        res = make([][]string, 0)
        rows, left, right = make([]bool, n), make([]bool, 2*n-1), make([]bool, 2*n-1)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0)
        return res
}

func dfs(arr [][]string, row int) {
        n := len(arr)
        if len(arr) == row {
                temp := make([]string, 0)
                for i := 0; i < n; i++ {
                        str := ""
                        for j := 0; j < n; j++ {
                                str = str + arr[i][j]
                        }
                        temp = append(temp, str)
                }
                res = append(res, temp)
                return
        }
        // 每列尝试
        for col := 0; col < n; col++ {
```

(续下页)

(接上页)

```
                if rows[col] == true || left[row-col+n-1] == true || right[row+col]
↪== true {
                        continue
                }
                rows[col], left[row-col+n-1], right[row+col] = true, true, true
                arr[row][col] = "Q"
                dfs(arr, row+1)
                arr[row][col] = "."
                rows[col], left[row-col+n-1], right[row+col] = false, false, false
        }
}

# 3
var res [][]string

func solveNQueens(n int) [][]string {
        res = make([][]string, 0)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0, 0, 0, 0)
        return res
}

func dfs(arr [][]string, row int, rows, left, right int) {
        n := len(arr)
        if len(arr) == row {
                temp := make([]string, 0)
                for i := 0; i < n; i++ {
                        str := ""
                        for j := 0; j < n; j++ {
                                str = str + arr[i][j]
                        }
                        temp = append(temp, str)
                }
                res = append(res, temp)
                return
```

(续下页)

(接上页)

```
        }
        // 每列尝试
        for col := 0; col < n; col++ {
                a := uint(col)
                b := uint(row - col + n - 1)
                c := uint(row + col)
                if ((rows>>a)&1) != 0 || ((left>>b)&1) != 0 || ((right>>c)&1) != 0 {
                        continue
                }
                arr[row][col] = "Q"
                dfs(arr, row+1, rows^(1<<a), left^(1<<b), right^(1<<c))
                arr[row][col] = "."
        }
}
```

3.13 52.N 皇后 II(3)

- 题目

```
n 皇后问题研究的是如何将 n 个皇后放置在 n×n 的棋盘上，并且使皇后彼此之间不能相互攻击。
上图为 8 皇后问题的一种解法。
给定一个整数 n，返回 n 皇后不同的解决方案的数量。
示例:输入：4 输出：2 解释：4 皇后问题存在如下两个不同的解法。
[
 [".Q..",  // 解法 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // 解法 2
  "Q...",
  "...Q",
  ".Q.."]
]
提示：皇后，是国际象棋中的棋子，意味着国王的妻子。皇后只做一件事，那就是“吃子”。
    当她遇见可以吃的棋子时，就迅速冲上去吃掉棋子。当然，她横、竖、斜都可走一或 N-1步，可进可退。
```

- 解题思路

```
var res int
var rows, left, right []bool
```

(续下页)

(接上页)

```
func totalNQueens(n int) int {
        res = 0
        rows, left, right = make([]bool, n), make([]bool, 2*n-1), make([]bool, 2*n-1)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0)
        return res
}

func dfs(arr [][]string, row int) {
        n := len(arr)
        if len(arr) == row {
                res++
                return
        }
        // 每列尝试
        for col := 0; col < n; col++ {
                if rows[col] == true || left[row-col+n-1] == true || right[row+col]
↪== true {
                        continue
                }
                rows[col], left[row-col+n-1], right[row+col] = true, true, true
                arr[row][col] = "Q"
                dfs(arr, row+1)
                arr[row][col] = "."
                rows[col], left[row-col+n-1], right[row+col] = false, false, false
        }
}

# 2
var res int

func totalNQueens(n int) int {
        res = 0
        // 初始化棋盘
```

(续下页)

(接上页)

```
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0)
        return res
}

func dfs(arr [][]string, row int) {
        if len(arr) == row {
                res++
                return
        }
        // 每列尝试
        for col := 0; col < len(arr[0]); col++ {
                if valid(arr, row, col) == false {
                        continue
                }
                arr[row][col] = "Q"
                dfs(arr, row+1)
                arr[row][col] = "."
        }
}

func valid(arr [][]string, row, col int) bool {
        n := len(arr)
        // 当前列判断(竖着)
        for row := 0; row < n; row++ {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        // 左上角
        for row, col := row-1, col-1; row >= 0 && col >= 0; row, col = row-1, col-1 {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        // 右上角
```

(续下页)

(接上页)

```
        for row, col := row-1, col+1; row >= 0 && col < n; row, col = row-1, col+1 {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        return true
}

# 3
var res int

func totalNQueens(n int) int {
        res = 0
        // 从第1行开始,上层是满足条件
        dfs(0, n, 0, 0, 0)
        return res
}

func dfs(row, n int, rows, left, right int) {
        if n == row {
                res++
                return
        }
        // 每列尝试
        for col := 0; col < n; col++ {
                a := uint(col)
                b := uint(row - col + n - 1)
                c := uint(row + col)
                if ((rows>>a)&1) != 0 || ((left>>b)&1) != 0 || ((right>>c)&1) != 0 {
                        continue
                }
                dfs(row+1, n, rows^(1<<a), left^(1<<b), right^(1<<c))
        }
}
```

3.14 57. 插入区间 (3)

- 题目

给出一个无重叠的 ，按照区间起始端点排序的区间列表。
在列表中插入一个新的区间，你需要确保列表中的区间仍然有序且不重叠（如果有必要的话，可以合并区间）。
示例 1: 输入: intervals = [[1,3],[6,9]], newInterval = [2,5] 输出: [[1,5],[6,9]]
示例 2: 输入: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
输出: [[1,2],[3,10],[12,16]]
解释: 这是因为新的区间 [4,8] 与 [3,5],[6,7],[8,10] 重叠。
注意: 输入类型已在 2019 年 4 月 15 日更改。请重置为默认代码定义以获取新的方法签名。

- 解题思路

```
func insert(intervals [][]int, newInterval []int) [][]int {
        res := make([][]int, 0)
        if len(intervals) == 0 {
                res = append(res, newInterval)
                return res
        }
        i := 0
        for ; i < len(intervals) && intervals[i][1] < newInterval[0]; i++ {
                res = append(res, intervals[i])
        }
        for ; i < len(intervals) && intervals[i][0] <= newInterval[1]; i++ {
                newInterval[0] = min(newInterval[0], intervals[i][0])
                newInterval[1] = max(newInterval[1], intervals[i][1])
        }
        res = append(res, newInterval)
        for ; i < len(intervals); i++ {
                res = append(res, intervals[i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}

# 2
func insert(intervals [][]int, newInterval []int) [][]int {
        if len(intervals) == 0 {
                return [][]int{newInterval}
        }
        i := 0
        for ; i < len(intervals) && intervals[i][1] < newInterval[0]; i++ {
        }
        left := i
        i = len(intervals) - 1
        for ; i >= 0 && intervals[i][0] > newInterval[1]; i-- {
        }
        right := i
        if left > right {
                return append(intervals[:left], append([][]int{newInterval},␣
↪intervals[left:]...)...)
        }
        newInterval[0] = min(newInterval[0], intervals[left][0])
        newInterval[1] = max(newInterval[1], intervals[right][1])
        return append(intervals[:left], append([][]int{newInterval},␣
↪intervals[right+1:]...)...)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
```

(续下页)

(接上页)

```
func insert(intervals [][]int, newInterval []int) [][]int {
        res := make([][]int, 0)
        intervals = append(intervals, newInterval)
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][0] < intervals[j][0]
        })
        res = append(res, intervals[0])
        for i := 1; i < len(intervals); i++ {
                arr := res[len(res)-1]
                if intervals[i][0] > arr[1] {
                        res = append(res, intervals[i])
                } else if intervals[i][1] > arr[1] {
                        res[len(res)-1][1] = intervals[i][1]
                }
        }
        return res
}
```

3.15 65. 有效数字 (1)

- 题目

```
验证给定的字符串是否可以解释为十进制数字。
例如:
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
" -90e3   " => true
" 1e" => false
"e3" => false
" 6e-1" => true
" 99e2.5 " => false
"53.5e93" => true
" --6 " => false
"-+3" => false
"95a54e53" => false
说明: 我们有意将问题陈述地比较模糊。在实现代码之前，你应当事先思考所有可能的情况。
这里给出一份可能存在于有效十进制数字中的字符列表:
    数字 0-9
    指数 - "e"
```

(续下页)

(接上页)

```
    正/负号 - "+"/"-"
    小数点 - "."
当然，在输入中，这些字符的上下文也很重要。
```

- 解题思路

```
func isNumber(s string) bool {
        s = strings.Trim(s, " ")
        if s == "" || len(s) == 0 || len(s) == 0 {
                return false
        }
        arr := []byte(s)
        i := 0
        numeric := scanInteger(&arr, &i)
        if i < len(arr) && arr[i] == '.' {
                i++
                numeric = scanUnsignedInteger(&arr, &i) || numeric
        }
        if i < len(arr) && (arr[i] == 'e' || arr[i] == 'E') {
                i++
                numeric = numeric && scanInteger(&arr, &i)
        }
        return numeric && len(arr) == i
}

func scanInteger(arr *[]byte, index *int) bool {
        if len(*arr) <= *index {
                return false
        }
        if (*arr)[*index] == '+' || (*arr)[*index] == '-' {
                *index++
        }
        return scanUnsignedInteger(arr, index)
}

func scanUnsignedInteger(arr *[]byte, index *int) bool {
        j := *index
        for *index < len(*arr) {
                if (*arr)[*index] < '0' || (*arr)[*index] > '9' {
                        break
                }
                *index++
        }
        return j < *index
```

(续下页)

(接上页)

```
}
```

3.16 68. 文本左右对齐 (1)

- 题目

```
给定一个单词数组和一个长度 maxWidth，重新排版单词，使其成为每行恰好有 maxWidth␣
↪个字符，
且左右两端对齐的文本。
你应该使用“贪心算法”来放置给定的单词；也就是说，尽可能多地往每行中放置单词。
必要时可用空格 ' ' 填充，使得每行恰好有 maxWidth 个字符。
要求尽可能均匀分配单词间的空格数量。
如果某一行单词间的空格不能均匀分配，则左侧放置的空格数要多于右侧的空格数。
文本的最后一行应为左对齐，且单词之间不插入额外的空格。
说明:单词是指由非空格字符组成的字符序列。
    每个单词的长度大于 0，小于等于 maxWidth。
    输入单词数组 words 至少包含一个单词。
示例:输入: words = ["This", "is", "an", "example", "of", "text", "justification."]
maxWidth = 16
输出:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
示例 2:输入: words = ["What","must","be","acknowledgment","shall","be"] maxWidth = 16
输出:
[
  "What   must   be",
  "acknowledgment  ",
  "shall be        "
]
解释: 注意最后一行的格式应为 "shall be    " 而不是 "shall     be",
     因为最后一行应为左对齐，而不是左右两端对齐。
     第二行同样为左对齐，这是因为这行只包含一个单词。
示例 3:
输入:words = ["Science","is","what","we","understand","well","enough","to","explain",
         "to","a","computer.","Art","is","everything","else","we","do"]
maxWidth = 20
输出:
[
  "Science  is  what we",
```

(续下页)

(接上页)

```
  "understand      well",
  "enough to explain to",
  "a  computer.  Art is",
  "everything  else  we",
  "do                  "
]
```

- 解题思路

```
func fullJustify(words []string, maxWidth int) []string {
        res := make([]string, 0)
        count := 0
        start := 0
        for i := 0; i < len(words); i++ {
                count = count + len(words[i])
                if count > maxWidth {
                        temp := justify(words, start, i-1, maxWidth)
                        res = append(res, temp)
                        start = i
                        if i == len(words)-1 {
                                count = 0
                                i--
                        } else {
                                count = len(words[i]) + 1
                        }
                } else if i == len(words)-1 {
                        temp := justify(words, start, i, maxWidth)
                        res = append(res, temp)
                } else {
                        count++
                }
        }
        return res
}

func justify(words []string, start, end int, maxWidth int) string {
        arr := make([]byte, maxWidth)
        for i := 0; i < len(arr); i++ {
                arr[i] = byte(' ')
        }
        index := 0
        // 文本的最后一行应为左对齐，且单词之间不插入额外的空格。
        if start == end || end == len(words)-1 {
                for i := start; i <= end; i++ {
```

(续下页)

(接上页)

```
                        copy(arr[index:], words[i])
                        index = index + len(words[i]) + 1
                }
        } else {
                // 要求尽可能均匀分配单词间的空格数量。
                //␣
↪如果某一行单词间的空格不能均匀分配，则左侧放置的空格数要多于右侧的空格数。
                count := end - start + 1
                left := maxWidth - count + 1
                for i := start; i <= end; i++ {
                        left = left - len(words[i])
                }
                space := left / (count - 1) // 均分
                mod := left % (count - 1)   // 多的放左边
                for i := start; i <= end; i++ {
                        copy(arr[index:], words[i])
                        index = index + len(words[i]) + 1 + space
                        if mod > 0 {
                                index++
                                mod--
                        }
                }
        }
        return string(arr)
}
```

3.17 72. 编辑距离 (2)

- 题目

```
给你两个单词 word1 和 word2，请你计算出将 word1 转换成 word2 所使用的最少操作数 。
你可以对一个单词进行如下三种操作：
    插入一个字符
    删除一个字符
    替换一个字符
示例 1：输入：word1 = "horse", word2 = "ros" 输出：3
解释：horse -> rorse (将 'h' 替换为 'r')
rorse -> rose (删除 'r')
rose -> ros (删除 'e')
示例 2：输入：word1 = "intention", word2 = "execution" 输出：5
解释：  intention -> inention (删除 't')
inention -> enention (将 'i' 替换为 'e')
```

(续下页)

(接上页)

```
enention -> exention (将 'n' 替换为 'x')
exention -> exection (将 'n' 替换为 'c')
exection -> execution (插入 'u')
```

- 解题思路

```
func minDistance(word1 string, word2 string) int {
        n1 := len(word1)
        n2 := len(word2)
        // dp[i][j]代表 word1的i位置转换成word2的j位置需要最少步数
        dp := make([][]int, n1+1)
        for i := 0; i < n1+1; i++ {
                dp[i] = make([]int, n2+1)
        }
        dp[0][0] = 0
        // 到word2[0]需要全部删除，有多少删除多少
        for i := 1; i <= n1; i++ {
                dp[i][0] = i
        }
        // 到word2[i]需要添加，有多少添加多少
        for i := 1; i <= n2; i++ {
                dp[0][i] = i
        }
        for i := 1; i <= n1; i++ {
                for j := 1; j <= n2; j++ {
                        if word1[i-1] == word2[j-1] {
                                dp[i][j] = dp[i-1][j-1] // 相同不需要操作
                        } else {
                                dp[i][j] = dp[i-1][j-1] + 1             // 替换
                                dp[i][j] = min(dp[i][j], dp[i][j-1]+1) // 插入
                                dp[i][j] = min(dp[i][j], dp[i-1][j]+1) // 删除
                        }
                }
        }
        return dp[n1][n2]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
# 2
var dp [][]int

func minDistance(word1 string, word2 string) int {
        dp = make([][]int, len(word1)+1)
        for i := 0; i < len(word1)+1; i++ {
                dp[i] = make([]int, len(word2)+1)
        }
        return helper(word1, word2, 0, 0)
}

func helper(word1, word2 string, i, j int) int {
        if dp[i][j] > 0 {
                return dp[i][j]
        }
        if i == len(word1) || j == len(word2) {
                return len(word1) - i + len(word2) - j
        }
        if word1[i] == word2[j] {
                return helper(word1, word2, i+1, j+1)
        }
        inserted := helper(word1, word2, i, j+1)
        deleted := helper(word1, word2, i+1, j)
        replaced := helper(word1, word2, i+1, j+1)
        dp[i][j] = min(inserted, min(deleted, replaced)) + 1
        return dp[i][j]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

3.18 76. 最小覆盖子串 (2)

- 题目

```
给你一个字符串 S、一个字符串 T 。
请你设计一种算法，可以在 O(n) 的时间复杂度内，从字符串 S 里面找出：包含 T␣
↪所有字符的最小子串。
示例：输入：S = "ADOBECODEBANC", T = "ABC" 输出："BANC"
提示：如果 S 中不存这样的子串，则返回空字符串 ""。
    如果 S 中存在这样的子串，我们保证它是唯一的答案。
```

- 解题思路

```
func minWindow(s string, t string) string {
        if len(s) < len(t) {
                return ""
        }
        window := make(map[byte]int)
        need := make(map[byte]int)
        for i := 0; i < len(t); i++ {
                need[t[i]]++
        }
        left, right := -1, -1
        minLength := math.MaxInt32
        for l, r := 0, 0; r < len(s); r++ {
                if r < len(s) && need[s[r]] > 0 {
                        window[s[r]]++
                }
                // 找到，然后left往右移
                for check(need, window) == true && l <= r {
                        if r-l+1 < minLength {
                                minLength = r - l + 1
                                left, right = l, r+1
                        }
                        if _, ok := need[s[l]]; ok {
                                window[s[l]]--
                        }
                        l++
                }
        }
        if left == -1 {
                return ""
        }
        return s[left:right]
```

(续下页)

(接上页)

```
}

func check(need, window map[byte]int) bool {
        for k, v := range need {
                if window[k] < v {
                        return false
                }
        }
        return true
}

# 2
func minWindow(s string, t string) string {
        if len(s) < len(t) {
                return ""
        }
        arr := make(map[byte]int)
        for i := 0; i < len(t); i++ {
                arr[t[i]]++
        }
        l, count := 0, 0
        res := ""
        minLength := math.MaxInt32
        for r := 0; r < len(s); r++ {
                arr[s[r]]--
                if arr[s[r]] >= 0 {
                        count++
                }
                // left往右边移动
                for count == len(t) {
                        if minLength > r-l+1 {
                                minLength = r - l + 1
                                res = s[l : r+1]
                        }
                        arr[s[l]]++
                        if arr[s[l]] > 0 {
                                count--
                        }
                        l++
                }
        }
        return res
}
```

3.19 84. 柱状图中最大的矩形 (5)

- 题目

给定 n 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。
求在该柱状图中，能够勾勒出来的矩形的最大面积。
以上是柱状图的示例，其中每个柱子的宽度为 1，给定的高度为 [2,1,5,6,2,3]。
图中阴影部分为所能勾勒出的最大矩形面积，其面积为 10 个单位。
示例:输入: [2,1,5,6,2,3]输出: 10

- 解题思路

```
func largestRectangleArea(heights []int) int {
        n := len(heights)
        res := 0
        for i := 0; i < n; i++ {
                height := heights[i]
                for j := i; j < n; j++ {
                        width := j - i + 1
                        height = min(height, heights[j])
                        res = max(res, width*height)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func largestRectangleArea(heights []int) int {
        n := len(heights)
        res := 0
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                height := heights[i]
                left, right := i, i
                for left > 0 && heights[left-1] >= height {
                        left--
                }
                for right < n-1 && heights[right+1] >= height {
                        right++
                }
                width := right - left + 1
                res = max(res, width*height)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func largestRectangleArea(heights []int) int {
        n := len(heights)
        res := 0
        left := make([]int, n)
        right := make([]int, n)
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                for len(stack) > 0 && heights[stack[len(stack)-1]] >= heights[i] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        left[i] = -1
                } else {
                        left[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }
        stack = make([]int, 0)
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 && heights[stack[len(stack)-1]] >= heights[i] {
```

(续下页)

(接上页)

```
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        right[i] = n
                } else {
                        right[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }
        for i := 0; i < n; i++ {
                res = max(res, heights[i]*(right[i]-left[i]-1))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func largestRectangleArea(heights []int) int {
        n := len(heights)
        res := 0
        left := make([]int, n)
        right := make([]int, n)
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                right[i] = n
        }
        for i := 0; i < n; i++ {
                for len(stack) > 0 && heights[stack[len(stack)-1]] >= heights[i] {
                        right[stack[len(stack)-1]] = i
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        left[i] = -1
                } else {
                        left[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
```

(续下页)

(接上页)

```
        }

        for i := 0; i < n; i++ {
                res = max(res, heights[i]*(right[i]-left[i]-1))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 5
func largestRectangleArea(heights []int) int {
        heights = append([]int{0}, heights...)
        heights = append(heights, 0)
        n := len(heights)
        res := 0
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                // 递增栈
                for len(stack) > 0 && heights[stack[len(stack)-1]] > heights[i] {
                        height := heights[stack[len(stack)-1]]
                        stack = stack[:len(stack)-1]
                        width := i - stack[len(stack)-1] - 1
                        res = max(res, height*width)
                }
                stack = append(stack, i)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

3.20 85. 最大矩形 (2)

- 题目

```
给定一个仅包含 0 和 1 的二维二进制矩阵，找出只包含 1 的最大矩形，并返回其面积。
示例:输入:
[
  ["1","0","1","0","0"],
  ["1","0","1","1","1"],
  ["1","1","1","1","1"],
  ["1","0","0","1","0"]
]
输出: 6
```

- 解题思路

```
func maximalRectangle(matrix [][]byte) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        height := make([]int, m) // 高度
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '0' {
                                height[j] = 0
                        } else {
                                height[j] = height[j] + 1
                        }
                }
                res = max(res, getMaxArea(height))
        }
        return res
}

func getMaxArea(heights []int) int {
        heights = append([]int{0}, heights...)
        heights = append(heights, 0)
        n := len(heights)
        res := 0
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                // 递增栈
```

(续下页)

(接上页)

```
                for len(stack) > 0 && heights[stack[len(stack)-1]] > heights[i] {
                        height := heights[stack[len(stack)-1]]
                        stack = stack[:len(stack)-1]
                        width := i - stack[len(stack)-1] - 1
                        res = max(res, height*width)
                }
                stack = append(stack, i)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maximalRectangle(matrix [][]byte) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        left, right, height := make([]int, m), make([]int, m), make([]int, m)
        for i := 0; i < m; i++ {
                right[i] = m
        }
        for i := 0; i < n; i++ {
                curLeft, curRight := 0, m
                // 高度
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '1' {
                                height[j]++
                        } else {
                                height[j] = 0
                        }
                }
                // 左边
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '1' {
                                left[j] = max(left[j], curLeft)
```

(续下页)

(接上页)

```
                        } else {
                                left[j] = 0
                                curLeft = j + 1
                        }
                }
                // 右边
                for j := m - 1; j >= 0; j-- {
                        if matrix[i][j] == '1' {
                                right[j] = min(right[j], curRight)
                        } else {
                                right[j] = m
                                curRight = j
                        }
                }
                for j := 0; j < m; j++ {
                        res = max(res, height[j]*(right[j]-left[j]))
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

3.21 87. 扰乱字符串 (2)

- 题目

```
给定一个字符串 s1，我们可以把它递归地分割成两个非空子字符串，从而将其表示为二叉树。
下图是字符串 s1 = "great" 的一种可能的表示形式。
    great
   /    \
  gr    eat
 / \    /  \
g   r  e   at
           / \
          a   t
在扰乱这个字符串的过程中，我们可以挑选任何一个非叶节点，然后交换它的两个子节点。
例如，如果我们挑选非叶节点 "gr" ，交换它的两个子节点，将会产生扰乱字符串 "rgeat" 。
    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t
我们将 "rgeat” 称作 "great" 的一个扰乱字符串。
同样地，如果我们继续交换节点 "eat" 和 "at" 的子节点，将会产生另一个新的扰乱字符串
↪"rgtae" 。
    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta  e
       / \
      t   a
我们将 "rgtae” 称作 "great" 的一个扰乱字符串。
给出两个长度相等的字符串 s1 和 s2，判断 s2 是否是 s1 的扰乱字符串。
示例 1:输入: s1 = "great", s2 = "rgeat" 输出: true
示例 2:输入: s1 = "abcde", s2 = "caebd" 输出: false
```

- 解题思路

```
func isScramble(s1 string, s2 string) bool {
        n, m := len(s1), len(s2)
        if n != m {
                return false
        }
```

(续下页)

(接上页)

```
        // dp[i][j][l]:表示s1从i开始，s2从j开始长度为l的两个子字符串是扰乱
        dp := make([][][]bool, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([][]bool, n+1)
                for j := 0; j <= n; j++ {
                        dp[i][j] = make([]bool, n+1)
                }
        }
        // 单个字符
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        dp[i][j][1] = s1[i] == s2[j]
                }
        }
        for k := 2; k <= n; k++ { // 枚举长度: 2-n
                for i := 0; i <= n-k; i++ { // s1起点
                        for j := 0; j <= n-k; j++ { // s2起点
                                dp[i][j][k] = false
                                // 长度为w，分为两部分，其中最少是1
                                for w := 1; w <= k-1; w++ {
                                        // 划分不交换：S1->T1, S2->T2
                                        // 划分交换： S1->T2, S2->T1
                                        if (dp[i][j][w] == true && dp[i+w][j+w][k-w]␣
↪== true) ||
                                                (dp[i][j+k-w][w] == true &&␣
↪dp[i+w][j][k-w] == true) {
                                                dp[i][j][k] = true
                                        }
                                }
                        }
                }
        }
        return dp[0][0][n]
}

# 2
func isScramble(s1 string, s2 string) bool {
        return dfs([]byte(s1), []byte(s2))
}

func dfs(arr1, arr2 []byte) bool {
        if compare(arr1, arr2) == false {
                return false
```

(续下页)

(接上页)

```
        }
        if len(arr1) <= 2 {
                return (len(arr1) == 2 && ((arr1[0] == arr2[0] && arr1[1] == arr2[1])||
                        (arr1[0] == arr2[1] && arr1[1] == arr2[0]))) ||
                        (len(arr1) == 1 && arr1[0] == arr2[0])
        }
        for i := 1; i < len(arr1); i++ {
                leftA, rightA := arr1[:i], arr1[i:]
                leftB, rightB := arr2[:i], arr2[i:]
                LB, RB := arr2[len(arr1)-i:], arr2[:len(arr1)-i]
                if (dfs(leftA, leftB) && dfs(rightA, rightB)) || (dfs(leftA, LB) &&dfs(rightA, RB)) {
                        return true
                }
        }
        return false
}

func compare(arr1, arr2 []byte) bool {
        if len(arr1) != len(arr2) {
                return false
        }
        arrA := make([]byte, 26)
        arrB := make([]byte, 26)
        for i := 0; i < len(arr1); i++ {
                arrA[arr1[i]-'a']++
                arrB[arr2[i]-'a']++
        }
        for i := 0; i < len(arrA); i++ {
                if arrA[i] != arrB[i] {
                        return false
                }
        }
        return true
}
```

3.22 97. 交错字符串 (3)

- 题目

```
给定三个字符串 s1, s2, s3, 验证 s3 是否是由 s1 和 s2 交错组成的。
示例 1: 输入: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac" 输出: true
示例 2: 输入: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc" 输出: false
```

- 解题思路

```
func isInterleave(s1 string, s2 string, s3 string) bool {
        n, m, t := len(s1), len(s2), len(s3)
        if n+m != t {
                return false
        }
        // dp[i][j]表示s1的前i个元素和s2的前j个元素是否能交错组成s3的前i+j个元素
        dp := make([][]bool, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]bool, m+1)
        }
        dp[0][0] = true
        for i := 0; i <= n; i++ {
                for j := 0; j <= m; j++ {
                        total := i + j - 1
                        if i > 0 && dp[i-1][j] == true && s1[i-1] == s3[total] {
                                dp[i][j] = true
                        }
                        if j > 0 && dp[i][j-1] == true && s2[j-1] == s3[total] {
                                dp[i][j] = true
                        }
                }
        }
        return dp[n][m]
}

# 2
func isInterleave(s1 string, s2 string, s3 string) bool {
        n, m, t := len(s1), len(s2), len(s3)
        if n+m != t {
                return false
        }
        // dp[j]表示s1的前i个元素和s2的前j个元素是否能交错组成s3的前i+j个元素
        dp := make([]bool, m+1)
        dp[0] = true
```

(续下页)

(接上页)

```
        for i := 0; i <= n; i++ {
                for j := 0; j <= m; j++ {
                        total := i + j - 1
                        if i > 0 {
                                if dp[j] == true && s1[i-1] == s3[total] {
                                        dp[j] = true
                                } else {
                                        dp[j] = false
                                }
                        }
                        if j > 0 {
                                if dp[j] == true || (dp[j-1] == true && s2[j-1] ==␣
↪s3[total]) {
                                        dp[j] = true
                                } else {
                                        dp[j] = false
                                }
                        }
                }
        }
        return dp[m]
}

# 3
func isInterleave(s1 string, s2 string, s3 string) bool {
        if len(s1)+len(s2) != len(s3) {
                return false
        }
        return dfs(s1, s2, s3, 0, 0, 0)
}

func dfs(s1, s2, s3 string, i, j, k int) bool {
        if k == len(s3) && i == len(s1) && j == len(s2) {
                return true
        }
        if k >= len(s3) {
                return false
        }
        if i < len(s1) {
                if s1[i] == s3[k] {
                        if dfs(s1, s2, s3, i+1, j, k+1) {
                                return true
                        }
```

(续下页)

(接上页)

```
                }
        }
        if j < len(s2) {
                if s2[j] == s3[k] {
                        if dfs(s1, s2, s3, i, j+1, k+1) {
                                return true
                        }
                }
        }
        return false
}
```

3.23 99. 恢复二叉搜索树 (4)

- 题目

```
二叉搜索树中的两个节点被错误地交换。
请在不改变其结构的情况下，恢复这棵树。
示例 1:输入: [1,3,null,null,2]
   1
  /
 3
  \
   2
输出: [3,1,null,null,2]
   3
  /
 1
  \
   2
示例 2:输入: [3,1,4,null,null,2]
  3
 / \
1   4
   /
  2
输出: [2,1,4,null,null,3]
  2
 / \
1   4
   /
  3
```

(续下页)

(接上页)

进阶:使用 O(n) 空间复杂度的解法很容易实现。你能想出一个只使用常数空间的解决方案吗？

- 解题思路

```
var arr []*TreeNode

func recoverTree(root *TreeNode) {
        arr = make([]*TreeNode, 0)
        dfs(root)
        a, b := -1, -1
        for i := 0; i < len(arr)-1; i++ {
                if arr[i].Val > arr[i+1].Val {
                        b = i + 1
                        if a == -1 {
                                a = i
                        }
                }
        }
        arr[a].Val, arr[b].Val = arr[b].Val, arr[a].Val
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
        arr = append(arr, root)
        dfs(root.Right)
}

# 2
var prev, first, second *TreeNode

func recoverTree(root *TreeNode) {
        prev, first, second = nil, nil, nil
        dfs(root)
        first.Val, second.Val = second.Val, first.Val
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
```

(续下页)

(接上页)

```
        if prev != nil && prev.Val > root.Val {
                second = root
                if first == nil {
                        first = prev
                } else {
                        return
                }
        }
        prev = root
        dfs(root.Right)
}

# 3
func recoverTree(root *TreeNode) {
        var prev, first, second *TreeNode
        stack := make([]*TreeNode, 0)
        for len(stack) > 0 || root != nil {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                root = stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if prev != nil && root.Val < prev.Val {
                        second = root
                        if first == nil {
                                first = prev
                        } else {
                                break
                        }
                }
                prev = root
                root = root.Right
        }
        first.Val, second.Val = second.Val, first.Val
}

# 4
func recoverTree(root *TreeNode) {
        var prev, temp, first, second *TreeNode
        for root != nil {
                temp = root.Left
                if temp != nil {
```

(续下页)

(接上页)

```
            // 当前root节点向左走一步，然后一直向右走至无法走为止
                    for temp.Right != nil && temp.Right != root {
                            temp = temp.Right
                    }
                    if temp.Right == nil {
                            temp.Right = root
                            root = root.Left
                            continue
                    } else {
                            temp.Right = nil
                    }
            }
            if prev != nil && prev.Val > root.Val {
                    second = root
                    if first == nil {
                            first = prev
                    }
            }
            prev = root
            root = root.Right
        }
        first.Val, second.Val = second.Val, first.Val
}
```

CHAPTER 4

0101-0200-Easy

4.1 101. 对称二叉树 (2)

- 题目

```
给定一个二叉树，检查它是否是镜像对称的。
例如，二叉树 [1,2,2,3,4,4,3] 是对称的。
    1
   / \
  2   2
 / \ / \
3  4 4  3
但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的:
    1
   / \
  2   2
   \   \
   3    3
说明:如果你可以运用递归和迭代两种方法解决这个问题，会很加分。
```

- 解答思路

```
// 递归
func isSymmetric(root *TreeNode) bool {
        if root == nil {
```

(续下页)

(接上页)

```
                return true
        }
        return recur(root.Left, root.Right)
}

func recur(left, right *TreeNode) bool {
        if left == nil && right == nil {
                return true
        }
        if left == nil || right == nil {
                return false
        }

        return left.Val == right.Val &&
                recur(left.Left, right.Right) &&
                recur(left.Right, right.Left)
}
// 迭代
func isSymmetric(root *TreeNode) bool {
        leftQ := make([]*TreeNode, 0)
        rightQ := make([]*TreeNode, 0)
        leftQ = append(leftQ, root)
        rightQ = append(rightQ, root)

        for len(leftQ) != 0 && len(rightQ) != 0 {
                leftCur, rightCur := leftQ[0], rightQ[0]
                leftQ, rightQ = leftQ[1:], rightQ[1:]

                if leftCur == nil && rightCur == nil {
                        continue
                } else if leftCur != nil && rightCur != nil && leftCur.Val ==␣
↪rightCur.Val {
                        leftQ = append(leftQ, leftCur.Left, leftCur.Right)
                        rightQ = append(rightQ, rightCur.Right, rightCur.Left)
                } else {
                        return false
                }
        }

        if len(leftQ) == 0 && len(rightQ) == 0 {
                return true
        } else {
                return false
```

(续下页)

(接上页)

```
        }
}
```

4.2 104. 二叉树的最大深度 (2)

- 题目

```
给定一个二叉树，找出其最大深度。
二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。
说明: 叶子节点是指没有子节点的节点。
示例: 给定二叉树 [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回它的最大深度 3 。
```

- 解答思路

```
// 递归
func maxDepth(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := maxDepth(root.Left)
        right := maxDepth(root.Right)

        return max(left, right) + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}


// 迭代
func maxDepth(root *TreeNode) int {
        if root == nil {
```

(续下页)

(接上页)

```
                return 0
        }
        queue := make([]*TreeNode,0)
        queue = append(queue,root)
        depth := 0

        for len(queue) > 0{
                length := len(queue)

                for i := 0; i < length; i++{
                        node := queue[0]
                        queue = queue[1:]
                        if node.Left != nil{
                                queue = append(queue,node.Left)
                        }
                        if node.Right != nil{
                                queue = append(queue,node.Right)
                        }
                }
                depth++
        }
        return depth
}
```

4.3 107. 二叉树的层次遍历 II(2)

- 题目

```
给定一个二叉树，返回其节点值自底向上的层次遍历。␣
↪(即按从叶子节点所在层到根节点所在的层，逐层从左向右遍历)
例如：给定二叉树 [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回其自底向上的层次遍历为：
[
  [15,7],
  [9,20],
  [3]
]
```

- 解题思路

```
// 迭代
func levelOrderBottom(root *TreeNode) [][]int {
        if root == nil {
                return nil
        }
        queue := make([]*TreeNode,0)
        out := make([][]int,0)
        queue = append(queue, root)

        for len(queue) != 0 {
                l := len(queue)
                arr := make([]int,0)
                for i := 0; i < l; i++ {
                        pop := queue[i]
                        arr = append(arr, pop.Val)
                        if pop.Left != nil {
                                queue = append(queue, pop.Left)
                        }
                        if pop.Right != nil {
                                queue = append(queue, pop.Right)
                        }
                }
                out = append(out, arr)
                queue = queue[l:]
        }

        out2 := make([][]int, len(out))
        for i := 0; i < len(out); i++ {
                out2[len(out)-1-i] = out[i]
        }

        return out2
}

// 递归
func levelOrderBottom(root *TreeNode) [][]int {
        result := make([][]int, 0)
        level := 0
        if root == nil {
                return result
        }

        orderBottom(root, &result, level)
```

(续下页)

(接上页)

```
        left, right := 0, len(result)-1
        for left < right {
                result[left], result[right] = result[right], result[left]
                left++
                right--
        }
        return result
}

func orderBottom(root *TreeNode, result *[][]int, level int) {
        if root == nil {
                return
        }
        if len(*result) > level {
                (*result)[level] = append((*result)[level], root.Val)
        } else {
                *result = append(*result, []int{root.Val})
        }
        orderBottom(root.Left, result, level+1)
        orderBottom(root.Right, result, level+1)
}
```

4.4 108. 将有序数组转换为二叉搜索树 (2)

- 题目

```
将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。
本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过
↪1。
示例:给定有序数组: [-10,-3,0,5,9],
一个可能的答案是: [0,-3,9,-10,null,5], 它可以表示下面这个高度平衡二叉搜索树:
      0
     / \
   -3   9
   /   /
 -10  5
```

- 解题思路

```
// 递归
func sortedArrayToBST(nums []int) *TreeNode {
```

(续下页)

(接上页)

```
        if len(nums) == 0 {
                return nil
        }

        mid := len(nums) / 2

        return &TreeNode{
                Val:   nums[mid],
                Left:  sortedArrayToBST(nums[:mid]),
                Right: sortedArrayToBST(nums[mid+1:]),
        }
}

// 迭代
type MyTreeNode struct {
        root  *TreeNode
        start int
        end   int
}

func sortedArrayToBST(nums []int) *TreeNode {
        if len(nums) == 0 {
                return nil
        }

        queue := make([]MyTreeNode, 0)
        root := &TreeNode{Val: 0}
        queue = append(queue, MyTreeNode{root, 0, len(nums)})
        for len(queue) > 0 {
                myRoot := queue[0]
                queue = queue[1:]
                start := myRoot.start
                end := myRoot.end
                mid := (start + end) / 2
                curRoot := myRoot.root
                curRoot.Val = nums[mid]
                if start < mid {
                        curRoot.Left = &TreeNode{Val: 0}
                        queue = append(queue, MyTreeNode{curRoot.Left, start, mid})
                }
                if mid+1 < end {
                        curRoot.Right = &TreeNode{Val: 0}
                        queue = append(queue, MyTreeNode{curRoot.Right, mid + 1, end})
```

(续下页)

(接上页)

```
                }
        }
        return root
}
```

4.5 110. 平衡二叉树 (3)

- 题目

```
给定一个二叉树，判断它是否是高度平衡的二叉树。
本题中，一棵高度平衡二叉树定义为：
    一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过1。
示例 1:给定二叉树 [3,9,20,null,null,15,7]
    3
   / \
  9  20
    /  \
   15   7
返回 true 。
示例 2:给定二叉树 [1,2,2,3,3,null,null,4,4]
       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
返回 false 。
```

- 解题思路

```
func isBalanced(root *TreeNode) bool {
        _, isBalanced := recur(root)
        return isBalanced

}

func recur(root *TreeNode) (int, bool) {
        if root == nil {
                return 0, true
        }
```

(续下页)

(接上页)

```
        leftDepth, leftIsBalanced := recur(root.Left)
        if leftIsBalanced == false{
                return 0,false
        }
        rightDepth, rightIsBalanced := recur(root.Right)
        if rightIsBalanced == false{
                return 0,false
        }

        if -1 <= leftDepth-rightDepth &&
                leftDepth-rightDepth <= 1 {
                return max(leftDepth, rightDepth) + 1, true
        }
        return 0, false
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func isBalanced(root *TreeNode) bool {
        return dfs(root) != -1
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        if left != -1 && right != -1 &&
                abs(left, right) <= 1 {
                return max(left, right) + 1
        }
        return -1
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

# 3
func isBalanced(root *TreeNode) bool {
        if root == nil {
                return true
        }
        if math.Abs(dfs(root.Left)-dfs(root.Right)) <= 1 {
                return isBalanced(root.Left) && isBalanced(root.Right)
        }
        return false
}

func dfs(root *TreeNode) float64 {
        if root == nil {
                return 0
        }
        return math.Max(dfs(root.Left), dfs(root.Right)) + 1
}
```

4.6 111. 二叉树的最小深度 (2)

- 题目

```
给定一个二叉树，找出其最小深度。
最小深度是从根节点到最近叶子节点的最短路径上的节点数量。
说明：叶子节点是指没有子节点的节点。
示例:给定二叉树 [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
```

(续下页)

(接上页)

```
   15   7
返回它的最小深度  2.
```

- 解题思路

```
// 递归
func minDepth(root *TreeNode) int {
        if root == nil {
                return 0
        } else if root.Left == nil {
                return 1 + minDepth(root.Right)
        } else if root.Right == nil {
                return 1 + minDepth(root.Left)
        } else {
                return 1 + min(minDepth(root.Left), minDepth(root.Right))
        }
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

// 广度优先搜索
func minDepth(root *TreeNode) int {
        if root == nil{
                return 0
        }

        list := make([]*TreeNode,0)
        list = append(list,root)
        depth := 1

        for len(list) > 0{
                length := len(list)
                for i := 0; i < length; i++{
                        node := list[0]
                        list = list[1:]
                        if node.Left == nil && node.Right == nil{
                                return depth
                        }
                        if node.Left != nil{
```

(续下页)

(接上页)

```
                                list = append(list,node.Left)
                        }
                        if node.Right != nil{
                                list = append(list,node.Right)
                        }
                }
                depth++
        }
        return depth
}
```

4.7 112. 路径总和 (2)

- 题目

```
给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目
说明：叶子节点是指没有子节点的节点。
示例：给定如下二叉树，以及目标和 sum = 22，
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \      \
        7    2      1
返回 true，因为存在目标和为 22 的根节点到叶子节点的路径 5->4->11->2。
```

- 解题思路

```
// 递归
func hasPathSum(root *TreeNode, sum int) bool {
        if root == nil {
                return false
        }
        sum = sum - root.Val
        if root.Left == nil && root.Right == nil {
                return sum == 0
        }
        return hasPathSum(root.Left, sum) || hasPathSum(root.Right, sum)
}

// 迭代
```

(续下页)

(接上页)

```
func hasPathSum(root *TreeNode, sum int) bool {
        if root == nil {
                return false
        }
        list1 := list.New()
        list2 := list.New()

        list1.PushFront(root)
        list2.PushFront(sum - root.Val)
        for list1.Len() > 0 {
                length := list1.Len()

                for i := 0; i < length; i++ {
                        node := list1.Remove(list1.Back()).(*TreeNode)
                        currentSum := list2.Remove(list2.Back()).(int)
                        if node.Left == nil && node.Right == nil && currentSum == 0 {
                                return true
                        }
                        if node.Left != nil {
                                list1.PushFront(node.Left)
                                list2.PushFront(currentSum - node.Left.Val)
                        }
                        if node.Right != nil {
                                list1.PushFront(node.Right)
                                list2.PushFront(currentSum - node.Right.Val)
                        }
                }
        }
        return false
}
```

4.8 118. 杨辉三角 (2)

- 题目

```
给定一个非负整数 numRows，生成杨辉三角的前 numRows 行。
在杨辉三角中，每个数是它左上方和右上方的数的和。
示例:输入: 5 输出:
[
     [1],
    [1,1],
   [1,2,1],
```

(续下页)

(接上页)

```
  [1,3,3,1],
 [1,4,6,4,1]
]
```

- 解题思路

```
// 动态规划
func generate(numRows int) [][]int {
        var result [][]int
        for i := 0; i < numRows; i++ {
                var row []int
                for j := 0; j <= i; j++ {
                        tmp := 1
                        if j == 0 || j == i {

                        } else {
                                tmp = result[i-1][j-1] + result[i-1][j]
                        }
                        row = append(row, tmp)
                }
                result = append(result, row)
        }
        return result
}

// 递推
func generate(numRows int) [][]int {
        res := make([][]int, 0)
        if numRows == 0 {
                return res
        }

        res = append(res, []int{1})
        if numRows == 1 {
                return res
        }

        for i := 1; i < numRows; i++ {
                res = append(res, genNext(res[i-1]))
        }
        return res
}

func genNext(p []int) []int {
```

(续下页)

(接上页)

```
        res := make([]int, 1, len(p)+1)
        res = append(res, p...)

        for i := 0; i < len(res)-1; i++ {
                res[i] = res[i] + res[i+1]
        }
        return res
}
```

4.9 119. 杨辉三角 II(3)

- 题目

```
给定一个非负索引 k，其中 k ≤ 33，返回杨辉三角的第 k 行。
在杨辉三角中，每个数是它左上方和右上方的数的和。
示例:输入: 3 输出: [1,3,3,1]
进阶: 你可以优化你的算法到 O(k) 空间复杂度吗？
```

- 解题思路

```
// 动态规划
func getRow(rowIndex int) []int {
        var result [][]int
        for i := 0; i < rowIndex+1; i++ {
                var row []int
                for j := 0; j <= i; j++ {
                        tmp := 1
                        if j == 0 || j == i {

                        } else {
                                tmp = result[i-1][j-1] + result[i-1][j]
                        }
                        row = append(row, tmp)
                }
                result = append(result, row)
        }
        return result[rowIndex]
}

// 递推
func getRow(rowIndex int) []int {
        res := make([]int,1,rowIndex+1)
```

(续下页)

(接上页)

```
        res[0] = 1
        if rowIndex == 0{
                return res
        }

        for i := 0; i < rowIndex; i++{
                res = append(res,1)
                for j := len(res) -2 ; j > 0; j--{
                        res[j] = res[j] + res[j-1]
                }

        }
        return res
}

// 二项式定理
func getRow(rowIndex int) []int {
        res := make([]int,rowIndex+1)
        res[0] = 1
        if rowIndex == 0{
                return res
        }

        // 公式
        // C(n,k) = n! /(k! * (n-k)!)
        // C(n,k) = (n-k+1)/k * C(n,k-1)
        for i := 1; i <= rowIndex; i++{
                res[i] = res[i-1] * (rowIndex-i+1)/i
        }
        return res
}
```

4.10 121. 买卖股票的最佳时机 (3)

- 题目

```
给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。
如果你最多只允许完成一笔交易（即买入和卖出一支股票），设计一个算法来计算你所能获取的最大利润。
注意你不能在买入股票前卖出股票。
示例 1:输入: [7,1,5,3,6,4] 输出: 5
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 =␣
↪6）的时候卖出，最大利润 = 6-1 = 5 。
```

(续下页)

(接上页)

```
    注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格。
示例 2:输入: [7,6,4,3,1] 输出: 0
解释: 在这种情况下, 没有交易完成, 所以最大利润为 0。
```

- 解题思路

```
// 暴力法
func maxProfit(prices []int) int {
        max := 0
        length := len(prices)

        for i := 0; i < length-1 ; i++{
                for j := i+1; j <= length-1; j++{
                        if prices[j] - prices[i] > max{
                                max = prices[j] - prices[i]
                        }
                }
        }
        return max
}

// 动态规划(从前到后)
func maxProfit(prices []int) int {
        if len(prices) < 2 {
                return 0
        }

        min := prices[0]
        profit := 0

        for i := 1; i < len(prices); i++ {
                if prices[i] < min {
                        min = prices[i]
                }
                if profit < prices[i]-min {
                        profit = prices[i] - min
                }
        }
        return profit
}


// 动态规划(从后到前)
func maxProfit(prices []int) int {
```

(续下页)

(接上页)

```
        if len(prices) < 2 {
                return 0
        }

        max := 0
        profit := 0

        for i := len(prices) - 1; i >= 0; i-- {
                if max < prices[i] {
                        max = prices[i]
                }
                if profit < max-prices[i] {
                        profit = max - prices[i]
                }
        }

        return profit
}
```

4.11 122. 买卖股票的最佳时机 II(2)

- 题目

```
给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。
设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。
注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
示例 1:输入: [7,1,5,3,6,4] 输出: 7
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，
这笔交易所能获得利润 = 5-1 = 4 。
     随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，
     这笔交易所能获得利润 = 6-3 = 3 。
示例 2: 输入: [1,2,3,4,5]输出: 4
解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，
这笔交易所能获得利润 = 5-1 = 4 。
     注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。
     因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。
示例 3:输入: [7,6,4,3,1] 输出: 0
解释: 在这种情况下，没有交易完成，所以最大利润为 0。
```

- 解题思路

```
func maxProfit(prices []int) int {
        max := 0
        for i := 1; i < len(prices); i++ {
                if prices[i] > prices[i-1] {
                        max = max + prices[i] - prices[i-1]
                }
        }
        return max
}

func maxProfit(prices []int) int {
        if len(prices) == 0 {
                return 0
        }
        i := 0
        valley := prices[0]
        peak := prices[0]
        profit := 0
        for i < len(prices)-1 {
                for i < len(prices)-1 && prices[i] >= prices[i+1] {
                        i++
                }
                valley = prices[i]
                for i < len(prices)-1 && prices[i] <= prices[i+1] {
                        i++
                }
                peak = prices[i]
                profit = profit + peak - valley
        }
        return profit
}
```

4.12 125. 验证回文串 (2)

- 题目

```
给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。
说明：本题中，我们将空字符串定义为有效的回文串。
示例 1:输入: "A man, a plan, a canal: Panama" 输出: true
示例 2:输入: "race a car" 输出: false
```

- 解题思路

```
func isPalindrome(s string) bool {
        s = strings.ToLower(s)
        i, j := 0, len(s)-1

        for i < j {
                for i < j && !isChar(s[i]) {
                        i++
                }
                for i < j && !isChar(s[j]) {
                        j--
                }
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

func isChar(c byte) bool {
        if ('a' <= c && c <= 'z') || ('0' <= c && c <= '9') {
                return true
        }
        return false
}

//
func isPalindrome(s string) bool {
        str := ""
        s = strings.ToLower(s)
        for _, value := range s {
                if (value >= '0' && value <= '9') || (value >= 'a' && value <= 'z') {
                        str += string(value)
                }
        }
        if len(str) == 0 {
                return true
        }
        i := 0
        j := len(str) - 1
        for i <= j {
                if str[i] != str[j] {
                        return false
```

(续下页)

(接上页)

```
                }
                i++
                j--
        }
        return true
}
```

4.13 136. 只出现一次的数字 (4)

- 题目

```
给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。
说明：你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？
示例 1:输入: [2,2,1] 输出: 1
示例 2:输入: [4,1,2,1,2] 输出: 4
```

- 解题思路

```
// 异或
func singleNumber(nums []int) int {
        res := 0
        for _, n := range nums {
                res = res ^ n
        }
        return res
}

// 哈希
func singleNumber(nums []int) int {
        m := make(map[int]int)

        for _,v := range nums{
                m[v]++
        }

        for k,v := range m{
                if v == 1{
                        return k
                }
        }
        return -1
}
```

(续下页)

(接上页)

```
// 暴力法
func singleNumber(nums []int) int {
        for i := 0; i < len(nums); i++ {
                flag := false
                for j := 0; j < len(nums); j++ {
                        if nums[i] == nums[j] && i != j {
                                flag = true
                                break
                        }
                }
                if flag == false {
                        return nums[i]
                }

        }
        return -1
}

// 排序遍历
func singleNumber(nums []int) int {
        sort.Ints(nums)
        for i := 0; i < len(nums);i = i+2{
                if i+1 == len(nums){
                        return nums[i]
                }
                if nums[i] != nums[i+1]{
                        return nums[i]
                }
        }
        return -1
}
```

4.14 141. 环形链表 (3)

- 题目

```
给定一个链表，判断链表中是否有环。
为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0
↪开始）。
如果 pos 是 -1，则在该链表中没有环。
示例 1：输入：head = [3,2,0,-4], pos = 1 输出：true
```

(续下页)

(接上页)

```
解释：链表中有一个环，其尾部连接到第二个节点。
示例 2：输入：head = [1,2], pos = 0 输出：true
解释：链表中有一个环，其尾部连接到第一个节点。
示例 3：输入：head = [1], pos = -1输出：false
解释：链表中没有环。
```

- 解题思路

```
func hasCycle(head *ListNode) bool {
        m := make(map[*ListNode]bool)
        for head != nil {
                if m[head] {
                        return true
                }
                m[head] = true
                head = head.Next
        }
        return false
}

// 双指针(快慢指针)
func hasCycle(head *ListNode) bool {
        if head == nil {
                return false
        }
        fast := head.Next
        for fast != nil && head != nil && fast.Next != nil {
                if fast == head {
                        return true
                }
                fast = fast.Next.Next
                head = head.Next
        }
        return false
}

# 3
func hasCycle(head *ListNode) bool {
        for head != nil {
                if head.Val == math.MaxInt32 {
                        return true
                }
                head.Val = math.MaxInt32
                head = head.Next
```

(续下页)

(接上页)

```
        }
        return false
}
```

4.15 155. 最小栈 (2)

- 题目

```
设计一个支持 push, pop, top 操作, 并能在常数时间内检索到最小元素的栈。
    push(x) -- 将元素 x 推入栈中。
    pop() -- 删除栈顶的元素。
    top() -- 获取栈顶元素。
    getMin() -- 检索栈中的最小元素。
示例:
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

- 解题思路

```
type item struct {
        min, x int
}
type MinStack struct {
        stack []item
}

func Constructor() MinStack {
        return MinStack{}
}

func (this *MinStack) Push(x int) {
        min := x
        if len(this.stack) > 0 && this.GetMin() < x {
                min = this.GetMin()
        }
        this.stack = append(this.stack, item{
```

(续下页)

(接上页)

```
                min: min,
                x:   x,
        })
}

func (this *MinStack) Pop() {
        this.stack = this.stack[:len(this.stack)-1]
}

func (this *MinStack) Top() int {
        if len(this.stack) == 0 {
                return 0
        }
        return this.stack[len(this.stack)-1].x
}

func (this *MinStack) GetMin() int {
        if len(this.stack) == 0 {
                return 0
        }
        return this.stack[len(this.stack)-1].min
}

//
type MinStack struct {
        data []int
        min  []int
}

func Constructor() MinStack {
        return MinStack{[]int{}, []int{}}
}

func (this *MinStack) Push(x int) {
        if len(this.data) == 0 || x <= this.GetMin() {
                this.min = append(this.min, x)
        }
        this.data = append(this.data, x)
}

func (this *MinStack) Pop() {
        x := this.data[len(this.data)-1]
        this.data = this.data[:len(this.data)-1]
```

(续下页)

(接上页)

```
        if x == this.GetMin() {
                this.min = this.min[:len(this.min)-1]
        }
}

func (this *MinStack) Top() int {
        if len(this.data) == 0 {
                return 0
        }
        return this.data[len(this.data)-1]
}

func (this *MinStack) GetMin() int {
        return this.min[len(this.min)-1]
}
```

4.16 160. 相交链表 (4)

- 题目

```
编写一个程序，找到两个单链表相交的起始节点。
如下面的两个链表：
在节点 c1 开始相交。
示例 1：
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB␣
↪= 3
输出：Reference of the node with value = 8
输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
示例 2：
输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
输出：Reference of the node with value = 2
输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。
在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。
示例 3：
输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
输出：null
输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。
由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
解释：这两个链表不相交，因此返回 null。
```

(续下页)

(接上页)

```
注意：
    如果两个链表没有交点，返回 null.
    在返回结果后，两个链表仍须保持原有的结构。
    可假定整个链表结构中没有循环。
    程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。
```

- 解题思路

```
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        ALength := 0
        A := headA
        for A != nil {
                ALength++
                A = A.Next
        }
        BLength := 0
        B := headB
        for B != nil {
                BLength++
                B = B.Next
        }

        pA := headA
        pB := headB
        if ALength > BLength {
                n := ALength - BLength
                for n > 0 {
                        pA = pA.Next
                        n--
                }
        } else {
                n := BLength - ALength
                for n > 0 {
                        pB = pB.Next
                        n--
                }
        }

        for pA != pB {
                pA = pA.Next
                pB = pB.Next
        }
        return pA
}
```

(续下页)

(接上页)

```
//
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        A, B := headA, headB
        for A != B {
                if A != nil {
                        A = A.Next
                } else {
                        A = headB
                }
                if B != nil {
                        B = B.Next
                } else {
                        B = headA
                }
        }
        return A
}

// 暴力法
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        A, B := headA, headB
        for A != nil {
                for B != nil {
                        if A == B {
                                return A
                        }
                        B = B.Next
                }
                A = A.Next
                B = headB
        }
        return nil
}

// 哈希表法
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for headA != nil {
                m[headA] = true
                headA = headA.Next
        }
```

(续下页)

(接上页)

```
        for headB != nil {
                if _, ok := m[headB]; ok {
                        return headB
                }
                headB = headB.Next
        }
        return nil
}
```

4.17 167. 两数之和 II - 输入有序数组 (4)

- 题目

```
给定一个已按照升序排列 的有序数组，找到两个数使得它们相加之和等于目标数。
函数应该返回这两个下标值 index1 和 index2，其中 index1 必须小于 index2。
说明:
    返回的下标值 (index1 和 index2) 不是从零开始的。
    你可以假设每个输入只对应唯一的答案，而且你不可以重复使用相同的元素。
示例:输入: numbers = [2, 7, 11, 15], target = 9 输出: [1,2]
解释: 2 与 7 之和等于目标数 9 。因此 index1 = 1, index2 = 2 。
```

- 解题思路

```
// 暴力法：2层循环遍历
func twoSum(nums []int, target int) []int {
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i]+nums[j] == target {
                                return []int{i + 1, j + 1}
                        }
                }
        }
        return []int{}
}

// 两遍哈希遍历
func twoSum(nums []int, target int) []int {
        m := make(map[int]int, len(nums))
        for k, v := range nums {
                m[v] = k
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                b := target - nums[i]
                if num, ok := m[b]; ok && num != i {
                        return []int{i + 1, m[b] + 1}
                }
        }
        return []int{}
}

// 一遍哈希遍历
func twoSum(numbers []int, target int) []int {
        m := make(map[int]int, len(numbers))

        for i, n := range numbers {
                if m[target-n] != 0 {
                        return []int{m[target-n], i + 1}
                }
                m[n] = i + 1
        }
        return nil
}

// 双指针法
func twoSum(numbers []int, target int) []int {
        first := 0
        last := len(numbers) - 1

        result := make([]int, 2)

        for {
                if numbers[first]+numbers[last] == target {
                        result[0] = first + 1
                        result[1] = last + 1
                        return result
                } else if numbers[first]+numbers[last] > target {
                        last--
                } else {
                        first++
                }
        }
}
```

4.18 168.Excel 表列名称 (2)

- 题目

```
给定一个正整数，返回它在 Excel 表中相对应的列名称。
例如，
    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB
    ...
示例 1:输入: 1 输出: "A"
示例 2:输入: 28 输出: "AB"
示例 3:输入: 701 输出: "ZY"
```

- 解题思路

```
// 求余模拟进制
func convertToTitle(n int) string {
        str := ""

        for n > 0 {
                n--
                str = string(byte(n%26)+'A') + str
                n /= 26
        }
        return str
}

// 递归计算
func convertToTitle(n int) string {
        if n <= 26{
                return string('A'+n-1)
        }
        y := n % 26
        if y == 0{
                // 26的倍数 如52%26=0 => AZ
                return convertToTitle((n-y-1)/26)+convertToTitle(26)
        }
        return convertToTitle((n-y)/26)+convertToTitle(y)
}
```

4.19 169. 多数元素 (5)

- 题目

```
给定一个大小为 n 的数组，找到其中的多数元素。多数元素是指在数组中出现次数大于 ⌊ n/2 ⌋ 的元素。
你可以假设数组是非空的，并且给定的数组总是存在多数元素。
示例 1:输入: [3,2,3]输出: 3
示例 2:输入: [2,2,1,1,1,2,2]输出: 2
```

- 解题思路

```
// 排序取半
func majorityElement(nums []int) int {
        sort.Ints(nums)
        return nums[len(nums)/2]
}

// 哈希法
func majorityElement(nums []int) int {
        m := make(map[int]int)
        result := 0
        for _, v := range nums{
                if _,ok := m[v];ok{
                        m[v]++
                }else {
                        m[v]=1
                }
                if m[v] > (len(nums)/2){
                        result = v
                }
        }
        return result
}

// Boyer-Moore投票算法
func majorityElement(nums []int) int {
        result, count := 0, 0
        for i := 0; i < len(nums); i++ {
                if count == 0 {
                        result = nums[i]
                        count++
                } else if result == nums[i] {
                        count++
```

(续下页)

(接上页)

```
                } else {
                        count--
                }
        }
        return result
}

// 位运算
func majorityElement(nums []int) int {
        if len(nums) == 1 {
                return nums[0]
        }
        result := int32(0)
        // 64位有坑
        mask := int32(1)
        for i := 0; i < 32; i++ {
                count := 0
                for j := 0; j < len(nums); j++ {
                        if mask&int32(nums[j]) == mask {
                                count++
                        }
                }
                if count > len(nums)/2 {
                        result = result | mask
                }
                mask = mask << 1
        }
        return int(result)
}

// 分治法
func majorityElement(nums []int) int {
        return majority(nums, 0, len(nums)-1)
}

func count(nums []int, target int, start int, end int) int {
        countNum := 0
        for i := start; i <= end; i++ {
                if nums[i] == target {
                        countNum++
                }
        }
        return countNum
```

(续下页)

(接上页)

```
}

func majority(nums []int, start, end int) int {
        if start == end {
                return nums[start]
        }

        mid := (start + end) / 2

        left := majority(nums, start, mid)
        right := majority(nums, mid+1, end)
        if left == right {
                return left
        }

        leftCount := count(nums, left, start, end)
        rightCount := count(nums, right, start, end)
        if leftCount > rightCount {
                return left
        }
        return right
}
```

4.20 171.Excel 表列序号 (1)

- 题目

```
给定一个Excel表格中的列名称，返回其相应的列序号。
例如，
    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28
    ...
示例 1:输入: "A" 输出: 1
示例 2:输入: "AB" 输出: 28
示例 3:输入: "ZY" 输出: 701
```

- 解题思路

```
func titleToNumber(s string) int {
        result := 0
        for i := 0; i < len(s); i++ {
                temp := int(s[i] - 'A' + 1)
                result = result*26 + temp
        }
        return result
}
```

4.21 172. 阶乘后的零 (1)

- 题目

```
给定一个整数 n，返回 n! 结果尾数中零的数量。
示例 1:输入: 3 输出: 0
解释: 3! = 6, 尾数中没有零。
示例 2:输入: 5输出: 1
解释: 5! = 120, 尾数中有 1 个零.
说明: 你算法的时间复杂度应为 O(log n) 。
```

- 解题思路

```
func trailingZeroes(n int) int {
        result := 0
        for n >= 5 {
                n = n / 5
                result = result + n
        }
        return result
}
```

4.22 189. 旋转数组 (4)

- 题目

```
给定一个数组，将数组中的元素向右移动 k 个位置，其中 k 是非负数。
示例 1:
输入: [1,2,3,4,5,6,7] 和 k = 3
输出: [5,6,7,1,2,3,4]
解释:
向右旋转 1 步: [7,1,2,3,4,5,6]
```

(续下页)

(接上页)

```
向右旋转 2 步: [6,7,1,2,3,4,5]
向右旋转 3 步: [5,6,7,1,2,3,4]
示例 2:
输入: [-1,-100,3,99] 和 k = 2
输出: [3,99,-1,-100]
解释:
向右旋转 1 步: [99,-1,-100,3]
向右旋转 2 步: [3,99,-1,-100]
说明:
    尽可能想出更多的解决方案，至少有三种不同的方法可以解决这个问题。
    要求使用空间复杂度为 O(1) 的 原地 算法。
```

- 解题思路

```
// 暴力法
func rotate(nums []int, k int) {
        n := len(nums)

        if k > n {
                k = k % n
        }
        if k == 0 || k == n {
                return
        }
        for i := 0; i < k; i++ {
                last := nums[len(nums)-1]
                for j := 0; j < len(nums); j++ {
                        nums[j], last = last, nums[j]
                }
        }
}

// 三次反转法
func rotate(nums []int, k int) {
        n := len(nums)

        if k > n {
                k = k % n
        }
        if k == 0 || k == n {
                return
        }
        reverse(nums, 0, n-1)
        reverse(nums, 0, k-1)
```

(续下页)

(接上页)

```
        reverse(nums, k, n-1)
}

func reverse(nums []int, i, j int) {
        for i < j {
                nums[i], nums[j] = nums[j], nums[i]
                i++
                j--
        }
}

// 使用额外的数组
func rotate(nums []int, k int) {
        n := len(nums)

        if k > n {
                k = k % n
        }
        if k == 0 || k == n {
                return
        }

        arr := make([]int, len(nums))
        for i := 0; i < len(nums); i++ {
                arr[(i+k)%len(nums)] = nums[i]
        }

        for i := 0; i < len(nums); i++ {
                nums[i] = arr[i]
        }
}

// 环形替换
func rotate(nums []int, k int) {
        n := len(nums)

        if k > n {
                k = k % n
        }
        if k == 0 || k == n {
                return
        }
        count := 0
```

(续下页)

(接上页)

```
        for i := 0; count < len(nums); i++ {
                current := i
                prev := nums[i]
                for {
                        next := (current + k) % len(nums)
                        nums[next], prev = prev, nums[next]
                        current = next
                        // fmt.Println(nums, prev)
                        count++
                        if i == current {
                                break
                        }
                }
        }
}
```

4.23 190. 颠倒二进制位 (3)

- 题目

```
颠倒给定的 32 位无符号整数的二进制位。
示例 1: 输入: 00000010100101000001111010011100 输出: 00111001011110000010100101000000
解释: 输入的二进制串 00000010100101000001111010011100 表示无符号整数 43261596,
      因此返回 964176192, 其二进制表示形式为 00111001011110000010100101000000。
示例 2: 输入: 11111111111111111111111111111101 输出: 10111111111111111111111111111111
解释: 输入的二进制串 11111111111111111111111111111101 表示无符号整数 4294967293,
      因此返回 3221225471 其二进制表示形式为 10101111110010110010011101101001。
提示:
    请注意, 在某些语言 (如 Java) 中, 没有无符号整数类型。
    在这种情况下, 输入和输出都将被指定为有符号整数类型, 并且不应影响您的实现,
    因为无论整数是有符号的还是无符号的, 其内部的二进制表示形式都是相同的。
    在 Java 中, 编译器使用二进制补码记法来表示有符号整数。
    因此, 在上面的 示例 2 中, 输入表示有符号整数 -3, 输出表示有符号整数 -1073741825。
进阶:
如果多次调用这个函数, 你将如何优化你的算法?
```

- 解题思路

```
func reverseBits(num uint32) uint32 {
        result := uint32(0)
        for i := 0; i < 32; i++ {
```

(续下页)

(接上页)

```
                last := num & 1                 // 取最后一位
                result = (result << 1) + last // 前移
                num = num >> 1
        }
        return result
}

//
func reverseBits(num uint32) uint32 {
        str := strconv.FormatUint(uint64(num), 2)
        rev := ""
        for i := len(str) - 1; i >= 0; i-- {
                rev = rev + str[i:i+1]
        }
        if len(rev) < 32 {
                rev = rev + strings.Repeat("0", 32-len(rev))
        }
        n, _ := strconv.ParseUint(rev, 2, 64)
        return uint32(n)
}

// 二进制交换
import (
        "github.com/imroc/biu"
)

func reverseBits(num uint32) uint32 {
        fmt.Println(biu.Uint32ToBinaryString(num))
        num = ((num & 0xffff0000) >> 16) | ((num & 0x0000ffff) << 16)
        num = ((num & 0xff00ff00) >> 8) | ((num & 0x00ff00ff) << 8)
        num = ((num & 0xf0f0f0f0) >> 4) | ((num & 0x0f0f0f0f) << 4)
        num = ((num & 0xcccccccc) >> 2) | ((num & 0x33333333) << 2)
        num = ((num & 0xaaaaaaaa) >> 1) | ((num & 0x55555555) << 1)
        return num
}
```

4.24 191. 位 1 的个数 (4)

- 题目

```
编写一个函数，输入是一个无符号整数，返回其二进制表达式中数字位数为 ‘1’ 的个数
（也被称为汉明重量）。
示例 1：输入：00000000000000000000000000001011 输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
示例 2：输入：00000000000000000000000010000000 输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
示例 3：输入：11111111111111111111111111111101 输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
提示：
    请注意，在某些语言（如 Java）中，没有无符号整数类型。
    在这种情况下，输入和输出都将被指定为有符号整数类型，并且不应影响您的实现，
    因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。
    在 Java 中，编译器使用二进制补码记法来表示有符号整数。
    因此，在上面的 示例 3 中，输入表示有符号整数 -3。
进阶:如果多次调用这个函数，你将如何优化你的算法？
```

- 解题思路

```
// 循环位计算
func hammingWeight(num uint32) int {
        count := 0
        for num != 0 {
                if num&1 == 1 {
                        count++
                }
                num = num >> 1
        }
        return count
}

//
func hammingWeight(num uint32) int {
        count := 0
        for num != 0 {
                num = num & (num - 1)
                count++
        }
        return count
}
```

(续下页)

(接上页)

```
#
func hammingWeight(num uint32) int {
        return strings.Count(strconv.FormatInt(int64(num),2),"1")
        // return strings.Count(fmt.Sprintf("%b",num),"1")
}

#
func hammingWeight(num uint32) int {
        count := 0
        flag := uint32(1)
        for flag != 0 {
                if num&flag == flag {
                        count++
                }
                flag = flag << 1
        }
        return count
}
```

4.25 198. 打家劫舍 (4)

- 题目

```
你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，
影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，
如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。
给定一个代表每个房屋存放金额的非负整数数组，计算你在不触动警报装置的情况下，能够偷窃到的最高金额。
示例 1:输入: [1,2,3,1] 输出: 4
解释: 偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。 偷窃到的最高金额 = 1␣
↪+ 3 = 4 。
示例 2: 输入: [2,7,9,3,1] 输出: 12
解释: 偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9), 接着偷窃 5 号房屋 (金额 =␣
↪1)。
     偷窃到的最高金额 = 2 + 9 + 1 = 12 。
```

- 解题思路

```
func rob(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
```

(续下页)

(接上页)

```
                return nums[0]
        }
        a := nums[0]
        b := max(a, nums[1])

        for i := 2; i < len(nums); i++ {
                a, b = b, max(a+nums[i], b)
        }
        return b
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        }
        if n == 1 {
                return nums[0]
        }
        dp := make([]int, n)
        dp[0] = nums[0]
        if nums[0] > nums[1] {
                dp[1] = nums[0]
        } else {
                dp[1] = nums[1]
        }
        for i := 2; i < n; i++ {
                dp[i] = max(dp[i-1], dp[i-2]+nums[i])
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

#
func rob(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
                return nums[0]
        }
        n := len(nums)
        dp := make([][]int, n)
        for n := range dp {
                dp[n] = make([]int, 2)
        }
        dp[0][0], dp[0][1] = 0, nums[0]
        for i := 1; i < n; i++ {
                dp[i][0] = max(dp[i-1][0], dp[i-1][1])
                dp[i][1] = dp[i-1][0] + nums[i]
        }
        return max(dp[n-1][0], dp[n-1][1])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func rob(nums []int) int {
        var a, b int
        for i, v := range nums {
                if i%2 == 0 {
                        a = max(a+v, b)
                } else {
                        b = max(a, b+v)
                }
        }
        return max(a, b)
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

CHAPTER 5

0101-0200-Medium

5.1 102. 二叉树的层序遍历 (2)

- 题目

```
给你一个二叉树，请你返回其按 层序遍历 得到的节点值。␣
↪（即逐层地，从左到右访问所有节点）。
示例：二叉树：[3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回其层次遍历结果：
[
  [3],
  [9,20],
  [15,7]
]
```

- 解题思路

```
func levelOrder(root *TreeNode) [][]int {
        res := make([][]int, 0)
        if root == nil {
```

(续下页)

(接上页)

```
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                temp := make([]int, 0)
                for i := 0; i < length; i++ {
                        node := list[i]
                        temp = append(temp, node.Val)
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
                                list = append(list, node.Right)
                        }
                }
                res = append(res, temp)
                list = list[length:]
        }
        return res
}

#
var res [][]int

func levelOrder(root *TreeNode) [][]int {
        res = make([][]int, 0)
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        if level == len(res) {
                res = append(res, []int{})
        }
        res[level] = append(res[level], root.Val)
```

(续下页)

(接上页)

```
        dfs(root.Left, level+1)
        dfs(root.Right, level+1)
}
```

5.2 103. 二叉树的锯齿形层次遍历 (2)

- 题目

```
给定一个二叉树，返回其节点值的锯齿形层次遍历。
（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。
例如：给定二叉树 [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回锯齿形层次遍历如下：
[
  [3],
  [20,9],
  [15,7]
]
```

- 解题思路

```
func zigzagLevelOrder(root *TreeNode) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                temp := make([]int, 0)
                for i := 0; i < length; i++ {
                        node := list[i]
                        temp = append(temp, node.Val)
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
```

(续下页)

(接上页)

```
                                list = append(list, node.Right)
                        }
                }
                if len(res)%2 == 1 {
                        for i := 0; i < len(temp)/2; i++ {
                                temp[i], temp[len(temp)-1-i] = temp[len(temp)-1-i],␣
↪temp[i]
                        }
                }
                res = append(res, temp)
                list = list[length:]
        }
        return res
}

#
var res [][]int

func zigzagLevelOrder(root *TreeNode) [][]int {
        res = make([][]int, 0)
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        if level == len(res) {
                res = append(res, []int{})
        }
        if level%2 == 1 {
                arr := res[level]
                arr = append([]int{root.Val}, arr...)
                res[level] = arr
        } else {
                res[level] = append(res[level], root.Val)
        }
        dfs(root.Left, level+1)
        dfs(root.Right, level+1)
}
```

5.3 105. 从前序与中序遍历序列构造二叉树 (3)

- 题目

```
根据一棵树的前序遍历与中序遍历构造二叉树。
注意:你可以假设树中没有重复的元素。
例如，给出
前序遍历 preorder = [3,9,20,15,7]
中序遍历 inorder = [9,3,15,20,7]
返回如下的二叉树:
    3
   / \
  9  20
    /  \
   15   7
```

- 解题思路

```
func buildTree(preorder []int, inorder []int) *TreeNode {
    for k := range inorder {
        if inorder[k] == preorder[0] {
            return &TreeNode{
                Val:   preorder[0],
                Left:  buildTree(preorder[1:k+1], inorder[0:k]),
                Right: buildTree(preorder[k+1:], inorder[k+1:]),
            }
        }
    }
    return nil
}

# 2
func buildTree(preorder []int, inorder []int) *TreeNode {
    if preorder == nil || len(preorder) == 0 {
        return nil
    }
    root := &TreeNode{
        Val: preorder[0],
    }
    length := len(preorder)
    stack := make([]*TreeNode, 0)
    stack = append(stack, root)
    index := 0
    for i := 1; i < length; i++ {
```

(续下页)

(接上页)

```
        value := preorder[i]
        node := stack[len(stack)-1]
        if node.Val != inorder[index] {
            node.Left = &TreeNode{Val: value}
            stack = append(stack, node.Left)
        } else {
            for len(stack) > 0 && stack[len(stack)-1].Val == inorder[index] {
                node = stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                index++
            }
            node.Right = &TreeNode{Val: value}
            stack = append(stack, node.Right)
        }
    }
    return root
}

#
func buildTree(preorder []int, inorder []int) *TreeNode {
    if len(preorder) == 0 {
        return nil
    }
    return helper(preorder, inorder)
}

func helper(preorder []int, inorder []int) *TreeNode {
    var root *TreeNode
    for k := range inorder {
        if inorder[k] == preorder[0] {
            root = &TreeNode{Val: preorder[0]}
            root.Left = helper(preorder[1:k+1], inorder[0:k])
            root.Right = helper(preorder[k+1:], inorder[k+1:])
        }
    }
    return root
}
```

5.4 106. 从中序与后序遍历序列构造二叉树 (3)

- 题目

```
根据一棵树的中序遍历与后序遍历构造二叉树。
注意:你可以假设树中没有重复的元素。
例如，给出
中序遍历 inorder = [9,3,15,20,7]
后序遍历 postorder = [9,15,7,20,3]
返回如下的二叉树:
    3
   / \
  9  20
    /  \
   15   7
```

- 解题思路

```
func buildTree(inorder []int, postorder []int) *TreeNode {
        last := len(postorder) - 1
        for k := range inorder {
                if inorder[k] == postorder[last] {
                        return &TreeNode{
                                Val:   postorder[last],
                                Left:  buildTree(inorder[0:k], postorder[0:k]),
                                Right: buildTree(inorder[k+1:], postorder[k:last]),
                        }
                }
        }
        return nil
}

#
func buildTree(inorder []int, postorder []int) *TreeNode {
        if len(postorder) == 0 {
                return nil
        }
        return helper(inorder, postorder)
}

func helper(inorder []int, postorder []int) *TreeNode {
        var root *TreeNode
        last := len(postorder) - 1
        for k := range inorder {
```

(续下页)

(接上页)

```
                if inorder[k] == postorder[last] {
                        root = &TreeNode{Val: postorder[last]}
                        root.Left = helper(inorder[0:k], postorder[0:k])
                        root.Right = helper(inorder[k+1:], postorder[k:last])
                }
        }
        return root
}

#
func buildTree(inorder []int, postorder []int) *TreeNode {
        if postorder == nil || len(postorder) == 0 {
                return nil
        }
        last := len(postorder) - 1
        root := &TreeNode{
                Val: postorder[last],
        }
        length := len(postorder)
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        index := last
        for i := length - 2; i >= 0; i-- {
                value := postorder[i]
                node := stack[len(stack)-1]
                if node.Val != inorder[index] {
                        node.Right = &TreeNode{Val: value}
                        stack = append(stack, node.Right)
                } else {
                        for len(stack) > 0 && stack[len(stack)-1].Val ==␣
↪inorder[index] {
                                node = stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                index--
                        }
                        node.Left = &TreeNode{Val: value}
                        stack = append(stack, node.Left)
                }
        }
        return root
}
```

5.5 109. 有序链表转换二叉搜索树 (2)

- 题目

```
给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。
本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过
↪1。
示例:给定的有序链表： [-10, -3, 0, 5, 9],
一个可能的答案是：[0, -3, 9, -10, null, 5], 它可以表示下面这个高度平衡二叉搜索树：
      0
     / \
   -3   9
   /   /
 -10  5
```

- 解题思路

```
func sortedListToBST(head *ListNode) *TreeNode {
        if head == nil {
                return nil
        }
        mid := find(head)
        if mid == head {
                return &TreeNode{Val: mid.Val}
        }
        return &TreeNode{
                Val:   mid.Val,
                Left:  sortedListToBST(head),
                Right: sortedListToBST(mid.Next),
        }
}

func find(head *ListNode) *ListNode {
        if head == nil {
                return nil
        }
        slow, fast := head, head
        var prev *ListNode
        for fast != nil && fast.Next != nil {
                prev = slow
                slow = slow.Next
                fast = fast.Next.Next
        }
        if prev != nil {
```

(续下页)

(接上页)

```
                prev.Next = nil
        }
        return slow
}

#
func sortedListToBST(head *ListNode) *TreeNode {
        if head == nil {
                return nil
        }
        arr := make([]int, 0)
        for head != nil {
                arr = append(arr, head.Val)
                head = head.Next
        }
        return sortArr(arr)
}

func sortArr(arr []int) *TreeNode {
        if len(arr) == 0 {
                return nil
        }
        return &TreeNode{
                Val:   arr[len(arr)/2],
                Left:  sortArr(arr[:len(arr)/2]),
                Right: sortArr(arr[len(arr)/2+1:]),
        }
}
```

5.6 113. 路径总和 II(2)

- 题目

```
给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。
说明: 叶子节点是指没有子节点的节点。
示例:给定如下二叉树，以及目标和 sum = 22，

              5
             / \
            4   8
           /   / \
          11  13  4
```

(续下页)

(接上页)

```
         /  \    / \
        7    2  5   1
返回:
[
   [5,4,11,2],
   [5,8,4,5]
]
```

- 解题思路

```
var res [][]int

func pathSum(root *TreeNode, sum int) [][]int {
        if root == nil {
                return nil
        }
        res = make([][]int, 0)
        var arr []int
        dfs(root, sum, arr)
        return res
}

func dfs(root *TreeNode, sum int, arr []int) {
        if root == nil {
                return
        }
        arr = append(arr, root.Val)
        if root.Val == sum && root.Left == nil && root.Right == nil {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
        }
        dfs(root.Left, sum-root.Val, arr)
        dfs(root.Right, sum-root.Val, arr)
        arr = arr[:len(arr)-1]
}

#
func pathSum(root *TreeNode, sum int) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        temp := make([]int, 0)
```

(续下页)

(接上页)

```
        stack := make([]*TreeNode, 0)
        visited := make(map[*TreeNode]bool)
        curSum := 0
        for root != nil || len(stack) > 0 {
                for root != nil {
                        temp = append(temp, root.Val)
                        curSum = curSum + root.Val
                        visited[root] = true
                        stack = append(stack, root)
                        root = root.Left
                }
                node := stack[len(stack)-1]
                if node.Right == nil || visited[node.Right] {
                        if node.Left == nil && node.Right == nil && curSum == sum {
                                tmp := make([]int, len(temp))
                                copy(tmp, temp)
                                res = append(res, tmp)
                        }
                        stack = stack[:len(stack)-1]
                        temp = temp[:len(temp)-1]
                        curSum = curSum - node.Val
                        root = nil
                } else {
                        root = node.Right
                }
        }
        return res
}
```

5.7 114. 二叉树展开为链表 (3)

- 题目

```
给定一个二叉树，原地将它展开为一个单链表。
例如，给定二叉树
    1
   / \
  2   5
 / \   \
3   4   6
将其展开为：
1
```

(续下页)

(接上页)

```
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

- 解题思路

```
// 将原左子树变为节点的右子树
// 再将原右子树变为当前右子树最右节点的右子树。
func flatten(root *TreeNode) {
        if root == nil {
                return
        }
        flatten(root.Left)
        flatten(root.Right)
        right := root.Right
        root.Right, root.Left = root.Left, nil
        for root.Right != nil {
                root = root.Right
        }
        root.Right = right
}

#
func flatten(root *TreeNode) {
        dfs(root, nil)
}

func dfs(root *TreeNode, pre *TreeNode) *TreeNode {
        if root == nil {
                return pre
        }
        pre = dfs(root.Right, pre)
        pre = dfs(root.Left, pre)
        root.Right, root.Left = pre, nil
        pre = root
        return pre
}
```

(续下页)

(接上页)

```
#
func flatten(root *TreeNode) {
        if root == nil {
                return
        }
        res := make([]*TreeNode, 0)
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                res = append(res, node)
                stack = stack[:len(stack)-1]
                if node.Right != nil {
                        stack = append(stack, node.Right)
                }
                if node.Left != nil {
                        stack = append(stack, node.Left)
                }
        }
        for i := 1; i < len(res); i++ {
                res[i-1].Left = nil
                res[i-1].Right = res[i]
        }
        res[len(res)-1].Left = nil
}
```

5.8 116. 填充每个节点的下一个右侧节点指针 (3)

- 题目

```
给定一个完美二叉树，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
填充它的每个 next 指针，让这个指针指向其下一个右侧节点。
如果找不到下一个右侧节点，则将 next 指针设置为 NULL。
初始状态下，所有 next 指针都被设置为 NULL。
示例：输入：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null,
```

(续下页)

(接上页)

```
↪"right":null,"val":4},"next":null,"right":
{"$id":"4","left":null,"next":null,"right":null,"val":5},"val":2},"next":null,"right":
{"$id":"5","left":
{"$id":"6","left":null,"next":null,"right":null,"val":6},"next":null,"right":
{"$id":"7","left":null,"next":null,"right":null,"val":7},"val":3},"val":1}
输出：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":
{"$id":"4","left":null,"next":{"$id":"5","left":null,"next":
{"$id":"6","left":null,"next":null,"right":null,"val":7},"right":null,"val":6},
"right":null,"val":5},"right":null,"val":4},"next":{"$id":"7","left":
{"$ref":"5"},"next":null,"right":{"$ref":"6"},"val":3},"right":
{"$ref":"4"},"val":2},"next":null,"right":{"$ref":"7"},"val":1}
解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next␣
↪指针，以指向其下一个右侧节点，如图 B 所示。
提示：你只能使用常量级额外空间。
    使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。
```

- 解题思路

```
func connect(root *Node) *Node {
        if root == nil {
                return nil
        }
        left := root.Left
        right := root.Right
        // 从上往下，连接最中间的
        for left != nil {
                left.Next = right
                left = left.Right
                right = right.Left
        }
        connect(root.Left)
        connect(root.Right)
        return root
}

# 2
func connect(root *Node) *Node {
        if root == nil {
                return nil
        }
        cur := root
        for cur.Left != nil {
                parent := cur
                for parent != nil {
```

(续下页)

(接上页)

```
                        parent.Left.Next = parent.Right // 左节点连接右节点
                        if parent.Next != nil {
                                // 图中的5->6 , 左子树的右节点->右子树的左节点
                                parent.Right.Next = parent.Next.Left
                        }
                        parent = parent.Next
                }
                cur = cur.Left // 移到下一层最左边
        }
        return root
}

# 3
func connect(root *Node) *Node {
        if root == nil {
                return nil
        }
        queue := make([]*Node, 0)
        if root.Left != nil {
                queue = append(queue, root.Left)
        }
        if root.Right != nil {
                queue = append(queue, root.Right)
        }
        for len(queue) > 0 {
                length := len(queue)
                i := 0
                for i = 0; i < length; i++ {
                        node := queue[i]
                        if node.Left != nil {
                                queue = append(queue, node.Left)
                        }
                        if node.Right != nil {
                                queue = append(queue, node.Right)
                        }
                        if i+1 < length {
                                node.Next = queue[i+1]
                        }
                }
                queue = queue[length:]
        }
        return root
}
```

5.9 117. 填充每个节点的下一个右侧节点指针 II(4)

- 题目

```
给定一个二叉树
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
填充它的每个 next 指针，让这个指针指向其下一个右侧节点。
如果找不到下一个右侧节点，则将 next 指针设置为 NULL。
初始状态下，所有 next 指针都被设置为 NULL。
进阶：你只能使用常量级额外空间。
    使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。
示例：输入：root = [1,2,3,4,5,null,7] 输出：[1,#,2,3,#,4,5,7,#]
解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next␣
↪指针，以指向其下一个右侧节点，如图 B 所示。
提示：
    树中的节点数小于 6000
    -100 <= node.val <= 100
```

- 解题思路

```
func connect(root *Node) *Node {
        if root == nil || (root.Left == nil && root.Right == nil) {
                return root
        }
        if root.Left != nil {
                root.Left.Next = root.Right
        }
        prev := root.Right
        if prev == nil {
                prev = root.Left
        }
        nextRoot := root.Next
        for nextRoot != nil && (nextRoot.Left == nil && nextRoot.Right == nil) {
                nextRoot = nextRoot.Next
        }
        if nextRoot != nil {
                if nextRoot.Left != nil {
                        prev.Next = nextRoot.Left
                } else {
```

(续下页)

(接上页)

```
                        prev.Next = nextRoot.Right
                }
        }
        connect(root.Right)
        connect(root.Left)
        return root
}

# 2
func connect(root *Node) *Node {
        if root == nil {
                return nil
        }
        queue := make([]*Node, 0)
        if root.Left != nil {
                queue = append(queue, root.Left)
        }
        if root.Right != nil {
                queue = append(queue, root.Right)
        }
        for len(queue) > 0 {
                length := len(queue)
                i := 0
                for i = 0; i < length; i++ {
                        node := queue[i]
                        if node.Left != nil {
                                queue = append(queue, node.Left)
                        }
                        if node.Right != nil {
                                queue = append(queue, node.Right)
                        }
                        if i+1 < length {
                                node.Next = queue[i+1]
                        }
                }
                queue = queue[length:]
        }
        return root
}

# 3
func connect(root *Node) *Node {
        if root == nil {
```

(续下页)

(接上页)

```
                return nil
        }
        cur := root
        for cur != nil {
                var prev, down *Node
                for cur != nil {
                        if cur.Left != nil {
                                if prev != nil {
                                        prev.Next = cur.Left
                                } else {
                                        down = cur.Left
                                }
                                prev = cur.Left
                        }
                        if cur.Right != nil {
                                if prev != nil {
                                        prev.Next = cur.Right
                                } else {
                                        down = cur.Right
                                }
                                prev = cur.Right
                        }
                        cur = cur.Next // 当前层级移动
                }
                cur = down // 移到下一层最左边
        }
        return root
}

# 4
func connect(root *Node) *Node {
        if root == nil {
                return nil
        }
        cur := root
        for cur != nil {
                down := &Node{}
                prev := down
                for cur != nil {
                        if cur.Left != nil {
                                prev.Next = cur.Left
                                prev = prev.Next
                        }
```

(续下页)

(接上页)

```
                        if cur.Right != nil {
                                prev.Next = cur.Right
                                prev = prev.Next
                        }
                        cur = cur.Next // 当前层级移动
                }
                cur = down.Next // 移到下一层最左边
        }
        return root
}
```

5.10 120. 三角形最小路径和 (5)

- 题目

```
给定一个三角形，找出自顶向下的最小路径和。每一步只能移动到下一行中相邻的结点上。
相邻的结点 在这里指的是 下标 与 上一层结点下标 相同或者等于 上一层结点下标 + 1␣
↪的两个结点。
例如，给定三角形：
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
自顶向下的最小路径和为 11（即，2 + 3 + 5 + 1 = 11）。
说明：如果你可以只使用 O(n) 的额外空间（n␣
↪为三角形的总行数）来解决这个问题，那么你的算法会很加分。
```

- 解题思路

```
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        dp[0][0] = triangle[0][0]
        for i := 1; i < n; i++ {
                dp[i][0] = dp[i-1][0] + triangle[i][0]
                for j := 1; j < i; j++ {
                        dp[i][j] = min(dp[i-1][j-1], dp[i-1][j]) + triangle[i][j]
```

(续下页)

(接上页)

```
                }
                dp[i][i] = dp[i-1][i-1] + triangle[i][i]
        }
        res := dp[n-1][0]
        for i := 1; i < n; i++ {
                res = min(res, dp[n-1][i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        dp := [2][]int{}
        for i := 0; i < 2; i++ {
                dp[i] = make([]int, n)
        }
        dp[0][0] = triangle[0][0]
        for i := 1; i < n; i++ {
                cur := i % 2
                prev := 1 - cur
                dp[cur][0] = dp[prev][0] + triangle[i][0]
                for j := 1; j < i; j++ {
                        dp[cur][j] = min(dp[prev][j-1], dp[prev][j]) + triangle[i][j]
                }
                dp[cur][i] = dp[prev][i-1] + triangle[i][i]
        }
        res := dp[(n-1)%2][0]
        for i := 1; i < n; i++ {
                res = min(res, dp[(n-1)%2][i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return b
        }
        return a
}

# 3
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        dp := make([]int, n)
        dp[0] = triangle[0][0]
        for i := 1; i < n; i++ {
                dp[i] = dp[i-1] + triangle[i][i]
                for j := i - 1; j > 0; j-- {
                        dp[j] = min(dp[j-1], dp[j]) + triangle[i][j]
                }
                dp[0] = dp[0] + triangle[i][0]
        }
        res := dp[0]
        for i := 1; i < n; i++ {
                res = min(res, dp[i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        for i := n - 2; i >= 0; i-- {
                for j := 0; j < len(triangle[i]); j++ {
                        triangle[i][j] = min(triangle[i+1][j], triangle[i+1][j+1]) +␣
↪triangle[i][j]
                }
        }
        return triangle[0][0]
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 5
var dp [][]int

func minimumTotal(triangle [][]int) int {
        dp = make([][]int, len(triangle))
        for i := 0; i < len(triangle); i++ {
                dp[i] = make([]int, len(triangle))
        }
        return dfs(triangle, 0, 0)
}

func dfs(triangle [][]int, i, j int) int {
        if i == len(triangle) {
                return 0
        }
        if dp[i][j] != 0 {
                return dp[i][j]
        }
        dp[i][j] = min(dfs(triangle, i+1, j), dfs(triangle, i+1, j+1)) +␣
↪triangle[i][j]
        return dp[i][j]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

5.11 127. 单词接龙 (2)

- 题目

```
给定两个单词（beginWord 和 endWord）和一个字典，找到从 beginWord 到 endWord␣
↪的最短转换序列的长度。
转换需遵循如下规则：
    每次转换只能改变一个字母。
    转换过程中的中间单词必须是字典中的单词。
说明：
    如果不存在这样的转换序列，返回 0。
    所有单词具有相同的长度。
    所有单词只由小写字母组成。
    字典中不存在重复的单词。
    你可以假设 beginWord 和 endWord 是非空的，且二者不相同。
示例 1:输入：
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
输出：5
解释：一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog",
     返回它的长度 5。
示例 2:输入：
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
输出：0
解释：endWord "cog" 不在字典中，所以无法进行转换。
```

- 解题思路

```
func ladderLength(beginWord string, endWord string, wordList []string) int {
        m := make(map[string]int)
        for i := 0; i < len(wordList); i++ {
                m[wordList[i]] = 1
        }
        if m[endWord] == 0 {
                return 0
        }
        preMap := make(map[string][]string)
        for i := 0; i < len(wordList); i++ {
                for j := 0; j < len(wordList[i]); j++ {
                        newStr := wordList[i][:j] + "*" + wordList[i][j+1:]
                        if _, ok := preMap[newStr]; !ok {
```

(续下页)

(接上页)

```
                                preMap[newStr] = make([]string, 0)
                        }
                        preMap[newStr] = append(preMap[newStr], wordList[i])
                }
        }
        visited := make(map[string]bool)
        count := 0
        queue := make([]string, 0)
        queue = append(queue, beginWord)
        for len(queue) > 0 {
                count++
                length := len(queue)
                for i := 0; i < length; i++ {
                        for j := 0; j < len(beginWord); j++ {
                                newStr := queue[i][:j] + "*" + queue[i][j+1:]
                                for _, word := range preMap[newStr] {
                                        if word == endWord {
                                                return count + 1
                                        }
                                        if visited[word] == false {
                                                visited[word] = true
                                                queue = append(queue, word)
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return 0
}

# 2
func ladderLength(beginWord string, endWord string, wordList []string) int {
        m := make(map[string]int)
        for i := 0; i < len(wordList); i++ {
                m[wordList[i]] = 1
        }
        if m[endWord] == 0 {
                return 0
        }
        queue := make([]string, 0)
        queue = append(queue, beginWord)
        count := 0
```

(续下页)

(接上页)

```
        for len(queue) > 0 {
                count++
                length := len(queue)
                for i := 0; i < length; i++ {
                        for _, word := range wordList {
                                diff := 0
                                for j := 0; j < len(queue[i]); j++ {
                                        if queue[i][j] != word[j] {
                                                diff++
                                        }
                                        if diff > 1 {
                                                break
                                        }
                                }
                                if diff == 1 && m[word] != 2 {
                                        if word == endWord {
                                                return count + 1
                                        }
                                        m[word] = 2
                                        queue = append(queue, word)
                                }
                        }
                }
                queue = queue[length:]
        }
        return 0
}
```

5.12 129. 求根到叶子节点数字之和 (2)

- 题目

```
给定一个二叉树，它的每个结点都存放一个 0-9␣
↪的数字，每条从根到叶子节点的路径都代表一个数字。
例如，从根到叶子节点路径 1->2->3 代表数字 123。
计算从根到叶子节点生成的所有数字之和。
说明: 叶子节点是指没有子节点的节点。
示例 1:输入: [1,2,3]
    1
   / \
  2   3
输出: 25
```

(续下页)

(接上页)

```
解释:
从根到叶子节点路径 1->2 代表数字 12.
从根到叶子节点路径 1->3 代表数字 13.
因此，数字总和 = 12 + 13 = 25.
示例 2:输入: [4,9,0,5,1]
    4
   / \
  9   0
 / \
5   1
输出: 1026
解释:
从根到叶子节点路径 4->9->5 代表数字 495.
从根到叶子节点路径 4->9->1 代表数字 491.
从根到叶子节点路径 4->0 代表数字 40.
因此，数字总和 = 495 + 491 + 40 = 1026。
```

- 解题思路

```
var res int

func sumNumbers(root *TreeNode) int {
        res = 0
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, sum int) {
        if root == nil {
                return
        }
        sum = sum*10 + root.Val
        if root.Left == nil && root.Right == nil {
                res = res + sum
        }
        dfs(root.Left, sum)
        dfs(root.Right, sum)
}

#
func sumNumbers(root *TreeNode) int {
        res := 0
        if root == nil {
                return res
```

(续下页)

(接上页)

```
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                for i := 0; i < length; i++ {
                        node := list[i]
                        value := node.Val
                        if node.Left == nil && node.Right == nil {
                                res = res + value
                        }
                        if node.Left != nil {
                                node.Left.Val = node.Left.Val + value*10
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
                                node.Right.Val = node.Right.Val + value*10
                                list = append(list, node.Right)
                        }
                }
                list = list[length:]
        }
        return res
}
```

5.13 130. 被围绕的区域 (2)

- 题目

```
给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。
找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。
示例：
X X X X
X O O X
X X O X
X O X X
运行你的函数后，矩阵变为：
X X X X
X X X X
X X X X
X O X X
解释：
```

(续下页)

(接上页)

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。
任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。
如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

- 解题思路

```
func solve(board [][]byte) {
        if board == nil || len(board) == 0 {
                return
        }
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        if (i == 0 || i == len(board)-1 || j == 0 || j ==␣
↪len(board[i])-1) &&
                                board[i][j] == 'O' {
                                dfs(board, i, j)
                        }
                }
        }
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        if board[i][j] == 'O' {
                                board[i][j] = 'X'
                        }
                        if board[i][j] == '#' {
                                board[i][j] = 'O'
                        }
                }
        }
}

func dfs(board [][]byte, i, j int) {
        if i < 0 || j < 0 || i >= len(board) || j >= len(board[0]) ||
                board[i][j] == '#' || board[i][j] == 'X' {
                return
        }
        board[i][j] = '#'
        dfs(board, i+1, j)
        dfs(board, i-1, j)
        dfs(board, i, j+1)
        dfs(board, i, j-1)
}

# 2
```

(续下页)

(接上页)

```
func solve(board [][]byte) {
        if board == nil || len(board) == 0 {
                return
        }
        n := len(board)
        m := len(board[0])
        fa = Init(n*m + 1)
        target := n * m
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if board[i][j] == 'O' {
                                if i == 0 || i == n-1 || j == 0 || j == m-1 {
                                        union(i*m+j, target)
                                } else {
                                        if board[i-1][j] == 'O' {
                                                union(i*m+j, (i-1)*m+j)
                                        }
                                        if board[i+1][j] == 'O' {
                                                union(i*m+j, (i+1)*m+j)
                                        }
                                        if board[i][j-1] == 'O' {
                                                union(i*m+j, i*m+j-1)
                                        }
                                        if board[i][j+1] == 'O' {
                                                union(i*m+j, i*m+j+1)
                                        }
                                }
                        }
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if board[i][j] == 'O' && find(i*m+j) != find(target) {
                                board[i][j] = 'X'
                        }
                }
        }
}

var fa []int

// 初始化
func Init(n int) []int {
```

(续下页)

(接上页)

```
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

5.14 131. 分割回文串 (2)

- 题目

```
给定一个字符串 s，将 s 分割成一些子串，使每个子串都是回文串。
返回 s 所有可能的分割方案。
示例:输入: "aab" 输出:
[
  ["aa","b"],
  ["a","a","b"]
]
```

- 解题思路

```
var res [][]string
```

(续下页)

(接上页)

```
func partition(s string) [][]string {
        res = make([][]string, 0)
        arr := make([]string, 0)
        dfs(s, 0, arr)
        return res
}

func dfs(s string, level int, arr []string) {
        if level == len(s) {
                temp := make([]string, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := level; i < len(s); i++ {
                str := s[level : i+1]
                if judge(str) == true {
                        dfs(s, i+1, append(arr, str))
                }
        }
}

func judge(s string) bool {
        for i := 0; i < len(s)/2; i++ {
                if s[i] != s[len(s)-1-i] {
                        return false
                }
        }
        return true
}

# 2
var res [][]string
var dp [][]bool

func partition(s string) [][]string {
        res = make([][]string, 0)
        arr := make([]string, 0)
        dp = make([][]bool, len(s))
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
                dp[r][r] = true
                for l := 0; l < r; l++ {
```

(续下页)

(接上页)

```
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
                        }
                }
        }
        dfs(s, 0, arr)
        return res
}

func dfs(s string, level int, arr []string) {
        if level == len(s) {
                temp := make([]string, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := level; i < len(s); i++ {
                str := s[level : i+1]
                if dp[level][i] == true {
                        dfs(s, i+1, append(arr, str))
                }
        }
}
```

5.15 133. 克隆图 (2)

- 题目

```
给你无向 连通 图中一个节点的引用，请你返回该图的 深拷贝（克隆）。
图中的每个节点都包含它的值 val（int） 和其邻居的列表（list[Node]）。
class Node {
    public int val;
    public List<Node> neighbors;
}
测试用例格式：简单起见，每个节点的值都和它的索引相同。
例如，第一个节点值为 1（val = 1），第二个节点值为 2（val = 2），以此类推。
该图在测试用例中使用邻接列表表示。
邻接列表 是用于表示有限图的无序列表的集合。每个列表都描述了图中节点的邻居集。
给定节点将始终是图中的第一个节点（值为 1）。你必须将 给定节点的拷贝
↪作为对克隆图的引用返回。
```

(续下页)

(接上页)

```
示例 1: 输入: adjList = [[2,4],[1,3],[2,4],[1,3]] 输出: [[2,4],[1,3],[2,4],[1,3]]
解释: 图中有 4 个节点。
节点 1 的值是 1, 它有两个邻居: 节点 2 和 4 。
节点 2 的值是 2, 它有两个邻居: 节点 1 和 3 。
节点 3 的值是 3, 它有两个邻居: 节点 2 和 4 。
节点 4 的值是 4, 它有两个邻居: 节点 1 和 3 。
示例 2: 输入: adjList = [[]] 输出: [[]]
解释: 输入包含一个空列表。该图仅仅只有一个值为 1 的节点, 它没有任何邻居。
示例 3: 输入: adjList = [] 输出: []
解释: 这个图是空的, 它不含任何节点。
示例 4: 输入: adjList = [[2],[1]] 输出: [[2],[1]]
提示:
    节点数不超过 100 。
    每个节点值 Node.val 都是唯一的, 1 <= Node.val <= 100。
    无向图是一个简单图, 这意味着图中没有重复的边, 也没有自环。
    由于图是无向的, 如果节点 p 是节点 q 的邻居, 那么节点 q 也必须是节点 p 的邻居。
    图是连通图, 你可以从给定节点访问到所有节点。
```

- 解题思路

```
var visited map[*Node]*Node

func cloneGraph(node *Node) *Node {
        visited = make(map[*Node]*Node)
        return clone(node)
}

func clone(node *Node) *Node {
        if node == nil {
                return node
        }
        if v, ok := visited[node]; ok {
                return v
        }
        newNode := &Node{
                Val:       node.Val,
                Neighbors: make([]*Node, len(node.Neighbors)),
        }
        visited[node] = newNode
        for i := 0; i < len(node.Neighbors); i++ {
                newNode.Neighbors[i] = clone(node.Neighbors[i])
        }
        return newNode
}
```

(续下页)

(接上页)

```
# 2
func cloneGraph(node *Node) *Node {
        if node == nil {
                return nil
        }
        queue := make([]*Node, 0)
        queue = append(queue, node)
        visited := make(map[*Node]*Node)
        visited[node] = &Node{
                Val:       node.Val,
                Neighbors: make([]*Node, len(node.Neighbors)),
        }
        for len(queue) > 0 {
                temp := queue[0]
                queue = queue[1:]
                for i, v := range temp.Neighbors {
                        if _, ok := visited[v]; !ok {
                                queue = append(queue, v)
                                visited[v] = &Node{
                                        Val:       v.Val,
                                        Neighbors: make([]*Node, len(v.Neighbors)),
                                }
                        }
                        visited[temp].Neighbors[i] = visited[v]
                }
        }
        return visited[node]
}
```

5.16 134. 加油站 (2)

- 题目

```
在一条环路上有 N 个加油站，其中第 i 个加油站有汽油 gas[i] 升。
你有一辆油箱容量无限的的汽车，从第 i 个加油站开往第 i+1 个加油站需要消耗汽油 cost[i]␣
↪升。
你从其中的一个加油站出发，开始时油箱为空。
如果你可以绕环路行驶一周，则返回出发时加油站的编号，否则返回 -1。
说明:
    如果题目有解，该答案即为唯一答案。
    输入数组均为非空数组，且长度相同。
```

(续下页)

(接上页)

```
    输入数组中的元素均为非负数。
示例 1:输入: gas  = [1,2,3,4,5] cost = [3,4,5,1,2] 输出: 3
解释:从 3 号加油站(索引为 3 处)出发，可获得 4 升汽油。此时油箱有 = 0 + 4 = 4 升汽油
开往 4 号加油站，此时油箱有 4 - 1 + 5 = 8 升汽油
开往 0 号加油站，此时油箱有 8 - 2 + 1 = 7 升汽油
开往 1 号加油站，此时油箱有 7 - 3 + 2 = 6 升汽油
开往 2 号加油站，此时油箱有 6 - 4 + 3 = 5 升汽油
开往 3 号加油站，你需要消耗 5 升汽油，正好足够你返回到 3 号加油站。
因此，3 可为起始索引。
示例 2:输入: gas  = [2,3,4] cost = [3,4,3]输出: -1
解释:你不能从 0 号或 1 号加油站出发，因为没有足够的汽油可以让你行驶到下一个加油站。
我们从 2 号加油站出发，可以获得 4 升汽油。 此时油箱有 = 0 + 4 = 4 升汽油
开往 0 号加油站，此时油箱有 4 - 3 + 2 = 3 升汽油
开往 1 号加油站，此时油箱有 3 - 3 + 3 = 3 升汽油
你无法返回 2 号加油站，因为返程需要消耗 4 升汽油，但是你的油箱只有 3 升汽油。
因此，无论怎样，你都不可能绕环路行驶一周。。
```

- 解题思路

```
func canCompleteCircuit(gas []int, cost []int) int {
        total, sum, start := 0, 0, 0
        for i := 0; i < len(gas); i++ {
                sum = sum + gas[i] - cost[i]
                total = total + gas[i] - cost[i]
                // 例如gas[i] - cost[i]的值为=> 1, 2, 3, 4, -11, 12
                // 1->-11 <0
                // 2->-11 <0
                // 3->-11 <0
                // 4->-11 <0
                // -11 < 0
                // i要是到不了j但是能到i和j之间的点(>=0),
                // 那么i和j之间的所有点都到不了b(<0)
                if sum < 0 {
                        start = i + 1
                        sum = 0
                }
        }
        if total < 0 {
                return -1
        }
        return start
}

#
```

(续下页)

(接上页)

```
func canCompleteCircuit(gas []int, cost []int) int {
        for i := 0; i < len(gas); i++ {
                total := 0
                for j := 0; j < len(gas); j++ {
                        total = total + gas[j]
                        if total < cost[j] {
                                break
                        } else {
                                if j == len(gas)-1 && total >= cost[j] {
                                        return i
                                }
                                total = total - cost[j]
                        }
                }
                gas = append(gas[1:], gas[0])
                cost = append(cost[1:], cost[0])
        }
        return -1
}
```

5.17 137. 只出现一次的数字 II(5)

- 题目

```
给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现了三次。找出那个只出现了一次的元
说明：
你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？
示例 1:输入: [2,2,3,2] 输出: 3
示例 2:输入: [0,1,0,1,0,1,99] 输出: 99
```

- 解题思路

```
func singleNumber(nums []int) int {
        m := make(map[int]int)
        for _, v := range nums {
                m[v]++
        }
        for k, v := range m {
                if v == 1 {
                        return k
                }
        }
```

(续下页)

(接上页)

```
        return 0
}

# 2
func singleNumber(nums []int) int {
        sort.Ints(nums)
        for i := 0; i < len(nums)-1; i=i+3{
                if nums[i] != nums[i+1]{
                        return nums[i]
                }
        }
        return nums[len(nums)-1]
}

# 3
func singleNumber(nums []int) int {
        var res int
        for i := 0; i < 64; i++ {
                count := 0
                for j := 0; j < len(nums); j++ {
                        if (nums[j]>>i)&1 == 1 {
                                count++
                        }
                }
                res = res | ((count % 3) << i) // 哪一位出现求余后1次，该位置为1
        }
        return res
}

# 4
func singleNumber(nums []int) int {
        a, b := 0, 0
        for i := 0; i < len(nums); i++ {
                a = (a ^ nums[i]) & (^b) // a: 保留出现1次的数
                b = (b ^ nums[i]) & (^a) // b: 保留出现2次的数
        }
        return a // 最后返回只出现1次的数
}

# 5
func singleNumber(nums []int) int {
        m := make(map[int]int)
        sum := 0
```

(续下页)

(接上页)

```
        singleSum := 0
        for _, v := range nums {
                if m[v] == 0{
                        singleSum = singleSum+v
                }
                m[v] = 1
                sum = sum + v
        }
        return (singleSum*3-sum)/2
}
```

5.18 138. 复制带随机指针的链表 (3)

- 题目

```
给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。
要求返回这个链表的 深拷贝。
我们用一个由 n 个节点组成的链表来表示输入/输出中的链表。每个节点用一个 [val, random_
↪index] 表示：
    val：一个表示 Node.val 的整数。
    random_index：随机指针指向的节点索引（范围从 0 到 n-1）；如果不指向任何节点，则为␣
↪ null 。
示例 1：输入：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
输出：[[7,null],[13,0],[11,4],[10,2],[1,0]]
示例 2：输入：head = [[1,1],[2,1]] 输出：[[1,1],[2,1]]
示例 3：输入：head = [[3,null],[3,0],[3,null]] 输出：[[3,null],[3,0],[3,null]]
示例 4：输入：head = [] 输出：[]
解释：给定的链表为空（空指针），因此返回 null。
提示：
    -10000 <= Node.val <= 10000
    Node.random 为空（null）或指向链表中的节点。
    节点数目不超过 1000 。
```

- 解题思路

```
var m map[*Node]*Node

func copyRandomList(head *Node) *Node {
        m = make(map[*Node]*Node)
        return copyList(head)
}
```

(续下页)

(接上页)

```
func copyList(head *Node) *Node {
        if head == nil {
                return head
        }
        if node, ok := m[head]; ok {
                return node
        }
        temp := &Node{
                Val:    head.Val,
                Next:   nil,
                Random: nil,
        }
        m[head] = temp
        temp.Next = copyList(head.Next)
        temp.Random = copyList(head.Random)
        return temp
}

# 2
func copyRandomList(head *Node) *Node {
        if head == nil {
                return nil
        }
        res := new(Node)
        m := make(map[*Node]*Node)
        temp := head
        p := res
        for temp != nil {
                node := &Node{
                        Val:    temp.Val,
                        Next:   nil,
                        Random: nil,
                }
                m[temp] = node
                p.Next = node
                p = p.Next
                temp = temp.Next

        }
        temp = head
        p = res.Next
        for temp != nil {
                p.Random = m[temp.Random]
```

(续下页)

(接上页)

```
                p = p.Next
                temp = temp.Next
        }
        return res.Next
}

# 3
func copyRandomList(head *Node) *Node {
        if head == nil {
                return nil
        }
        res := copyNext(head)
        res = copyRandom(res)
        res = cutEven(res)
        return res
}

// 原1-复制1-原2-复制2
func copyNext(head *Node) *Node {
        p := head
        for p != nil {
                node := new(Node)
                node.Val = p.Val
                node.Next = p.Next
                p.Next = node
                p = node.Next
        }
        return head
}

func copyRandom(head *Node) *Node {
        p := head
        for p != nil {
                if p.Random != nil {
                        p.Next.Random = p.Random.Next
                }
                p = p.Next.Next
        }
        return head
}

func cutEven(head *Node) *Node {
        oldNode := head
```

(续下页)

(接上页)

```
        newNode := head.Next
        cur := newNode
        for oldNode != nil {
                oldNode.Next = oldNode.Next.Next
                if newNode.Next != nil{
                        newNode.Next = newNode.Next.Next
                }
                oldNode = oldNode.Next
                newNode = newNode.Next
        }
        return cur
}
```

5.19 139. 单词拆分 (2)

- 题目

```
给定一个非空字符串 s 和一个包含非空单词列表的字典 wordDict,
判定 s 是否可以被空格拆分为一个或多个在字典中出现的单词。
说明:
    拆分时可以重复使用字典中的单词。
    你可以假设字典中没有重复的单词。
示例 1: 输入: s = "leetcode", wordDict = ["leet", "code"] 输出: true
解释: 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。
示例 2: 输入: s = "applepenapple", wordDict = ["apple", "pen"] 输出: true
解释: 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。
     注意你可以重复使用字典中的单词。
示例 3: 输入: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"] 输出:␣
↪false
```

- 解题思路

```
func wordBreak(s string, wordDict []string) bool {
        m := make(map[string]bool)
        for i := 0; i < len(wordDict); i++{
                m[wordDict[i]] = true
        }
        dp := make([]bool, len(s)+1)
        dp[0] = true
        for i := 1; i <= len(s); i++{
                for j := 0; j < i; j++{
                        if dp[j] == true && m[s[j:i]] == true{
```

(续下页)

(接上页)

```
                                dp[i] = true
                                break
                        }
                }
        }
        return dp[len(s)]
}

# 2
var m map[string]bool
var visited map[int]bool

func wordBreak(s string, wordDict []string) bool {
        m = make(map[string]bool)
        for i := 0; i < len(wordDict); i++ {
                m[wordDict[i]] = true
        }
        visited = make(map[int]bool)
        return wordbreak(s, 0)
}

func wordbreak(s string, start int) bool {
        if start == len(s) {
                return true
        }
        if _, ok := visited[start]; ok {
                return visited[start]
        }
        //递归
        for i := start; i < len(s); i++ {
                if _, ok := m[s[start:i+1]]; ok && wordbreak(s, i+1) {
                        visited[start] = true
                        return true
                }
        }
        visited[start] = false
        return false
}
```

5.20 142. 环形链表 II(3)

- 题目

```
给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。
为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0
↪开始）。
如果 pos 是 -1，则在该链表中没有环。
说明：不允许修改给定的链表。
示例 1：输入：head = [3,2,0,-4], pos = 1 输出：tail connects to node index 1
解释：链表中有一个环，其尾部连接到第二个节点。
示例 2：输入：head = [1,2], pos = 0 输出：tail connects to node index 0
解释：链表中有一个环，其尾部连接到第一个节点。
示例 3：输入：head = [1], pos = -1 输出：no cycle
解释：链表中没有环。
进阶：你是否可以不用额外空间解决此题？
```

- 解题思路

```
func detectCycle(head *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for head != nil {
                if m[head] {
                        return head
                }
                m[head] = true
                head = head.Next
        }
        return nil
}

# 2
func detectCycle(head *ListNode) *ListNode {
        if head == nil {
                return nil
        }
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
                if fast == slow {
                        break
                }
        }
```

(续下页)

(接上页)

```
        if fast == nil || fast.Next == nil {
                return nil
        }
        slow = head
        for fast != slow {
                fast = fast.Next
                slow = slow.Next
        }
        return slow
}

# 3
func detectCycle(head *ListNode) *ListNode {
        for head != nil {
                if head.Val == math.MaxInt32 {
                        return head
                }
                head.Val = math.MaxInt32
                head = head.Next
        }
        return head
}
```

5.21 143. 重排链表 (4)

- 题目

```
给定一个单链表 L: L0→L1→…→Ln-1→Ln ,
将其重新排列后变为: L0→Ln→L1→Ln-1→L2→Ln-2→…
你不能只是单纯的改变节点内部的值, 而是需要实际的进行节点交换。
示例 1:给定链表 1->2->3->4, 重新排列为 1->4->2->3.
示例 2:给定链表 1->2->3->4->5, 重新排列为 1->5->2->4->3.
```

- 解题思路

```
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
                return
        }
        cur := head
        arr := make([]*ListNode, 0)
        for cur != nil {
```

(续下页)

(接上页)

```
                arr = append(arr, cur)
                cur = cur.Next
        }
        res := make([]*ListNode, 0)
        for i := 0; i < len(arr)/2; i++ {
                res = append(res, arr[i], arr[len(arr)-1-i])
        }
        if len(arr)%2 == 1 {
                res = append(res, arr[len(arr)/2])
        }
        cur = head
        for i := 1; i < len(res); i++ {
                cur.Next = res[i]
                cur = cur.Next
        }
        cur.Next = nil
}

# 2
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
                return
        }
        cur, prev, next := head, head, head
        for cur != nil {
                next = cur.Next
                // prev 指向n-1 prev.next 指向n
                for prev = next; prev != nil && prev.Next != nil && prev.Next.Next !=␣
↪nil; {
                        prev = prev.Next
                }
                if prev != nil && prev.Next != nil {
                        cur.Next = prev.Next
                        prev.Next.Next = next
                        prev.Next = nil
                }
                cur = next
        }
}

# 3
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
```

(续下页)

(接上页)

```
                return
        }
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
        }
        second := reverse(slow.Next)
        slow.Next = nil
        cur := head
        count := 0
        for cur != nil && second != nil {
                a := cur.Next
                b := second.Next
                if count%2 == 0 {
                        cur.Next = second
                        cur = a
                } else {
                        second.Next = cur
                        second = b
                }
                count++
        }
}

func reverse(head *ListNode) *ListNode {
        var res *ListNode
        for head != nil {
                next := head.Next
                head.Next = res
                res = head
                head = next
        }
        return res
}

# 4
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
                return
        }
        length := 0
        cur := head
```

(续下页)

(接上页)

```
        for cur != nil {
                length++
                cur = cur.Next
        }
        helper(head, length)
}

func helper(head *ListNode, length int) *ListNode {
        if length == 1 {
                next := head.Next
                head.Next = nil
                return next
        }
        if length == 2 {
                next := head.Next.Next
                head.Next.Next = nil
                return next
        }
        tail := helper(head.Next, length-2)
        next := tail.Next
        temp := head.Next
        head.Next = tail
        tail.Next = temp
        return next
}
```

5.22 144. 二叉树的前序遍历 (3)

- 题目

```
给定一个二叉树，返回它的 前序 遍历。
示例:输入: [1,null,2,3]
   1
    \
     2
    /
   3
输出: [1,2,3]
进阶: 递归算法很简单，你可以通过迭代算法完成吗？
```

- 解题思路

```
var res []int

func preorderTraversal(root *TreeNode) []int {
        res = make([]int, 0)
        dfs(root)
        return res
}

func dfs(root *TreeNode) {
        if root != nil {
                res = append(res, root.Val)
                dfs(root.Left)
                dfs(root.Right)
        }
}

# 2
func preorderTraversal(root *TreeNode) []int {
        res := make([]int, 0)
        stack := make([]*TreeNode, 0)
        for len(stack) > 0 || root != nil {
                for root != nil {
                        res = append(res, root.Val)
                        stack = append(stack, root.Right)
                        root = root.Left
                }
                last := len(stack) - 1
                root = stack[last]
                stack = stack[:last]
        }
        return res
}

# 3
func preorderTraversal(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil {
                return res
        }
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
```

(续下页)

(接上页)

```
                res = append(res, node.Val)
                if node.Right != nil {
                        stack = append(stack, node.Right)
                }
                if node.Left != nil {
                        stack = append(stack, node.Left)
                }
        }
        return res
}
```

5.23 146.LRU 缓存机制 (1)

- 题目

```
运用你所掌握的数据结构，设计和实现一个 LRU (最近最少使用) 缓存机制。
它应该支持以下操作： 获取数据 get 和 写入数据 put 。
获取数据 get(key) - 如果关键字 (key)␣
↪存在于缓存中，则获取关键字的值（总是正数），否则返回 -1。
写入数据 put(key, value) - 如果关键字已经存在，则变更其数据值；
如果关键字不存在，则插入该组「关键字/值」。
当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。
进阶:你是否可以在 O(1) 时间复杂度内完成这两种操作？
示例:LRUCache cache = new LRUCache( 2 /* 缓存容量 */ );
cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // 返回  1
cache.put(3, 3);    // 该操作会使得关键字 2 作废
cache.get(2);       // 返回 -1 (未找到)
cache.put(4, 4);    // 该操作会使得关键字 1 作废
cache.get(1);       // 返回 -1 (未找到)
cache.get(3);       // 返回  3
cache.get(4);       // 返回  4
```

- 解题思路

```
type Node struct {
        key   int
        value int
        prev  *Node
        next  *Node
}
```

(续下页)

(接上页)

```
type LRUCache struct {
        cap    int
        header *Node
        tail   *Node
        m      map[int]*Node
}

func Constructor(capacity int) LRUCache {
        cache := LRUCache{
                cap:    capacity,
                header: &Node{},
                tail:   &Node{},
                m:      make(map[int]*Node, capacity),
        }
        cache.header.next = cache.tail
        cache.tail.prev = cache.header
        return cache
}

func (this *LRUCache) Get(key int) int {
        if node, ok := this.m[key]; ok {
                this.remove(node)
                this.putHead(node)
                return node.value
        }
        return -1
}

func (this *LRUCache) Put(key int, value int) {
        if node, ok := this.m[key]; ok {
                node.value = value
                this.remove(node)
                this.putHead(node)
                return
        }
        if this.cap <= len(this.m) {
                // 删除尾部
                deleteKey := this.tail.prev.key
                this.remove(this.tail.prev)
                delete(this.m, deleteKey)
        }
        // 插入到头部
```

(续下页)

(接上页)

```
        newNode := &Node{key: key, value: value}
        this.putHead(newNode)
        this.m[key] = newNode
}

// 删除尾部节点
func (this *LRUCache) remove(node *Node) {
        node.prev.next = node.next
        node.next.prev = node.prev
}

// 插入头部
func (this *LRUCache) putHead(node *Node) {
        next := this.header.next
        this.header.next = node
        node.next = next
        next.prev = node
        node.prev = this.header
}
```

5.24 147. 对链表进行插入排序 (2)

- 题目

```
对链表进行插入排序。
插入排序的动画演示如上。从第一个元素开始，该链表可以被认为已经部分排序（用黑色表示）。
每次迭代时，从输入数据中移除一个元素（用红色表示），并原地将其插入到已排好序的链表中。
插入排序算法：
    插入排序是迭代的，每次只移动一个元素，直到所有元素可以形成一个有序的输出列表。
    ␣
↪每次迭代中，插入排序只从输入数据中移除一个待排序的元素，找到它在序列中适当的位置，并将其插入。
    重复直到所有输入数据插入完为止。
示例 1：输入：4->2->1->3 输出：1->2->3->4
示例 2：输入：-1->5->3->4->0 输出：-1->0->3->4->5
```

- 解题思路

```
func insertionSortList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        res := &ListNode{Next: head}
```

(续下页)

(接上页)

```
        cur := head.Next
        head.Next = nil
        for cur != nil {
                next := cur.Next
                prev := res // 从头开始寻找插入点
                for prev.Next != nil && prev.Next.Val <= cur.Val {
                        prev = prev.Next
                }
                // 插入操作
                cur.Next = prev.Next
                prev.Next = cur
                // 指向下一个未排序节点
                cur = next
        }
        return res.Next
}

# 2
func insertionSortList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        arr := make([]*ListNode, 0)
        for head != nil {
                arr = append(arr, head)
                head = head.Next
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].Val < arr[j].Val
        })
        res := &ListNode{Next: head}
        cur := res
        arr[len(arr)-1].Next = nil
        for i := 0; i < len(arr); i++ {
                cur.Next = arr[i]
                cur = cur.Next
        }
        return res.Next
}
```

5.25 148. 排序链表 (3)

- 题目

```
在 O(n log n) 时间复杂度和常数级空间复杂度下，对链表进行排序。
示例 1:输入: 4->2->1->3 输出: 1->2->3->4
示例 2:输入: -1->5->3->4->0 输出: -1->0->3->4->5
```

- 解题思路

```
func sortList(head *ListNode) *ListNode {
        quickSort(head, nil)
        return head
}

func quickSort(head, end *ListNode) {
        if head == end || head.Next == end {
                return
        }
        temp := head.Val
        fast, slow := head.Next, head
        for fast != end {
                if fast.Val < temp {
                        slow = slow.Next
                        slow.Val, fast.Val = fast.Val, slow.Val
                }
                fast = fast.Next
        }
        slow.Val, head.Val = head.Val, slow.Val
        quickSort(head, slow)
        quickSort(slow.Next, end)
}

# 2
func sortList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        slow, fast := head, head.Next
        for fast != nil && fast.Next != nil {
                slow = slow.Next
                fast = fast.Next.Next
        }
        right := sortList(slow.Next)
```

(续下页)

(接上页)

```
        slow.Next = nil
        left := sortList(head)
        return mergeTwoLists(left, right)
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}

# 3
func sortList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        res := &ListNode{Next: head}
        cur := head
        var left, right *ListNode
        length := 0
        for cur != nil {
                length++
                cur = cur.Next
        }
        for i := 1; i < length; i = i * 2 {
                cur = res.Next
                tail := res
                for cur != nil {
```

(续下页)

(接上页)

```
                        left = cur
                        right = split(left, i)
                        cur = split(right, i)
                        tail.Next = mergeTwoLists(left, right)
                        for tail.Next != nil {
                                tail = tail.Next
                        }
                }
        }
        return res.Next
}

func split(head *ListNode, length int) *ListNode {
        cur := head
        var right *ListNode
        length--
        for length > 0 && cur != nil {
                length--
                cur = cur.Next
        }
        if cur == nil {
                return nil
        }
        right = cur.Next
        cur.Next = nil
        return right
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
```

(续下页)

(接上页)

```
        } else {
                temp.Next = l2
        }
        return res.Next
}
```

5.26 150. 逆波兰表达式求值 (1)

- 题目

```
根据 逆波兰表示法，求表达式的值。
有效的运算符包括 +, -, *, / 。每个运算对象可以是整数，也可以是另一个逆波兰表达式。
说明：
    整数除法只保留整数部分。
    给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0␣
↪的情况。
示例 1：输入：["2", "1", "+", "3", "*"] 输出：9
解释：该算式转化为常见的中缀算术表达式为：((2 + 1) * 3) = 9
示例 2：输入：["4", "13", "5", "/", "+"] 输出：6
解释：该算式转化为常见的中缀算术表达式为：(4 + (13 / 5)) = 6
示例 3：  输入：["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"]
输出：22
解释：
该算式转化为常见的中缀算术表达式为：
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
逆波兰表达式：
逆波兰表达式是一种后缀表达式，所谓后缀就是指算符写在后面。
    平常使用的算式则是一种中缀表达式，如 ( 1 + 2 ) * ( 3 + 4 ) 。
    该算式的逆波兰表达式写法为 ( ( 1 2 + ) ( 3 4 + ) * ) 。
逆波兰表达式主要有以下两个优点：
    去掉括号后表达式无歧义，上式即便写成 1 2 + 3 4 + * 也可以依据次序计算出正确结果。
  ␣
↪适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中。
```

- 解题思路

```
func evalRPN(tokens []string) int {
        stack := make([]int, 0)
        for _, v := range tokens {
                length := len(stack)
                if v == "+" || v == "-" || v == "*" || v == "/" {
                        a := stack[length-2]
                        b := stack[length-1]
                        stack = stack[:length-2]
                        var value int
                        if v == "+" {
                                value = a + b
                        } else if v == "-" {
                                value = a - b
                        } else if v == "*" {
                                value = a * b
                        } else {
                                value = a / b
                        }
                        stack = append(stack, value)
                } else {
                        value, _ := strconv.Atoi(v)
                        stack = append(stack, value)
                }
        }
        return stack[0]
}
```

5.27 151. 翻转字符串里的单词 (2)

- 题目

```
给定一个字符串，逐个翻转字符串中的每个单词。
示例 1: 输入: "the sky is blue" 输出: "blue is sky the"
示例 2: 输入: "  hello world!  " 输出: "world! hello"
解释: 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
示例 3: 输入: "a good   example" 输出: "example good a"
解释: 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
说明:
    无空格字符构成一个单词。
    输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
    如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
进阶: 请选用 C 语言的用户尝试使用 O(1) 额外空间复杂度的原地解法。
```

- 解题思路

```
func reverseWords(s string) string {
        arr := strings.Fields(s)
        for i := 0; i < len(arr)/2; i++{
                arr[i], arr[len(arr)-1-i] =  arr[len(arr)-1-i],arr[i]
        }
        return strings.Join(arr," ")
}

#
func reverseWords(s string) string {
        arr := make([]string, 0)
        i, j := 0, 0
        for i < len(s) && j <= len(s) {
                for i = j; i < len(s) && s[i] == ' '; i++ {
                }
                for j = i; j < len(s) && s[j] != ' '; j++ {
                }
                if i < j {
                        arr = append(arr, s[i:j])
                }
        }
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
        return strings.Join(arr, " ")
}
```

5.28 152. 乘积最大子数组 (2)

- 题目

```
给你一个整数数组 nums␣
↪，请你找出数组中乘积最大的连续子数组（该子数组中至少包含一个数字），
并返回该子数组所对应的乘积。
示例 1:输入: [2,3,-2,4] 输出: 6
解释: 子数组 [2,3] 有最大乘积 6。
示例 2:输入: [-2,0,-1] 输出: 0
解释: 结果不能为 2, 因为 [-2,-1] 不是子数组。
```

-解题思路

```
func maxProduct(nums []int) int {
        minValue, maxValue, res := nums[0], nums[0], nums[0]
        for i := 1; i < len(nums); i++ {
                minV, maxV := minValue, maxValue
                minValue = min(minV*nums[i], min(nums[i], maxV*nums[i]))
                maxValue = max(maxV*nums[i], max(nums[i], minV*nums[i]))
                res = max(res, maxValue)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func maxProduct(nums []int) int {
        res := math.MinInt64
        for i := 0; i < len(nums); i++ {
                temp := 1
                for j := i; j < len(nums); j++ {
                        temp = temp * nums[j]
                        if temp > res {
                                res = temp
                        }
                }
        }
        return res
}
```

5.29 153. 寻找旋转排序数组中的最小值 (2)

- 题目

```
假设按照升序排序的数组在预先未知的某个点上进行了旋转。
( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。
请找出其中最小的元素。
你可以假设数组中不存在重复元素。
示例 1:输入: [3,4,5,1,2]输出: 1
示例 2:输入: [4,5,6,7,0,1,2]输出: 0
```

- 解题思路

```
func findMin(nums []int) int {
        res := nums[0]
        for i := 1; i < len(nums); i++ {
                if nums[i] < res {
                        res = nums[i]
                }
        }
        return res
}

# 2
func findMin(nums []int) int {
        left, right := 0, len(nums)-1
        for left < right {
                mid := left + (right-left)/2
                if nums[mid] > nums[right] {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return nums[left]
}
```

5.30 162. 寻找峰值 (3)

- 题目

```
峰值元素是指其值大于左右相邻值的元素。
给定一个输入数组 nums，其中 nums[i] ≠ nums[i+1]，找到峰值元素并返回其索引。
数组可能包含多个峰值，在这种情况下，返回任何一个峰值所在位置即可。
你可以假设 nums[-1] = nums[n] = -∞。
示例 1:输入: nums = [1,2,3,1] 输出: 2
解释: 3 是峰值元素，你的函数应该返回其索引 2。
示例 2:输入: nums = [1,2,1,3,5,6,4] 输出: 1 或 5
解释: 你的函数可以返回索引 1，其峰值元素为 2;
     或者返回索引 5， 其峰值元素为 6。
说明:你的解法应该是 O(logN) 时间复杂度的。
```

- 解题思路

```
func findPeakElement(nums []int) int {
        n := len(nums)
        if n == 1 {
                return 0
        }
        for i := 0; i < n; i++ {
                if i == 0 && i+1 < n && nums[i] > nums[i+1] {
                        return i
                }
                if i == n-1 && i-1 >= 0 && nums[i] > nums[i-1] {
                        return i
                }
                if i-1 >= 0 && i+1 < n && nums[i] > nums[i+1] && nums[i] > nums[i-1] {
                        return i
                }
        }
        return -1
}

# 2
func findPeakElement(nums []int) int {
        n := len(nums)
        if n == 1 {
                return 0
        }
        left := 0
        right := n - 1
```

(续下页)

(接上页)

```
        for left < right {
                mid := left + (right-left)/2
                if nums[mid] > nums[mid+1] {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

# 3
func findPeakElement(nums []int) int {
        n := len(nums)
        if n == 1 {
                return 0
        }
        for i := 0; i < n-1; i++ {
                if nums[i] > nums[i+1] {
                        return i
                }
        }
        return n - 1
}
```

5.31 165. 比较版本号 (2)

- 题目

```
比较两个版本号 version1 和 version2。
如果 version1 > version2 返回 1，如果 version1 < version2 返回 -1， 除此之外返回 0。
你可以假设版本字符串非空，并且只包含数字和 . 字符。
 . 字符不代表小数点，而是用于分隔数字序列。
例如，2.5 不是“两个半”，也不是“差一半到三”，而是第二版中的第五个小版本。
你可以假设版本号的每一级的默认修订版号为 0。
例如，版本号 3.4 的第一级（大版本）和第二级（小版本）修订号分别为 3 和␣
↪4。其第三级和第四级修订号均为 0。
示例 1:输入: version1 = "0.1", version2 = "1.1" 输出: -1
示例 2:输入: version1 = "1.0.1", version2 = "1" 输出: 1
示例 3:输入: version1 = "7.5.2.4", version2 = "7.5.3" 输出: -1
示例 4: 输入: version1 = "1.01", version2 = "1.001" 输出: 0
解释: 忽略前导零，“01” 和 “001” 表示相同的数字 “1”。
```

(续下页)

(接上页)

```
示例 5: 输入: version1 = "1.0", version2 = "1.0.0" 输出: 0
解释: version1 没有第三级修订号, 这意味着它的第三级修订号默认为 “0” 。
提示:
    版本字符串由以点 (.) 分隔的数字字符串组成。这个数字字符串可能有前导零。
    版本字符串不以点开始或结束, 并且其中不会有两个连续的点。
```

- 解题思路

```
func compareVersion(version1 string, version2 string) int {
        arr1 := strings.Split(version1, ".")
        arr2 := strings.Split(version2, ".")
        for len(arr1) < len(arr2) {
                arr1 = append(arr1, "0")
        }
        for len(arr2) < len(arr1) {
                arr2 = append(arr2, "0")
        }
        for i := 0; i < len(arr1); i++ {
                a, _ := strconv.Atoi(arr1[i])
                b, _ := strconv.Atoi(arr2[i])
                if a > b {
                        return 1
                } else if a < b {
                        return -1
                }
        }
        return 0
}

#
func compareVersion(version1 string, version2 string) int {
        arr1 := strings.Split(version1, ".")
        arr2 := strings.Split(version2, ".")
        for len(arr1) < len(arr2) {
                arr1 = append(arr1, "0")
        }
        for len(arr2) < len(arr1) {
                arr2 = append(arr2, "0")
        }
        for i := 0; i < len(arr1); i++ {
                a := strings.TrimLeft(arr1[i], "0")
                b := strings.TrimLeft(arr2[i], "0")
                // 1 < 10 => 01 < 10
                for len(a) < len(b) {
```

(续下页)

(接上页)

```
                        a = "0" + a
                }
                for len(b) < len(a) {
                        b = "0" + b
                }
                for j := 0; j < len(a); j++ {
                        if a[j] < b[j] {
                                return -1
                        } else if a[j] > b[j] {
                                return 1
                        }
                }
        }
        return 0
}
```

5.32 166. 分数到小数 (1)

- 题目

```
给定两个整数，分别表示分数的分子 numerator 和分母 denominator，以字符串形式返回小数。
如果小数部分为循环小数，则将循环的部分括在括号内。
示例 1:输入: numerator = 1, denominator = 2 输出: "0.5"
示例 2:输入: numerator = 2, denominator = 1 输出: "2"
示例 3:输入: numerator = 2, denominator = 3 输出: "0.(6)"
```

- 解题思路

```
func fractionToDecimal(numerator int, denominator int) string {
        res := make([]string, 0)
        if numerator == 0 {
                return "0"
        }
        // 预处理
        flag := false
        if numerator < 0 {
                flag = !flag
                numerator = -numerator
        }
        if denominator < 0 {
                flag = !flag
                denominator = -denominator
```

(续下页)

(接上页)

```
        }
        if flag == true {
                res = append(res, "-")
        }
        a, b := numerator/denominator, numerator%denominator
        res = append(res, strconv.Itoa(a))
        if b == 0 {
                return strings.Join(res, "")
        }
        res = append(res, ".")
        m := make(map[int]int)
        last, index := -1, len(res)
        for b > 0 {
                b = b * 10
                if v, ok := m[b]; ok {
                        last = v
                        break
                } else {
                        m[b] = index
                }
                index++
                a, b = b/denominator, b%denominator
                res = append(res, strconv.Itoa(a))
        }
        if last != -1 {
                res = append(res[:last], append([]string{"("}, res[last:]...)...)
                res = append(res, ")")
        }
        return strings.Join(res, "")
}
```

5.33 173. 二叉搜索树迭代器 (2)

- 题目

```
实现一个二叉搜索树迭代器。你将使用二叉搜索树的根节点初始化迭代器。
调用 next() 将返回二叉搜索树中的下一个最小的数。
示例：BSTIterator iterator = new BSTIterator(root);
iterator.next();    // 返回 3
iterator.next();    // 返回 7
iterator.hasNext(); // 返回 true
iterator.next();    // 返回 9
```

(续下页)

(接上页)

```
iterator.hasNext(); // 返回 true
iterator.next();    // 返回 15
iterator.hasNext(); // 返回 true
iterator.next();    // 返回 20
iterator.hasNext(); // 返回 false
提示：next() 和 hasNext() 操作的时间复杂度是 O(1)，并使用 O(h) 内存，其中 h␣
↪是树的高度。
    你可以假设 next() 调用总是有效的，也就是说，当调用 next() 时，BST␣
↪中至少存在一个下一个最小的数。
```

- 解题思路

```
type BSTIterator struct {
        arr  []int
        root *TreeNode
}

func Constructor(root *TreeNode) BSTIterator {
        arr := make([]int, 0)
        inorder(root, &arr)
        return BSTIterator{
                arr:  arr,
                root: root,
        }
}

func inorder(root *TreeNode, nums *[]int) {
        if root == nil {
                return
        }
        inorder(root.Left, nums)
        *nums = append(*nums, root.Val)
        inorder(root.Right, nums)
}

func (this *BSTIterator) Next() int {
        if len(this.arr) == 0 {
                return -1
        }
        res := this.arr[0]
        this.arr = this.arr[1:]
        return res
}
```

(续下页)

(接上页)

```
func (this *BSTIterator) HasNext() bool {
        if len(this.arr) > 0 {
                return true
        }
        return false
}

# 2
type BSTIterator struct {
        stack []*TreeNode
}

func Constructor(root *TreeNode) BSTIterator {
        res := BSTIterator{}
        res.left(root)
        return res
}

func (this *BSTIterator) left(root *TreeNode) {
        for root != nil {
                this.stack = append(this.stack, root)
                root = root.Left
        }
}

func (this *BSTIterator) Next() int {
        node := this.stack[len(this.stack)-1]
        this.stack = this.stack[:len(this.stack)-1]
        if node.Right != nil {
                this.left(node.Right)
        }
        return node.Val
}

func (this *BSTIterator) HasNext() bool {
        return len(this.stack) > 0
}
```

5.34 179. 最大数 (2)

- 题目

```
给定一组非负整数，重新排列它们的顺序使之组成一个最大的整数。
示例 1:输入: [10,2] 输出: 210
示例 2:输入: [3,30,34,5,9] 输出: 9534330
说明: 输出结果可能非常大，所以你需要返回一个字符串而不是整数。
```

- 解题思路

```
func largestNumber(nums []int) string {
        arr := make([]string, 0)
        for i := 0; i < len(nums); i++ {
                arr = append(arr, strconv.Itoa(nums[i]))
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i]+arr[j] >= arr[j]+arr[i]
        })
        res := strings.Join(arr, "")
        if res[0] == '0' {
                return "0"
        }
        return res
}

#
func largestNumber(nums []int) string {
        sort.Slice(nums, func(i, j int) bool {
                return fmt.Sprintf("%d%d", nums[i], nums[j]) >=
                        fmt.Sprintf("%d%d", nums[j], nums[i])
        })
        res := ""
        for i := 0; i < len(nums); i++ {
                res = res + strconv.Itoa(nums[i])
        }
        if res[0] == '0' {
                return "0"
        }
        return res
}
```

5.35 187. 重复的 DNA 序列 (1)

- 题目

```
所有 DNA 都由一系列缩写为 A，C，G 和 T 的核苷酸组成，例如：“ACGAATTCCG”。
在研究 DNA 时，识别 DNA 中的重复序列有时会对研究非常有帮助。
编写一个函数来查找目标子串，目标子串的长度为 10，且在 DNA 字符串 s␣
↪中出现次数超过一次。
示例：输入：s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT" 输出：["AAAAACCCCC", "CCCCCAAAAA"]
```

- 解题思路

```
func findRepeatedDnaSequences(s string) []string {
        res := make([]string, 0)
        m := make(map[string]int)
        // 可以采用其他的形式作为key
        for i := 0; i < len(s)-9; i++ {
                m[s[i:i+10]]++
        }
        for k, v := range m {
                if v > 1 {
                        res = append(res, k)
                }
        }
        return res
}
```

5.36 199. 二叉树的右视图 (2)

- 题目

```
给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。
示例:输入：[1,2,3,null,5,null,4] 输出：[1, 3, 4]
解释:
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

- 解题思路

```
func rightSideView(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil{
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                res = append(res, list[0].Val)
                for i := 0; i < length; i++ {
                        node := list[i]
                        if node.Right != nil {
                                list = append(list, node.Right)
                        }
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                }
                list = list[length:]
        }
        return res
}

#
var res []int

func rightSideView(root *TreeNode) []int {
        res = make([]int, 0)
        if root == nil {
                return res
        }
        dfs(root, 1)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        if level > len(res) {
                res = append(res, root.Val)
        }
        dfs(root.Right, level+1)
```

(续下页)

(接上页)

```
        dfs(root.Left, level+1)
}
```

5.37 200. 岛屿数量 (2)

- 题目

```
给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。
岛屿总是被水包围，并且每座岛屿只能由水平方向或竖直方向上相邻的陆地连接形成。
此外，你可以假设该网格的四条边均被水包围。
示例 1:输入:
[
['1','1','1','1','0'],
['1','1','0','1','0'],
['1','1','0','0','0'],
['0','0','0','0','0']
]
输出: 1
示例 2:输入:
[
['1','1','0','0','0'],
['1','1','0','0','0'],
['0','0','1','0','0'],
['0','0','0','1','1']
]
输出: 3
解释: 每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。
```

- 解题思路

```
func numIslands(grid [][]byte) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == '1' {
                                dfs(grid, i, j)
                                res++
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func dfs(grid [][]byte, i, j int) {
        if i < 0 || j < 0 || i >= len(grid) || j >= len(grid[0]) ||
                grid[i][j] == '0' {
                return
        }
        grid[i][j] = '0'
        dfs(grid, i+1, j)
        dfs(grid, i-1, j)
        dfs(grid, i, j+1)
        dfs(grid, i, j-1)
}

# 2
func numIslands(grid [][]byte) int {
        n := len(grid)
        m := len(grid[0])
        fa = Init(n*m + 1)
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == '1' {
                                count++
                                grid[i][j] = '0'
                                if i >= 1 && grid[i-1][j] == '1' {
                                        union(i*m+j, (i-1)*m+j)
                                }
                                if i < n-1 && grid[i+1][j] == '1' {
                                        union(i*m+j, (i+1)*m+j)
                                }
                                if j >= 1 && grid[i][j-1] == '1' {
                                        union(i*m+j, i*m+j-1)
                                }
                                if j < m-1 && grid[i][j+1] == '1' {
                                        union(i*m+j, i*m+j+1)
                                }
                        }
                }
        }
        return getCount()
}

var fa []int
var count int
```

(续下页)

(接上页)

```
// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        count = 0
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
        }
}

func query(i, j int) bool {
        return find(i) == find(j)
}

func getCount() int {
        return count
}
```

CHAPTER 6

0101-0200-Hard

6.1 115. 不同的子序列 (2)

- 题目

```
给定一个字符串 S 和一个字符串 T，计算在 S 的子序列中 T 出现的个数。
一个字符串的一个子序列是指，通过删除一些（也可以不删除）字符且不干扰剩余字符相对位置所组成的新字符串
（例如，"ACE" 是 "ABCDE" 的一个子序列，而 "AEC" 不是）
题目数据保证答案符合 32 位带符号整数范围。
示例 1: 输入: S = "rabbbit", T = "rabbit" 输出: 3
解释: 如下图所示，有 3 种可以从 S 中得到 "rabbit" 的方案。
(上箭头符号 ^ 表示选取的字母)
rabbbit
^^^^ ^^
rabbbit
^^ ^^^^
rabbbit
^^^ ^^^
示例 2: 输入: S = "babgbag", T = "bag" 输出: 5
解释: 如下图所示，有 5 种可以从 S 中得到 "bag" 的方案。
(上箭头符号 ^ 表示选取的字母)
babgbag
^^ ^
babgbag
^^    ^
```

(续下页)

(接上页)

```
babgbag
^    ^^
babgbag
  ^  ^^
babgbag
    ^^^
```

- 解题思路

```
func numDistinct(s string, t string) int {
        dp := make([]int, len(t)+1)
        dp[0] = 1
        for i := 1; i <= len(s); i++ {
                for j := len(t); j >= 1; j-- {
                        if s[i-1] == t[j-1] {
                                dp[j] = dp[j] + dp[j-1]
                        }
                }
        }
        return dp[len(t)]
}

# 2
func numDistinct(s string, t string) int {
        // dp[i][j]为使用s的前i个字符能够最多组成多少个t的前j个字符
        dp := make([][]int, len(s)+1)
        for i := 0; i <= len(s); i++ {
                dp[i] = make([]int, len(t)+1)
        }
        for i := 0; i <= len(s); i++ {
                dp[i][0] = 1
        }
        for i := 1; i <= len(s); i++ {
                for j := 1; j <= len(t); j++ {
                        if s[i-1] == t[j-1] {
                                // s用最后一位的 +不用最后一位
                                dp[i][j] = dp[i-1][j-1] + dp[i-1][j]
                        } else {
                                dp[i][j] = dp[i-1][j]
                        }
                }
        }
        return dp[len(s)][len(t)]
}
```

6.2 123. 买卖股票的最佳时机 III(2)

- 题目

```
给定一个数组，它的第 i 个元素是一支给定的股票在第 i 天的价格。
设计一个算法来计算你所能获取的最大利润。你最多可以完成 两笔 交易。
注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
示例 1:输入：[3,3,5,0,0,3,1,4]
输出：6
解释：在第 4 天（股票价格 = 0）的时候买入，在第 6 天（股票价格 = 3）的时候卖出，
这笔交易所能获得利润 = 3-0 = 3 。
随后，在第 7 天（股票价格 = 1）的时候买入，在第 8 天 （股票价格 = 4）的时候卖出，
     这笔交易所能获得利润 = 4-1 = 3 。
示例 2:输入：[1,2,3,4,5] 输出：4
解释：在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，
这笔交易所能获得利润 = 5-1 = 4 。
     注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。
     因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。
示例 3:输入：[7,6,4,3,1]  输出：0
解释：在这个情况下，没有交易完成，所以最大利润为 0。
```

- 解题思路

```
func maxProfit(prices []int) int {
        n := len(prices)
        if n < 2 {
                return 0
        }
        maxK := 2
        // 第一维n个状态：n天
        // 第二维3个状态：0、1、2分别表示完成的交易次数
        // 第三维2个状态：0（不持有股票）、1（持有股票）
        dp := make([][3][2]int, n)
        for i := 0; i < n; i++ {
                for j := 0; j <= maxK; j++ {
                        if i == 0 {
                                if j == 0 {
                                        dp[i][j][1] = math.MinInt64
                                } else {
                                        dp[i][j][1] = -prices[i]
                                }
                        } else if j == 0 {
                                dp[i][j][0] = 0
                                dp[i][j][1] = math.MinInt64
```

(续下页)

(接上页)

```
                        } else {
                                dp[i][j][0] = max(dp[i-1][j][0], dp[i-
↪1][j][1]+prices[i])
                                dp[i][j][1] = max(dp[i-1][j][1], dp[i-1][j-1][0]-
↪prices[i])
                        }
                }
        }
        return dp[n-1][2][0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxProfit(prices []int) int {
        buy1, buy2 := math.MaxInt32, math.MaxInt32
        profit1, profit2 := 0, 0
        for i := 0; i < len(prices); i++ {
                value := prices[i]
                buy1 = min(buy1, value)
                profit1 = max(profit1, value-buy1)
                buy2 = min(buy2, value-profit1)
                profit2 = max(profit2, value-buy2)
        }
        return profit2
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}
```

6.3 124. 二叉树中的最大路径和 (2)

- 题目

```
给定一个非空二叉树，返回其最大路径和。
本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径至少包含一个节点，且不一定经过
示例 1:输入: [1,2,3]
       1
      / \
     2   3
输出: 6
示例 2:输入: [-10,9,20,null,null,15,7]
   -10
   / \
  9  20
    /  \
   15   7
输出: 42
```

- 解题思路

```
var res int

func maxPathSum(root *TreeNode) int {
        res = math.MinInt32
        dfs(root)
        return res
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := max(dfs(root.Left), 0)
        right := max(dfs(root.Right), 0)
        // 该顶点路径和=root.Val+2边和
        value := left + right + root.Val
        res = max(res, value)
        // 单分支
```

(续下页)

(接上页)

```
        return root.Val + max(left, right)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var res int

func maxPathSum(root *TreeNode) int {
        res = math.MinInt32
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        stack := make([]*TreeNode, 0)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
                stack = append(stack, node)
        }
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                res = max(res, node.Val)
                var left, right int
                if node.Left == nil {
                        left = 0
                } else {
                        left = max(node.Left.Val, 0)
                }
                if node.Right == nil {
                        right = 0
                } else {
                        right = max(node.Right.Val, 0)
```

(续下页)

(接上页)

```
                }
                sum := node.Val + left + right
                res = max(res, sum)
                node.Val = node.Val + max(left, right)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

6.4 126. 单词接龙 II

6.4.1 题目

```
给定两个单词 (beginWord 和 endWord) 和一个字典 wordList,
找出所有从 beginWord 到 endWord 的最短转换序列。转换需遵循如下规则:
    每次转换只能改变一个字母。
    转换后得到的单词必须是字典中的单词。
说明:如果不存在这样的转换序列, 返回一个空列表。
    所有单词具有相同的长度。
    所有单词只由小写字母组成。
    字典中不存在重复的单词。
    你可以假设 beginWord 和 endWord 是非空的, 且二者不相同。
示例 1:输入:beginWord = "hit", endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
输出:[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]
示例 2:输入:beginWord = "hit" endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
输出: []
解释: endWord "cog" 不在字典中, 所以不存在符合要求的转换序列。
```

6.4.2 解题思路

6.5 128. 最长连续序列 (4)

- 题目

```
给定一个未排序的整数数组，找出最长连续序列的长度。
要求算法的时间复杂度为 O(n)。
示例:输入: [100, 4, 200, 1, 3, 2] 输出: 4
解释: 最长连续序列是 [1, 2, 3, 4]。它的长度为 4。
```

- 解题思路

```
func longestConsecutive(nums []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = true
        }
        res := 0
        for i := 0; i < len(nums); i++ {
                if _, ok := m[nums[i]-1]; !ok {
                        cur := nums[i]
                        count := 1
                        for m[cur+1] == true {
                                count = count + 1
                                cur = cur + 1
                        }
                        res = max(res, count)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
func longestConsecutive(nums []int) int {
        if len(nums) <= 1 {
                return len(nums)
        }
        sort.Ints(nums)
        res := 1
        count := 1
        for i := 1; i < len(nums); i++ {
                if nums[i] == nums[i-1] {
                        continue
                } else if nums[i] == nums[i-1]+1 {
                        count++
                } else {
                        res = max(res, count)
                        count = 1
                }
        }
        res = max(res, count)
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func longestConsecutive(nums []int) int {
        m := make(map[int]int)
        res := 0
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] > 0 {
                        continue
                }
                left := m[nums[i]-1]
                right := m[nums[i]+1]
                sum := left + 1 + right
                res = max(res, sum)
                m[nums[i]] = sum
                m[nums[i]-left] = sum
                m[nums[i]+right] = sum
```

(续下页)

(接上页)

```
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func longestConsecutive(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        m := make(map[int]int)
        res := 1
        fa = Init(nums)
        for i := 0; i < len(nums); i++ {
                union(nums[i], nums[i]+1)
                m[nums[i]]++
        }
        for i := 0; i < len(nums); i++ {
                res = max(res, find(nums[i])-nums[i]+1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

var fa map[int]int

// 初始化
func Init(data []int) map[int]int {
        n := len(data)
        arr := make(map[int]int)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                arr[data[i]] = data[i]
        }
        return arr
}

// 查询
func find(x int) int {
        if _, ok := fa[x]; !ok {
                return math.MinInt32 // 特殊处理
        }
        res := x
        for res != fa[res] {
                res = fa[res]
        }
        return res
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x == y {
                return
        } else if x == math.MinInt32 || y == math.MinInt32 {
                return
        }
        fa[x] = y
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

6.6 132. 分割回文串 II(2)

- 题目

```
给定一个字符串 s，将 s 分割成一些子串，使每个子串都是回文串。
返回符合要求的最少分割次数。
示例:输入: "aab"输出: 1
解释: 进行一次分割就可将 s 分割成 ["aa","b"] 这样两个回文子串。
```

- 解题思路

```
func minCut(s string) int {
        if len(s) == 0 || len(s) == 1 {
                return 0
        }
        dp := make([]int, len(s)+1)
        dp[0] = -1
        dp[1] = 1
        for i := 1; i <= len(s); i++ {
                dp[i] = i - 1 // 长度N切分n-1次
                for j := 0; j < i; j++ {
                        if judge(s[j:i]) {
                                dp[i] = min(dp[i], dp[j]+1)
                        }
                }
        }
        return dp[len(s)]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func judge(s string) bool {
        for i := 0; i < len(s)/2; i++ {
                if s[i] != s[len(s)-1-i] {
                        return false
                }
        }
        return true
}

# 2
func minCut(s string) int {
        if len(s) == 0 || len(s) == 1 {
                return 0
        }
        dp := make([]int, len(s)+1)
        dp[0] = -1
        dp[1] = 1
        arr := getDP(s)
        for i := 1; i <= len(s); i++ {
```

(续下页)

(接上页)

```
                dp[i] = i - 1 // 长度N切分n-1次
                for j := 0; j < i; j++ {
                        if arr[j][i-1] == true {
                                dp[i] = min(dp[i], dp[j]+1)
                        }
                }
        }
        return dp[len(s)]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func getDP(s string) [][]bool {
        dp := make([][]bool, len(s))
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
                dp[r][r] = true
                for l := 0; l < r; l++ {
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
                        }
                }
        }
        return dp
}
```

6.7 135. 分发糖果 (2)

- 题目

```
老师想给孩子们分发糖果，有 N␣
↪个孩子站成了一条直线，老师会根据每个孩子的表现，预先给他们评分。
你需要按照以下要求，帮助老师给这些孩子分发糖果：
    每个孩子至少分配到 1 个糖果。
    相邻的孩子中，评分高的孩子必须获得更多的糖果。
```

(续下页)

(接上页)

```
那么这样下来，老师至少需要准备多少颗糖果呢？
示例 1:输入：[1,0,2] 输出：5
解释：你可以分别给这三个孩子分发 2、1、2 颗糖果。
示例 2:输入：[1,2,2] 输出：4
解释：你可以分别给这三个孩子分发 1、2、1 颗糖果。
     第三个孩子只得到 1 颗糖果，这已满足上述两个条件。
```

- 解题思路

```
func candy(ratings []int) int {
        arr := make([]int, len(ratings))
        // 规则1：每个孩子至少分配到 1 个糖果。
        for i := 0; i < len(arr); i++ {
                arr[i] = 1
        }
        for i := 1; i < len(ratings); i++ {
                if ratings[i] > ratings[i-1] && arr[i] <= arr[i-1] {
                        arr[i] = arr[i-1] + 1
                }
        }
        for i := len(ratings) - 2; i >= 0; i-- {
                if ratings[i] > ratings[i+1] && arr[i] <= arr[i+1] {
                        arr[i] = arr[i+1] + 1
                }
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                res = res + arr[i]
        }
        return res
}

# 2
func candy(ratings []int) int {
        n := len(ratings)
        left := make([]int, n)
        right := make([]int, n)
        // 规则1：每个孩子至少分配到 1 个糖果。
        for i := 0; i < n; i++ {
                left[i] = 1
                right[i] = 1
        }
        for i := 1; i < n; i++ {
                if ratings[i] > ratings[i-1] {
```

(续下页)

(接上页)

```
                        left[i] = left[i-1] + 1
                }
        }
        for i := n - 2; i >= 0; i-- {
                if ratings[i] > ratings[i+1] {
                        right[i] = right[i+1] + 1
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                res = res + max(left[i], right[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

6.8 140. 单词拆分 II(2)

- 题目

```
给定一个非空字符串 s 和一个包含非空单词列表的字典␣
↪wordDict，在字符串中增加空格来构建一个句子，
使得句子中所有的单词都在词典中。返回所有这些可能的句子。
说明：
    分隔时可以重复使用字典中的单词。
    你可以假设字典中没有重复的单词。
示例 1：输入:s = "catsanddog" wordDict = ["cat", "cats", "and", "sand", "dog"]
输出:[
  "cats and dog",
  "cat sand dog"
]
示例 2：输入:s = "pineapplepenapple"
wordDict = ["apple", "pen", "applepen", "pine", "pineapple"]
输出:[
  "pine apple pen apple",
  "pineapple pen apple",
```

(续下页)

(接上页)

```
  "pine applepen apple"
]
解释: 注意你可以重复使用字典中的单词。
示例 3: 输入:s = "catsandog" wordDict = ["cats", "dog", "sand", "and", "cat"] 输出:[]
```

- 解题思路

```
var res []string
var m map[string]bool

func wordBreak(s string, wordDict []string) []string {
        m = make(map[string]bool)
        for _, word := range wordDict {
                m[word] = true
        }
        dp := make([]bool, len(s)+1)
        dp[0] = true
        for i := 1; i <= len(s); i++ {
                for j := 0; j < i; j++ {
                        if dp[j] == true && m[s[j:i]] == true {
                                dp[i] = true
                                break
                        }
                }
        }
        if dp[len(s)] == false {
                return nil
        }
        res = make([]string, 0)
        dfs(s, make([]string, 0))
        return res
}

func dfs(str string, arr []string) {
        if len(str) == 0 {
                res = append(res, strings.Join(arr, " "))
                return
        }
        for i := 1; i <= len(str); i++ {
                if m[str[:i]] == true {
                        dfs(str[i:], append(arr, str[:i]))
                }
        }
}
```

(续下页)

(接上页)

```
# 2
var m map[string]bool
var visited map[int][]string

func wordBreak(s string, wordDict []string) []string {
        m = make(map[string]bool)
        visited = make(map[int][]string)
        for _, str := range wordDict {
                m[str] = true
        }
        return dfs(s, 0)
}

func dfs(s string, level int) []string {
        if str, ok := visited[level]; ok {
                return str
        }
        res := make([]string, 0)
        for i := level + 1; i <= len(s); i++ {
                if m[s[level:i]] {
                        if i != len(s) {
                                arr := dfs(s, i)
                                for _, str := range arr {
                                        res = append(res, s[level:i]+" "+str)
                                }
                        } else {
                                res = append(res, s[level:i])
                        }
                }
        }
        visited[level] = res
        return res
}
```

6.9 145. 二叉树的后序遍历 (4)

- 题目

```
给定一个二叉树，返回它的 后序 遍历。
示例:输入: [1,null,2,3]
   1
    \
     2
    /
   3
输出: [3,2,1] 进阶: 递归算法很简单，你可以通过迭代算法完成吗？
```

- 解题思路

```
var res []int

func postorderTraversal(root *TreeNode) []int {
        res = make([]int, 0)
        dfs(root)
        return res
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                dfs(root.Right)
                res = append(res, root.Val)
        }
}

# 2
func postorderTraversal(root *TreeNode) []int {
        if root == nil {
                return nil
        }
        res := make([]int, 0)
        if root.Left != nil {
                res = append(res, postorderTraversal(root.Left)...)
        }
        if root.Right != nil {
                res = append(res, postorderTraversal(root.Right)...)
        }
        res = append(res, root.Val)
```

(续下页)

(接上页)

```
        return res
}

# 3
func postorderTraversal(root *TreeNode) []int {
        if root == nil {
                return nil
        }
        res := make([]int, 0)
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                last := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if last != nil {
                        stack = append(stack, last)
                        stack = append(stack, nil)
                        if last.Right != nil {
                                stack = append(stack, last.Right)
                        }
                        if last.Left != nil {
                                stack = append(stack, last.Left)
                        }
                } else {
                        node := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        res = append(res, node.Val)
                }
        }
        return res
}

# 4
func postorderTraversal(root *TreeNode) []int {
        if root == nil {
                return nil
        }
        res := make([]int, 0)
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        // 根->右->左
        for len(stack) != 0 {
                node := stack[len(stack)-1]
```

(续下页)

(接上页)

```
                stack = stack[:len(stack)-1]
                if node.Left != nil {
                        stack = append(stack, node.Left)
                }
                if node.Right != nil {
                        stack = append(stack, node.Right)
                }
                res = append(res, node.Val)
        }

        for i := 0; i < len(res)/2; i++ {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
        }
        return res
}
```

6.10 149. 直线上最多的点数 (2)

- 题目

```
给定一个二维平面，平面上有 n 个点，求最多有多少个点在同一条直线上。
示例 1:输入: [[1,1],[2,2],[3,3]] 输出: 3
解释:
^
|
|        o
|     o
|  o
+------------->
0  1  2  3  4
示例 2:输入: [[1,1],[3,2],[5,3],[4,1],[2,3],[1,4]] 输出: 4
解释:
^
|
|  o
|     o        o
|        o
|  o        o
+-------------------->
0  1  2  3  4  5  6
```

- 解题思路

```
func maxPoints(points [][]int) int {
        n := len(points)
        if n < 3 {
                return n
        }
        res := 2
        m := make(map[[3]int]map[int]bool)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        // AX+BY+C=0
                        A := points[j][1] - points[i][1]
                        B := points[i][0] - points[j][0]
                        C := points[i][1]*points[j][0] - points[i][0]*points[j][1]
                        com := gcd(gcd(A, B), C)
                        A, B, C = A/com, B/com, C/com
                        node := [3]int{A, B, C}
                        if m[node] == nil {
                                m[node] = make(map[int]bool)
                        }
                        m[node][i] = true
                        m[node][j] = true
                        if len(m[node]) > res {
                                res = len(m[node])
                        }
                }
        }
        return res
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}

# 2
func maxPoints(points [][]int) int {
        res := 0
        n := len(points)
        if n < 3 {
                return n
        }
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := i + 1; j < n; j++ {
                        count := 2
                        x1 := points[i][0] - points[j][0]
                        y1 := points[i][1] - points[j][1]
                        for k := j + 1; k < n; k++ {
                                x2 := points[i][0] - points[k][0]
                                y2 := points[i][1] - points[k][1]
                                if x1*y2 == x2*y1 { // 斜率相同+1
                                        count++
                                }
                        }
                        if count > res {
                                res = count
                        }
                }
        }
        return res
}
```

6.11 154. 寻找旋转排序数组中的最小值 II(4)

- 题目

```
假设按照升序排序的数组在预先未知的某个点上进行了旋转。
( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。
请找出其中最小的元素。
注意数组中可能存在重复的元素。
示例 1: 输入: [1,3,5] 输出: 1
示例 2: 输入: [2,2,2,0,1]输出: 0
说明: 这道题是 寻找旋转排序数组中的最小值 的延伸题目。
    允许重复会影响算法的时间复杂度吗？会如何影响，为什么？
```

- 解题思路

```
func findMin(nums []int) int {
        left := 0
        right := len(nums) - 1
        for left < right {
                mid := left + (right-left)/2
                if nums[mid] > nums[right] {
                        left = mid + 1
                } else if nums[mid] < nums[right] {
```

(续下页)

(接上页)

```
                        right = mid
                } else {
                        right--
                }
        }
        return nums[left]
}

# 2
func findMin(nums []int) int {
        sort.Ints(nums)
        return nums[0]
}

# 3
func findMin(nums []int) int {
        for i := 1; i < len(nums); i++ {
                if nums[i] < nums[i-1] {
                        return nums[i]
                }
        }
        return nums[0]
}

# 4
func findMin(nums []int) int {
        left := 0
        right := len(nums) - 1
        mid := left
        for nums[left] >= nums[right] {
                if right-left == 1 {
                        mid = right
                        break
                }
                mid = (left + right) / 2
                if nums[left] == nums[right] && nums[mid] == nums[left] {
                        return minInorder(nums, left, right)
                }
                if nums[mid] >= nums[left] {
                        left = mid
                } else if nums[mid] <= nums[right] {
                        right = mid
                }
```

(续下页)

(接上页)

```
        }
        return nums[mid]
}

func minInorder(numbers []int, left, right int) int {
        result := numbers[left]
        for i := left + 1; i <= right; i++ {
                if result > numbers[i] {
                        result = numbers[i]
                }
        }
        return result
}
```

6.12 164. 最大间距 (2)

- 题目

```
给定一个无序的数组，找出数组在排序之后，相邻元素之间最大的差值。
如果数组元素个数小于 2，则返回 0。
示例 1:输入: [3,6,9,1] 输出: 3
解释: 排序后的数组是 [1,3,6,9]，其中相邻元素 (3,6) 和 (6,9) 之间都存在最大差值 3。
示例 2:输入: [10] 输出: 0
解释: 数组元素个数小于 2，因此返回 0。
说明:你可以假设数组中所有元素都是非负整数，且数值在 32 位有符号整数范围内。
    请尝试在线性时间复杂度和空间复杂度的条件下解决此问题。
```

- 解题思路

```
func maximumGap(nums []int) int {
        res := 0
        sort.Ints(nums)
        for i := 1; i < len(nums); i++ {
                if nums[i]-nums[i-1] > res {
                        res = nums[i] - nums[i-1]
                }
        }
        return res
}

# 2
func maximumGap(nums []int) int {
```

(续下页)

(接上页)

```
        if len(nums) <= 1 {
                return 0
        }
        res := 0
        minValue, maxValue := nums[0], nums[0]
        for i := 1; i < len(nums); i++ {
                minValue = min(minValue, nums[i])
                maxValue = max(maxValue, nums[i])
        }
        bucketSize := (maxValue-minValue)/len(nums) + 1
        bucketNum := (maxValue-minValue)/bucketSize + 1
        arr := make([][]int, bucketNum)
        for i := 0; i < len(nums); i++ {
                index := (nums[i] - minValue) / bucketSize
                if len(arr[index]) == 0 {
                        arr[index] = make([]int, 2)
                        arr[index][0], arr[index][1] = nums[i], nums[i]
                } else {
                        arr[index][0] = min(arr[index][0], nums[i])
                        arr[index][1] = max(arr[index][1], nums[i])
                }
        }
        prev := 0
        for i := 0; i < bucketNum; i++ {
                if len(arr[i]) == 0 {
                        continue
                }
                res = max(res, arr[i][0]-arr[prev][1])
                prev = i
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}
```

6.13 174. 地下城游戏 (3)

- 题目

```
一些恶魔抓住了公主（P）并将她关在了地下城的右下角。地下城是由 M x N␣
↪个房间组成的二维网格。
我们英勇的骑士（K）最初被安置在左上角的房间里，他必须穿过地下城并通过对抗恶魔来拯救公主。
骑士的初始健康点数为一个正整数。如果他的健康点数在某一时刻降至 0␣
↪或以下，他会立即死亡。
有些房间由恶魔守卫，因此骑士在进入这些房间时会失去健康点数
（若房间里的值为负整数，则表示骑士将损失健康点数）；
其他房间要么是空的（房间里的值为 0），
要么包含增加骑士健康点数的魔法球（若房间里的值为正整数，则表示骑士将增加健康点数）。
为了尽快到达公主，骑士决定每次只向右或向下移动一步。
编写一个函数来计算确保骑士能够拯救到公主所需的最低初始健康点数。
例如，考虑到如下布局的地下城，如果骑士遵循最佳路径 右 -> 右 -> 下 -> 下，
则骑士的初始健康点数至少为 7。
-2 (K)          -3          3
-5          -10          1
10          30          -5 (P)
说明:骑士的健康点数没有上限。
    任何房间都可能对骑士的健康点数造成威胁，也可能增加骑士的健康点数，
    包括骑士进入的左上角房间以及公主被监禁的右下角房间。
```

- 解题思路

```
func calculateMinimumHP(dungeon [][]int) int {
        n, m := len(dungeon), len(dungeon[0])
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = math.MaxInt32
                }
        }
        dp[n][m-1], dp[n-1][m] = 1, 1 // 结果最小为1
        for i := n - 1; i >= 0; i-- {
                for j := m - 1; j >= 0; j-- {
```

(续下页)

(接上页)

```
                        minValue := min(dp[i+1][j], dp[i][j+1])
                        dp[i][j] = max(minValue-dungeon[i][j], 1)
                }
        }
        return dp[0][0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var dp [][]int

func calculateMinimumHP(dungeon [][]int) int {
        n, m := len(dungeon), len(dungeon[0])
        dp = make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m+1)
        }
        return dfs(dungeon, n, m, 0, 0)
}

func dfs(dungeon [][]int, n, m, i, j int) int {
        if i == n-1 && j == m-1 {
                return max(1-dungeon[i][j], 1)
        }
        if dp[i][j] > 0 {
                return dp[i][j]
        }
        res := 0
        if i == n-1 {
                res = max(dfs(dungeon, n, m, i, j+1)-dungeon[i][j], 1)
```

(续下页)

(接上页)

```
        } else if j == m-1 {
                res = max(dfs(dungeon, n, m, i+1, j)-dungeon[i][j], 1)
        } else {
                minValue := min(dfs(dungeon, n, m, i, j+1), dfs(dungeon, n, m, i+1,␣
↪j))
                res = max(minValue-dungeon[i][j], 1)
        }
        dp[i][j] = res
        return dp[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func calculateMinimumHP(dungeon [][]int) int {
        if len(dungeon) == 0 {
                return 0
        }
        left, right := 1, math.MaxInt32
        for left <= right {
                mid := left + (right-left)/2
                if judge(dungeon, mid) == true {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return left
}

func judge(dungeon [][]int, hp int) bool {
```

(续下页)

(接上页)

```
        n, m := len(dungeon), len(dungeon[0])
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = math.MinInt32
                }
        }
        dp[0][1], dp[1][0] = hp, hp
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        value := max(dp[i-1][j], dp[i][j-1]) + dungeon[i-1][j-1]
                        if value <= 0 {
                                continue
                        }
                        dp[i][j] = value
                }
        }
        return dp[n][m] > 0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

6.14 188. 买卖股票的最佳时机 IV(3)

- 题目

```
给定一个数组，它的第 i 个元素是一支给定的股票在第 i 天的价格。
设计一个算法来计算你所能获取的最大利润。你最多可以完成 k 笔交易。
注意: 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
示例 1:输入: [2,4,1], k = 2 输出: 2
解释: 在第 1 天 (股票价格 = 2) 的时候买入，在第 2 天 (股票价格 = 4) 的时候卖出，
这笔交易所能获得利润 = 4-2 = 2 。
示例 2:输入: [3,2,6,5,0,3], k = 2 输出: 7
解释: 在第 2 天 (股票价格 = 2) 的时候买入，在第 3 天 (股票价格 = 6) 的时候卖出，
这笔交易所能获得利润 = 6-2 = 4 。
     随后，在第 5 天 (股票价格 = 0) 的时候买入，在第 6 天 (股票价格 = 3) 的时候卖出，
```

(续下页)

(接上页)

```
    这笔交易所能获得利润 = 3-0 = 3 。
```

- 解题思路

```
func maxProfit(k int, prices []int) int {
        res := 0
        if k >= len(prices)/2 {
                for i := 0; i < len(prices)-1; i++ {
                        if prices[i] < prices[i+1] {
                                res = res + prices[i+1] - prices[i]
                        }
                }
                return res
        }
        dp0, dp1 := make([]int, k+1), make([]int, k+1)
        for i := 0; i <= k; i++ {
                dp0[i] = 0
                dp1[i] = math.MinInt64
        }
        for i := 0; i < len(prices); i++ {
                for j := k; j >= 1; j-- {
                        dp0[j] = max(dp0[j], dp1[j]+prices[i])
                        dp1[j] = max(dp1[j], dp0[j-1]-prices[i])
                }
        }
        return dp0[k]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxProfit(k int, prices []int) int {
        res := 0
        if k >= len(prices)/2 {
                for i := 0; i < len(prices)-1; i++ {
                        if prices[i] < prices[i+1] {
                                res = res + prices[i+1] - prices[i]
                        }
                }
```

(续下页)

(接上页)

```
                return res
        }
        dp0, dp1 := make([][]int, len(prices)), make([][]int, len(prices))
        for i := 0; i < len(prices); i++ {
                dp0[i] = make([]int, k+1)
                dp1[i] = make([]int, k+1)
        }
        for i := 0; i <= k; i++ {
                dp1[0][i] = -prices[0]
        }
        for i := 1; i < len(prices); i++ {
                for j := 1; j <= k; j++ {
                        dp0[i][j] = max(dp0[i-1][j], dp1[i-1][j]+prices[i])
                        dp1[i][j] = max(dp1[i-1][j], dp0[i-1][j-1]-prices[i])
                }
        }
        return dp0[len(prices)-1][k]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxProfit(k int, prices []int) int {
        res := 0
        if k >= len(prices)/2 {
                for i := 0; i < len(prices)-1; i++ {
                        if prices[i] < prices[i+1] {
                                res = res + prices[i+1] - prices[i]
                        }
                }
                return res
        }
        dp := make([][][2]int, len(prices))
        for i := 0; i < len(prices); i++ {
                dp[i] = make([][2]int, k+1)
        }
        for i := 0; i <= k; i++ {
                dp[0][i][1] = -prices[0]
```

(续下页)

(接上页)

```
        }
        for i := 1; i < len(prices); i++ {
                for j := 1; j <= k; j++ {
                        dp[i][j][0] = max(dp[i-1][j][0], dp[i-1][j][1]+prices[i])
                        dp[i][j][1] = max(dp[i-1][j][1], dp[i-1][j-1][0]-prices[i])
                }
        }
        return dp[len(prices)-1][k][0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

CHAPTER 7

0201-0300-Easy

7.1 202. 快乐数 (2)

- 题目

```
编写一个算法来判断一个数 n 是不是快乐数。
「快乐数」定义为：对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和，
然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。
如果 可以变为  1，那么这个数就是快乐数。
如果 n 是快乐数就返回 True ；不是，则返回 False 。
示例：  输入：19 输出：true
解释：
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1
```

- 解题思路

```
func isHappy(n int) bool {
        now, next := n, nextValue(n)
        m := make(map[int]int)
        m[now] = 1
        for {
                if next == 1 {
```

(续下页)

(接上页)

```
                        break
                }
                if _, ok := m[next]; ok {
                        break
                } else {
                        m[next] = 1
                }
                next = nextValue(next)
        }

        if next == 1 {
                return true
        }

        return false
}

func nextValue(n int) int {
        ret := 0
        for n != 0 {
                ret = ret + (n%10)*(n%10)
                n = n / 10
        }
        return ret
}

#
func isHappy(n int) bool {
        now, next := n, nextValue(n)
        for now != next {
                now = nextValue(now)
                next = nextValue(nextValue(next))
        }
        if now == 1 {
                return true
        }
        return false
}

func nextValue(n int) int {
        ret := 0
        for n != 0 {
                ret = ret + (n%10)*(n%10)
```

(续下页)

(接上页)

```
                n = n / 10
        }
        return ret
}
```

7.2 203. 移除链表元素 (2)

- 题目

```
删除链表中等于给定值 val 的所有节点。
示例:
输入: 1->2->6->3->4->5->6, val = 6
输出: 1->2->3->4->5
```

- 解题思路

```
func removeElements(head *ListNode, val int) *ListNode {
        headPre := &ListNode{Next: head}
        temp := headPre

        for temp.Next != nil {
                if temp.Next.Val == val {
                        //delete
                        temp.Next = temp.Next.Next
                } else {
                        temp = temp.Next
                }
        }

        return headPre.Next
}

# 递归
func removeElements(head *ListNode, val int) *ListNode {
        if head == nil {
                return nil
        }
        head.Next = removeElements(head.Next, val)
        if head.Val == val {
                return head.Next
        }
        return head
```

(续下页)

(接上页)

```
}
```

7.3 204. 计数质数 (2)

- 题目

```
统计所有小于非负整数 n 的质数的数量。
示例:
输入: 10
输出: 4
解释: 小于 10 的质数一共有 4 个, 它们是 2, 3, 5, 7 。
```

- 解题思路

```
func countPrimes(n int) int {
        if n < 3 {
                return 0
        }
        notPrimes := make([]bool, n)
        count := 0
        for i := 2; i < n; i++ {
                if notPrimes[i] {
                        continue
                }
                for j := i*2 ; j < n; j += i  {
                        notPrimes[j] = true
                }
                count++
        }
        return count
}

#
func countPrimes(n int) int {
        if n < 3 {
                return 0
        }
        isPrimes := make([]bool, n)
        for i := range isPrimes {
                isPrimes[i] = true
        }
        for i := 2; i*i < n; i++ {
```

(续下页)

(接上页)

```
                if !isPrimes[i] {
                        continue
                }
                for j := i * i; j < n; j += i {
                        isPrimes[j] = false
                }
        }
        count := 0
        for i := 2; i < n; i++ {
                if isPrimes[i] {
                        count++
                }
        }
        return count
}
```

7.4 205. 同构字符串 (3)

- 题目

```
给定两个字符串 s 和 t，判断它们是否是同构的。
如果 s 中的字符可以被替换得到 t ，那么这两个字符串是同构的。
所有出现的字符都必须用另一个字符替换，同时保留字符的顺序。
两个字符不能映射到同一个字符上，但字符可以映射自己本身。

示例 1: 输入: s = "egg", t = "add" 输出: true
示例 2:输入: s = "foo", t = "bar" 输出: false
示例 3: 输入: s = "paper", t = "title" 输出: true

说明:你可以假设 s 和 t 具有相同的长度。
```

- 解题思路

```
func isIsomorphic(s string, t string) bool {
        if len(s) != len(t) {
                return false
        }

        m1 := make([]int, 256)
        m2 := make([]int, 256)

        for i := 0; i < len(s); i++ {
```

(续下页)

(接上页)

```
                a := int(s[i])
                b := int(t[i])
                if m1[a] != m2[b] {
                        return false
                }
                m1[a] = i + 1
                m2[b] = i + 1
        }
        return true
}

// 2
func isIsomorphic(s string, t string) bool {
        if len(s) != len(t) {
                return false
        }

        m := make(map[int]int)
        n := make(map[int]int)

        for i := 0; i < len(s); i++ {
                a := int(s[i])
                b := int(t[i])
                if m[a] == 0 && n[b] == 0 {
                        m[a] = b
                        n[b] = a
                } else if m[a] != b || n[b] != a {
                        return false
                }
        }
        return true
}


// 3
func isIsomorphic(s string, t string) bool {
        for i := 0; i < len(s); i++ {
                if strings.IndexByte(s[i+1:], s[i]) != strings.IndexByte(t[i+1:],␣
↪t[i]) {
                        return false
                }
        }
        return true
}
```

7.5 206. 反转链表 (4)

- 题目

```
反转一个单链表。
示例:
输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL
进阶:
你可以迭代或递归地反转链表。你能否用两种方法解决这道题?
```

- 解题思路

```
func reverseList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }

        result := reverseList(head.Next)
        head.Next.Next = head
        head.Next = nil
        return result
}

// 2
func reverseList(head *ListNode) *ListNode {
        var result *ListNode
        var temp *ListNode
        for head != nil {
                temp = head.Next
                head.Next = result
                result = head
                head = temp
        }
        return result
}

// 3
func reverseList(head *ListNode) *ListNode {
        result := &ListNode{}
        arr := make([]*ListNode, 0)
        for head != nil {
                arr = append(arr, head)
                head = head.Next
```

(续下页)

(接上页)

```
        }
        temp := result
        for i := len(arr) - 1; i >= 0; i-- {
                arr[i].Next = nil
                temp.Next = arr[i]
                temp = temp.Next
        }
        return result.Next
}

// 4
func reverseList(head *ListNode) *ListNode {
        var res *ListNode
        for {
                if head == nil {
                        break
                }
                res = &ListNode{head.Val, res}
                head = head.Next
        }
        return res
}
```

7.6 217. 存在重复元素 (2)

- 题目

```
给定一个整数数组，判断是否存在重复元素。
如果任意一值在数组中出现至少两次，函数返回 true 。如果数组中每个元素都不相同，则返回␣
↪false 。

示例 1:输入: [1,2,3,1] 输出: true
示例 2:输入: [1,2,3,4] 输出: false
示例 3:输入: [1,1,1,3,3,4,3,2,4,2] 输出: true
```

- 解题思路

```
func containsDuplicate(nums []int) bool {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                if _, ok := m[nums[i]]; ok {
                        return true
```

(续下页)

(接上页)

```
                } else {
                        m[nums[i]] = 1
                }
        }
        return false
}

#
func containsDuplicate(nums []int) bool {
        sort.Ints(nums)
        for i := 0; i < len(nums)-1; i++{
                if nums[i] == nums[i+1]{
                        return true
                }
        }
        return false
}
```

7.7 219. 存在重复元素 II(2)

- 题目

```
给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，
使得 nums [i] = nums [j]，并且 i 和 j 的差的 绝对值 至多为 k。

示例 1:输入: nums = [1,2,3,1], k = 3输出: true
示例 2:输入: nums = [1,0,1,1], k = 1 输出: true
示例 3:输入: nums = [1,2,3,1,2,3], k = 2 输出: false
```

- 解题思路

```
#
func containsNearbyDuplicate(nums []int, k int) bool {
        m := make(map[int]int)

        for i, n := range nums {
                if m[n] != 0 && (i+1)-m[n] <= k {
                        return true
                }
                m[n] = i + 1
        }
        return false
```

(续下页)

(接上页)

```
}

#
func containsNearbyDuplicate(nums []int, k int) bool {
        m := make(map[int]int)

        for i, n := range nums {
                if m[n] != 0 {
                        return true
                }
                m[n] = i + 1
                if len(m) > k {
                        delete(m, nums[i-k])
                }
        }
        return false
}
```

7.8 225. 用队列实现栈 (4)

- 题目

```
使用队列实现栈的下列操作:
    push(x) -- 元素 x 入栈
    pop() -- 移除栈顶元素
    top() -- 获取栈顶元素
    empty() -- 返回栈是否为空

注意:
    你只能使用队列的基本操作-- 也就是 push to back, peek/pop from front, size,
    和 is empty 这些操作是合法的。
    你所使用的语言也许不支持队列。 你可以使用 list 或者␣
↪deque (双端队列) 来模拟一个队列 ,
    只要是标准的队列操作即可。
    你可以假设所有操作都是有效的 (例如, 对一个空的栈不会调用 pop 或者 top 操作) 。
```

- 解题思路

```
type MyStack struct {
        arr []int
}
```

(续下页)

(接上页)

```
func Constructor() MyStack {
        return MyStack{}
}

func (m *MyStack) Push(x int) {
        m.arr = append(m.arr, x)
}

func (m *MyStack) Pop() int {
        if len(m.arr) == 0 {
                return 0
        }
        last := m.arr[len(m.arr)-1]
        m.arr = m.arr[0 : len(m.arr)-1]
        return last
}

func (m *MyStack) Top() int {
        if len(m.arr) == 0 {
                return 0
        }
        return m.arr[len(m.arr)-1]
}

func (m *MyStack) Empty() bool {
        if len(m.arr) == 0 {
                return true
        }
        return false
}

# 使用1个list实现
type MyStack struct {
        *list.List
}
func Constructor() MyStack {
        return MyStack{
                list.New(),
        }
}

func (m *MyStack) Push(x int) {
        m.PushBack(x)
```

(续下页)

(接上页)

```
}

func (m *MyStack) Pop() int {
        if m.Len() == 0 {
                return -1
        }
        return m.Remove(m.Back()).(int)
}

func (m *MyStack) Top() int {
        if m.Len() == 0 {
                return -1
        }
        return m.Back().Value.(int)
}

func (m *MyStack) Empty() bool {
        return m.Len() == 0
}

#  使用2个list实现
/*
入栈过程:
1、q1 为空，放入 q2，否则放入 q1
出栈过程:
1、q1为空：依次取出q2中的元素（除了最后一个），并且放入q1中␣
↪取出q2中的最后一个元素，返回结果
否则 依次取出q1中的元素（除了最后一个），并且放入q2中 取出q1中的最后一个元素，返回结果
*/
type MyStack struct {
        l1 *list.List
        l2 *list.List
}

func Constructor() MyStack {
        return MyStack{
                l1: list.New(),
                l2: list.New(),
        }
}

func (m *MyStack) Push(x int) {
        if m.l1.Len() == 0 {
```

(续下页)

(接上页)

```
                m.l2.PushBack(x)
        } else {
                m.l1.PushBack(x)
        }
}

func (m *MyStack) Pop() int {
        var top int
        if m.l1.Len() > 0 {
                for m.l1.Len() > 1 {
                        m.l2.PushBack(m.l1.Remove(m.l1.Front()))
                }
                top = m.l1.Remove(m.l1.Front()).(int)
        } else {
                for m.l2.Len() > 1 {
                        m.l1.PushBack(m.l2.Remove(m.l2.Front()))
                }
                top = m.l2.Remove(m.l2.Front()).(int)
        }
        return top
}

func (m *MyStack) Top() int {
        var top int
        if m.l1.Len() > 0 {
                for m.l1.Len() > 1 {
                        m.l2.PushBack(m.l1.Remove(m.l1.Front()))
                }
                top = m.l1.Back().Value.(int)
                m.l2.PushBack(m.l1.Remove(m.l1.Front()))

        } else {
                for m.l2.Len() > 1 {
                        m.l1.PushBack(m.l2.Remove(m.l2.Front()))
                }
                top = m.l2.Back().Value.(int)
                m.l1.PushBack(m.l2.Remove(m.l2.Front()))
        }
        return top
}

func (m *MyStack) Empty() bool {
        return m.l1.Len() == 0 && m.l2.Len() == 0
```

(续下页)

(接上页)

```
}

# 使用2个双端队列deque实现
type MyStack struct {
        l1 *Queue
        l2 *Queue
}

func Constructor() MyStack {
        return MyStack{
                l1: NewQueue(),
                l2: NewQueue(),
        }
}

func (m *MyStack) Push(x int) {
        m.l1.Push(x)
}

func (m *MyStack) Pop() int {
        if m.l2.Len() == 0 {
                m.l1, m.l2 = m.l2, m.l1
        }

        for m.l2.Len() > 1 {
                m.l1.Push(m.l2.Pop())
        }
        return m.l2.Pop()
}

func (m *MyStack) Top() int {
        res := m.Pop()
        m.l1.Push(res)
        return res
}

func (m *MyStack) Empty() bool {
        return (m.l1.Len() + m.l2.Len()) == 0
}

type Queue struct {
        nums []int
}
```

(续下页)

(接上页)

```
func NewQueue() *Queue {
        return &Queue{
                nums: []int{},
        }
}

func (q *Queue) Push(n int) {
        q.nums = append(q.nums, n)
}

func (q *Queue) Pop() int {
        if len(q.nums) == 0 {
                return 0
        }
        res := q.nums[0]
        q.nums = q.nums[1:]
        return res
}

func (q *Queue) Len() int {
        return len(q.nums)
}

func (q *Queue) IsEmpty() bool {
        return q.Len() == 0
}
```

7.9 226. 翻转二叉树 (2)

- 题目

```
翻转一棵二叉树。

示例：

输入：

     4
   /   \
  2     7
 / \   / \
```

(续下页)

(接上页)

```
1   3 6   9

输出：

     4
   /   \
  7     2
 / \   / \
9   6 3   1

备注:
这个问题是受到 Max Howell 的 原问题 启发的 :
    谷歌: 我们90%的工程师使用您编写的软件(Homebrew),
    但是您却无法在面试时在白板上写出翻转二叉树这道题, 这太糟糕了。
```

- 解题思路

```
func invertTree(root *TreeNode) *TreeNode {
        if root == nil || (root.Left == nil && root.Right == nil) {
                return root
        }
        root.Left, root.Right = invertTree(root.Right), invertTree(root.Left)
        return root
}

#
func invertTree(root *TreeNode) *TreeNode {
        if root == nil {
                return root
        }

        var queue []*TreeNode
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                node.Left, node.Right = node.Right, node.Left
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
```

(续下页)

(接上页)

```
        return root
}
```

7.10 231.2 的幂 (3)

- 题目

```
给定一个整数，编写一个函数来判断它是否是 2 的幂次方。

示例 1:输入: 1 输出: true 解释: 2^0 = 1
示例 2:输入: 16 输出: true 解释: 2^4 = 16
示例 3:输入: 218 输出: false
```

- 解题思路

```
func isPowerOfTwo(n int) bool {
        if n < 1 {
                return false
        }
        for n > 1 {
                if n%2 == 1 {
                        return false
                }
                n = n / 2
        }
        return true
}

#
func isPowerOfTwo(n int) bool {
        if n < 1 {
                return false
        }
        return n & (n-1) == 0
}

#
func isPowerOfTwo(n int) bool {
        if n < 1 {
                return false
        }
        if n == 1{
```

(续下页)

(接上页)

```
                return true
        }
        if n % 2 != 0{
                return false
        }
        return isPowerOfTwo(n/2)
}
```

7.11 232. 用栈实现队列 (3)

- 题目

```
使用栈实现队列的下列操作：
    push(x) -- 将一个元素放入队列的尾部。
    pop() -- 从队列首部移除元素。
    peek() -- 返回队列首部的元素。
    empty() -- 返回队列是否为空。

示例:
MyQueue queue = new MyQueue();
queue.push(1);
queue.push(2);
queue.peek();  // 返回 1
queue.pop();   // 返回 1
queue.empty(); // 返回 false

说明:
你只能使用标准的栈操作 -- 也就是只有 push to top, peek/pop from top, size,
和 is empty 操作是合法的。
你所使用的语言也许不支持栈。你可以使用 list 或者␣
↪deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。
假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。
```

- 解题思路

```
type MyQueue struct {
        a []int
}

func Constructor() MyQueue {
        return MyQueue{}
}
```

(续下页)

(接上页)

```
func (m *MyQueue) Push(x int) {
        m.a = append(m.a, x)
}

func (m *MyQueue) Pop() int {
        if len(m.a) == 0 {
                return 0
        }
        first := m.a[0]
        m.a = m.a[1:]
        return first
}

func (m *MyQueue) Peek() int {
        if len(m.a) == 0 {
                return 0
        }
        return m.a[0]
}

func (m *MyQueue) Empty() bool {
        if len(m.a) == 0 {
                return true
        }
        return false
}

# 使用2个栈实现
/*
入队：直接入栈a
出队：栈b为空，则把栈a中全部数据出栈进入栈b，然后出栈b,不为空直接出栈b
*/
type MyQueue struct {
        a, b *Stack
}

func Constructor() MyQueue {
        return MyQueue{
                a: NewStack(),
                b: NewStack(),
        }
}
```

(续下页)

(接上页)

```
func (m *MyQueue) Push(x int) {
        m.a.Push(x)
}

func (m *MyQueue) Pop() int {
        if m.b.Len() == 0 {
                for m.a.Len() > 0 {
                        m.b.Push(m.a.Pop())
                }
        }
        return m.b.Pop()
}

func (m *MyQueue) Peek() int {
        res := m.Pop()
        m.b.Push(res)
        return res
}

func (m *MyQueue) Empty() bool {
        return m.a.Len() == 0 && m.b.Len() == 0
}

type Stack struct {
        nums []int
}

func NewStack() *Stack {
        return &Stack{
                nums: []int{},
        }
}

func (s *Stack) Push(n int) {
        s.nums = append(s.nums, n)
}

func (s *Stack) Pop() int {
        res := s.nums[len(s.nums)-1]
        s.nums = s.nums[:len(s.nums)-1]
        return res
}
```

(续下页)

(接上页)

```
func (s *Stack) Len() int {
        return len(s.nums)
}

func (s *Stack) IsEmpty() bool {
        return s.Len() == 0
}

# 使用2个切片实现
type MyQueue struct {
        a []int
        b []int
}

func Constructor() MyQueue {
        return MyQueue{}
}

func (m *MyQueue) Push(x int) {
        m.a = append(m.a, x)
}

func (m *MyQueue) Pop() int {
        m.Peek()
        temp := m.b[len(m.b)-1]
        m.b = m.b[:len(m.b)-1]
        return temp
}

func (m *MyQueue) Peek() int {
        if len(m.b) == 0 {
                for len(m.a) > 0 {
                        m.b = append(m.b, m.a[len(m.a)-1])
                        m.a = m.a[:len(m.a)-1]
                }
        }
        if len(m.b) == 0 {
                return -1
        }
        return m.b[len(m.b)-1]
}
```

(续下页)

(接上页)

```
func (m *MyQueue) Empty() bool {
        return len(m.a) == 0 && len(m.b) == 0
}
```

7.12 234. 回文链表 (4)

- 题目

```
请判断一个链表是否为回文链表。
示例 1:输入: 1->2 输出: false
示例 2:输入: 1->2->2->1 输出: true
```

- 解题思路

```
func isPalindrome(head *ListNode) bool {
        m := make([]int, 0)
        for head != nil {
                m = append(m, head.Val)
                head = head.Next
        }
        i, j := 0, len(m)-1
        for i < j {
                if m[i] != m[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

# 2
func isPalindrome(head *ListNode) bool {
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
        }
        var pre *ListNode
        cur := slow
        for cur != nil{
                next := cur.Next
```

(续下页)

(接上页)

```
                cur.Next = pre
                pre = cur
                cur = next
        }
        for pre != nil{
                if head.Val != pre.Val{
                        return false
                }
                pre = pre.Next
                head = head.Next
        }
        return true
}

# 3
func isPalindrome(head *ListNode) bool {
        m := make([]int, 0)
        temp := head
        for temp != nil {
                m = append(m, temp.Val)
                temp = temp.Next
        }
        for head != nil {
                val := m[len(m)-1]
                m = m[:len(m)-1]
                if head.Val != val {
                        return false
                }
                head = head.Next
        }
        return true
}

# 4
var p *ListNode
func isPalindrome(head *ListNode) bool {
        if head == nil{
                return true
        }
        if p == nil{
                p = head
        }
        if isPalindrome(head.Next) && (p.Val == head.Val){
```

(续下页)

(接上页)

```
                p = p.Next
                return true
        }
        p = nil
        return false
}
```

7.13 235. 二叉搜索树的最近公共祖先 (2)

- 题目

```
给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。
百度百科中最近公共祖先的定义为: “对于有根树 T 的两个结点␣
↪p、q, 最近公共祖先表示为一个结点 x,
满足 x 是 p、q 的祖先且 x 的深度尽可能大 (一个节点也可以是它自己的祖先) 。”
例如, 给定如下二叉搜索树: root = [6,2,8,0,4,7,9,null,null,3,5]

示例 1:输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8 输出: 6
解释: 节点 2 和节点 8 的最近公共祖先是 6。

示例 2:输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4 输出: 2
解释: 节点 2 和节点 4 的最近公共祖先是 2, 因为根据定义最近公共祖先节点可以为节点本身。

说明:
    所有节点的值都是唯一的。
    p、q 为不同节点且均存在于给定的二叉搜索树中。
```

- 解题思路

```
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
        if p.Val > root.Val && q.Val > root.Val{
                return lowestCommonAncestor(root.Right, p, q)
        }else if p.Val < root.Val && q.Val < root.Val{
                return lowestCommonAncestor(root.Left, p, q)
        }else {
                return root
        }
}

#
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
        for root != nil{
```

(续下页)

(接上页)

```
                if p.Val > root.Val && q.Val > root.Val{
                        root = root.Right
                }else if p.Val < root.Val && q.Val < root.Val{
                        root = root.Left
                }else {
                        return root
                }
        }
        return nil
}
```

7.14 237. 删除链表中的节点 (1)

- 题目

```
请编写一个函数，使其可以删除某个链表中给定的（非末尾）节点，你将只被给定要求被删除的节点。
现有一个链表 -- head = [4,5,1,9]，它可以表示为：

示例 1: 输入：head = [4,5,1,9], node = 5  输出：[4,1,9]
解释：给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -
↪> 9.

示例 2:输入：head = [4,5,1,9], node = 1 输出：[4,5,9]
解释：给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -
↪> 9.

说明：
    链表至少包含两个节点。
    链表中所有节点的值都是唯一的。
    给定的节点为非末尾节点并且一定是链表中的一个有效节点。
    不要从你的函数中返回任何结果。
```

- 解题思路

```
func deleteNode(node *ListNode) {
        node.Val = node.Next.Val
        node.Next = node.Next.Next
}
```

7.15 242. 有效的字母异位词 (2)

- 题目

```
给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。
示例 1:输入: s = "anagram", t = "nagaram"输出: true
示例 2:输入: s = "rat", t = "car"输出: false

说明:你可以假设字符串只包含小写字母。
进阶:如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？
```

- 解题思路

```
func isAnagram(s string, t string) bool {
        if len(s) != len(t) {
                return false
        }

        sr := []rune(s)
        tr := []rune(t)

        rec := make(map[rune]int, len(sr))
        for i := range sr {
                rec[sr[i]]++
                rec[tr[i]]--
        }

        for _, n := range rec {
                if n != 0 {
                        return false
                }
        }
        return true
}

#
func isAnagram(s string, t string) bool {
        if len(s) != len(t) {
                return false
        }
        sArr := make([]int, len(s))
        tArr := make([]int, len(t))
        for i := 0; i < len(s); i++ {
                sArr[i] = int(s[i] - 'a')
```

(续下页)

(接上页)

```
                tArr[i] = int(t[i] - 'a')
        }
        sort.Ints(sArr)
        sort.Ints(tArr)
        for i := 0; i < len(s); i++ {
                if sArr[i] != tArr[i] {
                        return false
                }
        }
        return true
}
```

7.16 257. 二叉树的所有路径 (2)

- 题目

```
给定一个二叉树，返回所有从根节点到叶子节点的路径。
说明: 叶子节点是指没有子节点的节点。
示例:
输入:

   1
 /   \
2     3
 \
  5
输出: ["1->2->5", "1->3"]
解释: 所有根节点到叶子节点的路径为: 1->2->5, 1->3
```

- 解题思路

```
#
func binaryTreePaths(root *TreeNode) []string {
        if root == nil {
                return nil
        }

        res := make([]string, 0)
        var dfs func(string, *TreeNode)
        dfs = func(pre string, root *TreeNode) {
                if pre == "" {
                        pre = strconv.Itoa(root.Val)
```

(续下页)

(接上页)

```
                } else {
                        pre += "->" + strconv.Itoa(root.Val)
                }

                if root.Left != nil {
                        dfs(pre, root.Left)
                }

                if root.Right != nil {
                        dfs(pre, root.Right)
                }

                if root.Left == nil && root.Right == nil {
                        res = append(res, pre)
                }
        }

        dfs("", root)
        return res
}

#
func binaryTreePaths(root *TreeNode) []string {
        res := make([]string, 0)
        if root == nil {
                return res
        }
        var queue []*TreeNode
        var stringQueue []string
        queue = append(queue, root)
        stringQueue = append(stringQueue, strconv.Itoa(root.Val))
        for len(queue) > 0 {
                node := queue[0]
                path := stringQueue[0]
                queue = queue[1:]
                stringQueue = stringQueue[1:]
                if node.Left == nil && node.Right == nil {
                        res = append(res, path)
                }
                if node.Left != nil {
                        queue = append(queue, node.Left)
                        stringQueue = append(stringQueue, path+"->"+strconv.Itoa(node.
↪Left.Val))
```

(续下页)

(接上页)

```
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                        stringQueue = append(stringQueue, path+"->"+strconv.Itoa(node.
↪Right.Val))
                }
        }
        return res
}
```

7.17 258. 各位相加 (4)

- 题目

```
给定一个非负整数 num，反复将各个位上的数字相加，直到结果为一位数。
示例：输入：38 输出：2
解释：各位相加的过程为：3 + 8 = 11，1 + 1 = 2。 由于 2 是一位数，所以返回 2。
进阶：
你可以不使用循环或者递归，且在 O(1) 时间复杂度内解决这个问题吗？
```

- 解题思路

```
# 找规律1
func addDigits(num int) int {
        if num < 10 {
                return num
        }
        if num%9 == 0 {
                return 9
        }
        return num % 9
}

# 找规律2
func addDigits(num int) int {
        return (num-1)%9 + 1
}

# 模拟计算-字符串
func addDigits(num int) int {
        for num >= 10 {
                num = sumDigits(num)
```

(续下页)

(接上页)

```
        }
        return num
}

func sumDigits(num int) int {
        sumVal := 0
        str := strconv.Itoa(num)
        for i := range str {
                sumVal = sumVal + int(str[i]-'0')
        }
        return sumVal
}

# 模拟计算-递归
func addDigits(num int) int {
        sum := 0
        for num != 0 {
                sum = sum + num%10
                num = num / 10
        }
        if sum/10 == 0 {
                return sum
        }
        return addDigits(sum)
}
```

7.18 263. 丑数 (2)

- 题目

```
丑数就是只包含质因数 2, 3, 5 的正整数。

示例 1:输入: 6 输出: true 解释: 6 = 2 × 3
示例 2:输入: 8 输出: true 解释: 8 = 2 × 2 × 2
示例 3: 输入: 14 输出: false  解释: 14 不是丑数, 因为它包含了另外一个质因数 7。

说明:
1 是丑数。
输入不会超过 32 位有符号整数的范围: [−231,  231 − 1]。
```

- 解题思路

```
func isUgly(num int) bool {
        if num <= 0 {
                return false
        }
        if num <= 6 {
                return true
        }
        if num%2 == 0 {
                return isUgly(num / 2)
        }
        if num%3 == 0 {
                return isUgly(num / 3)
        }
        if num%5 == 0 {
                return isUgly(num / 5)
        }
        return false
}

# 迭代
func isUgly(num int) bool {
        if num <= 0 {
                return false
        }
        for num != 1 {
                if num%2 == 0 {
                        num = num / 2
                } else if num%3 == 0 {
                        num = num / 3
                } else if num%5 == 0 {
                        num = num / 5
                } else {
                        return false
                }
        }
        return true
}
```

7.19 268. 缺失数字 (5)

- 题目

```
给定一个包含 0, 1, 2, ..., n 中 n 个数的序列，找出 0 .. n 中没有出现在序列中的那个数。

示例 1:输入: [3,0,1]输出: 2
示例 2:输入: [9,6,4,2,3,5,7,0,1] 输出: 8

说明:你的算法应具有线性时间复杂度。你能否仅使用额外常数空间来实现?
```

- 解题思路

```
func missingNumber(nums []int) int {
        n := len(nums)
        sum := n * (n + 1) / 2
        for i := 0; i < n; i++ {
                sum = sum - nums[i]
        }
        return sum
}

# 2
func missingNumber(nums []int) int {
        sort.Ints(nums)
        for i := 0; i < len(nums); i++ {
                if nums[i] != i {
                        return i
                }
        }
        return len(nums)
}

# 3
func missingNumber(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                res = res ^ (i+1) ^ nums[i]
        }
        return res
}

# 4
func missingNumber(nums []int) int {
```

(续下页)

(接上页)

```
        n := len(nums)
        // 假设index=n
        index := n
        for i := 0; i < n; {
                // nums[i]到指定位置i后往后走
                if i == nums[i] {
                        i++
                        continue
                }
                if nums[i] == n {
                        index = i
                        i++
                        continue
                }
                nums[i], nums[nums[i]] = nums[nums[i]], nums[i]
        }
        return index
}

# 5
func missingNumber(nums []int) int {
        m := make(map[int]bool)
        for i := range nums{
                m[nums[i]] = true
        }
        for i := 0; i <= len(nums); i++{
                if m[i] == false{
                        return i
                }
        }
        return 0
}
```

7.20 278. 第一个错误的版本 (2)

- 题目

```
你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。
由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。
假设你有 n 个版本 [1, 2, ..., n]，你想找出导致之后所有版本出错的第一个错误的版本。
你可以通过调用 bool isBadVersion(version) 接口来判断版本号 version␣
↪是否在单元测试中出错。
```

(续下页)

(接上页)

```
实现一个函数来查找第一个错误的版本。你应该尽量减少对调用 API 的次数。

示例:
给定 n = 5, 并且 version = 4 是第一个错误的版本。

调用 isBadVersion(3) -> false
调用 isBadVersion(5) -> true
调用 isBadVersion(4) -> true
所以, 4 是第一个错误的版本。
```

- 解题思路

```
func firstBadVersion(n int) int {
        low := 1
        high := n
        for low <= high {
                mid := low + (high-low)/2
                if isBadVersion(mid) == false {
                        low = mid + 1
                } else if isBadVersion(mid) == true && isBadVersion(mid-1) == true {
                        high = mid - 1
                } else if isBadVersion(mid) == true && isBadVersion(mid-1) == false {
                        return mid
                }
        }
        return -1
}

#
func firstBadVersion(n int) int {
        low := 1
        high := n
        for low < high {
                mid := low + (high-low)/2
                if isBadVersion(mid) {
                        high = mid
                } else {
                        low = mid + 1
                }
        }
        return low
}
```

7.21 283. 移动零 (3)

- 题目

```
给定一个数组 nums，编写一个函数将所有 0 移动到数组的末尾，同时保持非零元素的相对顺序。
示例:
输入: [0,1,0,3,12]
输出: [1,3,12,0,0]
说明:
    必须在原数组上操作，不能拷贝额外的数组。
    尽量减少操作次数。
```

- 解题思路

```
func moveZeroes(nums []int) {
        length := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] != 0 {
                        nums[length] = nums[i]
                        length++
                }
        }

        for i := length; i < len(nums); i++ {
                nums[i] = 0
        }
}

#
func moveZeroes(nums []int) {
        length := 0
        for i:= 0; i < len(nums); i++ {
                nums[i], nums[length] = nums[length], nums[i]
                if nums[length] != 0 {
                        length++
                }
        }
}

#
func moveZeroes(nums []int) {
        arr := make([]int,len(nums))
        count := 0
        for i := range nums{
```

(续下页)

(接上页)

```
                if nums[i] != 0{
                        arr[count] = nums[i]
                        count++
                }
        }

        copy(nums, arr)
}
```

7.22 290. 单词规律 (2)

- 题目

```
给定一种规律 pattern 和一个字符串 str ，判断 str 是否遵循相同的规律。

这里的 遵循 指完全匹配，
例如， pattern 里的每个字母和字符串 str 中的每个非空单词之间存在着双向连接的对应规律。

示例1:输入: pattern = "abba", str = "dog cat cat dog"输出: true
示例 2:输入:pattern = "abba", str = "dog cat cat fish"输出: false
示例 3:输入: pattern = "aaaa", str = "dog cat cat dog"输出: false
示例 4:输入: pattern = "abba", str = "dog dog dog dog" 输出: false

说明:
你可以假设 pattern 只包含小写字母， str 包含了由单个空格分隔的小写字母。
```

- 解题思路

```
func wordPattern(pattern string, str string) bool {
        pa := strings.Split(pattern, "")
        sa := strings.Split(str, " ")
        if len(pa) != len(sa) {
                return false
        }
        length := len(pa)
        pMap := make(map[string]string, length)
        sMap := make(map[string]string, length)

        for i := 0; i < length; i++ {
                pStr, ok := pMap[pa[i]]
                sStr, ok1 := sMap[sa[i]]
```

(续下页)

(接上页)

```
                if (ok && pStr != sa[i]) || (ok1 && sStr != pa[i]) {
                        return false
                } else {
                        pMap[pa[i]] = sa[i]
                        sMap[sa[i]] = pa[i]
                }
        }
        return true
}

#
func wordPattern(pattern string, str string) bool {
        pa := strings.Split(pattern, "")
        sa := strings.Split(str, " ")
        if len(pa) != len(sa) {
                return false
        }
        return isMatch(pa, sa) && isMatch(sa, pa)
}

func isMatch(pa, sa []string) bool {
        length := len(pa)
        m := make(map[string]string, length)
        for i := 0; i < length; i++ {
                if w, ok := m[pa[i]]; ok && w != sa[i] {
                        return false
                } else {
                        m[pa[i]] = sa[i]
                }
        }
        return true
}
```

7.23 292.Nim 游戏 (1)

- 题目

```
你和你的朋友，两个人一起玩 Nim 游戏：桌子上有一堆石头，每次你们轮流拿掉 1 - 3 块石头。
拿掉最后一块石头的人就是获胜者。你作为先手。
你们是聪明人，每一步都是最优解。␣
↪编写一个函数，来判断你是否可以在给定石头数量的情况下赢得游戏。
```

(续下页)

(接上页)

```
示例:
输入: 4
输出: false
解释: 如果堆中有 4 块石头，那么你永远不会赢得比赛;
     因为无论你拿走 1 块、2 块 还是 3 块石头，最后一块石头总是会被你的朋友拿走。
```

- 解题思路

```
func canWinNim(n int) bool {
        // return n&3 != 0
        return n%4 != 0
}
```

7.24 299. 猜数字游戏 (2)

- 题目

```
你正在和你的朋友玩 猜数字（Bulls and Cows）游戏：你写下一个数字让你的朋友猜。
每次他猜测后，你给他一个提示，告诉他有多少位数字和确切位置都猜对了（称为“Bulls”,␣
↪公牛），
有多少位数字猜对了但是位置不对（称为“Cows”，奶牛）。
你的朋友将会根据提示继续猜，直到猜出秘密数字。

请写出一个根据秘密数字和朋友的猜测数返回提示的函数，用 A 表示公牛，用 B 表示奶牛。
请注意秘密数字和朋友的猜测数都可能含有重复数字。

示例 1:输入: secret = "1807", guess = "7810"输出: "1A3B"
解释: 1 公牛和 3 奶牛。公牛是 8，奶牛是 0，1 和 7。

示例 2:输入: secret = "1123", guess = "0111"输出: "1A1B"
解释: 朋友猜测数中的第一个 1 是公牛，第二个或第三个 1 可被视为奶牛。

说明: 你可以假设秘密数字和朋友的猜测数都只包含数字，并且它们的长度永远相等。
```

- 解题思路

```
func getHint(secret string, guess string) string {
        length := len(secret)
        right := 0
        wrongLoc := 0
        m := make(map[byte]int)
        n := make(map[byte]int)
```

(续下页)

(接上页)

```
        for i := 0; i < length; i++ {
                if secret[i] == guess[i] {
                        right++
                } else {
                        m[secret[i]]++
                        n[guess[i]]++
                }
        }
        for i := range m {
                if m[i] < n[i] {
                        wrongLoc = wrongLoc + m[i]
                } else {
                        wrongLoc = wrongLoc + n[i]
                }
        }

        return fmt.Sprintf("%dA%dB", right, wrongLoc)
}

#
func getHint(secret string, guess string) string {
        length := len(secret)
        right := 0
        wrongNum := 0
        m := make(map[int]int)
        for i := 0; i < length; i++ {
                if secret[i] == guess[i] {
                        right++
                }
                m[int(secret[i]-'0')]++
                m[int(guess[i]-'0')]--
        }
        for i := range m {
                if m[i] > 0{
                        wrongNum = wrongNum + m[i]
                }
        }
        // wrongLoc = 总数 - 猜对的数 - 猜错的数
        wrongLoc := length - right - wrongNum
        return fmt.Sprintf("%dA%dB", right, wrongLoc)
}
```

CHAPTER 8

0201-0300-Medium

8.1 201. 数字范围按位与 (2)

- 题目

```
给定范围 [m, n]，其中 0 <= m <= n <= 2147483647，返回此范围内所有数字的按位与（包含 m,
↪ n 两端点）。
示例 1: 输入: [5,7]输出: 4
示例 2:输入: [0,1]输出: 0
```

- 解题思路

```
func rangeBitwiseAnd(m int, n int) int {
        count := 0
        // 找m,n的32位二进制，前面相同的位数，然后后面添0
        for m != n {
                count++
                // 同时右移去除末尾1位
                m = m >> 1
                n = n >> 1
        }
        return m << count
}

# 2
```

(续下页)

(接上页)

```
func rangeBitwiseAnd(m int, n int) int {
        for m < n {
                n = n & (n - 1) // n抹去右边1位1
        }
        return n
}
```

8.2 207. 课程表 (2)

- 题目

```
你这个学期必须选修 numCourse 门课程，记为 0 到 numCourse-1 。
在选修某些课程之前需要一些先修课程。
例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们：[0,1]
给定课程总量以及它们的先决条件，请你判断是否可能完成所有课程的学习？
示例 1:输入：2, [[1,0]]  输出：true
解释：总共有 2 门课程。学习课程 1 之前，你需要完成课程 0。所以这是可能的。
示例 2:输入：2, [[1,0],[0,1]] 输出：false
解释：总共有 2 门课程。学习课程 1 之前，你需要先完成课程 0；并且学习课程 0 之前，
你还应先完成课程 1。这是不可能的。
提示：
    输入的先决条件是由 边缘列表 表示的图形，而不是 邻接矩阵 。详情请参见图的表示法。
    你可以假定输入的先决条件中没有重复的边。
    1 <= numCourses <= 10^5
```

- 解题思路

```
var res bool
var visited []int
var path []int
var edges [][]int

func canFinish(numCourses int, prerequisites [][]int) bool {
        res = true
        edges = make([][]int, numCourses) // 邻接表
        visited = make([]int, numCourses)
        path = make([]int, 0)
        for i := 0; i < len(prerequisites); i++ {
                // prev->cur
                prev := prerequisites[i][1]
                cur := prerequisites[i][0]
                edges[prev] = append(edges[prev], cur)
```

(续下页)

(接上页)

```
        }
        for i := 0; i < numCourses; i++ {
                if visited[i] == 0 {
                        dfs(i)
                }
                if res == false {
                        return false
                }
        }
        return res
}

func dfs(start int) {
        // 0 未搜索
        // 1 搜索中
        // 2 已完成
        visited[start] = 1
        for i := 0; i < len(edges[start]); i++ {
                out := edges[start][i]
                if visited[out] == 0 {
                        dfs(out)
                        if res == false {
                                return
                        }
                } else if visited[out] == 1 {
                        res = false
                        return
                }
        }
        visited[start] = 2
        path = append(path, start)
}

# 2
func canFinish(numCourses int, prerequisites [][]int) bool {
        edges := make([][]int, numCourses)
        path := make([]int, 0)
        inEdges := make([]int, numCourses)
        for i := 0; i < len(prerequisites); i++ {
                // prev->cur
                prev := prerequisites[i][1]
                cur := prerequisites[i][0]
                edges[prev] = append(edges[prev], cur)
```

(续下页)

(接上页)

```
                inEdges[cur]++ // 入度
        }
        // 入度为0
        queue := make([]int, 0)
        for i := 0; i < numCourses; i++ {
                if inEdges[i] == 0 {
                        queue = append(queue, i)
                }
        }
        for len(queue) > 0 {
                start := queue[0]
                queue = queue[1:]
                path = append(path, start)
                for i := 0; i < len(edges[start]); i++ {
                        out := edges[start][i]
                        inEdges[out]--
                        if inEdges[out] == 0 {
                                queue = append(queue, out)
                        }
                }
        }
        return len(path) == numCourses
}
```

8.3 208. 实现 Trie(前缀树)(2)

- 题目

```
实现一个 Trie (前缀树), 包含 insert, search, 和 startsWith 这三个操作。
示例:Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 true
trie.search("app");     // 返回 false
trie.startsWith("app"); // 返回 true
trie.insert("app");
trie.search("app");     // 返回 true
说明:你可以假设所有的输入都是由小写字母 a-z 构成的。
    保证所有输入均为非空字符串。
```

- 解题思路

```
type Trie struct {
        next   [26]*Trie
        ending int
}

func Constructor() Trie {
        return Trie{
                next:   [26]*Trie{},
                ending: 0,
        }
}

func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

func (this *Trie) Search(word string) bool {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}

func (this *Trie) StartsWith(prefix string) bool {
        temp := this
        for _, v := range prefix {
```

(续下页)

(接上页)

```
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        return true
}

# 2
type Trie struct {
        next   map[byte]*Trie
        ending int
}

/** Initialize your data structure here. */
func Constructor() Trie {
        return Trie{
                next:   make(map[byte]*Trie),
                ending: 0,
        }
}

/** Inserts a word into the trie. */
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := byte(v - 'a')
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   make(map[byte]*Trie),
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

/** Returns if the word is in the trie. */
func (this *Trie) Search(word string) bool {
        temp := this
        for _, v := range word {
                value := byte(v - 'a')
```

(续下页)

(接上页)

```
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}

/** Returns if there is any word in the trie that starts with the given prefix. */
func (this *Trie) StartsWith(prefix string) bool {
        temp := this
        for _, v := range prefix {
                value := byte(v - 'a')
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        return true
}
```

8.4 209. 长度最小的子数组 (3)

- 题目

```
给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 ≥ s 的长度最小的␣
↪连续 子数组，
并返回其长度。如果不存在符合条件的子数组，返回 0。
示例：输入：s = 7, nums = [2,3,1,2,4,3] 输出：2
解释：子数组 [4,3] 是该条件下的长度最小的子数组。
进阶：如果你已经完成了 O(n) 时间复杂度的解法，请尝试 O(n log n) 时间复杂度的解法。
```

- 解题思路

```
func minSubArrayLen(target int, nums []int) int {
        res := math.MaxInt32
        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum = sum + nums[j]
                        if sum >= target {
```

(续下页)

(接上页)

```
                                if res > j-i+1 {
                                        res = j - i + 1
                                }
                                break
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}

# 2
func minSubArrayLen(target int, nums []int) int {
        res := math.MaxInt32
        arr := make([]int, len(nums)+1)
        for i := 1; i <= len(nums); i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        for i := 1; i <= len(nums); i++ {
                target := target + arr[i-1]
                index := sort.SearchInts(arr, target)
                if index <= len(nums) {
                        if res > index-i+1 {
                                res = index - i + 1
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}

# 3
func minSubArrayLen(s int, nums []int) int {
        res := math.MaxInt32
        i, j := 0, 0
        sum := 0
        for ; j < len(nums); j++ {
                sum = sum + nums[j]
                for sum >= s {
```

(续下页)

(接上页)

```
                        if res > j-i+1 {
                                res = j - i + 1
                        }
                        sum = sum - nums[i]
                        i++
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}
```

8.5 210. 课程表 II(2)

- 题目

```
现在你总共有 n 门课需要选，记为 0 到 n-1。
在选修某些课程之前需要一些先修课程。
例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们: [0,1]
给定课程总量以及它们的先决条件，返回你为了学完所有课程所安排的学习顺序。
可能会有多个正确的顺序，你只要返回一种就可以了。如果不可能完成所有课程，返回一个空数组。
示例 1:输入: 2, [[1,0]]  输出: [0,1]
解释: 总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1]␣
↪。
示例 2:输入: 4, [[1,0],[2,0],[3,1],[3,2]] 输出: [0,1,2,3] or [0,2,1,3]
解释: 总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。
并且课程 1 和课程 2 都应该排在课程 0 之后。
     因此，一个正确的课程顺序是 [0,1,2,3] 。另一个正确的排序是 [0,2,1,3] 。
说明:输入的先决条件是由边缘列表表示的图形，而不是邻接矩阵。详情请参见图的表示法。
    你可以假定输入的先决条件中没有重复的边。
提示:
    这个问题相当于查找一个循环是否存在于有向图中。
    如果存在循环，则不存在拓扑排序，因此不可能选取所有课程进行学习。
    通过 DFS 进行拓扑排序 -␣
↪一个关于Coursera的精彩视频教程（21分钟），介绍拓扑排序的基本概念。
    拓扑排序也可以通过 BFS 完成。
```

- 解题思路

```
var res bool
var visited []int
```

(续下页)

(接上页)

```
var path []int
var edges [][]int

func findOrder(numCourses int, prerequisites [][]int) []int {
        res = true
        edges = make([][]int, numCourses) // 邻接表
        visited = make([]int, numCourses)
        path = make([]int, 0)
        for i := 0; i < len(prerequisites); i++ {
                // prev->cur
                prev := prerequisites[i][1]
                cur := prerequisites[i][0]
                edges[prev] = append(edges[prev], cur)
        }
        for i := 0; i < numCourses; i++ {
                if visited[i] == 0 {
                        dfs(i)
                }
                if res == false {
                        return nil
                }
        }
        for i := 0; i < len(path)/2; i++ {
                path[i], path[len(path)-1-i] = path[len(path)-1-i], path[i]
        }
        return path
}

func dfs(start int) {
        // 0 未搜索
        // 1 搜索中
        // 2 已完成
        visited[start] = 1
        for i := 0; i < len(edges[start]); i++ {
                out := edges[start][i]
                if visited[out] == 0 {
                        dfs(out)
                        if res == false {
                                return
                        }
                } else if visited[out] == 1 {
                        res = false
                        return
```

(续下页)

(接上页)

```
                }
        }
        visited[start] = 2
        path = append(path, start)
}

# 2
func findOrder(numCourses int, prerequisites [][]int) []int {
        edges := make([][]int, numCourses)
        path := make([]int, 0)
        inEdges := make([]int, numCourses)
        for i := 0; i < len(prerequisites); i++ {
                // prev->cur
                prev := prerequisites[i][1]
                cur := prerequisites[i][0]
                edges[prev] = append(edges[prev], cur)
                inEdges[cur]++ // 入度
        }
        // 入度为0
        queue := make([]int, 0)
        for i := 0; i < numCourses; i++ {
                if inEdges[i] == 0 {
                        queue = append(queue, i)
                }
        }
        for len(queue) > 0 {
                start := queue[0]
                queue = queue[1:]
                path = append(path, start)
                for i := 0; i < len(edges[start]); i++ {
                        out := edges[start][i]
                        inEdges[out]--
                        if inEdges[out] == 0 {
                                queue = append(queue, out)
                        }
                }
        }
        if len(path) != numCourses {
                return nil
        }
        return path
}
```

8.6 211. 添加与搜索单词-数据结构设计 (1)

- 题目

```
如果数据结构中有任何与word匹配的字符串，则bool search（word）返回true，否则返回false。
单词可能包含点“。” 点可以与任何字母匹配的地方。
请你设计一个数据结构，支持 添加新单词 和 查找字符串是否与任何先前添加的字符串匹配 。
实现词典类 WordDictionary ：
    WordDictionary() 初始化词典对象
    void addWord(word) 将 word 添加到数据结构中，之后可以对它进行匹配
    bool search(word) 如果数据结构中存在字符串与 word 匹配，则返回 true ；
    否则，返回  false 。word 中可能包含一些 '.' ，每个 . 都可以表示任何一个字母。
示例：输入：
["WordDictionary","addWord","addWord","addWord","search","search","search","search"]
[[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]
输出：[null,null,null,null,false,true,true,true]
解释：
WordDictionary wordDictionary = new WordDictionary();
wordDictionary.addWord("bad");
wordDictionary.addWord("dad");
wordDictionary.addWord("mad");
wordDictionary.search("pad"); // return False
wordDictionary.search("bad"); // return True
wordDictionary.search(".ad"); // return True
wordDictionary.search("b.."); // return True
提示：1 <= word.length <= 500
    addWord 中的 word 由小写英文字母组成
    search 中的 word 由 '.' 或小写英文字母组成
    最调用多 50000 次 addWord 和 search
```

- 解题思路

```
type Trie struct {
        next   [26]*Trie
        ending int
}

func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
```

(续下页)

(接上页)

```
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

func (this *Trie) Search(word string, k int) bool {
        temp := this
        for i := k; i < len(word); i++ {
                if word[i] == '.' {
                        for j := 0; j < len(temp.next); j++ {
                                if temp.next[j] != nil && temp.next[j].Search(word,␣
↪i+1) {
                                        return true
                                }
                        }
                        return false
                }
                value := word[i] - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}

type WordDictionary struct {
        trie *Trie
}

func Constructor() WordDictionary {
        return WordDictionary{trie: &Trie{}}
}

func (this *WordDictionary) AddWord(word string) {
        this.trie.Insert(word)
}
```

(续下页)

(接上页)

```
func (this *WordDictionary) Search(word string) bool {
        return this.trie.Search(word, 0)
}
```

8.7 213. 打家劫舍 II(3)

- 题目

```
你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。
这个地方所有的房屋都围成一圈，这意味着第一个房屋和最后一个房屋是紧挨着的。
同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。
给定一个代表每个房屋存放金额的非负整数数组，计算你在不触动警报装置的情况下，能够偷窃到的最高金额。
示例 1:输入: [2,3,2] 输出: 3
解释: 你不能先偷窃 1 号房屋（金额 = 2），然后偷窃 3 号房屋（金额 = 2），␣
↪因为他们是相邻的。
示例 2:输入: [1,2,3,1] 输出: 4
解释: 你可以先偷窃 1 号房屋（金额 = 1），然后偷窃 3 号房屋（金额 = 3）。
     偷窃到的最高金额 = 1 + 3 = 4 。
```

- 解题思路

```
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        } else if n == 1 {
                return nums[0]
        }
        dp1 := make([]int, n) // 从第一家开始打劫，最后一家不可选
        dp2 := make([]int, n) // 从第二家开始打劫，最后一家可以选
        dp1[0] = nums[0]
        dp1[1] = max(nums[0], nums[1])
        dp2[0] = 0
        dp2[1] = nums[1]
        for i := 2; i < n; i++ {
                dp1[i] = max(dp1[i-1], dp1[i-2]+nums[i])
                dp2[i] = max(dp2[i-1], dp2[i-2]+nums[i])
        }
        return max(dp1[n-2], dp2[n-1])
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 2
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        } else if n == 1 {
                return nums[0]
        } else if n == 2 {
                return max(nums[0], nums[1])
        }
        return max(getMax(nums[:n-1]), getMax(nums[1:]))
}

func getMax(nums []int) int {
        n := len(nums)
        dp := make([]int, n+1)
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])
        for i := 2; i < n; i++ {
                dp[i] = max(dp[i-1], dp[i-2]+nums[i])
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        } else if n == 1 {
```

(续下页)

(接上页)

```
                return nums[0]
        } else if n == 2 {
                return max(nums[0], nums[1])
        }
        return max(getMax(nums[:n-1]), getMax(nums[1:]))
}

func getMax(nums []int) int {
        var a, b int
        for i, v := range nums {
                if i%2 == 0 {
                        a = max(a+v, b)
                } else {
                        b = max(a, b+v)
                }
        }
        return max(a, b)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

8.8 215. 数组中的第 K 个最大元素 (3)

- 题目

```
在未排序的数组中找到第 k 个最大的元素。
请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。
示例 1:输入: [3,2,1,5,6,4] 和 k = 2 输出: 5
示例 2:输入: [3,2,3,1,2,4,5,5,6] 和 k = 4 输出: 4
说明:你可以假设 k 总是有效的，且 1 ≤ k ≤ 数组的长度。
```

- 解题思路

```
func findKthLargest(nums []int, k int) int {
        sort.Ints(nums)
        return nums[len(nums)-k]
}
```

(续下页)

(接上页)

```
# 2
func findKthLargest(nums []int, k int) int {
        heapSize := len(nums)
        buildMaxHeap(nums, heapSize)
        for i := len(nums) - 1; i >= len(nums)-k+1; i-- {
                nums[0], nums[i] = nums[i], nums[0]
                heapSize--
                maxHeapify(nums, 0, heapSize)
        }
        return nums[0]
}

func buildMaxHeap(a []int, heapSize int) {
        for i := heapSize / 2; i >= 0; i-- {
                maxHeapify(a, i, heapSize)
        }
}

func maxHeapify(a []int, i, heapSize int) {
        l, r, largest := i*2+1, i*2+2, i
        if l < heapSize && a[l] > a[largest] {
                largest = l
        }
        if r < heapSize && a[r] > a[largest] {
                largest = r
        }
        if largest != i {
                a[i], a[largest] = a[largest], a[i]
                maxHeapify(a, largest, heapSize)
        }
}

# 3
func findKthLargest(nums []int, k int) int {
        return findK(nums, 0, len(nums)-1, k)
}

func findK(nums []int, start, end int, k int) int {
        if start >= end {
                return nums[end]
        }
        index := partition(nums, start, end)
```

(续下页)

(接上页)

```
        if index+1 == k {
                return nums[index]
        } else if index+1 < k {
                return findK(nums, index+1, end, k)
        }
        return findK(nums, start, index-1, k)
}

func partition(nums []int, start, end int) int {
        temp := nums[end]
        i := start
        for j := start; j < end; j++ {
                if nums[j] > temp {
                        if i != j {
                                nums[i], nums[j] = nums[j], nums[i]
                        }
                        i++
                }
        }
        nums[i], nums[end] = nums[end], nums[i]
        return i
}
```

8.9 216. 组合总和 III(1)

- 题目

```
找出所有相加之和为 n 的 k 个数的组合。组合中只允许含有 1 - 9␣
↪的正整数，并且每种组合中不存在重复的数字。
说明：
    所有数字都是正整数。
    解集不能包含重复的组合。
示例 1:输入: k = 3, n = 7 输出: [[1,2,4]]
示例 2:输入: k = 3, n = 9 输出: [[1,2,6], [1,3,5], [2,3,4]]
```

- 解题思路

```
var res [][]int

func combinationSum3(k int, n int) [][]int {
        res = make([][]int, 0)
        arr := make([]int, 0)
```

(续下页)

(接上页)

```
        dfs(k, n, 1, arr)
        return res
}

func dfs(k, n int, level int, arr []int) {
        if k == 0 || n < 0 {
                if n == 0 {
                        temp := make([]int, len(arr))
                        copy(temp, arr)
                        res = append(res, temp)
                }
                return
        }
        for i := level; i <= 9; i++ {
                dfs(k-1, n-i, i+1, append(arr, i))
        }
}
```

8.10 220. 存在重复元素 III(2)

- 题目

```
在整数数组 nums 中，是否存在两个下标 i 和 j，使得 nums [i] 和 nums [j]␣
↪的差的绝对值小于等于 t ，
且满足 i 和 j 的差的绝对值也小于等于 k 。
如果存在则返回 true，不存在返回 false。
示例 1:输入: nums = [1,2,3,1], k = 3, t = 0 输出: true
示例 2:输入: nums = [1,0,1,1], k = 1, t = 2 输出: true
示例 3:输入: nums = [1,5,9,1,5,9], k = 2, t = 3 输出: false
```

- 解题思路

```
func containsNearbyAlmostDuplicate(nums []int, k int, t int) bool {
        if len(nums) <= 1 {
                return false
        }
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums) && j <= i+k; j++ {
                        if abs(nums[i], nums[j]) <= t {
                                return true
                        }
                }
```

(续下页)

(接上页)

```
        }
        return false
}

# 2
func containsNearbyAlmostDuplicate(nums []int, k int, t int) bool {
        if len(nums) <= 1 || t < 0 {
                return false
        }
        m := make(map[int]int)
        width := t + 1
        for i := 0; i < len(nums); i++ {
                key := getKey(nums[i], width)
                if _, ok := m[key]; ok {
                        return true
                }
                if value, ok := m[key-1]; ok && abs(nums[i], value) < width {
                        return true
                }
                if value, ok := m[key+1]; ok && abs(nums[i], value) < width {
                        return true
                }
                m[key] = nums[i]
                if i >= k {
                        // 满足i和j的差的绝对值也小于等于k
                        delete(m, getKey(nums[i-k], width))
                }
        }
        return false
}

func getKey(value, width int) int {
        if value < 0 {
                return (value+1)/width - 1
        }
        return value / width
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

8.11 221. 最大正方形 (3)

- 题目

```
在一个由 0 和 1 组成的二维矩阵内，找到只包含 1 的最大正方形，并返回其面积。
示例:输入:
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
输出: 4
```

- 解题思路

```
func maximalSquare(matrix [][]byte) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '1' {
                                res = max(res, 1)
                                minLength := min(n-i, m-j)
                                for k := 1; k < minLength; k++ {
                                        flag := true
                                        if matrix[i+k][j+k] == '0' {
                                                break
                                        }
                                        for l := 0; l < k; l++ {
                                                if matrix[i+k][j+l] == '0' ||␣
↪matrix[i+l][j+k] == '0' {
                                                        flag = false
                                                        break
                                                }
                                        }
                                        if flag == true {
                                                res = max(res, k+1)
                                        } else {
                                                break
                                        }
                                }
                        }
```

(续下页)

(接上页)

```
                }
        }
        return res * res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func maximalSquare(matrix [][]byte) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        dp[i][j] = int(matrix[i][j] - '0')
                        if dp[i][j] == 1 {
                                res = 1
                        }
                }
        }
        for i := 1; i < n; i++ {
                for j := 1; j < m; j++ {
                        if dp[i][j] == 1 {
                                dp[i][j] = min(dp[i-1][j-1], min(dp[i-1][j], dp[i][j-
↪1])) + 1
                                res = max(res, dp[i][j])
                        }
```

(续下页)

(接上页)

```
                }
        }
        return res * res

}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func maximalSquare(matrix [][]byte) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '1' {
                                res = max(res, int(matrix[i][j]-'0'))
                        }
                        if i == 0 || j == 0 {
                                continue
                        }
                        if matrix[i][j] == '1' {
                                a := int(matrix[i-1][j-1] - '0')
                                b := int(matrix[i-1][j] - '0')
                                c := int(matrix[i][j-1] - '0')
                                matrix[i][j] = byte(min(a, min(b, c)) + 1 + '0')
                                res = max(res, int(matrix[i][j]-'0'))
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res * res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

8.12 222. 完全二叉树的节点个数 (3)

- 题目

```
给出一个完全二叉树，求出该树的节点个数。
说明：
完全二叉树的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，
并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2h␣
↪个节点。
示例:输入:
    1
   / \
  2   3
 / \  /
4  5 6
输出: 6
```

- 解题思路

```
func countNodes(root *TreeNode) int {
        if root == nil {
                return 0
        }
        return 1 + countNodes(root.Left) + countNodes(root.Right)
```

(续下页)

(接上页)

```
}

# 2
func countNodes(root *TreeNode) int {
        if root == nil {
                return 0
        }
        res := 0
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                res++
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return res
}

# 3
func countNodes(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := getLevel(root.Left)
        right := getLevel(root.Right)
        if left == right{
                return 1<<left+countNodes(root.Right)
        }
        return countNodes(root.Left) + 1<<right
}

func getLevel(root *TreeNode)int  {
        level := 0
        for root != nil{
                level++
                root = root.Left
        }
```

(续下页)

(接上页)

```
        return level
}
```

8.13 223. 矩形面积 (1)

- 题目

```
在二维平面上计算出两个由直线构成的矩形重叠后形成的总面积。
每个矩形由其左下顶点和右上顶点坐标表示，如图所示。
示例:输入: -3, 0, 3, 4, 0, -1, 9, 2 输出: 45
说明: 假设矩形面积不会超出 int 的范围。
```

- 解题思路

```
func computeArea(A int, B int, C int, D int, E int, F int, G int, H int) int {
        left, right := max(A, E), min(C, G)
        bottom, top := max(B, F), min(D, H)
        area1, area2 := (C-A)*(D-B), (G-E)*(H-F)
        if left < right && bottom < top {
                return area1 + area2 - (right-left)*(top-bottom)
        }
        return area1 + area2
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

8.14 227. 基本计算器 II(2)

- 题目

```
实现一个基本的计算器来计算一个简单的字符串表达式的值。
字符串表达式仅包含非负整数，+， - ，*，/ 四种运算符和空格  。 整数除法仅保留整数部分。
示例 1:输入: "3+2*2" 输出: 7
示例 2:输入: " 3/2 " 输出: 1
示例 3:输入: " 3+5 / 2 " 输出: 5
说明:
    你可以假设所给定的表达式都是有效的。
    请不要使用内置的库函数 eval。
```

- 解题思路

```
func calculate(s string) int {
        stack := make([]int, 0)
        op := make([]int, 0)
        num := 0
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        num = 0
                        for i < len(s) && '0' <= s[i] && s[i] <= '9' {
                                num = num*10 + int(s[i]-'0')
                                i++
                        }
                        // 处理乘除计算
                        if len(op) > 0 && op[len(op)-1] > 1 {
                                if op[len(op)-1] == 2 {
                                        stack[len(stack)-1] = stack[len(stack)-1] *␣
↪num
                                } else {
                                        stack[len(stack)-1] = stack[len(stack)-1] /␣
↪num
                                }
                                op = op[:len(op)-1]
                        } else {
                                stack = append(stack, num)
                        }
                        i--
                } else if s[i] == '+' {
                        op = append(op, 1)
                } else if s[i] == '-' {
                        op = append(op, -1)
```

(续下页)

(接上页)

```
                } else if s[i] == '*' {
                        op = append(op, 2)
                } else if s[i] == '/' {
                        op = append(op, 3)
                }
        }
        // 处理加减
        for len(op) > 0 {
                stack[1] = stack[0] + stack[1]*op[0]
                stack = stack[1:]
                op = op[1:]
        }
        return stack[0]
}

# 2
func calculate(s string) int {
        s = strings.Trim(s, " ") // 避免"3/2 "的情况
        stack := make([]int, 0)
        num := 0
        sign := byte('+')
        for i := 0; i < len(s); i++ {
                if s[i] == ' ' {
                        continue
                }
                if '0' <= s[i] && s[i] <= '9' {
                        num = num*10 + int(s[i]-'0')
                }
                if s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' || i ==␣
↪len(s)-1 {
            // 处理前一个符号
                        switch sign {
                        case '+':
                                stack = append(stack, num)
                        case '-':
                                stack = append(stack, -num)
                        case '*':
                                prev := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                stack = append(stack, num*prev)
                        case '/':
                                prev := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
```

(续下页)

(接上页)

```
                                stack = append(stack, prev/num)
                        }
                        num = 0
                        sign = s[i]
                }
        }
        res := 0
        for i := 0; i < len(stack); i++ {
                res = res + stack[i]
        }
        return res
}
```

8.15 228. 汇总区间 (2)

- 题目

```
给定一个无重复元素的有序整数数组，返回数组区间范围的汇总。
示例 1:输入: [0,1,2,4,5,7]输出: ["0->2","4->5","7"]
解释: 0,1,2 可组成一个连续的区间; 4,5 可组成一个连续的区间。
示例 2:输入: [0,2,3,4,6,8,9]输出: ["0","2->4","6","8->9"]
解释: 2,3,4 可组成一个连续的区间; 8,9 可组成一个连续的区间。
```

- 解题思路

```
func summaryRanges(nums []int) []string {
        res := make([]string, 0)
        if len(nums) == 0 {
                return res
        }
        i, j := 0, 1
        for j < len(nums) {
                if nums[j]-nums[j-1] != 1 {
                        str := ""
                        if j-i > 1 {
                                str = strconv.Itoa(nums[i]) + "->" + strconv.
↪Itoa(nums[j-1])
                        } else {
                                str = strconv.Itoa(nums[i])
                        }
                        res = append(res, str)
                        i = j
```

(续下页)

(接上页)

```
                }
                j++
        }
        if j == len(nums) {
                str := ""
                if j-i > 1 {
                        str = strconv.Itoa(nums[i]) + "->" + strconv.Itoa(nums[j-1])
                } else {
                        str = strconv.Itoa(nums[i])
                }
                res = append(res, str)
        }
        return res
}

# 2
func summaryRanges(nums []int) []string {
        res := make([]string, 0)
        if len(nums) == 0 {
                return res
        }
        nums = append(nums, nums[0])
        i, j := 0, 1
        index := 0
        res = append(res, strconv.Itoa(nums[i]))
        for j = 1; j < len(nums); j++ {
                if nums[j]-nums[j-1] != 1 {
                        if j-i > 1 {
                                str := strconv.Itoa(nums[i]) + "->" + strconv.
↪Itoa(nums[j-1])
                                res[index] = str
                        }
                        res = append(res, strconv.Itoa(nums[j]))
                        i = j
                        index++
                }
        }
        return res[:index]
}
```

8.16 229. 求众数 II(2)

- 题目

```
给定一个大小为 n 的数组，找出其中所有出现超过 ⌊ n/3 ⌋ 次的元素。
说明：要求算法的时间复杂度为 O(n)，空间复杂度为 O(1)。
示例 1:输入：[3,2,3] 输出：[3]
示例 2:输入：[1,1,1,3,3,2,2,2] 输出：[1,2]
```

- 解题思路

```
func majorityElement(nums []int) []int {
        m := make(map[int]int)
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for k, v := range m {
                if v > len(nums)/3 {
                        res = append(res, k)
                }
        }
        return res
}

# 2
func majorityElement(nums []int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        a, b := nums[0], nums[0]
        countA, countB := 0, 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == a {
                        countA++
                        continue
                }
                if nums[i] == b {
                        countB++
                        continue
                }
                if countA == 0 {
                        a = nums[i]
```

(续下页)

(接上页)

```
                        countA++
                        continue
                }
                if countB == 0 {
                        b = nums[i]
                        countB++
                        continue
                }
                countA--
                countB--
        }
        countA, countB = 0, 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == a {
                        countA++
                } else if nums[i] == b {
                        countB++
                }
        }
        if countA > len(nums)/3 {
                res = append(res, a)
        }
        if countB > len(nums)/3 {
                res = append(res, b)
        }
        return res
}
```

8.17 230. 二叉搜索树中第 K 小的元素 (3)

- 题目

```
给定一个二叉搜索树，编写一个函数 kthSmallest 来查找其中第 k 个最小的元素。
说明：你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。
示例 1:输入: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
输出: 1
示例 2:输入: root = [5,3,6,2,4,null,null,1], k = 3
```

(续下页)

(接上页)

```
       5
      / \
     3   6
    / \
   2   4
  /
 1
输出: 3
进阶: 如果二叉搜索树经常被修改 (插入/删除操作) 并且你需要频繁地查找第 k 小的值,
你将如何优化 kthSmallest 函数?
```

- 解题思路

```
var res int
var index int

func kthSmallest(root *TreeNode, k int) int {
        res = 0
        index = k
        dfs(root)
        return res
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                index--
                if index == 0 {
                        res = root.Val
                        return
                }
                dfs(root.Right)
        }
}

# 2
func kthSmallest(root *TreeNode, k int) int {
        res := 0
        stack := make([]*TreeNode, 0)
        for k > 0 {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
```

(续下页)

(接上页)

```
                root = stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                res = root.Val
                k--
                root = root.Right
        }
        return res
}

# 3
var res []int

func kthSmallest(root *TreeNode, k int) int {
        res = make([]int, 0)
        dfs(root)
        return res[k-1]
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                res = append(res, root.Val)
                dfs(root.Right)
        }
}
```

8.18 236. 二叉树的最近公共祖先 (2)

- 题目

```
给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。
百度百科中最近公共祖先的定义为: “对于有根树 T 的两个结点␣
↪p、q, 最近公共祖先表示为一个结点 x,
满足 x 是 p、q 的祖先且 x 的深度尽可能大 (一个节点也可以是它自己的祖先) 。”
例如, 给定如下二叉树: root = [3,5,1,6,2,0,8,null,null,7,4]
示例 1:输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1输出: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。
示例 2:输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4输出: 5
解释: 节点 5 和节点 4 的最近公共祖先是节点␣
↪5。因为根据定义最近公共祖先节点可以为节点本身。
说明:所有节点的值都是唯一的。
```

- 解题思路

```
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Val == p.Val || root.Val == q.Val {
                return root
        }
        left := lowestCommonAncestor(root.Left, p, q)
        right := lowestCommonAncestor(root.Right, p, q)
        if left != nil && right != nil {
                return root
        }
        if left == nil {
                return right
        }
        return left
}

# 2
var m map[int]*TreeNode

func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        m = make(map[int]*TreeNode)
        dfs(root)
        visited := make(map[int]bool)
        for p != nil {
                visited[p.Val] = true
                p = m[p.Val]
        }
        for q != nil {
                if visited[q.Val] == true {
                        return q
                }
                q = m[q.Val]
        }
        return nil
}

func dfs(root *TreeNode) {
        if root == nil {
                return
```

(续下页)

(接上页)

```
        }
        if root.Left != nil {
                m[root.Left.Val] = root
                dfs(root.Left)
        }
        if root.Right != nil {
                m[root.Right.Val] = root
                dfs(root.Right)
        }
}
```

8.19 238. 除自身以外数组的乘积 (3)

- 题目

```
给你一个长度为 n 的整数数组 nums，其中 n > 1，返回输出数组 output ，
其中 output[i] 等于 nums 中除 nums[i] 之外其余各元素的乘积。
示例:输入: [1,2,3,4] 输出: [24,12,8,6]
提示：题目数据保证数组之中任意元素的全部前缀元素和后缀（甚至是整个数组）的乘积都在 32␣
↪位整数范围内。
说明：请不要使用除法，且在 O(n) 时间复杂度内完成此题。
进阶：你可以在常数空间复杂度内完成这个题目吗？（␣
↪出于对空间复杂度分析的目的，输出数组不被视为额外空间。）
```

- 解题思路

```
func productExceptSelf(nums []int) []int {
        left := make([]int, len(nums))
        right := make([]int, len(nums))
        res := make([]int, 0)
        left[0] = 1
        right[len(nums)-1] = 1
        for i := 1; i < len(nums); i++ {
                left[i] = left[i-1] * nums[i-1]
        }
        for i := len(nums) - 2; i >= 0; i-- {
                right[i] = right[i+1] * nums[i+1]
        }
        for i := 0; i < len(nums); i++ {
                res = append(res, left[i]*right[i])
        }
        return res
```

(续下页)

(接上页)

```
}

# 2
func productExceptSelf(nums []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                value := 1
                for j := 0; j < len(nums); j++ {
                        if i != j {
                                value = value * nums[j]
                        }
                }
                res = append(res, value)
        }
        return res
}

# 3
func productExceptSelf(nums []int) []int {
        res := make([]int, len(nums))
        res[0] = 1
        for i := 1; i < len(nums); i++ {
                res[i] = res[i-1] * nums[i-1]
        }
        value := 1
        for i := len(nums) - 1; i >= 0; i-- {
                res[i] = res[i] * value
                value = value * nums[i]
        }
        return res
}
```

8.20 240. 搜索二维矩阵 II(6)

- 题目

```
编写一个高效的算法来搜索 m x n 矩阵 matrix 中的一个目标值 target。该矩阵具有以下特性:
    每行的元素从左到右升序排列。
    每列的元素从上到下升序排列。
示例:现有矩阵 matrix 如下:
[
  [1,   4,  7, 11, 15],
```

(续下页)

(接上页)

```
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
给定 target = 5, 返回 true。
给定 target = 20, 返回 false。
```

- 解题思路

```
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == target {
                                return true
                        }
                }
        }
        return false
}

# 2
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        for j := 0; j < len(matrix[i]); j++ {
                                if matrix[i][j] == target {
                                        return true
                                }
                        }
                }
        }
```

(续下页)

(接上页)

```
        return false
}

# 3
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        res := binarySearch(matrix[i], target)
                        if res == true {
                                return true
                        }
                }
        }
        return false
}

func binarySearch(arr []int, target int) bool {
        left := 0
        right := len(arr) - 1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        return true
                } else if arr[mid] > target {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return false
}

# 4
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
```

(续下页)

(接上页)

```
        if len(matrix[0]) == 0 {
                return false
        }
        i := len(matrix) - 1
        j := 0
        for i >= 0 && j < len(matrix[0]) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        i--
                } else {
                        j++
                }
        }
        return false
}

# 5
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := 0
        j := len(matrix[0]) - 1
        for j >= 0 && i < len(matrix) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        j--
                } else {
                        i++
                }
        }
        return false
}

# 6
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
```

(续下页)

(接上页)

```
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                index := sort.SearchInts(matrix[i], target)
                if index < len(matrix[i]) && target == matrix[i][index] {
                        return true
                }
        }
        return false
}
```

8.21 241. 为运算表达式设计优先级 (2)

- 题目

```
给定一个含有数字和运算符的字符串，为表达式添加括号，改变其运算优先级以求出不同的结果。
你需要给出所有可能的组合的结果。有效的运算符号包含 +, - 以及 * 。

示例 1:输入: "2-1-1" 输出: [0, 2]
解释:
((2-1)-1) = 0
(2-(1-1)) = 2
示例 2:输入: "2*3-4*5" 输出: [-34, -14, -10, -10, 10]
解释:
(2*(3-(4*5))) = -34
((2*3)-(4*5)) = -14
((2*(3-4))*5) = -10
(2*((3-4)*5)) = -10
(((2*3)-4)*5) = 10
```

- 解题思路

```
func diffWaysToCompute(input string) []int {
        if value, err := strconv.Atoi(input); err == nil {
                return []int{value}
        }
        res := make([]int, 0)
        for i := 0; i < len(input); i++ {
                char := string(input[i])
                if char == "+" || char == "-" || char == "*" {
```

(续下页)

(接上页)

```
                        left := diffWaysToCompute(input[:i])
                        right := diffWaysToCompute(input[i+1:])
                        for _, leftNum := range left {
                                for _, rightNum := range right {
                                        temp := 0
                                        if char == "+" {
                                                temp = leftNum + rightNum
                                        } else if char == "-" {
                                                temp = leftNum - rightNum
                                        } else if char == "*" {
                                                temp = leftNum * rightNum
                                        }
                                        res = append(res, temp)
                                }
                        }
                }
        }
        return res
}

# 2
func diffWaysToCompute(input string) []int {
        if value, err := strconv.Atoi(input); err == nil {
                return []int{value}
        }
        numArr := make([]int, 0)
        opArr := make([]byte, 0)
        num := 0
        for i := 0; i < len(input); i++ {
                if input[i] == '+' || input[i] == '-' || input[i] == '*' {
                        opArr = append(opArr, input[i])
                        numArr = append(numArr, num)
                        num = 0
                        continue
                }
                num = num*10 + int(input[i]-'0')
        }
        numArr = append(numArr, num)
        n := len(numArr)
        dp := make([][][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([][]int, n)
                arr := make([]int, 0)
```

(续下页)

(接上页)

```
                arr = append(arr, numArr[i])
                dp[i][i] = arr
        }
        for k := 2; k <= n; k++ { // 长度
                for i := 0; i < n; i++ { // 起点
                        j := i + k - 1 // 终点
                        if j >= n {
                                break
                        }
                        temp := make([]int, 0)
                        for l := i; l < j; l++ { // 切割点
                                left := dp[i][l]
                                right := dp[l+1][j]
                                for a := 0; a < len(left); a++ {
                                        for b := 0; b < len(right); b++ {
                                                op := opArr[l]
                                                if op == '+' {
                                                        temp = append(temp,␣
↪left[a]+right[b])
                                                } else if op == '-' {
                                                        temp = append(temp, left[a]-
↪right[b])
                                                } else if op == '*' {
                                                        temp = append(temp,␣
↪left[a]*right[b])
                                                }
                                        }
                                }
                        }
                        dp[i][j] = temp
                }
        }
        return dp[0][n-1]
}
```

8.22 260. 只出现一次的数字 III(3)

- 题目

```
给定一个整数数组 nums，其中恰好有两个元素只出现一次，其余所有元素均出现两次。␣
↪找出只出现一次的那两个元素。
示例 :
输入: [1,2,1,3,2,5]
输出: [3,5]
注意:
    结果输出的顺序并不重要，对于上面的例子， [5, 3] 也是正确答案。
    你的算法应该具有线性时间复杂度。你能否仅使用常数空间复杂度来实现？
```

- 解题思路

```
func singleNumber(nums []int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for k, v := range m {
                if v == 1 {
                        res = append(res, k)
                }
        }
        return res
}

# 2
func singleNumber(nums []int) []int {
        a := 0
        for i := 0; i < len(nums); i++ {
                a = a ^ nums[i]
        }
        b := a & (-a) // x & (-x) 是保留位中最右边 1 ，且将其余的 1 设位 0 的方法。
        value := 0
        for i := 0; i < len(nums); i++ {
                if nums[i]&b == 0 {
                        value = value ^ nums[i]
                }
        }
        return []int{value, a ^ value}
}
```

(续下页)

(接上页)

```
# 3
func singleNumber(nums []int) []int {
        a := 0
        for i := 0; i < len(nums); i++ {
                a = a ^ nums[i]
        }
        b := 1
        for a&1 == 0 {
                a = a >> 1
                b = b << 1
        }
        res := []int{0, 0}
        for i := 0; i < len(nums); i++ {
                if nums[i]&b == 0 {
                        res[0] = res[0] ^ nums[i]
                } else {
                        res[1] = res[1] ^ nums[i]
                }
        }
        return res
}
```

8.23 264. 丑数 II(1)

- 题目

```
编写一个程序，找出第 n 个丑数。
丑数就是质因数只包含 2, 3, 5 的正整数。
示例:输入: n = 10 输出: 12
解释: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 是前 10 个丑数。
说明:
    1 是丑数。
    n 不超过1690。
```

- 解题思路

```
func nthUglyNumber(n int) int {
        dp := make([]int, n)
        dp[0] = 1
        // 丑数*2或3或5之后还是丑数
        idx2, idx3, idx5 := 0, 0, 0
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                dp[i] = min(dp[idx2]*2, min(dp[idx3]*3, dp[idx5]*5))
                if dp[i] == dp[idx2]*2 {
                        idx2++
                }
                if dp[i] == dp[idx3]*3 {
                        idx3++
                }
                if dp[i] == dp[idx5]*5 {
                        idx5++
                }
        }
        return dp[n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

8.24 274.H 指数 (2)

- 题目

```
给定一位研究者论文被引用次数的数组（被引用次数是非负整数）。编写一个方法，计算出研究者的h 指数。
h 指数的定义：h 代表“高引用次数”（high citations），
一名科研人员的 h 指数是指他（她）的 （N 篇论文中）总共有 h 篇论文分别被引用了至少 h次。
（其余的 N - h 篇论文每篇被引用次数 不超过 h 次。）
例如：某人的 h 指数是 20，这表示他已发表的论文中，每篇被引用了至少 20 次的论文总共有20 篇。
示例：输入：citations = [3,0,6,1,5]输出：3
解释：给定数组表示研究者总共有 5 篇论文，每篇论文相应的被引用了 3, 0, 6, 1, 5 次。
     由于研究者有 3 篇论文每篇 至少 被引用了 3 次，其余两篇论文每篇被引用 不多于 3次，
     所以她的 h 指数是 3。
提示：如果 h 有多种可能的值，h 指数是其中最大的那个。
```

- 解题思路

```
func hIndex(citations []int) int {
        sort.Ints(citations)
        for i := 0; i < len(citations); i++ {
                if citations[i] >= len(citations)-i {
                        return len(citations) - i
                }
        }
        return 0
}

#
func hIndex(citations []int) int {
        arr := make([]int, len(citations)+1)
        for i := 0; i < len(citations); i++ {
                if citations[i] >= len(citations) {
                        arr[len(citations)]++
                } else {
                        arr[citations[i]]++
                }
        }
        count := 0
        for i := len(citations); i >= 0; i-- {
                count = count + arr[i]
                if count >= i {
                        return i
                }
        }
        return 0
}
```

8.25 275.H 指数 II(2)

- 题目

```
给定一位研究者论文被引用次数的数组（被引用次数是非负整数），数组已经按照升序排列。
编写一个方法，计算出研究者的 h 指数。
h 指数的定义: “h 代表“高引用次数” (high citations)，
一名科研人员的 h 指数是指他（她）的 （N 篇论文中）总共有 h 篇论文分别被引用了至少 h
↪次。
（其余的 N - h 篇论文每篇被引用次数不多于 h 次。）"
示例:输入: citations = [0,1,3,5,6] 输出: 3
解释: 给定数组表示研究者总共有 5 篇论文，每篇论文相应的被引用了 0, 1, 3, 5, 6 次。
```

(续下页)

(接上页)

```
    由于研究者有 3 篇论文每篇至少被引用了 3 次，其余两篇论文每篇被引用不多于 3 次，
    所以她的 h 指数是 3。
说明:如果 h 有多有种可能的值 ，h 指数是其中最大的那个。
进阶：
    这是 H指数 的延伸题目，本题中的 citations 数组是保证有序的。
    你可以优化你的算法到对数时间复杂度吗？
```

- 解题思路

```
func hIndex(citations []int) int {
        for i := 0; i < len(citations); i++ {
                if citations[i] >= len(citations)-i {
                        return len(citations) - i
                }
        }
        return 0
}

#
func hIndex(citations []int) int {
        left := 0
        right := len(citations) - 1
        for left <= right {
                mid := left + (right-left)/2
                if citations[mid] == len(citations)-mid {
                        return len(citations) - mid
                } else if citations[mid] > len(citations)-mid {
                        right = mid - 1
                } else if citations[mid] < len(citations)-mid {
                        left = mid + 1
                }
        }
        return len(citations) - left
}
```

8.26 279. 完全平方数 (5)

- 题目

```
给定正整数 n，找到若干个完全平方数（比如 1，4，9，16，...）使得它们的和等于 n。
你需要让组成和的完全平方数的个数最少。
示例 1:输入：n = 12 输出：3
```

(续下页)

(接上页)

```
解释: 12 = 4 + 4 + 4.
示例 2:输入: n = 13 输出: 2
解释: 13 = 4 + 9.
```

- 解题思路

```
func numSquares(n int) int {
        dp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                dp[i] = math.MaxInt32
        }
        arr := make([]int, 0)
        arr = append(arr, 0)
        for i := 1; i*i <= n; i++ {
                arr = append(arr, i*i)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j*j <= i; j++ {
                        if i < arr[j] {
                                break
                        }
                        dp[i] = min(dp[i], dp[i-arr[j]]+1)
                }
        }
        return dp[n]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func numSquares(n int) int {
        dp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                dp[i] = i
                for j := 1; j*j <= i; j++ {
                        dp[i] = min(dp[i], dp[i-j*j]+1)
                }
        }
        return dp[n]
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func numSquares(n int) int {
        if n == 0 {
                return 0
        }
        list := make([]int, 0)
        list = append(list, n)
        level := 0
        for len(list) > 0 {
                level++
                length := len(list)
                for i := 0; i < length; i++ {
                        value := list[i]
                        for j := 1; j*j <= value; j++ {
                                if j*j == value {
                                        return level
                                }
                                list = append(list, value-j*j)
                        }
                }
                list = list[length:]
        }
        return level
}

# 4
var m map[int]int

func numSquares(n int) int {
        m = make(map[int]int)
        return dfs(n)
}

func dfs(n int) int {
```

(续下页)

(接上页)

```
        if m[n] > 0 {
                return m[n]
        }
        if n == 0 {
                return 0
        }
        count := math.MaxInt32
        for i := 1; i*i <= n; i++ {
                count = min(count, dfs(n-i*i)+1)
        }
        return count
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 5
func numSquares(n int) int {
        if judge(n) {
                return 1
        }
        res := n
        for res%4 == 0 {
                res = res / 4
        }
        if res%8 == 7 {
                return 4
        }
        for i := 1; i*i < n; i++ {
                if judge(n - i*i) {
                        return 2
                }
        }
        return 3
}

func judge(n int) bool {
        value := int(math.Sqrt(float64(n)))
        return value*value == n
}
```

8.27 284. 顶端迭代器 (2)

- 题目

```
给定一个迭代器类的接口，接口包含两个方法： next() 和 hasNext()。
设计并实现一个支持 peek() 操作的顶端迭代器 -- 其本质就是把原本应由 next()␣
↪方法返回的元素 peek() 出来。
示例:假设迭代器被初始化为列表 [1,2,3]。
调用 next() 返回 1，得到列表中的第一个元素。
现在调用 peek() 返回 2，下一个元素。在此之后调用 next() 仍然返回 2。
最后一次调用 next() 返回 3，末尾元素。在此之后调用 hasNext() 应该返回 false。
进阶：你将如何拓展你的设计？使之变得通用化，从而适应所有的类型，而不只是整数型？
```

- 解题思路

```
type PeekingIterator struct {
        Iter    *Iterator
        cache   int
        isCache bool
}

func Constructor(iter *Iterator) *PeekingIterator {
        return &PeekingIterator{
                Iter:    iter,
                cache:   0,
                isCache: false,
        }
}

func (this *PeekingIterator) hasNext() bool {
        return this.isCache || this.Iter.hasNext()
}

func (this *PeekingIterator) next() int {
        if this.isCache == false {
                return this.Iter.next()
        }
        res := this.cache
        this.isCache = false
        return res
}

func (this *PeekingIterator) peek() int {
        if this.isCache == false {
```

(续下页)

(接上页)

```
                this.cache = this.Iter.next()
                this.isCache = true
        }
        return this.cache
}

# 2
type PeekingIterator struct {
        Iter  *Iterator
        cache *int
}

func Constructor(iter *Iterator) *PeekingIterator {
        return &PeekingIterator{
                Iter:  iter,
                cache: nil,
        }
}

func (this *PeekingIterator) hasNext() bool {
        return this.cache != nil || this.Iter.hasNext()
}

func (this *PeekingIterator) next() int {
        if this.cache != nil {
                res := *this.cache
                this.cache = nil
                return res
        }
        return this.Iter.next()
}

func (this *PeekingIterator) peek() int {
        if this.cache == nil {
                value := this.Iter.next()
                this.cache = &value
        }
        return *this.cache
}
```

8.28 287. 寻找重复数 (8)

- 题目

```
给定一个包含 n + 1 个整数的数组 nums，其数字都在 1 到 n 之间（包括 1 和 n），
可知至少存在一个重复的整数。假设只有一个重复的整数，找出这个重复的数。
示例 1:输入: [1,3,4,2,2] 输出: 2
示例 2:输入: [3,1,3,4,2] 输出: 3
说明:
    不能更改原数组（假设数组是只读的）。
    只能使用额外的 O(1) 的空间。
    时间复杂度小于 O(n^2) 。
    数组中只有一个重复的数字，但它可能不止重复出现一次。
```

- 解题思路

```
func findDuplicate(nums []int) int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] > 0 {
                        return nums[i]
                }
                m[nums[i]] = 1
        }
        return -1
}

# 2
func findDuplicate(nums []int) int {
        sort.Ints(nums)
        for i := 1; i < len(nums); i++{
                if nums[i] == nums[i-1]{
                        return nums[i]
                }
        }
        return -1
}

# 3
func findDuplicate(nums []int) int {
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i] == nums[j] {
                                return nums[i]
```

(续下页)

(接上页)

```
                        }
                }
        }
        return -1
}

# 4
func findDuplicate(nums []int) int {
        left, right := 1, len(nums)-1
        res := -1
        for left <= right {
                mid := left + (right-left)/2
                count := 0
                for i := 0; i < len(nums); i++ {
                        if nums[i] <= mid {
                                count++
                        }
                }
                if count <= mid {
                        left = mid + 1
                } else {
                        right = mid - 1
                        res = mid
                }
        }
        return res
}

# 5
func findDuplicate(nums []int) int {
        slow, fast := nums[0], nums[nums[0]]
        for slow != fast {
                slow, fast = nums[slow], nums[nums[fast]]
        }
        slow = 0
        for slow != fast {
                slow, fast = nums[slow], nums[fast]
        }
        return slow
}

# 6
func findDuplicate(nums []int) int {
```

(续下页)

(接上页)

```
        arrV := [32]int{}
        arrI := [32]int{}
        res := 0
        for i := 0; i < len(nums); i++ {
                value := nums[i]
                index := i
                for j := 0; j < 32; j++ {
                        if value&(1<<j) > 0 {
                                arrV[j]++
                        }
                        if index > 0 && (index&(1<<j) > 0) {
                                arrI[j]++
                        }
                }
        }

        for i := 0; i < len(arrV); i++ {
                if arrV[i] > arrI[i] {
                        res = res ^ (1 << i)
                }
        }
        return res
}

# 7
func findDuplicate(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        for i := 1; i <= len(nums)-1; i++ {
                for i != nums[i] {
                        value := nums[i]
                        if value == nums[value] {
                                return value
                        }
                        nums[i], nums[value] = nums[value], nums[i]
                }
        }
        return nums[0]
}

# 8
func findDuplicate(nums []int) int {
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                index := abs(nums[i]) - 1
                if nums[index] > 0 {
                        nums[index] = -1 * nums[index]
                } else {
                        return abs(nums[i])
                }
        }
        return 0
}

func abs(a int) int {
        if a >= 0 {
                return a
        }
        return -a
}
```

8.29 289. 生命游戏 (2)

- 题目

```
根据 百度百科 ，生命游戏，简称为生命，是英国数学家约翰·何顿·康威在 1970␣
↪年发明的细胞自动机。
给定一个包含 m × n␣
↪个格子的面板，每一个格子都可以看成是一个细胞。每个细胞都具有一个初始状态：
1 即为活细胞（live），或 0 即为死细胞（dead）。
每个细胞与其八个相邻位置（水平，垂直，对角线）的细胞都遵循以下四条生存定律：
    如果活细胞周围八个位置的活细胞数少于两个，则该位置活细胞死亡；
    如果活细胞周围八个位置有两个或三个活细胞，则该位置活细胞仍然存活；
    如果活细胞周围八个位置有超过三个活细胞，则该位置活细胞死亡；
    如果死细胞周围正好有三个活细胞，则该位置死细胞复活；
根据当前状态，写一个函数来计算面板上所有细胞的下一个（一次更新后的）状态。
下一个状态是通过将上述规则同时应用于当前状态下的每个细胞所形成的，其中细胞的出生和死亡是同时发生的。
示例：输入：
[
  [0,1,0],
  [0,0,1],
  [1,1,1],
  [0,0,0]
]
输出：
```

(续下页)

(接上页)

```
[
  [0,0,0],
  [1,0,1],
  [0,1,1],
  [0,1,0]
]
进阶：
    你可以使用原地算法解决本题吗？请注意，面板上所有格子需要同时被更新：
    你不能先更新某些格子，然后使用它们的更新后的值再更新其他格子。
    ␣
↪本题中，我们使用二维数组来表示面板。原则上，面板是无限的，但当活细胞侵占了面板边界时会造成问题。
    你将如何解决这些问题？
```

- 解题思路

```
func gameOfLife(board [][]int) {
        temp := make([][]int, len(board))
        for i := 0; i < len(board); i++ {
                temp[i] = make([]int, len(board[i]))
        }
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        count := 0
                        for a := i - 1; a < i+1; a++ {
                                for b := j - 1; b < j+1; b++ {
                                        if 0 <= a && a < len(board) &&
                                                0 <= b && b < len(board[i]) &&␣
↪board[i][j] == 1 {
                                                count++
                                        }
                                }
                        }
                        if count < 2 || count > 3 {
                                temp[i][j] = 0
                        }
                        if count == 3 && temp[i][j] == 0 {
                                temp[i][j] = 1
                        }
                }
        }
        copy(board, temp)
}

# 2
```

(续下页)

(接上页)

```
func gameOfLife(board [][]int) {
        // 0 00 => 死
        // 1 01 => 活
        // 2 10 => 死=>活
        // 3 11 => 活=>死
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        count := 0
                        for a := i - 1; a <= i+1; a++ {
                                for b := j - 1; b <= j+1; b++ {
                                        if a == i && b == j {
                                                continue
                                        }
                                        if 0 <= a && a < len(board) &&
                                                0 <= b && b < len(board[i]) {
                                                count = count + board[a][b]%2
                                        }
                                }
                        }
                        if (count < 2 || count > 3) && board[i][j] == 1 {
                                board[i][j] = 3
                        }
                        if count == 3 && board[i][j] == 0 {
                                board[i][j] = 2
                        }
                }
        }
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        if board[i][j] == 2 {
                                board[i][j] = 1
                        } else if board[i][j] == 3 {
                                board[i][j] = 0
                        }
                }
        }
}
```

8.30 300. 最长上升子序列 (2)

- 题目

```
给定一个无序的整数数组，找到其中最长上升子序列的长度。
示例:输入: [10,9,2,5,3,7,101,18] 输出: 4
解释: 最长的上升子序列是 [2,3,7,101]，它的长度是 4。
说明:可能会有多种最长上升子序列的组合，你只需要输出对应的长度即可。
    你算法的时间复杂度应该为 O(n2) 。
进阶: 你能将算法的时间复杂度降低到 O(n log n) 吗?
```

- 解题思路

```
/*
dp[i] = max(dp[j]+1, dp[i]),其中0<=j<i, nums[j]<nums[i])
*/
func lengthOfLIS(nums []int) int {
        if len(nums) < 2 {
                return len(nums)
        }
        dp := make([]int, len(nums))
        res := 1
        for i := 0; i < len(nums); i++ {
                dp[i] = 1
                for j := 0; j < i; j++ {
                        if nums[j] < nums[i] {
                                dp[i] = max(dp[j]+1, dp[i])
                        }
                }
                res = max(res, dp[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func lengthOfLIS(nums []int) int {
        if len(nums) < 2 {
```

(续下页)

(接上页)

```
                return len(nums)
        }
        arr := make([]int, len(nums)+1)
        arr[1] = nums[0]
        res := 1
        for i := 1; i < len(nums); i++ {
                if arr[res] < nums[i] {
                        res++
                        arr[res] = nums[i]
                } else {
                        left, right := 1, res
                        index := 0
                        for left <= right {
                                mid := left + (right-left)/2
                                if arr[mid] < nums[i] {
                                        index = mid
                                        left = mid + 1
                                } else {
                                        right = mid - 1
                                }
                        }
                        arr[index+1] = nums[i]
                }
        }
        return res
}
```

CHAPTER 9

0201-0300-Hard

9.1 214. 最短回文串 (3)

- 题目

```
给定一个字符串 s，你可以通过在字符串前面添加字符将其转换为回文串。
找到并返回可以用这种方式转换的最短回文串。
示例 1:输入: "aacecaaa" 输出: "aaacecaaa"
示例 2:输入: "abcd" 输出: "dcbabcd"
```

- 解题思路

```
func shortestPalindrome(s string) string {
        str := reverse(s)
        i := 0
        for i = 0; i < len(s); i++ {
                if str[i:] == s[:len(s)-i] {
                        break
                }
        }
        return str[:i] + s
}

func reverse(s string) string {
        res := make([]byte, 0)
```

(续下页)

(接上页)

```
        for i := len(s) - 1; i >= 0; i-- {
                res = append(res, s[i])
        }
        return string(res)
}

# 2
func shortestPalindrome(s string) string {
        i := len(s)
        for {
                if isPalindrome(s[:i]) == true {
                        break
                }
                i--
        }
        res := s
        for j := i; j < len(s); j++ {
                res = string(s[j]) + res
        }
        return res
}

func isPalindrome(s string) bool {
        i, j := 0, len(s)-1
        for i < j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

# 3
func shortestPalindrome(s string) string {
        str := add(s)
        index := 0
        for i := len(str) / 2; i <= len(str)/2; i-- {
                j := i
                for ; j > 0; j-- {
                        if str[i-j] != str[i+j] {
                                break
```

(续下页)

(接上页)

```
                        }
                }
                if j == 0 {
                        index = i
                        break
                }
        }
        res := s
        for j := index; j < len(s); j++ {
                res = string(s[j]) + res
        }
        return res
}

func add(s string) string {
        var res []rune
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        return string(res)
}
```

9.2 218. 天际线问题

9.2.1 题目

```
```

9.2.2 解题思路

9.3 224. 基本计算器 (1)

- 题目

```
实现一个基本的计算器来计算一个简单的字符串表达式 s 的值。
示例 1: 输入: s = "1 + 1" 输出: 2
示例 2: 输入: s = " 2-1 + 2 " 输出: 3
示例 3: 输入: s = "(1+(4+5+2)-3)+(6+8)" 输出: 23
提示: 1 <= s.length <= 3* 105
s 由数字、'+'、'-'、'('、')'、和 ' ' 组成
s 表示一个有效的表达式
```

- 解题思路

```go
func calculate(s string) int {
        stack := make([]int, 0)
        num := 0
        res := 0
        sign := 1
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        num = 0
                        for ; i < len(s) && '0' <= s[i] && s[i] <= '9'; i++ {
                                num = num*10 + int(s[i]-'0')
                        }
                        res = res + sign*num
                        i--
                } else if s[i] == '+' {
                        sign = 1
                } else if s[i] == '-' {
                        sign = -1
                } else if s[i] == '(' {
                        stack = append(stack, res, sign)
                        res = 0
                        sign = 1
                } else if s[i] == ')' {
                        sign = stack[len(stack)-1]
                        prev := stack[len(stack)-2]
                        stack = stack[:len(stack)-2]
```

(续下页)

(接上页)

```
                        res = prev + sign*res*sign
                }
        }
        return res
}
```

9.4 233. 数字 1 的个数 (3)

- 题目

```
给定一个整数 n，计算所有小于等于 n 的非负整数中数字 1 出现的个数。
示例:输入: 13输出: 6
解释: 数字 1 出现在以下数字中: 1, 10, 11, 12, 13 。
```

- 解题思路

```
func countDigitOne(n int) int {
        res := 0
        digit := 1
        high := n / 10
        cur := n % 10
        low := 0
        for high != 0 || cur != 0 {
                if cur == 0 {
                        res = res + high*digit
                } else if cur == 1 {
                        res = res + high*digit + low + 1
                } else {
                        res = res + (high+1)*digit
                }
                low = low + cur*digit
                cur = high % 10
                high = high / 10
                digit = digit * 10
        }
        return res
}

# 2
func countDigitOne(n int) int {
        if n <= 0 {
                return 0
```

(续下页)

(接上页)

```
        }
        str := strconv.Itoa(n)
        return dfs(str)
}

func dfs(str string) int {
        if str == "" {
                return 0
        }
        first := int(str[0] - '0')
        if len(str) == 1 && first == 0 {
                return 0
        }
        if len(str) == 1 && first >= 1 {
                return 1
        }
        count := 0
        if first > 1 {
                count = int(math.Pow(float64(10), float64(len(str)-1)))
        } else if first == 1 {
                count, _ = strconv.Atoi(str[1:])
                count = count + 1
        }
        other := first * (len(str) - 1) * int(math.Pow(float64(10), float64(len(str)-
↪2)))
        numLeft := dfs(str[1:])
        return count + numLeft + other
}

# 3
func countDigitOne(n int) int {
        if n <= 0 {
                return 0
        }
        res := 0
        for i := 1; i <= n; i = i * 10 {
                left := n / i
                right := n % i
                res = res + (left+8)/10*i
                if left%10 == 1 {
                        res = res + right + 1
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

9.5 239. 滑动窗口最大值 (3)

- 题目

```
给定一个数组 nums，有一个大小为 k 的滑动窗口从数组的最左侧移动到数组的最右侧。
你只可以看到在滑动窗口内的 k 个数字。滑动窗口每次只向右移动一位。
返回滑动窗口中的最大值。
进阶：你能在线性时间复杂度内解决此题吗？
示例:输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3 输出: [3,3,5,5,6,7]
解释:
  滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
提示：
    1 <= nums.length <= 10^5
    -10^4 <= nums[i] <= 10^4
    1 <= k <= nums.length
```

- 解题思路

```
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        for i := 0; i < len(nums)-k+1; i++ {
                max := nums[i]
                for j := i; j < i+k; j++ {
                        if nums[j] > max {
                                max = nums[j]
                        }
                }
                res = append(res, max)
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        max := math.MaxInt32
        for i := 0; i < len(nums)-k+1; i++ {
                if i == 0 || nums[i-1] == max {
                        max = nums[i]
                        for j := i; j < i+k; j++ {
                                if nums[j] > max {
                                        max = nums[j]
                                }
                        }
                } else {
                        if nums[i+k-1] > max {
                                max = nums[i+k-1]
                        }
                }
                res = append(res, max)
        }
        return res
}

# 3
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        // 递减下标
        deque := make([]int, 0)
        for i := 0; i < k; i++ {
                for len(deque) > 0 && nums[i] >= nums[deque[len(deque)-1]] {
                        deque = deque[:len(deque)-1]
                }
                deque = append(deque, i)
        }
        for i := k; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                res = append(res, nums[deque[0]])
                for len(deque) > 0 && nums[i] >= nums[deque[len(deque)-1]] {
                        deque = deque[:len(deque)-1]
                }
                if len(deque) > 0 && deque[0] <= i-k {
                        deque = deque[1:]
                }
                deque = append(deque, i)
        }
        res = append(res, nums[deque[0]])
        return res
}

# 4
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        intHeap := make(IntHeap, 0, k)
        heap.Init(&intHeap)
        for i := 0; i < k; i++ {
                heap.Push(&intHeap, nums[i])
        }
        for i := k; i < len(nums); i++ {
                temp := heap.Pop(&intHeap).(int)
                res = append(res, temp)
                if temp != nums[i-k] {
                        intHeap.Remove(nums[i-k])
                        heap.Push(&intHeap, temp)
                        heap.Push(&intHeap, nums[i])
                } else {
                        heap.Push(&intHeap, nums[i])
                }
        }
        res = append(res, heap.Pop(&intHeap).(int))
        return res
}

type IntHeap []int

func (i IntHeap) Len() int {
        return len(i)
```

(续下页)

(接上页)

```
}

func (i IntHeap) Less(x, y int) bool {
        return i[x] > i[y]
}

func (i IntHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *IntHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *IntHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

func (i *IntHeap) Remove(x interface{}) {
        for j := 0; j < len(*i); j++ {
                if (*i)[j] == x {
                        *i = append((*i)[:j], (*i)[j+1:]...)
                        break
                }
        }
        heap.Init(i)
}
```

9.6 273. 整数转换英文表 (3)

9.6.1 题目

```
将非负整数转换为其对应的英文表示。可以保证给定输入小于 2^31 - 1 。
示例 1:输入: 123 输出: "One Hundred Twenty Three"
示例 2:输入: 12345 输出: "Twelve Thousand Three Hundred Forty Five"
示例 3:输入: 1234567
输出: "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
示例 4:输入: 1234567891
输出: "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven
Thousand Eight Hundred Ninety One"
```

9.6.2 解题思路

```
func numberToWords(num int) string {
        if num == 0 {
                return "Zero"
        }
        res := ""
        billion := num / 1000000000
        million := (num - billion*1000000000) / 1000000
        thousand := (num - billion*1000000000 - million*1000000) / 1000
        left := num - billion*1000000000 - million*1000000 - thousand*1000
        if billion != 0 {
                res += three(billion) + " Billion"
        }
        if million != 0 {
                if res != "" {
                        res += " "
                }
                res += three(million) + " Million"
        }
        if thousand != 0 {
                if res != "" {
                        res += " "
                }
                res += three(thousand) + " Thousand"
        }
        if left != 0 {
                if res != "" {
                        res += " "
                }
                res += three(left)
        }
        return res
}

func three(num int) string {
        hundred := num / 100
        left := num - hundred*100
        if hundred == 0 {
                return two(num)
        }
        res := transfer[hundred] + " Hundred"
        if left != 0 {
                res += " " + two(left)
```

(续下页)

(接上页)

```
        }
        return res
}

func two(num int) string {
        if num == 0 {
                return ""
        } else if num < 10 {
                return transfer[num]
        } else if num < 20 {
                return transfer[num]
        }
        ten := num / 10
        left := num - ten*10
        ten = ten * 10
        res := transfer[ten]
        if left != 0 {
                res += " " + transfer[left]
        }
        return res
}

var transfer = map[int]string{
        0:  "Zero",
        1:  "One",
        2:  "Two",
        3:  "Three",
        4:  "Four",
        5:  "Five",
        6:  "Six",
        7:  "Seven",
        8:  "Eight",
        9:  "Nine",
        10: "Ten",
        11: "Eleven",
        12: "Twelve",
        13: "Thirteen",
        14: "Fourteen",
        15: "Fifteen",
        16: "Sixteen",
        17: "Seventeen",
        18: "Eighteen",
        19: "Nineteen",
```

(续下页)

(接上页)

```
        20: "Twenty",
        30: "Thirty",
        40: "Forty",
        50: "Fifty",
        60: "Sixty",
        70: "Seventy",
        80: "Eighty",
        90: "Ninety",
}

# 2
func numberToWords(num int) string {
        if num == 0 {
                return "Zero"
        }
        return strings.Trim(dfs(num), " ")
}

func dfs(n int) string {
        if n < 20 {
                return transfer[n]
        }
        if n < 100 {
                return transfer[n/10*10] + dfs(n%10)
        }
        if n < 1000 {
                return transfer[n/100] + "Hundred " + dfs(n%100)
        }
        if n < 1000000 {
                return dfs(n/1000) + "Thousand " + dfs(n%1000)
        }
        if n < 1000000000 {
                return dfs(n/1000000) + "Million " + dfs(n%1000000)
        }
        return dfs(n/1000000000) + "Billion " + dfs(n%1000000000)
}

var transfer = map[int]string{
        1:  "One ",
        2:  "Two ",
        3:  "Three ",
        4:  "Four ",
        5:  "Five ",
```

(续下页)

(接上页)

```
        6:  "Six ",
        7:  "Seven ",
        8:  "Eight ",
        9:  "Nine ",
        10: "Ten ",
        11: "Eleven ",
        12: "Twelve ",
        13: "Thirteen ",
        14: "Fourteen ",
        15: "Fifteen ",
        16: "Sixteen ",
        17: "Seventeen ",
        18: "Eighteen ",
        19: "Nineteen ",
        20: "Twenty ",
        30: "Thirty ",
        40: "Forty ",
        50: "Fifty ",
        60: "Sixty ",
        70: "Seventy ",
        80: "Eighty ",
        90: "Ninety ",
}
```

9.7 282. 给表达式添加运算符 (2)

- 题目

```
给定一个仅包含数字0-9的字符串和一个目标值，在数字之间添加 二元 运算符（不是一元）+、-
↪或*，
返回所有能够得到目标值的表达式。
示例 1:输入: num = "123", target = 6 输出: ["1+2+3", "1*2*3"]
示例2:输入: num = "232", target = 8 输出: ["2*3+2", "2+3*2"]
示例 3:输入: num = "105", target = 5 输出: ["1*0+5","10-5"]
示例4:输入: num = "00", target = 0 输出: ["0+0", "0-0", "0*0"]
示例 5:输入: num = "3456237490", target = 9191 输出: []
提示: 0 <= num.length <= 10
num 仅含数字
```

- 解题思路

```
var res []string

func addOperators(num string, target int) []string {
        res = make([]string, 0)
        dfs(num, target, 0, "", 0, 0)
        return res
}

func dfs(num string, target int, index int, str string, value int, prev int) {
        if index == len(num) {
                if value == target {
                        res = append(res, str)
                }
                return
        }
        for i := index; i < len(num); i++ {
                if num[index] == '0' && index < i { // 105 5 => 1*05不符合要求
                        return
                }
                s := num[index : i+1]
                a, _ := strconv.Atoi(s)
                if index == 0 {
                        dfs(num, target, i+1, str+s, a, a)
                } else {
                        dfs(num, target, i+1, str+"+"+s, value+a, a)
                        dfs(num, target, i+1, str+"-"+s, value-a, -a)
                        dfs(num, target, i+1, str+"*"+s, value-prev+prev*a, prev*a)
                }
        }
}

# 2
var res []string

func addOperators(num string, target int) []string {
        res = make([]string, 0)
        dfs(num, target, 0, "")
        return res
}

func dfs(num string, target int, index int, str string) {
        if index == len(num) {
                // 全排列再计算
                if calculate(str) == target {
```

(续下页)

(接上页)

```
                        res = append(res, str)
                }
                return
        }
        for i := index; i < len(num); i++ {
                if num[index] == '0' && index < i { // 105 5 => 1*05不符合要求
                        return
                }
                s := num[index : i+1]
                if index == 0 {
                        dfs(num, target, i+1, str+s)
                } else {
                        dfs(num, target, i+1, str+"+"+s)
                        dfs(num, target, i+1, str+"-"+s)
                        dfs(num, target, i+1, str+"*"+s)
                }
        }
}

// leetcode227.基本计算器II
func calculate(s string) int {
        stack := make([]int, 0)
        num := 0
        sign := byte('+')
        for i := 0; i < len(s); i++ {
                if s[i] == ' ' {
                        continue
                }
                if '0' <= s[i] && s[i] <= '9' {
                        num = num*10 + int(s[i]-'0')
                }
                if s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' || i ==␣
↪len(s)-1 {
                        // 处理前一个符号
                        switch sign {
                        case '+':
                                stack = append(stack, num)
                        case '-':
                                stack = append(stack, -num)
                        case '*':
                                prev := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                stack = append(stack, num*prev)
```

(续下页)

(接上页)

```
                        case '/':
                                prev := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                stack = append(stack, prev/num)
                        }
                        num = 0
                        sign = s[i]
                }
        }
        res := 0
        for i := 0; i < len(stack); i++ {
                res = res + stack[i]
        }
        return res
}
```

9.8 295. 数据流的中位数 (1)

- 题目

```
中位数是有序列表中间的数。如果列表长度是偶数，中位数则是中间两个数的平均值。
例如，[2,3,4] 的中位数是 3
[2,3] 的中位数是 (2 + 3) / 2 = 2.5
设计一个支持以下两种操作的数据结构：
    void addNum(int num) - 从数据流中添加一个整数到数据结构中。
    double findMedian() - 返回目前所有元素的中位数。
示例：
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
进阶：
    如果数据流中所有整数都在 0 到 100 范围内，你将如何优化你的算法？
    如果数据流中 99% 的整数都在 0 到 100 范围内，你将如何优化你的算法？
```

- 解题思路

```
type MinHeap []int

func (i MinHeap) Len() int {
        return len(i)
```

(续下页)

(接上页)

```
}

func (i MinHeap) Less(x, y int) bool {
        return i[x] < i[y]
}

func (i MinHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *MinHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *MinHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

type MaxHeap []int

func (i MaxHeap) Len() int {
        return len(i)
}

func (i MaxHeap) Less(x, y int) bool {
        return i[x] > i[y]
}

func (i MaxHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *MaxHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *MaxHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

type MedianFinder struct {
```

(续下页)

(接上页)

```
        minArr *MinHeap
        maxArr *MaxHeap
}

func Constructor() MedianFinder {
        res := new(MedianFinder)
        res.minArr = new(MinHeap)
        res.maxArr = new(MaxHeap)
        heap.Init(res.minArr)
        heap.Init(res.maxArr)
        return *res
}

func (this *MedianFinder) AddNum(num int) {
        if this.maxArr.Len() == this.minArr.Len() {
                heap.Push(this.minArr, num)
                heap.Push(this.maxArr, heap.Pop(this.minArr))
        } else {
                heap.Push(this.maxArr, num)
                heap.Push(this.minArr, heap.Pop(this.maxArr))
        }
}

func (this *MedianFinder) FindMedian() float64 {
        if this.minArr.Len() == this.maxArr.Len() {
                return (float64((*this.maxArr)[0]) + float64((*this.minArr)[0])) / 2
        } else {
                return float64((*this.maxArr)[0])
        }
}
```

9.9 297. 二叉树的序列化与反序列化 (2)

- 题目

```
序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存
同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。
请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 /␣
↪反序列化算法执行逻辑，
你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。
示例：你可以将以下二叉树：
    1
```

(续下页)

(接上页)

```
   / \
  2   3
     / \
    4   5
序列化为 "[1,2,3,null,null,4,5]"
提示: 这与 LeetCode 目前使用的方式一致, 详情请参阅 LeetCode 序列化二叉树的格式。
你并非必须采取这种方式, 你也可以采用其他的方法解决这个问题。
说明: 不要使用类的成员 / 全局 /␣
↪静态变量来存储状态, 你的序列化和反序列化算法应该是无状态的。
```

- 解题思路

```
type Codec struct {
        res []string
}

func Constructor() Codec {
        return Codec{}
}

// Serializes a tree to a single string.
func (this *Codec) serialize(root *TreeNode) string {
        if root == nil {
                return "#"
        }
        return strconv.Itoa(root.Val) + "," + this.serialize(root.Left) + "," + this.
↪serialize(root.Right)
}

// Deserializes your encoded data to tree.
func (this *Codec) deserialize(data string) *TreeNode {
        this.res = strings.Split(data, ",")
        return this.dfsDeserialize()
}

func (this *Codec) dfsDeserialize() *TreeNode {
        node := this.res[0]
        this.res = this.res[1:]
        if node == "#" {
                return nil
        }
        value, _ := strconv.Atoi(node)
        return &TreeNode{
                Val:   value,
```

(续下页)

(接上页)

```
                Left:  this.dfsDeserialize(),
                Right: this.dfsDeserialize(),
        }
}

# 2
type Codec struct {
        res []string
}

func Constructor() Codec {
        return Codec{}
}

// Serializes a tree to a single string.
func (this *Codec) serialize(root *TreeNode) string {
        if root == nil {
                return ""
        }
        res := make([]string, 0)
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node != nil {
                        res = append(res, strconv.Itoa(node.Val))
                        queue = append(queue, node.Left, node.Right)
                } else {
                        res = append(res, "#")
                }
        }
        return strings.Join(res, ",")
}

// Deserializes your encoded data to tree.
func (this *Codec) deserialize(data string) *TreeNode {
        if len(data) == 0 || data == "" {
                return nil
        }
        res := strings.Split(data, ",")
        root := &TreeNode{}
        root.Val, _ = strconv.Atoi(res[0])
```

(续下页)

(接上页)

```
        res = res[1:]
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                if res[0] != "#" {
                        left, _ := strconv.Atoi(res[0])
                        queue[0].Left = &TreeNode{Val: left}
                        queue = append(queue, queue[0].Left)
                }
                if res[1] != "#" {
                        right, _ := strconv.Atoi(res[1])
                        queue[0].Right = &TreeNode{Val: right}
                        queue = append(queue, queue[0].Right)
                }
                queue = queue[1:]
                res = res[2:]
        }
        return root
}
```

CHAPTER 10

0301-0400-Easy

10.1 303. 区域和检索-数组不可变 (2)

- 题目

```
给定一个整数数组  nums，求出数组从索引 i 到 j  (i ≤ j) 范围内元素的总和，包含 i,  j
↪两点。
示例：
给定 nums = [-2, 0, 3, -5, 2, -1]，求和函数为 sumRange()
sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3

说明：
    你可以假设数组不可变。
    会多次调用 sumRange 方法。
```

- 解题思路

```
type NumArray struct {
        arr []int
}

func Constructor(nums []int) NumArray {
        size := len(nums)
```

(续下页)

(接上页)

```
        arr := make([]int, size+1)
        for i := 1; i <= size; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        return NumArray{arr: arr}
}

func (n *NumArray) SumRange(i int, j int) int {
        return n.arr[j+1] - n.arr[i]
}

#
type NumArray struct {
        arr []int
}

func Constructor(nums []int) NumArray {
        return NumArray{nums}
}

func (n *NumArray) SumRange(i int, j int) int {
        sum := 0
        for ; i <= j; i++ {
                sum = sum + n.arr[i]
        }
        return sum
}
```

10.2 326.3 的幂 (3)

- 题目

```
给定一个整数，写一个函数来判断它是否是 3 的幂次方。
示例 1: 输入: 27 输出: true
示例 2: 输入: 0 输出: false
示例 3: 输入: 9 输出: true
示例 4: 输入: 45 输出: false
进阶：你能不使用循环或者递归来完成本题吗？
```

- 解题思路

```
func isPowerOfThree(n int) bool {
        if n <= 0 {
                return false
        }
        for n > 1 {
                if n % 3 != 0{
                        return false
                }
                n = n / 3
        }
        return n == 1
}

#
func isPowerOfThree(n int) bool {
        if n <= 0 {
                return false
        }
        str := strconv.FormatInt(int64(n), 3)
        return str[0:1] == "1" && strings.Count(str, "0") == len(str)-1
}


#
func isPowerOfThree(n int) bool {
        if n <= 0 {
                return false
        }
        if n == 1 {
                return true
        }
        if n%3 != 0 {
                return false
        }
        return isPowerOfThree(n / 3)
}
```

10.3 342.4 的幂 (4)

- 题目

```
给定一个整数 (32 位有符号整数), 请编写一个函数来判断它是否是 4 的幂次方。
示例 1: 输入: 16 输出: true
示例 2: 输入: 5 输出: false
进阶: 你能不使用循环或者递归来完成本题吗?
```

- 解题思路

```
func isPowerOfFour(num int) bool {
        if num <= 0 {
                return false
        }

        for num > 1 {
                if num%4 != 0 {
                        return false
                }
                num = num / 4
        }
        return num == 1
}

#
func isPowerOfFour(num int) bool {
        if num <= 0 {
                return false
        }
        if num == 1{
                return true
        }
        if num % 4 != 0{
                return false
        }

        return isPowerOfFour(num/4)
}

#
func isPowerOfFour(num int) bool {
        if num <= 0 {
                return false
```

(续下页)

(接上页)

```
        }
        // return (num & (num-1) == 0) && (num-1)%3 == 0
        return (num&(num-1) == 0) && (num&0xaaaaaaaa == 0)
}

#
func isPowerOfFour(num int) bool {
        if num <= 0 {
                return false
        }
        str := strconv.FormatInt(int64(num), 4)
        return str[0:1] == "1" && strings.Count(str, "0") == len(str)-1
}
```

10.4 344. 反转字符串 (3)

- 题目

```
编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 char[] 的形式给出。
不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1)␣
↪的额外空间解决这一问题。
你可以假设数组中的所有字符都是 ASCII 码表中的可打印字符。

示例 1:  输入：["h","e","l","l","o"] 输出：["o","l","l","e","h"]
示例 2:  输入：["H","a","n","n","a","h"] 输出：["h","a","n","n","a","H"]
```

- 解题思路

```
func reverseString(s []byte) {
        i, j := 0, len(s)-1
        for i < j {
                s[i], s[j] = s[j], s[i]
                i++
                j--
        }
}

#
func reverseString(s []byte) {
        var reverse func(int, int)
        reverse = func(left, right int) {
                if left < right {
```

(续下页)

(接上页)

```
                        s[left], s[right] = s[right], s[left]
                        reverse(left+1, right-1)
                }
        }
        reverse(0, len(s)-1)
}

#
func reverseString(s []byte) {
        for i := 0; i < len(s)/2; i++ {
                s[i], s[len(s)-1-i] = s[len(s)-1-i], s[i]
        }
}
```

10.5 345. 反转字符串中的元音字母 (2)

- 题目

```
编写一个函数，以字符串作为输入，反转该字符串中的元音字母。

示例 1:输入: "hello"输出: "holle"
示例 2:输入: "leetcode"输出: "leotcede"
说明:元音字母不包含字母"y"。
```

- 解题思路

```
func reverseVowels(s string) string {
        bytes := []byte(s)
        length := len(s)
        i, j := 0, length-1
        for i < j {
                if !isvowels(bytes[i]) {
                        i++
                        continue
                }
                if !isvowels(bytes[j]) {
                        j--
                        continue
                }
                bytes[i], bytes[j] = bytes[j], bytes[i]
                i++
                j--
```

(续下页)

(接上页)

```
        }
        return string(bytes)
}

func isvowels(b byte) bool {
        return b == 'a' || b == 'e' || b == 'i' || b == 'o' || b == 'u' ||
                b == 'A' || b == 'E' || b == 'I' || b == 'O' || b == 'U'
}

#
func reverseVowels(s string) string {
        bytes := []byte(s)
        length := len(s)
        temp := make([]byte, 0)
        for i := 0; i < length; i++ {
                if isvowels(bytes[i]) {
                        temp = append(temp, bytes[i])
                }
        }
        count := 0
        for i := 0; i < length; i++ {
                if isvowels(bytes[i]) {
                        bytes[i] = temp[len(temp)-1-count]
                        count++
                }
        }
        return string(bytes)
}

func isvowels(b byte) bool {
        return b == 'a' || b == 'e' || b == 'i' || b == 'o' || b == 'u' ||
                b == 'A' || b == 'E' || b == 'I' || b == 'O' || b == 'U'
}
```

10.6 349. 两个数组的交集 (3)

- 题目

```
给定两个数组，编写一个函数来计算它们的交集。
示例 1:输入: nums1 = [1,2,2,1], nums2 = [2,2]输出: [2]
示例 2:输入: nums1 = [4,9,5], nums2 = [9,4,9,8,4]输出: [9,4]
说明:
```

(续下页)

(接上页)

输出结果中的每个元素一定是唯一的。
我们可以不考虑输出结果的顺序。

- 解题思路

```
func intersection(nums1 []int, nums2 []int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        for _, v := range nums1 {
                m[v] = 1
        }
        for _, v := range nums2 {
                if m[v] == 1 {
                        res = append(res, v)
                        m[v] += 1
                }
        }
        return res
}

#
func intersection(nums1 []int, nums2 []int) []int {
        m1 := make(map[int]bool)
        m2 := make(map[int]bool)
        res := make([]int, 0)
        for _, v := range nums1 {
                m1[v] = true
        }

        for _, v := range nums2 {
                if m1[v] != false {
                        m2[v] = true
                }
        }

        for k := range m2 {
                res = append(res, k)
        }
        return res
}

#
func intersection(nums1 []int, nums2 []int) []int {
        sort.Ints(nums1)
```

(续下页)

(接上页)

```
        sort.Ints(nums2)
        res := make([]int, 0)
        i := 0
        j := 0
        for i < len(nums1) && j < len(nums2) {
                if nums1[i] < nums2[j] {
                        i++
                } else if nums1[i] > nums2[j] {
                        j++
                } else {
                        if len(res) == 0 || res[len(res)-1] != nums1[i] {
                                res = append(res, nums1[i])
                        }
                        i++
                        j++
                }
        }
        return res
}
```

10.7 350. 两个数组的交集 II(3)

- 题目

```
给定两个数组，编写一个函数来计算它们的交集。
示例 1: 输入: nums1 = [1,2,2,1], nums2 = [2,2]  输出: [2,2]
示例 2: 输入: nums1 = [4,9,5], nums2 = [9,4,9,8,4] 输出: [4,9]
说明: 输出结果中每个元素出现的次数，应与元素在两个数组中出现的次数一致。
     我们可以不考虑输出结果的顺序。
进阶:
    如果给定的数组已经排好序呢？你将如何优化你的算法？
    如果 nums1 的大小比 nums2 小很多，哪种方法更优？
    如果 nums2␣
↪的元素存储在磁盘上，磁盘内存是有限的，并且你不能一次加载所有的元素到内存中，你该怎么办？
```

- 解题思路

```
func intersect(nums1 []int, nums2 []int) []int {
        m1 := make(map[int]int)
        res := make([]int, 0)
        for _, v := range nums1 {
                m1[v] += 1
```

(续下页)

(接上页)

```
        }

        for _, v := range nums2 {
                if m1[v] > 0 {
                        res = append(res, v)
                        m1[v]--
                }
        }
        return res
}

#
func intersect(nums1 []int, nums2 []int) []int {
        m1 := make(map[int]int)
        m2 := make(map[int]int)
        res := make([]int, 0)
        for _, v := range nums1 {
                m1[v]++
        }

        for _, v := range nums2 {
                if m1[v] != 0 && m1[v] > m2[v] {
                        m2[v]++
                }
        }

        for k := range m2 {
                for i := 0; i < m2[k]; i++ {
                        res = append(res, k)
                }
        }
        return res
}

#
func intersect(nums1 []int, nums2 []int) []int {
        sort.Ints(nums1)
        sort.Ints(nums2)
        res := make([]int, 0)
        i := 0
        j := 0
        for i < len(nums1) && j < len(nums2) {
                if nums1[i] < nums2[j] {
```

(续下页)

(接上页)

```
                        i++
                } else if nums1[i] > nums2[j] {
                        j++
                } else {
                        res = append(res, nums1[i])
                        i++
                        j++
                }
        }
        return res
}
```

10.8 367. 有效的完全平方数 (4)

- 题目

```
给定一个正整数 num，编写一个函数，如果 num 是一个完全平方数，则返回 True，否则返回␣
↪False。
说明：不要使用任何内置的库函数，如  sqrt。
示例 1：输入：16 输出：True
示例 2：输入：14 输出：False
```

- 解题思路

```
func isPerfectSquare(num int) bool {
        if num < 2 {
                return true
        }
        left := 2
        right := num / 2
        for left <= right {
                mid := left + (right-left)/2
                if mid*mid == num {
                        return true
                } else if mid*mid > num {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return false
}
```

(续下页)

(接上页)

```
#
func isPerfectSquare(num int) bool {
        if num < 2 {
                return true
        }
        x := num / 2
        for x*x > num {
                x = (x + num/x) / 2
        }
        return x*x == num
}

#
func isPerfectSquare(num int) bool {
        i := 1
        for num > 0 {
                num = num - i
                i = i + 2
        }
        return num == 0
}

#
func isPerfectSquare(num int) bool {
        i := 1
        for i * i < num{
                i++
        }
        return i * i == num
}
```

10.9 371. 两整数之和 (2)

- 题目

```
不使用运算符 + 和 - ，计算两整数a,b之和。
示例 1:输入: a = 1, b = 2 输出: 3
示例 2:输入: a = -2, b = 3 输出: 1
```

- 解题思路

```
func getSum(a int, b int) int {
        for b != 0 {
                a, b = a^b, (a&b)<<1
        }
        return a
}


#
func getSum(a int, b int) int {
        if b == 0 {
                return a
        }
        return getSum(a^b, (a&b)<<1)
}
```

10.10 374. 猜数字大小 (2)

- 题目

```
我们正在玩一个猜数字游戏。 游戏规则如下:
我从 1 到 n 选择一个数字。 你需要猜我选择了哪个数字。
每次你猜错了，我会告诉你这个数字是大了还是小了。
你调用一个预先定义好的接口 guess(int num)，它会返回 3 个可能的结果 (-1，1 或 0)：
-1 : 我的数字比较小
 1 : 我的数字比较大
 0 : 恭喜! 你猜对了!

示例 :输入: n = 10, pick = 6 输出: 6
```

- 解题思路

```
func guessNumber(n int) int {
        low := 1
        high := n
        for low < high{
                mid := low + (high-low)/2
                if guess(mid) == 0{
                        return mid
                }else if guess(mid) == 1{
                        low = mid + 1
                }else {
                        high = mid - 1
```

(续下页)

(接上页)

```
                }
        }
        return low
}

#
func guessNumber(n int) int {
        return binary(1, n)
}

func binary(left, right int) int {
        mid := left + (right-left)/2
        if guess(mid) == 1 {
                return binary(mid+1, right)
        } else if guess(mid) == -1 {
                return binary(left, mid-1)
        } else {
                return mid
        }
}
```

10.11 383. 赎金信 (3)

- 题目

```
给定一个赎金信 (ransom) 字符串和一个杂志(magazine)字符串，
判断第一个字符串 ransom 能不能由第二个字符串 magazines 里面的字符构成。
如果可以构成，返回 true ；否则返回 false。

(题目说明：为了不暴露赎金信字迹，要从杂志上搜索各个需要的字母，组成单词来表达意思。
杂志字符串中的每个字符只能在赎金信字符串中使用一次。)

注意：你可以假设两个字符串均只含有小写字母。
canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true
```

- 解题思路

```
func canConstruct(ransomNote string, magazine string) bool {
        index := [26]int{}
        for i := 0; i < len(magazine); i++ {
```

(续下页)

(接上页)

```
                index[magazine[i]-'a']++
        }

        for i := 0; i < len(ransomNote); i++ {
                index[ransomNote[i]-'a']--
                if index[ransomNote[i]-'a'] < 0 {
                        return false
                }
        }
        return true
}

#
func canConstruct(ransomNote string, magazine string) bool {
        index := make(map[uint8]int)
        for i := 0; i < len(magazine); i++ {
                index[magazine[i]-'a']++
        }

        for i := 0; i < len(ransomNote); i++ {
                index[ransomNote[i]-'a']--
                if index[ransomNote[i]-'a'] < 0 {
                        return false
                }
        }
        return true
}

#
func canConstruct(ransomNote string, magazine string) bool {
        ransomNoteArr := strings.Split(ransomNote, "")
        magazineArr := strings.Split(magazine, "")
        sort.Strings(ransomNoteArr)
        sort.Strings(magazineArr)

        i := 0
        j := 0
        for i < len(ransomNoteArr) && j < len(magazineArr) {
                if ransomNoteArr[i] > magazineArr[j] {
                        j++
                } else if ransomNoteArr[i] < magazineArr[j] {
                        return false
                } else {
```

(续下页)

(接上页)

```
                        i++
                        j++
                }
        }
        return i == len(ransomNote)
}
```

10.12 387. 字符串中的第一个唯一字符 (3)

- 题目

```
给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。
案例:
s = "leetcode"返回 0.
s = "loveleetcode",返回 2.
注意事项: 您可以假定该字符串只包含小写字母。
```

- 解题思路

```
func firstUniqChar(s string) int {
        m := [26]int{}
        for i := 0; i < len(s); i++ {
                m[s[i]-'a']++
        }
        for i := 0; i < len(s); i++ {
                if m[s[i]-'a'] == 1 {
                        return i
                }
        }
        return -1
}

#
func firstUniqChar(s string) int {
        m := make(map[uint8]int)
        for i := 0; i < len(s); i++ {
                m[s[i]-'a']++
        }
        for i := 0; i < len(s); i++ {
                if m[s[i]-'a'] == 1 {
                        return i
                }
```

(续下页)

(接上页)

```
        }
        return -1
}

#
func firstUniqChar(s string) int {
        for i := 0; i < len(s); i++ {
                flag := true
                for j := 0; j < len(s); j++ {
                        if s[i] == s[j] && i != j {
                                flag = false
                                break
                        }
                }
                if flag {
                        return i
                }
        }
        return -1
}
```

10.13 389. 找不同 (5)

- 题目

```
给定两个字符串 s 和 t，它们只包含小写字母。
字符串 t 由字符串 s 随机重排，然后在随机位置添加一个字母。
请找出在 t 中被添加的字母。
示例:输入：s = "abcd"t = "abcde"输出：e
解释：'e' 是那个被添加的字母。
```

- 解题思路

```
func findTheDifference(s string, t string) byte {
        m := [26]int{}
        bytest := []byte(t)
        bytess := []byte(s)
        for _, v := range bytest {
                m[v-'a']++
        }
        for _, v := range bytess {
                m[v-'a']--
```

(续下页)

(接上页)

```
        }
        for k := range m {
                if m[k] == 1 {
                        return byte(k + 'a')
                }
        }
        return 0
}

#
func findTheDifference(s string, t string) byte {
        m := make(map[byte]int)
        bytest := []byte(t)
        bytess := []byte(s)
        for _, v := range bytest {
                m[v]++
        }
        for _, v := range bytess {
                m[v]--
        }
        for k := range m {
                if m[k] == 1 {
                        return k
                }
        }
        return 0
}

#
func findTheDifference(s string, t string) byte {
        ch := byte(0)
        for _, value := range s {
                ch ^= byte(value)
        }
        for _, value := range t {
                ch ^= byte(value)
        }
        return ch
}

#
func findTheDifference(s string, t string) byte {
        ch := byte(0)
```

(续下页)

(接上页)

```
        for _, value := range t {
                ch += byte(value)
        }
        for _, value := range s {
                ch -= byte(value)
        }
        return ch
}

#
func findTheDifference(s string, t string) byte {
        sArr := strings.Split(s, "")
        tArr := strings.Split(t, "")
        sort.Strings(sArr)
        sort.Strings(tArr)
        for i := 0; i < len(sArr); i++{
                if sArr[i] != tArr[i]{
                        return []byte(tArr[i])[0]
                }
        }
        return []byte(tArr[len(tArr)-1])[0]
}
```

10.14 392. 判断子序列 (4)

- 题目

```
给定字符串 s 和 t ，判断 s 是否为 t 的子序列。
你可以认为 s 和 t 中仅包含英文小写字母。
字符串 t 可能会很长（长度 ~= 500,000），而 s 是个短字符串（长度 <=100）。
字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。
（例如，"ace"是"abcde"的一个子序列，而"aec"不是）。
示例 1:s = "abc", t = "ahbgdc"返回 true.
示例 2:s = "axc", t = "ahbgdc"返回 false.
后续挑战 :
如果有大量输入的 S，称作S1, S2, ... , Sk 其中 k >= 10亿，
你需要依次检查它们是否为 T 的子序列。在这种情况下，你会怎样改变代码？
```

- 解题思路

```
func isSubsequence(s string, t string) bool {
        if len(s) > len(t){
```

(续下页)

(接上页)

```
                return false
        }
        i := 0
        j := 0
        for i < len(s) && j < len(t){
                if s[i] == t[j]{
                        i++
                }
                j++
        }
        return i == len(s)
}

#
func isSubsequence(s string, t string) bool {
        for _, v := range s{
                idx := strings.IndexRune(t, v)
                if idx == -1{
                        return false
                }
                t = t[idx+1:]
        }
        return true
}

#
func isSubsequence(s string, t string) bool {
        m := make(map[uint8][]int)
        for i := 0; i < len(t); i++ {
                value := t[i] - 'a'
                if m[value] == nil {
                        m[value] = make([]int, 0)
                }
                m[value] = append(m[value], i)
        }
        prev := -1
        for i := 0; i < len(s); i++ {
                value := s[i] - 'a'
                left := 0
                right := len(m[value]) - 1
                if len(m[value]) == 0 {
                        return false
                }
```

(续下页)

(接上页)

```
                for left < right {
                        mid := left + (right-left)/2
                        if m[value][mid] > prev {
                                right = mid
                        } else {
                                left = mid + 1
                        }
                }
                if left > right || m[value][left] <= prev {
                        return false
                }
                prev = m[value][left]
        }
        return true
}

#
/*
状态定义: dp[i][j] 表示长度为i的字符串s是否为长度为j的字符串t的子序列
状态转移方程: 如果s[i] == t[j], 则dp[i][j] = dp[i-1][j-1]
如果s[i] != t[j], 则dp[i][j] = dp[i][j-1]
初始: dp[0][j] = true 表示空串s 是任意长度串t的子串
dp[i][0] = false 表示任意长度非空串s 不是空串t的字串
dp[i][0] = false 表示任意长度非空串s 不是空串t的字串
*/
func isSubsequence(s string, t string) bool {
        if len(s) == 0 {
                return true
        } else if len(s) > len(t) {
                return false
        }

        dp := make([][]bool, len(s)+1)
        for i := 0; i < len(s)+1; i++ {
                dp[i] = make([]bool, len(t)+1)
                dp[i][0] = false
        }
        for i := 0; i <= len(t); i++ {
                dp[0][i] = true
        }

        for i := 1; i <= len(s); i++ {
                for j := 1; j <= len(t); j++ {
```

(续下页)

(接上页)

```
                        if s[i-1] == t[j-1] {
                                dp[i][j] = dp[i-1][j-1]
                        } else {
                                dp[i][j] = dp[i][j-1]
                        }
                }
        }
        return dp[len(s)][len(t)]
}
```

CHAPTER 11

0301-0400-Medium

11.1 304. 二维区域和检索-矩阵不可变 (1)

- 题目

```
给定一个二维矩阵，计算其子矩形范围内元素的总和，该子矩阵的左上角为 (row1, col1) ，
右下角为 (row2, col2)。
Range Sum Query 2D
上图子矩阵左上角 (row1, col1) = (2, 1) ，右下角(row2, col2) = (4, 3)，
该子矩形内元素的总和为 8。
示例: 给定 matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]
]
sumRegion(2, 1, 4, 3) -> 8
sumRegion(1, 1, 2, 2) -> 11
sumRegion(1, 2, 2, 4) -> 12
说明:你可以假设矩阵不可变。
    会多次调用 sumRegion 方法。
    你可以假设 row1 ≤ row2 且 col1 ≤ col2。
```

- 解题思路

```
type NumMatrix struct {
        arr [][]int
}

func Constructor(matrix [][]int) NumMatrix {
        if matrix == nil || len(matrix) == 0 || matrix[0] == nil || len(matrix[0]) ==␣
↪0 {
                arr := make([][]int, 1)
                for i := 0; i < 1; i++ {
                        arr[i] = make([]int, 1)
                }
                return NumMatrix{arr: arr}
        }
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n+1)
        for i := 0; i < n+1; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        arr[i][j] = arr[i][j-1] + arr[i-1][j] - arr[i-1][j-1] +␣
↪matrix[i-1][j-1]
                }
        }
        return NumMatrix{arr: arr}
}

func (this *NumMatrix) SumRegion(row1 int, col1 int, row2 int, col2 int) int {
        return this.arr[row2+1][col2+1] - this.arr[row2+1][col1] - this.
↪arr[row1][col2+1] + this.arr[row1][col1]
}
```

11.2 306. 累加数 (1)

- 题目

```
累加数是一个字符串，组成它的数字可以形成累加序列。
一个有效的累加序列必须至少包含 3␣
↪个数。除了最开始的两个数以外，字符串中的其他数都等于它之前两个数相加的和。
给定一个只包含数字 '0'-'9' 的字符串，编写一个算法来判断给定输入是否是累加数。
说明: 累加序列里的数不会以 0 开头，所以不会出现 1, 2, 03 或者 1, 02, 3 的情况。
示例 1:输入: "112358" 输出: true
```

(续下页)

(接上页)

```
解释: 累加序列为: 1, 1, 2, 3, 5, 8 。1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8
示例 2:输入: "199100199" 输出: true
解释: 累加序列为: 1, 99, 100, 199。1 + 99 = 100, 99 + 100 = 199
进阶:你如何处理一个溢出的过大的整数输入?
```

- 解题思路

```
var res []int

func isAdditiveNumber(num string) bool {
        if len(num) < 3 {
                return false
        }
        res = make([]int, 0)
        dfs(num, 0, 0, 0, make([]int, 0))
        return len(res) >= 3
}

func dfs(s string, index, sum, prev int, path []int) bool {
        if index == len(s) {
                if len(path) >= 3 {
                        res = path
                }
                return len(path) >= 3
        }
        value := 0
        for i := index; i < len(s); i++ {
                // 0开头不满足要求(当前i=index的时候，可以为0，  避免错过1+0=1的情况)
                if s[index] == '0' && i > index {
                        break
                }
                value = value*10 + int(s[i]-'0')
                if len(path) >= 2 {
                        if value < sum {
                                continue
                        }
                        if value > sum {
                                break
                        }
                }
                if dfs(s, i+1, prev+value, value, append(path, value)) == true {
                        return true
                }
        }
```

(续下页)

(接上页)

```
        return false
}
```

11.3 307. 区域和检索-数组可修改 (3)

- 题目

```
给你一个数组 nums␣
↪，请你完成两类查询，其中一类查询要求更新数组下标对应的值，另一类查询要求返回数组中某个范围内元素的
实现 NumArray 类：
NumArray(int[] nums) 用整数数组 nums 初始化对象
void update(int index, int val) 将 nums[index] 的值更新为 val
int sumRange(int left, int right) 返回子数组 nums[left, right] 的总和
（即，nums[left] + nums[left + 1], ..., nums[right]）
示例：输入： ["NumArray", "sumRange", "update", "sumRange"]
[[[1, 3, 5]], [0, 2], [1, 2], [0, 2]]
输出： [null, 9, null, 8]
解释：
NumArray numArray = new NumArray([1, 3, 5]);
numArray.sumRange(0, 2); // 返回 9 , sum([1,3,5]) = 9
numArray.update(1, 2);   // nums = [1,2,5]
numArray.sumRange(0, 2); // 返回 8 , sum([1,2,5]) = 8
提示：1 <= nums.length <= 3 * 104
-100 <= nums[i] <= 100
0 <= index < nums.length
-100 <= val <= 100
0 <= left <= right < nums.length
最多调用 3 * 104 次 update 和 sumRange 方法
```

- 解题思路

```
type NumArray struct {
        arr    []int // 原数组
        b      []int // 分块和
        length int   // 分块长度
}

func Constructor(nums []int) NumArray {
        n := len(nums)
        per := int(math.Sqrt(float64(n)))
        length := int(math.Ceil(float64(n) / float64(per)))
        b := make([]int, length)
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                b[i/length] = b[i/length] + nums[i]
        }
        return NumArray{
                arr:    nums,
                b:      b,
                length: length,
        }
}

func (this *NumArray) Update(i int, val int) {
        index := i / this.length
        this.b[index] = this.b[index] - this.arr[i] + val
        this.arr[i] = val
}

func (this *NumArray) SumRange(i int, j int) int {
        res := 0
        a, b := i/this.length, j/this.length
        if a == b {
                for k := i; k <= j; k++ {
                        res = res + this.arr[k]
                }
        } else {
                // 分3段
                for k := i; k <= (a+1)*this.length-1; k++ {
                        res = res + this.arr[k]
                }
                for k := a + 1; k <= b-1; k++ {
                        res = res + this.b[k]
                }
                for k := b * this.length; k <= j; k++ {
                        res = res + this.arr[k]
                }
        }
        return res
}

# 2
type NumArray struct {
        origin []int // 原数组
        arr    []int // 线段树
        length int   // 长度
```

(续下页)

(接上页)

```
}

func Constructor(nums []int) NumArray {
        n := len(nums)
        arr := make([]int, 4*n+1)
        res := NumArray{
                origin: nums,
                arr:    arr,
                length: n,
        }
        for i := 0; i < n; i++ {
                res.UpdateArr(0, 1, res.length, i+1, nums[i]) // 从1开始，添加nums[i]
        }
        return res
}

func (this *NumArray) Update(index int, val int) {
        val, this.origin[index] = val-this.origin[index], val // 需要变更的大小
        this.UpdateArr(0, 1, this.length, index+1, val)       // 从1开始
}

func (this *NumArray) SumRange(left int, right int) int {
        return this.Query(0, 1, this.length, left+1, right+1) // 范围+1
}

func (this *NumArray) UpdateArr(id int, left, right, x int, value int) {
        if left > x || right < x {
                return
        }
        this.arr[id] = this.arr[id] + value
        if left == right {
                return
        }
        mid := left + (right-left)/2
        this.UpdateArr(2*id+1, left, mid, x, value)    // 左节点
        this.UpdateArr(2*id+2, mid+1, right, x, value) // 右节点
}

func (this *NumArray) Query(id int, left, right, queryLeft, queryRight int) int {
        if left > queryRight || right < queryLeft {
                return 0
        }
        if queryLeft <= left && right <= queryRight {
```

(续下页)

(接上页)

```
                return this.arr[id]
        }
        mid := left + (right-left)/2
        return this.Query(id*2+1, left, mid, queryLeft, queryRight) +
                this.Query(id*2+2, mid+1, right, queryLeft, queryRight)
}

# 3
type NumArray struct {
        origin []int // 原数组
        c      []int // 树状数组
        length int   // 长度
}

func Constructor(nums []int) NumArray {
        n := len(nums)
        arr := make([]int, n+1)
        res := NumArray{
                origin: nums,
                c:      arr,
                length: n,
        }
        // 单点修改
        for i := 0; i < n; i++ {
                res.UpData(i+1, nums[i]) // 注意下标，默认数组是从1开始
        }
        return res
}

func (this *NumArray) Update(index int, val int) {
        if index < 0 || index > this.length-1 {
                return
        }
        val, this.origin[index] = val-this.origin[index], val // 需要变更的大小
        this.UpData(index+1, val)
}

func (this *NumArray) SumRange(left int, right int) int {
        return this.GetSum(right+1) - this.GetSum(left)
}

func (this *NumArray) LowBit(x int) int {
        return x & (-x)
```

(续下页)

(接上页)

```
}

// 单点修改
func (this *NumArray) UpData(i, k int) { // 在i位置加上k
        for i <= this.length {
                this.c[i] = this.c[i] + k
                i = i + this.LowBit(i) // i = i + 2^k
        }
}

// 区间查询
func (this *NumArray) GetSum(i int) int {
        res := 0
        for i > 0 {
                res = res + this.c[i]
                i = i - this.LowBit(i)
        }
        return res
}
```

11.4 309. 最佳买卖股票时机含冷冻期 (2)

- 题目

```
给定一个整数数组，其中第 i 个元素代表了第 i 天的股票价格 。
设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）:
    你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
    卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。
示例:输入: [1,2,3,0,2] 输出: 3
解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]
```

- 解题思路

```
func maxProfit(prices []int) int {
        if len(prices) == 0 {
                return 0
        }
        n := len(prices)
        dp := make([][3]int, n)
        // 0 => 持有
        // 1 => 不持有，本日卖出，下一日冷冻期
        // 2 => 不持有，本日无卖出，下一日不是冷冻期
```

(续下页)

(接上页)

```
        dp[0][0] = -prices[0] // 第0天买入，亏损-price[0]
        for i := 1; i < n; i++ {
                dp[i][0] = max(dp[i-1][0], dp[i-1][2]-prices[i]) // 继续持有 or
↪可以操作，继续买入，导致今天持有股票
                dp[i][1] = dp[i-1][0] + prices[i]                // 卖出操作
                dp[i][2] = max(dp[i-1][1], dp[i-1][2])           //
↪昨天卖出，无股票，今天是冷冻期 or 昨天没股票，也不操作
        }
        return max(dp[n-1][1], dp[n-1][2]) // 最后一天操作，会导致利润变少，可以忽略
        // return max(dp[n-1][0], max(dp[n-1][1], dp[n-1][2]))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxProfit(prices []int) int {
        if len(prices) == 0 {
                return 0
        }
        n := len(prices)
        // a => 持有
        // b => 不持有，本日卖出，下一日冷冻期
        // c => 不持有，本日无卖出，下一日不是冷冻期
        var a, b, c int
        a = -prices[0] // 第0天买入，亏损-price[0]
        for i := 1; i < n; i++ {
                A := max(a, c-prices[i]) // 继续持有 or
↪可以操作，继续买入，导致今天持有股票
                B := a + prices[i]       // 卖出操作
                C := max(b, c)           // 昨天卖出，无股票，今天是冷冻期 or
↪昨天没股票，也不操作
                a, b, c = A, B, C
        }
        return max(b, c)
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}
```

11.5 310. 最小高度树 (1)

- 题目

```
树是一个无向图，其中任何两个顶点只通过一条路径连接。␣
↪换句话说，一个任何没有简单环路的连通图都是一棵树。
给你一棵包含n个节点的数，标记为0到n - 1 。
给定数字n和一个有 n - 1 条无向边的 edges列表（每一个边都是一对标签），
其中 edges[i] = [ai, bi] 表示树中节点 ai 和 bi 之间存在一条无向边。
可选择树中任何一个节点作为根。当选择节点 x 作为根节点时，设结果树的高度为 h 。
在所有可能的树中，具有最小高度的树（即，min(h)）被称为 最小高度树 。
请你找到所有的 最小高度树 并按 任意顺序 返回它们的根节点标签列表。
树的 高度 是指根节点和叶子节点之间最长向下路径上边的数量。
示例 1: 输入: n = 4, edges = [[1,0],[1,2],[1,3]] 输出: [1]
解释: 如图所示，当根是标签为 1 的节点时，树的高度是 1 ，这是唯一的最小高度树。
示例 2: 输入: n = 6, edges = [[3,0],[3,1],[3,2],[3,4],[5,4]] 输出: [3,4]
示例 3: 输入: n = 1, edges = [] 输出: [0]
示例 4: 输入: n = 2, edges = [[0,1]] 输出: [0,1]
提示: 1 <= n <= 2 * 104
edges.length == n - 1
0 <= ai, bi < n
ai != bi
所有 (ai, bi) 互不相同
给定的输入 保证 是一棵树，并且 不会有重复的边
```

- 解题思路

```
func findMinHeightTrees(n int, edges [][]int) []int {
        if n == 1 {
                return []int{0}
        }
        if n == 2 {
                return []int{0, 1}
        }
        m := make(map[int][]int)
        degree := make([]int, n)
        for i := 0; i < len(edges); i++ {
```

(续下页)

(接上页)

```
                a, b := edges[i][0], edges[i][1]
                m[a] = append(m[a], b)
                m[b] = append(m[b], a)
                degree[a]++
                degree[b]++
        }
        // 从叶子节点开始遍历
        queue := make([]int, 0)
        for i := 0; i < n; i++ {
                if degree[i] == 1 {
                        queue = append(queue, i)
                }
        }

        for n > 2 {
                total := len(queue)
                n = n - total
                for i := 0; i < total; i++ {
                        node := queue[i]
                        degree[node] = 0
                        for j := 0; j < len(m[node]); j++ {
                                temp := m[node][j]
                                if degree[temp] > 0 {
                                        degree[temp]--
                                        if degree[temp] == 1 {
                                                queue = append(queue, temp)
                                        }
                                }
                        }
                }
                queue = queue[total:]
        }
        res := make([]int, 0)
        for i := 0; i < len(queue); i++ {
                res = append(res, queue[i])
        }
        return res
}
```

11.6 313. 超级丑数 (2)

- 题目

```
编写一段程序来查找第 n 个超级丑数。
超级丑数是指其所有质因数都是长度为k的质数列表primes中的正整数。
示例:输入: n = 12, primes = [2,7,13,19] 输出: 32
解释: 给定长度为 4 的质数列表 primes = [2,7,13,19], 前 12 个超级丑数序列为: [1,2,4,7,
↪8,13,14,16,19,26,28,32] 。
说明:1是任何给定primes的超级丑数。
给定primes中的数字以升序排列。
0 < k ≤ 100, 0 < n ≤ 106, 0 < primes[i] < 1000 。
第n个超级丑数确保在 32 位有符整数范围内。
```

- 解题思路

```
func nthSuperUglyNumber(n int, primes []int) int {
        if n == 0 || n == 1 {
                return n
        }
        intHeap := &IntHeap{}
        heap.Init(intHeap)
        heap.Push(intHeap, 1)
        n--
        for n > 0 {
                x := heap.Pop(intHeap).(int)
                for intHeap.Len() > 0 && x == (*intHeap)[0] {
                        heap.Pop(intHeap)
                }
                for i := 0; i < len(primes); i++ {
                        heap.Push(intHeap, x*primes[i])
                }
                n--
        }
        return heap.Pop(intHeap).(int)
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

func (h IntHeap) Less(i, j int) bool {
```

(续下页)

(接上页)

```
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func nthSuperUglyNumber(n int, primes []int) int {
        arr := make([]int, n)
        arr[0] = 1
        times := make([]int, len(primes))
        for i := 1; i < n; i++ {
                next := math.MaxInt32
                for j, value := range times {
                        next = min(next, primes[j]*arr[value])
                }
                for j, value := range times {
                        if primes[j]*arr[value] == next {
                                times[j]++
                        }
                }
                arr[i] = next
        }
        return arr[n-1]
}

func min(x, y int) int {
        if x > y {
                return y
        }
        return x
}
```

11.7 318. 最大单词长度乘积 (2)

- 题目

```
给定一个字符串数组 words，找到 length(word[i]) * length(word[j]) 的最大值，
并且这两个单词不含有公共字母。你可以认为每个单词只包含小写字母。如果不存在这样的两个单词，返回
↪0。
示例 1:输入: ["abcw","baz","foo","bar","xtfn","abcdef"] 输出: 16
解释: 这两个单词为 "abcw", "xtfn"。
示例 2:输入: ["a","ab","abc","d","cd","bcd","abcd"] 输出: 4
解释: 这两个单词为 "ab", "cd"。
示例 3:输入: ["a","aa","aaa","aaaa"] 输出: 0
解释: 不存在这样的两个单词。
```

- 解题思路

```
func maxProduct(words []string) int {
        res := 0
        for i := 0; i < len(words); i++ {
                for j := i + 1; j < len(words); j++ {
                        if strings.ContainsAny(words[i], words[j]) == false &&
                                res < len(words[i])*len(words[j]) {
                                res = len(words[i]) * len(words[j])
                        }
                }
        }
        return res
}

# 2
func maxProduct(words []string) int {
        res := 0
        arr := make([]int, len(words))
        for i := 0; i < len(words); i++ {
                for _, char := range words[i] {
                        // 位或 只要有1，那么就是1
                        arr[i] = arr[i] | 1<<uint(char-'a')
                }
        }
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        if arr[i]&arr[j] == 0 && res < len(words[i])*len(words[j]) {
                                res = len(words[i]) * len(words[j])
                        }
```

(续下页)

(接上页)

```
                }
        }
        return res
}
```

11.8 319. 灯泡开关 (1)

- 题目

```
初始时有 n 个灯泡关闭。 第 1 轮，你打开所有的灯泡。 第 2 轮，每两个灯泡你关闭一次。
第 3 轮，每三个灯泡切换一次开关（如果关闭则开启，如果开启则关闭）。
第 i 轮，每 i 个灯泡切换一次开关。 对于第 n 轮，你只切换最后一个灯泡的开关。
找出 n 轮后有多少个亮着的灯泡。
示例:输入: 3输出: 1
解释: 初始时, 灯泡状态 [关闭, 关闭, 关闭].
第一轮后, 灯泡状态 [开启, 开启, 开启].
第二轮后, 灯泡状态 [开启, 关闭, 开启].
第三轮后, 灯泡状态 [开启, 关闭, 关闭].
你应该返回 1, 因为只有一个灯泡还亮着。
```

- 解题思路

```
func bulbSwitch(n int) int {
        // 第i个灯泡的反转次数等于它所有因子（包括1和i）的个数
        // 反转奇数次=>变成亮
        // 只有平方数才有奇数个因子
        return int(math.Sqrt(float64(n)))
}
```

11.9 322. 零钱兑换 (4)

- 题目

```
给定不同面额的硬币 coins 和一个总金额␣
↪amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。
如果没有任何一种硬币组合能组成总金额，返回 -1。
示例 1:输入: coins = [1, 2, 5], amount = 11 输出: 3
解释: 11 = 5 + 5 + 1
示例 2:输入: coins = [2], amount = 3 输出: -1
说明:你可以认为每种硬币的数量是无限的。
```

- 解题思路

```
func coinChange(coins []int, amount int) int {
        dp := make([]int, amount+1)
        for i := 1; i <= amount; i++ {
                dp[i] = -1
                for j := 0; j < len(coins); j++ {
                        prev := i - coins[j]
                        if i < coins[j] || dp[prev] == -1 {
                                continue
                        }
                        if dp[i] == -1 || dp[i] > dp[prev]+1 {
                                dp[i] = dp[prev] + 1
                        }
                }
        }
        return dp[amount]
}

# 2
func coinChange(coins []int, amount int) int {
        dp := make([]int, amount+1)
        for i := 0; i <= amount; i++ {
                dp[i] = amount + 1
        }
        dp[0] = 0
        for i := 0; i < len(coins); i++ {
                for j := coins[i]; j < amount+1; j++ {
                        dp[j] = min(dp[j], dp[j-coins[i]]+1)
                }
        }
        if dp[amount] == amount+1 {
                return -1
        }
        return dp[amount]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
```

(续下页)

(接上页)

```
var res int

func coinChange(coins []int, amount int) int {
        for i := 0; i < len(coins); i++ {
                for j := 0; j < len(coins)-1-i; j++ {
                        if coins[j] < coins[j+1] {
                                coins[j], coins[j+1] = coins[j+1], coins[j]
                        }
                }
        }
        res = math.MaxInt32
        dfs(coins, amount, 0, 0)
        if res == math.MaxInt32 {
                return -1
        }
        return res
}

func dfs(coins []int, amount int, count int, level int) {
        if amount == 0 {
                res = min(res, count)
                return
        }
        if level == len(coins) {
                return
        }
        for i := amount / coins[level]; i >= 0 && i+count < res; i-- {
                dfs(coins, amount-i*coins[level], count+i, level+1)
        }
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func coinChange(coins []int, amount int) int {
        if amount == 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        res := 1
        sort.Ints(coins)
        list := make([]int, 0)
        list = append(list, amount)
        arr := make([]bool, amount+1)
        arr[amount] = true
        for len(list) > 0 {
                length := len(list)
                for i := 0; i < length; i++ {
                        value := list[i]
                        for j := 0; j < len(coins); j++ {
                                next := value - coins[j]
                                if next == 0 {
                                        return res
                                }
                                if next < 0 {
                                        break
                                }
                                if arr[next] == false {
                                        list = append(list, next)
                                        arr[next] = true
                                }
                        }
                }
                list = list[length:]
                res++
        }
        return -1
}
```

11.10 324. 摆动排序 II(2)

- 题目

```
给定一个无序的数组 nums，将它重新排列成 nums[0] < nums[1] > nums[2] < nums[3]...␣
↪的顺序。
示例 1:输入: nums = [1, 5, 1, 1, 6, 4] 输出: 一个可能的答案是 [1, 4, 1, 5, 1, 6]
示例 2:输入: nums = [1, 3, 2, 2, 3, 1] 输出: 一个可能的答案是 [2, 3, 1, 3, 1, 2]
说明: 你可以假设所有输入都会得到有效的结果。
进阶:你能用 O(n) 时间复杂度和 / 或原地 O(1) 额外空间来实现吗？
```

- 解题思路

```
func wiggleSort(nums []int) {
        arr := make([]int, len(nums))
        copy(arr, nums)
        sort.Slice(arr, func(i, j int) bool {
                return arr[i] > arr[j]
        })
        a := 1
        for i := 0; i < len(arr)/2; i++ {
                nums[a] = arr[i]
                a = a + 2
        }
        a = 0
        for i := len(arr) / 2; i < len(arr); i++ {
                nums[a] = arr[i]
                a = a + 2
        }
}

# 2
func wiggleSort(nums []int) {
        arr := make([]int, len(nums))
        copy(arr, nums)
        sort.Ints(arr)
        j := len(nums)
        k := (len(nums) + 1) / 2
        for i := 0; i < len(nums); i++ {
                if i%2 == 1 {
                        j--
                        nums[i] = arr[j]
                } else {
                        k--
                        nums[i] = arr[k]
                }
        }
}
```

11.11 328. 奇偶链表 (3)

- 题目

```
给定一个单链表，把所有的奇数节点和偶数节点分别排在一起。
请注意，这里的奇数节点和偶数节点指的是节点编号的奇偶性，而不是节点的值的奇偶性。
请尝试使用原地算法完成。你的算法的空间复杂度应为 O(1)，时间复杂度应为 O(nodes)，nodes
↪为节点总数。
示例 1:输入: 1->2->3->4->5->NULL 输出: 1->3->5->2->4->NULL
示例 2:输入: 2->1->3->5->6->4->7->NULL  输出: 2->3->6->7->1->5->4->NULL
说明:应当保持奇数节点和偶数节点的相对顺序。
    链表的第一个节点视为奇数节点，第二个节点视为偶数节点，以此类推。
```

- 解题思路

```
func oddEvenList(head *ListNode) *ListNode {
        odd := &ListNode{}
        even := &ListNode{}
        a := odd
        b := even
        count := 1
        for head != nil {
                if count%2 == 1 {
                        a.Next = head
                        a = head
                } else {
                        b.Next = head
                        b = head
                }
                count++
                head = head.Next
        }
        b.Next = nil
        a.Next = even.Next
        return odd.Next
}

# 2
func oddEvenList(head *ListNode) *ListNode {
        odd := make([]*ListNode, 0)
        even := make([]*ListNode, 0)
        count := 1
        for head != nil {
                if count%2 == 1 {
```

(续下页)

(接上页)

```
                        odd = append(odd, head)
                } else {
                        even = append(even, head)
                }
                count++
                head = head.Next
        }
        temp := &ListNode{}
        node := temp
        for i := 0; i < len(odd); i++ {
                node.Next = odd[i]
                node = node.Next
        }
        for i := 0; i < len(even); i++ {
                node.Next = even[i]
                node = node.Next
        }
        node.Next = nil
        return temp.Next
}

# 3
func oddEvenList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        temp := head.Next // 第一个偶数
        odd, even := head, temp
        for odd.Next != nil && even.Next != nil {
                odd.Next = even.Next
                odd = odd.Next
                even.Next = odd.Next
                even = even.Next
        }
        odd.Next = temp // 第一个偶数接入奇数尾部
        return head
}
```

11.12 331. 验证二叉树的前序序列化 (2)

- 题目

```
序列化二叉树的一种方法是使用前序遍历。当我们遇到一个非空节点时，我们可以记录下这个节点的值。
如果它是一个空节点，我们可以使用一个标记值记录，例如 #。
     _9_
    /   \
   3     2
  / \   / \
 4   1  #  6
/ \ / \   / \
# # # #   # #
例如，上面的二叉树可以被序列化为字符串 "9,3,4,#,#,1,#,#,2,#,6,#,#"，其中 #␣
↪代表一个空节点。
给定一串以逗号分隔的序列，验证它是否是正确的二叉树的前序序列化。编写一个在不重构树的条件下的可行算法
每个以逗号分隔的字符或为一个整数或为一个表示 null 指针的 '#' 。
你可以认为输入格式总是有效的，例如它永远不会包含两个连续的逗号，比如 "1,,3" 。
示例 1:输入: "9,3,4,#,#,1,#,#,2,#,6,#,#" 输出: true
示例 2:输入: "1,#" 输出: false
示例 3:输入: "9,#,#,1" 输出: false
```

- 解题思路

```
func isValidSerialization(preorder string) bool {
        arr := strings.Split(preorder, ",")
        slot := 1
        for i := 0; i < len(arr); i++ {
                slot--
                if slot < 0 {
                        return false
                }
                if arr[i] != "#" {
                        slot = slot + 2
                }
        }
        return slot == 0
}

# 2
func isValidSerialization(preorder string) bool {
        arr := strings.Split(preorder, ",")
        stack := make([]string, 0)
        for i := 0; i < len(arr); i++ {
```

(续下页)

(接上页)

```
                for len(stack) > 0 && stack[len(stack)-1] == "#" && arr[i] == "#" {
                        stack = stack[:len(stack)-1]
                        if len(stack) == 0 {
                                return false
                        }
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, arr[i])
        }
        return len(stack) == 1 && stack[0] == "#"
}
```

11.13 332. 重新安排行程 (1)

- 题目

```
给定一个机票的字符串二维数组 [from,␣
↪to], 子数组中的两个成员分别表示飞机出发和降落的机场地点,
对该行程进行重新规划排序。所有这些机票都属于一个从 JFK (肯尼迪国际机场) 出发的先生,
所以该行程必须从 JFK 开始。
提示: 如果存在多种有效的行程, 请你按字符自然排序返回最小的行程组合。
例如, 行程 ["JFK", "LGA"] 与 ["JFK", "LGB"] 相比就更小, 排序更靠前
所有的机场都用三个大写字母表示 (机场代码)。
假定所有机票至少存在一种合理的行程。
所有的机票必须都用一次 且 只能用一次。
示例 1: 输入: [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]
输出: ["JFK", "MUC", "LHR", "SFO", "SJC"]
示例 2: 输入: [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]
输出: ["JFK","ATL","JFK","SFO","ATL","SFO"]
解释: 另一种有效的行程是["JFK","SFO","ATL","JFK","ATL","SFO
↪"]。但是它自然排序更大更靠后。
```

- 解题思路

```
var m map[string][]string
var res []string

func findItinerary(tickets [][]string) []string {
        res = make([]string, 0)
        m = make(map[string][]string)
        for i := 0; i < len(tickets); i++ {
                a, b := tickets[i][0], tickets[i][1]
```

(续下页)

(接上页)

```
                m[a] = append(m[a], b)
        }
        for _, v := range m {
                sort.Strings(v)
        }
        dfs("JFK")
        left, right := 0, len(res)-1
        for left < right {
                res[left], res[right] = res[right], res[left]
                left++
                right--
        }
        return res
}

func dfs(start string) {
        for len(m[start]) > 0 {
                node := m[start][0]
                m[start] = m[start][1:]
                dfs(node)
        }
        res = append(res, start)
}
```

11.14 334. 递增的三元子序列 (4)

- 题目

```
给定一个未排序的数组，判断这个数组中是否存在长度为 3 的递增子序列。
数学表达式如下:
    如果存在这样的 i, j, k,  且满足 0 ≤ i < j < k ≤ n-1,
    使得 arr[i] < arr[j] < arr[k] , 返回 true ; 否则返回 false 。
说明: 要求算法的时间复杂度为 O(n), 空间复杂度为 O(1) 。
示例 1:输入: [1,2,3,4,5] 输出: true
示例 2:输入: [5,4,3,2,1] 输出: false
```

- 解题思路

```
func increasingTriplet(nums []int) bool {
        a, b := math.MaxInt32, math.MaxInt32
        for i := 0; i < len(nums); i++ {
                if a >= nums[i] {
```

(续下页)

(接上页)

```
                        a = nums[i]
                } else if b >= nums[i] {
                        b = nums[i]
                } else {
                        return true
                }
        }
        return false
}

# 2
func increasingTriplet(nums []int) bool {
        if len(nums) < 3 {
                return false
        }
        a := make([]int, len(nums))
        b := make([]int, len(nums))
        a[0] = nums[0]
        b[len(b)-1] = nums[len(nums)-1]
        for i := 1; i < len(nums); i++ {
                a[i] = min(a[i-1], nums[i])
        }
        for i := len(nums) - 2; i >= 0; i-- {
                b[i] = max(b[i+1], nums[i])
        }
        for i := 0; i < len(nums); i++ {
                if a[i] < nums[i] && nums[i] < b[i] {
                        return true
                }
        }
        return false
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

# 3
func increasingTriplet(nums []int) bool {
        dp := make([]int, len(nums)+1)
        for i := 0; i < len(nums); i++ {
                for j := 0; j < i; j++ {
                        if nums[j] < nums[i] {
                                dp[i] = max(dp[i], dp[j]+1)
                        }
                }
                if dp[i] >= 2 {
                        return true
                }
        }
        return false
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func increasingTriplet(nums []int) bool {
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i] >= nums[j] {
                                continue
                        }
                        for k := j + 1; k < len(nums); k++ {
                                if nums[j] < nums[k] {
                                        return true
                                }
                        }
                }
        }
        return false
}
```

11.15 337. 打家劫舍 III(1)

- 题目

```
在上次打劫完一条街道之后和一圈房屋后，小偷又发现了一个新的可行窃的地区。
这个地区只有一个入口，我们称之为“根”。
除了“根”之外，每栋房子有且只有一个“父“房子与之相连。
一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。
如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。
计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。
示例 1:输入: [3,2,3,null,3,null,1]
     3
    / \
   2   3
    \   \
     3   1
输出: 7 解释: 小偷一晚能够盗取的最高金额 = 3 + 3 + 1 = 7.
示例 2:输入: [3,4,5,1,3,null,1]
     3
    / \
   4   5
  / \   \
 1   3   1
输出: 9解释: 小偷一晚能够盗取的最高金额 = 4 + 5 = 9.
```

- 解题思路

```
func rob(root *TreeNode) int {
        a, b := dfs(root)
        return max(a, b)
}

func dfs(root *TreeNode) (int, int) {
        if root == nil {
                return 0, 0
        }
        leftA, leftB := dfs(root.Left)
        rightA, rightB := dfs(root.Right)
        a := root.Val + leftB + rightB                // A=>偷
        b := max(leftA, leftB) + max(rightA, rightB) // B=>不偷
        return a, b
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}
```

11.16 338. 比特位计数 (4)

- 题目

```
给定一个非负整数 num。对于 0 ≤ i ≤ num 范围中的每个数字 i ，
计算其二进制数中的 1 的数目并将它们作为数组返回。
示例 1:输入: 2 输出: [0,1,1]
示例 2:输入: 5 输出: [0,1,1,2,1,2]
进阶:给出时间复杂度为O(n*sizeof(integer))的解答非常容易。但你可以在线性时间O(n)内用一趟扫描做到吗？
要求算法的空间复杂度为O(n)。
你能进一步完善解法吗？要求在C++或任何其他语言中不使用任何内置函数
（如 C++ 中的 __builtin_popcount）来执行此操作。
```

- 解题思路

```
func countBits(num int) []int {
        res := make([]int, num+1)
        for i := 1; i <= num; i++ {
                res[i] = res[i&(i-1)] + 1
        }
        return res
}

# 2
func countBits(num int) []int {
        res := make([]int, num+1)
        for i := 1; i <= num; i++ {
                if i%2 == 0 {
                        res[i] = res[i/2]
                } else {
                        res[i] = res[i-1] + 1
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 3
func countBits(num int) []int {
        res := make([]int, 0)
        for i := 0; i <= num; i++ {
                count := 0
                value := i
                for value != 0 {
                        if value%2 == 1 {
                                count++
                        }
                        value = value / 2
                }
                res = append(res, count)
        }
        return res
}

# 4
func countBits(num int) []int {
        res := make([]int, 0)
        for i := 0; i <= num; i++ {
                count := bits.OnesCount(uint(i))
                res = append(res, count)
        }
        return res
}
```

11.17 341. 扁平化嵌套列表迭代器 (2)

- 题目

```
给你一个嵌套的整型列表。请你设计一个迭代器，使其能够遍历这个整型列表中的所有整数。
列表中的每一项或者为一个整数，或者是另一个列表。其中列表的元素也可能是整数或是其他列表。
示例 1:输入: [[1,1],2,[1,1]] 输出: [1,1,2,1,1]
解释: 通过重复调用 next 直到 hasNext 返回 false，next 返回的元素的顺序应该是: [1,1,2,
↪1,1]。
示例 2:输入: [1,[4,[6]]] 输出: [1,4,6]
解释: 通过重复调用 next 直到 hasNext 返回 false，next 返回的元素的顺序应该是: [1,4,
↪6]。
```

- 解题思路

```
type NestedIterator struct {
        arr []*NestedInteger
}

func Constructor(nestedList []*NestedInteger) *NestedIterator {
        return &NestedIterator{arr: nestedList}
}

func (this *NestedIterator) Next() int {
        value := this.arr[0]
        this.arr = this.arr[1:]
        return value.GetInteger()
}

func (this *NestedIterator) HasNext() bool {
        if len(this.arr) == 0 {
                return false
        }
        if this.arr[0].IsInteger() {
                return true
        }
        this.arr = append(this.arr[0].GetList(), this.arr[1:]...)
        return this.HasNext()
}

# 2
type NestedIterator struct {
        arr []int
}

func Constructor(nestedList []*NestedInteger) *NestedIterator {
        arr := getList(nestedList)
        return &NestedIterator{arr: arr}
}

func getList(nestedList []*NestedInteger) []int {
        res := make([]int, 0)
        for i := 0; i < len(nestedList); i++ {
                if nestedList[i].IsInteger() == true {
                        res = append(res, nestedList[i].GetInteger())
                } else {
                        res = append(res, getList(nestedList[i].GetList())...)
                }
        }
```

(续下页)

(接上页)

```
        return res
}
func (this *NestedIterator) Next() int {
        value := this.arr[0]
        this.arr = this.arr[1:]
        return value
}

func (this *NestedIterator) HasNext() bool {
        return len(this.arr) > 0
}
```

11.18 343. 整数拆分 (2)

- 题目

```
给定一个正整数 n，将其拆分为至少两个正整数的和，并使这些整数的乘积最大化。␣
↪返回你可以获得的最大乘积。
示例 1:输入: 2 输出: 1
解释: 2 = 1 + 1, 1 × 1 = 1。
示例 2:输入: 10 输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36。
说明: 你可以假设 n 不小于 2 且不大于 58。
```

- 解题思路

```
func integerBreak(n int) int {
        if n < 2 {
                return 0
        }
        if n == 2 {
                return 1
        }
        if n == 3 {
                return 2
        }
        dp := make([]int, n+1)
        dp[0] = 0
        dp[1] = 1
        dp[2] = 2
        dp[3] = 3
        for i := 4; i <= n; i++ {
```

(续下页)

(接上页)

```
                max := 0
                for j := 1; j <= i/2; j++ {
                        length := dp[j] * dp[i-j]
                        if length > max {
                                max = length
                        }
                        dp[i] = max
                }
        }
        return dp[n]
}

#
func integerBreak(n int) int {
        if n < 2 {
                return 0
        }
        if n == 2 {
                return 1
        }
        if n == 3 {
                return 2
        }
        timesOf3 := n / 3
        if n-timesOf3*3 == 1 {
                timesOf3 = timesOf3 - 1
        }
        timesOf2 := (n - timesOf3*3) / 2
        return int(math.Pow(float64(2), float64(timesOf2))) *
                int(math.Pow(float64(3), float64(timesOf3)))
}
```

11.19 347. 前 K 个高频元素 (3)

- 题目

```
给定一个非空的整数数组，返回其中出现频率前 k 高的元素。
示例 1:输入: nums = [1,1,1,2,2,3], k = 2 输出: [1,2]
示例 2:输入: nums = [1], k = 1 输出: [1]
提示：你可以假设给定的 k 总是合理的，且 1 ≤ k ≤ 数组中不相同的元素的个数。
    你的算法的时间复杂度必须优于 O(n log n) , n 是数组的大小。
    题目数据保证答案唯一，换句话说，数组中前 k 个高频元素的集合是唯一的。
```

(续下页)

(接上页)

> 你可以按任意顺序返回答案。

- 解题思路

```
func topKFrequent(nums []int, k int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        arr := make([][2]int, 0)
        for k, v := range m {
                arr = append(arr, [2]int{k, v})
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] > arr[j][1]
        })
        res := make([]int, 0)
        for i := 0; i < k; i++ {
                res = append(res, arr[i][0])
        }
        return res
}

# 2
func topKFrequent(nums []int, k int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        var h IntHeap
        heap.Init(&h)
        for k, v := range m {
                heap.Push(&h, [2]int{k, v})
        }
        res := make([]int, 0)
        for h.Len() > 0 && k > 0 {
                k--
                node := heap.Pop(&h).([2]int)
                res = append(res, node[0])
        }
        return res
}

type IntHeap [][2]int
```

(续下页)

(接上页)

```
func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][1] > h[j][1] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([2]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 3
func topKFrequent(nums []int, k int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        arr := make([][]int, len(nums)+1)
        temp := make(map[int][]int)
        for key, value := range m {
                temp[value] = append(temp[value], key)
                arr[value] = append(arr[value], key)
        }
        res := make([]int, 0)
        for i := len(arr) - 1; i >= 0; i-- {
                // 避免出现0=>x次的情况
                if _, ok := temp[i]; ok {
                        for j := 0; j < len(arr[i]); j++ {
                                k--
                                if k < 0 {
                                        break
                                }
                                res = append(res, arr[i][j])
                        }
                }
        }
        return res
}
```

11.20 355. 设计推特 (2)

- 题目

```
设计一个简化版的推特(Twitter)，可以让用户实现发送推文，关注/取消关注其他用户，
能够看见关注人（包括自己）的最近十条推文。你的设计需要支持以下的几个功能：
postTweet(userId, tweetId): 创建一条新的推文
getNewsFeed(userId):␣
↪检索最近的十条推文。每个推文都必须是由此用户关注的人或者是用户自己发出的。
推文必须按照时间顺序由最近的开始排序。
follow(followerId, followeeId): 关注一个用户
unfollow(followerId, followeeId): 取消关注一个用户
示例:Twitter twitter = new Twitter();
// 用户1发送了一条新推文 (用户id = 1, 推文id = 5).
twitter.postTweet(1, 5);
// 用户1的获取推文应当返回一个列表，其中包含一个id为5的推文.
twitter.getNewsFeed(1);
// 用户1关注了用户2.
twitter.follow(1, 2);
// 用户2发送了一个新推文 (推文id = 6).
twitter.postTweet(2, 6);
// 用户1的获取推文应当返回一个列表，其中包含两个推文，id分别为 -> [6, 5].
// 推文id6应当在推文id5之前，因为它是在5之后发送的.
twitter.getNewsFeed(1);
// 用户1取消关注了用户2.
twitter.unfollow(1, 2);
// 用户1的获取推文应当返回一个列表，其中包含一个id为5的推文.
// 因为用户1已经不再关注用户2.
twitter.getNewsFeed(1);
```

- 解题思路

```
type Twitter struct {
        data [][2]int
        m    map[int]map[int]int
}

func Constructor() Twitter {
        return Twitter{
                data: make([][2]int, 0),
                m:    make(map[int]map[int]int),
        }
}
```

(续下页)

(接上页)

```
func (this *Twitter) PostTweet(userId int, tweetId int) {
        this.data = append(this.data, [2]int{userId, tweetId})
}

func (this *Twitter) GetNewsFeed(userId int) []int {
        res := make([]int, 0)
        for i := len(this.data) - 1; i >= 0; i-- { // 遍历发表的列表
                id, tid := this.data[i][0], this.data[i][1]
                if id == userId || this.m[userId][id] > 0 {
                        res = append(res, tid)
                }
                if len(res) == 10 {
                        return res
                }
        }
        return res
}

func (this *Twitter) Follow(followerId int, followeeId int) {
        if this.m[followerId] == nil {
                this.m[followerId] = make(map[int]int)
        }
        this.m[followerId][followeeId] = 1
}

func (this *Twitter) Unfollow(followerId int, followeeId int) {
        if this.m[followerId] != nil {
                this.m[followerId][followeeId] = 0
        }
}

# 2
type Twitter struct {
        data  map[int][][2]int
        m     map[int]map[int]int
        count int
}

func Constructor() Twitter {
        return Twitter{
                data:  make(map[int][][2]int),
                m:     make(map[int]map[int]int),
                count: 0,
```

(续下页)

(接上页)

```
        }
}

func (this *Twitter) PostTweet(userId int, tweetId int) {
        this.data[userId] = append(this.data[userId], [2]int{this.count, tweetId})
        this.count++
}

func (this *Twitter) GetNewsFeed(userId int) []int {
        res := make([]int, 0)
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := len(this.data[userId]) - 1; i >= 0; i-- {
                a, b := this.data[userId][i][0], this.data[userId][i][1]
                if intHeap.Len() == 10 && intHeap[0][0] > a {
                        break
                }
                heap.Push(&intHeap, [2]int{a, b})
                if intHeap.Len() > 10 {
                        heap.Pop(&intHeap)
                }
        }
        for k, v := range this.m[userId] {
                if k == userId || v == 0 {
                        continue
                }
                for i := len(this.data[k]) - 1; i >= 0; i-- {
                        a, b := this.data[k][i][0], this.data[k][i][1]
                        if intHeap.Len() == 10 && intHeap[0][0] > a {
                                break
                        }
                        heap.Push(&intHeap, [2]int{a, b})
                        if intHeap.Len() > 10 {
                                heap.Pop(&intHeap)
                        }
                }
        }
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([2]int)
                res = append([]int{node[1]}, res...)
        }
        return res
}
```

(续下页)

(接上页)

```
func (this *Twitter) Follow(followerId int, followeeId int) {
        if this.m[followerId] == nil {
                this.m[followerId] = make(map[int]int)
        }
        this.m[followerId][followeeId] = 1
}

func (this *Twitter) Unfollow(followerId int, followeeId int) {
        if this.m[followerId] != nil {
                this.m[followerId][followeeId] = 0
        }
}

type IntHeap [][2]int

func (h IntHeap) Len() int             { return len(h) }
func (h IntHeap) Less(i, j int) bool   { return h[i][0] < h[j][0] }
func (h IntHeap) Swap(i, j int)        { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([2]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}
```

11.21 357. 计算各个位数不同的数字个数 (3)

- 题目

```
给定一个非负整数 n，计算各位数字都不同的数字 x 的个数，其中 0 ≤ x < 10n。
示例:输入: 2 输出: 91
解释: 答案应为除去 11,22,33,44,55,66,77,88,99 外，在 [0,100) 区间内的所有数字。
```

- 解题思路

```
func countNumbersWithUniqueDigits(n int) int {
        if n == 0 {
                return 1
        }
```

(续下页)

(接上页)

```
        dp := make([]int, n+10)
        dp[1] = 10
        prev := 9
        /*
                n = 1, 1+9
                n = 2, 1+9+9*9
                n = 3, 1+9+9*9+9*9*8
                n = 4, 1+9+9*9+9*9*8+9*9*8*7
        */
        for i := 2; i <= 10; i++ {
                dp[i] = dp[i-1] + 9*prev
                prev = prev * (10 - i)
        }
        if n >= 10 {
                return dp[10]
        }
        return dp[n]
}

# 2
func countNumbersWithUniqueDigits(n int) int {
        if n == 0 {
                return 1
        }
        res := 1
        prev := 1
        for i := 1; i <= 10 && i <= n; i++ {
                res = res + 9*prev
                prev = prev * (10 - i)
        }
        return res
}

# 3
func countNumbersWithUniqueDigits(n int) int {
        if n == 0 {
                return 1
        }
        return dfs(n, 0, make([]bool, 10))
}

func dfs(n, index int, visited []bool) int {
        if index == n {
```

(续下页)

(接上页)

```
                return 0
        }
        res := 0
        for i := 0; i < 10; i++ {
                if n >= 2 && index == 1 && i == 0 {
                        continue
                }
                if visited[i] == true {
                        continue
                }
                visited[i] = true
                res = res + dfs(n, index+1, visited) + 1
                visited[i] = false
        }
        return res
}
```

11.22 365. 水壶问题 (3)

- 题目

```
有两个容量分别为 x升 和 y升 的水壶以及无限多的水。
请判断能否通过使用这两个水壶，从而可以得到恰好 z升 的水？
如果可以，最后请用以上水壶中的一或两个来盛放取得的 z升 水。
你允许：
    装满任意一个水壶
    清空任意一个水壶
    从一个水壶向另外一个水壶倒水，直到装满或者倒空
示例 1: (From the famous "Die Hard" example)
输入: x = 3, y = 5, z = 4  输出: True
示例 2:输入: x = 2, y = 6, z = 5 输出: False
```

- 解题思路

```
// ax+by=z
func canMeasureWater(x int, y int, z int) bool {
        if x > y {
                x, y = y, x
        }
        if x+y < z {
                return false
        }
```

(续下页)

(接上页)

```
        if x == 0 || y == 0 {
                return z == 0 || x+y == z
        }
        return z%gcd(x, y) == 0
}

func gcd(a, b int) int {
        if a%b == 0 {
                return b
        }
        return gcd(b, a%b)
}

# 2
func canMeasureWater(x int, y int, z int) bool {
        if z == 0 || z == x+y {
                return true
        } else if z > x+y || y == 0 {
                return false
        }
        return canMeasureWater(y, x%y, z%y)
}

# 3
func canMeasureWater(x int, y int, z int) bool {
        if x+y < z {
                return false
        }
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{0, 0})
        m := make(map[[2]int]bool)
        for len(queue) > 0 {
                a, b := queue[0][0], queue[0][1]
                queue = queue[1:]
                if m[[2]int{a, b}] == true {
                        continue
                }
                m[[2]int{a, b}] = true
                if a == z || b == z || a+b == z {
                        return true
                }
                // +x
                c, d := x, b
```

(续下页)

(接上页)

```
                queue = append(queue, [2]int{c, d})
                // +y
                c, d = a, y
                queue = append(queue, [2]int{c, d})
                // -x
                c, d = 0, b
                queue = append(queue, [2]int{c, d})
                // -y
                c, d = a, 0
                queue = append(queue, [2]int{c, d})
                // x->y
                if a > y-b {
                        c, d = a+b-y, y
                        queue = append(queue, [2]int{c, d})
                } else {
                        c, d = 0, a+b
                        queue = append(queue, [2]int{c, d})
                }
                // y->x
                if b > x-a {
                        c, d = x, a+b-x
                        queue = append(queue, [2]int{c, d})
                } else {
                        c, d = a+b, 0
                        queue = append(queue, [2]int{c, d})
                }
        }
        return false
}
```

11.23 368. 最大整除子集 (1)

- 题目

```
给出一个由无重复的正整数组成的集合，找出其中最大的整除子集，子集中任意一对 (Si，Sj)␣
↪都要满足：
Si % Sj = 0 或 Sj % Si = 0。
如果有多个目标子集，返回其中任何一个均可。
示例 1:输入: [1,2,3] 输出: [1,2] (当然, [1,3] 也正确)
示例 2:输入: [1,2,4,8] 输出: [1,2,4,8]
```

- 解题思路

```
func largestDivisibleSubset(nums []int) []int {
        n := len(nums)
        if n < 2 {
                return nums
        }
        sort.Ints(nums)
        dp := make([][]int, n)
        dp[0] = append(dp[0], nums[0])
        resLength := 0
        index := 0
        for i := 1; i < n; i++ {
                maxLength := 0
                arr := make([]int, 0)
                for j := 0; j < i; j++ {
                        // 可除以整除集合中的最大值=>属于该集合
                        if nums[i]%nums[j] == 0 && len(dp[j]) >= maxLength {
                                maxLength = len(dp[j])
                                arr = dp[j]
                        }
                }
                dp[i] = append(dp[i], append(arr, nums[i])...)
                if len(dp[i]) > resLength {
                        resLength = len(dp[i])
                        index = i
                }
        }
        return dp[index]
}
```

11.24 372. 超级次方 (2)

- 题目

```
你的任务是计算ab对1337 取模，a 是一个正整数，b␣
↪是一个非常大的正整数且会以数组形式给出。
示例 1: 输入: a = 2, b = [3] 输出: 8
示例 2: 输入: a = 2, b = [1,0] 输出: 1024
示例 3: 输入: a = 1, b = [4,3,3,8,5,2] 输出: 1
示例 4: 输入: a = 2147483647, b = [2,0,0] 输出: 1198
提示: 1 <= a <= 231 - 1
1 <= b.length <= 2000
0 <= b[i] <= 9
b 不含前导 0
```

- 解题思路

```
var mod int = 1337

func superPow(a int, b []int) int {
        a = a % mod
        if len(b) == 0 {
                return 1
        }
        x := mypow(a, b[len(b)-1])
        y := mypow(superPow(a, b[:len(b)-1]), 10)
        return x * y % mod
}

// a^n
func mypow(a int, n int) int {
        res := 1
        for n > 0 {
                if n%2 == 1 {
                        res = res * a % mod
                }
                a = a * a % mod
                n = n / 2
        }
        return res
}

# 2
var mod int = 1337

func superPow(a int, b []int) int {
        a = a % mod
        if len(b) == 0 {
                return 1
        }
        res := 1
        for i := 0; i < len(b); i++ {
                res = mypow(res, 10) * mypow(a, b[i]) % mod
        }
        return res
}

// a^n
func mypow(a int, n int) int {
        res := 1
```

(续下页)

(接上页)

```
        for n > 0 {
                if n%2 == 1 {
                        res = res * a % mod
                }
                a = a * a % mod
                n = n / 2
        }
        return res
}
```

11.25 373. 查找和最小的 K 对数字 (2)

- 题目

```
给定两个以升序排列的整形数组 nums1 和 nums2, 以及一个整数 k。
定义一对值(u,v)，其中第一个元素来自nums1，第二个元素来自 nums2。
找到和最小的 k 对数字(u1,v1), (u2,v2) ... (uk,vk)。
示例 1:输入: nums1 = [1,7,11], nums2 = [2,4,6], k = 3 输出: [1,2],[1,4],[1,6]
解释: 返回序列中的前 3 对数:
     [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
示例 2:输入: nums1 = [1,1,2], nums2 = [1,2,3], k = 2 输出: [1,1],[1,1]
解释: 返回序列中的前 2 对数:
    [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
示例 3:输入: nums1 = [1,2], nums2 = [3], k = 3  输出: [1,3],[2,3]
解释: 也可能序列中所有的数对都被返回:[1,3],[2,3]
```

- 解题思路

```
func kSmallestPairs(nums1 []int, nums2 []int, k int) [][]int {
        Heap := &NodeHeap{}
        heap.Init(Heap)
        for i := 0; i < len(nums1); i++ {
                for j := 0; j < len(nums2); j++ {
                        heap.Push(Heap, Node{
                                i: nums1[i],
                                j: nums2[j],
                        })
                        if Heap.Len() > k {
                                heap.Pop(Heap)
                        }
                }
        }
```

(续下页)

(接上页)

```
        res := make([][]int, 0)
        for Heap.Len() > 0 {
                node := heap.Pop(Heap).(Node)
                res = append(res, []int{node.i, node.j})
        }
        return res
}

type Node struct {
        i int
        j int
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

func (h NodeHeap) Less(i, j int) bool {
        return h[i].i+h[i].j > h[j].i+h[j].j
}

func (h NodeHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func kSmallestPairs(nums1 []int, nums2 []int, k int) [][]int {
        arr := make([][]int, 0)
        for i := 0; i < len(nums1); i++ {
                for j := 0; j < len(nums2); j++ {
                        arr = append(arr, []int{nums1[i], nums2[j]})
```

(续下页)

(接上页)

```
                }
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][0]+arr[i][1] < arr[j][0]+arr[j][1]
        })
        if len(arr) < k {
                return arr
        }
        return arr[:k]
}
```

11.26 375. 猜数字大小 II(3)

- 题目

```
我们正在玩一个猜数游戏，游戏规则如下：
我从1到 n 之间选择一个数字，你来猜我选了哪个数字。
每次你猜错了，我都会告诉你，我选的数字比你的大了或者小了。
然而，当你猜了数字 x 并且猜错了的时候，你需要支付金额为 x 的现金。
直到你猜到我选的数字，你才算赢得了这个游戏。
示例:n = 10, 我选择了8.
第一轮：你猜我选择的数字是5，我会告诉你，我的数字更大一些，然后你需要支付5块。
第二轮：你猜是7，我告诉你，我的数字更大一些，你支付7块。
第三轮：你猜是9，我告诉你，我的数字更小一些，你支付9块。
游戏结束。8 就是我选的数字。
你最终要支付 5 + 7 + 9 = 21 块钱。
给定n ≥ 1，计算你至少需要拥有多少现金才能确保你能赢得这个游戏。
```

- 解题思路

```
func getMoneyAmount(n int) int {
        dp := make([][]int, n+1) // 表示从[i,
↪j]之间选择一个数字来猜，你确保获胜所需要的最少现金
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
        }
        for i := n; i >= 1; i-- {
                for j := i + 1; j <= n; j++ {
                        minValue := math.MaxInt32
                        for k := i; k < j; k++ { // 可以选择[i-j]，其中最小值
                                minValue = min(minValue, max(dp[i][k-1],␣
↪dp[k+1][j])+k)
```

(续下页)

(接上页)

```
                        }
                        dp[i][j] = minValue
                }
        }
        return dp[1][n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func getMoneyAmount(n int) int {
        dp := make([][]int, n+1) // 表示从[i,
↪j]之间选择一个数字来猜，你确保获胜所需要的最少现金
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
        }
        for i := 2; i <= n; i++ {
                for j := 1; j+i-1 <= n; j++ {
                        minValue := math.MaxInt32
                        for k := j; k < i+j-1; k++ { // 可以选择[i-j]，其中最小值
                                minValue = min(minValue, max(dp[j][k-1], dp[k+1][i+j-
↪1])+k)
                        }
                        dp[j][i+j-1] = minValue
                }
        }
        return dp[1][n]
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var dp [][]int

func getMoneyAmount(n int) int {
        dp = make([][]int, n+1) // 表示从[i,
↪j]之间选择一个数字来猜，你确保获胜所需要的最少现金
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
        }
        return dfs(1, n)
}

func dfs(start, end int) int {
        if start >= end {
                return 0
        }
        if dp[start][end] > 0 {
                return dp[start][end]
        }
        minValue := math.MaxInt32
        for i := start; i <= end; i++ {
                res := i + max(dfs(i+1, end), dfs(start, i-1))
                minValue = min(minValue, res)
        }
        dp[start][end] = minValue
        return minValue
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

11.27 376. 摆动序列 (3)

- 题目

```
如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为摆动序列。
第一个差（如果存在的话）可能是正数或负数。少于两个元素的序列也是摆动序列。
例如，[1,7,4,9,2,5] 是一个摆动序列，因为差值 (6,-3,5,-7,3)是正负交替出现的。
相反，[1,4,7,2,5]和[1,7,4,5,5] 不是摆动序列，第一个序列是因为它的前两个差值都是正数，
第二个序列是因为它的最后一个差值为零。
给定一个整数序列，返回作为摆动序列的最长子序列的长度。
通过从原始序列中删除一些（也可以不删除）元素来获得子序列，剩下的元素保持其原始顺序。
示例 1:输入: [1,7,4,9,2,5] 输出: 6
解释: 整个序列均为摆动序列。
示例 2:输入: [1,17,5,10,13,15,10,5,16,8] 输出: 7
解释: 这个序列包含几个长度为 7 摆动序列，其中一个可为[1,17,10,13,10,16,8]。
示例 3:输入: [1,2,3,4,5,6,7,8,9] 输出: 2
进阶:你能否用O(n) 时间复杂度完成此题?
```

- 解题思路

```
func wiggleMaxLength(nums []int) int {
        n := len(nums)
        if n < 2 {
                return n
        }
        up := make([]int, n)    //␣
↪以前i个元素中的某一个为结尾的最长的上升摆动序列的长度
        down := make([]int, n) //␣
↪以前i个元素中的某一个为结尾的最长的下降摆动序列的长度。
        up[0] = 1
        down[0] = 1
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                if nums[i-1] < nums[i] { // 递增
                        up[i] = max(up[i-1], down[i-1]+1)
                        down[i] = down[i-1]
                } else if nums[i-1] > nums[i] { // 递减
                        up[i] = up[i-1]
                        down[i] = max(up[i-1]+1, down[i-1])
                } else {
                        up[i] = up[i-1]
                        down[i] = down[i-1]
                }
        }
        return max(up[n-1], down[n-1])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func wiggleMaxLength(nums []int) int {
        n := len(nums)
        if n < 2 {
                return n
        }
        up := 1   // 以前i个元素中的某一个为结尾的最长的上升摆动序列的长度
        down := 1 // 以前i个元素中的某一个为结尾的最长的下降摆动序列的长度。
        for i := 1; i < n; i++ {
                if nums[i-1] < nums[i] { // 递增
                        up = down + 1
                } else if nums[i-1] > nums[i] { // 递减

                        down = up + 1
                }
        }
        return max(up, down)
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}

# 3
func wiggleMaxLength(nums []int) int {
        n := len(nums)
        if n < 2 {
                return n
        }
        res := 1
        diff := nums[1] - nums[0]
        if diff != 0 {
                res = 2
        }
        for i := 2; i < n; i++ {
                newDiff := nums[i] - nums[i-1]
                if (diff >= 0 && newDiff < 0) ||
                        (diff <= 0 && newDiff > 0) {
                        res++
                        diff = newDiff
                }
        }
        return res
}
```

11.28 377. 组合总和 IV(2)

- 题目

```
给定一个由正整数组成且不存在重复数字的数组，找出和为给定目标正整数的组合的个数。
示例:nums = [1, 2, 3] target = 4
所有可能的组合为:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
请注意，顺序不同的序列被视作不同的组合。
```

(续下页)

(接上页)

因此输出为 7。
进阶：如果给定的数组中含有负数会怎么样？
问题会产生什么变化？
我们需要在题目中添加什么限制来允许负数的出现？

- 解题思路

```
func combinationSum4(nums []int, target int) int {
        // 等价于：
        // 假设你正在爬楼梯。需要n阶你才能到达楼顶。
        // 每次你可以爬num(num in nums)级台阶。
        // 你有多少种不同的方法可以爬到楼顶呢？
        dp := make([]int, target+1)
        dp[0] = 1 // 爬0楼1种解法
        for i := 1; i <= target; i++ {
                for j := 0; j < len(nums); j++ {
                        if i-nums[j] >= 0 {
                                dp[i] = dp[i] + dp[i-nums[j]]
                        }
                }
        }
        return dp[target]
}

# 2
var m map[int]int

func combinationSum4(nums []int, target int) int {
        m = make(map[int]int)
        res := dfs(nums, target)
        if res == -1 {
                return 0
        }
        return res
}

func dfs(nums []int, target int) int {
        if target == 0 {
                return 1
        }
        if target < 0 {
                return -1
        }
        if v, ok := m[target]; ok {
```

(续下页)

(接上页)

```
                return v
        }
        temp := 0
        for i := 0; i < len(nums); i++ {
                if dfs(nums, target-nums[i]) != -1 {
                        temp = temp + dfs(nums, target-nums[i])
                }
        }
        m[target] = temp
        return temp
}
```

11.29 378. 有序矩阵中第 K 小的元素 (3)

- 题目

```
给定一个 n x n 矩阵，其中每行和每列元素均按升序排序，找到矩阵中第 k 小的元素。
请注意，它是排序后的第 k 小元素，而不是第 k 个不同的元素。
示例：matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,
返回 13。
提示：你可以假设 k 的值永远是有效的，1 ≤ k ≤ n2 。
```

- 解题思路

```
func kthSmallest(matrix [][]int, k int) int {
        res := make([]int, 0)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        res = append(res, matrix[i][j])
                }
        }
        sort.Ints(res)
        return res[k-1]
}

# 2
func kthSmallest(matrix [][]int, k int) int {
```

(续下页)

(接上页)

```
        n := len(matrix)
        left, right := matrix[0][0], matrix[n-1][n-1]
        for left < right {
                mid := left + (right-left)/2
                if check(matrix, mid, k, n) {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

// 左下角查找
func check(matrix [][]int, mid, k, n int) bool {
        i := n - 1
        j := 0
        num := 0
        for i >= 0 && j < n {
                if matrix[i][j] <= mid {
                        // 往右,说明左边一列都小于mid
                        num = num + i + 1
                        j++
                } else {
                        // 往上
                        i--
                }
        }
        return num >= k
}

# 3
func kthSmallest(matrix [][]int, k int) int {
        var h IntHeap
        heap.Init(&h)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        heap.Push(&h, matrix[i][j])
                }
        }
        for h.Len() > 0 && k > 0 {
                k--
                res := heap.Pop(&h).(int)
```

(续下页)

(接上页)

```
                if k == 0 {
                        return res
                }
        }
        return 0
}

type IntHeap []int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i] < h[j] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.(int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}
```

11.30 380. 常数时间插入、删除和获取随机元素 (2)

- 题目

```
设计一个支持在平均 时间复杂度 O(1) 下，执行以下操作的数据结构。
    insert(val)：当元素 val 不存在时，向集合中插入该项。
    remove(val)：元素 val 存在时，从集合中移除该项。
    getRandom：随机返回现有集合中的一项。每个元素应该有相同的概率被返回。
示例 :
// 初始化一个空的集合。
RandomizedSet randomSet = new RandomizedSet();
// 向集合中插入 1 。返回 true 表示 1 被成功地插入。
randomSet.insert(1);
// 返回 false ，表示集合中不存在 2 。
randomSet.remove(2);
// 向集合中插入 2 。返回 true 。集合现在包含 [1,2] 。
randomSet.insert(2);
// getRandom 应随机返回 1 或 2 。
randomSet.getRandom();
// 从集合中移除 1 ，返回 true 。集合现在包含 [2] 。
randomSet.remove(1);
```

(续下页)

(接上页)

```
// 2 已在集合中，所以返回 false 。
randomSet.insert(2);
// 由于 2 是集合中唯一的数字，getRandom 总是返回 2 。
randomSet.getRandom();
```

- 解题思路

```
type RandomizedSet struct {
        m   map[int]int
        arr []int
}

func Constructor() RandomizedSet {
        return RandomizedSet{
                m:   make(map[int]int),
                arr: make([]int, 0),
        }
}

func (this *RandomizedSet) Insert(val int) bool {
        if _, ok := this.m[val]; ok {
                return false
        }
        this.arr = append(this.arr, val)
        this.m[val] = len(this.arr) - 1
        return true
}

func (this *RandomizedSet) Remove(val int) bool {
        if _, ok := this.m[val]; !ok {
                return false
        }
        index := this.m[val]
        this.arr[index], this.arr[len(this.arr)-1] = this.arr[len(this.arr)-1], this.
↪arr[index]
        this.m[this.arr[index]] = index
        this.arr = this.arr[:len(this.arr)-1]
        delete(this.m, val)
        return true
}

func (this *RandomizedSet) GetRandom() int {
        if len(this.arr) == 0 {
                return -1
```

(续下页)

(接上页)

```
        }
        index := rand.Intn(len(this.arr))
        return this.arr[index]
}

# 2
type RandomizedSet struct {
        m map[int]bool
}

func Constructor() RandomizedSet {
        return RandomizedSet{
                m: make(map[int]bool),
        }
}

func (this *RandomizedSet) Insert(val int) bool {
        if _, ok := this.m[val]; ok {
                return false
        }
        this.m[val] = true
        return true
}

func (this *RandomizedSet) Remove(val int) bool {
        if _, ok := this.m[val]; !ok {
                return false
        }
        delete(this.m, val)
        return true
}

func (this *RandomizedSet) GetRandom() int {
        if len(this.m) == 0 {
                return -1
        }
        index := rand.Intn(len(this.m))
        res := -1
        for res = range this.m {
                index--
                if index == -1 {
                        break
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

11.31 382. 链表随机节点 (1)

- 题目

```
给定一个单链表，随机选择链表的一个节点，并返回相应的节点值。保证每个节点被选的概率一样。
进阶:如果链表十分大且长度未知，如何解决这个问题？你能否使用常数级空间复杂度实现？
示例:// 初始化一个单链表 [1,2,3].
ListNode head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
Solution solution = new Solution(head);
// getRandom()方法应随机返回1,2,3中的一个，保证每个元素被返回的概率相等。
solution.getRandom();
```

- 解题思路

```
type Solution struct {
        head *ListNode
        r    *rand.Rand
}

func Constructor(head *ListNode) Solution {
        return Solution{
                head: head,
                r:    rand.New(rand.NewSource(time.Now().Unix())),
        }
}

func (this *Solution) GetRandom() int {
        res := this.head.Val
        cur := this.head
        count := 1
        for cur != nil {
                if this.r.Intn(count)+1 == count {
                        res = cur.Val
                }
                count++
                cur = cur.Next
```

(续下页)

(接上页)

```
        }
        return res
}
```

11.32 384. 打乱数组 (2)

- 题目

```
打乱一个没有重复元素的数组。
示例:
// 以数字集合 1, 2 和 3 初始化数组。
int[] nums = {1,2,3};
Solution solution = new Solution(nums);
// 打乱数组 [1,2,3] 并返回结果。任何 [1,2,3]的排列返回的概率应该相同。
solution.shuffle();
// 重设数组到它的初始状态[1,2,3]。
solution.reset();
// 随机返回数组[1,2,3]打乱后的结果。
solution.shuffle();
```

- 解题思路

```
type Solution struct {
        nums []int
}

func Constructor(nums []int) Solution {
        return Solution{nums: nums}
}

func (this *Solution) Reset() []int {
        return this.nums
}

func (this *Solution) Shuffle() []int {
        arr := make([]int, len(this.nums))
        copy(arr, this.nums)
        rand.Shuffle(len(arr), func(i, j int) {
                arr[i], arr[j] = arr[j], arr[i]
        })
        return arr
}
```

(续下页)

(接上页)

```
#
type Solution struct {
        nums []int
}

func Constructor(nums []int) Solution {
        return Solution{nums: nums}
}

func (this *Solution) Reset() []int {
        return this.nums
}

func (this *Solution) Shuffle() []int {
        arr := make([]int, len(this.nums))
        copy(arr, this.nums)
        res := make([]int, len(this.nums))
        for i := 0; i < len(res); i++{
                j := rand.Intn(len(arr))
                res[i] = arr[j]
                arr = append(arr[:j], arr[j+1:]...)
        }
        return res
}
```

11.33 385. 迷你语法分析器 (1)

- 题目

```
给定一个用字符串表示的整数的嵌套列表，实现一个解析它的语法分析器。
列表中的每个元素只可能是整数或整数嵌套列表
提示：你可以假定这些字符串都是格式良好的：
字符串非空
字符串不包含空格
字符串只包含数字0-9、[、-、,、]
示例 1：给定 s = "324",
你应该返回一个 NestedInteger 对象，其中只包含整数值 324。
示例 2：给定 s = "[123,[456,[789]]]",
返回一个 NestedInteger 对象包含一个有两个元素的嵌套列表：
1. 一个 integer 包含值 123
2. 一个包含两个元素的嵌套列表：
```

(续下页)

(接上页)

```
    i.  一个 integer 包含值 456
    ii. 一个包含一个元素的嵌套列表
         a. 一个 integer 包含值 789
```

- 解题思路

```
func deserialize(s string) *NestedInteger {
        if s[0] != '[' {
                res := &NestedInteger{}
                num, _ := strconv.Atoi(s)
                res.SetInteger(num)
                return res
        }
        stack := make([]NestedInteger, 0)
        str := ""
        for i := 0; i < len(s); i++ {
                if s[i] == '[' {
                        stack = append(stack, NestedInteger{})
                } else if s[i] == ']' {
                        if len(str) > 0 {
                                node := NestedInteger{}
                                num, _ := strconv.Atoi(str)
                                node.SetInteger(num)
                                stack[len(stack)-1].Add(node)
                        }
                        str = ""
                        if len(stack) > 1 { // 出栈
                                last := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                stack[len(stack)-1].Add(last)
                        }
                } else if s[i] == ',' {
                        if len(str) > 0 {
                                node := NestedInteger{}
                                num, _ := strconv.Atoi(str)
                                node.SetInteger(num)
                                stack[len(stack)-1].Add(node)
                        }
                        str = ""
                } else {
                        str = str + string(s[i])
                }
        }
        return &stack[len(stack)-1]
```

(续下页)

(接上页)

```
}
```

11.34 386. 字典序排数 (2)

- 题目

```
给定一个整数n, 返回从1到n的字典顺序。
例如, 给定 n =1 3, 返回 [1,10,11,12,13,2,3,4,5,6,7,8,9] 。
请尽可能的优化算法的时间复杂度和空间复杂度。 输入的数据n小于等于5,000,000。
```

- 解题思路

```
func lexicalOrder(n int) []int {
        res := make([]int, 0)
        num := 1
        for {
                if num <= n {
                        res = append(res, num)
                        num = num * 10
                } else {
                        num = num / 10
                        for num%10 == 9 {
                                num = num / 10
                        }
                        if num == 0 {
                                break
                        }
                        num++
                }
        }
        return res
}

# 2
var res []int

func lexicalOrder(n int) []int {
        res = make([]int, 0)
        for i := 1; i <= 9; i++ {
                dfs(n, i)
        }
        return res
```

(续下页)

(接上页)

```
}

func dfs(n, start int) {
        if start > n {
                return
        }
        // N叉树前序遍历
        res = append(res, start)
        for i := 0; i <= 9; i++ {
                dfs(n, start*10+i)
        }
}
```

11.35 388. 文件的最长绝对路径 (1)

- 题目

```
假设文件系统如下图所示：
这里将 dir 作为根目录中的唯一目录。dir 包含两个子目录 subdir1 和 subdir2 。
subdir1 包含文件 file1.ext 和子目录 subsubdir1；
subdir2 包含子目录 subsubdir2，该子目录下包含文件 file2.ext 。
在文本格式中，如下所示(→表示制表符)：
dir
→ subdir1
→ → file1.ext
→ → subsubdir1
→ subdir2
→ → subsubdir2
→ → → file2.ext
如果是代码表示，上面的文件系统可以写为 "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\
↪n
\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext" 。'\n' 和 '\t' 分别是换行符和制表符。
文件系统中的每个文件和文件夹都有一个唯一的 绝对路径 ，即必须打开才能到达文件/
↪目录所在位置的目录顺序，
所有路径用 '/' 连接。上面例子中，
指向 file2.ext 的绝对路径是 "dir/subdir2/subsubdir2/file2.ext" 。
每个目录名由字母、数字和/或空格组成，每个文件名遵循 name.extension 的格式，
其中名称和扩展名由字母、数字和/或空格组成。
给定一个以上述格式表示文件系统的字符串 input ，返回文件系统中 指向文件的最长绝对路径␣
↪的长度。
如果系统中没有文件，返回0。
示例 1：输入：input = "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext" 输出：20
```

(续下页)

(接上页)

```
解释：只有一个文件，绝对路径为 "dir/subdir2/file.ext" ，路径长度 20
路径 "dir/subdir1" 不含任何文件
示例 2：输入：input = "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2
\n\t\tsubsubdir2\n\t\t\tfile2.ext"
输出：32
解释：存在两个文件：
"dir/subdir1/file1.ext" ，路径长度 21
"dir/subdir2/subsubdir2/file2.ext" ，路径长度 32
返回 32 ，因为这是最长的路径
示例 3：输入：input = "a" 输出：0
解释：不存在任何文件
示例 4：输入：input = "file1.txt\nfile2.txt\nlongfile.txt" 输出：12
解释：根目录下有 3 个文件。
因为根目录中任何东西的绝对路径只是名称本身，所以答案是 "longfile.txt" ，路径长度为 12
提示：1 <= input.length <= 104
input 可能包含小写或大写的英文字母，一个换行符 '\n',
一个指表符 '\t'，一个点 '.'，一个空格 ' '，和数字。
```

- 解题思路

```
func lengthLongestPath(input string) int {
        res := 0
        arr := strings.Split(input, "\n")
        temp := make([]string, 0)
        for i := 0; i < len(arr); i++ {
                str := arr[i]
                count := strings.Count(arr[i], "\t")
                str = str[count:] // 去除\t后的字符串
                if count < len(temp) { // 多个\t的个数小于当前层级，需要去除
                        temp = temp[:count]
                }
                if strings.Contains(str, ".") {
                        length := len(str) + getLength(temp) + len(temp) //␣
↪len(temp)个分隔符
                        if length > res {
                                res = length
                        }
                } else {
                        temp = append(temp, str)
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func getLength(arr []string) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                res = res + len(arr[i])
        }
        return res
}
```

11.36 390. 消除游戏 (2)

- 题目

```
给定一个从1 到 n 排序的整数列表。
首先，从左到右，从第一个数字开始，每隔一个数字进行删除，直到列表的末尾。
第二步，在剩下的数字中，从右到左，从倒数第一个数字开始，每隔一个数字进行删除，直到列表开头。
我们不断重复这两步，从左到右和从右到左交替进行，直到只剩下一个数字。
返回长度为 n 的列表中，最后剩下的数字。
示例：输入：
n = 9,
1 2 3 4 5 6 7 8 9
2 4 6 8
2 6
6
输出:6
```

- 解题思路

```
func lastRemaining(n int) int {
        if n == 1 {
                return 1
        }
        // f(2k)/f(2k+1) = 2(k+1-f(k)), 最后奇数2k+1不影响结果
        return 2 * (n/2 + 1 - lastRemaining(n/2))
}

# 2
func lastRemaining(n int) int {
        return dfs(n, 1)
}

func dfs(n int, isLeftToRight int) int {
        if n == 1 {
```

(续下页)

(接上页)

```
                return 1
        }
        if n%2 == 1 { // 奇数
                return 2 * dfs(n/2, 1-isLeftToRight)
        }
        if isLeftToRight == 1 { // 偶数从左往右
                return 2 * dfs(n/2, 1-isLeftToRight)
        }
        // 偶数从右往左，如1，2，3，4，5，6，7，8 => 1，3，5，7
        return 2*dfs(n/2, 1-isLeftToRight) - 1
}
```

11.37 393.UTF-8 编码验证 (2)

- 题目

```
UTF-8 中的一个字符可能的长度为 1 到 4 字节，遵循以下的规则：
对于 1 字节的字符，字节的第一位设为0，后面7位为这个符号的unicode码。
对于 n 字节的字符 (n > 1)，第一个字节的前 n 位都设为1，第 n+1␣
↪位设为0，后面字节的前两位一律设为10。
剩下的没有提及的二进制位，全部为这个符号的unicode码。
这是 UTF-8 编码的工作方式：
   Char. number range  |        UTF-8 octet sequence
      (hexadecimal)    |              (binary)
   --------------------+---------------------------------------------
   0000 0000-0000 007F | 0xxxxxxx
   0000 0080-0000 07FF | 110xxxxx 10xxxxxx
   0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx
   0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
给定一个表示数据的整数数组，返回它是否为有效的 utf-8 编码。
注意:输入是整数数组。只有每个整数的最低 8␣
↪个有效位用来存储数据。这意味着每个整数只表示 1 字节的数据。
示例 1:data = [197, 130, 1]，表示 8 位的序列: 11000101 10000010 00000001.返回 true 。
这是有效的 utf-8 编码，为一个2字节字符，跟着一个1字节字符。
示例 2:data = [235, 140, 4]，表示 8 位的序列: 11101011 10001100 00000100. 返回 false␣
↪。
前 3 位都是 1 ，第 4 位为 0 表示它是一个3字节字符。
下一个字节是开头为 10 的延续字节，这是正确的。
但第二个延续字节不以 10 开头，所以是不符合规则的。
```

- 解题思路

```
func validUtf8(data []int) bool {
        count := 0
        for i := 0; i < len(data); i++ {
                if count == 0 {
                        if data[i]&0b11111000 == 0b11110000 {
                                count = 3
                        } else if data[i]&0b11110000 == 0b11100000 {
                                count = 2
                        } else if data[i]&0b11100000 == 0b11000000 {
                                count = 1
                        } else if data[i]&0b10000000 != 0b00000000 { // 其它以1开头的
                                return false
                        }
                } else {
                        if data[i]&0b10000000 != 0b10000000 {
                                return false
                        }
                        count--
                }
        }
        return count == 0
}

# 2
func validUtf8(data []int) bool {
        count := 0
        for i := 0; i < len(data); i++ {
                if count == 0 {
                        if data[i]>>3 == 0b11110 {
                                count = 3
                        } else if data[i]>>4 == 0b1110 {
                                count = 2
                        } else if data[i]>>5 == 0b110 {
                                count = 1
                        } else if data[i]>>7 != 0b0 { // 其它以1开头的
                                return false
                        }
                } else {
                        if data[i]>>6 != 0b10 {
                                return false
                        }
                        count--
                }
        }
```

(续下页)

(接上页)

```
        return count == 0
}
```

11.38 394. 字符串解码 (2)

- 题目

```
给定一个经过编码的字符串，返回它解码后的字符串。
编码规则为: k[encoded_string]，表示其中方括号内部的 encoded_string 正好重复 k 次。
注意 k 保证为正整数。
你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。
此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a␣
↪或 2[4] 的输入。
示例 1: 输入: s = "3[a]2[bc]" 输出: "aaabcbc"
示例 2: 输入: s = "3[a2[c]]" 输出: "accaccacc"
示例 3: 输入: s = "2[abc]3[cd]ef" 输出: "abcabccdcdcdef"
示例 4: 输入: s = "abc3[cd]xyz" 输出: "abccdcdcdxyz"
```

- 解题思路

```
func decodeString(s string) string {
        res := make([]byte, 0)
        numStack := make([]int, 0)
        lenStack := make([]int, 0)
        var count int
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        count = count*10 + int(s[i]-'0')
                } else if s[i] == '[' {
                        numStack = append(numStack, count)
                        count = 0
                        lenStack = append(lenStack, len(res))
                } else if s[i] == ']' {
                        c := numStack[len(numStack)-1]
                        numStack = numStack[:len(numStack)-1]
                        l := lenStack[len(lenStack)-1]
                        lenStack = lenStack[:len(lenStack)-1]
                        str := res[l:]
                        res = res[:l]
                        for i := 0; i < c; i++ {
                                res = append(res, str...)
                        }
```

(续下页)

(接上页)

```
                } else {
                        res = append(res, s[i])
                }
        }
        return string(res)
}

#
func decodeString(s string) string {
        res := ""
        count := 0
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        count = count*10 + int(s[i]-'0')
                } else if s[i] == '[' {
                        times := 0
                        i++
                        str := make([]byte, 0)
                        for s[i] != ']' || times != 0 {
                                if s[i] == '[' {
                                        times++
                                } else if s[i] == ']' {
                                        times--
                                }
                                str = append(str, s[i])
                                i++
                        }
                        temp := decodeString(string(str))
                        for j := 0; j < count; j++ {
                                res = res + (temp)
                        }
                        count = 0
                } else {
                        res = res + string(s[i])
                }
        }
        return res
}
```

11.39 395. 至少有 K 个重复字符的最长子串 (2)

- 题目

```
找到给定字符串（由小写字符组成）中的最长子串 T ， 要求 T 中的每一字符出现次数都不少于k 。输出 T 的长度。
示例 1:输入: s = "aaabb", k = 3输出: 3
最长子串为 "aaa" ，其中 'a' 重复了 3 次。
示例 2:输入:s = "ababbc", k = 2 输出:5
最长子串为 "ababb" ，其中 'a' 重复了 2 次， 'b' 重复了 3 次。
```

- 解题思路

```
func longestSubstring(s string, k int) int {
        res := 0
        m := make(map[byte]int) // 统计每个字符的个数
        for i := 0; i < len(s); i++ {
                m[s[i]]++
        }
        var split byte // 某个字符出现的次数：0<x<k
        for key, value := range m {
                if value < k {
                        split = key
                        break
                }
        }
        if split == 0 { // 字符出现次数都>=k
                return len(s)
        }
        arr := strings.Split(s, string(split)) // 以该字符切割的子串
        for i := 0; i < len(arr); i++ {
                temp := longestSubstring(arr[i], k)
                if temp > res {
                        res = temp
                }
        }
        return res
}

# 2
func longestSubstring(s string, k int) int {
        res := 0
        // 多次滑动窗口
        for i := 1; i < 26; i++ { // 枚举字符种类的个数
```

(续下页)

(接上页)

```
                m := make(map[byte]int)
                count := 0
                left := 0
                lessK := 0 // 字符出现次数<k的个数
                for right := 0; right < len(s); right++ {
                        if m[s[right]] == 0 {
                                count++
                                lessK++
                        }
                        m[s[right]]++
                        if m[s[right]] == k {
                                lessK--
                        }
                        for count > i { // 字母出现次数大于枚举的次数
                                if m[s[left]] == k {
                                        lessK++
                                }
                                m[s[left]]-- // 移动左窗口
                                if m[s[left]] == 0 {
                                        count--
                                        lessK--
                                }
                                left++
                        }
                        if lessK == 0 && right-left+1 > res { // 更新结果
                                res = right - left + 1
                        }
                }
        }
        return res
}
```

11.40 396. 旋转函数 (1)

- 题目

```
给定一个长度为 n 的整数数组A。
假设Bk是数组A顺时针旋转 k 个位置后的数组，我们定义A的“旋转函数”F为：
F(k) = 0 * Bk[0] + 1 * Bk[1] + ... + (n-1) * Bk[n-1]。
计算F(0), F(1), ..., F(n-1)中的最大值。
注意:可以认为 n 的值小于 10^5。
示例:A = [4, 3, 2, 6]
```

(续下页)

(接上页)

```
F(0) = (0 * 4) + (1 * 3) + (2 * 2) + (3 * 6) = 0 + 3 + 4 + 18 = 25
F(1) = (0 * 6) + (1 * 4) + (2 * 3) + (3 * 2) = 0 + 4 + 6 + 6 = 16
F(2) = (0 * 2) + (1 * 6) + (2 * 4) + (3 * 3) = 0 + 6 + 8 + 9 = 23
F(3) = (0 * 3) + (1 * 2) + (2 * 6) + (3 * 4) = 0 + 2 + 12 + 12 = 26
所以 F(0), F(1), F(2), F(3) 中的最大值是 F(3) = 26 。
```

- 解题思路

```
func maxRotateFunction(A []int) int {
        res := 0
        total := 0
        n := len(A)
        for i := 0; i < n; i++ {
                total = total + A[i]
                res = res + i*A[i]
        }
        temp := res
        for i := 0; i < n; i++ {
                // 最后移动到第一位，其他右移一位
                temp = temp + total      // 每一位都加一次
                temp = temp - n*A[n-1-i] // 最后一位删除
                if temp > res {
                        res = temp
                }
        }
        return res
}
```

11.41 397. 整数替换 (2)

- 题目

```
给定一个正整数 n，你可以做如下操作：
1. 如果 n 是偶数，则用 n / 2替换 n。
2. 如果 n 是奇数，则可以用 n + 1或n - 1替换 n。
n 变为 1 所需的最小替换次数是多少？
示例 1:输入: 8输出: 3
解释:8 -> 4 -> 2 -> 1
示例 2:输入: 7输出: 4
解释:7 -> 8 -> 4 -> 2 -> 1 或7 -> 6 -> 3 -> 2 -> 1
```

- 解题思路

```
func integerReplacement(n int) int {
        if n < 3 {
                return n - 1
        }
        if n == math.MinInt32 {
                return 32
        }
        if n%2 == 0 {
                return integerReplacement(n/2) + 1
        }
        a := integerReplacement(n+1) + 1
        b := integerReplacement(n-1) + 1
        if a > b {
                return b
        }
        return a
}

# 2
var m map[int]int

func integerReplacement(n int) int {
        m = make(map[int]int)
        return dfs(n)
}

func dfs(n int) int {
        if m[n] > 0 {
                return m[n]
        }
        if n == 1 {
                return 0
        }
        if n%2 == 0 {
                m[n] = dfs(n/2) + 1
        } else {
                m[n] = min(dfs(n-1), dfs(n+1)) + 1
        }
        return m[n]
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}
```

11.42 398. 随机数索引 (2)

- 题目

```
给定一个可能含有重复元素的整数数组，要求随机输出给定的数字的索引。
↪您可以假设给定的数字一定存在于数组中。
注意：数组大小可能非常大。 使用太多额外空间的解决方案将不会通过测试。
示例:int[] nums = new int[] {1,2,3,3,3};
Solution solution = new Solution(nums);
// pick(3) 应该返回索引 2,3 或者 4。每个索引的返回概率应该相等。
solution.pick(3);
// pick(1) 应该返回 0。因为只有nums[0]等于1。
solution.pick(1);
```

- 解题思路

```
type Solution struct {
        m map[int][]int
        r *rand.Rand
}

func Constructor(nums []int) Solution {
        res := Solution{
                m: make(map[int][]int),
                r: rand.New(rand.NewSource(time.Now().Unix())),
        }
        for i := 0; i < len(nums); i++ {
                res.m[nums[i]] = append(res.m[nums[i]], i)
        }
        return res
}

func (this *Solution) Pick(target int) int {
        arr := this.m[target]
        index := this.r.Intn(len(arr))
        return arr[index]
}
```

(续下页)

(接上页)

```
# 2
type Solution struct {
        nums []int
        r    *rand.Rand
}

func Constructor(nums []int) Solution {
        return Solution{
                nums: nums,
                r:    rand.New(rand.NewSource(time.Now().Unix())),
        }
}

func (this *Solution) Pick(target int) int {
        res := 0
        count := 1
        for i := 0; i < len(this.nums); i++ {
                if this.nums[i] == target {
            // 蓄水池采样法
                        if this.r.Intn(count)+1 == count {
                                res = i
                        }
                        count++
                }
        }
        return res
}
```

11.43 399. 除法求值 (3)

- 题目

```
给出方程式A / B = k, 其中A 和B 均为用字符串表示的变量, k 是一个浮点型数字。
根据已知方程式求解问题, 并返回计算结果。如果结果不存在, 则返回-1.0。
输入总是有效的。你可以假设除法运算中不会出现除数为 0 的情况, 且不存在任何矛盾的结果。
示例 1: 输入: equations = [["a","b"],["b","c"]], values = [2.0,3.0],
queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]
输出: [6.00000,0.50000,-1.00000,1.00000,-1.00000]
解释: 给定: a / b = 2.0, b / c = 3.0
问题: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
返回: [6.0, 0.5, -1.0, 1.0, -1.0 ]
示例 2: 输入: equations = [["a","b"],["b","c"],["bc","cd"]],
```

(续下页)

(接上页)

```
values = [1.5,2.5,5.0], queries = [["a","c"],["c","b"],["bc","cd"],["cd","bc"]]
输出：[3.75000,0.40000,5.00000,0.20000]
示例 3：输入：equations = [["a","b"]], values = [0.5],
queries = [["a","b"],["b","a"],["a","c"],["x","y"]]
输出：[0.50000,2.00000,-1.00000,-1.00000]
提示：1 <= equations.length <= 20
equations[i].length == 2
1 <= equations[i][0].length, equations[i][1].length <= 5
values.length ==equations.length
0.0 <values[i] <= 20.0
1 <= queries.length <= 20
queries[i].length == 2
1 <= queries[i][0].length, queries[i][1].length <= 5
equations[i][0], equations[i][1],queries[i][0], queries[i][1] 由小写英文字母与数字组成
```

- 解题思路

```
type Node struct {
        to    int
        value float64
}

func calcEquation(equations [][]string, values []float64, queries [][]string)␣
↪[]float64 {
        m := make(map[string]int) // 计算对应的id
        for i := 0; i < len(equations); i++ {
                a, b := equations[i][0], equations[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = len(m)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = len(m)
                }
        }
        arr := make([][]Node, len(m)) // 邻接表
        for i := 0; i < len(equations); i++ {
                a, b := m[equations[i][0]], m[equations[i][1]]
                arr[a] = append(arr[a], Node{to: b, value: values[i]})
                arr[b] = append(arr[b], Node{to: a, value: 1 / values[i]}) // 除以
        }
        res := make([]float64, len(queries))
        for i := 0; i < len(queries); i++ {
                a, okA := m[queries[i][0]]
                b, okB := m[queries[i][1]]
```

(续下页)

(接上页)

```
                if okA == false || okB == false {
                        res[i] = -1
                } else {
                        res[i] = bfs(arr, a, b) // 广度优先查找
                }
        }
        return res
}

func bfs(arr [][]Node, start, end int) float64 {
        temp := make([]float64, len(arr)) // 结果的比例
        temp[start] = 1
        queue := make([]int, 0)
        queue = append(queue, start)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node == end {
                        return temp[node]
                }
                for i := 0; i < len(arr[node]); i++ {
                        next := arr[node][i].to
                        if temp[next] == 0 {
                                temp[next] = temp[node] * arr[node][i].value
                                queue = append(queue, next)
                        }
                }
        }
        return -1
}

# 2
func calcEquation(equations [][]string, values []float64, queries [][]string)␣
↪[]float64 {
        m := make(map[string]int) // 计算对应的id
        for i := 0; i < len(equations); i++ {
                a, b := equations[i][0], equations[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = len(m)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = len(m)
                }
```

(续下页)

(接上页)

```
        }
        arr := make([][]float64, len(m)) // 邻接矩阵
        for i := 0; i < len(m); i++ {
                arr[i] = make([]float64, len(m))
        }
        for i := 0; i < len(equations); i++ {
                a, b := m[equations[i][0]], m[equations[i][1]]
                arr[a][b] = values[i]
                arr[b][a] = 1 / values[i]
        }
        for k := 0; k < len(arr); k++ { // Floyd
                for i := 0; i < len(arr); i++ {
                        for j := 0; j < len(arr); j++ {
                                if arr[i][k] > 0 && arr[k][j] > 0 {
                                        arr[i][j] = arr[i][k] * arr[k][j]
                                }
                        }
                }
        }
        res := make([]float64, len(queries))
        for i := 0; i < len(queries); i++ {
                a, okA := m[queries[i][0]]
                b, okB := m[queries[i][1]]
                if okA == false || okB == false || arr[a][b] == 0 {
                        res[i] = -1
                } else {
                        res[i] = arr[a][b]
                }
        }
        return res
}

# 3
func calcEquation(equations [][]string, values []float64, queries [][]string)␣
↪[]float64 {
        m := make(map[string]int) // 计算对应的id
        for i := 0; i < len(equations); i++ {
                a, b := equations[i][0], equations[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = len(m)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = len(m)
```

(续下页)

(接上页)

```
                }
        }
        fa, rank = Init(len(m))
        for i := 0; i < len(equations); i++ {
                a, b := m[equations[i][0]], m[equations[i][1]]
                union(a, b, values[i])
        }
        res := make([]float64, len(queries))
        for i := 0; i < len(queries); i++ {
                a, okA := m[queries[i][0]]
                b, okB := m[queries[i][1]]
                if okA == true && okB == true && find(a) == find(b) {
                        res[i] = rank[a] / rank[b]
                } else {
                        res[i] = -1
                }
        }
        return res
}

var fa []int
var rank []float64

// 初始化
func Init(n int) ([]int, []float64) {
        arr := make([]int, n)
        r := make([]float64, n)
        for i := 0; i < n; i++ {
                arr[i] = i
                r[i] = 1
        }
        return arr, r
}

// 查询
func find(x int) int {
        // 彻底路径压缩
        if fa[x] != x {
                origin := fa[x]
                fa[x] = find(fa[x])
                rank[x] = rank[x] * rank[origin] // 秩处理是难点
        }
        return fa[x]
```

(续下页)

(接上页)

```
}

// 合并
func union(i, j int, value float64) {
        x, y := find(i), find(j)
        rank[x] = value * rank[j] / rank[i] // 秩处理是难点
        fa[x] = y
}
```

11.44 400. 第 N 个数字 (2)

- 题目

```
在无限的整数序列 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...中找到第 n 个数字。
注意:n 是正数且在32位整数范围内 ( n < 231)。
示例 1:输入:3输出:3
示例 2:输入:11输出:0
说明:第11个数字在序列 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... 里是0, 它是10的一部分。
```

- 解题思路

```
func findNthDigit(n int) int {
        if n < 0 {
                return -1
        }
        digits := 1
        for {
                numbers := countOfIntegers(digits)
                if n < numbers*digits {
                        return digitAtIndex(n, digits)
                }
                n = n - numbers*digits
                digits++
        }
}

func countOfIntegers(n int) int {
        if n == 1 {
                return 10
        }
        count := math.Pow(float64(10), float64(n-1))
        return 9 * int(count)
```

(续下页)

(接上页)

```
}

func digitAtIndex(n, digits int) int {
        number := beginNumber(digits) + n/digits
        indexFromRight := digits - n%digits
        for i := 1; i < indexFromRight; i++ {
                number = number / 10
        }
        return number % 10
}

func beginNumber(digits int) int {
        if digits == 1 {
                return 0
        }
        return int(math.Pow(float64(10), float64(digits-1)))
}

# 2
func findNthDigit(n int) int {
        if n < 10 {
                return n
        }
        digits := 1
        count := 9
        number := 1
        for n-digits*count > 0 {
                n = n - digits*count
                digits++
                count = count * 10
                number = number * 10
        }
        number = number + (n-1)/digits
        index := (n - 1) % digits
        str := strconv.Itoa(number)
        return int(str[index] - '0')
}
```

CHAPTER 12

0301-0400-Hard

12.1 301. 删除无效的括号 (2)

- 题目

```
删除最小数量的无效括号，使得输入的字符串有效，返回所有可能的结果。
说明: 输入可能包含了除 ( 和 ) 以外的字符。
示例 1:输入: "()())()" 输出: ["()()()", "(())()"]
示例 2:输入: "(a)())()" 输出: ["(a)()()", "(a())()"]
示例 3:输入: ")(" 输出: [""]
```

- 解题思路

```
func removeInvalidParentheses(s string) []string {
        res := make([]string, 0)
        cur := make(map[string]int)
        cur[s] = 1
        for {
                for k := range cur {
                        if isValid(k) {
                                res = append(res, k)
                        }
                }
                if len(res) > 0 {
                        return res
```

(续下页)

(接上页)

```
                }
                next := make(map[string]int)
                for k := range cur {
                        for i := range k {
                                if k[i] == '(' || k[i] == ')' {
                                        str := k[:i] + k[i+1:]
                                        next[str] = 1
                                }
                        }
                }
                cur = next
        }
}

func isValid(s string) bool {
        left := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        left++
                } else if s[i] == ')' {
                        left--
                }
                if left < 0 {
                        return false
                }
        }
        return left == 0
}

# 2
var m map[string]bool
var max int

func removeInvalidParentheses(s string) []string {
        m = make(map[string]bool)
        res := make([]string, 0)
        max = 0
        dfs(s, 0, 0, "")
        for k := range m {
                res = append(res, k)
        }
        return res
}
```

(续下页)

(接上页)

```
func dfs(s string, start, count int, temp string) {
        if count < 0 {
                return
        }
        if start == len(s) {
                if count == 0 {
                        if len(temp) > max {
                                max = len(temp)
                                m = make(map[string]bool)
                                m[temp] = true
                        } else if max == len(temp) {
                                m[temp] = true
                        }
                }
                return
        }
        if s[start] == '(' {
                dfs(s, start+1, count+1, temp+"(")
        } else if s[start] == ')' {
                dfs(s, start+1, count-1, temp+")")
        } else {
                dfs(s, start+1, count, temp+string(s[start]))
        }
        if s[start] == '(' || s[start] == ')' {
                dfs(s, start+1, count, temp)
        }
}
```

12.2 312. 戳气球 (3)

- 题目

```
有 n 个气球，编号为0 到 n-1，每个气球上都标有一个数字，这些数字存在数组 nums 中。
现在要求你戳破所有的气球。如果你戳破气球 i ，就可以获得 nums[left] * nums[i] *␣
↪nums[right] 个硬币。
这里的 left 和 right 代表和 i 相邻的两个气球的序号。
注意当你戳破了气球 i 后，气球 left 和气球 right 就变成了相邻的气球。
求所能获得硬币的最大数量。
说明:你可以假设 nums[-1] = nums[n] = 1，但注意它们不是真实存在的所以并不能被戳破。
    0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100
示例:输入: [3,1,5,8] 输出: 167
```

(续下页)

(接上页)

```
解释: nums = [3,1,5,8] --> [3,5,8] -->   [3,8]   -->  [8]  --> []
     coins =  3*1*5      +  3*5*8    +  1*3*8      + 1*8*1   = 167
```

- 解题思路

```
var res [][]int
var arr []int

func maxCoins(nums []int) int {
        n := len(nums)
        arr = make([]int, n+2)
        arr[0], arr[len(arr)-1] = 1, 1
        for i := 1; i <= n; i++ {
                arr[i] = nums[i-1]
        }
        res = make([][]int, n+2)
        for i := 0; i < len(res); i++ {
                res[i] = make([]int, n+2)
                for j := 0; j < len(res[i]); j++ {
                        res[i][j] = -1
                }
        }
        return dfs(0, n+1)
}

// 将开区间(i,j)内的位置全部填满气球能够得到的最多硬币数
func dfs(left, right int) int {
        // 不满足3个
        if left+1 >= right {
                return 0
        }
        if res[left][right] != -1 {
                return res[left][right]
        }
        for i := left + 1; i < right; i++ {
                // 填充第i位，两边是arr[left],arr[right]
                sum := arr[left] * arr[i] * arr[right]
                sum = sum + dfs(left, i) + dfs(i, right)
                res[left][right] = max(res[left][right], sum)
        }
        return res[left][right]
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 2
func maxCoins(nums []int) int {
        n := len(nums)
        arr := make([]int, n+2)
        arr[0], arr[len(arr)-1] = 1, 1
        for i := 1; i <= n; i++ {
                arr[i] = nums[i-1]
        }
        dp := make([][]int, n+2)
        for i := 0; i < len(dp); i++ {
                dp[i] = make([]int, n+2)
        }
        // dp[i][j] 表示填满开区间(i,j)能得到的最多硬币数
        // i => left
        // k => i
        // j => right
        // i不能0->n+1
        for i := n - 1; i >= 0; i-- {
                for j := i + 2; j <= n+1; j++ {
                        for k := i + 1; k < j; k++ {
                                sum := arr[i] * arr[k] * arr[j]
                                sum = sum + dp[i][k] + dp[k][j]
                                dp[i][j] = max(dp[i][j], sum)
                        }
                }
        }
        return dp[0][n+1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
```

(续下页)

(接上页)

```
func maxCoins(nums []int) int {
        n := len(nums)
        arr := make([]int, n+2)
        arr[0], arr[len(arr)-1] = 1, 1
        for i := 1; i <= n; i++ {
                arr[i] = nums[i-1]
        }
        dp := make([][]int, n+2)
        for i := 0; i < len(dp); i++ {
                dp[i] = make([]int, n+2)
        }
        // dp[i][j] 表示填满开区间(i,j)能得到的最多硬币数
        // i => left
        // k => i
        // j => right
        for j := 2; j <= n+1; j++ {
                for i := j - 2; i >= 0; i-- {
                        for k := i + 1; k < j; k++ {
                                sum := arr[i] * arr[k] * arr[j]
                                sum = sum + dp[i][k] + dp[k][j]
                                dp[i][j] = max(dp[i][j], sum)
                        }
                }
        }
        return dp[0][n+1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

12.3 315. 计算右侧小于当前元素的个数 (3)

- 题目

```
给定一个整数数组 nums，按要求返回一个新数组counts。
数组 counts 有该性质： counts[i] 的值是 nums[i] 右侧小于nums[i] 的元素的数量。
示例：输入：nums = [5,2,6,1] 输出：[2,1,1,0]
解释：5 的右侧有 2 个更小的元素 (2 和 1)
```

(续下页)

(接上页)

```
2 的右侧仅有 1 个更小的元素 (1)
6 的右侧有 1 个更小的元素 (1)
1 的右侧有 0 个更小的元素
提示：0 <= nums.length <= 10^5
-10^4<= nums[i] <= 10^4
```

- 解题思路

```
func countSmaller(nums []int) []int {
        n := len(nums)
        res := make([]int, n)
        arr, _ := getLSH(nums)
        length = n
        c = make([]int, n+1)
        for i := n - 1; i >= 0; i-- {
                res[i] = getSum(arr[i] - 1)
                upData(arr[i], 1)
        }
        return res
}

func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var length int
var c []int // 树状数组
```

(续下页)

(接上页)

```
func lowBit(x int) int {
        return x & (-x)
}

// 单点修改
func upData(i, k int) { // 在i位置加上k
        for i <= length {
                c[i] = c[i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(i int) int {
        res := 0
        for i > 0 {
                res = res + c[i]
                i = i - lowBit(i)
        }
        return res
}

# 2
type Node struct {
        Value int
        Index int
}

var res []int

func countSmaller(nums []int) []int {
        n := len(nums)
        res = make([]int, n)
        arr := make([]Node, 0)
        for i := 0; i < n; i++ {
                arr = append(arr, Node{
                        Value: nums[i],
                        Index: i,
                })
        }
        mergeSort(arr)
        return res
```

(续下页)

(接上页)

```
}

func mergeSort(nums []Node) {
        n := len(nums)
        if n <= 1 {
                return
        }
        a := append([]Node{}, nums[:n/2]...)
        b := append([]Node{}, nums[n/2:]...)
        mergeSort(a) // a已经有序
        mergeSort(b) // b已经有序
        i, j := 0, 0
        for i = 0; i < len(a); i++ {
                for j < len(b) && a[i].Value > b[j].Value { // 统计比右边多多少个数
                        j++
                }
                res[a[i].Index] = res[a[i].Index] + j // 找到下标，然后加上次数
        }
        i, j = 0, 0
        for k := 0; k < len(nums); k++ { // 2个有序数组合并
                if i < len(a) && (j == len(b) || a[i].Value <= b[j].Value) {
                        nums[k] = a[i]
                        i++
                } else {
                        nums[k] = b[j]
                        j++
                }
        }
        return
}

# 3
func countSmaller(nums []int) []int {
        n := len(nums)
        res := make([]int, n)
        tempArr, m := getLSH(nums)
        total := len(tempArr)
        arr = make([]int, 4*total+1)
        for i := n - 1; i >= 0; i-- {
                index := m[nums[i]]
                res[i] = query(0, 1, total, 1, index-1)
                update(0, 1, total, index, 1)
        }
```

(续下页)

(接上页)

```
        return res
}

func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var arr []int // 线段树

func update(id int, left, right, x int, value int) {
        if left > x || right < x {
                return
        }
        arr[id] = arr[id] + value
        if left == right {
                return
        }
        mid := left + (right-left)/2
        update(2*id+1, left, mid, x, value)    // 左节点
        update(2*id+2, mid+1, right, x, value) // 右节点
}

func query(id int, left, right, queryLeft, queryRight int) int {
        if left > queryRight || right < queryLeft {
                return 0
        }
```

(续下页)

(接上页)

```
        if queryLeft <= left && right <= queryRight {
                return arr[id]
        }
        mid := left + (right-left)/2
        return query(id*2+1, left, mid, queryLeft, queryRight) +
                query(id*2+2, mid+1, right, queryLeft, queryRight)
}
```

12.4 316. 去除重复字母 (2)

- 题目

```
给你一个仅包含小写字母的字符串，请你去除字符串中重复的字母，使得每个字母只出现一次。
需保证返回结果的字典序最小（要求不能打乱其他字符的相对位置）。
示例 1:输入: "bcabc" 输出: "abc"
示例 2:输入: "cbacdcbc" 输出: "acdb"
注意: 该题与 1081
https://leetcode.cn/problems/smallest-subsequence-of-distinct-characters 相同
```

- 解题思路

```
func removeDuplicateLetters(s string) string {
        stack := make([]byte, 0)
        arr := [256]byte{}
        m := make(map[byte]bool)
        for i := 0; i < len(s); i++ {
                arr[s[i]]++
        }
        for i := 0; i < len(s); i++ {
                if m[s[i]] == true {
                        arr[s[i]]--
                        continue
                }
                // arr[栈顶]说明有重复元素
                // 栈顶>s[i]:说明字典序不满足
                for len(stack) > 0 && stack[len(stack)-1] > s[i] &&␣
↪arr[stack[len(stack)-1]] > 0 {
                        m[stack[len(stack)-1]] = false
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, s[i])
                arr[s[i]]--
```

(续下页)

(接上页)

```
                m[s[i]] = true
        }
        return string(stack)
}

# 2
func removeDuplicateLetters(s string) string {
        arr := [26]int{}
        pos := 0
        for i := 0; i < len(s); i++ {
                arr[s[i]-'a']++
        }
        for i := 0; i < len(s); i++ {
                if s[i] < s[pos] {
                        pos = i
                }
                arr[s[i]-'a']--
                if arr[s[i]-'a'] == 0 {
                        break
                }
        }
        if len(s) == 0 {
                return ""
        }
        newStr := strings.ReplaceAll(s[pos+1:], string(s[pos]), "")
        return string(s[pos]) + removeDuplicateLetters(newStr)
}
```

12.5 321. 拼接最大数 (1)

- 题目

```
给定长度分别为m和n的两个数组，其元素由0-9构成，表示两个自然数各位上的数字。
现在从这两个数组中选出 k (k <= m + n)个数字拼接成一个新的数，
要求从同一个数组中取出的数字保持其在原数组中的相对顺序。
求满足该条件的最大数。结果返回一个表示该最大数的长度为k的数组。
说明: 请尽可能地优化你算法的时间和空间复杂度。
示例1:输入:nums1 = [3, 4, 6, 5] nums2 = [9, 1, 2, 5, 8, 3] k = 5
输出:[9, 8, 6, 5, 3]
示例 2:输入:nums1 = [6, 7]nums2 = [6, 0, 4]k = 5
输出:[6, 7, 6, 0, 4]
示例 3:输入:nums1 = [3, 9]nums2 = [8, 9] k = 3
```

(续下页)

(接上页)

```
输出:[9, 8, 9]
```

- 解题思路

```
func maxNumber(nums1 []int, nums2 []int, k int) []int {
        res := make([]int, k)
        for i := 0; i <= k; i++ {
                if i <= len(nums1) && k-i <= len(nums2) {
                        a := check(nums1, i)
                        b := check(nums2, k-i)
                        c := merge(a, b)
                        if compare(res, c) == true {
                                res = c
                        }
                }
        }
        return res
}

func check(num []int, k int) []int {
        stack := make([]int, 0)
        count := len(num) - k
        for i := 0; i < len(num); i++ {
                for len(stack) > 0 && count > 0 && stack[len(stack)-1] < num[i] {
                        stack = stack[:len(stack)-1]
                        count--
                }
                stack = append(stack, num[i])
        }
        return stack[:k]
}

func merge(nums1, nums2 []int) []int {
        res := make([]int, len(nums1)+len(nums2))
        if len(nums1) == 0 {
                copy(res, nums2)
                return res
        }
        if len(nums2) == 0 {
                copy(res, nums1)
                return res
        }
        for i := 0; i < len(res); i++ {
                if compare(nums1, nums2) {
```

(续下页)

(接上页)

```
                        res[i], nums2 = nums2[0], nums2[1:]
                } else {
                        res[i], nums1 = nums1[0], nums1[1:]
                }
        }
        return res
}

func compare(nums1, nums2 []int) bool {
        for i := 0; i < len(nums1) && i < len(nums2); i++ {
                if nums1[i] != nums2[i] {
                        return nums1[i] < nums2[i]
                }
        }
        return len(nums1) < len(nums2)
}
```

12.6 327. 区间和的个数 (3)

- 题目

```
给你一个整数数组nums 以及两个整数lower 和 upper 。
求数组中，值位于范围 [lower, upper] （包含lower和upper）之内的 区间和的个数 。
区间和S(i, j)表示在nums中，位置从i到j的元素之和，包含i和j(i ≤ j)。
示例 1：输入：nums = [-2,5,-1], lower = -2, upper = 2 输出：3
解释：存在三个区间：[0,0]、[2,2] 和 [0,2] ，对应的区间和分别是：-2 、-1 、2 。
示例 2：输入：nums = [0], lower = 0, upper = 0 输出：1
提示：1 <= nums.length <= 105
-231 <= nums[i] <= 231 - 1
-105 <= lower <= upper <= 105
题目数据保证答案是一个 32 位 的整数
```

- 解题思路

```
func countRangeSum(nums []int, lower int, upper int) int {
        n := len(nums)
        arr := make([]int, n+1) // 前缀和
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + nums[i]
        }
        return countSum(arr, lower, upper)
}
```

(续下页)

(接上页)

```
func countSum(nums []int, lower int, upper int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        a := append([]int{}, nums[:n/2]...)
        b := append([]int{}, nums[n/2:]...)
        res := countSum(a, lower, upper) + countSum(b, lower, upper)
        left, right := 0, 0
        for i := 0; i < len(a); i++ {
                for left < len(b) && b[left]-a[i] < lower {
                        left++
                }
                for right < len(b) && b[right]-a[i] <= upper {
                        right++
                }
                res = res + right - left
        }
        i, j := 0, 0
        for k := 0; k < len(nums); k++ { // 2个有序数组合并
                if i < len(a) && (j == len(b) || a[i] <= b[j]) {
                        nums[k] = a[i]
                        i++
                } else {
                        nums[k] = b[j]
                        j++
                }
        }
        return res
}

# 2
func countRangeSum(nums []int, lower int, upper int) int {
        n := len(nums)
        preSum := make([]int, n+1) // 前缀和
        temp := make([]int, 0)
        temp = append(temp, 0) // 注意0
        for i := 0; i < n; i++ {
                preSum[i+1] = preSum[i] + nums[i]
                temp = append(temp, preSum[i+1], preSum[i+1]-lower, preSum[i+1]-upper)
        }
        tempArr, m := getLSH(temp) // 离散化
```

(续下页)

(接上页)

```
        total := len(tempArr)
        arr = make([]int, 4*total+1)
        update(0, 1, total, m[0], 1) // 注意0
        res := 0
        for i := 1; i < len(preSum); i++ {
                left := m[preSum[i]-upper]
                right := m[preSum[i]-lower]
                index := m[preSum[i]]
                count := query(0, 1, total, left, right)
                res = res + count
                update(0, 1, total, index, 1)
        }
        return res
}

// 离散化
func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var arr []int // 线段树

func update(id int, left, right, x int, value int) {
        if left > x || right < x {
                return
        }
```

(续下页)

(接上页)

```
        arr[id] = arr[id] + value
        if left == right {
                return
        }
        mid := left + (right-left)/2
        update(2*id+1, left, mid, x, value)    // 左节点
        update(2*id+2, mid+1, right, x, value) // 右节点
}

func query(id int, left, right, queryLeft, queryRight int) int {
        if left > queryRight || right < queryLeft {
                return 0
        }
        if queryLeft <= left && right <= queryRight {
                return arr[id]
        }
        mid := left + (right-left)/2
        return query(id*2+1, left, mid, queryLeft, queryRight) +
                query(id*2+2, mid+1, right, queryLeft, queryRight)
}

# 3
func countRangeSum(nums []int, lower int, upper int) int {
        n := len(nums)
        preSum := make([]int, n+1) // 前缀和
        temp := make([]int, 0)
        temp = append(temp, 0) // 注意0
        for i := 0; i < n; i++ {
                preSum[i+1] = preSum[i] + nums[i]
                temp = append(temp, preSum[i+1], preSum[i+1]-lower, preSum[i+1]-upper)
        }
        tempArr, m := getLSH(temp) // 离散化
        length = len(tempArr)
        c = make([]int, length+1)
        upData(m[0], 1) // 注意0
        res := 0
        for i := 1; i < len(preSum); i++ {
                left := m[preSum[i]-upper]
                right := m[preSum[i]-lower]
                index := m[preSum[i]]
                count := getSum(right) - getSum(left-1)
                res = res + count
                upData(index, 1)
```

(续下页)

(接上页)

```
        }
        return res
}

// 离散化
func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var length int
var c []int // 树状数组

func lowBit(x int) int {
        return x & (-x)
}

// 单点修改
func upData(i, k int) { // 在i位置加上k
        for i <= length {
                c[i] = c[i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(i int) int {
```

(续下页)

(接上页)

```
        res := 0
        for i > 0 {
                res = res + c[i]
                i = i - lowBit(i)
        }
        return res
}
```

12.7 329. 矩阵中的最长递增路径 (3)

- 题目

```
给定一个m x n 整数矩阵matrix ，找出其中 最长递增路径 的长度。
对于每个单元格，你可以往上，下，左，右四个方向移动。 你 不能 在 对角线␣
↪方向上移动或移动到 边界外（即不允许环绕）。
示例 1：输入：matrix = [[9,9,4],[6,6,8],[2,1,1]] 输出：4
解释：最长递增路径为[1, 2, 6, 9]。
示例 2：输入：matrix = [[3,4,5],[3,2,6],[2,2,1]]输出：4
解释：最长递增路径是[3, 4, 5, 6]。注意不允许在对角线方向上移动。
示例 3：输入：matrix = [[1]]输出：1
提示：m == matrix.length
n == matrix[i].length
1 <= m, n <= 200
0 <= matrix[i][j] <= 231 - 1
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var n, m int
var arr [][]int

func longestIncreasingPath(matrix [][]int) int {
        n, m = len(matrix), len(matrix[0])
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        res = max(res, dfs(matrix, i, j))
```

(续下页)

(接上页)

```
                }
        }
        return res
}

func dfs(matrix [][]int, i, j int) int {
        if arr[i][j] != 0 {
                return arr[i][j]
        }
        arr[i][j]++ // 长度为1
        for k := 0; k < 4; k++ {
                x, y := i+dx[k], j+dy[k]
                if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y] > matrix[i][j] {
                        arr[i][j] = max(arr[i][j], dfs(matrix, x, y)+1)
                }
        }
        return arr[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func longestIncreasingPath(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        queue := make([][2]int, 0) // 从最大数开始广度优先搜索
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        for k := 0; k < 4; k++ {
                                x, y := i+dx[k], j+dy[k]
                                if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y]␣
↪> matrix[i][j] {
```

(续下页)

(接上页)

```
                                        arr[i][j]++ // 四周大于当前的个数
                                }
                        }
                        if arr[i][j] == 0 { // 四周没有大于当前的数
                                queue = append(queue, [2]int{i, j})
                        }
                }
        }
        res := 0
        for len(queue) > 0 {
                res++
                length := len(queue)
                for i := 0; i < length; i++ {
                        a, b := queue[i][0], queue[i][1]
                        for k := 0; k < 4; k++ {
                                x, y := a+dx[k], b+dy[k]
                                if 0 <= x && x < n && 0 <= y && y < m && matrix[a][b]␣
↪> matrix[x][y] {
                                        arr[x][y]--
                                        if arr[x][y] == 0 { // 个数为0, 加入队列
                                                queue = append(queue, [2]int{x, y})
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return res
}

# 3
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func longestIncreasingPath(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        dp := make([][]int, n)
        temp := make([][3]int, 0)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        dp[i][j] = 1
                        temp = append(temp, [3]int{i, j, matrix[i][j]})
```

(续下页)

(接上页)

```
                }
        }
        sort.Slice(temp, func(i, j int) bool {
                return temp[i][2] < temp[j][2]
        })
        res := 1 // 一个数的时候，没有周围4个数，此时为1
        for i := 0; i < len(temp); i++ {
                a, b := temp[i][0], temp[i][1]
                for k := 0; k < 4; k++ {
                        x, y := a+dx[k], b+dy[k]
                        if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y] >␣
↪matrix[a][b] {
                                dp[x][y] = max(dp[x][y], dp[a][b]+1) // 更新长度
                                res = max(res, dp[x][y])
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

12.8 330. 按要求补齐数组 (1)

- 题目

```
给定一个已排序的正整数数组 nums，和一个正整数n 。从[1,␣
↪n]区间内选取任意个数字补充到nums中，
使得[1, n]区间内的任何数字都可以用nums中某几个数字的和来表示。
请输出满足上述要求的最少需要补充的数字个数。
示例1:输入: nums = [1,3], n = 6 输出: 1
解释: 根据 nums里现有的组合[1], [3], [1,3], 可以得出1, 3, 4。
现在如果我们将2添加到nums 中，组合变为: [1], [2], [3], [1,3], [2,3], [1,2,3]。
其和可以表示数字1, 2, 3, 4, 5, 6, 能够覆盖[1, 6]区间里所有的数。
所以我们最少需要添加一个数字。
示例 2:输入: nums = [1,5,10], n = 20 输出: 2
解释: 我们需要添加[2, 4]。
```

(续下页)

(接上页)

```
示例3:输入: nums = [1,2,2], n = 5 输出: 0
```

- 解题思路

```
func minPatches(nums []int, n int) int {
        res := 0
        target := 1
        for i := 0; target <= n; {
                // 对于正整数x, 如果区间[1,x-1]内的所有数字都已经被覆盖,
                // 加上x后, 则区间[1,2x-1]内的所有数字被覆盖。
                if i < len(nums) && nums[i] <= target {
                        // 贪心思路: 把当前在范围内的加上, target之前的[1,target-
↪1]都已经满足
                        target = target + nums[i]
                        i++
                } else {
                        // 没有就把target加入数组, 范围扩大1倍
                        target = target * 2
                        res++
                }
        }
        return res
}
```

12.9 335. 路径交叉 (2)

- 题目

```
给定一个含有n个正数的数组x。从点(0,0)开始, 先向北移动x[0]米,
然后向西移动x[1]米, 向南移动x[2]米, 向东移动x[3]米, 持续移动。
也就是说, 每次移动后你的方位会发生逆时针变化。
编写一个O(1)空间复杂度的一趟扫描算法, 判断你所经过的路径是否相交。
示例1:
┌──┐
│  │
└──┼─>
   │
输入: [2,1,1,2] 输出: true
示例2:
┌────┐
│    │
│
```

(续下页)

(接上页)

```
|
└──────────────>
输入: [1,2,3,4] 输出: false
示例 3:
┌──┐
|  |
└──┼>
输入: [1,1,1,1] 输出: true
```

- 解题思路

```
func isSelfCrossing(distance []int) bool {
        n := len(distance)
        if n < 4 {
                return false
        }
        for i := 3; i < n; i++ {
                // 长度为4相交
                if i >= 3 && distance[i] >= distance[i-2] &&
                        distance[i-1] <= distance[i-3] {
                        return true
                }
                // 长度为5相交
                if i >= 4 && distance[i]+distance[i-4] >= distance[i-2] &&
                        distance[i-1] == distance[i-3] {
                        return true
                }
                // 长度为6相交
                if i >= 5 && distance[i]+distance[i-4] >= distance[i-2] &&
                        distance[i-1]+distance[i-5] >= distance[i-3] &&
                        distance[i-2] > distance[i-4] &&
                        distance[i-1] < distance[i-3] {
                        return true
                }
        }
        return false
}

# 2
func isSelfCrossing(distance []int) bool {
        arr := append([]int{0, 0, 0, 0}, distance...)
        n := len(arr)
        i := 4
        // 圈变大
```

(续下页)

(接上页)

```
        for i < n && arr[i] > arr[i-2] {
                i++
        }
        if i == n {
                return false
        }
        if arr[i] >= arr[i-2]-arr[i-4] {
                arr[i-1] = arr[i-1] - arr[i-3]
        }
        i++
        // 圈变小
        for i < n && arr[i] < arr[i-2] {
                i++
        }
        if i == n {
                return false
        }
        return true
}
```

12.10 336. 回文对

12.10.1 题目

```
给定一组 互不相同 的单词， 找出所有不同的索引对(i, j)，使得列表中的两个单词，
words[i] + words[j]，可拼接成回文串。
示例 1: 输入: ["abcd","dcba","lls","s","sssll"] 输出: [[0,1],[1,0],[3,2],[2,4]]
解释: 可拼接成的回文串为 ["dcbaabcd","abcddcba","slls","llssssll"]
示例 2: 输入: ["bat","tab","cat"] 输出: [[0,1],[1,0]]
解释: 可拼接成的回文串为 ["battab","tabbat"]
```

12.10.2 解题思路

12.11 354. 俄罗斯套娃信封问题 (3)

- 题目

给定一些标记了宽度和高度的信封，宽度和高度以整数对形式 (w, h) 出现。
当另一个信封的宽度和高度都比这个信封大的时候，这个信封就可以放进另一个信封里，如同俄罗斯套娃一样。
请计算最多能有多少个信封能组成一组“俄罗斯套娃”信封（即可以把一个信封放到另一个信封里面）。
说明：不允许旋转信封。
示例:输入：envelopes = [[5,4],[6,4],[6,7],[2,3]] 输出：3
解释：最多信封的个数为 3，组合为：[2,3] => [5,4] => [6,7]。

- 解题思路

```
func maxEnvelopes(envelopes [][]int) int {
        if len(envelopes) <= 1 {
                return len(envelopes)
        }
        // 宽[0] 高[1]排序
        sort.Slice(envelopes, func(i, j int) bool {
                if envelopes[i][0] == envelopes[j][0] {
                        return envelopes[i][1] < envelopes[j][1]
                }
                return envelopes[i][0] < envelopes[j][0]
        })
        // 第i个信封套几个
        dp := make([]int, len(envelopes))
        for i := 0; i < len(dp); i++ {
                dp[i] = 1
        }
        res := 1
        for i := 1; i < len(envelopes); i++ {
                for j := 0; j < i; j++ {
                        if envelopes[i][0] > envelopes[j][0] &&
                                envelopes[i][1] > envelopes[j][1] {
                                dp[i] = max(dp[i], dp[j]+1)
                        }
                }
                res = max(res, dp[i])
        }
```

(续下页)

(接上页)

```
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxEnvelopes(envelopes [][]int) int {
        if len(envelopes) <= 1 {
                return len(envelopes)
        }
        // 宽[0] 高[1]排序
        sort.Slice(envelopes, func(i, j int) bool {
                if envelopes[i][0] == envelopes[j][0] {
                        return envelopes[i][1] < envelopes[j][1]
                }
                return envelopes[i][0] < envelopes[j][0]
        })
        arr := make([]int, 0)
        for i := 0; i < len(envelopes); i++ {
                left := 0
                right := len(arr) - 1
                for left <= right {
                        mid := left + (right-left)/2
                        if envelopes[i][1] > arr[mid] {
                                left = mid + 1
                        } else {
                                right = mid - 1
                        }
                }
                if left >= len(arr) {
                        arr = append(arr, envelopes[i][1])
                } else {
                        arr[left] = envelopes[i][1]
                }
        }
        return len(arr)
}
```

(续下页)

(接上页)

```
# 3
func maxEnvelopes(envelopes [][]int) int {
        if len(envelopes) <= 1 {
                return len(envelopes)
        }
        // 宽[0] 高[1]排序
        sort.Slice(envelopes, func(i, j int) bool {
                if envelopes[i][0] == envelopes[j][0] {
                        return envelopes[i][1] < envelopes[j][1]
                }
                return envelopes[i][0] < envelopes[j][0]
        })
        // 第i个信封套几个
        dp := make([]int, len(envelopes))
        dp[0] = 1
        res := 1
        for i := 1; i < len(envelopes); i++ {
                temp := 0
                for j := i - 1; j >= 0; j-- {
                        if envelopes[i][0] > envelopes[j][0] &&
                                envelopes[i][1] > envelopes[j][1] && dp[j] > temp {
                                temp = dp[j]
                        }
                }
                dp[i] = temp + 1
                if dp[i] > res {
                        res = dp[i]
                }
        }
        return res
}
```

12.12 391. 完美矩形 (1)

- 题目

```
我们有 N 个与坐标轴对齐的矩形，其中 N > 0，判断它们是否能精确地覆盖一个矩形区域。
每个矩形用左下角的点和右上角的点的坐标来表示。例如，一个单位正方形可以表示为 [1,1,2,
↪2]。
( 左下角的点的坐标为 (1, 1) 以及右上角的点的坐标为 (2, 2) )。
示例 1:rectangles = [
  [1,1,3,3],
```

(续下页)

(接上页)

```
  [3,1,4,2],
  [3,2,4,4],
  [1,3,2,4],
  [2,3,3,4]
]
返回 true。5个矩形一起可以精确地覆盖一个矩形区域。
示例2:rectangles = [
  [1,1,2,3],
  [1,3,2,4],
  [3,1,4,2],
  [3,2,4,4]
]
返回 false。两个矩形之间有间隔，无法覆盖成一个矩形。
示例 3:rectangles = [
  [1,1,3,3],
  [3,1,4,2],
  [1,3,2,4],
  [3,2,4,4]
]
返回 false。图形顶端留有间隔，无法覆盖成一个矩形。
示例 4:rectangles = [
  [1,1,3,3],
  [3,1,4,2],
  [1,3,2,4],
  [2,2,4,4]
]
返回 false。因为中间有相交区域，虽然形成了矩形，但不是精确覆盖。
```

- 解题思路

```
func isRectangleCover(rectangles [][]int) bool {
        minX, maxX := math.MaxInt32, math.MinInt32
        minY, maxY := math.MaxInt32, math.MinInt32
        res := 0
        m := make(map[string]bool)
        for i := 0; i < len(rectangles); i++ {
                a, b, c, d := rectangles[i][0], rectangles[i][1], rectangles[i][2],␣
↪rectangles[i][3]
                minX = min(minX, a)
                minY = min(minY, b)
                maxX = max(maxX, c)
                maxY = max(maxY, d)
                res = res + (c-a)*(d-b)
                arr := []string{
```

(续下页)

(接上页)

```
            fmt.Sprintf("%d-%d", a, b),
            fmt.Sprintf("%d-%d", c, d),
                        fmt.Sprintf("%d-%d", a, d),
            fmt.Sprintf("%d-%d", c, b),
        }
                for _, v := range arr {
                        if _, ok := m[v]; ok {
                                delete(m, v)
                        } else {
                                m[v] = true
                        }
                }
        }
        if (maxX-minX)*(maxY-minY) != res { //␣
↪满足条件1: 所有小矩形相加的面积之和要等于完美矩形的面积
                return false
        }
        if len(m) != 4 || // 满足条件2: 除最终大矩形4个顶点外其它点都出现偶数次
                m[fmt.Sprintf("%d-%d", minX, minY)] == false ||
        m[fmt.Sprintf("%d-%d", minX, maxY)] == false ||
                m[fmt.Sprintf("%d-%d", maxX, minY)] == false ||
        m[fmt.Sprintf("%d-%d", maxX, maxY)] == false {
                return false
        }
        return true
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

CHAPTER 13

0401-0500-Easy

13.1 401. 二进制手表 (3)

- 题目

```
二进制手表顶部有 4 个 LED 代表小时 (0-11)，底部的 6 个 LED 代表分钟 (0-59)。
每个 LED 代表一个 0 或 1，最低位在右侧。
例如，上面的二进制手表读取 “3:25”。
给定一个非负整数 n 代表当前 LED 亮着的数量，返回所有可能的时间。

案例:输入: n = 1
返回: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16", "0:32"]

注意事项:
输出的顺序没有要求。
小时不会以零开头，比如 “01:00” 是不允许的，应为 “1:00”。
分钟必须由两位数组成，可能会以零开头，比如 “10:2” 是无效的，应为 “10:02”。
```

- 解题思路

```
func binCount(num int) int {
        count := make([]int, 0)
        for num != 0 {
                temp := num % 2
                count = append(count, temp)
```

(续下页)

(接上页)

```
                num = num / 2
        }
        countNum := 0
        for i := 0; i < len(count); i++ {
                if count[i] == 1 {
                        countNum++
                }
        }
        return countNum
}

func readBinaryWatch(num int) []string {
        res := make([]string, 0)
        for i := 0; i < 12; i++ {
                for j := 0; j < 60; j++ {
                        if binCount(i)+binCount(j) == num {
                                res = append(res, fmt.Sprintf("%d:%02d", i, j))
                        }
                }
        }
        return res
}

#
func readBinaryWatch(num int) []string {
        res := make([]string, 0)
        for i := 0; i < 12; i++ {
                for j := 0; j < 60; j++ {
                        hour := fmt.Sprintf("%b", i)
                        minute := fmt.Sprintf("%b", j)
                        if strings.Count(hour, "1")+strings.Count(minute, "1") == num
↪{
                                res = append(res, fmt.Sprintf("%d:%02d", i, j))
                        }
                }
        }
        return res
}

#
func readBinaryWatch(num int) []string {
        res := make([]string, 0)
        ledS := make([]bool, 10)
```

(续下页)

(接上页)

```
        var dfs func(int, int)
        dfs = func(idx, num int) {
                if num == 0 {
                        // 满足条件
                        m, h := get(ledS[:6]), get(ledS[6:])
                        if h < 12 && m < 60 {
                                res = append(res, fmt.Sprintf("%d:%02d", h, m))
                        }
                        return
                }
                for i := idx; i < 11-num; i++ {
                        ledS[i] = true
                        dfs(i+1, num-1)
                        ledS[i] = false
                }
        }
        dfs(0, num)
        return res
}

func get(ledS []bool) int {
        bs := []int{1, 2, 4, 8, 16, 32}
        var sum int
        size := len(ledS)
        for i := 0; i < size; i++ {
                if ledS[i] {
                        sum += bs[i]
                }
        }
        return sum
}
```

13.2 404. 左叶子之和 (2)

- 题目

```
计算给定二叉树的所有左叶子之和。
示例:
    3
   / \
  9  20
```

(续下页)

(接上页)

```
    /  \
   15   7
在这个二叉树中，有两个左叶子，分别是 9 和 15，所以返回 24
```

- 解题思路

```
func sumOfLeftLeaves(root *TreeNode) int {
        if root == nil {
                return 0
        }
        if root.Left == nil {
                return sumOfLeftLeaves(root.Right)
        }
        if root.Left.Left == nil && root.Left.Right == nil {
                return root.Left.Val + sumOfLeftLeaves(root.Right)
        }
        return sumOfLeftLeaves(root.Left) + sumOfLeftLeaves(root.Right)
}


#
func sumOfLeftLeaves(root *TreeNode) int {
        sum := 0
        if root == nil{
                return 0
        }
        queue := make([]*TreeNode,0)
        queue = append(queue, root)
        for len(queue) > 0{
                node := queue[0]
                queue = queue[1:]
                if node.Left != nil && node.Left.Left == nil && node.Left.Right == nil
↪{
                        sum = sum + node.Left.Val
                }
                if node.Left != nil{
                        queue = append(queue, node.Left)
                }
                if node.Right != nil{
                        queue = append(queue, node.Right)
                }
        }
        return sum
}
```

13.3 405. 数字转换为十六进制数 (2)

- 题目

```
给定一个整数，编写一个算法将这个数转换为十六进制数。对于负整数，我们通常使用 补码运算␣
↪方法。
注意:
    十六进制中所有字母(a-f)都必须是小写。
    十六进制字符串中不能包含多余的前导零。
    如果要转化的数为0，那么以单个字符'0
↪'来表示；对于其他情况，十六进制字符串中的第一个字符将不会是0字符。
    给定的数确保在32位有符号整数范围内。
    不能使用任何由库提供的将数字直接转换或格式化为十六进制的方法。
示例 1:  输入:26 输出:"1a"
示例 2:  输入:-1 输出:"ffffffff"
```

- 解题思路

```
var h = []string{
        "0", "1", "2", "3", "4", "5", "6", "7",
        "8", "9", "a", "b", "c", "d", "e", "f",
}

func toHex(num int) string {
        hex := ""
        if num == 0 {
                return "0"
        }

        for i := 0; i < 8 && num != 0; i++ {
                hex = h[num&15] + hex
                num = num >> 4
        }
        return hex
}

#
var h = []string{
        "0", "1", "2", "3", "4", "5", "6", "7",
        "8", "9", "a", "b", "c", "d", "e", "f",
}

func toHex(num int) string {
        res := ""
```

(续下页)

(接上页)

```
        if num == 0{
                return "0"
        }
        if num < 0 {
                num = num + 4294967296
        }

        for num != 0{
                temp := num % 16
                res = h[temp] + res
                num = num / 16
        }
        return res
}
```

13.4 409. 最长回文串 (2)

- 题目

```
给定一个包含大写字母和小写字母的字符串，找到通过这些字母构造成的最长的回文串。
在构造过程中，请注意区分大小写。比如 "Aa" 不能当做一个回文字符串。
注意:假设字符串的长度不会超过 1010。
示例 1:输入:"abccccdd"输出:7
解释:我们可以构造的最长的回文串是"dccaccd", 它的长度是 7。
```

- 解题思路

```
func longestPalindrome(s string) int {
        ret := 0
        a := [123]int{}
        for i := range s {
                a[s[i]]++
        }
        hasOdd := 0
        for i := range a {
                if a[i] == 0 {
                        continue
                }
                if a[i] % 2 == 0 {
                        ret += a[i]
                } else {
                        ret += a[i] - 1
```

(续下页)

(接上页)

```
                        hasOdd = 1
                }
        }
        return ret + hasOdd
}

#
func longestPalindrome(s string) int {
        ret := 0
        a := make(map[byte]int)
        for i := range s {
                a[s[i]]++
        }
        hasOdd := 0
        for i := range a {
                if a[i] == 0 {
                        continue
                }
                if a[i]%2 == 0 {
                        ret += a[i]
                } else {
                        ret += a[i] - 1
                        hasOdd = 1
                }
        }
        return ret + hasOdd
}
```

13.5 412.Fizz Buzz(1)

- 题目

```
写一个程序，输出从 1 到 n 数字的字符串表示。
1. 如果 n 是3的倍数，输出“Fizz”；
2. 如果 n 是5的倍数，输出“Buzz”；
3.如果 n 同时是3和5的倍数，输出 “FizzBuzz”。

示例：n = 15,
返回:
[
    "1",
    "2",
```

(续下页)

(接上页)

```
    "Fizz",
    "4",
    "Buzz",
    "Fizz",
    "7",
    "8",
    "Fizz",
    "Buzz",
    "11",
    "Fizz",
    "13",
    "14",
    "FizzBuzz"
]
```

- 解题思路

```
func fizzBuzz(n int) []string {
        ret := make([]string, n)
        for i := range ret {
                x := i + 1
                switch {
                case x%15 == 0:
                        ret[i] = "FizzBuzz"
                case x%5 == 0:
                        ret[i] = "Buzz"
                case x%3 == 0:
                        ret[i] = "Fizz"
                default:
                        ret[i] = strconv.Itoa(x)
                }
        }
        return ret
}
```

13.6 414. 第三大的数 (2)

- 题目

```
给定一个非空数组，返回此数组中第三大的数。如果不存在，则返回数组中最大的数。要求算法时间复杂度必须是O
示例 1:输入: [3, 2, 1]输出: 1
解释: 第三大的数是 1.
```

(续下页)

(接上页)

```
示例 2:输入: [1, 2]输出: 2
解释: 第三大的数不存在, 所以返回最大的数 2 .
示例 3:输入: [2, 2, 3, 1]输出: 1
解释:␣
↪注意, 要求返回第三大的数, 是指第三大且唯一出现的数。存在两个值为2的数, 它们都排第二。
```

- 解题思路

```
func thirdMax(nums []int) int {
        max1, max2, max3 := math.MinInt64, math.MinInt64, math.MinInt64
        for _, n := range nums {
                if n == max1 || n == max2 {
                        continue
                }
                switch {
                case max1 < n:
                        max1, max2, max3 = n, max1, max2
                case max2 < n:
                        max2, max3 = n, max2
                case max3 < n:
                        max3 = n
                }
        }

        if max3 == math.MinInt64 {
                return max1
        }
        return max3
}

#
func thirdMax(nums []int) int {

        sort.Ints(nums)
        if len(nums) < 3 {
                return nums[len(nums)-1]
        }

        k := 2
        maxValue := nums[len(nums)-1]
        for i := len(nums) - 2; i >= 0; i-- {
                if nums[i] != nums[i+1] {
                        k--
                }
```

(续下页)

(接上页)

```
                if k == 0 {
                        return nums[i]
                }
        }
        return maxValue
}
```

13.7 415. 字符串相加 (2)

- 题目

```
给定两个字符串形式的非负整数 num1 和num2 ，计算它们的和。

注意：
num1 和num2 的长度都小于 5100.
num1 和num2 都只包含数字 0-9.
num1 和num2 都不包含任何前导零。
你不能使用任何🅵建 BigInteger 库， 也不能直接将输入的字符串转换为整数形式。
```

- 解题思路

```
func addStrings(num1 string, num2 string) string {
        if len(num1) > len(num2) {
                num1, num2 = num2, num1
        }
        n1, n2 := len(num1), len(num2)
        a1, a2 := []byte(num1), []byte(num2)
        carry := byte(0)
        buf := make([]byte, n2+1)
        buf[0] = '1'

        for i := 1; i <= n2; i++ {
                if i <= n1 {
                        buf[n2+1-i] = a1[n1-i] - '0'
                }
                buf[n2+1-i] = buf[n2+1-i] + a2[n2-i] + carry

                if buf[n2+1-i] > '9' {
                        buf[n2+1-i] = buf[n2+1-i] - 10
                        carry = byte(1)
                } else {
                        carry = byte(0)
```

(续下页)

(接上页)

```
                }
        }
        if carry == 1 {
                return string(buf)
        }
        return string(buf[1:])
}

#
func addStrings(num1 string, num2 string) string {
        if len(num1) > len(num2) {
                num1, num2 = num2, num1
        }
        n1, n2 := len(num1), len(num2)
        a1, a2 := []byte(num1), []byte(num2)
        a1 = reverse(a1)
        a2 = reverse(a2)

        carry := 0
        buf := make([]byte, 0)
        for i := 0; i < n2; i++ {
                temp := 0
                if i < n1 {
                        temp = int(a1[i] - '0')
                }
                temp = int(a2[i]-'0') + temp + carry
                if temp > 9 {
                        buf = append(buf, byte(temp-10+'0'))
                        carry = 1
                } else {
                        buf = append(buf, byte(temp+'0'))
                        carry = 0
                }
        }
        if carry == 1 {
                buf = append(buf, byte('1'))
        }
        return string(reverse(buf))
}

func reverse(arr []byte) []byte {
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
```

(续下页)

(接上页)

```
        }
        return arr
}
```

13.8 434. 字符串中的单词数 (2)

- 题目

```
统计字符串中的单词个数，这里的单词指的是连续的不是空格的字符。
请注意，你可以假定字符串里不包括任何不可打印的字符。
示例:输入: "Hello, my name is John"输出: 5
解释: 这里的单词是指连续的不是空格的字符，所以 "Hello," 算作 1 个单词。
```

- 解题思路

```
func countSegments(s string) int {
        if len(s) == 0 {
                return 0
        }
        return len(strings.Fields(s))
}

#
func countSegments(s string) int {
        count := 0
        for i := 0; i < len(s); i++{
                if (i == 0 || s[i-1] == ' ') && s[i] != ' '{
                        count++
                }
        }
        return count
}
```

13.9 437. 路径总和 III(4)

- 题目

```
给定一个二叉树，它的每个结点都存放着一个整数值。
找出路径和等于给定数值的路径总数。
路径不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。
```

(续下页)

(接上页)

```
二叉树不超过1000个节点，且节点数值范围是 [-1000000,1000000] 的整数。
示例：
root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

      10
     /  \
    5   -3
   / \    \
  3   2   11
 / \   \
3  -2   1

返回 3。和等于 8 的路径有：
1.  5 -> 3
2.  5 -> 2 -> 1
3.  -3 -> 11
```

- 解题思路

```
func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
        }
        res := 0
        var helper func(*TreeNode, int)
        helper = func(node *TreeNode, sum int) {
                if node == nil {
                        return
                }
                sum = sum - node.Val
                // 路径不需要从根节点开始，也不需要在叶子节点结束
                if sum == 0 {
                        res++
                }
                helper(node.Left, sum)
                helper(node.Right, sum)
        }
        helper(root, sum)
        return res + pathSum(root.Left, sum) + pathSum(root.Right, sum)
}

#
func helper(node *TreeNode, sum int) int {
```

(续下页)

(接上页)

```
        if node == nil {
                return 0
        }
        sum = sum - node.Val
        res := 0
        if sum == 0 {
                res = 1
        }
        return res + helper(node.Left, sum) + helper(node.Right, sum)
}

func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
        }
        return helper(root, sum) + pathSum(root.Left, sum) + pathSum(root.Right, sum)
}

# 迭代+递归
func helper(node *TreeNode, sum int, curSum int) int {
        res := 0
        curSum = curSum + node.Val
        if curSum == sum {
                res++
        }
        if node.Left != nil {
                res += helper(node.Left, sum, curSum)
        }
        if node.Right != nil {
                res += helper(node.Right, sum, curSum)
        }
        return res
}

func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        res := 0
        for len(queue) > 0 {
                node := queue[0]
```

(续下页)

(接上页)

```
                queue = queue[1:]
                tempSum := 0
                res += helper(node, sum, tempSum)
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return res
}

#
func helper(node *TreeNode, sum int, path []int, level int) int {
        if node == nil {
                return 0
        }
        res := 0
        if sum == node.Val {
                res = 1
        }
        temp := node.Val
        for i := level - 1; i >= 0; i-- {
                temp = temp + path[i]
                if temp == sum {
                        res++
                }
        }
        path[level] = node.Val
        return res + helper(node.Left, sum, path, level+1) +
                helper(node.Right, sum, path, level+1)
}

func pathSum(root *TreeNode, sum int) int {
        return helper(root, sum, make([]int, 1001), 0)
}
```

13.10 441. 排列硬币 (3)

- 题目

```
你总共有 n 枚硬币，你需要将它们摆成一个阶梯形状，第 k 行就必须正好有 k 枚硬币。
给定一个数字 n，找出可形成完整阶梯行的总行数。
n 是一个非负整数，并且在32位有符号整型的范围内。
示例 1:n = 5
硬币可排列成以下几行:
¤
¤ ¤
¤ ¤
因为第三行不完整，所以返回2.
示例 2:n = 8
硬币可排列成以下几行:
¤
¤ ¤
¤ ¤ ¤
¤ ¤
因为第四行不完整，所以返回3.
```

- 解题思路

```
func arrangeCoins(n int) int {
        return int(math.Sqrt(float64(2*n)+0.25) - 0.5)
}

#
func arrangeCoins(n int) int {
        i := 1
        for i <= n{
                n = n - i
                i++
        }
        return i-1
}

#
func arrangeCoins(n int) int {
        if n == 0{
                return 0
        }
        left, right := 1, n
        for left < right{
```

(续下页)

(接上页)

```
                mid := left + (right-left)/2
                if mid * (mid+1)/2 < n{
                        left = mid + 1
                }else {
                        right = mid
                }
        }
        if left*(left+1)/2 == n{
                return left
        }
        return left-1
}
```

13.11 443. 压缩字符串 (1)

- 题目

```
给定一组字符，使用原地算法将其压缩。
压缩后的长度必须始终小于或等于原数组长度。
数组的每个元素应该是长度为1 的字符（不是 int 整数类型）。
在完成原地修改输入数组后，返回数组的新长度。
进阶：你能否仅使用O(1) 空间解决问题？

示例 1：输入：["a","a","b","b","c","c","c"]
输出：返回6，输入数组的前6个字符应该是：["a","2","b","2","c","3"]
说明："aa"被"a2"替代。"bb"被"b2"替代。"ccc"被"c3"替代。

示例 2：输入：["a"]
输出：返回1，输入数组的前1个字符应该是：["a"]
说明：没有任何字符串被替代。

示例 3：输入：["a","b","b","b","b","b","b","b","b","b","b","b","b"]
输出：返回4，输入数组的前4个字符应该是：["a","b","1","2"]。
说明：由于字符"a"不重复，所以不会被压缩。"bbbbbbbbbbbb"被“b12”替代。
注意每个数字在数组中都有它自己的位置。
注意：
    所有字符都有一个ASCII值在[35, 126]区间内。
    1 <= len(chars) <= 1000。
```

- 解题思路

```
func compress(chars []byte) int {
        j := 0
        count := 1
        for i := 0; i < len(chars); i++ {
                char := chars[i]
                if i+1 < len(chars) && char == chars[i+1] {
                        count++
                } else {
                        chars[j] = char
                        j++
                        if count > 1 {
                                for _, num := range strconv.Itoa(count) {
                                        chars[j] = byte(num)
                                        j++
                                }
                        }
                        count = 1
                }
        }
        return j
}
```

13.12 447. 回旋镖的数量 (1)

- 题目

```
给定平面上 n 对不同的点，“回旋镖” 是由点表示的元组 (i, j, k) ，
其中 i 和 j 之间的距离和 i 和 k 之间的距离相等（需要考虑元组的顺序）。

找到所有回旋镖的数量。你可以假设 n 最大为 500，所有点的坐标在闭区间 [-10000, 10000]␣
↪中。
示例:
输入:[[0,0],[1,0],[2,0]]
输出:2
解释:两个回旋镖为 [[1,0],[0,0],[2,0]] 和 [[1,0],[2,0],[0,0]]
```

- 解题思路

```
func numberOfBoomerangs(points [][]int) int {
        res := 0
        size := len(points)
        if size < 3 {
                return 0
```

(续下页)

(接上页)

```
        }
        for i := 0; i < size; i++ {
                dMap := make(map[int]int, size)
                for j := 0; j < size; j++ {
                        if i == j {
                                continue
                        }
                        d := dSquare(points[i], points[j])
                        if _, ok := dMap[d]; ok {
                                dMap[d]++
                        } else {
                                dMap[d] = 1
                        }
                }
                // 相同距离的v个点，总共有 v*(v-1)种排列
                for _, v := range dMap {
                        res = res + v*(v-1)
                }
        }
        return res
}

func dSquare(p1, p2 []int) int {
        x := p2[0] - p1[0]
        y := p2[1] - p1[1]
        return x*x + y*y
}
```

13.13 448. 找到所有数组中消失的数字 (3)

- 题目

```
给定一个范围在 1 ≤ a[i] ≤ n ( n = 数组大小 ) 的␣
↪整型数组，数组中的元素一些出现了两次，另一些只出现一次。
找到所有在 [1, n] 范围之间没有出现在数组中的数字。
您能在不使用额外空间且时间复杂度为O(n)的情况下完成这个任务吗?␣
↪你可以假定返回的数组不算在额外空间内。

示例:输入:[4,3,2,7,8,2,3,1]输出:[5,6]
```

- 解题思路

```
func findDisappearedNumbers(nums []int) []int {
        for i := 0; i < len(nums); i++ {
                for nums[i] != nums[nums[i]-1] {
                        nums[i], nums[nums[i]-1] = nums[nums[i]-1], nums[i]
                }
        }
        res := make([]int, 0)
        for i, n := range nums {
                if n != i+1 {
                        res = append(res, i+1)
                }
        }
        return res
}

#
func findDisappearedNumbers(nums []int) []int {
        for i := 0; i < len(nums); i++ {
                value := nums[i]
                if value < 0{
                        value = -value
                }
                if nums[value-1] > 0{
                        nums[value-1] = -nums[value-1]
                }
        }
        res := make([]int, 0)
        for key, value := range nums {
                if value > 0{
                        res = append(res, key+1)
                }
        }
        return res
}

#
func findDisappearedNumbers(nums []int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = 1
        }
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                if _, ok := m[i+1]; !ok {
```

(续下页)

(接上页)

```
                        res = append(res, i+1)
                }
        }
        return res
}
```

13.14 453. 最小移动次数使数组元素相等 (2)

- 题目

```
给定一个长度为 n 的非空整数数组，找到让数组所有元素相等的最小移动次数。每次移动可以使
↪n - 1 个元素增加 1。
示例:输入:[1,2,3]输出:3
解释:只需要3次移动（注意每次移动会增加两个元素的值）：
[1,2,3]  =>  [2,3,3]  =>  [3,4,3]  =>  [4,4,4]
```

- 解题思路

```
func minMoves(nums []int) int {
        sum := 0
        min := nums[0]
        for _, n := range nums {
                sum += n
                if min > n {
                        min = n
                }
        }
        return sum - min*len(nums)
}

#
func minMoves(nums []int) int {
        sum := 0
        sort.Ints(nums)
        for i := 1; i < len(nums); i++{
                sum = sum + nums[i] - nums[0]
        }
        return sum
}
```

13.15 455. 分发饼干 (1)

- 题目

```
假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。
对每个孩子 i ，都有一个胃口值 gi ，这是能让孩子们满足胃口的饼干的最小尺寸；
并且每块饼干 j ，都有一个尺寸 sj 。
如果 sj >= gi ，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。
你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。

注意：你可以假设胃口值为正。一个小朋友最多只能拥有一块饼干。

示例 1:输入: [1,2,3], [1,1]  输出: 1
解释: 你有三个孩子和两块小饼干，3个孩子的胃口值分别是: 1,2,3。
虽然你有两块小饼干，由于他们的尺寸都是1，你只能让胃口值是1的孩子满足。所以你应该输出1。

示例 2:输入: [1,2], [1,2,3] 输出: 2
解释: 你有两个孩子和三块小饼干，2个孩子的胃口值分别是1,2。
你拥有的饼干数量和尺寸都足以让所有孩子满足。所以你应该输出2.
```

- 解题思路

```go
func findContentChildren(g []int, s []int) int {
        sort.Ints(g)
        sort.Ints(s)
        var i, j int
        for i < len(g) && j < len(s) {
                if g[i] <= s[j] {
                        i++
                }
                j++
        }
        return i
}
```

13.16 459. 重复的子字符串 (2)

- 题目

```
给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。
给定的字符串只含有小写英文字母，并且长度不超过10000。
示例 1:输入: "abab"输出: True
解释: 可由子字符串 "ab" 重复两次构成。
```

(续下页)

(接上页)

```
示例 2:输入: "aba"输出: False
示例 3:输入: "abcabcabcabc"输出: True
解释: 可由子字符串 "abc" 重复四次构成。 (或者子字符串 "abcabc" 重复两次构成。)
```

- 解题思路

```
func repeatedSubstringPattern(s string) bool {
        if len(s) == 0 {
                return false
        }

        size := len(s)
        ss := (s + s)[1 : size*2-1]
        return strings.Contains(ss, s)
}

#
func repeatedSubstringPattern(s string) bool {
        if len(s) == 0 {
                return false
        }
        size := len(s)
        for i := 1; i < size; i++ {
                if size%i == 0 {
                        count := size / i
                        if strings.Repeat(s[0:i], count) == s {
                                return true
                        }
                }
        }
        return false
}
```

13.17 461. 汉明距离 (3)

- 题目

```
两个整数之间的汉明距离指的是这两个数字对应二进制位不同的位置的数目。
给出两个整数 x 和 y, 计算它们之间的汉明距离。
注意:
0 ≤ x, y < 231.
示例:
```

(续下页)

(接上页)

```
输入: x = 1, y = 4输出: 2
解释:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑
上面的箭头指出了对应二进制位不同的位置。
```

- 解题思路

```
func hammingDistance(x int, y int) int {
        x = x ^ y
        res := 0
        for x > 0 {
                if x&1 == 1{
                        res++
                }
                x = x >> 1
        }
        return res
}

#
func hammingDistance(x int, y int) int {
        x = x ^ y
        res := 0
        for x > 0 {
                res++
                x = x & (x-1)
        }
        return res
}

#
func hammingDistance(x int, y int) int {
        x = x ^ y
        return bits.OnesCount(uint(x))
}
```

13.18 463. 岛屿的周长 (3)

- 题目

```
给定一个包含 0 和 1 的二维网格地图，其中 1 表示陆地 0 表示水域。
网格中的格子水平和垂直方向相连（对角线方向不相连）。
整个网格被水完全包围，但其中恰好有一个岛屿（或者说，一个或多个表示陆地的格子相连组成的岛屿）。
岛屿中没有“湖”（“湖” 指水域在岛屿内部且不和岛屿周围的水相连）。格子是边长为 1␣
↪的正方形。
网格为长方形，且宽度和高度均不超过 100 。计算这个岛屿的周长。

示例 :
输入:
[[0,1,0,0],
 [1,1,1,0],
 [0,1,0,0],
 [1,1,0,0]]
输出: 16
```

- 解题思路

```
func islandPerimeter(grid [][]int) int {
        var dx = []int{-1, 1, 0, 0}
        var dy = []int{0, 0, -1, 1}
        m, n := len(grid), len(grid[0])
        res := 0
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if grid[i][j] == 0 {
                                continue
                        }
                        res += 4
                        for k := 0; k < 4; k++ {
                                x := i + dx[k]
                                y := j + dy[k]
                                if (0 <= x && x < m && 0 <= y && y < n) && grid[x][y]␣
↪== 1 {
                                        res--
                                }
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func islandPerimeter(grid [][]int) int {
        m, n := len(grid), len(grid[0])
        res := 0
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if grid[i][j] == 0 {
                                continue
                        }
                        res += 4
                        if i > 0 && grid[i-1][j] == 1 {
                                res -= 2
                        }
                        if j > 0 && grid[i][j-1] == 1 {
                                res -= 2
                        }
                }
        }
        return res
}

#
func islandPerimeter(grid [][]int) int {
        m, n := len(grid), len(grid[0])
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if grid[i][j] == 1 {
                                return dfs(grid, i, j)
                        }
                }
        }
        return 0
}

func dfs(grid [][]int, i, j int) int {
        // 边界+1
        if !(0 <= i && i < len(grid) && 0 <= j && j < len(grid[0])) {
                return 1
        }
        // 水域+1
        if grid[i][j] == 0 {
                return 1
```

(续下页)

(接上页)

```
        }
        if grid[i][j] != 1 {
                return 0
        }
        grid[i][j] = 2
        return dfs(grid, i-1, j) +
                dfs(grid, i+1, j) +
                dfs(grid, i, j-1) +
                dfs(grid, i, j+1)
}
```

13.19 475. 供暖器 (2)

- 题目

```
冬季已经来临。 你的任务是设计一个有固定加热半径的供暖器向所有房屋供暖。
现在，给出位于一条水平线上的房屋和供暖器的位置，找到可以覆盖所有房屋的最小加热半径。
所以，你的输入将会是房屋和供暖器的位置。你将输出供暖器的最小加热半径。
说明：
    给出的房屋和供暖器的数目是非负数且不会超过 25000。
    给出的房屋和供暖器的位置均是非负数且不会超过10^9。
    只要房屋位于供暖器的半径内(包括在边缘上)，它就可以得到供暖。
    所有供暖器都遵循你的半径标准，加热的半径也一样。

示例 1:输入: [1,2,3],[2] 输出: 1
解释: 仅在位置2上有一个供暖器。如果我们将加热半径设为1，那么所有房屋就都能得到供暖。

示例 2:输入: [1,2,3,4],[1,4] 输出: 1
解释: 在位置1, 4上有两个供暖器。我们需要将加热半径设为1，这样所有房屋就都能得到供暖。
```

- 解题思路

```
func findRadius(houses []int, heaters []int) int {
        if len(heaters) == 0 {
                return 0
        }
        sort.Ints(houses)
        sort.Ints(heaters)
        res := 0
        j := 0
        for i := 0; i < len(houses); i++ {
                // 找到最近的一个供暖器，>=确保出现重复的供暖器会往后走
```

(续下页)

(接上页)

```
                for j < len(heaters)-1 &&
                        Abs(houses[i], heaters[j]) >= Abs(houses[i], heaters[j+1]) {
                        j++
                }
                res = Max(Abs(houses[i], heaters[j]), res)
        }
        return res
}

func Abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

func Max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func findRadius(houses []int, heaters []int) int {
        if len(heaters) == 0 {
                return 0
        }
        sort.Ints(houses)
        sort.Ints(heaters)
        res := 0
        length := len(heaters)
        for i := 0; i < len(houses); i++ {
                left := 0
                right := length - 1
                for left < right {
                        mid := left + (right-left)/2
                        if heaters[mid] < houses[i] {
                                left = mid + 1
                        } else {
                                right = mid
                        }
                }
```

(续下页)

(接上页)

```
                dis := 0
                if heaters[left] < houses[i] {
                        dis = houses[i] - heaters[left]
                } else if heaters[left] > houses[i] {
                        if left == 0 {
                                dis = heaters[0] - houses[i]
                        } else {
                                dis = Min(heaters[left]-houses[i], houses[i]-
↪heaters[left-1])
                        }
                }
                res = Max(res, dis)
        }
        return res
}

func Min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func Max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

13.20 476. 数字的补数 (3)

- 题目

```
给定一个正整数，输出它的补数。补数是对该数的二进制表示取反。

示例 1:输入: 5 输出: 2
解释: 5 的二进制表示为 101（没有前导零位），其补数为 010。所以你需要输出 2 。
示例 2:输入: 1 输出: 0
解释: 1 的二进制表示为 1（没有前导零位），其补数为 0。所以你需要输出 0 。
注意:
    给定的整数保证在 32 位带符号整数的范围内。
```

(续下页)

(接上页)

```
    你可以假定二进制数不包含前导零位。
    本题与 1009 https://leetcode.cn/problems/complement-of-base-10-integer/ 相同
```

- 解题思路

```
func findComplement(num int) int {
        temp := 1
        for num >= temp {
                temp = temp << 1
        }
        return temp - 1 - num
}

#
func findComplement(num int) int {
        temp := num
        res := 0
        for temp > 0 {
                temp = temp >> 1
                res = res << 1
                res++
        }
        return res ^ num
}

#
func findComplement(num int) int {
        res := 0
        if num == 0 {
                return 1
        }
        if num == 1 {
                return 0
        }

        exp := 1
        for num > 0 {
                temp := num % 2
                if temp == 0 {
                        res = res + exp
                        exp = exp * 2
                } else {
                        exp = exp * 2
                }
```

(续下页)

(接上页)

```
                num = num / 2
        }
        return res
}
```

13.21 482. 密钥格式化 (2)

- 题目

```
有一个密钥字符串 S ，只包含字母，数字以及 '-'（破折号）。
其中， N 个 '-' 将字符串分成了 N+1 组。

给你一个数字 K，请你重新格式化字符串，除了第一个分组以外，每个分组要包含 K 个字符；
而第一个分组中，至少要包含 1 个字符。
两个分组之间需要用 '-'（破折号）隔开，并且将所有的小写字母转换为大写字母。
给定非空字符串 S 和数字 K，按照上面描述的规则进行格式化。

示例 1：输入：S = "5F3Z-2e-9-w", K = 4 输出："5F3Z-2E9W"
解释：字符串 S 被分成了两个部分，每部分 4 个字符；注意，两个额外的破折号需要删掉。

示例 2：输入：S = "2-5g-3-J", K = 2 输出："2-5G-3J"
解释：字符串 S 被分成了 3 个部分，按照前面的规则描述，
第一部分的字符可以少于给定的数量，其余部分皆为 2 个字符。
提示：
    S 的长度可能很长，请按需分配大小。K 为正整数。
    S 只包含字母数字（a-z，A-Z，0-9）以及破折号'-'
    S 非空
```

- 解题思路

```
func licenseKeyFormatting(S string, K int) string {
        arr := strings.Join(strings.Split(strings.ToUpper(S), "-"), "")
        count := len(arr) / K
        first := len(arr) % K
        if first > 0 {
                count++
        }
        str := arr[:first]
        if first != 0 {
                count = count - 1
        }
        for i := 0; i < count; i++ {
```

(续下页)

(接上页)

```
                str = str + "-" + arr[first+i*K:first+(i+1)*K]
        }
        return strings.Trim(str, "-")
}

#
func licenseKeyFormatting(S string, K int) string {
        res := make([]rune, 0)
        temp := []rune(S)
        count := 0
        for i := len(temp) - 1; i >= 0; i-- {
                value := temp[i]
                if value >= 'a' {
                        value = value - 'a' + 'A'
                }
                if value == '-' {
                        continue
                }
                count++
                res = append([]rune{value}, res...)
                if count == K {
                        res = append([]rune{'-'}, res...)
                        count = 0
                }
        }
        if len(res) == 0 {
                return ""
        }
        if res[0] == '-' {
                res = res[1:]
        }
        return string(res)
}
```

13.22 485. 最大连续 1 的个数 (2)

- 题目

```
给定一个二进制数组， 计算其中最大连续1的个数。
示例 1:输入: [1,1,0,1,1,1]输出: 3
解释: 开头的两位和最后的三位都是连续1，所以最大连续1的个数是 3.
注意:
```

(续下页)

(接上页)

```
    输入的数组只包含 0 和1。
    输入数组的长度是正整数，且不超过 10,000。
```

- 解题思路

```
func findMaxConsecutiveOnes(nums []int) int {
        max := 0
        for i, j := 0, -1; i < len(nums); i++ {
                if nums[i] == 0 {
                        j = i
                } else {
                        if max < i-j {
                                max = i - j
                        }
                }
        }
        return max
}

#
func findMaxConsecutiveOnes(nums []int) int {
        max := 0
        count := 0
        for _, v := range nums {
                if v == 1 {
                        count++
                } else {
                        if count > max {
                                max = count
                        }
                        count = 0
                }
        }
        if count > max {
                max = count
        }
        return max
}
```

13.23 492. 构造矩形 (1)

- 题目

```
作为一位web开发者， 懂得怎样去规划一个页面的尺寸是很重要的。
现给定一个具体的矩形页面面积，你的任务是设计一个长度为 L 和宽度为 W且满足以下要求的矩形的页面。要求：

1. 你设计的矩形页面必须等于给定的目标面积。
2. 宽度 W 不应大于长度 L，换言之，要求 L >= W 。
3. 长度 L 和宽度 W 之间的差距应当尽可能小。

你需要按顺序输出你设计的页面的长度 L 和宽度 W。
示例：
输入：4 输出：[2, 2]
解释：目标面积是 4， 所有可能的构造方案有 [1,4], [2,2], [4,1]。
但是根据要求2，[1,4] 不符合要求；根据要求3，[2,2] 比 [4,1] 更能符合要求.
所以输出长度 L 为 2， 宽度 W 为 2。
说明：
    给定的面积不大于 10,000,000 且为正整数。
    你设计的页面的长度和宽度必须都是正整数。
```

- 解题思路

```
func constructRectangle(area int) []int {
        for i := int(math.Sqrt(float64(area))); i > 1; i-- {
                if area%i == 0 {
                        return []int{area / i, i}
                }
        }
        return []int{area, 1}
}
```

13.24 496. 下一个更大元素 I(3)

- 题目

```
给定两个没有重复元素的数组nums1 和 nums2 ，其中nums1 是 nums2 的子集。
找到 nums1 中每个元素在 nums2 中的下一个比其大的值。
nums1 中数字 x 的下一个更大元素是指 x 在 nums2 中对应位置的右边的第一个比 x 大的元素。
如果不存在，对应位置输出 -1 。
示例 1:
输入: nums1 = [4,1,2], nums2 = [1,3,4,2].
```

(续下页)

(接上页)

```
输出: [-1,3,-1]
解释:
    对于num1中的数字4，你无法在第二个数组中找到下一个更大的数字，因此输出 -1。
    对于num1中的数字1，第二个数组中数字1右边的下一个较大数字是 3。
    对于num1中的数字2，第二个数组中没有下一个更大的数字，因此输出 -1。

示例 2:
输入: nums1 = [2,4], nums2 = [1,2,3,4].
输出: [3,-1]
解释:
    对于 num1 中的数字 2 ，第二个数组中的下一个较大数字是 3 。
    对于 num1 中的数字 4 ，第二个数组中没有下一个更大的数字，因此输出 -1 。
提示:
    nums1和nums2中所有元素是唯一的。
    nums1和nums2 的数组大小都不超过1000。
```

- 解题思路

```
func nextGreaterElement(nums1 []int, nums2 []int) []int {
        m := make(map[int]int)
        for i, n := range nums2 {
                m[n] = i
        }
        res := make([]int, len(nums1))
        for i, n := range nums1 {
                res[i] = -1
                for j := m[n] + 1; j < len(nums2); j++ {
                        if n < nums2[j] {
                                res[i] = nums2[j]
                                break
                        }
                }
        }
        return res
}

#
func nextGreaterElement(nums1 []int, nums2 []int) []int {
        m := make(map[int]int)
        res := make([]int, len(nums1))
        for i := 0; i < len(nums2); i++ {
                for j := i + 1; j < len(nums2); j++ {
                        if nums2[j] > nums2[i] {
                                m[nums2[i]] = nums2[j]
```

(续下页)

(接上页)

```
                                break
                        }
                }
        }
        for key, value := range nums1 {
                if _, ok := m[value]; ok {
                        res[key] = m[value]
                } else {
                        res[key] = -1
                }
        }
        return res
}

#
func nextGreaterElement(nums1 []int, nums2 []int) []int {
        m := make(map[int]int)
        res := make([]int, len(nums1))
        stack := make([]int, 0)
        for i := 0; i < len(nums2); i++ {
                if len(stack) > 0 {
                        for len(stack) > 0 && nums2[i] > stack[len(stack)-1] {
                                top := stack[len(stack)-1]
                                m[top] = nums2[i]
                                stack = stack[:len(stack)-1]
                        }
                }
                stack = append(stack, nums2[i])
        }
        for key, value := range nums1 {
                if _, ok := m[value]; ok {
                        res[key] = m[value]
                } else {
                        res[key] = -1
                }
        }
        return res
}
```

13.25 500. 键盘行 (4)

- 题目

```
给定一个单词列表，只返回可以使用在键盘同一行的字母打印出来的单词。键盘如下图所示。
示例：
输入：["Hello", "Alaska", "Dad", "Peace"]
输出：["Alaska", "Dad"]
注意：
    你可以重复使用键盘上同一字符。
    你可以假设输入的字符串将只包含字母。
```

- 解题思路

```
func findWords(words []string) []string {
        m := make(map[byte]int)
        m['q'] = 1
        m['w'] = 1
        m['e'] = 1
        m['r'] = 1
        m['t'] = 1
        m['y'] = 1
        m['u'] = 1
        m['i'] = 1
        m['o'] = 1
        m['p'] = 1
        m['a'] = 2
        m['s'] = 2
        m['d'] = 2
        m['f'] = 2
        m['g'] = 2
        m['h'] = 2
        m['j'] = 2
        m['k'] = 2
        m['l'] = 2
        m['z'] = 3
        m['x'] = 3
        m['c'] = 3
        m['v'] = 3
        m['b'] = 3
        m['n'] = 3
        m['m'] = 3

        res := make([]string, 0)
```

(续下页)

(接上页)

```
        for i := 0; i < len(words); i++ {
                b := []byte(strings.ToLower(words[i]))
                level := m[b[0]]
                flag := true
                for j := 1; j < len(b); j++ {
                        if m[b[j]] != level {
                                flag = false
                                break
                        }
                }
                if flag {
                        res = append(res, words[i])
                }
        }
        return res
}

#
var qRow = map[byte]bool{
        'q': true,
        'w': true,
        'e': true,
        'r': true,
        't': true,
        'y': true,
        'u': true,
        'i': true,
        'o': true,
        'p': true,
}

var aRow = map[byte]bool{
        'a': true,
        's': true,
        'd': true,
        'f': true,
        'g': true,
        'h': true,
        'j': true,
        'k': true,
        'l': true,
}
```

(续下页)

(接上页)

```
var zRow = map[byte]bool{
        'z': true,
        'x': true,
        'c': true,
        'v': true,
        'b': true,
        'n': true,
        'm': true,
}

func findWords(words []string) []string {
        res := make([]string, 0, len(words))
        for _, word := range words {
                w := strings.ToLower(word)
                if isAllIn(w, qRow) || isAllIn(w, aRow) || isAllIn(w, zRow) {
                        res = append(res, word)
                }
        }
        return res
}

func isAllIn(s string, Row map[byte]bool) bool {
        for i := range s {
                if !Row[s[i]] {
                        return false
                }
        }
        return true
}

#
func findWords(words []string) []string {
        res := make([]string, 0, len(words))
        for _, word := range words {
                w := strings.ToLower(word)
                flag := 0
                for _, m := range w {
                        switch m {
                        case 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p':
                                if flag != 0 && flag != 1 {
                                        flag = 4
                                        break
                                }
```

(续下页)

(接上页)

```
                                flag = 1
                        case 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l':
                                if flag != 0 && flag != 2 {
                                        flag = 4
                                        break
                                }
                                flag = 2
                        case 'z', 'x', 'c', 'v', 'b', 'n', 'm':
                                if flag != 0 && flag != 3 {
                                        flag = 4
                                        break
                                }
                                flag = 3
                        default:
                                flag = 4
                        }
                }
                if flag != 0 && flag != 4 {
                        res = append(res, word)
                }
        }
        return res
}

#
func findWords(words []string) []string {
        res := make([]string, 0, len(words))
        q := "qwertyuiopQWERTYUIOP"
        a := "asdfghjklASDFGHJKL"
        z := "zxcvbnmZXCVBNM"
        for _, word := range words {
                qLen, aLen, zLen := 0, 0, 0
                for i := 0; i < len(word); i++ {
                        if strings.Contains(q, string(word[i])) {
                                qLen++
                        }
                        if strings.Contains(a, string(word[i])) {
                                aLen++
                        }
                        if strings.Contains(z, string(word[i])) {
                                zLen++
                        }
                }
```

(续下页)

(接上页)

```
                if qLen == len(word) || aLen == len(word) || zLen == len(word) {
                        res = append(res, word)
                }
        }
        return res
}
```

CHAPTER 14

0401-0500-Medium

14.1 402. 移掉 K 位数字 (1)

- 题目

```
给定一个以字符串表示的非负整数 num，移除这个数中的 k 位数字，使得剩下的数字最小。
注意:num 的长度小于 10002 且 ≥ k。
    num 不会包含任何前导零。
示例 1 :输入: num = "1432219", k = 3 输出: "1219"
解释: 移除掉三个数字 4, 3, 和 2 形成一个新的最小的数字 1219。
示例 2 :输入: num = "10200", k = 1 输出: "200"
解释: 移掉首位的 1 剩下的数字为 200. 注意输出不能有任何前导零。
示例 3 :输入: num = "10", k = 2 输出: "0"
解释: 从原数字移除所有的数字，剩余为空就是0。
```

- 解题思路

```
func removeKdigits(num string, k int) string {
        stack := make([]byte, 0)
        res := ""
        for i := 0; i < len(num); i++ {
                value := num[i]
                //␣
↪栈顶元素打大于后面的元素，摘除栈顶元素（因为前面的更大，需要删除了才能变的最小）
                for len(stack) > 0 && stack[len(stack)-1] > value && k > 0 {
```

(续下页)

(接上页)

```
                        stack = stack[:len(stack)-1]
                        k--
                }
                stack = append(stack, value)
        }
        stack = stack[:len(stack)-k]
        res = strings.TrimLeft(string(stack), "0")
        if res == "" {
                return "0"
        }
        return res
}
```

14.2 406. 根据身高重建队列 (2)

- 题目

```
假设有打乱顺序的一群人站成一个队列。
每个人由一个整数对(h,␣
↪k)表示，其中h是这个人的身高，k是排在这个人前面且身高大于或等于h的人数。
编写一个算法来重建这个队列。
注意：  总人数少于1100人。
示例  输入:[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]
输出:[[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]
```

- 解题思路

```
func reconstructQueue(people [][]int) [][]int {
        sort.Slice(people, func(i, j int) bool {
                if people[i][0] == people[j][0] {
                        return people[i][1] < people[j][1] // k 递增
                }
                return people[i][0] > people[j][0] // 升高 递减
        })
        for i := 0; i < len(people); i++ {
                index := people[i][1]
                p := people[i]
                copy(people[index+1:i+1], people[index:i+1]) // 后移
                people[index] = p
        }
        return people
}
```

(续下页)

(接上页)

```
# 2
func reconstructQueue(people [][]int) [][]int {
        sort.Slice(people, func(i, j int) bool {
                if people[i][0] == people[j][0] {
                        return people[i][1] < people[j][1] // k 递增
                }
                return people[i][0] > people[j][0] // 升高 递减
        })
        for i := 0; i < len(people); i++ {
                index := people[i][1]
                p := people[i]
                // copy(people[index+1:i+1], people[index:i+1]) // 后移
                for j := i; j > index; j-- {
                        people[j] = people[j-1]
                }
                people[index] = p
        }
        return people
}
```

14.3 413. 等差数列划分 (3)

- 题目

```
如果一个数列至少有三个元素，并且任意两个相邻元素之差相同，则称该数列为等差数列。
例如，以下数列为等差数列:
1, 3, 5, 7, 9
7, 7, 7, 7
3, -1, -5, -9
以下数列不是等差数列。
1, 1, 2, 5, 7
数组 A 包含 N 个数，且索引从0开始。
数组 A 的一个子数组划分为数组 (P, Q)，P 与 Q 是整数且满足 0<=P<Q<N 。
如果满足以下条件，则称子数组(P, Q)为等差数组:
元素 A[P], A[p + 1], ..., A[Q - 1], A[Q] 是等差的。并且P + 1 < Q 。
函数要返回数组 A 中所有为等差数组的子数组个数。
示例:A = [1, 2, 3, 4]
返回: 3, A 中有三个子等差数组: [1, 2, 3], [2, 3, 4] 以及自身 [1, 2, 3, 4]。
```

- 解题思路

```
func numberOfArithmeticSlices(A []int) int {
        n := len(A)
        if n < 3 {
                return 0
        }
        res := 0
        count := 2
        diff := A[1] - A[0]
        for i := 2; i < n; i++ {
                a := A[i] - A[i-1]
                if a == diff {
                        count++
                } else {
                        if count >= 3 {
                                res = res + getValue(count)
                        }
                        count = 2
                        diff = a
                }
        }
        if count >= 3 {
                res = res + getValue(count)
        }
        return res
}

func getValue(num int) int {
        n := num - 2
        return n * (n + 1) / 2
}

# 2
func numberOfArithmeticSlices(A []int) int {
        n := len(A)
        if n < 3 {
                return 0
        }
        res := 0
        dp := make([]int, n)
        for i := 2; i < n; i++ {
                if A[i]-A[i-1] == A[i-1]-A[i-2] {
                        dp[i] = dp[i-1] + 1
                }
                res = res + dp[i]
```

(续下页)

(接上页)

```
        }
        return res
}

# 3
func numberOfArithmeticSlices(A []int) int {
        n := len(A)
        if n < 3 {
                return 0
        }
        res := 0
        for i := 0; i < n-2; i++ {
                diff := A[i+1] - A[i]
                for j := i + 2; j < n; j++ {
                        if A[j]-A[j-1] == diff {
                                res++
                        } else {
                                break
                        }
                }
        }
        return res
}
```

14.4 416. 分割等和子集 (3)

- 题目

```
给定一个只包含正整数的非空数组。是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。
注意:每个数组中的元素不会超过 100
    数组的大小不会超过 200
示例 1:输入: [1, 5, 11, 5] 输出: true
解释: 数组可以分割成 [1, 5, 5] 和 [11].
示例 2:输入: [1, 2, 3, 5] 输出: false
解释: 数组不能分割成两个元素和相等的子集.
```

- 解题思路

```
func canPartition(nums []int) bool {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
```

(续下页)

(接上页)

```
        }
        if sum%2 == 1 {
                return false
        }
        target := sum / 2
        // 题目转换为0-1背包问题，容量为sum/2
        dp := make([][]bool, len(nums)+1)
        for i := 0; i <= len(nums); i++ {
                dp[i] = make([]bool, target+1)
                dp[i][0] = true
        }
        for i := 1; i <= len(nums); i++ {
                for j := 1; j <= target; j++ {
                        if j-nums[i-1] < 0 {
                                dp[i][j] = dp[i-1][j]
                        } else {
                                dp[i][j] = dp[i-1][j] || dp[i-1][j-nums[i-1]]
                        }
                }
        }
        return dp[len(nums)][target]
}

# 2
func canPartition(nums []int) bool {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum%2 == 1 {
                return false
        }
        target := sum / 2
        // 题目转换为0-1背包问题，容量为sum/2
        dp := make([]bool, target+1)
        dp[0] = true
        for i := 0; i < len(nums); i++ {
                for j := target; j >= 0; j-- {
                        if j-nums[i] >= 0 && dp[j-nums[i]] == true {
                                dp[j] = true
                        }
                }
        }
```

(续下页)

(接上页)

```
        return dp[target]
}

# 3
func canPartition(nums []int) bool {
        sort.Ints(nums)
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum%2 == 1 {
                return false
        }
        target := sum / 2
        return dfs(nums, target, 0)
}

func dfs(nums []int, target int, index int) bool {
        if target == 0 {
                return true
        }
        for i := index; i < len(nums); i++ {
                if index < i && nums[i] == nums[i-1] {
                        continue
                }
                if target-nums[i] < 0 {
                        return false
                }
                if dfs(nums, target-nums[i], i+1) == true {
                        return true
                }
        }
        return false
}
```

14.5 417. 太平洋大西洋水流问题 (2)

- 题目

```
给定一个 m x n 的非负整数矩阵来表示一片大陆上各个单元格的高度。
“太平洋”处于大陆的左边界和上边界，而“大西洋”处于大陆的右边界和下边界。
规定水流只能按照上、下、左、右四个方向流动，且只能从高到低或者在同等高度上流动。
请找出那些水流既可以流动到“太平洋”，又能流动到“大西洋”的陆地单元的坐标。
提示：输出坐标的顺序不重要
m 和 n 都小于150
示例：给定下面的 5x5 矩阵：
太平洋 ~   ~   ~   ~   ~
       ~  1   2   2   3  (5) *
       ~  3   2   3  (4) (4) *
       ~  2   4  (5)  3   1  *
       ~ (6) (7)  1   4   5  *
       ~ (5)  1   1   2   4  *
          *   *   *   *   * 大西洋
返回:[[0, 4], [1, 3], [1, 4], [2, 2], [3, 0], [3, 1], [4, 0]] (上图中带括号的单元).
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var n, m int

func pacificAtlantic(heights [][]int) [][]int {
        res := make([][]int, 0)
        n, m = len(heights), len(heights[0])
        A, B := make([][]bool, n), make([][]bool, n)
        for i := 0; i < n; i++ {
                A[i], B[i] = make([]bool, m), make([]bool, m)
        }
        for i := 0; i < n; i++ { // 枚举左右两边往中间走
                dfs(heights, A, i, 0) // 最左边（同上边）走到A
                dfs(heights, B, i, m-1)
        }
        for j := 0; j < m; j++ { // 枚举上下两边往中间走
                dfs(heights, A, 0, j) // 最上边（同左边）走到A
                dfs(heights, B, n-1, j)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if A[i][j] == true && B[i][j] == true {
```

(续下页)

(接上页)

```
                                res = append(res, []int{i, j})
                        }
                }
        }
        return res
}

func dfs(heights [][]int, visited [][]bool, i, j int) {
        visited[i][j] = true
        for k := 0; k < 4; k++ {
                x, y := dx[k]+i, dy[k]+j
                if 0 <= x && x < n && 0 <= y && y < m &&
                        heights[x][y] >= heights[i][j] && visited[x][y] == false {
                        dfs(heights, visited, x, y)
                }
        }
}

# 2
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var queue [][2]int
var n, m int

func pacificAtlantic(heights [][]int) [][]int {
        res := make([][]int, 0)
        n, m = len(heights), len(heights[0])
        queue = make([][2]int, 0)
        A, B := make([][]bool, n), make([][]bool, n)
        for i := 0; i < n; i++ {
                A[i], B[i] = make([]bool, m), make([]bool, m)
        }
        for i := 0; i < n; i++ { // 枚举左右两边往中间走
                queue = append(queue, [2]int{i, 0})
                A[i][0] = true
                bfs(heights, A) // 最左边（同上边）走到A
                queue = append(queue, [2]int{i, m - 1})
                B[i][m-1] = true
                bfs(heights, B)
        }
        for j := 0; j < m; j++ { // 枚举上下两边往中间走
                queue = append(queue, [2]int{0, j})
                A[0][j] = true
```

(续下页)

(接上页)

```
                bfs(heights, A) // 最上边（同左边）走到A
                queue = append(queue, [2]int{n - 1, j})
                B[n-1][j] = true
                bfs(heights, B)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if A[i][j] == true && B[i][j] == true {
                                res = append(res, []int{i, j})
                        }
                }
        }
        return res
}

func bfs(heights [][]int, visited [][]bool) {
        for len(queue) > 0 {
                i, j := queue[0][0], queue[0][1]
                queue = queue[1:]
                for k := 0; k < 4; k++ {
                        x, y := dx[k]+i, dy[k]+j
                        if 0 <= x && x < n && 0 <= y && y < m &&
                                heights[x][y] >= heights[i][j] && visited[x][y] ==␣
↪false {
                                visited[x][y] = true
                                queue = append(queue, [2]int{x, y})
                        }
                }
        }
}
```

14.6 419. 甲板上的战舰 (3)

- 题目

```
给定一个二维的甲板， 请计算其中有多少艘战舰。战舰用'X'表示，空位用'.
↪'表示。你需要遵守以下规则：
给你一个有效的甲板，仅由战舰或者空位组成。
战舰只能水平或者垂直放置。换句话说,战舰只能由1xN (1 行, N 列)组成，
或者Nx1 (N 行, 1 列)组成，其中N可以是任意大小。
两艘战舰之间至少有一个水平或垂直的空位分隔- 即没有相邻的战舰。
示例 ：
```

(续下页)

(接上页)

```
X..X
...X
...X
在上面的甲板中有2艘战舰。
无效样例 :
...X
XXXX
...X
你不会收到这样的无效甲板- 因为战舰之间至少会有一个空位将它们分开。
进阶:你可以用一次扫描算法，只使用O(1)额外空间，并且不修改甲板的值来解决这个问题吗？
```

- 解题思路

```
func countBattleships(board [][]byte) int {
        res := 0
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        if board[i][j] == 'X' {
                                if (i > 0 && board[i-1][j] == 'X') ||
                                        (j > 0 && board[i][j-1] == 'X') {
                                        continue
                                }
                                res++
                        }
                }
        }
        return res
}

# 2
func countBattleships(board [][]byte) int {
        res := 0
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[i]); j++ {
                        if board[i][j] == 'X' {
                                res++
                                board[i][j] = '.'
                                for x := i + 1; x < len(board); x++ {
                                        if board[x][j] == 'X' {
                                                board[x][j] = '.'
                                        } else {
                                                break
                                        }
                                }
```

(续下页)

(接上页)

```
                                for y := j + 1; y < len(board[i]); y++ {
                                        if board[i][y] == 'X' {
                                                board[i][y] = '.'
                                        } else {
                                                break
                                        }
                                }
                        }
                }
        }
        return res
}

# 3
func countBattleships(board [][]byte) int {
        a, b := len(board), len(board[0])
        fa = Init(a * b)
        m := make(map[int]bool)
        arr := make([]int, 0)
        for i := 0; i < a; i++ {
                for j := 0; j < b; j++ {
                        if board[i][j] == 'X' {
                                if i < a-1 && board[i][j] == board[i+1][j] {
                                        union(i*b+j, i*b+b+j)
                                }
                                if j < b-1 && board[i][j] == board[i][j+1] {
                                        union(i*b+j, i*b+j+1)
                                }
                                arr = append(arr, i*b+j)
                        }
                }
        }
        for i := 0; i < len(arr); i++ {
                m[find(arr[i])] = true
        }
        return len(m)
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

func union(a, b int) {
        x, y := find(a), find(b)
        if x != y {
                fa[x] = y
        }
}

// 彻底路径压缩
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}
```

14.7 421. 数组中两个数的最大异或值 (2)

- 题目

```
给你一个整数数组 nums ，返回 nums[i] XOR nums[j] 的最大运算结果，其中 0 ≤ i ≤ j < n 。
进阶：你可以在 O(n) 的时间解决这个问题吗？
示例 1：输入：nums = [3,10,5,25,2,8] 输出：28
解释：最大运算结果是 5 XOR 25 = 28.
示例 2：输入：nums = [0] 输出：0
示例 3：输入：nums = [2,4] 输出：6
示例 4：输入：nums = [8,10,2] 输出：10
示例 5：输入：nums = [14,70,53,83,49,91,36,80,92,51,66,70] 输出：127
提示：1 <= nums.length <= 2 * 104
0 <= nums[i] <= 231 - 1
```

- 解题思路

```
func findMaximumXOR(nums []int) int {
        res := 0
        target := 0
        for i := 31; i >= 0; i-- { // 枚举每一位（第i位，从右到左），判断该为能否为1
```

(续下页)

(接上页)

```
                m := make(map[int]bool)
                target = target | (1 << i) // target第i位置1
                for j := 0; j < len(nums); j++ {
                        m[nums[j]&target] = true // 高位&: 取前缀
                }
                temp := res | (1 << i) // 假设结果第i位为1
                // a ^ b = temp
                // temp ^ a = b
                for k := range m {
                        if _, ok := m[temp^k]; ok {
                                res = temp // 能取到1
                                break
                        }
                }
        }
        return res
}

# 2
func findMaximumXOR(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        res := 0
        root := Trie{
                next: make([]*Trie, 2), // 0和1
        }
        for i := 0; i < n; i++ {
                temp := &root
                for j := 31; j >= 0; j-- {
                        value := (nums[i] >> j) & 1
                        if temp.next[value] == nil {
                                temp.next[value] = &Trie{
                                        next: make([]*Trie, 2),
                                }
                        }
                        temp = temp.next[value]
                }
        }
        for i := 0; i < n; i++ {
                temp := &root
                cur := 0
```

(续下页)

(接上页)

```
                for j := 31; j >= 0; j-- {
                        value := (nums[i] >> j) & 1
                        if temp.next[value^1] != nil { // 能取到1
                                cur = cur | (1 << j) // 结果在该位可以为1
                                temp = temp.next[value^1]
                        } else {
                                temp = temp.next[value]
                        }
                }
                res = max(res, cur)
        }
        return res
}

type Trie struct {
        next []*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

14.8 423. 从英文中重建数字 (1)

- 题目

```
给定一个非空字符串，其中包含字母顺序打乱的英文单词表示的数字0-
↪9。按升序输出原始的数字。
注意:输入只包含小写英文字母。
输入保证合法并可以转换为原始的数字，这意味着像 "abc" 或 "zerone" 的输入是不允许的。
输入字符串的长度小于 50,000。
示例 1:输入: "owoztneoer"输出: "012" (zeroonetwo)
示例 2:输入: "fviefuro"输出: "45" (fourfive)
```

- 解题思路

```
func originalDigits(s string) string {
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
```

(续下页)

(接上页)

```
                m[s[i]]++
        }
        // 利用字母唯一性
        arr := [10]int{}
        arr[0] = m['z']
        arr[2] = m['w']
        arr[4] = m['u']
        arr[6] = m['x']
        arr[8] = m['g']
        arr[3] = m['t'] - arr[2] - arr[8]
        arr[1] = m['o'] - arr[0] - arr[2] - arr[4]
        arr[7] = m['s'] - arr[6]
        arr[5] = m['v'] - arr[7]
        arr[9] = m['i'] - arr[5] - arr[6] - arr[8]
        res := make([]byte, 0)
        for i := 0; i < 10; i++ {
                for j := 0; j < arr[i]; j++ {
                        res = append(res, byte(i+'0'))
                }
        }
        return string(res)
}
```

14.9 424. 替换后的最长重复字符 (3)

- 题目

```
给你一个仅由大写英文字母组成的字符串，你可以将任意位置上的字符替换成另外的字符，总共可最多替换k次。
在执行上述操作后，找到包含重复字母的最长子串的长度。
注意:字符串长度 和 k 不会超过104。
示例 1:输入:s = "ABAB", k = 2输出: 4
解释:用两个'A'替换为两个'B',反之亦然。
示例 2:输入:s = "AABABBA", k = 1 输出: 4
解释:将中间的一个'A'替换为'B',字符串变为 "AABBBBA"。
子串 "BBBB" 有最长重复字母，答案为 4。
```

- 解题思路

```
func characterReplacement(s string, k int) int {
        if s == "" {
                return 0
        }
```

(续下页)

(接上页)

```
        res := 0
        left := 0
        count := 0
        arr := [26]int{}
        for right := 0; right < len(s); right++ {
                arr[s[right]-'A']++
                if arr[s[right]-'A'] > count {
                        count = arr[s[right]-'A']
                }
                for right-left+1-count > k {
                        arr[s[left]-'A']--
                        left++
                }
                if right-left+1 > res {
                        res = right - left + 1
                }
        }
        return res
}

# 2
func characterReplacement(s string, k int) int {
        if s == "" {
                return 0
        }
        left := 0
        count := 0
        arr := [26]int{}
        for right := 0; right < len(s); right++ {
                arr[s[right]-'A']++
                if arr[s[right]-'A'] > count {
                        count = arr[s[right]-'A']
                }
                if right-left+1-count > k { // 窗口内不同字符超过k个开始收缩左边界
                        arr[s[left]-'A']--
                        left++
                }
        }
        return len(s) - left
}

# 3
func characterReplacement(s string, k int) int {
```

(续下页)

(接上页)

```
        if s == "" {
                return 0
        }
        res := 0
        for i := 0; i < len(s); i++ {
                temp := k
                j := i + 1
                for ; j < len(s); j++ {
                        if s[j] != s[i] {
                                if temp == 0 {
                                        break
                                }
                                temp--
                        }
                }
                if j-i+temp > len(s) {
                        return len(s)
                }
                if j-i+temp > res {
                        res = j - i + temp
                }
        }
        return res
}
```

14.10 427. 建立四叉树 (2)

- 题目

```
给你一个 n * n 矩阵 grid ，矩阵由若干 0 和 1 组成。请你用四叉树表示该矩阵 grid 。
你需要返回能表示矩阵的 四叉树 的根结点。
注意，当 isLeaf 为 False 时，你可以把 True 或者 False␣
↪赋值给节点，两种值都会被判题机制 接受 。
四叉树数据结构中，每个内部节点只有四个子节点。此外，每个节点都有两个属性：
val：储存叶子结点所代表的区域的值。1 对应 True，0 对应 False；
isLeaf: 当这个节点是一个叶子结点时为 True，如果它有 4 个子节点则为 False 。
class Node {
    public boolean val;
  public boolean isLeaf;
  public Node topLeft;
  public Node topRight;
  public Node bottomLeft;
```

(续下页)

(接上页)

```
  public Node bottomRight;
}
我们可以按以下步骤为二维区域构建四叉树：
如果当前网格的值相同（即，全为 0 或者全为 1），将 isLeaf 设为 True ，
将 val 设为网格相应的值，并将四个子节点都设为 Null 然后停止。
如果当前网格的值不同，将 isLeaf 设为 False， 将 val␣
↪设为任意值，然后如下图所示，将当前网格划分为四个子网格。
使用适当的子网格递归每个子节点。
如果你想了解更多关于四叉树的内容，可以参考 wiki 。
四叉树格式：输出为使用层序遍历后四叉树的序列化形式，其中 null␣
↪表示路径终止符，其下面不存在节点。
它与二叉树的序列化非常相似。唯一的区别是节点以列表形式表示 [isLeaf, val] 。
如果 isLeaf 或者 val 的值为 True ，则表示它在列表[isLeaf, val] 中的值为 1 ；
如果 isLeaf 或者 val 的值为 False ，则表示值为 0 。
示例 1：输入：grid = [[0,1],[1,0]] 输出：[[0,1],[1,0],[1,1],[1,1],[1,0]]
解释：此示例的解释如下：
请注意，在下面四叉树的图示中，0 表示 false，1 表示 True 。
示例 2：输入：grid = [[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0],[1,1,1,1,1,1,1,1],
[1,1,1,1,1,1,1,1],[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0],[1,1,1,1,0,0,
↪0,0]]
输出：[[0,1],[1,1],[0,1],[1,1],[1,0],null,null,null,null,[1,0],[1,0],[1,1],[1,1]]
解释：网格中的所有值都不相同。我们将网格划分为四个子网格。
topLeft，bottomLeft 和 bottomRight 均具有相同的值。
topRight 具有不同的值，因此我们将其再分为 4 个子网格，这样每个子网格都具有相同的值。
解释如下图所示：
示例 3：输入：grid = [[1,1],[1,1]] 输出：[[1,1]]
示例 4：输入：grid = [[0]] 输出：[[1,0]]
示例 5：输入：grid = [[1,1,0,0],[1,1,0,0],[0,0,1,1],[0,0,1,1]] 输出：[[0,1],[1,1],[1,
↪0],[1,0],[1,1]]
提示：n == grid.length == grid[i].length
n == 2^x 其中 0 <= x <= 6
```

- 解题思路

```
func construct(grid [][]int) *Node {
        n := len(grid)
        return dfs(grid, 0, 0, n, n)
}

func dfs(grid [][]int, x1, y1, x2, y2 int) *Node {
        if x1+1 == x2 {
                value := false
                if grid[x1][y1] == 1 {
                        value = true
```

(续下页)

(接上页)

```
                }
                return &Node{
                        Val:    value,
                        IsLeaf: true,
                }
        }
        midX := (x1 + x2) / 2
        midY := (y1 + y2) / 2
        tL := dfs(grid, x1, y1, midX, midY)
        tR := dfs(grid, x1, midY, midX, y2)
        bL := dfs(grid, midX, y1, x2, midY)
        bR := dfs(grid, midX, midY, x2, y2)
        if tL.IsLeaf == true && tR.IsLeaf == true && bL.IsLeaf == true && bR.IsLeaf␣
↪== true &&
                ((tL.Val == true && tR.Val == true && bL.Val == true && bR.Val ==␣
↪true) ||
                        (tL.Val == false && tR.Val == false && bL.Val == false && bR.
↪Val == false)) {
                return &Node{
                        Val:    tL.Val,
                        IsLeaf: true,
                }
        }
        return &Node{
                Val:         false,
                IsLeaf:      false,
                TopLeft:     tL,
                TopRight:    tR,
                BottomLeft:  bL,
                BottomRight: bR,
        }
}

# 2
func construct(grid [][]int) *Node {
        n := len(grid)
        return dfs(grid, 0, 0, n, n)
}

func dfs(grid [][]int, x1, y1, x2, y2 int) *Node {
        isLeaf := true
        for i := x1; i < x2; i++ {
                for j := y1; j < y2; j++ {
```

(续下页)

(接上页)

```
                        if grid[i][j] != grid[x1][y1] {
                                isLeaf = false
                                break
                        }
                }
        }
        if isLeaf == true {
                return &Node{
                        Val:    grid[x1][y1] == 1,
                        IsLeaf: true,
                }
        }
        midX := (x1 + x2) / 2
        midY := (y1 + y2) / 2
        tL := dfs(grid, x1, y1, midX, midY)
        tR := dfs(grid, x1, midY, midX, y2)
        bL := dfs(grid, midX, y1, x2, midY)
        bR := dfs(grid, midX, midY, x2, y2)
        return &Node{
                Val:         false,
                IsLeaf:      false,
                TopLeft:     tL,
                TopRight:    tR,
                BottomLeft:  bL,
                BottomRight: bR,
        }
}
```

14.11 429.N 叉树的层序遍历 (2)

- 题目

```
给定一个 N 叉树，返回其节点值的层序遍历。 (即从左到右，逐层遍历)。
例如，给定一个 3叉树 :
返回其层序遍历:
[
     [1],
     [3,2,4],
     [5,6]
]
说明:
    树的深度不会超过 1000。
```

(续下页)

(接上页)

树的节点总数不会超过 5000。

- 解题思路

```
func levelOrder(root *Node) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        queue := make([]*Node, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                temp := make([]int, 0)
                length := len(queue)
                for i := 0; i < length; i++ {
                        if queue[i] != nil {
                                temp = append(temp, queue[i].Val)
                                for j := 0; j < len(queue[i].Children); j++ {
                                        queue = append(queue, queue[i].Children[j])
                                }
                        }
                }
                res = append(res, temp)
                queue = queue[length:]
        }
        return res
}

# 2
var res [][]int

func levelOrder(root *Node) [][]int {
        res = make([][]int, 0)
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *Node, level int) {
        if root == nil {
                return
        }
```

(续下页)

(接上页)

```
        if len(res) == level {
                res = append(res, make([]int, 0))
        }
        res[level] = append(res[level], root.Val)
        for i := 0; i < len(root.Children); i++ {
                dfs(root.Children[i], level+1)
        }
}
```

14.12 430. 扁平化多级双向链表 (3)

- 题目

```
多级双向链表中，除了指向下一个节点和前一个节点指针之外，它还有一个子链表指针，可能指向单独的双向链表。
这些子列表也可能会有一个或多个自己的子项，依此类推，生成多级数据结构，如下面的示例所示。
给你位于列表第一级的头节点，请你扁平化列表，使所有结点出现在单级双链表中。
示例 1：输入：head = [1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12]
输出：[1,2,3,7,8,11,12,9,10,4,5,6]
解释：输入的多级列表如下图所示：
扁平化后的链表如下图：
示例 2：输入：head = [1,2,null,3] 输出：[1,3,2]
解释：输入的多级列表如下图所示：
  1---2---NULL
  |
  3---NULL
示例 3：输入：head = [] 输出：[]
如何表示测试用例中的多级链表？
以 示例 1 为例：
 1---2---3---4---5---6--NULL
         |
         7---8---9---10--NULL
             |
             11--12--NULL
序列化其中的每一级之后：
[1,2,3,4,5,6,null]
[7,8,9,10,null]
[11,12,null]
为了将每一级都序列化到一起，我们需要每一级中添加值为 null␣
↪的元素，以表示没有节点连接到上一级的上级节点。
[1,2,3,4,5,6,null]
[null,null,7,8,9,10,null]
[null,11,12,null]
```

(续下页)

(接上页)

```
合并所有序列化结果，并去除末尾的 null 。
[1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12]
提示：节点数目不超过 1000
1 <= Node.val <= 10^5
```

- 解题思路

```
func flatten(root *Node) *Node {
        if root == nil {
                return nil
        }
        res := &Node{}
        cur := res
        for root != nil {
                cur.Next = root
                root.Prev = cur
                cur = cur.Next
                root = root.Next
                // 处理子节点
                if cur.Child != nil {
                        ch := flatten(cur.Child)
                        cur.Child = nil
                        cur.Next = ch
                        ch.Prev = cur
                        // 指针移动
                        for cur.Next != nil {
                                cur = cur.Next
                        }
                }
        }
        res.Next.Prev = nil
        return res.Next
}

# 2
var arr []*Node

func flatten(root *Node) *Node {
        arr = make([]*Node, 0)
        dfs(root)
        for i := 0; i < len(arr); i++ {
                if i+1 < len(arr) {
                        arr[i].Next = arr[i+1]
                }
```

(续下页)

(接上页)

```
                if i > 0 {
                        arr[i].Prev = arr[i-1]
                }
                arr[i].Child = nil
        }
        return root
}

func dfs(root *Node) {
        if root == nil {
                return
        }
        arr = append(arr, root)
        dfs(root.Child)
        dfs(root.Next)
}

# 3
func flatten(root *Node) *Node {
        cur := root
        stack := make([]*Node, 0)
        for cur != nil {
                // 处理child
                if cur.Child != nil {
                        if cur.Next != nil {
                                stack = append(stack, cur.Next)
                        }
                        cur.Child.Prev = cur
                        cur.Next = cur.Child
                        cur.Child = nil
                        continue
                }
                if cur.Next != nil {
            cur.Child = nil
                        cur = cur.Next
                        continue
                }
                if len(stack) == 0 {
                        break
                }
                last := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                cur.Next = last
```

(续下页)

(接上页)

```
                last.Prev = cur
                cur = last
        }
        return root
}
```

14.13 433. 最小基因变化 (3)

- 题目

```
一条基因序列由一个带有8个字符的字符串表示，其中每个字符都属于 "A", "C", "G", "T
↪"中的任意一个。
假设我们要调查一个基因序列的变化。一次基因变化意味着这个基因序列中的一个字符发生了变化。
例如，基因序列由"AACCGGTT"变化至"AACCGGTA"即发生了一次基因变化。
与此同时，每一次基因变化的结果，都需要是一个合法的基因串，即该结果属于一个基因库。
现在给定3个参数 — start, end, bank, 分别代表起始基因序列，目标基因序列及基因库，
请找出能够使起始基因序列变化为目标基因序列所需的最少变化次数。如果无法实现目标变化，请返回␣
↪-1。
注意：起始基因序列默认是合法的，但是它并不一定会出现在基因库中。
如果一个起始基因序列需要多次变化，那么它每一次变化之后的基因序列都必须是合法的。
假定起始基因序列与目标基因序列是不一样的。
示例 1: start: "AACCGGTT" end:   "AACCGGTA" bank: ["AACCGGTA"]
返回值: 1
示例 2: start: "AACCGGTT" end:   "AAACGGTA" bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]
返回值: 2
示例 3: start: "AAAAACCC" end:   "AACCCCCC" bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]
返回值: 3
```

- 解题思路

```
func minMutation(start string, end string, bank []string) int {
        arr := []byte{'A', 'T', 'C', 'G'}
        m := make(map[string]bool)
        for i := 0; i < len(bank); i++ {
                m[bank[i]] = true
        }
        if _, ok := m[end]; ok == false {
                return -1
        }
        res := 0
        queue := make([]string, 0)
        queue = append(queue, start)
```

(续下页)

(接上页)

```
        for len(queue) > 0 {
                res++
                length := len(queue)
                for i := 0; i < length; i++ {
                        str := queue[i]
                        for j := 0; j < len(str); j++ {
                                for k := 0; k < len(arr); k++ {
                                        if arr[k] != str[j] {
                                                newStr := str[:j] + string(arr[k]) +␣
↪str[j+1:]
                                                if _, ok := m[newStr]; ok {
                                                        queue = append(queue, newStr)
                                                        delete(m, newStr)
                                                }
                                                if newStr == end {
                                                        return res
                                                }
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return -1
}

# 2
var res int

func minMutation(start string, end string, bank []string) int {
        res = math.MaxInt32
        m := make(map[string]bool)
        for i := 0; i < len(bank); i++ {
                m[bank[i]] = true
        }
        if _, ok := m[end]; ok == false {
                return -1
        }
        dfs(start, end, 0, bank, make([]bool, len(bank)))
        if res == math.MaxInt32 {
                return -1
        }
        return res
```

(续下页)

(接上页)

```
}

func dfs(start string, end string, index int, bank []string, visited []bool) {
        if start == end {
                if index < res {
                        res = index
                }
                return
        }
        for i := 0; i < len(bank); i++ {
                if visited[i] == false && judge(start, bank[i]) {
                        visited[i] = true
                        dfs(bank[i], end, index+1, bank, visited)
                        visited[i] = false
                }
        }
}

func judge(a, b string) bool {
        count := 0
        for i := 0; i < len(a); i++ {
                if a[i] != b[i] {
                        count++
                }
        }
        return count == 1
}

# 3
func minMutation(start string, end string, bank []string) int {
        arr := []byte{'A', 'T', 'C', 'G'}
        m := make(map[string]bool)
        for i := 0; i < len(bank); i++ {
                m[bank[i]] = true
        }
        if _, ok := m[end]; ok == false {
                return -1
        }
        res := 0
        queueA := make([]string, 0)
        queueA = append(queueA, start)
        queueB := make([]string, 0)
        queueB = append(queueB, end)
```

(续下页)

(接上页)

```
        for len(queueA) > 0 {
                res++
                if len(queueA) > len(queueB) {
                        queueA, queueB = queueB, queueA
                }
                length := len(queueA)
                for i := 0; i < length; i++ {
                        str := queueA[i]
                        for j := 0; j < len(str); j++ {
                                for k := 0; k < len(arr); k++ {
                                        if arr[k] != str[j] {
                                                newStr := str[:j] + string(arr[k]) +␣
↪str[j+1:]
                                                if _, ok := m[newStr]; ok {
                                                        queueA = append(queueA,␣
↪newStr)
                                                        delete(m, newStr)
                                                }
                                                for l := 0; l < len(queueB); l++ {
                                                        if newStr == queueB[l] {
                                                                return res
                                                        }
                                                }
                                        }
                                }
                        }
                }
                queueA = queueA[length:]
        }
        return -1
}
```

14.14 435. 无重叠区间 (4)

- 题目

```
给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。
注意:可以认为区间的终点总是大于它的起点。
    区间 [1,2] 和 [2,3] 的边界相互“接触”，但没有相互重叠。
示例 1:输入: [ [1,2], [2,3], [3,4], [1,3] ]输出: 1
解释: 移除 [1,3] 后，剩下的区间没有重叠。
示例 2:输入: [ [1,2], [1,2], [1,2] ]输出: 2
```

(续下页)

(接上页)

```
解释: 你需要移除两个 [1,2] 来使剩下的区间没有重叠。
示例 3:输入: [ [1,2], [2,3] ] 输出: 0
解释: 你不需要移除任何区间，因为它们已经是无重叠的了。
```

- 解题思路

```
func eraseOverlapIntervals(intervals [][]int) int {
        if len(intervals) == 0 {
                return 0
        }
        // 按照结束时间排序
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][1] < intervals[j][1]
        })
        count := 1
        end := intervals[0][1]
        for i := 0; i < len(intervals); i++ {
                node := intervals[i]
                if node[0] >= end {
                        end = node[1]
                        count++
                }
        }
        return len(intervals) - count
}

# 2
func eraseOverlapIntervals(intervals [][]int) int {
        if len(intervals) == 0 {
                return 0
        }
        // 按照结束时间排序
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][1] < intervals[j][1]
        })
        count := 0
        end := intervals[0][1]
        for i := 1; i < len(intervals); i++ {
                node := intervals[i]
                if node[0] >= end {
                        end = node[1]
                } else {
                        if node[1] < end {
                                end = node[1]
```

(续下页)

(接上页)

```
                        }
                        count++
                }
        }
        return count
}

# 3
func eraseOverlapIntervals(intervals [][]int) int {
        if len(intervals) == 0 {
                return 0
        }
        // 按照结束时间排序
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][1] < intervals[j][1]
        })
        dp := make([]int, len(intervals))
        dp[0] = 1
        res := 1
        for i := 1; i < len(intervals); i++ {
                count := 0
                for j := i - 1; j >= 0; j-- {
                        if intervals[j][1] <= intervals[i][0] {
                                count = max(dp[j], count)
                        }
                }
                dp[i] = count + 1
                res = max(res, dp[i])
        }
        return len(intervals) - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func eraseOverlapIntervals(intervals [][]int) int {
        if len(intervals) == 0 {
                return 0
```

(续下页)

(接上页)

```
        }
        // 按照结束时间排序
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][1] < intervals[j][1]
        })
        dp := make([]int, len(intervals))
        dp[0] = 1
        res := 1
        for i := 1; i < len(intervals); i++ {
                count := 0
                for j := i - 1; j >= 0; j-- {
                        if intervals[j][1] <= intervals[i][0] {
                                count = max(dp[j], count)
                                break
                        }
                }
                dp[i] = max(dp[i-1], count+1)
                res = max(res, dp[i])
        }
        return len(intervals) - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

14.15 436. 寻找右区间 (2)

- 题目

```
给你一个区间数组 intervals ，其中intervals[i] = [starti, endi] ，且每个starti 都 不同␣
↪。
区间 i 的 右侧区间 可以记作区间 j ，并满足 startj>= endi ，且 startj 最小化 。
返回一个由每个区间 i 的 右侧区间 的最小起始位置组成的数组。如果某个区间 i␣
↪不存在对应的 右侧区间 ，则下标 i 处的值设为 -1 。
示例 1：输入：intervals = [[1,2]] 输出：[-1]
解释：集合中只有一个区间，所以输出-1。
示例 2：输入：intervals = [[3,4],[2,3],[1,2]] 输出：[-1, 0, 1]
解释：对于 [3,4] ，没有满足条件的“右侧”区间。
```

(续下页)

(接上页)

```
对于 [2,3] ，区间[3,4]具有最小的“右”起点;
对于 [1,2] ，区间[2,3]具有最小的“右”起点。
示例 3：输入：intervals = [[1,4],[2,3],[3,4]] 输出：[-1, 2, -1]
解释：对于区间 [1,4] 和 [3,4] ，没有满足条件的“右侧”区间。
对于 [2,3] ，区间 [3,4] 有最小的“右”起点。
提示：1 <=intervals.length <= 2 * 104
intervals[i].length == 2
-106 <= starti <= endi <= 106
每个间隔的起点都 不相同
```

- 解题思路

```
func findRightInterval(intervals [][]int) []int {
        m := make(map[int]int)
        n := len(intervals)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = -1
                m[intervals[i][0]] = i // 存储start对应的下标，因为起点都不相同
        }
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][0] == intervals[j][0] {
                        return intervals[i][1] < intervals[j][1]
                }
                return intervals[i][0] < intervals[j][0]
        })
        for i := 0; i < n; i++ {
                for j := i; j < n; j++ { //  有坑注意：可以跟自己相比 [[1,1],[3,4]] =>
↪ [0,-1]
                        // 满足startj >= endi的取最小值
                        if intervals[i][1] <= intervals[j][0] {
                                index := m[intervals[i][0]]
                                res[index] = m[intervals[j][0]]
                                break
                        }
                }
        }
        return res
}

# 2
func findRightInterval(intervals [][]int) []int {
        m := make(map[int]int)
        n := len(intervals)
```

(续下页)

(接上页)

```
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = -1
                m[intervals[i][0]] = i // 存储start对应的下标，因为起点都不相同
        }
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][0] == intervals[j][0] {
                        return intervals[i][1] < intervals[j][1]
                }
                return intervals[i][0] < intervals[j][0]
        })
        for i := 0; i < n; i++ {
                target := intervals[i][1]
                index := m[intervals[i][0]]
                left, right := i, n-1
                for left <= right {
                        mid := left + (right-left)/2
                        if target <= intervals[mid][0] {
                                res[index] = m[intervals[mid][0]]
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                }
        }
        return res
}
```

14.16 438. 找到字符串中所有字母异位词 (2)

- 题目

```
给定一个字符串 s 和一个非空字符串 p，找到 s 中所有是 p␣
↪的字母异位词的子串，返回这些子串的起始索引。
字符串只包含小写英文字母，并且字符串 s 和 p 的长度都不超过 20100。
说明：字母异位词指字母相同，但排列不同的字符串。
    不考虑答案输出的顺序。
示例 1:输入: s: "cbaebabacd" p: "abc"输出: [0, 6]
解释:起始索引等于 0 的子串是 "cba", 它是 "abc" 的字母异位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的字母异位词。
示例 2:输入: s: "abab" p: "ab"输出: [0, 1, 2]
解释:起始索引等于 0 的子串是 "ab", 它是 "ab" 的字母异位词。
```

(续下页)

(接上页)

```
起始索引等于 1 的子串是 "ba", 它是 "ab" 的字母异位词。
起始索引等于 2 的子串是 "ab", 它是 "ab" 的字母异位词。
```

- 解题思路

```
func findAnagrams(s string, p string) []int {
        res := make([]int, 0)
        if len(p) > len(s) {
                return res
        }
        arr1, arr2 := [26]int{}, [26]int{}
        for i := 0; i < len(p); i++ {
                arr1[p[i]-'a']++
                arr2[s[i]-'a']++
        }
        for i := 0; i < len(s)-len(p); i++ {
                if arr1 == arr2 {
                        res = append(res, i)
                }
                arr2[s[i]-'a']--
                arr2[s[i+len(p)]-'a']++
        }
        if arr1 == arr2 {
                res = append(res, len(s)-len(p))
        }
        return res
}

# 2
func findAnagrams(s string, p string) []int {
        res := make([]int, 0)
        if len(p) > len(s) {
                return res
        }
        m1, m2 := make(map[byte]int), make(map[byte]int)
        for i := 0; i < len(p); i++ {
                m1[p[i]-'a']++
                m2[s[i]-'a']++
        }
        for i := 0; i < len(s)-len(p); i++ {
                if compare(m1, m2) {
                        res = append(res, i)
                }
                m2[s[i]-'a']--
```

(续下页)

(接上页)

```
                if m2[s[i]-'a'] == 0 {
                        delete(m2, s[i]-'a')
                }
                m2[s[i+len(p)]-'a']++
        }
        if compare(m1, m2) {
                res = append(res, len(s)-len(p))
        }
        return res
}

func compare(m1, m2 map[byte]int) bool {
        if len(m1) != len(m2) {
                return false
        }
        for k := range m1 {
                if m2[k] != m1[k] {
                        return false
                }
        }
        return true
}
```

14.17 442. 数组中重复的数据 (5)

- 题目

```
给定一个整数数组 a，其中1 ≤ a[i] ≤ n （n为数组长度），
↪其中有些元素出现两次而其他元素出现一次。
找到所有出现两次的元素。
你可以不用到任何额外空间并在O(n)时间复杂度内解决这个问题吗？
示例：输入：[4,3,2,7,8,2,3,1] 输出:[2,3]
```

- 解题思路

```
func findDuplicates(nums []int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for k, v := range m {
```

(续下页)

(接上页)

```
                if v == 2 {
                        res = append(res, k)
                }
        }
        return res
}

# 2
func findDuplicates(nums []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                index := abs(nums[i]) - 1
                if nums[index] < 0 {
                        res = append(res, abs(nums[i]))
                } else {
                        nums[index] = -nums[index]
                }
        }
        return res
}

func abs(a int) int {
        if a >= 0 {
                return a
        }
        return -a
}

# 3
func findDuplicates(nums []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                for nums[nums[i]-1] != nums[i] {
                        nums[nums[i]-1], nums[i] = nums[i], nums[nums[i]-1]
                }
        }
        for i := 0; i < len(nums); i++ {
                if nums[i]-1 != i {
                        res = append(res, nums[i])
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 4
func findDuplicates(nums []int) []int {
        res := make([]int, 0)
        n := len(nums)
        for i := 0; i < n; i++ {
                index := nums[i]%(n+1) - 1
                nums[index] = nums[index] + (n + 1)
        }
        for i := 0; i < n; i++ {
                if nums[i]/(n+1) == 2 {
                        res = append(res, i+1)
                }
        }
        return res
}

# 5
func findDuplicates(nums []int) []int {
        if len(nums) == 0 {
                return nil
        }
        sort.Ints(nums)
        res := make([]int, 0)
        prev := nums[0]
        count := 1
        for i := 1; i < len(nums); i++ {
                if prev == nums[i] {
                        count++
                } else {
                        if count == 2 {
                                res = append(res, nums[i-1])
                        }
                        prev = nums[i]
                        count = 1
                }
        }
        if count == 2 {
                res = append(res, nums[len(nums)-1])
        }
        return res
}
```

14.18 445. 两数相加 II(3)

- 题目

给你两个 非空 链表来代表两个非负整数。数字最高位位于链表开始位置。
它们的每个节点只存储一位数字。将这两数相加会返回一个新的链表。
你可以假设除了数字 0 之外，这两个数字都不会以零开头。
进阶：如果输入链表不能修改该如何处理？换句话说，你不能对列表中的节点进行翻转。
示例：输入：(7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4) 输出：7 -> 8 -> 0 -> 7

- 解题思路

```
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        l1 = reverse(l1)
        l2 = reverse(l2)
        res := &ListNode{}
        cur := res
        carry := 0
        for l1 != nil || l2 != nil || carry > 0 {
                sum := carry
                if l1 != nil {
                        sum += l1.Val
                        l1 = l1.Next
                }
                if l2 != nil {
                        sum += l2.Val
                        l2 = l2.Next
                }
                carry = sum / 10 // 进位
                cur.Next = &ListNode{Val: sum % 10}
                cur = cur.Next
        }
        return reverse(res.Next)
}

func reverse(head *ListNode) *ListNode {
        var result *ListNode
        var temp *ListNode
        for head != nil {
                temp = head.Next
                head.Next = result
                result = head
                head = temp
        }
```

(续下页)

(接上页)

```
        return result
}

# 2
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        stack1 := make([]int, 0)
        stack2 := make([]int, 0)
        for l1 != nil {
                stack1 = append(stack1, l1.Val)
                l1 = l1.Next
        }
        for l2 != nil {
                stack2 = append(stack2, l2.Val)
                l2 = l2.Next
        }
        var res *ListNode
        carry := 0
        for len(stack1) > 0 || len(stack2) > 0 || carry > 0 {
                if len(stack1) > 0 {
                        carry = carry + stack1[len(stack1)-1]
                        stack1 = stack1[:len(stack1)-1]
                }
                if len(stack2) > 0 {
                        carry = carry + stack2[len(stack2)-1]
                        stack2 = stack2[:len(stack2)-1]
                }
                temp := &ListNode{
                        Val:  carry % 10,
                        Next: res,
                }
                carry = carry / 10
                res = temp
        }
        return res
}

# 3
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        a, b := l1, l2
        length1, length2 := 0, 0
        for a != nil {
                length1++
                a = a.Next
```

(续下页)

(接上页)

```
        }
        for b != nil {
                length2++
                b = b.Next
        }
        res, carry := add(l1, l2, length1, length2)
        if carry > 0 {
                return &ListNode{Val: carry, Next: res}
        }
        return res
}

func add(l1, l2 *ListNode, length1, length2 int) (res *ListNode, carry int) {
        if l1 != nil && l2 != nil {
                if l1.Next == nil && l2.Next == nil {
                        val := l1.Val + l2.Val
                        carry = val / 10
                        res = &ListNode{Val: val % 10, Next: nil}
                        return
                }
        }
        a := &ListNode{}
        var b, n int
        if length1 > length2 {
                a, b = add(l1.Next, l2, length1-1, length2)
                n = l1.Val + b
        } else if length1 < length2 {
                a, b = add(l1, l2.Next, length1, length2-1)
                n = l2.Val + b
        } else {
                a, b = add(l1.Next, l2.Next, length1-1, length2-1)
                n = l1.Val + l2.Val + b
        }
        res = &ListNode{Val: n % 10, Next: a}
        carry = n / 10
        return
}
```

14.19 449. 序列化和反序列化二叉搜索树 (2)

- 题目

```
序列化是将数据结构或对象转换为一系列位的过程，以便它可以存储在文件或内存缓冲区中，或通过网络连接链路
以便稍后在同一个或另一个计算机环境中重建。
设计一个算法来序列化和反序列化二叉搜索树。 对序列化/反序列化算法的工作方式没有限制。
您只需确保二叉搜索树可以序列化为字符串，并且可以将该字符串反序列化为最初的二叉搜索树。
编码的字符串应尽可能紧凑。
注意：不要使用类成员/全局/静态变量来存储状态。␣
↪你的序列化和反序列化算法应该是无状态的。
```

- 解题思路

```
type Codec struct {
        res []string
}

func Constructor() Codec {
        return Codec{}
}

func (this *Codec) serialize(root *TreeNode) string {
        if root == nil {
                return "#"
        }
        return strconv.Itoa(root.Val) + "," + this.serialize(root.Left) + "," + this.
↪serialize(root.Right)
}

func (this *Codec) deserialize(data string) *TreeNode {
        this.res = strings.Split(data, ",")
        return this.dfsDeserialize()
}

func (this *Codec) dfsDeserialize() *TreeNode {
        node := this.res[0]
        this.res = this.res[1:]
        if node == "#" {
                return nil
        }
        value, _ := strconv.Atoi(node)
        return &TreeNode{
                Val:   value,
```

(续下页)

(接上页)

```
                Left:  this.dfsDeserialize(),
                Right: this.dfsDeserialize(),
        }
}

# 2
type Codec struct {
        res []string
}

func Constructor() Codec {
        return Codec{}
}

func (this *Codec) serialize(root *TreeNode) string {
        if root == nil {
                return ""
        }
        res := make([]string, 0)
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node != nil {
                        res = append(res, strconv.Itoa(node.Val))
                        queue = append(queue, node.Left, node.Right)
                } else {
                        res = append(res, "#")
                }
        }
        return strings.Join(res, ",")
}

func (this *Codec) deserialize(data string) *TreeNode {
        if len(data) == 0 || data == "" {
                return nil
        }
        res := strings.Split(data, ",")
        root := &TreeNode{}
        root.Val, _ = strconv.Atoi(res[0])
        res = res[1:]
        queue := make([]*TreeNode, 0)
```

(续下页)

(接上页)

```
        queue = append(queue, root)
        for len(queue) > 0 {
                if res[0] != "#" {
                        left, _ := strconv.Atoi(res[0])
                        queue[0].Left = &TreeNode{Val: left}
                        queue = append(queue, queue[0].Left)
                }
                if res[1] != "#" {
                        right, _ := strconv.Atoi(res[1])
                        queue[0].Right = &TreeNode{Val: right}
                        queue = append(queue, queue[0].Right)
                }
                queue = queue[1:]
                res = res[2:]
        }
        return root
}
```

14.20 450. 删除二叉搜索树中的节点 (2)

- 题目

```
给定一个二叉搜索树的根节点 root 和一个值 key，删除二叉搜索树中的 key 对应的节点，
并保证二叉搜索树的性质不变。返回二叉搜索树（有可能被更新）的根节点的引用。
一般来说，删除节点可分为两个步骤：
    首先找到需要删除的节点；
    如果找到了，删除它。
说明： 要求算法时间复杂度为 O(h)，h 为树的高度。
示例:root = [5,3,6,2,4,null,7] key = 3
    5
   / \
  3   6
 / \   \
2   4   7
给定需要删除的节点值是 3，所以我们首先找到 3 这个节点，然后删除它。
一个正确的答案是 [5,4,6,2,null,null,7]，如下图所示。
    5
   / \
  4   6
 /     \
2       7
另一个正确答案是 [5,2,6,null,4,null,7]。
```

(续下页)

(接上页)

```
    5
   / \
  2   6
   \   \
    4   7
```

- 解题思路

```
func deleteNode(root *TreeNode, key int) *TreeNode {
        if root == nil {
                return nil
        }
        if key < root.Val {
                root.Left = deleteNode(root.Left, key)
        } else if key > root.Val {
                root.Right = deleteNode(root.Right, key)
        } else {
                if root.Left == nil {
                        return root.Right
                }
                if root.Right == nil {
                        return root.Left
                }
                // 找到右节点的最小值，把左节点给最小值
                minNode := root.Right
                for minNode.Left != nil {
                        minNode = minNode.Left
                }
                minNode.Left = root.Left
                root = root.Right
        }
        return root
}

# 2
func deleteNode(root *TreeNode, key int) *TreeNode {
        if root == nil {
                return nil
        }
        if key < root.Val {
                root.Left = deleteNode(root.Left, key)
        } else if key > root.Val {
                root.Right = deleteNode(root.Right, key)
        } else {
```

(续下页)

(接上页)

```
                if root.Left == nil {
                        return root.Right
                }
                if root.Right == nil {
                        return root.Left
                }
                // 找到左节点的最大值，把右节点给最大值
                maxNode := root.Left
                for maxNode.Right != nil {
                        maxNode = maxNode.Right
                }
                maxNode.Right = root.Right
                root = root.Left
        }
        return root
}
```

14.21 451. 根据字符出现频率排序 (2)

- 题目

```
给定一个字符串，请将字符串里的字符按照出现的频率降序排列。
示例 1:输入:"tree"输出:"eert"
解释:'e'出现两次，'r'和't'都只出现一次。
因此'e'必须出现在'r'和't'之前。此外，"eetr"也是一个有效的答案。
示例 2:输入:"cccaaa"输出:"cccaaa"
解释:'c'和'a'都出现三次。此外，"aaaccc"也是有效的答案。
注意"cacaca"是不正确的，因为相同的字母必须放在一起。
示例 3:输入:"Aabb"输出:"bbAa"
解释:此外，"bbaA"也是一个有效的答案，但"Aabb"是不正确的。
注意'A'和'a'被认为是两种不同的字符。
```

- 解题思路

```
func frequencySort(s string) string {
        m := make(map[int]int)
        for i := 0; i < len(s); i++ {
                m[int(s[i])]++
        }
        arr := make([][2]int, 0)
        for k, v := range m {
                arr = append(arr, [2]int{k, v})
```

(续下页)

(接上页)

```
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] > arr[j][1]
        })
        res := ""
        for i := range arr {
                for j := 0; j < arr[i][1]; j++ {
                        res = res + string(arr[i][0])
                }
        }
        return res
}

# 2
func frequencySort(s string) string {
        m := make(map[byte]string)
        for i := 0; i < len(s); i++ {
                m[s[i]] = m[s[i]] + string(s[i])
        }
        var h HeapString
        heap.Init(&h)
        for _, v := range m {
                heap.Push(&h, v)
        }
        res := ""
        for h.Len() > 0 {
                str := heap.Pop(&h).(string)
                res = res + str
        }
        return res
}

type HeapString []string

func (h HeapString) Len() int {
        return len(h)
}

func (h HeapString) Less(i int, j int) bool {
        return len(h[i]) >= len(h[j])
}

func (h HeapString) Swap(i int, j int) {
```

(续下页)

(接上页)

```
        h[i], h[j] = h[j], h[i]
}

func (h *HeapString) Push(x interface{}) {
        *h = append(*h, x.(string))
}

func (h *HeapString) Pop() interface{} {
        n := len(*h)
        val := (*h)[n-1]
        *h = (*h)[:n-1]
        return val
}
```

14.22 452. 用最少数量的箭引爆气球 (1)

- 题目

```
在二维空间中有许多球形的气球。对于每个气球，提供的输入是水平方向上，气球直径的开始和结束坐标。
由于它是水平的，所以纵坐标并不重要，因此只要知道开始和结束的横坐标就足够了。开始坐标总是小于结束坐标
一支弓箭可以沿着 x 轴从不同点完全垂直地射出。
在坐标 x 处射出一支箭，若有一个气球的直径的开始和结束坐标为 xstart，xend，
且满足 xstart≤ x ≤ xend，则该气球会被引爆。可以射出的弓箭的数量没有限制。
弓箭一旦被射出之后，可以无限地前进。我们想找到使得所有气球全部被引爆，所需的弓箭的最小数量。
给你一个数组 points ，其中 points [i] = [xstart,xend]␣
↪，返回引爆所有气球所必须射出的最小弓箭数。
示例 1：输入：points = [[10,16],[2,8],[1,6],[7,12]] 输出：2
解释：对于该样例，x = 6 可以射爆 [2,8],[1,6] 两个气球，以及 x = 11 射爆另外两个气球
示例 2：输入：points = [[1,2],[3,4],[5,6],[7,8]] 输出：4
示例 3：输入：points = [[1,2],[2,3],[3,4],[4,5]] 输出：2
示例 4：输入：points = [[1,2]] 输出：1
示例 5：输入：points = [[2,3],[2,3]] 输出：1
提示：0 <= points.length <= 104
points[i].length == 2
-231 <= xstart <xend <= 231 - 1
```

- 解题思路

```
func findMinArrowShots(points [][]int) int {
        if len(points) == 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        sort.Slice(points, func(i, j int) bool {
                return points[i][1] < points[j][1]
        })
        right := points[0][1]
        res := 1
        for i := 0; i < len(points); i++ {
                if points[i][0] > right {
                        right = points[i][1]
                        res++
                }
        }
        return res
}
```

14.23 454. 四数相加 II(2)

- 题目

```
给定四个包含整数的数组列表 A , B , C , D ,计算有多少个元组 (i, j, k, l) ,
使得 A[i] + B[j] + C[k] + D[l] = 0。
为了使问题简单化，所有的 A, B, C, D 具有相同的长度 N，且 0 ≤ N ≤ 500 。
所有整数的范围在 -228 到 228 - 1 之间，最终结果不会超过 231 - 1 。
例如:输入:
A = [ 1, 2]
B = [-2,-1]
C = [-1, 2]
D = [ 0, 2]
输出:2
解释:两个元组如下:
1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0
2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0
```

- 解题思路

```
func fourSumCount(A []int, B []int, C []int, D []int) int {
        res := 0
        m := make(map[int]int)
        for _, a := range A {
                for _, b := range B {
                        m[a+b]++
                }
        }
```

(续下页)

(接上页)

```
        for _, c := range C {
                for _, d := range D {
                        res = res + m[0-c-d]
                }
        }
        return res
}

# 2
func fourSumCount(A []int, B []int, C []int, D []int) int {
        res := 0
        mA := make(map[int]int)
        mB := make(map[int]int)
        for _, a := range A {
                for _, b := range B {
                        mA[a+b]++
                }
        }
        for _, c := range C {
                for _, d := range D {
                        mB[c+d]++
                }
        }
        for k, v := range mA {
                res = res + v*mB[-k]
        }
        return res
}
```

14.24 456.132 模式 (3)

- 题目

```
给定一个整数序列：a1, a2, ..., an, 一个132模式的子序列ai, aj, ak被定义为：
当 i < j < k 时，ai < ak < aj。
设计一个算法，当给定有n 个数字的序列时，验证这个序列中是否含有132模式的子序列。
注意：n 的值小于15000。
示例1:输入：[1, 2, 3, 4] 输出：False
解释：序列中不存在132模式的子序列。
示例 2:输入：[3, 1, 4, 2]输出：True
解释：序列中有 1 个132模式的子序列： [1, 4, 2].
示例 3:输入：[-1, 3, 2, 0] 输出：True
```

(续下页)

(接上页)

```
解释: 序列中有 3 个132模式的的子序列: [-1, 3, 2], [-1, 3, 0] 和 [-1, 2, 0].
```

- 解题思路

```
func find132pattern(nums []int) bool {
        if len(nums) < 3 {
                return false
        }
        minArr := make([]int, len(nums)) // minArr[i]是[0,i]中的最小值
        minArr[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                minArr[i] = min(nums[i], minArr[i-1])
        }
        stack := make([]int, 0)
        for j := len(nums) - 1; j >= 0; j-- {
                // a[i]<a[k]<a[j] => min[j]<a[k]<a[j]
                if nums[j] > minArr[j] {
                        for len(stack) > 0 && stack[len(stack)-1] <= minArr[j] {
                                stack = stack[:len(stack)-1]
                        }
                        if len(stack) > 0 && stack[len(stack)-1] < nums[j] {
                                return true
                        }
                        stack = append(stack, nums[j])
                }
        }
        return false
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func find132pattern(nums []int) bool {
        if len(nums) < 3 {
                return false
        }
        minArr := make([]int, len(nums)) // minArr[i]是[0,i]中的最小值
        minArr[0] = nums[0]
        for i := 1; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                minArr[i] = min(nums[i], minArr[i-1])
        }
        for j := len(nums) - 2; j >= 0; j-- {
                if nums[j] != minArr[j] {
                        for k := j + 1; k < len(nums); k++ {
                                if nums[k] > minArr[j] && nums[k] < nums[j] {
                                        return true
                                }
                        }
                }
        }
        return false
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func find132pattern(nums []int) bool {
        if len(nums) < 3 {
                return false
        }
        stack := make([]int, 0)
        maxValue := math.MinInt32
        for i := len(nums) - 1; i >= 0; i-- {
                // i < k
                if nums[i] < maxValue {
                        return true
                }
                for len(stack) > 0 && nums[i] > stack[len(stack)-1] {
                        last := len(stack) - 1
                        maxValue = max(maxValue, stack[last])
                        stack = stack[:last]
                }
                stack = append(stack, nums[i])
        }
        return false
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

14.25 457. 环形数组循环 (3)

- 题目

```
给定一个含有正整数和负整数的环形数组 nums。 如果某个索引中的数 k 为正数，则向前移动 k␣
↪个索引。
相反，如果是负数 (-k)，则向后移动 k 个索引。
因为数组是环形的，所以可以假设最后一个元素的下一个元素是第一个元素，而第一个元素的前一个元素是最后一
确定 nums 中是否存在循环（或周期）。循环必须在相同的索引处开始和结束并且循环长度 > 1。
此外，一个循环中的所有运动都必须沿着同一方向进行。换句话说，一个循环中不能同时包括向前的运动和向后的
示例 1：输入：[2,-1,1,2,2] 输出：true
解释：存在循环，按索引 0 -> 2 -> 3 -> 0 。循环长度为 3 。
示例 2：输入：[-1,2] 输出：false
解释：按索引 1 -> 1 -> 1 ... 的运动无法构成循环，因为循环的长度为 1␣
↪。根据定义，循环的长度必须大于 1 。
示例 3:输入：[-2,1,-1,-2,-2] 输出：false
解释：按索引 1 -> 2 -> 1 -> ... 的运动无法构成循环，因为按索引 1 -> 2␣
↪的运动是向前的运动，
而按索引 2 -> 1 的运动是向后的运动。一个循环中的所有运动都必须沿着同一方向进行。
提示：-1000 ≤ nums[i] ≤ 1000
    nums[i] ≠ 0
    0 ≤ nums.length ≤ 5000
进阶：你能写出时间时间复杂度为 O(n) 和额外空间复杂度为 O(1) 的算法吗？
```

- 解题思路

```
func circularArrayLoop(nums []int) bool {
        n := len(nums)
        for i := 0; i < n; i++ {
                if nums[i] == 0 { // 原数组没有0，被标记为0，跳过
                        continue
                }
                slow, fast := i, getNext(nums, n, i)
                // 保证是全正或者全负
                for nums[slow]*nums[fast] > 0 && nums[slow]*nums[getNext(nums, n,␣
```

(续下页)

(接上页)

```
↪fast)] > 0 {
                        if slow == fast {
                                if slow == getNext(nums, n, slow) { //␣
↪等于本身，退出继续寻找
                                        break
                                }
                                return true
                        }
                        slow = getNext(nums, n, slow)
                        fast = getNext(nums, n, getNext(nums, n, fast))
                }
                temp := i
                for nums[temp]*nums[getNext(nums, n, temp)] > 0 {
                        nums[temp] = 0 // 标记为0
                        temp = getNext(nums, n, temp)
                }
        }
        return false
}

func getNext(nums []int, n, cur int) int {
        return ((cur+nums[cur])%n + n) % n
}

# 2
func circularArrayLoop(nums []int) bool {
        n := len(nums)
        for i := 0; i < n; i++ {
                slow, fast := i, getNext(nums, n, i)
                // 保证是同方向
                for nums[slow]*nums[fast] > 0 && nums[slow]*nums[getNext(nums, n,␣
↪fast)] > 0 {
                        if slow == fast {
                                if slow == getNext(nums, n, slow) { //␣
↪等于本身，退出继续寻找
                                        break
                                }
                                return true
                        }
                        slow = getNext(nums, n, slow)
                        fast = getNext(nums, n, getNext(nums, n, fast))
                }
        }
```

(续下页)

(接上页)

```
        return false
}

func getNext(nums []int, n, cur int) int {
        return ((cur+nums[cur])%n + n) % n
}

# 3
func circularArrayLoop(nums []int) bool {
        n := len(nums)
        for i := 0; i < n; i++ {
                if judge(nums, n, i) == true {
                        return true
                }
        }
        return false
}

func judge(nums []int, n, cur int) bool {
        start := cur
        dir := 1
        if nums[cur] < 0 {
                dir = -1
        }
        count := 1
        for {
                if count > n {
                        return false
                }
                next := ((cur+nums[cur])%n + n) % n
                if (dir > 0 && nums[next] < 0) || (dir < 0 && nums[next] > 0) {
                        return false
                }
                if next == start { // 走到起点
                        return count > 1
                }
                count++
                cur = next
        }
}
```

14.26 462. 最少移动次数使数组元素相等 II(2)

- 题目

```
给定一个非空整数数组，找到使所有数组元素相等所需的最小移动数，其中每次移动可将选定的一个元素加1或减1
您可以假设数组的长度最多为10000。
例如:输入: [1,2,3]输出: 2
说明: 只有两个动作是必要的 (记得每一步仅可使其中一个元素加1或减1) :
[1,2,3]  =>  [2,2,3]  =>  [2,2,2]
```

- 解题思路

```
func minMoves2(nums []int) int {
        sort.Ints(nums)
        target := nums[len(nums)/2]
        res := 0
        for i := 0; i < len(nums); i++ {
                res = res + abs(target-nums[i])
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func minMoves2(nums []int) int {
        sort.Ints(nums)
        res := 0
        left, right := 0, len(nums)-1
        for left < right {
                // 无论选哪个数作为最终值，前后第n个值需要移动之和不变
                // nums[right]-target+target-nums[left] = nums[right] - nums[left]
                res = res + nums[right] - nums[left]
                left++
                right--
        }
        return res
}
```

14.27 464. 我能赢吗 (3)

- 题目

```
在 "100 game" 这个游戏中，两名玩家轮流选择从 1 到 10 的任意整数，累计整数和，
先使得累计整数和达到或超过 100 的玩家，即为胜者。
如果我们将游戏规则改为 “玩家不能重复使用整数” 呢？
例如，两个玩家可以轮流从公共整数池中抽取从 1 到 15 的整数（不放回），直到累计整数和 >
↪= 100。
给定一个整数maxChoosableInteger（整数池中可选择的最大数）和另一个整数desiredTotal（累计和），
判断先出手的玩家是否能稳赢（假设两位玩家游戏时都表现最佳）？
你可以假设maxChoosableInteger不会大于 20，desiredTotal不会大于 300。
示例：输入：maxChoosableInteger = 10 desiredTotal = 11输出： false
解释：无论第一个玩家选择哪个整数，他都会失败。
第一个玩家可以选择从 1 到 10 的整数。
如果第一个玩家选择 1，那么第二个玩家只能选择从 2 到 10 的整数。
第二个玩家可以通过选择整数 10（那么累积和为 11 >= desiredTotal），从而取得胜利.
同样地，第一个玩家选择任意其他整数，第二个玩家都会赢。
```

- 解题思路

```
var m []bool

func canIWin(maxChoosableInteger int, desiredTotal int) bool {
        a := maxChoosableInteger
        if a*(a+1)/2 < desiredTotal {
                return false
        }
        m = make([]bool, 1<<a)
        return dfs(a, desiredTotal, 0)
}

func dfs(a, b int, status int) bool {
        if m[status] == true {
                return true
        }
        for i := 1; i <= a; i++ {
                cur := 1 << (i - 1)
                if cur&status > 0 { // 当前位(i-1)为1
                        continue
                }
                if b <= i { // 当前选的数可以赢
                        m[status] = true
                        return true
```

(续下页)

(接上页)

```
                }
                next := status | cur            // 按位或运算：status第(i-1)变为1
                if dfs(a, b-i, next) == false { // 如果下个人要输的话，当前人要赢
                        m[status] = true
                        return true
                }
        }
        m[status] = false
        return false
}

# 2
var m map[int]bool

func canIWin(maxChoosableInteger int, desiredTotal int) bool {
        a := maxChoosableInteger
        if a*(a+1)/2 < desiredTotal {
                return false
        }
        m = make(map[int]bool)
        return dfs(a, desiredTotal, 0)
}

func dfs(a, b int, status int) bool {
        if v, ok := m[status]; ok {
                return v
        }
        for i := 1; i <= a; i++ {
                cur := 1 << (i - 1)
                next := status | cur                                          //␣
↪按位或运算：status第(i-1)变为1
                if cur&status == 0 && (b <= i || dfs(a, b-i, next) == false) { //␣
↪当前位(i-1)为1
                        m[status] = true
                        return true
                }
        }
        m[status] = false
        return false
}

# 3
func canIWin(maxChoosableInteger int, desiredTotal int) bool {
```

(续下页)

(接上页)

```
        a, b := maxChoosableInteger, desiredTotal
        if a*(a+1)/2 < b {
                return false
        }
        total := 1 << a
        dp := make([]bool, total)
        for i := total - 1; i >= 0; i-- { // 枚举状态
                sum := 0
                for k := 0; k < a; k++ { // 状态和
                        if i&(1<<k) > 0 { // i: 对应状态为1位置上和
                                sum = sum + (k + 1)
                        }
                }
                for k := 0; k < a; k++ {
                        if i&(1<<k) > 0 {
                                continue
                        }
                        prev := i | (1 << k)
                        // >=剩下值，或者之前为false
                        if k+1 >= desiredTotal-sum || dp[prev] == false {
                                dp[i] = true
                        }
                }
        }
        return dp[0]
}
```

14.28 467. 环绕字符串中唯一的子字符串 (1)

- 题目

```
把字符串 s 看作是“abcdefghijklmnopqrstuvwxyz”的无限环绕字符串，
所以s 看起来是这样的："...zabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcd...
↪.".
现在我们有了另一个字符串 p 。
你需要的是找出 s 中有多少个唯一的 p 的非空子串，
尤其是当你的输入是字符串 p ，你需要输出字符串s 中 p 的不同的非空子串的数目。
注意：p仅由小写的英文字母组成，p 的大小可能超过 10000。
示例1:输入："a" 输出：1
解释：字符串 S 中只有一个"a"子字符。
示例 2:输入："cac" 输出：2
解释：字符串 S 中的字符串“cac”只有两个子串“a”、“c”。.
```

(续下页)

(接上页)

```
示例 3:输入: "zab" 输出: 6
解释: 在字符串 S 中有六个子串“z”、“a”、“b”、“za”、“ab”、“zab”。.
```

- 解题思路

```
func findSubstringInWraproundString(p string) int {
        dp := [26]int{}
        count := 0
        for i := 0; i < len(p); i++ {
                value := int(p[i] - 'a')
                if i > 0 && (value-int(p[i-1]-'a')-1)%26 == 0 {
                        count++
                } else {
                        count = 1
                }
                dp[value] = max(dp[value], count)
        }
        res := 0
        for i := 0; i < len(dp); i++ {
                res = res + dp[i]
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

14.29 468. 验证 IP 地址 (1)

- 题目

```
编写一个函数来验证输入的字符串是否是有效的 IPv4 或IPv6 地址。
如果是有效的 IPv4 地址，返回 "IPv4" ；
如果是有效的 IPv6 地址，返回 "IPv6" ；
如果不是上述类型的 IP 地址，返回 "Neither" 。
IPv4地址由十进制数和点来表示，每个地址包含 4 个十进制数，其范围为0 -255，用(".")分割。
比如，172.16.254.1；
同时，IPv4 地址内的数不会以 0 开头。比如，地址172.16.254.01 是不合法的。
```

(续下页)

(接上页)

```
IPv6地址由 8 组 16 进制的数字来表示，每组表示16 比特。这些组数字通过 (":")分割。
比如,2001:0db8:85a3:0000:0000:8a2e:0370:7334 是一个有效的地址。
而且，我们可以加入一些以 0 开头的数字，字母可以使用大写，也可以是小写。
所以，2001:db8:85a3:0:0:8A2E:0370:7334 也是一个有效的 IPv6 address地址
(即，忽略 0 开头，忽略大小写)。
然而，我们不能因为某个组的值为 0，而使用一个空的组，以至于出现 (::) 的情况。
比如，2001:0db8:85a3::8A2E:0370:7334 是无效的 IPv6 地址。
同时，在 IPv6 地址中，多余的 0 也是不被允许的。
比如，02001:0db8:85a3:0000:0000:8a2e:0370:7334 是无效的。
示例 1: 输入: IP = "172.16.254.1" 输出: "IPv4"
解释: 有效的 IPv4 地址，返回 "IPv4"
示例 2: 输入: IP = "2001:0db8:85a3:0:0:8A2E:0370:7334" 输出: "IPv6"
解释: 有效的 IPv6 地址，返回 "IPv6"
示例 3: 输入: IP = "256.256.256.256" 输出: "Neither"
解释: 既不是 IPv4 地址，又不是 IPv6 地址
示例 4: 输入: IP = "2001:0db8:85a3:0:0:8A2E:0370:7334:" 输出: "Neither"
示例 5: 输入: IP = "1e1.4.5.6"输出: "Neither"
提示: IP 仅由英文字母，数字，字符 '.' 和 ':' 组成。
```

- 解题思路

```
var defaultRes string = "Neither"

func validIPAddress(IP string) string {
        if strings.Contains(IP, ":") {
                return checkIPv6(IP)
        }
        return checkIPv4(IP)
}

func checkIPv4(ip string) string {
        arr := strings.Split(ip, ".")
        if len(arr) != 4 {
                return defaultRes
        }
        for i := 0; i < len(arr); i++ {
                num, err := strconv.Atoi(arr[i])
                if err != nil || num > 255 || num < 0 {
                        return defaultRes
                }
                if len(arr[i]) > 1 && arr[i][0] == '0' || (i == 0 && num == 0) {
                        return defaultRes
                }
        }
```

(续下页)

(接上页)

```
        return "IPv4"
}

func checkIPv6(ip string) string {
        arr := strings.Split(ip, ":")
        if len(arr) != 8 {
                return defaultRes
        }
        for i := 0; i < len(arr); i++ {
                if arr[i] == "" || len(arr[i]) > 4 {
                        return defaultRes
                }
                for j := 0; j < len(arr[i]); j++ {
                        if (arr[i][j] > 'F' && arr[i][j] < 'a') ||
                                (arr[i][j] > 'f' && arr[i][j] <= 'z') {
                                return defaultRes
                        }
                }
        }
        return "IPv6"
}
```

14.30 470. 用 Rand7() 实现 Rand10()(2)

- 题目

```
已有方法rand7可生成 1 到 7 范围内的均匀随机整数，
试写一个方法rand10生成 1 到 10 范围内的均匀随机整数。
不要使用系统的Math.random()方法。
示例 1:输入: 1 输出: [7]
示例 2:输入: 2 输出: [8,4]
示例 3:输入: 3 输出: [8,1,10]
提示:rand7已定义。
传入参数:n表示rand10的调用次数。
进阶:rand7()调用次数的期望值是多少?
你能否尽量少调用 rand7() ?
```

- 解题思路

```
func rand10() int {
        for {
                a := rand7()
```

(续下页)

(接上页)

```
                b := rand7()
                target := a + (b-1)*7
                if target <= 40 {
                        return target%10 + 1
                }
        }
}

# 2
func rand10() int {
        for {
                a := rand7()
                b := rand7()
                target := a + (b-1)*7
                if target <= 40 { // 1-49
                        return target%10 + 1
                }
                a = rand7()
                b = target - 40
                target = a + (b-1)*7 // 1-63
                if target <= 60 {
                        return target%10 + 1
                }
                a = rand7()
                b = target - 60
                target = a + (b-1)*7 // 1-21
                if target <= 20 {
                        return target%10 + 1
                }
        }
}
```

14.31 473. 火柴拼正方形 (2)

- 题目

```
还记得童话《卖火柴的小女孩》吗？现在，你知道小女孩有多少根火柴，
请找出一种能使用所有火柴拼成一个正方形的方法。不能折断火柴，可以把火柴连接起来，并且每根火柴都要用到
输入为小女孩拥有火柴的数目，每根火柴用其长度表示。输出即为是否能用所有的火柴拼成正方形。
示例1:输入: [1,1,2,2,2] 输出: true
解释: 能拼成一个边长为2的正方形，每边两根火柴。
示例2:输入: [3,3,3,3,4] 输出: false
```

(续下页)

(接上页)

```
解释: 不能用所有火柴拼成一个正方形。
注意:给定的火柴长度和在0到10^9之间。
火柴数组的长度不超过15。
```

- 解题思路

```
func makesquare(nums []int) bool {
        n := len(nums)
        if nums == nil || n < 4 {
                return false
        }
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum%4 != 0 {
                return false
        }
        // 从大到小排序; 从小到大可能会超时
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] > nums[j]
        })
        res := make([]int, 4)
        return dfs(nums, res, sum/4, 0)
}

func dfs(nums []int, res []int, target int, level int) bool {
        if len(nums) == level {
                for i := 0; i < len(res); i++ {
                        if res[i] != target {
                                return false
                        }
                }
                return true
        }
        for i := 0; i < len(res); i++ {
                if nums[level]+res[i] > target {
                        continue
                }
                res[i] = res[i] + nums[level]
                if dfs(nums, res, target, level+1) == true {
                        return true
                }
                res[i] = res[i] - nums[level]
```

(续下页)

(接上页)

```
        }
        return false
}

# 2
func makesquare(nums []int) bool {
        n := len(nums)
        if nums == nil || n < 4 {
                return false
        }
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum%4 != 0 {
                return false
        }
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] > nums[j]
        })
        visited := make([]bool, len(nums))
        for i := 0; i < 4; i++ {
                if dfs(nums, sum/4, 0, 0, visited) == false {
                        return false
                }
        }
        return true
}

func dfs(nums []int, target int, sum int, level int, visited []bool) bool {
        if sum == target {
                return true
        }
        if len(nums) == level {

        }
        for i := level; i < len(nums); i++ {
                if visited[i] == false && sum <= target {
                        visited[i] = true
                        if dfs(nums, target, sum+nums[i], level+1, visited) {
                                return true
                        }
                        visited[i] = false
```

(续下页)

(接上页)

```
                }
        }
        return false
}
```

14.32 474. 一和零 (2)

- 题目

```
给你一个二进制字符串数组 strs 和两个整数 m 和 n 。
请你找出并返回 strs 的最大子集的大小，该子集中 最多 有 m 个 0 和 n 个 1 。
如果 x 的所有元素也是 y 的元素，集合 x 是集合 y 的 子集 。
示例 1：输入：strs = ["10", "0001", "111001", "1", "0"], m = 5, n = 3 输出：4
解释：最多有 5 个 0 和 3 个 1 的最大子集是 {"10","0001","1","0"} ，因此答案是 4 。
其他满足题意但较小的子集包括 {"0001","1"} 和 {"10","1","0"} 。
{"111001"} 不满足题意，因为它含 4 个 1 ，大于 n 的值 3 。
示例 2：输入：strs = ["10", "0", "1"], m = 1, n = 1 输出：2
解释：最大的子集是 {"0", "1"} ，所以答案是 2 。
提示：1 <= strs.length <= 600
1 <= strs[i].length <= 100
strs[i]仅由'0' 和'1' 组成
1 <= m, n <= 100
```

- 解题思路

```
func findMaxForm(strs []string, m int, n int) int {
        dp := make([][]int, m+1)
        for i := 0; i <= m; i++ {
                dp[i] = make([]int, n+1)
        }
        for i := 0; i < len(strs); i++ {
                a, b := getCount(strs[i])
                for j := m; j >= a; j-- {
                        for k := n; k >= b; k-- {
                                dp[j][k] = max(dp[j][k], dp[j-a][k-b]+1)
                        }
                }
        }
        return dp[m][n]
}

func getCount(str string) (a, b int) {
```

(续下页)

(接上页)

```
        a, b = 0, 0
        for i := 0; i < len(str); i++ {
                if str[i] == '0' {
                        a++
                } else {
                        b++
                }
        }
        return a, b
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func findMaxForm(strs []string, m int, n int) int {
        dp := make([][][]int, len(strs)+1)
        for i := 0; i <= len(strs); i++ {
                dp[i] = make([][]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = make([]int, n+1)
                }
        }
        for i := 1; i <= len(strs); i++ {
                a, b := getCount(strs[i-1])
                for j := 0; j <= m; j++ {
                        for k := 0; k <= n; k++ {
                                dp[i][j][k] = dp[i-1][j][k]
                                if a <= j && b <= k {
                                        dp[i][j][k] = max(dp[i-1][j][k], dp[i-1][j-
↪a][k-b]+1)
                                }
                        }
                }
        }
        return dp[len(strs)][m][n]
}

func getCount(str string) (a, b int) {
```

(续下页)

(接上页)

```
        a, b = 0, 0
        for i := 0; i < len(str); i++ {
                if str[i] == '0' {
                        a++
                } else {
                        b++
                }
        }
        return a, b
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

14.33 477. 汉明距离总和 (1)

- 题目

```
两个整数的汉明距离 指的是这两个数字的二进制数对应位不同的数量。
计算一个数组中，任意两个数之间汉明距离的总和。
示例:输入: 4, 14, 2 输出: 6
解释: 在二进制表示中，4表示为0100，14表示为1110，2表示为0010。
（这样表示是为了体现后四位之间关系）
所以答案为:
HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 =␣
↪6.
注意:数组中元素的范围为从0到10^9。
数组的长度不超过10^4。
```

- 解题思路

```
func totalHammingDistance(nums []int) int {
        res := 0
        for i := 0; i < 32; i++ {
                total := 0
                for j := 0; j < len(nums); j++ {
                        total = total + (nums[j]>>i)&1
                }
```

(续下页)

(接上页)

```
                // 汉明距离：两个数字的二进制数对应位不同的数量
                // 该位1的数量*该位0的数量
                res = res + total*(len(nums)-total)
        }
        return res
}
```

14.34 478. 在圆内随机生成点 (1)

- 题目

```
给定圆的半径和圆心的 x、y 坐标，写一个在圆中产生均匀随机点的函数randPoint。
说明:输入值和输出值都将是浮点数。
圆的半径和圆心的 x、y 坐标将作为参数传递给类的构造函数。
圆周上的点也认为是在圆中。
randPoint返回一个包含随机点的x坐标和y坐标的大小为2的数组。
示例 1：输入：["Solution","randPoint","randPoint","randPoint"] [[1,0,0],[],[],[]]
输出：[null,[-0.72939,-0.65505],[-0.78502,-0.28626],[-0.83119,-0.19803]]
示例 2：输入：["Solution","randPoint","randPoint","randPoint"] [[10,5,-7.5],[],[],[]]
输出：[null,[11.52438,-8.33273],[2.46992,-16.21705],[11.13430,-12.42337]]
输入语法说明：输入是两个列表：调用成员函数名和调用的参数。Solution的构造函数有三个参数，
圆的半径、圆心的 x 坐标、圆心的 y 坐标。randPoint没有参数。
输入参数是一个列表，即使参数为空，也会输入一个 [] 空列表。
```

- 解题思路

```
type Solution struct {
        x      float64
        y      float64
        radius float64
}

func Constructor(radius float64, x_center float64, y_center float64) Solution {
        return Solution{
                x:      x_center,
                y:      y_center,
                radius: radius,
        }
}

func (this *Solution) RandPoint() []float64 {
        for {
```

(续下页)

(接上页)

```
                a := this.x - this.radius + 2*this.radius*rand.Float64()
                b := this.y - this.radius + 2*this.radius*rand.Float64()
                if (a-this.x)*(a-this.x)+(b-this.y)*(b-this.y) < this.radius*this.
↪radius {
                        return []float64{a, b}
                }
        }
}
```

14.35 481. 神奇字符串 (2)

- 题目

```
神奇的字符串S只包含 '1' 和 '2', 并遵守以下规则:
字符串 S 是神奇的, 因为串联字符 '1' 和 '2' 的连续出现次数会生成字符串 S 本身。
字符串S的前几个元素如下: S = “1221121221221121122 ......”
如果我们将S 中连续的 1 和 2 进行分组, 它将变成:
1 22 11 2 1 22 1 22 11 2 11 22 ......
并且每个组中 '1' 或 '2' 的出现次数分别是:
1 2 2 1 1 2 1 2 2 1 2 2 ......
你可以看到上面的出现次数就是 S 本身。
给定一个整数 N 作为输入, 返回神奇字符串 S中前 N 个数字中的 '1' 的数目。
注意: N 不会超过 100,000。
示例: 输入: 6 输出: 3
解释: 神奇字符串 S 的前 6 个元素是 “12211”, 它包含三个 1, 因此返回 3。
```

- 解题思路

```
func magicalString(n int) int {
        if n == 0 {
                return 0
        }
        if n <= 3 {
                return 1
        }
        str := []byte("122")
        res := 1
        index := 2
        for i := 2; i < n; i++ {
                if str[i] == '2' {
                        if str[index] == '2' {
                                str = append(str, []byte{'1', '1'}...)
```

(续下页)

(接上页)

```
                        } else {
                                str = append(str, []byte{'2', '2'}...)
                        }
                        index = index + 2
                } else {
                        res++
                        if str[index] == '2' {
                                str = append(str, '1')
                        } else {
                                str = append(str, '2')
                        }
                        index = index + 1
                }
        }
        return res
}

# 2
func magicalString(n int) int {
        if n == 0 {
                return 0
        }
        if n <= 3 {
                return 1
        }
        str := []byte("122")
        flag := true
        for i := 2; i < n; i++ {
                count := str[i] - '0'
                if flag == true {
                        for count > 0 {
                                str = append(str, '1')
                                count--
                        }
                        flag = false
                } else {
                        for count > 0 {
                                str = append(str, '2')
                                count--
                        }
                        flag = true
                }
        }
```

(续下页)

(接上页)

```
        res := 0
        for i := 0; i < n; i++ {
                if str[i] == '1' {
                        res++
                }
        }
        return res
}
```

14.36 486. 预测赢家 (3)

- 题目

```
给定一个表示分数的非负整数数组。 玩家 1 从数组任意一端拿取一个分数，
随后玩家 2 继续从剩余数组任意一端拿取分数，然后玩家 1 拿，…… 。
每次一个玩家只能拿取一个分数，分数被拿取之后不再可取。直到没有剩余分数可取时游戏结束。
最终获得分数总和最多的玩家获胜。
给定一个表示分数的数组，预测玩家1是否会成为赢家。你可以假设每个玩家的玩法都会使他的分数最大化。
示例 1: 输入: [1, 5, 2] 输出: False
解释: 一开始，玩家1可以从1和2中进行选择。
如果他选择 2（或者 1），那么玩家 2 可以从 1（或者 2 ）和 5 中进行选择。
如果玩家 2 选择了 5 ，那么玩家 1 则只剩下 1（或者 2 ）可选。
所以，玩家 1 的最终分数为 1 + 2 = 3，而玩家 2 为 5 。
因此，玩家 1 永远不会成为赢家，返回 False 。
示例 2: 输入: [1, 5, 233, 7] 输出: True
解释: 玩家 1 一开始选择 1 。然后玩家 2 必须从 5 和 7 中进行选择。
无论玩家 2 选择了哪个，玩家 1 都可以选择 233 。
     最终，玩家 1（234 分）比玩家 2（12 分）获得更多的分数，所以返回 True，表示玩家 1␣
↪可以成为赢家。
提示: 1 <= 给定的数组长度 <= 20.
    数组里所有分数都为非负数且不会大于 10000000 。
    如果最终两个玩家的分数相等，那么玩家 1 仍为赢家。
```

- 解题思路

```
func PredictTheWinner(nums []int) bool {
        return dfs(nums, 0, len(nums)-1) >= 0
}

func dfs(nums []int, start, end int) int {
        if start > end {
                return 0
```

(续下页)

(接上页)

```
        }
        // 玩家得分：自己得分-对手得分
        left := nums[start] - dfs(nums, start+1, end)
        right := nums[end] - dfs(nums, start, end-1)
        return max(left, right)

}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func PredictTheWinner(nums []int) bool {
        dp := make([]int, len(nums))
        for i := 0; i < len(nums); i++ {
                dp[i] = nums[i]
        }
        for i := len(nums) - 2; i >= 0; i-- {
                for j := i + 1; j < len(nums); j++ {
                        dp[j] = max(nums[i]-dp[j], nums[j]-dp[j-1])
                }
        }
        return dp[len(nums)-1] >= 0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func PredictTheWinner(nums []int) bool {
        n := len(nums)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                dp[i][i] = nums[i]
```

(续下页)

(接上页)

```
        }
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        // 玩家得分: 自己得分-对手得分
                        dp[i][j] = max(nums[i]-dp[i+1][j], nums[j]-dp[i][j-1])
                }
        }
        return dp[0][n-1] >= 0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

14.37 491. 递增子序列 (2)

- 题目

```
给定一个整型数组, 你的任务是找到所有该数组的递增子序列, 递增子序列的长度至少是2。
示例:输入: [4, 6, 7, 7]
输出: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]]
说明:给定数组的长度不会超过15。
    数组中的整数范围是 [-100,100]。
    给定数组中可能包含重复数字, 相等的数字应该被视为递增的一种情况。
```

- 解题思路

```
var res [][]int

func findSubsequences(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, 0, math.MinInt32, make([]int, 0))
        return res
}

func dfs(nums []int, index int, prev int, arr []int) {
        if index == len(nums) {
                if len(arr) >= 2 {
                        temp := make([]int, len(arr))
```

(续下页)

(接上页)

```
                        copy(temp, arr)
                        res = append(res, temp)
                }
                return
        }
        if prev <= nums[index] {
                arr = append(arr, nums[index])
                dfs(nums, index+1, nums[index], arr)
                arr = arr[:len(arr)-1]
        }
        if prev != nums[index] {
                dfs(nums, index+1, prev, arr)
        }
}

# 2
var res [][]int

func findSubsequences(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, 0, make([]int, 0))
        return res
}

func dfs(nums []int, index int, arr []int) {
        if len(arr) >= 2 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
        }
        m := make(map[int]bool)
        for i := index; i < len(nums); i++ {
                if m[nums[i]] == true || (len(arr) > 0 && nums[i] < arr[len(arr)-1]) {
                        continue
                }
                m[nums[i]] = true
                dfs(nums, i+1, append(arr, nums[i]))
        }
}
```

14.38 494. 目标和 (5)

- 题目

```
给定一个非负整数数组，a1, a2, ..., an, 和一个目标数，S。现在你有两个符号 + 和 -。
对于数组中的任意一个整数，你都可以从 + 或 -中选择一个符号添加在前面。
返回可以使最终数组和为目标数 S 的所有添加符号的方法数。
示例：输入：nums: [1, 1, 1, 1, 1], S: 3 输出：5
解释：
-1+1+1+1+1 = 3
+1-1+1+1+1 = 3
+1+1-1+1+1 = 3
+1+1+1-1+1 = 3
+1+1+1+1-1 = 3
一共有5种方法让最终目标和为3。
提示：数组非空，且长度不会超过 20 。
    初始的数组的和不会超过 1000 。
    保证返回的最终结果能被 32 位整数存下。
```

- 解题思路

```
func findTargetSumWays(nums []int, S int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
                if nums[0] == 0 && S == 0 {
                        return 2
                }
                if nums[0] == S || nums[0] == -S {
                        return 1
                }
        }
        value := nums[0]
        nums = nums[1:]
        return findTargetSumWays(nums, S-value) + findTargetSumWays(nums, S+value)
}

# 2
func findTargetSumWays(nums []int, S int) int {
        dp := make(map[int]int)
        dp[nums[0]]++
        dp[-nums[0]]++
        for i := 1; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                temp := make(map[int]int)
                for k, v := range dp {
                        temp[k-nums[i]] = temp[k-nums[i]] + v
                        temp[k+nums[i]] = temp[k+nums[i]] + v
                }
                dp = temp
        }
        return dp[S]
}

# 3
var res int

func findTargetSumWays(nums []int, S int) int {
        res = 0
        dfs(nums, 0, S)
        return res
}

func dfs(nums []int, index int, target int) {
        if index == len(nums) {
                if target == 0 {
                        res++
                }
                return
        }
        dfs(nums, index+1, target+nums[index])
        dfs(nums, index+1, target-nums[index])
}


# 4
func findTargetSumWays(nums []int, S int) int {
        sum := 0
        // 非负整数数组
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum < int(math.Abs(float64(S))) || (sum+S)%2 == 1 {
                return 0
        }
        // 一个正数x,一个负数背包y => x+y=sum, x-y=S => (sum+S)/2=x
        target := (sum + S) / 2
```

(续下页)

(接上页)

```
        dp := make([]int, target+1)
        dp[0] = 1
        for i := 1; i <= len(nums); i++ {
        // 从后往前，避免覆盖
                for j := target; j >= 0; j-- {
                        if j >= nums[i-1] {
                                // 背包足够大，都选
                                dp[j] = dp[j] + dp[j-nums[i-1]]
                        } else {
                                // 容量不够，不选
                                dp[j] = dp[j]
                        }
                }
        }
        return dp[target]
}

# 5
func findTargetSumWays(nums []int, S int) int {
        sum := 0
        // 非负整数数组
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum < int(math.Abs(float64(S))) || (sum+S)%2 == 1 {
                return 0
        }
        // 一个正数x,一个负数背包y => x+y=sum, x-y=S => (sum+S)/2=x
        target := (sum + S) / 2
        // 在前i个物品中选择，若当前背包的容量为j，则最多有x种方法可以恰好装满背包。
        dp := make([][]int, len(nums)+1)
        for i := 0; i <= len(nums); i++ {
                dp[i] = make([]int, target+1)
                dp[i][0] = 1 // 容量为0，  只有都不选
        }
        for i := 1; i <= len(nums); i++ {
                for j := 0; j <= target; j++ {
                        if j >= nums[i-1] {
                                // 背包足够大，都选
                                dp[i][j] = dp[i-1][j] + dp[i-1][j-nums[i-1]]
                        } else {
                                // 容量不够，不选
                                dp[i][j] = dp[i-1][j]
```

(续下页)

(接上页)

```
                        }
                }
        }
        return dp[len(nums)][target]
}
```

14.39 495. 提莫攻击 (1)

- 题目

```
在《英雄联盟》的世界中，有一个叫 “提莫” 的英雄，
他的攻击可以让敌方英雄艾希（编者注：寒冰射手）进入中毒状态。
现在，给出提莫对艾希的攻击时间序列和提莫攻击的中毒持续时间，你需要输出艾希的中毒状态总时长。
你可以认为提莫在给定的时间点进行攻击，并立即使艾希处于中毒状态。
示例1:输入: [1,4], 2 输出: 4
原因: 第 1 秒初，提莫开始对艾希进行攻击并使其立即中毒。中毒状态会维持 2 秒钟，直到第␣
↪2 秒末结束。
第 4 秒初，提莫再次攻击艾希，使得艾希获得另外 2 秒中毒时间。 所以最终输出 4 秒。
示例2:输入: [1,2], 2 输出: 3
原因: 第 1 秒初，提莫开始对艾希进行攻击并使其立即中毒。中毒状态会维持 2 秒钟，直到第␣
↪2 秒末结束。
但是第 2 秒初，提莫再次攻击了已经处于中毒状态的艾希。
由于中毒状态不可叠加，提莫在第 2 秒初的这次攻击会在第 3 秒末结束。 所以最终输出 3 。
提示: 你可以假定时间序列数组的总长度不超过 10000。
   ␣
↪你可以假定提莫攻击时间序列中的数字和提莫攻击的中毒持续时间都是非负整数，并且不超过␣
↪10,000,000。
```

- 解题思路

```
func findPoisonedDuration(timeSeries []int, duration int) int {
        res := 0
        if len(timeSeries) == 0 {
                return 0
        }
        for i := 0; i < len(timeSeries)-1; i++ {
                res = res + min(timeSeries[i+1]-timeSeries[i], duration)
        }
        return res + duration
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}
```

14.40 497. 非重叠矩形中的随机点 (1)

- 题目

```
给定一个非重叠轴对齐矩形的列表 rects，写一个函数 pick␣
↪随机均匀地选取矩形覆盖的空间中的整数点。
提示：整数点是具有整数坐标的点。
矩形周边上的点包含在矩形覆盖的空间中。
第 i 个矩形 rects [i] = [x1, y1, x2, y2]，其中[x1, y1] 是左下角的整数坐标，[x2, y2]␣
↪是右上角的整数坐标。
每个矩形的长度和宽度不超过 2000。
1 <= rects.length<= 100
pick 以整数坐标数组[p_x, p_y]的形式返回一个点。
pick 最多被调用10000次。
示例 1：输入： ["Solution","pick","pick","pick"] [[[[1,1,5,5]]],[],[],[]] 输出：[null,
↪[4,1],[4,1],[3,3]]
示例 2：输入： ["Solution","pick","pick","pick","pick","pick"] [[[[-2,-2,-1,-1],[1,0,
↪3,0]]],[],[],[],[],[]]
输出：[null,[-1,-2],[2,0],[-2,-1],[3,0],[-2,-2]]
输入语法的说明：
输入是两个列表：调用的子例程及其参数。
Solution 的构造函数有一个参数，即矩形数组 rects。pick␣
↪没有参数。参数总是用列表包装的，即使没有也是如此。
```

- 解题思路

```
type Solution struct {
        nums  []int   // 前缀和
        total int     // 总和
        rects [][]int // 原数组
}

func Constructor(rects [][]int) Solution {
        arr := make([]int, 0)
        total := 0
        for i := 0; i < len(rects); i++ {
```

(续下页)

(接上页)

```
                x1, y1, x2, y2 := rects[i][0], rects[i][1], rects[i][2], rects[i][3]
                total = total + (x2-x1+1)*(y2-y1+1) // x点的个数 *␣
↪y点的个数: 注意包含边+1
                arr = append(arr, total)
        }
        return Solution{nums: arr, total: total, rects: rects}
}

// leetcode528.按权重随机选择
func (this *Solution) Pick() []int {
        target := rand.Intn(this.total) // 目标值
        left, right := 0, len(this.nums)
        for left < right {
                mid := left + (right-left)/2
                if this.nums[mid] <= target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        temp := this.rects[left]
        x1, y1, x2, y2 := temp[0], temp[1], temp[2], temp[3]
        width := x2 - x1 + 1
        height := y2 - y1 + 1
        start := this.nums[left] - width*height // 前缀和-目标值
        return []int{x1 + (target-start)%width, y1 + (target-start)/width}
}
```

14.41 498. 对角线遍历 (2)

- 题目

```
给定一个含有 M x N 个元素的矩阵（M 行，N␣
↪列），请以对角线遍历的顺序返回这个矩阵中的所有元素，
对角线遍历如下图所示。
示例:
输入:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
```

(续下页)

(接上页)

```
输出：  [1,2,4,7,5,3,6,8,9]
说明:给定矩阵中的元素总数不会超过 100000 。
```

- 解题思路

```
func findDiagonalOrder(matrix [][]int) []int {
        res := make([]int, 0)
        if len(matrix) == 0 {
                return res
        }
        n, m := len(matrix), len(matrix[0])
        if n == 1 {
                return matrix[0]
        }
        i, j := 0, 0
        flag := false
        for j < m {
                a, b := i, j
                temp := make([]int, 0)
                temp = append(temp, matrix[a][b])
                // 从左下往右上
                for a != 0 && b != m-1 {
                        a--
                        b++
                        temp = append(temp, matrix[a][b])
                }
                if flag == true {
                        reverse(temp)
                        flag = false
                } else {
                        flag = true
                }
                res = append(res, temp...)
                if i != n-1 {
                        i++
                } else {
                        j++
                }
        }
        return res
}

func reverse(arr []int) {
        for i := 0; i < len(arr)/2; i++ {
```

(续下页)

(接上页)

```
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
}

# 2
func findDiagonalOrder(matrix [][]int) []int {
        res := make([]int, 0)
        if len(matrix) == 0 {
                return res
        }
        n, m := len(matrix), len(matrix[0])
        if n == 1 {
                return matrix[0]
        }
        // 右边拼接一个相同的，依次遍历
        for i := 0; i < n+m-1; i++ {
                temp := make([]int, 0)
                var a, b int
                if i < m {
                        a = 0
                        b = i
                } else {
                        a = i - m + 1
                        b = m - 1
                }
                for a < n && b >= 0 {
                        temp = append(temp, matrix[a][b])
                        a, b = a+1, b-1
                }
                if i%2 == 0 {
                        reverse(temp)
                }
                res = append(res, temp...)
        }
        return res
}

func reverse(arr []int) {
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
}
```

CHAPTER 15

0401-0500-Hard

15.1 403. 青蛙过河 (4)

- 题目

```
一只青蛙想要过河。␣
↪假定河流被等分为若干个单元格，并且在每一个单元格内都有可能放有一块石子（也有可能没有）。
青蛙可以跳上石子，但是不可以跳入水中。
给你石子的位置列表 stones（用单元格序号 升序 表示），
请判定青蛙能否成功过河（即能否在最后一步跳至最后一块石子上）。
开始时，青蛙默认已站在第一块石子上，并可以假定它第一步只能跳跃一个单位
（即只能从单元格 1 跳至单元格 2 ）。
如果青蛙上一步跳跃了k个单位，那么它接下来的跳跃距离只能选择为k - 1、k或k + 1 个单位。
另请注意，青蛙只能向前方（终点的方向）跳跃。
示例 1：输入：stones = [0,1,3,5,6,8,12,17] 输出：true
解释：青蛙可以成功过河，按照如下方案跳跃：跳 1 个单位到第 2 块石子，
然后跳 2 个单位到第 3 块石子，接着 跳 2 个单位到第 4 块石子，
然后跳 3 个单位到第 6 块石子，跳 4 个单位到第 7 块石子，
最后，跳 5 个单位到第 8 个石子（即最后一块石子）。
示例 2：输入：stones = [0,1,2,3,4,8,9,11] 输出：false
解释：这是因为第 5 和第 6 个石子之间的间距太大，没有可选的方案供青蛙跳跃过去。
提示：2 <= stones.length <= 2000
0 <= stones[i] <= 231 - 1
stones[0] == 0
```

- 解题思路

```
var m [][]int

func canCross(stones []int) bool {
        n := len(stones)
        m = make([][]int, n)
        for i := 0; i < n; i++ {
                m[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        m[i][j] = -1
                }
        }
        return dfs(stones, 0, 0) == 1
}

func dfs(stones []int, index int, k int) int {
        if m[index][k] >= 0 {
                return m[index][k]
        }
        for i := index + 1; i < len(stones); i++ {
                next := stones[i] - stones[index]
                if k-1 <= next && next <= k+1 { // k-1、k或 k+1
                        if dfs(stones, i, next) == 1 {
                                m[index][k] = 1
                                return 1
                        }
                }
        }
        if index == len(stones)-1 {
                m[index][k] = 1
        } else {
                m[index][k] = 0
        }
        return m[index][k]
}

# 2
func canCross(stones []int) bool {
        n := len(stones)
        m := make(map[int]map[int]int)
        for i := 0; i < n; i++ {
                m[stones[i]] = make(map[int]int)
        }
        m[0][0] = 1
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                for k := range m[stones[i]] {
                        for next := k - 1; next <= k+1; next++ {
                                if next > 0 {
                                        if _, ok := m[stones[i]+next]; ok {
                                                m[stones[i]+next][next] = 1
                                        }
                                }
                        }
                }
        }
        return len(m[stones[n-1]]) > 0
}

# 3
func canCross(stones []int) bool {
        n := len(stones)
        dp := make([][]bool, n+1)
        for i := 0; i < n; i++ {
                dp[i] = make([]bool, n+1)
        }
        dp[0][0] = true
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
                        k := stones[i] - stones[j]
                        if k <= i {
                                dp[i][k] = dp[j][k-1] || dp[j][k] || dp[j][k+1] //␣
↪满足其一
                                if i == n-1 && dp[i][k] == true {
                                        return true
                                }
                        }
                }
        }
        return false
}

# 4
type Node struct {
        index int
        size  int
}
```

(续下页)

(接上页)

```
func canCross(stones []int) bool {
        n := len(stones)
        m := make(map[int]bool)
        for i := 0; i < n; i++ {
                m[stones[i]] = true
        }
        isVisited := make(map[Node]bool)

        stack := make([]Node, 0)
        node := Node{
                index: 0,
                size:  0,
        }
        stack = append(stack, node)
        isVisited[node] = true
        for len(stack) > 0 {
                node = stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if node.index == stones[n-1] {
                        return true
                }
                for k := node.size - 1; k <= node.size+1; k++ {
                        next := node.index + k
                        if k > stones[n-1] {
                                continue
                        }
                        temp := Node{
                                index: next,
                                size:  k,
                        }
                        if k > 0 && m[next] == true && isVisited[temp] == false {
                                isVisited[temp] = true
                                stack = append(stack, temp)
                        }
                }
        }
        return false
}
```

15.2 410. 分割数组的最大值 (3)

- 题目

```
给定一个非负整数数组和一个整数 m，你需要将这个数组分成 m 个非空的连续子数组。
设计一个算法使得这 m 个子数组各自和的最大值最小。
注意:数组长度 n 满足以下条件:
    1 ≤ n ≤ 1000
    1 ≤ m ≤ min(50, n)
示例:输入: nums = [7,2,5,10,8] m = 2输出:18
解释:一共有四种方法将nums分割为2个子数组。
其中最好的方式是将其分为[7,2,5] 和 [10,8],
因为此时这两个子数组各自的和的最大值为18，在所有情况中最小。
```

- 解题思路

```
func splitArray(nums []int, m int) int {
        left, right := 0, 0 // 最小值，最大值
        for i := 0; i < len(nums); i++ {
                right = right + nums[i]
                if left < nums[i] {
                        left = nums[i]
                }
        }
        for left < right {
                mid := left + (right-left)/2
                // 继续尝试
                if check(nums, mid, m) {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

// 区间和的最大值为target时，所得出的区间数
func check(arr []int, target int, m int) bool {
        sum := 0
        count := 1
        for i := 0; i < len(arr); i++ {
                // 当sum加上当前值超过了target
                // 我们就把当前取的值作为新的一段分割子数组的开头，并将count加1
                if sum+arr[i] > target {
```

(续下页)

(接上页)

```
                        count++
                        sum = arr[i]
                } else {
                        sum = sum + arr[i]
                }
        }
        return count <= m
}

# 2
func splitArray(nums []int, m int) int {
        n := len(nums)
        dp := make([][]int, n+1)
        sub := make([]int, n+1)
        for i := 0; i < n+1; i++ {
                dp[i] = make([]int, m+1)
                for j := 0; j < m+1; j++ {
                        dp[i][j] = math.MaxInt32
                }
        }
        for i := 0; i < n; i++ {
                sub[i+1] = sub[i] + nums[i]
        }
        // dp[i][j]表示前i个数字被分割成j段的结果
        // 0<=k<i枚举所有可以被分成j-1段的情况
        // 前 k个数被分割为j-1段，而第 k+1到第i个数为第j段
        // dp[i][j] = min{max(dp[k][j-1], sum(k+1, i))}
        dp[0][0] = 0
        for i := 1; i <= n; i++ {
                // 分成m段，可能不够分，最多分min(i,m)
                for j := 1; j <= min(i, m); j++ {
                        for k := 0; k < i; k++ {
                                dp[i][j] = min(dp[i][j], max(dp[k][j-1], sub[i]-
↪sub[k]))
                        }
                }
        }
        return dp[n][m]
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func splitArray(nums []int, m int) int {
        left, right := 0, 0 // 最小值，最大值
        for i := 0; i < len(nums); i++ {
                right = right + nums[i]
                if left < nums[i] {
                        left = nums[i]
                }
        }
        for left < right {
                mid := left + (right-left)/2
                // 分割数太多，继续尝试
                if check(nums, mid, m) > m {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

// 区间和的最大值为target时，所得出的区间数
func check(arr []int, target int, m int) int {
        sum := 0
        count := 1
        for i := 0; i < len(arr); i++ {
                // 当sum加上当前值超过了target
                // 我们就把当前取的值作为新的一段分割子数组的开头，并将count加1
                if sum+arr[i] > target {
                        count++
                        sum = 0
                }
```

(续下页)

(接上页)

```
                sum = sum + arr[i]
        }
        return count
}
```

15.3 440. 字典序的第 K 小数字 (1)

- 题目

```
给定整数n和k，找到1到n中字典序第k小的数字。
注意：1 ≤ k ≤ n ≤ 109。
示例 :输入: n: 13   k: 2 输出: 10
解释:字典序的排列是 [1, 10, 11, 12, 13, 2, 3, 4, 5, 6, 7, 8, 9]，所以第二小的数字是␣
↪10。
```

- 解题思路

```
func findKthNumber(n int, k int) int {
        pre := 1
        count := 1
        for count < k {
                total := getCount(pre, n)
                if count+total > k { // 超过了，*10继续尝试，缩小范围
                        pre = pre * 10
                        count++
                } else if count+total <= k { // 小了，尝试下一个数字开头，扩大范围
                        pre++
                        count = count + total
                }
        }
        return pre
}

// 以数字i开头且不超过最大数字n的数字个数
func getCount(pre, n int) int {
        count := 0
        a := pre
        b := pre + 1
        for a <= n {
                if b < n+1 {
                        count = count + b - a
                } else {
```

(续下页)

(接上页)

```
                        count = count + n + 1 - a
                }
                a = a * 10
                b = b * 10
        }
        return count
}
```

15.4 446. 等差数列划分 II-子序列 (1)

- 题目

```
如果一个数列至少有三个元素，并且任意两个相邻元素之差相同，则称该数列为等差数列。
例如，以下数列为等差数列:
1, 3, 5, 7, 9
7, 7, 7, 7
3, -1, -5, -9
以下数列不是等差数列。
1, 1, 2, 5, 7
数组 A 包含 N 个数，且索引从 0 开始。
该数组子序列将划分为整数序列(P0, P1, ..., Pk)，满足 0 ≤ P0 < P1 < ... < Pk < N。
如果序列 A[P0]，A[P1]，...，A[Pk-1]，A[Pk] 是等差的，那么数组 A 的子序列␣
↪(P0，P1，…，PK) 称为等差序列。
值得注意的是，这意味着 k ≥ 2。
函数要返回数组 A 中所有等差子序列的个数。
输入包含 N 个整数。每个整数都在 -231 和 231-1 之间，另外 0 ≤ N ≤ 1000。保证输出小于␣
↪231-1。
示例：输入：[2, 4, 6, 8, 10] 输出：7
解释：  所有的等差子序列为:
[2,4,6]
[4,6,8]
[6,8,10]
[2,4,6,8]
[4,6,8,10]
[2,4,6,8,10]
[2,6,10]
```

- 解题思路

```
func numberOfArithmeticSlices(nums []int) int {
        n := len(nums)
        dp := make([]map[int]int, n) // dp[i][d]代表以A[i]结束且公差为d的等差数列个数
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                dp[i] = make(map[int]int)
        }
        res := 0
        for i := 1; i < n; i++ {
                for j := 0; j < i; j++ {
                        diff := nums[i] - nums[j]
                        dp[i][diff] = dp[i][diff] + dp[j][diff] + 1
                        res = res + dp[j][diff]
                }
        }
        return res
}
```

15.5 458. 可怜的小猪 (2)

- 题目

```
有 buckets 桶液体，其中 正好 有一桶含有毒药，其余装的都是水。
它们从外观看起来都一样。为了弄清楚哪只水桶含有毒药，你可以喂一些猪喝，通过观察猪是否会死进行判断。
不幸的是，你只有minutesToTest 分钟时间来确定哪桶液体是有毒的。
喂猪的规则如下：
选择若干活猪进行喂养
可以允许小猪同时饮用任意数量的桶中的水，并且该过程不需要时间。
小猪喝完水后，必须有 minutesToDie␣
↪分钟的冷却时间。在这段时间里，你只能观察，而不允许继续喂猪。
过了 minutesToDie 分钟后，所有喝到毒药的猪都会死去，其他所有猪都会活下来。
重复这一过程，直到时间用完。
给你桶的数目 buckets ，minutesToDie 和 minutesToTest␣
↪，返回在规定时间内判断哪个桶有毒所需的 最小 猪数。
示例 1：输入：buckets = 1000, minutesToDie = 15, minutesToTest = 60 输出：5
示例 2：输入：buckets = 4, minutesToDie = 15, minutesToTest = 15 输出：2
示例 3：输入：buckets = 4, minutesToDie = 15, minutesToTest = 30 输出：2
提示：1 <= buckets <= 1000
1 <=minutesToDie <=minutesToTest <= 100
```

- 解题思路

```
func poorPigs(buckets int, minutesToDie int, minutesToTest int) int {
        count := minutesToTest/minutesToDie + 1 // 最多喝几次水
        res := 0
        target := 1
```

(续下页)

(接上页)

```
        // n^res<buckets
        for target < buckets {
                target = target * count
                res++
        }
        return res
}

# 2
func poorPigs(buckets int, minutesToDie int, minutesToTest int) int {
        count := minutesToTest/minutesToDie + 1 // 最多喝几次水
        res := math.Log(float64(buckets)) / math.Log(float64(count))
        return int(math.Ceil(res))
}
```

15.6 460.LFU 缓存 (2)

- 题目

```
请你为 最不经常使用（LFU）缓存算法设计并实现数据结构。它应该支持以下操作：get 和 put。
    get(key) - 如果键存在于缓存中，则获取键的值（总是正数），否则返回 -1。
    put(key, value) - 如果键已存在，则变更其值；如果键不存在，请插入键值对。
    当缓存达到其容量时，则应该在插入新项之前，使最不经常使用的项无效。
   ␣
↪在此问题中，当存在平局（即两个或更多个键具有相同使用频率）时，应该去除最久未使用的键。
「项的使用次数」就是自插入该项以来对其调用 get 和 put␣
↪函数的次数之和。使用次数会在对应项被移除后置为 0 。
进阶：你是否可以在 O(1) 时间复杂度内执行两项操作？
示例：LFUCache cache = new LFUCache( 2 /* capacity (缓存容量) */ );
cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // 返回 1
cache.put(3, 3);    // 去除 key 2
cache.get(2);       // 返回 -1 (未找到key 2)
cache.get(3);       // 返回 3
cache.put(4, 4);    // 去除 key 1
cache.get(1);       // 返回 -1 (未找到 key 1)
cache.get(3);       // 返回 3
cache.get(4);       // 返回 4
```

- 解题思路

```
type Node struct {
        key   int
        value int
        count int
        next  *Node
        prev  *Node
}

type LFUCache struct {
        cap     int
        minFreq int
        kv      map[int]*Node
        fk      map[int]*DoubleList
}

func Constructor(capacity int) LFUCache {
        return LFUCache{
                cap:     capacity,
                kv:      make(map[int]*Node),
                fk:      make(map[int]*DoubleList),
                minFreq: 0,
        }
}

func (this *LFUCache) Get(key int) int {
        node, ok := this.kv[key]
        if ok == false {
                return -1
        }
        this.increaseFreq(node)
        return node.value
}

func (this *LFUCache) Put(key int, value int) {
        if this.cap <= 0 {
                return
        }
        // 已经存在的key, 修改并增加次数
        node, ok := this.kv[key]
        if ok {
                node.value = value
                this.increaseFreq(node)
                return
        }
```

(续下页)

(接上页)

```
        if this.cap <= len(this.kv) {
                last := this.remove()
                delete(this.kv, last.key)
        }
        temp := &Node{
                key:   key,
                value: value,
                count: 1,
        }
        this.kv[key] = temp
        if this.fk[1] == nil {
                this.fk[1] = NewDoubleList()
        }
        this.fk[1].Push(temp)
        this.minFreq = 1 // 新插入的频率为1
}

func (this *LFUCache) increaseFreq(node *Node) {
        freq := node.count
        this.fk[freq].Remove(node)
        if this.fk[freq+1] == nil {
                this.fk[freq+1] = NewDoubleList()
        }
        node.count++
        this.fk[freq+1].Push(node)
        if this.fk[freq].head.next == this.fk[freq].tail {
                if freq == this.minFreq {
                        this.minFreq++
                }
        }
        return
}

func (this *LFUCache) remove() *Node {
        temp := this.fk[this.minFreq]
        last := temp.tail.prev
        temp.Remove(temp.tail.prev) // 删除尾部节点
        return last
}

type DoubleList struct {
        head *Node
        tail *Node
```

(续下页)

(接上页)

```
}

func NewDoubleList() *DoubleList {
        node := &DoubleList{
                head: &Node{},
                tail: &Node{},
        }
        node.head.next = node.tail
        node.tail.prev = node.head
        return node
}

// 插入头部
func (this *DoubleList) Push(node *Node) {
        next := this.head.next
        node.next = next
        node.prev = this.head
        next.prev = node
        this.head.next = node
}

// 删除指定节点
func (this *DoubleList) Remove(node *Node) {
        node.prev.next = node.next
        node.next.prev = node.prev
        node.next = nil
        node.prev = nil
}

# 2
type Node struct {
        key   int
        value int
        count int
}

type LFUCache struct {
        cap     int
        minFreq int
        kv      map[int]*list.Element
        fk      map[int]*list.List
}
```

(续下页)

(接上页)

```
func Constructor(capacity int) LFUCache {
        return LFUCache{
                cap:     capacity,
                minFreq: 0,
                kv:      make(map[int]*list.Element),
                fk:      make(map[int]*list.List),
        }
}

func (this *LFUCache) Get(key int) int {
        node, ok := this.kv[key]
        if ok == false {
                return -1
        }
        return this.increaseFreq(node)
}

func (this *LFUCache) Put(key int, value int) {
        data, ok := this.kv[key]
        if ok {
                node := data.Value.(*Node)
                node.value = value
                this.increaseFreq(data)
                return
        }
        if this.cap == len(this.kv) {
                cur, ok := this.fk[this.minFreq]
                if ok == false {
                        return
                }
                deleteKey := cur.Front()
                cur.Remove(deleteKey)
                delete(this.kv, deleteKey.Value.(*Node).key)
        }
        temp := &Node{
                key:   key,
                value: value,
                count: 1,
        }
        if _, ok := this.fk[1]; ok == false {
                this.fk[1] = list.New()
        }
        res := this.fk[1].PushBack(temp)
```

(续下页)

(接上页)

```
        this.kv[key] = res
        this.minFreq = 1
}

func (this *LFUCache) increaseFreq(data *list.Element) int {
        node := data.Value.(*Node)
        cur, ok := this.fk[node.count]
        if ok == false {
                return -1
        }
        cur.Remove(data)
        if cur.Len() == 0 && this.minFreq == node.count {
                this.minFreq++
        }
        node.count++
        if this.fk[node.count] == nil {
                this.fk[node.count] = list.New()
        }
        res := this.fk[node.count].PushBack(node)
        this.kv[node.key] = res
        return node.value
}
```

15.7 466. 统计重复个数 (1)

- 题目

```
由 n 个连接的字符串 s 组成字符串 S，记作S = [s,n]。例如，["abc",3]=“abcabcabc”。
如果我们可以从 s2中删除某些字符使其变为 s1，则称字符串 s1可以从字符串 s2 获得。
例如，根据定义，"abc" 可以从 “abdbec” 获得，但不能从 “acbbe” 获得。
现在给你两个非空字符串 s1和 s2（每个最多 100 个字符长）和两个整数 0 ≤ n1 ≤ 106和 1 ≤␣
↪n2 ≤ 106。
现在考虑字符串 S1 和 S2，其中 S1=[s1,n1]、S2=[s2,n2] 。
请你找出一个可以满足使[S2,M] 从 S1获得的最大整数 M 。
示例：输入： s1 ="acb",n1 = 4 s2 ="ab",n2 = 2 返回：2
```

- 解题思路

```
func getMaxRepetitions(s1 string, n1 int, s2 string, n2 int) int {
        count := 0
        j := 0
        for a := 0; a < n1; a++ {
```

(续下页)

(接上页)

```
                for i := 0; i < len(s1); i++ {
                        if s1[i] == s2[j] {
                                if j == len(s2)-1 {
                                        j = 0
                                        count++
                                } else {
                                        j++
                                }
                        }
                }
        }
        return count / n2
}
```

15.8 472. 连接词 (1)

- 题目

```
给你一个 不含重复 单词的字符串数组 words ，请你找出并返回 words 中的所有 连接词 。
连接词 定义为：一个完全由给定数组中的至少两个较短单词组成的字符串。
示例 1：输入：words = ["cat","cats","catsdogcats","dog","dogcatsdog","hippopotamuses",
↪"rat","ratcatdogcat"]
输出：["catsdogcats","dogcatsdog","ratcatdogcat"]
解释："catsdogcats" 由 "cats", "dog" 和 "cats" 组成;
     "dogcatsdog" 由 "dog", "cats" 和 "dog" 组成;
     "ratcatdogcat" 由 "rat", "cat", "dog" 和 "cat" 组成。
示例 2：输入：words = ["cat","dog","catdog"] 输出：["catdog"]
提示：1 <= words.length <= 104
0 <= words[i].length <= 1000
words[i] 仅由小写字母组成
0 <= sum(words[i].length) <= 105
```

- 解题思路

```
func findAllConcatenatedWordsInADict(words []string) []string {
        res := make([]string, 0)
        sort.Slice(words, func(i, j int) bool {
                return len(words[i]) < len(words[j])
        })
        root := Trie{}
        for i := 0; i < len(words); i++ {
                if words[i] == "" {
```

(续下页)

(接上页)

```
                        continue
                }
                if root.Dfs(words[i]) == true {
                        res = append(res, words[i])
                } else {
                        root.Insert(words[i])
                }
        }
        return res
}

type Trie struct {
        next   [26]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int       // 次数（可以改为bool）
}

// 插入word
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

func (this *Trie) Dfs(word string) bool {
        if word == "" {
                return true
        }
        temp := this
        for i := 0; i < len(word); i++ {
                value := word[i] - 'a'
                temp = temp.next[value]
                if temp == nil {
                        return false
                }
```

(续下页)

(接上页)

```
                if temp.ending > 0 && this.Dfs(word[i+1:]) == true {
                        return true
                }
        }
        return false
}
```

15.9 479. 最大回文数乘积 (1)

- 题目

```
你需要找到由两个 n 位数的乘积组成的最大回文数。
由于结果会很大，你只需返回最大回文数 mod 1337得到的结果。
示例:输入: 2 输出: 987
解释: 99 x 91 = 9009, 9009 % 1337 = 987
说明:n 的取值范围为[1,8]。
```

- 解题思路

```
func largestPalindrome(n int) int {
        if n == 1 {
                return 9
        }
        last := int(math.Pow10(n)) - 1 // 10^n-1 : n=3 999
    first := last / 10              // 10^(n-1)-1 n=3 99
        for i := last; i > first; i-- {
                target := makePalindrome(i) // 依次生成对应的递减回文数: 如98=>9889
                for j := last; j > first && target < j*j; j-- {
                        if target%j == 0 { // 该回文数满足要求
                                return target % 1337
                        }
                }
        }
        return 0
}

func makePalindrome(num int) int {
        str := strconv.Itoa(num)
        arr := []byte(str)
        for i := len(str) - 1; i > -1; i-- {
                arr = append(arr, str[i])
        }
```

(续下页)

(接上页)

```
        res, _ := strconv.Atoi(string(arr))
        return res
}
```

15.10 480. 滑动窗口中位数

15.10.1 题目

```
中位数是有序序列最中间的那个数。如果序列的长度是偶数，则没有最中间的数；此时中位数是最中间的两个数的
例如：[2,3,4]，中位数是3
[2,3]，中位数是 (2 + 3) / 2 = 2.5
给你一个数组 nums，有一个长度为 k 的窗口从最左端滑动到最右端。窗口中有 k␣
↪个数，每次窗口向右移动 1 位。
你的任务是找出每次窗口移动后得到的新窗口中元素的中位数，并输出由它们组成的数组。
示例：给出nums = [1,3,-1,-3,5,3,6,7]，以及k = 3。
窗口位置                      中位数
---------------               -----
[1  3  -1] -3  5  3  6  7       1
 1 [3  -1  -3] 5  3  6  7      -1
 1  3 [-1  -3  5] 3  6  7      -1
 1  3  -1 [-3  5  3] 6  7       3
 1  3  -1  -3 [5  3  6] 7       5
 1  3  -1  -3  5 [3  6  7]      6
因此，返回该滑动窗口的中位数数组[1,-1,-1,3,5,6]。
提示：你可以假设k始终有效，即：k 始终小于等于输入的非空数组的元素个数。
与真实值误差在 10 ^ -5 以内的答案将被视作正确答案。
```

15.10.2 解题思路

```

```

15.11 493. 翻转对 (4)

- 题目

```
给定一个数组nums，如果i < j且nums[i] > 2*nums[j]我们就将(i, j)称作一个重要翻转对。
你需要返回给定数组中的重要翻转对的数量。
示例 1:输入: [1,3,2,3,1] 输出: 2
```

(续下页)

(接上页)

```
示例 2:输入: [2,4,3,5,1] 输出: 3
注意:
给定数组的长度不会超过50000。
输入数组中的所有数字都在32位整数的表示范围内。
```

- 解题思路

```
func reversePairs(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        a := append([]int{}, nums[:n/2]...)
        b := append([]int{}, nums[n/2:]...)
        res := reversePairs(a) + reversePairs(b) // 递归后，a、b分别有序
        i, j := 0, 0
        for i = 0; i < len(a); i++ {
                for j < len(b) && a[i] > 2*b[j] { // 统计
                        j++
                }
                res = res + j
        }
        i, j = 0, 0
        for k := 0; k < len(nums); k++ { // 2个有序数组合并
                if i < len(a) && (j == len(b) || a[i] <= b[j]) {
                        nums[k] = a[i]
                        i++
                } else {
                        nums[k] = b[j]
                        j++
                }
        }
        return res
}

# 2
func reversePairs(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        temp := make([]int, 0)
        for i := 0; i < n; i++ {
                // 会存在负数的情况: [-5,-5] => -5 > -10
```

(续下页)

(接上页)

```
                // 所以加入 2*nums[i] 参与离散化，后面直接查询2*nums[i]
                temp = append(temp, nums[i], 2*nums[i])
        }
        arr, m := getLSH(temp)
        length = len(arr)
        c = make([]int, length+1)
        res := 0
        for i := 0; i < n; i++ {
                index := m[2*nums[i]]
                // nums[i] > 2*nums[j] => x > 2*nums[i]
                count := getSum(index) // 统计之前插入了多少个小于等于2*nums[i]的数
                res = res + i - count  // i-count：剩下的就是大于2*nums[i]的数
                upData(m[nums[i]], 1)  // nums[i]次数+1
        }
        return res
}

// 离散化
func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var length int
var c []int // 树状数组

func lowBit(x int) int {
```

(续下页)

(接上页)

```
        return x & (-x)
}

// 单点修改
func upData(i, k int) { // 在i位置加上k
        for i <= length {
                c[i] = c[i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(i int) int {
        res := 0
        for i > 0 {
                res = res + c[i]
                i = i - lowBit(i)
        }
        return res
}

# 3
func reversePairs(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        temp := make([]int, 0)
        for i := 0; i < n; i++ {
                // 会存在负数的情况: [-5,-5] => -5 > -10
                // 所以加入 2*nums[i] 参与离散化, 后面直接查询2*nums[i]
                temp = append(temp, nums[i], 2*nums[i])
        }
        arr, m := getLSH(temp)
        length = len(arr)
        c = make([]int, length+1)
        res := 0
        for i := n - 1; i >= 0; i-- {
                index := m[nums[i]]
                // nums[i] > 2*nums[j] => 求小于nums[i], 其中2*nums[j]已经插入
                count := getSum(index - 1) // 统计插入了多少个小于nums[i]的数
                res = res + count          //
                upData(m[2*nums[i]], 1)    // 2*nums[j]次数+1
```

(续下页)

(接上页)

```
        }
        return res
}

// 离散化
func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var length int
var c []int // 树状数组

func lowBit(x int) int {
        return x & (-x)
}

// 单点修改
func upData(i, k int) { // 在i位置加上k
        for i <= length {
                c[i] = c[i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(i int) int {
```

(续下页)

(接上页)

```
        res := 0
        for i > 0 {
                res = res + c[i]
                i = i - lowBit(i)
        }
        return res
}

# 4
func reversePairs(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        temp := make([]int, 0)
        for i := 0; i < n; i++ {
                // 会存在负数的情况：[-5,-5] => -5 > -10
                // 所以加入 2*nums[i] 参与离散化，后面直接查询2*nums[i]
                temp = append(temp, nums[i], 2*nums[i])
        }
        tempArr, m := getLSH(temp)
        total := len(tempArr)
        arr = make([]int, 4*total+1)
        res := 0
        for i := n - 1; i >= 0; i-- {
                index := m[nums[i]]
                // nums[i] > 2*nums[j] => 求小于nums[i]，其中2*nums[j]已经插入
                count := query(0, 1, total, 1, index-1) //␣
↪统计插入了多少个小于nums[i]的数
                res = res + count                       //
                update(0, 1, total, m[2*nums[i]], 1)    // 2*nums[j]次数+1
        }
        return res
}

// 离散化
func getLSH(a []int) ([]int, map[int]int) {
        n := len(a)
        arr := make([]int, n)
        copy(arr, a)
        sort.Ints(arr) // 排序
        m := make(map[int]int)
        m[arr[0]] = 1
```

(续下页)

(接上页)

```
        index := 1
        for i := 1; i < n; i++ {
                if arr[i] != arr[i-1] {
                        index++
                        m[arr[i]] = index
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = m[a[i]]
        }
        return res, m
}

var arr []int // 线段树

func update(id int, left, right, x int, value int) {
        if left > x || right < x {
                return
        }
        arr[id] = arr[id] + value
        if left == right {
                return
        }
        mid := left + (right-left)/2
        update(2*id+1, left, mid, x, value)    // 左节点
        update(2*id+2, mid+1, right, x, value) // 右节点
}

func query(id int, left, right, queryLeft, queryRight int) int {
        if left > queryRight || right < queryLeft {
                return 0
        }
        if queryLeft <= left && right <= queryRight {
                return arr[id]
        }
        mid := left + (right-left)/2
        return query(id*2+1, left, mid, queryLeft, queryRight) +
                query(id*2+2, mid+1, right, queryLeft, queryRight)
}
```

CHAPTER 16

0501-0600-Easy

16.1 501. 二叉搜索树中的众数 (2)

- 题目

```
给定一个有相同值的二叉搜索树（BST），找出 BST 中的所有众数（出现频率最高的元素）。
假定 BST 有如下定义：
    结点左子树中所含结点的值小于等于当前结点的值
    结点右子树中所含结点的值大于等于当前结点的值
    左子树和右子树都是二叉搜索树
例如：
给定 BST [1,null,2,2],
   1
    \
     2
    /
   2
返回[2].
提示：如果众数超过1个，不需考虑输出顺序
进阶：你可以不使用额外的空间吗？（假设由递归产生的隐式调用栈的开销不被计算在内）
```

- 解题思路

```
func findMode(root *TreeNode) []int {
        m := map[int]int{}
```

(续下页)

(接上页)

```
        dfs(root, m)
        max := -1
        res := make([]int, 0)
        for i, v := range m {
                if max <= v {
                        if max < v {
                                max = v
                                res = res[0:0]
                        }
                        res = append(res, i)
                }
        }
        return res
}

func dfs(root *TreeNode, rec map[int]int) {
        if root == nil {
                return
        }
        rec[root.Val]++
        dfs(root.Left, rec)
        dfs(root.Right, rec)
}

#
var max int
var res []int
var cur int
var count int

func findMode(root *TreeNode) []int {
        res = make([]int, 0)
        max, cur, count = 0, 0, 0
        dfs(root)
        return res
}

// 中序遍历保证利用二叉搜索树的性质，得到的结果是升序的
func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
```

(续下页)

(接上页)

```
        if root.Val != cur {
                count = 0
        }
        count++
        if max < count {
                max = count
                res = []int{root.Val}
        } else if max == count {
                res = append(res, root.Val)
        }
        cur = root.Val
        dfs(root.Right)
}
```

16.2 504. 七进制数 (3)

- 题目

```
给定一个整数，将其转化为7进制，并以字符串形式输出。
示例 1:输入: 100 输出: "202"
示例 2: 输入: -7 输出: "-10"
注意: 输入范围是 [-1e7, 1e7] 。
```

- 解题思路

```
func convertToBase7(num int) string {
        if num == 0 {
                return "0"
        }

        minus := ""
        if num < 0 {
                minus = "-"
                num = -1 * num
        }

        s := ""
        for num > 0 {
                s = fmt.Sprintf("%d", num%7) + s
                num = num / 7
        }
        return minus + s
```

(续下页)

(接上页)

```
}

#
func convertToBase7(num int) string {
        return strconv.FormatInt(int64(num), 7)
}

#
func convertToBase7(num int) string {
        if num < 0 {
                return "-" + convertToBase7(-1*num)
        }
        if num < 7 {
                return strconv.Itoa(num)
        }
        return convertToBase7(num/7) + strconv.Itoa(num%7)
}
```

16.3 506. 相对名次 (1)

- 题目

```
给出 N 名运动员的成绩，找出他们的相对名次并授予前三名对应的奖牌。
前三名运动员将会被分别授予 “金牌”，“银牌” 和 “ 铜牌”
("Gold Medal", "Silver Medal", "Bronze Medal") 。
(注：分数越高的选手，排名越靠前。)

示例 1:
输入: [5, 4, 3, 2, 1]
输出: ["Gold Medal", "Silver Medal", "Bronze Medal", "4", "5"]
解释: 前三名运动员的成绩为前三高的，
因此将会分别被授予 “金牌”，“银牌”和“铜牌” ("Gold Medal", "Silver Medal" and
↪"Bronze Medal").
余下的两名运动员，我们只需要通过他们的成绩计算将其相对名次即可。
提示:
    N 是一个正整数并且不会超过 10000。
    所有运动员的成绩都不相同。
```

- 解题思路

```
func findRelativeRanks(nums []int) []string {
        temp := make([]int, len(nums))
```

(续下页)

(接上页)

```
        copy(temp, nums)
        sort.Ints(temp)
        m := make(map[int]string)
        for i := 0; i < len(temp); i++ {
                if i == len(temp)-1 {
                        m[temp[i]] = "Gold Medal"
                } else if i == len(temp)-2 {
                        m[temp[i]] = "Silver Medal"
                } else if i == len(temp)-3 {
                        m[temp[i]] = "Bronze Medal"
                } else {
                        m[temp[i]] = strconv.Itoa(len(temp) - i)
                }
        }
        res := make([]string,0)
        for i := 0; i < len(nums); i++ {
                res = append(res, m[nums[i]])
        }
        return res
}
```

16.4 507. 完美数 (1)

- 题目

```
对于一个 正整数，如果它和除了它自身以外的所有正因子之和相等，我们称它为“完美数”。
给定一个 整数 n， 如果他是完美数，返回 True，否则返回 False
示例：输入：28 输出：True 解释：28 = 1 + 2 + 4 + 7 + 14
提示：输入的数字 n 不会超过 100,000,000. (1e8)
```

- 解题思路

```
func checkPerfectNumber(num int) bool {
        if num == 1 {
                return false
        }
        sum := 1
        for i := 2; i <= num/i; i++ {
                if num%i == 0 {
                        sum = sum + i + (num / i)
                }
        }
```

(续下页)

(接上页)

```
        return sum == num
}
```

16.5 509. 斐波那契数 (6)

- 题目

```
斐波那契数，通常用 F(n) 表示，形成的序列称为斐波那契数列。
该数列由 0 和 1 开始，后面的每一项数字都是前面两项数字的和。也就是：
F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), 其中 N > 1.
给定 N，计算 F(N)。
示例 1：输入：2输出：1
解释：F(2) = F(1) + F(0) = 1 + 0 = 1.
示例 2：输入：3输出：2
解释：F(3) = F(2) + F(1) = 1 + 1 = 2.
示例 3：输入：4输出：3
解释：F(4) = F(3) + F(2) = 2 + 1 = 3.
提示：
    0 ≤ N ≤ 30
```

- 解题思路

```
func fib(N int) int {
        if N == 0 {
                return 0
        }
        if N == 1 {
                return 1
        }
        n1, n2 := 0, 1
        for i := 2; i <= N; i++ {
                n1, n2 = n2, n1+n2
        }
        return n2
}

#
func fib(N int) int {
        if N == 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        if N == 1 {
                return 1
        }
        res := make([]int, N+1)
        res[0] = 0
        res[1] = 1
        for i := 2; i <= N; i++ {
                res[i] = res[i-1] + res[i-2]
        }
        return res[N]
}

#
func fib(N int) int {
        if N == 0 {
                return 0
        }
        if N == 1 {
                return 1
        }
        return fib(N-1) + fib(N-2)
}

#
func fib(N int) int {
        temp1 := (1 + math.Sqrt(5)) / 2
        temp2 := (1 - math.Sqrt(5)) / 2
        fn := math.Round((math.Pow(temp1, float64(N))- math.Pow(temp2, float64(N)))/␣
↪math.Sqrt(5))
        return int(fn)
}

# 5
func fib(N int) int {
        if N == 0 {
                return 0
        }
        /*
                ans = [Fn+1 Fn
                           Fn Fn-1]
                    = [ 1 0
                              0 1]
        */
```

(续下页)

(接上页)

```
        ans := matrix{
                a: 1,
                b: 0,
                c: 0,
                d: 1,
        }
        m := matrix{
                a: 1,
                b: 1,
                c: 1,
                d: 0,
        }
        for N > 0 {
                if N%2 == 1 {
                        ans = multi(ans, m)
                }
                m = multi(m, m)
                N = N >> 1
        }
        return ans.b
}

/*
a b
c d
*/
type matrix struct {
        a, b, c, d int
}

// 矩阵乘法
func multi(x, y matrix) matrix {
        newA := x.a*y.a + x.b*y.c
        newB := x.a*y.b + x.b*y.d
        newC := x.c*y.a + x.d*y.c
        newD := x.c*y.b + x.d*y.d
        return matrix{
                a: newA,
                b: newB,
                c: newC,
                d: newD,
        }
}
```

(续下页)

(接上页)

```
# 6
func fib(N int) int {
        if N == 0 {
                return 0
        }
        /*
                ans = [Fn+1 Fn
                           Fn Fn-1]
                        = [ 1 0
                                  0 1]
        */
        ans := matrix{
                a: 1,
                b: 0,
                c: 0,
                d: 1,
        }
        m := matrix{
                a: 1,
                b: 1,
                c: 1,
                d: 0,
        }
        for N > 0 {
                ans = multi(ans, m)
                N--
        }
        return ans.b
}

/*
a b
c d
*/
type matrix struct {
        a, b, c, d int
}

// 矩阵乘法
func multi(x, y matrix) matrix {
        newA := x.a*y.a + x.b*y.c
        newB := x.a*y.b + x.b*y.d
```

(续下页)

(接上页)

```
        newC := x.c*y.a + x.d*y.c
        newD := x.c*y.b + x.d*y.d
        return matrix{
                a: newA,
                b: newB,
                c: newC,
                d: newD,
        }
}
```

16.6 520. 检测大写字母 (2)

- 题目

```
给定一个单词，你需要判断单词的大写使用是否正确。
我们定义，在以下情况时，单词的大写用法是正确的：
    全部字母都是大写，比如"USA"。
    单词中所有字母都不是大写，比如"leetcode"。
    如果单词不只含有一个字母，只有首字母大写， 比如 "Google"。
否则，我们定义这个单词没有正确使用大写字母。
示例 1:输入: "USA"输出: True
示例 2:输入: "FlaG"输出: False
注意: 输入是由大写和小写拉丁字母组成的非空单词。
```

- 解题思路

```
func detectCapitalUse(word string) bool {
        if word == "" {
                return false
        }
        count := 0
        for i := 0; i < len(word); i++ {
                if word[i] >= 'A' && word[i] <= 'Z' {
                        count++
                }
        }

        if count == 0 || count == len(word) ||
                (count == 1 && word[0] >= 'A' && word[0] <= 'Z') {
                return true
        }
        return false
```

(续下页)

(接上页)

```
}

#
func detectCapitalUse(word string) bool {
        pattern := "(^[a-z]+)$|(^[A-Z]+)$|(^[A-Z]{1}[a-z]*)$"
        isMatch, _ := regexp.MatchString(pattern, word)
        return isMatch
}
```

16.7 521. 最长特殊序列 I(1)

- 题目

```
给你两个字符串，请你从这两个字符串中找出最长的特殊序列。
「最长特殊序列」定义如下：该序列为某字符串独有的最长子序列（即不能是其他字符串的子序列）。

子序列␣
↪可以通过删去字符串中的某些字符实现，但不能改变剩余字符的相对顺序。空序列为所有字符串的子序列，任何

输入为两个字符串，输出最长特殊序列的长度。如果不存在，则返回 -1。

示例 1：输入："aba", "cdc" 输出：3
解释：最长特殊序列可为 "aba"（或 "cdc"），两者均为自身的子序列且不是对方的子序列。
示例 2：输入：a = "aaa", b = "bbb"输出：3
示例 3：输入：a = "aaa", b = "aaa"输出：-1
提示：
    两个字符串长度均处于区间 [1 - 100] 。
    字符串中的字符仅含有 'a'~'z' 。
```

- 解题思路

```
func findLUSlength(a string, b string) int {
        if a == b {
                return -1
        }
        return max(len(a), len(b))
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

16.8 530. 二叉搜索树的最小绝对差 (3)

- 题目

```
给你一棵所有节点为非负值的二叉搜索树，请你计算树中任意两节点的差的绝对值的最小值。
示例：
输入：

   1
    \
     3
    /
   2

输出：1
解释：
最小绝对差为 1，其中 2 和 1 的差的绝对值为 1（或者 2 和 3）。
提示：
    树中至少有 2 个节点。
    本题与 783 https://leetcode.cn/problems/minimum-distance-between-bst-nodes/ 相同
```

- 解题思路

```
var minDiff, previous int
func getMinimumDifference(root *TreeNode) int {
        minDiff, previous  = math.MaxInt32, math.MaxInt32
        dfs(root)
        return minDiff
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)

        newDiff := diff(previous, root.Val)
        if minDiff > newDiff {
                minDiff = newDiff
```

(续下页)

(接上页)

```
        }
        previous = root.Val
        dfs(root.Right)
}

func diff(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func getMinimumDifference(root *TreeNode) int {
        arr := make([]int, 0)
        dfs(root, &arr)
        minDiff := arr[1] - arr[0]
        for i := 2; i < len(arr); i++ {
                if minDiff > arr[i]-arr[i-1] {
                        minDiff = arr[i] - arr[i-1]
                }
        }
        return minDiff
}

func dfs(root *TreeNode, arr *[]int) {
        if root == nil {
                return
        }
        dfs(root.Left, arr)
        *arr = append(*arr, root.Val)
        dfs(root.Right, arr)
}
```

16.9 532. 数组中的 K-diff 数对 (3)

- 题目

```
给定一个整数数组和一个整数 k, 你需要在数组里找到不同的 k-diff 数对。
这里将 k-diff 数对定义为一个整数对 (i, j), 其中 i 和 j␣
↪都是数组中的数字, 且两数之差的绝对值是 k.
```

(续下页)

(接上页)

```
示例 1: 输入: [3, 1, 4, 1, 5], k = 2 输出: 2
解释: 数组中有两个 2-diff 数对, (1, 3) 和 (3, 5)。
尽管数组中有两个1, 但我们只应返回不同的数对的数量。
示例 2:输入:[1, 2, 3, 4, 5], k = 1 输出: 4
解释: 数组中有四个 1-diff 数对, (1, 2), (2, 3), (3, 4) 和 (4, 5)。
示例 3:输入: [1, 3, 1, 5, 4], k = 0 输出: 1
解释: 数组中只有一个 0-diff 数对, (1, 1)。
注意:
    数对 (i, j) 和数对 (j, i) 被算作同一数对。
    数组的长度不超过10,000。
    所有输入的整数的范围在 [-1e7, 1e7]。
```

- 解题思路

```
func findPairs(nums []int, k int) int {
        if k < 0 {
                return 0
        }
        record := make(map[int]int)
        for _, num := range nums {
                record[num]++
        }
        res := 0
        if k == 0 {
                for _, count := range record {
                        if count > 1 {
                                res++
                        }
                }
                return res
        } else {
                for n := range record {
                        if record[n-k] > 0 {
                                res++
                        }
                }
                return res
        }
}

#
func findPairs(nums []int, k int) int {
        if k < 0 {
                return 0
```

(续下页)

(接上页)

```
        }
        m := make(map[int]bool)
        res := make(map[int]bool)
        for _, value := range nums {
                if m[value-k] {
                        res[value-k] = true
                }
                if m[value+k] {
                        res[value] = true
                }
                m[value] = true
        }
        return len(res)
}
```

16.10 538. 把二叉搜索树转换为累加树 (2)

- 题目

```
给定一个二叉搜索树（Binary Search Tree），把它转换成为累加树（Greater Tree），
使得每个节点的值是原来的节点值加上所有大于它的节点值之和。
例如：
输入：原始二叉搜索树：
              5
            /   \
           2     13
输出：转换为累加树：
             18
            /   \
          20     13
注意：
本题和 1038: https://leetcode.cn/problems/binary-search-tree-to-greater-sum-tree/ 相同
```

- 解题思路

```
func convertBST(root *TreeNode) *TreeNode {
        sum := 0
        dfs(root, &sum)
        return root
}

func dfs(root *TreeNode, sum *int) {
```

(续下页)

(接上页)

```
        if root == nil {
                return
        }
        dfs(root.Right, sum)
        *sum = *sum + root.Val
        root.Val = *sum
        dfs(root.Left, sum)
}

#
func convertBST(root *TreeNode) *TreeNode {
        if root == nil {
                return root
        }
        stack := make([]*TreeNode, 0)
        temp := root
        sum := 0
        for {
                if temp != nil {
                        stack = append(stack, temp)
                        temp = temp.Right
                } else if len(stack) != 0 {
                        temp = stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        temp.Val = temp.Val + sum
                        sum = temp.Val
                        temp = temp.Left
                } else {
                        break
                }
        }
        return root
}
```

16.11 541. 反转字符串 II(2)

- 题目

```
给定一个字符串和一个整数 k，你需要对从字符串开头算起的每个 2k␣
↪个字符的前k个字符进行反转。
如果剩余少于 k 个字符，则将剩余的所有全部反转。
如果有小于 2k 但大于或等于 k 个字符，则反转前 k 个字符，并将剩余的字符保持原样。
```

(续下页)

(接上页)

```
示例:
输入: s = "abcdefg", k = 2
输出: "bacdfeg"
要求:
    该字符串只包含小写的英文字母。
    给定字符串的长度和 k 在[1, 10000]范围内。
```

- 解题思路

```
func reverseStr(s string, k int) string {
        arr := []byte(s)
        for i := 0; i < len(s); i = i + 2*k {
                j := min(i+k, len(s))
                reverse(arr[i:j])
        }
        return string(arr)
}

func reverse(arr []byte) {
        i, j := 0, len(arr)-1
        for i < j {
                arr[i], arr[j] = arr[j], arr[i]
                i++
                j--
        }
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

#
func reverseStr(s string, k int) string {
        arr := []byte(s)
        for i := 0; i < len(s); i = i + k {
                if i%(2*k) == 0 {
                        j := i + k
                        if len(arr) < j {
                                j = len(arr)
                        }
```

(续下页)

(接上页)

```
                        reverse(arr[i:j])
                }
        }
        return string(arr)
}

func reverse(arr []byte) {
        i, j := 0, len(arr)-1
        for i < j {
                arr[i], arr[j] = arr[j], arr[i]
                i++
                j--
        }
}
```

16.12 543. 二叉树的直径 (2)

- 题目

```
给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。
这条路径可能穿过也可能不穿过根结点。
示例 :
给定二叉树
          1
         / \
        2   3
       / \
      4   5
返回 3, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。
注意：两结点之间的路径长度是以它们之间边的数目表示。
```

- 解题思路

```
var res int
func diameterOfBinaryTree(root *TreeNode) int {
        res = 0
        dfs(root)
        return res
}

func dfs(root *TreeNode) int {
        if root == nil {
```

(续下页)

(接上页)

```
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        path := max(left, right)
        res = max(left+right, res) // 当前节点最大直径与当前保存最大值比较
        return path + 1 // 以该节点为根的最大深度
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func diameterOfBinaryTree(root *TreeNode) int {
        if root == nil {
                return 0
        }
        max := 0
        stack := make([]*TreeNode, 0)
        m := make(map[*TreeNode]int)

        cur := root
        var prev *TreeNode
        for cur != nil || len(stack) != 0 {
                for cur != nil {
                        stack = append(stack, cur)
                        cur = cur.Left
                }
                cur = stack[len(stack)-1]
                if cur.Right == nil || cur.Right == prev {
                        cur = stack[len(stack)-1]
                        stack = stack[:len(stack)-1]

                        leftLen := 0
                        rightLen := 0
                        if v, ok := m[cur.Left]; ok {
                                leftLen = v
                        }
                        if v, ok := m[cur.Right]; ok {
```

(续下页)

(接上页)

```
                                rightLen = v
                        }
                        if leftLen > rightLen {
                                m[cur] = leftLen + 1
                        } else {
                                m[cur] = rightLen + 1
                        }
                        if max < leftLen+rightLen {
                                max = leftLen + rightLen
                        }
                        prev = cur
                        cur = nil
                } else {
                        cur = cur.Right
                }
        }
        return max
}
```

16.13 551. 学生出勤记录 I(2)

- 题目

```
给定一个字符串来代表一个学生的出勤记录，这个记录仅包含以下三个字符：
    'A' : Absent，缺勤
    'L' : Late，迟到
    'P' : Present，到场
如果一个学生的出勤记录中不超过一个'A'(缺勤)并且不超过两个连续的'L'(迟到),
↪那么这个学生会被奖赏。
你需要根据这个学生的出勤记录判断他是否会被奖赏。

示例 1:输入: "PPALLP"输出: True
示例 2:输入: "PPALLL"输出: False
```

- 解题思路

```
func checkRecord(s string) bool {
        if strings.Count(s, "A") <= 1 && strings.Count(s, "LLL") == 0 {
                return true
        }
        return false
}
```

(续下页)

(接上页)

```
#
func checkRecord(s string) bool {
        aNum := 0
        lNum := 0
        for i := 0; i < len(s); i++ {
                if s[i] == 'A' {
                        aNum++
                }
                if s[i] == 'L' {
                        lNum++
                } else {
                        lNum = 0
                }
                if aNum == 2 {
                        return false
                }
                if lNum == 3 {
                        return false
                }
        }
        return true
}
```

16.14 557. 反转字符串中的单词 III(2)

- 题目

```
给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。
示例 1:
输入: "Let's take LeetCode contest"
输出: "s'teL ekat edoCteeL tsetnoc"

注意: 在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。
```

- 解题思路分析

```
func reverseWords(s string) string {
        strS := strings.Split(s, " ")
        for i, s := range strS {
                strS[i] = reverse(s)
```

(续下页)

(接上页)

```
        }
        return strings.Join(strS, " ")
}

func reverse(s string) string {
        arr := []byte(s)
        i, j := 0, len(arr)-1
        for i < j {
                arr[i], arr[j] = arr[j], arr[i]
                i++
                j--
        }
        return string(arr)
}

#
func reverseWords(s string) string {
        arr := []byte(s)
        j := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] == ' ' {
                        reverse(arr, j, i-1)
                        j = i + 1
                }
        }
        reverse(arr, j, len(arr)-1)
        return string(arr)
}

func reverse(arr []byte, i, j int) []byte {
        for i < j {
                arr[i], arr[j] = arr[j], arr[i]
                i++
                j--
        }
        return arr
}
```

16.15 559.N 叉树的最大深度 (2)

- 题目

```
给定一个 N 叉树，找到其最大深度。
最大深度是指从根节点到最远叶子节点的最长路径上的节点总数。
例如，给定一个 3叉树 :
我们应返回其最大深度，3。
说明:
    树的深度不会超过 1000。
    树的节点总不会超过 5000。
```

- 解题思路

```
func maxDepth(root *Node) int {
        if root == nil {
                return 0
        }
        depth := 0
        for _, node := range root.Children {
                depth = max(depth, maxDepth(node))
        }
        return depth + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func maxDepth(root *Node) int {
        if root == nil {
                return 0
        }
        queue := make([]*Node, 0)
        depth := 0
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        temp := queue[0]
```

(续下页)

(接上页)

```
                        for _, node := range temp.Children {
                                queue = append(queue, node)
                        }
                        queue = queue[1:]
                }
                depth++
        }
        return depth
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

16.16 561. 数组拆分 I(2)

- 题目

```
给定长度为 2n 的数组, 你的任务是将这些数分成 n 对, 例如 (a1, b1), (a2, b2), ..., (an,␣
↪bn) ,
使得从1 到 n 的 min(ai, bi) 总和最大。

示例 1:输入: [1,4,3,2]输出: 4
解释: n 等于 2, 最大总和为 4 = min(1, 2) + min(3, 4).
提示:
    n 是正整数,范围在 [1, 10000].
    数组中的元素范围在 [-10000, 10000].
```

- 解题思路

```
func arrayPairSum(nums []int) int {
        sort.Ints(nums)
        sum := 0
        for k, v := range nums {
                if k%2 == 0 {
                        sum = sum + v
                }
        }
        return sum
```

(续下页)

(接上页)

```
}

#
func arrayPairSum(nums []int) int {
        var arr [20010]int
        for _, num := range nums {
                arr[num+10000]++
        }
        sum := 0
        needAdd := true
        for num, count := range arr {
                for count > 0 {
                        if needAdd {
                                sum = sum + num - 10000
                        }
                        needAdd = !needAdd
                        count--
                }
        }
        return sum
}
```

16.17 563. 二叉树的坡度 (2)

- 题目

```
给定一个二叉树，计算整个树的坡度。
一个树的节点的坡度定义即为，该节点左子树的结点之和和右子树结点之和的差的绝对值。空结点的的坡度是0。
整个树的坡度就是其所有节点的坡度之和。

示例：

输入：
          1
        /   \
       2     3
输出：1
解释：
结点的坡度 2 : 0
结点的坡度 3 : 0
结点的坡度 1 : |2-3| = 1
树的坡度 : 0 + 0 + 1 = 1
```

(续下页)

(接上页)

```
注意:
    任何子树的结点的和不会超过32位整数的范围。
    坡度的值不会超过32位整数的范围。
```

- 解题思路

```
var total int

func findTilt(root *TreeNode) int {
        total = 0
        dfs(root)
        return total
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        total = total + abs(left, right)
        return left + right + root.Val // 返回节点之和
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func findTilt(root *TreeNode) int {
        if root == nil {
                return 0
        }
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        list := make([]*TreeNode, 0)
        total := 0
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[0 : len(stack)-1]
                list = append([]*TreeNode{node}, list...)
```

(续下页)

(接上页)

```
                if node.Left != nil {
                        stack = append(stack, node.Left)
                }
                if node.Right != nil {
                        stack = append(stack, node.Right)
                }
        }
        for i := range list {
                node := list[i]
                left := 0
                right := 0
                if node.Left != nil {
                        left = node.Left.Val
                }
                if node.Right != nil {
                        right = node.Right.Val
                }
                total = total + abs(left, right)
                node.Val = left + right + node.Val
        }
        return total
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

16.18 566. 重塑矩阵 (2)

- 题目

```
在MATLAB中，有一个非常有用的函数␣
↪reshape，它可以将一个矩阵重塑为另一个大小不同的新矩阵，但保留其原始数据。
给出一个由二维数组表示的矩阵，以及两个正整数r和c，分别表示想要的重构的矩阵的行数和列数。
重构后的矩阵需要将原始矩阵的所有元素以相同的行遍历顺序填充。
如果具有给定参数的reshape操作是可行且合理的，则输出新的重塑矩阵；否则，输出原始矩阵。

示例 1:输入:
nums =
```

(续下页)

(接上页)

```
[[1,2],
 [3,4]]
r = 1, c = 4
输出:
[[1,2,3,4]]
解释:行遍历nums的结果是 [1,2,3,4]。新的矩阵是 1 * 4 矩阵,␣
↪用之前的元素值一行一行填充新矩阵。

示例 2:输入:
nums =
[[1,2],
 [3,4]]
r = 2, c = 4
输出:
[[1,2],
 [3,4]]
解释:没有办法将 2 * 2 矩阵转化为 2 * 4 矩阵。 所以输出原矩阵。
注意:
    给定矩阵的宽和高范围在 [1, 100]。
    给定的 r 和 c 都是正数。
```

- 解题思路

```
func matrixReshape(nums [][]int, r int, c int) [][]int {
        row, col := len(nums), len(nums[0])
        if (row*col != r*c) || (row == r && col == c) {
                return nums
        }
        res := make([][]int, r)
        for i := 0; i < len(res); i++ {
                res[i] = make([]int, c)
        }
        for i := 0; i < r*c; i++ {
                res[i/c][i%c] = nums[i/col][i%col]
        }
        return res
}

#
func matrixReshape(nums [][]int, r int, c int) [][]int {
        row, col := len(nums), len(nums[0])
        if (row*col != r*c) || (row == r && col == c) {
                return nums
        }
```

(续下页)

(接上页)

```
        res := make([][]int, 0)
        arr := make([]int, 0)
        count := 0
        for _, num := range nums {
                for _, value := range num {
                        arr = append(arr, value)
                        count++
                        if count == c {
                                res = append(res, arr)
                                arr = []int{}
                                count = 0
                        }
                }
        }
        return res
}
```

16.19 572. 另一个树的子树 (3)

- 题目

```
给定两个非空二叉树 s 和 t，检验 s 中是否包含和 t 具有相同结构和节点值的子树。
s 的一个子树包括 s 的一个节点和这个节点的所有子孙。s 也可以看做它自身的一棵子树。
示例 1:
给定的树 s:
     3
    / \
   4   5
  / \
 1   2

给定的树 t:
   4
  / \
 1   2
返回 true，因为 t 与 s 的一个子树拥有相同的结构和节点值。

示例 2:
给定的树 s:

     3
    / \
```

(续下页)

(接上页)

```
   4   5
  / \
 1   2
    /
   0
给定的树 t:
   4
  / \
 1   2
返回 false。
```

- 解题思路

```
func isSubtree(s *TreeNode, t *TreeNode) bool {
        if s == nil {
                return false
        }
        return isSame(s, t) || isSubtree(s.Left, t) || isSubtree(s.Right, t)
}

func isSame(s *TreeNode, t *TreeNode) bool {
        if s == nil || t == nil{
                return t == s
        }
        return isSame(s.Left, t.Left) && isSame(s.Right, t.Right) && s.Val == t.Val
}

#
func isSubtree(s *TreeNode, t *TreeNode) bool {
        sStr := dfs(s, "")
        tStr := dfs(t, "")
        return strings.Contains(sStr, tStr)
}

func dfs(s *TreeNode, pre string) string {
        if s == nil {
                return pre
        }
        return fmt.Sprintf("#%d%s%s", s.Val, dfs(s.Left, "l"), dfs(s.Right, "r"))
}

#
func isSubtree(s *TreeNode, t *TreeNode) bool {
        sStr := preOrder(s)
```

(续下页)

(接上页)

```
        tStr := preOrder(t)
        return strings.Contains(sStr, tStr)
}

func preOrder(root *TreeNode) string {
        if root == nil {
                return ""
        }
        res := "!"
        stack := make([]*TreeNode,0)
        temp := root
        for {
                for temp != nil{
                        res += strconv.Itoa(temp.Val)
                        res += "!"
                        stack = append(stack, temp)
                        temp = temp.Left
                }
                res += "#!"
                if len(stack) > 0{
                        node := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        temp = node.Right
                }else {
                        break
                }
        }
        return res
}
```

16.20 575. 分糖果 (2)

- 题目

```
给定一个偶数长度的数组，其中不同的数字代表着不同种类的糖果，每一个数字代表一个糖果。
你需要把这些糖果平均分给一个弟弟和一个妹妹。返回妹妹可以获得的最大糖果的种类数。

示例 1:输入: candies = [1,1,2,2,3,3] 输出: 3
解析: 一共有三种种类的糖果，每一种都有两个。
     最优分配方案: 妹妹获得[1,2,3],弟弟也获得[1,2,3]。这样使妹妹获得糖果的种类数最多。

示例 2 : 输入: candies = [1,1,2,3] 输出: 2
```

(续下页)

(接上页)

```
解析：妹妹获得糖果[2,3],弟弟获得糖果[1,1]，妹妹有两种不同的糖果，弟弟只有一种。
这样使得妹妹可以获得的糖果种类数最多。

注意:
    数组的长度为[2, 10,000]，并且确定为偶数。
    数组中数字的大小在范围[-100,000, 100,000]内。
```

- 解题思路

```
func distributeCandies(candies []int) int {
        n := len(candies)
        r := make(map[int]bool, n)
        for _, c := range candies {
                r[c] = true
        }
        return min(len(r), n/2)
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

#
func distributeCandies(candies []int) int {
        length := len(candies)
        half := length / 2
        count := 1
        sort.Ints(candies)
        for i := 1; i < length; i++ {
                if candies[i] != candies[i-1] {
                        count++
                }
        }
        if count >= half {
                return half
        }
        return count
}
```

16.21 581. 最短无序连续子数组 (3)

- 题目

```
给定一个整数数组，你需要寻找一个连续的子数组，如果对这个子数组进行升序排序，那么整个数组都会变为升序
你找到的子数组应是最短的，请输出它的长度。
示例 1:输入: [2, 6, 4, 8, 10, 9, 15] 输出: 5
解释: 你只需要对 [6, 4, 8, 10, 9] 进行升序排序，那么整个表都会变为升序排序。
说明 :
    输入的数组长度范围在 [1, 10,000]。
    输入的数组可能包含重复元素 ，所以升序的意思是<=。
```

- 解题思路

```
func findUnsortedSubarray(nums []int) int {
        length := len(nums)
        left, right := 0, -1
        min, max := nums[length-1], nums[0]
        for i := 1; i < length; i++ {
                if max <= nums[i] {
                        max = nums[i]
                } else {
                        right = i
                }
                j := length - i - 1
                if min >= nums[j] {
                        min = nums[j]
                } else {
                        left = j
                }
        }
        return right - left + 1
}

#
func findUnsortedSubarray(nums []int) int {
        length := len(nums)
        right := -1
        max := nums[0]
        for i := 1; i < length; i++ {
                if max <= nums[i] {
                        max = nums[i]
                } else {
                        right = i
```

(续下页)

(接上页)

```
                }
        }
        if right == 0 {
                // 针对升序，特殊处理
                // 如去掉判断
                // 需要保证left,right初始值满足right-left+1=0
                return 0
        }
        left := 0
        min := nums[length-1]
        for i := length - 2; i >= 0; i-- {
                if min >= nums[i] {
                        min = nums[i]
                } else {
                        left = i
                }
        }
        return right - left + 1
}

#
func findUnsortedSubarray(nums []int) int {
        temp := make([]int,len(nums))
        copy(temp,nums)
        sort.Ints(temp)
        i, j := 0, len(nums)-1
        for i < len(nums) && nums[i] == temp[i]{
                i++
        }
        for i+1 < j && nums[j] == temp[j]{
                j--
        }
        return j-i+1
}
```

16.22 589.N 叉树的前序遍历 (2)

- 题目

```
给定一个 N 叉树，返回其节点值的前序遍历。
例如，给定一个 3叉树 :
返回其前序遍历: [1,3,5,6,2,4]。
说明: 递归法很简单，你可以使用迭代法完成此题吗?
```

- 解题思路

```
var res []int

func preorder(root *Node) []int {
        res = make([]int, 0)
        dfs(root)
        return res
}

func dfs(root *Node) {
        if root == nil {
                return
        }
        res = append(res, root.Val)
        for _, value := range root.Children {
                dfs(value)
        }
}

#
func preorder(root *Node) []int {
        res := make([]int, 0)
        if root == nil {
                return res
        }
        stack := make([]*Node, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                temp := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                res = append(res, temp.Val)
                for i := len(temp.Children) - 1; i >= 0; i-- {
                        stack = append(stack, temp.Children[i])
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

16.23 590.N 叉树的后序遍历 (2)

- 题目

```
给定一个 N 叉树，返回其节点值的后序遍历。

例如，给定一个 3叉树 :

返回其后序遍历: [5,6,3,2,4,1].
说明: 递归法很简单，你可以使用迭代法完成此题吗?
```

- 解题思路

```
var res []int
func postorder(root *Node) []int {
        res = make([]int, 0)
        dfs(root)
        return res
}

func dfs(root *Node) {
        if root == nil {
                return
        }
        for _, value := range root.Children {
                dfs(value)
        }
        res = append(res, root.Val)
}

#
// 后序: (左右)根
// 前序: 根(左右)=>根(右左)=>左右根
func postorder(root *Node) []int {
        res := make([]int, 0)
        if root == nil {
                return res
        }
```

(续下页)

(接上页)

```
        stack := make([]*Node, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                temp := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                res = append(res, temp.Val)
                for i := 0; i < len(temp.Children); i++ {
                        stack = append(stack, temp.Children[i])
                }
        }
        for i := 0; i < len(res)/2; i++ {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
        }
        return res
}
```

16.24 594. 最长和谐子序列 (2)

- 题目

```
和谐数组是指一个数组里元素的最大值和最小值之间的差别正好是1。
现在，给定一个整数数组，你需要在所有可能的子序列中找到最长的和谐子序列的长度。
示例 1:输入: [1,3,2,2,5,2,3,7]输出: 5
原因: 最长的和谐数组是: [3,2,2,2,3].
说明: 输入的数组长度最大不超过20,000.
```

- 解题思路

```
func findLHS(nums []int) int {
        m := make(map[int]int, len(nums))
        for _, n := range nums {
                m[n]++
        }
        res := 0
        for key, value := range m {
                value2, ok := m[key+1]
                if ok {
                        t := value + value2
                        if res < t {
                                res = t
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

#
func findLHS(nums []int) int {
        sort.Ints(nums)
        res := 0
        left := 0
        for i := 0; i < len(nums); i++ {
                for nums[i]-nums[left] > 1 {
                        left++
                }
                if nums[i]-nums[left] == 1 {
                        if res < i-left+1 {
                                res = i - left + 1
                        }
                }
        }
        return res
}
```

16.25 598. 范围求和 II(1)

- 题目

```
给定一个初始元素全部为 0，大小为 m*n 的矩阵 M 以及在 M 上的一系列更新操作。
操作用二维数组表示，其中的每个操作用一个含有两个正整数 a 和 b 的数组表示，
含义是将所有符合 0 <= i < a 以及 0 <= j < b 的元素 M[i][j] 的值都增加 1。
在执行给定的一系列操作后，你需要返回矩阵中含有最大整数的元素个数。

示例 1:输入: m = 3, n = 3operations = [[2,2],[3,3]] 输出: 4
解释: 初始状态, M =
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
执行完操作 [2,2] 后, M =
[[1, 1, 0],
 [1, 1, 0],
 [0, 0, 0]]
执行完操作 [3,3] 后, M =
[[2, 2, 1],
```

(续下页)

(接上页)

```
 [2, 2, 1],
 [1, 1, 1]]

M 中最大的整数是 2, 而且 M 中有4个值为2的元素。因此返回 4。
注意:
    m 和 n 的范围是 [1,40000]。
    a 的范围是 [1,m], b 的范围是 [1,n]。
    操作数目不超过 10000。
```

- 解题思路

```
func maxCount(m int, n int, ops [][]int) int {
        for _, o := range ops {
                m = min(m, o[0])
                n = min(n, o[1])
        }
        return m * n
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}
```

16.26 599. 两个列表的最小索引总和 (2)

- 题目

```
假设Andy和Doris想在晚餐时选择一家餐厅，并且他们都有一个表示最喜爱餐厅的列表，每个餐厅的名字用字符串表
你需要帮助他们用最少的索引和找出他们共同喜爱的餐厅。␣
↪如果答案不止一个，则输出所有答案并且不考虑顺序。
你可以假设总是存在一个答案。

示例 1:输入:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]
输出: ["Shogun"]
解释: 他们唯一共同喜爱的餐厅是“Shogun”。

示例 2:
```

(续下页)

(接上页)

```
输入:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["KFC", "Shogun", "Burger King"]
输出: ["Shogun"]
解释: 他们共同喜爱且具有最小索引和的餐厅是 “Shogun” , 它有最小的索引和1(0+1)。

提示:
    两个列表的长度范围都在 [1, 1000]内。
    两个列表中的字符串的长度将在[1, 30]的范围内。
    下标从0开始, 到列表的长度减1。
    两个列表都没有重复的元素。
```

- 解题思路

```
func findRestaurant(list1 []string, list2 []string) []string {
        if len(list1) > len(list2) {
                list1, list2 = list2, list1
        }
        m2 := make(map[string]int, len(list2))
        for i := range list2 {
                m2[list2[i]] = i
        }
        min := 2000
        res := make([]string, 0, 1000)
        for key, value := range list1 {
                if key2, ok := m2[value]; ok {
                        if min == key+key2 {
                                res = append(res, value)
                        }
                        if min > key+key2 {
                                min = key + key2
                                res = []string{value}
                        }
                }
        }
        return res
}

#
func findRestaurant(list1 []string, list2 []string) []string {
        min := 2000
        res := make([]string, 0, 1000)
        for key1, value1 := range list1 {
                for key2, value2 := range list2{
```

(续下页)

(接上页)

```
                        if value1 == value2{
                                if min == key1+key2 {
                                        res = append(res, value1)
                                }
                                if min > key1+key2 {
                                        min = key1 + key2
                                        res = []string{value1}
                                }
                        }
                }
        }
        return res
}
```

CHAPTER 17

0501-0600-Medium

17.1 503. 下一个更大元素 II(2)

- 题目

```
给定一个循环数组（最后一个元素的下一个元素是数组的第一个元素），输出每个元素的下一个更大元素。
数字 x 的下一个更大的元素是按数组遍历顺序，这个数字之后的第一个比它更大的数，
这意味着你应该循环地搜索它的下一个更大的数。如果不存在，则输出 -1。
示例 1:输入: [1,2,1] 输出: [2,-1,2]
解释: 第一个 1 的下一个更大的数是 2; 数字 2 找不到下一个更大的数;
第二个 1 的下一个最大的数需要循环搜索，结果也是 2。
注意: 输入数组的长度不会超过 10000。
```

- 解题思路

```
func nextGreaterElements(nums []int) []int {
        res := make([]int, len(nums))
        if len(nums) == 0 {
                return res
        }
        for i := 0; i < len(nums); i++ {
                res[i] = -1
        }
        stack := make([]int, 0)
        for i := 0; i < len(nums)*2; i++ {
```

(续下页)

(接上页)

```
                index := i % len(nums)
                for len(stack) > 0 && nums[index] > nums[stack[len(stack)-1]] {
                        if res[stack[len(stack)-1]] == -1 {
                                res[stack[len(stack)-1]] = nums[index]
                        }
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, index)
        }
        return res
}

# 2
func nextGreaterElements(nums []int) []int {
        res := make([]int, len(nums))
        if len(nums) == 0 {
                return res
        }
        stack := make([]int, 0)
        for i := 2*len(nums) - 1; i >= 0; i-- {
                index := i % len(nums)
                for len(stack) > 0 && nums[index] >= stack[len(stack)-1] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        res[index] = -1
                } else {
                        res[index] = stack[len(stack)-1]
                }
                stack = append(stack, nums[index])
        }
        return res
}
```

17.2 508. 出现次数最多的子树元素和 (1)

- 题目

```
给你一个二叉树的根结点，请你找出出现次数最多的子树元素和。
一个结点的「子树元素和」定义为以该结点为根的二叉树上所有结点的元素之和（包括结点本身）。
你需要返回出现次数最多的子树元素和。
如果有多个元素出现的次数相同，返回所有出现次数最多的子树元素和（不限顺序）。
```

(续下页)

(接上页)

```
示例 1: 输入:
  5
 /  \
2   -3
返回[2, -3, 4], 所有的值均只出现一次, 以任意顺序返回所有值。
示例2: 输入:
  5
 /  \
2   -5
返回[2], 只有 2 出现两次, -5 只出现 1 次。
提示: 假设任意子树元素和均可以用 32 位有符号整数表示。
```

- 解题思路

```
var m map[int]int

func findFrequentTreeSum(root *TreeNode) []int {
        m = make(map[int]int)
        res := make([]int, 0)
        dfs(root)
        maxValue := 0
        for k, v := range m {
                if v > maxValue {
                        maxValue = v
                        res = []int{k}
                } else if maxValue == v {
                        res = append(res, k)
                }
        }
        return res
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        sum := root.Val + dfs(root.Left) + dfs(root.Right)
        m[sum]++
        return sum
}
```

17.3 513. 找树左下角的值 (2)

- 题目

```
给定一个二叉树，在树的最后一行找到最左边的值。
示例 1:输入:
    2
   / \
  1   3
输出:1
示例 2:输入:
        1
       / \
      2   3
     /   / \
    4   5   6
       /
      7
输出:7
```

- 解题思路

```
func findBottomLeftValue(root *TreeNode) int {
        res := 0
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                res = queue[0].Val
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                }
                queue = queue[length:]
        }
        return res
}

# 2
var res int
```

(续下页)

(接上页)

```
var maxLevel int

func findBottomLeftValue(root *TreeNode) int {
        res = 0
        maxLevel = -1
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        dfs(root.Left, level+1)
        if level > maxLevel {
                maxLevel = level
                res = root.Val
        }
        dfs(root.Right, level+1)
}
```

17.4 515. 在每个树行中找最大值 (2)

- 题目

```
您需要在二叉树的每一行中找到最大的值。
示例：输入：
          1
         / \
        3   2
       / \   \
      5   3   9
输出：[1, 3, 9]
```

- 解题思路

```
func largestValues(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil {
```

(续下页)

(接上页)

```
                return res
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                maxValue := math.MinInt32
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                        if maxValue < queue[i].Val {
                                maxValue = queue[i].Val
                        }
                }
                res = append(res, maxValue)
                queue = queue[length:]
        }
        return res
}

# 2
var res []int

func largestValues(root *TreeNode) []int {
        res = make([]int, 0)
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        if level >= len(res) {
                res = append(res, math.MinInt32)
        }
```

(续下页)

(接上页)

```
        res[level] = max(res[level], root.Val)
        dfs(root.Left, level+1)
        dfs(root.Right, level+1)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

17.5 516. 最长回文子序列 (3)

- 题目

```
给定一个字符串 s ，找到其中最长的回文子序列，并返回该序列的长度。可以假设 s␣
↪的最大长度为 1000 。
示例 1:输入:"bbbab"输出:4
一个可能的最长回文子序列为 "bbbb"。
示例 2:输入:"cbbd"输出:2
一个可能的最长回文子序列为 "bb"。
提示:
    1 <= s.length <= 1000
    s 只包含小写英文字母
```

- 解题思路

```
func longestPalindromeSubseq(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        n := len(s)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                dp[i][i] = 1
        }
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        if s[i] == s[j] {
                                dp[i][j] = dp[i+1][j-1] + 2 // 内层+2
```

(续下页)

(接上页)

```
                        } else {
                                dp[i][j] = max(dp[i+1][j], dp[i][j-1])
                        }
                }
        }
        return dp[0][n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestPalindromeSubseq(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        n := len(s)
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = 1
        }
        for i := n - 1; i >= 0; i-- {
                prev := 0
                for j := i + 1; j < n; j++ {
                        temp := dp[j]
                        if s[i] == s[j] {
                                dp[j] = prev + 2 // 内层+2
                        } else {
                                dp[j] = max(dp[j], dp[j-1])
                        }
                        prev = temp
                }
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

# 3
var dp [][]int

func longestPalindromeSubseq(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        n := len(s)
        dp = make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        return dfs(s, 0, n-1)
}

func dfs(s string, i, j int) int {
        if i == j {
                return 1
        }
        if i > j {
                return 0
        }
        if dp[i][j] > 0 {
                return dp[i][j]
        }
        if s[i] == s[j] {
                dp[i][j] = dfs(s, i+1, j-1) + 2
        } else {
                dp[i][j] = max(dfs(s, i+1, j), dfs(s, i, j-1))
        }
        return dp[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

17.6 518. 零钱兑换 II(2)

- 题目

```
给定不同面额的硬币和一个总金额。写出函数来计算可以凑成总金额的硬币组合数。假设每一种面额的硬币有无限
示例 1:输入: amount = 5, coins = [1, 2, 5] 输出: 4
解释: 有四种方式可以凑成总金额:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
示例 2:输入: amount = 3, coins = [2] 输出: 0
解释: 只用面额2的硬币不能凑成总金额3。
示例 3:输入: amount = 10, coins = [10]  输出: 1
注意:你可以假设:
    0 <= amount (总金额) <= 5000
    1 <= coin (硬币面额) <= 5000
    硬币种类不超过 500 种
    结果符合 32 位符号整数
```

- 解题思路

```
func change(amount int, coins []int) int {
        n := len(coins)
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, amount+1)
                dp[i][0] = 1 // 金额为0的情况, 只有都不选, 组合情况为1
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= amount; j++ {
                        if j-coins[i-1] >= 0 {
                                dp[i][j] = dp[i-1][j] + dp[i][j-coins[i-1]]
                        } else {
                                dp[i][j] = dp[i-1][j]
                        }
                }
        }
        return dp[n][amount]
}

# 2
func change(amount int, coins []int) int {
        n := len(coins)
```

(续下页)

(接上页)

```
        dp := make([]int, amount+1)
        dp[0] = 1
        for i := 1; i <= n; i++ {
                for j := 1; j <= amount; j++ {
                        if j-coins[i-1] >= 0 {
                                dp[j] = dp[j] + dp[j-coins[i-1]]
                        }
                }
        }
        return dp[amount]
}
```

17.7 519. 随机翻转矩阵 (1)

- 题目

```
题中给出一个 n_rows 行 n_cols 列的二维矩阵，且所有值被初始化为 0。
要求编写一个 flip 函数，均匀随机的将矩阵中的 0 变为 1，并返回该值的位置下标 [row_id,
↪col_id];
同样编写一个 reset 函数，将所有的值都重新置为 0。
尽量最少调用随机函数 Math.random()，并且优化时间和空间复杂度。
注意:1 <= n_rows, n_cols <= 10000
0 <= row.id < n_rows 并且 0 <= col.id < n_cols
当矩阵中没有值为 0 时，不可以调用 flip 函数
调用 flip 和 reset 函数的次数加起来不会超过 1000 次
示例 1: 输入: ["Solution","flip","flip","flip","flip"] [[2,3],[],[],[],[]]
输出: [null,[0,1],[1,2],[1,0],[1,1]]
示例 2: 输入: ["Solution","flip","flip","reset","flip"] [[1,2],[],[],[],[]]
输出: [null,[0,0],[0,1],null,[0,0]]
输入语法解释:
输入包含两个列表: 被调用的子程序和他们的参数。Solution 的构造函数有两个参数，分别为 n_
↪rows 和 n_cols。
flip和 reset 没有参数，参数总会以列表形式给出，哪怕该列表为空
```

- 解题思路

```
type Solution struct {
        m     map[int]bool
        rows  int
        cols  int
        total int
}
```

(续下页)

(接上页)

```
func Constructor(n_rows int, n_cols int) Solution {
        return Solution{
                m:     make(map[int]bool),
                rows:  n_rows,
                cols:  n_cols,
                total: 0,
        }
}

func (this *Solution) Flip() []int {
        if this.total >= this.rows*this.cols {
                return nil
        }
        for {
                index := rand.Intn(this.rows * this.cols)
                if this.m[index] == true {
                        continue
                }
                a, b := index/this.cols, index%this.cols
                this.m[index] = true
                this.total++
                return []int{a, b}
        }
}

func (this *Solution) Reset() {
        this.m = make(map[int]bool)
        this.total = 0
}
```

17.8 522. 最长特殊序列 II(3)

- 题目

```
给定字符串列表，你需要从它们中找出最长的特殊序列。
最长特殊序列定义如下：该序列为某字符串独有的最长子序列（即不能是其他字符串的子序列）。
子序列可以通过删去字符串中的某些字符实现，但不能改变剩余字符的相对顺序。
空序列为所有字符串的子序列，任何字符串为其自身的子序列。
输入将是一个字符串列表，输出是最长特殊序列的长度。如果最长特殊序列不存在，返回 -1 。
示例：输入："aba", "cdc", "eae" 输出：3
提示：所有给定的字符串长度不会超过 10 。
```

(续下页)

(接上页)

给定字符串列表的长度将在 [2, 50] 之间。

- 解题思路

```
func findLUSlength(strs []string) int {
        m := make(map[string]int)
        for i := 0; i < len(strs); i++ {
                total := 1 << (len(strs[i]))
                for j := 0; j < total; j++ {
                        s := ""
                        for k := 0; k < len(strs[i]); k++ {
                                if (j>>k)&1 != 0 {
                                        s = s + string(strs[i][k])
                                }
                        }
                        m[s]++
                }
        }
        res := -1
        for k, v := range m {
                if v == 1 && len(k) > res {
                        res = len(k)
                }
        }
        return res
}

# 2
func findLUSlength(strs []string) int {
        res := -1
        var j int
        for i := 0; i < len(strs); i++ {
                for j = 0; j < len(strs); j++ {
                        if i == j {
                                continue
                        }
                        if judge(strs[i], strs[j]) == true {
                                break
                        }
                }
                if j == len(strs) && len(strs[i]) > res {
                        res = len(strs[i])
                }
        }
```

(续下页)

(接上页)

```
        return res
}

func judge(a, b string) bool {
        j := 0
        for i := 0; i < len(b) && j < len(a); i++ {
                if a[j] == b[i] {
                        j++
                }
        }
        return j == len(a)
}

# 3
func findLUSlength(strs []string) int {
        sort.Slice(strs, func(i, j int) bool {
                return len(strs[i]) > len(strs[j])
        })
        res := -1
        var j int
        for i := 0; i < len(strs); i++ {
                for j = 0; j < len(strs); j++ {
                        if i == j {
                                continue
                        }
                        if judge(strs[i], strs[j]) == true {
                                break
                        }
                }
                if j == len(strs) {
                        return len(strs[i])
                }
        }
        return res
}

func judge(a, b string) bool {
        j := 0
        for i := 0; i < len(b) && j < len(a); i++ {
                if a[j] == b[i] {
                        j++
                }
```

(续下页)

(接上页)

```
        }
        return j == len(a)
}
```

17.9 523. 连续的子数组和 (2)

- 题目

```
给定一个包含 非负数 的数组和一个目标 整数␣
↪k，编写一个函数来判断该数组是否含有连续的子数组，
其大小至少为 2，且总和为 k 的倍数，即总和为 n*k，其中 n 也是一个整数。
示例 1: 输入: [23,2,4,6,7], k = 6 输出: True
解释: [2,4] 是一个大小为 2 的子数组，并且和为 6。
示例 2: 输入: [23,2,6,4,7], k = 6 输出: True
解释: [23,2,6,4,7]是大小为 5 的子数组，并且和为 42。
说明:
    数组的长度不会超过 10,000 。
    你可以认为所有数字总和在 32 位有符号整数范围内。
```

- 解题思路

```
func checkSubarraySum(nums []int, k int) bool {
        if len(nums) == 0 {
                return false
        }
        m := make(map[int]int)
        m[0] = -1
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if k != 0 {
                        sum = sum % k
                }
                if _, ok := m[sum]; ok {
                        // 确保数组大小至少为2
                        if i-m[sum] >= 2 {
                                return true
                        }
                } else {
                        m[sum] = i
                }
        }
```

(续下页)

(接上页)

```
        return false
}

# 2
func checkSubarraySum(nums []int, k int) bool {
        if len(nums) == 0 {
                return false
        }
        arr := make([]int, len(nums))
        arr[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                arr[i] = arr[i-1] + nums[i]
        }
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        sum := arr[j] - arr[i] + nums[i]
                        if sum == k || (k != 0 && sum%k == 0) {
                                return true
                        }
                }
        }
        return false
}
```

17.10 524. 通过删除字母匹配到字典里最长单词 (2)

- 题目

```
给你一个字符串 s 和一个字符串数组 dictionary␣
↪作为字典，找出并返回字典中最长的字符串，该字符串可以通过删除 s 中的某些字符得到。
如果答案不止一个，返回长度最长且字典序最小的字符串。如果答案不存在，则返回空字符串。
示例 1：输入：s = "abpcplea", dictionary = ["ale","apple","monkey","plea"] 输出：
↪"apple"
示例 2：输入：s = "abpcplea", dictionary = ["a","b","c"] 输出："a"
提示：1 <= s.length <= 1000
1 <= dictionary.length <= 1000
1 <= dictionary[i].length <= 1000
s 和 dictionary[i] 仅由小写英文字母组成
```

- 解题思路

```
func findLongestWord(s string, dictionary []string) string {
        sort.Slice(dictionary, func(i, j int) bool {
                if len(dictionary[i]) == len(dictionary[j]) {
                        return dictionary[i] < dictionary[j]
                }
                return len(dictionary[i]) > len(dictionary[j])
        })
        for i := 0; i < len(dictionary); i++ {
                if isSubsequence(dictionary[i], s) {
                        return dictionary[i]
                }
        }
        return ""
}

// leetcode392.判断子序列
func isSubsequence(s string, t string) bool {
        if len(s) > len(t) {
                return false
        }
        i := 0
        j := 0
        for i < len(s) && j < len(t) {
                if s[i] == t[j] {
                        i++
                }
                j++
        }
        return i == len(s)
}

# 2
func findLongestWord(s string, dictionary []string) string {
        res := ""
        for i := 0; i < len(dictionary); i++ {
                if isSubsequence(dictionary[i], s) {
                        if len(dictionary[i]) > len(res) ||
                                (len(dictionary[i]) == len(res) && dictionary[i] <␣
↪res) {
                                res = dictionary[i]
                        }
                }
        }
        return res
```

(续下页)

(接上页)

```
}

// leetcode392.判断子序列
func isSubsequence(s string, t string) bool {
        if len(s) > len(t) {
                return false
        }
        i := 0
        j := 0
        for i < len(s) && j < len(t) {
                if s[i] == t[j] {
                        i++
                }
                j++
        }
        return i == len(s)
}
```

17.11 525. 连续数组 (1)

- 题目

```
给定一个二进制数组, 找到含有相同数量的 0 和 1 的最长连续子数组（的长度）。
示例 1:输入: [0,1] 输出: 2
说明: [0, 1] 是具有相同数量0和1的最长连续子数组。
示例 2:输入: [0,1,0] 输出: 2
说明: [0, 1] (或 [1, 0]) 是具有相同数量0和1的最长连续子数组。
注意:给定的二进制数组的长度不会超过50000。
```

- 解题思路

```
func findMaxLength(nums []int) int {
   res := 0
   m := make(map[int]int)
   m[0] = -1
   total := 0
   for i := 0; i < len(nums); i++ {
           if nums[i] == 0 {
                   total--
           } else {
                   total++
           }
```

(续下页)

(接上页)

```
            if first, ok := m[total]; !ok {
                    m[total] = i
            } else {
                    if i-first > res {
                            res = i - first
                    }
            }
    }
    return res
}
```

17.12 526. 优美的排列 (2)

- 题目

```
假设有从 1 到 N 的N个整数，如果从这N个数字中成功构造出一个数组，
使得数组的第 i位 (1 <= i <= N)␣
↪满足如下两个条件中的一个，我们就称这个数组为一个优美的排列。条件:
第i位的数字能被i整除
i 能被第 i 位上的数字整除
现在给定一个整数 N，请问可以构造多少个优美的排列？
示例1:输入: 2 输出: 2
解释:
第 1 个优美的排列是 [1, 2]:
  第 1 个位置 (i=1) 上的数字是1，1能被 i (i=1) 整除
  第 2 个位置 (i=2) 上的数字是2，2能被 i (i=2) 整除
第 2 个优美的排列是 [2, 1]:
  第 1 个位置 (i=1) 上的数字是2，2能被 i (i=1) 整除
  第 2 个位置 (i=2) 上的数字是1，i (i=2) 能被 1 整除
说明:N 是一个正整数，并且不会超过15。
```

- 解题思路

```
var res [][]int

func countArrangement(n int) int {
        res = make([][]int, 0)
        dfs(n, make([]int, 0), make([]bool, n+1))
        fmt.Println(res)
        return len(res)
}
```

(续下页)

(接上页)

```
func dfs(n int, path []int, visited []bool) {
        if len(path) == n {
                temp := make([]int, len(path))
                copy(temp, path)
                res = append(res, temp)
                return
        }
        for i := 1; i <= n; i++ {
                index := len(path) + 1
                if visited[i] == false && (i%index == 0 || index%i == 0) {
                        visited[i] = true
                        dfs(n, append(path, i), visited)
                        visited[i] = false
                }
        }
}

# 2
var res int

func countArrangement(n int) int {
        res = 0
        dfs(n, make([]int, 0), make([]bool, n+1))
        return res
}

func dfs(n int, path []int, visited []bool) {
        if len(path) == n {
                res++
                return
        }
        for i := 1; i <= n; i++ {
                index := len(path) + 1
                if visited[i] == false && (i%index == 0 || index%i == 0) {
                        visited[i] = true
                        dfs(n, append(path, i), visited)
                        visited[i] = false
                }
        }
}
```

17.13 528. 按权重随机选择 (1)

- 题目

```
给定一个正整数数组w，其中w[i]代表下标 i的权重（下标从 0 开始），
请写一个函数pickIndex，它可以随机地获取下标 i，选取下标 i的概率与w[i]成正比。
例如，对于 w = [1, 3]，挑选下标 0 的概率为 1 / (1 + 3)= 0.25 （即，25%），
而选取下标 1 的概率为 3 / (1 + 3)= 0.75（即，75%）。
也就是说，选取下标 i 的概率为 w[i] / sum(w) 。
示例 1：输入：["Solution","pickIndex"] [[[1]],[]]输出： [null,0]
解释：Solution solution = new Solution([1]);
solution.pickIndex(); // 返回 0，因为数组中只有一个元素，所以唯一的选择是返回下标 0。
示例 2：输入： ["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex
↪"]
[[[1,3]],[],[],[],[],[]]
输出：[null,1,1,1,1,0]
解释：Solution solution = new Solution([1, 3]);
solution.pickIndex(); // 返回 1，返回下标 1，返回该下标概率为 3/4 。
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 0，返回下标 0，返回该下标概率为 1/4 。
由于这是一个随机问题，允许多个答案，因此下列输出都可以被认为是正确的：
[null,1,1,1,1,0]
[null,1,1,1,1,1]
[null,1,1,1,0,0]
[null,1,1,1,0,1]
[null,1,0,1,0,0]
......
诸若此类。
提示：1 <= w.length <= 10000
1 <= w[i] <= 10^5
pickIndex将被调用不超过10000次
```

- 解题思路

```go
type Solution struct {
        nums  []int
        total int
}

func Constructor(w []int) Solution {
        total := 0
        arr := make([]int, len(w)) // 前缀和
```

(续下页)

(接上页)

```
        for i := 0; i < len(w); i++ {
                total = total + w[i]
                arr[i] = total
        }
        return Solution{
                nums:  arr,
                total: total,
        }
}

func (this *Solution) PickIndex() int {
        target := rand.Intn(this.total)
        left, right := 0, len(this.nums)
        for left < right {
                mid := left + (right-left)/2
                if this.nums[mid] <= target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}
```

17.14 529. 扫雷游戏 (2)

- 题目

```
让我们一起来玩扫雷游戏!
给定一个代表游戏板的二维字符矩阵。 'M' 代表一个未挖出的地雷，'E'␣
↪代表一个未挖出的空方块，
'B' 代表没有相邻（上，下，左，右，和所有4个对角线）地雷的已挖出的空白方块，
数字（'1' 到 '8'）表示有多少地雷与这块已挖出的方块相邻，'X' 则表示一个已挖出的地雷。
现在给出在所有未挖出的方块中（'M'或者'E
↪'）的下一个点击位置（行和列索引），根据以下规则，
返回相应位置被点击后对应的面板：
    如果一个地雷（'M'）被挖出，游戏就结束了- 把它改为 'X'。
    如果一个没有相邻地雷的空方块（'E'）被挖出，修改它为（'B'），
    并且所有和其相邻的未挖出方块都应该被递归地揭露。
    如果一个至少与一个地雷相邻的空方块（'E'）被挖出，修改它为数字（'1'到'8
↪'），表示相邻地雷的数量。
    如果在此次点击中，若无更多方块可被揭露，则返回面板。
```

(续下页)

(接上页)

```
示例 1: 输入: [['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'M', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E']]
Click : [3,0]
输出: [['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]
解释:示例 2: 输入:
[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]
Click : [1,2]
输出:
[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'X', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]
解释:
注意: 输入矩阵的宽和高的范围为 [1,50]。
    点击的位置只能是未被挖出的方块 ('M' 或者 'E
↪'), 这也意味着面板至少包含一个可点击的方块。
    输入面板不会是游戏结束的状态 (即有地雷已被挖出) 。
    简单起见, 未提及的规则在这个问题中可被忽略。
    例如, 当游戏结束时你不需要挖出所有地雷, 考虑所有你可能赢得游戏或标记方块的情况。
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0, 1, 1, -1, -1}
var dy = []int{0, 0, -1, 1, 1, -1, 1, -1}

func updateBoard(board [][]byte, click []int) [][]byte {
        x, y := click[0], click[1]
        if board[x][y] == 'M' {
                board[x][y] = 'X'
        } else {
                dfs(board, x, y)
        }
        return board
}

func dfs(board [][]byte, x, y int) {
```

(续下页)

(接上页)

```
        count := 0
        for i := 0; i < 8; i++ {
                newX := dx[i] + x
                newY := dy[i] + y
                if newX < 0 || newX >= len(board) || newY < 0 || newY >=␣
↪len(board[0]) {
                        continue
                }
                if board[newX][newY] == 'M' {
                        count++
                }
        }
        if count > 0 {
                board[x][y] = byte(count + '0')
        } else {
                board[x][y] = 'B'
                for i := 0; i < 8; i++ {
                        newX := dx[i] + x
                        newY := dy[i] + y
                        if newX < 0 || newX >= len(board) ||
                                newY < 0 || newY >= len(board[0]) ||
                                board[newX][newY] != 'E' {
                                continue
                        }
                        dfs(board, newX, newY)
                }
        }
}

# 2
var dx = []int{-1, 1, 0, 0, 1, 1, -1, -1}
var dy = []int{0, 0, -1, 1, 1, -1, 1, -1}

func updateBoard(board [][]byte, click []int) [][]byte {
        x, y := click[0], click[1]
        if board[x][y] == 'M' {
                board[x][y] = 'X'
        } else {
                bfs(board, x, y)
        }
        return board
}
```

(续下页)

(接上页)

```
func bfs(board [][]byte, x, y int) {
        visited := make([][]bool, len(board))
        for i := 0; i < len(board); i++ {
                visited[i] = make([]bool, len(board[i]))
        }
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{x, y})
        visited[x][y] = true
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                count := 0
                a := node[0]
                b := node[1]
                for j := 0; j < 8; j++ {
                        newX := dx[j] + a
                        newY := dy[j] + b
                        if newX < 0 || newX >= len(board) ||
                                newY < 0 || newY >= len(board[0]) ||
                                visited[newX][newY] == true {
                                continue
                        }
                        if board[newX][newY] == 'M' {
                                count++
                        }
                }
                if count > 0 {
                        board[a][b] = byte(count + '0')
                } else {
                        board[a][b] = 'B'
                        for j := 0; j < 8; j++ {
                                newX := dx[j] + a
                                newY := dy[j] + b
                                if newX < 0 || newX >= len(board) ||
                                        newY < 0 || newY >= len(board[0]) ||
                                        board[newX][newY] != 'E' ||
                                        visited[newX][newY] == true {
                                        continue
                                }
                                queue = append(queue, [2]int{newX, newY})
                                visited[newX][newY] = true
                        }
                }
```

(续下页)

(接上页)

```
        }
}
```

17.15 537. 复数乘法 (2)

- 题目

```
给定两个表示复数的字符串。
返回表示它们乘积的字符串。注意，根据定义 i2 = -1 。
示例 1:输入: "1+1i", "1+1i" 输出: "0+2i"
解释: (1 + i) * (1 + i) = 1 + i2 + 2 * i = 2i ，你需要将它转换为 0+2i 的形式。
示例 2:输入: "1+-1i", "1+-1i" 输出: "0+-2i"
解释: (1 - i) * (1 - i) = 1 + i2 - 2 * i = -2i ，你需要将它转换为 0+-2i 的形式。
注意:输入字符串不包含额外的空格。
输入字符串将以a+bi 的形式给出，其中整数 a 和 b 的范围均在 [-100, 100]␣
↪之间。输出也应当符合这种形式。
```

- 解题思路

```
func complexNumberMultiply(a string, b string) string {
        a1, a2 := getValue(a)
        b1, b2 := getValue(b)
        return fmt.Sprintf("%d+%di", a1*b1-a2*b2, a1*b2+a2*b1)
}

func getValue(str string) (a, b int) {
        arr := strings.Split(str, "+")
        a, _ = strconv.Atoi(arr[0])
        b, _ = strconv.Atoi(arr[1][:len(arr[1])-1])
        return a, b
}

# 2
func complexNumberMultiply(a string, b string) string {
        var a1, a2, b1, b2 int
        fmt.Sscanf(a, "%d+%di", &a1, &a2)
        fmt.Sscanf(b, "%d+%di", &b1, &b2)
        return fmt.Sprintf("%d+%di", a1*b1-a2*b2, a1*b2+a2*b1)
}
```

17.16 539. 最小时间差 (2)

- 题目

```
给定一个 24 小时制（小时:分钟 "HH:MM
↪"）的时间列表，找出列表中任意两个时间的最小时间差并以分钟数表示。
示例 1: 输入: timePoints = ["23:59","00:00"] 输出: 1
示例 2: 输入: timePoints = ["00:00","23:59","00:00"]输出: 0
提示: 2 <= timePoints <= 2 * 104
timePoints[i] 格式为 "HH:MM"
```

- 解题思路

```
func findMinDifference(timePoints []string) int {
        m := make(map[int]bool)
        for i := 0; i < len(timePoints); i++ {
                value := getValue(timePoints[i])
                if _, ok := m[value]; ok {
                        return 0
                }
                m[value] = true
        }
        arr := make([]int, 0)
        for k := range m {
                arr = append(arr, k)
        }
        sort.Ints(arr)
        res := math.MaxInt32
        arr = append(arr, arr[0]+1440)
        for i := 1; i < len(arr); i++ {
                if res > arr[i]-arr[i-1] {
                        res = arr[i] - arr[i-1]
                }
        }
        return res
}

func getValue(str string) int {
        hour, _ := strconv.Atoi(str[:2])
        minute, _ := strconv.Atoi(str[3:])
        return hour*60 + minute
}

# 2
```

(续下页)

(接上页)

```
func findMinDifference(timePoints []string) int {
        arr := make([]int, 0)
        for i := 0; i < len(timePoints); i++ {
                value := getValue(timePoints[i])
                arr = append(arr, value)
        }
        sort.Ints(arr)
        res := math.MaxInt32
        arr = append(arr, arr[0]+1440)
        for i := 1; i < len(arr); i++ {
                if res > arr[i]-arr[i-1] {
                        res = arr[i] - arr[i-1]
                }
        }
        return res
}

func getValue(str string) int {
        hour, _ := strconv.Atoi(str[:2])
        minute, _ := strconv.Atoi(str[3:])
        return hour*60 + minute
}
```

17.17 540. 有序数组中的单一元素 (3)

- 题目

```
给定一个只包含整数的有序数组，每个元素都会出现两次，唯有一个数只会出现一次，找出这个数。
示例 1:输入: [1,1,2,3,3,4,4,8,8]输出: 2
示例 2:输入: [3,3,7,7,10,11,11]输出: 10
注意: 您的方案应该在 O(log n)时间复杂度和 O(1)空间复杂度中运行。
```

- 解题思路

```
func singleNonDuplicate(nums []int) int {
        for i := 0; i < len(nums)-1; i = i + 2 {
                if nums[i] != nums[i+1] {
                        return nums[i]
                }
        }
        return nums[len(nums)-1]
}
```

(续下页)

(接上页)

```
# 2
func singleNonDuplicate(nums []int) int {
        n := len(nums)
        left, right := 0, n-1
        for left < right {
                mid := left + (right-left)/2
                if mid%2 == 1 {
                        mid--
                }
                if nums[mid] == nums[mid+1] {
                        left = mid + 2
                } else {
                        right = mid
                }
        }
        return nums[left]
}

# 3
func singleNonDuplicate(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                res = res ^ nums[i]
        }
        return res
}
```

17.18 542.01 矩阵 (3)

- 题目

```
给定一个由 0 和 1 组成的矩阵，找出每个元素到最近的 0 的距离。
两个相邻元素间的距离为 1 。
示例 1: 输入:
0 0 0
0 1 0
0 0 0
输出:
0 0 0
0 1 0
0 0 0
```

(续下页)

(接上页)

```
示例 2: 输入:
0 0 0
0 1 0
1 1 1
输出:
0 0 0
0 1 0
1 2 1
注意:
    给定矩阵的元素个数不超过 10000。
    给定矩阵中至少有一个元素是 0。
    矩阵中的元素只在四个方向上相邻: 上、下、左、右。
```

- 解题思路

```
func updateMatrix(matrix [][]int) [][]int {
        n := len(matrix)
        m := len(matrix[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 1 {
                                dp[i][j] = math.MaxInt32 / 10
                                if i > 0 {
                                        dp[i][j] = min(dp[i][j], dp[i-1][j]+1)
                                }
                                if j > 0 {
                                        dp[i][j] = min(dp[i][j], dp[i][j-1]+1)
                                }
                        } else {
                                dp[i][j] = 0
                        }
                }
        }
        for i := n - 1; i >= 0; i-- {
                for j := m - 1; j >= 0; j-- {
                        if dp[i][j] > 1 {
                                if i < n-1 {
                                        dp[i][j] = min(dp[i][j], dp[i+1][j]+1)
                                }
                                if j < m-1 {
                                        dp[i][j] = min(dp[i][j], dp[i][j+1]+1)
                                }
```

(续下页)

(接上页)

```
                        }
                }
        }
        return dp
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func updateMatrix(matrix [][]int) [][]int {
        n := len(matrix)
        m := len(matrix[0])
        queue := make([][2]int, 0)
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 0 {
                                queue = append(queue, [2]int{i, j})
                        } else {
                                matrix[i][j] = -1
                        }
                }
        }
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                for i := 0; i < 4; i++ {
                        x := node[0] + dx[i]
                        y := node[1] + dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y] == -1 {
                                matrix[x][y] = matrix[node[0]][node[1]] + 1
                                queue = append(queue, [2]int{x, y})
                        }
                }
        }
        return matrix
```

(续下页)

(接上页)

```
}

# 3
func updateMatrix(matrix [][]int) [][]int {
        n := len(matrix)
        m := len(matrix[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 1 {
                                matrix[i][j] = math.MaxInt32 / 10
                                if i > 0 {
                                        matrix[i][j] = min(matrix[i][j], matrix[i-
↪1][j]+1)
                                }
                                if j > 0 {
                                        matrix[i][j] = min(matrix[i][j], matrix[i][j-
↪1]+1)
                                }
                        } else {
                                matrix[i][j] = 0
                        }
                }
        }
        for i := n - 1; i >= 0; i-- {
                for j := m - 1; j >= 0; j-- {
                        if matrix[i][j] > 1 {
                                if i < n-1 {
                                        matrix[i][j] = min(matrix[i][j],␣
↪matrix[i+1][j]+1)
                                }
                                if j < m-1 {
                                        matrix[i][j] = min(matrix[i][j],␣
↪matrix[i][j+1]+1)
                                }
                        }
                }
        }
        return matrix
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}
```

17.19 547. 省份数量 (3)

- 题目

```
有 n 个城市，其中一些彼此相连，另一些没有相连。如果城市 a 与城市 b 直接相连，
且城市 b 与城市 c 直接相连，那么城市 a 与城市 c 间接相连。
省份 是一组直接或间接相连的城市，组内不含其他没有相连的城市。
给你一个 n x n 的矩阵 isConnected ，其中 isConnected[i][j] = 1 表示第 i 个城市和第 j␣
↪个城市直接相连，
而 isConnected[i][j] = 0 表示二者不直接相连。
返回矩阵中 省份 的数量。
示例 1：输入：isConnected = [[1,1,0],[1,1,0],[0,0,1]]输出：2
示例 2：输入：isConnected = [[1,0,0],[0,1,0],[0,0,1]] 输出：3
提示：1 <= n <= 200
n == isConnected.length
n == isConnected[i].length
isConnected[i][j] 为 1 或 0
isConnected[i][i] == 1
isConnected[i][j] == isConnected[j][i]
```

- 解题思路

```
func findCircleNum(M [][]int) int {
        n := len(M)
        fa = Init(n)
        count = n
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if M[i][j] == 1 {
                                union(i, j)
                        }
                }
        }
        return getCount()
}

var fa []int
var count int
```

(续下页)

(接上页)

```
// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        count = n
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
        }
}

func query(i, j int) bool {
        return find(i) == find(j)
}

func getCount() int {
        return count
}

# 2
var arr []bool

func findCircleNum(M [][]int) int {
        n := len(M)
```

(续下页)

(接上页)

```
        arr = make([]bool, n)
        res := 0
        for i := 0; i < n; i++ {
                if arr[i] == false {
                        dfs(M, i)
                        res++
                }
        }
        return res
}

func dfs(M [][]int, index int) {
        for i := 0; i < len(M); i++ {
                if arr[i] == false && M[index][i] == 1 {
                        arr[i] = true
                        dfs(M, i)
                }
        }
}

# 3
func findCircleNum(M [][]int) int {
        n := len(M)
        arr := make([]bool, n)
        res := 0
        queue := make([]int, 0)
        for i := 0; i < n; i++ {
                if arr[i] == false {
                        queue = append(queue, i)
                        for len(queue) > 0 {
                                node := queue[0]
                                queue = queue[1:]
                                arr[node] = true
                                for j := 0; j < n; j++ {
                                        if M[node][j] == 1 && arr[j] == false {
                                                queue = append(queue, j)
                                        }
                                }
                        }
                        res++
                }
        }
        return res
}
```

17.20 553. 最优除法 (1)

- 题目

```
给定一组正整数，相邻的整数之间将会进行浮点除法操作。例如，[2,3,4] -> 2 / 3 / 4 。
但是，你可以在任意位置添加任意数目的括号，来改变算数的优先级。
你需要找出怎么添加括号，才能得到最大的结果，并且返回相应的字符串格式的表达式。你的表达式不应该含有冗
示例：输入：[1000,100,10,2] 输出："1000/(100/10/2)"
解释:1000/(100/10/2) = 1000/((100/10)/2) = 200
但是，以下加粗的括号 "1000/((100/10)/2)" 是冗余的，
因为他们并不影响操作的优先级，所以你需要返回 "1000/(100/10/2)"。
其他用例：
1000/(100/10)/2 = 50
1000/(100/(10/2)) = 50
1000/100/10/2 = 0.5
1000/100/(10/2) = 2
说明:输入数组的长度在 [1, 10] 之间。
数组中每个元素的大小都在 [2, 1000] 之间。
每个测试用例只有一个最优除法解。
```

- 解题思路

```
func optimalDivision(nums []int) string {
        res := make([]string,0)
        for i := 0; i < len(nums); i++{
                res = append(res, strconv.Itoa(nums[i]))
        }
        if len(res) < 3{
                return strings.Join(res, "/")
        }
        return res[0]+"/("+strings.Join(res[1:],"/") + ")"
}
```

17.21 554. 砖墙 (1)

- 题目

```
你的面前有一堵矩形的、由多行砖块组成的砖墙。这些砖块高度相同但是宽度不同。
你现在要画一条自顶向下的、穿过最少砖块的垂线。
砖墙由行的列表表示。 每一行都是一个代表从左至右每块砖的宽度的整数列表。
如果你画的线只是从砖块的边缘经过，就不算穿过这块砖。
你需要找出怎样画才能使这条线穿过的砖块数量最少，并且返回穿过的砖块数量。
你不能沿着墙的两个垂直边缘之一画线，这样显然是没有穿过一块砖的。
```

(续下页)

(接上页)

```
示例：输入：[[1,2,2,1],
      [3,1,2],
      [1,3,2],
      [2,4],
      [3,1,2],
      [1,3,1,1]]
输出：2
解释：提示：每一行砖块的宽度之和应该相等，并且不能超过 INT_MAX。
每一行砖块的数量在[1,10,000] 范围内，墙的高度在[1,10,000] 范围内，总的砖块数量不超过␣
↪20,000。
```

- 解题思路

```
func leastBricks(wall [][]int) int {
        maxCount := 0
        m := make(map[int]int)
        for i := 0; i < len(wall); i++ {
                index := 0
                for j := 0; j < len(wall[i])-1; j++ {
                        index = index + wall[i][j]
            m[index]++ // 保留去除开头和结尾的位置(空隙地方)
                        if maxCount <= m[index] {
                                maxCount = m[index]
                        }
                }
        }
        return len(wall) - maxCount
}
```

17.22 556. 下一个更大元素 III(2)

- 题目

```
给你一个正整数n ，请你找出符合条件的最小整数，其由重新排列␣
↪n中存在的每位数字组成，并且其值大于 n 。
如果不存在这样的正整数，则返回 -1 。
注意 ，返回的整数应当是一个 32 位整数 ，如果存在满足题意的答案，但不是 32 位整数␣
↪，同样返回 -1 。
示例 1：输入：n = 12 输出：21
示例 2：输入：n = 21 输出：-1
提示：1 <= n <= 231 - 1
```

- 解题思路

```
func nextGreaterElement(n int) int {
        arr := make([]int, 0)
        for n > 0 {
                arr = append(arr, n%10)
                n = n / 10
        }
        reverse(arr, 0, len(arr)-1)
        arr = nextPermutation(arr)
        if arr == nil {
                return -1
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                res = res*10 + arr[i]
                if res > math.MaxInt32 {
                        return -1
                }
        }
        return res
}

// leetcode31.下一个排列
func nextPermutation(nums []int) []int {
        n := len(nums)
        left := n - 2
        // 以12385764为例，从后往前找到5<7 的升序情况，目标值为左边的数5
        for left >= 0 && nums[left] >= nums[left+1] {
                left--
        }
        if left >= 0 { // 存在升序的情况
                right := n - 1
                // 从后往前，找到第一个大于目标值的数，如6>5，然后交换
                for right >= 0 && nums[right] <= nums[left] {
                        right--
                }
                nums[left], nums[right] = nums[right], nums[left]
        } else {
                return nil
        }
        reverse(nums, left+1, n-1)
        return nums
}

func reverse(nums []int, left, right int) {
```

(续下页)

(接上页)

```
        for left < right {
                nums[left], nums[right] = nums[right], nums[left]
                left++
                right--
        }
}

# 2
func nextGreaterElement(n int) int {
        nums := []byte(strconv.Itoa(n))
        length := len(nums)
        left := length - 2
        // 以12385764为例，从后往前找到5<7 的升序情况，目标值为左边的数5
        for left >= 0 && nums[left] >= nums[left+1] {
                left--
        }
        if left >= 0 { // 存在升序的情况
                right := length - 1
                // 从后往前，找到第一个大于目标值的数，如6>5，然后交换
                for right >= 0 && nums[right] <= nums[left] {
                        right--
                }
                nums[left], nums[right] = nums[right], nums[left]
        } else {
                return -1
        }
        left = left + 1
        right := length - 1
        for left < right {
                nums[left], nums[right] = nums[right], nums[left]
                left++
                right--
        }
        res, _ := strconv.Atoi(string(nums))
        if res > math.MaxInt32 {
                return -1
        }
        return res
}
```

17.23 558. 四叉树交集 (1)

- 题目

```
二进制矩阵中的所有元素不是 0 就是 1 。
给你两个四叉树，quadTree1 和 quadTree2。其中 quadTree1 表示一个 n * n 二进制矩阵，
而 quadTree2 表示另一个 n * n 二进制矩阵。
请你返回一个表示 n * n 二进制矩阵的四叉树，它是 quadTree1 和 quadTree2␣
↪所表示的两个二进制矩阵进行 按位逻辑或运算 的结果。
注意，当 isLeaf 为 False 时，你可以把 True 或者 False␣
↪赋值给节点，两种值都会被判题机制 接受 。
四叉树数据结构中，每个内部节点只有四个子节点。此外，每个节点都有两个属性：
val：储存叶子结点所代表的区域的值。1 对应 True，0 对应 False；
isLeaf: 当这个节点是一个叶子结点时为 True，如果它有 4 个子节点则为 False 。
class Node {
    public boolean val;
  public boolean isLeaf;
  public Node topLeft;
  public Node topRight;
  public Node bottomLeft;
  public Node bottomRight;
}
我们可以按以下步骤为二维区域构建四叉树：
如果当前网格的值相同（即，全为 0 或者全为 1），将 isLeaf 设为 True ，
将 val 设为网格相应的值，并将四个子节点都设为 Null 然后停止。
如果当前网格的值不同，将 isLeaf 设为 False， 将 val␣
↪设为任意值，然后如下图所示，将当前网格划分为四个子网格。
使用适当的子网格递归每个子节点。
如果你想了解更多关于四叉树的内容，可以参考 wiki 。
四叉树格式：
输出为使用层序遍历后四叉树的序列化形式，其中 null 表示路径终止符，其下面不存在节点。
它与二叉树的序列化非常相似。唯一的区别是节点以列表形式表示 [isLeaf, val] 。
如果 isLeaf 或者 val 的值为 True ，则表示它在列表[isLeaf, val] 中的值为 1 ；
如果 isLeaf 或者 val 的值为 False ，则表示值为 0 。
示例 1：输入：quadTree1 = [[0,1],[1,1],[1,1],[1,0],[1,0]],
quadTree2 = [[0,1],[1,1],[0,1],[1,1],[1,0],null,null,null,null,[1,0],[1,0],[1,1],[1,
↪1]]
输出：[[0,0],[1,1],[1,1],[1,1],[1,0]]
解释：quadTree1 和 quadTree2 如上所示。由四叉树所表示的二进制矩阵也已经给出。
如果我们对这两个矩阵进行按位逻辑或运算，则可以得到下面的二进制矩阵，由一个作为结果的四叉树表示。
注意，我们展示的二进制矩阵仅仅是为了更好地说明题意，你无需构造二进制矩阵来获得结果四叉树。
示例 2：输入：quadTree1 = [[1,0]], quadTree2 = [[1,0]] 输出：[[1,0]]
解释：两个数所表示的矩阵大小都为 1*1，值全为 0
结果矩阵大小为 1*1，值全为 0 。
```

(续下页)

(接上页)

```
示例 3：输入：quadTree1 = [[0,0],[1,0],[1,0],[1,1],[1,1]] , quadTree2 = [[0,0],[1,1],
↪[1,1],[1,0],[1,1]]
输出：[[1,1]]
示例 4：输入：quadTree1 = [[0,0],[1,1],[1,0],[1,1],[1,1]],
quadTree2 = [[0,0],[1,1],[0,1],[1,1],[1,1],null,null,null,null,[1,1],[1,0],[1,0],[1,
↪1]]
输出：[[0,0],[1,1],[0,1],[1,1],[1,1],null,null,null,null,[1,1],[1,0],[1,0],[1,1]]
示例 5：输入：quadTree1 = [[0,1],[1,0],[0,1],[1,1],[1,0],null,null,null,null,[1,0],[1,
↪0],[1,1],[1,1]],
quadTree2 = [[0,1],[0,1],[1,0],[1,1],[1,0],[1,0],[1,0],[1,1],[1,1]]
输出：[[0,0],[0,1],[0,1],[1,1],[1,0],[1,0],[1,0],[1,1],[1,1],[1,0],[1,0],[1,1],[1,1]]
提示：quadTree1 和 quadTree2 都是符合题目要求的四叉树，每个都代表一个 n * n 的矩阵。
n == 2^x ，其中 0 <= x <= 9.
```

- 解题思路

```
func intersect(quadTree1 *Node, quadTree2 *Node) *Node {
        res := &Node{}
        if quadTree1.IsLeaf == true { // 叶子节点
                if quadTree1.Val == true {
                        return quadTree1
                }
                return quadTree2
        }
        if quadTree2.IsLeaf == true { // 叶子节点
                if quadTree2.Val == true {
                        return quadTree2
                }
                return quadTree1
        }
        tL := intersect(quadTree1.TopLeft, quadTree2.TopLeft)
        tR := intersect(quadTree1.TopRight, quadTree2.TopRight)
        bL := intersect(quadTree1.BottomLeft, quadTree2.BottomLeft)
        bR := intersect(quadTree1.BottomRight, quadTree2.BottomRight)
        // 叶子节点判断
        if tL.IsLeaf == true && tR.IsLeaf == true && bL.IsLeaf == true && bR.IsLeaf
↪== true &&
                tL.Val == tR.Val && tR.Val == bL.Val && bL.Val == bR.Val {
                res.IsLeaf = true
                res.Val = tL.Val // 4个值都相同
                return res
        }
        res.TopLeft = tL
        res.TopRight = tR
```

(续下页)

(接上页)

```
        res.BottomLeft = bL
        res.BottomRight = bR
        res.Val = false
        res.IsLeaf = false
        return res
}
```

17.24 560. 和为 K 的子数组 (4)

- 题目

```
给定一个整数数组和一个整数 k，你需要找到该数组中和为 k 的连续的子数组的个数。
示例 1 :输入:nums = [1,1,1], k = 2 输出: 2 , [1,1] 与 [1,1] 为两种不同的情况。
说明:数组的长度为 [1, 20,000]。
```

- 解题思路

```
func subarraySum(nums []int, k int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum = sum + nums[j]
                        if sum == k {
                                res++
                        }
                }
        }
        return res
}

# 2
func subarraySum(nums []int, k int) int {
        if len(nums) == 0 {
                return 0
        }
        res := 0
        arr := make([]int, len(nums)+1)
        arr[0] = 0
        for i := 1; i <= len(nums); i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
```

(续下页)

(接上页)

```
        for i := 0; i <= len(nums); i++ {
                for j := 0; j < i; j++ {
                        if arr[i]-arr[j] == k {
                                res++
                        }
                }
        }
        return res
}

# 3
func subarraySum(nums []int, k int) int {
        res := 0
        m := make(map[int]int)
        m[0] = 1 // 保证第一个k的存在
        sum := 0
        // sum[i:j]= sum[0:j]-sum[0:i], 把sum[i:j]设为k,
        // 于是可以转化为sum[0:j]-k=sum[0:i]
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if _, ok := m[sum-k]; ok {
                        res = res + m[sum-k]
                }
                m[sum]++
        }
        return res
}

# 4
func subarraySum(nums []int, k int) int {
        res := 0
        m := make(map[int][]int)
        m[0] = []int{-1} // 保证第一个k的存在
        sum := 0
        // sum[i:j]= sum[0:j]-sum[0:i], 把sum[i:j]设为k,
        // 于是可以转化为sum[0:j]-k=sum[0:i]
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if _, ok := m[sum-k]; ok {
                        res = res + len(m[sum-k])
            // 输出满足条件的子数组下标
                        // for _, v := range m[sum-k] {
                        //         fmt.Println(v+1, i)
```

(续下页)

(接上页)

```
                        // }
                }
                m[sum] = append(m[sum], i)
        }
        return res
}
```

17.25 565. 数组嵌套 (4)

- 题目

```
索引从0开始长度为N的数组A，包含0到N - 1的所有整数。找到最大的集合S并返回其大小，
其中 S[i] = {A[i], A[A[i]], A[A[A[i]]], ... }且遵守以下的规则。
假设选择索引为i的元素A[i]为S的第一个元素，S的下一个元素应该是A[A[i]],
之后是A[A[A[i]]]... 以此类推，不断添加直到S出现重复的元素。
示例1:输入: A = [5,4,0,3,1,6,2] 输出: 4
解释:  A[0] = 5, A[1] = 4, A[2] = 0, A[3] = 3, A[4] = 1, A[5] = 6, A[6] = 2.
其中一种最长的 S[K]:
S[0] = {A[0], A[5], A[6], A[2]} = {5, 6, 2, 0}
提示：N是[1, 20,000]之间的整数。
A中不含有重复的元素。
A中的元素大小在[0, N-1]之间。
```

- 解题思路

```
func arrayNesting(nums []int) int {
        m := make(map[int]bool)
        res := 0
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] == true {
                        continue
                }
                count := 0
                cur := i
                for {
                        count++
                        m[cur] = true
                        cur = nums[cur]
                        if cur == i {
                                break
                        }
                }
```

(续下页)

(接上页)

```
                if count > res {
                        res = count
                }
        }
        return res
}

# 2
func arrayNesting(nums []int) int {
        m := make(map[int]bool)
        res := 0
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] == true {
                        continue
                }
                count := 0
                cur := i
                for {
                        count++
                        m[cur] = true
                        cur = nums[cur]
                        if m[cur] == true {
                                break
                        }
                }
                if count > res {
                        res = count
                }
        }
        return res
}

# 3
func arrayNesting(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                count := 1
                for nums[i] != i {
                        count++
                        nums[i], nums[nums[i]] = nums[nums[i]], nums[i]
                }
                if count > res {
                        res = count
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 4
func arrayNesting(nums []int) int {
        res := 0
        fa = Init(len(nums))
        for i := 0; i < len(nums); i++ {
                union(i, nums[i])
        }
        m := make(map[int]int)
        for i := 0; i < len(fa); i++ {
                m[find(i)]++
        }
        for _, v := range m {
                if v > res {
                        res = v
                }
        }
        return res
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
```

(续下页)

(接上页)

```
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
        }
}
```

17.26 567. 字符串的排列 (2)

- 题目

```
给定两个字符串 s1 和 s2，写一个函数来判断 s2 是否包含 s1 的排列。
换句话说，第一个字符串的排列之一是第二个字符串的子串。
示例1:输入: s1 = "ab" s2 = "eidbaooo" 输出: True
解释: s2 包含 s1 的排列之一 ("ba").
示例2:输入: s1= "ab" s2 = "eidboaoo" 输出: False
注意: 输入的字符串只包含小写字母
两个字符串的长度都在 [1, 10,000] 之间
```

- 解题思路

```
func checkInclusion(s1 string, s2 string) bool {
        if len(s1) > len(s2) {
                return false
        }
        arr1, arr2 := [26]int{}, [26]int{}
        for i := 0; i < len(s1); i++ {
                arr1[s1[i]-'a']++
                arr2[s2[i]-'a']++
        }
        for i := 0; i < len(s2)-len(s1); i++ {
                if arr1 == arr2 {
                        return true
                }
                arr2[s2[i]-'a']--
                arr2[s2[i+len(s1)]-'a']++
        }
        return arr1 == arr2
}
```

(续下页)

(接上页)

```
# 2
func checkInclusion(s1 string, s2 string) bool {
        if len(s1) > len(s2) {
                return false
        }
        m1, m2 := make(map[byte]int), make(map[byte]int)
        for i := 0; i < len(s1); i++ {
                m1[s1[i]-'a']++
                m2[s2[i]-'a']++
        }
        for i := 0; i < len(s2)-len(s1); i++ {
                if compare(m1, m2) {
                        return true
                }
                m2[s2[i]-'a']--
                if m2[s2[i]-'a'] == 0 {
                        delete(m2, s2[i]-'a')
                }
                m2[s2[i+len(s1)]-'a']++
        }
        return compare(m1, m2)
}

func compare(m1, m2 map[byte]int) bool {
        if len(m1) != len(m2) {
                return false
        }
        for k := range m1 {
                if m2[k] != m1[k] {
                        return false
                }
        }
        return true
}
```

17.27 576. 出界的路径数 (2)

- 题目

```
给定一个 m × n 的网格和一个球。球的起始坐标为(i,j)，你可以将球移到相邻的单元格内，
或者往上、下、左、右四个方向上移动使球穿过网格边界。但是，你最多可以移动N次。
找出可以将球移出边界的路径数量。答案可能非常大，返回 结果 mod 109+ 7 的值。
示例 1：输入：m = 2, n = 2, N = 2, i = 0, j = 0 输出：6
解释：
示例 2：输入：m = 1, n = 3, N = 3, i = 0, j = 1 输出：12
解释：
说明:球一旦出界，就不能再被移动回网格内。
网格的长度和高度在 [1,50] 的范围内。
N 在 [0,50] 的范围内。
```

- 解题思路

```
var dp [60][60][60]int
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}
var mod = 1000000007

func findPaths(m int, n int, N int, i int, j int) int {
        dp = [60][60][60]int{}
        for i := 0; i < 60; i++ {
                for j := 0; j < 60; j++ {
                        for k := 0; k < 60; k++ {
                                dp[i][j][k] = -1
                        }
                }
        }
        return dfs(m, n, i+1, j+1, N) // 下标取正
}

func dfs(m, n, i, j, k int) int {
        if k < 0 { // 次数够了
                return 0
        }
        if i < 1 || i > m || j < 1 || j > n { // 出界次数+1
                return 1
        }
        if dp[i][j][k] != -1 {
                return dp[i][j][k]
        }
```

(续下页)

(接上页)

```
        dp[i][j][k] = 0
        for a := 0; a < 4; a++ { // 上下左右4个方向
                x := i + dx[a]
                y := j + dy[a]
                dp[i][j][k] = (dp[i][j][k] + dfs(m, n, x, y, k-1)) % mod
        }
        return dp[i][j][k]
}

# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}
var mod = 1000000007

func findPaths(m int, n int, N int, i int, j int) int {
        dp := [60][60][60]int{}
        for k := 1; k <= N; k++ {
                for a := 0; a < m; a++ {
                        for b := 0; b < n; b++ {
                                for i := 0; i < 4; i++ {
                                        x := a + dx[i]
                                        y := b + dy[i]
                                        if x < 0 || x >= m || y < 0 || y >= n { //␣
↪出界次数+1
                                                dp[a][b][k]++
                                        } else {
                                                dp[a][b][k] = (dp[a][b][k] +␣
↪dp[x][y][k-1]) % mod
                                        }
                                }
                        }
                }
        }
        return dp[i][j][N] % mod
}
```

17.28 583. 两个字符串的删除操作 (3)

- 题目

```
给定两个单词word1和word2，找到使得word1和word2相同所需的最小步数，
每步可以删除任意一个字符串中的一个字符。
示例：输入："sea", "eat" 输出：2
解释：第一步将"sea"变为"ea"，第二步将"eat"变为"ea"
提示：给定单词的长度不超过500。
给定单词中的字符只含有小写字母。
```

- 解题思路

```
func minDistance(word1 string, word2 string) int {
        a, b := len(word1), len(word2)
        // 最长公共子序列
        dp := make([][]int, a+1)
        for i := 0; i <= a; i++ {
                dp[i] = make([]int, b+1)
        }
        for i := 1; i <= a; i++ {
                for j := 1; j <= b; j++ {
                        if word1[i-1] == word2[j-1] {
                                dp[i][j] = dp[i-1][j-1] + 1
                        } else {
                                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
                        }
                }
        }
        return a + b - 2*dp[a][b]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minDistance(word1 string, word2 string) int {
        a, b := len(word1), len(word2)
        dp := make([][]int, a+1)
        for i := 0; i <= a; i++ {
```

(续下页)

(接上页)

```
                dp[i] = make([]int, b+1)
                dp[i][0] = i
        }
        for i := 0; i <= b; i++ {
                dp[0][i] = i
        }
        for i := 1; i <= a; i++ {
                for j := 1; j <= b; j++ {
                        if word1[i-1] == word2[j-1] {
                                dp[i][j] = dp[i-1][j-1]
                        } else {
                                dp[i][j] = min(dp[i][j-1], dp[i-1][j]) + 1
                        }
                }
        }
        return dp[a][b]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var m [][]int

func minDistance(word1 string, word2 string) int {
        a, b := len(word1), len(word2)
        m = make([][]int, a+1)
        for i := 0; i <= a; i++ {
                m[i] = make([]int, b+1)
                for j := 0; j <= b; j++ {
                        m[i][j] = -1
                }
        }
        total := dfs(word1, word2, 0, 0)
        return a + b - 2*total
}

func dfs(word1 string, word2 string, i, j int) int {
        if len(word1) == i || len(word2) == j {
```

(续下页)

(接上页)

```
                return 0
        }
        if m[i][j] > -1 {
                return m[i][j]
        }
        if word1[i] == word2[j] {
                m[i][j] = dfs(word1, word2, i+1, j+1) + 1
        } else {
                m[i][j] = max(dfs(word1, word2, i, j+1), dfs(word1, word2, i+1, j))
        }
        return m[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

17.29 592. 分数加减运算 (1)

- 题目

```
给定一个表示分数加减运算表达式的字符串，你需要返回一个字符串形式的计算结果。这个结果应该是不可约分的
如果最终结果是一个整数，例如2，你需要将它转换成分数形式，其分母为1。所以在上述例子中,␣
↪2应该被转换为2/1。
示例1:输入:"-1/2+1/2" 输出: "0/1"
示例 2:输入:"-1/2+1/2+1/3" 输出: "1/3"
示例 3:输入:"1/3-1/2" 输出: "-1/6"
示例 4:输入:"5/3+1/3" 输出: "2/1"
说明:输入和输出字符串只包含'0' 到'9'的数字，以及'/', '+' 和'-'。
输入和输出分数格式均为±分子/分母。如果输入的第一个分数或者输出的分数是正数，则'+
↪'会被省略掉。
输入只包含合法的最简分数，每个分数的分子与分母的范围是[1,
↪10]。如果分母是1，意味着这个分数实际上是一个整数。
输入的分数个数范围是 [1,10]。
最终结果的分子与分母保证是 32 位整数范围内的有效整数。
```

- 解题思路

```
func fractionAddition(expression string) string {
        s := strings.ReplaceAll(expression, "-", "+-") // 替换-为+-
        s = strings.ReplaceAll(s, "/", "+")            // 替换/为+
        temp := strings.Split(s, "+")                  // 根据+切割
        arr := make([]int, 0)
        for i := 0; i < len(temp); i++ {
                if temp[i] == "" { // 第一个数的分子如果为负数，+切割会为空
                        continue
                }
                value, _ := strconv.Atoi(temp[i])
                arr = append(arr, value)
        }
        a, b := 0, 1
        for i := 0; i < len(arr); i = i + 2 { // 遍历每个数：分子+分母
                c, d := arr[i], arr[i+1]
                // a/b+c/d=ad/bd+bc/bd=(ad+bc)/bd
                a = a*d + b*c
                b = b * d
                g := gcd(a, b)
                if g < 0 { // 约分为负数，需要转换为正数，这样确保分母一直为正数
                        g = -g
                }
                a, b = a/g, b/g
        }
        return fmt.Sprintf("%d/%d", a, b)
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}
```

17.30 593. 有效的正方形 (2)

- 题目

```
给定二维空间中四点的坐标，返回四点是否可以构造一个正方形。
一个点的坐标（x，y）由一个有两个整数的整数数组表示。
示例:输入: p1 = [0,0], p2 = [1,1], p3 = [1,0], p4 = [0,1] 输出: True
注意:所有输入整数都在 [-10000, 10000] 范围内。
```

(续下页)

(接上页)

一个有效的正方形有四个等长的正长和四个等角（90度角）。
输入点没有顺序。

- 解题思路

```
func validSquare(p1 []int, p2 []int, p3 []int, p4 []int) bool {
        m := make(map[int]int)
        m[getDis(p1, p2)]++
        m[getDis(p1, p3)]++
        m[getDis(p1, p4)]++
        m[getDis(p2, p3)]++
        m[getDis(p2, p4)]++
        m[getDis(p3, p4)]++
        a, b := 0, 0
        for k, v := range m {
                if v == 2 {
                        a = k
                } else if v == 4 {
                        b = k
                } else {
                        return false
                }
        }
        return len(m) == 2 && a == 2*b
}

func getDis(a, b []int) int {
        return (a[0]-b[0])*(a[0]-b[0]) + (a[1]-b[1])*(a[1]-b[1])
}

# 2
func validSquare(p1 []int, p2 []int, p3 []int, p4 []int) bool {
        arr := make([]int, 0)
        arr = append(arr, getDis(p1, p2))
        arr = append(arr, getDis(p1, p3))
        arr = append(arr, getDis(p1, p4))
        arr = append(arr, getDis(p2, p3))
        arr = append(arr, getDis(p2, p4))
        arr = append(arr, getDis(p3, p4))
        sort.Ints(arr)
        return arr[0] > 0 && arr[0] == arr[3] && arr[4] == arr[5] && arr[0]*2 ==␣
↪arr[4]
}
```

(续下页)

(接上页)

```
func getDis(a, b []int) int {
        return (a[0]-b[0])*(a[0]-b[0]) + (a[1]-b[1])*(a[1]-b[1])
}
```

CHAPTER 18

0501-0600-Hard

18.1 502.IPO(2)

- 题目

```
假设 力扣（LeetCode）即将开始其 IPO。为了以更高的价格将股票卖给风险投资公司，
力扣 希望在 IPO 之前开展一些项目以增加其资本。
由于资源有限，它只能在 IPO 之前完成最多 k 个不同的项目。
帮助 力扣 设计完成最多 k 个不同项目后得到最大总资本的方式。
给定若干个项目。对于每个项目 i，它都有一个纯利润 Pi，并且需要最小的资本 Ci␣
↪来启动相应的项目。
最初，你有 W 资本。当你完成一个项目时，你将获得纯利润，且利润将被添加到你的总资本中。
总而言之，从给定项目中选择最多 k␣
↪个不同项目的列表，以最大化最终资本，并输出最终可获得的最多资本。
示例 1:输入: k=2, W=0, Profits=[1,2,3], Capital=[0,1,1].
输出: 4
解释:由于你的初始资本为 0，你尽可以从 0 号项目开始。
在完成后，你将获得 1 的利润，你的总资本将变为 1。
此时你可以选择开始 1 号或 2 号项目。
由于你最多可以选择两个项目，所以你需要完成 2 号项目以获得最大的资本。
因此，输出最后最大化的资本，为 0 + 1 + 3 = 4。
注意:假设所有输入数字都是非负整数。
    表示利润和资本的数组的长度不超过 50000。
    答案保证在 32 位有符号整数范围内。
```

- 解题思路

```
func findMaximizedCapital(k int, W int, Profits []int, Capital []int) int {
        maxProfit := &ProfitNode{}
        minCapital := &CapitalNode{}
        heap.Init(maxProfit)
        heap.Init(minCapital)
        for i := 0; i < len(Profits); i++ {
                heap.Push(minCapital, Node{
                        Profits: Profits[i],
                        Capital: Capital[i],
                })
        }
        for i := 0; i < k; i++ {
                for minCapital.Len() > 0 {
                        node := heap.Pop(minCapital).(Node)
                        if node.Capital <= W {
                                heap.Push(maxProfit, node)
                        } else {
                                heap.Push(minCapital, node)
                                break
                        }
                }
                if maxProfit.Len() == 0 {
                        return W
                }
                node := heap.Pop(maxProfit).(Node)
                W = W + node.Profits
        }
        return W
}

type Node struct {
        Profits int
        Capital int
}

type ProfitNode []Node

func (h ProfitNode) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h ProfitNode) Less(i, j int) bool {
```

(续下页)

(接上页)

```
        return h[i].Profits > h[j].Profits
}

func (h ProfitNode) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *ProfitNode) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *ProfitNode) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

type CapitalNode []Node

func (h CapitalNode) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h CapitalNode) Less(i, j int) bool {
        return h[i].Capital < h[j].Capital
}

func (h CapitalNode) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *CapitalNode) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *CapitalNode) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
```

(续下页)

(接上页)

```
type Node struct {
        profit  int
        capital int
}

func findMaximizedCapital(k int, W int, Profits []int, Capital []int) int {
        arr := make([]Node, 0)
        for i := 0; i < len(Profits); i++ {
                arr = append(arr, Node{
                        profit:  Profits[i],
                        capital: Capital[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].profit > arr[j].profit
        })
        index := 0
        for k > 0 {
                if index == len(arr) {
                        return W
                }
                // 挑选一个满足条件的项目，利润最大即可
                if arr[index].capital <= W {
                        k--
                        W = W + arr[index].profit
                        arr = append(arr[:index], arr[index+1:]...)
                        index = 0
                        continue
                }
                index++
        }
        return W
}
```

18.2 514. 自由之路 (2)

- 题目

```
电子游戏“辐射4”中，任务“通向自由”要求玩家到达名为“Freedom Trail␣
↪Ring”的金属表盘，
并使用表盘拼写特定关键词才能开门。
给定一个字符串ring，表示刻在外环上的编码；给定另一个字符串key，表示需要拼写的关键词。
```

(续下页)

(接上页)

```
您需要算出能够拼写关键词中所有字符的最少步数。
最初，ring的第一个字符与12:00方向对齐。
您需要顺时针或逆时针旋转 ring 以使key的一个字符在 12:00 方向对齐，
然后按下中心按钮，以此逐个拼写完key中的所有字符。
旋转ring拼出 key 字符key[i]的阶段中：
您可以将ring顺时针或逆时针旋转一个位置，计为1步。
旋转的最终目的是将字符串ring的一个字符与 12:00␣
↪方向对齐，并且这个字符必须等于字符key[i] 。
如果字符key[i]已经对齐到12:00方向，您需要按下中心按钮进行拼写，这也将算作1 步。
按完之后，您可以开始拼写key的下一个字符（下一阶段），直至完成所有拼写。
示例：输入：ring = "godding", key = "gd" 输出：4
解释：对于 key 的第一个字符 'g'，已经在正确的位置，我们只需要1步来拼写这个字符。
 对于 key 的第二个字符 'd'，我们需要逆时针旋转 ring "godding" 2步使它变成 "ddinggo"。
 当然，我们还需要1步进行拼写。
 因此最终的输出是 4。
提示：ring 和key的字符串长度取值范围均为1 至100；
两个字符串中都只有小写字符，并且均可能存在重复字符；
字符串key一定可以由字符串 ring旋转拼出。
```

- 解题思路

```
func findRotateSteps(ring string, key string) int {
        maxValue := math.MaxInt32 / 10
        n := len(key)
        m := len(ring)
        dp := make([][]int, n) // dp[i][j] => key[:i+1], ring[:j+1]的最少步数
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        dp[i][j] = maxValue
                }
        }
        arr := [26][]int{}
        for i := 0; i < len(ring); i++ {
                value := int(ring[i] - 'a')
                arr[value] = append(arr[value], i)
        }
        for _, v := range arr[key[0]-'a'] {
                dp[0][v] = min(v, m-v) + 1 // 移到次数
        }
        for i := 1; i < n; i++ {
                for _, j := range arr[key[i]-'a'] { // 枚举当前字母位置
                        for _, k := range arr[key[i-1]-'a'] { // 枚举上一个字母位置
                                minValue := min(abs(j-k), m-abs(j-k))
```

(续下页)

(接上页)

```
                                dp[i][j] = min(dp[i][j], dp[i-1][k]+minValue+1)
                        }
                }
        }
        res := math.MaxInt32
        for i := 0; i < m; i++ {
                res = min(res, dp[n-1][i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
var dp [][]int
var arr [26][]int

func findRotateSteps(ring string, key string) int {
        n := len(key)
        m := len(ring)
        dp = make([][]int, n) // dp[i][j] => key[:i+1], ring[:j+1]的最少步数
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        dp[i][j] = -1
                }
        }
        arr = [26][]int{}
        for i := 0; i < len(ring); i++ {
                value := int(ring[i] - 'a')
                arr[value] = append(arr[value], i)
```

(续下页)

(接上页)

```
        }
        return n + dfs(key, ring, 0, 0)
}

func dfs(key, ring string, keyIndex, ringIndex int) int {
        if keyIndex == len(key) {
                return 0
        }
        if dp[keyIndex][ringIndex] != -1 {
                return dp[keyIndex][ringIndex]
        }
        cur := int(key[keyIndex] - 'a')
        res := math.MaxInt32
        for _, v := range arr[cur] {
                minValue := min(abs(ringIndex-v), len(ring)-abs(ringIndex-v))
                res = min(res, minValue+dfs(key, ring, keyIndex+1, v))
        }
        dp[keyIndex][ringIndex] = res
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

18.3 517. 超级洗衣机 (1)

- 题目

```
假设有␣
↪n台超级洗衣机放在同一排上。开始的时候，每台洗衣机内可能有一定量的衣服，也可能是空的。
在每一步操作中，你可以选择任意 m (1 ≤ m ≤␣
↪n) 台洗衣机，与此同时将每台洗衣机的一件衣服送到相邻的一台洗衣机。
给定一个非负整数数组代表从左至右每台洗衣机中的衣物数量，请给出能让所有洗衣机中剩下的衣物的数量相等的
如果不能使每台洗衣机中衣物的数量相等，则返回 -1。
示例 1: 输入: [1,0,5] 输出: 3
解释: 第一步:    1     0 <-- 5    =>    1     1     4
第二步:    1 <-- 1 <-- 4    =>    2     1     3
第三步:    2     1 <-- 3    =>    2     2     2
示例 2: 输入: [0,3,0] 输出: 2
解释:  第一步:    0 <-- 3     0    =>    1     2     0
第二步:    1     2 --> 0    =>    1     1     1
示例 3:输入: [0,2,0] 输出: -1
解释:  不可能让所有三个洗衣机同时剩下相同数量的衣物。
提示: n 的范围是 [1, 10000]。
在每台超级洗衣机中，衣物数量的范围是 [0, 1e5]。
```

- 解题思路

```
func findMinMoves(machines []int) int {
        n := len(machines)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + machines[i]
        }
        if sum%n != 0 { // 先判断
                return -1
        }
        per := sum / n // 最终每个洗衣机里面的衣服数
        for i := 0; i < n; i++ {
                machines[i] = machines[i] - per //␣
↪计算每个洗衣机需要拿出或者需要放入的衣服数
        }
        maxValue := 0
        curSum := 0
        res := 0
        // 注意: 选择任意m台，不要求连续
        // 2种情况:
        // 1、数组的最大值: 是取出，每次一件，会有最大值的次数
```

(续下页)

(接上页)

```
        // 2、前缀和的最大绝对值：前面多余或者需要的数量
        for i := 0; i < n; i++ {
                curSum = curSum + machines[i]                //␣
↪前缀和：需要移动curSum件衣服到当前节点
                maxValue = max(maxValue, abs(curSum))        // 需要移动的最大值
                res = max(res, max(maxValue, machines[i])) //␣
↪取：数组的最大值和数组前缀和的绝对值的最大值中的较大值
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

18.4 535.TinyURL 的加密与解密 (2)

- 题目

```
TinyURL是一种URL简化服务，比如：当你输入一个URLhttps://leetcode.com/problems/design-
↪tinyurl时，
它将返回一个简化的URLhttp://tinyurl.com/4e9iAk.
要求：设计一个 TinyURL 的加密encode和解密decode的方法。
你的加密和解密算法如何设计和运作是没有限制的，你只需要保证一个URL可以被加密成一个TinyURL，
并且这个TinyURL可以用解密方法恢复成原本的URL。
```

- 解题思路

```
type Codec struct {
        m     map[string]string
        index int
}
```

(续下页)

(接上页)

```
func Constructor() Codec {
        return Codec{
                m:     make(map[string]string),
                index: 1,
        }
}

// Encodes a URL to a shortened URL.
func (this *Codec) encode(longUrl string) string {
        res := "http://tinyurl.com/" + strconv.Itoa(this.index)
        this.m[res] = longUrl
        this.index++
        return res
}

// Decodes a shortened URL to its original URL.
func (this *Codec) decode(shortUrl string) string {
        return this.m[shortUrl]
}

# 2
type Codec struct {
        m     map[string]string
        index int
}

func Constructor() Codec {
        return Codec{
                m:     make(map[string]string),
                index: 1,
        }
}

// Encodes a URL to a shortened URL.
func (this *Codec) encode(longUrl string) string {
        str := "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
        res := "http://tinyurl.com/"
        this.index++
        count := this.index
        temp := make([]byte, 0)
        for count > 0 {
                temp = append(temp, str[count%62])
```

(续下页)

(接上页)

```
                count = count / 62
        }
        res = res + string(temp)
        this.m[res] = longUrl
        return res
}

// Decodes a shortened URL to its original URL.
func (this *Codec) decode(shortUrl string) string {
        return this.m[shortUrl]
}
```

18.5 546. 移除盒子

18.5.1 题目

```
给出一些不同颜色的盒子，盒子的颜色由数字表示，即不同的数字表示不同的颜色。
你将经过若干轮操作去去掉盒子，直到所有的盒子都去掉为止。
每一轮你可以移除具有相同颜色的连续 k 个盒子（k>= 1），这样一轮之后你将得到 k * k␣
↪个积分。
当你将所有盒子都去掉之后，求你能获得的最大积分和。
示例 1：输入：boxes = [1,3,2,2,2,3,4,3,1] 输出：23
解释：[1, 3, 2, 2, 2, 3, 4, 3, 1]
----> [1, 3, 3, 4, 3, 1] (3*3=9 分)
----> [1, 3, 3, 3, 1] (1*1=1 分)
----> [1, 1] (3*3=9 分)
----> [] (2*2=4 分)
示例 2：输入：boxes = [1,1,1] 输出：9
示例 3：输入：boxes = [1] 输出：1
提示：1 <= boxes.length <= 100
1 <= boxes[i]<= 100
```

18.5.2 解题思路

18.6 552. 学生出勤记录 II(1)

- 题目

```
给定一个正整数n，返回长度为 n 的所有可被视为可奖励的出勤记录的数量。
答案可能非常大，你只需返回结果mod 109 + 7的值。
学生出勤记录是只包含以下三个字符的字符串：
'A' : Absent，缺勤
'L' : Late，迟到
'P' : Present，到场
如果记录不包含多于一个'A'（缺勤）或超过两个连续的'L'（迟到），则该记录被视为可奖励的。
示例 1:输入: n = 2 输出: 8
解释: 有8个长度为2的记录将被视为可奖励:
"PP" , "AP", "PA", "LP", "PL", "AL", "LA", "LL"
只有"AA"不会被视为可奖励，因为缺勤次数超过一次。
注意: n 的值不会超过100000。
```

- 解题思路

```
var mod = 1000000007

func checkRecord(n int) int {
        dp := [6]int{}
        dp[0] = 1 // 0A 0L
        dp[1] = 1 // 0A 1L
        dp[2] = 0 // 0A 2L
        dp[3] = 1 // 1A 0L
        dp[4] = 0 // 1A 1L
        dp[5] = 0 // 1A 2L
        for i := 2; i <= n; i++ {
                temp := [6]int{}
                temp[0] = (dp[0] + dp[1] + dp[2]) % mod                          // +P
                temp[1] = dp[0]                                                  // +L
                temp[2] = dp[1]                                                  // +L
                temp[3] = (dp[0] + dp[1] + dp[2] + dp[3] + dp[4] + dp[5]) % mod //␣
↪0、1、2+A，3、4、5+P
                temp[4] = dp[3]                                                  // +L
                temp[5] = dp[4]                                                  // +L
                dp = temp
        }
        res := 0
        for i := 0; i < len(dp); i++ {
                res = (res + dp[i]) % 1000000007
        }
```

(续下页)

(接上页)

```
        return res
}
```

18.7 600. 不含连续 1 的非负整数 (1)

- 题目

```
给定一个正整数 n，找出小于或等于 n 的非负整数中，其二进制表示不包含连续的1的个数。
示例 1:输入: 5 输出: 5
解释: 下面是带有相应二进制表示的非负整数<= 5:
0 : 0
1 : 1
2 : 10
3 : 11
4 : 100
5 : 101
其中，只有整数3违反规则（有两个连续的1），其他5个满足规则。
说明: 1 <= n <= 109
```

- 解题思路

```
func findIntegers(n int) int {
        dp := make(map[int]int) // dp[i]=>
↪高度为i，根节点为0的满二叉树，不包含连续1的路径数量（下标从1开始）
        dp[1] = 1               // 高度为1的时候，只有0等1种情况
        dp[2] = 2               // 高度为2的时候，有00、01等2种情况
        for i := 3; i <= 32; i++ {
                dp[i] = dp[i-1] + dp[i-2] // 左子树(0)+右子树的左子树(10)的数量
        }
        res := 0
        prev := 0
        for i := 32; i >= 1; i-- { // 从最高位开始遍历进行替换（下标从1开始）
                if n&(1<<(i-1)) > 0 { // 第i位为1，替换该位，前缀不变
                        // 1xxx1xx => 0xxxxxx => 该位变为0
                        // 1xxx1xx => 1xxx0xx => 该位变为0，前缀不变
                        res = res + dp[i] //␣
↪高度为i: 高度=i，根节点为0的都小于n（把i位的1替换为0）
                        if prev == 1 {    //␣
↪出现连续1退出，比如后面使用1011xx的前缀就不满足题目要求了
                                break
                        }
                        prev = 1
```

(续下页)

(接上页)

```
                } else {
                        prev = 0
                }
                if i == 1 { //␣
↪如果能走到最后1位，也要算上n，说明n也满足条件(小于等于n,即n也算1次)
                        res++
                }
        }
        return res
}
```

CHAPTER 19

0601-0700-Easy

19.1 605. 种花问题 (3)

- 题目

```
假设你有一个很长的花坛，一部分地块种植了花，另一部分却没有。
可是，花卉不能种植在相邻的地块上，它们会争夺水源，两者都会死去。
给定一个花坛（表示为一个数组包含0和1，其中0表示没种植花，1表示种植了花），和一个数 n␣
↪。
能否在不打破种植规则的情况下种入 n 朵花？能则返回True，不能则返回False。

示例 1:输入: flowerbed = [1,0,0,0,1], n = 1 输出: True
示例 2:输入: flowerbed = [1,0,0,0,1], n = 2 输出: False

注意:
    数组内已种好的花不会违反种植规则。
    输入的数组长度范围为 [1, 20000]。
    n 是非负整数，且不会超过输入数组的大小。
```

- 解题思路

```
func canPlaceFlowers(flowerbed []int, n int) bool {
        length := len(flowerbed)
        // 判断条件
        // 1:当前元素是0
```

(续下页)

(接上页)

```
        // 2.前一个元素是0，或者当前是第一个元素
        // 3.后一个元素是0，或者当前是最后一个元素
        for i := 0; i < length; i++ {
                if flowerbed[i] == 0 &&
                        (i == 0 || flowerbed[i-1] == 0) &&
                        (i == length-1 || flowerbed[i+1] == 0) {
                        flowerbed[i] = 1
                        n--
                        if n <= 0 {
                                return true
                        }
                }
        }
        return n <= 0
}

#
func canPlaceFlowers(flowerbed []int, n int) bool {
        length := len(flowerbed)
        count := 0
        temp := 1
        // 以0开头，计算情况同以0结束，为向中间情况靠齐，可以特殊处理把temp初始化为1
        // 中间计算可以种花，value = (temp-1)/2
        // 最后结束如果为偶数，value=temp/2
        for i := 0; i < length; i++ {
                if flowerbed[i] == 1 {
                        count = count + (temp-1)/2
                        temp = 0
                } else {
                        temp++
                }
        }
        count = count + temp/2
        return n <= count
}

#
func canPlaceFlowers(flowerbed []int, n int) bool {
        flowerbed = append([]int{0}, flowerbed...)
        flowerbed = append(flowerbed, []int{0, 1}...)
        count := 0
        temp := 0
        // 首补0，尾补0，1，统一一种情况
```

(续下页)

(接上页)

```
        for i := 0; i < len(flowerbed); i++ {
                if flowerbed[i] == 1 {
                        count = count + (temp-1)/2
                        temp = 0
                } else {
                        temp++
                }
        }
        return n <= count
}
```

19.2 606. 根据二叉树创建字符串 (2)

- 题目

```
你需要采用前序遍历的方式，将一个二叉树转换成一个由括号和整数组成的字符串。
空节点则用一对空括号 "()"␣
↪表示。而且你需要省略所有不影响字符串与原始二叉树之间的一对一映射关系的空括号对。
示例 1:
输入: 二叉树: [1,2,3,4]
       1
     /   \
    2     3
   /
  4
输出: "1(2(4))(3)"
解释: 原本将是“1(2(4)())(3())”，
在你省略所有不必要的空括号对之后，
它将是“1(2(4))(3)”。
示例 2:
输入: 二叉树: [1,2,3,null,4]
       1
     /   \
    2     3
     \
      4
输出: "1(2()(4))(3)"
解释: 和第一个示例相似，
除了我们不能省略第一个对括号来中断输入和输出之间的一对一映射关系。
```

- 解题思路

```
func tree2str(t *TreeNode) string {
        if t == nil {
                return ""
        }
        res := strconv.Itoa(t.Val)
        if t.Left == nil && t.Right == nil {
                return res
        }
        res += "(" + tree2str(t.Left) + ")"
        if t.Right != nil{
                res += "(" + tree2str(t.Right) + ")"
        }
        return res
}

#
func tree2str(t *TreeNode) string {
        if t == nil {
                return ""
        }
        stack := make([]*TreeNode, 0)
        m := make(map[*TreeNode]bool)
        stack = append(stack, t)
        res := ""
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                if _, ok := m[node]; ok {
                        stack = stack[:len(stack)-1]
                        res = res + ")"
                } else {
                        m[node] = true
                        res = res + "(" + strconv.Itoa(node.Val)
                        if node.Left == nil && node.Right != nil {
                                res = res + "()"
                        }
                        if node.Right != nil {
                                stack = append(stack, node.Right)
                        }
                        if node.Left != nil {
                                stack = append(stack, node.Left)
                        }
                }
        }
        return res[1 : len(res)-1]
```

(续下页)

(接上页)

```
}
```

19.3 617. 合并二叉树 (2)

- 题目

```
给定两个二叉树，想象当你将它们中的一个覆盖到另一个上时，两个二叉树的一些节点便会重叠。
你需要将他们合并为一个新的二叉树。合并的规则是如果两个节点重叠，那么将他们的值相加作为节点合并后的新
否则不为 NULL 的节点将直接作为新二叉树的节点。

示例 1:
输入:
        Tree 1                     Tree 2
          1                         2
         / \                       / \
        3   2                     1   3
       /                           \   \
      5                             4   7
输出:
合并后的树:
             3
            / \
           4   5
          / \   \
         5   4   7
注意: 合并必须从两个树的根节点开始。
```

- 解题思路

```
func mergeTrees(t1 *TreeNode, t2 *TreeNode) *TreeNode {
        if t1 == nil {
                return t2
        }
        if t2 == nil {
                return t1
        }
        t1.Val = t1.Val + t2.Val
        t1.Left = mergeTrees(t1.Left, t2.Left)
        t1.Right = mergeTrees(t1.Right, t2.Right)
        return t1
}
```

(续下页)

(接上页)

```
#
func mergeTrees(t1 *TreeNode, t2 *TreeNode) *TreeNode {
        if t1 == nil {
                return t2
        }
        if t2 == nil {
                return t1
        }
        list := make([]*TreeNode, 0)
        list = append(list, t1)
        list = append(list, t2)
        for len(list) > 0 {
                node1 := list[0]
                node2 := list[1]
                node1.Val = node1.Val + node2.Val
                if node1.Left != nil && node2.Left != nil {
                        list = append(list, node1.Left)
                        list = append(list, node2.Left)
                } else if node1.Left == nil && node2.Left != nil {
                        node1.Left = node2.Left
                }
                if node1.Right != nil && node2.Right != nil {
                        list = append(list, node1.Right)
                        list = append(list, node2.Right)
                } else if node1.Right == nil && node2.Right != nil {
                        node1.Right = node2.Right
                }
                list = list[2:]
        }
        return t1
}
```

19.4 628. 三个数的最大乘积 (2)

- 题目

```
给定一个整型数组，在数组中找出由三个数组成的最大乘积，并输出这个乘积。
示例 1:输入: [1,2,3]输出: 6
示例 2:输入: [1,2,3,4]输出: 24
注意:
    给定的整型数组长度范围是[3,104]，数组中所有的元素范围是[-1000, 1000]。
    输入的数组中任意三个数的乘积不会超出32位有符号整数的范围。
```

- 解题思路

```
func maximumProduct(nums []int) int {
        sort.Ints(nums)
        return max(nums[0]*nums[1]*nums[len(nums)-1],
                nums[len(nums)-3]*nums[len(nums)-2]*nums[len(nums)-1])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func maximumProduct(nums []int) int {
        max1, max2, max3 := math.MinInt32, math.MinInt32, math.MinInt32
        min1, min2 := math.MaxInt32, math.MaxInt32
        for i := 0; i < len(nums); i++ {
                if nums[i] <= min1 {
                        min2 = min1
                        min1 = nums[i]
                } else if nums[i] <= min2 {
                        min2 = nums[i]
                }
                if nums[i] >= max1 {
                        max3 = max2
                        max2 = max1
                        max1 = nums[i]
                } else if nums[i] >= max2 {
                        max3 = max2
                        max2 = nums[i]
                } else if nums[i] >= max3 {
                        max3 = nums[i]
                }
        }
        return max(min1*min2*max1, max1*max2*max3)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}
```

19.5 633. 平方数之和 (2)

- 题目

```
给定一个非负整数 c ，你要判断是否存在两个整数 a 和 b，使得 a2 + b2 = c。
示例1:输入: 5 输出: True 解释: 1 * 1 + 2 * 2 = 5
示例2:输入: 3 输出: False
```

- 解题思路

```
func judgeSquareSum(c int) bool {
        if c < 0 {
                return false
        }
        i, j := 0, int(math.Sqrt(float64(c)))
        for i <= j {
                current := i*i + j*j
                if current < c {
                        i++
                } else if current > c {
                        j--
                } else {
                        return true
                }
        }
        return false
}

#
func judgeSquareSum(c int) bool {
        for i := 0; i <= int(math.Sqrt(float64(c))); i++ {
                b := c - i*i
                s := int(math.Sqrt(float64(b)))
                if s*s == b {
                        return true
                }
        }
        return false
}
```

19.6 637. 二叉树的层平均值 (2)

- 题目

```
给定一个非空二叉树，返回一个由每层节点平均值组成的数组.
示例 1:
输入:
    3
   / \
  9  20
    /  \
   15   7
输出: [3, 14.5, 11]
解释:第0层的平均值是 3,  第1层是 14.5, 第2层是 11. 因此返回 [3, 14.5, 11].
注意:
    节点值的范围在32位有符号整数范围内。
```

- 解题思路

```
func averageOfLevels(root *TreeNode) []float64 {
        var sum, node []int
        res := make([]float64, 0)
        sum = append(sum, root.Val)
        node = append(node, 1)
        sum, node = dfs(root, sum, node, 1)
        for i := 0; i < len(sum); i++ {
                res = append(res, float64(sum[i])/float64(node[i]))
        }
        return res
}

func dfs(root *TreeNode, sum, node []int, level int) ([]int, []int) {
        if root == nil || (root.Left == nil && root.Right == nil) {
                return sum, node
        }
        if level >= len(sum) {
                sum = append(sum, 0)
                node = append(node, 0)
        }
        if root.Left != nil {
                sum[level] += root.Left.Val
                node[level]++
        }
        if root.Right != nil {
```

(续下页)

(接上页)

```
                sum[level] += root.Right.Val
                node[level]++
        }
        sum, node = dfs(root.Left, sum, node, level+1)
        sum, node = dfs(root.Right, sum, node, level+1)
        return sum, node
}

#
func averageOfLevels(root *TreeNode) []float64 {
        res := make([]float64, 0)
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                sum := 0
                for i := 0; i < length; i++ {
                        sum = sum + list[i].Val
                        if list[i].Left != nil {
                                list = append(list, list[i].Left)
                        }
                        if list[i].Right != nil {
                                list = append(list, list[i].Right)
                        }
                }
                res = append(res, float64(sum)/float64(length))
                list = list[length:]
        }
        return res
}
```

19.7 643. 子数组最大平均数 I(3)

- 题目

```
给定 n 个整数，找出平均数最大且长度为 k 的连续子数组，并输出该最大平均数。
示例 1:输入: [1,12,-5,-6,50,3], k = 4 输出: 12.75
解释: 最大平均数 (12-5-6+50)/4 = 51/4 = 12.75
注意:
    1 <= k <= n <= 30,000。
    所给数据范围 [-10,000，10,000]。
```

- 解题思路

```
func findMaxAverage(nums []int, k int) float64 {
        temp := 0
        for i := 0; i < k; i++ {
                temp = temp + nums[i]
        }
        max := temp
        for i := k; i < len(nums); i++ {
                temp = temp + nums[i] - nums[i-k]
                if max < temp {
                        max = temp
                }
        }
        return float64(max) / float64(k)
}

#
func findMaxAverage(nums []int, k int) float64 {
        max := math.MinInt32
        for i := 0; i < len(nums); i++ {
                if i + k > len(nums){
                        break
                }
                sum := 0
                for j := i; j < i+k; j++{
                        sum = sum+nums[j]
                }
                if sum > max{
                        max = sum
                }
        }
        return float64(max) / float64(k)
}

#
func findMaxAverage(nums []int, k int) float64 {
        sum := make([]int, len(nums))
        sum[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                sum[i] = sum[i-1] + nums[i]
        }
        max := sum[k-1]
        for i := k; i < len(nums); i++ {
                if sum[i]-sum[i-k] > max {
                        max = sum[i] - sum[i-k]
```

(续下页)

(接上页)

```
                }
        }
        return float64(max) / float64(k)
}
```

19.8 645. 错误的集合 (5)

- 题目

```
集合 S 包含从1到 n 的整数。不幸的是，因为数据错误，
导致集合里面某一个元素复制了成了集合里面的另外一个元素的值，导致集合丢失了一个整数并且有一个元素重复
给定一个数组 nums 代表了集合 S 发生错误后的结果。
你的任务是首先寻找到重复出现的整数，再找到丢失的整数，将它们以数组的形式返回。

示例 1:输入: nums = [1,2,2,4]输出: [2,3]
注意:
        给定数组的长度范围是 [2, 10000]。
    给定的数组是无序的。
```

- 解题思路

```
func findErrorNums(nums []int) []int {
        newNums := make([]int, len(nums))
        var repeatNum int
        for _, v := range nums {
                if newNums[v-1] != 0 {
                        repeatNum = v
                }
                newNums[v-1] = v
        }
        for i, v := range newNums {
                if v == 0 {
                        return []int{repeatNum, i + 1}
                }
        }
        return []int{0, 0}
}

#
func findErrorNums(nums []int) []int {
        repeatNum := 0
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                n := abs(nums[i])
                if nums[n-1] < 0 {
                        repeatNum = n
                } else {
                        nums[n-1] = -nums[n-1]
                }
        }
        misNum := 0
        for i, v := range nums {
                if v > 0 {
                        misNum = i + 1
                        break
                }
        }
        return []int{repeatNum, misNum}
}

func abs(a int) int {
        if a > 0 {
                return a
        }
        return -a
}

#
func findErrorNums(nums []int) []int {
        res := 0
        // 异或得到repeatedNum^misNum
        for i := 0; i < len(nums); i++ {
                res = res ^ (i + 1) ^ (nums[i])
        }
        // 找到第一位不是0的
        h := 1
        for res&h == 0 {
                h = h << 1
        }
        a := 0
        b := 0
        for i := range nums {
                if h&nums[i] == 0 {
                        a ^= nums[i]
                } else {
                        b ^= nums[i]
```

(续下页)

(接上页)

```
                }
                if h&(i+1) == 0 {
                        a ^= i + 1
                } else {
                        b ^= i + 1
                }
        }
        for i := range nums {
                if nums[i] == b {
                        return []int{b, a}
                }
        }
        return []int{a, b}
}

#
func findErrorNums(nums []int) []int {
        m := make(map[int]int)
        n := len(nums)
        sum := 0
        repeatNum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if _, ok := m[nums[i]]; ok {
                        repeatNum = nums[i]
                }
                m[nums[i]] = 1
        }
        return []int{repeatNum, n*(n+1)/2 - sum + repeatNum}
}

#
func findErrorNums(nums []int) []int {
        sort.Ints(nums)
        n := len(nums)
        sum := 0
        repeatNum := nums[0]
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if i < len(nums)-1 && nums[i] == nums[i+1] {
                        repeatNum = nums[i]
                }
        }
```

(续下页)

(接上页)

```
        return []int{repeatNum, n*(n+1)/2 - sum + repeatNum}
}
```

19.9 653. 两数之和 IV 输入 BST(4)

- 题目

```
给定一个二叉搜索树和一个目标结果，如果 BST␣
↪中存在两个元素且它们的和等于给定的目标结果，则返回 true。
案例 1:
输入:
    5
   / \
  3   6
 / \   \
2   4   7
Target = 9
输出: True
案例 2:
输入:
    5
   / \
  3   6
 / \   \
2   4   7
Target = 28 输出: False
```

- 解题思路

```
func findTarget(root *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        m := map[int]int{}
        return dfs(root, k, m)
}

func dfs(node *TreeNode, k int, m map[int]int) bool {
        if node == nil {
                return false
        }
        if _, ok := m[k-node.Val]; ok {
```

(续下页)

(接上页)

```
                return true
        }
        m[node.Val] = node.Val
        return dfs(node.Left, k, m) || dfs(node.Right, k, m)
}

#
func dfs(root, searchRoot *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        found := findNode(searchRoot, k-root.Val)
        if found != nil && found != root {
                return true
        }
        return dfs(root.Left, searchRoot, k) ||
                dfs(root.Right, searchRoot, k)
}

func findNode(root *TreeNode, target int) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Val == target {
                return root
        }
        if root.Val < target {
                return findNode(root.Right, target)
        }
        return findNode(root.Left, target)
}

#
func findTarget(root *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        m := make(map[int]int)
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[len(queue)-1]
                queue = queue[:len(queue)-1]
```

(续下页)

(接上页)

```
                if _, ok := m[k-node.Val]; ok {
                        return true
                }
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
                m[node.Val] = 1
        }
        return false
}

#
var arr []int

func findTarget(root *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        arr = make([]int, 0)
        dfs(root)
        i := 0
        j := len(arr) - 1
        for i < j {
                if arr[i]+arr[j] == k {
                        return true
                } else if arr[i]+arr[j] > k {
                        j--
                } else {
                        i++
                }
        }
        return false
}

func dfs(node *TreeNode) {
        if node == nil {
                return
        }
        dfs(node.Left)
        arr = append(arr, node.Val)
```

(续下页)

(接上页)

```
        dfs(node.Right)
}
```

19.10 657. 机器人能否返回原点 (2)

- 题目

```
在二维平面上，有一个机器人从原点 (0, 0) 开始。
给出它的移动顺序，判断这个机器人在完成移动后是否在 (0, 0) 处结束。
移动顺序由字符串表示。字符 move[i] 表示其第 i 次移动。
机器人的有效动作有 R（右），L（左），U（上）和 D（下）。
如果机器人在完成所有动作后返回原点，则返回 true。否则，返回 false。

注意：机器人“面朝”的方向无关紧要。 “R” 将始终使机器人向右移动一次，“L” 将始终向左移动等。
此外，假设每次移动机器人的移动幅度相同。

示例 1:输入: "UD" 出: true
解释：机器人向上移动一次，然后向下移动一次。
所有动作都具有相同的幅度，因此它最终回到它开始的原点。因此，我们返回 true。

示例 2:输入: "LL"输出: false
解释：机器人向左移动两次。它最终位于原点的左侧，距原点有两次 “移动” 的距离。
我们返回 false，因为它在移动结束时没有返回原点。
```

- 解题思路

```
func judgeCircle(moves string) bool {
        return strings.Count(moves, "U") == strings.Count(moves, "D") &&
                strings.Count(moves, "L") == strings.Count(moves, "R")
}

#
func judgeCircle(moves string) bool {
        x, y := 0, 0
        for i := range moves {
                switch i {
                case 'U':
                        y = y + 1
                case 'D':
                        y = y - 1
                case 'L':
```

(续下页)

(接上页)

```
                        x = x - 1
                case 'R':
                        x = x + 1
                }
        }
        return x == 0 && y == 0
}
```

19.11 661. 图片平滑器 (2)

- 题目

```
包含整数的二维矩阵 M␣
↪表示一个图片的灰度。你需要设计一个平滑器来让每一个单元的灰度成为平均灰度 (向下舍入)␣
↪,
平均灰度的计算是周围的8个单元和它本身的值求平均，如果周围的单元格不足八个，则尽可能多的利用它们。

示例 1:
输入:
[[1,1,1],
 [1,0,1],
 [1,1,1]]
输出:
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
解释:
对于点 (0,0), (0,2), (2,0), (2,2): 平均(3/4) = 平均(0.75) = 0
对于点 (0,1), (1,0), (1,2), (2,1): 平均(5/6) = 平均(0.83333333) = 0
对于点 (1,1): 平均(8/9) = 平均(0.88888889) = 0
注意:
    给定矩阵中的整数范围为 [0, 255]。
    矩阵的长和宽的范围均为 [1, 150]。
```

- 解题思路

```
func imageSmoother(M [][]int) [][]int {
        res := make([][]int, len(M))
        for i := range res {
                res[i] = make([]int, len(M[0]))
                for j := range res[i] {
                        res[i][j] = getValue(M, i, j)
```

(续下页)

(接上页)

```
                }
        }
        return res
}

func getValue(M [][]int, r, c int) int {
        value, count := 0, 0
        for i := r - 1; i < r+2; i++ {
                for j := c - 1; j < c+2; j++ {
                        if 0 <= i && i < len(M) && 0 <= j && j < len(M[0]) {
                                value = value + M[i][j]
                                count++
                        }
                }
        }
        return value / count
}

#
func imageSmoother(M [][]int) [][]int {
        res := make([][]int, len(M))
        for i := range res {
                res[i] = make([]int, len(M[0]))
                for j := range res[i] {
                        value, count := 0, 0
                        for r := i - 1; r <= i+1; r++ {
                                for c := j - 1; c <= j+1; c++ {
                                        if 0 <= r && r < len(M) && 0 <= c && c <␣
↪len(M[0]) {
                                                value = value + M[r][c]
                                                count++
                                        }
                                }
                        }
                        res[i][j] = value / count
                }
        }
        return res
}
```

19.12 665. 非递减数列 (3)

- 题目

```
给你一个长度为 n 的整数数组，请你判断在 最多 改变 1␣
↪个元素的情况下，该数组能否变成一个非递减数列。
我们是这样定义一个非递减数列的： 对于数组中所有的 i (1 <= i < n)，总满足 array[i] <=␣
↪array[i + 1]。

示例 1:输入: nums = [4,2,3] 输出: true
解释: 你可以通过把第一个4变成1来使得它成为一个非递减数列。
示例 2:输入: nums = [4,2,1] 输出: false
解释: 你不能在只改变一个元素的情况下将其变为非递减数列。

说明:
    1 <= n <= 10 ^ 4
    - 10 ^ 5 <= nums[i] <= 10 ^ 5
```

- 解题思路

```
func checkPossibility(nums []int) bool {
        for i := 0; i < len(nums); i++ {
                res := make([]int,0)
                res = append(res, nums[0:i]...)
                res = append(res, nums[i+1:]...)
                if isSort(res) {
                        return true
                }
        }
        return false
}

func isSort(nums []int) bool {
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] > nums[i+1] {
                        return false
                }
        }
        return true
}

#
func checkPossibility(nums []int) bool {
        for i := 1; i < len(nums); i++{
```

(续下页)

(接上页)

```
                if nums[i-1] > nums[i]{
                        pre := deepCopy(nums)
                        pre[i-1] = pre[i]
                        next := deepCopy(nums)
                        next[i] = next[i-1]
                        return sort.IsSorted(sort.IntSlice(pre)) || sort.
↪IsSorted(sort.IntSlice(next))
                }
        }
        return true
}

func deepCopy(nums []int) []int  {
        res := make([]int, len(nums))
        copy(res,nums)
        return res
}

#
func checkPossibility(nums []int) bool {
        count := 0
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] > nums[i+1] {
                        if count == 1 {
                                return false
                        } else if i == 0 {
                                // 4 2 3 => 2 2 3
                                nums[i] = nums[i+1]
                                count++
                        } else if nums[i-1] > nums[i+1] {
                                // 3 4 2 => 3 4 4
                                nums[i+1] = nums[i]
                                count++
                        } else {
                                // 1 4 2 =>  1 2 2
                                nums[i] = nums[i+1]
                                count++
                        }
                }
        }
        return true
}
```

19.13 669. 修剪二叉搜索树 (2)

- 题目

```
给定一个二叉搜索树，同时给定最小边界L 和最大边界 R。
通过修剪二叉搜索树，使得所有节点的值在[L, R]中 (R>=L) 。
你可能需要改变树的根节点，所以结果应当返回修剪好的二叉搜索树的新的根节点。

示例 1:
输入:
    1
   / \
  0   2

  L = 1
  R = 2
输出:
    1
      \
       2

示例 2:
输入:
    3
   / \
  0   4
   \
    2
   /
  1
  L = 1
  R = 3
输出:
      3
     /
   2
  /
 1
```

- 解题思路

```
func trimBST(root *TreeNode, L int, R int) *TreeNode {
        if root == nil {
                return nil
```

(续下页)

(接上页)

```
        }
        if root.Val < L {
                return trimBST(root.Right, L, R)
        }
        if R < root.Val {
                return trimBST(root.Left, L, R)
        }
        root.Left = trimBST(root.Left, L, R)
        root.Right = trimBST(root.Right, L, R)
        return root
}

#
func trimBST(root *TreeNode, L int, R int) *TreeNode {
        if root == nil {
                return nil
        }
        // 找到根节点
        for root.Val < L || root.Val > R {
                if root.Val < L {
                        root = root.Right
                } else {
                        root = root.Left
                }
        }

        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        cur := root
        temp := root
        for len(stack) > 0 {
                cur = stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if cur.Left != nil {
                        if cur.Left.Val >= L {
                                // 左节点>=L, 继续向左
                                stack = append(stack, cur.Left)
                        } else {
                                // 在当前左节点, 向它的右节点找到满足<L的值,
                                // 并把当前左指针指向找到的值
                                // 如示例2里面的, 3的Left指向找到的2
                                // 然后入栈继续在2找
                                temp = cur.Left
```

(续下页)

(接上页)

```
                                for temp != nil && temp.Val < L {
                                        temp = temp.Right
                                }
                                cur.Left = temp
                                if temp != nil {
                                        stack = append(stack, temp)
                                }
                        }
                }
                if cur.Right != nil {
                        if cur.Right.Val <= R {
                                stack = append(stack, cur.Right)
                        } else {
                                temp = cur.Right
                                for temp != nil && temp.Val > R {
                                        temp = temp.Left
                                }
                                cur.Right = temp
                                if temp != nil {
                                        stack = append(stack, temp)
                                }
                        }
                }
        }
        return root
}
```

19.14 671. 二叉树中第二小的节点 (3)

- 题目

```
给定一个非空特殊的二叉树，每个节点都是正数，并且每个节点的子节点数量只能为 2 或 0。
如果一个节点有两个子节点的话，那么这个节点的值不大于它的子节点的值。
给出这样的一个二叉树，你需要输出所有节点中的第二小的值。如果第二小的值不存在的话，输出␣
↪-1 。
示例 1:
输入:
    2
   / \
  2   5
     / \
    5   7
```

(续下页)

(接上页)

```
输出：5
说明：最小的值是 2 ，第二小的值是 5 。
示例 2:
输入：
    2
   / \
  2   2
输出：-1
说明：最小的值是 2, 但是不存在第二小的值。
```

- 解题思路

```
var arr []int

func findSecondMinimumValue(root *TreeNode) int {
        arr = make([]int, 0)
        dfs(root)
        min, second := math.MaxInt32, math.MaxInt32
        flag := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] < min {
                        second = min
                        min = arr[i]
                } else if min < arr[i] && arr[i] <= second {
                        flag = 1
                        second = arr[i]
                }
        }
        if second == math.MaxInt32 && flag == 0 {
                return -1
        }
        return second
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        arr = append(arr, root.Val)
        dfs(root.Left)
        dfs(root.Right)
}
```

(续下页)

(接上页)

```
#
func dfs(root *TreeNode, val int) int {
        if root == nil {
                return -1
        }
        if root.Val > val {
                return root.Val
        }
        left := dfs(root.Left, val)
        right := dfs(root.Right, val)
        if left == -1 {
                return right
        }
        if right == -1 {
                return left
        }
        return min(left, right)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}


#
func findSecondMinimumValue(root *TreeNode) int {
        min, second := root.Val, math.MaxInt32
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        flag := 0
        for len(queue) > 0 {
                node := queue[len(queue)-1]
                queue = queue[:len(queue)-1]
                if node.Val < min {
                        second = min
                        min = node.Val
                } else if min < node.Val && node.Val <= second {
                        flag = 1
                        second = node.Val
                }
```

(续下页)

(接上页)

```
                if node.Left != nil {
                        // 有0个或2节点
                        queue = append(queue, node.Left)
                        queue = append(queue, node.Right)
                }
        }
        if second == math.MaxInt32 && flag == 0 {
                return -1
        }
        return second
}
```

19.15 674. 最长连续递增序列 (3)

- 题目

```
给定一个未经排序的整数数组，找到最长且连续的的递增序列。
示例 1:输入: [1,3,5,4,7] 输出: 3
解释: 最长连续递增序列是 [1,3,5], 长度为3。
尽管 [1,3,5,7] 也是升序的子序列, 但它不是连续的, 因为5和7在原数组里被4隔开。
示例 2:输入: [2,2,2,2,2] 输出: 1
解释: 最长连续递增序列是 [2], 长度为1。
```

- 解题思路

```
func findLengthOfLCIS(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        res := 1
        i, j := 0, 1
        for j < len(nums) {
                for j < len(nums) && nums[j-1] < nums[j] {
                        j++
                }
                if res < j-i {
                        res = j - i
                }
                i = j
                j++
        }
        return res
```

(续下页)

(接上页)

```
}

#
// 状态转移方程
// 若nums[i-1]<nums[i], 则dp[i]=dp[i-1]+1; 否则dp[i]=1
func findLengthOfLCIS(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        res := 1
        dp := make([]int,len(nums))
        for i := 0; i < len(nums); i++{
                dp[i] = 1
        }
        for i := 1; i < len(nums); i++{
                if nums[i-1] < nums[i]{
                        dp[i] = dp[i-1]+1
                }
                if dp[i] > res{
                        res = dp[i]
                }
        }
        return res
}
```

19.16 680. 验证回文字符串 II(2)

- 题目

```
给定一个非空字符串 s，最多删除一个字符。判断是否能成为回文字符串。
示例 1:输入: "aba" 输出: True
示例 2:输入: "abca"输出: True 解释: 你可以删除c字符。
注意:
    字符串只包含从 a-z 的小写字母。字符串的最大长度是50000。
```

- 解题思路

```
func validPalindrome(s string) bool {
        i := 0
        j := len(s) - 1
        for i < j {
                if s[i] != s[j] {
```

(续下页)

(接上页)

```
                        return isPalindrome(s, i, j-1) || isPalindrome(s, i+1, j)
                }
                i++
                j--
        }
        return true
}

func isPalindrome(s string, i, j int) bool {
        for i < j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

#
func validPalindrome(s string) bool {
        length := len(s)
        if length < 2 {
                return true
        }
        if s[0] == s[length-1] {
                return validPalindrome(s[1 : length-1])
        }
        return isPalindrome(s[0:length-1]) || isPalindrome(s[1:length])
}

func isPalindrome(s string) bool {
        i := 0
        j := len(s) - 1
        for i < j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}
```

19.17 682. 棒球比赛 (1)

- 题目

```
你现在是棒球比赛记录员。
给定一个字符串列表，每个字符串可以是以下四种类型之一：
1.整数（一轮的得分）：直接表示您在本轮中获得的积分数。
2. "+"（一轮的得分）：表示本轮获得的得分是前两轮有效 回合得分的总和。
3. "D"（一轮的得分）：表示本轮获得的得分是前一轮有效 回合得分的两倍。
4. "C"（一个操作，这不是一个回合的分数）：表示您获得的最后一个有效
↪回合的分数是无效的，应该被移除。

每一轮的操作都是永久性的，可能会对前一轮和后一轮产生影响。
你需要返回你在所有回合中得分的总和。
示例 1:输入：["5","2","C","D","+"] 输出：30
解释：
第1轮：你可以得到5分。总和是：5。
第2轮：你可以得到2分。总和是：7。
操作1：第2轮的数据无效。总和是：5。
第3轮：你可以得到10分（第2轮的数据已被删除）。总数是：15。
第4轮：你可以得到5 + 10 = 15分。总数是：30。

示例 2:输入：["5","-2","4","C","D","9","+","+"] 输出：27
解释：
第1轮：你可以得到5分。总和是：5。
第2轮：你可以得到-2分。总数是：3。
第3轮：你可以得到4分。总和是：7。
操作1：第3轮的数据无效。总数是：3。
第4轮：你可以得到-4分（第三轮的数据已被删除）。总和是：-1。
第5轮：你可以得到9分。总数是：8。
第6轮：你可以得到-4 + 9 = 5分。总数是13。
第7轮：你可以得到9 + 5 = 14分。总数是27。

注意：
    输入列表的大小将介于1和1000之间。
    列表中的每个整数都将介于-30000和30000之间。
```

- 解题思路

```
func calPoints(ops []string) int {
        stacks := make([]int, 0)
        for i := range ops {
                switch ops[i] {
                case "+":
```

(续下页)

(接上页)

```
                        r1 := stacks[len(stacks)-1]
                        r2 := stacks[len(stacks)-2]
                        stacks = append(stacks, r1+r2)
                case "D":
                        r1 := stacks[len(stacks)-1]
                        stacks = append(stacks, 2*r1)
                case "C":
                        stacks = stacks[:len(stacks)-1]
                default:
                        tempInt, _ := strconv.Atoi(ops[i])
                        stacks = append(stacks, tempInt)
                }
        }
        res := 0
        for _, value := range stacks {
                res = res + value
        }
        return res
}
```

19.18 686. 重复叠加字符串匹配 (2)

- 题目

```
给定两个字符串 A 和 B, 寻找重复叠加字符串A的最小次数,
使得字符串B成为叠加后的字符串A的子串, 如果不存在则返回 -1。

举个例子, A = "abcd", B = "cdabcdab"。
答案为 3, 因为 A 重复叠加三遍后为 “abcdabcdabcd”,
此时 B 是其子串; A 重复叠加两遍后为"abcdabcd", B 并不是其子串。

注意:
A 与 B 字符串的长度在1和10000区间范围内。
```

- 解题思路

```
func repeatedStringMatch(A string, B string) int {
        times := len(B) / len(A)
        // 要确保B是A的子串, 就要最少重复len(B)/len(A)次A次, 最多len(B)/len(A)+2次
        // 如长度为 len(B) = 6, len(A) = 3,至少重复2次
        // 长度为len(B) = 7, len(A) = 3, 至少重复3次
        // 另外如B="cabcabca", A="abc",需要重复4次
```

(续下页)

(接上页)

```
        for i := times; i <= times+2; i++ {
                if strings.Contains(strings.Repeat(A, i), B) {
                        return i
                }
        }
        return -1
}

#
func repeatedStringMatch(A string, B string) int {
        temp := A
        count := 1
        for len(temp) < len(B) {
                temp = temp + A
                count++
        }
        if strings.Contains(temp, B) {
                return count
        }
        temp = temp + A
        if strings.Contains(temp, B) {
                return count + 1
        }
        return -1
}
```

19.19 687. 最长同值路径 (3)

- 题目

```
给定一个二叉树，找到最长的路径，这个路径中的每个节点具有相同值。␣
↪这条路径可以经过也可以不经过根节点。
注意：两个节点之间的路径长度由它们之间的边数表示。
示例 1:
输入:
              5
             / \
            4   5
           / \   \
          1   1   5

输出:2
```

(续下页)

(接上页)

```
示例 2:
输入:
              1
             / \
            4   5
           / \   \
          4   4   5
输出:2
注意: 给定的二叉树不超过10000个结点。 树的高度不超过1000。
```

- 解题思路

```
var maxLen int

func longestUnivaluePath(root *TreeNode) int {
        maxLen = 0
        dfs(root)
        return maxLen
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        l, r := 0, 0
        if root.Left != nil && root.Val == root.Left.Val {
                l = left + 1
        }
        if root.Right != nil && root.Val == root.Right.Val {
                r = right + 1
        }
        if l+r > maxLen {
                maxLen = l + r
        }
        return max(l, r)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

#
var maxLen int

func longestUnivaluePath(root *TreeNode) int {
        maxLen = 0
        if root == nil {
                return 0
        }
        dfs(root, root.Val)
        return maxLen
}

func dfs(root *TreeNode, val int) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left, root.Val)
        right := dfs(root.Right, root.Val)
        if left+right > maxLen {
                maxLen = left + right
        }
        if root.Val == val {
                return max(left, right) + 1
        }
        return 0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 参考543.二叉树的直径做法
func longestUnivaluePath(root *TreeNode) int {
        res := 0
        stack := make([]*TreeNode, 0)
        m := make(map[*TreeNode]int)

        cur := root
```

(续下页)

(接上页)

```
        var prev *TreeNode
        for cur != nil || len(stack) != 0 {
                for cur != nil {
                        stack = append(stack, cur)
                        cur = cur.Left
                }
                cur = stack[len(stack)-1]
                if cur.Right == nil || cur.Right == prev {
                        cur = stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        leftLen := 0
                        rightLen := 0
                        if v, ok := m[cur.Left]; ok {
                                leftLen = v
                        }
                        if v, ok := m[cur.Right]; ok {
                                rightLen = v
                        }
                        var left, right int
                        if cur.Left != nil && cur.Val == cur.Left.Val {
                                left = leftLen + 1
                        }
                        if cur.Right != nil && cur.Val == cur.Right.Val {
                                right = rightLen + 1
                        }

                        if left+right > res {
                                res = left + right
                        }
                        if left > right {
                                m[cur] = left
                        } else {
                                m[cur] = right
                        }
                        prev = cur
                        cur = nil
                } else {
                        cur = cur.Right
                }
        }
        return res
}
```

19.20 690. 员工的重要性 (2)

- 题目

```
给定一个保存员工信息的数据结构，它包含了员工唯一的id，重要度 和 直系下属的id。
比如，员工1是员工2的领导，员工2是员工3的领导。他们相应的重要度为15，10，5。
那么员工1的数据结构是[1，15，[2]]，员工2的数据结构是[2，10，[3]]，员工3的数据结构是[3，
↪ 5，[]]。
注意虽然员工3也是员工1的一个下属，但是由于并不是直系下属，因此没有体现在员工1的数据结构中。
现在输入一个公司的所有员工信息，以及单个员工id，返回这个员工和他所有下属的重要度之和。

示例 1:输入：[[1，5，[2，3]]，[2，3，[]]，[3，3，[]]]，1 输出：11
解释：
员工1自身的重要度是5，他有两个直系下属2和3，而且2和3的重要度均为3。因此员工1的总重要度是␣
↪5 + 3 + 3 = 11。
注意：
    一个员工最多有一个直系领导，但是可以有多个直系下属
    员工数量不超过2000。
```

- 解题思路

```
func getImportance(employees []*Employee, id int) int {
        if len(employees) == 0 {
                return 0
        }
        var root *Employee
        for i := 0; i < len(employees); i++ {
                if employees[i].Id == id {
                        root = employees[i]
                }
        }
        if root == nil {
                return 0
        }
        res := root.Importance
        for i := range root.Subordinates {
                res = res + getImportance(employees, root.Subordinates[i])
        }
        return res
}

#
func getImportance(employees []*Employee, id int) int {
        if len(employees) == 0 {
```

(续下页)

(接上页)

```
                return 0
        }
        m := make(map[int]*Employee)
        for i := 0; i < len(employees); i++ {
                m[employees[i].Id] = employees[i]
        }
        root := m[id]
        if root == nil {
                return 0
        }
        res := 0
        list := make([]*Employee, 0)
        list = append(list, root)
        for len(list) > 0 {
                node := list[0]
                list = list[1:]
                res = res + node.Importance
                for i := range node.Subordinates {
                        if value, ok := m[node.Subordinates[i]]; ok {
                                list = append(list, value)
                        }
                }
        }
        return res
}
```

19.21 693. 交替位二进制数 (4)

- 题目

```
给定一个正整数，检查他是否为交替位二进制数：换句话说，就是他的二进制数相邻的两个位数永不相等。
示例 1: 输入: 5 输出: True
解释:5的二进制数是: 101
示例 2:输入: 7 输出: False
解释: 7的二进制数是: 111
示例 3:输入: 11 输出: False
解释: 11的二进制数是: 1011
示例 4:输入: 10 输出: True
解释: 10的二进制数是: 1010
```

- 解题思路

```
func hasAlternatingBits(n int) bool {
        str := strconv.FormatInt(int64(n), 2)
        for i := 1; i < len(str); i++ {
                if str[i] == str[i-1] {
                        return false
                }
        }
        return true
}

#
/*
示例1:
1. n=1010
2. n>>1=101
3. n=n^(n>>1)=1010^101=1111
4. n&(n+1)=1111&(10000)=0

示例2:
1. n=101
2. n>>1=10
3. n=n^(n>>1)=101^10=111
4. n&(n+1)=111&(1000)=0
*/
func hasAlternatingBits(n int) bool {
        n = n ^ (n >> 1)
        return n&(n+1) == 0
}

#
// n (10|01)&3(11)=10|01
func hasAlternatingBits(n int) bool {
        temp := n & 3
        if temp != 1 && temp != 2 {
                return false
        }
        for n > 0 {
                if n&3 != temp {
                        return false
                }
                n = n >> 2
        }
        return true
}
```

(续下页)

(接上页)

```
#
// n (10|01)&3(11)=10|01
func hasAlternatingBits(n int) bool {
        temp := n & 3
        if temp != 1 && temp != 2 {
                return false
        }
        for n > 0 {
                if n&3 != temp {
                        return false
                }
                n = n >> 2
        }
        return true
}

#
func hasAlternatingBits(n int) bool {
        pre := n & 1
        n = n >> 1
        for n > 0 {
                if n&1 == pre {
                        return false
                }
                pre = n & 1
                n = n >> 1
        }
        return true
}
```

19.22 696. 计数二进制子串 (3)

- 题目

```
给定一个字符串 s，计算具有相同数量0和1的非空(连续)子字符串的数量，
并且这些子字符串中的所有0和所有1都是组合在一起的。
重复出现的子串要计算它们出现的次数。

示例 1 :输入: "00110011" 输出: 6
解释: 有6个子串具有相同数量的连续1和0: “0011”, “01”, “1100”, “10”, “0011”␣
↪和 “01”。
```

(续下页)

(接上页)

```
请注意，一些重复出现的子串要计算它们出现的次数。
另外，“00110011”不是有效的子串，因为所有的0（和1）没有组合在一起。

示例 2 :输入: "10101"输出: 4
解释: 有4个子串：“10”，“01”，“10”，“01”，它们具有相同数量的连续1和0。
注意：
    s.length 在1到50,000之间。
    s 只包含“0”或“1”字符。
```

- 解题思路

```
func countBinarySubstrings(s string) int {
        res := 0
        cur := 1
        pre := 0
        for i := 0; i < len(s)-1; i++ {
                if s[i] == s[i+1] {
                        cur++
                } else {
                        if pre > cur {
                                res = res + cur
                        } else {
                                res = res + pre
                        }
                        pre = cur
                        cur = 1
                }
        }
        if pre > cur {
                return res + cur
        }
        return res + pre
}

#
func countBinarySubstrings(s string) int {
        res := 0
        arr := make([]int, 0)
        arr = append(arr, 1)
        for i := 1; i < len(s); i++ {
                if s[i] == s[i-1] {
                        arr[len(arr)-1]++
                } else {
                        arr = append(arr, 1)
```

(续下页)

(接上页)

```
                }
        }
        for i := 0; i < len(arr)-1; i++ {
                if arr[i] > arr[i+1] {
                        res = res + arr[i+1]
                } else {
                        res = res + arr[i]
                }
        }
        return res
}

#
var count int

func countBinarySubstrings(s string) int {
        count = 0
        for i := 1; i < len(s); i++ {
                if s[i-1] == '0' && s[i] == '1' {
                        CountString(s, i-1, i)
                }
                if s[i-1] == '1' && s[i] == '0' {
                        CountString(s, i-1, i)
                }
        }
        return count
}

func CountString(s string, left, right int) {
        leftStr := s[left]
        rightStr := s[right]
        for left >= 0 && right < len(s) && s[left] == leftStr && s[right] == rightStr
↪{
                left--
                right++
                count++
        }
}
```

19.23 697. 数组的度 (3)

- 题目

```
给定一个非空且只包含非负数的整数数组 nums,␣
↪数组的度的定义是指数组里任一元素出现频数的最大值。
你的任务是找到与 nums 拥有相同大小的度的最短连续子数组，返回其长度。

示例 1:输入: [1, 2, 2, 3, 1]输出: 2
解释: 输入数组的度是2，因为元素1和2的出现频数最大，均为2.
连续子数组里面拥有相同度的有如下所示:
[1, 2, 2, 3, 1], [1, 2, 2, 3], [2, 2, 3, 1], [1, 2, 2], [2, 2, 3], [2, 2]
最短连续子数组[2, 2]的长度为2，所以返回2.

示例 2:输入: [1,2,2,3,1,4,2] 输出: 6

注意:
    nums.length 在1到50,000区间范围内。
    nums[i] 是一个在0到49,999范围内的整数。
```

- 解题思路

```
type node struct {
        count int
        left  int
        right int
}

func findShortestSubArray(nums []int) int {
        m := make(map[int]*node, 0)
        for k, v := range nums {
                if nd, ok := m[v]; ok {
                        nd.count = nd.count + 1
                        nd.right = k
                } else {
                        m[v] = &node{
                                count: 1,
                                left:  k,
                                right: k,
                        }
                }
        }
        maxNode := new(node)
        for _, v := range m {
```

(续下页)

(接上页)

```
                if v.count > maxNode.count {
                        maxNode = v
                } else if v.count == maxNode.count &&
                        v.right-v.left < maxNode.right-maxNode.left {
                        maxNode = v
                }
        }
        return maxNode.right - maxNode.left + 1
}

#
func findShortestSubArray(nums []int) int {
        size := len(nums)
        if size < 2 {
                return size
        }
        first := make(map[int]int)
        count := make(map[int]int)
        maxCount := 1
        minLen := size
        for i, n := range nums {
                count[n]++
                if count[n] == 1 {
                        first[n] = i
                } else {
                        length := i - first[n] + 1
                        if maxCount < count[n] ||
                                (maxCount == count[n] && minLen > length) {
                                maxCount = count[n]
                                minLen = length
                        }
                }
        }
        if len(count) == size {
                return 1
        }
        return minLen
}

#
func findShortestSubArray(nums []int) int {
        size := len(nums)
        if size < 2 {
```

(续下页)

(接上页)

```
                return size
        }
        res := 0
        maxLen := 0
        m := make(map[int][]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = append(m[nums[i]], i)
        }
        for _, v := range m{
                if len(v) > maxLen {
                        maxLen = len(v)
                        res = v[len(v)-1] - v[0] + 1
                } else if len(v) == maxLen && v[len(v)-1]-v[0]+1 < res {
                        res = v[len(v)-1] - v[0] + 1
                }
        }
        return res
}
```

19.24 700. 二叉搜索树中的搜索 (2)

- 题目

```
给定二叉搜索树（BST）的根节点和一个值。 你需要在BST中找到节点值等于给定值的节点。
返回以该节点为根的子树。 如果节点不存在，则返回 NULL。
例如，
给定二叉搜索树:
        4
       / \
      2   7
     / \
    1   3
和值: 2
你应该返回如下子树:
      2
     / \
    1   3
在上述示例中，如果要找的值是 5，但因为没有节点值为 5，我们应该返回 NULL。
```

- 解题思路

```
func searchBST(root *TreeNode, val int) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Val < val {
                return searchBST(root.Right, val)
        } else if root.Val > val {
                return searchBST(root.Left, val)
        }
        return root
}

#
func searchBST(root *TreeNode, val int) *TreeNode {
        if root == nil {
                return nil
        }
        stack := make([]*TreeNode, 0)
        if root.Val == val {
                return root
        } else if root.Val > val && root.Left != nil {
                stack = append(stack, root.Left)
        } else if root.Val < val && root.Right != nil {
                stack = append(stack, root.Right)
        }
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if node.Val == val {
                        return node
                } else if node.Val > val && node.Left != nil {
                        stack = append(stack, node.Left)
                } else if node.Val < val && node.Right != nil {
                        stack = append(stack, node.Right)
                }
        }
        return nil
}
```

CHAPTER 20

0601-0700-Medium

20.1 609. 在系统中查找重复文件 (1)

- 题目

```
给定一个目录信息列表，包括目录路径，以及该目录中的所有包含内容的文件，
您需要找到文件系统中的所有重复文件组的路径。
一组重复的文件至少包括二个具有完全相同内容的文件。
输入列表中的单个目录信息字符串的格式如下：
"root/d1/d2/.../dm f1.txt(f1_content) f2.txt(f2_content) ... fn.txt(fn_content)"
这意味着有 n 个文件（f1.txt,f2.txt...fn.txt 的内容分别是
f1_content,f2_content...fn_content）在目录root/d1/d2/.../dm下。
注意：n>=1 且 m>=0。如果 m=0，则表示该目录是根目录。
该输出是重复文件路径组的列表。对于每个组，它包含具有相同内容的文件的所有文件路径。
文件路径是具有下列格式的字符串：
"directory_path/file_name.txt"
示例 1：输入：
["root/a 1.txt(abcd) 2.txt(efgh)", "root/c 3.txt(abcd)", "root/c/d 4.txt(efgh)",
"root 4.txt(efgh)"]
输出：[["root/a/2.txt","root/c/d/4.txt","root/4.txt"],["root/a/1.txt","root/c/3.txt"]]
注：最终输出不需要顺序。
您可以假设目录名、文件名和文件内容只有字母和数字，并且文件内容的长度在 [1, 50]␣
↪的范围内。
给定的文件数量在 [1, 20000] 个范围内。
您可以假设在同一目录中没有任何文件或目录共享相同的名称。
```

(续下页)

(接上页)

```
您可以假设每个给定的目录信息代表一个唯一的目录。目录路径和文件信息用一个空格分隔。
超越竞赛的后续行动：
假设您有一个真正的文件系统，您将如何搜索文件？广度搜索还是宽度搜索？
如果文件内容非常大（GB级别），您将如何修改您的解决方案？
如果每次只能读取 1 kb 的文件，您将如何修改解决方案？
修改后的解决方案的时间复杂度是多少？其中最耗时的部分和消耗内存的部分是什么？如何优化？
如何确保您发现的重复文件不是误报？
```

- 解题思路

```
func findDuplicate(paths []string) [][]string {
        res := make([][]string, 0)
        m := make(map[string][]string)
        for i := 0; i < len(paths); i++ {
                arr := strings.Split(paths[i], " ")
                for j := 1; j < len(arr); j++ {
                        index := strings.LastIndexByte(arr[j], '(')
                        content := arr[j][index+1 : len(arr[j])-1]
                        m[content] = append(m[content], arr[0]+"/"+arr[j][:index])
                }
        }
        for _, v := range m {
                if len(v) > 1 {
                        res = append(res, v)
                }
        }
        return res
}
```

20.2 611. 有效三角形的个数 (3)

- 题目

```
给定一个包含非负整数的数组，你的任务是统计其中可以组成三角形三条边的三元组个数。
示例 1:输入: [2,2,3,4] 输出: 3
解释:有效的组合是:
2,3,4 (使用第一个 2)
2,3,4 (使用第二个 2)
2,2,3
注意: 数组长度不超过1000。
    数组里整数的范围为 [0, 1000]。
```

- 解题思路

```
func triangleNumber(nums []int) int {
        res := 0
        n := len(nums)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        for k := j + 1; k < n; k++ {
                                if nums[i]+nums[j] > nums[k] &&
                                        nums[i]+nums[k] > nums[j] &&
                                        nums[j]+nums[k] > nums[i] {
                                        res++
                                }
                        }
                }
        }
        return res
}

# 2
func triangleNumber(nums []int) int {
        sort.Ints(nums)
        res := 0
        n := len(nums)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        for k := j + 1; k < n; k++ {
                                if nums[i]+nums[j] > nums[k] {
                                        res++
                                } else {
                                        break
                                }
                        }
                }
        }
        return res
}

# 3
func triangleNumber(nums []int) int {
        sort.Ints(nums)
        res := 0
        n := len(nums)
        for i := 0; i < n; i++ {
                if nums[i] == 0 {
                        continue
```

(续下页)

(接上页)

```
                }
                left, right := i+1, i+2
                for left < n-1 && nums[left] != 0 {
                        for right < n && nums[i]+nums[left] > nums[right] {
                                right++
                        }
                        res = res + right - left - 1
                        left++
                }
        }
        return res
}
```

20.3 621. 任务调度器 (2)

- 题目

```
给定一个用字符数组表示的 CPU 需要执行的任务列表。
其中包含使用大写的 A - Z 字母表示的26 种不同种类的任务。
任务可以以任意顺序执行，并且每个任务都可以在 1 个单位时间内执行完。
CPU 在任何一个单位时间内都可以执行一个任务，或者在待命状态。
然而，两个相同种类的任务之间必须有长度为 n 的冷却时间，
因此至少有连续 n 个单位时间内 CPU 在执行不同的任务，或者在待命状态。
你需要计算完成所有任务所需要的最短时间。
示例 ： 输入：tasks = ["A","A","A","B","B","B"], n = 2 输出：8
解释：A -> B -> (待命) -> A -> B -> (待命) -> A -> B.
     在本示例中，两个相同类型任务之间必须间隔长度为 n = 2 的冷却时间，
     而执行一个任务只需要一个单位时间，所以中间出现了（待命）状态。
提示：
    任务的总个数为 [1, 10000]。
    n 的取值范围为 [0, 100]。
```

- 解题思路

```
func leastInterval(tasks []byte, n int) int {
        arr := [26]int{}
        maxValue := 0
        for i := 0; i < len(tasks); i++ {
                arr[tasks[i]-'A']++
                if arr[tasks[i]-'A'] > maxValue {
                        maxValue = arr[tasks[i]-'A']
                }
```

(续下页)

(接上页)

```
        }
        res := (maxValue - 1) * (n + 1) // 完成所有任务至少需要(max-1)*(n+1)+1
        for i := 0; i < len(arr); i++ {
                if arr[i] == maxValue {
                        res++
                }
        }
        return max(res, len(tasks))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func leastInterval(tasks []byte, n int) int {
        arr := make([]int, 26)
        for i := 0; i < len(tasks); i++ {
                arr[tasks[i]-'A']++
        }
        sort.Ints(arr)
        res := 0
        for arr[25] > 0 {
                i := 0
                for i <= n { // 每次安排n+1个
                        if arr[25] == 0 {
                                break
                        }
                        if i < 26 && arr[25-i] > 0 {
                                arr[25-i]--
                        }
                        res++
                        i++
                }
                sort.Ints(arr)
        }
        return res
}
```

20.4 622. 设计循环队列 (2)

- 题目

```
设计你的循环队列实现。
循环队列是一种线性数据结构，其操作表现基于␣
↪FIFO（先进先出）原则并且队尾被连接在队首之后以形成一个循环。
它也被称为“环形缓冲器”。
循环队列的一个好处是我们可以利用这个队列之前用过的空间。
在一个普通队列里，一旦一个队列满了，我们就不能插入下一个元素，即使在队列前面仍有空间。
但是使用循环队列，我们能使用这些空间去存储新的值。
你的实现应该支持如下操作：
    MyCircularQueue(k): 构造器，设置队列长度为 k 。
    Front: 从队首获取元素。如果队列为空，返回 -1 。
    Rear: 获取队尾元素。如果队列为空，返回 -1 。
    enQueue(value): 向循环队列插入一个元素。如果成功插入则返回真。
    deQueue(): 从循环队列中删除一个元素。如果成功删除则返回真。
    isEmpty(): 检查循环队列是否为空。
    isFull(): 检查循环队列是否已满。
示例：
MyCircularQueue circularQueue = new MyCircularQueue(3); // 设置长度为 3
circularQueue.enQueue(1);  // 返回 true
circularQueue.enQueue(2);  // 返回 true
circularQueue.enQueue(3);  // 返回 true
circularQueue.enQueue(4);  // 返回 false，队列已满
circularQueue.Rear();  // 返回 3
circularQueue.isFull();  // 返回 true
circularQueue.deQueue();  // 返回 true
circularQueue.enQueue(4);  // 返回 true
circularQueue.Rear();  // 返回 4
提示：
    所有的值都在 0 至 1000 的范围内；
    操作数将在 1 至 1000 的范围内；
    请不要使用内置的队列库。
```

- 解题思路

```
type MyCircularQueue struct {
        queue []int
        k     int
}

func Constructor(k int) MyCircularQueue {
        return MyCircularQueue{
```

(续下页)

(接上页)

```
                queue: make([]int, 0),
                k:     k,
        }
}

func (this *MyCircularQueue) EnQueue(value int) bool {
        if len(this.queue) == this.k {
                return false
        }
        this.queue = append(this.queue, value)
        return true
}

func (this *MyCircularQueue) DeQueue() bool {
        if len(this.queue) == 0 {
                return false
        }
        this.queue = this.queue[1:]
        return true
}

func (this *MyCircularQueue) Front() int {
        if len(this.queue) == 0 {
                return -1
        }
        return this.queue[0]
}

func (this *MyCircularQueue) Rear() int {
        if len(this.queue) == 0 {
                return -1
        }
        return this.queue[len(this.queue)-1]
}

func (this *MyCircularQueue) IsEmpty() bool {
        return len(this.queue) == 0
}

func (this *MyCircularQueue) IsFull() bool {
        return len(this.queue) == this.k
}
```

(续下页)

(接上页)

```
# 2
type MyCircularQueue struct {
        queue []int
        k     int
        front int // 队首
        rear  int // 队尾
}

func Constructor(k int) MyCircularQueue {
        return MyCircularQueue{
                queue: make([]int, k+1),
                k:     k + 1,
                front: 0,
                rear:  0,
        }
}

func (this *MyCircularQueue) EnQueue(value int) bool {
        if this.IsFull() {
                return false
        }
        // 队尾入队
        this.queue[this.rear] = value
        this.rear++
        if this.rear == this.k {
                this.rear = 0
        }
        return true
}

func (this *MyCircularQueue) DeQueue() bool {
        if this.IsEmpty() {
                return false
        }
        // 队尾出队
        this.front++
        if this.front == this.k {
                this.front = 0
        }
        return true
}

func (this *MyCircularQueue) Front() int {
```

(续下页)

(接上页)

```
        if this.IsEmpty() {
                return -1
        }
        return this.queue[this.front]
}

func (this *MyCircularQueue) Rear() int {
        if this.IsEmpty() {
                return -1
        }
        prev := this.rear - 1
        if prev < 0 {
                prev = this.k - 1
        }
        return this.queue[prev]
}

func (this *MyCircularQueue) IsEmpty() bool {
        return this.front == this.rear
}

func (this *MyCircularQueue) IsFull() bool {
        next := this.rear + 1
        if next == this.k {
                next = 0
        }
        return next == this.front
}
```

20.5 623. 在二叉树中增加一行 (2)

- 题目

```
给定一个二叉树，根节点为第1层，深度为 1。在其第d层追加一行值为v的节点。
添加规则：给定一个深度值 d （正整数），针对深度为 d-1 层的每一非空节点 N，
为 N 创建两个值为v的左子树和右子树。
将N 原先的左子树，连接为新节点v 的左子树；将N 原先的右子树，连接为新节点v 的右子树。
如果 d 的值为 1，深度 d - 1 不存在，则创建一个新的根节点 v，原先的整棵树将作为 v
↪的左子树。
示例 1:输入：二叉树如下所示：
       4
     /   \
```

(续下页)

(接上页)

```
    2     6
   / \   /
  3   1 5
v = 1
d = 2
输出:
       4
      / \
     1   1
    /     \
   2       6
  / \     /
 3   1   5
示例 2:输入: 二叉树如下所示:
      4
     /
    2
   / \
  3   1
v = 1
d = 3
输出:
      4
     /
    2
   / \
  1   1
 /     \
3       1
注意:输入的深度值 d 的范围是: [1, 二叉树最大深度 + 1]。
输入的二叉树至少有一个节点。
```

- 解题思路

```
func addOneRow(root *TreeNode, v int, d int) *TreeNode {
        if root == nil {
                return &TreeNode{Val: v}
        }
        if d == 1 {
                return &TreeNode{Val: v, Left: root}
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        var level = 1
```

(续下页)

(接上页)

```
        for len(queue) > 0 {
                level++
                length := len(queue)
                if level == d {
                        for i := 0; i < length; i++ {
                                queue[i].Left = &TreeNode{
                                        Left: queue[i].Left,
                                        Val:  v,
                                }
                                queue[i].Right = &TreeNode{
                                        Right: queue[i].Right,
                                        Val:   v,
                                }
                        }
                }
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                }
                queue = queue[length:]
        }
        return root
}

# 2
func addOneRow(root *TreeNode, v int, d int) *TreeNode {
        if root == nil {
                return &TreeNode{Val: v}
        }
        if d == 1 {
                return &TreeNode{Val: v, Left: root}
        }
        dfs(root, v, d)
        return root
}

func dfs(root *TreeNode, v int, d int) {
        if root == nil {
                return
```

(续下页)

(接上页)

```
        }
        if d == 2 {
                root.Left = &TreeNode{
                        Val:  v,
                        Left: root.Left,
                }
                root.Right = &TreeNode{
                        Val:   v,
                        Right: root.Right,
                }
                return
        }
        dfs(root.Left, v, d-1)
        dfs(root.Right, v, d-1)
}
```

20.6 636. 函数的独占时间 (2)

- 题目

```
给出一个非抢占单线程CPU的 n 个函数运行日志，找到函数的独占时间。
每个函数都有一个唯一的 Id，从 0 到 n-1，函数可能会递归调用或者被其他函数调用。
日志是具有以下格式的字符串：function_id：start_or_end：timestamp。
例如："0:start:0"表示函数 0 从 0 时刻开始运行。"0:end:0"表示函数 0 在 0 时刻结束。
函数的独占时间定义是在该方法中花费的时间，调用其他函数花费的时间不算该函数的独占时间。
你需要根据函数的 Id 有序地返回每个函数的独占时间。
示例 1:输入：n = 2
logs =
["0:start:0",
 "1:start:2",
 "1:end:5",
 "0:end:6"]
输出:[3, 4]
说明：函数 0 在时刻 0 开始，在执行了  2个时间单位结束于时刻 1。
现在函数 0 调用函数 1，函数 1 在时刻 2 开始，执行 4 个时间单位后结束于时刻 5。
函数 0 再次在时刻 6 开始执行，并在时刻 6 结束运行，从而执行了 1 个时间单位。
所以函数 0 总共的执行了 2 +1 =3 个时间单位，函数 1 总共执行了 4 个时间单位。
说明：输入的日志会根据时间戳排序，而不是根据日志Id排序。
你的输出会根据函数Id排序，也就意味着你的输出数组中序号为 0 的元素相当于函数 0␣
↪的执行时间。
两个函数不会在同时开始或结束。
函数允许被递归调用，直到运行结束。
```

(续下页)

(接上页)

```
1 <= n <= 100
```

- 解题思路

```
type Node struct {
        Id        int
        StartTime int
        Wait      int
}

func exclusiveTime(n int, logs []string) []int {
        res := make([]int, n)
        stack := make([]Node, 0)
        for i := 0; i < len(logs); i++ {
                arr := strings.Split(logs[i], ":")
                id, _ := strconv.Atoi(arr[0])
                if arr[1] == "start" {
                        start, _ := strconv.Atoi(arr[2])
                        stack = append(stack, Node{
                                Id:        id,
                                StartTime: start,
                                Wait:      0,
                        })
                } else {
                        end, _ := strconv.Atoi(arr[2])
                        node := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        total := end - node.StartTime + 1 - node.Wait
                        res[node.Id] = res[node.Id] + total
                        if len(stack) > 0 {
                                wait := end - node.StartTime + 1
                                stack[len(stack)-1].Wait = stack[len(stack)-1].Wait +␣
↪wait
                        }
                }
        }
        return res
}

# 2
func exclusiveTime(n int, logs []string) []int {
        res := make([]int, n)
        stack := make([]int, 0)
        var prev int
```

(续下页)

(接上页)

```
        for i := 0; i < len(logs); i++ {
                arr := strings.Split(logs[i], ":")
                id, _ := strconv.Atoi(arr[0])
                if arr[1] == "start" {
                        start, _ := strconv.Atoi(arr[2])
                        if len(stack) > 0 {
                                lastId := stack[len(stack)-1]
                                res[lastId] = res[lastId] + start - prev
                        }
                        stack = append(stack, id)
                        prev = start
                } else {
                        end, _ := strconv.Atoi(arr[2])
                        lastId := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        res[lastId] = res[lastId] + end - prev + 1
                        prev = end + 1
                }
        }
        return res
}
```

20.7 638. 大礼包 (4)

- 题目

```
在LeetCode商店中， 有许多在售的物品。
然而，也有一些大礼包，每个大礼包以优惠的价格捆绑销售一组物品。
现给定每个物品的价格，每个大礼包包含物品的清单，以及待购物品清单。请输出确切完成待购清单的最低花费。
每个大礼包的由一个数组中的一组数据描述，最后一个数字代表大礼包的价格，
其他数字分别表示内含的其他种类物品的数量。
任意大礼包可无限次购买。
示例 1:输入: [2,5], [[3,0,5],[1,2,10]], [3,2] 输出: 14
解释: 有A和B两种物品，价格分别为¥2和¥5。
大礼包1，你可以以¥5的价格购买3A和0B。
大礼包2， 你可以以¥10的价格购买1A和2B。
你需要购买3个A和2个B， 所以你付了¥10购买了1A和2B（大礼包2），以及¥4购买2A。
示例 2:输入: [2,3,4], [[1,1,0,4],[2,2,1,9]], [1,2,1] 输出: 11
解释: A, B, C的价格分别为¥2, ¥3, ¥4.
你可以用¥4购买1A和1B，也可以用¥9购买2A, 2B和1C。
你需要买1A, 2B和1C，所以你付了¥4买了1A和1B（大礼包1），以及¥3购买1B， ¥4购买1C。
你不可以购买超出待购清单的物品，尽管购买大礼包2更加便宜。
```

(续下页)

(接上页)

```
说明:最多6种物品， 100种大礼包。
每种物品，你最多只需要购买6个。
你不可以购买超出待购清单的物品，即使更便宜。
提示：n == price.length
n == needs.length
1 <= n <= 6
0 <= price[i] <= 10
0 <= needs[i] <= 10
1 <= special.length <= 100
special[i].length == n + 1
0 <= special[i][j] <= 50
```

- 解题思路

```
func shoppingOffers(price []int, special [][]int, needs []int) int {
        return dfs(price, special, needs)
}

func dfs(price []int, special [][]int, needs []int) int {
        res := 0
        for i := 0; i < len(needs); i++ { // 默认：走单品所需要的总价格
                res = res + needs[i]*price[i]
        }
        for i := 0; i < len(special); i++ { // 遍历每个礼包，每次取1份尝试
                temp := make([]int, len(needs))
                copy(temp, needs) // 复制，避免还原
                j := 0
                for j = 0; j < len(temp); j++ {
                        if temp[j] < special[i][j] { //␣
↪剪枝：不满足当前礼包要求，提前退出
                                break
                        }
                        temp[j] = temp[j] - special[i][j]
                }
                if j == len(temp) { // 可以取该礼包，继续递归
                        res = min(res, dfs(price, special, temp)+special[i][j]) //␣
↪递归，取最小
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return b
        }
        return a
}

# 2
func shoppingOffers(price []int, special [][]int, needs []int) int {
        return dfs(price, special, needs)
}

func dfs(price []int, special [][]int, needs []int) int {
        res := 0
        for i := 0; i < len(needs); i++ { // 默认：走单品所需要的总价格
                res = res + needs[i]*price[i]
        }
        for i := 0; i < len(special); i++ { // 遍历每个礼包，每次取1份尝试
                j := 0
                for j = 0; j < len(needs); j++ {
                        if needs[j] < special[i][j] { //␣
↪剪枝：不满足当前礼包要求，提前退出
                                break
                        }
                }
                if j == len(needs) { // 可以取该礼包，继续递归
                        for k := 0; k < len(needs); k++ {
                                needs[k] = needs[k] - special[i][k]
                        }
                        res = min(res, dfs(price, special, needs)+special[i][j]) //␣
↪递归，取最小
                        for k := 0; k < len(needs); k++ {
                                needs[k] = needs[k] + special[i][k]
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
# 3
var m map[string]int

func shoppingOffers(price []int, special [][]int, needs []int) int {
        m = make(map[string]int)
        return dfs(price, special, needs)
}

func dfs(price []int, special [][]int, needs []int) int {
        if v, ok := m[getString(needs)]; ok {
                return v
        }
        res := 0
        for i := 0; i < len(needs); i++ { // 默认：走单品所需要的总价格
                res = res + needs[i]*price[i]
        }
        for i := 0; i < len(special); i++ { // 遍历每个礼包，每次取1份尝试
                j := 0
                for j = 0; j < len(needs); j++ {
                        if needs[j] < special[i][j] { //␣
↪剪枝：不满足当前礼包要求，提前退出
                                break
                        }
                }
                if j == len(needs) { // 可以取该礼包，继续递归
                        for k := 0; k < len(needs); k++ {
                                needs[k] = needs[k] - special[i][k]
                        }
                        res = min(res, dfs(price, special, needs)+special[i][j]) //␣
↪递归，取最小
                        for k := 0; k < len(needs); k++ {
                                needs[k] = needs[k] + special[i][k]
                        }
                }
        }
        m[getString(needs)] = res
        return res
}

func getString(arr []int) string {
        res := ""
        for i := 0; i < len(arr); i++ {
```

(续下页)

(接上页)

```
                res = res + fmt.Sprintf("%d,", arr[i])
        }
        return strings.TrimRight(res, ",")
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func shoppingOffers(price []int, special [][]int, needs []int) int {
        res := 0
        for i := 0; i < len(needs); i++ { // 默认：走单品所需要的总价格
                res = res + needs[i]*price[i]
        }
        for i := 0; i < len(special); i++ { // 遍历每个礼包，每次取1份尝试
                temp := make([]int, len(needs))
                copy(temp, needs) // 复制，避免还原
                j := 0
                for j = 0; j < len(temp); j++ {
                        if temp[j] < special[i][j] { //␣
↪剪枝：不满足当前礼包要求，提前退出
                                break
                        }
                        temp[j] = temp[j] - special[i][j]
                }
                if j == len(temp) { // 可以取该礼包，继续递归
                        res = min(res, shoppingOffers(price, special,␣
↪temp)+special[i][j]) // 递归，取最小
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

20.8 640. 求解方程 (1)

- 题目

```
求解一个给定的方程，将x以字符串"x=#value"的形式返回。该方程仅包含'+'，' -
↪'操作，变量x和其对应系数。
如果方程没有解，请返回“No solution”。
如果方程有无限解，则返回“Infinite solutions”。
如果方程中只有一个解，要保证返回值x是一个整数。
示例 1：输入："x+5-3+x=6+x-2" 输出："x=2"
示例 2:输入："x=x" 输出："Infinite solutions"
示例 3:输入："2x=x" 输出："x=0"
示例 4:输入："2x+3x-6x=x+2" 输出："x=-1"
示例 5:输入："x=x+2" 输出："No solution"
```

- 解题思路

```
func solveEquation(equation string) string {
        arr := strings.Split(equation, "=")
        left, right := split(arr[0]), split(arr[1])
        l, r := getValue(left) // l左边x的系数，r右边值
        a, b := getValue(right)
        l, r = l-a, r-b
        if l == r && l == 0 {
                return "Infinite solutions"
        } else if l == 0 && r != 0 {
                return "No solution"
        }
        return "x=" + fmt.Sprintf("%d", r/l)
}

func getValue(arr []string) (l, r int) {
        for i := 0; i < len(arr); i++ {
                s := arr[i]
                if strings.Contains(s, "x") == true {
                        s = strings.ReplaceAll(s, "x", "")
                        if s == "" || s == "+" || s == "-" {
                                s = s + "1"
                        }
                        value, _ := strconv.Atoi(s)
                        l = l + value
                } else {
                        value, _ := strconv.Atoi(s)
                        r = r - value
```

(续下页)

(接上页)

```
                }
        }
        return l, r
}

func split(str string) (res []string) {
        prev := ""
        for i := 0; i < len(str); i++ {
                if str[i] == '+' || str[i] == '-' {
                        if len(prev) > 0 {
                                res = append(res, prev)
                                prev = ""
                        }
                }
                prev = prev + string(str[i])
        }
        res = append(res, prev)
        return res
}
```

20.9 641. 设计循环双端队列 (3)

- 题目

```
设计实现双端队列。
你的实现需要支持以下操作：
MyCircularDeque(k)：构造函数,双端队列的大小为k。
insertFront()：将一个元素添加到双端队列头部。 如果操作成功返回 true。
insertLast()：将一个元素添加到双端队列尾部。如果操作成功返回 true。
deleteFront()：从双端队列头部删除一个元素。 如果操作成功返回 true。
deleteLast()：从双端队列尾部删除一个元素。如果操作成功返回 true。
getFront()：从双端队列头部获得一个元素。如果双端队列为空，返回 -1。
getRear()：获得双端队列的最后一个元素。如果双端队列为空，返回 -1。
isEmpty()：检查双端队列是否为空。
isFull()：检查双端队列是否满了。
示例：MyCircularDeque circularDeque = new MycircularDeque(3); // 设置容量大小为3
circularDeque.insertLast(1);                                  // 返回 true
circularDeque.insertLast(2);                                  // 返回 true
circularDeque.insertFront(3);                                  // 返回 true
circularDeque.insertFront(4);                                  // 已经满了，返回 false
circularDeque.getRear();                                     // 返回 2
circularDeque.isFull();                                        // 返回 true
```

(续下页)

(接上页)

```
circularDeque.deleteLast();                                    // 返回 true
circularDeque.insertFront(4);                                    // 返回 true
circularDeque.getFront();                                     // 返回 4
提示：所有值的范围为 [1, 1000]
操作次数的范围为 [1, 1000]
请不要使用内置的双端队列库。
```

- 解题思路

```
type MyCircularDeque struct {
        arr    []int
        head   int
        tail   int
        length int
}

// leetcode 622.设计循环队列
func Constructor(k int) MyCircularDeque {
        return MyCircularDeque{
                arr:    make([]int, k+1),
                head:   0,
                tail:   0,
                length: k + 1,
        }
}

func (this *MyCircularDeque) InsertFront(value int) bool {
        if this.IsFull() {
                return false
        }
        this.head = (this.head - 1 + this.length) % this.length // 入队：队头-1
        this.arr[this.head] = value
        return true
}

func (this *MyCircularDeque) InsertLast(value int) bool {
        if this.IsFull() {
                return false
        }
        this.arr[this.tail] = value
        this.tail = (this.tail + 1 + this.length) % this.length // 入队：队尾+1
        return true
}
```

(续下页)

(接上页)

```
func (this *MyCircularDeque) DeleteFront() bool {
        if this.IsEmpty() {
                return false
        }
        this.head = (this.head + 1) % this.length // 出队：队头+1
        return true
}

func (this *MyCircularDeque) DeleteLast() bool {
        if this.IsEmpty() {
                return false
        }
        this.tail = (this.tail - 1 + this.length) % this.length // 出队：队尾-1
        return true
}

func (this *MyCircularDeque) GetFront() int {
        if this.IsEmpty() {
                return -1
        }
        return this.arr[this.head]
}

func (this *MyCircularDeque) GetRear() int {
        if this.IsEmpty() {
                return -1
        }
        index := (this.tail - 1 + this.length) % this.length
        return this.arr[index]
}

func (this *MyCircularDeque) IsEmpty() bool {
        return this.head == this.tail
}

func (this *MyCircularDeque) IsFull() bool {
        return (this.tail+1)%this.length == this.head
}

# 2
type MyCircularDeque struct {
        head   *Node
        tail   *Node
```

(续下页)

(接上页)

```
        length int
        cap    int
}

type Node struct {
        value int
        pre   *Node
        next  *Node
}

func Constructor(k int) MyCircularDeque {
        return MyCircularDeque{
                cap: k,
        }
}

func (this *MyCircularDeque) InsertFront(value int) bool {
        if this.length == this.cap {
                return false
        }
        node := &Node{
                value: value,
        }
        if this.length == 0 {
                this.head = node
                this.tail = node
        } else {
                node.next = this.head
                this.head.pre = node
                this.head = node
        }
        this.length++
        return true
}

func (this *MyCircularDeque) InsertLast(value int) bool {
        if this.length == this.cap {
                return false
        }
        node := &Node{
                value: value,
        }
        if this.length == 0 {
```

(续下页)

(接上页)

```
                this.head = node
                this.tail = node
        } else {
                node.pre = this.tail
                this.tail.next = node
                this.tail = node
        }
        this.length++
        return true
}

func (this *MyCircularDeque) DeleteFront() bool {
        if this.length == 0 {
                return false
        }
        if this.length == 1 {
                this.head, this.tail = nil, nil
        } else {
                this.head = this.head.next
                this.head.pre = nil
        }
        this.length--
        return true
}

func (this *MyCircularDeque) DeleteLast() bool {
        if this.length == 0 {
                return false
        }
        if this.length == 1 {
                this.head, this.tail = nil, nil
        } else {
                this.tail = this.tail.pre
                this.tail.next = nil
        }
        this.length--
        return true
}

func (this *MyCircularDeque) GetFront() int {
        if this.length == 0 {
                return -1
        }
```

(续下页)

(接上页)

```
        return this.head.value
}

func (this *MyCircularDeque) GetRear() int {
        if this.length == 0 {
                return -1
        }
        return this.tail.value
}

func (this *MyCircularDeque) IsEmpty() bool {
        return this.length == 0
}

func (this *MyCircularDeque) IsFull() bool {
        return this.length == this.cap
}

# 3
type MyCircularDeque struct {
        arr    []int
        length int
        cap    int
}

func Constructor(k int) MyCircularDeque {
        return MyCircularDeque{
                arr:    make([]int, 0),
                length: 0,
                cap:    k,
        }
}

func (this *MyCircularDeque) InsertFront(value int) bool {
        if this.IsFull() {
                return false
        }
        this.arr = append([]int{value}, this.arr...)
        this.length++
        return true
}

func (this *MyCircularDeque) InsertLast(value int) bool {
```

(续下页)

(接上页)

```
        if this.IsFull() {
                return false
        }
        this.arr = append(this.arr, value)
        this.length++
        return true
}

func (this *MyCircularDeque) DeleteFront() bool {
        if this.IsEmpty() {
                return false
        }
        this.arr = this.arr[1:]
        this.length--
        return true
}

func (this *MyCircularDeque) DeleteLast() bool {
        if this.IsEmpty() {
                return false
        }
        this.arr = this.arr[:this.length-1]
        this.length--
        return true
}

func (this *MyCircularDeque) GetFront() int {
        if this.IsEmpty() {
                return -1
        }
        return this.arr[0]
}

func (this *MyCircularDeque) GetRear() int {
        if this.IsEmpty() {
                return -1
        }
        return this.arr[this.length-1]
}

func (this *MyCircularDeque) IsEmpty() bool {
        return this.length == 0
}
```

(续下页)

(接上页)

```
func (this *MyCircularDeque) IsFull() bool {
        return this.length == this.cap
}
```

20.10 646. 最长数对链 (2)

- 题目

```
给出n个数对。在每一个数对中，第一个数字总是比第二个数字小。
现在，我们定义一种跟随关系，当且仅当b < c时，数对(c, d)才可以跟在(a, b)后面。
我们用这种形式来构造一个数对链。
给定一个数对集合，找出能够形成的最长数对链的长度。
你不需要用到所有的数对，你可以以任何顺序选择其中的一些数对来构造。
示例：输入：[[1,2], [2,3], [3,4]] 输出：2
解释：最长的数对链是 [1,2] -> [3,4]
提示：给出数对的个数在[1, 1000] 范围内。
```

- 解题思路

```
func findLongestChain(pairs [][]int) int {
        sort.Slice(pairs, func(i, j int) bool {
                if pairs[i][1] == pairs[j][1] {
                        return pairs[i][0] < pairs[j][0]
                }
                return pairs[i][1] < pairs[j][1]
        })
        res := 0
        cur := math.MinInt32
        for i := 0; i < len(pairs); i++ {
                if cur < pairs[i][0] {
                        cur = pairs[i][1]
                        res++
                }
        }
        return res
}

# 2
func findLongestChain(pairs [][]int) int {
        sort.Slice(pairs, func(i, j int) bool {
                if pairs[i][1] == pairs[j][1] {
```

(续下页)

(接上页)

```
                        return pairs[i][0] < pairs[j][0]
                }
                return pairs[i][1] < pairs[j][1]
        })
        dp := make([]int, len(pairs))
        for i := 0; i < len(pairs); i++ {
                dp[i] = 1
        }
        res := 0
        for i := 0; i < len(pairs); i++ {
                for j := 0; j < i; j++ {
                        if pairs[j][1] < pairs[i][0] {
                                dp[i] = max(dp[i], dp[j]+1)
                        }
                }
                res = max(res, dp[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

20.11 647. 回文子串 (5)

- 题目

```
给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。
具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。
示例 1: 输入: "abc" 输出: 3
解释: 三个回文子串: "a", "b", "c"
示例 2: 输入: "aaa" 输出: 6
解释: 6个回文子串: "a", "a", "a", "aa", "aa", "aaa"
提示: 输入的字符串长度不会超过 1000 。
```

- 解题思路

```
func countSubstrings(s string) int {
        n := len(s)
        res := 0
        for i := 0; i < 2*n-1; i++ {
                left, right := i/2, i/2+i%2
                for ; 0 <= left && right < n && s[left] == s[right]; left, right =␣
↪left-1, right+1 {
                        res++
                }
        }
        return res
}

# 2
func countSubstrings(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        str := add(s)
        length := len(str)
        res := 0
        for i := 0; i < length; i++ {
                curLength := search(str, i)
                res = res + curLength/2 + curLength%2
        }
        return res
}

func add(s string) string {
        var res []rune
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        return string(res)
}

func search(s string, center int) int {
        i := center - 1
        j := center + 1
        step := 0
        for ; i >= 0 && j < len(s) && s[i] == s[j]; i, j = i-1, j+1 {
                step++
```

(续下页)

(接上页)

```
        }
        return step
}

# 3
func countSubstrings(s string) int {
        var res []rune
        res = append(res, '$')
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        res = append(res, '!')
        str := string(res)
        n := len(str) - 1
        arr := make([]int, n)
        leftMax, rightMax, result := 0, 0, 0
        for i := 1; i < n; i++ {
                if i <= rightMax {
                        arr[i] = min(rightMax-i+1, arr[2*leftMax-i])
                } else {
                        arr[i] = 1
                }
                for str[i+arr[i]] == str[i-arr[i]] {
                        arr[i]++
                }
                if i+arr[i]-1 > rightMax {
                        leftMax = i
                        rightMax = i + arr[i] - 1
                }
                result = result + arr[i]/2
        }
        return result
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
# 4
func countSubstrings(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        dp := make([][]bool, len(s))
        res := 0
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
                dp[r][r] = true
                res++
                for l := 0; l < r; l++ {
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
                        }
                        if dp[l][r] == true {
                                res++
                        }
                }
        }
        return res
}

# 5
func countSubstrings(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        res := len(s)
        for i := 0; i < len(s)-1; i++ {
                for j := i + 1; j < len(s); j++ {
                        if s[i] == s[j] && judge(s, i, j) == true {
                                res++
                        }
                }
        }
        return res
}

func judge(s string, i, j int) bool {
        for i <= j {
```

(续下页)

(接上页)

```
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}
```

20.12 648. 单词替换 (2)

- 题目

```
在英语中，我们有一个叫做词根(root)的概念，
它可以跟着其他一些词组成另一个较长的单词——我们称这个词为继承词(successor)。
例如，词根an，跟随着单词other(其他)，可以形成新的单词another(另一个)。
现在，给定一个由许多词根组成的词典和一个句子。
你需要将句子中的所有继承词用词根替换掉。如果继承词有许多可以形成它的词根，则用最短的词根替换它。
你需要输出替换之后的句子。
示例 1：输入：dictionary = ["cat","bat","rat"],
sentence = "the cattle was rattled by the battery"
输出："the cat was rat by the bat"
示例 2：输入：dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"
输出："a a b c"
示例 3：输入：dictionary = ["a", "aa", "aaa", "aaaa"],
sentence = "a aa a aaaa aaa aaa aaa aaaaaa bbb baba ababa"
输出："a a a a a a a a bbb baba a"
示例 4：输入：dictionary = ["catt","cat","bat","rat"],
sentence = "the cattle was rattled by the battery"
输出："the cat was rat by the bat"
示例 5：输入：dictionary = ["ac","ab"],
sentence = "it is abnormal that this solution is accepted"
输出："it is ab that this solution is ac"
提示：1 <= dictionary.length<= 1000
1 <= dictionary[i].length <= 100
dictionary[i]仅由小写字母组成。
1 <= sentence.length <= 10^6
sentence仅由小写字母和空格组成。
sentence 中单词的总量在范围 [1, 1000] 内。
sentence 中每个单词的长度在范围 [1, 1000] 内。
sentence 中单词之间由一个空格隔开。
sentence没有前导或尾随空格。
```

- 解题思路

```
func replaceWords(dictionary []string, sentence string) string {
        sort.Strings(dictionary)
        arr := strings.Split(sentence, " ")
        for i := 0; i < len(arr); i++ {
                for _, v := range dictionary {
                        if strings.HasPrefix(arr[i], v) {
                                arr[i] = v
                                break
                        }
                }
        }
        return strings.Join(arr, " ")
}

# 2
func replaceWords(dictionary []string, sentence string) string {
        trie := Constructor()
        for i := 0; i < len(dictionary); i++ {
                trie.Insert(dictionary[i])
        }
        arr := strings.Split(sentence, " ")
        for i := 0; i < len(arr); i++ {
                result := trie.Search(arr[i])
                if result != "" {
                        arr[i] = result
                }
        }
        return strings.Join(arr, " ")
}

type Trie struct {
        next   [26]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int       // 次数（可以改为bool）
}

func Constructor() Trie {
        return Trie{
                next:   [26]*Trie{},
                ending: 0,
        }
}

// 插入word
```

(续下页)

(接上页)

```
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

// 查找
func (this *Trie) Search(word string) string {
        temp := this
        res := ""
        for _, v := range word {
                res = res + string(v)
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return ""
                }
                if temp.ending > 0 {
                        return res
                }
        }
        return ""
}
```

20.13 649.Dota2 参议院 (1)

- 题目

```
Dota2 的世界里有两个阵营：Radiant(天辉)和Dire(夜魇)
Dota2 参议院由来自两派的参议员组成。现在参议院希望对一个 Dota2 游戏里的改变作出决定。
他们以一个基于轮为过程的投票进行。在每一轮中，每一位参议员都可以行使两项权利中的一项：
禁止一名参议员的权利：参议员可以让另一位参议员在这一轮和随后的几轮中丧失所有的权利。
宣布胜利：如果参议员发现有权利投票的参议员都是同一个阵营的，他可以宣布胜利并决定在游戏中的有关变化。
给定一个字符串代表每个参议员的阵营。字母 “R” 和 “D”␣
```

(续下页)

(接上页)

```
↪分别代表了Radiant（天辉）和Dire（夜魇）。
然后，如果有 n 个参议员，给定字符串的大小将是n。
以轮为基础的过程从给定顺序的第一个参议员开始到最后一个参议员结束。
这一过程将持续到投票结束。所有失去权利的参议员将在过程中被跳过。
假设每一位参议员都足够聪明，会为自己的政党做出最好的策略，
你需要预测哪一方最终会宣布胜利并在 Dota2 游戏中决定改变。输出应该是Radiant或Dire。
示例 1：输入："RD" 输出："Radiant"
解释：第一个参议员来自 Radiant 阵营并且他可以使用第一项权利让第二个参议员失去权力，
因此第二个参议员将被跳过因为他没有任何权利。
然后在第二轮的时候，第一个参议员可以宣布胜利，因为他是唯一一个有投票权的人
示例 2：输入："RDD" 输出："Dire"
解释：第一轮中,第一个来自 Radiant 阵营的参议员可以使用第一项权利禁止第二个参议员的权利
第二个来自 Dire 阵营的参议员会被跳过因为他的权利被禁止
第三个来自 Dire 阵营的参议员可以使用他的第一项权利禁止第一个参议员的权利
因此在第二轮只剩下第三个参议员拥有投票的权利,于是他可以宣布胜利
提示：给定字符串的长度在 [1, 10,000] 之间.
```

- 解题思路

```
func predictPartyVictory(senate string) string {
        r, d := make([]int, 0), make([]int, 0)
        for i := 0; i < len(senate); i++ {
                if senate[i] == 'R' {
                        r = append(r, i)
                } else {
                        d = append(d, i)
                }
        }
        for len(r) > 0 && len(d) > 0 {
                if r[0] < d[0] {
                        r = append(r, r[0]+len(senate))
                } else {
                        d = append(d, d[0]+len(senate))
                }
                r = r[1:]
                d = d[1:]
        }
        if len(r) > 0 {
                return "Radiant"
        }
        return "Dire"
}
```

20.14 650. 只有两个键的键盘 (2)

- 题目

```
最初在一个记事本上只有一个字符 'A'。你每次可以对这个记事本进行两种操作:
    Copy All (复制全部) : 你可以复制这个记事本中的所有字符(部分的复制是不允许的)。
    Paste (粘贴) : 你可以粘贴你上一次复制的字符。
给定一个数字 n 。你需要使用最少的操作次数, 在记事本中打印出恰好 n 个 'A'。
输出能够打印出 n 个 'A' 的最少操作次数。
示例 1:输入: 3 输出: 3
解释:最初, 我们只有一个字符 'A'。
第 1 步, 我们使用 Copy All 操作。
第 2 步, 我们使用 Paste 操作来获得 'AA'。
第 3 步, 我们使用 Paste 操作来获得 'AAA'。
说明: n 的取值范围是 [1, 1000] 。
```

- 解题思路

```
func minSteps(n int) int {
        dp := make([]int, n+3)
        if n <= 1 {
                return 0
        }
        dp[0] = 0
        dp[1] = 0
        dp[2] = 2
        for i := 3; i <= n; i++ {
                minValue := i
                for j := i / 2; j >= 2; j-- {
                        if i%j == 0 {
                                minValue = dp[j] + i/j
                                break
                        }
                }
                dp[i] = minValue
        }
        return dp[n]
}

# 2
func minSteps(n int) int {
        res := 0
        for i := 2; i <= n; i++ {
                for n%i == 0 {
```

(续下页)

(接上页)

```
                        res = res + i
                        n = n / i
                }
        }
        return res
}
```

20.15 652. 寻找重复的子树 (1)

- 题目

```
给定一棵二叉树，返回所有重复的子树。对于同一类的重复子树，你只需要返回其中任意一棵的根结点即可。
两棵树重复是指它们具有相同的结构以及相同的结点值。
示例 1：
        1
       / \
      2   3
     /   / \
    4   2   4
       /
      4
下面是两个重复的子树：
      2
     /
    4
和 4
因此，你需要以列表的形式返回上述重复子树的根结点。
```

- 解题思路

```
var m map[string]int
var res []*TreeNode

func findDuplicateSubtrees(root *TreeNode) []*TreeNode {
        m = make(map[string]int)
        res = make([]*TreeNode, 0)

        dfs(root)
        return res
}

func dfs(root *TreeNode) string {
```

(续下页)

(接上页)

```
        if root == nil {
                return "#"
        }
        value := strconv.Itoa(root.Val) + "," + dfs(root.Left) + "," + dfs(root.Right)
        m[value]++
        if m[value] == 2 {
                res = append(res, root)
        }
        return value
}
```

20.16 654. 最大二叉树 (2)

- 题目

```
给定一个不含重复元素的整数数组。一个以此数组构建的最大二叉树定义如下：
    二叉树的根是数组中的最大元素。
    左子树是通过数组中最大值左边部分构造出的最大二叉树。
    右子树是通过数组中最大值右边部分构造出的最大二叉树。
通过给定的数组构建最大二叉树，并且输出这个树的根节点。
示例 ：输入：[3,2,1,6,0,5] 输出：
返回下面这棵树的根节点：
      6
    /   \
   3     5
    \    /
     2  0
       \
        1
提示：给定的数组的大小在 [1, 1000] 之间。
```

- 解题思路

```
func constructMaximumBinaryTree(nums []int) *TreeNode {
        if len(nums) == 0 {
                return nil
        }
        index := 0
        maxValue := nums[0]
        for i := 1; i < len(nums); i++ {
                if nums[i] > maxValue {
                        maxValue = nums[i]
```

(续下页)

(接上页)

```
                        index = i
                }
        }
        return &TreeNode{
                Val:   maxValue,
                Left:  constructMaximumBinaryTree(nums[:index]),
                Right: constructMaximumBinaryTree(nums[index+1:]),
        }
}

# 2
func constructMaximumBinaryTree(nums []int) *TreeNode {
        if len(nums) == 0 {
                return nil
        }
        stack := make([]*TreeNode, 0)
        var cur *TreeNode
        for i := 0; i < len(nums); i++ {
                cur = &TreeNode{
                        Val: nums[i],
                }
                // 递减栈
                for len(stack) > 0 && stack[len(stack)-1].Val < cur.Val {
                        top := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        // top选择cur或者栈顶数据作为父节点
                        if len(stack) > 0 && stack[len(stack)-1].Val < cur.Val {
                                stack[len(stack)-1].Right = top
                        } else {
                                cur.Left = top
                        }
                }
                stack = append(stack, cur)
        }
        // 没有右边节点，栈顶元素作为第二个栈顶元素的右节点
        for len(stack) > 0 {
                cur = stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if len(stack) > 0 {
                        stack[len(stack)-1].Right = cur
                }
        }
        return cur
}
```

20.17 655. 输出二叉树 (1)

- 题目

```
在一个 m*n 的二维字符串数组中输出二叉树，并遵守以下规则：
行数m应当等于给定二叉树的高度。
列数n应当总是奇数。
根节点的值（以字符串格式给出）应当放在可放置的第一行正中间。
根节点所在的行与列会将剩余空间划分为两部分（左下部分和右下部分）。你应该将左子树输出在左下部分，右子
左下和右下部分应当有相同的大小。即使一个子树为空而另一个非空，你不需要为空的子树输出任何东西，但仍需
然而，如果两个子树都为空则不需要为它们留出任何空间。
每个未使用的空间应包含一个空的字符串""。
使用相同的规则输出子树。
示例 1:输入:
     1
    /
   2
输出:
[["", "1", ""],
 ["2", "", ""]]
示例 2:输入:
     1
    / \
   2   3
    \
     4
输出:
[["", "", "", "1", "", "", ""],
 ["", "2", "", "", "", "3", ""],
 ["", "", "4", "", "", "", ""]]
示例 3:输入:
      1
     / \
    2   5
   /
  3
 /
4
输出:
[["",  "",  "",  "",  "",  "",  "",  "1",  "",  "",  "",  "",  "",  "",  ""]
 ["",  "",  "",  "2",  "",  "",  "",  "",  "",  "",  "",  "5",  "",  "",  ""]
 ["",  "3",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  ""]
 ["4",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  "",  ""]]
注意: 二叉树的高度在范围 [1, 10] 中。
```

- 解题思路

```
var res [][]string

func printTree(root *TreeNode) [][]string {
        h := getHeightDFS(root)
        w := (1 << h) - 1
        res = make([][]string, h)
        for i := 0; i < h; i++ {
                res[i] = make([]string, w)
        }
        dfs(root, 0, 0, w-1)
        return res
}

func dfs(root *TreeNode, h, left, right int) {
        if root == nil {
                return
        }
        mid := left + (right-left)/2
        res[h][mid] = strconv.Itoa(root.Val)
        dfs(root.Left, h+1, left, mid-1)
        dfs(root.Right, h+1, mid+1, right)
}

func getHeightDFS(root *TreeNode) int {
        if root == nil {
                return 0
        }
        return 1 + max(getHeightDFS(root.Left), getHeightDFS(root.Right))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

20.18 658. 找到 K 个最接近的元素 (3)

- 题目

```
给定一个排序好的数组arr ，两个整数 k 和 x ，从数组中找到最靠近 x（两数之差最小）的 k个数。返回的结果必须要是按升序排好的。
整数 a 比整数 b 更接近 x 需要满足：
|a - x| < |b - x| 或者
|a - x| == |b - x| 且 a < b
示例 1：输入：arr = [1,2,3,4,5], k = 4, x = 3 输出：[1,2,3,4]
示例 2：输入：arr = [1,2,3,4,5], k = 4, x = -1 输出：[1,2,3,4]
提示：1 <= k <= arr.length
1 <= arr.length<= 104
数组里的每个元素与x 的绝对值不超过 104
```

- 解题思路

```
func findClosestElements(arr []int, k int, x int) []int {
        sort.Slice(arr, func(i, j int) bool { //按差值的绝对值排序，相同按照值大小排序
                if abs(arr[i]-x) == abs(arr[j]-x) {
                        return arr[i] < arr[j]
                }
                return abs(arr[i]-x) < abs(arr[j]-x)
        })
        res := arr[:k]
        sort.Ints(res)
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func findClosestElements(arr []int, k int, x int) []int {
        left, right := 0, len(arr)-k
        for left < right {
                mid := left + (right-left)/2   // 枚举左边
                if x-arr[mid] > arr[mid+k]-x { // 看x离left和right哪个远
                        left = mid + 1
```

(续下页)

(接上页)

```
                } else {
                        right = mid
                }
        }
        return arr[left : left+k]
}

# 3
func findClosestElements(arr []int, k int, x int) []int {
        left, right := 0, len(arr)-1
        for i := 1; i <= len(arr)-k; i++ {
                if x-arr[left] <= arr[right]-x { //␣
↪看x离left和right哪个远，远的就移动 相等就right-1
                        right--
                } else {
                        left++
                }
        }
        return arr[left : right+1]
}
```

20.19 659. 分割数组为连续子序列 (2)

- 题目

```
给你一个按升序排序的整数数组␣
↪num（可能包含重复数字），请你将它们分割成一个或多个长度至少为 3 的子序列，
其中每个子序列都由连续整数组成。
如果可以完成上述分割，则返回 true ；否则，返回 false 。
示例 1：输入：[1,2,3,3,4,5] 输出：True
解释:你可以分割出这样两个连续子序列 :
1, 2, 3
3, 4, 5
示例 2：输入：[1,2,3,3,4,4,5,5] 输出：True
解释：你可以分割出这样两个连续子序列 :
1, 2, 3, 4, 5
3, 4, 5
示例 3：输入：[1,2,3,4,4,5] 输出：False
提示：1 <= nums.length <= 10000
```

- 解题思路

```
func isPossible(nums []int) bool {
        m := make(map[int]*IntHeap)
        for i := 0; i < len(nums); i++ {
                v := nums[i]
                if m[v] == nil {
                        intHeap := make(IntHeap, 0)
                        heap.Init(&intHeap)
                        m[v] = &intHeap
                }
                length := 0
                if h := m[v-1]; h != nil {
                        length = heap.Pop(h).(int) // 找到最短的以v-1结尾的长度
                        if m[v-1].Len() == 0 {
                                delete(m, v-1)
                        }
                }
                temp := m[v]
                heap.Push(temp, length+1)
        }
        for _, v := range m {
                if (*v)[0] < 3 {
                        return false
                }
        }
        return true
}

type IntHeap []int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i] < h[j] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.(int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 2
func isPossible(nums []int) bool {
        m := make(map[int]int)
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        count := make(map[int]int) // 以某个数字结尾的连续子序列的个数
        for i := 0; i < len(nums); i++ {
                v := nums[i]
                if m[v] == 0 {
                        continue
                }
                if count[v-1] > 0 { // 添加到前一个数之后
                        m[v]--
                        count[v-1]--
                        count[v]++
                } else if m[v+1] > 0 && m[v+2] > 0 { // 没有，生成1个新的
                        m[v]--
                        m[v+1]--
                        m[v+2]--
                        count[v+2]++
                } else {
                        return false
                }
        }
        return true
}
```

20.20 662. 二叉树最大宽度 (2)

- 题目

```
给定一个二叉树，编写一个函数来获取这个树的最大宽度。树的宽度是所有层中的最大宽度。
这个二叉树与满二叉树（full binary tree）结构相同，但一些节点为空。
每一层的宽度被定义为两个端点（该层最左和最右的非空节点，两端点间的null节点也计入长度）之间的长度。
示例 1:输入:
           1
         /   \
        3     2
       / \     \
      5   3     9
输出: 4
解释: 最大值出现在树的第 3 层，宽度为 4 (5,3,null,9)。
示例 2:输入:
          1
```

(续下页)

(接上页)

```
          /
        3
       / \
      5   3
输出: 2
解释: 最大值出现在树的第 3 层, 宽度为 2 (5,3)。
示例 3:输入:
          1
         / \
        3   2
       /
      5
输出: 2
解释: 最大值出现在树的第 2 层, 宽度为 2 (3,2)。
示例 4:输入:
          1
         / \
        3   2
       /     \
      5       9
     /         \
    6           7
输出: 8
解释: 最大值出现在树的第 4 层, 宽度为 8 (6,null,null,null,null,null,null,7)。
注意: 答案在32位有符号整数的表示范围内。
```

- 解题思路

```
func widthOfBinaryTree(root *TreeNode) int {
        res := 1
        if root == nil {
                return 0
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        arr := make([]int, 0)
        arr = append(arr, 1)
        for len(queue) > 0 {
                if arr[len(arr)-1]-arr[0]+1 > res {
                        res = arr[len(arr)-1] - arr[0] + 1
                }
                length := len(queue)
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
```

(续下页)

(接上页)

```
                                queue = append(queue, queue[i].Left)
                                arr = append(arr, arr[i]*2)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                                arr = append(arr, arr[i]*2+1)
                        }
                }
                queue = queue[length:]
                arr = arr[length:]
        }
        return res
}

# 2
var res int
var m map[int]int

func widthOfBinaryTree(root *TreeNode) int {
        if root == nil {
                return 0
        }
        res = 0
        m = make(map[int]int)
        dfs(root, 0, 1)
        return res
}

func dfs(root *TreeNode, level int, id int) {
        if root == nil {
                return
        }
        if _, ok := m[level]; !ok {
                m[level] = id
        }
        if id-m[level]+1 > res {
                res = id - m[level] + 1
        }
        dfs(root.Left, level+1, id*2)
        dfs(root.Right, level+1, id*2+1)
}
```

20.21 667. 优美的排列 II(1)

- 题目

```
给你两个整数 n 和 k ，请你构造一个答案列表 answer ，该列表应当包含从 1 到 n 的 n␣
↪个不同正整数，并同时满足下述条件：
假设该列表是 answer =[a1, a2, a3, ... , an] ，
那么列表 [|a1 - a2|, |a2 - a3|, |a3 - a4|, ... , |an-1 - an|] 中应该有且仅有 k␣
↪个不同整数。
返回列表 answer 。如果存在多种答案，只需返回其中 任意一种 。
示例 1：输入：n = 3, k = 1
输出：[1, 2, 3]
解释：[1, 2, 3] 包含 3 个范围在 1-3 的不同整数，并且 [1, 1] 中有且仅有 1 个不同整数：1
示例 2：输入：n = 3, k = 2 输出：[1, 3, 2]
解释：[1, 3, 2] 包含 3 个范围在 1-3 的不同整数，并且 [2, 1] 中有且仅有 2␣
↪个不同整数：1 和 2
提示：1 <= k < n <= 104
```

- 解题思路

```
func constructArray(n int, k int) []int {
        if n == k {
                return nil
        }
        res := make([]int, n)
        // 构建等差数列为1：共n-k个数=>1
        for i := 1; i <= n-k; i++ {
                res[i-1] = i
        }
        // 构建交错队列：最大值和最小值交错出现，这样差值各不相同
        // n=10, k=7 => [1 2 3 4 5 6 7 10 8 9]
        // 剩下k个数（与等差数列相连）：共k个差值，依次1、2、3、...，去除1后共k-
↪1个差值
        left := n - k + 1
        right := n
        count := 0
        for i := n - k + 1; i <= n; i++ {
                if count%2 == 1 {
                        res[i-1] = left
                        left++
                } else {
                        res[i-1] = right
                        right--
                }
```

(续下页)

(接上页)

```
                count++
        }
        return res
}
```

20.22 670. 最大交换 (3)

- 题目

```
给定一个非负整数，你至多可以交换一次数字中的任意两位。返回你能得到的最大值。
示例 1 :输入: 2736 输出: 7236
解释: 交换数字2和数字7。
示例 2 :输入: 9973 输出: 9973
解释: 不需要交换。
注意:给定数字的范围是 [0, 10^8]
```

- 解题思路

```
func maximumSwap(num int) int {
        if num <= 11 {
                return num
        }
        res := num
        arr := []byte(strconv.Itoa(num))
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        tempArr := make([]byte, len(arr))
                        copy(tempArr, arr)
                        tempArr[i], tempArr[j] = tempArr[j], tempArr[i]
                        newValue, _ := strconv.Atoi(string(tempArr))
                        if newValue > res {
                                res = newValue
                        }
                }
        }
        return res
}

# 2
func maximumSwap(num int) int {
        if num <= 11 {
                return num
```

(续下页)

(接上页)

```
        }
        res := num
        arr := []byte(strconv.Itoa(num))
        temp := [10]int{}
        for i := 0; i < len(arr); i++ {
                temp[arr[i]-'0'] = i // 每个数字最后一次出现的位置
        }
        for i := 0; i < len(arr); i++ {
                // 寻找最后面比当前数字大并且最大数字并进行交换
                for j := 9; j > int(arr[i]-'0'); j-- {
                        if temp[j] > i {
                                arr[i], arr[temp[j]] = arr[temp[j]], arr[i]
                                res, _ = strconv.Atoi(string(arr))
                                return res
                        }
                }
        }
        return res
}

# 3
func maximumSwap(num int) int {
        if num <= 11 {
                return num
        }
        res := num
        arr := []byte(strconv.Itoa(num))
        temp := [10]int{}
        for i := 0; i < len(arr); i++ {
                temp[arr[i]-'0'] = i // 每个数字最后一次出现的位置
        }
        tempArr := []byte(strconv.Itoa(num))
        sort.Slice(tempArr, func(i, j int) bool {
                return tempArr[i] > tempArr[j]
        })

        for i := 0; i < len(arr); i++ {
                if arr[i] != tempArr[i] {
                        arr[i], arr[temp[int(tempArr[i]-'0')]] =␣
↪arr[temp[int(tempArr[i]-'0')]], arr[i]
                        res, _ = strconv.Atoi(string(arr))
                        return res
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

20.23 672. 灯泡开关 II(2)

- 题目

```
现有一个房间，墙上挂有n只已经打开的灯泡和 4␣
↪个按钮。在进行了m次未知操作后，你需要返回这n只灯泡可能有多少种不同的状态。
假设这 n 只灯泡被编号为 [1, 2, 3 ..., n]，这 4 个按钮的功能如下：
将所有灯泡的状态反转（即开变为关，关变为开）
将编号为偶数的灯泡的状态反转
将编号为奇数的灯泡的状态反转
将编号为 3k+1 的灯泡的状态反转 (k = 0, 1, 2, ...)
示例 1:输入: n = 1, m = 1. 输出: 2
说明: 状态为: [开], [关]
示例 2:输入: n = 2, m = 1. 输出: 3
说明: 状态为: [开, 关], [关, 开], [关, 关]
示例 3:输入: n = 3, m = 1. 输出: 4
说明: 状态为: [关, 开, 关], [开, 关, 开], [关, 关, 关], [关, 开, 开].
注意: n和m 都属于 [0, 1000].
```

- 解题思路

```
func flipLights(n int, presses int) int {
        // 找规律:
        // 同一按钮操作2次，结果不变
        // 操作状态有16种；m > 4 后，状态在m=3或者m=4之间切换
        // m = 0:  0000
        // m = 1: 1000、0100、0010、0001
        // m = 2:
        //        还原: 0000
        //  新增: 1100、1010、1001, 0110、0101、0011
        // m = 3:
        // 还原: 1000、0100、0010、0001
        // 新增: 1110、0111、1011、1101
        // m = 4:
        // 还原: 0000、1100、1010、1001, 0110、0101、0011
        // 新增: 1111
        if presses == 0 {
                return 1
```

(续下页)

(接上页)

```
        }
        if n == 1 {
                return 2
        }
        if n == 2 {
                if presses == 1 {
                        return 3
                }
                return 4
        }
        if presses == 1 {
                return 4
        } else if presses == 2 {
                return 7
        }
        return 8
}
```

20.24 673. 最长递增子序列的个数 (1)

- 题目

```
给定一个未排序的整数数组，找到最长递增子序列的个数。
示例 1:输入: [1,3,5,4,7] 输出: 2
解释: 有两个最长递增子序列，分别是 [1, 3, 4, 7] 和[1, 3, 5, 7]。
示例 2:输入: [2,2,2,2,2] 输出: 5
解释: 最长递增子序列的长度是1，并且存在5个子序列的长度为1，因此输出5。
注意: 给定的数组长度不超过 2000 并且结果一定是32位有符号整数。
```

- 解题思路

```
func findNumberOfLIS(nums []int) int {
        n := len(nums)
        if n == 0 || nums == nil {
                return 0
        }
        dp := make([]int, n)
        count := make([]int, n)
        maxValue := 0
        for i := 0; i < n; i++ {
                dp[i] = 1
                count[i] = 1
```

(续下页)

(接上页)

```
                for j := 0; j < i; j++ {
                        if nums[j] < nums[i] {
                                if dp[i] < dp[j]+1 {
                                        count[i] = count[j]
                                } else if dp[i] == dp[j]+1 {
                                        count[i] = count[i] + count[j]
                                }
                                dp[i] = max(dp[j]+1, dp[i])
                        }
                }
                maxValue = max(maxValue, dp[i])
        }
        res := 0
        for i := 0; i < n; i++ {
                if dp[i] == maxValue {
                        res = res + count[i]
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

20.25 676. 实现一个魔法字典 (3)

- 题目

```
设计一个使用单词列表进行初始化的数据结构，单词列表中的单词 互不相同 。
如果给出一个单词，请判定能否只将这个单词中一个字母换成另一个字母，使得所形成的新单词存在于你构建的字
实现 MagicDictionary 类：
MagicDictionary() 初始化对象
void buildDict(String[]dictionary) 使用字符串数组dictionary␣
↪设定该数据结构，dictionary 中的字符串互不相同
bool search(String searchWord) 给定一个字符串 searchWord ，
判定能否只将字符串中 一个␣
↪字母换成另一个字母，使得所形成的新字符串能够与字典中的任一字符串匹配。
如果可以，返回 true ；否则，返回 false 。
```

(续下页)

(接上页)

```
示例：输入["MagicDictionary", "buildDict", "search", "search", "search", "search"]
[[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]
输出[null, null, false, true, false, false]
解释 MagicDictionary magicDictionary = new MagicDictionary();
magicDictionary.buildDict(["hello", "leetcode"]);
magicDictionary.search("hello"); // 返回 False
magicDictionary.search("hhllo"); // 将第二个 'h' 替换为 'e' 可以匹配 "hello"
↪, 所以返回 True
magicDictionary.search("hell"); // 返回 False
magicDictionary.search("leetcoded"); // 返回 False
提示：1 <=dictionary.length <= 100
1 <=dictionary[i].length <= 100
dictionary[i] 仅由小写英文字母组成
dictionary 中的所有字符串 互不相同
1 <=searchWord.length <= 100
searchWord 仅由小写英文字母组成
buildDict 仅在 search 之前调用一次
最多调用 100 次 search
```

- 解题思路

```
type MagicDictionary struct {
        m map[int][]string
}

func Constructor() MagicDictionary {
        return MagicDictionary{m: map[int][]string{}}
}

func (this *MagicDictionary) BuildDict(dictionary []string) {
        for i := 0; i < len(dictionary); i++ {
                this.m[len(dictionary[i])] = append(this.m[len(dictionary[i])],
↪dictionary[i])
        }
}

func (this *MagicDictionary) Search(searchWord string) bool {
        if len(this.m[len(searchWord)]) == 0 {
                return false
        }
        for i := 0; i < len(this.m[len(searchWord)]); i++ {
                word := this.m[len(searchWord)][i]
                count := 0
                for j := 0; j < len(searchWord); j++ {
```

(续下页)

(接上页)

```
                        if word[j] != searchWord[j] {
                                count++
                                if count > 1 {
                                        break
                                }
                        }
                }
                if count == 1 {
                        return true
                }
        }
        return false
}

# 2
type MagicDictionary struct {
        arr []string
}

func Constructor() MagicDictionary {
        return MagicDictionary{arr: make([]string, 0)}
}

func (this *MagicDictionary) BuildDict(dictionary []string) {
        this.arr = dictionary
}

func (this *MagicDictionary) Search(searchWord string) bool {
        for i := 0; i < len(this.arr); i++ {
                word := this.arr[i]
                if len(word) != len(searchWord) {
                        continue
                }
                count := 0
                for j := 0; j < len(searchWord); j++ {
                        if word[j] != searchWord[j] {
                                count++
                                if count > 1 {
                                        break
                                }
                        }
                }
                if count == 1 {
```

(续下页)

(接上页)

```
                        return true
                }
        }
        return false
}

# 3
type MagicDictionary struct {
        next   [26]*MagicDictionary // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int                  // 次数（可以改为bool）
}

func Constructor() MagicDictionary {
        return MagicDictionary{
                next:   [26]*MagicDictionary{},
                ending: 0,
        }
}

func (this *MagicDictionary) BuildDict(dictionary []string) {
        for i := 0; i < len(dictionary); i++ {
                word := dictionary[i]
                temp := this
                for _, v := range word {
                        value := v - 'a'
                        if temp.next[value] == nil {
                                temp.next[value] = &MagicDictionary{
                                        next:   [26]*MagicDictionary{},
                                        ending: 0,
                                }
                        }
                        temp = temp.next[value]
                }
                temp.ending++
        }
}

func (this *MagicDictionary) Search(searchWord string) bool {
        cur := this
        arr := []byte(searchWord)
        for i := 0; i < len(searchWord); i++ {
                b := searchWord[i]
                for j := 0; j < 26; j++ {
```

(续下页)

(接上页)

```
                        if j+'a' == int(b) {
                                continue
                        }
                        arr[i] = byte('a' + j)
                        if cur.SearchWord(string(arr[i:])) == true {
                                return true
                        }
                }
                arr[i] = b
                if cur.next[int(b-'a')] == nil {
                        return false
                }
                cur = cur.next[int(b-'a')]
        }
        return false
}

func (this *MagicDictionary) SearchWord(word string) bool {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}
```

20.26 677. 键值映射 (3)

- 题目

```
实现一个 MapSum 类，支持两个方法，insert和sum：
MapSum() 初始化 MapSum 对象
void insert(String key, int val) 插入 key-val 键值对，字符串表示键 key ，整数表示值␣
↪val 。
如果键 key 已经存在，那么原来的键值对将被替代成新的键值对。
int sum(string prefix) 返回所有以该前缀 prefix 开头的键 key 的值的总和。
示例：输入： ["MapSum", "insert", "sum", "insert", "sum"]
```

(续下页)

(接上页)

```
[[], ["apple", 3], ["ap"], ["app", 2], ["ap"]]
输出：[null, null, 3, null, 5]
解释： MapSum mapSum = new MapSum();
mapSum.insert("apple", 3);
mapSum.sum("ap");           // return 3 (apple = 3)
mapSum.insert("app", 2);
mapSum.sum("ap");           // return 5 (apple + app = 3 + 2 = 5)
提示：1 <= key.length, prefix.length <= 50
key 和 prefix 仅由小写英文字母组成
1 <= val <= 1000
最多调用 50 次 insert 和 sum
```

- 解题思路

```
type MapSum struct {
        val  int
        next map[int32]*MapSum
}

func Constructor() MapSum {
        return MapSum{
                val:  0,
                next: make(map[int32]*MapSum),
        }
}

func (this *MapSum) Insert(key string, val int) {
        node := this
        for _, v := range key {
                if _, ok := node.next[v]; ok == false {
                        temp := Constructor()
                        node.next[v] = &temp
                }
                node = node.next[v]
        }
        node.val = val
}

func (this *MapSum) Sum(prefix string) int {
        node := this
        for _, v := range prefix {
                if _, ok := node.next[v]; ok == false {
                        return 0
                }
```

(续下页)

(接上页)

```
                node = node.next[v]
        }
        res := 0
        queue := make([]*MapSum, 0)
        queue = append(queue, node)
        for len(queue) > 0 {
                temp := queue[0]
                queue = queue[1:]
                res = res + temp.val
                for _, v := range temp.next {
                        queue = append(queue, v)
                }
        }
        return res
}

# 2
type MapSum struct {
        m    map[string]int
        data map[string]map[string]bool
}

func Constructor() MapSum {
        return MapSum{
                m:    make(map[string]int),
                data: make(map[string]map[string]bool),
        }
}

func (this *MapSum) Insert(key string, val int) {
        this.m[key] = val
        for i := 1; i <= len(key); i++ {
                str := key[:i]
                if _, ok := this.data[str]; ok == false {
                        this.data[str] = make(map[string]bool)
                }
                this.data[str][key] = true
        }
}

func (this *MapSum) Sum(prefix string) int {
        res := 0
        for key := range this.data[prefix] {
```

(续下页)

(接上页)

```
                res = res + this.m[key]
        }
        return res
}

# 3
type MapSum struct {
        m map[string]int
}

func Constructor() MapSum {
        return MapSum{
                m: make(map[string]int),
        }
}

func (this *MapSum) Insert(key string, val int) {
        this.m[key] = val
}

func (this *MapSum) Sum(prefix string) int {
        res := 0
        for key, value := range this.m {
                if strings.HasPrefix(key, prefix) {
                        res = res + value
                }
        }
        return res
}
```

20.27 678. 有效的括号字符串 (4)

- 题目

```
给定一个只包含三种字符的字符串：（，）和␣
↪*，写一个函数来检验这个字符串是否为有效字符串。
有效字符串具有如下规则：
    任何左括号（必须有相应的右括号）。
    任何右括号）必须有相应的左括号（。
    左括号（必须在对应的右括号之前）。
    * 可以被视为单个右括号），或单个左括号（，或一个空字符串。
    一个空字符串也被视为有效字符串。
```

(续下页)

(接上页)

```
示例 1:输入: "()" 输出: True
示例 2:输入: "(*)" 输出: True
示例 3:输入: "(*))" 输出: True
注意:字符串大小将在 [1, 100] 范围内。
```

- 解题思路

```
func checkValidString(s string) bool {
        // 第1次把星号当左括号看
        left, right := 0, 0
        for i := 0; i < len(s); i++ {
                if s[i] == ')' {
                        right++
                } else {
                        left++
                }
                if right > left {
                        return false
                }
        }
        // 第2次把星号当右括号看
        left, right = 0, 0
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '(' {
                        left++
                } else {
                        right++
                }
                if left > right {
                        return false
                }
        }
        return true
}

# 2
func checkValidString(s string) bool {
        stackL := make([]int, 0)
        stackS := make([]int, 0)
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        stackL = append(stackL, i)
                } else if s[i] == '*' {
                        stackS = append(stackS, i)
```

(续下页)

(接上页)

```
                } else {
                        if len(stackL) > 0 {
                                stackL = stackL[:len(stackL)-1]
                        } else if len(stackS) > 0 {
                                stackS = stackS[:len(stackS)-1]
                        } else {
                                return false
                        }
                }
        }
        if len(stackL) > len(stackS) {
                return false
        }
        for len(stackL) > 0 && len(stackS) > 0 {
                a, b := stackL[len(stackL)-1], stackS[len(stackS)-1]
                if a > b {
                        return false
                }
                stackL = stackL[:len(stackL)-1]
                stackS = stackS[:len(stackS)-1]
        }
        if len(stackL) == 0 {
                return true
        }
        return false
}

# 3
func checkValidString(s string) bool {
        return dfs(s, 0, 0)
}

func dfs(s string, index, count int) bool {
        if count < 0 {
                return false
        }
        for i := index; i < len(s); i++ {
                if s[i] == '(' {
                        count++
                } else if s[i] == ')' {
                        if count == 0 {
                                return false
                        }
```

(续下页)

(接上页)

```
                        count--
                } else if s[i] == '*' {
                        return dfs(s, i+1, count+1) || dfs(s, i+1, count-1) || dfs(s,␣
↪i+1, count)
                }
        }
        return count == 0
}

# 4
func checkValidString(s string) bool {
        maxLeft, minLeft := 0, 0 // 可以有最多left和最少left的数量
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        maxLeft++
                        minLeft++
                } else if s[i] == '*' {
                        maxLeft++        // *当(用
                        if minLeft > 0 { // *当)用
                                minLeft--
                        }
                } else if s[i] == ')' {
                        maxLeft--
                        if maxLeft < 0 {
                                return false
                        }
                        if minLeft > 0 {
                                minLeft--
                        }
                }
        }
        return minLeft == 0
}
```

20.28 684. 冗余连接 (1)

- 题目

```
在本问题中，树指的是一个连通且无环的无向图。
输入一个图，该图由一个有着N个节点 (节点值不重复1, 2, ..., N) 的树及一条附加的边构成。
附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。
结果图是一个以边组成的二维数组。每一个边的元素是一对[u, v]，满足u <␣
```

(续下页)

(接上页)

```
↪v，表示连接顶点u和v的无向图的边。
返回一条可以删去的边，使得结果图是一个有着N个节点的树。
如果有多个答案，则返回二维数组中最后出现的边。答案边[u, v] 应满足相同的格式u < v
示例 1：输入：[[1,2], [1,3], [2,3]] 输出：[2,3]
解释：给定的无向图为：
  1
 / \
2 - 3
示例 2：输入：[[1,2], [2,3], [3,4], [1,4], [1,5]] 输出：[1,4]
解释：给定的无向图为：
5 - 1 - 2
    |   |
    4 - 3
注意:输入的二维数组大小在 3 到 1000。
二维数组中的整数在1到N之间，其中N是输入数组的大小。
```

- 解题思路

```
func findRedundantConnection(edges [][]int) []int {
        n := len(edges) + 1
        fa := make([]int, n)
        for i := 0; i < n; i++ {
                fa[i] = i
        }
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                if find(fa, a) == find(fa, b) {
                        return edges[i]
                }
                union(fa, a, b)
        }
        return nil
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}

func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
                a = fa[a]
        }
        return a
```

(续下页)

(接上页)

```
}
```

20.29 688. “马” 在棋盘上的概率 (2)

- 题目

```
已知一个NxN的国际象棋棋盘，棋盘的行号和列号都是从 0 开始。即最左上角的格子记为(0,␣
↪0)，最右下角的记为(N-1, N-1)。
现有一个 “马”（也译作 “骑士”）位于(r, c)，并打算进行K 次移动。
如下图所示，国际象棋的 “马” 每一步先沿水平或垂直方向移动 2␣
↪个格子，然后向与之相垂直的方向再移动 1 个格子，共有 8 个可选的位置。
现在 “马”␣
↪每一步都从可选的位置（包括棋盘外部的）中独立随机地选择一个进行移动，直到移动了K次或跳到了棋盘外面。
求移动结束后，“马” 仍留在棋盘上的概率。
示例：输入：3, 2, 0, 0 输出：0.0625
解释：输入的数据依次为 N, K, r, c
第 1 步时，有且只有 2 种走法令 “马” 可以留在棋盘上（跳到（1,2）或（2,1））。
对于以上的两种情况，各自在第2步均有且只有2种走法令 “马” 仍然留在棋盘上。
所以 “马” 在结束后仍在棋盘上的概率为 0.0625。
注意：N 的取值范围为 [1, 25]
K的取值范围为 [0, 100]
开始时，“马” 总是位于棋盘上
```

- 解题思路

```
var dx = []int{2, 2, 1, 1, -1, -1, -2, -2}
var dy = []int{1, -1, 2, -2, 2, -2, 1, -1}

func knightProbability(n int, k int, row int, column int) float64 {
        dp := make([][][]float64, n) // dp[i][j][k]在位置[i,j]移动k步
        for i := 0; i < n; i++ {
                dp[i] = make([][]float64, n)
                for j := 0; j < n; j++ {
                        dp[i][j] = make([]float64, k+1)
                }
        }
        dp[row][column][0] = float64(1)
        for a := 1; a <= k; a++ {
                for i := 0; i < n; i++ {
                        for j := 0; j < n; j++ {
                                for b := 0; b < 8; b++ {
                                        x := i + dx[b]
```

(续下页)

(接上页)

```
                                        y := j + dy[b]
                                        if 0 <= x && x < n && 0 <= y && y < n {
                                                dp[i][j][a] = dp[i][j][a] +␣
↪dp[x][y][a-1]/8.0
                                        }
                                }
                        }
                }
        }
        res := float64(0)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        res = res + dp[i][j][k]
                }
        }
        return res
}

# 2
var dx = []int{2, 2, 1, 1, -1, -1, -2, -2}
var dy = []int{1, -1, 2, -2, 2, -2, 1, -1}

func knightProbability(n int, k int, row int, column int) float64 {
        dp := make([][]float64, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]float64, n)
        }
        dp[row][column] = float64(1)
        for a := 1; a <= k; a++ {
                temp := make([][]float64, n)
                for i := 0; i < n; i++ {
                        temp[i] = make([]float64, n)
                }
                for i := 0; i < n; i++ {
                        for j := 0; j < n; j++ {
                                for b := 0; b < 8; b++ {
                                        x := i + dx[b]
                                        y := j + dy[b]
                                        if 0 <= x && x < n && 0 <= y && y < n {
                                                temp[i][j] = temp[i][j] + dp[x][y]/8.0
                                        }
                                }
                        }
```

(续下页)

(接上页)

```
                }
                dp = temp
        }
        res := float64(0)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        res = res + dp[i][j]
                }
        }
        return res
}
```

20.30 692. 前 K 个高频单词 (2)

- 题目

```
给一非空的单词列表，返回前 k 个出现次数最多的单词。
返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率，按字母顺序排序。
示例 1: 输入: ["i", "love", "leetcode", "i", "love", "coding"], k = 2 输出: ["i",
↪"love"]
解析: "i" 和 "love" 为出现次数最多的两个单词，均为2次。
    注意，按字母顺序 "i" 在 "love" 之前。
示例 2:
输入: ["the", "day", "is", "sunny", "the", "the", "the", "sunny", "is", "is"], k = 4
输出: ["the", "is", "sunny", "day"]
解析: "the", "is", "sunny" 和 "day" 是出现次数最多的四个单词，出现次数依次为 4, 3, 2␣
↪和 1 次。
注意: 假定 k 总为有效值， 1 ≤ k ≤ 集合元素数。
    输入的单词均由小写字母组成。
扩展练习： 尝试以 O(n log k) 时间复杂度和 O(n) 空间复杂度解决。
```

- 解题思路

```
func topKFrequent(words []string, k int) []string {
        m := make(map[string]int)
        for _, v := range words {
                m[v]++
        }
        nodeHeap := &Heap{}
        heap.Init(nodeHeap)
        for key, value := range m {
                heap.Push(nodeHeap, Node{
```

(续下页)

(接上页)

```
                        str: key,
                        num: value,
                })
        }
        fmt.Println(nodeHeap)
        var res []string
        for i := 0; i < k; i++ {
                value := heap.Pop(nodeHeap).(Node)
                res = append(res, value.str)
        }
        return res
}

type Node struct {
        str string
        num int
}

type Heap []Node

func (h Heap) Len() int {
        return len(h)
}

func (h Heap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h Heap) Less(i, j int) bool {
        if h[i].num == h[j].num {
                return h[i].str < h[j].str
        }
        return h[i].num > h[j].num
}

func (h *Heap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

func (h *Heap) Push(x interface{}) {
        *h = append(*h, x.(Node))
```

(续下页)

(接上页)

```
}

# 2
func topKFrequent(words []string, k int) []string {
        var res []string
        m := make(map[string]int)
        for _, v := range words {
                m[v]++
        }
        var arr []Node
        for k, v := range m {
                arr = append(arr, Node{
                        str: k,
                        num: v,
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i].num == arr[j].num {
                        return arr[i].str < arr[j].str
                }
                return arr[i].num > arr[j].num
        })
        for i := 0; i < k; i++ {
                res = append(res, arr[i].str)
        }
        return res
}

type Node struct {
        str string
        num int
}
```

20.31 695. 岛屿的最大面积 (2)

- 题目

```
给定一个包含了一些 0 和 1 的非空二维数组 grid 。
一个 岛屿 是由一些相邻的 1 (代表土地) 构成的组合，这里的「相邻」要求两个 1␣
↪必须在水平或者竖直方向上相邻。
你可以假设 grid 的四个边缘都被 0（代表水）包围着。
找到给定的二维数组中最大的岛屿面积。(如果没有岛屿，则返回面积为 0 。)
```

(续下页)

(接上页)

```
示例 1:
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
对于上面这个给定矩阵应返回 6。注意答案不应该是 11␣
↪, 因为岛屿只能包含水平或垂直的四个方向的 1 。
示例 2:[[0,0,0,0,0,0,0,0]]
对于上面这个给定的矩阵, 返回 0。
注意: 给定的矩阵grid 的长度和宽度都不超过 50。
```

- 解题思路

```
func maxAreaOfIsland(grid [][]int) int {
        maxArea := 0
        for i := range grid {
                for j := range grid[i] {
                        maxArea = max(maxArea, getArea(grid, i, j))
                }
        }
        return maxArea
}

func getArea(grid [][]int, i, j int) int {
        if grid[i][j] == 0 {
                return 0
        }
        grid[i][j] = 0
        area := 1
        if i != 0 {
                area = area + getArea(grid, i-1, j)
        }
        if j != 0 {
                area = area + getArea(grid, i, j-1)
        }
        if i != len(grid)-1 {
                area = area + getArea(grid, i+1, j)
        }
        if j != len(grid[0])-1 {
                area = area + getArea(grid, i, j+1)
```

(续下页)

(接上页)

```
        }
        return area
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxAreaOfIsland(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 1 {
                                value := dfs(grid, i, j)
                                if value > res {
                                        res = value
                                }
                        }
                }
        }
        return res
}

func dfs(grid [][]int, i, j int) int {
        if i < 0 || j < 0 || i >= len(grid) || j >= len(grid[0]) ||
                grid[i][j] == 0 {
                return 0
        }
        grid[i][j] = 0
        res := 1
        res = res + dfs(grid, i+1, j)
        res = res + dfs(grid, i-1, j)
        res = res + dfs(grid, i, j+1)
        res = res + dfs(grid, i, j-1)
        return res
}
```

20.32 698. 划分为 k 个相等的子集 (3)

- 题目

```
给定一个整数数组nums 和一个正整数 k，找出是否有可能把这个数组分成 k个非空子集，其总和都相等。
示例 1：输入： nums = [4, 3, 2, 3, 5, 2, 1], k = 4 输出： True
说明： 有可能将其分成 4 个子集（5），（1,4），（2,3），（2,3）等于总和。
提示：1 <= k <= len(nums) <= 16
0 < nums[i] < 10000
```

- 解题思路

```
func canPartitionKSubsets(nums []int, k int) bool {
        if k == 1 {
                return true
        }
        n := len(nums)
        sort.Ints(nums)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
        }
        if sum%k != 0 { // 分不开：false
                return false
        }
        target := sum / k
        if nums[n-1] > target { // 有1个大于平均值：false
                return false
        }
        total := 1 << n
        arr := make([]int, total)
        dp := make([]bool, total)
        dp[0] = true
        for i := 0; i < total; i++ { // 枚举状态
                if dp[i] == false {
                        continue
                }
                for j := 0; j < n; j++ { // 基于当前状态，添加1个数
                        if i&(1<<j) > 0 { // 第j位为1，跳过
                                continue
                        }
                        next := i | (1 << j) // 添加完后的值
                        if dp[next] == true {
```

(续下页)

(接上页)

```
                                continue
                        }
                        if arr[i]+nums[j] <= target {
                                arr[next] = (arr[i] + nums[j])%target
                                dp[next] = true
                        } else {
                                break // 已经排好序，后面会更大
                        }
                }
        }
        return dp[total-1]
}

# 2
func canPartitionKSubsets(nums []int, k int) bool {
        if k == 1 {
                return true
        }
        n := len(nums)
        sort.Ints(nums)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
        }
        if sum%k != 0 { // 分不开：false
                return false
        }
        target := sum / k
        if nums[n-1] > target { // 有1个大于平均值：false
                return false
        }
        return dfs(nums, k, target, 0, 0, make([]bool, n))
}

func dfs(nums []int, k int, target int, index int, count int, visited []bool) bool {
        if k == 0 {
                return true
        }
        if count == target {
                return dfs(nums, k-1, target, 0, 0, visited) // k减少一个，从头开始
        }
        for i := index; i < len(nums); i++ {
                if visited[i] == true { // nums[i]使用过
```

(续下页)

(接上页)

```
                        continue
                }
                if count+nums[i] > target { // 大于目标值
                        continue
                }
                visited[i] = true
                count = count + nums[i]
                if dfs(nums, k, target, i+1, count, visited) == true {
                        return true
                }
                count = count - nums[i]
                visited[i] = false
        }
        return false
}

# 3
func canPartitionKSubsets(nums []int, k int) bool {
        if k == 1 {
                return true
        }
        n := len(nums)
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] > nums[j]
        })
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
        }
        if sum%k != 0 { // 分不开：false
                return false
        }
        target := sum / k
        if nums[0] > target { // 有1个大于平均值：false
                return false
        }
        return dfs(nums, k, target, 0, make([]int, k))
}

// 不做剪枝，需要排序从大到小
func dfs(nums []int, k int, target int, index int, sum []int) bool {
        if index == len(nums) {
                return true
```

(续下页)

(接上页)

```
        }
        for i := 0; i < k; i++ {
                if sum[i] < target && sum[i]+nums[index] <= target {
                        sum[i] = sum[i] + nums[index]
                        if dfs(nums, k, target, index+1, sum) == true {
                                return true
                        }
                        sum[i] = sum[i] - nums[index]
                }
        }
        return false
}
```

CHAPTER 21

0601-0700-Hard

21.1 629.K 个逆序对数组 (2)

- 题目

```
给出两个整数n和k，找出所有包含从1到n的数字，且恰好拥有k个逆序对的不同的数组的个数。
逆序对的定义如下：对于数组的第i个和第j个元素，如果满i<j且a[i]>
↪a[j]，则其为一个逆序对；否则不是。
由于答案可能很大，只需要返回 答案 mod 109+ 7 的值。
示例 1:输入：n = 3, k = 0输出：1
解释：只有数组 [1,2,3] 包含了从1到3的整数并且正好拥有 0 个逆序对。
示例 2:输入：n = 3, k = 1输出：2
解释：数组 [1,3,2] 和 [2,1,3] 都有 1 个逆序对。
说明:n的范围是 [1, 1000] 并且 k 的范围是 [0, 1000]。
```

- 解题思路

```
var mod = 1000000007

func kInversePairs(n int, k int) int {
        dp := make([][]int, n+1) // dp[n][k]表示1-n的排列中，包含k个逆序对
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
                dp[i][0] = 1
        }
```

(续下页)

(接上页)

```
        for i := 1; i <= n; i++ {
                for j := 1; j <= k; j++ {
                        // 前面i-1个数，i个插入位
                        // 插入到最后，不增加： f(i,j) = f(i,j) + f(i-1,j)
                        // 插入到倒数第2个，增加：f(i,j) = f(i,j) + f(i-1,j-1)
                        // ...
                        // 插入到倒数第i个，增加：f(i,j) = f(i,j) + f(i-1,j-i+1)
                        // f(i,j) = f(i-1,j) + f(i-1,j-1) + ... + f(i-1,j-i+1)
                        for l := max(0, j-i+1); l <= j; l++ {
                                dp[i][j] = (dp[i][j] + dp[i-1][l]) % mod
                        }
                }
        }
        return dp[n][k]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 1
var mod = 1000000007

func kInversePairs(n int, k int) int {
        dp := make([][]int, n+1) // dp[n][k]表示1-n的排列中，包含k个逆序对
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
        }
        for i := 1; i <= n; i++ {
                dp[i][0] = 1
                // 最多i*(i-1)/2
                for j := 1; j <= k && j <= i*(i-1)/2; j++ {
                        // 前面i-1个数，i个插入位
                        // 插入到最后，不增加： f(i,j) = f(i,j) + f(i-1,j)
                        // 插入到倒数第2个，增加：f(i,j) = f(i,j) + f(i-1,j-1)
                        // ...
                        // 插入到倒数第i个，增加：f(i,j) = f(i,j) + f(i-1,j-i+1)
                        // => f(i,j) = f(i-1,j) + f(i-1,j-1) + ... + f(i-1,j-i+1)
                        // f(i,j-1) = f(i-1, j-1)+ ...+f(i-1, j-i)
                        // f(i,j) - f(i,j-1) = f(i-1,j)-f(i-1, j-i)
```

(续下页)

(接上页)

```
                        // => f(i, j) = f(i,j-1)+f(i-1,j)-f(i-1,j-i)
                        if j >= i {
                                dp[i][j] = dp[i][j-1] + dp[i-1][j] - dp[i-1][j-i]
                        } else {
                                dp[i][j] = dp[i][j-1] + dp[i-1][j]
                        }
                        if dp[i][j] >= 0 {
                                dp[i][j] = dp[i][j] % mod
                        } else {
                                dp[i][j] = (dp[i][j] + mod) % mod
                        }
                }
        }
        return dp[n][k]
}

# 2
var mod = 1000000007

func kInversePairs(n int, k int) int {
        dp := make([]int, k+1) // dp[k]包含k个逆序对的方案数
        dp[0] = 1
        sum := make([]int, k+2)
        sum[1] = 1
        for i := 1; i <= n; i++ {
                // 最多i*(i-1)/2
                for j := 1; j <= k && j <= i*(i-1)/2; j++ {
                        // 前面i-1个数，i个插入位
                        // 插入到最后，不增加： f(i,j) = f(i,j) + f(i-1,j)
                        // 插入到倒数第2个，增加：f(i,j) = f(i,j) + f(i-1,j-1)
                        // ...
                        // 插入到倒数第i个，增加：f(i,j) = f(i,j) + f(i-1,j-i+1)
                        // => f(i,j) = f(i-1,j) + f(i-1,j-1) + ... + f(i-1,j-i+1)
                        // => f(j) = sum[j+1] - sum[j-i+1]
                        dp[j] = (sum[j+1] - sum[max(0, j-i+1)]) % mod
                }
                for j := 1; j <= k; j++ {
                        sum[j+1] = sum[j] + dp[j]
                }
        }
        return dp[k]
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

21.2 630. 课程表 III(2)

- 题目

```
这里有 n 门不同的在线课程，他们按从 1 到 n编号。
每一门课程有一定的持续上课时间（课程时间）t 以及关闭时间第 d天。
一门课要持续学习 t 天直到第 d 天时要完成，你将会从第 1 天开始。
给出 n 个在线课程用 (t, d) 对表示。你的任务是找出最多可以修几门课。
示例：输入：[[100, 200], [200, 1300], [1000, 1250], [2000, 3200]] 输出：3
解释：这里一共有 4 门课程，但是你最多可以修 3 门：
首先，修第一门课时，它要耗费 100 天，你会在第 100 天完成，在第 101 天准备下门课。
第二，修第三门课时，它会耗费 1000 天，所以你将在第 1100 天的时候完成它，以及在第 1101␣
↪天开始准备下门课程。
第三，修第二门课时，它会耗时 200 天，所以你将会在第 1300 天时完成它。
第四门课现在不能修，因为你将会在第 3300 天完成它，这已经超出了关闭日期。
提示:整数 1 <= d, t, n <= 10,000 。
你不能同时修两门课程。
```

- 解题思路

```
func scheduleCourse(courses [][]int) int {
        sort.Slice(courses, func(i, j int) bool {
                return courses[i][1] < courses[j][1]
        })
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        sum := 0
        for i := 0; i < len(courses); i++ {
                count, endTime := courses[i][0], courses[i][1]
                if sum+count <= endTime { // 时间充足，学习
                        sum = sum + count
                        heap.Push(&intHeap, count)
                } else if intHeap.Len() > 0 && count < intHeap[0] {
                        // 当前花费时间比之前时间最大耗时少，放弃之前最大耗时的课程
                        top := heap.Pop(&intHeap).(int) // 最大放弃
```

(续下页)

(接上页)

```
                        sum = sum - top                 // 减去最大
                        sum = sum + count               // 添加当前
                        heap.Push(&intHeap, count)
                }
        }
        return intHeap.Len()
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func scheduleCourse(courses [][]int) int {
        sort.Slice(courses, func(i, j int) bool {
                return courses[i][1] < courses[j][1]
        })
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        sum := 0
        res := 0
        for i := 0; i < len(courses); i++ {
```

(续下页)

(接上页)

```
                count, endTime := courses[i][0], courses[i][1]
                if sum+count <= endTime { // 时间充足，学习
                        sum = sum + count
                        res++
                        heap.Push(&intHeap, count)
                } else if intHeap.Len() > 0 && count < intHeap[0] {
                        // 当前花费时间比之前时间最大耗时少，放弃之前最大耗时的课程
                        top := heap.Pop(&intHeap).(int) // 最大放弃
                        sum = sum - top                 // 减去最大
                        sum = sum + count               // 添加当前
                        heap.Push(&intHeap, count)
                }
        }
        return res
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

21.3 632. 最小区间 (2)

- 题目

```
你有k个 非递减排列 的整数列表。找到一个 最小␣
↪区间，使得k个列表中的每个列表至少有一个数包含在其中。
我们定义如果b-a < d-c或者在b-a == d-c时a < c，则区间 [a,b] 比 [c,d] 小。
示例 1：输入：nums = [[4,10,15,24,26], [0,9,12,20], [5,18,22,30]] 输出：[20,24]
解释： 列表 1：[4, 10, 15, 24, 26]，24 在区间 [20,24] 中。
列表 2：[0, 9, 12, 20]，20 在区间 [20,24] 中。
列表 3：[5, 18, 22, 30]，22 在区间 [20,24] 中。
示例 2：输入：nums = [[1,2,3],[1,2,3],[1,2,3]] 输出：[1,1]
示例 3：输入：nums = [[10,10],[11,11]] 输出：[10,11]
示例 4：输入：nums = [[10],[11]] 输出：[10,11]
示例 5：输入：nums = [[1],[2],[3],[4],[5],[6],[7]] 输出：[1,7]
提示：nums.length == k
1 <= k <= 3500
1 <= nums[i].length <= 50
-105 <= nums[i][j] <= 105
nums[i] 按非递减顺序排列
```

- 解题思路

```
func smallestRange(nums [][]int) []int {
        nodeHeap := make(NodeHeap, 0) // 堆的大小为n
        heap.Init(&nodeHeap)
        maxValue, n := math.MinInt32, len(nums)
        // 问题可以转化为，从n个列表中各取1个数，使得这n个数中的最大值与最小值的差最小
        for i := 0; i < n; i++ {
                maxValue = max(maxValue, nums[i][0]) // 获取n个数的最大值
                heap.Push(&nodeHeap, Node{Id: i, Value: nums[i][0]})
                nums[i] = nums[i][1:] // 数组缩小，也可以使用下标标记
        }
        res := []int{math.MinInt32 / 10, math.MaxInt32 / 10}
        for { // 从小到大，每从堆取出一个最小值，再从所在组取出下一个较大的数放回去
                node := heap.Pop(&nodeHeap).(Node)       // 小根堆，取最小值
                if maxValue-node.Value < res[1]-res[0] { // 更新范围：最大值-最小值
                        res = []int{node.Value, maxValue}
                }
                if len(nums[node.Id]) == 0 { // 退出条件：某一个数组首先访问完
                        break
                }
                heap.Push(&nodeHeap, Node{Id: node.Id, Value: nums[node.Id][0]})
                maxValue = max(maxValue, nums[node.Id][0]) // 更新最大值
```

(续下页)

(接上页)

```
                nums[node.Id] = nums[node.Id][1:]
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type Node struct {
        Id    int
        Value int
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h NodeHeap) Less(i, j int) bool {
        return h[i].Value < h[j].Value
}

func (h NodeHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
```

(续下页)

(接上页)

```
func smallestRange(nums [][]int) []int {
        n := len(nums)
        m := make(map[int][]int)
        minValue, maxValue := math.MaxInt32, math.MinInt32
        for i := 0; i < n; i++ {
                for j := 0; j < len(nums[i]); j++ {
                        minValue = min(minValue, nums[i][j])
                        maxValue = max(maxValue, nums[i][j])
                        m[nums[i][j]] = append(m[nums[i][j]], i) // 存的是值对应的下标
                }
        }
        res := []int{minValue, maxValue}
        left, right := minValue, minValue // 双指针
        window := make(map[int]int)       // 滑动窗口：包含n个列的时候，更新范围
        for ; right <= maxValue; right++ {
                if len(m[right]) > 0 {
                        for i := 0; i < len(m[right]); i++ {
                                window[m[right][i]]++ // 添加进窗口
                        }
                        for len(window) == n {
                                if right-left < res[1]-res[0] {
                                        res = []int{left, right}
                                }
                                for i := 0; i < len(m[left]); i++ {
                                        window[m[left][i]]--
                                        if window[m[left][i]] == 0 {
                                                delete(window, m[left][i])
                                        }
                                }
                                left++
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

21.4 664. 奇怪的打印机 (3)

- 题目

```
有台奇怪的打印机有以下两个特殊要求:
打印机每次只能打印同一个字符序列。
每次可以在任意起始和结束位置打印新字符，并且会覆盖掉原来已有的字符。
给定一个只包含小写英文字母的字符串，你的任务是计算这个打印机打印它需要的最少次数。
示例 1:输入: "aaabbb" 输出: 2
解释: 首先打印 "aaa" 然后打印 "bbb"。
示例 2:输入: "aba" 输出: 2
解释: 首先打印 "aaa" 然后在第二个位置打印 "b" 覆盖掉原来的字符 'a'。
```

- 解题思路

```
var dp [][]int

func strangePrinter(s string) int {
        n := len(s)
        dp = make([][]int, n) // dp[i][j] => 打印S[i], S[i+1], ..., S[j]所需的次数
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        return dfs(s, 0, n-1)
}

func dfs(s string, i, j int) int {
        if i > j {
                return 0
        }
        if dp[i][j] > 0 {
                return dp[i][j]
        }
        res := dfs(s, i+1, j) + 1 // 单独打印i
        for k := i + 1; k <= j; k++ {
```

(续下页)

(接上页)

```
                if s[i] == s[k] { // 相同的时候，打印i-k
                        res = min(res, dfs(s, i, k-1)+dfs(s, k+1, j))
                }
        }
        dp[i][j] = res
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func strangePrinter(s string) int {
        n := len(s)
        dp := make([][]int, n) // dp[i][j] => 打印S[i], S[i+1], ..., S[j]所需的次数
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                dp[i][i] = 1
        }
        for length := 2; length <= n; length++ {
                for i := 0; i+length-1 < n; i++ {
                        j := i + length - 1
                        dp[i][j] = dp[i+1][j] + 1     // 单独打印i
                        for k := i + 1; k <= j; k++ { // 相同的时候，打印i-k
                                if s[i] == s[k] {
                                        dp[i][j] = min(dp[i][j], dp[i+1][k-
↪1]+dp[k][j])
                                }
                        }
                }
        }
        return dp[0][n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
```

(续下页)

(接上页)

```
}

# 3
func strangePrinter(s string) int {
        n := len(s)
        dp := make([][]int, n+1) // dp[i][j] => 打印S[i], S[i+1], ..., S[j]所需的次数
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
                dp[i][i] = 1
        }
        for length := 2; length <= n; length++ {
                for i := 0; i+length-1 < n; i++ {
                        j := i + length - 1
                        dp[i][j] = dp[i+1][j] + 1
                        for k := i + 1; k <= j; k++ {
                                if s[i] == s[k] {
                                        dp[i][j] = min(dp[i][j], dp[i][k-
↪1]+dp[k+1][j])
                                }
                        }
                }
        }
        return dp[0][n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

21.5 668. 乘法表中第 k 小的数 (1)

- 题目

```
几乎每一个人都用乘法表。但是你能在乘法表中快速找到第k小的数字吗？
给定高度m、宽度n 的一张m * n的乘法表，以及正整数k，你需要返回表中第k小的数字。
例1: 输入: m = 3, n = 3, k = 5 输出: 3
解释: 乘法表:
1        2        3
2        4        6
```

(续下页)

(接上页)

```
3         6         9
第5小的数字是 3 (1, 2, 2, 3, 3).
例 2: 输入: m = 2, n = 3, k = 6 输出: 6
解释: 乘法表:
1         2         3
2         4         6
第6小的数字是 6 (1, 2, 2, 3, 4, 6).
注意: m 和n的范围在 [1, 30000] 之间。
k 的范围在 [1, m * n] 之间。
```

- 解题思路

```
func findKthNumber(m int, n int, k int) int {
        left := 1
        right := m * n
        for left < right {
                mid := left + (right-left)/2
                total := judge(m, n, k, mid)
                if total == true { // 满足条件, 继续找
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

func judge(m int, n int, k int, target int) bool {
        count := 0
        for i := 1; i <= m; i++ {
                count = count + min(target/i, n) // 当前行全部满足+n, 部分满足+target/
↪i
        }
        return count >= k
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

21.6 675. 为高尔夫比赛砍树

21.6.1 题目

```
你被请来给一个要举办高尔夫比赛的树林砍树。树林由一个m x n 的矩阵表示， 在这个矩阵中：
0 表示障碍，无法触碰
1表示地面，可以行走
比 1 大的数表示有树的单元格，可以行走，数值表示树的高度
每一步，你都可以向上、下、左、右四个方向之一移动一个单位，如果你站的地方有一棵树，那么你可以决定是否
你需要按照树的高度从低向高砍掉所有的树，每砍过一颗树，该单元格的值变为␣
↪1（即变为地面）。
你将从（0，0）点开始工作，返回你砍完所有树需要走的最小步数。␣
↪如果你无法砍完所有的树，返回 -1 。
可以保证的是，没有两棵树的高度是相同的，并且你至少需要砍倒一棵树。
示例 1：输入：forest = [[1,2,3],[0,0,4],[7,6,5]] 输出：6
解释：沿着上面的路径，你可以用 6 步，按从最矮到最高的顺序砍掉这些树。
示例 2：输入：forest = [[1,2,3],[0,0,0],[7,6,5]] 输出：-1
解释：由于中间一行被障碍阻塞，无法访问最下面一行中的树。
示例 3：输入：forest = [[2,3,4],[0,0,5],[8,7,6]] 输出：6
解释：可以按与示例 1 相同的路径来砍掉所有的树。
(0,0) 位置的树，可以直接砍去，不用算步数。
提示：m == forest.length
n == forest[i].length
1 <= m, n <= 50
0 <= forest[i][j] <= 109
```

21.6.2 解题思路

21.7 679.24 点游戏

21.7.1 题目

21.7.2 解题思路

21.8 685. 冗余连接 II

21.8.1 题目

```
在本问题中，有根树指满足以下条件的有向图。该树只有一个根节点，所有其他节点都是该根节点的后继。
每一个节点只有一个父节点，除了根节点没有父节点。
输入一个有向图，该图由一个有着N个节点 (节点值不重复1, 2, ..., N)␣
↪的树及一条附加的边构成。
附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。
结果图是一个以边组成的二维数组。
每一个边 的元素是一对 [u, v]，用以表示有向图中连接顶点 u 和顶点 v 的边，其中 u 是 v␣
↪的一个父节点。
返回一条能删除的边，使得剩下的图是有N个节点的有根树。若有多个答案，返回最后出现在给定二维数组的答案。
示例1:输入: [[1,2], [1,3], [2,3]] 输出: [2,3]
解释: 给定的有向图如下:
  1
 / \
v   v
2-->3
示例 2:输入: [[1,2], [2,3], [3,4], [4,1], [1,5]] 输出: [4,1]
解释: 给定的有向图如下:
5 <- 1 -> 2
     ^    |
     |    v
     4 <- 3
注意:二维数组大小的在3到1000范围内。
二维数组中的每个整数在1到N之间，其中 N 是二维数组的大小。
```

21.8.2 解题思路

21.9 689. 三个无重叠子数组的最大和

21.9.1 题目

给你一个整数数组 nums 和一个整数 k ，找出三个长度为 k 、互不重叠、且全部数字和（3 * k 项）最大的子数组，并返回这三个子数组。
以下标的数组形式返回结果，数组中的每一项分别指示每个子数组的起始位置（下标从 0 开始）。如果有多个结果，返回字典序最小的一个。
示例 1：输入：nums = [1,2,1,2,6,7,5,1], k = 2 输出：[0,3,5]
解释：子数组 [1, 2], [2, 6], [7, 5] 对应的起始下标为 [0, 3, 5]。
也可以取 [2, 1], 但是结果 [1, 3, 5] 在字典序上更大。
示例 2：输入：nums = [1,2,1,2,1,2,1,2,1], k = 2 输出：[0,2,4]
提示：1 <= nums.length <= 2 * 104
1 <= nums[i] <216
1 <= k <= floor(nums.length / 3)

21.9.2 解题思路

CHAPTER 22

0701-0800-Easy

22.1 703. 数据流中的第 K 大元素 (2)

- 题目

```
设计一个找到数据流中第K大元素的类（class）。注意是排序后的第K大元素，不是第K个不同的元素。
你的 KthLargest 类需要一个同时接收整数 k 和整数数组nums␣
↪的构造器，它包含数据流中的初始元素。
每次调用 KthLargest.add，返回当前数据流中第K大的元素。

示例:
int k = 3;
int[] arr = [4,5,8,2];
KthLargest kthLargest = new KthLargest(3, arr);
kthLargest.add(3);   // returns 4
kthLargest.add(5);   // returns 5
kthLargest.add(10);  // returns 5
kthLargest.add(9);   // returns 8
kthLargest.add(4);   // returns 8
说明:
你可以假设 nums 的长度≥ k-1 且k ≥ 1。
```

- 解题思路

```
type KthLargest struct {
        k    int
        heap intHeap
}

func Constructor(k int, nums []int) KthLargest {
        h := intHeap(nums)
        heap.Init(&h)

        for len(h) > k {
                heap.Pop(&h)
        }
        return KthLargest{
                k:    k,
                heap: h,
        }
}

func (k *KthLargest) Add(val int) int {
        heap.Push(&k.heap, val)
        if len(k.heap) > k.k {
                heap.Pop(&k.heap)
        }
        return k.heap[0]
}

// 内置heap，实现接口
/*
type Interface interface {
        sort.Interface
        Push(x interface{}) // add x as element Len()
        Pop() interface{}   // remove and return element Len() - 1.
}
*/
type intHeap []int

func (h intHeap) Len() int {
        return len(h)
}

func (h intHeap) Less(i, j int) bool {
        return h[i] < h[j]
}
```

(续下页)

(接上页)

```
func (h intHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *intHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *intHeap) Pop() interface{} {
        res := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return res
}

#
type KthLargest struct {
        nums []int
        k    int
}

func Constructor(k int, nums []int) KthLargest {
        if k < len(nums) {
                sort.Ints(nums)
                nums = nums[len(nums)-k:]
        }
        // 向上调整
        Up(nums)
        return KthLargest{
                nums: nums,
                k:    k,
        }
}

func (k *KthLargest) Add(val int) int {
        if k.k > len(k.nums) {
                k.nums = append(k.nums, val)
                Up(k.nums)
        } else {
                if val > k.nums[0] {
                        // 在堆顶，向下调整
                        k.nums[0] = val
                        Down(k.nums, 0)
                }
```

(续下页)

(接上页)

```
        }
        return k.nums[0]
}

func Down(nums []int, index int) {
        length := len(nums)
        minIndex := index
        for {
                left := 2*index + 1
                right := 2*index + 2
                if left < length && nums[left] < nums[minIndex] {
                        minIndex = left
                }
                if right < length && nums[right] < nums[minIndex] {
                        minIndex = right
                }
                if minIndex == index {
                        break
                }
                swap(nums, index, minIndex)
                index = minIndex
        }
}

func Up(nums []int) {
        length := len(nums)
        for i := length/2 - 1; i >= 0; i-- {
                minIndex := i
                left := 2*i + 1
                right := 2*i + 2
                if left < length && nums[left] < nums[minIndex] {
                        minIndex = left
                }
                if right < length && nums[right] < nums[minIndex] {
                        minIndex = right
                }
                if i != minIndex {
                        swap(nums, i, minIndex)
                }
        }
}

func swap(nums []int, i, j int) {
```

(续下页)

(接上页)

```
        nums[i], nums[j] = nums[j], nums[i]
}
```

22.2 704. 二分查找 (3)

- 题目

```
给定一个 n 个元素有序的（升序）整型数组 nums 和一个目标值 target  ，
写一个函数搜索 nums 中的 target，如果目标值存在返回下标，否则返回 -1。
示例 1:输入: nums = [-1,0,3,5,9,12], target = 9 输出: 4
解释: 9 出现在 nums 中并且下标为 4
示例 2:输入: nums = [-1,0,3,5,9,12], target = 2 输出: -1
解释: 2 不存在 nums 中因此返回 -1
提示：
    你可以假设 nums 中的所有元素是不重复的。
    n 将在 [1, 10000]之间。
    nums 的每个元素都将在 [-9999, 9999]之间。
```

- 解题思路

```
func search(nums []int, target int) int {
        left, right:= 0, len(nums)-1
        for left <= right {
                mid := left + (right-left) / 2
                switch {
                case nums[mid] < target:
                        left = mid + 1
                case nums[mid] > target:
                        right = mid - 1
                default:
                        return mid
                }
        }
        return -1
}

#
func search(nums []int, target int) int {
        if nums[0] > target || nums[len(nums)-1] < target {
                return -1
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        return i
                }
                if nums[i] > target {
                        return -1
                }
        }
        return -1
}

#
func search(nums []int, target int) int {
        if len(nums) == 0 {
                return -1
        }
        mid := len(nums) / 2
        if nums[mid] == target {
                return mid
        } else if nums[mid] > target {
                return search(nums[:mid], target)
        } else {
                result := search(nums[mid+1:], target)
                if result == -1 {
                        return result
                }
                return mid + 1 + result
        }
}
```

22.3 705. 设计哈希集合 (2)

- 题目

```
不使用任何内建的哈希表库设计一个哈希集合
具体地说，你的设计应该包含以下的功能
    add(value)：向哈希集合中插入一个值。
    contains(value) ：返回哈希集合中是否存在这个值。
    remove(value)：将给定值从哈希集合中删除。如果哈希集合中没有这个值，什么也不做。
示例：
MyHashSet hashSet = new MyHashSet();
hashSet.add(1);
```

(续下页)

(接上页)

```
hashSet.add(2);
hashSet.contains(1);    // 返回 true
hashSet.contains(3);    // 返回 false (未找到)
hashSet.add(2);
hashSet.contains(2);    // 返回 true
hashSet.remove(2);
hashSet.contains(2);    // 返回  false (已经被删除)
注意:
    所有的值都在 [0, 1000000]的范围内。
    操作的总数目在[1, 10000]范围内。
    不要使用内建的哈希集合库。
```

- 解题思路

```
type MyHashSet struct {
        table []bool
}

func Constructor() MyHashSet {
        return MyHashSet{
                table: make([]bool, 1000001),
        }
}

func (m *MyHashSet) Add(key int) {
        m.table[key] = true
}

func (m *MyHashSet) Remove(key int) {
        m.table[key] = false
}

func (m *MyHashSet) Contains(key int) bool {
        return m.table[key]
}

#
type MyHashSet struct {
        table [10000][]int
}

func Constructor() MyHashSet {
        return MyHashSet{
                table: [10000][]int{},
```

(续下页)

(接上页)

```
        }
}

func (m *MyHashSet) Add(key int) {
        for _, v := range m.table[key%10000] {
                if v == key {
                        return
                }
        }
        m.table[key%10000] = append(m.table[key%10000], key)
}

func (m *MyHashSet) Remove(key int) {
        for k, v := range m.table[key%10000] {
                if v == key {
                        m.table[key%10000] = append(m.table[key%10000][:k], m.
↪table[key%10000][k+1:]...)
                }
        }
}

func (m *MyHashSet) Contains(key int) bool {
        for _, v := range m.table[key%10000] {
                if v == key {
                        return true
                }
        }
        return false
}
```

22.4 706. 设计哈希映射 (2)

- 题目

```
具体地说，你的设计应该包含以下的功能
    put(key, value)：向哈希映射中插入(键,
↪值)的数值对。如果键对应的值已经存在，更新这个值。
    get(key)：返回给定的键所对应的值，如果映射中不包含这个键，返回-1。
    remove(key)：如果映射中存在这个键，删除这个数值对。
示例：
MyHashMap hashMap = new MyHashMap();
hashMap.put(1, 1);
```

(续下页)

(接上页)

```
hashMap.put(2, 2);
hashMap.get(1);            // 返回 1
hashMap.get(3);            // 返回 -1 (未找到)
hashMap.put(2, 1);         // 更新已有的值
hashMap.get(2);            // 返回 1
hashMap.remove(2);         // 删除键为2的数据
hashMap.get(2);            // 返回 -1 (未找到)
注意：
    所有的值都在 [0, 1000000]的范围内。
    操作的总数目在[1, 10000]范围内。
    不要使用内建的哈希库。
```

- 解题思路

```
type MyHashMap struct {
        table []int
}

func Constructor() MyHashMap {
        return MyHashMap{
                table: make([]int, 1000001),
        }
}

func (this *MyHashMap) Put(key int, value int) {
        this.table[key] = value + 1
}

func (this *MyHashMap) Get(key int) int {
        return this.table[key] - 1
}

func (this *MyHashMap) Remove(key int) {
        this.table[key] = 0
}

#
type MyHashMap struct {
        keys  [10000][]int
        value [10000][]int
}

func Constructor() MyHashMap {
        return MyHashMap{
```

(续下页)

(接上页)

```
                keys:  [10000][]int{},
                value: [10000][]int{},
        }
}

func (m *MyHashMap) Put(key int, value int) {
        for k, v := range m.keys[key%10000] {
                if v == key {
                        m.value[key%10000][k] = value
                        return
                }
        }
        m.keys[key%10000] = append(m.keys[key%10000], key)
        m.value[key%10000] = append(m.value[key%10000], value)
}

func (m *MyHashMap) Get(key int) int {
        for k, v := range m.keys[key%10000] {
                if v == key {
                        return m.value[key%10000][k]
                }
        }
        return -1
}

func (m *MyHashMap) Remove(key int) {
        for k, v := range m.keys[key%10000] {
                if v == key {
                        m.keys[key%10000] = append(m.keys[key%10000][:k], m.keys[key
↪%10000][k+1:]...)
                        m.value[key%10000] = append(m.value[key%10000][:k], m.
↪value[key%10000][k+1:]...)
                }
        }
}
```

22.5 709. 转换成小写字母 (2)

- 题目

```
实现函数 ToLowerCase()，该函数接收一个字符串参数 str，
并将该字符串中的大写字母转换成小写字母，之后返回新的字符串。

示例 1: 输入: "Hello" 输出: "hello"
示例 2: 输入: "here" 输出: "here"
示例 3: 输入: "LOVELY" 输出: "lovely"
```

- 解题思路

```
func toLowerCase(str string) string {
        return strings.ToLower(str)
}

#
func toLowerCase(str string) string {
        arr := []byte(str)
        for i := 0; i < len(arr); i++{
                if arr[i] >='A' && arr[i] <= 'Z'{
                        arr[i] = arr[i] - 'A' + 'a'
                }
        }
        return string(arr)
}
```

22.6 717.1 比特与 2 比特字符 (3)

- 题目

```
现给一个由若干比特组成的字符串。问最后一个字符是否必定为一个一比特字符。给定的字符串总是由0结束。
示例 1:输入: bits = [1, 0, 0] 输出: True
解释: 唯一的编码方式是一个两比特字符和一个一比特字符。所以最后一个字符是一比特字符。
示例 2:输入: bits = [1, 1, 1, 0] 输出: False
解释: 唯一的编码方式是两比特字符和两比特字符。所以最后一个字符不是一比特字符。
注意:
    1 <= len(bits) <= 1000.
    bits[i] 总是0 或 1.
```

- 解题思路

```
func isOneBitCharacter(bits []int) bool {
        n := len(bits)
        i := 0
        for i < n-1 {
                // 逢1加2,0加1位
                if bits[i] == 1 {
                        i = i + 2
                } else {
                        i++
                }
        }
        return i == n-1
}

#
func isOneBitCharacter(bits []int) bool {
        n := len(bits)
        count := 0
        // 统计末尾1的个数，偶数正确，奇数错误
        for i := n - 2; i >= 0; i-- {
                if bits[i] == 0 {
                        break
                } else {
                        count++
                }
        }
        // return count & 1 == 0
        return count%2 == 0
}

#
func isOneBitCharacter(bits []int) bool {
        return helper(bits, 0)
}

func helper(bits []int, left int) bool {
        if left == len(bits)-1 {
                return bits[left] == 0
        }
        if left < len(bits)-1 {
                if bits[left] == 0 {
                        return helper(bits, left+1)
                }
                if bits[left] == 1 {
```

(续下页)

(接上页)

```
                        return helper(bits, left+2)
                }
        }
        return false
}
```

22.7 720. 词典中最长的单词 (2)

- 题目

```
给出一个字符串数组words组成的一本英语词典。
从中找出最长的一个单词，该单词是由words词典中其他单词逐步添加一个字母组成。
若其中有多个可行的答案，则返回答案中字典序最小的单词。
若无答案，则返回空字符串。

示例 1:输入: words = ["w","wo","wor","worl", "world"] 输出: "world"
解释: 单词"world"可由"w", "wo", "wor", 和 "worl"添加一个字母组成。
示例 2:输入: words = ["a", "banana", "app", "appl", "ap", "apply", "apple"] 输出:
↪"apple"
解释:  "apply"和"apple"都能由词典中的单词组成。但是"apple"得字典序小于"apply"。
注意:
    所有输入的字符串都只包含小写字母。
    words数组长度范围为[1,1000]。
    words[i]的长度范围为[1,30]。
```

- 解题思路

```
func longestWord(words []string) string {
        if len(words) == 0 {
                return ""
        } else if len(words) == 1 && len(words[0]) > 1 {
                return ""
        }
        sort.Strings(words)
        m := make(map[string]bool)
        res := words[0]
        for _, w := range words {
                n := len(w)
                if n == 1 {
                        m[w] = true
                } else if m[w[:n-1]] {
                        m[w] = true
```

(续下页)

(接上页)

```
                        if len(res) < len(w) {
                                res = w
                        }
                }
        }
        return res
}

#
type Trie struct {
        children [26]*Trie
        index    int
}

func Constructor(str string) Trie {
        return Trie{}
}

func (t *Trie) insert(str string, index int) {
        cur := t
        for i := 0; i < len(str); i++ {
                j := str[i] - 'a'
                if cur.children[j] == nil {
                        cur.children[j] = &Trie{}
                }
                cur = cur.children[j]
        }
        cur.index = index
}

func (t *Trie) bfs(words []string) string {
        res := ""
        stack := make([]*Trie, 0)
        stack = append(stack, t)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]

                if node.index > 0 || node == t {
                        if node != t {
                                word := words[node.index-1]
                                if len(word) > len(res) || (len(word) == len(res) &&␣
↪word < res) {
```

(续下页)

(接上页)

```
                                        res = word
                                }
                        }
                        for i := 0; i < len(node.children); i++ {
                                if node.children[i] != nil {
                                        stack = append(stack, node.children[i])
                                }
                        }
                }
        }
        return res
}

func longestWord(words []string) string {
        t := Trie{}
        for i := 0; i < len(words); i++ {
                t.insert(words[i], i+1)
        }
        return t.bfs(words)
}
```

22.8 724. 寻找数组的中心索引 (2)

- 题目

```
给定一个整数类型的数组 nums，请编写一个能够返回数组“中心索引”的方法。
我们是这样定义数组中心索引的：数组中心索引的左侧所有元素相加的和等于右侧所有元素相加的和。
如果数组不存在中心索引，那么我们应该返回 -
↪1。如果数组有多个中心索引，那么我们应该返回最靠近左边的那一个。

示例 1:输入: nums = [1, 7, 3, 6, 5, 6] 输出: 3
解释:  索引3 (nums[3] = 6) 的左侧数之和(1 + 7 + 3 = 11)，与右侧数之和(5 + 6 =␣
↪11)相等。
同时，3 也是第一个符合要求的中心索引。
示例 2:输入: nums = [1, 2, 3] 输出: -1
解释: 数组中不存在满足此条件的中心索引。
说明:
    nums 的长度范围为 [0, 10000]。
    任何一个 nums[i] 将会是一个范围在 [-1000, 1000]的整数。
```

- 解题思路

```
func pivotIndex(nums []int) int {
        sum := 0
        for i := range nums {
                sum = sum + nums[i]
        }
        left := 0
        for i := range nums {
                if left*2+nums[i] == sum {
                        return i
                }
                left = left + nums[i]
        }
        return -1
}

#
func pivotIndex(nums []int) int {
        if len(nums) == 0 {
                return -1
        }
        arr := make([]int, len(nums))
        arr[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                arr[i] = arr[i-1] + nums[i]
        }
        for i := 0; i < len(nums); i++ {
                var left, right int
                if i == 0 {
                        left = 0
                } else {
                        left = arr[i-1]
                }
                r := i + 1
                if r > len(nums)-1 {
                        right = 0
                } else {
                        right = arr[len(nums)-1] - arr[i]
                }
                if left == right {
                        return i
                }
        }
        return -1
}
```

22.9 728. 自除数 (2)

- 题目

```
自除数 是指可以被它包含的每一位数除尽的数。
例如，128 是一个自除数，因为 128 % 1 == 0，128 % 2 == 0，128 % 8 == 0。
还有，自除数不允许包含 0 。
给定上边界和下边界数字，输出一个列表，列表的元素是边界（含边界）内所有的自除数。
示例 1：输入： 上边界left = 1, 下边界right = 22
输出： [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]
注意：
    每个输入参数的边界满足 1 <= left <= right <= 10000。
```

- 解题思路

```
func selfDividingNumbers(left int, right int) []int {
        res := make([]int, 0)
        for i := left; i <= right; i++ {
                if isSelfDividing(i) {
                        res = append(res, i)
                }
        }
        return res
}

func isSelfDividing(n int) bool {
        temp := n
        for temp > 0 {
                d := temp % 10
                temp = temp / 10
                if d == 0 || n%d != 0 {
                        return false
                }
        }
        return true
}

#
func selfDividingNumbers(left int, right int) []int {
        res := make([]int, 0)
        for i := left; i <= right; i++ {
                if isSelfDividing(i) {
                        res = append(res, i)
                }
```

(续下页)

(接上页)

```
        }
        return res
}

func isSelfDividing(n int) bool {
        str := strconv.Itoa(n)
        for _, v := range str{
                if v == '0' || int32(n) % (v-'0') != 0{
                        return false
                }
        }
        return true
}
```

22.10 733. 图像渲染 (2)

- 题目

```
有一幅以二维整数数组表示的图画，每一个整数表示该图画的像素值大小，数值在 0 到 65535之间。
给你一个坐标 (sr, sc) 表示图像渲染开始的像素值（行 ，列）和一个新的颜色值newColor，让你重新上色这幅图像。
为了完成上色工作，从初始坐标开始，记录初始坐标的上下左右四个方向上像素值与初始坐标相同的相连像素点，
接着再记录这四个方向上符合条件的像素点与他们对应四个方向上像素值与初始坐标相同的相连像素点，……，重
将所有有记录的像素点的颜色值改为新的颜色值。
最后返回经过上色渲染后的图像。

示例 1:输入: image = [[1,1,1],[1,1,0],[1,0,1]] sr = 1, sc = 1, newColor = 2
输出: [[2,2,2],[2,2,0],[2,0,1]]
解析: 在图像的正中间，(坐标(sr,sc)=(1,1)),
在路径上所有符合条件的像素点的颜色都被更改成2。
注意，右下角的像素没有更改为2，
因为它不是在上下左右四个方向上与初始点相连的像素点。

注意:
    image 和 image[0] 的长度在范围 [1, 50] 内。
    给出的初始点将满足 0 <= sr < image.length 和 0 <= sc < image[0].length。
    image[i][j] 和 newColor 表示的颜色值在范围 [0, 65535]内。
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func floodFill(image [][]int, sr int, sc int, newColor int) [][]int {
        oldColor := image[sr][sc]
        if oldColor == newColor {
                return image
        }
        m, n := len(image), len(image[0])
        list := make([][]int, 1)
        list[0] = []int{sr, sc}

        for len(list) > 0 {
                node := list[0]
                list = list[1:]
                image[node[0]][node[1]] = newColor
                for i := 0; i < 4; i++ {
                        x := node[0] + dx[i]
                        y := node[1] + dy[i]
                        if 0 <= x && x < m && 0 <= y && y < n &&
                                image[x][y] == oldColor {
                                list = append(list, []int{x, y})
                        }
                }
        }
        return image
}

#
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func floodFill(image [][]int, sr int, sc int, newColor int) [][]int {
        if sr < 0 || sc < 0 || sr >= len(image) ||
                sc >= len(image[sr]) || image[sr][sc] == newColor {
                return image
        }
        oldColor := image[sr][sc]
        image[sr][sc] = newColor
        for i := 0; i < 4; i++ {
                x := sr + dx[i]
                y := sc + dy[i]
                if 0 <= x && x < len(image) && 0 <= y && y < len(image[x]) &&
                        image[x][y] == oldColor {
```

(续下页)

(接上页)

```
                        floodFill(image, x, y, newColor)
                }
        }
        return image
}
```

22.11 744. 寻找比目标字母大的最小字母 (3)

- 题目

```
给你一个排序后的字符列表 letters ，列表中只包含小写英文字母。
另给出一个目标字母 target，请你寻找在这一有序列表里比目标字母大的最小字母。
在比较时，字母是依序循环出现的。举个例子：
    如果目标字母 target = 'z' 并且字符列表为 letters = ['a', 'b']，则答案返回 'a'
示例：
输入:letters = ["c", "f", "j"] target = "a" 输出: "c"
输入:letters = ["c", "f", "j"] target = "c" 输出: "f"
输入:letters = ["c", "f", "j"] target = "d" 输出: "f"
输入:letters = ["c", "f", "j"] target = "g" 输出: "j"
输入:letters = ["c", "f", "j"] target = "j" 输出: "c"
输入:letters = ["c", "f", "j"] target = "k" 输出: "c"
提示：
    letters长度范围在[2, 10000]区间内。
    letters 仅由小写字母组成，最少包含两个不同的字母。
    目标字母target 是一个小写字母。
```

- 解题思路

```
func nextGreatestLetter(letters []byte, target byte) byte {
        for i := 0; i < len(letters); i++ {
                if letters[i] > target {
                        return letters[i]
                }
        }
        return letters[0]
}

#
func nextGreatestLetter(letters []byte, target byte) byte {
        n := len(letters)
        i := sort.Search(n, func(i int) bool {
                return target < letters[i]
```

(续下页)

(接上页)

```
        })
        return letters[i%n]
}

#
func nextGreatestLetter(letters []byte, target byte) byte {
        left := 0
        right := len(letters) - 1
        for left <= right {
                mid := left + (right-left)/2
                if letters[mid] <= target {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return letters[left%len(letters)]
}
```

22.12 746. 使用最小花费爬楼梯 (3)

- 题目

```
数组的每个索引做为一个阶梯，第 i个阶梯对应着一个非负数的体力花费值␣
↪cost[i](索引从0开始)。
每当你爬上一个阶梯你都要花费对应的体力花费值，然后你可以选择继续爬一个阶梯或者爬两个阶梯。
您需要找到达到楼层顶部的最低花费。在开始时，你可以选择从索引为 0 或 1␣
↪的元素作为初始阶梯。
示例 1:输入: cost = [10, 15, 20] 输出: 15
解释: 最低花费是从cost[1]开始，然后走两步即可到阶梯顶，一共花费15。
示例 2:输入: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]输出: 6
解释: 最低花费方式是从cost[0]开始，逐个经过那些1，跳过cost[3]，一共花费6。
注意:
    cost 的长度将会在 [2, 1000]。
    每一个 cost[i] 将会是一个Integer类型，范围为 [0, 999]。
```

- 解题思路

```
/*
用dp[i]表示爬i个台阶所需要的成本，所以dp[0]=0，dp[1]=0
每次爬i个楼梯，计算的都是从倒数第一个结束，还是从倒数第二个结束
动态转移方程为:
```

(续下页)

(接上页)

```
dp[i] = min{dp[i-2]+cost[i-2] , dp[i-1]+cost[i-1]};
*/
func minCostClimbingStairs(cost []int) int {
        n := len(cost)
        dp := make([]int, n+1)
        dp[0] = 0
        dp[1] = 0
        for i := 2; i <= n; i++ {
                dp[i] = min(dp[i-1]+cost[i-1], dp[i-2]+cost[i-2])
        }
        return dp[n]
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

#
func minCostClimbingStairs(cost []int) int {
        a := 0
        b := 0
        for i := 2; i <= len(cost); i++ {
                a, b = b, min(b+cost[i-1], a+cost[i-2])
        }
        return b
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

#
var arr []int

func minCostClimbingStairs(cost []int) int {
        arr = make([]int, len(cost)+1)
        return ClimbingStais(cost, len(cost))
```

(续下页)

(接上页)

```
}

func ClimbingStais(cost []int, i int) int {
        if i == 0 || i == 1 {
                return 0
        }
        if arr[i] == 0 {
                arr[i] = min(ClimbingStais(cost, i-1)+cost[i-1],
                        ClimbingStais(cost, i-2)+cost[i-2])
        }
        return arr[i]
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}
```

22.13 747. 至少是其他数字两倍的最大数 (3)

- 题目

```
在一个给定的数组nums中，总是存在一个最大元素 。
查找数组中的最大元素是否至少是数组中每个其他数字的两倍。
如果是，则返回最大元素的索引，否则返回-1。

示例 1:输入: nums = [3, 6, 1, 0] 输出: 1
解释: 6是最大的整数, 对于数组中的其他整数,6大于数组中其他元素的两倍。
6的索引是1, 所以我们返回1.
示例 2:输入: nums = [1, 2, 3, 4] 输出: -1
解释: 4没有超过3的两倍大, 所以我们返回 -1.
提示:
    nums 的长度范围在[1, 50].
    每个 nums[i] 的整数范围在 [0, 100].
```

- 解题思路

```
func dominantIndex(nums []int) int {
        n := len(nums)
        if n == 1 {
```

(续下页)

(接上页)

```
                return 0
        }
        maxIndex, secondMaxIndex := 0, 1
        if nums[maxIndex] < nums[secondMaxIndex] {
                maxIndex, secondMaxIndex = secondMaxIndex, maxIndex
        }

        for i := 2; i < n; i++ {
                if nums[maxIndex] < nums[i] {
                        maxIndex, secondMaxIndex = i, maxIndex
                } else if nums[secondMaxIndex] < nums[i] {
                        secondMaxIndex = i
                }
        }
        if nums[maxIndex] >= 2*nums[secondMaxIndex] {
                return maxIndex
        }
        return -1
}

#
func dominantIndex(nums []int) int {
        n := len(nums)
        if n == 1 {
                return 0
        }
        maxValue := nums[0]
        index := 0

        for i := 1; i < n; i++ {
                if nums[i] > maxValue {
                        maxValue = nums[i]
                        index = i
                }
        }
        for i := 0; i < n; i++ {
                if i == index {
                        continue
                }
                if nums[i]*2 > maxValue {
                        return -1
                }
        }
```

(续下页)

(接上页)

```
        return index
}

#
func dominantIndex(nums []int) int {
        n := len(nums)
        if n == 1 {
                return 0
        }
        temp := make([]int, len(nums))
        copy(temp, nums)
        sort.Ints(temp)
        maxValue := temp[len(temp)-1]
        if maxValue < 2*temp[len(temp)-2] {
                return -1
        }
        for i := 0; i < n; i++ {
                if nums[i] == maxValue {
                        return i
                }
        }
        return -1
}
```

22.14 748. 最短完整词 (3)

- 题目

```
如果单词列表（words）中的一个单词包含牌照（licensePlate）中所有的字母，那么我们称之为完整词。
在所有完整词中，最短的单词我们称之为最短完整词。
单词在匹配牌照中的字母时不区分大小写，比如牌照中的 "P" 依然可以匹配单词中的 "p" 字母。
我们保证一定存在一个最短完整词。当有多个单词都符合最短完整词的匹配条件时取单词列表中最靠前的一个。
牌照中可能包含多个相同的字符，比如说：对于牌照 "PP"，单词 "pair" 无法匹配，但是
↪"supper" 可以匹配。

示例 1：输入：licensePlate = "1s3 PSt", words = ["step", "steps", "stripe", "stepple"]
输出："steps"
说明：最短完整词应该包括 "s"、"p"、"s" 以及 "t"。
对于 "step" 它只包含一个 "s"␣
↪所以它不符合条件。同时在匹配过程中我们忽略牌照中的大小写。
示例 2：输入：licensePlate = "1s3 456", words = ["looks", "pest", "stew", "show"]
输出："pest"
```

(续下页)

(接上页)

```
说明：存在 3 个包含字母 "s" 且有着最短长度的完整词，但我们返回最先出现的完整词。
注意：
    牌照（licensePlate）的长度在区域[1, 7]中。
    牌照（licensePlate）将会包含数字、空格、或者字母（大写和小写）。
    单词列表（words）长度在区间 [10, 1000] 中。
    每一个单词 words[i] 都是小写，并且长度在区间 [1, 15] 中。
```

- 解题思路

```
func shortestCompletingWord(licensePlate string, words []string) string {
        m := make(map[byte]int)
        licensePlate = strings.ToLower(licensePlate)
        for i := 0; i < len(licensePlate); i++ {
                if licensePlate[i] >= 'a' && licensePlate[i] <= 'z' {
                        m[licensePlate[i]]++
                }
        }
        res := ""
        for i := 0; i < len(words); i++ {
                if len(words[i]) >= len(res) && res != "" {
                        continue
                }
                tempM := make(map[byte]int)
                for j := 0; j < len(words[i]); j++ {
                        tempM[words[i][j]]++
                }
                flag := true
                for k := range m {
                        if tempM[k] < m[k] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = words[i]
                }
        }
        return res
}

#
func shortestCompletingWord(licensePlate string, words []string) string {
        m := make([]int, 26)
        licensePlate = strings.ToLower(licensePlate)
```

(续下页)

(接上页)

```
        for i := 0; i < len(licensePlate); i++ {
                if licensePlate[i] >= 'a' && licensePlate[i] <= 'z' {
                        m[licensePlate[i]-'a']++
                }
        }
        res := ""
        for i := 0; i < len(words); i++ {
                if len(words[i]) >= len(res) && res != "" {
                        continue
                }
                tempM := make([]int, 26)
                for j := 0; j < len(words[i]); j++ {
                        tempM[words[i][j]-'a']++
                }
                flag := true
                for k := range m {
                        if tempM[k] < m[k] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = words[i]
                }
        }
        return res
}

#
func shortestCompletingWord(licensePlate string, words []string) string {
        m := make([]int, 26)
        licensePlate = strings.ToLower(licensePlate)
        for i := 0; i < len(licensePlate); i++ {
                if licensePlate[i] >= 'a' && licensePlate[i] <= 'z' {
                        m[licensePlate[i]-'a']++
                }
        }
        var lists [16][]string
        for _, word := range words {
                lists[len(word)] = append(lists[len(word)], word)
        }
        for _, list := range lists {
                for _, word := range list {
```

(续下页)

(接上页)

```
                        tempM := make([]int, 26)
                        for i := 0; i < len(word); i++ {
                                tempM[word[i]-'a']++
                        }
                        flag := true
                        for k := range m {
                                if tempM[k] < m[k] {
                                        flag = false
                                        break
                                }
                        }
                        if flag == true {
                                return word
                        }
                }
        }
        return ""
}
```

22.15 762. 二进制表示中质数个计算置位 (2)

- 题目

```
给定两个整数 L 和 R ，找到闭区间 [L, R] 范围内，计算置位位数为质数的整数个数。
（注意，计算置位代表二进制表示中1的个数。例如 21 的二进制表示 10101 有 3␣
↪个计算置位。还有，1 不是质数。）
示例 1: 输入: L = 6, R = 10 输出: 4
解释:
6 -> 110 (2 个计算置位，2 是质数)
7 -> 111 (3 个计算置位，3 是质数)
9 -> 1001 (2 个计算置位，2 是质数)
10-> 1010 (2 个计算置位，2 是质数)

示例 2: 输入: L = 10, R = 15 输出: 5
解释:
10 -> 1010 (2 个计算置位，2 是质数)
11 -> 1011 (3 个计算置位，3 是质数)
12 -> 1100 (2 个计算置位，2 是质数)
13 -> 1101 (3 个计算置位，3 是质数)
14 -> 1110 (3 个计算置位，3 是质数)
15 -> 1111 (4 个计算置位，4 不是质数)
```

(续下页)

(接上页)

```
注意：
    L, R 是 L <= R 且在 [1, 10^6] 中的整数。
    R - L 的最大值为 10000。
```

- 解题思路

```
func countPrimeSetBits(L int, R int) int {
        primes := [...]int{
                2:  1,
                3:  1,
                5:  1,
                7:  1,
                11: 1,
                13: 1,
                17: 1,
                19: 1,
                23: 1,
                29: 1,
                31: 1,
        }
        res := 0
        for i := L; i <= R; i++ {
                bits := 0
                for n := i; n > 0; n >>= 1 {
                        // bits = bits + n & 1
                        bits = bits + n % 2
                }
                res = res + primes[bits]
        }
        return res
}
```

22.16 766. 托普利茨矩阵 (2)

- 题目

```
如果一个矩阵的每一方向由左上到右下的对角线上具有相同元素，那么这个矩阵是托普利茨矩阵。
给定一个 M x N 的矩阵，当且仅当它是托普利茨矩阵时返回 True。
示例 1:输入:
matrix = [
  [1,2,3,4],
  [5,1,2,3],
```

(续下页)

(接上页)

```
  [9,5,1,2]
]
输出: True
解释:
在上述矩阵中, 其对角线为:"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]"。
各条对角线上的所有元素均相同, 因此答案是True。
示例 2:输入:
matrix = [
  [1,2],
  [2,2]
]
输出: False
解释: 对角线"[1, 2]"上的元素不同。
说明:
    matrix 是一个包含整数的二维数组。
    matrix 的行数和列数均在 [1, 20]范围内。
    matrix[i][j] 包含的整数在 [0, 99]范围内。
进阶:
   ␣
↪如果矩阵存储在磁盘上, 并且磁盘内存是有限的, 因此一次最多只能将一行矩阵加载到内存中, 该怎么办?
    如果矩阵太大以至于只能一次将部分行加载到内存中, 该怎么办?
```

- 解题思路

```
func isToeplitzMatrix(matrix [][]int) bool {
        m, n := len(matrix), len(matrix[0])
        for i := 0; i < m-1; i++ {
                for j := 0; j < n-1; j++ {
                        if matrix[i][j] != matrix[i+1][j+1] {
                                return false
                        }
                }
        }
        return true
}

#
func isToeplitzMatrix(matrix [][]int) bool {
        m := make(map[int]int)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[0]); j++ {
                        if value, ok := m[i-j]; ok {
                                if matrix[i][j] != value {
                                        return false
```

(续下页)

(接上页)

```
                                }
                        } else {
                                m[i-j] = matrix[i][j]
                        }
                }
        }
        return true
}
```

22.17 771. 宝石与石头 (3)

- 题目

```
给定字符串J 代表石头中宝石的类型，和字符串 S代表你拥有的石头。
S 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。
J 中的字母不重复，J 和 S中的所有字符都是字母。字母区分大小写，因此"a"和"A
↪"是不同类型的石头。
示例 1:输入: J = "aA", S = "aAAbbbb"输出: 3
示例 2:输入: J = "z", S = "ZZ"输出: 0

注意:
    S 和 J 最多含有50个字母。
     J 中的字符不重复。
```

- 解题思路

```
func numJewelsInStones(J string, S string) int {
        res := 0
        for _, v := range J {
                res = res + strings.Count(S, string(v))
        }
        return res
}


#
func numJewelsInStones(J string, S string) int {
        m := make(map[byte]bool)
        for i := range J {
                m[J[i]] = true
        }
        res := 0
```

(续下页)

(接上页)

```
        for i := range S {
                if m[S[i]] {
                        res++
                }
        }
        return res
}

#
func numJewelsInStones(J string, S string) int {
        res := 0
        for _, v := range J {
                for _, s := range S {
                        if v == s {
                                res++
                        }
                }
        }
        return res
}
```

22.18 783. 二叉搜索树节点最小距离 (3)

- 题目

```
给定一个二叉搜索树的根节点 root，返回树中任意两节点的差的最小值。
示例：输入：root = [4,2,6,1,3,null,null] 输出：1
解释:注意，root是树节点对象(TreeNode object)，而不是数组。
给定的树 [4,2,6,1,3,null,null] 可表示为下图：
          4
        /   \
      2      6
     / \
    1   3
最小的差值是 1，它是节点1和节点2的差值，也是节点3和节点2的差值。
注意：
    二叉树的大小范围在 2 到 100。
    二叉树总是有效的，每个节点的值都是整数，且不重复。
    本题与 530：https://leetcode.cn/problems/minimum-absolute-difference-in-bst/ 相同
```

- 解题思路

```
var minDiff, previous int

func minDiffInBST(root *TreeNode) int {
        minDiff, previous = math.MaxInt32, math.MaxInt32
        dfs(root)
        return minDiff
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
        newDiff := diff(previous, root.Val)
        if minDiff > newDiff {
                minDiff = newDiff
        }
        previous = root.Val
        dfs(root.Right)
}

func diff(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func minDiffInBST(root *TreeNode) int {
        arr := make([]int, 0)
        dfs(root, &arr)
        min := arr[1] - arr[0]
        for i := 2; i < len(arr); i++ {
                if min > arr[i]-arr[i-1] {
                        min = arr[i] - arr[i-1]
                }
        }
        return min
}

func dfs(root *TreeNode, arr *[]int) {
        if root == nil {
                return
```

(续下页)

(接上页)

```
        }
        dfs(root.Left, arr)
        *arr = append(*arr, root.Val)
        dfs(root.Right, arr)
}

#
func minDiffInBST(root *TreeNode) int {
        arr := make([]int, 0)
        stack := make([]*TreeNode, 0)
        min := math.MaxInt32
        for root != nil || len(stack) > 0 {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                arr = append(arr, node.Val)
                if len(arr) > 1 {
                        temp := node.Val - arr[len(arr)-2]
                        if min > temp {
                                min = temp
                        }
                }
                root = node.Right
        }
        return min
}
```

22.19 784. 字母大小写全排列 (4)

- 题目

```
给定一个字符串S，通过将字符串S中的每个字母转变大小写，我们可以获得一个新的字符串。
返回所有可能得到的字符串集合。

示例：
输入：S = "a1b2" 输出：["a1b2", "a1B2", "A1b2", "A1B2"]
输入：S = "3z4" 输出：["3z4", "3Z4"]
输入：S = "12345" 输出：["12345"]
注意：
```

(续下页)

(接上页)

```
S 的长度不超过12。
S 仅由数字和字母组成。
```

- 解题思路

```
func letterCasePermutation(S string) []string {
        res := make([]string, 1)
        for i := 0; i < len(S); i++ {
                if string(S[i]) >= "0" && string(S[i]) <= "9" {
                        newRes := make([]string, 0)
                        for _, v := range res {
                                newRes = append(newRes, v+string(S[i]))
                        }
                        res = newRes
                } else if b, ok := check(S[i]); ok {
                        first := string(b[0])
                        second := string(b[1])
                        newRes := make([]string, 0)
                        for _, v := range res {
                                newRes = append(newRes, v+first)
                                newRes = append(newRes, v+second)
                        }
                        res = newRes
                }
        }
        return res
}

func check(b byte) ([]byte, bool) {
        if 'a' <= b && b <= 'z' {
                return []byte{b - 'a' + 'A', b}, true
        }
        if 'A' <= b && b <= 'Z' {
                return []byte{b, b - 'A' + 'a'}, true
        }
        return []byte{b}, false
}

#
func letterCasePermutation(S string) []string {
        size := len(S)
        if size == 0 {
                return []string{""}
        }
```

(续下页)

(接上页)

```
        postfixs := make([]string, 1)
        lastByte := S[size-1]
        postfixs[0] = string(lastByte)
        if b, ok := check(lastByte); ok {
                postfixs = append(postfixs, string(b))
        }
        prefixs := letterCasePermutation(S[:size-1])
        res := make([]string, 0)
        for _, pre := range prefixs {
                for _, post := range postfixs {
                        res = append(res, pre+post)
                }
        }
        return res
}

func check(b byte) (byte, bool) {
        if 'a' <= b && b <= 'z' {
                return b - 'a' + 'A', true
        }
        if 'A' <= b && b <= 'Z' {
                return b - 'A' + 'a', true
        }
        return 0, false
}

#
func letterCasePermutation(S string) []string {
        S = strings.ToLower(S)
        res := []string{S}
        for i := range S {
                if S[i] >= 'a' {
                        n := len(res)
                        for j := 0; j < n; j++ {
                                temp := []byte(res[j])
                                temp[i] = S[i] - 'a' + 'A'
                                res = append(res, string(temp))
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
var res []string

func letterCasePermutation(S string) []string {
        res = make([]string, 0)
        dfs([]byte(S), 0)
        return res
}

func dfs(arr []byte, level int) {
        if level == len(arr) {
                res = append(res, string(arr))
                return
        }
        if arr[level] >= 'a' && arr[level] <= 'z' {
                dfs(arr, level+1)
                arr[level] = arr[level] - 'a' + 'A' // 大写
                dfs(arr, level+1)
                arr[level] = arr[level] - 'A' + 'a' // 小写
        } else if arr[level] >= 'A' && arr[level] <= 'Z' {
                dfs(arr, level+1)
                arr[level] = arr[level] - 'A' + 'a' // 小写
                dfs(arr, level+1)
                arr[level] = arr[level] - 'a' + 'A' // 大写
        } else {
                dfs(arr, level+1)
        }
        return
}
```

22.20 788. 旋转数字 (4)

- 题目

```
我们称一个数 X 为好数，如果它的每位数字逐个地被旋转 180 度后，
我们仍可以得到一个有效的，且和 X 不同的数。要求每位数字都要被旋转。
如果一个数的每位数字被旋转以后仍然还是一个数字， 则这个数是有效的。0，1，和 8␣
↪被旋转后仍然是它们自己；
2 和 5 可以互相旋转成对方（在这种情况下，它们以不同的方向旋转，换句话说，2 和 5␣
↪互为镜像）；
6 和 9 同理，除了这些以外其他的数字旋转以后都不再是有效的数字。
```

(续下页)

(接上页)

```
现在我们有一个正整数 N，计算从 1 到 N 中有多少个数 X 是好数？
示例：输入：10 输出：4
解释：在[1, 10]中有四个好数： 2, 5, 6, 9。注意 1 和 10 不是好数，
↪因为他们在旋转之后不变。
提示：N 的取值范围是 [1, 10000]。
```

- 解题思路

```
func rotatedDigits(N int) int {
        count := 0
        for i := 2; i <= N; i++ {
                if isValid(i) {
                        count++
                }
        }
        return count
}

func isValid(n int) bool {
        valid := false
        for n > 0 {
                switch n % 10 {
                case 2, 5, 6, 9:
                        valid = true
                case 3, 4, 7:
                        return false
                }
                n = n / 10
        }
        return valid
}

#
func rotatedDigits(N int) int {
        count := 0
        for i := 2; i <= N; i++ {
                if isValid(i, false) {
                        count++
                }
        }
        return count
}

func isValid(n int, flag bool) bool {
```

(续下页)

(接上页)

```
        if n == 0 {
                return flag
        }
        switch n % 10 {
        case 3, 4, 7:
                return false
        case 0, 1, 8:
                return isValid(n/10, flag)
        case 2, 5, 6, 9:
                return isValid(n/10, true)
        }
        return false
}

#
// 每个数字由(i/10)和(i%10)组成
// dp[i]={dp[i/10],dp[i%10]}
func rotatedDigits(N int) int {
        dp := []int{0, 0, 1, -1, -1, 1, 1, -1, 0, 1}
        if N >= 10 {
                dp = append(dp, make([]int, N-9)...)
        }
        res := 0
        for i := 0; i <= N; i++ {
                if dp[i/10] == -1 || dp[i%10] == -1 {
                        dp[i] = -1
                } else if dp[i] = dp[i/10] | dp[i%10]; dp[i] == 1 {
                        // arr[i/10] = 1/0 arr[i%10] == 1/0
                        // 异或操作，确保把0，1，8组成的数字剔除
                        // 0|0 == 0
                        // 0|1 == 1
                        // 1|0 == 1
                        // 1|1 == 1
                        res++
                }
        }
        return res
}
```

22.21 796. 旋转字符串 (2)

- 题目

```
给定两个字符串, A 和 B。
A 的旋转操作就是将 A 最左边的字符移动到最右边。
例如, 若 A = 'abcde', 在移动一次之后结果就是'bcdea' 。
如果在若干次旋转操作之后, A 能变成B, 那么返回True。

示例 1: 输入: A = 'abcde', B = 'cdeab' 输出: true
示例 2:输入: A = 'abcde', B = 'abced' 输出: false
注意: A 和 B 长度不超过 100。
```

- 解题思路

```
func rotateString(A string, B string) bool {
        return len(A) == len(B) && strings.Contains(A+A, B)
}

#
func rotateString(A string, B string) bool {
        if A == B {
                return true
        }
        if len(A) != len(B) {
                return false
        }
        for i := 0; i < len(A); i++ {
                A = A[1:] + string(A[0])
                if A == B {
                        return true
                }
        }
        return false
}
```

CHAPTER 23

0701-0800-Medium

23.1 701. 二叉搜索树中的插入操作 (2)

- 题目

```
给定二叉搜索树（BST）的根节点和要插入树中的值，将值插入二叉搜索树。␣
↪返回插入后二叉搜索树的根节点。
输入数据 保证 ，新值和原始二叉搜索树中的任意节点值都不同。
注意，可能存在多种有效的插入方式，只要树在插入后仍保持为二叉搜索树即可。 你可以返回␣
↪任意有效的结果 。
示例 1：输入：root = [4,2,7,1,3], val = 5 输出：[4,2,7,1,3,5]
解释：另一个满足题目要求可以通过的树是：
示例 2：输入：root = [40,20,60,10,30,50,70], val = 25
输出：[40,20,60,10,30,50,70,null,null,25]
示例 3：输入：root = [4,2,7,1,3,null,null,null,null,null,null], val = 5
输出：[4,2,7,1,3,5]
提示：给定的树上的节点数介于 0 和 10^4 之间
每个节点都有一个唯一整数值，取值范围从 0 到 10^8
-10^8 <= val <= 10^8
新值和原始二叉搜索树中的任意节点值都不同
```

- 解题思路

```
func insertIntoBST(root *TreeNode, val int) *TreeNode {
        if root == nil {
```

(续下页)

(接上页)

```
                return &TreeNode{
                        Val: val,
                }
        }
        temp := root
        for temp != nil {
                if temp.Val > val {
                        if temp.Left == nil {
                                temp.Left = &TreeNode{
                                        Val: val,
                                }
                                break
                        }
                        temp = temp.Left
                } else {
                        if temp.Right == nil {
                                temp.Right = &TreeNode{
                                        Val: val,
                                }
                                break
                        }
                        temp = temp.Right
                }
        }
        return root
}

# 2
func insertIntoBST(root *TreeNode, val int) *TreeNode {
        if root == nil {
                return &TreeNode{
                        Val: val,
                }
        }
        if root.Val > val {
                root.Left = insertIntoBST(root.Left, val)
        } else {
                root.Right = insertIntoBST(root.Right, val)
        }
        return root
}
```

23.2 707. 设计链表 (2)

- 题目

```
设计链表的实现。您可以选择使用单链表或双链表。
单链表中的节点应该具有两个属性：val和next。val是当前节点的值，next是指向下一个节点的指针/
↪引用。
如果要使用双向链表，则还需要一个属性prev以指示链表中的上一个节点。假设链表中的所有节点都是␣
↪0-index 的。
在链表类中实现这些功能：
get(index)：获取链表中第index个节点的值。如果索引无效，则返回-1。
addAtHead(val)：在链表的第一个元素之前添加一个值为val的节点。插入后，新节点将成为链表的第一个节点。
addAtTail(val)：将值为val 的节点追加到链表的最后一个元素。
addAtIndex(index,val)：在链表中的第index个节点之前添加值为val 的节点。
如果index等于链表的长度，则该节点将附加到链表的末尾。
如果 index 大于链表长度，则不会插入节点。如果index小于0，则在头部插入节点。
deleteAtIndex(index)：如果索引index 有效，则删除链表中的第index 个节点。
示例：MyLinkedList linkedList = new MyLinkedList();
linkedList.addAtHead(1);
linkedList.addAtTail(3);
linkedList.addAtIndex(1,2);   //链表变为1-> 2-> 3
linkedList.get(1);            //返回2
linkedList.deleteAtIndex(1);  //现在链表是1-> 3
linkedList.get(1);            //返回3
提示：所有val值都在[1, 1000]之内。
操作次数将在[1, 1000]之内。
请不要使用内置的 LinkedList 库。
```

- 解题思路

```
type MyLinkedList struct {
        size int
        head *Node
        tail *Node
}

type Node struct {
        value int
        next  *Node
}

func Constructor() MyLinkedList {
        tail := &Node{}
        head := &Node{next: tail}
```

(续下页)

(接上页)

```
        return MyLinkedList{
                head: head,
                tail: tail,
        }
}

func (this *MyLinkedList) Get(index int) int {
        if index < 0 || this.size <= index {
                return -1
        }
        i := 0
        cur := this.head.next
        for i < index {
                i++
                cur = cur.next
        }
        return cur.value
}

func (this *MyLinkedList) AddAtHead(val int) {
        node := &Node{value: val}
        node.next = this.head.next
        this.head.next = node
        this.size++
}

func (this *MyLinkedList) AddAtTail(val int) {
        this.tail.value = val
        node := &Node{}
        this.tail.next = node
        this.tail = node
        this.size++
}

func (this *MyLinkedList) AddAtIndex(index int, val int) {
        switch {
        case index < 0 || this.size < index:
                return
        case index == 0:
                this.AddAtHead(val)
                return
        case index == this.size:
                this.AddAtTail(val)
```

(续下页)

(接上页)

```
                return
        }
        i := -1
        cur := this.head
        for i+1 < index {
                i++
                cur = cur.next
        }
        node := &Node{value: val}
        node.next = cur.next
        cur.next = node
        this.size++
}

func (this *MyLinkedList) DeleteAtIndex(index int) {
        if index < 0 || this.size <= index {
                return
        }
        i := -1
        cur := this.head
        for i+1 < index {
                i++
                cur = cur.next
        }
        cur.next = cur.next.next
        this.size--
}

# 2
type MyLinkedList struct {
        head *Node
        size int
}

type Node struct {
        value int
        next  *Node
}

func Constructor() MyLinkedList {
        return MyLinkedList{
                head: &Node{},
                size: 0,
```

(续下页)

(接上页)

```
        }
}

func (this *MyLinkedList) Get(index int) int {
        if index < 0 || index >= this.size {
                return -1
        }
        prev := this.head
        for i := 1; i <= index; i++ {
                prev = prev.next
        }
        return prev.next.value
}

func (this *MyLinkedList) AddAtHead(val int) {
        this.AddAtIndex(0, val)
}

func (this *MyLinkedList) AddAtTail(val int) {
        this.AddAtIndex(this.size, val)
}

func (this *MyLinkedList) AddAtIndex(index int, val int) {
        if index < 0 || index > this.size {
                return
        }
        prev := this.head
        for i := 1; i <= index; i++ {
                prev = prev.next
        }
        node := &Node{
                value: val,
                next:  nil,
        }
        node.next = prev.next
        prev.next = node
        this.size++
}

func (this *MyLinkedList) DeleteAtIndex(index int) {
        if index < 0 || index >= this.size {
                return
        }
```

(续下页)

(接上页)

```
        prev := this.head
        for i := 1; i <= index; i++ {
                prev = prev.next
        }
        prev.next = prev.next.next
        this.size--
}
```

23.3 712. 两个字符串的最小 ASCII 删除和 (3)

- 题目

```
给定两个字符串s1, s2, 找到使两个字符串相等所需删除字符的ASCII值的最小和。
示例 1:输入: s1 = "sea", s2 = "eat" 输出: 231
解释: 在 "sea" 中删除 "s" 并将 "s" 的值(115)加入总和。
在 "eat" 中删除 "t" 并将 116 加入总和。
结束时, 两个字符串相等, 115 + 116 = 231 就是符合条件的最小和。
示例2:输入: s1 = "delete", s2 = "leet" 输出: 403
解释: 在 "delete" 中删除 "dee" 字符串变成 "let",
将 100[d]+101[e]+101[e] 加入总和。在 "leet" 中删除 "e" 将 101[e] 加入总和。
结束时, 两个字符串都等于 "let", 结果即为 100+101+101+101 = 403 。
如果改为将两个字符串转换为 "lee" 或 "eet", 我们会得到 433 或 417 的结果, 比答案更大。
注意:0 < s1.length, s2.length <= 1000。
所有字符串中的字符ASCII值在[97, 122]之间。
```

- 解题思路

```
func minimumDeleteSum(s1 string, s2 string) int {
        a, b := len(s1), len(s2)
        // 最长公共子序列
        dp := make([][]int, a+1)
        for i := 0; i <= a; i++ {
                dp[i] = make([]int, b+1)
        }
        for i := 1; i <= a; i++ {
                for j := 1; j <= b; j++ {
                        if s1[i-1] == s2[j-1] {
                                dp[i][j] = dp[i-1][j-1] + int(s1[i-1])
                        } else {
                                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
                        }
                }
```

(续下页)

(接上页)

```
        }
        return sumAscii(s1) + sumAscii(s2) - 2*dp[a][b]
}

func sumAscii(s string) int {
        res := 0
        for i := 0; i < len(s); i++ {
                res = res + int(s[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minimumDeleteSum(s1 string, s2 string) int {
        a, b := len(s1), len(s2)
        dp := make([][]int, a+1)
        for i := 0; i <= a; i++ {
                dp[i] = make([]int, b+1)
                if i > 0 {
                        dp[i][0] = dp[i-1][0] + int(s1[i-1])
                }
        }
        for i := 1; i <= b; i++ {
                dp[0][i] = dp[0][i-1] + int(s2[i-1])
        }

        for i := 1; i <= a; i++ {
                for j := 1; j <= b; j++ {
                        if s1[i-1] == s2[j-1] {
                                dp[i][j] = dp[i-1][j-1]
                        } else {
                                dp[i][j] = min(dp[i][j-1]+int(s2[j-1]), dp[i-
↪1][j]+int(s1[i-1]))
                        }
                }
        }
```

(续下页)

(接上页)

```
        return dp[a][b]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var m [][]int

func minimumDeleteSum(s1 string, s2 string) int {
        a, b := len(s1), len(s2)
        m = make([][]int, a+1)
        for i := 0; i <= a; i++ {
                m[i] = make([]int, b+1)
                for j := 0; j <= b; j++ {
                        m[i][j] = -1
                }
        }

        total := dfs(s1, s2, 0, 0)
        return sumAscii(s1) + sumAscii(s2) - 2*total
}

func dfs(s1 string, s2 string, i, j int) int {
        if len(s1) == i || len(s2) == j {
                return 0
        }
        if m[i][j] > -1 {
                return m[i][j]
        }
        if s1[i] == s2[j] {
                m[i][j] = dfs(s1, s2, i+1, j+1) + int(s1[i])
        } else {
                m[i][j] = max(dfs(s1, s2, i, j+1), dfs(s1, s2, i+1, j))
        }
        return m[i][j]
}

func sumAscii(s string) int {
```

(续下页)

(接上页)

```
        res := 0
        for i := 0; i < len(s); i++ {
                res = res + int(s[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

23.4 713. 乘积小于 K 的子数组 (1)

- 题目

```
给定一个正整数数组 nums。
找出该数组内乘积小于 k 的连续的子数组的个数。
示例 1:输入: nums = [10,5,2,6], k = 100 输出: 8
解释: 8个乘积小于100的子数组分别为: [10], [5], [2], [6], [10,5], [5,2], [2,6], [5,2,
↪6]。
需要注意的是 [10,5,2] 并不是乘积小于100的子数组。
说明:0 < nums.length <= 50000
    0 < nums[i] < 1000
    0 <= k < 10^6
```

- 解题思路

```
func numSubarrayProductLessThanK(nums []int, k int) int {
        if k <= 1 {
                return 0
        }
        res := 0
        left := 0
        total := 1
        for right := 0; right < len(nums); right++ {
                total = total * nums[right]
                for k <= total {
                        total = total / nums[left]
                        left++
```

(续下页)

(接上页)

```
                }
                res = res + right - left + 1
        }
        return res
}
```

23.5 714. 买卖股票的最佳时机含手续费 (2)

- 题目

```
给定一个整数数组 prices，其中第 i 个元素代表了第 i 天的股票价格 ；
非负整数 fee 代表了交易股票的手续费用。
你可以无限次地完成交易，但是你每笔交易都需要付手续费。
如果你已经购买了一个股票，在卖出它之前你就不能再继续购买股票了。
返回获得利润的最大值。
注意：这里的一笔交易指买入持有并卖出股票的整个过程，每笔交易你只需要为支付一次手续费。
示例 1:输入：prices = [1, 3, 2, 8, 4, 9], fee = 2 输出：8
解释：能够达到的最大利润:
在此处买入 prices[0] = 1
在此处卖出 prices[3] = 8
在此处买入 prices[4] = 4
在此处卖出 prices[5] = 9
总利润：((8 - 1) - 2) + ((9 - 4) - 2) = 8.
注意：
    0 < prices.length <= 50000.
    0 < prices[i] < 50000.
    0 <= fee < 50000.
```

- 解题思路

```
func maxProfit(prices []int, fee int) int {
        dp0, dp1 := 0, math.MinInt32
        for i := 0; i < len(prices); i++ {
                temp := dp0
                dp0 = max(dp0, dp1+prices[i])
                dp1 = max(dp1, temp-prices[i]-fee)
        }
        return dp0
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}

# 2
func maxProfit(prices []int, fee int) int {
        dp := make([][2]int, len(prices))
        dp[0][0] = 0
        dp[0][1] = -prices[0]
        for i := 1; i < len(prices); i++ {
                dp[i][0] = max(dp[i-1][0], dp[i-1][1]+prices[i]-fee)
                dp[i][1] = max(dp[i-1][1], dp[i-1][0]-prices[i])
        }
        return dp[len(prices)-1][0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

23.6 718. 最长重复子数组 (3)

- 题目

```
给两个整数数组 A 和 B ，返回两个数组中公共的、长度最长的子数组的长度。
示例：输入：A: [1,2,3,2,1] B: [3,2,1,4,7]
输出：3
解释：长度最长的公共子数组是 [3, 2, 1] 。
提示：1 <= len(A), len(B) <= 1000
    0 <= A[i], B[i] < 100
```

- 解题思路

```
func findLength(A []int, B []int) int {
        n, m := len(A), len(B)
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m+1)
```

(续下页)

(接上页)

```
        }
        res := math.MinInt32
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if A[i-1] == B[j-1] {
                                dp[i][j] = dp[i-1][j-1] + 1
                        }
                        if dp[i][j] > res {
                                res = dp[i][j]
                        }
                }
        }
        return res
}

# 2
func findLength(A []int, B []int) int {
        n, m := len(A), len(B)
        dp := make([]int, m+1)
        res := math.MinInt32
        for i := 1; i <= n; i++ {
                for j := m; j >= 1; j-- {
                        if A[i-1] == B[j-1] {
                                dp[j] = dp[j-1] + 1
                        } else {
                                dp[j] = 0 // 需要清0
                        }
                        if dp[j] > res {
                                res = dp[j]
                        }
                }
        }
        return res
}

# 3
func findLength(A []int, B []int) int {
        n, m := len(A), len(B)
        res := math.MinInt32
        for i := 0; i < n; i++ {
                length := min(n-i, m)
                maxLength := getMaxLength(A, B, i, 0, length)
                res = max(res, maxLength)
```

(续下页)

(接上页)

```
        }
        for i := 0; i < m; i++ {
                length := min(n, m-i)
                maxLength := getMaxLength(A, B, 0, i, length)
                res = max(res, maxLength)
        }
        return res
}

func getMaxLength(A, B []int, a, b int, length int) int {
        res := 0
        count := 0
        for i := 0; i < length; i++ {
                if A[a+i] == B[b+i] {
                        count++
                } else {
                        count = 0
                }
                res = max(res, count)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

23.7 721. 账户合并 (1)

- 题目

```
给定一个列表 accounts，每个元素 accounts[i]是一个字符串列表，
其中第一个元素 accounts[i][0]是名称 (name)，其余元素是 emails 表示该账户的邮箱地址。
现在，我们想合并这些账户。如果两个账户都有一些共同的邮箱地址，则两个账户必定属于同一个人。
请注意，即使两个账户具有相同的名称，它们也可能属于不同的人，因为人们可能具有相同的名称。
一个人最初可以拥有任意数量的账户，但其所有账户都具有相同的名称。
合并账户后，按以下格式返回账户：每个账户的第一个元素是名称，其余元素是按顺序排列的邮箱地址。
账户本身可以以任意顺序返回。
示例 1：输入：accounts = [["John", "johnsmith@mail.com", "john00@mail.com"],
["John", "johnnybravo@mail.com"], ["John", "johnsmith@mail.com",
"john_newyork@mail.com"], ["Mary", "mary@mail.com"]]
输出：
[["John", 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com'],
["John", "johnnybravo@mail.com"], ["Mary", "mary@mail.com"]]
解释：第一个和第三个 John 是同一个人，因为他们有共同的邮箱地址 "johnsmith@mail.com"。
第二个 John 和 Mary 是不同的人，因为他们的邮箱地址没有被其他帐户使用。
可以以任何顺序返回这些列表，例如答案 [['Mary', 'mary@mail.com'],
['John', 'johnnybravo@mail.com'],
['John', 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com']]␣
↪也是正确的。
提示：accounts的长度将在[1, 1000]的范围内。
accounts[i]的长度将在[1, 10]的范围内。
accounts[i][j]的长度将在[1, 30]的范围内。
```

- 解题思路

```go
func accountsMerge(accounts [][]string) [][]string {
        n := len(accounts)
        fa = Init(n)
        res := make([][]string, 0)
        m := make(map[string]int)
        for i := 0; i < len(accounts); i++ {
                for j := 1; j < len(accounts[i]); j++ {
                        email := accounts[i][j]
                        if id, ok := m[email]; ok { // 邮箱重复出现，合并账户
                                union(i, id)
                        } else {
                                m[email] = i
                        }
                }
        }
```

(续下页)

(接上页)

```
        temp := make([][]string, n)
        for k, v := range m {
                target := find(v)
                temp[target] = append(temp[target], k)
        }
        for i := 0; i < len(temp); i++ {
                if len(temp[i]) > 0 {
                        arr := temp[i]
                        sort.Strings(arr)
                        arr = append([]string{accounts[i][0]}, arr...)
                        res = append(res, arr)
                }
        }
        return res
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}
```

23.8 722. 删除注释 (1)

- 题目

```
给一个 C++␣
↪程序，删除程序中的注释。这个程序source是一个数组，其中source[i]表示第i行源码。
这表示每行源码由\n分隔。
在 C++ 中有两种注释风格，行内注释和块注释。
字符串// 表示行注释，表示//和其右侧的其余字符应该被忽略。
字符串/* 表示一个块注释，它表示直到*/的下一个（非重叠）出现的所有字符都应该被忽略。
（阅读顺序为从左到右）非重叠是指，字符串/*/
↪并没有结束块注释，因为注释的结尾与开头相重叠。
第一个有效注释优先于其他注释：如果字符串//出现在块注释中会被忽略。
同样，如果字符串/*出现在行或块注释中也会被忽略。
如果一行在删除注释之后变为空字符串，那么不要输出该行。即，答案列表中的每个字符串都是非空的。
样例中没有控制字符，单引号或双引号字符。比如，source = "string s = "/* Not a comment.␣
↪*/";"
不会出现在测试样例里。（此外，没有其他内容（如定义或宏）会干扰注释。）
我们保证每一个块注释最终都会被闭合， 所以在行或块注释之外的/*总是开始新的注释。
最后，隐式换行符可以通过块注释删除。 有关详细信息，请参阅下面的示例。
从源代码中删除注释后，需要以相同的格式返回源代码。
示例1:输入：
source = ["/*Test program */", "int main()", "{ ", "  // variable declaration ",
"int a, b, c;", "/* This is a test", "   multiline  ", "   comment for ",
"   testing */", "a = b + c;", "}"]
示例代码可以编排成这样：
/*Test program */
int main()
{
  // variable declaration
int a, b, c;
/* This is a test
   multiline
   comment for
   testing */
a = b + c;
}
输出：["int main()","{ ","  ","int a, b, c;","a = b + c;","}"]
编排后：
int main()
{

int a, b, c;
a = b + c;
```

(续下页)

(接上页)

```
}
解释：第 1 行和第 6-9 行的字符串 /* 表示块注释。第 4 行的字符串 // 表示行注释。
示例 2:输入： source = ["a/*comment", "line", "more_comment*/b"]
输出：["ab"]
解释：原始的 source 字符串是 "a/*comment\nline\nmore_comment*/b",␣
↪其中我们用粗体显示了换行符。
删除注释后，隐含的换行符被删除，留下字符串 "ab" 用换行符分隔成数组时就是 ["ab"].
注意:source的长度范围为[1, 100].
source[i]的长度范围为[0, 80].
每个块注释都会被闭合。
给定的源码中不会有单引号、双引号或其他控制字符。
```

- 解题思路

```
func removeComments(source []string) []string {
        res := make([]string, 0)
        flag := false // 判断是否是块注释
        temp := make([]byte, 0)
        for i := 0; i < len(source); i++ {
                str := source[i]
                j := 0
                for j < len(str) {
                        if flag == false && j+1 < len(str) && str[j] == '/' &&␣
↪str[j+1] == '*' {
                                flag = true
                                j = j + 2
                                continue
                        }
                        if flag == true && j+1 < len(str) && str[j] == '*' &&␣
↪str[j+1] == '/' {
                                flag = false
                                j = j + 2
                                continue
                        }
                        if flag == false && j+1 < len(str) && str[j] == '/' &&␣
↪str[j+1] == '/' {
                                break
                        }
                        if flag == false {
                                temp = append(temp, str[j])
                        }
                        j++
                }
                if flag == false && len(temp) > 0 {
```

(续下页)

(接上页)

```
                        res = append(res, string(temp))
                        temp = make([]byte, 0)
                }
        }
        return res
}
```

23.9 725. 分隔链表 (2)

- 题目

```
给定一个头结点为 root 的链表, 编写一个函数以将链表分隔为 k 个连续的部分。
每部分的长度应该尽可能的相等: 任意两部分的长度差距不能超过 1, 也就是说可能有些部分为␣
↪null。
这k个部分应该按照在链表中出现的顺序进行输出, 并且排在前面的部分的长度应该大于或等于后面的长度。
返回一个符合上述规则的链表的列表。
举例: 1->2->3->4, k = 5 // 5 结果 [ [1], [2], [3], [4], null ]
示例 1: 输入: root = [1, 2, 3], k = 5 输出: [[1],[2],[3],[],[]]
解释:输入输出各部分都应该是链表, 而不是数组。
例如, 输入的结点 root 的 val= 1, root.next.val = 2, \root.next.next.val = 3, 且 root.
↪next.next.next = null。
第一个输出 output[0] 是 output[0].val = 1, output[0].next = null。
最后一个元素 output[4] 为 null, 它代表了最后一个部分为空链表。
示例 2: 输入: root = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 3
输出: [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]
解释:输入被分成了几个连续的部分, 并且每部分的长度相差不超过1.
↪前面部分的长度大于等于后面部分的长度。
提示:root 的长度范围: [0, 1000].
输入的每个节点的大小范围: [0, 999].
k的取值范围: [1, 50].
```

- 解题思路

```
func splitListToParts(root *ListNode, k int) []*ListNode {
        res := make([]*ListNode, 0)
        cur := root
        length := 0
        for cur != nil {
                length++
                cur = cur.Next
        }
        a, b := length/k, length%k
```

(续下页)

(接上页)

```
        for i := 0; i < k; i++ {
                node := &ListNode{Next: nil}
                temp := node
                for j := 0; j < a; j++ {
                        temp.Next = &ListNode{
                                Val:  root.Val,
                                Next: nil,
                        }
                        temp = temp.Next
                        root = root.Next
                }
                if b > 0 {
                        temp.Next = &ListNode{
                                Val:  root.Val,
                                Next: nil,
                        }
                        temp = temp.Next
                        root = root.Next
                        b = b - 1
                }
                res = append(res, node.Next)
        }
        return res
}

# 2
func splitListToParts(root *ListNode, k int) []*ListNode {
        res := make([]*ListNode, 0)
        cur := root
        length := 0
        for cur != nil {
                length++
                cur = cur.Next
        }
        a, b := length/k, length%k
        for i := 0; i < k; i++ {
                if root == nil {
                        res = append(res, nil)
                        continue
                }
                node := root
                for j := 1; j < a && root.Next != nil; j++ {
                        root = root.Next
```

(续下页)

(接上页)

```
                }
                if b > 0 {
                        root = root.Next
                        b--
                }
                temp := root.Next
                root.Next = nil
                root = temp
                res = append(res, node)
        }
        return res
}
```

23.10 729. 我的日程安排表 I(2)

- 题目

```
实现一个 MyCalendar␣
↪类来存放你的日程安排。如果要添加的时间内没有其他安排，则可以存储这个新的日程安排。
MyCalendar 有一个 book(int start, int end)方法。它意味着在 start 到 end␣
↪时间内增加一个日程安排，
注意，这里的时间是半开区间，即 [start, end), 实数x 的范围为， start <= x < end。
当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生重复预订。
每次调用 MyCalendar.
↪book方法时，如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true。
否则，返回 false并且不要将该日程安排添加到日历中。
请按照以下步骤调用 MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.
↪book(start, end)
示例 1:MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true
解释: 第一个日程安排可以添加到日历中. 第二个日程安排不能添加到日历中，因为时间 15␣
↪已经被第一个日程安排预定了。
第三个日程安排可以添加到日历中，因为第一个日程安排并不包含时间 20 。
说明:每个测试用例，调用MyCalendar.book函数最多不超过1000次。
调用函数MyCalendar.book(start, end)时，start 和end 的取值范围为[0, 10^9]。
```

- 解题思路

```
type MyCalendar struct {
        arr [][2]int
```

(续下页)

(接上页)

```
}

func Constructor() MyCalendar {
        return MyCalendar{arr: make([][2]int, 0)}
}

func (this *MyCalendar) Book(start int, end int) bool {
        for i := 0; i < len(this.arr); i++ {
                if this.arr[i][0] < end && start < this.arr[i][1] {
                        return false
                }
        }
        this.arr = append(this.arr, [2]int{start, end})
        return true
}

# 2
type MyCalendar struct {
        root *Node
}

func Constructor() MyCalendar {
        return MyCalendar{root: nil}
}

func (this *MyCalendar) Book(start int, end int) bool {
        node := &Node{
                start: start,
                end:   end,
                left:  nil,
                right: nil,
        }
        if this.root == nil {
                this.root = node
                return true
        }
        return this.root.Insert(node)
}

type Node struct {
        start int
        end   int
        left  *Node
```

(续下页)

(接上页)

```
        right *Node
}

func (this *Node) Insert(node *Node) bool {
        if node.start >= this.end {
                if this.right == nil {
                        this.right = node
                        return true
                }
                return this.right.Insert(node)
        } else if node.end <= this.start {
                if this.left == nil {
                        this.left = node
                        return true
                }
                return this.left.Insert(node)
        }
        return false
}
```

23.11 731. 我的日程安排表 II(1)

- 题目

```
实现一个 MyCalendar␣
↪类来存放你的日程安排。如果要添加的时间内不会导致三重预订时，则可以存储这个新的日程安排。
MyCalendar 有一个 book(int start, int end)方法。
它意味着在 start 到 end 时间内增加一个日程安排，注意，这里的时间是半开区间，
即 [start, end), 实数x 的范围为， start <= x < end。
当三个日程安排有一些时间上的交叉时（例如三个日程安排都在同一时间内），就会产生三重预订。
每次调用 MyCalendar.
↪book方法时，如果可以将日程安排成功添加到日历中而不会导致三重预订，返回 true。
否则，返回 false 并且不要将该日程安排添加到日历中。
请按照以下步骤调用MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.
↪book(start, end)
示例: MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(50, 60); // returns true
MyCalendar.book(10, 40); // returns true
MyCalendar.book(5, 15); // returns false
MyCalendar.book(5, 10); // returns true
MyCalendar.book(25, 55); // returns true
```

(续下页)

(接上页)

```
解释： 前两个日程安排可以添加至日历中。␣
↪第三个日程安排会导致双重预订，但可以添加至日历中。
第四个日程安排活动（5,15）不能添加至日历中，因为它会导致三重预订。
第五个日程安排（5,10）可以添加至日历中，因为它未使用已经双重预订的时间10。
第六个日程安排（25,55）可以添加至日历中，因为时间 [25,40] 将和第三个日程安排双重预订；
时间 [40,50] 将单独预订，时间 [50,55）将和第二个日程安排双重预订。
提示：每个测试用例，调用MyCalendar.book函数最多不超过1000次。
调用函数MyCalendar.book(start, end)时，start 和end 的取值范围为[0, 10^9]。
```

- 解题思路

```
type MyCalendarTwo struct {
        first  [][2]int
        second [][2]int
}

func Constructor() MyCalendarTwo {
        return MyCalendarTwo{}
}

func (this *MyCalendarTwo) Book(start int, end int) bool {
        for i := 0; i < len(this.second); i++ {
                a, b := this.second[i][0], this.second[i][1]
                if start < b && end > a { // 跟第二个重复
                        return false
                }
        }
        for i := 0; i < len(this.first); i++ {
                a, b := this.first[i][0], this.first[i][1]
                if start < b && end > a {
                        // 插入重叠的部分
                        this.second = append(this.second, [2]int{max(start, a),␣
↪min(end, b)})
                }
        }
        this.first = append(this.first, [2]int{start, end})
        return true
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

23.12 735. 行星碰撞 (2)

- 题目

```
给定一个整数数组 asteroids，表示在同一行的行星。
对于数组中的每一个元素，其绝对值表示行星的大小，正负表示行星的移动方向（正表示向右移动，负表示向左移
每一颗行星以相同的速度移动。
找出碰撞后剩下的所有行星。碰撞规则：两个行星相互碰撞，较小的行星会爆炸。
如果两颗行星大小相同，则两颗行星都会爆炸。两颗移动方向相同的行星，永远不会发生碰撞。
示例 1:输入: asteroids = [5, 10, -5] 输出: [5, 10]
解释: 10 和 -5 碰撞后只剩下 10。 5 和 10 永远不会发生碰撞。
示例 2:输入: asteroids = [8, -8] 输出: []
解释: 8 和 -8 碰撞后，两者都发生爆炸。
示例 3:输入: asteroids = [10, 2, -5] 输出: [10]
解释: 2 和 -5 发生碰撞后剩下 -5。10 和 -5 发生碰撞后剩下 10。
示例 4:输入: asteroids = [-2, -1, 1, 2] 输出: [-2, -1, 1, 2]
解释: -2 和 -1 向左移动，而 1 和 2 向右移动。
由于移动方向相同的行星不会发生碰撞，所以最终没有行星发生碰撞。
说明:数组asteroids 的长度不超过10000。
每一颗行星的大小都是非零整数，范围是[-1000, 1000]。
```

- 解题思路

```
func asteroidCollision(asteroids []int) []int {
        left := make([]int, 0)
        right := make([]int, 0)
        for i := 0; i < len(asteroids); i++ {
                if asteroids[i] > 0 {
                        right = append(right, asteroids[i])
                } else {
                        if len(right) > 0 {
                                for {
                                        if len(right) == 0 {
```

(续下页)

(接上页)

```
                                                left = append(left, asteroids[i])
                                                break
                                        }
                                        sum := asteroids[i] + right[len(right)-1]
                                        if sum == 0 {
                                                right = right[:len(right)-1]
                                                break
                                        } else if sum > 0 {
                                                break
                                        } else {
                                                right = right[:len(right)-1]
                                        }
                                }
                        } else {
                                left = append(left, asteroids[i])
                        }
                }

        }
        return append(left, right...)
}

# 2
func asteroidCollision(asteroids []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(asteroids); i++ {
                value := asteroids[i]
                for value < 0 && len(res) > 0 && res[len(res)-1] > 0 {
                        sum := value + res[len(res)-1]
                        if sum >= 0 {
                                value = 0
                        }
                        if sum <= 0 {
                                res = res[:len(res)-1]
                        }
                }
                if value != 0 {
                        res = append(res, value)
                }
        }
        return res
}
```

23.13 738. 单调递增的数字 (2)

- 题目

```
给定一个非负整数N，找出小于或等于N的最大的整数，同时这个整数需要满足其各个位数上的数字是单调递增。
（当且仅当每个相邻位数上的数字x和y满足x <= y时，我们称这个整数是单调递增的。）
示例 1:输入: N = 10 输出: 9
示例 2:输入: N = 1234 输出: 1234
示例 3:输入: N = 332 输出: 299
说明: N是在[0, 10^9]范围内的一个整数。
```

- 解题思路

```
func monotoneIncreasingDigits(N int) int {
        arr := []byte(strconv.Itoa(N))
        i := 1
        for i < len(arr) && arr[i-1] <= arr[i] {
                i++
        }
        // 前面有逆序的
        if i < len(arr) {
                // 前面减去1，如：332=>2xx要减2次
                for i > 0 && arr[i] < arr[i-1] {
                        arr[i-1]--
                        i--
                }
                i++
                for ; i < len(arr); i++ {
                        arr[i] = '9'
                }
        }
        res, _ := strconv.Atoi(string(arr))
        return res
}

# 2
func monotoneIncreasingDigits(N int) int {
        arr := []byte(strconv.Itoa(N))
        maxValue := -1
        index := -1
        for i := 0; i < len(arr)-1; i++ {
                if int(arr[i]-'0') > maxValue {
                        maxValue = int(arr[i] - '0')
                        index = i
```

(续下页)

(接上页)

```
                }
                if arr[i] > arr[i+1] {
                        arr[index]--
                        for j := index + 1; j < len(arr); j++ {
                                arr[j] = '9'
                        }
                        break
                }
        }
        res, _ := strconv.Atoi(string(arr))
        return res
}
```

23.14 739. 每日温度 (3)

- 题目

```
请根据每日 气温 列表，重新生成一个列表。
对应位置的输出为：要想观测到更高的气温，至少需要等待的天数。
如果气温在这之后都不会升高，请在该位置用 0 来代替。
例如，给定一个列表 temperatures = [73, 74, 75, 71, 69, 72, 76, 73],
你的输出应该是 [1, 1, 4, 2, 1, 1, 0, 0]。
提示：气温 列表长度的范围是 [1, 30000]。每个气温的值的均为华氏度，都是在 [30, 100]␣
↪范围内的整数。
```

- 解题思路

```
func dailyTemperatures(temperatures []int) []int {
        res := make([]int, len(temperatures))
        stack := make([]int, 0) // 栈保存递减数据的下标
        for i := 0; i < len(temperatures); i++ {
                for len(stack) > 0 && temperatures[i] > temperatures[stack[len(stack)-
↪1]] {
                        last := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        res[last] = i - last
                }
                stack = append(stack, i)
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func dailyTemperatures(temperatures []int) []int {
        res := make([]int, len(temperatures))
        arr := make([]int, 101)
        for i := 0; i < len(arr); i++ {
                arr[i] = math.MaxInt64
        }
        for i := len(temperatures) - 1; i >= 0; i-- {
                temp := math.MaxInt64
                for t := temperatures[i] + 1; t < 101; t++ {
                        if arr[t] < temp {
                                temp = arr[t]
                        }
                }
                if temp < math.MaxInt64 {
                        res[i] = temp - i
                }
                arr[temperatures[i]] = i
        }
        return res
}

# 3
func dailyTemperatures(temperatures []int) []int {
        j := 0
        for i := 0; i < len(temperatures); i++ {
                for j = i + 1; j < len(temperatures); j++ {
                        if temperatures[j] > temperatures[i] {
                                temperatures[i] = j - i
                                break
                        }
                }
                if j == len(temperatures) {
                        temperatures[i] = 0
                }
        }
        return temperatures
}
```

23.15 740. 删除与获得点数 (2)

- 题目

```
给定一个整数数组nums，你可以对它进行一些操作。
每次操作中，选择任意一个nums[i]，删除它并获得nums[i]的点数。
之后，你必须删除每个等于nums[i] - 1或nums[i] + 1的元素。
开始你拥有 0 个点数。返回你能通过这些操作获得的最大点数。
示例 1:输入: nums = [3, 4, 2] 输出: 6
解释: 删除 4 来获得 4 个点数，因此 3 也被删除。
之后，删除 2 来获得 2 个点数。总共获得 6 个点数。
示例2:输入: nums = [2, 2, 3, 3, 3, 4] 输出: 9
解释: 删除 3 来获得 3 个点数，接着要删除两个 2 和 4 。
之后，再次删除 3 获得 3 个点数，再次删除 3 获得 3 个点数。
总共获得 9 个点数。
注意:nums的长度最大为20000。
每个整数nums[i]的大小都在[1, 10000]范围内。
```

- 解题思路

```
func deleteAndEarn(nums []int) int {
        count := make([]int, 10001)
        for i := 0; i < len(nums); i++ {
                count[nums[i]]++
        }
        dp := make([]int, 10001)
        dp[1] = count[1]
        dp[2] = max(dp[1], count[2]*2)
        for i := 2; i < 10001; i++ {
                dp[i] = max(dp[i-1], dp[i-2]+i*count[i])
        }
        return dp[10000]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}


# 2
func deleteAndEarn(nums []int) int {
```

(续下页)

(接上页)

```
        count := make([]int, 10001)
        for i := 0; i < len(nums); i++ {
                count[nums[i]]++
        }
        a, b := 0, 0 // a使用i, b不使用i
        prev := -1
        for i := 0; i < 10001; i++ {
                if count[i] > 0 {
                        maxValue := max(a, b)
                        if prev != i-1 { // 不等于上一个, 使用最大值
                                a = i*count[i] + maxValue
                                b = maxValue
                        } else { // 等于上一个, 使用b
                                a = i*count[i] + b
                                b = maxValue
                        }
                        prev = i
                }
        }
        return max(a, b)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

23.16 743. 网络延迟时间 (5)

- 题目

```
有 n 个网络节点, 标记为1到 n。
给你一个列表times, 表示信号经过 有向 边的传递时间。
times[i] = (ui, vi, wi), 其中ui是源节点, vi是目标节点, ␣
↪wi是一个信号从源节点传递到目标节点的时间。
现在, 从某个节点K发出一个信号。需要多久才能使所有节点都收到信号？如果不能使所有节点收到信号, 返回-
↪1 。
示例 1: 输入: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2 输出: 2
示例 2: 输入: times = [[1,2,1]], n = 2, k = 1 输出: 1
示例 3: 输入: times = [[1,2,1]], n = 2, k = 2 输出: -1
```

(续下页)

(接上页)

```
提示：1 <= k <= n <= 100
1 <= times.length <= 6000
times[i].length == 3
1 <= ui, vi <= n
ui != vi
0 <= wi <= 100
所有 (ui, vi) 对都 互不相同（即，不含重复边）
```

- 解题思路

```
func networkDelayTime(times [][]int, n int, k int) int {
        maxValue := math.MaxInt32 / 10
        arr := make([][]int, n) // i=>j的最短距离
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = maxValue
                }
        }
        for i := 0; i < len(times); i++ {
                a, b, c := times[i][0]-1, times[i][1]-1, times[i][2] // a=>b
                arr[a][b] = c
        }
        dis := make([]int, n) // k到其他点的距离
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[k-1] = 0
        visited := make([]bool, n)
        for i := 0; i < n; i++ {
                target := -1 // 寻找未访问的距离起点最近点
                for j := 0; j < len(visited); j++ {
                        if visited[j] == false && (target == -1 || dis[j] <␣
↪dis[target]) {
                                target = j
                        }
                }
                visited[target] = true
                for j := 0; j < len(arr[target]); j++ { // 更新距离
                        dis[j] = min(dis[j], dis[target]+arr[target][j])
                }
        }
        res := 0
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                if dis[i] == maxValue {
                        return -1
                }
                res = max(res, dis[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func networkDelayTime(times [][]int, n int, k int) int {
        maxValue := math.MaxInt32 / 10
        arr := make([][][2]int, n) // 邻接表：i=>j的集合
        for i := 0; i < len(times); i++ {
                a, b, c := times[i][0]-1, times[i][1]-1, times[i][2] // a=>b
                arr[a] = append(arr[a], [2]int{b, c})
        }
        dis := make([]int, n) // k到其他点的距离
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[k-1] = 0
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [2]int{k - 1, 0})
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([2]int) // 距离起点最近的点
                a := node[0]
                if dis[a] < node[1] { // 大于最短距离，跳过
                        continue
```

(续下页)

(接上页)

```
                }
                for i := 0; i < len(arr[a]); i++ {
                        b, c := arr[a][i][0], arr[a][i][1]
                        if dis[a]+c < dis[b] { // 更新距离
                                dis[b] = dis[a] + c
                                heap.Push(&intHeap, [2]int{b, dis[b]})
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                if dis[i] == maxValue {
                        return -1
                }
                res = max(res, dis[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][1] < h[j][1]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
```

(续下页)

(接上页)

```
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 3
func networkDelayTime(times [][]int, n int, k int) int {
        maxValue := math.MaxInt32 / 10
        dis := make([]int, n) // k到其他点的距离
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[k-1] = 0
        for k := 0; k < n-1; k++ {
                flag := true
                for i := 0; i < len(times); i++ {
                        a, b, c := times[i][0]-1, times[i][1]-1, times[i][2] // a=>b
                        if dis[a]+c < dis[b] {
                                flag = false
                                dis[b] = dis[a] + c
                        }
                }
                if flag == true {
                        break
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                if dis[i] == maxValue {
                        return -1
                }
                res = max(res, dis[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

# 4
func networkDelayTime(times [][]int, n int, k int) int {
        maxValue := math.MaxInt32 / 10
        arr := make([][][2]int, n) // 邻接表：i=>j的集合
        for i := 0; i < len(times); i++ {
                a, b, c := times[i][0]-1, times[i][1]-1, times[i][2] // a=>b
                arr[a] = append(arr[a], [2]int{b, c})
        }
        dis := make([]int, n) // k到其他点的距离
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[k-1] = 0
        queue := make([]int, 0)
        queue = append(queue, k-1)
        for len(queue) > 0 {
                a := queue[0]
                queue = queue[1:]
                for i := 0; i < len(arr[a]); i++ {
                        b, c := arr[a][i][0], arr[a][i][1]
                        if dis[a]+c < dis[b] { // 更新距离
                                dis[b] = dis[a] + c
                                queue = append(queue, b)
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                if dis[i] == maxValue {
                        return -1
                }
                res = max(res, dis[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

# 5
func networkDelayTime(times [][]int, n int, k int) int {
        maxValue := math.MaxInt32 / 10
        arr := make([][]int, n) // 邻接表：i=>j的集合
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        if i == j {
                                continue
                        }
                        arr[i][j] = maxValue
                }
        }
        for i := 0; i < len(times); i++ {
                a, b, c := times[i][0]-1, times[i][1]-1, times[i][2] // a=>b
                arr[a][b] = c
        }
        for p := 0; p < n; p++ { // floyd
                for i := 0; i < n; i++ {
                        for j := 0; j < n; j++ {
                                arr[i][j] = min(arr[i][j], arr[i][p]+arr[p][j])
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                if arr[k-1][i] == maxValue {
                        return -1
                }
                res = max(res, arr[k-1][i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

23.17 752. 打开转盘锁 (1)

- 题目

```
你有一个带有四个圆形拨轮的转盘锁。
每个拨轮都有10个数字： '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' 。
每个拨轮可以自由旋转：例如把 '9' 变为  '0', '0' 变为 '9'␣
↪。每次旋转都只能旋转一个拨轮的一位数字。
锁的初始数字为 '0000' ，一个代表四个拨轮的数字的字符串。
列表 deadends 包含了一组死亡数字，一旦拨轮的数字和列表里的任何一个元素相同，
这个锁将会被永久锁定，无法再被旋转。
字符串 target␣
↪代表可以解锁的数字，你需要给出最小的旋转次数，如果无论如何不能解锁，返回 -1。
示例 1:输入：deadends = ["0201","0101","0102","1212","2002"], target = "0202" 输出：6
解释：可能的移动序列为 "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" ->
↪"0202"。
注意 "0000" -> "0001" -> "0002" -> "0102" -> "0202" 这样的序列是不能解锁的，
因为当拨动到 "0102" 时这个锁就会被锁定。
示例 2:输入：deadends = ["8888"], target = "0009" 输出：1
解释：  把最后一位反向旋转一次即可 "0000" -> "0009"。
示例 3:输入：deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"],
target = "8888" 输出：-1
解释：无法旋转到目标数字且不被锁定。
示例 4:输入：deadends = ["0000"], target = "8888" 输出：-1
提示：
    死亡列表 deadends 的长度范围为 [1, 500]。
    目标数字 target 不会在 deadends 之中。
    每个 deadends 和 target 中的字符串的数字会在 10,000 个可能的情况 '0000' 到 '9999'␣
↪中产生。
```

- 解题思路

```
func openLock(deadends []string, target string) int {
        m := make(map[string]int)
```

(续下页)

(接上页)

```
        m["0000"] = 0
        for i := 0; i < len(deadends); i++ {
                if deadends[i] == "0000" {
                        return -1
                }
                m[deadends[i]] = 0
        }
        if target == "0000" {
                return 0
        }
        if _, ok := m[target]; ok {
                return -1
        }
        queue := make([]string, 0)
        queue = append(queue, "0000")
        res := 0
        dir := []int{1, -1}
        for len(queue) > 0 {
                res++
                length := len(queue)
                for i := 0; i < length; i++ {
                        str := queue[i]
                        for j := 0; j < 4; j++ {
                                for k := 0; k < len(dir); k++ {
                                        char := string((int(str[j]-'0')+10+dir[k])%10␣
↪+ '0')
                                        newStr := str[:j] + char + str[j+1:]
                                        if _, ok := m[newStr]; ok {
                                                continue
                                        }
                                        queue = append(queue, newStr)
                                        m[newStr] = 1
                                        if newStr == target {
                                                return res
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return -1
}
```

23.18 754. 到达终点数字 (2)

- 题目

```
在一根无限长的数轴上，你站在0的位置。终点在target的位置。
每次你可以选择向左或向右移动。第 n 次移动（从 1 开始），可以走 n 步。
返回到达终点需要的最小移动次数。
示例 1:输入: target = 3 输出: 2
解释:第一次移动，从 0 到 1 。
第二次移动，从 1 到 3 。
示例 2:输入: target = 2 输出: 3
解释: 第一次移动，从 0 到 1 。
第二次移动，从 1 到 -1 。
第三次移动，从 -1 到 2 。
注意:target是在[-10^9, 10^9]范围中的非零整数。
```

- 解题思路

```
func reachNumber(target int) int {
        n := abs(target) // 负数转为正数，负数正数本质上都一样
        k := 0
        // S=1+...+k >= target
        // 差值：diff = S-target
        // diff为偶数：可以找到1~k之间的一个组合之和为diff/2
        // diff为奇数：需要考虑S=1~k+1或者S=1~K+2的情况，使得新diff为偶数
        for n > 0 {
                k = k + 1
                n = n - k
        }
        if n%2 == 0 {
                return k
        }
        return k + 1 + k%2
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func reachNumber(target int) int {
```

(续下页)

(接上页)

```
        n := abs(target) // 负数转为正数，负数正数本质上都一样
        k := 1
        // S=1+...+k >= target
        // 差值：diff = S-target
        // diff为偶数：可以找到1~k之间的一个组合之和为diff/2
        // diff为奇数：需要考虑S=1~k+1或者S=1~K+2的情况，使得新diff为偶数
        // =>求S>=target 并且使用diff为偶数
        for {
                sum := k * (k + 1) / 2
                if sum >= n && (sum-n)%2 == 0 {
                        return k
                }
                k++
        }
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

23.19 756. 金字塔转换矩阵

23.19.1 题目

```
现在，我们用一些方块来堆砌一个金字塔。 每个方块用仅包含一个字母的字符串表示。
使用三元组表示金字塔的堆砌规则如下：
对于三元组 ABC ，C 为顶层方块，方块 A 、B 分别作为方块 C 下一层的的左、右子块。
当且仅当 ABC 是被允许的三元组，我们才可以将其堆砌上。
初始时，给定金字塔的基层bottom，用一个字符串表示。一个允许的三元组列表allowed，每个三元组用一个长度为
↪3 的字符串表示。
如果可以由基层一直堆到塔尖就返回 true ，否则返回 false 。
示例 1：输入：bottom = "BCD", allowed = ["BCG", "CDE", "GEA", "FFF"] 输出：true
解释：可以堆砌成这样的金字塔：
    A
   / \
  G   E
 / \ / \
B   C   D
```

(续下页)

(接上页)

```
因为符合 BCG、CDE 和 GEA 三种规则。
示例 2: 输入: bottom = "AABA", allowed = ["AAA", "AAB", "ABA", "ABB", "BAC"]␣
↪输出: false
解释: 无法一直堆到塔尖。
注意, 允许存在像 ABC 和 ABD 这样的三元组, 其中 C != D。
提示: bottom 的长度范围在[2, 8]。
allowed 的长度范围在[0, 200]。
方块的标记字母范围为{'A', 'B', 'C', 'D', 'E', 'F', 'G'}。
```

23.19.2 解题思路

```

```

23.20 763. 划分字母区间 (2)

- 题目

```
字符串 S␣
↪由小写字母组成。我们要把这个字符串划分为尽可能多的片段, 同一个字母只会出现在其中的一个片段。
返回一个表示每个字符串片段的长度的列表。
示例 1: 输入: S = "ababcbacadefegdehijhklij" 输出: [9,7,8]
解释: 划分结果为 "ababcbaca", "defegde", "hijhklij"。
每个字母最多出现在一个片段中。
像 "ababcbacadefegde", "hijhklij" 的划分是错误的, 因为划分的片段数较少。
提示: S的长度在[1, 500]之间。
    S只包含小写字母 'a' 到 'z' 。
```

- 解题思路

```
func partitionLabels(S string) []int {
        m := make(map[byte]int)
        for i := 0; i < len(S); i++ {
                m[S[i]] = i
        }
        res := make([]int, 0)
        left := 0
        right := 0
        for i := 0; i < len(S); i++ {
                right = max(right, m[S[i]])
                if i == right {
```

(续下页)

(接上页)

```
                        res = append(res, right-left+1)
                        left = right + 1
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func partitionLabels(S string) []int {
        res := make([]int, 0)
        left := 0
        right := 0
        for i := 0; i < len(S); i++ {
                right = max(right, strings.LastIndex(S, string(S[i])))
                if i == right {
                        res = append(res, right-left+1)
                        left = right + 1
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

23.21 767. 重构字符串 (2)

- 题目

```
给定一个字符串S，检查是否能重新排布其中的字母，使得两相邻的字符不同。
若可行，输出任意可行的结果。若不可行，返回空字符串。
示例 1:输入: S = "aab" 输出: "aba"
示例 2:输入: S = "aaab" 输出: ""
注意: S 只包含小写字母并且长度在[1, 500]区间内。
```

- 解题思路

```
func reorganizeString(S string) string {
        n := len(S)
        if n <= 1 {
                return S
        }
        res := make([]byte, 0)
        m := make(map[byte]int)
        for _, v := range S {
                m[byte(v)]++
        }
        nodeHeap := &Heap{}
        heap.Init(nodeHeap)
        for k, v := range m {
                if v > (n+1)/2 {
                        return ""
                }
                heap.Push(nodeHeap, Node{
                        char: k,
                        num:  v,
                })
        }
        for nodeHeap.Len() >= 2 {
                node1 := heap.Pop(nodeHeap).(Node)
                node2 := heap.Pop(nodeHeap).(Node)
                res = append(res, node1.char, node2.char)
                node1.num--
                node2.num--
                if node1.num > 0 {
                        heap.Push(nodeHeap, node1)
                }
                if node2.num > 0 {
                        heap.Push(nodeHeap, node2)
```

(续下页)

(接上页)

```
                }
        }
        if nodeHeap.Len() > 0 {
                t := heap.Pop(nodeHeap).(Node)
                res = append(res, t.char)
        }
        return string(res)
}

type Node struct {
        char byte
        num  int
}

type Heap []Node

func (h Heap) Len() int {
        return len(h)
}

func (h Heap) Less(i, j int) bool {
        return h[i].num > h[j].num
}

func (h Heap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *Heap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *Heap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func reorganizeString(S string) string {
        arr := make([]node, 26)
        maxCount := 0
        for _, char := range S {
```

(续下页)

(接上页)

```
                index := char - 'a'
                arr[index].char = char
                arr[index].num++
                if arr[index].num > maxCount {
                        maxCount = arr[index].num
                }
        }
        if maxCount > (len(S)+1)/2 {
                return ""
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].num >= arr[j].num
        })
        res := make([]rune, len(S))
        var index int
        // 先偶后奇
        for i := 0; i < 2; i++ {
                for j := i; j < len(S); j = j + 2 {
                        if arr[index].num == 0 {
                                index++
                        }
                        res[j] = arr[index].char
                        arr[index].num--
                }
        }
        return string(res)
}
```

23.22 769. 最多能完成排序的块 (1)

- 题目

```
数组arr是[0, 1, ..., arr.length - 1]的一种排列，我们将这个数组分割成几个“块”，
并将这些块分别进行排序。之后再连接起来，使得连接的结果和按升序排序后的原数组相同。
我们最多能将数组分成多少块？
示例 1:输入: arr = [4,3,2,1,0] 输出: 1
解释:将数组分成2块或者更多块，都无法得到所需的结果。
例如，分成 [4, 3], [2, 1, 0] 的结果是 [3, 4, 0, 1, 2]，这不是有序的数组。
示例 2:输入: arr = [1,0,2,3,4] 输出: 4
解释:我们可以把它分成两块，例如 [1, 0], [2, 3, 4]。
然而，分成 [1, 0], [2], [3], [4] 可以得到最多的块数。
注意:arr 的长度在 [1, 10] 之间。
```

(续下页)

(接上页)

```
arr[i]是 [0, 1, ..., arr.length - 1]的一种排列。
```

- 解题思路

```
func maxChunksToSorted(arr []int) int {
        res := 0
        maxValue := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] > maxValue {
                        maxValue = arr[i]
                }
                if maxValue == i {
                        res++
                }
        }
        return res
}
```

23.23 775. 全局倒置与局部倒置 (3)

- 题目

```
数组A是[0, 1, ..., N - 1]的一种排列，N 是数组A的长度。
全局倒置指的是 i,j满足0 <= i < j < N 并且A[i] > A[j],
局部倒置指的是 i 满足0 <= i < N并且A[i] > A[i+1]。
当数组A中全局倒置的数量等于局部倒置的数量时，返回 true 。
示例 1:输入: A = [1,0,2] 输出: true
解释: 有 1 个全局倒置，和 1 个局部倒置。
示例 2:输入: A = [1,2,0] 输出: false
解释: 有 2 个全局倒置，和 1 个局部倒置。
注意:A 是[0, 1, ..., A.length - 1]的一种排列
A 的长度在[1, 5000]之间
这个问题的时间限制已经减少了。
```

- 解题思路

```
func isIdealPermutation(A []int) bool {
        if len(A) < 3 {
                return true
        }
        // 局部倒置首先是一个全局倒置，因此无需统计局部倒置
        // 只需要判断是否存在 0<=i<k<j<N 并且A[i]>A[j]
```

(续下页)

(接上页)

```
        maxValue := A[0] // 前2位数组最大值，如果存在maxValue > A[i]，则不会相等
        for i := 2; i < len(A); i++ {
                if maxValue > A[i] {
                        return false
                }
                if maxValue < A[i-1] {
                        maxValue = A[i-1]
                }
        }
        return true
}

# 2
func isIdealPermutation(A []int) bool {
        if len(A) < 3 {
                return true
        }
        minValue := len(A)
        for i := len(A) - 1; i >= 2; i-- {
                if A[i] < minValue {
                        minValue = A[i]
                }
                if A[i-2] > minValue {
                        return false
                }
        }
        return true
}

# 3
func isIdealPermutation(A []int) bool {
        if len(A) < 3 {
                return true
        }
        for i := 0; i < len(A); i++ {
                if abs(i-A[i]) > 1 {
                        return false
                }
        }
        return true
}

func abs(a int) int {
```

(续下页)

(接上页)

```
        if a < 0 {
                return -a
        }
        return a
}
```

23.24 777. 在 LR 字符串中交换相邻字符 (1)

- 题目

```
在一个由 'L' , 'R' 和 'X' 三个字符组成的字符串（例如"RXXLRXRXL"）中进行移动操作。
一次移动操作指用一个"LX"替换一个"XL"，或者用一个"XR"替换一个"RX"。
现给定起始字符串start和结束字符串end，请编写代码，
当且仅当存在一系列移动操作使得start可以转换成end时， 返回True。
示例 :输入: start = "RXXLRXRXL", end = "XRLXXRRLX" 输出: True
解释:我们可以通过以下几步将start转换成end:
RXXLRXRXL ->
XRXLRXRXL ->
XRLXRXRXL ->
XRLXXRRXL ->
XRLXXRRLX
提示: 1 <= len(start) = len(end) <= 10000。
start和end中的字符串仅限于'L', 'R'和'X'。
```

- 解题思路

```
func canTransform(start string, end string) bool {
        if strings.ReplaceAll(start, "X", "") != strings.ReplaceAll(end, "X", "") {
                return false
        }
        j := 0
        for i := 0; i < len(start); i++ {
                if start[i] == 'L' { // LX=>XL, L是往右的
                        for end[j] != 'L' {
                                j++
                        }
                        if i < j {
                                return false
                        }
                        j++
                }
        }
```

(续下页)

(接上页)

```
        j = 0
        for i := 0; i < len(start); i++ {
                if start[i] == 'R' { // XR=>RX, R是往左的
                        for end[j] != 'R' {
                                j++
                        }
                        if i > j {
                                return false
                        }
                        j++
                }
        }
        return true
}
```

23.25 779. 第 K 个语法符号 (3)

- 题目

```
在第一行我们写上一个 0。接下来的每一行，将前一行中的0替换为01，1替换为10。
给定行数 N 和序数 K，返回第 N 行中第 K个字符。（K从1开始）
例子:输入：N = 1, K = 1 输出：0
输入：N = 2, K = 1 输出：0
输入：N = 2, K = 2 输出：1
输入：N = 4, K = 5 输出：1
解释：
第一行：0
第二行：01
第三行：0110
第四行：01101001
注意：  N 的范围 [1, 30]. K 的范围 [1, 2^(N-1)].
```

- 解题思路

```
func kthGrammar(N int, K int) int {
        if K == 1 {
                return 0
        }
    // N行K的数是由N-1行(K+1)/2的数来的
        temp := kthGrammar(N-1, (K+1)/2)
        if K%2 == 1 {
                return temp
```

(续下页)

(接上页)

```
        }
        return 1 - temp
}

# 2
func kthGrammar(N int, K int) int {
        if K == 1 {
                return 0
        }
        total := int(math.Pow(2, float64(N-1)))
        half := total / 2
        if K <= half {
                return kthGrammar(N-1, K)
        }
        return 1 - kthGrammar(N-1, K-half)
}

# 3
func kthGrammar(N int, K int) int {
        return bits.OnesCount(uint(K-1))%2
}
```

23.26 781. 森林中的兔子 (2)

- 题目

```
森林中，每个兔子都有颜色。其中一些兔子（可能是全部）告诉你还有多少其他的兔子和自己有相同的颜色。
我们将这些回答放在 answers 数组里。
返回森林中兔子的最少数量。
示例:输入: answers = [1, 1, 2] 输出: 5
解释:两只回答了 "1" 的兔子可能有相同的颜色，设为红色。
之后回答了 "2" 的兔子不会是红色，否则他们的回答会相互矛盾。
设回答了 "2" 的兔子为蓝色。
此外，森林中还应有另外 2 只蓝色兔子的回答没有包含在数组中。
因此森林中兔子的最少数量是 5: 3 只回答的和 2 只没有回答的。
输入: answers = [10, 10, 10] 输出: 11
输入: answers = [] 输出: 0
说明: answers 的长度最大为1000。
    answers[i] 是在 [0, 999] 范围内的整数。
```

- 解题思路

```
func numRabbits(answers []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(answers); i++ {
                value := answers[i]
                if m[value] == 0 {
                        res = res + value + 1
                }
                m[value]++
                if m[value] == value+1 {
                        m[value] = 0
                }
        }
        return res
}

# 2
func numRabbits(answers []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(answers); i++ {
                value := answers[i]
                m[value]++
        }
        for k, v := range m {
                target := k + 1
                res = res + v/target*target
                if v%target > 0 {
                        res = res + target
                }
        }
        return res
}
```

23.27 785. 判断二分图 (3)

- 题目

```
存在一个 无向图 ，图中有 n 个节点。其中每个节点都有一个介于 0 到 n - 1␣
↪之间的唯一编号。
给你一个二维数组 graph ，其中 graph[u] 是一个节点数组，由节点 u␣
↪的邻接节点组成。形式上，对于graph[u] 中的每个 v ，都存在一条位于节点 u 和节点 v␣
```

(续下页)

(接上页)

```
↪之间的无向边。该无向图同时具有以下属性：
不存在自环（graph[u] 不包含 u）。
不存在平行边（graph[u] 不包含重复值）。
如果 v 在 graph[u] 内，那么 u 也应该在 graph[v] 内（该图是无向图）
这个图可能不是连通图，也就是说两个节点 u 和 v 之间可能不存在一条连通彼此的路径。
二分图 定义：如果能将一个图的节点集合分割成两个独立的子集 A 和 B ，
并使图中的每一条边的两个节点一个来自 A 集合，一个来自 B 集合，就将这个图称为 二分图 。
如果图是二分图，返回 true ；否则，返回 false 。
示例 1：输入：graph = [[1,2,3],[0,2],[0,1,3],[0,2]] 输出：false
解释：不能将节点分割成两个独立的子集，以使每条边都连通一个子集中的一个节点与另一个子集中的一个节点。
示例 2：输入：graph = [[1,3],[0,2],[1,3],[0,2]] 输出：true
解释：可以将节点分成两组：{0, 2} 和 {1, 3} 。
提示：graph.length == n
1 <= n <= 100
0 <= graph[u].length < n
0 <= graph[u][i] <= n - 1
graph[u] 不会包含 u
graph[u] 的所有值 互不相同
如果 graph[u] 包含 v，那么 graph[v] 也会包含 u
```

- 解题思路

```
// 思路同leetcode886.可能的二分法
var m map[int]int

func isBipartite(graph [][]int) bool {
        n := len(graph)
        m = make(map[int]int) // 分组： 0一组，1一组
        for i := 0; i < n; i++ {
                // 没有被分配过，分配到0一组
                if _, ok := m[i]; ok == false && dfs(graph, i, 0) == false {
                        return false
                }
        }
        return true
}

func dfs(arr [][]int, index int, value int) bool {
        if v, ok := m[index]; ok {
                return v == value // 已经分配，查看是否同一组
        }
        m[index] = value
        for i := 0; i < len(arr[index]); i++ {
                target := arr[index][i]
```

(续下页)

(接上页)

```
                if dfs(arr, target, 1-value) == false { //␣
↪不喜欢的人，分配到对立组：1-value
                        return false
                }
        }
        return true
}

# 2
func isBipartite(graph [][]int) bool {
        n := len(graph)
        m := make(map[int]int) // 分组： 0一组，1一组
        for i := 0; i < n; i++ {
                // 没有被分配过，分配到0一组
                if _, ok := m[i]; ok == true {
                        continue
                }
                m[i] = 0
                queue := make([]int, 0)
                queue = append(queue, i)
                for len(queue) > 0 {
                        node := queue[0]
                        queue = queue[1:]
                        for i := 0; i < len(graph[node]); i++ {
                                target := graph[node][i]
                                if _, ok := m[target]; ok == false {
                                        m[target] = 1 - m[node] // 相反一组
                                        queue = append(queue, target)
                                } else if m[node] == m[target] { //␣
↪已经分配，查看是否同一组
                                        return false
                                }
                        }
                }
        }
        return true
}

# 3
func isBipartite(graph [][]int) bool {
        n := len(graph)
        fa = Init(n)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < len(graph[i]); j++ {
                        target := graph[i][j]
                        if find(i) == find(target) { // 和不喜欢的人在相同组，失败
                                return false
                        }
                        union(graph[i][0], target) // 不喜欢的人在同一组
                }
        }
        return true
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        for x != fa[x] {
                fa[x] = fa[fa[x]]
                x = fa[x]
        }
        return x
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}
```

23.28 787.K 站中转内最便宜的航班 (4)

- 题目

```
有 n 个城市通过一些航班连接。给你一个数组flights ，
其中flights[i] = [fromi, toi, pricei] ，表示该航班都从城市 fromi 开始，以价格 pricei␣
↪抵达 toi。
现在给定所有的城市和航班，以及出发城市 src 和目的地 dst，你的任务是找到出一条最多经过␣
↪k站中转的路线，
使得从 src 到 dst 的 价格最便宜 ，并返回该价格。 如果不存在这样的路线，则输出 -1。
示例 1: 输入: n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]] src = 0, dst = 2, k = 1␣
↪输出: 200
解释: 城市航班图如下
从城市 0 到城市 2 在 1 站中转以内的最便宜价格是 200，如图中红色所示。
示例 2: 输入: n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]] src = 0, dst = 2, k = 0␣
↪输出: 500
解释: 城市航班图如下
从城市 0 到城市 2 在 0 站中转以内的最便宜价格是 500，如图中蓝色所示。
提示: 1 <= n <= 100
0 <= flights.length <= (n * (n - 1) / 2)
flights[i].length == 3
0 <= fromi, toi < n
fromi != toi
1 <= pricei <= 104
航班没有重复，且不存在自环
0 <= src, dst, k < n
src != dst
```

- 解题思路

```
func findCheapestPrice(n int, flights [][]int, src int, dst int, k int) int {
        maxValue := math.MaxInt32 / 10
        dp := make([][]int, k+2) // dp[i][j] =>␣
↪经过i次航班到j地需要的最少花费 (k次中转需要k+1次航班)
        for i := 0; i <= k+1; i++ {
                dp[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        dp[i][j] = maxValue
                }
        }
        dp[0][src] = 0 // 到开始地为0
        for i := 1; i <= k+1; i++ {
                for j := 0; j < len(flights); j++ {
                        a, b, c := flights[j][0], flights[j][1], flights[j][2] // a=>
```

(续下页)

(接上页)

```
↪b c
                        dp[i][b] = min(dp[i][b], dp[i-1][a]+c)
                }
        }
        res := maxValue
        for i := 1; i <= k+1; i++ {
                res = min(res, dp[i][dst])
        }
        if res == maxValue {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func findCheapestPrice(n int, flights [][]int, src int, dst int, k int) int {
        maxValue := math.MaxInt32 / 10
        dp := make([]int, n) // dp[i] => 到j地需要的最少花费（k次中转需要k+1次航班）
        for i := 0; i < n; i++ {
                dp[i] = maxValue
        }
        dp[src] = 0 // 到开始地为0
        res := maxValue
        for i := 1; i <= k+1; i++ {
                temp := make([]int, n)
                for j := 0; j < n; j++ {
                        temp[j] = maxValue
                }
                for j := 0; j < len(flights); j++ {
                        a, b, c := flights[j][0], flights[j][1], flights[j][2] // a=>
↪b c
                        temp[b] = min(temp[b], dp[a]+c)

                }
                res = min(res, temp[dst])
                dp = temp
```

(续下页)

(接上页)

```
        }
        if res == maxValue {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func findCheapestPrice(n int, flights [][]int, src int, dst int, k int) int {
        maxValue := math.MaxInt32 / 10
        prices := make([]int, n)
        arr := make([][][2]int, n)
        for i := 0; i < n; i++ {
                prices[i] = maxValue
        }
        prices[src] = 0
        for i := 0; i < len(flights); i++ {
                a, b, c := flights[i][0], flights[i][1], flights[i][2] // a=>b c
                arr[a] = append(arr[a], [2]int{b, c})
        }
        queue := make([][3]int, 0)
        queue = append(queue, [3]int{1, src, prices[src]}) // 次数，起点，价格
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node[0] > k+1 { // 大于k+1次退出
                        break
                }
                cur, value := node[1], node[2]
                for i := 0; i < len(arr[cur]); i++ {
                        b, c := arr[cur][i][0], arr[cur][i][1]
                        if prices[b] > c+value {
                                prices[b] = c + value
                                queue = append(queue, [3]int{node[0] + 1, b,␣
↪prices[b]})
                        }
```

(续下页)

(接上页)

```
                }
        }
        if prices[dst] == maxValue {
                return -1
        }
        return prices[dst]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func findCheapestPrice(n int, flights [][]int, src int, dst int, k int) int {
        maxValue := math.MaxInt32 / 10
        dis := make([]int, n)
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[src] = 0 // 到开始地为0
        for i := 1; i <= k+1; i++ {
                temp := make([]int, n)
                copy(temp, dis)
                for j := 0; j < len(flights); j++ {
                        a, b, c := flights[j][0], flights[j][1], flights[j][2] // a=>
↪b c
                        temp[b] = min(temp[b], dis[a]+c)
                }
                dis = temp
        }
        if dis[dst] == maxValue {
                return -1
        }
        return dis[dst]
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}
```

23.29 789. 逃脱阻碍者 (1)

- 题目

```
你在进行一个简化版的吃豆人游戏。你从 [0, 0] 点开始出发，你的目的地是target = [xtarget,
↪ ytarget] 。
地图上有一些阻碍者，以数组 ghosts 给出，第 i 个阻碍者从ghosts[i] = [xi, yi]出发。
所有输入均为 整数坐标 。
每一回合，你和阻碍者们可以同时向东，西，南，北四个方向移动，每次可以移动到距离原位置␣
↪1 个单位 的新位置。
当然，也可以选择 不动 。所有动作 同时 发生。
如果你可以在任何阻碍者抓住你 之前␣
↪到达目的地（阻碍者可以采取任意行动方式），则被视为逃脱成功。
如果你和阻碍者同时到达了一个位置（包括目的地）都不算是逃脱成功。
只有在你有可能成功逃脱时，输出 true ；否则，输出 false 。
示例 1：输入：ghosts = [[1,0],[0,3]], target = [0,1] 输出：true
解释：你可以直接一步到达目的地 (0,1) ，在 (1, 0) 或者 (0, 3)␣
↪位置的阻碍者都不可能抓住你。
示例 2：输入：ghosts = [[1,0]], target = [2,0] 输出：false
解释：你需要走到位于 (2, 0) 的目的地，但是在 (1, 0) 的阻碍者位于你和目的地之间。
示例 3：输入：ghosts = [[2,0]], target = [1,0] 输出：false
解释：阻碍者可以和你同时达到目的地。
示例 4：输入：ghosts = [[5,0],[-10,-2],[0,-5],[-2,-2],[-7,1]], target = [7,7]␣
↪输出：false
示例 5：输入：ghosts = [[-1,0],[0,1],[-1,0],[0,1],[-1,0]], target = [0,0] 输出：true
提示：1 <= ghosts.length <= 100
ghosts[i].length == 2
-104 <= xi, yi <= 104
同一位置可能有 多个阻碍者 。
target.length == 2
-104 <= xtarget, ytarget <= 104
```

- 解题思路

```
func escapeGhosts(ghosts [][]int, target []int) bool {
        a := abs(target[0]) + abs(target[1])
        for i := 0; i < len(ghosts); i++ {
                b := abs(ghosts[i][0]-target[0]) + abs(ghosts[i][1]-target[1])
                if b <= a {
```

(续下页)

(接上页)

```
                        return false
                }
        }
        return true
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

23.30 790. 多米诺和托米诺平铺 (2)

- 题目

```
有两种形状的瓷砖：一种是2x1 的多米诺形，另一种是形如"L"␣
↪的托米诺形。两种形状都可以旋转。
XX  <- 多米诺
XX  <- "L" 托米诺
X
给定N 的值，有多少种方法可以平铺2 x N 的面板？返回值 mod 10^9 + 7。
（平铺指的是每个正方形都必须有瓷砖覆盖。
两个平铺不同，当且仅当面板上有四个方向上的相邻单元中的两个，使得恰好有一个平铺有一个瓷砖占据两个正方
示例:输入：3 输出：5
解释：下面列出了五种不同的方法，不同字母代表不同瓷砖：
XYZ XXZ XYY XXY XYY
XYZ YYZ XZZ XYY XXY
提示：N 的范围是[1, 1000]
```

- 解题思路

```
var mod = 1000000007

func numTilings(N int) int {
        dp := [4]int{}
        dp[0] = 1
        for i := 0; i < N; i++ {
                temp := [4]int{}
                temp[0] = (dp[0] + dp[3]) % mod
                temp[1] = (dp[0] + dp[2]) % mod
```

(续下页)

(接上页)

```
                temp[2] = (dp[0] + dp[1]) % mod
                temp[3] = (dp[0] + dp[1] + dp[2]) % mod
                dp = temp
        }
        return dp[0]
}

# 2
func numTilings(N int) int {
        dp := make([]int, N+3)
        dp[1] = 1
        dp[2] = 2
        dp[3] = 5
        for i := 4; i <= N; i++ {
                dp[i] = (2*dp[i-1] + dp[i-3]) % 1000000007
        }
        return dp[N]
}
```

23.31 791. 自定义字符串排序 (3)

- 题目

```
字符串S和 T 只包含小写字符。在S中，所有字符只会出现一次。
S 已经根据某种规则进行了排序。我们要根据S中的字符顺序对T进行排序。
更具体地说，如果S中x在y之前出现，那么返回的字符串中x也应出现在y之前。
返回任意一种符合条件的字符串T。
示例: 输入:
S = "cba"
T = "abcd"
输出: "cbad"
解释: S中出现了字符 "a", "b", "c", 所以 "a", "b", "c" 的顺序应该是 "c", "b", "a".
由于 "d" 没有在S中出现，它可以放在T的任意位置. "dcba", "cdba", "cbda" 都是合法的输出。
注意:S的最大长度为26，其中没有重复的字符。
T的最大长度为200。
S和T只包含小写字符。
```

- 解题思路

```
func customSortString(S string, T string) string {
        m := make(map[uint8]int)
        for i := 0; i < len(S); i++ {
```

(续下页)

(接上页)

```
                m[S[i]] = i
        }
        arr := []byte(T)
        sort.Slice(arr, func(i, j int) bool {
                return m[arr[i]] < m[arr[j]]
        })
        return string(arr)
}

# 2
func customSortString(S string, T string) string {
        count := make([]int, 26)
        for i := 0; i < len(T); i++ {
                count[T[i]-'a']++
        }
        res := make([]byte, 0)
        for i := 0; i < len(S); i++ {
                for j := 0; j < count[S[i]-'a']; j++ {
                        res = append(res, S[i])
                }
                count[S[i]-'a'] = 0
        }
        for i := 0; i < 26; i++ {
                for j := 0; j < count[i]; j++ {
                        res = append(res, byte(i+'a'))
                }
        }
        return string(res)
}

# 3
func customSortString(S string, T string) string {
        res := []byte(T)
        index := 0
        for i := 0; i < len(S); i++ {
                for j := 0; j < len(res); j++ {
                        if res[j] == S[i] {
                                res[j], res[index] = res[index], res[j]
                                index++
                        }
                }
        }
        return string(res)
}
```

23.32 792. 匹配子序列的单词数 (2)

- 题目

```
给定字符串 S 和单词字典 words, 求words[i]中是S的子序列的单词个数。
示例:输入: S = "abcde" words = ["a", "bb", "acd", "ace"] 输出: 3
解释: 有三个是S 的子序列的单词: "a", "acd", "ace"。
注意:所有在words和S里的单词都只由小写字母组成。
S 的长度在[1, 50000]。
words的长度在[1, 5000]。
words[i]的长度在[1, 50]。
```

- 解题思路

```
func numMatchingSubseq(S string, words []string) int {
        res := 0
        m := make(map[string]int)
        for i := 0; i < len(words); i++ {
                m[words[i]]++
        }
        for k, v := range m {
                if judge(S, k) == true {
                        res = res + v
                }
        }
        return res
}

func judge(S string, str string) bool {
        for i, j := 0, 0; i < len(S) && j < len(str); i++ {
                if S[i] == str[j] {
                        j++
                }
                if j == len(str) {
                        return true
                }
        }
        return false
}

# 2
func numMatchingSubseq(S string, words []string) int {
        res := 0
        for i := 0; i < len(words); i++ {
```

(续下页)

(接上页)

```
                if len(words[i]) > len(S) {
                        continue
                }
                k := 0
                for j := 0; j < len(S); j++ {
                        if S[j] == words[i][k] {
                                k++
                                if k == len(words[i]) {
                                        res++
                                        break
                                }
                        }
                }
        }
        return res
}
```

23.33 794. 有效的井字游戏 (1)

- 题目

```
用字符串数组作为井字游戏的游戏板board。当且仅当在井字游戏过程中，玩家有可能将字符放置成游戏板所显示的
↪true。
该游戏板是一个 3 x 3 数组，由字符" "，"X"和"O"组成。字符" "代表一个空位。
以下是井字游戏的规则：
玩家轮流将字符放入空位（" "）中。
第一个玩家总是放字符 “X”，且第二个玩家总是放字符 “O”。
“X” 和 “O” 只允许放置在空位中，不允许对已放有字符的位置进行填充。
当有 3 个相同（且非空）的字符填充任何行、列或对角线时，游戏结束。
当所有位置非空时，也算为游戏结束。
如果游戏结束，玩家不允许再放置字符。
示例 1:输入: board = ["O ", " ", " "] 输出: false
解释: 第一个玩家总是放置 “X”。
示例 2:输入: board = ["XOX", " X ", "   "] 输出: false
解释: 玩家应该是轮流放置的。
示例 3:输入: board = ["XXX", "   ", "OOO"] 输出: false
示例 4:输入: board = ["XOX", "O O", "XOX"] 输出: true
说明:游戏板board是长度为 3 的字符串数组，其中每个字符串board[i]的长度为3。
board[i][j]是集合{" ", "X", "O"}中的一个字符。
```

- 解题思路

```
func validTicTacToe(board []string) bool {
        var XCount, OCount int
        n := len(board)
        rows := make([][2]int, n)          // 行
        cols := make([][2]int, n)          // 列
        left, right := [2]int{}, [2]int{} // 对角线
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if board[i][j] == 'X' {
                                XCount++
                                rows[i][0]++
                                cols[j][0]++
                                if i == j {
                                        left[0]++
                                }
                                if i == n-1-j {
                                        right[0]++
                                }
                        } else if board[i][j] == 'O' {
                                OCount++
                                rows[i][1]++
                                cols[j][1]++
                                if i == j {
                                        left[1]++
                                }
                                if i == n-1-j {
                                        right[1]++
                                }
                        }
                }
        }
        if XCount != OCount && XCount-1 != OCount {
                return false
        }
        for i := 0; i < n; i++ { // 行列判断
                if (rows[i][0] == n || cols[i][0] == n) && XCount-1 != OCount {
                        return false
                }
                if (rows[i][1] == n || cols[i][1] == n) && XCount != OCount {
                        return false
                }
        }
        if (left[0] == n || right[0] == n) && XCount-1 != OCount { // 对角线判断
                return false
```

(续下页)

(接上页)

```
        }
        if (left[1] == n || right[1] == n) && XCount != OCount {
                return false
        }
        return true
}
```

23.34 795. 区间子数组个数 (4)

- 题目

```
给定一个元素都是正整数的数组A, 正整数 L以及R(L <= R)。
求连续、非空且其中最大元素满足大于等于L小于等于R的子数组个数。
例如 : 输入: A = [2, 1, 4, 3] L = 2 R = 3  输出: 3
解释: 满足条件的子数组: [2], [2, 1], [3].
注意:L, R 和A[i] 都是整数, 范围在[0, 10^9]。
数组A的长度范围在[1, 50000]。
```

- 解题思路

```
func numSubarrayBoundedMax(nums []int, left int, right int) int {
        // L~R范围的组合数=0~R范围的组合数- 0~L-1范围的组合数
        return count(nums, right) - count(nums, left-1)
}

func count(nums []int, target int) int {
        res := 0
        total := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] <= target {
                        total++
                } else {
                        total = 0
                }
                res = res + total
        }
        return res
}

# 2
func numSubarrayBoundedMax(nums []int, left int, right int) int {
        // L~R范围的组合数=0~R范围的组合数- 0~L-1范围的组合数
```

(续下页)

(接上页)

```
        return count(nums, right) - count(nums, left-1)
}

func count(nums []int, target int) int {
        res := 0
        n := len(nums)
        dp := make([]int, n)
        if nums[0] <= target {
                dp[0] = 1
        }
        res = res + dp[0]
        for i := 1; i < len(nums); i++ {
                if nums[i] <= target {
                        dp[i] = dp[i-1] + 1
                }
                res = res + dp[i]
        }
        return res
}

# 3
func numSubarrayBoundedMax(nums []int, left int, right int) int {
        // L~R范围的组合数=0~R范围的组合数- 0~L-1范围的组合数
        return count(nums, right) - count(nums, left-1)
}

func count(nums []int, target int) int {
        res := 0
        total := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] <= target {
                        total++
                } else {
                        res = res + (total+1)*total/2
                        total = 0
                }
        }
        res = res + (total+1)*total/2
        return res
}

# 4
func numSubarrayBoundedMax(nums []int, left int, right int) int {
```

(续下页)

(接上页)

```
        res := 0
        j := -1
        count := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] > right {
                        j = i
                }
                if nums[i] >= left {
                        count = i - j // 满足要求，如果大于right，则count=0
                }
                res = res + count
        }
        return res
}
```

23.35 797. 所有可能的路径 (1)

- 题目

```
给一个有n个结点的有向无环图，找到所有从0到n-1的路径并输出（不要求按顺序）
二维数组的第 i 个数组中的单元都表示有向图中 i 号结点所能到达的下一些结点
（译者注：有向图是有方向的，即规定了 a→b 你就不能从 b→a ）空就是没有下一个结点了。
示例 1：输入：graph = [[1,2],[3],[3],[]] 输出：[[0,1,3],[0,2,3]]
解释：有两条路径 0 -> 1 -> 3 和 0 -> 2 -> 3
示例 2：输入：graph = [[4,3,1],[3,2,4],[3],[4],[]]
输出：[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]
示例 3：输入：graph = [[1],[]] 输出：[[0,1]]
示例 4：输入：graph = [[1,2,3],[2],[3],[]] 输出：[[0,1,2,3],[0,2,3],[0,3]]
示例 5：输入：graph = [[1,3],[2],[3],[]] 输出：[[0,1,2,3],[0,3]]
提示：结点的数量会在范围[2, 15]内。
你可以把路径以任意顺序输出，但在路径内的结点的顺序必须保证。
```

- 解题思路

```
var res [][]int

func allPathsSourceTarget(graph [][]int) [][]int {
        res = make([][]int, 0)
        dfs(graph, 0, len(graph)-1, make([]int, 0))
        return res
}
```

(续下页)

(接上页)

```
func dfs(graph [][]int, cur, target int, path []int) {
        if cur == target {
                path = append(path, cur)
                temp := make([]int, len(path))
                copy(temp, path)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(graph[cur]); i++ {
                dfs(graph, graph[cur][i], target, append(path, cur))
        }
}
```

23.36 799. 香槟塔 (1)

- 题目

```
我们把玻璃杯摆成金字塔的形状，其中第一层有1个玻璃杯，
第二层有2个，依次类推到第100层，每个玻璃杯(250ml)将盛有香槟。
从顶层的第一个玻璃杯开始倾倒一些香槟，当顶层的杯子满了，
任何溢出的香槟都会立刻等流量的流向左右两侧的玻璃杯。
当左右两边的杯子也满了，就会等流量的流向它们左右两边的杯子，依次类推。
（当最底层的玻璃杯满了，香槟会流到地板上）
例如，在倾倒一杯香槟后，最顶层的玻璃杯满了。倾倒了两杯香槟后，第二层的两个玻璃杯各自盛放一半的香槟。
在倒三杯香槟后，第二层的香槟满了 - 此时总共有三个满的玻璃杯。
在倒第四杯后，第三层中间的玻璃杯盛放了一半的香槟，他两边的玻璃杯各自盛放了四分之一的香槟，如下图所示
现在当倾倒了非负整数杯香槟后，返回第 i 行 j 个玻璃杯所盛放的香槟占玻璃杯容积的比例（i
↪和 j都从0开始）。
示例 1:输入: poured(倾倒香槟总杯数) = 1, query_glass(杯子的位置数) = 1, query_
↪row(行数) = 1
输出: 0.0
解释:
↪我们在顶层（下标是（0, 0））倒了一杯香槟后，没有溢出，因此所有在顶层以下的玻璃杯都是空的。
示例 2:输入: poured(倾倒香槟总杯数) = 2, query_glass(杯子的位置数) = 1, query_
↪row(行数) = 1
输出: 0.5
解释:
↪我们在顶层（下标是（0, 0）倒了两杯香槟后，有一杯量的香槟将从顶层溢出，位于（1, 0）的玻璃杯和（1, 1
注意:poured的范围[0, 10 ^ 9]。
query_glass和query_row的范围[0, 99]。
```

- 解题思路

```
func champagneTower(poured int, query_row int, query_glass int) float64 {
        n, m := query_row, query_glass
        dp := make([][]float64, n+2)
        for i := 0; i < n+2; i++ {
                dp[i] = make([]float64, n+2)
        }
        dp[0][0] = float64(poured) // 初始值
        for i := 0; i <= n; i++ {
                for j := 0; j <= i; j++ {
                        if dp[i][j] > 1 {
                                dp[i+1][j] = dp[i+1][j] + (dp[i][j]-1)/2.0      //␣
↪往左分
                                dp[i+1][j+1] = dp[i+1][j+1] + (dp[i][j]-1)/2.0 //␣
↪往右分
                        }
                }
        }
        if dp[n][m] > 1 {
                return 1.0
        }
        return dp[n][m]
}
```

CHAPTER 24

0701-0800-Hard

24.1 710. 黑名单中的随机数 (1)

- 题目

```
给定一个包含 [0, n ) 中独特的整数的黑名单 B, 写一个函数从 [ 0, n ) 中返回一个不在 B␣
↪中的随机整数。
对它进行优化使其尽量少调用系统方法 Math.random() 。
提示:1 <= N <= 1000000000
0 <= B.length < min(100000, N)
[0, N)不包含N, 详细参见interval notation。
示例 1:输入: ["Solution","pick","pick","pick"] [[1,[]],[],[],[]]
输出: [null,0,0,0]
示例 2:输入: ["Solution","pick","pick","pick"] [[2,[]],[],[],[]]
输出: [null,1,1,1]
示例 3:输入: ["Solution","pick","pick","pick"] [[3,[1]],[],[],[]]
Output: [null,0,0,2]
示例 4:输入: ["Solution","pick","pick","pick"] [[4,[2]],[],[],[]]
输出: [null,1,3,1]
输入语法说明:
输入是两个列表: 调用成员函数名和调用的参数。Solution的构造函数有两个参数, N和黑名单B。
pick没有参数, 输入参数是一个列表, 即使参数为空, 也会输入一个 [] 空列表。
```

- 解题思路

```
type Solution struct {
        m map[int]int
        N int
}

func Constructor(N int, blacklist []int) Solution {
        m := make(map[int]int)
        temp := make(map[int]bool)
        for i := 0; i < len(blacklist); i++ {
                temp[blacklist[i]] = true
        }
        length := N - len(blacklist)
        arr := make([]int, 0) // 需要替换为较大数
        for i := 0; i < len(blacklist); i++ {
                if blacklist[i] < length {
                        arr = append(arr, blacklist[i])
                }
        }
        a := make([]int, 0) // 没有使用过的较大数
        for i := length; i < N; i++ {
                if temp[i] == false {
                        a = append(a, i)
                }
        }

        for i := 0; i < len(a); i++ {
                m[arr[i]] = a[i]
        }
        return Solution{
                m: m,
                N: length,
        }
}

func (this *Solution) Pick() int {
        index := rand.Intn(this.N)
        if value, ok := this.m[index]; ok {
                return value
        }
        return index
}
```

24.2 719. 找出第 k 小的距离对 (2)

- 题目

```
给定一个整数数组，返回所有数对之间的第 k 个最小距离。一对 (A, B) 的距离被定义为 A 和␣
↪B 之间的绝对差值。
示例 1:输入：nums = [1,3,1] k = 1 输出：0
解释：所有数对如下：
(1,3) -> 2
(1,1) -> 0
(3,1) -> 2
因此第 1 个最小距离的数对是 (1,1)，它们之间的距离为 0。
提示:2 <= len(nums) <= 10000.
0 <= nums[i] < 1000000.
1 <= k <= len(nums) * (len(nums) - 1) / 2.
```

- 解题思路

```
func smallestDistancePair(nums []int, k int) int {
        sort.Ints(nums)
        n := len(nums)
        left, right := 0, nums[n-1]-nums[0]
        for left < right {
                mid := left + (right-left)/2
                if judge(nums, mid, k) == true {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

func judge(nums []int, mid, k int) bool {
        count := 0
        left := 0
        for right := 0; right < len(nums); right++ {
                for nums[right]-nums[left] > mid {
                        left++
                }
                count = count + right - left
        }
        return k <= count
}
```

(续下页)

(接上页)

```
# 2
func smallestDistancePair(nums []int, k int) int {
        sort.Ints(nums)
        n := len(nums)
        arr := make([]int, nums[n-1]-nums[0]+1)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        arr[nums[j]-nums[i]]++
                }
        }
        count := 0
        for i := 0; i < len(arr); i++ {
                count = count + arr[i]
                if count >= k {
                        return i
                }
        }
        return 0
}
```

24.3 732. 我的日程安排表 III(2)

- 题目

```
当 k 个日程安排有一些时间上的交叉时（例如 k 个日程安排都在同一时间内），就会产生 k 次预订。
给你一些日程安排 [start, end) ，请你在每个日程安排添加后，返回一个整数 k ，表示所有先前日程安排会产生的最大 k 次预订。
实现一个 MyCalendarThree 类来存放你的日程安排，你可以一直添加新的日程安排。
MyCalendarThree() 初始化对象。
int book(int start, int end) 返回一个整数 k ，表示日历中存在的 k 次预订的最大值。
示例：输入： ["MyCalendarThree", "book", "book", "book", "book", "book", "book"]
[[], [10, 20], [50, 60], [10, 40], [5, 15], [5, 10], [25, 55]]
输出：[null, 1, 1, 2, 3, 3, 3]
解释：MyCalendarThree myCalendarThree = new MyCalendarThree();
myCalendarThree.book(10, 20); // 返回 1 ，第一个日程安排可以预订并且不存在相交，所以最大 k 次预订是 1 次预订。
myCalendarThree.book(50, 60); // 返回 1 ，第二个日程安排可以预订并且不存在相交，所以最大 k 次预订是 1 次预订。
myCalendarThree.book(10, 40); // 返回 2 ，第三个日程安排 [10, 40) 与第一个日程安排相交，所以最大 k 次预订是 2 次预订。
```

(续下页)

(接上页)

```
myCalendarThree.book(5, 15); // 返回 3 ，剩下的日程安排的最大 k 次预订是 3 次预订。
myCalendarThree.book(5, 10); // 返回 3
myCalendarThree.book(25, 55); // 返回 3
提示：0 <= start < end <= 109
每个测试用例，调用 book函数最多不超过400次
```

- 解题思路

```
type MyCalendarThree struct {
        m map[int]int
}

func Constructor() MyCalendarThree {
        return MyCalendarThree{m: make(map[int]int)}
}

func (this *MyCalendarThree) Book(start int, end int) int {
        this.m[start]++
        this.m[end]--
        arr := make([]int, 0)
        for k := range this.m {
                arr = append(arr, k)
        }
        sort.Ints(arr)
        res := 0
        sum := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + this.m[arr[i]]
                if sum > res {
                        res = sum
                }
        }
        return res
}

# 2
type MyCalendarThree struct {
        root *Node
}

func Constructor() MyCalendarThree {
        return MyCalendarThree{root: &Node{
                start: 0,
                end:   1000000000,
```

(续下页)

(接上页)

```
        }}
}

func (this *MyCalendarThree) Book(start int, end int) int {
        return this.root.Insert(start, end)
}

type Node struct {
        start int
        end   int
        count int
        delay int // 延迟更新线段树
        left  *Node
        right *Node
}

func (root *Node) getMid() int {
        return (root.start + root.end) / 2
}

func (root *Node) Left() *Node {
        if root.left == nil {
                root.left = &Node{
                        start: root.start,
                        end:   root.getMid(),
                }
        }
        return root.left
}

func (root *Node) Right() *Node {
        if root.right == nil {
                root.right = &Node{
                        start: root.getMid(),
                        end:   root.end,
                }
        }
        return root.right
}

func (root *Node) Insert(s, e int) int {
        if s <= root.start && root.end <= e { // 包含
                root.delay++
```

(续下页)

(接上页)

```
                root.count++
        } else if s < root.end && root.start < e { // 相交
                // 自上向下延迟更新
                root.Left().count = root.Left().count + root.delay
                root.Left().delay = root.Left().delay + root.delay
                root.Right().count = root.Right().count + root.delay
                root.Right().delay = root.Right().delay + root.delay
                root.delay = 0
                a := root.Left().Insert(s, e)
                b := root.Right().Insert(s, e)
                root.count = max(root.count, max(a, b))
        }
        return root.count
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

24.4 757. 设置交集大小至少为 2(2)

- 题目

```
一个整数区间[a, b](a < b) 代表着从a到b的所有连续整数，包括a和b。
给你一组整数区间intervals，请找到一个最小的集合 S，使得 S␣
↪里的元素与区间intervals中的每一个整数区间都至少有2个元素相交。
输出这个最小集合S的大小。
示例 1:输入: intervals = [[1, 3], [1, 4], [2, 5], [3, 5]] 输出: 3
解释:考虑集合 S = {2, 3, 4}. S与intervals中的四个区间都有至少2个相交的元素。
且这是S最小的情况，故我们输出3。
示例 2:输入: intervals = [[1, 2], [2, 3], [2, 4], [4, 5]] 输出: 5
解释: 最小的集合S = {1, 2, 3, 4, 5}.
注意:intervals的长度范围为[1, 3000]。
intervals[i]长度为2，分别代表左、右边界。
intervals[i][j] 的值是[0, 10^8]范围内的整数。
```

- 解题思路

```
func intersectionSizeTwo(intervals [][]int) int {
        n := len(intervals)
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = 2 // 每个区间还需要找到的交点的个数默认为2
        }
        res := 0
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][0] == intervals[j][0] {
                        return intervals[i][1] > intervals[j][1]
                }
                return intervals[i][0] < intervals[j][0]
        })
        for i := n - 1; i >= 0; i-- {
                start := intervals[i][0]
                for k := start; k < start+arr[i]; k++ { //␣
↪一般最多取前面2个（start、start+1）
                        for j := i - 1; j >= 0; j-- { // 往前遍历
                                if arr[j] > 0 && k <= intervals[j][1] { //␣
↪当前的start或者start+1小于前面的end，交点arr[j]-1
                                        arr[j]--
                                }
                        }
                }
                res = res + arr[i]
        }
        return res
}

# 2
func intersectionSizeTwo(intervals [][]int) int {
        n := len(intervals)
        arr := []int{-1, -1}
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][1] == intervals[j][1] {
                        return intervals[i][0] > intervals[j][0]
                }
                return intervals[i][1] < intervals[j][1]
        })
        for i := 0; i < n; i++ {
                start, end := intervals[i][0], intervals[i][1]
                a, b := arr[len(arr)-2], arr[len(arr)-1]
                if start <= a { // 上一个开始值已经包括当前start
                        continue
```

(续下页)

(接上页)

```
                }
                if b < start { // 当前开始大于之前结束，把[end-1,end]包括
                        arr = append(arr, end-1)
                }
                arr = append(arr, end)
        }
        return len(arr) - 2
}
```

24.5 765. 情侣牵手 (2)

- 题目

```
N 对情侣坐在连续排列的 2N 个座位上，想要牵到对方的手。
计算最少交换座位的次数，以便每对情侣可以并肩坐在一起。␣
↪一次交换可选择任意两人，让他们站起来交换座位。
人和座位用0到2N-1的整数表示，情侣们按顺序编号，
第一对是(0, 1)，第二对是(2, 3)，以此类推，最后一对是(2N-2, 2N-1)。
这些情侣的初始座位row[i]是由最初始坐在第 i 个座位上的人决定的。
示例 1:输入: row = [0, 2, 1, 3] 输出: 1
解释: 我们只需要交换row[1]和row[2]的位置即可。
示例 2:输入: row = [3, 2, 0, 1] 输出: 0
解释: 无需交换座位，所有的情侣都已经可以手牵手了。
说明:len(row) 是偶数且数值在[4, 60]范围内。
可以保证row 是序列0...len(row)-1的一个全排列。
```

- 解题思路

```
func minSwapsCouples(row []int) int {
        n := len(row) / 2
        fa := make([]int, n)
        for i := 0; i < n; i++ {
                fa[i] = i
        }
        // 将每张沙发上的两个人员编号union一下，如果本来编号就相同，则表示两个人是一类
        for i := 0; i < len(row); i = i + 2 {
                a, b := row[i]/2, row[i+1]/2
                union(fa, a, b)
        }
        res := 0
        for i := 0; i < n; i++ {
                // 几个相同，就有几个环
```

(续下页)

(接上页)

```
                if find(fa, i) == i {
                        res++
                }
        }
        // 如果3组1个环，需要的次数是3-1=2，另外4组1个环，需要的次数是4-1=3。4+3-2=5
        // 组数-减去环数
        return n - res
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}

func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
                a = fa[a]
        }
        return a
}

# 2
func minSwapsCouples(row []int) int {
        res := 0
        for i := 0; i < len(row); i = i + 2 {
                a, b := row[i], row[i+1]
                if b == a^1 {
                        continue
                }
                res = res + 1
                for j := i + 1; j < len(row); j++ {
                        if row[j] == a^1 {
                                row[j], row[i+1] = row[i+1], row[j]
                                break
                        }
                }
        }
        return res
}
```

24.6 768. 最多能完成排序的块 II(4)

- 题目

```
这个问题和“最多能完成排序的块”相似，但给定数组中的元素可以重复，输入数组最大长度为2000，其中的元素
arr是一个可能包含重复元素的整数数组，我们将这个数组分割成几个“块”，并将这些块分别进行排序。
之后再连接起来，使得连接的结果和按升序排序后的原数组相同。
我们最多能将数组分成多少块？
示例1:输入: arr = [5,4,3,2,1] 输出: 1
解释: 将数组分成2块或者更多块，都无法得到所需的结果。
例如，分成 [5, 4], [3, 2, 1] 的结果是 [4, 5, 1, 2, 3]，这不是有序的数组。
示例 2:输入: arr = [2,1,3,4,4] 输出: 4
解释: 我们可以把它分成两块，例如 [2, 1], [3, 4, 4]。
然而，分成 [2, 1], [3], [4], [4] 可以得到最多的块数。
注意: arr的长度在[1, 2000]之间。
arr[i]的大小在[0, 10**8]之间。
```

- 解题思路

```
func maxChunksToSorted(arr []int) int {
        res := 0
        n := len(arr)
        target := make([]int, n)
        copy(target, arr)
        sort.Ints(target)
        m := make(map[int]int)
        count := 0
        for i := 0; i < n; i++ {
                m[arr[i]]++
                if m[arr[i]] == 0 {
                        count--
                } else if m[arr[i]] == 1 {
                        count++
                }
                m[target[i]]--
                if m[target[i]] == 0 {
                        count--
                } else if m[target[i]] == -1 {
                        count++
                }
                if count == 0 {
                        res++
                }
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func maxChunksToSorted(arr []int) int {
        res := 0
        n := len(arr)
        target := make([]int, n)
        copy(target, arr)
        sort.Ints(target)
        diff := 0 // 不同
        for i := 0; i < n; i++ {
                diff = diff + arr[i] - target[i]
                if diff == 0 { // 累计次数抵消后为0，次数+1
                        res++
                }
        }
        return res
}

# 3
func maxChunksToSorted(arr []int) int {
        res := 0
        n := len(arr)
        m := make(map[int]int)
        temp := make([][2]int, n)
        for i := 0; i < n; i++ {
                m[arr[i]]++
                temp[i] = [2]int{arr[i], m[arr[i]]}
        }
        target := make([][2]int, n)
        copy(target, temp)
        sort.Slice(target, func(i, j int) bool {
                if target[i][0] == target[j][0] {
                        return target[i][1] < target[j][1]
                }
                return target[i][0] < target[j][0]
        })
        cur := temp[0]
        for i := 0; i < n; i++ {
                if compare(cur, temp[i]) == true { // 小于temp[i]更新
                        cur = temp[i]
                }
```

(续下页)

(接上页)

```
                if cur == target[i] {
                        res++
                }
        }
        return res
}

func compare(a, b [2]int) bool {
        if a[0] == b[0] {
                return a[1] < b[1]
        }
        return a[0] < b[0]
}

# 4
func maxChunksToSorted(arr []int) int {
        stack := make([]int, 0) // 递增栈
        for i := 0; i < len(arr); i++ {
                if len(stack) > 0 && arr[i] < stack[len(stack)-1] {
                        top := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        for len(stack) > 0 && arr[i] < stack[len(stack)-1] {
                                stack = stack[:len(stack)-1]
                        }
                        stack = append(stack, top)
                } else {
                        stack = append(stack, arr[i])
                }
        }
        return len(stack)
}
```

24.7 773. 滑动谜题

24.7.1 题目

```
在一个 2 x 3 的板上（board）有 5 块砖瓦，用数字 1~5 来表示，以及一块空缺用0来表示.
一次移动定义为选择0与一个相邻的数字（上下左右）进行交换.
最终当板board的结果是[[1,2,3],[4,5,0]]谜板被解开。
给出一个谜板的初始状态，返回最少可以通过多少次移动解开谜板，如果不能解开谜板，则返回 -
↪1 。
```

(续下页)

(接上页)

```
示例：输入：board = [[1,2,3],[4,0,5]] 输出：1
解释：交换 0 和 5 ，1 步完成
输入：board = [[1,2,3],[5,4,0]] 输出：-1
解释：没有办法完成谜板
输入：board = [[4,1,2],[5,0,3]] 输出：5
解释：最少完成谜板的最少移动次数是 5 ，
一种移动路径：
尚未移动：[[4,1,2],[5,0,3]]
移动 1 次：[[4,1,2],[0,5,3]]
移动 2 次：[[0,1,2],[4,5,3]]
移动 3 次：[[1,0,2],[4,5,3]]
移动 4 次：[[1,2,0],[4,5,3]]
移动 5 次：[[1,2,3],[4,5,0]]
输入：board = [[3,2,4],[1,5,0]] 输出：14
提示：board是一个如上所述的 2 x 3 的数组.
board[i][j]是一个[0, 1, 2, 3, 4, 5]的排列.
```

24.7.2 解题思路

24.8 778. 水位上升的泳池中游泳 (4)

- 题目

```
在一个 N x N 的坐标方格grid 中，每一个方格的值 grid[i][j] 表示在位置 (i,j)␣
↪的平台高度。
现在开始下雨了。当时间为t时，此时雨水导致水池中任意位置的水位为t。
你可以从一个平台游向四周相邻的任意一个平台，但是前提是此时水位必须同时淹没这两个平台。
假定你可以瞬间移动无限距离，也就是默认在方格内部游动是不耗时的。
当然，在你游泳的时候你必须待在坐标方格里面。
你从坐标方格的左上平台 (0，0) 出发。最少耗时多久你才能到达坐标方格的右下平台(N-1，N-
↪1)？
示例 1:输入：[[0,2],[1,3]] 输出：3
解释:时间为0时，你位于坐标方格的位置为 (0，0)。
此时你不能游向任意方向，因为四个相邻方向平台的高度都大于当前时间为 0 时的水位。
等时间到达 3 时，你才可以游向平台 (1，1). 因为此时的水位是␣
↪3，坐标方格中的平台没有比水位 3 更高的，
所以你可以游向坐标方格中的任意位置
示例2:输入：[[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,
```

(续下页)

(接上页)

```
↪6]] 输出：16
解释：
 0  1  2  3  4
24 23 22 21  5
12 13 14 15 16
11 17 18 19 20
10  9  8  7  6
最终的路线用加粗进行了标记。
我们必须等到时间为 16，此时才能保证平台 (0, 0) 和 (4, 4) 是连通的
提示:2 <= N <= 50.
grid[i][j] 是 [0, ..., N*N - 1] 的排列。
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func swimInWater(grid [][]int) int {
        n := len(grid)
        visited := make([][]bool, n)
        for i := 0; i < n; i++ {
                visited[i] = make([]bool, n)
        }
        visited[0][0] = true
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [3]int{0, 0, grid[0][0]})
        res := 0
        for {
                node := heap.Pop(&intHeap).([3]int)
                a, b, c := node[0], node[1], node[2]
                res = max(res, c) // 更新时间
                // 经过时间t以后，可以瞬间从坐标[0,0]到坐标[N-1, N-1]。
                if a == n-1 && b == n-1 {
                        return res
                }
                for i := 0; i < 4; i++ {
                        x, y := a+dx[i], b+dy[i]
                        if 0 <= x && x < n && 0 <= y && y < n && visited[x][y] ==␣
↪false {
                                visited[x][y] = true
                                heap.Push(&intHeap, [3]int{x, y, grid[x][y]})
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type IntHeap [][3]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][2] < h[j][2]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([3]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func swimInWater(grid [][]int) int {
        n := len(grid)
        return sort.Search(n*n-1, func(target int) bool {
```

(续下页)

(接上页)

```
                if target < grid[0][0] {
                        return false
                }
                queue := make([][2]int, 0)
                queue = append(queue, [2]int{0, 0})
                visited := make([][]bool, n)
                for i := 0; i < n; i++ {
                        visited[i] = make([]bool, n)
                }
                for len(queue) > 0 {
                        a, b := queue[0][0], queue[0][1]
                        queue = queue[1:]
                        if a == n-1 && b == n-1 {
                                return true
                        }
                        for i := 0; i < 4; i++ {
                                x, y := a+dx[i], b+dy[i]
                                if 0 <= x && x < n && 0 <= y && y < n &&
                                        visited[x][y] == false && grid[x][y] <=␣
↪target {
                                        queue = append(queue, [2]int{x, y})
                                        visited[x][y] = true
                                }
                        }
                }
                return false
        })
        return 0
}

# 3
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func swimInWater(grid [][]int) int {
        n := len(grid)
        left, right := 0, n*n-1
        res := 0
        for left <= right {
                mid := left + (right-left)/2 // 二分枚举最大值
                if mid < grid[0][0] {
                        left = mid + 1
                        continue
```

(续下页)

(接上页)

```
                }
                queue := make([][2]int, 0)
                queue = append(queue, [2]int{0, 0})
                visited := make([][]bool, n)
                for i := 0; i < n; i++ {
                        visited[i] = make([]bool, n)
                }
                for len(queue) > 0 {
                        a, b := queue[0][0], queue[0][1]
                        queue = queue[1:]
                        for i := 0; i < 4; i++ {
                                x, y := a+dx[i], b+dy[i]
                                if 0 <= x && x < n && 0 <= y && y < n &&
                                        visited[x][y] == false && grid[x][y] <= mid {
                                        queue = append(queue, [2]int{x, y})
                                        visited[x][y] = true
                                }
                        }
                }
                if visited[n-1][n-1] == true { // 缩小范围
                        res = mid
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res
}

# 4
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func swimInWater(grid [][]int) int {
        n := len(grid)
        arr := make([][2]int, n*n)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        arr[grid[i][j]] = [2]int{i, j} // 高度对应的位置
                }
        }
        fa = Init(n * n)
        for i := 0; i < len(arr); i++ {
```

(续下页)

(接上页)

```
                a, b := arr[i][0], arr[i][1]
                for j := 0; j < 4; j++ {
                        x, y := a+dx[j], b+dy[j]
                        if 0 <= x && x < n && 0 <= y && y < n && grid[x][y] <= i {
                                union(x*n+y, a*n+b)
                        }
                }
                if query(0, n*n-1) {
                        return i
                }
        }
        return 0
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

24.9 780. 到达终点 (2)

- 题目

```
从点(x, y)可以转换到(x, x+y) 或者(x+y, y)。
给定一个起点(sx, sy)和一个终点(tx,␣
↪ty)，如果通过一系列的转换可以从起点到达终点，则返回 True，否则返回False。
示例:输入: sx = 1, sy = 1, tx = 3, ty = 5 输出: True
解释:可以通过以下一系列转换从起点转换到终点:
(1, 1) -> (1, 2)
(1, 2) -> (3, 2)
(3, 2) -> (3, 5)
输入: sx = 1, sy = 1, tx = 2, ty = 2 输出: False
输入: sx = 1, sy = 1, tx = 1, ty = 1 输出: True
注意:sx, sy, tx, ty是范围在[1, 10^9]的整数。
```

- 解题思路

```
func reachingPoints(sx int, sy int, tx int, ty int) bool {
        for tx >= sx && ty >= sy {
                if tx == ty {
                        break
                }
                if tx > ty {
                        // (tx,ty) => (tx-ty,ty)
                        if ty > sy {
                                tx = tx % ty
                        } else {
                                return (tx-sx)%sy == 0
                        }
                } else {
                        if tx > sx {
                                ty = ty % tx
                        } else {
                                return (ty-sy)%sx == 0
                        }
                }
        }
        return tx == sx && ty == sy
}

# 2
func reachingPoints(sx int, sy int, tx int, ty int) bool {
        for tx > sx && ty > sy {
```

(续下页)

(接上页)

```
                if tx > ty {
                        tx = tx % ty
                } else {
                        ty = ty % tx
                }
        }
        if tx == sx { // (x,y) => (x, kx+y)
                return ty >= sy && (ty-sy)%sx == 0
        }
        if ty == sy { // (x,y) => (x+ky,y)
                return tx >= sx && (tx-sx)%sy == 0
        }
        return false
}
```

24.10 786. 第 K 个最小的素数分数 (3)

- 题目

```
给你一个按递增顺序排序的数组 arr 和一个整数 k 。数组 arr 由 1 和若干 素数␣
↪组成，且其中所有整数互不相同。
对于每对满足 0 <= i < j < arr.length 的 i 和 j ，可以得到分数 arr[i] / arr[j] 。
那么第k个最小的分数是多少呢？以长度为 2 的整数数组返回你的答案，这里answer[0] ==␣
↪arr[i]且answer[1] == arr[j] 。
示例 1：输入：arr = [1,2,3,5], k = 3 输出：[2,5]
解释：已构造好的分数,排序后如下所示:
1/5, 1/3, 2/5, 1/2, 3/5, 2/3
很明显第三个最小的分数是 2/5
示例 2：输入：arr = [1,7], k = 1 输出：[1,7]
提示：2 <= arr.length <= 1000
1 <= arr[i] <= 3 * 104
arr[0] == 1
arr[i] 是一个 素数 ，i > 0
arr 中的所有数字 互不相同 ，且按 严格递增 排序
1 <= k <= arr.length * (arr.length - 1) / 2
```

- 解题思路

```
func kthSmallestPrimeFraction(arr []int, k int) []int {
        n := len(arr)
        nums := make([][]int, 0)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := i + 1; j < n; j++ {
                        nums = append(nums, []int{arr[i], arr[j]})
                }
        }
        sort.Slice(nums, func(i, j int) bool {
                // a/b c/d => ad<bc
                a, b, c, d := nums[i][0], nums[i][1], nums[j][0], nums[j][1]
                return a*d < b*c
        })
        return nums[k-1]
}

# 2
func kthSmallestPrimeFraction(arr []int, k int) []int {
        n := len(arr)
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for j := 1; j < n; j++ {
                heap.Push(&intHeap, []int{arr[0], arr[j], 0, j}) // 0/j␣
↪递减：分子最小，分母依次增大
        }
        for i := 1; i <= k-1; i++ { // 取k-1个数(k从1开始)
                node := heap.Pop(&intHeap).([]int)
                x, y := node[2], node[3]
                if x+1 < y { // 下标 x+1 < y
                        heap.Push(&intHeap, []int{arr[x+1], arr[y], x + 1, y})
                }
        }
        return []int{intHeap[0][0], intHeap[0][1]}
}

type IntHeap [][]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][0]*h[j][1] < h[i][1]*h[j][0] } //␣
↪a*d < b*c
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
```

(续下页)

(接上页)

```
        return x
}

# 3
func kthSmallestPrimeFraction(arr []int, k int) []int {
        n := len(arr)
        left, right := 0.0, 1.0
        for {
                mid := left + (right-left)/2
                count := 0
                x, y := 0, 1 // 记录最大的分子/分母
                i := -1
                for j := 1; j < n; j++ {
                        for float64(arr[i+1])/float64(arr[j]) < mid { // 小于目标
                                i++
                                if arr[i]*y > arr[j]*x { // 更新: a/b > c/d => a*d >␣
↪bc 保存a,b
                                        x, y = arr[i], arr[j]
                                }
                        }
                        count = count + (i + 1) // 除以当前arr[j],总计几个数小于mid
                }
                if count == k {
                        return []int{x, y}
                } else if count < k {
                        left = mid
                } else {
                        right = mid
                }
        }
}
```

24.11 793. 阶乘函数后 K 个零 (1)

- 题目

```
f(x)是x!末尾是 0 的数量。 (回想一下x! = 1 * 2 * 3 * ... * x, 且 0! = 1 )
例如, f(3) = 0, 因为 3! = 6 的末尾没有 0 ; 而 f(11) = 2, 因为 11!= 39916800 末端有 2␣
↪个 0 。
给定K, 找出多少个非负整数 x, 能满足 f(x) = K 。
示例 1: 输入: K = 0 输出: 5
解释: 0!, 1!, 2!, 3!, and 4!均符合 K = 0 的条件。
```

(续下页)

(接上页)

```
示例 2: 输入: K = 5 输出: 0
解释: 没有匹配到这样的 x!, 符合 K = 5 的条件。
提示: K 是范围在[0, 10^9]的整数。
```

- 解题思路

```
func preimageSizeFZF(k int) int {
        left, right := 0, 5*k+1
        for left < right {
                mid := left + (right-left)/2
                target := trailingZeroes(mid)
                if target == k {
                        return 5 // 能找到一定会存在连续5个数的阶乘末尾0的个数为k
                } else if target < k {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return 0 // 不存在返回0
}

// leetcode 172.阶乘后的零
// n!末尾0的个数
func trailingZeroes(n int) int {
        result := 0
        for n >= 5 {
                n = n / 5
                result = result + n
        }
        return result
}
```

CHAPTER 25

0801-0900-Easy

25.1 804. 唯一摩尔斯密码词 (1)

- 题目

```
国际摩尔斯密码定义一种标准编码方式，将每个字母对应于一个由一系列点和短线组成的字符串，
比如："a" 对应 ".-", "b" 对应 "-...", "c" 对应 "-.-.", 等等。
为了方便，所有26个英文字母对应摩尔斯密码表如下：
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..",
"--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]
给定一个单词列表，每个单词可以写成每个字母对应摩尔斯密码的组合。
例如，"cab" 可以写成 "-.-..--...", (即 "-.-." + "-..." + ".-"字符串的结合)。
我们将这样一个连接过程称作单词翻译。
返回我们可以获得所有词不同单词翻译的数量。
例如：
输入：words = ["gin", "zen", "gig", "msg"]
输出：2
解释：
各单词翻译如下：
"gin" -> "--...-."
"zen" -> "--...-."
"gig" -> "--...--."
"msg" -> "--...--."

共有 2 种不同翻译，"--...-." 和 "--...--.".
```

(续下页)

(接上页)

```
注意：
    单词列表words 的长度不会超过 100。
    每个单词 words[i]的长度范围为 [1, 12]。
    每个单词 words[i]只包含小写字母。
```

- 解题思路

```
var table = []string{
        ".-", "-...", "-.-.", "-..", ".",
        "..-.", "--.", "....", "..", ".---",
        "-.-", ".-..", "--", "-.", "---",
        ".--.", "--.-", ".-.", "...", "-",
        "..-", "...-", ".--", "-..-", "-.--",
        "--..",
}

func uniqueMorseRepresentations(words []string) int {
        res := make(map[string]bool)
        for _, w := range words {
                b := ""
                for i := 0; i < len(w); i++ {
                        b = b + table[w[i]-'a']
                }
                res[b] = true
        }
        return len(res)
}
```

25.2 806. 写字符串需要的行数 (1)

- 题目

```
我们要把给定的字符串 S 从左到右写到每一行上，每一行的最大宽度为100个单位，
如果我们在写某个字母的时候会使这行超过了100 个单位，那么我们应该把这个字母写到下一行。
我们给定了一个数组 widths ，这个数组 widths[0] 代表 'a' 需要的单位，
widths[1] 代表 'b' 需要的单位，...， widths[25] 代表 'z' 需要的单位。
现在回答两个问题：至少多少行能放下S，以及最后一行使用的宽度是多少个单位？
将你的答案作为长度为2的整数列表返回。

示例 1:输入:
widths = [10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
↪10]
```

(续下页)

(接上页)

```
S = "abcdefghijklmnopqrstuvwxyz"
输出: [3, 60]
解释:
所有的字符拥有相同的占用单位10。所以书写所有的26个字母,
我们需要2个整行和占用60个单位的一行。

示例 2:输入:
widths = [4,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
↪10]
S = "bbbcccdddaaa"
输出: [2, 4]
解释:
除去字母'a'所有的字符都是相同的单位10, 并且字符串 "bbbcccdddaa" 将会覆盖 9 * 10 + 2 *␣
↪4 = 98 个单位.
最后一个字母 'a' 将会被写到第二行, 因为第一行只剩下2个单位了。
所以, 这个答案是2行, 第二行有4个单位宽度。
注:
    字符串 S 的长度在 [1, 1000] 的范围。
    S 只包含小写字母。
    widths 是长度为 26的数组。
    widths[i] 值的范围在 [2, 10]。
```

- 解题思路

```
func numberOfLines(widths []int, S string) []int {
        res := []int{0, 0}
        if len(S) == 0 {
                return res
        }
        res[0] = 1

        for i := 0; i < len(S); i++ {
                if res[1]+widths[S[i]-'a'] > 100 {
                        res[0]++
                        res[1] = widths[S[i]-'a']
                } else {
                        res[1] = res[1] + widths[S[i]-'a']
                }
        }
        return res
}
```

25.3 811. 子域名访问计数 (2)

- 题目

```
一个网站域名，如"discuss.leetcode.com"，包含了多个子域名。
作为顶级域名，常用的有"com"，下一级则有"leetcode.com"，最低的一级为"discuss.leetcode.
↪com"。
当我们访问域名"discuss.leetcode.com"时，也同时访问了其父域名"leetcode.com
↪"以及顶级域名 "com"。
给定一个带访问次数和域名的组合，要求分别计算每个域名被访问的次数。
其格式为访问次数+空格+地址，例如："9001 discuss.leetcode.com"。
接下来会给出一组访问次数和域名组合的列表cpdomains 。
要求解析出所有域名的访问次数，输出格式和输入格式相同，不限定先后顺序。

示例 1:输入：["9001 discuss.leetcode.com"]
输出： ["9001 discuss.leetcode.com", "9001 leetcode.com", "9001 com"]
说明：
例子中仅包含一个网站域名："discuss.leetcode.com"。
按照前文假设，子域名"leetcode.com"和"com"都会被访问，所以它们都被访问了9001次。

示例 2输入：
["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]
输出：
["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org","5 org",
"1 intel.mail.com","951 com"]
说明：
按照假设，会访问"google.mail.com" 900次，"yahoo.com" 50次，
"intel.mail.com" 1次，"wiki.org" 5次。
而对于父域名，会访问"mail.com" 900+1 = 901次，"com" 900 + 50 + 1 = 951次，和 "org" 5␣
↪次。

注意事项：
    cpdomains 的长度小于 100。
    每个域名的长度小于100。
    每个域名地址包含一个或两个"."符号。
    输入中任意一个域名的访问次数都小于10000。
```

- 解题思路

```
func subdomainVisits(cpdomains []string) []string {
        m := make(map[string]int)
        for _, domains := range cpdomains {
                domain, count := parse(domains)
                isNew := true
```

(续下页)

(接上页)

```
                for isNew {
                        m[domain] = m[domain] + count
                        domain, isNew = cut(domain)
                }
        }
        return getResult(m)
}

func parse(s string) (string, int) {
        ss := strings.Split(s, " ")
        count, _ := strconv.Atoi(ss[0])
        return ss[1], count
}

func cut(s string) (string, bool) {
        index := strings.Index(s, ".")
        if index == -1 {
                return "", false
        }
        return s[index+1:], true
}

func getResult(m map[string]int) []string {
        res := make([]string, 0, len(m))
        for k, v := range m {
                res = append(res, fmt.Sprintf("%d %s", v, k))
        }
        return res
}

#
func subdomainVisits(cpdomains []string) []string {
        m := make(map[string]int)
        for _, domains := range cpdomains {
                arr := strings.Split(domains, " ")
                count, _ := strconv.Atoi(arr[0])
                tempArr := getSubdomains(arr[1])
                for i := 0; i < len(tempArr); i++ {
                        m[tempArr[i]] += count
                }
        }
        res := make([]string, 0)
        for k, v := range m {
```

(续下页)

(接上页)

```
                res = append(res, strconv.Itoa(v)+" "+k)
        }
        return res
}

func getSubdomains(s string) []string {
        res := make([]string, 0)
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '.' {
                        res = append(res, s[i+1:])
                }
        }
        res = append(res, s)
        return res
}
```

25.4 812. 最大三角形面积 (2)

- 题目

```
给定包含多个点的集合，从其中取三个点组成三角形，返回能组成的最大三角形的面积。
示例:输入: points = [[0,0],[0,1],[1,0],[0,2],[2,0]] 输出: 2
解释: 这五个点如下图所示。组成的橙色三角形是最大的，面积为2。
注意:
    3 <= points.length <= 50.
    不存在重复的点。
     -50 <= points[i][j] <= 50.
    结果误差值在 10^-6 以内都认为是正确答案。
```

- 解题思路

```
func largestTriangleArea(points [][]int) float64 {
        maxArea := 0.0
        n := len(points)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        for k := j + 1; k < n; k++ {
                                if area(points[i], points[j], points[k]) > maxArea {
                                        maxArea = area(points[i], points[j],␣
↪points[k])
                                }
                        }
```

(续下页)

(接上页)

```
                }
        }
        return maxArea
}

// 三角形面积=|(x1 * y2 + x2 * y3 + x3 * y1 - y1 * x2 - y2 * x3 - y3 * x1)|/2
func area(p1, p2, p3 []int) float64 {
        return abs(
                p1[0]*p2[1]+p2[0]*p3[1]+p3[0]*p1[1]-
                        p1[0]*p3[1]-p2[0]*p1[1]-p3[0]*p2[1]) / 2
}

func abs(num int) float64 {
        if num < 0 {
                num = -num
        }
        return float64(num)
}

#
func largestTriangleArea(points [][]int) float64 {
        maxArea := 0.0
        n := len(points)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        for k := j + 1; k < n; k++ {
                                // p = (a+b+c)/2
                                // s = p*(p-a)*(p-b)*(p-c)
                                a := length(points[i], points[j])
                                b := length(points[i], points[k])
                                c := length(points[j], points[k])
                                p := (a + b + c) / 2
                                area := math.Sqrt(p * (p - a) * (p - b) * (p - c))
                                if area > maxArea {
                                        maxArea = area
                                }
                        }
                }
        }
        return maxArea
}

// 求两点距离
```

(续下页)

(接上页)

```
func length(p1, p2 []int) float64 {
        l := (p1[0]-p2[0])*(p1[0]-p2[0]) + (p1[1]-p2[1])*(p1[1]-p2[1])
        return math.Sqrt(float64(l))
}
```

25.5 819. 最常见的单词 (2)

- 题目

```
给定一个段落 (paragraph) 和一个禁用单词列表
↪(banned)。返回出现次数最多，同时不在禁用列表中的单词。
题目保证至少有一个词不在禁用列表中，而且答案唯一。
禁用列表中的单词用小写字母表示，不含标点符号。段落中的单词不区分大小写。答案都是小写字母。

示例：输入：
paragraph = "Bob hit a ball, the hit BALL flew far after it was hit."
banned = ["hit"]
输出："ball"
解释：
"hit" 出现了3次，但它是一个禁用的单词。
"ball" 出现了2次
↪(同时没有其他单词出现2次)，所以它是段落里出现次数最多的，且不在禁用列表中的单词。
注意，所有这些单词在段落里不区分大小写，标点符号需要忽略（即使是紧挨着单词也忽略，
↪比如 "ball,"），
"hit"不是最终的答案，虽然它出现次数更多，但它在禁用单词列表中。
提示：
    1 <= 段落长度 <= 1000
    0 <= 禁用单词个数 <= 100
    1 <= 禁用单词长度 <= 10
    答案是唯一的，且都是小写字母（即使在 paragraph
↪里是大写的，即使是一些特定的名词，答案都是小写的。）
    paragraph 只包含字母、空格和下列标点符号!?',;.
    不存在没有连字符或者带有连字符的单词。
    单词里只包含字母，不会出现省略号或者其他标点符号。
```

- 解题思路

```
func mostCommonWord(paragraph string, banned []string) string {
        isBanned := make(map[string]bool)
        for _, b := range banned {
                isBanned[b] = true
        }
```

(续下页)

(接上页)

```
        chars := []string{"!", "?", ",", "'", ";", "."}
        for _, c := range chars {
                paragraph = strings.Replace(paragraph, c, " ", -1)
        }
        p := strings.ToLower(paragraph)
        ss := strings.Fields(p)
        count := make(map[string]int)
        for _, v := range ss {
                if isBanned[v] {
                        continue
                }
                count[v]++
        }
        res := ""
        max := 0
        for s, c := range count {
                if max < c {
                        max = c
                        res = s
                }
        }
        return res
}

#
func mostCommonWord(paragraph string, banned []string) string {
        isBanned := make(map[string]bool)
        for _, b := range banned {
                isBanned[b] = true
        }
        count := make(map[string]int)
        length := len(paragraph)
        for i := 0; i < length; i++ {
                for i < length && !isChar(paragraph[i]) {
                        i++
                }
                j := i
                temp := ""
                for ; j < length; j++ {
                        if !isChar(paragraph[j]) {
                                break
                        }
                        if paragraph[j] >= 'A' && paragraph[j] <= 'Z' {
```

(续下页)

(接上页)

```
                                temp = temp + string(paragraph[j]-'A'+'a')
                        } else {
                                temp = temp + string(paragraph[j])
                        }
                }
                i = j
                if isBanned[temp] {
                        continue
                }
                count[temp]++
        }
        res := ""
        max := 0
        for s, c := range count {
                if max < c {
                        max = c
                        res = s
                }
        }
        return res
}

func isChar(b byte) bool {
        if (b >= 'a' && b <= 'z') || (b >= 'A' && b <= 'Z') {
                return true
        }
        return false
}
```

25.6 821. 字符的最短距离 (3)

- 题目

```
给定一个字符串 S 和一个字符 C。返回一个代表字符串 S 中每个字符到字符串 S 中的字符 C␣
↪的最短距离的数组。
示例 1:输入: S = "loveleetcode", C = 'e'
输出: [3, 2, 1, 0, 1, 0, 0, 1, 2, 2, 1, 0]
说明:
    字符串 S 的长度范围为 [1, 10000]。
    C 是一个单字符，且保证是字符串 S 里的字符。
    S 和 C 中的所有字母均为小写字母。
```

- 解题思路

```
func shortestToChar(S string, C byte) []int {
        n := len(S)
        res := make([]int, n)
        for i := range res {
                res[i] = 100001
        }

        left, right := -n, 2*n
        for i := 0; i < n; i++ {
                j := n - i - 1
                if S[i] == C {
                        left = i
                }
                if S[j] == C {
                        right = j
                }
                // i从0->n-1 跟左边的C比较得到最近的距离
                // j从n-1->0 跟右边的C比较得到最近的距离
                res[i] = min(res[i], dist(i, left))
                res[j] = min(res[j], dist(j, right))
        }
        return res
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

func dist(i, j int) int {
        if i > j {
                return i - j
        }
        return j - i
}

#
func shortestToChar(S string, C byte) []int {
        n := len(S)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                if S[i] == C {
```

(续下页)

(接上页)

```
                        res[i] = 0
                        continue
                }
                min := n
                for j := i + 1; j < n; j++ {
                        if S[j] == C {
                                if min > j-i {
                                        min = j - i
                                }
                                break
                        }
                }
                for j := i - 1; j >= 0; j-- {
                        if S[j] == C {
                                if min > i-j {
                                        min = i - j
                                }
                                break
                        }
                }
                res[i] = min
        }
        return res
}

#
func shortestToChar(S string, C byte) []int {
        n := len(S)
        res := make([]int, n)
        arr := make([]int, 0)
        for i := 0; i < len(S); i++ {
                if S[i] == C {
                        arr = append(arr, i)
                }
        }
        for i := 0; i < n; i++ {
                min := n
                for _, value := range arr {
                        if value == i {
                                min = 0
                                break
                        }
                        if min > dist(i, value) {
```

(续下页)

(接上页)

```
                                min = dist(i, value)
                        }
                }
                res[i] = min
        }
        return res
}

func dist(i, j int) int {
        if i > j {
                return i - j
        }
        return j - i
}
```

25.7 824. 山羊拉丁文 (2)

- 题目

```
给定一个由空格分割单词的句子 S。每个单词只包含大写或小写字母。
我们要将句子转换为 “Goat Latin”（一种类似于 猪拉丁文 - Pig Latin 的虚构语言）。
山羊拉丁文的规则如下：
    如果单词以元音开头（a, e, i, o, u），在单词后添加"ma"。
    例如，单词"apple"变为"applema"。
    如果单词以辅音字母开头（即非元音字母），移除第一个字符并将它放到末尾，之后再添加
↪"ma"。
    例如，单词"goat"变为"oatgma"。
    根据单词在句子中的索引，在单词最后添加与索引相同数量的字母'a'，索引从1开始。
    例如，在第一个单词后添加"a"，在第二个单词后添加"aa"，以此类推。
返回将 S 转换为山羊拉丁文后的句子。

示例 1:输入："I speak Goat Latin"
输出："Imaa peaksmaaa oatGmaaaa atinLmaaaaa"
示例 2:输入："The quick brown fox jumped over the lazy dog"
输出："heTmaa uickqmaaa rownbmaaaa oxfmaaaaa umpedjmaaaaaa overmaaaaaaa hetmaaaaaaaa
azylmaaaaaaaaa ogdmaaaaaaaaaa"
说明：
    S 中仅包含大小写字母和空格。单词间有且仅有一个空格。
    1 <= S.length <= 150。
```

- 解题思路

```
func toGoatLatin(S string) string {
        ss := strings.Split(S, " ")
        for i := range ss {
                ss[i] = handle(ss[i], i)
        }
        return strings.Join(ss, " ")
}

func handle(s string, i int) string {
        postfix := "ma" + strings.Repeat("a", i+1)
        if isBeginWithVowel(s) {
                return s + postfix
        }
        return s[1:] + s[0:1] + postfix
}

func isBeginWithVowel(s string) bool {
        switch s[0] {
        case 'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U':
                return true
        default:
                return false
        }
}

#
func toGoatLatin(S string) string {
        res := ""
        begin := 1
        count := 1
        temp := ""
        for i := 0; i < len(S); i++ {
                if S[i] == ' ' {
                        res = res + temp + strings.Repeat("a", count) + " "
                        count++
                        begin = 1
                } else {
                        if begin == 1 {
                                begin = 0
                                if isBeginWithVowel(S[i]) {
                                        res = res + string(S[i])
                                        temp = "ma"
                                } else {
                                        temp = string(S[i]) + "ma"
```

(续下页)

(接上页)

```
                                }
                        } else {
                                res = res + string(S[i])
                        }
                }
        }
        return res + temp + strings.Repeat("a", count)
}

func isBeginWithVowel(b byte) bool {
        switch b {
        case 'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U':
                return true
        default:
                return false
        }
}
```

25.8 830. 较大分组的位置 (2)

- 题目

```
在一个由小写字母构成的字符串 S 中，包含由一些连续的相同字符所构成的分组。
例如，在字符串 S = "abbxxxxzyy" 中，就含有 "a", "bb", "xxxx", "z" 和 "yy"␣
↪这样的一些分组。
我们称所有包含大于或等于三个连续字符的分组为较大分组。找到每一个较大分组的起始和终止位置。
最终结果按照字典顺序输出。
示例 1:输入: "abbxxxxzzy"输出: [[3,6]]
解释: "xxxx" 是一个起始于 3 且终止于 6 的较大分组。
示例 2:输入: "abc"输出: []
解释: "a","b" 和 "c" 均不是符合要求的较大分组。
示例 3:输入: "abcdddeeeeaabbbcd"输出: [[3,5],[6,9],[12,14]]
说明:  1 <= S.length <= 1000
```

- 解题思路

```
func largeGroupPositions(S string) [][]int {
        res := make([][]int, 0, len(S)/3)
        left := 0
        for right := 0; right < len(S); right++ {
                if right == len(S)-1 || S[right] != S[right+1] {
                        if right-left+1 >= 3 {
```

(续下页)

(接上页)

```
                                res = append(res, []int{left, right})
                        }
                        left = right + 1
                }
        }
        return res
}

#
func largeGroupPositions(S string) [][]int {
        res := make([][]int, 0, len(S)/3)
        left, right := 0, 1
        for ; right < len(S); right++ {
                if S[left] != S[right] {
                        left = right
                        continue
                }
                if right-left+1 == 3 {
                        res = append(res, []int{left, right})
                } else if right-left+1 > 3 {
                        res[len(res)-1][1] = right
                }
        }
        return res
}
```

25.9 832. 翻转图像 (2)

- 题目

```
给定一个二进制矩阵 A，我们想先水平翻转图像，然后反转图像并返回结果。
水平翻转图片就是将图片的每一行都进行翻转，即逆序。例如，水平翻转 [1, 1, 0] 的结果是␣
↪[0, 1, 1]。
反转图片的意思是图片中的 0 全部被 1 替换， 1 全部被 0 替换。例如，反转 [0, 1, 1]␣
↪的结果是 [1, 0, 0]。

示例 1:
输入: [[1,1,0],[1,0,1],[0,0,0]]
输出: [[1,0,0],[0,1,0],[1,1,1]]
解释: 首先翻转每一行: [[0,1,1],[1,0,1],[0,0,0]];
     然后反转图片: [[1,0,0],[0,1,0],[1,1,1]]
```

(续下页)

(接上页)

```
示例 2:
输入: [[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]
输出: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]
解释: 首先翻转每一行: [[0,0,1,1],[1,0,0,1],[1,1,1,0],[0,1,0,1]];
     然后反转图片: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

说明:
    1 <= A.length = A[0].length <= 20
    0 <= A[i][j] <= 1
```

- 解题思路

```
func flipAndInvertImage(A [][]int) [][]int {
        for k := 0; k < len(A); k++ {
                i, j := 0, len(A[k])-1
                for i < j {
                        A[k][i], A[k][j] = invert(A[k][j]), invert(A[k][i])
                        i++
                        j--
                }
                if i == j {
                        A[k][i] = invert(A[k][i])
                }
        }
        return A
}

func invert(i int) int {
        if i == 0 {
                return 1
        }
        return 0
}

#
func flipAndInvertImage(A [][]int) [][]int {
        for k := 0; k < len(A); k++ {
                i, j := 0, len(A[k])-1
                for i < j {
                        A[k][i], A[k][j] = A[k][j], A[k][i]
                        i++
                        j--
                }
                for i := 0; i < len(A[k]); i++ {
```

(续下页)

(接上页)

```
                        A[k][i] = A[k][i] ^ 1
                }
        }
        return A
}
```

25.10 836. 矩形重叠 (3)

- 题目

```
矩形以列表 [x1, y1, x2, y2] 的形式表示，其中 (x1, y1) 为左下角的坐标，(x2, y2)␣
↪是右上角的坐标。
如果相交的面积为正，则称两矩形重叠。需要明确的是，只在角或边接触的两个矩形不构成重叠。
给出两个矩形，判断它们是否重叠并返回结果。
示例 1：输入：rec1 = [0,0,2,2], rec2 = [1,1,3,3] 输出：true
示例 2：输入：rec1 = [0,0,1,1], rec2 = [1,0,2,1] 输出：false
提示：
    两个矩形 rec1 和 rec2 都以含有四个整数的列表的形式给出。
    矩形中的所有坐标都处于 -10^9 和 10^9 之间。
    x 轴默认指向右，y 轴默认指向上。
    你可以仅考虑矩形是正放的情况。
```

- 解题思路

```
func isRectangleOverlap(rec1 []int, rec2 []int) bool {
        // 满足条件
        if rec1[1] < rec2[3] && rec1[0] < rec2[2] &&
                rec2[1] < rec1[3] && rec2[0] < rec1[2] {
                return true
        }
        return false
}

#
func isRectangleOverlap(rec1 []int, rec2 []int) bool {
        // 不满足条件，rec2固定，rec1在rec2的方位
        // 左侧：rec1[2] <= rec2[0]
        // 右侧：rec1[0] >= rec2[2]
        // 上方：rec1[1] >= rec2[3]
        // 下方：rec1[3] <= rec2[1]
        if rec1[2] <= rec2[0] ||
                rec1[3] <= rec2[1] ||
```

(续下页)

(接上页)

```
                rec1[0] >= rec2[2] ||
                rec1[1] >= rec2[3] {
                return false
        }
        return true
}

#
func isRectangleOverlap(rec1 []int, rec2 []int) bool {
        return min(rec1[2], rec2[2]) > max(rec1[0], rec2[0]) &&
                min(rec1[3], rec2[3]) > max(rec1[1], rec2[1])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

25.11 840. 矩阵中的幻方 (2)

- 题目

```
3 x 3 的幻方是一个填充有从 1 到 9 的不同数字的 3 x 3 矩阵，
其中每行，每列以及两条对角线上的各数之和都相等。
给定一个由整数组成的 grid，其中有多少个 3 × 3 的 “幻方” ␣
↪子矩阵？（每个子矩阵都是连续的）。

示例：输入：[[4,3,8,4],
      [9,5,1,9],
      [2,7,6,2]]
输出：1
解释：
下面的子矩阵是一个 3 x 3 的幻方：
```

(续下页)

(接上页)

```
438
951
276
而这一个不是:
384
519
762
总的来说，在本示例所给定的矩阵中只有一个 3 x 3 的幻方子矩阵。
提示:
    1 <= grid.length <= 10
    1 <= grid[0].length <= 10
    0 <= grid[i][j] <= 15
```

- 解题思路

```
func numMagicSquaresInside(grid [][]int) int {
        m, n := len(grid), len(grid[0])
        res := 0
        for i := 0; i+2 < m; i++ {
                for j := 0; j+2 < n; j++ {
                        if grid[i+1][j+1] != 5{
                                continue
                        }
                        if !available(i, j, grid) {
                                continue
                        }
                        if grid[i][j]+grid[i][j+1]+grid[i][j+2] == 15 &&
                                grid[i+1][j]+grid[i+1][j+1]+grid[i+1][j+2] == 15 &&
                                grid[i+2][j]+grid[i+2][j+1]+grid[i+2][j+2] == 15 &&
                                grid[i][j]+grid[i+1][j]+grid[i+2][j] == 15 &&
                                grid[i][j+1]+grid[i+1][j+1]+grid[i+2][j+1] == 15 &&
                                grid[i][j+2]+grid[i+1][j+2]+grid[i+2][j+2] == 15 &&
                                grid[i][j]+grid[i+1][j+1]+grid[i+2][j+2] == 15 &&
                                grid[i][j+2]+grid[i+1][j+1]+grid[i+2][j] == 15 {
                                res++
                        }
                }
        }
        return res
}

func available(x, y int, g [][]int) bool {
        tmp := [16]int{}
        for i := x; i <= x+2; i++ {
```

(续下页)

(接上页)

```
                for j := y; j <= y+2; j++ {
                        tmp[g[i][j]]++
                }
        }

        for i := 1; i <= 9; i++ {
                if tmp[i] != 1 {
                        return false
                }
        }
        return true
}

#
func numMagicSquaresInside(grid [][]int) int {
        m, n := len(grid), len(grid[0])
        res := 0
        for i := 0; i+2 < m; i++ {
                for j := 0; j+2 < n; j++ {
                        if grid[i+1][j+1] != 5 {
                                continue
                        }
                        if available(i, j, grid) {
                                res++
                        }
                }
        }
        return res
}

var m = map[string]bool{
        "816357492": true,
        "834159672": true,
        "618753294": true,
        "672159834": true,
        "492357816": true,
        "438951276": true,
        "294753618": true,
        "276951438": true,
}

func available(x, y int, g [][]int) bool {
        str := ""
```

(续下页)

(接上页)

```
        for i := x; i <= x+2; i++ {
                for j := y; j <= y+2; j++ {
                        str = str + strconv.Itoa(g[i][j])
                }
        }
        if m[str] {
                return true
        }
        return false
}
```

25.12 844. 比较含退格的字符串 (2)

- 题目

```
给定 S 和 T␣
↪两个字符串，当它们分别被输入到空白的文本编辑器后，判断二者是否相等，并返回结果。
# 代表退格字符。
注意：如果对空文本输入退格字符，文本继续为空。
示例 1：输入：S = "ab#c", T = "ad#c"输出：true
解释：S 和 T 都会变成 “ac”。
示例 2：输入：S = "ab##", T = "c#d#"输出：true
解释：S 和 T 都会变成 “”。
示例 3：输入：S = "a##c", T = "#a#c"输出：true
解释：S 和 T 都会变成 “c”。
示例 4：输入：S = "a#c", T = "b" 输出：false
解释：S 会变成 “c”，但 T 仍然是 “b”。
提示：

    1 <= S.length <= 200
    1 <= T.length <= 200
    S 和 T 只含有小写字母以及字符 '#'。
进阶：
    你可以用 O(N) 的时间复杂度和 O(1) 的空间复杂度解决该问题吗？
```

- 解题思路

```
func backspaceCompare(S string, T string) bool {
        return check(S) == check(T)
}

func check(str string) string {
```

(续下页)

(接上页)

```
        res := make([]string, 0)
        for _, v := range str {
                if string(v) == "#" {
                        if len(res) != 0 {
                                res = res[:len(res)-1]
                        }
                } else {
                        res = append(res, string(v))
                }
        }
        return strings.Join(res, "")
}

#
func backspaceCompare(S string, T string) bool {
        return check(S) == check(T)
}

func check(S string) string {
        str := ""
        count := 0
        for i := len(S) - 1; i >= 0; i-- {
                if S[i] == '#' {
                        count++
                } else {
                        if count != 0 {
                                count--
                                continue
                        }
                        str = string(S[i]) + str
                }
        }
        return str
}
```

25.13 849. 到最近的人的最大距离 (4)

- 题目

```
在一排座位（seats）中，1 代表有人坐在座位上，0 代表座位上是空的。
至少有一个空座位，且至少有一人坐在座位上。
亚历克斯希望坐在一个能够使他与离他最近的人之间的距离达到最大化的座位上。
返回他到离他最近的人的最大距离。
示例 1：输入：[1,0,0,0,1,0,1]输出：2
解释：如果亚历克斯坐在第二个空位（seats[2]）上，他到离他最近的人的距离为 2 。
如果亚历克斯坐在其它任何一个空位上，他到离他最近的人的距离为 1 。
因此，他到离他最近的人的最大距离是 2 。
示例 2：输入：[1,0,0,0]输出：3
解释：  如果亚历克斯坐在最后一个座位上，他离最近的人有 3 个座位远。
这是可能的最大距离，所以答案是 3 。
提示：
    1 <= seats.length <= 20000
    seats 中只含有 0 和 1，至少有一个 0，且至少有一个 1。
```

- 解题思路

```
func maxDistToClosest(seats []int) int {
        n := len(seats)
        left := make([]int, n)
        right := make([]int, n)
        for i := 0; i < n; i++ {
                left[i], right[i] = n, n
        }
        for i := 0; i < n; i++ {
                if seats[i] == 1 {
                        left[i] = 0
                } else if seats[i] != 1 && i > 0 {
                        left[i] = left[i-1] + 1
                }
        }
        for i := n - 1; i >= 0; i-- {
                if seats[i] == 1 {
                        right[i] = 0
                } else if seats[i] != 1 && i < n-1 {
                        right[i] = right[i+1] + 1
                }
        }
        res := 0
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                if seats[i] == 0 {
                        if min(left[i], right[i]) > res {
                                res = min(left[i], right[i])
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func maxDistToClosest(seats []int) int {
        n := len(seats)
        res := 0
        left := -1
        right := 0
        for i := 0; i < n; i++ {
                if seats[i] == 1 {
                        left = i
                } else {
                        // 找到右边有人的位置
                        for (right < n && seats[right] == 0) || right < i {
                                right++
                        }
                        leftLen := 0
                        rightLen := 0
                        if left == -1 {
                                leftLen = n
                        } else {
                                leftLen = i - left
                        }
                        if right == n {
                                rightLen = n
                        } else {
                                rightLen = right - i
                        }
                        if min(leftLen, rightLen) > res {
```

(续下页)

(接上页)

```
                                res = min(leftLen, rightLen)
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func maxDistToClosest(seats []int) int {
        n := len(seats)
        var arr []int
        for i := 0; i < n; i++ {
                if seats[i] == 1 {
                        arr = append(arr, i)
                }
        }
        if len(arr) == 0 {
                return 0
        }
        max := -1
        for i := 0; i < n-1; i++ {
                if arr[i+1]-arr[i] > max {
                        max = arr[i+1] - arr[i]
                }
        }
        max = max / 2
        // 判断首尾
        if arr[0] > max {
                max = arr[0]
        }
        if n-arr[len(arr)-1]-1 > max {
                max = n - arr[len(arr)-1] - 1
        }
        return max
}
```

(续下页)

(接上页)

```
#
func maxDistToClosest(seats []int) int {
        res := 0
        count := 0
        for i := 0; i < len(seats); i++ {
                if count == i {
                        res = count
                } else {
                        res = max(res, (count+count%2)/2)
                }
                if seats[i] == 1 {
                        count = 0
                } else {
                        count++
                }
        }
        return max(res, count)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

25.14 852. 山脉数组的峰顶索引 (3)

- 题目

```
我们把符合下列属性的数组 A 称作山脉：
    A.length >= 3
    存在 0 < i < A.length - 1 使得
    A[0] < A[1] < ... A[i-1] < A[i] > A[i+1] > ... > A[A.length - 1]
给定一个确定为山脉的数组，返回任何满足
A[0] < A[1] < ... A[i-1] < A[i] > A[i+1] > ... > A[A.length - 1] 的 i 的值。
示例 1：输入：[0,1,0]输出：1
示例 2：输入：[0,2,1,0]输出：1
提示：
    3 <= A.length <= 10000
    0 <= A[i] <= 10^6
    A 是如上定义的山脉
```

- 解题思路

```
func peakIndexInMountainArray(A []int) int {
        n := len(A)
        for i := 0; i < n-1; i++ {
                if A[i] > A[i+1] {
                        return i
                }
        }
        return 0
}

#
func peakIndexInMountainArray(A []int) int {
        left, right := 0, len(A)-1
        for {
                mid := left + (right-left)/2
                if A[mid] > A[mid+1] && A[mid] > A[mid-1] {
                        return mid
                }
                if A[mid] > A[mid-1] {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
}

# 3
func peakIndexInMountainArray(arr []int) int {
        n := len(arr)
        return sort.Search(n-1, func(i int) bool {
                return arr[i] > arr[i+1]
        })
}
```

25.15 859. 亲密字符串 (2)

- 题目

```
给定两个由小写字母构成的字符串 A 和 B ，只要我们可以通过交换 A 中的两个字母得到与 B
↪相等的结果，
就返回 true ；否则返回 false 。
```

(续下页)

(接上页)

```
示例 1: 输入: A = "ab", B = "ba" 输出: true
示例 2: 输入: A = "ab", B = "ab"输出: false
示例 3: 输入: A = "aa", B = "aa" 输出: true
示例 4: 输入: A = "aaaaaaabc", B = "aaaaaaacb" 输出: true
示例 5: 输入: A = "", B = "aa" 输出: false
提示:
    0 <= A.length <= 20000
    0 <= B.length <= 20000
    A 和 B 仅由小写字母构成。
```

- 解题思路

```
func buddyStrings(A string, B string) bool {
        if len(A) != len(B) {
                return false
        }
        if A == B {
                return hasDouble(A)
        }
        count := 2
        strA, strB := "", ""
        i := 0
        for ; count > 0 && i < len(A); i++ {
                if A[i] != B[i] {
                        strA = string(A[i]) + strA
                        strB = strB + string(B[i])
                        count--
                }
        }
        return count == 0 && strA == strB && A[i:] == B[i:]
}

func hasDouble(s string) bool {
        seen := [26]bool{}
        for i := range s {
                b := s[i] - 'a'
                if seen[b] {
                        return true
                }
                seen[b] = true
        }
        return false
}
```

(续下页)

(接上页)

```
#
func buddyStrings(A string, B string) bool {
        if len(A) != len(B) {
                return false
        }
        if A == B {
                return hasDouble(A)
        }
        first := -1
        second := -1
        for i := 0; i < len(A); i++ {
                if A[i] != B[i] {
                        if first == -1 {
                                first = i
                        } else if second == -1 {
                                second = i
                        } else {
                                return false
                        }
                }
        }
        return A[first] == B[second] && A[second] == B[first]
}

func hasDouble(s string) bool {
        seen := [26]int{}
        for i := range s {
                b := s[i] - 'a'
                if seen[b] >= 1 {
                        return true
                }
                seen[b] = 1
        }
        return false
}
```

25.16 860. 柠檬水找零 (1)

- 题目

```
在柠檬水摊上，每一杯柠檬水的售价为 5 美元。
顾客排队购买你的产品，（按账单 bills 支付的顺序）一次购买一杯。
每位顾客只买一杯柠檬水，然后向你付 5 美元、10 美元或 20 美元。
你必须给每个顾客正确找零，也就是说净交易是每位顾客向你支付 5 美元。
注意，一开始你手头没有任何零钱。
如果你能给每位顾客正确找零，返回 true ，否则返回 false 。

示例 1：输入：[5,5,5,10,20] 输出：true
解释：
前 3 位顾客那里，我们按顺序收取 3 张 5 美元的钞票。
第 4 位顾客那里，我们收取一张 10 美元的钞票，并返还 5 美元。
第 5 位顾客那里，我们找还一张 10 美元的钞票和一张 5 美元的钞票。
由于所有客户都得到了正确的找零，所以我们输出 true。
示例 2：输入：[5,5,10] 输出：true
示例 3：输入：[10,10] 输出：false
示例 4：输入：[5,5,10,10,20] 输出：false
解释：
前 2 位顾客那里，我们按顺序收取 2 张 5 美元的钞票。
对于接下来的 2 位顾客，我们收取一张 10 美元的钞票，然后返还 5 美元。
对于最后一位顾客，我们无法退回 15 美元，因为我们现在只有两张 10 美元的钞票。
由于不是每位顾客都得到了正确的找零，所以答案是 false。
提示：
    0 <= bills.length <= 10000
    bills[i] 不是 5 就是 10 或是 20
```

- 解题思路

```
func lemonadeChange(bills []int) bool {
        fives, tens := 0, 0
        for _, b := range bills {
                switch b {
                case 5:
                        fives++
                case 10:
                        fives--
                        tens++
                case 20:
                        if tens > 0 {
                                tens--
                                fives--
```

(续下页)

(接上页)

```
                        } else {
                                fives = fives - 3
                        }
                }
                if fives < 0 || tens < 0 {
                        return false
                }
        }
        return true
}
```

25.17 867. 转置矩阵 (1)

- 题目

```
给定一个矩阵 A， 返回 A 的转置矩阵。
矩阵的转置是指将矩阵的主对角线翻转，交换矩阵的行索引与列索引。
示例 1: 输入: [[1,2,3],[4,5,6],[7,8,9]]输出: [[1,4,7],[2,5,8],[3,6,9]]
示例 2: 输入: [[1,2,3],[4,5,6]]输出: [[1,4],[2,5],[3,6]]
提示:
    1 <= A.length <= 1000
    1 <= A[0].length <= 1000
```

- 解题思路

```
func transpose(A [][]int) [][]int {
        m, n := len(A), len(A[0])
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                res[i] = make([]int, m)
        }
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        res[j][i] = A[i][j]
                }
        }
        return res
}
```

25.18 868. 二进制间距 (3)

- 题目

```
给定一个正整数 N，找到并返回 N 的二进制表示中两个连续的 1 之间的最长距离。
如果没有两个连续的 1，返回 0 。
示例 1：输入：22 输出：2
解释：22 的二进制是 0b10110 。
在 22 的二进制表示中，有三个 1，组成两对连续的 1 。
第一对连续的 1 中，两个 1 之间的距离为 2 。
第二对连续的 1 中，两个 1 之间的距离为 1 。
答案取两个距离之中最大的，也就是 2 。
示例 2：输入：5 输出：2
解释：5 的二进制是 0b101 。
示例 3：输入：6 输出：1
解释：  6 的二进制是 0b110 。
示例 4：输入：8 输出：0
解释：  8 的二进制是 0b1000 。
在 8 的二进制表示中没有连续的 1，所以返回 0 。
提示：1 <= N <= 10^9
```

- 解题思路

```
func binaryGap(N int) int {
        arr := make([]int, 0)
        index := 0
        for N > 0 {
                if N%2 == 1 {
                        arr = append(arr, index)
                }
                index++
                N = N / 2
        }
        res := 0
        for i := 0; i < len(arr)-1; i++ {
                if arr[i+1]-arr[i] > res {
                        res = arr[i+1] - arr[i]
                }
        }
        return res
}

#
func binaryGap(N int) int {
```

(续下页)

(接上页)

```
        res := 0
        count := 0
        for N > 0 {
                if N%2 == 1 {
                        if count > res {
                                res = count
                        }
                        count = 1
                } else if count > 0 {
                        count++
                }
                N = N / 2
        }
        return res
}

#
func binaryGap(N int) int {
        res := 0
        str := strconv.FormatInt(int64(N), 2)
        j := -1
        for i := 0; i < len(str); i++ {
                if str[i] == '1' {
                        if j == -1 {
                                j = i
                        } else {
                                if i-j > res {
                                        res = i - j
                                }
                                j = i
                        }
                }
        }
        return res
}
```

25.19 872. 叶子相似的树 (2)

- 题目

```
请考虑一颗二叉树上所有的叶子，这些叶子的值按从左到右的顺序排列形成一个叶值序列 。
举个例子，如上图所示，给定一颗叶值序列为 (6, 7, 4, 9, 8) 的树。
如果有两颗二叉树的叶值序列是相同，那么我们就认为它们是 叶相似 的。
如果给定的两个头结点分别为 root1 和 root2 的树是叶相似的，则返回 true；否则返回 false。
提示：
    给定的两颗树可能会有 1 到 200 个结点。
    给定的两颗树上的值介于 0 到 200 之间。
```

- 解题思路

```
var a1, a2 []int

func leafSimilar(root1 *TreeNode, root2 *TreeNode) bool {
        a1 = make([]int, 0)
        a2 = make([]int, 0)
        dfs(root1, &a1)
        dfs(root2, &a2)
        if len(a1) != len(a2) {
                return false
        }
        for i := 0; i < len(a1); i++ {
                if a1[i] != a2[i] {
                        return false
                }
        }
        return true
}

func dfs(root *TreeNode, arr *[]int) {
        if root != nil {
                if root.Left == nil && root.Right == nil {
                        *arr = append(*arr, root.Val)
                        return
                }
                dfs(root.Left, arr)
                dfs(root.Right, arr)
        }
}
```

(续下页)

(接上页)

```
#
func leafSimilar(root1 *TreeNode, root2 *TreeNode) bool {
        a1 := make([]int, 0)
        a2 := make([]int, 0)
        bfs(root1, &a1)
        bfs(root2, &a2)
        if len(a1) != len(a2) {
                return false
        }
        for i := 0; i < len(a1); i++ {
                if a1[i] != a2[i] {
                        return false
                }
        }
        return true
}

func bfs(root *TreeNode, arr *[]int) {
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if node.Left == nil && node.Right == nil {
                        *arr = append(*arr, node.Val)
                }
                if node.Right != nil {
                        stack = append(stack, node.Right)
                }
                if node.Left != nil {
                        stack = append(stack, node.Left)
                }
        }
}
```

25.20 874. 模拟行走机器人 (2)

- 题目

```
机器人在一个无限大小的网格上行走，从点 (0, 0)␣
↪处开始出发，面向北方。该机器人可以接收以下三种类型的命令：
    -2：向左转 90 度
    -1：向右转 90 度
    1 <= x <= 9：向前移动 x 个单位长度
在网格上有一些格子被视为障碍物。
第 i 个障碍物位于网格点 (obstacles[i][0], obstacles[i][1])
机器人无法走到障碍物上，它将会停留在障碍物的前一个网格方块上，但仍然可以继续该路线的其余部分。
返回从原点到机器人的最大欧式距离的平方。

示例 1：输入：commands = [4,-1,3], obstacles = [] 输出：25
解释：机器人将会到达 (3, 4)
示例 2：输入：commands = [4,-1,4,-2,4], obstacles = [[2,4]] 输出：65
解释：机器人在左转走到 (1, 8) 之前将被困在 (1, 4) 处
提示：
    0 <= commands.length <= 10000
    0 <= obstacles.length <= 10000
    -30000 <= obstacle[i][0] <= 30000
    -30000 <= obstacle[i][1] <= 30000
    答案保证小于 2 ^ 31
```

- 解题思路

```
// 上、右、下、左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func robotSim(commands []int, obstacles [][]int) int {
        i := 0 // 方向， 0上， 1右， 2下， 3左
        x := 0
        y := 0
        res := 0
        m := map[string]bool{}
        for _, v := range obstacles {
                str := strconv.Itoa(v[0]) + "," + strconv.Itoa(v[1])
                m[str] = true
        }
        for _, v := range commands {
                if v == -2 {
                        i = (i + 3) % 4 // 左转
```

(续下页)

(接上页)

```
                } else if v == -1 {
                        i = (i + 1) % 4 // 右转
                } else {
                        for v > 0 {
                                ddx := x + dx[i]
                                ddy := y + dy[i]
                                tp := strconv.Itoa(ddx) + "," + strconv.Itoa(ddy)
                                if _, ok := m[tp]; ok {
                                        // 有障碍物，停止
                                        break
                                } else {
                                        x = ddx
                                        y = ddy
                                        if x*x+y*y > res {
                                                res = x*x + y*y
                                        }
                                }
                                v--
                        }
                }
        }
        return res
}

#
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func robotSim(commands []int, obstacles [][]int) int {
        m := make(map[string]bool, 10000)
        for _, o := range obstacles {
                i, j := o[0], o[1]
                m[encode(i, j)] = true
        }
        x, y, res := 0, 0, 0
        index := 0
        for _, c := range commands {
                index = (index + 4) % 4
                switch {
                case c == -2:
                        index--
                case c == -1:
                        index++
```

(续下页)

(接上页)

```
                default:
                        dx1, dy1 := dx[index], dy[index]
                        for c > 0 && !m[encode(x+dx1, y+dy1)] {
                                c--
                                x = x + dx1
                                y = y + dy1
                        }
                        if x*x+y*y > res {
                                res = x*x + y*y
                        }
                }
        }
        return res
}

func encode(x, y int) string {
        return strconv.Itoa(x) + "," + strconv.Itoa(y)
}
```

25.21 876. 链表的中间结点 (3)

- 题目

```
给定一个带有头结点 head 的非空单链表，返回链表的中间结点。
如果有两个中间结点，则返回第二个中间结点。
示例 1：输入：[1,2,3,4,5] 输出：此列表中的结点 3 (序列化形式：[3,4,5])
返回的结点值为 3 。 (测评系统对该结点序列化表述是 [3,4,5])。
注意，我们返回了一个 ListNode 类型的对象 ans，这样：
ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, 以及 ans.next.next.next = NULL.

示例 2：输入：[1,2,3,4,5,6] 输出：此列表中的结点 4 (序列化形式：[4,5,6])
由于该列表有两个中间结点，值分别为 3 和 4，我们返回第二个结点。
提示：
    给定链表的结点数介于 1 和 100 之间。
```

- 解题思路

```
func middleNode(head *ListNode) *ListNode {
        slow, fast := head, head
        for fast != nil && fast.Next != nil {
                slow = slow.Next
                fast = fast.Next.Next
```

(续下页)

(接上页)

```
        }
        return slow
}

#
func middleNode(head *ListNode) *ListNode {
        res := make([]*ListNode, 0)
        for head != nil {
                res = append(res, head)
                head = head.Next
        }
        return res[len(res)/2]
}

#
func middleNode(head *ListNode) *ListNode {
        count := 0
        temp := head
        for temp != nil {
                count++
                temp = temp.Next
        }
        mid := count / 2
        for head != nil {
                if mid == 0 {
                        return head
                }
                head = head.Next
                mid--
        }
        return head
}
```

25.22 883. 三维形体投影面积 (2)

- 题目

```
在 N * N 的网格中，我们放置了一些与 x，y，z 三轴对齐的 1 * 1 * 1 立方体。
每个值 v = grid[i][j] 表示 v 个正方体叠放在单元格 (i, j) 上。
现在，我们查看这些立方体在 xy、yz 和 zx 平面上的投影。
投影就像影子，将三维形体映射到一个二维平面上。
在这里，从顶部、前面和侧面看立方体时，我们会看到“影子”。
```

(续下页)

(接上页)

```
返回所有三个投影的总面积。

示例 1: 输入: [[2]] 输出: 5
示例 2: 输入: [[1,2],[3,4]] 输出: 17
解释: 这里有该形体在三个轴对齐平面上的三个投影(“阴影部分”)。
示例 3: 输入: [[1,0],[0,2]] 输出: 8
示例 4: 输入: [[1,1,1],[1,0,1],[1,1,1]] 输出: 14
示例 5: 输入: [[2,2,2],[2,1,2],[2,2,2]] 输出: 21
提示:
    1 <= grid.length = grid[0].length <= 50
    0 <= grid[i][j] <= 50
```

- 解题思路

```
// 1.xy面，grid[i][j]>0的个数累加
// 2.xz面，行的最大值累加
// 3.yz面，列的最大值累加
func projectionArea(grid [][]int) int {
        yz := [51]int{}
        xz := [51]int{}
        res := 0
        for i, line := range grid {
                for j, k := range line {
                        if k == 0 {
                                continue
                        }
                        res++
                        yz[i] = max(yz[i], k)
                        xz[j] = max(xz[j], k)
                }
        }
        for i := range yz {
                res = res + yz[i] + xz[i]
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
#
func projectionArea(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                yz := 0
                xz := 0
                // 每一行最大值之和，每一列最大值之和
                for j := 0; j < len(grid); j++ {
                        if grid[i][j] > 0 {
                                res++
                        }
                        if yz < grid[i][j] {
                                yz = grid[i][j]
                        }
                        if xz < grid[j][i] {
                                xz = grid[j][i]
                        }
                }
                res = res + yz + xz
        }
        return res
}
```

25.23 884. 两句话中的不常见单词 (2)

- 题目

```
给定两个句子 A 和 B 。 (句子是一串由空格分隔的单词。每个单词仅由小写字母组成。)
如果一个单词在其中一个句子中只出现一次，在另一个句子中却没有出现，那么这个单词就是不常见的。
返回所有不常用单词的列表。
您可以按任何顺序返回列表。
示例 1: 输入: A = "this apple is sweet", B = "this apple is sour"输出: ["sweet","sour
↪"]
示例 2: 输入: A = "apple apple", B = "banana"输出: ["banana"]
提示:
    0 <= A.length <= 200
    0 <= B.length <= 200
    A 和 B 都只包含空格和小写字母。
```

- 解题思路

```
func uncommonFromSentences(A string, B string) []string {
        m := map[string]int{}
        arrA := strings.Fields(A)
        arrB := strings.Fields(B)
        for _, v := range arrA {
                m[v]++
        }
        for _, v := range arrB {
                m[v]++
        }
        res := make([]string, 0)
        for k, v := range m {
                if v == 1 {
                        res = append(res, k)
                }
        }
        return res
}

#
func uncommonFromSentences(A string, B string) []string {
        m := map[string]int{}
        A = A + " " + B + " "
        j := 0
        for i := 0; i < len(A); i++ {
                if A[i] == ' ' {
                        m[A[j:i]]++
                        j = i + 1
                }
        }
        res := make([]string, 0)
        for k, v := range m {
                if v == 1 {
                        res = append(res, k)
                }
        }
        return res
}
```

25.24 888. 公平的糖果交换 (2)

- 题目

```
爱丽丝和鲍勃有不同大小的糖果棒：A[i] 是爱丽丝拥有的第 i 块糖的大小，B[j]␣
↪是鲍勃拥有的第 j 块糖的大小。
因为他们是朋友，所以他们想交换一个糖果棒，这样交换后，他们都有相同的糖果总量。
（一个人拥有的糖果总量是他们拥有的糖果棒大小的总和。）
返回一个整数数组 ans，其中 ans[0] 是爱丽丝必须交换的糖果棒的大小，ans[1] 是 Bob␣
↪必须交换的糖果棒的大小。
如果有多个答案，你可以返回其中任何一个。保证答案存在。
示例 1：输入：A = [1,1], B = [2,2] 输出：[1,2]
示例 2：输入：A = [1,2], B = [2,3] 输出：[1,2]
示例 3：输入：A = [2], B = [1,3] 输出：[2,3]
示例 4：输入：A = [1,2,5], B = [2,4] 输出：[5,4]
提示：
    1 <= A.length <= 10000
    1 <= B.length <= 10000
    1 <= A[i] <= 100000
    1 <= B[i] <= 100000
    保证爱丽丝与鲍勃的糖果总量不同。
    答案肯定存在。
```

- 解题思路

```
func fairCandySwap(A []int, B []int) []int {
        m := make(map[int]bool)
        sumA := 0
        sumB := 0
        for _, v := range A {
                sumA = sumA + v
                m[v] = true
        }
        for _, v := range B {
                sumB = sumB + v
        }
        half := (sumA - sumB) / 2
        a, b := 0, 0
        // sumA-A[i]+B[j] == sumB-B[j]+A[i]
        // sumA-sumB=2(A[i]-B[j])
        // (sumA-sumB)/2 = A[i]-B[j]
        for _, b = range B {
                a = b + half
                if m[a] == true {
```

(续下页)

(接上页)

```
                        return []int{a, b}
                }
        }
        return nil
}

#
func fairCandySwap(A []int, B []int) []int {
        sumA := 0
        sumB := 0
        for _, v := range A {
                sumA = sumA + v
        }
        for _, v := range B {
                sumB = sumB + v
        }
        for i := 0; i < len(A); i++ {
                for j := 0; j < len(B); j++ {
                        if sumA-A[i]+B[j] == sumB-B[j]+A[i] {
                                return []int{A[i], B[j]}
                        }
                }
        }
        return nil
}
```

25.25 892. 三维形体的表面积 (2)

- 题目

```
在 N * N 的网格上，我们放置一些 1 * 1 * 1  的立方体。
每个值 v = grid[i][j] 表示 v 个正方体叠放在对应单元格 (i, j) 上。
请你返回最终形体的表面积。
示例 1：输入：[[2]] 输出：10
示例 2：输入：[[1,2],[3,4]] 输出：34
示例 3：输入：[[1,0],[0,2]] 输出：16
示例 4：输入：[[1,1,1],[1,0,1],[1,1,1]] 输出：32
示例 5：输入：[[2,2,2],[2,1,2],[2,2,2]]输出：46
提示：
    1 <= N <= 50
    0 <= grid[i][j] <= 50
```

- 解题思路

```
// 第1步：总表面积是个数*6
// 第2步：同一位置，从2层以上开始，每升高一层，减少2个面
// 第3步：左右位置，每相邻一个，减少2个面
// 第4步：前后位置，每相邻一个，减少2个面
func surfaceArea(grid [][]int) int {
        sum := 0
        for i, rows := range grid {
                for j := range rows {
                        sum = sum + grid[i][j]*6
                        if grid[i][j] > 1 {
                                sum = sum - (grid[i][j]-1)*2
                        }
                        if j > 0 {
                                sum = sum - min(grid[i][j], grid[i][j-1])*2
                        }
                        if i > 0 {
                                sum = sum - min(grid[i][j], grid[i-1][j])*2
                        }
                }
        }
        return sum
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
// 上、右、下、左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func surfaceArea(grid [][]int) int {
        sum := 0
        for i, rows := range grid {
                for j := range rows {
                        sum = sum + grid[i][j]*6
                        if grid[i][j] > 1 {
                                sum = sum - (grid[i][j]-1)*2
                        }
                        for k := 0; k < 4; k++ {
```

(续下页)

(接上页)

```
                                x, y := i+dx[k], j+dy[k]
                                if x >= 0 && x < len(grid) && y >= 0 && y <␣
↪len(grid[0]) {
                                        sum = sum - min(grid[i][j], grid[x][y])
                                }
                        }
                }
        }
        return sum
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

25.26 893. 特殊等价字符串组 (2)

- 题目

```
你将得到一个字符串数组 A。
如果经过任意次数的移动，S == T，那么两个字符串 S 和 T 是特殊等价的。
一次移动包括选择两个索引 i 和 j，且 i % 2 == j % 2，交换 S[j] 和 S [i]。
现在规定，A 中的特殊等价字符串组是 A 的非空子集 S，
这样不在 S 中的任何字符串与 S 中的任何字符串都不是特殊等价的。
返回 A 中特殊等价字符串组的数量。

示例 1：输入：["a","b","c","a","c","c"] 输出：3
解释：3 组 ["a","a"], ["b"], ["c","c","c"]
示例 2：输入：["aa","bb","ab","ba"] 输出：4
解释：4 组 ["aa"], ["bb"], ["ab"], ["ba"]
示例 3：输入：["abc","acb","bac","bca","cab","cba"] 输出：3
解释：3 组 ["abc","cba"], ["acb","bca"], ["bac","cab"]
示例 4：输入：["abcd","cdab","adcb","cbad"] 输出：1
解释：1 组 ["abcd","cdab","adcb","cbad"]
提示：
    1 <= A.length <= 1000
    1 <= A[i].length <= 20
    所有 A[i] 都具有相同的长度。
    所有 A[i] 都只由小写字母组成。
```

- 解题思路

```
func numSpecialEquivGroups(A []string) int {
        groups := make(map[[26]int]bool)
        for _, a := range A {
                count := [26]int{}
                i := 0
                for i = 0; i < len(A[0]); i++ {
                        count[a[i]-'a']++
                        if i%2 == 0 {
                                count[a[i]-'a'] += 1000
                        }
                }
                groups[count] = true
        }
        return len(groups)
}

#
func numSpecialEquivGroups(A []string) int {
        groups := make(map[string]bool)
        for _, a := range A {
                odd := make([]byte, 0)
                even := make([]byte, 0)
                for i := 0; i < len(a); i++ {
                        if i%2 == 0 {
                                even = append(even, a[i]-'a')
                        } else {
                                odd = append(odd, a[i]-'a')
                        }
                }
                sort.Slice(odd, func(i, j int) bool {
                        return odd[i] < odd[j]
                })
                sort.Slice(even, func(i, j int) bool {
                        return even[i] < even[j]
                })
                groups[string(odd)+string(even)] = true
        }
        return len(groups)
}
```

25.27 896. 单调数列 (3)

- 题目

```
如果数组是单调递增或单调递减的，那么它是单调的。
如果对于所有 i <= j，A[i] <= A[j]，那么数组 A 是单调递增的。
如果对于所有 i <= j，A[i]> = A[j]，那么数组 A 是单调递减的。
当给定的数组 A 是单调数组时返回 true，否则返回 false。
示例 1：输入：[1,2,2,3]输出：true
示例 2：输入：[6,5,4,4]输出：true
示例 3：输入：[1,3,2]输出：false
示例 4：输入：[1,2,4,5]输出：true
示例 5：输入：[1,1,1]输出：true
提示：
    1 <= A.length <= 50000
    -100000 <= A[i] <= 100000
```

- 解题思路

```
func isMonotonic(A []int) bool {
        toEnd := true
        toFirst := true
        for i := 0; i < len(A)-1; i++ {
                if A[i] > A[i+1] {
                        toEnd = false
                }
                if A[i] < A[i+1] {
                        toFirst = false
                }
        }
        return toEnd || toFirst
}

#
func isMonotonic(A []int) bool {
        return inc(A) || desc(A)
}

func inc(A []int) bool {
        for i := 0; i < len(A)-1; i++ {
                if A[i] > A[i+1] {
                        return false
                }
        }
```

(续下页)

(接上页)

```
        return true
}

func desc(A []int) bool {
        for i := 0; i < len(A)-1; i++ {
                if A[i] < A[i+1] {
                        return false
                }
        }
        return true
}

#
func isMonotonic(A []int) bool {
        if len(A) == 1 {
                return true
        }
        temp := A[len(A)-1] - A[0]
        for i := 0; i < len(A)-1; i++ {
                if temp > 0 && A[i] > A[i+1] {
                        return false
                } else if temp < 0 && A[i] < A[i+1] {
                        return false
                } else if temp == 0 && A[i] != A[i+1] {
                        return false
                }
        }
        return true
}
```

25.28 897. 递增顺序查找树 (3)

- 题目

```
给你一个树，请你 按中序遍历 重新排列树，使树中最左边的结点现在是树的根，
并且每个结点没有左子结点，只有一个右子结点。
示例 ：输入：[5,3,6,2,4,null,8,1,null,null,null,7,9]
       5
      / \
    3    6
   / \    \
  2   4    8
```

(续下页)

(接上页)

```
 /         / \
1         7   9
输出：[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]
 1
  \
   2
    \
     3
      \
       4
        \
         5
          \
           6
            \
             7
              \
               8
                \
                 9
提示：
    给定树中的结点数介于 1 和 100 之间。
    每个结点都有一个从 0 到 1000 范围内的唯一整数值。
```

- 解题思路

```
func increasingBST(root *TreeNode) *TreeNode {
        arr := make([]int, 0)
        dfs(root, &arr)
        if len(arr) == 0 {
                return root
        }
        newRoot := &TreeNode{Val: arr[0]}
        cur := newRoot
        for i := 1; i < len(arr); i++ {
                cur.Right = &TreeNode{Val: arr[i]}
                cur = cur.Right
        }
        return newRoot
}

func dfs(node *TreeNode, arr *[]int) {
        if node == nil {
                return
```

(续下页)

(接上页)

```
        }
        dfs(node.Left, arr)
        *arr = append(*arr, node.Val)
        dfs(node.Right, arr)
}

#
var prev *TreeNode

func increasingBST(root *TreeNode) *TreeNode {
        prev = &TreeNode{}
        head := prev
        dfs(root)
        return head.Right
}

func dfs(node *TreeNode) {
        if node == nil {
                return
        }
        dfs(node.Left)
        node.Left = nil
        prev.Right = node
        prev = node
        dfs(node.Right)
}

#
func increasingBST(root *TreeNode) *TreeNode {
        stack := make([]*TreeNode, 0)
        newRoot := &TreeNode{}
        stack = append(stack, root)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                if node.Right != nil {
                        stack = append(stack, node.Right)
                        node.Right = nil
                        continue
                }
                stack = stack[:len(stack)-1]
                node.Right = newRoot.Right
                newRoot.Right = node
                if node.Left != nil {
```

(续下页)

(接上页)

```
                        stack = append(stack, node.Left)
                        node.Left = nil
                }
        }
        return newRoot.Right
}
```

CHAPTER 26

0801-0900-Medium

26.1 801. 使序列递增的最小交换次数 (1)

- 题目

```
我们有两个长度相等且不为空的整型数组A和B。
我们可以交换A[i]和B[i]的元素。注意这两个元素在各自的序列中应该处于相同的位置。
在交换过一些元素之后，数组A和B都应该是严格递增的
（数组严格递增的条件仅为A[0] < A[1] < A[2] < ... < A[A.length - 1]）。
给定数组A和B，请返回使得两个数组均保持严格递增状态的最小交换次数。假设给定的输入总是有效的。
示例:输入: A = [1,3,5,4], B = [1,2,3,7] 输出: 1
解释: 交换 A[3] 和 B[3] 后，两个数组如下:
A = [1, 3, 5, 7] , B = [1, 2, 3, 4]
两个数组均为严格递增的。
注意:A, B两个数组的长度总是相等的，且长度的范围为[1, 1000]。
A[i], B[i]均为[0, 2000]区间内的整数。
```

- 解题思路

```
func minSwap(A []int, B []int) int {
        n := len(A)
        dp := make([][2]int, n)
        dp[0][0] = 0 // dp[i][0] 第i个位置不换
        dp[0][1] = 1 // dp[i][1] 第i个位置换
        for i := 1; i < n; i++ {
```

(续下页)

(接上页)

```
                if A[i-1] < A[i] && B[i-1] < B[i] {
                        if A[i-1] < B[i] && B[i-1] < A[i] { // 可换可不换
                                dp[i][0] = min(dp[i-1][0], dp[i-1][1])
                                dp[i][1] = min(dp[i-1][0], dp[i-1][1]) + 1
                        } else {
                                dp[i][0] = dp[i-1][0]     // 不交换则上一轮也不交换
                                dp[i][1] = dp[i-1][1] + 1 // 交换则上一轮也交换
                        }
                } else {
                        dp[i][0] = dp[i-1][1]     // 不交换则上一轮必须交换
                        dp[i][1] = dp[i-1][0] + 1 // 交换，则上一轮不能交换
                }
        }
        return min(dp[n-1][0], dp[n-1][1])
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

26.2 802. 找到最终的安全状态 (2)

- 题目

```
在有向图中，从某个节点和每个转向处开始出发，沿着图的有向边走。如果到达的节点是终点（即它没有连出的有
如果从起始节点出发，最后必然能走到终点，就认为起始节点是 最终安全 的。
更具体地说，对于最终安全的起始节点而言，存在一个自然数 k ，无论选择沿哪条有向边行走␣
↪，走了不到 k 步后必能停止在一个终点上。
返回一个由图中所有最终安全的起始节点组成的数组作为答案。答案数组中的元素应当按 升序␣
↪排列。
该有向图有 n 个节点，按 0 到 n - 1 编号，其中 n 是graph的节点数。
图以下述形式给出：graph[i] 是编号 j 节点的一个列表，满足 (i, j) 是图的一条有向边。
示例 1：输入：graph = [[1,2],[2,3],[5],[0],[5],[],[]] 输出：[2,4,5,6]
解释：示意图如上。
示例 2：输入：graph = [[1,2,3,4],[1,2],[3,4],[0,4],[]] 输出：[4]
提示：n == graph.length
1 <= n <= 104
0 <= graph[i].legnth <= n
graph[i] 按严格递增顺序排列。
```

(续下页)

(接上页)

图中可能包含自环。
图中边的数目在范围 [1, 4 * 104] 内。

- 解题思路

```
func eventualSafeNodes(graph [][]int) []int {
        n := len(graph)
        safeArr := make([]bool, n) // 安全节点
        arr := make([][]int, n)
        m := make(map[int]map[int]bool)
        queue := make([]int, 0)
        for i := 0; i < n; i++ {
                m[i] = make(map[int]bool)
                if len(graph[i]) == 0 { // 没有出节点，是安全节点
                        queue = append(queue, i)
                }
                for j := 0; j < len(graph[i]); j++ {
                        a, b := i, graph[i][j]     // a=>b
                        arr[b] = append(arr[b], a) // 反向b=>a
                        m[a][b] = true             // a=>b
                }
        }
        res := make([]int, 0)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                safeArr[node] = true
                for i := 0; i < len(arr[node]); i++ { // 反向边遍历
                        index := arr[node][i]
                        delete(m[index], node)  // 删除
                        if len(m[index]) == 0 { // 没有出节点
                                queue = append(queue, index)
                        }
                }
        }
        for i := 0; i < n; i++ {
                if safeArr[i] == true {
                        res = append(res, i)
                }
        }
        return res
}

# 2
```

(续下页)

(接上页)

```
func eventualSafeNodes(graph [][]int) []int {
        n := len(graph)
        visited := make([]int, n) // 0: 未访问、1: 在访问（有环）、2: 安全
        res := make([]int, 0)
        for i := 0; i < n; i++ {
                if dfs(graph, i, visited) == true {
                        res = append(res, i)
                }
        }
        return res
}

func dfs(graph [][]int, index int, visited []int) bool {
        if visited[index] > 0 {
                return visited[index] == 2
        }
        visited[index] = 1
        for i := 0; i < len(graph[index]); i++ {
                next := graph[index][i]
                if visited[next] == 1 || dfs(graph, next, visited) == false {
                        return false
                }
        }
        visited[index] = 2
        return true
}
```

26.3 807. 保持城市天际线 (1)

- 题目

```
在二维数组grid中，grid[i][j]代表位于某处的建筑物的高度。
我们被允许增加任何数量（不同建筑物的数量可能不同）的建筑物的高度。 高度 0
↪也被认为是建筑物。
最后，从新数组的所有四个方向（即顶部，底部，左侧和右侧）观看的“天际线”必须与原始数组的天际线相同。
城市的天际线是从远处观看时，由所有建筑物形成的矩形的外部轮廓。 请看下面的例子。
建筑物高度可以增加的最大总和是多少？
例子：输入： grid = [[3,0,8,4],[2,4,5,7],[9,2,6,3],[0,3,1,0]] 输出： 35
解释： The grid is:
[ [3, 0, 8, 4],
  [2, 4, 5, 7],
  [9, 2, 6, 3],
```

(续下页)

(接上页)

```
  [0, 3, 1, 0] ]
从数组竖直方向（即顶部，底部）看“天际线”是：[9, 4, 8, 7]
从水平水平方向（即左侧，右侧）看“天际线”是：[8, 7, 9, 3]
在不影响天际线的情况下对建筑物进行增高后，新数组如下：
gridNew = [ [8, 4, 8, 7],
            [7, 4, 7, 7],
            [9, 4, 8, 7],
            [3, 3, 3, 3] ]
说明：1 < grid.length = grid[0].length <= 50。
grid[i][j] 的高度范围是： [0, 100]。
一座建筑物占据一个grid[i][j]：换言之，它们是 1 x 1 x grid[i][j] 的长方体。
```

- 解题思路

```
func maxIncreaseKeepingSkyline(grid [][]int) int {
        res := 0
        n, m := len(grid), len(grid[0])
        row := make([]int, n)
        col := make([]int, m)
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        row[i] = max(row[i], grid[i][j])
                        col[j] = max(col[j], grid[i][j])
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        res = res + min(row[i], col[j]) - grid[i][j]
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}
```

26.4 808. 分汤 (1)

- 题目

```
有A和B 两种类型的汤。一开始每种类型的汤有N毫升。有四种分配操作：
提供 100ml 的汤A 和 0ml 的汤B。
提供 75ml 的汤A 和 25ml 的汤B。
提供 50ml 的汤A 和 50ml 的汤B。
提供 25ml 的汤A 和 75ml 的汤B。
当我们把汤分配给某人之后，汤就没有了。每个回合，我们将从四种概率同为0.
↪25的操作中进行分配选择。
如果汤的剩余量不足以完成某次操作，我们将尽可能分配。当两种类型的汤都分配完时，停止操作。
注意不存在先分配100 ml汤B的操作。
需要返回的值： 汤A先分配完的概率 + 汤A和汤B同时分配完的概率 / 2。
示例： 输入： N = 50 输出： 0.625
解释:如果我们选择前两个操作，A将首先变为空。对于第三个操作，A和B会同时变为空。
对于第四个操作，B将首先变为空。
所以A变为空的总概率加上A和B同时变为空的概率的一半是 0.25 *(1 + 1 + 0.5 + 0)= 0.625。
注释： 0 <= N <= 10^9。
返回值在10^-6的范围将被认为是正确的。
```

- 解题思路

```
func soupServings(N int) float64 {
        n := N / 25
        if N%25 > 0 {
                n = n + 1
        }
        if n >= 500 {
                return 1.0
        }
        // 当给定i毫升的A和j毫升的B的概率
        // dp[i][j]的概率=0.25*(dp[i-4][j]+dp[i-3][j-1]+dp[i-2][j-2]+dp[i-1][j-3])
        dp := make([][]float64, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]float64, n+1)
        }
        dp[0][0] = 0.5
        for i := 1; i <= n; i++ {
```

(续下页)

(接上页)

```
                dp[i][0] = 0
                dp[0][i] = 1
        }
        for i := 1; i <= n; i++ {
                a1 := max(i-4, 0)
                a2 := max(i-3, 0)
                a3 := max(i-2, 0)
                a4 := max(i-1, 0)
                for j := 1; j <= n; j++ {
                        b1 := max(j, 0)
                        b2 := max(j-1, 0)
                        b3 := max(j-2, 0)
                        b4 := max(j-3, 0)
                        dp[i][j] = 0.25 * (dp[a1][b1] + dp[a2][b2] + dp[a3][b3] +␣
↪dp[a4][b4])
                }
        }
        return dp[n][n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

26.5 809. 情感丰富的文字 (1)

- 题目

```
有时候人们会用重复写一些字母来表示额外的感受，比如 "hello" -> "heeellooo", "hi" ->
↪"hiii"。
我们将相邻字母都相同的一串字符定义为相同字母组，例如：'h", "eee", "ll", "ooo"。
对于一个给定的字符串 S ，如果另一个单词能够通过将一些字母组扩张从而使其和 S 相同，
我们将这个单词定义为可扩张的（stretchy）。
扩张操作定义如下：选择一个字母组（包含字母c），然后往其中添加相同的字母c使其长度达到␣
↪3 或以上。
例如，以"hello" 为例，我们可以对字母组"o" 扩张得到 "hellooo"，
但是无法以同样的方法得到 "helloo" 因为字母组 "oo" 长度小于3。
此外，我们可以进行另一种扩张 "ll" -> "llllll" 以获得"hellllllooo"。
如果S = "hellllllooo"，那么查询词"hello" 是可扩张的，
```

(续下页)

(接上页)

```
因为可以对它执行这两种扩张操作使得query = "hello" -> "hellooo" ->"hellllllooo" = S。
输入一组查询单词，输出其中可扩张的单词数量。
示例：输入：  S = "heeellooo" words = ["hello", "hi", "helo"] 输出：1
解释：我们能通过扩张 "hello" 的 "e" 和 "o" 来得到 "heeellooo"。
我们不能通过扩张 "helo" 来得到 "heeellooo" 因为 "ll" 的长度小于 3 。
说明：0 <= len(S) <= 100。
0 <= len(words) <= 100。
0 <= len(words[i]) <= 100。
S和所有在words中的单词都只由小写字母组成。
```

- 解题思路

```
func expressiveWords(S string, words []string) int {
        res := 0
        arr := getCount(S)
        for i := 0; i < len(words); i++ {
                temp := getCount(words[i])
                if len(temp) != len(arr) {
                        continue
                }
                flag := true
                for j := 0; j < len(arr); j = j + 2 {
                        if arr[j] != temp[j] {
                                flag = false
                                break
                        }
                        if arr[j+1] == temp[j+1] {
                                continue
                        }
                        if arr[j+1] < 3 || arr[j+1] < temp[j+1] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res++
                }
        }
        return res
}

func getCount(str string) []int {
        res := make([]int, 0)
        count := 1
```

(续下页)

(接上页)

```
        for i := 0; i < len(str); i++ {
                if i == len(str)-1 || str[i] != str[i+1] {
                        res = append(res, int(str[i]), count)
                        count = 1
                } else {
                        count++
                }
        }
        return res
}
```

26.6 813. 最大平均值和的分组 (3)

- 题目

```
我们将给定的数组A分成K个相邻的非空子数组␣
↪，我们的分数由每个子数组内的平均值的总和构成。
计算我们所能得到的最大分数是多少。
注意我们必须使用 A 数组中的每一个数进行分组，并且分数不一定需要是整数。
示例:输入: A = [9,1,2,3,9] K = 3 输出: 20
解释: A 的最优分组是[9], [1, 2, 3], [9]. 得到的分数是 9 + (1 + 2 + 3) / 3 + 9 = 20.
我们也可以把 A 分成[9, 1], [2], [3, 9].
这样的分组得到的分数为 5 + 2 + 6 = 13, 但不是最大值.
说明:1 <= A.length <= 100.
1 <= A[i] <= 10000.
1 <= K <= A.length.
答案误差在10^-6内被视为是正确的。
```

- 解题思路

```
func largestSumOfAverages(A []int, K int) float64 {
        n := len(A)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + A[i]
        }
        dp := make([]float64, n) // dp[i]=>A[i:]的最大平均值
        for i := 0; i < n; i++ {
                dp[i] = float64(arr[n]-arr[i]) / float64(n-i) // 划分为1组
        }
        for k := 1; k < K; k++ { // K组可以划分K-1次
                for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                        for j := i + 1; j < n; j++ {
                                target := dp[j] + float64(arr[j]-arr[i])/float64(j-i)
                                dp[i] = math.Max(dp[i], target)
                        }
                }
        }
        return dp[0]
}

# 2
func largestSumOfAverages(A []int, K int) float64 {
        n := len(A)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + A[i]
        }
        dp := make([][]float64, n) // dp[i]=>A[i:]的最大平均值
        for i := 0; i < n; i++ {
                dp[i] = make([]float64, K)
                dp[i][0] = float64(arr[n]-arr[i]) / float64(n-i) // 划分为1组
        }
        for k := 1; k < K; k++ { // K组可以划分K-1次
                for i := 0; i < n; i++ {
                        for j := i + 1; j < n; j++ {
                                target := dp[j][k-1] + float64(arr[j]-arr[i])/
↪float64(j-i)
                                dp[i][k] = math.Max(dp[i][k], target)
                        }
                }
        }
        return dp[0][K-1]
}

# 3
func largestSumOfAverages(A []int, K int) float64 {
        n := len(A)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + A[i]
        }
        dp := make([][]float64, n+1)
        for i := 1; i <= n; i++ {
                dp[i] = make([]float64, K+1)
```

(续下页)

(接上页)

```
                dp[i][1] = float64(arr[i]) / float64(i) // 划分为1组
        }
        for i := 1; i <= n; i++ {
                for k := 2; k <= K && k <= i; k++ {
                        for j := 1; j < i; j++ {
                                target := dp[j][k-1] + float64(arr[i]-arr[j])/
↪float64(i-j)
                                dp[i][k] = math.Max(dp[i][k], target)
                        }
                }
        }
        return dp[n][K]
}
```

26.7 814. 二叉树剪枝 (1)

- 题目

```
给定二叉树根结点 root ，此外树的每个结点的值要么是 0，要么是 1。
返回移除了所有不包含 1 的子树的原二叉树。
( 节点 X 的子树为 X 本身，以及所有 X 的后代。)
示例1:输入: [1,null,0,0,1] 输出: [1,null,0,null,1]
解释: 只有红色节点满足条件“所有不包含 1 的子树”。 右图为返回的答案。
示例2:输入: [1,0,1,0,0,0,1] 输出: [1,null,1,null,1]
示例3:输入: [1,1,0,1,1,0,1,0] 输出: [1,1,0,1,1,null,1]
说明:给定的二叉树最多有 100 个节点。
    每个节点的值只会为 0 或 1 。
```

- 解题思路

```
func pruneTree(root *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        root.Left = pruneTree(root.Left)
        root.Right = pruneTree(root.Right)
        if root.Left == nil && root.Right == nil && root.Val == 0 {
                return nil
        }
        return root
}
```

26.8 816. 模糊坐标 (1)

- 题目

```
我们有一些二维坐标，如"(1, 3)"或"(2, 0.5)
↪"，然后我们移除所有逗号，小数点和空格，得到一个字符串S。
返回所有可能的原始字符串到一个列表中。
原始的坐标表示法不会存在多余的零，所以不会出现类似于"00", "0.0", "0.00", "1.0", "001",
"00.01"或一些其他更小的数来表示坐标。
此外，一个小数点前至少存在一个数，所以也不会出现“.1”形式的数字。
最后返回的列表可以是任意顺序的。而且注意返回的两个数字中间（逗号之后）都有一个空格。
示例 1:输入: "(123)" 输出: ["(1, 23)", "(12, 3)", "(1.2, 3)", "(1, 2.3)"]
示例 2:输入: "(00011)" 输出: ["(0.001, 1)", "(0, 0.011)"]
解释: 0.0, 00, 0001 或 00.01 是不被允许的。
示例 3: 输入: "(0123)"
输出: ["(0, 123)", "(0, 12.3)", "(0, 1.23)", "(0.1, 23)", "(0.1, 2.3)", "(0.12, 3)"]
示例 4:输入: "(100)" 输出: [(10, 0)]
解释: 1.0 是不被允许的。
提示:4 <= S.length <= 12.
S[0] = "(", S[S.length - 1] = ")", 且字符串S中的其他元素都是数字。
```

- 解题思路

```
func ambiguousCoordinates(s string) []string {
        res := make([]string, 0)
        str := s[1 : len(s)-1]
        n := len(str)
        for i := 1; i <= n-1; i++ { // 枚举左右2边
                left := str[:i]
                right := str[i:]
                a := process(left)
                b := process(right)
                for j := 0; j < len(a); j++ {
                        for k := 0; k < len(b); k++ {
                                res = append(res, "("+a[j]+", "+b[k]+")")
                        }
                }
        }
        return res
}

func process(str string) []string {
        res := make([]string, 0)
        n := len(str)
```

(续下页)

(接上页)

```
        for i := 1; i <= n; i++ {
                left := str[:i]
                right := str[i:]
                if judge(left, right) == true {
                        if i == n {
                                res = append(res, left+""+right)
                        } else {
                                res = append(res, left+"."+right)
                        }
                }
        }
        return res
}

func judge(a, b string) bool {
        if (len(a) >= 2 && a[0] == '0') ||
                (len(b) >= 1 && b[len(b)-1] == '0') {
                return false
        }
        return true
}
```

26.9 817. 链表组件 (2)

- 题目

```
给定链表头结点head，该链表上的每个结点都有一个 唯一的整型值 。
同时给定列表G，该列表是上述链表中整型值的一个子集。
返回列表G中组件的个数，这里对组件的定义为：链表中一段最长连续结点的值（该值必须在列表G中）构成的集合
示例1：输入：head: 0->1->2->3G = [0, 1, 3] 输出：2
解释：链表中,0 和 1 是相连接的，且 G 中不包含 2，所以 [0, 1] 是 G 的一个组件，
同理 [3] 也是一个组件，故返回 2。
示例 2：输入：head: 0->1->2->3->4 G = [0, 3, 1, 4] 输出：2
解释：链表中，0 和 1 是相连接的，3 和 4 是相连接的，所以 [0, 1] 和 [3, 4]␣
↪是两个组件，故返回 2。
提示：如果N是给定链表head的长度，1 <= N <= 10000。
链表中每个结点的值所在范围为[0, N - 1]。
1 <= G.length <= 10000
G 是链表中所有结点的值的一个子集.
```

- 解题思路

```
func numComponents(head *ListNode, G []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(G); i++ {
                m[G[i]] = true
        }
        res := 0
        for head != nil {
                if m[head.Val] == true &&
                        (head.Next == nil || m[head.Next.Val] == false) {
                        res++
                }
                head = head.Next
        }
        return res
}

# 2
func numComponents(head *ListNode, G []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(G); i++ {
                m[G[i]] = true
        }
        res := 0
        flag := false
        for head != nil {
                if m[head.Val] == true {
                        if flag == false {
                                res++
                                flag = true
                        }
                } else {
                        flag = false
                }
                head = head.Next
        }
        return res
}
```

26.10 820. 单词的压缩编码 (3)

- 题目

```
给定一个单词列表，我们将这个列表编码成一个索引字符串 S 与一个索引列表 A。
例如，如果这个列表是 ["time", "me", "bell"],
我们就可以将其表示为 S = "time#bell#" 和 indexes = [0, 2, 5]。
对于每一个索引，我们可以通过从字符串 S 中索引的位置开始读取字符串，直到 "#" 结束，
来恢复我们之前的单词列表。
那么成功对给定单词列表进行编码的最小字符串长度是多少呢？
示例： 输入: words = ["time", "me", "bell"] 输出: 10
说明: S = "time#bell#" ,  indexes = [0, 2, 5] 。
提示：1 <= words.length <= 2000
    1 <= words[i].length <= 7
    每个单词都是小写字母 。
```

- 解题思路

```
func minimumLengthEncoding(words []string) int {
        res := 0
        m := make(map[string]bool)
        for i := 0; i < len(words); i++ {
                m[words[i]] = true
        }
        for k := range m {
                for i := 1; i < len(k); i++ {
                        delete(m, k[i:])
                }
        }
        for k := range m {
                res = res + len(k) + 1
        }
        return res
}

# 2
func minimumLengthEncoding(words []string) int {
        res := 0
        m := make(map[string]bool)
        for i := 0; i < len(words); i++ {
                m[words[i]] = true
        }
        words = make([]string, 0)
        for k := range m {
```

(续下页)

(接上页)

```
                words = append(words, k)
        }
        sort.Slice(words, func(i, j int) bool {
                return len(words[i]) < len(words[j])
        })
        for i := len(words) - 1; i >= 0; i-- {
                if m[words[i]] == false {
                        continue
                }
                for j := i - 1; j >= 0; j-- {
                        if strings.HasSuffix(words[i], words[j]) == true {
                                m[words[j]] = false
                        }
                }
        }
        for k := range m {
                if m[k] == true {
                        res = res + len(k) + 1
                }
        }
        return res
}

# 3
func minimumLengthEncoding(words []string) int {
        res := 0
        arr := make([]string, 0)
        for k := range words {
                arr = append(arr, reverse(words[k]))
        }
        sort.Strings(arr)
        for i := 0; i < len(arr)-1; i++ {
                length := len(arr[i])
                if length <= len(arr[i+1]) && arr[i] == arr[i+1][:length] {
                        continue
                }
                res = res + length + 1
        }
        return res + len(arr[len(arr)-1]) + 1
}

func reverse(str string) string {
        res := make([]byte, 0)
```

(续下页)

(接上页)

```
        for i := len(str) - 1; i >= 0; i-- {
                res = append(res, str[i])
        }
        return string(res)
}
```

26.11 822. 翻转卡片游戏 (1)

- 题目

```
在桌子上有 N
↪张卡片，每张卡片的正面和背面都写着一个正数（正面与背面上的数有可能不一样）。
我们可以先翻转任意张卡片，然后选择其中一张卡片。
如果选中的那张卡片背面的数字 X
↪与任意一张卡片的正面的数字都不同，那么这个数字是我们想要的数字。
哪个数是这些想要的数字中最小的数（找到这些数中的最小值）呢？如果没有一个数字符合要求的，输出
↪0。
其中，fronts[i]和backs[i]分别代表第i张卡片的正面和背面的数字。
如果我们通过翻转卡片来交换正面与背面上的数，那么当初在正面的数就变成背面的数，背面的数就变成正面的数
示例：输入：fronts = [1,2,4,4,7], backs = [1,3,4,1,3] 输出：2
解释：假设我们翻转第二张卡片，那么在正面的数变成了 [1,3,4,4,7] ， 背面的数变成了 [1,2,
↪4,1,3]。
接着我们选择第二张卡片，因为现在该卡片的背面的数是 2，2 与任意卡片上正面的数都不同，
所以 2 就是我们想要的数字。
提示：1 <= fronts.length == backs.length<=1000
1 <=fronts[i]<= 2000
1 <= backs[i]<= 2000
```

- 解题思路

```
func flipgame(fronts []int, backs []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(fronts); i++ {
                if fronts[i] == backs[i] { // 前后相同，不能选
                        m[fronts[i]] = true
                }
        }
        res := math.MaxInt32
        for i := 0; i < len(fronts); i++ {
                if m[fronts[i]] == false { // 不相同
                        if fronts[i] < res {
                                res = fronts[i]
```

(续下页)

(接上页)

```
                        }
                }
                if m[backs[i]] == false { // 不相同
                        if backs[i] < res {
                                res = backs[i]
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}
```

26.12 823. 带因子的二叉树 (2)

- 题目

```
给出一个含有不重复整数元素的数组，每个整数均大于 1。
我们用这些整数来构建二叉树，每个整数可以使用任意次数。
其中：每个非叶结点的值应等于它的两个子结点的值的乘积。
满足条件的二叉树一共有多少个？返回的结果应模除 10 ** 9 + 7。
示例 1:输入：A = [2, 4] 输出：3
解释：我们可以得到这些二叉树：[2], [4], [4, 2, 2]
示例 2:输入：A = [2, 4, 5, 10] 输出：7
解释：我们可以得到这些二叉树：[2], [4], [5], [10], [4, 2, 2], [10, 2, 5], [10, 5, 2].
提示:1 <= A.length <=1000.
2 <=A[i]<=10 ^ 9.
```

- 解题思路

```
var mod = 1000000007

func numFactoredBinaryTrees(arr []int) int {
        sort.Ints(arr)
        n := len(arr)
        m := make(map[int]int)
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = 1
                m[arr[i]] = i
        }
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
                        if arr[i]%arr[j] == 0 {
                                c := arr[i] / arr[j]
                                if v, ok := m[c]; ok {
                                        dp[i] = (dp[i] + dp[j]*dp[v]) % mod
                                }
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                res = (res + dp[i]) % mod
        }
        return res
}

# 2
var mod = 1000000007

func numFactoredBinaryTrees(arr []int) int {
        sort.Ints(arr)
        n := len(arr)
        dp := make(map[int]int)
        res := 0
        for i := 0; i < n; i++ {
                dp[arr[i]] = 1
                for j := 0; j < i; j++ {
                        if arr[i]%arr[j] == 0 {
                                c := arr[i] / arr[j]
                                dp[arr[i]] = (dp[arr[i]] + dp[arr[j]]*dp[c]) % mod
                        }
                }
                res = (res + dp[arr[i]]) % mod
        }
        return res
}
```

26.13 825. 适龄的朋友 (2)

- 题目

```
人们会互相发送好友请求，现在给定一个包含有他们年龄的数组，ages[i]表示第 i 个人的年龄。
当满足以下任一条件时，A 不能给 B（A、B不为同一人）发送好友请求：
age[B]<= 0.5 * age[A]+ 7
age[B]> age[A]
age[B]> 100 &&age[A]< 100
否则，A 可以给 B 发送好友请求。
注意如果 A 向 B 发出了请求，不等于 B 也一定会向A␣
↪发出请求。而且，人们不会给自己发送好友请求。
求总共会发出多少份好友请求?
示例 1: 输入: [16,16] 输出: 2
解释: 二人可以互发好友申请。
示例 2: 输入: [16,17,18] 输出: 2
解释: 好友请求可产生于 17 -> 16, 18 -> 17.
示例 3: 输入: [20,30,100,110,120] 输出: 3
解释: 好友请求可产生于 110 -> 100, 120 -> 110, 120 -> 100.
提示: 1 <= ages.length<= 20000.
1 <= ages[i] <= 120.
```

- 解题思路

```
func numFriendRequests(ages []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(ages); i++ {
                m[ages[i]]++
        }
        for k, v := range m {
                for key, value := range m {
                        if key <= k/2+7 || key > k || (key > 100 && k < 100) {
                                continue
                        } else if k == key {
                                res = res + v*(value-1)
                        } else {
                                res = res + v*value
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func numFriendRequests(ages []int) int {
        res := 0
        arr := [121]int{}
        for i := 0; i < len(ages); i++ {
                arr[ages[i]]++
        }
        for a := 0; a <= 120; a++ {
                countA := arr[a]
                for b := 0; b <= 120; b++ {
                        countB := arr[b]
                        if a/2+7 >= b {
                                continue
                        }
                        if a < b {
                                continue
                        }
                        if a < 100 && 100 < b {
                                continue
                        }
                        res = res + countA*countB
                        if a == b {
                                res = res - countA
                        }
                }
        }
        return res
}
```

26.14 826. 安排工作以达到最大收益 (2)

- 题目

```
有一些工作：difficulty[i]表示第 i 个工作的难度，profit[i] 表示第 i 个工作的收益。
现在我们有一些工人。worker[i] 是第 i 个工人的能力，即该工人只能完成难度小于等于␣
↪worker[i] 的工作。
每一个工人都最多只能安排一个工作，但是一个工作可以完成多次。
举个例子，如果 3 个工人都尝试完成一份报酬为 1 的同样工作，那么总收益为 $3。
如果一个工人不能完成任何工作，他的收益为 $0 。
我们能得到的最大收益是多少？
示例：输入：difficulty = [2,4,6,8,10], profit = [10,20,30,40,50], worker = [4,5,6,7]
输出：100
```

(续下页)

(接上页)

```
解释: 工人被分配的工作难度是 [4,4,6,6] ，分别获得 [20,20,30,30] 的收益。
提示:1 <= difficulty.length = profit.length <= 10000
1 <= worker.length <= 10000
difficulty[i], profit[i], worker[i] 的范围是[1, 10^5]
```

- 解题思路

```
type Node struct {
        difficulty int
        profit     int
}

func maxProfitAssignment(difficulty []int, profit []int, worker []int) int {
        arr := make([]Node, 0)
        for i := 0; i < len(difficulty); i++ {
                arr = append(arr, Node{
                        difficulty: difficulty[i],
                        profit:     profit[i],
                })
        }
        sort.Ints(worker)
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].difficulty < arr[j].difficulty
        })
        res := 0
        index := 0
        maxProfit := 0
        for i := 0; i < len(worker); i++ {
                // 找到工人收益最大
                for index < len(arr) && worker[i] >= arr[index].difficulty {
                        maxProfit = max(maxProfit, arr[index].profit)
                        index++
                }
                res = res + maxProfit
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
func maxProfitAssignment(difficulty []int, profit []int, worker []int) int {
        res := 0
        arr := make([]int, 100001) // 难度对应的最大利润
        for i := 0; i < len(difficulty); i++ {
                arr[difficulty[i]] = max(arr[difficulty[i]], profit[i])
        }
        maxProfit := arr[0]
        for i := 1; i < 100001; i++ {
                maxProfit = max(maxProfit, arr[i])
                arr[i] = maxProfit
        }
        for i := 0; i < len(worker); i++ {
                res = res + arr[worker[i]]
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

26.15 831. 隐藏个人信息 (2)

- 题目

```
给你一条个人信息字符串 S，它可能是一个 邮箱地址 ，也可能是一串 电话号码 。
我们将隐藏它的隐私信息，通过如下规则：
1. 电子邮箱
定义名称 name 是长度大于等于 2 （length ≥ 2），并且只包含小写字母 a-z 和大写字母 A-Z␣
↪的字符串。
电子邮箱地址由名称 name 开头，紧接着是符号 '@'，后面接着一个名称 name，
再接着一个点号 '.'，然后是一个名称 name。
电子邮箱地址确定为有效的，并且格式是 "name1@name2.name3"。
为了隐藏电子邮箱，所有的名称 name 必须被转换成小写的，
并且第一个名称name 的第一个字母和最后一个字母的中间的所有字母由 5 个 '*' 代替。
2. 电话号码
电话号码是一串包括数字0-9，以及 {'+', '-', '(', ')', ''} 这几个字符的字符串。
```

(续下页)

(接上页)

```
你可以假设电话号码包含 10 到 13 个数字。
电话号码的最后 10 个数字组成本地号码，在这之前的数字组成国际号码。
注意，国际号码是可选的。我们只暴露最后 4 个数字并隐藏所有其他数字。
本地号码是有格式的，并且如 "***-***-1111" 这样显示，这里的 1 表示暴露的数字。
为了隐藏有国际号码的电话号码，像"+111 111 111 1111"，我们以 "+***-***-***-1111"␣
↪的格式来显示。
在本地号码前面的 '+' 号和第一个 '-' 号仅当电话号码中包含国际号码时存在。
例如，一个 12 位的电话号码应当以 "+**-" 开头进行显示。
注意：像 "(", ")", " "␣
↪这样的不相干的字符以及不符合上述格式的额外的减号或者加号都应当被删除。
最后，将提供的信息正确隐藏后返回。
示例 1：输入："LeetCode@LeetCode.com" 输出："l*****e@leetcode.com"
解释： 所有的名称转换成小写，第一个名称的第一个字符和最后一个字符中间由 5 个星号代替。
因此，"leetcode" -> "l*****e"。
示例 2：输入："AB@qq.com" 输出："a*****b@qq.com"
解释:第一个名称"ab"的第一个字符和最后一个字符的中间必须有 5 个星号
因此，"ab" -> "a*****b"。
示例 3：输入："1(234)567-890" 输出："***-***-7890"
解释:10 个数字的电话号码，那意味着所有的数字都是本地号码。
示例 4：输入："86-(10)12345678" 输出："+**-***-***-5678"
解释:12 位数字，2 个数字是国际号码另外 10 个数字是本地号码 。
注意:S.length<=40。
邮箱的长度至少是 8。
电话号码的长度至少是 10。
```

- 解题思路

```
func maskPII(S string) string {
        if strings.Contains(S, "@") {
                S = strings.ToLower(S)
                arr := strings.Split(S, "@")
                return arr[0][:1] + "*****" + arr[0][len(arr[0])-1:] + "@" + arr[1]
        }
        res := make([]byte, 0)
        for i := 0; i < len(S); i++ {
                if '0' <= S[i] && S[i] <= '9' {
                        res = append(res, S[i])
                }
        }
        if len(res) == 10 {
                return "***-***-" + string(res[len(res)-4:])
        } else if len(res) == 11 {
                return "+*-***-***-" + string(res[len(res)-4:])
        } else if len(res) == 12 {
```

(续下页)

(接上页)

```
                return "+**-***-***-" + string(res[len(res)-4:])
        } else if len(res) == 13 {
                return "+***-***-***-" + string(res[len(res)-4:])
        }
        return string(res)
}

# 2
func maskPII(S string) string {
        if strings.Contains(S, "@") {
                S = strings.ToLower(S)
                arr := strings.Split(S, "@")
                return arr[0][:1] + "*****" + arr[0][len(arr[0])-1:] + "@" + arr[1]
        }
        res := make([]byte, 0)
        for i := 0; i < len(S); i++ {
                if '0' <= S[i] && S[i] <= '9' {
                        res = append(res, S[i])
                }
        }
        n := len(res)
        str := "***-***-" + string(res[n-4:])
        if n > 10 {
                return "+" + strings.Repeat("*", n-10) + "-" + str
        }
        return str
}
```

26.16 833. 字符串中的查找与替换 (2)

- 题目

```
某个字符串 S 需要执行一些替换操作，用新的字母组替换原有的字母组（不一定大小相同）。
每个替换操作具有 3 个参数：起始索引 i，源字 x 和目标字 y。
规则是：如果 x 从原始字符串 S 中的位置 i 开始，那么就用 y 替换出现的␣
↪x。如果没有，则什么都不做。
举个例子，如果 S= “abcd” 并且替换操作 i = 2，x = “cd”，y = “ffff”，
那么因为 “cd” 从原始字符串 S 中的位置 2 开始，所以用 “ffff” 替换它。
再来看 S = “abcd” 上的另一个例子，如果一个替换操作 i = 0，x = “ab”，y = “eee”，
以及另一个替换操作 i = 2，x = “ec”，y = “ffff”，那么第二个操作将不会执行，
因为原始字符串中S[2] = 'c'，与 x[0] = 'e' 不匹配。
所有这些操作同时发生。保证在替换时不会有任何重叠：S = "abc",
```

(续下页)

(接上页)

```
indexes = [0, 1],sources = ["ab","bc"] 不是有效的测试用例。
示例 1: 输入: S = "abcd", indexes = [0,2], sources = ["a","cd"], targets = ["eee",
↪"ffff"]
输出: "eeebffff"
解释: "a" 从 S 中的索引 0 开始, 所以它被替换为 "eee"。
"cd" 从 S 中的索引 2 开始, 所以它被替换为 "ffff"。
示例 2: 输入: S = "abcd", indexes = [0,2], sources = ["ab","ec"], targets = ["eee",
↪"ffff"]
输出: "eeecd"
解释: "ab" 从 S 中的索引 0 开始, 所以它被替换为 "eee"。
"ec" 没有从原始的 S 中的索引 2 开始, 所以它没有被替换。
提示: 0 <= S.length <= 1000
S 仅由小写英文字母组成
0 <= indexes.length <= 100
0 <= indexes[i] < S.length
sources.length == indexes.length
targets.length == indexes.length
1 <= sources[i].length, targets[i].length <= 50
sources[i] 和 targets[i] 仅由小写英文字母组成
```

- 解题思路

```
type Node struct {
        index  int
        source string
        target string
}

func findReplaceString(S string, indexes []int, sources []string, targets []string)␣
↪string {
        arr := make([]Node, 0)
        for i := 0; i < len(indexes); i++ {
                arr = append(arr, Node{
                        index:  indexes[i],
                        source: sources[i],
                        target: targets[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].index < arr[j].index
        })
        res := ""
        left := 0
        for i := 0; i < len(arr); i++ {
```

(续下页)

(接上页)

```
                if left < arr[i].index {
                        res = res + S[left:arr[i].index]
                        left = arr[i].index
                }
                start := arr[i].index
                end := arr[i].index + len(arr[i].source)
                if end <= len(S) && S[start:end] == arr[i].source {
                        res = res + arr[i].target
                        left = end
                }
        }
        if left < len(S) {
                res = res + S[left:]
        }
        return res
}

# 2
func findReplaceString(S string, indexes []int, sources []string, targets []string)␣
↪string {
        m := make(map[int]int)
        for i := 0; i < len(indexes); i++ {
                if S[indexes[i]:indexes[i]+len(sources[i])] == sources[i] {
                        m[indexes[i]] = i
                }
        }
        res := make([]byte, 0)
        for i := 0; i < len(S); {
                if v, ok := m[i]; ok {
                        res = append(res, targets[v]...)
                        i = i + len(sources[v])
                } else {
                        res = append(res, S[i])
                        i++
                }
        }
        return string(res)
}
```

26.17 835. 图像重叠 (2)

- 题目

```
给你两个图像 img1 和 img2 ，两个图像的大小都是 n x n，用大小相同的二维正方形矩阵表示。
（并且为二进制矩阵，只包含若干 0 和若干 1 ）
转换其中一个图像，向左，右，上，或下滑动任何数量的单位，并把它放在另一个图像的上面。
之后，该转换的 重叠 是指两个图像都具有 1 的位置的数目。
（请注意，转换 不包括 向任何方向旋转。）
最大可能的重叠是多少？
示例 1：输入：img1 = [[1,1,0],[0,1,0],[0,1,0]], img2 = [[0,0,0],[0,1,1],[0,0,1]]输出：3
解释：将 img1 向右移动 1 个单位，再向下移动 1 个单位。
两个图像都具有 1 的位置的数目是 3（用红色标识）。
示例 2：输入：img1 = [[1]], img2 = [[1]] 输出：1
示例 3：输入：img1 = [[0]], img2 = [[0]] 输出：0
提示：n == img1.length
n == img1[i].length
n == img2.length
n == img2[i].length
1 <= n <= 30
img1[i][j] 为 0 或 1
img2[i][j] 为 0 或 1
```

- 解题思路

```
func largestOverlap(img1 [][]int, img2 [][]int) int {
        res := 0
        n := len(img1)
        arr := make([][]int, 2*n+1)
        for i := 0; i <= 2*n; i++ {
                arr[i] = make([]int, 2*n+1)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if img1[i][j] == 1 {
                                for a := 0; a < n; a++ {
                                        for b := 0; b < n; b++ {
                                                if img2[a][b] == 1 {
                                                        arr[i-a+n][j-b+n]++ // i-a, j-b是移动的，+n修正负数
                                                }
                                        }
```

(续下页)

(接上页)

```
                                }
                        }
                }
        }
        for i := 0; i <= 2*n; i++ {
                for j := 0; j <= 2*n; j++ {
                        res = max(res, arr[i][j])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func largestOverlap(img1 [][]int, img2 [][]int) int {
        res := 0
        n := len(img1)
        A := make([][2]int, 0)
        B := make([][2]int, 0)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if img1[i][j] == 1 {
                                A = append(A, [2]int{i, j})
                        }
                        if img2[i][j] == 1 {
                                B = append(B, [2]int{i, j})
                        }
                }
        }
        m := make(map[[2]int]int)
        for i := 0; i < len(A); i++ {
                for j := 0; j < len(B); j++ {
                        a := B[j][0] - A[i][0]
                        b := B[j][1] - A[i][1]
                        m[[2]int{a, b}]++
                        res = max(res, m[[2]int{a, b}])
                }
```

(续下页)

(接上页)

```
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

26.18 837. 新 21 点 (2)

- 题目

```
爱丽丝参与一个大致基于纸牌游戏 “21点” 规则的游戏，描述如下：
爱丽丝以 0 分开始，并在她的得分少于 K 分时抽取数字。
抽取时，她从 [1, W] 的范围中随机获得一个整数作为分数进行累计，其中 W 是整数。
每次抽取都是独立的，其结果具有相同的概率。
当爱丽丝获得不少于 K 分时，她就停止抽取数字。 爱丽丝的分数不超过 N 的概率是多少？
示例 1：输入：N = 10, K = 1, W = 10 输出：1.00000
说明：爱丽丝得到一张卡，然后停止。
示例 2：输入：N = 6, K = 1, W = 10 输出：0.60000
说明：爱丽丝得到一张卡，然后停止。
在 W = 10 的 6 种可能下，她的得分不超过 N = 6 分。
示例 3：输入：N = 21, K = 17, W = 10 输出：0.73278
提示：0 <= K <= N <= 10000
1 <= W <= 10000
如果答案与正确答案的误差不超过 10^-5，则该答案将被视为正确答案通过。
此问题的判断限制时间已经减少。
```

- 解题思路

```
func new21Game(N int, K int, W int) float64 {
        if K == 0 {
                return 1.0
        }
        dp := make([]float64, K+W) // 得分区间
        for i := K; i <= N && i < K+W; i++ {
                dp[i] = 1.0
        }
        /*for i := K-1; i >= 0; i--{
```

(续下页)

(接上页)

```
                for j := 1; j <= W; j++{ // 每次选择1~W
                        dp[i] = dp[i] + dp[i+j]/float64(W)
                }
        }*/
        dp[K-1] = 1.0 * (float64(min(N-K+1, W))) / float64(W)
        for i := K - 2; i >= 0; i-- {
                dp[i] = dp[i+1] - (dp[i+W+1]-dp[i+1])/float64(W)
        }
        return dp[0]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func new21Game(N int, K int, W int) float64 {
        if K == 0 {
                return 1.0
        }
        dp := make([]float64, K+W) // 为当前手中牌面为i点时获胜的概率
        var sum float64
        for i := K; i <= K+W-1; i++ {
                if i <= N {
                        dp[i] = 1
                }
                sum = sum + dp[i]
        }

        for i := K - 1; i >= 0; i-- {
                dp[i] = sum / float64(W)
                sum = sum - dp[i+W] + dp[i]
        }
        return dp[0]
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}
```

26.19 838. 推多米诺 (3)

- 题目

```
一行中有 N 张多米诺骨牌，我们将每张多米诺骨牌垂直竖立。
在开始时，我们同时把一些多米诺骨牌向左或向右推。
每过一秒，倒向左边的多米诺骨牌会推动其左侧相邻的多米诺骨牌。
同样地，倒向右边的多米诺骨牌也会推动竖立在其右侧的相邻多米诺骨牌。
如果同时有多米诺骨牌落在一张垂直竖立的多米诺骨牌的两边，由于受力平衡，
↪该骨牌仍然保持不变。
就这个问题而言，我们会认为正在下降的多米诺骨牌不会对其它正在下降或已经下降的多米诺骨牌施加额外的力。
给定表示初始状态的字符串 "S" 。如果第 i 张多米诺骨牌被推向左边，则 S[i] = 'L';
如果第 i 张多米诺骨牌被推向右边，则 S[i] = 'R'; 如果第 i 张多米诺骨牌没有被推动，则
↪S[i] = '.'。
返回表示最终状态的字符串。
示例 1: 输入: ".L.R...LR..L.." 输出: "LL.RR.LLRRLL.."
示例 2: 输入: "RR.L" 输出: "RR.L"
说明: 第一张多米诺骨牌没有给第二张施加额外的力。
提示: 0 <= N <= 10^5
表示多米诺骨牌状态的字符串只含有 'L', 'R'; 以及 '.';
```

- 解题思路

```
func pushDominoes(dominoes string) string {
        n := len(dominoes)
        arr := append([]byte{'L'}, dominoes...)
        arr = append(arr, 'R') // 前面补1个L，后面补1个R
        temp := make([]byte, n+2)
        for string(temp) != string(arr) { // 模拟每一秒：当没有变化的时候退出
                copy(temp, arr) // 存储之前1秒的情况
                for i := 1; i <= n; i++ {
                        if arr[i] == '.' { // 当前位置为.进行判断2边情况
                                // 讨论2种变的情况
                                if temp[i-1] == 'R' && temp[i+1] != 'L' {
                                        arr[i] = 'R' // 1、左边向右边倒，右边为R或者.
                                } else if temp[i+1] == 'L' && temp[i-1] != 'R' {
                                        arr[i] = 'L' // 2、右边向左边倒，左边为L或者.
                                }
                        }
```

(续下页)

(接上页)

```
                }
        }
        return string(arr[1 : n+1])
}

# 2
func pushDominoes(dominoes string) string {
        n := len(dominoes)
        arr := make([]int, n)
        value := 0
        for i := 0; i < n; i++ { // 1、从左往右
                // 计算左边的受力，R=n，L=0，.的时候随距离减1
                if dominoes[i] == 'R' {
                        value = n
                } else if dominoes[i] == 'L' {
                        value = 0
                } else {
                        value = max(0, value-1)
                }
                arr[i] = arr[i] + value
        }
        value = 0
        for i := n - 1; i >= 0; i-- { // 2、从右往左
                // 计算右边的受力，R=0，L=R，.的时候随距离减1
                if dominoes[i] == 'L' {
                        value = n
                } else if dominoes[i] == 'R' {
                        value = 0
                } else {
                        value = max(0, value-1)
                }
                arr[i] = arr[i] - value
        }
        res := []byte(dominoes)
        for i := 0; i < n; i++ {
                // 哪边受力大，结果随哪边
                if arr[i] > 0 {
                        res[i] = 'R'
                } else if arr[i] < 0 {
                        res[i] = 'L'
                } else {
                        res[i] = '.'
                }
```

(续下页)

(接上页)

```
        }
        return string(res)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func pushDominoes(dominoes string) string {
        s := ""
        for s != dominoes { // 模拟每一秒：当没有变化的时候退出
                s = dominoes
                dominoes = strings.ReplaceAll(dominoes, "R.L", "X") // 把R.
↪L这种不变的临时替换为1个不影响的X
                dominoes = strings.ReplaceAll(dominoes, ".L", "LL") // 向左倒
                dominoes = strings.ReplaceAll(dominoes, "R.", "RR") // 向右倒
                dominoes = strings.ReplaceAll(dominoes, "X", "R.L") // 替换回来
        }
        return dominoes
}
```

26.20 841. 钥匙和房间 (2)

- 题目

```
有 N 个房间，开始时你位于 0 号房间。每个房间有不同的号码：0，1，2，...，N-1，
并且房间里可能有一些钥匙能使你进入下一个房间。
在形式上，对于每个房间 i 都有一个钥匙列表 rooms[i]，
每个钥匙 rooms[i][j] 由 [0,1，...，N-1] 中的一个整数表示，其中 N = rooms.length。
钥匙 rooms[i][j] = v 可以打开编号为 v 的房间。
最初，除 0 号房间外的其余所有房间都被锁住。
你可以自由地在房间之间来回走动。
如果能进入每个房间返回 true，否则返回 false。
示例 1：输入：[[1],[2],[3],[]]输出：true
解释：  我们从 0 号房间开始，拿到钥匙 1。
之后我们去 1 号房间，拿到钥匙 2。
然后我们去 2 号房间，拿到钥匙 3。
最后我们去了 3 号房间。
```

(续下页)

(接上页)

```
由于我们能够进入每个房间，我们返回 true。
示例 2：输入：[[1,3],[3,0,1],[2],[0]]
输出：false
解释：我们不能进入 2 号房间。
提示：1 <= rooms.length <= 1000
    0 <= rooms[i].length <= 1000
    所有房间中的钥匙数量总计不超过 3000。
```

- 解题思路

```
var visited []bool
var total int

func canVisitAllRooms(rooms [][]int) bool {
        n := len(rooms)
        total = 0
        visited = make([]bool, n)
        dfs(rooms, 0)
        return total == n
}

func dfs(rooms [][]int, start int) {
        visited[start] = true
        total++
        for _, room := range rooms[start] {
                if visited[room] == false {
                        dfs(rooms, room)
                }
        }
}

# 2
func canVisitAllRooms(rooms [][]int) bool {
        n := len(rooms)
        total := 0
        visited := make([]bool, n)
        visited[0] = true
        queue := make([]int, 0)
        queue = append(queue, 0)
        for len(queue) > 0 {
                start := queue[0]
                queue = queue[1:]
                total++
                for _, room := range rooms[start] {
```

(续下页)

(接上页)

```
                        if visited[room] == false {
                                visited[room] = true
                                queue = append(queue, room)
                        }
                }
        }
        return total == n
}
```

26.21 842. 将数组拆分成斐波那契序列 (1)

- 题目

```
给定一个数字字符串 S，比如 S = "123456579"，我们可以将它分成斐波那契式的序列 [123,␣
↪456, 579]。
形式上，斐波那契式序列是一个非负整数列表 F，且满足：
0 <= F[i] <= 2^31 - 1，（也就是说，每个整数都符合 32 位有符号整数类型）；
F.length >= 3;
对于所有的0 <= i < F.length - 2，都有 F[i] + F[i+1] = F[i+2] 成立。
另外，请注意，将字符串拆分成小块时，每个块的数字一定不要以零开头，除非这个块是数字 0␣
↪本身。
返回从 S 拆分出来的任意一组斐波那契式的序列块，如果不能拆分则返回 []。
示例 1：输入："123456579" 输出：[123,456,579]
示例 2：输入："11235813" 输出：[1,1,2,3,5,8,13]
示例 3：输入："112358130" 输出：[]
解释：这项任务无法完成。
示例 4：输入："0123" 输出：[]
解释：每个块的数字不能以零开头，因此 "01"，"2"，"3" 不是有效答案。
示例 5：输入："1101111" 输出：[110, 1, 111]
解释：输出 [11,0,11,11] 也同样被接受。
提示：1 <= S.length<= 200
字符串 S 中只含有数字。
```

- 解题思路

```
var res []int

func splitIntoFibonacci(S string) []int {
        res = make([]int, 0)
        dfs(S, 0, 0, 0, make([]int, 0))
        return res
}
```

(续下页)

(接上页)

```
func dfs(s string, index, sum, prev int, path []int) bool {
        if index == len(s) {
                if len(path) >= 3 {
                        res = path
                }
                return len(path) >= 3
        }
        value := 0
        for i := index; i < len(s); i++ {
                // 0开头不满足要求(当前i=index的时候，可以为0，避免错过1+0=1的情况)
                if s[index] == '0' && i > index {
                        break
                }
                value = value*10 + int(s[i]-'0')
                if value > math.MaxInt32 {
                        break
                }
                if len(path) >= 2 {
                        if value < sum {
                                continue
                        }
                        if value > sum {
                                break
                        }
                }
                if dfs(s, i+1, prev+value, value, append(path, value)) == true {
                        return true
                }
        }
        return false
}
```

26.22 845. 数组中的最长山脉 (3)

- 题目

```
我们把数组 A 中符合下列属性的任意连续子数组 B 称为 “山脉”：
    B.length >= 3
    存在 0 < i < B.length - 1 使得
    B[0] < B[1] < ... B[i-1] < B[i] > B[i+1] > ... > B[B.length - 1]
（注意：B 可以是 A 的任意子数组，包括整个数组 A。）
```

(续下页)

(接上页)

```
给出一个整数数组 A，返回最长 “山脉” 的长度。
如果不含有 “山脉” 则返回 0。
示例 1：输入：[2,1,4,7,3,2,5] 输出：5
解释：最长的 “山脉” 是 [1,4,7,3,2]，长度为 5。
示例 2：输入：[2,2,2] 输出：0
解释：不含 “山脉”。
提示：0 <= A.length <= 10000
    0 <= A[i] <= 10000
```

- 解题思路

```
func longestMountain(A []int) int {
        n := len(A)
        left := make([]int, len(A))
        right := make([]int, len(A))
        for i := 1; i < n; i++ {
                if A[i-1] < A[i] {
                        left[i] = left[i-1] + 1
                }
        }
        for i := n - 2; i >= 0; i-- {
                if A[i+1] < A[i] {
                        right[i] = right[i+1] + 1
                }
        }
        res := 0
        for i := 1; i < n-1; i++ {
                if left[i] > 0 && right[i] > 0 {
                        res = max(res, left[i]+right[i]+1)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestMountain(A []int) int {
        n := len(A)
```

(续下页)

(接上页)

```
        left := 0
        res := 0
        for left+2 < n {
                // left指向左侧山脚, right寻找右侧山脚
                right := left + 1
                if A[left] < A[left+1] {
                        for right+1 < n && A[right] < A[right+1] {
                                right++
                        }
                        if right+1 < n && A[right] > A[right+1] {
                                for right+1 < n && A[right] > A[right+1] {
                                        right++
                                }
                                if right-left+1 > res {
                                        res = right - left + 1
                                }
                        } else {
                                right++
                        }
                }
                left = right
        }
        return res
}

# 3
func longestMountain(A []int) int {
        n := len(A)
        res := 0
        for i := 1; i < n-1; i++ {
                left, right := 0, 0
                for j := i - 1; j >= 0; j-- {
                        if A[j] < A[j+1] {
                                left++
                        } else {
                                break
                        }
                }
                for j := i + 1; j < n; j++ {
                        if A[j] < A[j-1] {
                                right++
                        } else {
                                break
```

(续下页)

(接上页)

```
                        }
                }
                if left > 0 && right > 0 {
                        res = max(res, left+right+1)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

26.23 846. 一手顺子 (3)

- 题目

```
爱丽丝有一手（hand）由整数数组给定的牌。
现在她想把牌重新排列成组，使得每个组的大小都是 W，且由 W 张连续的牌组成。
如果她可以完成分组就返回 true，否则返回 false。
注意：此题目与 1296 重复：
示例 1：输入：hand = [1,2,3,6,2,3,4,7,8], W = 3 输出：true
解释：爱丽丝的手牌可以被重新排列为 [1,2,3]，[2,3,4]，[6,7,8]。
示例 2：输入：hand = [1,2,3,4,5], W = 4 输出：false
解释：爱丽丝的手牌无法被重新排列成几个大小为 4 的组。
提示：1 <= hand.length <= 10000
0 <= hand[i]<= 10^9
1 <= W <= hand.length
```

- 解题思路

```
func isNStraightHand(hand []int, W int) bool {
        n := len(hand)
        if n%W != 0 {
                return false
        }
        if W == 1 {
                return true
        }
```

(续下页)

(接上页)

```
        sort.Ints(hand)
        for i := 0; i < n; i++ {
                if hand[i] >= 0 {
                        count := 1
                        for j := i + 1; j < n; j++ {
                                if hand[j] > hand[i]+count {
                                        break
                                }
                                if hand[j] >= 0 && hand[j] == hand[i]+count {
                                        hand[j] = -1
                                        count++
                                        if count == W {
                                                break
                                        }
                                }
                        }
                        if count != W {
                                return false
                        }
                        hand[i] = -1
                }
        }
        return true
}

# 2
func isNStraightHand(hand []int, W int) bool {
        n := len(hand)
        if n%W != 0 {
                return false
        }
        if W == 1 {
                return true
        }
        arr := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(hand); i++ {
                if m[hand[i]] == 0 {
                        arr = append(arr, hand[i])
                }
                m[hand[i]]++
        }
        sort.Ints(arr)
```

(续下页)

(接上页)

```
        for i := 0; i < len(arr); i++ {
                if m[arr[i]] > 0 {
                        for j := 1; j < W; j++ {
                                value := arr[i] + j
                                m[value] = m[value] - m[arr[i]]
                                if m[value] < 0 {
                                        return false
                                }
                        }
                }
        }
        return true
}

# 3
func isNStraightHand(hand []int, W int) bool {
        n := len(hand)
        if n%W != 0 {
                return false
        }
        if W == 1 {
                return true
        }
        m := make(map[int]int)
        for i := 0; i < len(hand); i++ {
                m[hand[i]]++
        }
        sort.Ints(hand)
        for i := 0; i < len(hand); i++ {
                value := m[hand[i]]
                if value > 0 {
                        for j := 0; j < W; j++ {
                                if m[hand[i]+j] < value {
                                        return false
                                }
                                m[hand[i]+j] = m[hand[i]+j] - value
                        }
                }
        }
        return true
}
```

26.24 848. 字母移位 (2)

- 题目

```
有一个由小写字母组成的字符串 S，和一个整数数组 shifts。
我们将字母表中的下一个字母称为原字母的 移位（由于字母表是环绕的， 'z'将会变成'a'）。
例如·，shift('a') = 'b', shift('t') = 'u',， 以及shift('z') = 'a'。
对于每个shifts[i] = x， 我们会将 S中的前i+1个字母移位x次。
返回将所有这些移位都应用到 S 后最终得到的字符串。
示例：输入：S = "abc", shifts = [3,5,9] 输出："rpl"
解释： 我们以 "abc" 开始。
将 S 中的第 1 个字母移位 3 次后，我们得到 "dbc"。
再将 S 中的前 2 个字母移位 5 次后，我们得到 "igc"。
最后将 S 中的这 3 个字母移位 9 次后，我们得到答案 "rpl"。
提示：1 <= S.length = shifts.length <= 20000
0 <= shifts[i] <= 10 ^ 9
```

- 解题思路

```
func shiftingLetters(S string, shifts []int) string {
        arr := []byte(S)
        shifts = append(shifts, 0)
        for i := len(S) - 1; i >= 0; i-- {
                shifts[i] = (shifts[i] + shifts[i+1]) % 26
                arr[i] = 'a' + (S[i]-'a'+byte(shifts[i]))%26
        }
        return string(arr)
}

# 2
func shiftingLetters(S string, shifts []int) string {
        sum := 0
        for i := 0; i < len(shifts); i++ {
                sum = (sum + shifts[i]) % 26
        }
        arr := []byte(S)
        for i := 0; i < len(S); i++ {
                count := int(S[i] - 'a')
                arr[i] = byte((count+sum)%26 + 'a')
                sum = ((sum-shifts[i])%26 + 26) % 26
        }
        return string(arr)
}
```

26.25 851. 喧闹和富有 (1)

- 题目

```
在一组 N 个人（编号为0, 1, 2, ..., N-
↪1）中，每个人都有不同数目的钱，以及不同程度的安静（quietness）。
为了方便起见，我们将编号为x的人简称为 "personx"。
如果能够肯定 personx比 persony更有钱的话，我们会说richer[i] = [x,␣
↪y]。注意richer可能只是有效观察的一个子集。
另外，如果 personx的安静程度为q，我们会说quiet[x] = q。
现在，返回答案answer，其中answer[x] = y的前提是，在所有拥有的钱不少于personx的人中，
persony是最安静的人（也就是安静值quiet[y]最小的人）。
示例：输入：richer = [[1,0],[2,1],[3,1],[3,7],[4,3],[5,3],[6,3]], quiet = [3,2,5,4,6,
↪1,7,0]
输出：[5,5,2,5,4,5,6,7]
解释： answer[0] = 5,
person 5 比 person 3 有更多的钱，person 3 比 person 1 有更多的钱，person 1 比 person␣
↪0 有更多的钱。
唯一较为安静（有较低的安静值 quiet[x]）的人是 person 7,
但是目前还不清楚他是否比 person 0 更有钱。
answer[7] = 7,
在所有拥有的钱肯定不少于 person 7 的人中(这可能包括 person 3, 4, 5, 6 以及 7),
最安静(有较低安静值 quiet[x])的人是 person 7。
其他的答案也可以用类似的推理来解释。
提示：1 <= quiet.length = N <= 500
0 <= quiet[i] < N, 所有quiet[i]都不相同。
0 <= richer.length <= N * (N-1) / 2
0 <= richer[i][j] < N
richer[i][0] != richer[i][1]
richer[i]都是不同的。
对richer的观察在逻辑上是一致的。
```

- 解题思路

```
var res []int

func loudAndRich(richer [][]int, quiet []int) []int {
        arr := make(map[int][]int)
        n := len(quiet)
        res = make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = -1
        }
        for i := 0; i < len(richer); i++ { // 有向无环图
```

(续下页)

(接上页)

```
                a, b := richer[i][0], richer[i][1]
                arr[b] = append(arr[b], a) // a比b更有钱
        }
        for i := 0; i < n; i++ {
                dfs(quiet, arr, i)
        }
        return res
}

func dfs(quiet []int, arr map[int][]int, i int) int {
        if res[i] == -1 {
                res[i] = i // 首先最安静的人等于本身
                for j := 0; j < len(arr[i]); j++ {
                        next := dfs(quiet, arr, arr[i][j]) //␣
↪递归找到arr[i][j]对应的最安静的人
                        if quiet[next] < quiet[res[i]] {
                                res[i] = next
                        }
                }
        }
        return res[i]
}
```

26.26 853. 车队 (1)

- 题目

```
N 辆车沿着一条车道驶向位于target英里之外的共同目的地。
每辆车i以恒定的速度speed[i] (英里/小时)，从初始位置position[i] (英里) ␣
↪沿车道驶向目的地。
一辆车永远不会超过前面的另一辆车，但它可以追上去，并与前车以相同的速度紧接着行驶。
此时，我们会忽略这两辆车之间的距离，也就是说，它们被假定处于相同的位置。
车队是一些由行驶在相同位置、具有相同速度的车组成的非空集合。注意，一辆车也可以是一个车队。
即便一辆车在目的地才赶上了一个车队，它们仍然会被视作是同一个车队。
会有多少车队到达目的地?
示例: 输入: target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3] 输出: 3
解释:  从 10 和 8 开始的车会组成一个车队，它们在 12 处相遇。
从 0 处开始的车无法追上其它车，所以它自己就是一个车队。
从 5 和 3 开始的车会组成一个车队，它们在 6 处相遇。
请注意，在到达目的地之前没有其它车会遇到这些车队，所以答案是 3。
提示: 0 <= N <= 10 ^ 4
0 < target<= 10 ^ 6
```

(续下页)

(接上页)

```
0 <speed[i] <= 10 ^ 6
0 <= position[i] < target
所有车的初始位置各不相同。
```

- 解题思路

```
type Node struct {
        Position int
        Left     float64
}

func carFleet(target int, position []int, speed []int) int {
        if len(position) == 0 {
                return 0
        }
        arr := make([]Node, 0)
        for i := 0; i < len(position); i++ {
                arr = append(arr, Node{
                        Position: position[i],
                        Left:     float64(target-position[i]) / float64(speed[i]),
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].Position > arr[j].Position
        })
        res := 1
        prev := arr[0].Left
        for i := 1; i < len(arr); i++ {
                if prev < arr[i].Left {
                        res++
                        prev = arr[i].Left
                }
        }
        return res
}
```

26.27 855. 考场就座

26.27.1 题目

```
在考场里，一排有N个座位，分别编号为0, 1, 2, ..., N-1。
当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。
如果有多个这样的座位，他会坐在编号最小的座位上。(另外，如果考场里没有人，那么学生就坐在
↪0 号座位上。)
返回ExamRoom(int N)类，它有两个公开的函数：其中，函数ExamRoom.
↪seat()会返回一个int（整型数据），代表学生坐的位置；
函数ExamRoom.leave(int p)代表坐在座位 p 上的学生现在离开了考场。
每次调用ExamRoom.leave(p)时都保证有学生坐在座位p上。
示例：输入：["ExamRoom","seat","seat","seat","seat","leave","seat"], [[10],[],[],[],
↪[],[4],[]]
输出：[null,0,9,4,2,null,5]
解释：ExamRoom(10) -> null
seat() -> 0，没有人在考场里，那么学生坐在 0 号座位上。
seat() -> 9，学生最后坐在 9 号座位上。
seat() -> 4，学生最后坐在 4 号座位上。
seat() -> 2，学生最后坐在 2 号座位上。
leave(4) -> null
seat() -> 5，学生最后坐在 5 号座位上。
提示：1 <= N <= 10^9
在所有的测试样例中ExamRoom.seat()和ExamRoom.leave()最多被调用10^4次。
保证在调用ExamRoom.leave(p)时有学生正坐在座位 p 上。
```

26.27.2 解题思路

```

```

26.28 856. 括号的分数 (3)

- 题目

```
给定一个平衡括号字符串S，按下述规则计算该字符串的分数：
() 得 1 分。
AB 得A + B分，其中 A 和 B 是平衡括号字符串。
(A) 得2 * A分，其中 A 是平衡括号字符串。
示例 1：输入： "()" 输出： 1
示例 2：输入： "(())" 输出： 2
```

(续下页)

(接上页)

```
示例3: 输入: "()()" 输出: 2
示例4: 输入: "((()()))" 输出: 6
提示: S是平衡括号字符串, 且只含有(和)。
2 <= S.length <= 50
```

- 解题思路

```
func scoreOfParentheses(S string) int {
        res := 0
        count := 0
        for i := 0; i < len(S); i++ {
                if S[i] == '(' {
                        count++
                } else {
                        count--
                        if S[i-1] == '(' {
                                res = res + 1<<count
                        }
                }
        }
        return res
}

# 2
func scoreOfParentheses(S string) int {
        stack := make([]int, 0)
        stack = append(stack, 0)
        for i := 0; i < len(S); i++ {
                if S[i] == '(' {
                        stack = append(stack, 0)
                } else {
                        a, b := stack[len(stack)-1], stack[len(stack)-2]
                        stack = stack[:len(stack)-2]
                        stack = append(stack, b+max(2*a, 1))
                }
        }
        return stack[0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

# 3
func scoreOfParentheses(S string) int {
        return dfs(S, 0, len(S))
}

func dfs(S string, left, right int) int {
        res := 0
        count := 0
        for i := left; i < right; i++ {
                if S[i] == '(' {
                        count++
                } else {
                        count--
                }
                if count == 0 {
                        if i-left == 1 {
                                res++
                        } else {
                                res = res + 2*dfs(S, left+1, i)
                        }
                        left = i + 1
                }
        }
        return res
}
```

26.29 858. 镜面反射

26.29.1 题目

```
有一个特殊的正方形房间，每面墙上都有一面镜子。除西南角以外，每个角落都放有一个接受器，编号为0，1，以
正方形房间的墙壁长度为p，一束激光从西南角射出，首先会与东墙相遇，入射点到接收器 0
↪的距离为 q 。
返回光线最先遇到的接收器的编号（保证光线最终会遇到一个接收器）。
示例：输入： p = 2, q = 1 输出： 2
解释： 这条光线在第一次被反射回左边的墙时就遇到了接收器 2 。
提示：1 <= p <= 1000
0 <= q <= p
```

26.29.2 解题思路

26.30 861. 翻转矩阵后的得分 (2)

- 题目

```
有一个二维矩阵A 其中每个元素的值为0或1。
移动是指选择任一行或列，并转换该行或列中的每一个值：将所有 0 都更改为 1，将所有 1都更改为 0。
在做出任意次数的移动后，将该矩阵的每一行都按照二进制数来解释，矩阵的得分就是这些数字的总和。
返回尽可能高的分数。
示例：输入：[[0,0,1,1],[1,0,1,0],[1,1,0,0]] 输出：39
解释：转换为 [[1,1,1,1],[1,0,0,1],[1,1,1,1]]
0b1111 + 0b1001 + 0b1111 = 15 + 9 + 15 = 39
提示：1 <= A.length <= 20
1 <= A[0].length <= 20
A[i][j]是0 或1
```

- 解题思路

```
func matrixScore(A [][]int) int {
        var res int
        if len(A) == 0 || len(A[0]) == 0 {
                return 0
        }
        // 翻转行，每行第一个为0则翻转
        for i := 0; i < len(A); i++ {
                if A[i][0] == 0 {
                        for j := 0; j < len(A[i]); j++ {
                                A[i][j] = 1 - A[i][j]
                        }
                }
        }
        // 翻转列,每列1的数量大于0则翻转
        for j := 0; j < len(A[0]); j++ {
                a, b := 0, 0
                for i := 0; i < len(A); i++ {
                        if A[i][j] == 0 {
                                a++
                        } else {
                                b++
```

(续下页)

(接上页)

```
                        }
                }
                if a <= b {
                        continue
                }
                for i := 0; i < len(A); i++ {
                        A[i][j] = 1 - A[i][j]
                }
        }
        for i := 0; i < len(A); i++ {
                sum := 0
                for j := 0; j < len(A[i]); j++ {
                        sum = sum*2 + A[i][j]
                }
                res = res + sum
        }
        return res
}

# 2
func matrixScore(A [][]int) int {
        var res int
        if len(A) == 0 || len(A[0]) == 0 {
                return 0
        }
        n := len(A)
        m := len(A[0])
        // 首先每行第一个都为1求和，n个长度为m的1x...x
        // 这样保证第一列全为1
        res = res + n*(1<<(m-1))
        for j := 1; j < m; j++ {
                a, b := 0, 0
                for i := 0; i < n; i++ {
                        if A[i][0] == 0 && A[i][j] == 0 { // 需要翻转
                                b++
                        } else if A[i][0] == 1 && A[i][j] == 1 { // 不需要翻转
                                b++
                        } else {
                                a++
                        }
                }
                // 1比0多，不需要翻转
                if a <= b {
```

(续下页)

(接上页)

```
                        res = res + b*(1<<(m-1-j))
                } else {
                        res = res + a*(1<<(m-1-j))
                }
        }
        return res
}
```

26.31 863. 二叉树中所有距离为 K 的结点 (1)

- 题目

```
给定一个二叉树（具有根结点root），一个目标结点target，和一个整数值 K 。
返回到目标结点 target 距离为 K 的所有结点的值的列表。 答案可以以任何顺序返回。
示例 1：输入：root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, K = 2 输出：[7,4,1]
解释：所求结点为与目标结点（值为 5）距离为 2 的结点，
值分别为 7，4，以及 1
注意，输入的 "root" 和 "target" 实际上是树上的结点。
上面的输入仅仅是对这些对象进行了序列化描述。
提示：给定的树是非空的。
树上的每个结点都具有唯一的值0 <= node.val <= 500。
目标结点target是树上的结点。
0 <= K <= 1000.
```

- 解题思路

```
var m map[int]*TreeNode // 存储值对应的父节点
var res []int

func distanceK(root *TreeNode, target *TreeNode, K int) []int {
        m = make(map[int]*TreeNode)
        res = make([]int, 0)
        dfs(root)                    // 生成值对应的父节点
        findDfs(K, target, nil, 0) // 遍历
        return res
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        if root.Left != nil {
```

(续下页)

(接上页)

```
                m[root.Left.Val] = root
                dfs(root.Left)
        }
        if root.Right != nil {
                m[root.Right.Val] = root
                dfs(root.Right)
        }
}

func findDfs(K int, node *TreeNode, prev *TreeNode, dis int) {
        if node == nil {
                return
        }
        if dis == K {
                res = append(res, node.Val)
                return
        }
        if node.Left != prev { // 防止重复
                findDfs(K, node.Left, node, dis+1)
        }
        if node.Right != prev { // 防止重复
                findDfs(K, node.Right, node, dis+1)
        }
        if m[node.Val] != prev { // 防止重复：搜索父节点
                findDfs(K, m[node.Val], node, dis+1)
        }
}
```

26.32 865. 具有所有最深节点的最小子树 (2)

- 题目

```
给定一个根为root的二叉树，每个节点的深度是 该节点到根的最短距离 。
如果一个节点在 整个树 的任意节点之间具有最大的深度，则该节点是 最深的 。
一个节点的 子树 是该节点加上它的所有后代的集合。
返回能满足 以该节点为根的子树中包含所有最深的节点 这一条件的具有最大深度的节点。
注意：本题与力扣 1123 重复：
示例 1：输入：root = [3,5,1,6,2,0,8,null,null,7,4] 输出：[2,7,4]
解释： 我们返回值为 2 的节点，在图中用黄色标记。
在图中用蓝色标记的是树的最深的节点。
注意，节点 5、3 和 2 包含树中最深的节点，但节点 2 的子树最小，因此我们返回它。
示例 2：输入：root = [1] 输出：[1]
```

(续下页)

(接上页)

```
解释：根节点是树中最深的节点。
示例 3：输入：root = [0,1,3,null,2] 输出：[2]
解释：树中最深的节点为 2 ，有效子树为节点 2、1 和 0 的子树，但节点 2 的子树最小。
提示：树中节点的数量介于1 和500 之间。
0 <= Node.val <= 500
每个节点的值都是独一无二的。
```

- 解题思路

```
func subtreeWithAllDeepest(root *TreeNode) *TreeNode {
        res, _ := dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) (*TreeNode, int) {
        if root == nil {
                return root, level
        }
        leftNode, left := dfs(root.Left, level+1)
        rightNode, right := dfs(root.Right, level+1)
        if left == right {
                return root, left + 1
        } else if left > right {
                return leftNode, left + 1
        }
        return rightNode, right + 1
}

# 2
func subtreeWithAllDeepest(root *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        if left == right {
                return root
        } else if left > right {
                return subtreeWithAllDeepest(root.Left)
        }
        return subtreeWithAllDeepest(root.Right)
}

func dfs(root *TreeNode) int {
```

(续下页)

(接上页)

```
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        return 1 + max(left, right)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

26.33 866. 回文素数 (1)

- 题目

```
求出大于或等于N的最小回文素数。
回顾一下，如果一个数大于 1，且其因数只有 1 和它自身，那么这个数是素数。
例如，2，3，5，7，11 以及13 是素数。
回顾一下，如果一个数从左往右读与从右往左读是一样的，那么这个数是回文数。
例如，12321 是回文数。
示例 1：输入：6 输出：7
示例2：输入：8 输出：11
示例3：输入：13 输出：101
提示：1 <= N <= 10^8
答案肯定存在，且小于2 * 10^8。
```

- 解题思路

```
func primePalindrome(N int) int {
    if 8 <= N && N <= 11 { // 特判：偶数位的回文数都可以被11整除
                return 11
        }
        for i := 1; i < 100000; i++ {
                s := strconv.Itoa(i)
                temp := []byte(s)
                for i := 0; i < len(s)/2; i++ {
                        temp[i], temp[len(s)-1-i] = temp[len(s)-1-i], temp[i]
                }
```

(续下页)

(接上页)

```
                target, _ := strconv.Atoi(s + string(temp[1:]))
                if target >= N && isPrime(target) {
                        return target
                }
        }
        return -1
}

func isPrime(n int) bool {
        if n < 2 {
                return false
        }
        for i := 2; i*i <= n; i++ {
                if n%i == 0 {
                        return false
                }
        }
        return true
}
```

26.34 869. 重新排序得到 2 的幂 (1)

- 题目

```
给定正整数 N，我们按任何顺序（包括原始顺序）将数字重新排序，注意其前导数字不能为零。
如果我们可以通过上述方式得到2 的幂，返回 true；否则，返回 false。
示例 1：输入：1 输出：true
示例 2：输入：10 输出：false
示例 3：输入：16 输出：true
示例 4：输入：24 输出：false
示例 5：输入：46 输出：true
提示：1 <= N <= 10^9
```

- 解题思路

```
func reorderedPowerOf2(N int) bool {
        arr := getCount(N)
        for i := 0; i < 31; i++ {
                if arr == getCount(1<<i) {
                        return true
                }
        }
```

(续下页)

(接上页)

```
        return false
}

func getCount(n int) [10]int {
        arr := [10]int{}
        for n > 0 {
                arr[n%10]++
                n = n / 10
        }
        return arr
}
```

26.35 870. 优势洗牌 (1)

- 题目

```
给定两个大小相等的数组 A 和 B，A 相对于 B 的优势可以用满足 A[i] > B[i] 的索引 i␣
↪的数目来描述。
返回 A 的任意排列，使其相对于 B 的优势最大化。
示例 1：输入：A = [2,7,11,15], B = [1,10,4,11] 输出：[2,11,7,15]
示例 2：输入：A = [12,24,8,32], B = [13,25,32,11] 输出：[24,32,8,12]
提示：  1 <= A.length = B.length <= 10000
    0 <= A[i] <= 10^9
    0 <= B[i] <= 10^9
```

- 解题思路

```
func advantageCount(A []int, B []int) []int {
        res := make([]int, len(A))
        sort.Ints(A)
        arr := make([][2]int, 0)
        for i := 0; i < len(B); i++ {
                arr = append(arr, [2]int{i, B[i]})
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] < arr[j][1]
        })
        left, right := 0, len(A)-1
        for i := 0; i < len(A); i++ {
                if A[i] > arr[left][1] { // 满足条件放前面
                        index := arr[left][0]
                        left++
```

(续下页)

(接上页)

```
                        res[index] = A[i]
                } else { // 不满足条件放后面
                        index := arr[right][0]
                        right--
                        res[index] = A[i]
                }
        }
        return res
}
```

26.36 873. 最长的斐波那契子序列的长度 (2)

- 题目

```
如果序列X_1, X_2, ..., X_n满足下列条件，就说它是斐波那契式的：
n >= 3
对于所有i + 2 <= n，都有X_i + X_{i+1} = X_{i+2}
给定一个严格递增的正整数数组形成序列，找到 A␣
↪中最长的斐波那契式的子序列的长度。如果一个不存在，返回0 。
（回想一下，子序列是从原序列 A中派生出来的，它从 A中删掉任意数量的元素（也可以不删），
而不改变其余元素的顺序。例如，[3, 5, 8]是[3, 4, 5, 6, 7, 8]的一个子序列）
示例 1： 输入：[1,2,3,4,5,6,7,8] 输出：5
解释:最长的斐波那契式子序列为：[1,2,3,5,8] 。
示例2： 输入：[1,3,7,11,12,14,18] 输出：3
解释:最长的斐波那契式子序列有：
[1,11,12]，[3,11,14] 以及 [7,11,18] 。
提示：3 <= A.length <= 1000
1 <= A[0] < A[1] < ... < A[A.length - 1] <= 10^9
（对于以 Java，C，C++，以及C# 的提交，时间限制被减少了 50%）
```

- 解题思路

```
func lenLongestFibSubseq(arr []int) int {
        n := len(arr)
        m := make(map[int]bool)
        for i := 0; i < n; i++ {
                m[arr[i]] = true
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        count := 2
```

(续下页)

(接上页)

```
                        a, b := arr[i], arr[j]
                        for m[a+b] == true {
                                count++
                                a, b = b, a+b
                        }
                        if count > res && count > 2 {
                                res = count
                        }
                }
        }
        return res
}

# 2
func lenLongestFibSubseq(arr []int) int {
        n := len(arr)
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                m[arr[i]] = i
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        dp[i][j] = 2
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
                        index, ok := m[arr[i]-arr[j]]
                        if ok && arr[index] < arr[j] {
                                dp[j][i] = dp[index][j] + 1
                                if dp[j][i] > 2 && dp[j][i] > res {
                                        res = dp[j][i]
                                }
                        }
                }
        }
        return res
}
```

26.37 875. 爱吃香蕉的珂珂 (2)

- 题目

```
珂珂喜欢吃香蕉。这里有 N 堆香蕉，第 i 堆中有 piles[i] 根香蕉。警卫已经离开了，将在 H␣
↪小时后回来。
珂珂可以决定她吃香蕉的速度 K （单位：根/
↪小时）。每个小时，她将会选择一堆香蕉，从中吃掉 K 根。
如果这堆香蕉少于 K 根，她将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉。
珂珂喜欢慢慢吃，但仍然想在警卫回来前吃掉所有的香蕉。
返回她可以在 H 小时内吃掉所有香蕉的最小速度 K（K 为整数）。
示例 1：输入：piles = [3,6,7,11], H = 8 输出：4
示例 2：输入：piles = [30,11,23,4,20], H = 5 输出：30
示例 3：输入：piles = [30,11,23,4,20], H = 6 输出：23
提示：1 <= piles.length <= 10^4
    piles.length <= H <= 10^9
    1 <= piles[i] <= 10^9
```

- 解题思路

```
func minEatingSpeed(piles []int, H int) int {
        maxValue := piles[0]
        for i := 1; i < len(piles); i++ {
                maxValue = max(maxValue, piles[i])
        }
        left, right := 1, maxValue
        for left < right {
                mid := left + (right-left)/2
                if judge(piles, mid, H) == true {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

func judge(piles []int, speed int, H int) bool {
        total := 0
        for i := 0; i < len(piles); i++ {
                total = total + piles[i]/speed
                if piles[i]%speed > 0 {
                        total = total + 1
                }
```

(续下页)

(接上页)

```
        }
        return total > H
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minEatingSpeed(piles []int, h int) int {
        maxValue := piles[0]
        for i := 1; i < len(piles); i++ {
                maxValue = max(maxValue, piles[i])
        }
        return sort.Search(maxValue, func(speed int) bool {
                if speed == 0 { // 避免除0
                        return false
                }
                total := 0
                for i := 0; i < len(piles); i++ {
                        total = total + piles[i]/speed
                        if piles[i]%speed > 0 {
                                total = total + 1
                        }
                }
                return total <= h
        })

}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

26.38 877. 石子游戏 (3)

- 题目

```
亚历克斯和李用几堆石子在做游戏。偶数堆石子排成一行，每堆都有正整数颗石子 piles[i] 。
游戏以谁手中的石子最多来决出胜负。石子的总数是奇数，所以没有平局。
亚历克斯和李轮流进行，亚历克斯先开始。 每回合，玩家从行的开始或结束处取走整堆石头。
这种情况一直持续到没有更多的石子堆为止，此时手中石子最多的玩家获胜。
假设亚历克斯和李都发挥出最佳水平，当亚历克斯赢得比赛时返回 true ，当李赢得比赛时返回␣
↪false 。
示例：输入：[5,3,4,5] 输出：true
解释：亚历克斯先开始，只能拿前 5 颗或后 5 颗石子 。
假设他取了前 5 颗，这一行就变成了 [3,4,5] 。
如果李拿走前 3 颗，那么剩下的是 [4,5]，亚历克斯拿走后 5 颗赢得 10 分。
如果李拿走后 5 颗，那么剩下的是 [3,4]，亚历克斯拿走后 4 颗赢得 9 分。
这表明，取前 5 颗石子对亚历克斯来说是一个胜利的举动，所以我们返回 true 。
提示：2 <= piles.length <= 500
    piles.length 是偶数。
    1 <= piles[i] <= 500
    sum(piles) 是奇数。
```

- 解题思路

```
func stoneGame(piles []int) bool {
        dp := make([]int, len(piles))
        for i := 0; i < len(piles); i++ {
                dp[i] = piles[i]
        }
        for i := len(piles) - 2; i >= 0; i-- {
                for j := i + 1; j < len(piles); j++ {
                        dp[j] = max(piles[i]-dp[j], piles[j]-dp[j-1])
                }
        }
        return dp[len(piles)-1] >= 0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
func stoneGame(piles []int) bool {
        n := len(piles)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                dp[i][i] = piles[i]
        }
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        // 玩家得分: 自己得分-对手得分
                        dp[i][j] = max(piles[i]-dp[i+1][j], piles[j]-dp[i][j-1])
                }
        }
        return dp[0][n-1] >= 0
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func stoneGame(piles []int) bool {
        return true
}
```

26.39 880. 索引处的解码字符串 (2)

- 题目

```
给定一个编码字符串 S。请你找出 解码字符串 并将其写入磁带。
解码时，从编码字符串中 每次读取一个字符 ，并采取以下步骤:
如果所读的字符是字母，则将该字母写在磁带上。
如果所读的字符是数字 (例如 d)，则整个当前磁带总共会被重复写d-1 次。
现在，对于给定的编码字符串 S 和索引 K，查找并返回解码字符串中的第K个字母。
示例 1: 输入: S = "leet2code3", K = 10 输出: "o"
解释: 解码后的字符串为 "leetleetcodeleetleetcodeleetleetcode"。
字符串中的第 10 个字母是 "o"。
示例 2: 输入: S = "ha22", K = 5 输出: "h"
解释: 解码后的字符串为 "hahahaha"。第 5 个字母是 "h"。
```

(续下页)

(接上页)

```
示例 3: 输入: S = "a2345678999999999999999", K = 1 输出: "a"
解释: 解码后的字符串为 "a" 重复 8301530446056247680 次。第 1 个字母是 "a"。
提示: 2 <= S.length <= 100
S只包含小写字母与数字 2 到 9 。
S以字母开头。
1 <= K <= 10^9
题目保证 K 小于或等于解码字符串的长度。
解码后的字符串保证少于2^63个字母。
```

- 解题思路

```
func decodeAtIndex(S string, K int) string {
        count := 0 // 字符个数
        n := len(S)
        for i := 0; i < n; i++ { // 先统计最终字符串总长数
                if '0' <= S[i] && S[i] <= '9' {
                        value := int(S[i] - '0')
                        count = count * value // 重写d-1次, 长度是原来的d倍
                } else {
                        count++ // 非数字, 长度+1
                }
        }
        for i := n - 1; i >= 0; i-- {
                K = K % count                                   // 缩小范围
                if K == 0 && 'a' <= S[i] && S[i] <= 'z' { // 找到目标字符
                        return string(S[i])
                }
                if '0' <= S[i] && S[i] <= '9' { // 范围缩小
                        value := int(S[i] - '0')
                        count = count / value
                } else {
                        count-- // 长度-1
                }
        }
        return ""
}

# 2
func decodeAtIndex(S string, K int) string {
        count := 0 // 字符个数
        n := len(S)
        for i := 0; i < n; i++ { // 先统计最终字符串总长数
                if '0' <= S[i] && S[i] <= '9' {
                        value := int(S[i] - '0')
```

(续下页)

(接上页)

```
                        prev := count
                        count = count * value // 重写d-1次，长度是原来的d倍
                        if count >= K {
                                return decodeAtIndex(S[:i], (K-1)%prev+1) //␣
↪缩小范围：避免求余出现0的情况
                        }
                } else {
                        count++ // 非数字，长度+1
                        if count == K {
                                return string(S[i])
                        }
                }
        }
        return ""
}
```

26.40 881. 救生艇 (1)

- 题目

```
第 i 个人的体重为 people[i]，每艘船可以承载的最大重量为 limit。
每艘船最多可同时载两人，但条件是这些人的重量之和最多为 limit。
返回载到每一个人所需的最小船数。(保证每个人都能被船载)。
示例 1：输入：people = [1,2]，limit = 3输出：1 解释：1 艘船载 (1, 2)
示例 2：输入：people = [3,2,2,1]，limit = 3 输出：3
解释：3 艘船分别载 (1, 2)，(2) 和 (3)
示例 3：输入：people = [3,5,3,4]，limit = 5 输出：4
解释：4 艘船分别载 (3)，(3)，(4)，(5)
提示：1 <= people.length <= 50000
    1 <= people[i] <= limit <= 30000
```

- 解题思路

```
func numRescueBoats(people []int, limit int) int {
        res := 0
        sort.Ints(people)
        i, j := 0, len(people)-1
        for i <= j {
                if people[i]+people[j] <= limit {
                        i++
                        j--
                } else {
```

(续下页)

(接上页)

```
                        j--
                }
                res++
        }
        return res
}
```

26.41 885. 螺旋矩阵 III(2)

- 题目

```
在R行C列的矩阵上，我们从(r0, c0)面朝东面开始
这里，网格的西北角位于第一行第一列，网格的东南角位于最后一行最后一列。
现在，我们以顺时针按螺旋状行走，访问此网格中的每个位置。
每当我们移动到网格的边界之外时，我们会继续在网格之外行走（但稍后可能会返回到网格边界）。
最终，我们到过网格的所有R * C个空间。
按照访问顺序返回表示网格位置的坐标列表。
示例 1：输入：R = 1, C = 4, r0 = 0, c0 = 0 输出：[[0,0],[0,1],[0,2],[0,3]]
示例 2：输入：R = 5, C = 6, r0 = 1, c0 = 4 输出：[[1,4],[1,5],[2,5],[2,4],[2,3],[1,3],
↪[0,3],[0,4],
[0,5],[3,5],[3,4],[3,3],[3,2],[2,2],[1,2],[0,2],[4,5],
[4,4],[4,3],[4,2],[4,1],[3,1],[2,1],[1,1],[0,1],[4,0],[3,0],[2,0],[1,0],[0,0]]
提示：1 <= R <= 100
1 <= C <= 100
0 <= r0 < R
0 <= c0 < C
```

- 解题思路

```
// 顺时针：上右下左
// 本题：右、下、左、上
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func spiralMatrixIII(rows int, cols int, rStart int, cStart int) [][]int {
        res := make([][]int, 0)
        total := rows * cols
        x, y := rStart, cStart
        res = append(res, []int{x, y})
        index := 1
        if total == 1 {
                return res
```

(续下页)

(接上页)

```
        }
        for k := 1; k < 2*(rows+cols); k = k + 2 {
                for i := 0; i < 4; i++ {
                        step := k + i/2 // 本次移动的步数，分别是2次的倍数
                        for j := 0; j < step; j++ {
                                x = x + dx[i]
                                y = y + dy[i]
                                if 0 <= x && x < rows && 0 <= y && y < cols {
                                        res = append(res, []int{x, y})
                                        index++
                                        if index == total {
                                                return res
                                        }
                                }
                        }

                }
        }
        return res
}

# 2
// 顺时针：上右下左
// 本题：右、下、左、上
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func spiralMatrixIII(rows int, cols int, rStart int, cStart int) [][]int {
        res := make([][]int, 0)
        total := rows * cols
        x, y := rStart, cStart
        index := 0
        step := 2
        dir := 0
        for index < total {
                for i := 0; i < step/2; i++ { // 本次移动的步数
                        if 0 <= x && x < rows && 0 <= y && y < cols {
                                res = append(res, []int{x, y})
                                index++
                                if index == total {
                                        return res
                                }
                        }
```

(续下页)

(接上页)

```
                        x = x + dx[dir]
                        y = y + dy[dir]
                }
                dir = (dir + 1) % 4
                step++
        }
        return res
}
```

26.42 886. 可能的二分法 (3)

- 题目

```
给定一组N人（编号为1, 2, ..., N），我们想把每个人分进任意大小的两组。
每个人都可能不喜欢其他人，那么他们不应该属于同一组。
形式上，如果 dislikes[i] = [a, b]，表示不允许将编号为 a 和 b 的人归入同一组。
当可以用这种方法将所有人分进两组时，返回 true；否则返回 false。
示例 1：输入：N = 4, dislikes = [[1,2],[1,3],[2,4]] 输出：true
解释：group1 [1,4], group2 [2,3]
示例 2：输入：N = 3, dislikes = [[1,2],[1,3],[2,3]] 输出：false
示例 3：输入：N = 5, dislikes = [[1,2],[2,3],[3,4],[4,5],[1,5]] 输出：false
提示：1 <= N <= 2000
0 <= dislikes.length <= 10000
dislikes[i].length == 2
1 <= dislikes[i][j] <= N
dislikes[i][0] < dislikes[i][1]
对于 dislikes[i] == dislikes[j] 不存在 i != j
```

- 解题思路

```
var arr [][]int
var m map[int]int

func possibleBipartition(n int, dislikes [][]int) bool {
        m = make(map[int]int) // 分组： 0一组，1一组
        arr = make([][]int, n+1)
        for i := 0; i < len(dislikes); i++ {
                a, b := dislikes[i][0], dislikes[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        for i := 1; i <= n; i++ {
```

(续下页)

(接上页)

```
                // 没有被分配过，分配到0一组
                if _, ok := m[i]; ok == false && dfs(i, 0) == false {
                        return false
                }
        }
        return true
}

func dfs(index int, value int) bool {
        if v, ok := m[index]; ok {
                return v == value // 已经分配，查看是否同一组
        }
        m[index] = value
        for i := 0; i < len(arr[index]); i++ {
                target := arr[index][i]
                if dfs(target, 1-value) == false { // 不喜欢的人，分配到对立组：1-
↪value
                        return false
                }
        }
        return true
}

# 2
func possibleBipartition(n int, dislikes [][]int) bool {
        m := make(map[int]int) // 分组： 0一组，1一组
        arr := make([][]int, n+1)
        for i := 0; i < len(dislikes); i++ {
                a, b := dislikes[i][0], dislikes[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        for i := 1; i <= n; i++ {
                // 没有被分配过，分配到0一组
                if _, ok := m[i]; ok == true {
                        continue
                }
                m[i] = 0
                queue := make([]int, 0)
                queue = append(queue, i)
                for len(queue) > 0 {
                        node := queue[0]
                        queue = queue[1:]
```

(续下页)

(接上页)

```
                        for i := 0; i < len(arr[node]); i++ {
                                target := arr[node][i]
                                if _, ok := m[target]; ok == false {
                                        m[target] = 1 - m[node] // 相反一组
                                        queue = append(queue, target)
                                } else if m[node] == m[target] { //␣
↪已经分配，查看是否同一组
                                        return false
                                }
                        }
                }
        }
        return true
}

# 3
func possibleBipartition(n int, dislikes [][]int) bool {
        fa = Init(n + 1)
        arr := make([][]int, n+1)
        for i := 0; i < len(dislikes); i++ {
                a, b := dislikes[i][0], dislikes[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        for i := 1; i <= n; i++ {
                for j := 0; j < len(arr[i]); j++ {
                        target := arr[i][j]
                        if find(i) == find(target) { // 和不喜欢的人在相同组，失败
                                return false
                        }
                        union(arr[i][0], target) // 不喜欢的人在同一组
                }
        }
        return true
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
```

(续下页)

(接上页)

```
        }
        return arr
}

// 查询
func find(x int) int {
        for x != fa[x] {
                fa[x] = fa[fa[x]]
                x = fa[x]
        }
        return x
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}
```

26.43 889. 根据前序和后序遍历构造二叉树 (1)

- 题目

```
返回与给定的前序和后序遍历匹配的任何二叉树。
pre和post遍历中的值是不同的正整数。
示例：输入：pre = [1,2,4,5,3,6,7], post = [4,5,2,6,7,3,1] 输出：[1,2,3,4,5,6,7]
提示：1 <= pre.length == post.length <= 30
pre[]和post[]都是1, 2, ..., pre.length的排列
每个输入保证至少有一个答案。如果有多个答案，可以返回其中一个。
```

- 解题思路

```
func constructFromPrePost(pre []int, post []int) *TreeNode {
        if len(pre) == 0 {
                return nil
        }
        root := &TreeNode{
                Val: pre[0],
        }
        if len(pre) == 1 {
                return root
        }
        index := len(pre)
```

(续下页)

(接上页)

```
        for i := 0; i < len(post); i++ {
                if post[i] == pre[1] {
                        index = i
                        break
                }
        }
        root.Left = constructFromPrePost(pre[1:index+2], post[:index+1])
        root.Right = constructFromPrePost(pre[index+2:], post[index+1:])
        return root
}
```

26.44 890. 查找和替换模式 (2)

- 题目

```
你有一个单词列表words和一个模式pattern，你想知道 words 中的哪些单词与模式匹配。
如果存在字母的排列 p，使得将模式中的每个字母 x 替换为 p(x) 之后，
我们就得到了所需的单词，那么单词与模式是匹配的。
（回想一下，字母的排列是从字母到字母的双射：每个字母映射到另一个字母，没有两个字母映射到同一个字母。）
返回 words 中与给定模式匹配的单词列表。
你可以按任何顺序返回答案。
示例：输入：words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"
输出：["mee","aqq"]
解释："mee" 与模式匹配，因为存在排列 {a -> m, b -> e, ...}。
"ccc" 与模式不匹配，因为 {a -> c, b -> c, ...} 不是排列。
因为 a 和 b 映射到同一个字母。
提示：1 <= words.length <= 50
1 <= pattern.length = words[i].length<= 20
```

- 解题思路

```
func findAndReplacePattern(words []string, pattern string) []string {
        res := make([]string, 0)
        for i := 0; i < len(words); i++ {
                m1, m2 := make(map[byte]byte), make(map[byte]byte)
                flag := true
                for j := 0; j < len(pattern); j++ {
                        x, y := pattern[j], words[i][j]
                        a, ok1 := m1[x]
                        b, ok2 := m2[y]
                        if ok1 == false && ok2 == false {
                                m1[x] = y
```

(续下页)

(接上页)

```
                                m2[y] = x
                        }
                        if (ok1 == true && ok2 == false) || (ok1 == false && ok2 ==␣
↪true) ||
                                (ok1 == true && ok2 == true && (a != y || b != x)) {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = append(res, words[i])
                }
        }
        return res
}

# 2
func findAndReplacePattern(words []string, pattern string) []string {
        res := make([]string, 0)
        for i := 0; i < len(words); i++ {
                m1, m2 := make(map[byte]byte), make(map[byte]byte)
                flag := true
                for j := 0; j < len(pattern); j++ {
                        x, y := pattern[j], words[i][j]
                        if m1[x] == 0 && m2[y] == 0 {
                                m1[x] = y
                                m2[y] = x
                        } else if (m1[x] == y && m2[y] == x) == false {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = append(res, words[i])
                }
        }
        return res
}
```

26.45 894. 所有可能的满二叉树 (3)

- 题目

```
满二叉树是一类二叉树，其中每个结点恰好有 0 或 2 个子结点。
返回包含 N 个结点的所有可能满二叉树的列表。 答案的每个元素都是一个可能树的根结点。
答案中每个树的每个结点都必须有 node.val=0。
你可以按任何顺序返回树的最终列表。
示例：输入：7
输出：[[0,0,0,null,null,0,0,null,null,0,0],[0,0,0,null,null,0,0,0,0],
[0,0,0,0,0,0,0],[0,0,0,0,0,null,null,null,null,0,0],[0,0,0,0,0,null,null,0,0]]
解释：
提示：1 <= N <= 20
```

- 解题思路

```
var res map[int][]*TreeNode

func allPossibleFBT(n int) []*TreeNode {
        res = make(map[int][]*TreeNode)
        dfs(n)
        return res[n]
}

func dfs(n int) []*TreeNode {
        if _, ok := res[n]; !ok {
                arr := make([]*TreeNode, 0)
                if n == 1 {
                        arr = append(arr, &TreeNode{Val: 0})
                } else if n%2 == 1 { // N只为奇数
                        for i := 0; i < n; i++ {
                                j := n - 1 - i
                                for _, left := range dfs(i) {
                                        for _, right := range dfs(j) {
                                                root := &TreeNode{Val: 0}
                                                root.Left = left
                                                root.Right = right
                                                arr = append(arr, root)
                                        }
                                }
                        }
                }
                res[n] = arr
        }
```

(续下页)

(接上页)

```
        return res[n]
}

# 2
func allPossibleFBT(n int) []*TreeNode {
        res := make(map[int][]*TreeNode)
        res[1] = []*TreeNode{{Val: 0}}
        for index := 3; index <= n; index = index + 2 {
                for i := 1; i < index; i = i + 2 {
                        j := index - 1 - i
                        for _, left := range res[i] {
                                for _, right := range res[j] {
                                        res[index] = append(res[index], &TreeNode{
                                                Val:   0,
                                                Left:  left,
                                                Right: right,
                                        })
                                }
                        }
                }
        }
        return res[n]
}

# 3
func allPossibleFBT(n int) []*TreeNode {
        if n == 1 {
                return []*TreeNode{{Val: 0}}
        }
        res := make([]*TreeNode, 0)
        for i := 1; i < n; i = i + 2 {
                leftResult := allPossibleFBT(i)
                rightResult := allPossibleFBT(n - 1 - i)
                for _, left := range leftResult {
                        for _, right := range rightResult {
                                res = append(res, &TreeNode{
                                        Val:   0,
                                        Left:  left,
                                        Right: right,
                                })
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

26.46 898. 子数组按位或操作 (2)

- 题目

```
我们有一个非负整数数组A。
对于每个（连续的）子数组B =[A[i], A[i+1], ..., A[j]] （i <= j），
我们对B中的每个元素进行按位或操作，获得结果A[i] | A[i+1] | ... | A[j]。
返回可能结果的数量。 （多次出现的结果在最终答案中仅计算一次。）
示例 1: 输入: [0] 输出: 1
解释: 只有一个可能的结果 0 。
示例 2: 输入: [1,1,2] 输出: 3
解释: 可能的子数组为 [1], [1], [2], [1, 1], [1, 2], [1, 1, 2]。
产生的结果为 1, 1, 2, 1, 3, 3 。
有三个唯一值, 所以答案是 3 。
示例3: 输入: [1,2,4] 输出: 6
解释: 可能的结果是 1, 2, 3, 4, 6, 以及 7 。
提示: 1 <= A.length <= 50000
0 <= A[i] <= 10^9
```

- 解题思路

```
func subarrayBitwiseORs(arr []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(arr); i++ {
                m[arr[i]] = true
                temp := 0
                for j := i - 1; j >= 0; j-- {
                        if temp|arr[i] == temp { // 都为1, 进行下去无意义, 避免超时
                                break
                        }
                        temp = temp | arr[j]
                        m[temp|arr[i]] = true
                }
        }
        return len(m)
}

# 2
func subarrayBitwiseORs(arr []int) int {
```

(续下页)

(接上页)

```
        m := make(map[int]bool)
        for i := 0; i < len(arr); i++ {
                m[arr[i]] = true
                for j := i - 1; j >= 0; j-- {
                        if arr[j]|arr[i] == arr[j] {
                                break
                        }
                        arr[j] = arr[j] | arr[i]
                        m[arr[j]] = true
                }
        }
        return len(m)
}
```

26.47 900.RLE 迭代器 (2)

- 题目

```
编写一个遍历游程编码序列的迭代器。
迭代器由 RLEIterator(int[] A) 初始化，其中A是某个序列的游程编码。
更具体地，对于所有偶数 i，A[i] 告诉我们在序列中重复非负整数值 A[i + 1] 的次数。
迭代器支持一个函数：next(int n)，它耗尽接下来的 n 个元素 (n >=␣
↪1) 并返回以这种方式耗去的最后一个元素。
如果没有剩余的元素可供耗尽，则 next返回-1 。
例如，我们以A = [3,8,0,9,2,5]开始，这是序列[8,8,8,5,5]的游程编码。
这是因为该序列可以读作 “三个八，零个九，两个五”。
示例：输入：["RLEIterator","next","next","next","next"], [[[3,8,0,9,2,5]],[2],[1],[1],
↪[2]]
输出：[null,8,8,5,-1]
解释：RLEIterator 由 RLEIterator([3,8,0,9,2,5]) 初始化。
这映射到序列 [8,8,8,5,5]。
然后调用 RLEIterator.next 4次。
.next(2) 耗去序列的 2 个项，返回 8。现在剩下的序列是 [8, 5, 5]。
.next(1) 耗去序列的 1 个项，返回 8。现在剩下的序列是 [5, 5]。
.next(1) 耗去序列的 1 个项，返回 5。现在剩下的序列是 [5]。
.next(2) 耗去序列的 2 个项，返回 -1。 这是由于第一个被耗去的项是 5，
但第二个项并不存在。由于最后一个要耗去的项不存在，我们返回 -1。
提示：0 <= A.length <= 1000
A.length是偶数。
0 <= A[i] <= 10^9
每个测试用例最多调用1000次RLEIterator.next(int n)。
每次调用RLEIterator.next(int n)都有1 <= n <= 10^9。
```

- 解题思路

```
type RLEIterator struct {
        arr   []int
        index int // 第几个元素
        count int // 元素的第几次
}

func Constructor(encoding []int) RLEIterator {
        return RLEIterator{
                arr:   encoding,
                index: 0,
                count: 0,
        }
}

func (this *RLEIterator) Next(n int) int {
        for this.index < len(this.arr) {
                if n+this.count > this.arr[this.index] {
                        n = n - (this.arr[this.index] - this.count)
                        this.count = 0
                        this.index = this.index + 2
                } else {
                        this.count = this.count + n
                        return this.arr[this.index+1]
                }
        }
        return -1
}

# 2
type RLEIterator struct {
        arr    []int
        total  int
        values []int
        cur    int
}

func Constructor(encoding []int) RLEIterator {
        total := 0
        arr := make([]int, 0) // 前缀和
        values := make([]int, 0)
        for i := 0; i < len(encoding); i = i + 2 {
                if encoding[i] == 0 {
                        continue
```

(续下页)

(接上页)

```
                }
                total = total + encoding[i]
                arr = append(arr, total)
                values = append(values, encoding[i+1])
        }
        return RLEIterator{
                arr:    arr,
                total:  total,
                cur:    0,
                values: values,
        }
}

func (this *RLEIterator) Next(n int) int {
        target := this.cur + n
        this.cur = target
        if target > this.total {
                return -1
        }
        left, right := 0, len(this.arr)
        for left < right {
                mid := left + (right-left)/2
                if this.arr[mid] < target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return this.values[left]
}
```

CHAPTER 27

0801-0900-Hard

27.1 810. 黑板异或游戏 (1)

- 题目

```
黑板上写着一个非负整数数组 nums[i] 。Alice 和 Bob 轮流从黑板上擦掉一个数字，Alice␣
↪先手。
如果擦除一个数字后，剩余的所有数字按位异或运算得出的结果等于 0␣
↪的话，当前玩家游戏失败。
(另外，如果只剩一个数字，按位异或运算得到它本身；如果无数字剩余，按位异或运算结果为0。)
并且，轮到某个玩家时，如果当前黑板上所有数字按位异或运算结果等于 0，这个玩家获胜。
假设两个玩家每步都使用最优解，当且仅当 Alice 获胜时返回 true。
示例：输入：nums = [1, 1, 2] 输出：false
解释： Alice 有两个选择：擦掉数字 1 或 2。
如果擦掉 1，数组变成 [1, 2]。剩余数字按位异或得到 1 XOR 2 = 3。
那么 Bob 可以擦掉任意数字，因为 Alice 会成为擦掉最后一个数字的人，她总是会输。
如果 Alice 擦掉 2，那么数组变成[1, 1]。剩余数字按位异或得到 1 XOR 1 = 0。Alice␣
↪仍然会输掉游戏。
提示：1 <= N <= 1000
0 <= nums[i] <= 2^16
```

- 解题思路

```
func xorGame(nums []int) bool {
        n := len(nums)
```

(续下页)

(接上页)

```
        res := 0
        for i := 0; i < n; i++ {
                res = res ^ nums[i]
        }
        return res == 0 || n%2 == 0
}
```

27.2 815. 公交路线 (2)

- 题目

```
给你一个数组 routes ，表示一系列公交线路，其中每个 routes[i] 表示一条公交线路，第 i␣
↪辆公交车将会在上面循环行驶。
例如，路线 routes[0] = [1, 5, 7]
表示第 0 辆公交车会一直按序列 1 -> 5 -> 7 -> 1 -> 5 -> 7 -> 1 -> ...␣
↪这样的车站路线行驶。
现在从 source 车站出发（初始时不在公交车上），要前往 target 车站。␣
↪期间仅可乘坐公交车。
求出 最少乘坐的公交车数量 。如果不可能到达终点车站，返回 -1 。
示例 1：输入：routes = [[1,2,7],[3,6,7]], source = 1, target = 6 输出：2
解释：最优策略是先乘坐第一辆公交车到达车站 7 ，然后换乘第二辆公交车到车站 6 。
示例 2：输入：routes = [[7,12],[4,5,15],[6],[15,19],[9,12,13]], source = 15, target =␣
↪12 输出：-1
提示：1 <= routes.length <= 500.
1 <= routes[i].length <= 105
routes[i] 中的所有值 互不相同
sum(routes[i].length) <= 105
0 <= routes[i][j] < 106
0 <= source, target < 106
```

- 解题思路

```
func numBusesToDestination(routes [][]int, source int, target int) int {
        if source == target {
                return 0
        }
        n := len(routes)
        m := make(map[int][]int) // 该公交车经过第几条线路的数组
        for i := 0; i < n; i++ {
                for j := 0; j < len(routes[i]); j++ {
                        node := routes[i][j]
                        m[node] = append(m[node], i)
```

(续下页)

(接上页)

```
                }
        }
        count := 1
        queue := make([]int, 0)
        queue = append(queue, source)
        visited := make(map[int]bool) // 第几条线路访问情况
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        node := queue[i]
                        for j := 0; j < len(m[node]); j++ {
                                if visited[m[node][j]] == false {
                                        visited[m[node][j]] = true
                                        for k := 0; k < len(routes[m[node][j]]); k++ {
                                                if routes[m[node][j]][k] == target {
                                                        return count
                                                }
                                                queue = append(queue,␣
↪routes[m[node][j]][k])
                                        }
                                }
                        }
                }
                count++
                queue = queue[length:]
        }
        return -1
}

# 2
func numBusesToDestination(routes [][]int, source int, target int) int {
        if source == target {
                return 0
        }
        n := len(routes)
        m := make(map[int][]int) // 该公交车经过第几条线路的数组
        dis := make([]int, n)    // source到第i条线路的距离
        arr := make([][]bool, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]bool, n)
                dis[i] = -1
        }
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < len(routes[i]); j++ {
                        node := routes[i][j]
                        m[node] = append(m[node], i)
                        for _, v := range m[node] {
                                arr[i][v] = true
                                arr[v][i] = true
                        }
                }
        }
        queue := make([]int, 0)
        for _, v := range m[source] {
                dis[v] = 1
                queue = append(queue, v)
        }
        for len(queue) > 0 { // 广度优先计算source所在公交站台，到其它站台的最小距离
                node := queue[0]
                queue = queue[1:]
                for i := 0; i < n; i++ {
                        if arr[node][i] == true && dis[i] == -1 { //␣
↪node可以到i，且i未访问过
                                dis[i] = dis[node] + 1
                                queue = append(queue, i)
                        }
                }
        }
        res := math.MaxInt32                    //  最短路径
        for i := 0; i < len(m[target]); i++ { // 遍历多终点，找最小
                v := m[target][i]
                if dis[v] != -1 && dis[v] < res {
                        res = dis[v]
                }
        }
        if res < math.MaxInt32 {
                return res
        }
        return -1
}
```

27.3 827. 最大人工岛

27.3.1 题目

```
给你一个大小为 n x n 二进制矩阵 grid 。最多 只能将一格0 变成1 。
返回执行此操作后，grid 中最大的岛屿面积是多少？
岛屿 由一组上、下、左、右四个方向相连的1 形成。
示例 1:输入: grid = [[1, 0], [0, 1]] 输出: 3
解释: 将一格0变成1，最终连通两个小岛得到面积为 3 的岛屿。
示例 2:输入: grid = [[1, 1], [1, 0]] 输出: 4
解释: 将一格0变成1，岛屿的面积扩大为 4。
示例 3:输入: grid = [[1, 1], [1, 1]] 输出: 4
解释: 没有0可以让我们变成1，面积依然为 4。
提示：n == grid.length
n == grid[i].length
1 <= n <= 500
grid[i][j] 为 0 或 1
```

27.3.2 解题思路

```

```

27.4 828. 统计子串中的唯一字符 (4)

- 题目

```
我们定义了一个函数 countUniqueChars(s) 来统计字符串 s␣
↪中的唯一字符，并返回唯一字符的个数。
例如：s = "LEETCODE" ，则其中 "L", "T","C","O","D"␣
↪都是唯一字符，因为它们只出现一次，所以 countUniqueChars(s) = 5 。
本题将会给你一个字符串 s ，我们需要返回 countUniqueChars(t) 的总和，其中 t 是 s␣
↪的子字符串。
注意，某些子字符串可能是重复的，但你统计时也必须算上这些重复的子字符串（也就是说，你必须统计␣
↪s 的所有子字符串中的唯一字符）。
由于答案可能非常大，请将结果 mod 10 ^ 9 + 7 后再返回。
示例 1：输入：s = "ABC" 输出：10
解释：所有可能的子串为："A","B","C","AB","BC" 和 "ABC"。
     其中，每一个子串都由独特字符构成。
     所以其长度总和为：1 + 1 + 1 + 2 + 2 + 3 = 10
示例 2：输入：s = "ABA" 输出：8
```

(续下页)

(接上页)

```
解释: 除了 countUniqueChars("ABA") = 1 之外, 其余与示例 1 相同。
示例 3: 输入: s = "LEETCODE" 输出: 92
提示: 0 <= s.length <= 10^4
s 只包含大写英文字符
```

- 解题思路

```
var mod = 1000000007

func uniqueLetterString(s string) int {
        res := 0
        var j, k int
        n := len(s)
        for i := 0; i < n; i++ { // 计算当前字符 作为 唯一字符 的子串的次数
                for j = i - 1; 0 <= j && s[i] != s[j]; j-- {
                }
                for k = i + 1; k < n && s[i] != s[k]; k++ {
                }
                res = (res + (i-j)*(k-i)) % mod // 总数: left * right
        }
        return res
}

# 2
var mod = 1000000007
var cur [26]int

func uniqueLetterString(s string) int {
        res := 0
        n := len(s)
        cur = [26]int{}    // 记录每个字符的下标进度
        arr := [26][]int{} // 记录每个字符的所有下标列表
        for i := 0; i < n; i++ {
                index := int(s[i] - 'A')
                arr[index] = append(arr[index], i)
        }
        sum := 0 // 26种字符, 作为唯一字符的总次数
        for i := 0; i < 26; i++ {
                arr[i] = append(arr[i], n, n) // 补2个n
                sum = sum + getDiff(arr, i)
        }
        for i := 0; i < n; i++ {
                res = (res + sum) % mod
                index := int(s[i] - 'A')
```

(续下页)

(接上页)

```
                prev := getDiff(arr, index)
                cur[index]++
                sum = sum + getDiff(arr, index) - prev // 更新次数
        }
        return res
}

func getDiff(arr [26][]int, index int) int { // 第i+1个和第i个下标的距离
        i := cur[index]
        return arr[index][i+1] - arr[index][i]
}

# 3
var mod = 1000000007

func uniqueLetterString(s string) int {
        res := 0
        n := len(s)
        arr := [26][]int{} // 记录每个字符的所有下标列表
        for i := 0; i < n; i++ {
                index := int(s[i] - 'A')
                arr[index] = append(arr[index], i)
        }
        for i := 0; i < 26; i++ {
                for j := 0; j < len(arr[i]); j++ {
                        var prev, next int
                        cur := arr[i][j]
                        if j == 0 {
                                prev = -1
                        } else {
                                prev = arr[i][j-1]
                        }
                        if j == len(arr[i])-1 {
                                next = n
                        } else {
                                next = arr[i][j+1]
                        }
                        res = (res + (cur-prev)*(next-cur)) % mod
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 4
var mod = 1000000007

func uniqueLetterString(s string) int {
        res := 0
        n := len(s)
        arr := [26][2]int{} // 记录每个字符的最后2个字符下标
        for i := 0; i < 26; i++ {
                arr[i] = [2]int{-1, -1}
        }
        sum := 0
        for i := 0; i < n; i++ {
                index := int(s[i] - 'A')
                cur := i - arr[index][1]
                prev := arr[index][1] - arr[index][0]
                sum = sum + cur - prev
                res = (res + sum) % mod
                // 更新下标
                arr[index][0] = arr[index][1]
                arr[index][1] = i
        }
        return res
}
```

27.5 829. 连续整数求和 (4)

- 题目

```
给定一个正整数 N，试求有多少组连续正整数满足所有数字之和为 N?
示例 1:输入: 5 输出: 2
解释: 5 = 5 = 2 + 3，共有两组连续整数([5],[2,3])求和后为 5。
示例 2:输入: 9 输出: 3
解释: 9 = 9 = 4 + 5 = 2 + 3 + 4
示例 3:输入: 15 输出: 4
解释: 15 = 15 = 8 + 7 = 4 + 5 + 6 = 1 + 2 + 3 + 4 + 5
说明:1 <= N <= 10 ^ 9
```

- 解题思路

```
func consecutiveNumbersSum(N int) int {
        res := 1
        sum := 0
```

(续下页)

(接上页)

```
        for i := 1; i < N; i++ {
                sum = sum + i // 1~i
                left := N - sum
                if left > 0 {
                        if left%(i+1) == 0 { // 划分为i+1个数
                                res++
                        }
                } else {
                        break
                }
        }
        return res
}

# 2
func consecutiveNumbersSum(N int) int {
        res := 1
        for i := 1; ; i++ {
                N = N - i
                if N > 0 {
                        if N%(i+1) == 0 { // 划分为i+1个数
                                res++
                        }
                } else {
                        break
                }
        }
        return res
}

# 3
func consecutiveNumbersSum(N int) int {
        res := 1
        // N=(x+1)+(x+2)+…+(x+k) = kx+k*(k+1)/2
        // 2N=k(2x+k+1)
        target := int(math.Sqrt(float64(2 * N)))
        for i := 1; i < target; i++ {
                left := N - i*(i+1)/2
                if left%(i+1) == 0 { // 长度i+1
                        res++
                }
        }
        return res
```

(续下页)

(接上页)

```
}

# 4
func consecutiveNumbersSum(N int) int {
        res := 0
        i := 1
        for N > 0 {
                if N%i == 0 {
                        res++
                }
                N = N - i
                i++
        }
        return res
}
```

27.6 834. 树中距离之和

27.6.1 题目

```
给定一个无向、连通的树。树中有 N 个标记为 0...N-1 的节点以及 N-1条边。
第 i 条边连接节点edges[i][0] 和 edges[i][1]。
返回一个表示节点 i 与其他所有节点距离之和的列表 ans。
示例 1:输入: N = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]] 输出: [8,12,6,10,10,10]
解释:
如下为给定的树的示意图:
  0
 / \
1   2
   /|\
  3 4 5
我们可以计算出 dist(0,1) + dist(0,2) + dist(0,3) + dist(0,4) + dist(0,5)
也就是 1 + 1 + 2 + 2 + 2 = 8。 因此，answer[0] = 8，以此类推。
说明:1 <= N <= 10000
```

27.6.2 解题思路

27.7 839. 相似字符串组 (1)

- 题目

```
如果交换字符串X 中的两个不同位置的字母，使得它和字符串Y 相等，那么称 X 和 Y
↪两个字符串相似。
如果这两个字符串本身是相等的，那它们也是相似的。
例如，"tars" 和 "rats" 是相似的 (交换 0 与 2 的位置)；
"rats" 和 "arts" 也是相似的，但是 "star" 不与 "tars"，"rats"，或 "arts" 相似。
总之，它们通过相似性形成了两个关联组：{"tars", "rats", "arts"} 和 {"star"}。
注意，"tars" 和 "arts" 是在同一组中，即使它们并不相似。
形式上，对每个组而言，要确定一个单词在组中，只需要这个词和该组中至少一个单词相似。
给你一个字符串列表 strs。列表中的每个字符串都是 strs
↪中其它所有字符串的一个字母异位词。请问 strs 中有多少个相似字符串组？
示例 1：输入：strs = ["tars","rats","arts","star"] 输出：2
示例 2：输入：strs = ["omv","ovm"] 输出：1
提示：1 <= strs.length <= 300
1 <= strs[i].length <= 300
strs[i] 只包含小写字母。
strs 中的所有单词都具有相同的长度，且是彼此的字母异位词。
备注：字母异位词 (anagram)，一种把某个字符串的字母的位置 (顺序) 加以改换所形成的新词。
```

- 解题思路

```
func numSimilarGroups(strs []string) int {
        n := len(strs)
        fa = Init(n)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if judge(strs[i], strs[j]) == true { // 满足条件，连通
                                union(i, j)
                        }
                }
        }
        return count
}

func judge(a, b string) bool {
        if a == b {
```

(续下页)

(接上页)

```
                return true
        }
        count := 0
        for i := 0; i < len(a); i++ {
                if a[i] != b[i] {
                        count++
                        if count > 2 {
                                return false
                        }
                }
        }
        return true
}

var fa []int
var count int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        count = n
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
```

(续下页)

(接上页)

```
        }
}
```

27.8 857. 雇佣 K 名工人的最低成本 (1)

- 题目

```
有 N名工人。第i名工人的工作质量为quality[i]，其最低期望工资为wage[i]。
现在我们想雇佣K名工人组成一个工资组。在雇佣一组 K␣
↪名工人时，我们必须按照下述规则向他们支付工资：
对工资组中的每名工人，应当按其工作质量与同组其他工人的工作质量的比例来支付工资。
工资组中的每名工人至少应当得到他们的最低期望工资。
返回组成一个满足上述条件的工资组至少需要多少钱。
示例 1：输入： quality = [10,20,5], wage = [70,50,30], K = 2 输出： 105.00000
解释： 我们向 0 号工人支付 70，向 2 号工人支付 35。
示例 2：输入： quality = [3,1,10,10,1], wage = [4,8,2,2,7], K = 3 输出： 30.66667
解释： 我们向 0 号工人支付 4，向 2 号和 3 号分别支付 13.33333。
提示：1 <= K <= N <= 10000，其中N = quality.length = wage.length
1 <= quality[i] <= 10000
1 <= wage[i] <= 10000
与正确答案误差在10^-5之内的答案将被视为正确的。
```

- 解题思路

```
func mincostToHireWorkers(quality []int, wage []int, k int) float64 {
        n := len(quality)
        arr := make([][2]int, n)
        for i := 0; i < n; i++ {
                arr[i] = [2]int{quality[i], wage[i]}
        }
        sort.Slice(arr, func(i, j int) bool {
                a := float64(arr[i][1]) / float64(arr[i][0])
                b := float64(arr[j][1]) / float64(arr[j][0])
                return a < b
        })
        res := math.MaxFloat64
        var sum float64
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        //␣
↪枚举最低时薪或者性价比，然后在能接受最低时薪的人中选择工作质量总和最小的k个人
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                cur := float64(arr[i][1]) / float64(arr[i][0]) // 性价比: 最低工资/
↪质量, 比率从小到大遍历
                heap.Push(&intHeap, arr[i][0])                 // 质量高优先淘汰
                sum = sum + float64(arr[i][0])                 // 质量和
                if intHeap.Len() > k {
                        node := heap.Pop(&intHeap).(int)
                        sum = sum - float64(node)
                }
                if intHeap.Len() == k && cur*sum < res {
                        res = cur * sum
                }
        }
        return res
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

27.9 862. 和至少为 K 的最短子数组 (2)

- 题目

```
返回 A 的最短的非空连续子数组的长度，该子数组的和至少为 K 。
如果没有和至少为K的非空子数组，返回-1。
示例 1：输入：A = [1], K = 1 输出：1
示例 2：输入：A = [1,2], K = 4 输出：-1
示例 3：输入：A = [2,-1,2], K = 3 输出：3
提示：1 <= A.length <= 50000
-10 ^ 5<= A[i] <= 10 ^ 5
1 <= K <= 10 ^ 9
```

- 解题思路

```
func shortestSubarray(nums []int, k int) int {
        res := math.MaxInt32
        n := len(nums)
        arr := make([]int, n+1)
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        queue := make([]int, 0) // 递增队列：队首坐标小，队尾坐标大
        for i := 0; i <= n; i++ {
                for len(queue) > 0 && arr[i] <= arr[queue[len(queue)-1]] { //␣
↪前缀和小于队尾值：出队
                        queue = queue[:len(queue)-1]
                }
                for len(queue) > 0 && arr[i]-arr[queue[0]] >= k { // 差值大于等于k
                        res = min(res, i-queue[0])
                        queue = queue[1:]
                }
                queue = append(queue, i)
        }
        if res == math.MaxInt32 {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}

# 2
var arr []int

func shortestSubarray(nums []int, k int) int {
        res := math.MaxInt32
        n := len(nums)
        arr = make([]int, n+1) // 前缀和在堆里面参与比较，使用全局变量
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i <= n; i++ {
                for intHeap.Len() > 0 && arr[i]-arr[intHeap[0]] >= k {
                        res = min(res, i-intHeap[0])
                        heap.Pop(&intHeap)
                }
                heap.Push(&intHeap, i)
        }
        if res == math.MaxInt32 {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
```

(续下页)

(接上页)

```
        return arr[h[i]] < arr[h[j]]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

27.10 871. 最低加油次数 (3)

- 题目

```
汽车从起点出发驶向目的地，该目的地位于出发位置东面 target英里处。
沿途有加油站，每个station[i]代表一个加油站，它位于出发位置东面station[i][0]英里处，
并且有station[i][1]升汽油。
假设汽车油箱的容量是无限的，其中最初有startFuel升燃料。它每行驶 1 英里就会用掉 1␣
↪升汽油。
当汽车到达加油站时，它可能停下来加油，将所有汽油从加油站转移到汽车中。
为了到达目的地，汽车所必要的最低加油次数是多少？如果无法到达目的地，则返回 -1 。
注意：如果汽车到达加油站时剩余燃料为 0，它仍然可以在那里加油。
如果汽车到达目的地时剩余燃料为 0，仍然认为它已经到达目的地。
示例 1： 输入：target = 1, startFuel = 1, stations = []输出：0
解释：我们可以在不加油的情况下到达目的地。
示例 2：输入：target = 100, startFuel = 1, stations = [[10,100]] 输出：-1
解释：我们无法抵达目的地，甚至无法到达第一个加油站。
示例 3：输入：target = 100, startFuel = 10, stations = [[10,60],[20,30],[30,30],[60,
↪40]]
输出：2
解释：我们出发时有 10 升燃料。
我们开车来到距起点 10 英里处的加油站，消耗 10 升燃料。将汽油从 0 升加到 60 升。
然后，我们从 10 英里处的加油站开到 60 英里处的加油站（消耗 50 升燃料），
并将汽油从 10 升加到 50 升。然后我们开车抵达目的地。
我们沿途在1两个加油站停靠，所以返回 2 。
```

(续下页)

(接上页)

```
提示：1 <= target, startFuel, stations[i][1] <= 10^9
0 <= stations.length <= 500
0 < stations[0][0] < stations[1][0] < ... < stations[stations.length-1][0] < target
```

- 解题思路

```
func minRefuelStops(target int, startFuel int, stations [][]int) int {
        res := 0
        total := startFuel
        if total >= target {
                return 0
        }
        Heap := &IntHeap{}
        heap.Init(Heap)
        for i := 0; i < len(stations); i++ {
                for total < stations[i][0] {
                        if Heap.Len() == 0 {
                                return -1
                        }
                        total += heap.Pop(Heap).(int)
                        res++
                }
                heap.Push(Heap, stations[i][1])
        }
        for total < target {
                if Heap.Len() == 0 {
                        return -1
                }
                total += heap.Pop(Heap).(int)
                res++
        }
        return res
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}
```

(续下页)

(接上页)

```
func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func minRefuelStops(target int, startFuel int, stations [][]int) int {
        n := len(stations)
        dp := make([]int, n+1)
        dp[0] = startFuel
        for i := 0; i < n; i++ {
                for j := i; j >= 0; j-- {
                        if dp[j] >= stations[i][0] {
                                dp[j+1] = max(dp[j+1], dp[j]+stations[i][1])
                        }
                }
        }
        for i := 0; i <= n; i++ {
                if dp[i] >= target {
                        return i
                }
        }
        return -1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
```

(续下页)

(接上页)

```
func minRefuelStops(target int, startFuel int, stations [][]int) int {
        n := len(stations)
        // dp[i][j]经过第i个加油站加油j次能够到达的最远距离
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
                dp[i][0] = startFuel
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= i; j++ {
                        // 不加油
                        if dp[i-1][j] >= stations[i-1][0] {
                                dp[i][j] = dp[i-1][j]
                        }
                        // 加油
                        if dp[i-1][j-1] >= stations[i-1][0] {
                                dp[i][j] = max(dp[i][j], dp[i-1][j-1]+stations[i-
↪1][1])
                        }
                }
        }
        for i := 0; i <= n; i++ {
                if dp[n][i] >= target {
                        return i
                }
        }
        return -1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

27.11 878. 第 N 个神奇数字 (3)

- 题目

```
如果正整数可以被 A 或 B 整除，那么它是神奇的。
返回第 N 个神奇数字。由于答案可能非常大，返回它模10^9 + 7的结果。
示例 1：输入：N = 1, A = 2, B = 3 输出：2
示例2：输入：N = 4, A = 2, B = 3 输出：6
示例 3：输入：N = 5, A = 2, B = 4 输出：10
示例 4：输入：N = 3, A = 6, B = 4 输出：8
提示：1 <= N<= 10^9
2 <= A<= 40000
2 <= B<= 40000
```

- 解题思路

```
var mod = 1000000007

func nthMagicalNumber(n int, a int, b int) int {
        ab := lcm(a, b)
        left := 1
        right := int(math.Pow10(15))
        for left <= right {
                mid := left + (right-left)/2
                total := mid/a + mid/b - mid/ab
                if total == n {
                        if mid%a == 0 || mid%b == 0 {
                                return mid % mod
                        }
                        right = mid - 1
                } else if total < n {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return left % mod
}

// 求最小公倍数
func lcm(x, y int) int {
        return x * y / gcd(x, y)
}
```

(续下页)

(接上页)

```
// 求最大公约数
func gcd(a, b int) int {
        if a < b {
                a, b = b, a
        }
        if a%b == 0 {
                return b
        }
        return gcd(a%b, b)
}

# 2
var mod = 1000000007

func nthMagicalNumber(n int, a int, b int) int {
        ab := lcm(a, b)
        left := 0
        right := int(math.Pow10(15))
        for left < right {
                mid := left + (right-left)/2
                total := mid/a + mid/b - mid/ab
                if total >= n {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left % mod
}

// 求最小公倍数
func lcm(x, y int) int {
        return x * y / gcd(x, y)
}

// 求最大公约数
func gcd(a, b int) int {
        if a < b {
                a, b = b, a
        }
        if a%b == 0 {
                return b
        }
```

(续下页)

(接上页)

```
        return gcd(a%b, b)
}

# 3
var mod = 1000000007

func nthMagicalNumber(n int, a int, b int) int {
        ab := lcm(a, b)
        // 设 ab为A，B的最小公倍数，如果 x(x<=ab)是神奇数字，那么x+ab也是神奇数字
        m := ab/a + ab/b - 1 // 有m个神奇数字小于ab
        // n = m*q + r
        q := n / m
        r := n % m
        res := q * ab % mod
        if r == 0 {
                return res
        }
        arr := []int{a, b}
        for i := 0; i < r-1; i++ { // 计算剩下的r-1个神奇数字
                if arr[0] <= arr[1] {
                        arr[0] = arr[0] + a
                } else {
                        arr[1] = arr[1] + b
                }
        }
        res = res + min(arr[0], arr[1])
        return res % mod
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

// 求最小公倍数
func lcm(x, y int) int {
        return x * y / gcd(x, y)
}

// 求最大公约数
func gcd(a, b int) int {
```

(续下页)

(接上页)

```
        if a < b {
                a, b = b, a
        }
        if a%b == 0 {
                return b
        }
        return gcd(a%b, b)
}
```

27.12 879. 盈利计划 (3)

- 题目

```
集团里有 n 名员工，他们可以完成各种各样的工作创造利润。
第i种工作会产生profit[i]的利润，它要求group[i]名成员共同参与。
如果成员参与了其中一项工作，就不能参与另一项工作。
工作的任何至少产生minProfit 利润的子集称为 盈利计划 。并且工作的成员总数最多为 n 。
有多少种计划可以选择？因为答案很大，所以 返回结果模10^9 + 7的值。
示例 1：输入：n = 5, minProfit = 3, group = [2,2], profit = [2,3] 输出：2
解释：至少产生 3 的利润，该集团可以完成工作 0 和工作 1 ，或仅完成工作 1 。
总的来说，有两种计划。
示例 2：输入：n = 10, minProfit = 5, group = [2,3,5], profit = [6,7,8] 输出：7
解释：至少产生 5 的利润，只要完成其中一种工作就行，所以该集团可以完成任何工作。
有 7 种可能的计划：(0)，(1)，(2)，(0,1)，(0,2)，(1,2)，以及 (0,1,2) 。
提示：1 <= n <= 100
0 <= minProfit <= 100
1 <= group.length <= 100
1 <= group[i] <= 100
profit.length == group.length
0 <= profit[i] <= 100
```

- 解题思路

```
var mod = 1000000007

func profitableSchemes(n int, minProfit int, group []int, profit []int) int {
        length := len(group)
        dp := make([][][]int, length+1) // dp[i][j][k] =>␣
↪在前i个工作，j个员工，利润最少为k 的计划数
        for i := 0; i <= length; i++ {
                dp[i] = make([][]int, n+1)
                for j := 0; j <= n; j++ {
```

(续下页)

(接上页)

```
                        dp[i][j] = make([]int, minProfit+1)
                }
        }
        dp[0][0][0] = 1
        for i := 0; i < length; i++ {
                profitValue := profit[i]
                groupValue := group[i]
                for j := 0; j <= n; j++ {
                        for k := 0; k <= minProfit; k++ {
                                if j < groupValue { // 人数不够
                                        dp[i+1][j][k] = dp[i][j][k]
                                } else { // 人数足够
                    // 利润为负：说明前面j-groupValue个组可以不干活，产生利润为0
                                        maxValue := max(0, k-profitValue) //␣
↪工作利润至少为k,而不是工作利润恰好为k
                                        dp[i+1][j][k] = (dp[i][j][k] + dp[i][j-
↪groupValue][maxValue]) % mod
                                }
                        }
                }
        }
        res := 0
        for j := 0; j <= n; j++ {
                res = (res + dp[length][j][minProfit]) % mod
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var mod = 1000000007

func profitableSchemes(n int, minProfit int, group []int, profit []int) int {
        length := len(group)
        dp := make([][]int, n+1) // dp[j][k] => 在j个员工，利润最少为k 的计划数
        for j := 0; j <= n; j++ {
                dp[j] = make([]int, minProfit+1)
```

(续下页)

(接上页)

```
                // 这里都为1后，就不需要求和
                dp[j][0] = 1 //
↪对于最小工作利润为0的情况，无论当前在工作的员工有多少人，我们总能提供一种方案
        }
        for i := 0; i < length; i++ {
                profitValue := profit[i]
                groupValue := group[i]
                for j := n; j >= groupValue; j-- {
                        for k := minProfit; k >= 0; k-- {
                // 利润为负：说明前面j-groupValue个组可以不干活，产生利润为0
                                maxValue := max(0, k-profitValue) // 工作利润至少为k,
↪而不是工作利润恰好为k
                                dp[j][k] = (dp[j][k] + dp[j-groupValue][maxValue]) %
↪mod
                        }
                }
        }
        return dp[n][minProfit]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
var mod = 1000000007

func profitableSchemes(n int, minProfit int, group []int, profit []int) int {
        length := len(group)
        dp := make([][]int, n+1) // dp[j][k] => 在j个员工，利润最少为k 的累计计划数
        for j := 0; j <= n; j++ {
                dp[j] = make([]int, minProfit+1)
        }
        dp[0][0] = 1
        for i := 0; i < length; i++ {
                profitValue := profit[i]
                groupValue := group[i]
                for j := n; j >= groupValue; j-- {
                        for k := minProfit; k >= 0; k-- {
                // 利润为负：说明前面j-groupValue个组可以不干活，产生利润为0
```

(续下页)

(接上页)

```
                                maxValue := max(0, k-profitValue) // 工作利润至少为k,
↪而不是工作利润恰好为k
                                dp[j][k] = (dp[j][k] + dp[j-groupValue][maxValue]) %␣
↪mod
                        }
                }
        }
        res := 0
        for j := 0; j <= n; j++ {
                res = (res + dp[j][minProfit]) % mod
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

27.13 887. 鸡蛋掉落 (2)

- 题目

```
你将获得 K 个鸡蛋，并可以使用一栋从 1 到 N  共有 N 层楼的建筑。
每个蛋的功能都是一样的，如果一个蛋碎了，你就不能再把它掉下去。
你知道存在楼层 F ，满足 0 <= F <= N 任何从高于 F 的楼层落下的鸡蛋都会碎，
从 F 楼层或比它低的楼层落下的鸡蛋都不会破。
每次移动，你可以取一个鸡蛋（如果你有完整的鸡蛋）并把它从任一楼层 X 扔下（满足 1 <= X
↪<= N）。
你的目标是确切地知道 F 的值是多少。
无论 F 的初始值如何，你确定 F 的值的最小移动次数是多少？
示例 1：输入：K = 1, N = 2 输出：2
解释：鸡蛋从 1 楼掉落。如果它碎了，我们肯定知道 F = 0 。
否则，鸡蛋从 2 楼掉落。如果它碎了，我们肯定知道 F = 1 。
如果它没碎，那么我们肯定知道 F = 2 。
因此，在最坏的情况下我们需要移动 2 次以确定 F 是多少。
示例 2：输入：K = 2, N = 6 输出：3
示例 3：输入：K = 3, N = 14 输出：4
提示：1 <= K <= 100
    1 <= N <= 10000
```

- 解题思路

```
func superEggDrop(K int, N int) int {
        if K == 1 {
                return N
        }
        if N == 1 {
                return 1
        }
        dp := make([][]int, K+1)
        for i := 0; i <= K; i++ {
                dp[i] = make([]int, N+1)
        }
        for i := 0; i <= N; i++ {
                dp[1][i] = i // 1个鸡蛋N层楼，需要移动N次
        }
        for i := 1; i <= K; i++ {
                dp[i][1] = 1 // i个鸡蛋1层楼，只需要移动1次
        }
        for i := 2; i <= K; i++ {
                for j := 2; j <= N; j++ {
                        if dp[i][j] == 0 {
                                dp[i][j] = N // 最多N次，默认值
                        }
                        for x := 1; x <= j; x++ { // x是目标楼层，不断尝试x,找到最小值
                                // dp[i][j-x] 没碎  dp[i-1][x-1] 碎了
                                value := max(dp[i][j-x], dp[i-1][x-1]) + 1
                                dp[i][j] = min(dp[i][j], value)
                        }
                }
        }
        return dp[K][N]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}

# 2
func superEggDrop(K int, N int) int {
        // dp[i][j] 有i次操作，j个鸡蛋时能测出的最高的楼层数
        dp := make([][]int, K+1)
        for i := 0; i <= K; i++ {
                dp[i] = make([]int, N+1)
        }
        for j := 1; j <= N; j++ { // 操作次数
                for i := 1; i <= K; i++ { //  K个蛋
                        // dp[i][j-1](没碎)+dp[i-1][j-1](碎了)+当前
                        dp[i][j] = dp[i][j-1] + dp[i-1][j-1] + 1
                        if dp[i][j] >= N {
                                return j
                        }
                }
        }
        return N
}
```

CHAPTER 28

0901-1000-Easy

28.1 905. 按奇偶排序数组 (4)

- 题目

```
给定一个非负整数数组 A，返回一个数组，在该数组中， A␣
↪的所有偶数元素之后跟着所有奇数元素。
你可以返回满足此条件的任何数组作为答案。
示例：输入：[3,1,2,4] 输出：[2,4,3,1]
输出 [4,2,3,1]，[2,4,1,3] 和 [4,2,1,3] 也会被接受。
提示：
    1 <= A.length <= 5000
    0 <= A[i] <= 5000
```

- 解题思路

```
func sortArrayByParity(A []int) []int {
        i := 0
        j := len(A)-1
        for i < j{
                if A[i] % 2== 0{
                        i++
                }else if A[j] % 2 == 1{
                        j--
                }else {
```

(续下页)

(接上页)

```
                        A[i], A[j] = A[j], A[i]
                }
        }
        return A
}

#
func sortArrayByParity(A []int) []int {
        i := 0
        j := len(A) - 1
        for i < j {
                for i < j && A[i]%2 == 0 {
                        i++
                }
                for i < j && A[j]%2 == 1 {
                        j--
                }
                A[i], A[j] = A[j], A[i]
        }
        return A
}

#
func sortArrayByParity(A []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(A); i++ {
                if A[i]%2 == 0 {
                        res = append(res, A[i])
                }
        }
        for i := 0; i < len(A); i++ {
                if A[i]%2 == 1 {
                        res = append(res, A[i])
                }
        }
        return res
}

#
func sortArrayByParity(A []int) []int {
        count := 0
        for i := 0; i < len(A); i++{
                if A[i] % 2 == 0{
```

(续下页)

(接上页)

```
                        A[count],A[i] = A[i], A[count]
                        count++
                }
        }
        return A
}
```

28.2 908. 最小差值 I(2)

- 题目

```
给你一个整数数组 A，对于每个整数 A[i]，我们可以选择处于区间 [-K, K] 中的任意数 x ，
将 x 与 A[i] 相加，结果存入 A[i] 。
在此过程之后，我们得到一些数组 B。
返回 B 的最大值和 B 的最小值之间可能存在的最小差值。
示例 1：输入：A = [1], K = 0 输出：0
解释：B = [1]
示例 2：输入：A = [0,10], K = 2 输出：6
解释：B = [2,8]
示例 3：输入：A = [1,3,6], K = 3 输出：0
解释：B = [3,3,3] 或 B = [4,4,4]
提示：
    1 <= A.length <= 10000
    0 <= A[i] <= 10000
    0 <= K <= 10000
```

- 解题思路

```
func smallestRangeI(A []int, K int) int {
        if len(A) == 1 {
                return 0
        }
        sort.Ints(A)
        if A[len(A)-1]-A[0] > 2*K {
                return A[len(A)-1] - A[0] - 2*K
        }
        return 0
}

#
func smallestRangeI(A []int, K int) int {
        if len(A) == 1 {
```

(续下页)

(接上页)

```
                return 0
        }
        min := A[0]
        max := A[0]
        for i := 0; i < len(A); i++ {
                if A[i] > max {
                        max = A[i]
                }
                if A[i] < min {
                        min = A[i]
                }
        }
        if max-min > 2*K {
                return max - min - 2*K
        }
        return 0
}
```

28.3 914. 卡牌分组

- 题目

```
给定一副牌，每张牌上都写着一个整数。
此时，你需要选定一个数字 X，使我们可以将整副牌按下述规则分成 1 组或更多组：
    每组都有 X 张牌。
    组内所有的牌上都写着相同的整数。
仅当你可选的 X >= 2 时返回 true。

示例 1：输入：[1,2,3,4,4,3,2,1] 输出：true
解释：可行的分组是 [1,1]，[2,2]，[3,3]，[4,4]
示例 2：输入：[1,1,1,2,2,2,3,3] 输出：false
解释：没有满足要求的分组。
示例 3：输入：[1] 输出：false
解释：没有满足要求的分组。
示例 4：输入：[1,1] 输出：true
解释：可行的分组是 [1,1]
示例 5：输入：[1,1,2,2,2,2] 输出：true
解释：可行的分组是 [1,1]，[2,2]，[2,2]
提示：
    1 <= deck.length <= 10000
    0 <= deck[i] < 10000
```

- 解题思路

```
func hasGroupsSizeX(deck []int) bool {
        if len(deck) < 2 {
                return false
        }
        m := make(map[int]int)
        for i := 0; i < len(deck); i++ {
                m[deck[i]]++
        }
        v := m[deck[0]]
        for _, value := range m {
                v = gcd(v, value)
                if v < 2 {
                        return false
                }
        }
        return true
}

func gcd(x, y int) int {
        a := x % y
        if a > 0 {
                return gcd(y, a)
        }
        return y
}

#
func hasGroupsSizeX(deck []int) bool {
        if len(deck) < 2 {
                return false
        }
        m := make(map[int]int)
        for i := 0; i < len(deck); i++ {
                m[deck[i]]++
        }
        for i := 2; i <= len(deck); i++ {
                flag := true
                if len(deck)%i == 0 {
                        for _, v := range m {
                                if v%i != 0 {
                                        flag = false
                                        break
                                }
                        }
```

(续下页)

(接上页)

```
                        if flag == true {
                                return true
                        }
                }
        }
        return false
}
```

28.4 917. 仅仅反转字母 (4)

- 题目

```
给定一个字符串 S，返回 “反转后的”␣
↪字符串，其中不是字母的字符都保留在原地，而所有字母的位置发生反转。
示例 1：输入："ab-cd" 输出："dc-ba"
示例 2：输入："a-bC-dEf-ghIj" 输出："j-Ih-gfE-dCba"
示例 3：输入："Test1ng-Leet=code-Q!" 输出："Qedo1ct-eeLg=ntse-T!"
提示：
    S.length <= 100
    33 <= S[i].ASCIIcode <= 122
    S 中不包含 \ or "
```

- 解题思路

```
func reverseOnlyLetters(S string) string {
        i := 0
        j := len(S) - 1
        arr := []byte(S)
        for i < j {
                for i < j && !isLetter(arr[i]) {
                        i++
                }
                for i < j && !isLetter(arr[j]) {
                        j--
                }
                arr[i], arr[j] = arr[j], arr[i]
                i++
                j--
        }
        return string(arr)
}
```

(续下页)

(接上页)

```
func isLetter(b byte) bool {
        if (b >= 'a' && b <= 'z') || (b >= 'A' && b <= 'Z') {
                return true
        }
        return false
}

#
func reverseOnlyLetters(S string) string {
        i := 0
        j := len(S) - 1
        arr := []byte(S)
        for i < j {
                if !isLetter(arr[i]) {
                        i++
                } else if !isLetter(arr[j]) {
                        j--
                } else {
                        arr[i], arr[j] = arr[j], arr[i]
                        i++
                        j--
                }
        }
        return string(arr)
}

func isLetter(b byte) bool {
        if (b >= 'a' && b <= 'z') || (b >= 'A' && b <= 'Z') {
                return true
        }
        return false
}

#
func reverseOnlyLetters(S string) string {
        i := 0
        j := len(S) - 1
        arr := []rune(S)
        for i < j {
                if !unicode.IsLetter(arr[i]) {
                        i++
                } else if !unicode.IsLetter(arr[j]) {
                        j--
```

(续下页)

(接上页)

```
                } else {
                        arr[i], arr[j] = arr[j], arr[i]
                        i++
                        j--
                }
        }
        return string(arr)
}

#
func reverseOnlyLetters(S string) string {
        stack := make([]rune, 0)
        res := make([]rune, 0)
        arr := []rune(S)
        for i := 0; i < len(arr); i++ {
                if unicode.IsLetter(arr[i]) {
                        stack = append(stack, arr[i])
                }
        }
        for i := 0; i < len(arr); i++ {
                if unicode.IsLetter(arr[i]) {
                        res = append(res, stack[len(stack)-1])
                        stack = stack[:len(stack)-1]
                } else {
                        res = append(res, arr[i])
                }
        }
        return string(res)
}
```

28.5 922. 按奇偶排序数组 II(3)

- 题目

```
给定一个非负整数数组 A， A 中一半整数是奇数，一半整数是偶数。
对数组进行排序，以便当 A[i] 为奇数时，i 也是奇数；当 A[i] 为偶数时， i 也是偶数。
你可以返回任何满足上述条件的数组作为答案。
示例：输入：[4,2,5,7]输出：[4,5,2,7]
解释：[4,7,2,5]，[2,5,4,7]，[2,7,4,5] 也会被接受。
提示：
    2 <= A.length <= 20000
    A.length % 2 == 0
```

(续下页)

(接上页)

```
    0 <= A[i] <= 1000
```

- 解题思路

```
func sortArrayByParityII(A []int) []int {
        i := 0
        j := 1
        for i < len(A) || j < len(A) {
                for i < len(A) && A[i]%2 == 0 {
                        i = i + 2
                }
                for j < len(A) && A[j]%2 == 1 {
                        j = j + 2
                }
                if i >= len(A) || j >= len(A) {
                        break
                }
                A[i], A[j] = A[j], A[i]
        }
        return A
}

#
func sortArrayByParityII(A []int) []int {
        i := 0
        j := 1
        for i < len(A) {
                for A[i]%2 != 0 {
                        if A[j]%2 == 0 {
                                A[i], A[j] = A[j], A[i]
                        } else {
                                j = j + 2
                        }
                }
                i = i + 2
        }
        return A
}

#
func sortArrayByParityII(A []int) []int {
        i := 0
        j := 1
        res := make([]int, len(A))
```

(续下页)

(接上页)

```
        for k := 0; k < len(A); k++ {
                if A[k]%2 == 0 {
                        res[i] = A[k]
                        i = i + 2
                } else {
                        res[j] = A[k]
                        j = j + 2
                }
        }
        return res
}
```

28.6 925. 长按键入 (2)

- 题目

```
你的朋友正在使用键盘输入他的名字 name。偶尔，在键入字符 c 时，按键可能会被长按，
而字符可能被输入 1 次或多次。
你将会检查键盘输入的字符 typed。
如果它对应的可能是你的朋友的名字（其中一些字符可能被长按），那么就返回 True。

示例 1：  输入：name = "alex", typed = "aaleex" 输出：true
解释：'alex' 中的 'a' 和 'e' 被长按。
示例 2：  输入：name = "saeed", typed = "ssaaedd" 输出：false
解释：'e' 一定需要被键入两次，但在 typed 的输出中不是这样。
示例 3：  输入：name = "leelee", typed = "lleeelee" 输出：true
示例 4：  输入：name = "laiden", typed = "laiden" 输出：true
解释：长按名字中的字符并不是必要的。
提示：
    name.length <= 1000
    typed.length <= 1000
    name 和 typed 的字符都是小写字母。
```

- 解题思路

```
func isLongPressedName(name string, typed string) bool {
        i := 0
        j := 0
        for j < len(typed) {
                if i == len(name) {
                        i = len(name) - 1
                }
```

(续下页)

(接上页)

```
                if name[i] == typed[j] {
                        // 正确的话，保证i == len(name) && j == len(typed)
                        i++
                        j++
                } else {
                        if i == 0 {
                                return false
                        }
                        if name[i-1] != typed[j] {
                                return false
                        } else {
                                j++
                        }
                }
        }
        return i == len(name) && j == len(typed)
}

#
func isLongPressedName(name string, typed string) bool {
        i := 1
        j := 1
        countI := 0
        countJ := 0
        for i < len(name) || j < len(typed) {
                for i < len(name) && name[i] == name[i-1] {
                        i++
                        countI++
                }
                for j < len(typed) && typed[j] == typed[j-1] {
                        j++
                        countJ++
                }
                if name[i-1] != typed[j-1] || countJ < countI {
                        return false
                }
                i++
                j++
                countI = 0
                countJ = 0
        }
        return name[len(name)-1] == typed[len(typed)-1]
}
```

28.7 929. 独特的电子邮件地址 (2)

- 题目

```
每封电子邮件都由一个本地名称和一个域名组成，以 @ 符号分隔。
例如，在 alice@leetcode.com中， alice 是本地名称，而 leetcode.com 是域名。
除了小写字母，这些电子邮件还可能包含 '.' 或 '+'。
如果在电子邮件地址的本地名称部分中的某些字符之间添加句点 ('.')，
则发往那里的邮件将会转发到本地名称中没有点的同一地址。
例如，"alice.z@leetcode.com” 和 “alicez@leetcode.com” 会转发到同一电子邮件地址。
（请注意，此规则不适用于域名。）
如果在本地名称中添加加号 ('+')，则会忽略第一个加号后面的所有内容。
这允许过滤某些电子邮件，例如 m.y+name@email.com 将转发到 my@email.com。
（同样，此规则不适用于域名。）
可以同时使用这两个规则。
给定电子邮件列表␣
↪emails，我们会向列表中的每个地址发送一封电子邮件。实际收到邮件的不同地址有多少？
示例：
输入：["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com",
"testemail+david@lee.tcode.com"]
输出：2
解释：实际收到邮件的是 "testemail@leetcode.com" 和 "testemail@lee.tcode.com"。
提示：
    1 <= emails[i].length <= 100
    1 <= emails.length <= 100
    每封 emails[i] 都包含有且仅有一个 '@' 字符。
```

- 解题思路

```
func numUniqueEmails(emails []string) int {
        m := make(map[string]bool)
        for i := 0; i < len(emails); i++ {
                addr := ""
                arr := strings.Split(emails[i], "@")
                for j := 0; j < len(arr[0]); j++ {
                        if arr[0][j] == '+' {
                                break
                        } else if arr[0][j] == '.' {
                                continue
                        } else {
                                addr = addr + string(arr[0][j])
                        }
                }
                m[addr+"@"+arr[1]] = true
```

(续下页)

(接上页)

```
        }
        return len(m)
}

#
func numUniqueEmails(emails []string) int {
        m := make(map[string]bool)
        for i := 0; i < len(emails); i++ {
                addr := ""
                isBreak := false
                for j := 0; j < len(emails[i]); j++ {
                        if emails[i][j] == '+' {
                                isBreak = true
                        } else if emails[i][j] == '.' {
                                continue
                        } else if emails[i][j] == '@' {
                                addr = addr + emails[i][j:]
                                break
                        } else if isBreak == true {
                        } else {
                                addr = addr + string(emails[i][j])
                        }
                }
                m[addr] = true
        }
        return len(m)
}
```

28.8 933. 最近的请求次数 (2)

- 题目

```
写一个 RecentCounter 类来计算最近的请求。
它只有一个方法：ping(int t)，其中 t 代表以毫秒为单位的某个时间。
返回从 3000 毫秒前到现在的 ping 数。
任何处于 [t - 3000, t] 时间范围之内的 ping 都将会被计算在内，包括当前（指 t 时刻）的ping。
保证每次对 ping 的调用都使用比之前更大的 t 值。

示例：
输入：inputs = ["RecentCounter","ping","ping","ping","ping"],
inputs = [[],[1],[100],[3001],[3002]]
```

(续下页)

(接上页)

```
输出：[null,1,2,3,3]
提示：
    每个测试用例最多调用 10000 次 ping。
    每个测试用例会使用严格递增的 t 值来调用 ping。
    每次调用 ping 都有 1 <= t <= 10^9。
```

- 解题思路

```
type RecentCounter struct {
        arr []int
}

func Constructor() RecentCounter {
        return RecentCounter{
                arr: make([]int, 0),
        }
}

func (r *RecentCounter) Ping(t int) int {
        r.arr = append(r.arr, t)
        res := 1
        for i := len(r.arr) - 2; i >= 0; i-- {
                if t-r.arr[i] <= 3000 {
                        res++
                } else {
                        r.arr = r.arr[i+1:]
                        break
                }
        }
        return res
}

#
type RecentCounter struct {
        arr []int
}

func Constructor() RecentCounter {
        return RecentCounter{
                arr: make([]int, 0),
        }
}

func (r *RecentCounter) Ping(t int) int {
```

(续下页)

(接上页)

```
        r.arr = append(r.arr, t)
        start := t - 3000
        for len(r.arr) > 0 && r.arr[0] < start {
                r.arr = r.arr[1:]
        }
        return len(r.arr)
}
```

28.9 937. 重新排列日志文件 (2)

- 题目

```
你有一个日志数组 logs。每条日志都是以空格分隔的字串。
对于每条日志，其第一个字为字母与数字混合的 标识符。
    除标识符之外，所有字均由小写字母组成的，称为 字母日志
    除标识符之外，所有字均由数字组成的，称为 数字日志
题目所用数据保证每个日志在其标识符后面至少有一个字。
请按下述规则将日志重新排序：
    所有 字母日志 都排在 数字日志 之前。
    字母日志␣
↪在内容不同时，忽略标识符后，按内容字母顺序排序；在内容相同时，按标识符排序；
    数字日志 应该按原来的顺序排列。
返回日志的最终顺序。
示例 ：
输入：["a1 9 2 3 1","g1 act car","zo4 4 7","ab1 off key dog","a8 act zoo"]
输出：["g1 act car","a8 act zoo","ab1 off key dog","a1 9 2 3 1","zo4 4 7"]
提示：
    0 <= logs.length <= 100
    3 <= logs[i].length <= 100
    logs[i] 保证有一个标识符，并且标识符后面有一个字。
```

- 解题思路

```
func reorderLogFiles(logs []string) []string {
        numLogs := make([]string, 0)
        wordLogs := make([]string, 0)
        for key := range logs {
                for i := 0; i < len(logs[key]); i++ {
                        if logs[key][i] == ' ' && i != len(logs[key])-1 {
                                if strings.ContainsAny(logs[key][i+1:], "0123456789")
↪{
                                        numLogs = append(numLogs, logs[key])
```

(续下页)

(接上页)

```
                                } else {
                                        wordLogs = append(wordLogs, logs[key])
                                }
                                break
                        }
                }
        }
        sort.Slice(wordLogs, func(i, j int) bool {
                firstIndex := strings.Index(wordLogs[i], " ")
                secondIndex := strings.Index(wordLogs[j], " ")
                if wordLogs[i][firstIndex+1:] == wordLogs[j][secondIndex+1:] {
                        return wordLogs[i][:firstIndex] < wordLogs[j][:secondIndex]
                }
                return wordLogs[i][firstIndex+1:] < wordLogs[j][secondIndex+1:]
        })
        return append(wordLogs, numLogs...)
}

#
type Logs []string

func (l Logs) Len() int {
        return len(l)
}

func (l Logs) Less(i, j int) bool {
        firstIndex := strings.Index(l[i], " ")
        secondIndex := strings.Index(l[j], " ")
        if l[i][firstIndex+1:] == l[j][secondIndex+1:] {
                return l[i][:firstIndex] < l[j][:secondIndex]
        }
        return l[i][firstIndex+1:] < l[j][secondIndex+1:]
}

func (l Logs) Swap(i, j int) {
        l[i], l[j] = l[j], l[i]
}

func reorderLogFiles(logs []string) []string {
        numLogs := make([]string, 0)
        wordLogs := make([]string, 0)
        for key := range logs {
                for i := 0; i < len(logs[key]); i++ {
```

(续下页)

(接上页)

```
                        if logs[key][i] == ' ' && i != len(logs[key])-1 {
                                if strings.ContainsAny(logs[key][i+1:], "0123456789")
↪{
                                        numLogs = append(numLogs, logs[key])
                                } else {
                                        wordLogs = append(wordLogs, logs[key])
                                }
                                break
                        }
                }
        }
        sort.Sort(Logs(wordLogs))
        return append(wordLogs, numLogs...)
}
```

28.10 938. 二叉搜索树的范围和 (2)

- 题目

```
给定二叉搜索树的根结点 root，返回 L 和 R（含）之间的所有结点的值的和。
二叉搜索树保证具有唯一的值。
示例 1：输入：root = [10,5,15,3,7,null,18], L = 7, R = 15 输出：32
示例 2：输入：root = [10,5,15,3,7,13,18,1,null,6], L = 6, R = 10 输出：23
提示：
    树中的结点数量最多为 10000 个。
    最终的答案保证小于 2^31。
```

- 解题思路

```
func rangeSumBST(root *TreeNode, L int, R int) int {
        if root == nil {
                return 0
        }
        if root.Val < L {
                return rangeSumBST(root.Right, L, R)
        }
        if root.Val > R {
                return rangeSumBST(root.Left, L, R)
        }
        return root.Val + rangeSumBST(root.Right, L, R) + rangeSumBST(root.Left, L, R)
}
```

(续下页)

(接上页)

```
#
func rangeSumBST(root *TreeNode, L int, R int) int {
        if root == nil {
                return 0
        }
        stack := make([]*TreeNode, 0)
        if root.Val > R && root.Left != nil {
                stack = append(stack, root.Left)
        } else if root.Val < L && root.Right != nil {
                stack = append(stack, root.Right)
        } else {
                stack = append(stack, root)
        }
        res := 0
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if node.Val <= R && node.Val >= L {
                        if node.Left != nil {
                                stack = append(stack, node.Left)
                        }
                        if node.Right != nil {
                                stack = append(stack, node.Right)
                        }
                        res = res + node.Val
                } else if node.Val > R && node.Left != nil {
                        stack = append(stack, node.Left)
                } else if node.Val < L && node.Right != nil {
                        stack = append(stack, node.Right)
                }
        }
        return res
}
```

28.11 941. 有效的山脉数组 (2)

- 题目

```
定一个整数数组 A，如果它是有效的山脉数组就返回 true，否则返回 false。
让我们回顾一下，如果 A 满足下述条件，那么它是一个山脉数组：
    A.length >= 3
    在 0 < i < A.length - 1 条件下，存在 i 使得：
```

(续下页)

(接上页)

```
        A[0] < A[1] < ... A[i-1] < A[i]
        A[i] > A[i+1] > ... > A[A.length - 1]
示例 1：输入：[2,1] 输出：false
示例 2：输入：[3,5,5] 输出：false
示例 3：输入：[0,3,2,1] 输出：true
提示：
    0 <= A.length <= 10000
    0 <= A[i] <= 10000
```

- 解题思路

```
func validMountainArray(A []int) bool {
        if len(A) < 3 {
                return false
        }
        pre := A[0]
        i := 0
        for i = 1; i < len(A); i++ {
                if pre < A[i] {
                        pre = A[i]
                } else if pre == A[i] {
                        return false
                } else {
                        break
                }
        }
        if i >= len(A) || i == 1 {
                return false
        }
        for ; i < len(A); i++ {
                if pre > A[i] {
                        pre = A[i]
                } else if pre == A[i] {
                        return false
                } else {
                        return false
                }
        }
        return true
}

#
func validMountainArray(A []int) bool {
        if len(A) < 3 {
```

(续下页)

(接上页)

```
                return false
        }
        i := 0
        j := len(A) - 1
        for i < j && A[i] < A[i+1] {
                i++
        }
        for i < j && A[j] < A[j-1] {
                j--
        }
        return i == j && i != 0 && j != len(A)-1
}
```

28.12 942. 增减字符串匹配 (1)

- 题目

```
给定只含 "I"（增大）或 "D"（减小）的字符串 S ，令 N = S.length。
返回 [0, 1, ..., N] 的任意排列 A 使得对于所有 i = 0, ..., N-1, 都有：
    如果 S[i] == "I", 那么 A[i] < A[i+1]
    如果 S[i] == "D", 那么 A[i] > A[i+1]
示例 1：输出："IDID" 输出：[0,4,1,3,2]
示例 2：输出："III" 输出：[0,1,2,3]
示例 3：输出："DDI" 输出：[3,2,0,1]
提示：  1 <= S.length <= 10000  S 只包含字符 "I" 或 "D"。
```

- 解题思路

```
func diStringMatch(S string) []int {
        res := make([]int, len(S)+1)
        left := 0
        right := len(S)
        for i := 0; i < len(S); i++ {
                if S[i] == 'I' {
                        res[i] = left
                        left++
                } else {
                        res[i] = right
                        right--
                }
        }
        res[len(S)] = left
```

(续下页)

(接上页)

```
        return res
}
```

28.13 944. 删列造序 (1)

- 题目

```
给定由 N 个小写字母字符串组成的数组 A，其中每个字符串长度相等。
你需要选出一组要删掉的列 D，对 A 执行删除操作，使 A 中剩余的每一列都是 非降序 排列的，
然后请你返回 D.length 的最小可能值。
删除 操作的定义是：选出一组要删掉的列，删去 A 中对应列中的所有字符，
形式上，第 n 列为 [A[0][n], A[1][n], ..., A[A.length-1][n]]）。（可以参见␣
↪删除操作范例）
示例 1：输入：["cba", "daf", "ghi"] 输出：1
解释：  当选择 D = {1}，删除后 A 的列为：["c","d","g"] 和 ["a","f","i
↪"]，均为非降序排列。
若选择 D = {}，那么 A 的列 ["b","a","h"] 就不是非降序排列了。
示例 2：输入：["a", "b"] 输出：0 解释：D = {}
示例 3：输入：["zyx", "wvu", "tsr"] 输出：3 解释：D = {0, 1, 2}
提示：
    1 <= A.length <= 100
    1 <= A[i].length <= 1000
删除操作范例：
比如，有 A = ["abcdef", "uvwxyz"]，
要删掉的列为 {0, 2, 3}，删除后 A 为["bef", "vyz"]，
A 的列分别为["b","v"], ["e","y"], ["f","z"]。
```

- 解题思路

```
func minDeletionSize(A []string) int {
        res := 0
        if len(A) == 1 {
                return res
        }
        for i := 0; i < len(A[0]); i++ {
                for j := 1; j < len(A); j++ {
                        if A[j][i] < A[j-1][i] {
                                res++
                                break
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

28.14 949. 给定数字能组成的最大时间 (2)

- 题目

```
给定一个由 4 位数字组成的数组，返回可以设置的符合 24 小时制的最大时间。
最小的 24 小时制时间是 00:00，而最大的是 23:59。从 00:00␣
↪（午夜）开始算起，过得越久，时间越大。
以长度为 5 的字符串返回答案。如果不能确定有效时间，则返回空字符串。
示例 1：输入：[1,2,3,4] 输出："23:41"
示例 2：输入：[5,5,5,5] 输出：""
提示：
    A.length == 4
    0 <= A[i] <= 9
```

- 解题思路

```
func largestTimeFromDigits(A []int) string {
        res := ""
        for i := 0; i < 4; i++ {
                for j := 0; j < 4; j++ {
                        for k := 0; k < 4; k++ {
                                for l := 0; l < 4; l++ {
                                        if i != j && i != k && i != l &&
                                                j != k && j != l && k != l {
                                                hour := A[i]*10 + A[j]
                                                minute := A[k]*10 + A[l]
                                                if hour <= 23 && minute <= 59 {
                                                        ans := fmt.Sprintf("%02d:%02d
↪", hour, minute)
                                                        if ans > res && res != "" {
                                                                res = ans
                                                        } else if res == "" {
                                                                res = ans
                                                        }
                                                }
                                        }
                                }
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

#
var arr []string

func largestTimeFromDigits(A []int) string {
        res := ""
        arr = make([]string, 0)
        dfs(A, 0, len(A)-1)
        for i := range arr {
                if (arr[i] > res && res != "") || (res == "") {
                        res = arr[i]
                }
        }
        return res
}

func dfs(A []int, start, length int) {
        if start == length {
                hour := A[0]*10 + A[1]
                minute := A[2]*10 + A[3]
                if hour <= 23 && minute <= 59 {
                        ans := fmt.Sprintf("%02d:%02d", hour, minute)
                        arr = append(arr, ans)
                }
        } else {
                for i := start; i <= length; i++ {
                        A[i], A[start] = A[start], A[i]
                        dfs(A, start+1, length)
                        A[i], A[start] = A[start], A[i]
                }
        }
}
```

28.15 953. 验证外星语词典 (2)

- 题目

```
某种外星语也使用英文小写字母，但可能顺序 order␣
↪不同。字母表的顺序（order）是一些小写字母的排列。
给定一组用外星语书写的单词 words，以及其字母表的顺序␣
↪order，只有当给定的单词在这种外星语中按字典序排列时，
返回 true；否则，返回 false。
示例 1：
输入：words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"
输出：true
解释：在该语言的字母表中，'h' 位于 'l' 之前，所以单词序列是按字典序排列的。
示例 2：
输入：words = ["word","world","row"], order = "worldabcefghijkmnpqstuvxyz"
输出：false
解释：在该语言的字母表中，'d' 位于 'l' 之后，那么 words[0] >␣
↪words[1]，因此单词序列不是按字典序排列的。
示例 3：
输入：words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"
输出：false
解释：当前三个字符 "app" 匹配时，第二个字符串相对短一些，然后根据词典编纂规则 "apple"␣
↪> "app"，因为 'l' > '∅'，其中 '∅' 是空白字符，定义为比任何其他字符都小（更多信息）。
提示：
    1 <= words.length <= 100
    1 <= words[i].length <= 20
    order.length == 26
    在 words[i] 和 order 中的所有字符都是英文小写字母。
```

- 解题思路

```
func isAlienSorted(words []string, order string) bool {
        newWords := make([]string, len(words))
        m := make(map[byte]int)
        for i := 0; i < len(order); i++ {
                m[order[i]] = i
        }
        for i := 0; i < len(words); i++ {
                str := ""
                for j := 0; j < len(words[i]); j++ {
                        str = str + string(m[words[i][j]]+'a')
                }
                newWords[i] = str
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(newWords)-1; i++ {
                if newWords[i] > newWords[i+1] {
                        return false
                }
        }
        return true
}

#
func isAlienSorted(words []string, order string) bool {
        m := make(map[byte]int)
        for i := 0; i < len(order); i++ {
                m[order[i]] = i
        }

        for i := 0; i < len(words)-1; i++ {
                length := len(words[i])
                if len(words[i+1]) < length {
                        length = len(words[i+1])
                }
                for j := 0; j < length; j++ {
                        if m[words[i][j]] < m[words[i+1][j]] {
                                break
                        }
                        if m[words[i][j]] > m[words[i+1][j]] {
                                return false
                        }
                        if j == length-1 {
                                if len(words[i]) > len(words[i+1]) {
                                        return false
                                }
                        }
                }
        }
        return true
}
```

28.16 961. 重复 N 次的元素 (5)

- 题目

```
在大小为 2N 的数组 A 中有 N+1 个不同的元素，其中有一个元素重复了 N 次。
返回重复了 N 次的那个元素。
示例 1：输入：[1,2,3,3] 输出：3
示例 2：输入：[2,1,2,5,3,2] 输出：2
示例 3：输入：[5,1,5,2,5,3,5,4] 输出：5
提示：
    4 <= A.length <= 10000
    0 <= A[i] < 10000
    A.length 为偶数
```

- 解题思路

```
func repeatedNTimes(A []int) int {
        m := make(map[int]int)
        for i := 0; i < len(A); i++ {
                if _, ok := m[A[i]]; ok {
                        return A[i]
                }
                m[A[i]]++
        }
        return 0
}

#
func repeatedNTimes(A []int) int {
        m := make(map[int]int)
        for i := 0; i < len(A); i++ {
                if _, ok := m[A[i]]; ok {
                        return A[i]
                }
                m[A[i]]++
        }
        return 0
}

#
func repeatedNTimes(A []int) int {
        m := make(map[int]int)
        for i := 0; i < len(A); i++ {
                m[A[i]]++
```

(续下页)

(接上页)

```
        }
        for key, value := range m {
                if value == len(A)/2 {
                        return key
                }
        }
        return 0
}

# 4
func repeatedNTimes(A []int) int {
        for i := 0; i < len(A)-2; i++ {
                if A[i] == A[i+1] || A[i] == A[i+2] {
                        return A[i]
                }
        }
        return A[len(A)-1]
}

# 5
func repeatedNTimes(A []int) int {
        for i := 0; i < len(A); i++ {
                for j := i + 1; j < len(A); j++ {
                        if A[i] == A[j] {
                                return A[i]
                        }
                }
        }
        return A[len(A)-1]
}
```

28.17 965. 单值二叉树 (4)

- 题目

```
如果二叉树每个节点都具有相同的值，那么该二叉树就是单值二叉树。
只有给定的树是单值二叉树时，才返回 true；否则返回 false。
示例 1：输入：[1,1,1,1,1,null,1] 输出：true
示例 2：输入：[2,2,2,5,2] 输出：false
提示：
    给定树的节点数范围是 [1, 100]。
    每个节点的值都是整数，范围为 [0, 99] 。
```

- 解题思路

```
var arr []int

func isUnivalTree(root *TreeNode) bool {
        if root == nil {
                return true
        }
        arr = make([]int, 0)
        dfs(root)
        for i := 1; i < len(arr); i++ {
                if arr[i] != arr[i-1] {
                        return false
                }
        }
        return true
}

func dfs(root *TreeNode) {
        if root != nil {
                arr = append(arr, root.Val)
                dfs(root.Left)
                dfs(root.Right)
        }
}

#
var value int
var res bool

func isUnivalTree(root *TreeNode) bool {
        if root == nil {
                return true
        }
        res = true
        value = root.Val
        dfs(root)
        return res
}

func dfs(root *TreeNode) {
        if root != nil {
                if root.Val != value {
                        res = false
                        return
```

(续下页)

(接上页)

```
                }
                dfs(root.Left)
                dfs(root.Right)
        }
}

#
func isUnivalTree(root *TreeNode) bool {
        if root == nil {
                return true
        }
        if (root.Left != nil && root.Left.Val != root.Val) ||
                (root.Right != nil && root.Right.Val != root.Val) {
                return false
        }
        return isUnivalTree(root.Left) && isUnivalTree(root.Right)
}

#
func isUnivalTree(root *TreeNode) bool {
        if root == nil {
                return true
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        value := root.Val
        for len(queue) > 0 {
                node := queue[len(queue)-1]
                queue = queue[:len(queue)-1]
                if node.Val != value {
                        return false
                }
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return true
}
```

28.18 970. 强整数 (2)

- 题目

```
给定两个正整数 x 和 y，如果某一整数等于 x^i + y^j，其中整数 i >= 0 且 j >= 0，
那么我们认为该整数是一个强整数。
返回值小于或等于 bound 的所有强整数组成的列表。
你可以按任何顺序返回答案。在你的回答中，每个值最多出现一次。
示例 1：输入：x = 2, y = 3, bound = 10 输出：[2,3,4,5,7,9,10]
解释：
2 = 2^0 + 3^0
3 = 2^1 + 3^0
4 = 2^0 + 3^1
5 = 2^1 + 3^1
7 = 2^2 + 3^1
9 = 2^3 + 3^0
10 = 2^0 + 3^2
示例 2：输入：x = 3, y = 5, bound = 15
输出：[2,4,6,8,10,14]
提示：
    1 <= x <= 100
    1 <= y <= 100
    0 <= bound <= 10^6
```

- 解题思路

```
func powerfulIntegers(x int, y int, bound int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        if bound < 2 {
                return res
        }
        for i := 1; i < bound; i = i * x {
                for j := 1; i+j <= bound; j = j * y {
                        if _, ok := m[i+j]; !ok {
                                res = append(res, i+j)
                                m[i+j] = 1
                        }
                        if y == 1 {
                                break
                        }
                }
                if x == 1 {
                        break
```

(续下页)

(接上页)

```
                }
        }
        return res
}

#
func powerfulIntegers(x int, y int, bound int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        if bound < 2 {
                return res
        }
        xArr := make([]int, 0)
        yArr := make([]int, 0)
        for i := 1; i < bound; i = i * x {
                xArr = append(xArr, i)
                if x == 1 {
                        break
                }
        }
        for i := 1; i < bound; i = i * y {
                yArr = append(yArr, i)
                if y == 1 {
                        break
                }
        }
        for i := 0; i < len(xArr); i++ {
                for j := 0; j < len(yArr); j++ {
                        if xArr[i]+yArr[j] <= bound && m[xArr[i]+yArr[j]] == 0 {
                                res = append(res, xArr[i]+yArr[j])
                                m[xArr[i]+yArr[j]] = 1
                        }
                }
        }
        return res
}
```

28.19 976. 三角形的最大周长 (2)

- 题目

```
给定由一些正数（代表长度）组成的数组␣
↪A，返回由其中三个长度组成的、面积不为零的三角形的最大周长。
如果不能形成任何面积不为零的三角形，返回 0。
示例 1：输入：[2,1,2] 输出：5
示例 2：输入：[1,2,1] 输出：0
示例 3：输入：[3,2,3,4] 输出：10
示例 4：输入：[3,6,2,3] 输出：8
提示：
3 <= A.length <= 10000
1 <= A[i] <= 10^6
```

- 解题思路

```
func largestPerimeter(A []int) int {
        sort.Ints(A)
        for i := len(A) - 3; i >= 0; i-- {
                if A[i]+A[i+1] > A[i+2] {
                        return A[i] + A[i+1] + A[i+2]
                }
        }
        return 0
}

#
func largestPerimeter(A []int) int {
        if len(A) < 3 {
                return 0
        }
        for i := 0; i < len(A)-1; i++ {
                for j := 0; j < len(A)-1-i; j++ {
                        if A[j] > A[j+1] {
                                A[j], A[j+1] = A[j+1], A[j]
                        }
                }
                if i >= 2 {
                        index := len(A) - 1 - i
                        if A[index]+A[index+1] > A[index+2] {
                                return A[index] + A[index+1] + A[index+2]
                        }
                }
```

(续下页)

(接上页)

```
        }
        if A[0]+A[1] > A[2] {
                return A[0] + A[1] + A[2]
        }
        return 0
}
```

28.20 977. 有序数组的平方 (3)

- 题目

```
给定一个按非递减顺序排序的整数数组␣
↪A，返回每个数字的平方组成的新数组，要求也按非递减顺序排序。
示例 1：输入：[-4,-1,0,3,10] 输出：[0,1,9,16,100]
示例 2：输入：[-7,-3,2,3,11] 输出：[4,9,9,49,121]
提示：
    1 <= A.length <= 10000
    -10000 <= A[i] <= 10000
    A 已按非递减顺序排序。
```

- 解题思路

```
func sortedSquares(A []int) []int {
        res := make([]int, len(A))
        i := 0
        j := len(A) - 1
        index := len(A) - 1
        for i <= j {
                if A[i]*A[i] < A[j]*A[j] {
                        res[index] = A[j] * A[j]
                        j--
                } else {
                        res[index] = A[i] * A[i]
                        i++
                }
                index--
        }
        return res
}

#
func sortedSquares(A []int) []int {
```

(续下页)

(接上页)

```
        res := make([]int, 0)
        for i := 0; i < len(A); i++ {
                res = append(res, A[i]*A[i])
        }
        sort.Ints(res)
        return res
}

#
func sortedSquares(A []int) []int {
        res := make([]int, len(A))
        res[0] = A[0] * A[0]
        j := 0
        for i := 1; i < len(A); i++ {
                value := A[i] * A[i]
                for j = i - 1; j >= 0; j-- {
                        if value < res[j] {
                                res[j+1] = res[j]
                        } else {
                                break
                        }
                }
                res[j+1] = value
        }
        return res
}
```

28.21 985. 查询后的偶数和 (1)

- 题目

```
给出一个整数数组 A 和一个查询数组 queries。
对于第 i 次查询，有 val = queries[i][0], index = queries[i][1],
我们会把 val 加到 A[index] 上。然后，第 i 次查询的答案是 A 中偶数值的和。
（此处给定的 index = queries[i][1] 是从 0 开始的索引，每次查询都会永久修改数组 A。）
返回所有查询的答案。你的答案应当以数组 answer 给出，answer[i] 为第 i 次查询的答案。
示例：输入：A = [1,2,3,4], queries = [[1,0],[-3,1],[-4,0],[2,3]]
输出：[8,6,2,4]
解释：开始时，数组为 [1,2,3,4]。
将 1 加到 A[0] 上之后，数组为 [2,2,3,4]，偶数值之和为 2 + 2 + 4 = 8。
将 -3 加到 A[1] 上之后，数组为 [2,-1,3,4]，偶数值之和为 2 + 4 = 6。
将 -4 加到 A[0] 上之后，数组为 [-2,-1,3,4]，偶数值之和为 -2 + 4 = 2。
```

(续下页)

(接上页)

```
将 2 加到 A[3] 上之后，数组为 [-2,-1,3,6]，偶数值之和为 -2 + 6 = 4。

提示：
    1 <= A.length <= 10000
    -10000 <= A[i] <= 10000
    1 <= queries.length <= 10000
    -10000 <= queries[i][0] <= 10000
    0 <= queries[i][1] < A.length
```

- 解题思路

```
func sumEvenAfterQueries(A []int, queries [][]int) []int {
        res := make([]int, 0)
        sum := 0
        for _, value := range A {
                if value%2 == 0 {
                        sum = sum + value
                }
        }
        for i := 0; i < len(queries); i++ {
                value := queries[i][0]
                index := queries[i][1]
                if A[index]%2 == 0 {
                        sum = sum - A[index]
                }
                A[index] = A[index] + value
                if A[index]%2 == 0 {
                        sum = sum + A[index]
                }
                res = append(res, sum)
        }
        return res
}
```

28.22 989. 数组形式的整数加法 (4)

- 题目

```
对于非负整数 X 而言，X 的数组形式是每位数字按从左到右的顺序形成的数组。
例如，如果 X = 1231，那么其数组形式为 [1,2,3,1]。
给定非负整数 X 的数组形式 A，返回整数 X+K 的数组形式。
示例 1：输入：A = [1,2,0,0]，K = 34 输出：[1,2,3,4]
```

(续下页)

(接上页)

```
解释：1200 + 34 = 1234

示例 2：
输入：A = [2,7,4], K = 181
输出：[4,5,5]
解释：274 + 181 = 455

示例 3：
输入：A = [2,1,5], K = 806
输出：[1,0,2,1]
解释：215 + 806 = 1021

示例 4：
输入：A = [9,9,9,9,9,9,9,9,9,9], K = 1
输出：[1,0,0,0,0,0,0,0,0,0,0]
解释：9999999999 + 1 = 10000000000

提示：
    1 <= A.length <= 10000
    0 <= A[i] <= 9
    0 <= K <= 10000
    如果 A.length > 1，那么 A[0] != 0
```

- 解题思路

```
func addToArrayForm(A []int, K int) []int {
        B := make([]int, 0)
        for K > 0 {
                B = append([]int{K % 10}, B...)
                K = K / 10
        }
        length := len(A)
        if len(B) > len(A) {
                length = len(B)
        }
        res := make([]int, length)
        flag := 0
        i := len(A) - 1
        j := len(B) - 1
        count := 0
        for i >= 0 && j >= 0 {
                sum := A[i] + B[j] + flag
                if sum >= 10 {
                        sum = sum - 10
```

(续下页)

(接上页)

```
                        flag = 1
                } else {
                        flag = 0
                }
                res[length-1-count] = sum
                i--
                j--
                count++
        }
        for i >= 0 {
                sum := A[i] + flag
                if sum >= 10 {
                        sum = sum - 10
                        flag = 1
                } else {
                        flag = 0
                }
                res[length-1-count] = sum
                i--
                count++
        }
        for j >= 0 {
                sum := B[j] + flag
                if sum >= 10 {
                        sum = sum - 10
                        flag = 1
                } else {
                        flag = 0
                }
                res[length-1-count] = sum
                j--
                count++
        }
        if flag == 1 {
                return append([]int{1}, res...)
        }
        return res
}

#
func addToArrayForm(A []int, K int) []int {
        A[len(A)-1] = A[len(A)-1] + K
        carry := 0
```

(续下页)

(接上页)

```
        for i := len(A) - 1; i >= 0; i-- {
                carry = A[i] / 10
                A[i] = A[i] % 10
                if i > 0 {
                        A[i-1] = A[i-1] + carry
                }
        }
        for carry > 0 {
                A = append([]int{carry % 10}, A...)
                carry = carry / 10
        }
        return A
}

#
func addToArrayForm(A []int, K int) []int {
        i := len(A) - 1
        res := make([]int, 0)
        for i >= 0 || K > 0 {
                if i >= 0 {
                        K = K + A[i]
                }
                res = append(res, K%10)
                K = K / 10
                i--
        }
        for i := 0; i < len(res)/2; i++ {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
        }
        return res
}

#
func addToArrayForm(A []int, K int) []int {
        i := len(A) - 1
        res := make([]int, 0)
        for i >= 0 || K > 0 {
                if i >= 0 {
                        K = K + A[i]
                }
                res = append([]int{K % 10}, res...)
                K = K / 10
                i--
```

(续下页)

(接上页)

```
        }
        return res
}
```

28.23 993. 二叉树的堂兄弟节点 (2)

- 题目

```
在二叉树中，根节点位于深度 0 处，每个深度为 k 的节点的子节点位于深度 k+1 处。
如果二叉树的两个节点深度相同，但父节点不同，则它们是一对堂兄弟节点。
我们给出了具有唯一值的二叉树的根节点 root，以及树中两个不同节点的值 x 和 y。
只有与值 x 和 y 对应的节点是堂兄弟节点时，才返回 true。否则，返回 false。
示例 1：输入：root = [1,2,3,4], x = 4, y = 3 输出：false
示例 2：输入：root = [1,2,3,null,4,null,5], x = 5, y = 4 输出：true
示例 3：输入：root = [1,2,3,null,4], x = 2, y = 3 输出：false

提示：
    二叉树的节点数介于 2 到 100 之间。
    每个节点的值都是唯一的、范围为 1 到 100 的整数。
```

- 解题思路

```
func isCousins(root *TreeNode, x int, y int) bool {
        xNode, xDepth := dfs(root, x, 0)
        yNode, yDepth := dfs(root, y, 0)
        return xDepth == yDepth && xNode != yNode
}

func dfs(root *TreeNode, value int, level int) (*TreeNode, int) {
        if root == nil {
                return nil, -1
        }
        if root.Val == value {
                return nil, 0
        }
        if (root.Left != nil && root.Left.Val == value) ||
                (root.Right != nil && root.Right.Val == value) {
                return root, level + 1
        }
        leftNode, leftLevel := dfs(root.Left, value, level+1)
        if leftNode != nil {
                return leftNode, leftLevel
```

(续下页)

(接上页)

```
        }
        return dfs(root.Right, value, level+1)
}

#
func isCousins(root *TreeNode, x int, y int) bool {
        if root == nil {
                return true
        }
        fatherM := make(map[int]int)
        levelM := make(map[int]int)
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        level := 0
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        node := queue[i]
                        levelM[node.Val] = level
                        if node.Left != nil {
                                fatherM[node.Left.Val] = node.Val
                                queue = append(queue, node.Left)
                        }
                        if node.Right != nil {
                                fatherM[node.Right.Val] = node.Val
                                queue = append(queue, node.Right)
                        }
                }
                queue = queue[length:]
                level++
        }
        return levelM[x] == levelM[y] && fatherM[x] != fatherM[y]
}

#
var fatherM map[int]int
var levelM map[int]int

func isCousins(root *TreeNode, x int, y int) bool {
        fatherM = make(map[int]int)
        levelM = make(map[int]int)
        dfs(root, nil, 0)
        return levelM[x] == levelM[y] && fatherM[x] != fatherM[y]
```

(续下页)

(接上页)

```
}

func dfs(root *TreeNode, father *TreeNode, level int) {
        if root == nil {
                return
        }
        if father == nil {
                fatherM[root.Val] = 0
        } else {
                fatherM[root.Val] = father.Val
        }
        levelM[root.Val] = level
        dfs(root.Left, root, level+1)
        dfs(root.Right, root, level+1)
}
```

28.24 997. 找到小镇的法官 (2)

- 题目

```
在一个小镇里，按从 1 到 N 标记了 N 个人。传言称，这些人中有一个是小镇上的秘密法官。
如果小镇的法官真的存在，那么：
    小镇的法官不相信任何人。
    每个人（除了小镇法官外）都信任小镇的法官。
    只有一个人同时满足属性 1 和属性 2 。
给定数组 trust，该数组由信任对 trust[i] = [a, b] 组成，表示标记为 a 的人信任标记为 b␣
↪的人。
如果小镇存在秘密法官并且可以确定他的身份，请返回该法官的标记。否则，返回 -1。
示例 1：输入：N = 2, trust = [[1,2]] 输出：2
示例 2：输入：N = 3, trust = [[1,3],[2,3]] 输出：3
示例 3：输入：N = 3, trust = [[1,3],[2,3],[3,1]] 输出：-1
示例 4：输入：N = 3, trust = [[1,2],[2,3]] 输出：-1
示例 5：输入：N = 4, trust = [[1,3],[1,4],[2,3],[2,4],[4,3]] 输出：3
提示：
    1 <= N <= 1000
    trust.length <= 10000
    trust[i] 是完全不同的
    trust[i][0] != trust[i][1]
    1 <= trust[i][0], trust[i][1] <= N
```

- 解题思路

```
func findJudge(N int, trust [][]int) int {
        arr := make([]int, N+1)
        for i := range trust {
                arr[trust[i][0]] = -1
                if arr[trust[i][1]] == -1 {
                        continue
                }
                arr[trust[i][1]]++
        }
        for i := 1; i <= N; i++ {
                if arr[i] == N-1 {
                        return i
                }
        }
        return -1
}

#
func findJudge(N int, trust [][]int) int {
        out := make([]int, N+1)
        in := make([]int, N+1)
        for i := range trust {
                out[trust[i][0]] = -1
                in[trust[i][1]]++
        }
        for i := 1; i <= N; i++ {
                // 出度为0，入度为N-1
                if out[i] == 0 && in[i] == N-1 {
                        return i
                }
        }
        return -1
}
```

28.25 999. 可以被一步捕获的棋子数 (2)

- 题目

```
在一个 8 x 8 的棋盘上，有一个白色的车（Rook），用字符 'R'␣
↪表示。棋盘上还可能存在空方块，
白色的象（Bishop）以及黑色的卒（pawn），分别用字符 '.', 'B' 和 'p' 表示。
不难看出，大写字符表示的是白棋，小写字符表示的是黑棋。
```

(续下页)

(接上页)

```
车按国际象棋中的规则移动。东，西，南，北四个基本方向任选其一，然后一直向选定的方向移动，
直到满足下列四个条件之一：
    棋手选择主动停下来。
    棋子因到达棋盘的边缘而停下。
    棋子移动到某一方格来捕获位于该方格上敌方（黑色）的卒，停在该方格内。
    车不能进入/越过已经放有其他友方棋子（白色的象）的方格，停在友方棋子前。
你现在可以控制车移动一次，请你统计有多少敌方的卒处于你的捕获范围内（即，可以被一步捕获的棋子数）。
示例 1：输入：
[[".",".",".",".",".",".",".",".","."],[".",".",".",".","p",".",".",".",".","."],
[".",".",".","R",".",".",".","p"],[".",".",".",".",".",".",".",".",".","."],
[".",".",".",".",".",".",".",".","."],[".",".",".",".","p",".",".",".",".","."],
[".",".",".",".",".",".",".",".","."],[".",".",".",".",".",".",".",".",".","."]]
输出：3
解释：在本例中，车能够捕获所有的卒。
示例 2：输入：
[[".",".",".",".",".",".",".",".","."],[".","p","p","p","p","p",".",".","."],
[".","p","p","B","p","p",".",".","."],[".","p","B","R","B","p",".",".","."],
[".","p","p","B","p","p",".",".","."],[".","p","p","p","p","p",".",".","."],
[".",".",".",".",".",".",".",".","."],[".",".",".",".",".",".",".",".",".","."]]
输出：0
解释：象阻止了车捕获任何卒。
示例 3：输入：
[[".",".",".",".",".",".",".",".","."],[".",".",".",".","p",".",".",".",".","."],
[".",".",".","p",".",".",".",".","."],["p","p",".","R",".","p","B",".","."],
[".",".",".",".",".",".",".",".","."],[".",".",".","B",".",".",".",".","."],
[".",".",".","p",".",".",".",".","."],[".",".",".",".",".",".",".",".",".","."]]
输出：3
解释：  车可以捕获位置 b5, d6 和 f5 的卒。
提示：
    board.length == board[i].length == 8
    board[i][j] 可以是 'R', '.', 'B' 或 'p'
    只有一个格子上存在 board[i][j] == 'R'
```

- 解题思路

```
func numRookCaptures(board [][]byte) int {
        res := 0
        var x, y int
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[0]); j++ {
                        if board[i][j] == 'R' {
                                x = i
                                y = j
```

(续下页)

(接上页)

```
                                break
                        }
                }
        }
        // 向右
        for i := y; i < 8 && board[x][i] != 'B'; i++ {
                if board[x][i] == 'p' {
                        res++
                        break
                }
        }
        // 向左
        for i := y; i >= 0 && board[x][i] != 'B'; i-- {
                if board[x][i] == 'p' {
                        res++
                        break
                }
        }
        // 向下
        for i := x; i < 8 && board[i][y] != 'B'; i++ {
                if board[i][y] == 'p' {
                        res++
                        break
                }
        }
        // 向上
        for i := x; i >= 0 && board[i][y] != 'B'; i-- {
                if board[i][y] == 'p' {
                        res++
                        break
                }
        }
        return res
}

#
func numRookCaptures(board [][]byte) int {
        res := 0
        var x, y int
        var dx = []int{-1, 1, 0, 0}
        var dy = []int{0, 0, -1, 1}
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[0]); j++ {
```

(续下页)

(接上页)

```
                        if board[i][j] == 'R' {
                                x = i
                                y = j
                                break
                        }
                }
        }
        for i := 0; i < 4; i++ {
                newX := x + dx[i]
                newY := y + dy[i]
                for newX >= 0 && newX < len(board) && newY >= 0 && newY <␣
↪len(board[0]) {
                        if board[newX][newY] == 'B' {
                                break
                        }
                        if board[newX][newY] == 'p' {
                                res++
                                break
                        }
                        newX = newX + dx[i]
                        newY = newY + dy[i]
                }
        }
        return res
}
```

CHAPTER 29

0901-1000-Medium

29.1 901. 股票价格跨度 (1)

- 题目

```
编写一个 StockSpanner 类，它收集某些股票的每日报价，并返回该股票当日价格的跨度。
今天股票价格的跨度被定义为股票价格小于或等于今天价格的最大连续日数（从今天开始往回数，包括今天）。
例如，如果未来7天股票的价格是 [100, 80, 60, 70, 60, 75, 85]，
那么股票跨度将是 [1, 1, 1, 2, 1, 4, 6]。
示例：
输入：["StockSpanner","next","next","next","next","next","next","next"],
[[],[100],[80],[60],[70],[60],[75],[85]]
输出：[null,1,1,1,2,1,4,6]
解释：首先，初始化 S = StockSpanner()，然后：
S.next(100) 被调用并返回 1，
S.next(80) 被调用并返回 1，
S.next(60) 被调用并返回 1，
S.next(70) 被调用并返回 2，
S.next(60) 被调用并返回 1，
S.next(75) 被调用并返回 4，
S.next(85) 被调用并返回 6。
注意（例如）S.next(75) 返回 4，因为截至今天的最后 4 个价格
(包括今天的价格 75) 小于或等于今天的价格。
提示：
    调用 StockSpanner.next(int price) 时，将有 1 <= price <= 10^5。
```

(续下页)

(接上页)

```
    每个测试用例最多可以调用  10000 次 StockSpanner.next。
    在所有测试用例中，最多调用 150000 次 StockSpanner.next。
    此问题的总时间限制减少了 50%。
```

- 解题思路

```
type StockSpanner struct {
        prices []int
        count  []int
}

func Constructor() StockSpanner {
        return StockSpanner{
                prices: make([]int, 0),
                count:  make([]int, 0),
        }
}

func (this *StockSpanner) Next(price int) int {
        res := 1
        for len(this.prices) > 0 && this.prices[len(this.prices)-1] <= price {
                this.prices = this.prices[:len(this.prices)-1]
                temp := this.count[len(this.count)-1]
                this.count = this.count[:len(this.count)-1]
                res = res + temp
        }
        this.prices = append(this.prices, price)
        this.count = append(this.count, res)
        return res
}
```

29.2 904. 水果成篮 (1)

- 题目

```
在一排树中，第 i 棵树产生tree[i] 型的水果。
你可以从你选择的任何树开始，然后重复执行以下步骤：
把这棵树上的水果放进你的篮子里。如果你做不到，就停下来。
移动到当前树右侧的下一棵树。如果右边没有树，就停下来。
请注意，在选择一颗树后，你没有任何选择：
你必须执行步骤 1，然后执行步骤 2，然后返回步骤 1，然后执行步骤 2，依此类推，直至停止。
你有两个篮子，每个篮子可以携带任何数量的水果，但你希望每个篮子只携带一种类型的水果。
```

(续下页)

(接上页)

```
用这个程序你能收集的水果树的最大总量是多少？
示例 1：输入：[1,2,1] 输出：3
解释：我们可以收集 [1,2,1]。
示例 2：输入：[0,1,2,2] 输出：3
解释：我们可以收集 [1,2,2]
如果我们从第一棵树开始，我们将只能收集到 [0, 1]。
示例 3：输入：[1,2,3,2,2] 输出：4
解释：我们可以收集 [2,3,2,2]
如果我们从第一棵树开始，我们将只能收集到 [1, 2]。
示例 4：输入：[3,3,3,1,2,1,1,2,3,3,4] 输出：5
解释：我们可以收集 [1,2,1,1,2]
如果我们从第一棵树或第八棵树开始，我们将只能收集到 4 棵水果树。
提示：1 <= tree.length <= 40000
0 <= tree[i] < tree.length
```

- 解题思路

```
func totalFruit(tree []int) int {
        res := 0
        m := make(map[int]int)
        left := 0
        total := 0
        for i := 0; i < len(tree); i++ {
                target := tree[i]
                if _, ok := m[target]; ok {
                        m[target]++
                } else {
                        for len(m) >= 2 {
                                m[tree[left]]--
                                total--
                                if m[tree[left]] == 0 {
                                        delete(m, tree[left])
                                }
                                left++
                        }
                        m[target] = 1
                }
                total++
                if total > res {
                        res = total
                }
        }
        return res
}
```

29.3 907. 子数组的最小值之和 (3)

- 题目

```
给定一个整数数组 A，找到 min(B)的总和，其中 B 的范围为A 的每个（连续）子数组。
由于答案可能很大，因此返回答案模 10^9 + 7。
示例：输入：[3,1,2,4] 输出：17
解释：子数组为 [3]，[1]，[2]，[4]，[3,1]，[1,2]，[2,4]，[3,1,2]，[1,2,4]，[3,1,2,4]。
最小值为 3，1，2，4，1，1，2，1，1，1，和为 17。
提示：1 <= A <= 30000
1 <= A[i] <= 30000
```

- 解题思路

```
func sumSubarrayMins(arr []int) int {
        if len(arr) == 0 {
                return 0
        }
        if len(arr) == 1 {
                return arr[0]
        }
        index := findMinValue(arr)
        left := index
        right := len(arr) - 1 - index
        //␣
↪1是自己，left是左边left个数+自己，right是右边right个数+自己，left*right是左边+自己+右边的个数
        count := 1 + left + right + left*right
        res := arr[index] * count % 1000000007
        res = res + sumSubarrayMins(arr[:index]) // 左边
        if index < len(arr)-1 {
                res = res + sumSubarrayMins(arr[index+1:]) // 右边
        }
        return res % 1000000007
}

func findMinValue(arr []int) int {
        minValue := arr[0]
        minIndex := 0
        for i := 1; i < len(arr); i++ {
                if arr[i] < minValue {
                        minValue = arr[i]
                        minIndex = i
                }
        }
```

(续下页)

(接上页)

```
        return minIndex
}

# 2
func sumSubarrayMins(arr []int) int {
        res := 0
        stack := make([]int, 0) // 递增栈
        stack = append(stack, -1)
        total := 0
        for i := 0; i < len(arr); i++ {
                for len(stack) > 1 && arr[i] < arr[stack[len(stack)-1]] { // 小于栈顶
                        prev := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        total = total - arr[prev]*(prev-stack[len(stack)-1])
                }
                stack = append(stack, i)
                total = total + arr[i]*(i-stack[len(stack)-2])
                res = (res + total) % 1000000007
        }
        return res % 1000000007
}

# 3
func sumSubarrayMins(arr []int) int {
        res := 0
        stack := make([][2]int, 0) // 递增栈
        sum := 0
        for i := 0; i < len(arr); i++ {
                count := 1
                for len(stack) > 0 && arr[i] < stack[len(stack)-1][0] { // 小于栈顶
                        prev := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        sum = sum - prev[0]*prev[1]
                        count = count + prev[1]
                }
                stack = append(stack, [2]int{arr[i], count})
                sum = sum + arr[i]*count
                res = (res + sum) % 1000000007
        }
        return res % 1000000007
}

# 4
```

(续下页)

(接上页)

```
func sumSubarrayMins(arr []int) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                left, right := i-1, i+1
                for ; left >= 0; left-- {
                        if arr[left] < arr[i] {
                                break
                        }
                }
                for ; right < len(arr); right++ {
                        if arr[right] <= arr[i] { // 注意边界
                                break
                        }
                }
                sum := arr[i] * (i - left) * (right - i)
                res = (res + sum) % 1000000007
        }
        return res % 1000000007
}
```

29.4 909. 蛇梯棋 (1)

- 题目

```
N x N 的棋盘board 上，按从1 到 N*N的数字给方格编号，编号␣
↪从左下角开始，每一行交替方向。
例如，一块 6 x 6 大小的棋盘，编号如下：
r 行 c 列的棋盘，按前述方法编号，棋盘格中可能存在 “蛇” 或 “梯子”；
如果 board[r][c] != -1，那个蛇或梯子的目的地将会是 board[r][c]。
玩家从棋盘上的方格1 （总是在最后一行、第一列）开始出发。
每一回合，玩家需要从当前方格 x 开始出发，按下述要求前进：
选定目标方格：选择从编号 x+1，x+2，x+3，x+4，x+5，或者 x+6 的方格中选出一个目标方格 s␣
↪，目标方格的编号 <= N*N。
该选择模拟了掷骰子的情景，无论棋盘大小如何，你的目的地范围也只能处于区间 [x+1, x+6]␣
↪之间。
传送玩家：如果目标方格 S␣
↪处存在蛇或梯子，那么玩家会传送到蛇或梯子的目的地。否则，玩家传送到目标方格 S。
注意，玩家在每回合的前进过程中最多只能爬过蛇或梯子一次：就算目的地是另一条蛇或梯子的起点，你也不会继
返回达到方格 N*N 所需的最少移动次数，如果不可能，则返回 -1。
示例：输入：[
[-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1],
```

(续下页)

(接上页)

```
[-1,-1,-1,-1,-1,-1],
[-1,35,-1,-1,13,-1],
[-1,-1,-1,-1,-1,-1],
[-1,15,-1,-1,-1,-1]]
输出：4
解释：首先，从方格 1 [第 5 行，第 0 列] 开始。
你决定移动到方格 2，并必须爬过梯子移动到到方格 15。
然后你决定移动到方格 17 [第 3 行，第 5 列]，必须爬过蛇到方格 13。
然后你决定移动到方格 14，且必须通过梯子移动到方格 35。
然后你决定移动到方格 36，游戏结束。
可以证明你需要至少 4 次移动才能到达第 N*N 个方格，所以答案是 4。
提示：2 <= board.length = board[0].length<= 20
board[i][j]介于1和N*N之间或者等于-1。
编号为1的方格上没有蛇或梯子。
编号为N*N的方格上没有蛇或梯子。
```

- 解题思路

```
func snakesAndLadders(board [][]int) int {
        n := len(board)
        m := make(map[int]int) // 保存到达坐标对应的移动次数
        m[1] = 0
        queue := make([]int, 0)
        queue = append(queue, 1)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node == n*n {
                        return m[node]
                }
                for i := node + 1; i <= node+6 && i <= n*n; i++ {
                        _, a, b := getIndex(i, n)
                        next := -1
                        if board[a][b] == -1 {
                                next = i
                        } else {
                                next = board[a][b]
                        }
                        if _, ok := m[next]; !ok {
                                m[next] = m[node] + 1
                                queue = append(queue, next)
                        }
                }
        }
```

(续下页)

(接上页)

```
        return -1
}

func getIndex(i int, n int) (int, int, int) {
        var x, y int
        x = (i - 1) / n // 行
        y = (i - 1) % n // 列
        if x%2 == 1 { // 奇数行需要反转
                y = n - 1 - y // 反转
        }
        x = n - 1 - x // 反转
        return x*n + y, x, y
}
```

29.5 910. 最小差值 II(1)

- 题目

```
给你一个整数数组 A，对于每个整数 A[i]，可以选择x = -K或是x = K （K 总是非负整数），
并将x加到A[i]中。
在此过程之后，得到数组B。
返回 B的最大值和 B的最小值之间可能存在的最小差值。
示例 1：输入：A = [1], K = 0 输出：0
解释：B = [1]
示例 2：输入：A = [0,10], K = 2 输出：6
解释：B = [2,8]
示例 3：输入：A = [1,3,6], K = 3 输出：3
解释：B = [4,6,3]
提示：1 <= A.length <= 10000
0 <= A[i] <= 10000
0 <= K <= 10000
```

- 解题思路

```
func smallestRangeII(A []int, K int) int {
        sort.Ints(A)
        n := len(A)
        res := A[n-1] - A[0]
        // 排序后，为了最小差值，必定是A[0,i]+K,A[i+1:]-K
        // 最小值落在A[0]+K, A[i+1]-K之中
        // 最大值落在A[n-1]-K, A[i]+K之中
        for i := 0; i < n-1; i++ {
```

(续下页)

(接上页)

```
                minValue := min(A[0]+K, A[i+1]-K)
                maxValue := max(A[n-1]-K, A[i]+K)
                res = min(maxValue-minValue, res)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

29.6 911. 在线选举 (2)

- 题目

```
在选举中，第i张票是在时间为times[i]时投给persons[i]的。
现在，我们想要实现下面的查询函数： TopVotedCandidate.q(int t) 将返回在t␣
↪时刻主导选举的候选人的编号。
在t␣
↪时刻投出的选票也将被计入我们的查询之中。在平局的情况下，最近获得投票的候选人将会获胜。
示例：输入：["TopVotedCandidate","q","q","q","q","q","q"],
[[[0,1,1,0,0,1,0],[0,5,10,15,20,25,30]],[3],[12],[25],[15],[24],[8]]
输出：[null,0,1,1,0,0,1]
解释：时间为 3，票数分布情况是 [0]，编号为 0 的候选人领先。
时间为 12，票数分布情况是 [0,1,1]，编号为 1 的候选人领先。
时间为 25，票数分布情况是 [0,1,1,0,0,1]，编号为 1␣
↪的候选人领先（因为最近的投票结果是平局）。
在时间 15、24 和 8 处继续执行 3 个查询。
提示：1 <= persons.length = times.length <= 5000
0 <= persons[i] <= persons.length
times是严格递增的数组，所有元素都在[0, 10^9]范围中。
每个测试用例最多调用10000次TopVotedCandidate.q。
```

(续下页)

(接上页)

TopVotedCandidate.q(int t)被调用时总是满足t >= times[0]。

- 解题思路

```
type TopVotedCandidate struct {
        result []int // 存放对应时间的候选人
        times  []int
}

func Constructor(persons []int, times []int) TopVotedCandidate {
        n := len(persons)
        arr := make([]int, n+1)
        maxCount := 0
        index := persons[0]
        res := make([]int, n)
        for i := 0; i < n; i++ {
                id := persons[i]
                arr[id]++
                if arr[id] >= maxCount {
                        maxCount = arr[id]
                        index = id
                }
                res[i] = index
        }
        return TopVotedCandidate{
                result: res,
                times:  times,
        }
}

func (this *TopVotedCandidate) Q(t int) int {
        left, right := 0, len(this.times)
        for left < right {
                mid := left + (right-left)/2
                if this.times[mid] == t {
                        return this.result[mid]
                } else if this.times[mid] > t {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return this.result[left-1]
}
```

(续下页)

(接上页)

```
# 2
type TopVotedCandidate struct {
        result []int // 存放对应时间的候选人
        times  []int
}

func Constructor(persons []int, times []int) TopVotedCandidate {
        n := len(persons)
        arr := make([]int, n+1)
        maxCount := 0
        index := persons[0]
        res := make([]int, n)
        for i := 0; i < n; i++ {
                id := persons[i]
                arr[id]++
                if arr[id] >= maxCount {
                        maxCount = arr[id]
                        index = id
                }
                res[i] = index
        }
        return TopVotedCandidate{
                result: res,
                times:  times,
        }
}

func (this *TopVotedCandidate) Q(t int) int {
        left, right := 0, len(this.times)-1
        for left <= right {
                mid := left + (right-left)/2
                if this.times[mid] > t {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return this.result[left-1]
}
```

29.7 912. 排序数组 (7)

- 题目

```
给你一个整数数组 nums，请你将该数组升序排列。
示例 1：输入：nums = [5,2,3,1] 输出：[1,2,3,5]
示例 2：输入：nums = [5,1,1,2,0,0] 输出：[0,0,1,1,2,5]
提示：1 <= nums.length <= 50000
    -50000 <= nums[i] <= 50000
```

- 解题思路

```
func sortArray(nums []int) []int {
        sort.Ints(nums)
        return nums
}

# 2
func sortArray(nums []int) []int {
        for i := 1; i < len(nums); i++ {
                pos := i - 1 // 已经有序的最后一个下标
                cur := nums[i]
                for pos >= 0 && nums[pos] > cur {
                        nums[pos+1] = nums[pos] // 后移
                        pos--
                }
                nums[pos+1] = cur
        }
        return nums
}

# 3
func sortArray(nums []int) []int {
        n := len(nums)
        for gap := n / 2; gap > 0; gap = gap / 2 {
                for i := gap; i < n; i++ {
                        j := i
                        cur := nums[i]
                        for j-gap >= 0 && cur < nums[j-gap] {
                                nums[j] = nums[j-gap]
                                j = j - gap
                        }
                        nums[j] = cur
                }
```

(续下页)

(接上页)

```
        }
        return nums
}

# 4
func sortArray(nums []int) []int {
        n := len(nums)
        if n < 2 {
                return nums
        }
        return merge(sortArray(nums[:len(nums)/2]), sortArray(nums[len(nums)/2:]))
}

func merge(left, right []int) []int {
        res := make([]int, 0)
        for len(left) > 0 && len(right) > 0 {
                if left[0] <= right[0] {
                        res = append(res, left[0])
                        left = left[1:]
                } else {
                        res = append(res, right[0])
                        right = right[1:]
                }
        }
        if len(left) > 0 {
                res = append(res, left...)
        }
        if len(right) > 0 {
                res = append(res, right...)
        }
        return res
}

# 5
func sortArray(nums []int) []int {
        quick(nums, 0, len(nums)-1)
        return nums
}

func quick(arr []int, left, right int) {
        if left >= right {
                return
        }
```

(续下页)

(接上页)

```
        index := partition(arr, left, right)
        quick(arr, left, index-1)
        quick(arr, index+1, right)
}

func partition(arr []int, left, right int) int {
        baseValue := arr[left] // 基准值
        for left < right {
                for baseValue <= arr[right] && left < right {
                        right-- // 依次查找大于基准值的位置
                }
                arr[left] = arr[right]
                for arr[left] <= baseValue && left < right {
                        left++ // 依次查找小于基准值的位置
                }
                arr[right] = arr[left]
        }
        arr[right] = baseValue
        return right
}

# 6
type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
```

(续下页)

(接上页)

```
        *h = (*h)[:len(*h)-1]
        return value
}

func sortArray(nums []int) []int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < len(nums); i++ {
                heap.Push(&intHeap, nums[i])
        }
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                value := heap.Pop(&intHeap).(int)
                res = append(res, value)
        }
        return res
}

# 7
func sortArray(nums []int) []int {
        buildHeap(nums, len(nums))
        for i := len(nums) - 1; i > 0; i-- {
                nums[0], nums[i] = nums[i], nums[0]
                heapModify(nums, 0, i-1)
        }
        return nums
}

func buildHeap(arr []int, length int) {
        for i := len(arr)/2 - 1; i >= 0; i-- {
                heapModify(arr, i, length-1)
        }
}

func heapModify(arr []int, start, end int) {
        temp := arr[start]
        for left := 2*start + 1; left <= end; left = 2*left + 1 {
                if left < end && arr[left] < arr[left+1] {
                        left++
                }
                if arr[start] < arr[left] {
                        arr[start] = arr[left]
                        start = left
```

(续下页)

(接上页)

```
                }
                arr[start] = temp
        }
}
```

29.8 915. 分割数组 (2)

- 题目

```
给定一个数组A，将其划分为两个不相交（没有公共元素）的连续子数组left和right，使得：
left中的每个元素都小于或等于right中的每个元素。
left 和right都是非空的。
left要尽可能小。
在完成这样的分组后返回left的长度。可以保证存在这样的划分方法。
示例 1：输入：[5,0,3,8,6] 输出：3
解释：left = [5,0,3]，right = [8,6]
示例 2：输入：[1,1,1,0,6,12] 输出：4
解释：left = [1,1,1,0]，right = [6,12]
提示：2 <= A.length<= 30000
0 <= A[i] <= 10^6
可以保证至少有一种方法能够按题目所描述的那样对 A 进行划分。
```

- 解题思路

```
func partitionDisjoint(A []int) int {
        n := len(A)
        maxLeft := make([]int, n)
        minRight := make([]int, n)
        maxValue := A[0]
        for i := 0; i < n; i++ {
                if maxValue < A[i] {
                        maxValue = A[i]
                }
                maxLeft[i] = maxValue
        }
        minValue := A[n-1]
        for i := n - 1; i >= 0; i-- {
                if minValue > A[i] {
                        minValue = A[i]
                }
                minRight[i] = minValue
        }
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                if maxLeft[i-1] <= minRight[i] {
                        return i
                }
        }
        return -1
}

# 2
func partitionDisjoint(A []int) int {
        n := len(A)
        res := 0
        maxLeft := A[0]
        maxValue := A[0]
        for i := 1; i < n; i++ {
                maxValue = max(maxValue, A[i])
                if A[i] < maxLeft {
                        res = i
                        maxLeft = maxValue
                }
        }
        return res + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

29.9 916. 单词子集 (1)

- 题目

```
我们给出两个单词数组 A和B。每个单词都是一串小写字母。
现在，如果b 中的每个字母都出现在 a 中，包括重复出现的字母，那么称单词 b 是单词 a␣
↪的子集。
例如，“wrr” 是 “warrior” 的子集，但不是 “world” 的子集。
如果对 B 中的每一个单词b，b 都是 a 的子集，那么我们称A 中的单词 a 是通用的。
你可以按任意顺序以列表形式返回A 中所有的通用单词。
示例 1： 输入：A = ["amazon","apple","facebook","google","leetcode"], B = ["e","o"]
```

(续下页)

(接上页)

```
输出：["facebook","google","leetcode"]
示例 2：输入：A = ["amazon","apple","facebook","google","leetcode"], B = ["l","e"]
输出：["apple","google","leetcode"]
示例 3：输入：A = ["amazon","apple","facebook","google","leetcode"], B = ["e","oo"]
输出：["facebook","google"]
示例 4：输入：A = ["amazon","apple","facebook","google","leetcode"], B = ["lo","eo"]
输出：["google","leetcode"]
示例 5：输入：A = ["amazon","apple","facebook","google","leetcode"], B = ["ec","oc",
↪"ceo"]
输出：["facebook","leetcode"]
提示：1 <= A.length, B.length <= 10000
1 <= A[i].length, B[i].length<= 10
A[i]和B[i]只由小写字母组成。
A[i]中所有的单词都是独一无二的，也就是说不存在i != j使得A[i] == A[j]。
```

- 解题思路

```
func wordSubsets(A []string, B []string) []string {
        maxArr := make([]int, 26) // B中的字母的最大频次
        for i := 0; i < len(B); i++ {
                temp := count(B[i])
                for i := 0; i < 26; i++ {
                        maxArr[i] = max(maxArr[i], temp[i])
                }
        }
        res := make([]string, 0)
        for i := 0; i < len(A); i++ {
                temp := count(A[i])
                flag := true
                for i := 0; i < 26; i++ {
                        if temp[i] < maxArr[i] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = append(res, A[i])
                }
        }
        return res
}

func count(str string) [26]int {
        arr := [26]int{}
```

(续下页)

(接上页)

```
        for i := 0; i < len(str); i++ {
                arr[str[i]-'a']++
        }
        return arr
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

29.10 918. 环形子数组的最大和 (1)

- 题目

```
给定一个由整数数组 A表示的环形数组 C，求 C的非空子数组的最大可能和。
在此处，环形数组意味着数组的末端将会与开头相连呈环状。
（形式上，当0 <= i < A.length时C[i] = A[i]，且当i >= 0时C[i+A.length] = C[i]）
此外，子数组最多只能包含固定缓冲区 A中的每个元素一次。
（形式上，对于子数组C[i], C[i+1], ..., C[j],
不存在i <= k1, k2 <= j其中k1 % A.length= k2 % A.length）
示例 1：输入：[1,-2,3,-2] 输出：3
解释：从子数组 [3] 得到最大和 3
示例 2：输入：[5,-3,5] 输出：10
解释：从子数组 [5,5] 得到最大和 5 + 5 = 10
示例 3：输入：[3,-1,2,-1] 输出：4
解释：从子数组 [2,-1,3] 得到最大和 2 + (-1) + 3 = 4
示例 4：输入：[3,-2,2,-3] 输出：3
解释：从子数组 [3] 和 [3,-2,2] 都可以得到最大和 3
示例 5：输入：[-2,-3,-1] 输出：-1
解释：从子数组 [-1] 得到最大和 -1
提示：-30000 <= A[i] <= 30000
1 <= A.length <= 30000
```

- 解题思路

```
func maxSubarraySumCircular(A []int) int {
        n := len(A)
        // leetcode53.最大子序和 的变形
        total := A[0]                   // 总和
```

(续下页)

(接上页)

```
        minValue, sumMin := A[0], A[0] // 找到连续数组之和的最小值
        maxValue, sumMax := A[0], A[0] // 找到连续数组之和的最大值
        for i := 1; i < n; i++ {
                total = total + A[i]
                sumMin = min(sumMin+A[i], A[i])
                minValue = min(minValue, sumMin)
                sumMax = max(sumMax+A[i], A[i])
                maxValue = max(maxValue, sumMax)
        }
        // 2种情况：取最大值即可
        // 1: 目标数组不需要环，求最大值
        // 2: 需要成环，总和减去最小值
        if total == minValue { // 极端情况全小于0：如果和=最小值，返回最大值
                return maxValue
        }
        return max(maxValue, total-minValue)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

29.11 919. 完全二叉树插入器 (1)

- 题目

```
完全二叉树是每一层（除最后一层外）都是完全填充（即，节点数达到最大）的，并且所有的节点都尽可能地集中
设计一个用完全二叉树初始化的数据结构CBTInserter，它支持以下几种操作：
CBTInserter(TreeNode root)使用头节点为root的给定树初始化该数据结构；
CBTInserter.insert(int v) 向树中插入一个新节点，节点类型为 TreeNode，值为 v 。
使树保持完全二叉树的状态，并返回插入的新节点的父节点的值；
CBTInserter.get_root() 将返回树的头节点。
```

(续下页)

(接上页)

```
示例 1: 输入: inputs = ["CBTInserter","insert","get_root"], inputs = [[[1]],[2],[]]␣
↪输出: [null,1,[1,2]]
示例 2: 输入: inputs = ["CBTInserter","insert","insert","get_root"], inputs = [[[1,2,
↪3,4,5,6]],[7],[8],[]]
输出: [null,3,4,[1,2,3,4,5,6,7,8]]
提示: 最初给定的树是完全二叉树, 且包含1到1000个节点。
每个测试用例最多调用CBTInserter.insert 操作10000次。
给定节点或插入节点的每个值都在0到5000之间。
```

- 解题思路

```
type CBTInserter struct {
        root *TreeNode
        arr  []*TreeNode
}

func Constructor(root *TreeNode) CBTInserter {
        arr := make([]*TreeNode, 0)
        queue := make([]*TreeNode, 0)
        arr = append(arr, root)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                                arr = append(arr, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                                arr = append(arr, queue[i].Right)
                        }
                }
                queue = queue[length:]
        }
        return CBTInserter{root: root, arr: arr}
}

func (this *CBTInserter) Insert(v int) int {
        newNode := &TreeNode{Val: v}
        this.arr = append(this.arr, newNode)
        n := len(this.arr)
        target := this.arr[n/2-1]
        if n%2 == 0 {
```

(续下页)

(接上页)

```
                target.Left = newNode
        } else {
                target.Right = newNode
        }
        return target.Val
}

func (this *CBTInserter) Get_root() *TreeNode {
        return this.root
}
```

29.12 921. 使括号有效的最少添加 (3)

- 题目

```
给定一个由 '(' 和 ')' 括号组成的字符串 S，我们需要添加最少的括号（ '(' 或是 ')
↪'，可以在任何位置），
以使得到的括号字符串有效。
从形式上讲，只有满足下面几点之一，括号字符串才是有效的：
    它是一个空字符串，或者
    它可以被写成 AB （A 与 B 连接），其中 A 和 B 都是有效字符串，或者
    它可以被写作 (A)，其中 A 是有效字符串。
给定一个括号字符串，返回为使结果字符串有效而必须添加的最少括号数。
示例 1：输入："())" 输出：1
示例 2：输入："(((" 输出：3
示例 3：输入："()" 输出：0
示例 4：输入："()))((" 输出：4
提示：S.length <= 1000  S 只包含 '(' 和 ')' 字符。
```

- 解题思路

```
func minAddToMakeValid(S string) int {
        stack := make([]byte,0)
        res := 0
        for i := 0; i < len(S); i++{
                if S[i] == '('{
                        stack = append(stack, '(')
                }else {
                        if len(stack) == 0{
                                res++
                        }else {
                                stack = stack[:len(stack)-1]
```

(续下页)

(接上页)

```
                        }
                }
        }
        return res+len(stack)
}

# 2
func minAddToMakeValid(S string) int {
        left := 0
        right := 0
        for i := 0; i < len(S); i++ {
                if S[i] == '(' {
                        left++
                } else {
                        if left > 0 {
                                left--
                        } else {
                                right++
                        }
                }
        }
        return left + right
}

# 3
func minAddToMakeValid(S string) int {
        for strings.Contains(S, "()") == true {
                S = strings.ReplaceAll(S, "()", "")
        }
        return len(S)
}
```

29.13 923. 三数之和的多种可能 (4)

- 题目

```
给定一个整数数组A，以及一个整数target作为目标值，
返回满足 i < j < k 且A[i] + A[j] + A[k] == target的元组i, j, k的数量。
由于结果会非常大，请返回 结果除以 10^9 + 7 的余数。
示例 1：输入：A = [1,1,2,2,3,3,4,4,5,5], target = 8 输出：20
解释：按值枚举（A[i], A[j], A[k]）：
(1, 2, 5) 出现 8 次；
```

(续下页)

(接上页)

```
(1, 3, 4) 出现 8 次；
(2, 2, 4) 出现 2 次；
(2, 3, 3) 出现 2 次。
示例 2：输入：A = [1,1,2,2,2,2], target = 5 输出：12
解释：A[i] = 1, A[j] = A[k] = 2 出现 12 次：
我们从 [1,1] 中选择一个 1，有 2 种情况，
从 [2,2,2,2] 中选出两个 2，有 6 种情况。
提示：3 <= A.length <= 3000
0 <= A[i] <= 100
0 <= target <= 300
```

- 解题思路

```
func threeSumMulti(arr []int, target int) int {
        res := 0
        sort.Ints(arr)
        for i := 0; i < len(arr)-2; i++ {
                targetTemp := target - arr[i]
                j := i + 1
                k := len(arr) - 1
                for j < k {
                        if arr[j]+arr[k] < targetTemp {
                                j++
                        } else if arr[j]+arr[k] > targetTemp {
                                k--
                        } else {
                                if arr[j] != arr[k] { // 2数不重复
                                        left, right := 1, 1
                                        for j+1 < k && arr[j] == arr[j+1] {
                                                left++
                                                j++
                                        }
                                        for j+1 < k && arr[k] == arr[k-1] {
                                                right++
                                                k--
                                        }
                                        res = (res + left*right) % 1000000007
                                        j++
                                        k--
                                } else { // 2数重复
                                        res = (res + (k-j+1)*(k-j)/2) % 1000000007
                                        break
                                }
                        }
```

(续下页)

(接上页)

```
                }
        }
        return res % 1000000007
}

# 2
func threeSumMulti(arr []int, target int) int {
        res := 0
        countArr := make([]int, 101)
        for i := 0; i < len(arr); i++ {
                countArr[arr[i]]++
        }
        for i := 0; i <= 100; i++ {
                // i < j < k
                for j := i + 1; j <= 100; j++ {
                        k := target - i - j
                        if j < k && k <= 100 {
                                res = (res + countArr[i]*countArr[j]*countArr[k]) %␣
↪1000000007
                        }
                }
                // i == j < k
                k := target - 2*i
                if i < k && k <= 100 {
                        res = (res + countArr[i]*(countArr[i]-1)/2*countArr[k]) %␣
↪1000000007
                }
                // i < j == k
                if (target-i)%2 == 0 {
                        j := (target - i) / 2
                        if i < j && j <= 100 {
                                res = (res + countArr[i]*countArr[j]*(countArr[j]-1)/
↪2) % 1000000007
                        }
                }
        }
        // i==j==k
        if target%3 == 0 {
                i := target / 3
                if 0 <= i && i <= 100 {
                        res = (res + countArr[i]*(countArr[i]-1)*(countArr[i]-2)/6) %␣
↪1000000007
                }
```

(续下页)

(接上页)

```
        }
        return res % 1000000007
}

# 3
func threeSumMulti(arr []int, target int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                res = (res + m[target-arr[i]]) % 1000000007 // 以当前i为k, 求有多少i+j
                for j := 0; j < i; j++ {
                        m[arr[i]+arr[j]]++
                }
        }
        return res % 1000000007
}

# 4
func threeSumMulti(arr []int, target int) int {
        n := len(arr)
        dp := make([][4][]int, n+1) // 虑前i个数时, 从中选出j个数, 组成k大小的方案数
        for i := 0; i <= n; i++ {
                dp[i] = [4][]int{}
                for j := 0; j < 4; j++ {
                        dp[i][j] = make([]int, target+1)
                }
        }
        for i := 0; i <= n; i++ {
                dp[i][0][0] = 1
        }
        for i := 1; i <= n; i++ {
                for j := 1; j < 4; j++ {
                        for k := 0; k <= target; k++ {
                                if k >= arr[i-1] { // 可选
                                        dp[i][j][k] = (dp[i][j][k] + dp[i-1][j-1][k-
↪arr[i-1]]) % 1000000007
                                }
                                dp[i][j][k] = (dp[i][j][k] + dp[i-1][j][k]) %␣
↪1000000007 // 不选
                        }
                }
        }
        return dp[n][3][target]
}
```

29.14 926. 将字符串翻转到单调递增 (3)

- 题目

```
如果一个由'0' 和 '1'组成的字符串，
是以一些 '0'（可能没有 '0'）后面跟着一些 '1'（也可能没有 '1
↪'）的形式组成的，那么该字符串是单调递增的。
我们给出一个由字符 '0' 和 '1'组成的字符串S，我们可以将任何'0' 翻转为'1'或者将'1'翻转为
↪'0'。
返回使 S 单调递增的最小翻转次数。
示例 1：输入："00110" 输出：1
解释：我们翻转最后一位得到 00111.
示例 2：输入："010110" 输出：2
解释：我们翻转得到 011111，或者是 000111。
示例 3：输入："00011000" 输出：2
解释：我们翻转得到 00000000。
提示：1 <= S.length <= 20000
S 中只包含字符'0'和'1'
```

- 解题思路

```
func minFlipsMonoIncr(S string) int {
    n := len(S)
    dpA := make([]int, n) // 0 结尾
    dpB := make([]int, n) // 1 结尾
    if S[0] == '1' {
            dpA[0] = 1
    } else {
            dpB[0] = 1
    }
    for i := 1; i < n; i++ {
            if S[i] == '1' {
                    dpA[i] = dpA[i-1] + 1             // 需要改为0
                    dpB[i] = min(dpB[i-1], dpA[i-1]) // 1结尾和0结尾的最小值
            } else {
                    dpA[i] = dpA[i-1]                    // 不需要改为0
                    dpB[i] = min(dpB[i-1], dpA[i-1]) + 1 // 1结尾和0结尾的最小值+1
            }
    }
    return min(dpA[n-1], dpB[n-1])
}

func min(a, b int) int {
    if a > b {
```

(续下页)

(接上页)

```
            return b
    }
    return a
}

# 2
func minFlipsMonoIncr(S string) int {
    n := len(S)
    a := 0 // 0 结尾
    b := 0 // 1 结尾
    if S[0] == '1' {
            a = 1
    } else {
            b = 1
    }
    for i := 1; i < n; i++ {
            if S[i] == '1' {
                    a, b = a+1, min(a, b)
            } else {
                    a, b = a, min(a, b)+1
            }
    }
    return min(a, b)
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
}

# 3
func minFlipsMonoIncr(S string) int {
    n := len(S)
    arr := make([]int, n+1)
    for i := 1; i <= n; i++ {
            arr[i] = arr[i-1]
            if S[i-1] == '1' {
                    arr[i]++
            }
    }
    res := n
```

(续下页)

(接上页)

```
    for i := 0; i <= n; i++ {
            left := arr[i]
            right := n - i - (arr[n] - arr[i])
            res = min(res, left+right)
    }
    return res
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
}
```

29.15 930. 和相同的二元子数组 (3)

- 题目

```
在由若干0和1 组成的数组A中，有多少个和为 S的非空子数组。
示例：输入：A = [1,0,1,0,1], S = 2 输出：4
解释： 如下面黑体所示，有 4 个满足题目要求的子数组：
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
提示：A.length <= 30000
0 <= S <= A.length
A[i]为0或1
```

- 解题思路

```
func numSubarraysWithSum(A []int, S int) int {
        m := make(map[int]int)
        res := 0
        sum := 0
        for i := 0; i < len(A); i++ {
                sum = sum + A[i]
                if sum == S {
                        res++
                }
                res = res + m[sum-S]
```

(续下页)

(接上页)

```
                m[sum]++
        }
        return res
}

# 2
func numSubarraysWithSum(A []int, S int) int {
        m := make(map[int]int)
        res := 0
        sum := 0
        m[0] = 1
        for i := 0; i < len(A); i++ {
                sum = sum + A[i]
                res = res + m[sum-S]
                m[sum]++
        }
        return res
}

# 3
func numSubarraysWithSum(A []int, S int) int {
        res := 0
        sum := 0
        left := 0
        for i := 0; i < len(A); i++ {
                sum = sum + A[i]
                if sum > S { // 左指针右移
                        for left < i && sum != S {
                                sum = sum - A[left]
                                left++
                        }
                }
                if S == sum {
                        tempSum := sum
                        j := left
                        for j <= i && tempSum == S { //␣
↪以该节点为结尾有多少个子数组sum==S
                                res++
                                tempSum = tempSum - A[j]
                                j++
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

29.16 931. 下降路径最小和 (2)

- 题目

```
给你一个 n x n 的 方形 整数数组matrix ，请你找出并返回通过 matrix 的下降路径 的
↪最小和 。
下降路径 可以从第一行中的任何元素开始，并从每一行中选择一个元素。
在下一行选择的元素和当前行所选元素最多相隔一列（即位于正下方或者沿对角线向左或者向右的第一个元素）。
具体来说，位置 (row, col) 的下一个元素应当是
(row + 1, col - 1)、(row + 1, col) 或者 (row + 1, col + 1) 。
示例 1：输入：matrix = [[2,1,3],[6,5,4],[7,8,9]] 输出：13
解释：下面是两条和最小的下降路径，用加粗标注：
[[2,1,3],      [[2,1,3],
 [6,5,4],       [6,5,4],
 [7,8,9]]       [7,8,9]]
示例 2：输入：matrix = [[-19,57],[-40,-5]] 输出：-59
解释：下面是一条和最小的下降路径，用加粗标注：
[[-19,57],
 [-40,-5]]
示例 3：输入：matrix = [[-48]] 输出：-48
提示：n == matrix.length
n == matrix[i].length
1 <= n <= 100
-100 <= matrix[i][j] <= 100
```

- 解题思路

```
func minFallingPathSum(matrix [][]int) int {
        n := len(matrix)
        for i := 1; i < n; i++ {
                for j := 0; j < n; j++ {
                        minValue := matrix[i-1][j]
                        if j > 0 {
                                minValue = min(minValue, matrix[i-1][j-1])
                        }
                        if j < n-1 {
                                minValue = min(minValue, matrix[i-1][j+1])
                        }
                        matrix[i][j] = matrix[i][j] + minValue
```

(续下页)

(接上页)

```
                }
        }
        res := matrix[n-1][0]
        for i := 0; i < n; i++ {
                res = min(res, matrix[n-1][i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minFallingPathSum(matrix [][]int) int {
        n := len(matrix)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if i == 0 {
                                dp[i][j] = matrix[i][j]
                                continue
                        }
                        minValue := dp[i-1][j]
                        if j > 0 {
                                minValue = min(minValue, dp[i-1][j-1])
                        }
                        if j < n-1 {
                                minValue = min(minValue, dp[i-1][j+1])
                        }
                        dp[i][j] = dp[i][j] + minValue + matrix[i][j]
                }
        }
        res := dp[n-1][0]
        for i := 0; i < n; i++ {
                res = min(res, dp[n-1][i])
        }
```

(续下页)

(接上页)

```
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

29.17 932. 漂亮数组 (3)

- 题目

```
对于某些固定的N，如果数组A是整数1，2，...，N组成的排列，使得：
对于每个i < j，都不存在k 满足i < k < j使得A[k] * 2 = A[i] + A[j]。
那么数组 A是漂亮数组。
给定N，返回任意漂亮数组A（保证存在一个）。
示例 1：输入：4 输出：[2,1,4,3]
示例 2：输入：5 输出：[3,1,2,5,4]
提示：1 <= N <= 1000
```

- 解题思路

```
var m map[int][]int

func beautifulArray(N int) []int {
        m = make(map[int][]int)
        return dfs(N)
}

func dfs(n int) []int {
        if v, ok := m[n]; ok {
                return v
        }
        res := make([]int, n)
        if n == 1 {
                res[0] = 1
        } else {
                // A[k] * 2 = A[i] + A[j], i < k < j
                //␣
↪要想等式恒不成立，一个简单的办法就是让i部分的数都是奇数，j部分的数都是偶数。
```

(续下页)

(接上页)

```
                index := 0
                for _, v := range dfs((n + 1) / 2) { // 1-N中有(N+1)/2个奇数
                        res[index] = 2*v - 1
                        index++
                }
                for _, v := range dfs(n / 2) { // 1-N中有N/2个偶数
                        res[index] = 2 * v
                        index++
                }
        }
        m[n] = res
        return res
}

# 2
func beautifulArray(N int) []int {
        if N == 1 {
                return []int{1}
        }
        res := make([]int, 0)
        a := beautifulArray((N + 1) / 2)
        b := beautifulArray(N / 2)
        for i := 0; i < len(a); i++ {
                res = append(res, a[i]*2-1)
        }
        for i := 0; i < len(b); i++ {
                res = append(res, b[i]*2)
        }
        return res
}

# 3
func beautifulArray(N int) []int {
        res := make([]int, N)
        for i := 0; i < N; i++ {
                res[i] = i + 1
        }
        sort.Slice(res, func(i, j int) bool {
                k := 0
                for res[i]&k == res[j]&k {
                        k++
                }
                return res[i]&k < res[j]&k
```

(续下页)

(接上页)

```
        })
        return res
}
```

29.18 934. 最短的桥 (1)

- 题目

```
在给定的二维二进制数组A中，存在两座岛。（岛是由四面相连的 1 形成的一个最大组。）
现在，我们可以将0变为1，以使两座岛连接起来，变成一座岛。
返回必须翻转的0 的最小数目。（可以保证答案至少是 1 。）
示例 1：输入：A = [[0,1],[1,0]] 输出：1
示例 2：输入：A = [[0,1,0],[0,0,0],[0,0,1]] 输出：2
示例 3：输入：A = [[1,1,1,1,1],[1,0,0,0,1],[1,0,1,0,1],[1,0,0,0,1],[1,1,1,1,1]]␣
↪输出：1
提示：2 <= A.length == A[0].length <= 100
A[i][j] == 0 或 A[i][j] == 1
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var queue [][2]int
var n int

func shortestBridge(grid [][]int) int {
        n = len(grid)
        queue = make([][2]int, 0)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if grid[i][j] == 1 {
                                dfs(grid, i, j)  // 深度优先搜索找边界
                                return bfs(grid) // 广度优先搜索找距离
                        }
                }
        }
        return 0
}

func dfs(grid [][]int, i, j int) {
        if i < 0 || i >= n || j < 0 || j >= n {
                return
```

(续下页)

(接上页)

```
        }
        if grid[i][j] == 0 {
                queue = append(queue, [2]int{i, j}) // 加入边界
        } else if grid[i][j] == 1 {
                grid[i][j] = 2 // 标记为2
                for k := 0; k < 4; k++ {
                        x, y := i+dx[k], j+dy[k]
                        dfs(grid, x, y)
                }
        }
}

func bfs(grid [][]int) int {
        res := 0
        for len(queue) > 0 {
                res++
                length := len(queue)
                for i := 0; i < length; i++ {
                        a, b := queue[i][0], queue[i][1]
                        for j := 0; j < 4; j++ {
                                x, y := a+dx[j], b+dy[j]
                                if 0 <= x && x < n && 0 <= y && y < n {
                                        if grid[x][y] == 2 {
                                                continue
                                        }
                                        if grid[x][y] == 1 {
                                                return res
                                        }
                                        queue = append(queue, [2]int{x, y})
                                        grid[x][y] = 2
                                }
                        }
                }
                queue = queue[length:]
        }
        return res
}
```

29.19 935. 骑士拨号器 (1)

- 题目

```
国际象棋中的骑士可以按下图所示进行移动：.
这一次，我们将“骑士”␣
↪放在电话拨号盘的任意数字键（如上图所示）上，接下来，骑士将会跳N-1 步。
每一步必须是从一个数字键跳到另一个数字键。
每当它落在一个键上（包括骑士的初始位置），都会拨出键所对应的数字，总共按下N 位数字。
你能用这种方式拨出多少个不同的号码？
因为答案可能很大，所以输出答案模10^9 + 7。
示例 1：输入：1 输出：10
示例 2：输入：2 输出：20
示例 3：输入：3 输出：46
提示：1 <= N <= 5000
```

- 解题思路

```
func knightDialer(n int) int {
        dp := make([][10]int, n)
        for i := 0; i < 10; i++ {
                dp[0][i] = 1
        }
        for i := 1; i < n; i++ {
                dp[i][0] = dp[i-1][4] + dp[i-1][6]
                dp[i][1] = dp[i-1][6] + dp[i-1][8]
                dp[i][2] = dp[i-1][7] + dp[i-1][9]
                dp[i][3] = dp[i-1][4] + dp[i-1][8]
                dp[i][4] = dp[i-1][0] + dp[i-1][3] + dp[i-1][9]
                dp[i][5] = 0
                dp[i][6] = dp[i-1][0] + dp[i-1][1] + dp[i-1][7]
                dp[i][7] = dp[i-1][2] + dp[i-1][6]
                dp[i][8] = dp[i-1][1] + dp[i-1][3]
                dp[i][9] = dp[i-1][2] + dp[i-1][4]
                for j := 0; j < 10; j++ {
                        dp[i][j] = dp[i][j] % 1000000007
                }
        }
        res := 0
        for i := 0; i < 10; i++ {
                res = (res + dp[n-1][i]) % 1000000007
        }
        return res
}
```

29.20 939. 最小面积矩形 (2)

- 题目

```
给定在 xy 平面上的一组点，确定由这些点组成的矩形的最小面积，其中矩形的边平行于 x 轴和y 轴。
如果没有任何矩形，就返回 0。
示例 1：输入：[[1,1],[1,3],[3,1],[3,3],[2,2]] 输出：4
示例 2：输入：[[1,1],[1,3],[3,1],[3,3],[4,1],[4,3]] 输出：2
提示：1 <= points.length <= 500
0 <=points[i][0] <=40000
0 <=points[i][1] <=40000
所有的点都是不同的。
```

- 解题思路

```
func minAreaRect(points [][]int) int {
        res := math.MaxInt32
        m := make(map[string]bool)
        n := len(points)
        for i := 0; i < n; i++ {
                a, b := points[i][0], points[i][1]
                m[fmt.Sprintf("%d,%d", a, b)] = true
        }
        for i := 0; i < n; i++ {
                a, b := points[i][0], points[i][1]
                for j := i + 1; j < n; j++ {
                        c, d := points[j][0], points[j][1]
                        if a == c || b == d {
                                continue
                        }
                        area := abs((a - c) * (b - d))
                        target1 := fmt.Sprintf("%d,%d", a, d) // 枚举另外2个点
                        target2 := fmt.Sprintf("%d,%d", c, b)
                        if area < res && m[target1] == true && m[target2] == true {
                                res = area
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}
```

(续下页)

(接上页)

```
func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func minAreaRect(points [][]int) int {
        res := math.MaxInt32
        m := make(map[int][]int)
        n := len(points)
        for i := 0; i < n; i++ {
                a, b := points[i][0], points[i][1]
                m[a] = append(m[a], b) // 列分组
        }
        rows := make([]int, 0)
        for k := range m {
                rows = append(rows, k)
        }
        sort.Ints(rows)
        prevM := make(map[string]int)
        for _, v := range rows { // 按列枚举
                arr := m[v]
                sort.Ints(arr)
                for i := 0; i < len(arr); i++ {
                        for j := i + 1; j < len(arr); j++ {
                                y1, y2 := arr[i], arr[j]
                                target := fmt.Sprintf("%d,%d", y1, y2)
                                if index, ok := prevM[target]; ok {
                                        area := (y2 - y1) * (v - index)
                                        res = min(res, area)
                                }
                                prevM[target] = v
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

29.21 945. 使数组唯一的最小增量 (3)

- 题目

```
给定整数数组 A，每次 move 操作将会选择任意 A[i]，并将其递增 1。
返回使 A 中的每个值都是唯一的最少操作次数。
示例 1:输入：[1,2,2] 输出：1
解释：经过一次 move 操作，数组将变为 [1, 2, 3]。
示例 2:输入：[3,2,1,2,1,7] 输出：6
解释：经过 6 次 move 操作，数组将变为 [3, 4, 1, 2, 5, 7]。
可以看出 5 次或 5 次以下的 move 操作是不能让数组的每个值唯一的。
提示：
    0 <= A.length <= 40000
    0 <= A[i] < 40000
```

- 解题思路

```
func minIncrementForUnique(A []int) int {
        res := 0
        sort.Ints(A)
        for i := 0; i < len(A)-1; i++ {
                if A[i] >= A[i+1] {
                        value := A[i] - A[i+1] + 1
                        res = res + value
                        A[i+1] = A[i+1] + value
                }
        }
        return res
}

# 2
func minIncrementForUnique(A []int) int {
        res := 0
        arr := make([]int, 80001)
```

(续下页)

(接上页)

```
        for i := 0; i < len(A); i++ {
                arr[A[i]]++
        }
        sum := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] >= 2 {
                        sum = sum + arr[i] - 1
                        res = res - i*(arr[i]-1)
                } else if arr[i] == 0 && sum > 0 {
                        sum--
                        res = res + i
                }
        }
        return res
}

# 3
var arr []int

func minIncrementForUnique(A []int) int {
        res := 0
        arr = make([]int, 21)
        for i := 0; i < len(arr); i++ {
                arr[i] = -1
        }
        for i := 0; i < len(A); i++ {
                b := findNext(A[i])
                res = res + b - A[i]
        }
        return res
}

func findNext(a int) int {
        b := arr[a]
        if b == -1 {
                arr[a] = a
                return a
        }
        b = findNext(b + 1)
        arr[a] = b
        return b
}
```

29.22 946. 验证栈序列 (2)

- 题目

```
给定 pushed 和 popped 两个序列，每个序列中的 值都不重复，
只有当它们可能是在最初空栈上进行的推入 push 和弹出 pop 操作序列的结果时，返回␣
↪true；否则，返回 false 。
示例 1：输入：pushed = [1,2,3,4,5], popped = [4,5,3,2,1] 输出：true
解释：我们可以按以下顺序执行：
push(1), push(2), push(3), push(4), pop() -> 4,
push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
示例 2：输入：pushed = [1,2,3,4,5], popped = [4,3,5,1,2] 输出：false
解释：1 不能在 2 之前弹出。
提示：
    0 <= pushed.length == popped.length <= 1000
    0 <= pushed[i], popped[i] < 1000
    pushed 是 popped 的排列。
```

- 解题思路

```
func validateStackSequences(pushed []int, popped []int) bool {
        stack := make([]int, 0)
        j := 0
        for i := 0; i < len(pushed); i++ {
                stack = append(stack, pushed[i])
                for len(stack) > 0 && stack[len(stack)-1] == popped[j] {
                        stack = stack[:len(stack)-1]
                        j++
                }
        }
        if len(stack) == 0 {
                return true
        }
        return false
}

# 2
func validateStackSequences(pushed []int, popped []int) bool {
        stack := make([]int, 0)
        res := false
        i := 0
        j := 0
        for j < len(popped) {
                for len(stack) == 0 || stack[len(stack)-1] != popped[j] {
```

(续下页)

(接上页)

```
                        if i == len(pushed) {
                                break
                        }
                        stack = append(stack, pushed[i])
                        i++
                }
                if stack[len(stack)-1] != popped[j] {
                        break
                }
                stack = stack[:len(stack)-1]
                j++
        }
        if len(stack) == 0 && j == len(popped) {
                res = true
        }
        return res
}
```

29.23 947. 移除最多的同行或同列石头 (2)

- 题目

```
我们将石头放置在二维平面中的一些整数坐标点上。每个坐标点上最多只能有一块石头。
每次 move 操作都会移除一块所在行或者列上有其他石头存在的石头。
请你设计一个算法，计算最多能执行多少次 move 操作？
示例 1：输入：stones = [[0,0],[0,1],[1,0],[1,2],[2,1],[2,2]] 输出：5
示例 2：输入：stones = [[0,0],[0,2],[1,1],[2,0],[2,2]] 输出：3
示例 3：输入：stones = [[0,0]] 输出：0
提示：1 <= stones.length <= 1000
0 <= stones[i][j] < 10000
```

- 解题思路

```
func removeStones(stones [][]int) int {
        fa := make([]int, 20000)
        for i := 0; i < 20000; i++ {
                fa[i] = i
        }
        for i := 0; i < len(stones); i++ {
                a, b := stones[i][0], stones[i][1]
                // 连接同一行、列
                union(fa, a, b+10000)
```

(续下页)

(接上页)

```
        }
        m := make(map[int]bool)
        for i := 0; i < len(stones); i++ {
                a := stones[i][0]
                m[find(fa, a)] = true
        }
        return len(stones) - len(m)
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}

func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
                a = fa[a]
        }
        return a
}

# 2
var arr [][]int
var m []bool

func removeStones(stones [][]int) int {
        n := len(stones)
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, 0)
        }
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if stones[i][0] == stones[j][0] || // 同行
                                stones[i][1] == stones[j][1] { // 同列
                                arr[i] = append(arr[i], j)
                                arr[j] = append(arr[j], i)
                        }
                }
        }
        m = make([]bool, n)
        count := 0
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                count = count + dfs(i)
        }
        return len(stones) - count
}

func dfs(index int) int {
        if m[index] == true {
                return 0
        }
        m[index] = true
        for i := 0; i < len(arr[index]); i++ {
                dfs(arr[index][i])
        }
        return 1
}
```

29.24 948. 令牌放置 (1)

- 题目

```
你的初始 能量 为P，初始 分数 为0，只有一包令牌 tokens 。
其中 tokens[i] 是第 i 个令牌的值（下标从 0 开始）。
令牌可能的两种使用方法如下：
如果你至少有token[i]点 能量 ，可以将令牌 i 置为正面朝上，失去token[i]点 能量␣
↪，并得到1分 。
如果我们至少有1分 ，可以将令牌 i 置为反面朝上，获得token[i] 点 能量 ，并失去1分 。
每个令牌 最多 只能使用一次，使用 顺序不限 ，不需 使用所有令牌。
在使用任意数量的令牌后，返回我们可以得到的最大 分数 。
示例 1：输入：tokens = [100]，P = 50 输出：0
解释：无法使用唯一的令牌，因为能量和分数都太少了。
示例 2：输入：tokens = [100,200]，P = 150 输出：1
解释：令牌 0 正面朝上，能量变为 50，分数变为 1 。不必使用令牌 1␣
↪，因为你无法使用它来提高分数。
示例 3：输入：tokens = [100,200,300,400]，P = 200 输出：2
解释：按下面顺序使用令牌可以得到 2 分：
1. 令牌 0 正面朝上，能量变为 100 ，分数变为 1
2. 令牌 3 正面朝下，能量变为 500 ，分数变为 0
3. 令牌 1 正面朝上，能量变为 300 ，分数变为 1
4. 令牌 2 正面朝上，能量变为 0 ，分数变为 2
提示：0 <= tokens.length <= 1000
0 <= tokens[i],P < 104
```

- 解题思路

```
func bagOfTokensScore(tokens []int, P int) int {
        sort.Ints(tokens)
        res := 0
        maxValue := 0
        left, right := 0, len(tokens)-1
        for left <= right {
                if tokens[left] <= P { // 分够：消耗分数，增加能量
                        P = P - tokens[left]
                        left++
                        maxValue++
                } else if maxValue > 0 { // 分不够：消耗能量，增加分数
                        P = P + tokens[right]
                        right--
                        maxValue--
                } else {
                        break
                }
                if maxValue > res {
                        res = maxValue
                }
        }
        return res
}
```

29.25 950. 按递增顺序显示卡牌 (3)

- 题目

```
牌组中的每张卡牌都对应有一个唯一的整数。你可以按你想要的顺序对这套卡片进行排序。
最初，这些卡牌在牌组里是正面朝下的（即，未显示状态）。
现在，重复执行以下步骤，直到显示所有卡牌为止：
从牌组顶部抽一张牌，显示它，然后将其从牌组中移出。
如果牌组中仍有牌，则将下一张处于牌组顶部的牌放在牌组的底部。
如果仍有未显示的牌，那么返回步骤 1。否则，停止行动。
返回能以递增顺序显示卡牌的牌组顺序。
答案中的第一张牌被认为处于牌堆顶部。
示例：输入：[17,13,11,2,3,5,7] 输出：[2,13,3,11,5,17,7]
解释：我们得到的牌组顺序为 [17,13,11,2,3,5,7]（这个顺序不重要），然后将其重新排序。
重新排序后，牌组以 [2,13,3,11,5,17,7] 开始，其中 2 位于牌组的顶部。
我们显示 2，然后将 13 移到底部。牌组现在是 [3,11,5,17,7,13]。
我们显示 3，并将 11 移到底部。牌组现在是 [5,17,7,13,11]。
我们显示 5，然后将 17 移到底部。牌组现在是 [7,13,11,17]。
```

(续下页)

(接上页)

```
我们显示 7，并将 13 移到底部。牌组现在是 [11,17,13]。
我们显示 11，然后将 17 移到底部。牌组现在是 [13,17]。
我们展示 13，然后将 17 移到底部。牌组现在是 [17]。
我们显示 17。
由于所有卡片都是按递增顺序排列显示的，所以答案是正确的。
提示：1 <= A.length <= 1000
1 <= A[i] <= 10^6
对于所有的i != j，A[i] != A[j]
```

- 解题思路

```
func deckRevealedIncreasing(deck []int) []int {
        n := len(deck)
        sort.Ints(deck)
        res := make([]int, n)
        temp := make([]int, n)
        for i := 0; i < n; i++ {
                temp[i] = i
        }
        for i := 0; i < n; i++ {
                first := temp[0]
                temp = temp[1:]
                res[first] = deck[i]
                if len(temp) > 0 {
                        first := temp[0]
                        temp = temp[1:]
                        temp = append(temp, first)
                }
        }
        return res
}

# 2
func deckRevealedIncreasing(deck []int) []int {
        n := len(deck)
        sort.Ints(deck)
        if n <= 2 {
                return deck
        }
        res := make([]int, 0)
        res = append(res, deck[n-1])
        for i := n - 2; i >= 0; i-- {
                // 插入一个数a，要将序列尾部的数b拿出来，组成[a,
↪b]放入序列头部，形成新序列。
```

(续下页)

(接上页)

```
                last := res[len(res)-1]
                res = res[:len(res)-1]
                res = append([]int{deck[i], last}, res...)
        }
        return res
}

# 3
func deckRevealedIncreasing(deck []int) []int {
        n := len(deck)
        sort.Ints(deck)
        if n <= 2 {
                return deck
        }
        res := make([]int, 0)
        res = append(res, deck[n-1])
        for i := n - 2; i >= 0; i-- {
                res = append(res, deck[i])
                first := res[0]
                res = res[1:]
                res = append(res, first)
        }
        last := res[len(res)-1]
        res = res[:len(res)-1]
        res = append([]int{last}, res...)
        for i := 0; i < n/2; i++ {
                res[i], res[n-1-i] = res[n-1-i], res[i]
        }
        return res
}
```

29.26 951. 翻转等价二叉树 (1)

- 题目

```
我们可以为二叉树 T␣
↪定义一个翻转操作，如下所示：选择任意节点，然后交换它的左子树和右子树。
只要经过一定次数的翻转操作后，能使 X 等于 Y，我们就称二叉树 X 翻转等价于二叉树 Y。
编写一个判断两个二叉树是否是翻转等价的函数。这些树由根节点 root1 和 root2 给出。
示例：输入：root1 = [1,2,3,4,5,6,null,null,null,7,8],
root2 =[1,3,2,null,6,4,5,null,null,null,null,8,7]
输出：true
```

(续下页)

(接上页)

```
解释：我们翻转值为 1，3 以及 5 的三个节点。
提示：每棵树最多有 100 个节点。
    每棵树中的每个值都是唯一的、在 [0, 99] 范围内的整数。
```

- 解题思路

```
func flipEquiv(root1 *TreeNode, root2 *TreeNode) bool {
        if root1 == nil && root2 == nil {
                return true
        }
        if (root1 == nil && root2 != nil) || (root1 != nil && root2 == nil) {
                return false
        }
        if root1.Val == root2.Val {
                return (flipEquiv(root1.Left, root2.Left) && flipEquiv(root1.Right,␣
↪root2.Right)) ||
                        (flipEquiv(root1.Left, root2.Right) && flipEquiv(root1.Right,␣
↪root2.Left))
        }
        return false
}
```

29.27 954. 二倍数对数组 (1)

- 题目

```
给定一个长度为偶数的整数数组 A，只有对 A 进行重组后可以满足 “对于每个 0 <= i <␣
↪len(A) / 2，
都有 A[2 * i + 1] = 2 * A[2 * i]” 时，返回 true；否则，返回 false。
示例 1：输入：[3,1,3,6] 输出：false
示例 2：输入：[2,1,2,6] 输出：false
示例 3：输入：[4,-2,2,-4] 输出：true
解释：我们可以用 [-2,-4] 和 [2,4] 这两组组成 [-2,-4,2,4] 或是 [2,4,-2,-4]
示例 4：输入：[1,2,4,16,8,4] 输出：false
提示：0 <= A.length <= 30000
    A.length 为偶数
    -100000 <= A[i] <= 100000
```

- 解题思路

```
func canReorderDoubled(A []int) bool {
        m := make(map[int]int)
```

(续下页)

(接上页)

```
        for i := 0; i < len(A); i++ {
                m[A[i]]++
        }
        sort.Slice(A, func(i, j int) bool {
                return abs(A[i]) < abs(A[j])
        })
        for i := 0; i < len(A); i++ {
                if m[A[i]] == 0 {
                        continue
                }
                if m[2*A[i]] == 0 {
                        return false
                }
                m[A[i]]--
                m[2*A[i]]--
        }
        return true
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

29.28 955. 删列造序 II(1)

- 题目

```
给定由 n 个字符串组成的数组 strs，其中每个字符串长度相等。
选取一个删除索引序列，对于 strs 中的每个字符串，删除对应每个索引处的字符。
比如，有 strs = ["abcdef", "uvwxyz"]，删除索引序列{0, 2, 3}，删除后 strs 为["bef",
↪"vyz"]。
假设，我们选择了一组删除索引 answer，那么在执行删除操作之后，最终得到的数组的元素是按
字典序（strs[0] <= strs[1] <= strs[2] ... <= strs[n - 1]）排列的，
然后请你返回 answer.length的最小可能值。
示例 1：输入：strs = ["ca","bb","ac"] 输出：1
解释： 删除第一列后，strs = ["a", "b", "c"]。
现在 strs 中元素是按字典排列的 （即，strs[0] <= strs[1] <= strs[2]）。
我们至少需要进行 1 次删除，因为最初 strs 不是按字典序排列的，所以答案是 1。
示例 2：输入：strs = ["xc","yb","za"] 输出：0
```

(续下页)

(接上页)

```
解释：strs 的列已经是按字典序排列了，所以我们不需要删除任何东西。
注意 strs 的行不需要按字典序排列。
也就是说，strs[0][0] <= strs[0][1] <= ... 不一定成立。
示例 3：输入：strs = ["zyx","wvu","tsr"] 输出：3
解释：我们必须删掉每一列。
提示：n == strs.length
1 <= n <= 100
1 <= strs[i].length <= 100
strs[i] 由小写英文字母组成
```

- 解题思路

```
func minDeletionSize(strs []string) int {
        res := 0
        cur := make([]string, len(strs))
        for i := 0; i < len(strs[0]); i++ {
                temp := make([]string, len(strs))
                copy(temp, cur)
                for j := 0; j < len(temp); j++ {
                        temp[j] = temp[j] + string(strs[j][i])
                }
                if judge(temp) == true { // 加上当前列后，是升序的就用，不是就删除
                        cur = temp
                } else {
                        res++
                }
        }
        return res
}

func judge(strs []string) bool {
        for i := 0; i < len(strs)-1; i++ {
                if strs[i] > strs[i+1] {
                        return false
                }
        }
        return true
}
```

29.29 957.N 天后的牢房

29.29.1 题目

```
8 间牢房排成一排，每间牢房不是有人住就是空着。
每天，无论牢房是被占用或空置，都会根据以下规则进行更改：
如果一间牢房的两个相邻的房间都被占用或都是空的，那么该牢房就会被占用。
否则，它就会被空置。
（请注意，由于监狱中的牢房排成一行，所以行中的第一个和最后一个房间无法有两个相邻的房间。）
我们用以下方式描述监狱的当前状态：如果第 i 间牢房被占用，则 cell[i]==1，否则
↪cell[i]==0。
根据监狱的初始状态，在 N 天后返回监狱的状况（和上述 N 种变化）。
示例 1：输入：cells = [0,1,0,1,1,0,0,1], N = 7 输出：[0,0,1,1,0,0,0,0]
解释：下表概述了监狱每天的状况：
Day 0: [0, 1, 0, 1, 1, 0, 0, 1]
Day 1: [0, 1, 1, 0, 0, 0, 0, 0]
Day 2: [0, 0, 0, 0, 1, 1, 1, 0]
Day 3: [0, 1, 1, 0, 0, 1, 0, 0]
Day 4: [0, 0, 0, 0, 0, 1, 0, 0]
Day 5: [0, 1, 1, 1, 0, 1, 0, 0]
Day 6: [0, 0, 1, 0, 1, 1, 0, 0]
Day 7: [0, 0, 1, 1, 0, 0, 0, 0]
示例 2：输入：cells = [1,0,0,1,0,0,1,0], N = 1000000000 输出：[0,0,1,1,1,1,1,0]
提示：cells.length == 8
cells[i]的值为 0 或 1
1 <= N <= 10^9
```

29.29.2 解题思路

29.30 958. 二叉树的完全性检验 (2)

- 题目

```
给定一个二叉树，确定它是否是一个完全二叉树。
百度百科中对完全二叉树的定义如下：
若设二叉树的深度为 h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数，
第 h 层所有的结点都连续集中在最左边，这就是完全二叉树。（注：第 h 层可能包含 1～ 2h
↪个节点。）
```

(续下页)

(接上页)

```
示例 1：输入：[1,2,3,4,5,6] 输出：true
解释：最后一层前的每一层都是满的（即，结点值为 {1} 和 {2,3} 的两层），
且最后一层中的所有结点（{4,5,6}）都尽可能地向左。
示例 2：输入：[1,2,3,4,5,null,7] 输出：false
解释：值为 7 的结点没有尽可能靠向左侧。
提示：树中将会有 1 到 100 个结点。
```

- 解题思路

```
var m map[int]bool

func isCompleteTree(root *TreeNode) bool {
        m = make(map[int]bool)
        dfs(root, 1)
        for i := 1; i <= len(m); i++ {
                if m[i] == false {
                        return false
                }
        }
        return true
}

func dfs(root *TreeNode, id int) {
        if root == nil {
                return
        }
        m[id] = true
        dfs(root.Left, id*2)
        dfs(root.Right, id*2+1)
}

# 2
func isCompleteTree(root *TreeNode) bool {
        if root == nil {
                return true
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        prev := root
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if prev == nil && node != nil {
                        return false
```

(续下页)

(接上页)

```
                }
                if node != nil {
                        queue = append(queue, node.Left)
                        queue = append(queue, node.Right)
                }
                prev = node
        }
        return true
}
```

29.31 959. 由斜杠划分区域 (2)

- 题目

```
在由 1 x 1 方格组成的 N x N 网格grid 中，每个 1 x 1方块由 /、\␣
↪或空格构成。这些字符会将方块划分为一些共边的区域。
（请注意，反斜杠字符是转义的，因此 \ 用 "\\"表示。）。
返回区域的数目。
示例 1：输入：
[
 " /",
 "/ "
]
输出：2
解释：2x2 网格如下：
示例 2：输入：
[
 " /",
 "  "
]
输出：1
解释：2x2 网格如下：
示例 3：输入：
[
 "\\/",
 "/\\"
]
输出：4
解释：（回想一下，因为 \ 字符是转义的，所以 "\\/" 表示 \/，而 "/\\" 表示 /\。）
2x2 网格如下：
示例 4：输入：
[
```

(续下页)

(接上页)

```
 "/\\",
 "\\/"
]
输出：5
解释：（回想一下，因为 \ 字符是转义的，所以 "/\\" 表示 /\，而 "\\/" 表示 \/。）
2x2 网格如下：
示例 5：输入：
[
 "//",
 "/ "
]
输出：3
解释：2x2 网格如下：
提示：1 <= grid.length == grid[0].length <= 30
grid[i][j] 是'/'、'\'、或' '。
```

- 解题思路

```
func regionsBySlashes(grid []string) int {
        n := len(grid)
        fa = Init(n * n * 4)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        index := 4 * (i*n + j) //␣
↪扩大4倍，每个格子拆分为4个：0、1、2、3（顺时针）
                        if grid[i][j] == '/' {
                                union(index, index+3)   // 合并0、3
                                union(index+1, index+2) // 合并1、2
                        } else if grid[i][j] == '\\' {
                                union(index, index+1)   // 合并0、1
                                union(index+2, index+3) // 合并2、3
                        } else {
                                union(index, index+1) // 合并0、1、2、3
                                union(index+1, index+2)
                                union(index+2, index+3)
                        }
                        if j < n-1 {
                                union(index+1, index+7) // 向右合并
                        }
                        if i < n-1 {
                                union(index+2, index+4*n) // 向下合并
                        }
                }
        }
```

(续下页)

(接上页)

```
        return getCount()
}

var fa []int
var count int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        count = n
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
        }
}

func query(i, j int) bool {
        return find(i) == find(j)
}

func getCount() int {
        return count
}
```

(续下页)

(接上页)

```
# 2
func regionsBySlashes(grid []string) int {
        n := len(grid)
        arr := make([][]int, 3*n)
        for i := 0; i < 3*n; i++ {
                arr[i] = make([]int, 3*n)
        }
        for i := 0; i < n; i++ { // 放大处理
                for j := 0; j < n; j++ {
                        if grid[i][j] == '/' { // 9宫格
                                arr[i*3+2][j*3] = 1
                                arr[i*3+1][j*3+1] = 1
                                arr[i*3][j*3+2] = 1
                        } else if grid[i][j] == '\\' {
                                arr[i*3+2][j*3+2] = 1
                                arr[i*3+1][j*3+1] = 1
                                arr[i*3][j*3] = 1
                        }
                }
        }
        res := 0
        for i := 0; i < 3*n; i++ {
                for j := 0; j < 3*n; j++ {
                        if arr[i][j] == 0 {
                                dfs(arr, i, j)
                                res++
                        }
                }
        }
        return res
}

func dfs(arr [][]int, i, j int) {
        if 0 <= i && i < len(arr) && 0 <= j && j < len(arr[0]) && arr[i][j] == 0 {
                arr[i][j] = 1
                dfs(arr, i+1, j)
                dfs(arr, i-1, j)
                dfs(arr, i, j-1)
                dfs(arr, i, j+1)
        }
}
```

29.32 962. 最大宽度坡 (4)

- 题目

```
给定一个整数数组A，坡是元组(i, j)，其中i < j且A[i] <= A[j]。这样的坡的宽度为j - i。
找出A中的坡的最大宽度，如果不存在，返回 0 。
示例 1：输入：[6,0,8,2,1,5] 输出：4
解释： 最大宽度的坡为 (i, j) = (1, 5)：A[1] = 0 且 A[5] = 5.
示例 2：输入：[9,8,1,0,1,9,4,0,4,1] 输出：7
解释：最大宽度的坡为 (i, j) = (2, 9)：A[2] = 1 且 A[9] = 1.
提示：2 <= A.length <= 50000
0 <= A[i] <= 50000
```

- 解题思路

```
type Node struct {
   Value int
   Index int
}

func maxWidthRamp(A []int) int {
   res := 0
   n := len(A)
   arr := make([]Node, n)
   for i := 0; i < n; i++ {
           arr[i] = Node{
                   Value: A[i],
                   Index: i,
           }
   }
   sort.Slice(arr, func(i, j int) bool {
           if arr[i].Value == arr[j].Value {
                   return arr[i].Index < arr[j].Index
           }
           return arr[i].Value < arr[j].Value
   })
   minIndex := n // 保存当前遇到的最小下标
   for i := 0; i < n; i++ {
           res = max(res, arr[i].Index-minIndex)
           minIndex = min(minIndex, arr[i].Index)
   }
   return res
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
   if a > b {
           return a
   }
   return b
}

func min(a, b int) int {
   if a > b {
           return b
   }
   return a
}

# 2
type Node struct {
   Value int
   Index int
}

func maxWidthRamp(A []int) int {
   res := 0
   n := len(A)
   arr := make([]Node, 0) // Value递增数组
   arr = append(arr, Node{
           Value: A[n-1],
           Index: n - 1,
   })
   for i := n - 2; i >= 0; i-- {
           left, right := 0, len(arr)
           for left < right {
                   mid := left + (right-left)/2
                   if arr[mid].Value < A[i] { // 不满足条件
                           left = mid + 1
                   } else {
                           right = mid
                   }
           }
           if left < len(arr) {
                   index := arr[left].Index
                   res = max(res, index-i)
           } else {
                   arr = append(arr, Node{
```

(续下页)

(接上页)

```
                            Value: A[i],
                            Index: i,
                    })
            }
    }
    return res
}

func max(a, b int) int {
    if a > b {
            return a
    }
    return b
}

# 3
func maxWidthRamp(A []int) int {
    res := 0
    n := len(A)
    stack := make([]int, 0) // 递减栈
    for i := 0; i < n; i++ {
            if len(stack) == 0 || A[stack[len(stack)-1]] > A[i] {
                    stack = append(stack, i)
            }
    }
    for i := n - 1; i >= 0; i-- {
            for len(stack) > 0 && A[stack[len(stack)-1]] <= A[i] {
                    index := stack[len(stack)-1] // 坡底index能形成的最大宽度
                    stack = stack[:len(stack)-1]
                    res = max(res, i-index)
            }
    }
    return res
}

func max(a, b int) int {
    if a > b {
            return a
    }
    return b
}

# 4
```

(续下页)

(接上页)

```
func maxWidthRamp(A []int) int {
   res := 0
   for i := 0; i < len(A); i++ {
           for j := len(A) - 1; j > i+res; j-- {
                   if A[i] <= A[j] {
                           res = max(res, j-i)
                           break
                   }
           }
   }
   return res
}

func max(a, b int) int {
   if a > b {
           return a
   }
   return b
}
```

29.33 963. 最小面积矩形 II

29.33.1 题目

```
给定在 xy␣
↪平面上的一组点，确定由这些点组成的任何矩形的最小面积，其中矩形的边不一定平行于 x␣
↪轴和 y 轴。
如果没有任何矩形，就返回 0。
示例 1：输入：[[1,2],[2,1],[1,0],[0,1]] 输出：2.00000
解释：最小面积的矩形出现在 [1,2],[2,1],[1,0],[0,1] 处，面积为 2。
示例 2：输入：[[0,1],[2,1],[1,1],[1,0],[2,0]] 输出：1.00000
解释：最小面积的矩形出现在 [1,0],[1,1],[2,1],[2,0] 处，面积为 1。
示例 3：输入：[[0,3],[1,2],[3,1],[1,3],[2,1]] 输出：0
解释：没法从这些点中组成任何矩形。
示例 4：输入：[[3,1],[1,1],[0,1],[2,1],[3,3],[3,2],[0,2],[2,3]] 输出：2.00000
解释：最小面积的矩形出现在 [2,1],[2,3],[3,3],[3,1] 处，面积为 2。
提示：1 <= points.length <= 50
0 <=points[i][0] <=40000
0 <=points[i][1] <=40000
所有的点都是不同的。
与真实值误差不超过 10^-5的答案将视为正确结果。
```

29.33.2 解题思路

29.34 966. 元音拼写检查器 (1)

- 题目

```
在给定单词列表wordlist的情况下，我们希望实现一个拼写检查器，将查询单词转换为正确的单词。
对于给定的查询单词query，拼写检查器将会处理两类拼写错误：
大小写：如果查询匹配单词列表中的某个单词（不区分大小写），则返回的正确单词与单词列表中的大小写相同。
例如：wordlist = ["yellow"], query = "YellOw": correct = "yellow"
例如：wordlist = ["Yellow"], query = "yellow": correct = "Yellow"
例如：wordlist = ["yellow"], query = "yellow": correct = "yellow"
元音错误：如果在将查询单词中的元音（‘a’、‘e’、‘i’、‘o’、‘u’）分别替换为任何元音后，
能与单词列表中的单词匹配（不区分大小写），则返回的正确单词与单词列表中的匹配项大小写相同。
例如：wordlist = ["YellOw"], query = "yollow": correct = "YellOw"
例如：wordlist = ["YellOw"], query = "yeellow": correct = ""（无匹配项）
例如：wordlist = ["YellOw"], query = "yllw": correct = ""（无匹配项）
此外，拼写检查器还按照以下优先级规则操作：
当查询完全匹配单词列表中的某个单词（区分大小写）时，应返回相同的单词。
当查询匹配到大小写问题的单词时，您应该返回单词列表中的第一个这样的匹配项。
当查询匹配到元音错误的单词时，您应该返回单词列表中的第一个这样的匹配项。
如果该查询在单词列表中没有匹配项，则应返回空字符串。
给出一些查询 queries，返回一个单词列表 answer，
其中 answer[i] 是由查询 query = queries[i] 得到的正确单词。
示例：输入：wordlist = ["KiTe","kite","hare","Hare"],
queries = ["kite","Kite","KiTe","Hare","HARE","Hear","hear","keti","keet","keto"]
输出：["kite","KiTe","KiTe","Hare","hare","","","KiTe","","KiTe"]
提示：1 <= wordlist.length <= 5000
1 <= queries.length <= 5000
1 <= wordlist[i].length <= 7
1 <= queries[i].length <= 7
wordlist 和queries中的所有字符串仅由英文字母组成。
```

- 解题思路

```
func spellchecker(wordlist []string, queries []string) []string {
        m := make(map[string]bool)
        mLower := make(map[string]string)
        mVowel := make(map[string]string)
        for i := 0; i < len(wordlist); i++ {
                m[wordlist[i]] = true
```

(续下页)

(接上页)

```
                lowerStr := strings.ToLower(wordlist[i])
                if _, ok := mLower[lowerStr]; ok == false {
                        mLower[lowerStr] = wordlist[i]
                }
                vowelStr := changeVowel(lowerStr)
                if _, ok := mVowel[vowelStr]; ok == false {
                        mVowel[vowelStr] = wordlist[i]
                }
        }
        for i := 0; i < len(queries); i++ {
                if m[queries[i]] == true {
                        continue
                }
                lowerStr := strings.ToLower(queries[i])
                if v, ok := mLower[lowerStr]; ok {
                        queries[i] = v
                        continue
                }
                vowelStr := changeVowel(lowerStr)
                if v, ok := mVowel[vowelStr]; ok {
                        queries[i] = v
                        continue
                }
                queries[i] = ""
        }
        return queries
}

func changeVowel(str string) string {
        str = strings.ReplaceAll(str, "a", "?")
        str = strings.ReplaceAll(str, "e", "?")
        str = strings.ReplaceAll(str, "i", "?")
        str = strings.ReplaceAll(str, "o", "?")
        str = strings.ReplaceAll(str, "u", "?")
        return str
}
```

29.35 967. 连续差相同的数字 (2)

- 题目

```
返回所有长度为 n 且满足其每两个连续位上的数字之间的差的绝对值为 k 的 非负整数 。
请注意，除了 数字 0 本身之外，答案中的每个数字都 不能 有前导零。
例如，01 有一个前导零，所以是无效的；但 0是有效的。
你可以按 任何顺序 返回答案。
示例 1：输入：n = 3, k = 7 输出：[181,292,707,818,929]
解释：注意，070 不是一个有效的数字，因为它有前导零。
示例 2：输入：n = 2, k = 1 输出：[10,12,21,23,32,34,43,45,54,56,65,67,76,78,87,89,98]
示例 3：输入：n = 2, k = 0 输出：[11,22,33,44,55,66,77,88,99]
示例 4：输入：n = 2, k = 1
输出：[10,12,21,23,32,34,43,45,54,56,65,67,76,78,87,89,98]
示例 5：输入：n = 2, k = 2
输出：[13,20,24,31,35,42,46,53,57,64,68,75,79,86,97]
提示：2 <= n <= 9
0 <= k <= 9
```

- 解题思路

```
var res []int

func numsSameConsecDiff(n int, k int) []int {
        res = make([]int, 0)
        for i := 1; i <= 9; i++ {
                dfs(i, n, k, i)
        }
        return res
}

func dfs(index, n, k, value int) {
        if n == 1 {
                res = append(res, value)
                return
        }
        a := index + k
        if 0 <= a && a <= 9 && k != 0 {
                value = value*10 + a
                dfs(a, n-1, k, value)
                value = (value - a) / 10
        }
        b := index - k
        if 0 <= b && b <= 9 {
```

(续下页)

(接上页)

```
                value = value*10 + b
                dfs(b, n-1, k, value)
                value = (value - b) / 10
        }
}

# 2
func numsSameConsecDiff(n int, k int) []int {
        dp := make([][]int, n)
        dp[0] = []int{1, 2, 3, 4, 5, 6, 7, 8, 9}
        for i := 1; i < n; i++ {
                temp := make([]int, 0)
                for j := 0; j < len(dp[i-1]); j++ {
                        num := dp[i-1][j]
                        a := num%10 - k
                        if a >= 0 {
                                temp = append(temp, num*10+a)
                        }
                        b := num%10 + k
                        if b <= 9 && k > 0 {
                                temp = append(temp, num*10+b)
                        }
                }
                dp[i] = temp
        }
        return dp[n-1]
}
```

29.36 969. 煎饼排序 (1)

- 题目

```
给定数组 A，我们可以对其进行煎饼翻转：我们选择一些正整数 k <= A.length，
然后反转 A 的前 k 个元素的顺序。
我们要执行零次或多次煎饼翻转（按顺序一次接一次地进行）以完成对数组 A 的排序。
返回能使 A 排序的煎饼翻转操作所对应的 k 值序列。
任何将数组排序且翻转次数在 10 * A.length 范围内的有效答案都将被判断为正确。
示例 1：输入：[3,2,4,1] 输出：[4,2,4,3]
解释：我们执行 4 次煎饼翻转，k 值分别为 4，2，4，和 3。
初始状态 A = [3, 2, 4, 1]
第一次翻转后 (k=4)：A = [1, 4, 2, 3]
第二次翻转后 (k=2)：A = [4, 1, 2, 3]
```

(续下页)

(接上页)

```
第三次翻转后 (k=4): A = [3, 2, 1, 4]
第四次翻转后 (k=3): A = [1, 2, 3, 4], 此时已完成排序。
示例 2: 输入: [1,2,3] 输出: []
解释: 输入已经排序, 因此不需要翻转任何内容。
请注意, 其他可能的答案, 如[3, 3], 也将被接受。
提示: 1 <= A.length <= 100
    A[i] 是 [1, 2, ..., A.length] 的排列
```

- 解题思路

```
func pancakeSort(arr []int) []int {
        res := make([]int, 0)
        n := len(arr) - 1
        for n >= 0 {
                maxValue := arr[0]
                index := 0
                for i := 1; i <= n; i++ {
                        if arr[i] > maxValue {
                                maxValue = arr[i]
                                index = i
                        }
                }
                if index == n {
                        n--
                        continue
                }
                if index != 0 {
                        res = append(res, index+1)
                        reverse(arr, 0, index)
                }
                res = append(res, n+1)
                reverse(arr, 0, n)
                n--
        }
        return res
}

func reverse(arr []int, left, right int) {
        for left < right {
                arr[left], arr[right] = arr[right], arr[left]
                left++
                right--
        }
}
```

29.37 971. 翻转二叉树以匹配先序遍历 (2)

- 题目

```
给你一棵二叉树的根节点 root ，树中有 n 个节点，每个节点都有一个不同于其他节点且处于 1␣
↪到 n 之间的值。
另给你一个由 n 个值组成的行程序列 voyage ，表示 预期 的二叉树 先序遍历 结果。
通过交换节点的左右子树，可以 翻转 该二叉树中的任意节点。例，翻转节点 1 的效果如下：
请翻转 最少 的树中节点，使二叉树的 先序遍历 与预期的遍历行程voyage相匹配 。
如果可以，则返回 翻转的␣
↪所有节点的值的列表。你可以按任何顺序返回答案。如果不能，则返回列表 [-1]。
示例 1：输入：root = [1,2], voyage = [2,1] 输出：[-1]
解释：翻转节点无法令先序遍历匹配预期行程。
示例 2：输入：root = [1,2,3], voyage = [1,3,2] 输出：[1]
解释：交换节点 2 和 3 来翻转节点 1 ，先序遍历可以匹配预期行程。
示例 3：输入：root = [1,2,3], voyage = [1,2,3] 输出：[]
解释：先序遍历已经匹配预期行程，所以不需要翻转节点。
提示：树中的节点数目为 n
n == voyage.length
1 <= n <= 100
1 <= Node.val, voyage[i] <= n
树中的所有值 互不相同
voyage 中的所有值 互不相同
```

- 解题思路

```
var res []int
var arr []int

func flipMatchVoyage(root *TreeNode, voyage []int) []int {
        res = make([]int, 0)
        arr = make([]int, len(voyage))
        copy(arr, voyage)
        dfs(root)
        return res
}

func dfs(root *TreeNode) bool {
        if root == nil {
                return true
        }
        if root.Val != arr[0] { // 根不满足直接返回
                res = []int{-1}
                return false
```

(续下页)

(接上页)

```
        }
        if root.Left != nil && root.Right != nil && root.Right.Val == arr[1] { // 交换
                res = append(res, root.Val)
                root.Left, root.Right = root.Right, root.Left
        }
        arr = arr[1:]
        if dfs(root.Left) == false {
                return false
        }
        return dfs(root.Right)
}

# 2
func flipMatchVoyage(root *TreeNode, voyage []int) []int {
        res := make([]int, 0)
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        index := 0
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if node == nil {
                        continue
                }
                if node.Val != voyage[index] {
                        return []int{-1}
                }
                index++
                if node.Left != nil && node.Right != nil && node.Left.Val !=␣
↪voyage[index] {
                        res = append(res, node.Val)
                        stack = append(stack, node.Left) // 翻转
                        stack = append(stack, node.Right)
                } else {
                        stack = append(stack, node.Right)
                        stack = append(stack, node.Left)
                }
        }
        return res
}
```

29.38 973. 最接近原点的 K 个点 (3)

- 题目

```
我们有一个由平面上的点组成的列表 points。需要从中找出 K 个距离原点 (0, 0) 最近的点。
（这里，平面上两点之间的距离是欧几里德距离。）
你可以按任何顺序返回答案。除了点坐标的顺序之外，答案确保是唯一的。
示例 1：输入：points = [[1,3],[-2,2]], K = 1 输出：[[-2,2]]
解释： (1, 3) 和原点之间的距离为 sqrt(10),
(-2, 2) 和原点之间的距离为 sqrt(8),
由于 sqrt(8) < sqrt(10), (-2, 2) 离原点更近。
我们只需要距离原点最近的 K = 1 个点，所以答案就是 [[-2,2]]。
示例 2：输入：points = [[3,3],[5,-1],[-2,4]], K = 2 输出：[[3,3],[-2,4]]
（答案 [[-2,4],[3,3]] 也会被接受。）
提示： 1 <= K <= points.length <= 10000
    -10000 < points[i][0] < 10000
    -10000 < points[i][1] < 10000
```

- 解题思路

```
func kClosest(points [][]int, K int) [][]int {
        sort.Slice(points, func(i, j int) bool {
                return points[i][0]*points[i][0]+points[i][1]*points[i][1] <
                        points[j][0]*points[j][0]+points[j][1]*points[j][1]
        })
        return points[:K]
}

# 2
func kClosest(points [][]int, K int) [][]int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < len(points); i++ {
                heap.Push(&intHeap, points[i])
        }
        res := make([][]int, 0)
        for i := 0; i < K; i++ {
                value := heap.Pop(&intHeap).([]int)
                res = append(res, value)
        }
        return res
}

type IntHeap [][]int
```

(续下页)

(接上页)

```
func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][0]*h[i][0]+h[i][1]*h[i][1] <
                h[j][0]*h[j][0]+h[j][1]*h[j][1]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 3
func kClosest(points [][]int, K int) [][]int {
        quick(points, 0, len(points)-1, K)
        return points[:K]
}

func quick(points [][]int, left, right int, K int) {
        if left >= right {
                return
        }
        for {
                target := partition(points, left, right)
                if target == K-1 {
                        return
                }
                if target < K-1 {
                        left = target + 1
                } else {
```

(续下页)

(接上页)

```
                        right = target - 1
                }
        }
}

func partition(points [][]int, left, right int) int {
        baseValue := points[left]
        for left < right {
                for left < right && dist(baseValue) <= dist(points[right]) {
                        right--
                }
                points[left] = points[right]
                for left < right && dist(points[left]) <= dist(baseValue) {
                        left++
                }
                points[right] = points[left]
        }
        points[right] = baseValue
        return right
}

func dist(points []int) int {
        return points[0]*points[0] + points[1]*points[1]
}
```

29.39 974. 和可被 K 整除的子数组 (2)

- 题目

```
给定一个整数数组 A，返回其中元素之和可被 K 整除的（连续、非空）子数组的数目。
示例：输入：A = [4,5,0,-2,-3,1], K = 5 输出：7
解释： 有 7 个子数组满足其元素之和可被 K = 5 整除：
[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]
提示： 1 <= A.length <= 30000
    -10000 <= A[i] <= 10000
    2 <= K <= 10000
```

- 解题思路

```
func subarraysDivByK(A []int, K int) int {
        m := make(map[int]int)
        m[0] = 1
```

(续下页)

(接上页)

```
        sum := 0
        res := 0
        for i := 0; i < len(A); i++ {
                sum = sum + A[i]
                value := (sum%K + K) % K
                res = res + m[value]
                m[value]++
        }
        return res
}

# 2
func subarraysDivByK(A []int, K int) int {
        m := make(map[int]int)
        m[0] = 1
        sum := 0
        res := 0
        for i := 0; i < len(A); i++ {
                sum = sum + A[i]
                value := (sum%K + K) % K
                m[value]++
        }
        for _, v := range m {
                res = res + v*(v-1)/2
        }
        return res
}
```

29.40 978. 最长湍流子数组 (3)

- 题目

```
当 A的子数组A[i], A[i+1], ..., A[j]满足下列条件时，我们称其为湍流子数组：
若i <= k < j，当 k为奇数时，A[k] > A[k+1]，且当 k 为偶数时，A[k] < A[k+1]；
或 若i <= k < j，当 k 为偶数时，A[k] > A[k+1]，且当 k为奇数时，A[k] < A[k+1]。
也就是说，如果比较符号在子数组中的每个相邻元素对之间翻转，则该子数组是湍流子数组。
返回 A 的最大湍流子数组的长度。
示例 1：输入：[9,4,2,10,7,8,8,1,9] 输出：5
解释：(A[1] > A[2] < A[3] > A[4] < A[5])
示例 2：输入：[4,8,12,16] 输出：2
示例 3：输入：[100] 输出：1
提示：1 <= A.length <= 40000
```

(续下页)

(接上页)

```
0 <= A[i] <= 10^9
```

- 解题思路

```
func maxTurbulenceSize(arr []int) int {
        n := len(arr)
        up := make([]int, n)
        down := make([]int, n)
        up[0] = 1
        down[0] = 1
        for i := 1; i < n; i++ {
                down[i] = 1
                up[i] = 1
                if arr[i] > arr[i-1] {
                        up[i] = down[i-1] + 1
                } else if arr[i] < arr[i-1] {
                        down[i] = up[i-1] + 1
                }
        }
        res := 1
        for i := 0; i < n; i++ {
                res = max(res, up[i])
                res = max(res, down[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxTurbulenceSize(arr []int) int {
        n := len(arr)
        up := 1
        down := 1
        res := 1
        for i := 1; i < n; i++ {
                if arr[i] > arr[i-1] {
                        up, down = down+1, 1
                } else if arr[i] < arr[i-1] {
```

(续下页)

(接上页)

```
                        up, down = 1, up+1
                } else {
                        up, down = 1, 1
                }
                res = max(res, max(up, down))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxTurbulenceSize(arr []int) int {
        n := len(arr)
        res := 1
        left, right := 0, 0
        for right < n-1 {
                if left == right {
                        if arr[left] == arr[left+1] {
                                left++
                        }
                        right++
                } else {
                        if arr[right-1] < arr[right] && arr[right] > arr[right+1] {
                                right++
                        } else if arr[right-1] > arr[right] && arr[right] <␣
↪arr[right+1] {
                                right++
                        } else {
                                left = right
                        }
                }
                res = max(res, right-left+1)
        }
        return res
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}
```

29.41 979. 在二叉树中分配硬币 (1)

- 题目

```
给定一个有 N 个结点的二叉树的根结点 root，树中的每个结点上都对应有 node.val 枚硬币，
并且总共有 N 枚硬币。
在一次移动中，我们可以选择两个相邻的结点，然后将一枚硬币从其中一个结点移动到另一个结点。
(移动可以是从父结点到子结点，或者从子结点移动到父结点。)。
返回使每个结点上只有一枚硬币所需的移动次数。
示例 1：输入：[3,0,0] 输出：2
解释：从树的根结点开始，我们将一枚硬币移到它的左子结点上，一枚硬币移到它的右子结点上。
示例 2：输入：[0,3,0] 输出：3
解释：从根结点的左子结点开始，我们将两枚硬币移到根结点上 [移动两次]。
然后，我们把一枚硬币从根结点移到右子结点上。
示例 3：输入：[1,0,2] 输出：2
示例 4：输入：[1,0,0,null,3] 输出：4
提示：1<= N <= 100
0 <= node.val <= N
```

- 解题思路

```
var res int

func distributeCoins(root *TreeNode) int {
        res = 0
        dfs(root)
        return res
}

func dfs(root *TreeNode) int { // 该节点子树多余/需要金币数量
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        res = res + abs(left) + abs(right)
```

(续下页)

(接上页)

```
        return left + right + root.Val - 1
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

29.42 981. 基于时间的键值存储 (1)

- 题目

```
创建一个基于时间的键值存储类TimeMap，它支持下面两个操作：
1. set(string key, string value, int timestamp)
存储键key、值value，以及给定的时间戳timestamp。
2. get(string key, int timestamp)
返回先前调用set(key, value, timestamp_prev)所存储的值，其中timestamp_prev <=␣
↪timestamp。
如果有多个这样的值，则返回对应最大的timestamp_prev的那个值。
如果没有值，则返回空字符串（""）。
示例 1：输入：inputs = ["TimeMap","set","get","get","set","get","get"],
inputs = [[],["foo","bar",1],["foo",1],["foo",3],["foo","bar2",4],["foo",4],["foo",5]]
输出：[null,null,"bar","bar",null,"bar2","bar2"]
解释：  TimeMap kv;
kv.set("foo", "bar", 1); // 存储键 "foo" 和值 "bar" 以及时间戳 timestamp = 1
kv.get("foo", 1);  // 输出 "bar"
kv.get("foo", 3); // 输出 "bar" 因为在时间戳 3 和时间戳 2 处没有对应 "foo" 的值，
所以唯一的值位于时间戳 1 处（即 "bar"）  kv.set("foo", "bar2", 4);
kv.get("foo", 4); // 输出 "bar2"
kv.get("foo", 5); // 输出 "bar2"
示例 2：输入：inputs = ["TimeMap","set","set","get","get","get","get","get"],
inputs = [[],["love","high",10],["love","low",20],["love",5],["love",10],
["love",15],["love",20],["love",25]]
输出：[null,null,null,"","high","high","low","low"]
提示：所有的键/值字符串都是小写的。
所有的键/值字符串长度都在[1, 100]范围内。
所有TimeMap.set操作中的时间戳timestamps 都是严格递增的。
1 <= timestamp <= 10^7
TimeMap.set 和TimeMap.get函数在每个测试用例中将（组合）调用总计120000 次。
```

- 解题思路

```go
type Node struct {
        timestamp int
        str       string
}

type TimeMap struct {
        m map[string][]Node
}

func Constructor() TimeMap {
        return TimeMap{m: make(map[string][]Node)}
}

func (this *TimeMap) Set(key string, value string, timestamp int) {
        this.m[key] = append(this.m[key], Node{
                timestamp: timestamp,
                str:       value,
        })
}

func (this *TimeMap) Get(key string, timestamp int) string {
        arr := this.m[key]
        n := len(arr)
        if n == 0 || (timestamp < arr[0].timestamp) {
                return ""
        }
        if timestamp >= arr[n-1].timestamp {
                return arr[n-1].str
        }
        left, right := 0, n
        for left < right {
                mid := left + (right-left)/2
                if arr[mid].timestamp == timestamp {
                        return arr[mid].str
                } else if arr[mid].timestamp < timestamp {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return arr[left].str
}
```

29.43 983. 最低票价 (3)

- 题目

```
在一个火车旅行很受欢迎的国度，你提前一年计划了一些火车旅行。
在接下来的一年里，你要旅行的日子将以一个名为 days 的数组给出。每一项是一个从 1 到 365 的整数。
火车票有三种不同的销售方式：
    一张为期一天的通行证售价为 costs[0] 美元；
    一张为期七天的通行证售价为 costs[1] 美元；
    一张为期三十天的通行证售价为 costs[2] 美元。
通行证允许数天无限制的旅行。 例如，如果我们在第 2 天获得一张为期 7 天的通行证，
那么我们可以连着旅行 7 天：第 2 天、第 3 天、第 4 天、第 5 天、第 6 天、第 7 天和第 8 天。
返回你想要完成在给定的列表 days 中列出的每一天的旅行所需要的最低消费。
示例 1：输入：days = [1,4,6,7,8,20], costs = [2,7,15] 输出：11
解释：  例如，这里有一种购买通行证的方法，可以让你完成你的旅行计划：
在第 1 天，你花了 costs[0] = $2 买了一张为期 1 天的通行证，它将在第 1 天生效。
在第 3 天，你花了 costs[1] = $7 买了一张为期 7 天的通行证，它将在第 3, 4, ..., 9 天生效。
在第 20 天，你花了 costs[0] = $2 买了一张为期 1 天的通行证，它将在第 20 天生效。
你总共花了 $11，并完成了你计划的每一天旅行。
示例 2：输入：days = [1,2,3,4,5,6,7,8,9,10,30,31], costs = [2,7,15] 输出：17
解释：  例如，这里有一种购买通行证的方法，可以让你完成你的旅行计划：
在第 1 天，你花了 costs[2] = $15 买了一张为期 30 天的通行证，它将在第 1, 2, ..., 30 天生效。
在第 31 天，你花了 costs[0] = $2 买了一张为期 1 天的通行证，它将在第 31 天生效。
你总共花了 $17，并完成了你计划的每一天旅行。
提示：1 <= days.length <= 365
    1 <= days[i] <= 365
    days 按顺序严格递增
    costs.length == 3
    1 <= costs[i] <= 1000
```

- 解题思路

```
var dp [366]int
var m map[int]bool

func mincostTickets(days []int, costs []int) int {
        dp = [366]int{}
        m = make(map[int]bool)
        for i := 0; i < len(days); i++ {
                m[days[i]] = true
```

(续下页)

(接上页)

```
        }
        return dfs(1, costs)
}

func dfs(day int, costs []int) int {
        if day > 365 {
                return 0
        }
        if dp[day] > 0 {
                return dp[day]
        }
        if m[day] == true {
                dp[day] = min(min(dfs(day+1, costs)+costs[0], dfs(day+7,␣
↪costs)+costs[1]),
                        dfs(day+30, costs)+costs[2])
        } else {
                dp[day] = dfs(day+1, costs)
        }
        return dp[day]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var dp [366]int
var duration = []int{1, 7, 30}

func mincostTickets(days []int, costs []int) int {
        dp = [366]int{}
        return dfs(0, costs, days)
}

func dfs(day int, costs []int, days []int) int {
        if day >= len(days) {
                return 0
        }
        if dp[day] > 0 {
                return dp[day]
```

(续下页)

(接上页)

```
        }
        dp[day] = math.MaxInt32
        j := day
        for i := 0; i < 3; i++ {
                for ; j < len(days) && days[j] < days[day]+duration[i]; j++ {
                }
                dp[day] = min(dp[day], dfs(j, costs, days)+costs[i])
        }
        return dp[day]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func mincostTickets(days []int, costs []int) int {
        n := days[len(days)-1] + 1
        dp := make([]int, n)
        for i := 0; i < len(days); i++ {
                dp[days[i]] = 1 // 出行日
        }
        for i := 1; i < n; i++ {
                if dp[i] > 0 {
                        dp[i] = min(dp[i-1]+costs[0],
                                min(dp[max(i-7, 0)]+costs[1], dp[max(i-30,␣
↪0)]+costs[2]))
                } else {
                        dp[i] = dp[i-1]
                }
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

29.44 984. 不含 AAA 或 BBB 的字符串 (2)

- 题目

```
给定两个整数A和B，返回任意字符串 S，要求满足：
S 的长度为A + B，且正好包含A个 'a'字母与B个 'b'字母；
子串'aaa'没有出现在S中；
子串'bbb' 没有出现在S中。
示例 1：输入：A = 1, B = 2 输出："abb"
解释："abb", "bab" 和 "bba" 都是正确答案。
示例 2：输入：A = 4, B = 1 输出："aabaa"
提示：0 <= A <= 100
0 <= B <= 100
对于给定的 A 和 B，保证存在满足要求的 S。
```

- 解题思路

```
func strWithout3a3b(a int, b int) string {
        res := make([]byte, 0)
        for a > 0 || b > 0 {
                flagA := false
                if len(res) >= 2 && res[len(res)-1] == res[len(res)-2] {
                        if res[len(res)-1] == 'b' {
                                flagA = true
                        }
                } else if a >= b {
                        flagA = true
                }
                if flagA == true {
                        res = append(res, 'a')
                        a--
                } else {
                        res = append(res, 'b')
                        b--
```

(续下页)

(接上页)

```
                }
        }
        return string(res)
}

# 2
func strWithout3a3b(a int, b int) string {
        res := make([]byte, 0)
        for a > 0 && b > 0 {
                if a > b {
                        res = append(res, []byte{'a', 'a', 'b'}...)
                        a = a - 2
                        b--
                } else if a == b {
                        res = append(res, []byte{'a', 'b'}...)
                        a--
                        b--
                } else {
                        res = append(res, []byte{'b', 'b', 'a'}...)
                        a--
                        b = b - 2
                }
        }
        for a > 0 {
                res = append(res, 'a')
                a--
        }
        for b > 0 {
                res = append(res, 'b')
                b--
        }
        return string(res)
}
```

29.45 986. 区间列表的交集 (2)

- 题目

```
给定两个由一些 闭区间 组成的列表，firstList 和 secondList ，
其中 firstList[i] = [starti, endi] 而secondList[j] = [startj, endj] 。
每个区间列表都是成对 不相交 的，并且 已经排序 。
返回这 两个区间列表的交集 。
```

(续下页)

(接上页)

```
形式上，闭区间[a, b]（其中a <= b）表示实数x的集合，而a <= x <= b 。
两个闭区间的 交集 是一组实数，要么为空集，要么为闭区间。例如，[1, 3] 和 [2, 4]␣
↪的交集为 [2, 3] 。
示例 1：输入：firstList = [[0,2],[5,10],[13,23],[24,25]],
secondList = [[1,5],[8,12],[15,24],[25,26]]
输出：[[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
示例 2：输入：firstList = [[1,3],[5,9]], secondList = [] 输出：[]
示例 3：输入：firstList = [], secondList = [[4,8],[10,12]] 输出：[]
示例 4：输入：firstList = [[1,7]], secondList = [[3,10]] 输出：[[3,7]]
提示：0 <= firstList.length, secondList.length <= 1000
firstList.length + secondList.length >= 1
0 <= starti < endi <= 109
endi < starti+1
0 <= startj < endj <= 109
endj < startj+1
```

- 解题思路

```
func intervalIntersection(A [][]int, B [][]int) [][]int {
        res := make([][]int, 0)
        i, j := 0, 0
        for i < len(A) && j < len(B) {
                if A[i][0] <= B[j][1] && B[j][0] <= A[i][1] {
                        left := max(A[i][0], B[j][0])
                        right := min(A[i][1], B[j][1])
                        res = append(res, []int{left, right})
                }
                if A[i][1] < B[j][1] {
                        i++
                } else {
                        j++
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}

# 2
func intervalIntersection(A [][]int, B [][]int) [][]int {
        res := make([][]int, 0)
        i, j := 0, 0
        for i < len(A) && j < len(B) {
                left := max(A[i][0], B[j][0])
                right := min(A[i][1], B[j][1])
                if left <= right {
                        res = append(res, []int{left, right})
                }
                if A[i][1] < B[j][1] {
                        i++
                } else {
                        j++
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

29.46 987. 二叉树的垂序遍历 (2)

- 题目

```
给定二叉树，按垂序遍历返回其结点值。
对位于(X, Y)的每个结点而言，其左右子结点分别位于(X-1, Y-1)和(X+1, Y-1)。
把一条垂线从X = -infinity移动到X = +infinity，
每当该垂线与结点接触时，我们按从上到下的顺序报告结点的值（Y坐标递减）。
如果两个结点位置相同，则首先报告的结点值较小。
按X坐标顺序返回非空报告的列表。每个报告都有一个结点值列表。
示例 1：输入：[3,9,20,null,null,15,7] 输出：[[9],[3,15],[20],[7]]
解释：  在不丧失其普遍性的情况下，我们可以假设根结点位于 (0, 0)：
然后，值为 9 的结点出现在 (-1, -1)；
值为 3 和 15 的两个结点分别出现在 (0, 0) 和 (0, -2)；
值为 20 的结点出现在 (1, -1)；
值为 7 的结点出现在 (2, -2)。
示例 2：输入：[1,2,3,4,5,6,7] 输出：[[4],[2],[1,5,6],[3],[7]]
解释：根据给定的方案，值为 5 和 6 的两个结点出现在同一位置。
然而，在报告 "[1,5,6]" 中，结点值 5 排在前面，因为 5 小于 6。
提示：树的结点数介于 1和1000之间。
每个结点值介于0和1000之间。
```

- 解题思路

```
var m map[int][][2]int

func verticalTraversal(root *TreeNode) [][]int {
        m = make(map[int][][2]int)
        res := make([][]int, 0)
        dfs(root, 0, 0)
        arr := make([]int, 0)
        for k := range m {
                arr = append(arr, k)
        }
        sort.Ints(arr)
        for i := 0; i < len(arr); i++ {
                temp := m[arr[i]]
                sort.Slice(temp, func(i, j int) bool {
                        if temp[i][1] == temp[j][1] {
                                return temp[i][0] < temp[j][0]
                        }
                        return temp[i][1] < temp[j][1]
                })
                tempArr := make([]int, 0)
```

(续下页)

(接上页)

```
                for j := 0; j < len(temp); j++ {
                        tempArr = append(tempArr, temp[j][0])
                }
                res = append(res, tempArr)
        }
        return res
}

func dfs(root *TreeNode, x, y int) {
        if root == nil {
                return
        }
        m[x] = append(m[x], [2]int{root.Val, y})
        dfs(root.Left, x-1, y+1)
        dfs(root.Right, x+1, y+1)
}

# 2
var m map[int][][2]int

func verticalTraversal(root *TreeNode) [][]int {
        m = make(map[int][][2]int)
        res := make([][]int, 0)
        bfs(root, 0, 0)
        arr := make([]int, 0)
        for k := range m {
                arr = append(arr, k)
        }
        sort.Ints(arr)
        for i := 0; i < len(arr); i++ {
                temp := m[arr[i]]
                sort.Slice(temp, func(i, j int) bool {
                        if temp[i][1] == temp[j][1] {
                                return temp[i][0] < temp[j][0]
                        }
                        return temp[i][1] < temp[j][1]
                })
                tempArr := make([]int, 0)
                for j := 0; j < len(temp); j++ {
                        tempArr = append(tempArr, temp[j][0])
                }
                res = append(res, tempArr)
        }
```

(续下页)

(接上页)

```
        return res
}

func bfs(root *TreeNode, x, y int) {
        if root == nil {
                return
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        queueArr := make([][2]int, 0)
        queueArr = append(queueArr, [2]int{0, 0})
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        node := queue[i]
                        x, y := queueArr[i][0], queueArr[i][1]
                        m[x] = append(m[x], [2]int{node.Val, y})
                        if node.Left != nil {
                                queue = append(queue, node.Left)
                                queueArr = append(queueArr, [2]int{x - 1, y + 1})
                        }
                        if node.Right != nil {
                                queue = append(queue, node.Right)
                                queueArr = append(queueArr, [2]int{x + 1, y + 1})
                        }
                }
                queue = queue[length:]
                queueArr = queueArr[length:]
        }
}
```

29.47 988. 从叶结点开始的最小字符串 (2)

- 题目

```
给定一颗根结点为root的二叉树，树中的每一个结点都有一个从0 到25的值，
分别代表字母'a' 到'z': 值0 代表'a'，值1代表'b'，依此类推。
找出按字典序最小的字符串，该字符串从这棵树的一个叶结点开始，到根结点结束。
（小贴士：字符串中任何较短的前缀在字典序上都是较小的：例如，在字典序上"ab" 比"aba
↪"要小。叶结点是指没有子结点的结点。）
示例 1：输入：[0,1,2,3,4,3,4] 输出："dba"
示例 2：输入：[25,1,3,1,3,0,2] 输出："adz"
```

(续下页)

(接上页)

```
示例 3: 输入: [2,2,1,null,1,0,null,0] 输出: "abc"
提示: 给定树的结点数介于1 和8500之间。
树中的每个结点都有一个介于0和25之间的值。
```

- 解题思路

```
var res string

func smallestFromLeaf(root *TreeNode) string {
        res = ""
        dfs(root, make([]byte, 0))
        return res
}

func dfs(root *TreeNode, arr []byte) {
        if root == nil {
                return
        }
        arr = append(arr, byte('a'+root.Val))
        if root.Left == nil && root.Right == nil {
                for i := 0; i < len(arr)/2; i++ {
                        arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
                }
                if string(arr) < res || res == "" {
                        res = string(arr)
                }
                for i := 0; i < len(arr)/2; i++ {
                        arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
                }
                return
        }
        dfs(root.Left, arr)
        dfs(root.Right, arr)
}

# 2
var res string

func smallestFromLeaf(root *TreeNode) string {
        res = ""
        dfs(root, make([]byte, 0))
        return res
}
```

(续下页)

(接上页)

```
func dfs(root *TreeNode, arr []byte) {
        if root == nil {
                return
        }
        arr = append([]byte{byte('a' + root.Val)}, arr...)
        if root.Left == nil && root.Right == nil {
                if string(arr) < res || res == "" {
                        res = string(arr)
                }
                return
        }
        dfs(root.Left, arr)
        dfs(root.Right, arr)
}
```

29.48 990. 等式方程的可满足性 (1)

- 题目

```
给定一个由表示变量之间关系的字符串方程组成的数组，每个字符串方程 equations[i]␣
↪的长度为 4，
并采用两种不同的形式之一："a==b" 或"a!=b"。在这里，a 和 b␣
↪是小写字母（不一定不同），表示单字母变量名。
只有当可以将整数分配给变量名，以便满足所有给定的方程时才返回true，否则返回 false。
示例 1：输入：["a==b","b!=a"] 输出：false
解释：如果我们指定，a = 1 且 b = 1，那么可以满足第一个方程，但无法满足第二个方程。
没有办法分配变量同时满足这两个方程。
示例 2：输入：["b==a","a==b"] 输出：true
解释：我们可以指定 a = 1 且 b = 1 以满足满足这两个方程。
示例 3：输入：["a==b","b==c","a==c"] 输出：true
示例 4：输入：["a==b","b!=c","c==a"] 输出：false
示例 5：输入：["c==c","b==d","x!=z"] 输出：true
提示：1 <= equations.length <= 500
equations[i].length == 4
equations[i][0] 和equations[i][3]是小写字母
equations[i][1] 要么是'='，要么是'!'
equations[i][2]是'='
```

- 解题思路

```
func equationsPossible(equations []string) bool {
        fa := make([]int, 26)
```

(续下页)

(接上页)

```
        for i := 0; i < 26; i++ {
                fa[i] = i
        }
        for i := 0; i < len(equations); i++ {
                if equations[i][1] == '=' {
                        a, b := int(equations[i][0]-'a'), int(equations[i][3]-'a')
                        union(fa, a, b)
                }
        }
        for i := 0; i < len(equations); i++ {
                if equations[i][1] == '!' {
                        a, b := int(equations[i][0]-'a'), int(equations[i][3]-'a')
                        if find(fa, a) == find(fa, b) {
                                return false
                        }
                }
        }
        return true
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}

func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
                a = fa[a]
        }
        return a
}
```

29.49 991. 坏了的计算器 (2)

- 题目

```
在显示着数字的坏计算器上，我们可以执行以下两种操作：
双倍（Double）：将显示屏上的数字乘 2；
递减（Decrement）：将显示屏上的数字减 1 。
最初，计算器显示数字X。
返回显示数字Y所需的最小操作数。
示例 1：输入：X = 2, Y = 3 输出：2
```

(续下页)

(接上页)

```
解释：先进行双倍运算，然后再进行递减运算 {2 -> 4 -> 3}.
示例 2：输入：X = 5, Y = 8 输出：2
解释：先递减，再双倍 {5 -> 4 -> 8}.
示例 3：输入：X = 3, Y = 10 输出：3
解释：先双倍，然后递减，再双倍 {3 -> 6 -> 5 -> 10}.
示例 4：输入：X = 1024, Y = 1 输出：1023
解释：执行递减运算 1023 次
提示：1 <= X <= 10^9
1 <= Y <= 10^9
```

- 解题思路

```
func brokenCalc(X int, Y int) int {
        if X > Y {
                return X - Y
        }
        res := 0
        for X < Y {
                if Y%2 == 0 {
                        Y = Y / 2
                        res++
                } else {
                        Y = (Y + 1) / 2
                        res = res + 2
                }
        }
        return res + X - Y
}

# 2
func brokenCalc(X int, Y int) int {
        if X > Y {
                return X - Y
        }
        res := 0
        for X < Y {
                res++
                if Y%2 == 1 {
                        Y++
                } else {
                        Y = Y / 2
                }
        }
        return res + X - Y
```

(续下页)

(接上页)

```
}
```

29.50 994. 腐烂的橘子 (2)

- 题目

```
在给定的网格中，每个单元格可以有以下三个值之一：
    值 0 代表空单元格；
    值 1 代表新鲜橘子；
    值 2 代表腐烂的橘子。
每分钟，任何与腐烂的橘子（在 4 个正方向上）相邻的新鲜橘子都会腐烂。
返回直到单元格中没有新鲜橘子为止所必须经过的最小分钟数。如果不可能，返回 -1。
示例 1：输入：[[2,1,1],[1,1,0],[0,1,1]] 输出：4
示例 2：输入：[[2,1,1],[0,1,1],[1,0,1]] 输出：-1
解释：左下角的橘子（第 2 行， 第 0 列）永远不会腐烂，因为腐烂只会发生在 4 个正向上。
示例 3：输入：[[0,2]] 输出：0
解释：因为 0 分钟时已经没有新鲜橘子了，所以答案就是 0 。
提示：
    1 <= grid.length <= 10
    1 <= grid[0].length <= 10
    grid[i][j] 仅为 0、1 或 2
```

- 解题思路

```
// 上、右、下、左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func orangesRotting(grid [][]int) int {
        queue := make([][]int, 0)
        count := 0
        times := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 2 {
                                queue = append(queue, []int{i, j})
                        } else if grid[i][j] == 1 {
                                count = count + 1
                        }
                }
        }
        for len(queue) > 0 && count > 0 {
```

(续下页)

(接上页)

```
                times++
                length := len(queue)
                for i := 0; i < length; i++ {
                        for j := 0; j < 4; j++ {
                                x := queue[i][0] + dx[j]
                                y := queue[i][1] + dy[j]
                                if x >= 0 && x < len(grid) &&
                                        y >= 0 && y < len(grid[0]) && grid[x][y] == 1
↪{
                                        grid[x][y] = 2
                                        queue = append(queue, []int{x, y})
                                        count--
                                }
                        }
                }
                queue = queue[length:]
        }
        if count > 0 {
                return -1
        }
        return times
}

# 2
// 上、右、下、左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func orangesRotting(grid [][]int) int {
        queue := make([][]int, 0)
        count := 0
        times := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 2 {
                                queue = append(queue, []int{i, j})
                        } else if grid[i][j] == 1 {
                                count = count + 1
                        }
                }
        }
        for len(queue) > 0 {
                times++
```

(续下页)

(接上页)

```
                length := len(queue)
                for i := 0; i < length; i++ {
                        for j := 0; j < 4; j++ {
                                x := queue[i][0] + dx[j]
                                y := queue[i][1] + dy[j]
                                if x >= 0 && x < len(grid) &&
                                        y >= 0 && y < len(grid[0]) && grid[x][y] == 1
↪{
                                        grid[x][y] = 2
                                        queue = append(queue, []int{x, y})
                                        count--
                                }
                        }
                }
                queue = queue[length:]
                if len(queue) == 0 {
                        times--
                }
        }
        if count > 0 {
                return -1
        }
        return times
}
```

29.51 998. 最大二叉树 II(2)

- 题目

```
最大树定义：一个树，其中每个节点的值都大于其子树中的任何其他值。
给出最大树的根节点 root。
就像之前的问题那样，给定的树是从列表A (root = Construct(A))
递归地使用下述Construct(A)例程构造的：
如果A为空，返回null
否则，令A[i]作为 A 的最大元素。创建一个值为A[i]的根节点 root
root的左子树将被构建为Construct([A[0], A[1], ..., A[i-1]])
root的右子树将被构建为 Construct([A[i+1], A[i+2], ..., A[A.length - 1]])
返回root
请注意，我们没有直接给定A，只有一个根节点root = Construct(A).
假设 B 是 A 的副本，并在末尾附加值 val。题目数据保证 B中的值是不同的。
返回Construct(B)。
示例 1：输入：root = [4,1,3,null,null,2], val = 5 输出：[5,4,null,1,3,null,null,2]
```

(续下页)

(接上页)

```
解释：A = [1,4,2,3], B = [1,4,2,3,5]
示例 2：输入：root = [5,2,4,null,1], val = 3 输出：[5,2,4,null,1,null,3]
解释：A = [2,1,5,4], B = [2,1,5,4,3]
示例 3：输入：root = [5,2,3,null,1], val = 4 输出：[5,2,4,null,1,3]
解释：A = [2,1,5,3], B = [2,1,5,3,4]
提示：1 <= B.length <= 100
```

- 解题思路

```
func insertIntoMaxTree(root *TreeNode, val int) *TreeNode {
   if root == nil {
           return &TreeNode{
                   Val:   val,
           }
   }
   if val > root.Val {
           return &TreeNode{
                   Val:   val,
                   Left:  root,
           }
   }
   root.Right = insertIntoMaxTree(root.Right, val)
   return root
}

# 2
func insertIntoMaxTree(root *TreeNode, val int) *TreeNode {
   newRoot := root
   if root.Val < val {
           newRoot = &TreeNode{
                   Val:  val,
                   Left: root,
           }
   } else if root.Right == nil {
           root.Right = &TreeNode{Val: val}
   } else {
           root.Right = insertIntoMaxTree(root.Right, val)
   }
   return newRoot
}
```

CHAPTER 30

0901-1000-Hard

30.1 902. 最大为 N 的数字组合 (1)

- 题目

```
我们有一组排序的数字 D，它是 {'1','2','3','4','5','6','7','8','9'}
↪的非空子集。（请注意，'0' 不包括在内。）
现在，我们用这些数字进行组合写数字，想用多少次就用多少次。
例如D = {'1','3','5'}，我们可以写出像'13', '551', '1351315'这样的数字。
返回可以用 D 中的数字写出的小于或等于 N 的正整数的数目。
示例 1：输入：D = ["1","3","5","7"], N = 100 输出：20
解释：可写出的 20 个数字是：
1, 3, 5, 7, 11, 13, 15, 17, 31, 33, 35, 37, 51, 53, 55, 57, 71, 73, 75, 77.
示例 2：输入：D = ["1","4","9"], N = 1000000000 输出：29523
解释：我们可以写 3 个一位数字，9 个两位数字，27 个三位数字，
81 个四位数字，243 个五位数字，729 个六位数字，
2187 个七位数字，6561 个八位数字和 19683 个九位数字。
总共，可以使用D中的数字写出 29523 个整数。
提示：D 是按排序顺序的数字 '1'-'9' 的子集。
1 <= N <= 10^9
```

- 解题思路

```
func atMostNGivenDigitSet(digits []string, n int) int {
        str := fmt.Sprintf("%d", n)
```

(续下页)

(接上页)

```
        m := len(digits)
        k := len(str)
        dp := make([]int, k+1) // dp[i] 表示 长度为i的N（最后i位）的个数
        dp[0] = 1
        // 1、考虑k位数的可能
        for i := 1; i <= k; i++ {
                value := str[k-i] - '0'
                for j := 0; j < len(digits); j++ {
                        v := digits[j][0] - '0'
                        if v < value { // 小于，剩下i-1位随便用： +  m^(i-1)
                                dp[i] = dp[i] + int(math.Pow(float64(m), float64(i-
↪1)))
                        } else if v == value { // 等于：考虑i-1位
                                dp[i] = dp[i] + dp[i-1]
                        }
                }
        }
        // 2、考虑非k位数的可能
        for i := 1; i < k; i++ {
                dp[k] = dp[k] + int(math.Pow(float64(m), float64(i))) // + m^i
        }
        return dp[k]
}
```

30.2 927. 三等分 (1)

- 题目

```
给定一个由 0 和 1 组成的数组A，将数组分成␣
↪3个非空的部分，使得所有这些部分表示相同的二进制值。
如果可以做到，请返回任何[i, j]，其中 i+1 < j，这样一来：
A[0], A[1], ..., A[i]组成第一部分；
A[i+1], A[i+2], ..., A[j-1]作为第二部分；
A[j], A[j+1], ..., A[A.length - 1] 是第三部分。
这三个部分所表示的二进制值相等。
如果无法做到，就返回[-1, -1]。
注意，在考虑每个部分所表示的二进制时，应当将其看作一个整体。例如，[1,1,
↪0]表示十进制中的6，而不会是3。
此外，前导零也是被允许的，所以[0,1,1] 和[1,1]表示相同的值。
示例 1：输入：[1,0,1,0,1] 输出：[0,3]
示例 2：输出：[1,1,0,1,1] 输出：[-1,-1]
提示：3 <= A.length <= 30000
```

(续下页)

(接上页)

```
A[i] == 0或A[i] == 1
```

- 解题思路

```
func threeEqualParts(arr []int) []int {
        res := []int{-1, -1}
        n := len(arr)
        indexArr := make([]int, 0)
        for i := 0; i < n; i++ {
                if arr[i] == 1 {
                        indexArr = append(indexArr, i)
                }
        }
        count := len(indexArr)
        if count == 0 {
                return []int{0, 2}
        }
        if count%3 != 0 {
                return res
        }
        // 3个部分，每个部分的1的个数都是一样的
        a, b := count/3, count/3*2 // 第2、3组开始的位置
        i, j, k := indexArr[0], indexArr[a], indexArr[b]
        for k < n && arr[i] == arr[j] && arr[j] == arr[k] {
                i++
                j++
                k++
        }
        if k == n {
                return []int{i - 1, j}
        }
        return res
}
```

30.3 940. 不同的子序列 II(3)

- 题目

```
给定一个字符串S，计算S的不同非空子序列的个数。
因为结果可能很大，所以返回答案模 10^9 + 7.
示例 1：输入："abc" 输出：7
解释：7 个不同的子序列分别是 "a", "b", "c", "ab", "ac", "bc", 以及 "abc"。
```

(续下页)

(接上页)

```
示例 2: 输入: "aba" 输出: 6
解释: 6 个不同的子序列分别是 "a", "b", "ab", "ba", "aa" 以及 "aba"。
示例 3: 输入: "aaa" 输出: 3
解释: 3 个不同的子序列分别是 "a", "aa" 以及 "aaa"。
提示: S只包含小写字母。
1 <= S.length <= 2000
```

- 解题思路

```
var mod = 1000000007

func distinctSubseqII(s string) int {
        n := len(s)
        dp := make([]int, n+1)  // dp[i] 表示s[0:i+1]的不同子序列数
        m := make(map[byte]int) // 保存上次出现的下标
        for i := 0; i < n; i++ {
                if index, ok := m[s[i]]; ok { // 该字符出现过，取上次坐标
                        dp[i+1] = (2*dp[i] - dp[index] + mod) % mod // 减去重复的
                } else {
                        dp[i+1] = (2*dp[i] + 1) % mod // 这个不包含空序列
                }
                m[s[i]] = i
        }
        return dp[n] % mod
}

# 2
var mod = 1000000007

func distinctSubseqII(s string) int {
        n := len(s)
        dp := make([]int, n+1)  // dp[i] 表示s[0:i+1]的不同子序列数
        dp[0] = 1               // 空序列
        m := make(map[byte]int) // 保存上次出现的下标
        for i := 0; i < n; i++ {
                if index, ok := m[s[i]]; ok { // 该字符出现过，取上次坐标
                        dp[i+1] = (2*dp[i] - dp[index] + mod) % mod // 减去重复的
                } else {
                        // 如 ab 含有4种情况: "" "a" "b" "ab"
                        // 如加入c后的abc 含有8种情况: "" "a" "b" "ab" "c" "ac" "bc"
↪"abc"
                        dp[i+1] = 2 * dp[i] % mod // 不重复翻倍
                }
                m[s[i]] = i
```

(续下页)

(接上页)

```
        }
        return (dp[n] - 1) % mod // 去除空序列
}

# 3
var mod = 1000000007

func distinctSubseqII(s string) int {
        n := len(s)
        dp := make([]int, n+1) // dp[i]表示：长度为i的组合数
        dp[0] = 1
        for i := 1; i <= n; i++ {
                for j := 1; j < i; j++ {
                        if s[i-1] != s[j-1] {
                                dp[i] = (dp[i] + dp[j]) % mod
                        }
                }
                dp[i]++ // 当前dp[i]时：最长为i只有1种
        }
        res := 0
        for i := 1; i <= n; i++ {
                res = (res + dp[i]) % mod
        }
        return res
}
```

30.4 956. 最高的广告牌 (5)

- 题目

```
你正在安装一个广告牌，并希望它高度最大。这块广告牌将有两个钢制支架，两边各一个。每个钢支架的高度必须
你有一堆可以焊接在一起的钢筋 rods。
举个例子，如果钢筋的长度为 1、2 和 3，则可以将它们焊接在一起形成长度为 6 的支架。
返回广告牌的最大可能安装高度。如果没法安装广告牌，请返回 0。
示例 1：输入：[1,2,3,6] 输出：6
解释：我们有两个不相交的子集 {1,2,3} 和 {6}，它们具有相同的和 sum = 6。
示例 2：输入：[1,2,3,4,5,6] 输出：10
解释：我们有两个不相交的子集 {2,3,5} 和 {4,6}，它们具有相同的和 sum = 10。
示例 3：输入：[1,2] 输出：0
解释：没法安装广告牌，所以返回 0。
提示：0 <= rods.length <= 20
    1 <= rods[i] <= 1000
```

(续下页)

(接上页)

```
    钢筋的长度总和最多为 5000
```

- 解题思路

```
const MinValue = math.MinInt32 / 100

// 官方题解(从后往前比较难理解)
// dp[i][s] 表示当我们可以使用 rods[j] (j >= i) 时能得到的最大 score
// dp[i][s] = max(dp[i+1][s], dp[i+1][s-rods[i]], rods[i] + dp[i+1][s+rods[i]])
// 例如:rods=[1,2,3,6],可以有dp[1][1]=5,
// 在写下1之后,可以写下+2,+3,-6使得剩下的rods[i:]获得score为5
var dp [][]int

func tallestBillboard(rods []int) int {
        dp = make([][]int, len(rods))
        for i := 0; i < len(rods); i++ {
                dp[i] = make([]int, 10001)
        }
        res := dfs(rods, 0, 5000)
        return res
}

func dfs(rods []int, index, total int) int {
        if index == len(rods) {
                if total == 5000 {
                        return 0
                }
                return MinValue
        }
        if dp[index][total] != 0 {
                return dp[index][total]
        }
        res := dfs(rods, index+1, total)
        res = max(res, dfs(rods, index+1, total-rods[index]))
        res = max(res, dfs(rods, index+1, total+rods[index])+rods[index])
        dp[index][total] = res
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

# 2
func tallestBillboard(rods []int) int {
        dp := make(map[int]int) // 保存和为i中正整数的和
        dp[0] = 0
        // 每个rods[i]有3个选择，+rods[i]、-rods[i]、0
        for i := 0; i < len(rods); i++ {
                value := rods[i]
                temp := make(map[int]int)
                for k, v := range dp {
                        temp[k+value] = max(temp[k+value], v+value) // +value>
↪0，是正整数，需要加上
                        temp[k] = max(temp[k], v)
                        temp[k-value] = max(temp[k-value], v) // -value<0，不需要加上
                }
                dp = temp
        }
        return dp[0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func tallestBillboard(rods []int) int {
        dp := make(map[int]int) // 保存高度差为i时最低一边的高度
        dp[0] = 0
        // 每个rods[i]有3个选择，+rods[i]、-rods[i]、0
        for i := 0; i < len(rods); i++ {
                temp := make(map[int]int)
                for k, v := range dp {
                        temp[k] = v
                }
                for k, v := range temp {
                        dp[k+rods[i]] = max(dp[k+rods[i]], v)                    ␣
↪  // 往高的一侧加，高度差变大，但最低一边高度不变
                        dp[k] = dp[k]                                            ␣
↪  // 不加，高度差不变，可忽略
```

(续下页)

(接上页)

```
                        dp[abs(k-rods[i])] = max(dp[abs(k-rods[i])], v+min(k,␣
↪rods[i])) // 往低的一侧加，高度差有变化，高度差有增长
                }
        }
        return dp[0]
}

func abs(a int) int {
        if a >= 0 {
                return a
        }
        return -a
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
// dp[i][s] 表示当我们可以使用 rods[j] (j >= i) 时能得到的最大 score
// dp[i][s] = max(dp[i+1][s], dp[i+1][s-rods[i]], rods[i] + dp[i+1][s+rods[i]])
// 例如:rods=[1,2,3,6],可以有dp[1][1]=5,
// 在写下1之后,可以写下+2,+3,-6使得剩下的rods[i:]获得score为5
const MinValue = math.MinInt32 / 100

func tallestBillboard(rods []int) int {
        dp := make([][]int, len(rods)+1)
        for i := 0; i < len(rods)+1; i++ {
                dp[i] = make([]int, 10001)
        }
        for j := 0; j < len(dp[len(rods)]); j++ {
                dp[len(rods)][j] = MinValue
        }
```

(续下页)

(接上页)

```
        sum := 0
        for i := 0; i < len(rods); i++ {
                sum = sum + rods[i]
        }
        m := 2*sum
        dp[len(rods)][sum] = 0
        for i := len(rods) - 1; i >= 0; i-- {
                for s := rods[i]; s <= m-rods[i]; s++ {
                        dp[i][s] = max(dp[i+1][s], max(dp[i+1][s-rods[i]],␣
↪rods[i]+dp[i+1][s+rods[i]]))
                }
        }
        return dp[0][sum]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 5
func tallestBillboard(rods []int) int {
        leftM := makeM(rods, 0, len(rods)/2)
        rightM := makeM(rods, len(rods)/2, len(rods))
        res := 0
        for k := range leftM {
                if rightM[k] > 0 {
                        res = max(res, leftM[k]+rightM[-k])
                }
        }
        return res
}

func makeM(rods []int, left, right int) map[int]int {
        dp := make([][2]int, 100001)
        dp[0] = [2]int{0, 0}
        count := 1
        for i := left; i < right; i++ {
                length := count
                for j := 0; j < length; j++ {
                        dp[count] = [2]int{dp[j][0] + rods[i], dp[j][1]}
```

(续下页)

(接上页)

```
                        count++
                        dp[count] = [2]int{dp[j][0], dp[j][1] + rods[i]}
                        count++
                }
        }
        m := make(map[int]int)
        for i := 0; i < count; i++ {
                a := dp[i][0]
                b := dp[i][1]
                m[a-b] = max(m[a-b], a)
        }
        return m
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

30.5 968. 监控二叉树 (2)

- 题目

```
给定一个二叉树，我们在树的节点上安装摄像头。
节点上的每个摄影头都可以监视其父对象、自身及其直接子对象。
计算监控树的所有节点所需的最小摄像头数量。
示例 1：输入：[0,0,null,0,0] 输出：1
解释：如图所示，一台摄像头足以监控所有节点。
示例 2：输入：[0,0,null,0,null,0,null,null,0] 输出：2
解释：需要至少两个摄像头来监视树的所有节点。 上图显示了摄像头放置的有效位置之一。
提示：给定树的节点数的范围是[1, 1000]。 每个节点的值都是 0。
```

- 解题思路

```
var maxValue = math.MaxInt32 / 10

func minCameraCover(root *TreeNode) int {
        _, res, _ := dfs(root)
        return res
}
```

(续下页)

(接上页)

```
func dfs(root *TreeNode) (a, b, c int) {
        if root == nil {
                return maxValue, 0, 0
        }
        la, lb, lc := dfs(root.Left)
        ra, rb, rc := dfs(root.Right)
        a = lc + rc + 1                 //␣
↪root必须放置摄像头的情况下，覆盖整棵树需要的摄像头数目。
        b = min(a, min(la+rb, ra+lb)) // 覆盖整棵树需要的摄像头数目，
↪无论root是否放置摄像头。
        c = min(a, lb+rb)               // 覆盖两棵子树需要的摄像头数目，
↪无论节点root本身是否被监控到。
        return a, b, c
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var res int

func minCameraCover(root *TreeNode) int {
        res = 0
        if dfs(root) == 0 { // root未被监控
                res++
        }
        return res
}

// 0=>未被监控， 1=>已监控， 2=>安装监控
func dfs(root *TreeNode) int {
        if root == nil {
                return 1 // 空节点假设有监控
        }
        left, right := dfs(root.Left), dfs(root.Right)
        if left == 1 && right == 1 { //␣
↪左右节点都有监控，比如都是空节点，那么当前节点是无未被监控
                return 0
```

(续下页)

(接上页)

```
        }
        if left+right >= 3 { // 1+2 / 2+1 / 2+2 => 当前节点已经被监控
                return 1
        }
        res++ // 1+0, 0+1, 0+2, 2+0, 0+0 => 需要安装监控
        return 2
}
```

30.6 980. 不同路径 III(1)

- 题目

```
在二维网格 grid 上，有 4 种类型的方格：
1 表示起始方格。且只有一个起始方格。
2 表示结束方格，且只有一个结束方格。
0 表示我们可以走过的空方格。
-1 表示我们无法跨越的障碍。
返回在四个方向（上、下、左、右）上行走时，从起始方格到结束方格的不同路径的数目。
每一个无障碍方格都要通过一次，但是一条路径中不能重复通过同一个方格。
示例 1：输入：[[1,0,0,0],[0,0,0,0],[0,0,2,-1]] 输出：2
解释：我们有以下两条路径：
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2)
2. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2)
示例 2：输入：[[1,0,0,0],[0,0,0,0],[0,0,0,2]] 输出：4
解释：我们有以下四条路径：
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2),(2,3)
2. (0,0),(0,1),(1,1),(1,0),(2,0),(2,1),(2,2),(1,2),(0,2),(0,3),(1,3),(2,3)
3. (0,0),(1,0),(2,0),(2,1),(2,2),(1,2),(1,1),(0,1),(0,2),(0,3),(1,3),(2,3)
4. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2),(2,3)
示例 3：输入：[[0,1],[2,0]] 输出：0
解释：没有一条路能完全穿过每一个空的方格一次。
请注意，起始和结束方格可以位于网格中的任意位置。
提示：1 <= grid.length * grid[0].length <= 20
```

- 解题思路

```
var res int
var n, m int

func uniquePathsIII(grid [][]int) int {
        res = 0
        n, m = len(grid), len(grid[0])
```

(续下页)

(接上页)

```
        visited := make([][]bool, n)
        for i := 0; i < n; i++ {
                visited[i] = make([]bool, m)
        }
        x, y := 0, 0
        count := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if grid[i][j] == 1 {
                                x, y = i, j
                                continue
                        }
                        if grid[i][j] == 0 {
                                count++
                        }
                }
        }
        dfs(grid, x, y, visited, count)
        return res
}

func dfs(grid [][]int, i, j int, visited [][]bool, count int) {
        if i < 0 || i >= n || j < 0 || j >= m ||
                grid[i][j] == -1 || visited[i][j] == true {
                return
        }
        if grid[i][j] == 2 {
                if count == -1 { // 包括起始点1
                        res++
                }
                return
        }
        visited[i][j] = true
        dfs(grid, i, j+1, visited, count-1)
        dfs(grid, i, j-1, visited, count-1)
        dfs(grid, i+1, j, visited, count-1)
        dfs(grid, i-1, j, visited, count-1)
        visited[i][j] = false
}
```

30.7 992.K 个不同整数的子数组 (2)

- 题目

```
给定一个正整数数组 A，如果 A的某个子数组中不同整数的个数恰好为 K，则称 A
↪的这个连续、不一定不同的子数组为好子数组。
（例如，[1,2,3,1,2] 中有3个不同的整数：1，2，以及3。）
返回A中好子数组的数目。
示例 1：输入：A = [1,2,1,2,3], K = 2 输出：7
解释：恰好由 2 个不同整数组成的子数组：[1,2], [2,1], [1,2], [2,3], [1,2,1], [2,1,2],
↪[1,2,1,2].
示例 2：输入：A = [1,2,1,3,4], K = 3 输出：3
解释：恰好由 3 个不同整数组成的子数组：[1,2,1,3], [2,1,3], [1,3,4].
提示：1 <= A.length <= 20000
1 <= A[i] <= A.length
1 <= K <= A.length
```

- 解题思路

```
func subarraysWithKDistinct(nums []int, k int) int {
        return getK(nums, k) - getK(nums, k-1) // 恰好K个不同=最多K个不同-最多K-
↪1个不同
}

// 求最多K个不同
func getK(nums []int, k int) int {
        res := 0
        n := len(nums)
        m := make(map[int]int)
        left, right := 0, 0
        count := 0
        for ; right < n; right++ {
                cur := nums[right]
                if m[cur] == 0 {
                        count++ // 次数+1
                }
                m[cur]++
                for count > k {
                        m[nums[left]]--
                        if m[nums[left]] == 0 {
                                count--
                        }
                        left++
                }
```

(续下页)

(接上页)

```
                res = res + right - left
        }
        return res
}

# 2
func subarraysWithKDistinct(nums []int, k int) int {
        res := 0
        n := len(nums)
        m1 := make(map[int]int)
        m2 := make(map[int]int)
        left1, left2 := 0, 0
        count1, count2 := 0, 0
        for i := 0; i < n; i++ {
                cur := nums[i]
                if m1[cur] == 0 {
                        count1++
                }
                if m2[cur] == 0 {
                        count2++
                }
                m1[cur]++
                m2[cur]++
                for count1 > k {
                        m1[nums[left1]]--
                        if m1[nums[left1]] == 0 {
                                count1--
                        }
                        left1++
                }
                for count2 > k-1 {
                        m2[nums[left2]]--
                        if m2[nums[left2]] == 0 {
                                count2--
                        }
                        left2++
                }
                res = res + left2 - left1
        }
        return res
}
```

30.8 995.K 连续位的最小翻转次数 (4)

- 题目

```
在仅包含 0 和 1 的数组 A 中，一次 K 位翻转包括选择一个长度为 K 的（连续）子数组，
同时将子数组中的每个 0 更改为 1，而每个 1 更改为 0。
返回所需的 K 位翻转的最小次数，以便数组没有值为 0 的元素。如果不可能，返回 -1。
示例 1：输入：A = [0,1,0], K = 1 输出：2
解释：先翻转 A[0]，然后翻转 A[2]。
示例 2：输入：A = [1,1,0], K = 2 输出：-1
解释：无论我们怎样翻转大小为 2 的子数组，我们都不能使数组变为 [1,1,1]。
示例 3：输入：A = [0,0,0,1,0,1,1,0], K = 3 输出：3
解释：翻转 A[0],A[1],A[2]:A变成 [1,1,1,1,0,1,1,0]
翻转 A[4],A[5],A[6]:A变成 [1,1,1,1,1,0,0,0]
翻转 A[5],A[6],A[7]:A变成 [1,1,1,1,1,1,1,1]
提示：1 <= A.length <=30000
1 <= K <= A.length
```

- 解题思路

```
func minKBitFlips(nums []int, k int) int {
        n := len(nums)
        arr := make([]int, n+1) // 差分数组
        res := 0
        count := 0 // 翻转奇数偶数次
        for i := 0; i < n; i++ {
                count = count + arr[i]
                if (count+nums[i])%2 == 0 { // 当前位置翻转count次后是0
                        if i+k > n { // 剩下达不到k个一组
                                return -1
                        }
                        count++    // 翻转+1
                        arr[i]++   // 差分数组i 加1
                        arr[i+k]-- // 差分数组i+k 减1
                        res++
                }
        }
        return res
}

# 2
func minKBitFlips(nums []int, k int) int {
        n := len(nums)
        res := 0
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                if nums[i] == 0 {
                        if i+k > n {
                                return -1
                        }
                        for j := i; j < i+k; j++ {
                                nums[j] = 1 - nums[j]
                        }
                        res++
                }
        }
        return res
}

# 3
func minKBitFlips(nums []int, k int) int {
        n := len(nums)
        arr := make([]int, n+1) // 差分数组
        res := 0
        count := 0 // 翻转奇数偶数次
        for i := 0; i < n; i++ {
                count = (count + arr[i]) % 2
                if (count+nums[i])%2 == 0 { // 当前位置翻转count次后是0
                        if i+k > n { // 剩下达不到k个一组
                                return -1
                        }
                        count = 1 - count // 翻转+1
                        arr[i]++          // 差分数组i 加1
                        arr[i+k]--        // 差分数组i+k 减1
                        res++
                }
        }
        return res
}

# 4
func minKBitFlips(nums []int, k int) int {
        n := len(nums)
        res := 0
        count := 0 // 翻转奇数偶数次
        for i := 0; i < n; i++ {
                if i >= k && nums[i-k] > 1 {
                        count = 1 - count
```

(续下页)

(接上页)

```
                }
                if (nums[i]+count)%2 == 0 { // 是1翻转奇数次变为1，是0翻转偶数次为0
                        if i+k > n {
                                return -1
                        }
                        res++
                        count = 1 - count
                        nums[i] = nums[i] + 2 // 标记
                }
        }
        return res
}
```

30.9 996. 正方形数组的数目

30.9.1 题目

```
给定一个非负整数数组A，如果该数组每对相邻元素之和是一个完全平方数，则称这一数组为正方形数组。

返回 A 的正方形排列的数目。两个排列 A1 和 A2 不同的充要条件是存在某个索引 i，使得␣
↪A1[i] != A2[i]。



示例 1：

输入：[1,17,8]
输出：2
解释：
[1,8,17] 和 [17,8,1] 都是有效的排列。
示例 2：

输入：[2,2,2]
输出：1


提示：

1 <= A.length <= 12
0 <= A[i] <= 1e9
```

(续下页)

(接上页)

来源：力扣（LeetCode）
链接：https://leetcode.cn/problems/number-of-squareful-arrays
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

30.9.2 解题思路

CHAPTER 31

1001-1100-Easy

31.1 1002. 查找常用字符 (2)

- 题目

```
给定仅有小写字母组成的字符串数组␣
↪A，返回列表中的每个字符串中都显示的全部字符（包括重复字符）组成的列表。
例如，如果一个字符在每个字符串中出现 3 次，但不是 4 次，则需要在最终答案中包含该字符␣
↪3 次。
你可以按任意顺序返回答案。
示例 1：输入：["bella","label","roller"] 输出：["e","l","l"]
示例 2：输入：["cool","lock","cook"] 输出：["c","o"]
提示：
    1 <= A.length <= 100
    1 <= A[i].length <= 100
    A[i][j] 是小写字母
```

- 解题思路

```
func commonChars(A []string) []string {
        arr := [26]int{}
        for _, v := range A[0] {
                arr[v-'a']++
        }
        for i := 1; i < len(A); i++ {
```

(续下页)

(接上页)

```
                temp := [26]int{}
                for _, v := range A[i] {
                        temp[v-'a']++
                }
                for i := 0; i < len(arr); i++ {
                        arr[i] = min(arr[i], temp[i])
                }
        }
        res := make([]string, 0)
        for i := 0; i < len(arr); i++ {
                if arr[i] > 0 {
                        for j := 0; j < arr[i]; j++ {
                                res = append(res, string('a'+i))
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func commonChars(A []string) []string {
        arr := make([][26]int, len(A))
        for i := 0; i < len(A); i++ {
                for j := 0; j < len(A[i]); j++ {
                        arr[i][A[i][j]-'a']++
                }
        }
        res := make([]string, 0)
        for j := 0; j < 26; j++ {
                minValue := arr[0][j]
                for i := 1; i < len(arr); i++ {
                        minValue = min(minValue, arr[i][j])
                }
                for minValue > 0 {
                        res = append(res, string(j+'a'))
                        minValue--
```

(续下页)

(接上页)

```
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

31.2 1005.K 次取反后最大化的数组和 (4)

- 题目

```
给定一个整数数组 A，我们只能用以下方法修改该数组：
我们选择某个个索引 i 并将 A[i] 替换为 -A[i]，然后总共重复这个过程 K 次。
（我们可以多次选择同一个索引 i。）
以这种方式修改数组后，返回数组可能的最大和。
示例 1：输入：A = [4,2,3], K = 1 输出：5
解释：选择索引 (1,) ，然后 A 变为 [4,-2,3]。
示例 2：输入：A = [3,-1,0,2], K = 3 输出：6
解释：选择索引 (1, 2, 2) ，然后 A 变为 [3,1,0,2]。
示例 3：输入：A = [2,-3,-1,5,-4], K = 2 输出：13
解释：选择索引 (1, 4) ，然后 A 变为 [2,3,-1,5,4]。
提示：
    1 <= A.length <= 10000
    1 <= K <= 10000
    -100 <= A[i] <= 100
```

- 解题思路

```
func largestSumAfterKNegations(A []int, K int) int {
        sort.Ints(A)
        i := 0
        for i < len(A) && K > 0 {
                if A[i] < 0 {
                        A[i] = -A[i]
                        i++
                        K--
                } else {
```

(续下页)

(接上页)

```
                        break
                }
        }
        sort.Ints(A)
        if K%2 == 1 {
                A[0] = -A[0]
        }
        return sum(A)
}

func sum(A []int) int {
        res := 0
        for i := 0; i < len(A); i++ {
                res = res + A[i]
        }
        return res
}

# 2
func largestSumAfterKNegations(A []int, K int) int {
        sort.Ints(A)
        i := 0
        for i < len(A)-1 && K > 0 {
                A[i] = -A[i]
                if A[i] > 0 && A[i] > A[i+1] {
                        i++
                }
                K--
        }
        return sum(A)
}

func sum(A []int) int {
        res := 0
        for i := 0; i < len(A); i++ {
                res = res + A[i]
        }
        return res
}

# 3
func largestSumAfterKNegations(A []int, K int) int {
        arr := make([]int, 201)
```

(续下页)

(接上页)

```
        for i := 0; i < len(A); i++ {
                arr[A[i]+100]++
        }
        i := 0
        for K > 0 {
                for arr[i] == 0 {
                        i++
                }
                if i > 100 {
                        break
                }
                arr[i]--
                arr[200-i]++
                K--
        }
        if K%2 == 1 && i != 100 {
                for j := i; j < len(arr); j++ {
                        if arr[j] > 0 {
                                arr[j]--
                                arr[200-j]++
                                break
                        }
                }
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                res = res + (i-100)*arr[i]
        }
        return res
}

# 4
func largestSumAfterKNegations(A []int, K int) int {
        for K > 0 {
                minIndex, minValue := findMin(A)
                if minValue > 0 {
                        break
                }
                A[minIndex] = -A[minIndex]
                K--
        }
        if K%2 == 1 {
                minIndex, _ := findMin(A)
```

(续下页)

(接上页)

```
                A[minIndex] = -A[minIndex]
        }
        res := 0
        for i := 0; i < len(A); i++ {
                res = res + A[i]
        }
        return res
}

func findMin(A []int) (int, int) {
        res := A[0]
        index := 0
        for i := 1; i < len(A); i++ {
                if res > A[i] {
                        res = A[i]
                        index = i
                }
        }
        return index, res
}
```

31.3 1009. 十进制整数的反码 (3)

- 题目

```
每个非负整数 N 都有其二进制表示。例如， 5 可以被表示为二进制 "101", 11 可以用二进制
↪"1011" 表示，
依此类推。注意，除 N = 0 外，任何二进制表示中都不含前导零。
二进制的反码表示是将每个 1 改为 0 且每个 0 变为 1。例如，二进制数 "101"␣
↪的二进制反码为 "010"。
给你一个十进制数 N，请你返回其二进制表示的反码所对应的十进制整数。
示例 1：输入：5 输出：2
解释：5 的二进制表示为 "101"，其二进制反码为 "010"，也就是十进制中的 2 。
示例 2：输入：7 输出：0
解释：7 的二进制表示为 "111"，其二进制反码为 "000"，也就是十进制中的 0 。
示例 3：输入：10 输出：5
解释：10 的二进制表示为 "1010"，其二进制反码为 "0101"，也就是十进制中的 5 。
提示：
    0 <= N < 10^9
    本题与 476：https://leetcode.cn/problems/number-complement/ 相同
```

- 解题思路

```
/*
101+010=1000=111+1
*/
func bitwiseComplement(N int) int {
        temp := 2
        for N >= temp {
                temp = temp << 1
        }
        return temp - 1 - N
}

#
/*
101^111=010
*/
func bitwiseComplement(N int) int {
        temp := N
        res := 1
        for temp > 1 {
                temp = temp >> 1
                res = res << 1
                res++
        }
        return res ^ N
}

#
func bitwiseComplement(N int) int {
        res := 0
        if N == 0 {
                return 1
        }
        if N == 1 {
                return 0
        }
        exp := 1
        for N > 0 {
                if N%2 == 0 {
                        res = res + exp
                }
                exp = exp * 2
                N = N / 2
        }
        return res
```

(续下页)

(接上页)

```
}
```

31.4 1010. 总持续时间可被 60 整除的歌曲 (2)

- 题目

```
在歌曲列表中，第 i 首歌曲的持续时间为 time[i] 秒。
返回其总持续时间（以秒为单位）可被 60 整除的歌曲对的数量。
形式上，我们希望索引的数字 i 和 j 满足  i < j 且有 (time[i] + time[j]) % 60 == 0。
示例 1：输入：[30,20,150,100,40] 输出：3
解释：这三对的总持续时间可被 60 整数：
(time[0] = 30, time[2] = 150)：总持续时间 180
(time[1] = 20, time[3] = 100)：总持续时间 120
(time[1] = 20, time[4] = 40)：总持续时间 60
示例 2：输入：[60,60,60] 输出：3
解释：所有三对的总持续时间都是 120，可以被 60 整数。
提示：
    1 <= time.length <= 60000
    1 <= time[i] <= 500
```

- 解题思路

```
func numPairsDivisibleBy60(time []int) int {
        m := make(map[int]int)
        for i := 0; i < len(time); i++ {
                m[time[i]%60]++
        }
        res := 0
        for key, value := range m {
                if key == 0 || key == 30 {
                        res = res + (value-1)*value/2
                } else {
                        if v, ok := m[60-key]; ok && v > 0 {
                                res = res + v*value
                                m[key] = 0
                                m[60-key] = 0
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func numPairsDivisibleBy60(time []int) int {
        res := 0
        arr := make([]int,60)
        for i := range time{
                if time[i] % 60 == 0{
                        res = res + arr[0]
                }else {
                        res = res + arr[60-time[i]%60]
                }
                arr[time[i]%60]++
        }
        return res
}
```

31.5 1013. 将数组分成和相等的三个部分 (2)

- 题目

```
给你一个整数数组 A，只有可以将其划分为三个和相等的非空部分时才返回 true，否则返回
↪false。
形式上，如果可以找出索引 i+1 < j 且满足 (A[0] + A[1] + ... + A[i]
== A[i+1] + A[i+2] + ... + A[j-1]
== A[j] + A[j-1] + ... + A[A.length - 1]) 就可以将数组三等分。
示例 1：输入：[0,2,1,-6,6,-7,9,1,2,0,1] 输出：true
解释：0 + 2 + 1 = -6 + 6 - 7 + 9 + 1 = 2 + 0 + 1
示例 2：输入：[0,2,1,-6,6,7,9,-1,2,0,1] 输出：false
示例 3：输入：[3,3,6,5,-2,2,5,1,-9,4] 输出：true
解释：3 + 3 = 6 = 5 - 2 + 2 + 5 + 1 - 9 + 4

提示：
    3 <= A.length <= 50000
    -10^4 <= A[i] <= 10^4
```

- 解题思路

```
func canThreePartsEqualSum(A []int) bool {
        length := len(A)
        if length < 3 {
                return false
        }
        sum := 0
```

(续下页)

(接上页)

```
        for i := 0; i < length; i++ {
                sum = sum + A[i]
        }
        if sum%3 != 0 {
                return false
        }
        target := sum / 3
        count := 0
        temp := 0
        for i := 0; i < len(A); i++ {
                temp = temp + A[i]
                if temp == target {
                        temp = 0
                        count++
                }
        }
        if count >= 3 {
                return true
        }
        return false
}

#
func canThreePartsEqualSum(A []int) bool {
        length := len(A)
        if length < 3 {
                return false
        }
        sum := 0
        for i := 0; i < length; i++ {
                sum = sum + A[i]
        }
        if sum%3 != 0 {
                return false
        }
        target := sum / 3
        left, right := 1, len(A)-2
        leftValue, rightValue := A[0], A[len(A)-1]
        for left < right {
                for left < right && leftValue != target {
                        leftValue = leftValue + A[left]
                        left++
                }
```

(续下页)

(接上页)

```
                for left < right && rightValue != target {
                        rightValue = rightValue + A[right]
                        right--
                }
                if leftValue == target && rightValue == target {
                        return true
                }
        }
        return false
}
```

31.6 1018. 可被 5 整除的二进制前缀 (1)

- 题目

```
给定由若干 0 和 1 组成的数组 A。我们定义 N_i:
从 A[0] 到 A[i] 的第 i 个子数组被解释为一个二进制数（从最高有效位到最低有效位）。
返回布尔值列表 answer，只有当 N_i 可以被 5 整除时，答案 answer[i] 为 true，否则为␣
↪false。
示例 1：输入：[0,1,1] 输出：[true,false,false]
解释：
输入数字为 0, 01, 011；也就是十进制中的 0, 1, 3 。只有第一个数可以被 5 整除，因此␣
↪answer[0] 为真。
示例 2：输入：[1,1,1] 输出：[false,false,false]
示例 3：输入：[0,1,1,1,1,1] 输出：[true,false,false,false,true,false]
示例 4：输入：[1,1,1,0,1] 输出：[false,false,false,false,false]
提示：
    1 <= A.length <= 30000
    A[i] 为 0 或 1
```

- 解题思路

```
func prefixesDivBy5(A []int) []bool {
        res := make([]bool, len(A))
        temp := 0
        for i := 0; i < len(A); i++ {
                temp = (temp*2 + A[i]) % 5
                if temp == 0 {
                        res[i] = true
                }
        }
        return res
```

(续下页)

(接上页)

```
}
```

31.7 1021. 删除最外层的括号 (3)

- 题目

```
有效括号字符串为空 ("")、"(" + A + ")" 或 A + B，其中 A 和 B 都是有效的括号字符串，+代表字符串的连接。
例如，""，"()"，"(())()" 和 "(()(()))" 都是有效的括号字符串。
如果有效字符串 S 非空，且不存在将其拆分为 S = A+B的方法，我们称其为原语 (primitive)，
其中 A 和 B 都是非空有效括号字符串。
给出一个非空有效字符串 S，考虑将其进行原语化分解，
使得：S = P_1 + P_2 + ... + P_k，其中 P_i 是有效括号字符串原语。
对 S 进行原语化分解，删除分解中每个原语字符串的最外层括号，返回 S 。
示例 1：输入："(()())(())" 输出："()()()"
解释：
输入字符串为 "(()())(())"，原语化分解得到 "(()())" + "(())"，
删除每个部分中的最外层括号后得到 "()()" + "()" = "()()()"。

示例 2：输入："(()())(())(()(()))" 输出："()()()()(())"
解释：
输入字符串为 "(()())(())(()(()))"，原语化分解得到 "(()())" + "(())" + "(()(()))"，
删除每个部分中的最外层括号后得到 "()()" + "()" + "()(())" = "()()()()(())"。
示例 3：输入："()()" 输出：""
解释：  输入字符串为 "()()"，原语化分解得到 "()" + "()"，
删除每个部分中的最外层括号后得到 "" + "" = ""。
提示：
    S.length <= 10000
    S[i] 为 "(" 或 ")"
    S 是一个有效括号字符串
```

- 解题思路

```
func removeOuterParentheses(S string) string {
        if len(S) == 0 {
                return ""
        }
        res := ""
        stack := make([]byte, 0)
        stack = append(stack, S[0])
        last := 0
```

(续下页)

(接上页)

```
        for i := 1; i < len(S); i++ {
                if len(stack) > 0 && S[i] == ')' && stack[len(stack)-1] == '(' {
                        stack = stack[:len(stack)-1]
                        if len(stack) == 0 {
                                res = res + S[last+1:i]
                                last = i + 1
                        }
                } else {
                        stack = append(stack, S[i])
                }
        }
        return res
}

#
func removeOuterParentheses(S string) string {
        res := ""
        count := 0
        last := 0
        for i := 0; i < len(S); i++ {
                if S[i] == '(' {
                        count++
                } else {
                        count--
                }
                if count == 1 && S[i] == '(' {
                        last = i
                }
                if count == 0 {
                        res = res + S[last+1:i]
                }
        }
        return res
}

#
func removeOuterParentheses(S string) string {
        if len(S) == 0 {
                return ""
        }
        res := ""
        stack := make([]byte, 0)
        for i := 0; i < len(S); i++ {
```

(续下页)

(接上页)

```
                if S[i] == ')' {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) > 0 {
                        res = res + string(S[i])
                }
                if S[i] == '(' {
                        stack = append(stack, S[i])
                }
        }
        return res
}
```

31.8 1022. 从根到叶的二进制数之和 (2)

- 题目

```
给出一棵二叉树，其上每个结点的值都是 0 或 1␣
↪。每一条从根到叶的路径都代表一个从最高有效位开始的二进制数。
例如，如果路径为 0 -> 1 -> 1 -> 0 -> 1，那么它表示二进制数 01101，也就是 13 。
对树上的每一片叶子，我们都要找出从根到该叶子的路径所表示的数字。
以 10^9 + 7 为模，返回这些数字之和。
示例：输入：[1,0,1,0,1,0,1] 输出：22
解释：(100) + (101) + (110) + (111) = 4 + 5 + 6 + 7 = 22
提示：
    树中的结点数介于 1 和 1000 之间。
    node.val 为 0 或 1 。
```

- 解题思路

```
var res int

func sumRootToLeaf(root *TreeNode) int {
        res = 0
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, sum int) {
        if root == nil {
                return
        }
```

(续下页)

(接上页)

```
        sum = sum*2 + root.Val
        if root.Left == nil && root.Right == nil {
                res = (res + sum) % 1000000007
        }
        dfs(root.Left, sum)
        dfs(root.Right, sum)
}

#
type Node struct {
        node *TreeNode
        sum  int
}

func sumRootToLeaf(root *TreeNode) int {
        res := 0
        stack := make([]Node, 0)
        stack = append(stack, Node{
                node: root,
                sum:  0,
        })
        for len(stack) > 0 {
                node, sum := stack[len(stack)-1].node, stack[len(stack)-1].sum
                stack = stack[:len(stack)-1]
                sum = sum*2 + node.Val
                if node.Left == nil && node.Right == nil {
                        res = (res + sum) % 1000000007
                }
                if node.Left != nil {
                        stack = append(stack, Node{
                                node: node.Left,
                                sum:  sum,
                        })
                }
                if node.Right != nil {
                        stack = append(stack, Node{
                                node: node.Right,
                                sum:  sum,
                        })
                }
        }
        return res
}
```

31.9 1025. 除数博弈 (2)

- 题目

```
爱丽丝和鲍勃一起玩游戏，他们轮流行动。爱丽丝先手开局。
最初，黑板上有一个数字 N 。在每个玩家的回合，玩家需要执行以下操作：
    选出任一 x，满足 0 < x < N 且 N % x == 0 。
    用 N - x 替换黑板上的数字 N 。
如果玩家无法执行这些操作，就会输掉游戏。
只有在爱丽丝在游戏中取得胜利时才返回 True，否则返回␣
↪false。假设两个玩家都以最佳状态参与游戏。
示例 1：输入：2 输出：true
解释：爱丽丝选择 1，鲍勃无法进行操作。
示例 2：输入：3 输出：false
解释：爱丽丝选择 1，鲍勃也选择 1，然后爱丽丝无法进行操作。
提示：
    1 <= N <= 1000
```

- 解题思路

```
func divisorGame(N int) bool {
        return N % 2 == 0
}

#
func divisorGame(N int) bool {
        dp := make([]bool, N+1)
        dp[1] = false // 1的时候爱丽丝没有选择，失败
        for i := 2; i <= N; i++ {
                for j := 1; j < i; j++ {
                        if i%j == 0 && dp[i-j] == false {
                                dp[i] = true
                        }
                }
        }
        return dp[N]
}
```

31.10 1029. 两地调度 (2)

- 题目

```
公司计划面试 2N 人。第 i 人飞往 A 市的费用为 costs[i][0]，飞往 B 市的费用为␣
↪costs[i][1]。
返回将每个人都飞到某座城市的最低费用，要求每个城市都有 N 人抵达。
示例：输入：[[10,20],[30,200],[400,50],[30,20]] 输出：110
解释：
第一个人去 A 市，费用为 10。
第二个人去 A 市，费用为 30。
第三个人去 B 市，费用为 50。
第四个人去 B 市，费用为 20。
最低总费用为 10 + 30 + 50 + 20 = 110，每个城市都有一半的人在面试。
提示：
    1 <= costs.length <= 100
    costs.length 为偶数
    1 <= costs[i][0], costs[i][1] <= 1000
```

- 解题思路

```
func twoCitySchedCost(costs [][]int) int {
        sort.Slice(costs, func(i, j int) bool {
                return costs[i][0]-costs[i][1] < costs[j][0]-costs[j][1]
        })
        res := 0
        for i := 0; i < len(costs); i++ {
                if i < len(costs)/2 {
                        res = res + costs[i][0]
                } else {
                        res = res + costs[i][1]
                }
        }
        return res
}

#
func twoCitySchedCost(costs [][]int) int {
        n := len(costs)
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
                for j := i + 1; j <= n; j++ {
                        // dp[i][j]表示i个人飞往A市的次数为j的最低费用
```

(续下页)

(接上页)

```
                        // 无效掉j>i的情况，比如i=3, j=4
                        // 因为不存在3个人飞往A市次数为4次的情况
                        dp[i][j] = 100000000
                }
        }
        for i := 1; i <= n; i++ {
                dp[i][0] = dp[i-1][0] + costs[i-1][1]
                for j := 1; j <= i; j++ {
                        // dp[i][j]表示i个人飞往A市的次数为j的最低费用
                        // 其中i-1个人飞往A市的次数为j+当前飞往B市的费用
                        // 其中i-1个人飞往A市的次数为j-1+当前飞往A市的费用
                        dp[i][j] = min(dp[i-1][j]+costs[i-1][1], dp[i-1][j-1]+costs[i-
↪1][0])
                }
        }
        return dp[n][n/2]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

31.11 1030. 距离顺序排列矩阵单元格 (3)

- 题目

```
给出 R 行 C 列的矩阵，其中的单元格的整数坐标为 (r, c)，满足 0 <= r < R 且 0 <= c < C。
另外，我们在该矩阵中给出了一个坐标为 (r0, c0) 的单元格。
返回矩阵中的所有单元格的坐标，并按到 (r0, c0) 的距离从最小到最大的顺序排，
其中，两单元格(r1, c1) 和 (r2, c2) 之间的距离是曼哈顿距离，|r1 - r2| + |c1 - c2|。
（你可以按任何满足此条件的顺序返回答案。）
示例 1：输入：R = 1, C = 2, r0 = 0, c0 = 0 输出：[[0,0],[0,1]]
解释：从 (r0, c0) 到其他单元格的距离为：[0,1]
示例 2：输入：R = 2, C = 2, r0 = 0, c0 = 1 输出：[[0,1],[0,0],[1,1],[1,0]]
解释：从 (r0, c0) 到其他单元格的距离为：[0,1,1,2]
[[0,1],[1,1],[0,0],[1,0]] 也会被视作正确答案。
示例 3：输入：R = 2, C = 3, r0 = 1, c0 = 2 输出：[[1,2],[0,2],[1,1],[0,1],[1,0],[0,0]]
解释：从 (r0, c0) 到其他单元格的距离为：[0,1,1,2,2,3]
其他满足题目要求的答案也会被视为正确，例如 [[1,2],[1,1],[0,2],[1,0],[0,1],[0,0]]。
```

(续下页)

(接上页)

```
提示：
    1 <= R <= 100
    1 <= C <= 100
    0 <= r0 < R
    0 <= c0 < C
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func allCellsDistOrder(R int, C int, r0 int, c0 int) [][]int {
        res := make([][]int, 0)
        visited := make([][]bool, R)
        for i := 0; i < R; i++ {
                visited[i] = make([]bool, C)
        }
        list := make([][]int, 0)
        list = append(list, []int{r0, c0})
        visited[r0][c0] = true
        for len(list) > 0 {
                x1, y1 := list[0][0], list[0][1]
                res = append(res, []int{x1, y1})
                list = list[1:]
                for i := 0; i < 4; i++ {
                        x := x1 + dx[i]
                        y := y1 + dy[i]
                        if (0 <= x && x < R && 0 <= y && y < C) && visited[x][y] ==␣
↪false {
                                visited[x][y] = true
                                list = append(list, []int{x, y})
                        }
                }
        }
        return res
}

#
func allCellsDistOrder(R int, C int, r0 int, c0 int) [][]int {
        res := make([][]int, 0)
        for i := 0; i < R; i++ {
                for j := 0; j < C; j++ {
                        res = append(res, []int{i, j})
                }
```

(续下页)

(接上页)

```
        }
        sort.Slice(res, func(i, j int) bool {
                return abs(res[i][0], r0)+abs(res[i][1], c0) <
                        abs(res[j][0], r0)+abs(res[j][1], c0)
        })
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func allCellsDistOrder(R int, C int, r0 int, c0 int) [][]int {
        res := make([][]int, 0)
        m := make(map[int][][]int)
        max := 0
        for i := 0; i < R; i++ {
                for j := 0; j < C; j++ {
                        length := abs(i, r0) + abs(j, c0)
                        m[length] = append(m[length], []int{i, j})
                        if length > max {
                                max = length
                        }
                }
        }
        for i := 0; i <= max; i++ {
                for j := 0; j < len(m[i]); j++ {
                        res = append(res, m[i][j])
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

31.12 1033. 移动石子直到连续 (2)

- 题目

```
三枚石子放置在数轴上，位置分别为 a，b，c。
每一回合，我们假设这三枚石子当前分别位于位置 x, y, z 且 x < y < z。
从位置 x 或者是位置 z 拿起一枚石子，并将该石子移动到某一整数位置 k 处，其中 x < k < z␣
↪且 k != y。
当你无法进行任何移动时，即，这些石子的位置连续时，游戏结束。
要使游戏结束，你可以执行的最小和最大移动次数分别是多少？ 以长度为 2␣
↪的数组形式返回答案：
answer = [minimum_moves, maximum_moves]
示例 1：输入：a = 1, b = 2, c = 5 输出：[1, 2]
解释：将石子从 5 移动到 4 再移动到 3，或者我们可以直接将石子移动到 3。
示例 2：输入：a = 4, b = 3, c = 2 输出：[0, 0]
解释：我们无法进行任何移动。
提示：
    1 <= a <= 100
    1 <= b <= 100
    1 <= c <= 100
    a != b, b != c, c != a
```

- 解题思路

```
func numMovesStones(a int, b int, c int) []int {
        arr := []int{a, b, c}
        sort.Ints(arr)
        a, b, c = arr[0], arr[1], arr[2]
        if a < b && b < c {
                if b-a == 1 && c-b == 1 {
                        return []int{0, 0}
                } else if b-a > 2 && c-b > 2 {
                        return []int{2, c - a - 2}
                } else {
                        return []int{1, c - a - 2}
                }
        }
        return []int{0, 0}
}

#
func numMovesStones(a int, b int, c int) []int {
        if a > b {
                a, b = b, a
```

(续下页)

(接上页)

```
        }
        if b > c {
                b, c = c, b
        }
        if a > b {
                a, b = b, a
        }
        if a < b && b < c {
                if b-a == 1 && c-b == 1 {
                        return []int{0, 0}
                } else if b-a > 2 && c-b > 2 {
                        return []int{2, c - a - 2}
                } else {
                        return []int{1, c - a - 2}
                }
        }
        return []int{0, 0}
}
```

31.13 1037. 有效的回旋镖 (3)

- 题目

```
回旋镖定义为一组三个点，这些点各不相同且不在一条直线上。
给出平面上三个点组成的列表，判断这些点是否可以构成回旋镖。
示例 1：输入：[[1,1],[2,3],[3,2]] 输出：true
示例 2：输入：[[1,1],[2,2],[3,3]] 输出：false
提示：
    points.length == 3
    points[i].length == 2
    0 <= points[i][j] <= 100
```

- 解题思路

```
// k1=(y1-y0)/(x1-x0) = k2 = (y2-y1)/(x2-x1)
// (x1-x0)*(y2-y1) = (x2-x1)*(y1-y0)
func isBoomerang(points [][]int) bool {
        return (points[1][0]-points[0][0])*(points[2][1]-points[1][1]) !=
                (points[2][0]-points[1][0])*(points[1][1]-points[0][1])
}

#
```

(续下页)

(接上页)

```
// 鞋带公式
// S=|(x1 * y2 + x2 * y3 + x3 * y1 - y1 * x2 - y2 * x3 - y3 * x1)|/2
// S!=0组成三角形
func isBoomerang(points [][]int) bool {
        return␣
↪points[0][0]*points[1][1]+points[1][0]*points[2][1]+points[2][0]*points[0][1] !=
                points[0][1]*points[1][0]+points[1][1]*points[2][0]+points[2][1]*points[0][0]
}

#
func isBoomerang(points [][]int) bool {
        side1 := side(points[0], points[1])
        side2 := side(points[1], points[2])
        side3 := side(points[0], points[2])
        return side1+side2 > side3 &&
                side2+side3 > side1 &&
                side1+side3 > side2
}

func side(arr1, arr2 []int) float64 {
        res := (arr1[0]-arr2[0])*(arr1[0]-arr2[0]) +
                (arr1[1]-arr2[1])*(arr1[1]-arr2[1])
        return math.Sqrt(float64(res))
}
```

31.14 1039. 多边形三角剖分的最低得分 (3)

- 题目

```
给定N，想象一个凸N边多边形，其顶点按顺时针顺序依次标记为A[0], A[i], ..., A[N-1]。
假设您将多边形剖分为 N-2 个三角形。对于每个三角形，该三角形的值是顶点标记的乘积，
三角剖分的分数是进行三角剖分后所有 N-2 个三角形的值之和。
返回多边形进行三角剖分后可以得到的最低分。
示例 1：输入：[1,2,3] 输出：6
解释：多边形已经三角化，唯一三角形的分数为 6。
示例 2：输入：[3,7,4,5] 输出：144
解释：有两种三角剖分，可能得分分别为：3*7*5 + 4*5*7 = 245，或 3*4*5 + 3*4*7 =␣
↪144。最低分数为 144。
示例 3：输入：[1,3,1,4,1,5] 输出：13
解释：最低分数三角剖分的得分情况为 1*1*3 + 1*1*4 + 1*1*5 + 1*1*1 = 13。
提示：3 <= A.length <= 50
1 <= A[i] <= 100
```

- 解题思路

```
func minScoreTriangulation(values []int) int {
        n := len(values)
        dp := make([][]int, n) // dp[i][j]表示从i到j序列的最低分
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        // 点0和n-1，因为总有某个点j与点0和n-1构成三角形
        // 这样就把1个多边形，转换为2个或者3个区域
        // dp[0][n-1] = min(dp[0][j] + dp[j][n-1] + A[0]*A[k]*A[n-1])
        for i := n - 3; i >= 0; i-- {
                for j := i + 2; j < n; j++ {
                        for k := i + 1; k < j; k++ {
                                sum := dp[i][k] + dp[k][j] +␣
↪values[i]*values[k]*values[j]
                                if dp[i][j] == 0 {
                                        dp[i][j] = sum
                                } else {
                                        dp[i][j] = min(dp[i][j], sum)
                                }
                        }
                }
        }
        return dp[0][n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minScoreTriangulation(values []int) int {
        n := len(values)
        dp := make([][]int, n) // dp[i][j]表示从i到j序列的最低分
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        // 点0和n-1，因为总有某个点j与点0和n-1构成三角形
        // 这样就把1个多边形，转换为2个或者3个区域
        // dp[0][n-1] = min(dp[0][j] + dp[j][n-1] + A[0]*A[k]*A[n-1])
        for j := 2; j < n; j++ {
```

(续下页)

(接上页)

```
                for i := j - 2; i >= 0; i-- {
                        for k := i + 1; k < j; k++ {
                                sum := dp[i][k] + dp[k][j] +␣
↪values[i]*values[k]*values[j]
                                if dp[i][j] == 0 {
                                        dp[i][j] = sum
                                } else {
                                        dp[i][j] = min(dp[i][j], sum)
                                }
                        }
                }
        }
        return dp[0][n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var dp [][]int

func minScoreTriangulation(values []int) int {
        n := len(values)
        dp = make([][]int, n) // dp[i][j]表示从i到j序列的最低分
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        return dfs(values, 0, n-1)
}

func dfs(values []int, i, j int) int {
        if i+1 == j {
                return 0
        }
        if dp[i][j] > 0 {
                return dp[i][j]
        }
        res := math.MaxInt32
        for k := i + 1; k < j; k++ {
```

(续下页)

(接上页)

```
                sum := dfs(values, i, k) + dfs(values, k, j) +␣
↪values[i]*values[k]*values[j]
                res = min(res, sum)
        }
        dp[i][j] = res
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

31.15 1042. 不邻接植花 (1)

- 题目

```
有 N 个花园，按从 1 到 N 标记。在每个花园中，你打算种下四种花之一。
paths[i] = [x, y] 描述了花园 x 到花园 y 的双向路径。
另外，没有花园有 3 条以上的路径可以进入或者离开。
你需要为每个花园选择一种花，使得通过路径相连的任何两个花园中的花的种类互不相同。
以数组形式返回选择的方案作为答案 answer，其中 answer[i] 为在第 (i+1)␣
↪个花园中种植的花的种类。
花的种类用  1, 2, 3, 4 表示。保证存在答案。
示例 1：输入：N = 3, paths = [[1,2],[2,3],[3,1]] 输出：[1,2,3]
示例 2：输入：N = 4, paths = [[1,2],[3,4]] 输出：[1,2,1,2]
示例 3：输入：N = 4, paths = [[1,2],[2,3],[3,4],[4,1],[1,3],[2,4]] 输出：[1,2,3,4]
提示：
    1 <= N <= 10000
    0 <= paths.size <= 20000
    不存在花园有 4 条或者更多路径可以进入或离开。
    保证存在答案。
```

- 解题思路

```
func gardenNoAdj(N int, paths [][]int) []int {
        res := make([]int, N+1)
        arr := make([][]int, N+1)
        for i := 0; i < len(paths); i++ {
                arr[paths[i][0]] = append(arr[paths[i][0]], paths[i][1])
```

(续下页)

(接上页)

```
                arr[paths[i][1]] = append(arr[paths[i][1]], paths[i][0])
        }
        for i := 1; i <= N; i++ {
                m := map[int]int{
                        1: 1,
                        2: 2,
                        3: 3,
                        4: 4,
                }
                for j := 0; j < len(arr[i]); j++ {
                        if res[arr[i][j]] > 0 {
                                delete(m, res[arr[i][j]])
                        }
                }
                for k := range m {
                        res[i] = k
                        break
                }
        }
        return res[1:]
}
```

31.16 1046. 最后一块石头的重量 (2)

- 题目

```
有一堆石头，每块石头的重量都是正整数。
每一回合，从中选出两块 最重的 石头，然后将它们一起粉碎。假设石头的重量分别为 x 和y，且 x <= y。
那么粉碎的可能结果如下：
    如果 x == y，那么两块石头都会被完全粉碎；
    如果 x != y，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 y-x。
最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 0。
示例：输入：[2,7,4,1,8,1] 输出：1
解释：
先选出 7 和 8，得到 1，所以数组转换为 [2,4,1,1,1]，
再选出 2 和 4，得到 2，所以数组转换为 [2,1,1,1]，
接着是 2 和 1，得到 1，所以数组转换为 [1,1,1]，
最后选出 1 和 1，得到 0，最终数组转换为 [1]，这就是最后剩下那块石头的重量。
提示：
    1 <= stones.length <= 30
    1 <= stones[i] <= 1000
```

- 解题思路

```
type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

func lastStoneWeight(stones []int) int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < len(stones); i++ {
                heap.Push(&intHeap, stones[i])
        }
        for intHeap.Len() > 1 {
                a := heap.Pop(&intHeap).(int)
                b := heap.Pop(&intHeap).(int)
                if a > b {
                        heap.Push(&intHeap, a-b)
                }
        }
        if intHeap.Len() > 0 {
                res := heap.Pop(&intHeap).(int)
                return res
        }
        return 0
```

(续下页)

(接上页)

```
}

#
func lastStoneWeight(stones []int) int {
        length := len(stones)
        if length == 1 {
                return stones[0]
        }
        sort.Ints(stones)
        for stones[length-2] != 0 {
                stones[length-1] = stones[length-1] - stones[length-2]
                stones[length-2] = 0
                sort.Ints(stones)
        }
        return stones[length-1]
}
```

31.17 1047. 删除字符串中的所有相邻重复项 (2)

- 题目

```
给出由小写字母组成的字符串 S，重复项删除操作会选择两个相邻且相同的字母，并删除它们。
在 S 上反复执行重复项删除操作，直到无法继续删除。
在完成所有重复项删除操作后返回最终的字符串。答案保证唯一。
示例：输入："abbaca" 输出："ca"
解释：例如，在 "abbaca" 中，我们可以删除 "bb"␣
↪由于两字母相邻且相同，这是此时唯一可以执行删除操作的重复项。
之后我们得到字符串 "aaca"，其中又只有 "aa" 可以执行重复项删除操作，所以最后的字符串为
↪"ca"。
提示：
    1 <= S.length <= 20000
    S 仅由小写英文字母组成。
```

- 解题思路

```
func removeDuplicates(S string) string {
        stack := make([]int32, 0)
        for _, v := range S {
                stack = append(stack, v)
                for len(stack) > 1 && stack[len(stack)-1] == stack[len(stack)-2] {
                        stack = stack[:len(stack)-2]
                }
```

(续下页)

(接上页)

```
        }
        return string(stack)
}

#
func removeDuplicates(S string) string {
        arr := []byte(S)
        for {
                flag := false
                for i := 0; i < len(arr)-1; i++ {
                        if arr[i] == arr[i+1] {
                                if i+2 != len(arr) {
                                        arr = append(arr[:i], arr[i+2:]...)
                                } else {
                                        arr = arr[:i]
                                }
                                flag = true
                                break
                        }
                }
                if flag == false {
                        break
                }
        }
        return string(arr)
}
```

31.18 1051. 高度检查器 (2)

- 题目

```
学校在拍年度纪念照时，一般要求学生按照 非递减 的高度顺序排列。
请你返回能让所有学生以 非递减 高度排列的最小必要移动人数。
注意，当一组学生被选中时，他们之间可以以任何可能的方式重新排序，而未被选中的学生应该保持不动。
示例：输入：heights = [1,1,4,2,1,3] 输出：3
解释：当前数组：[1,1,4,2,1,3] 目标数组：[1,1,1,2,3,4]
在下标 2 处（从 0 开始计数）出现 4 vs 1 ，所以我们必须移动这名学生。
在下标 4 处（从 0 开始计数）出现 1 vs 3 ，所以我们必须移动这名学生。
在下标 5 处（从 0 开始计数）出现 3 vs 4 ，所以我们必须移动这名学生。
示例 2：输入：heights = [5,1,2,3,4] 输出：5
示例 3：输入：heights = [1,2,3,4,5] 输出：0
提示：
```

(续下页)

(接上页)

```
    1 <= heights.length <= 100
    1 <= heights[i] <= 100
```

- 解题思路

```
func heightChecker(heights []int) int {
        temp := make([]int, len(heights))
        copy(temp, heights)
        sort.Ints(temp)
        res := 0
        for i := 0; i < len(temp); i++ {
                if temp[i] != heights[i] {
                        res++
                }
        }
        return res
}

#
func heightChecker(heights []int) int {
        arr := make([]int, 101)
        for i := 0; i < len(heights); i++ {
                arr[heights[i]]++
        }
        res := 0
        j := 0
        for i := 1; i <= 100; i++ {
                for arr[i] > 0 {
                        if heights[j] != i {
                                res++
                        }
                        arr[i]--
                        j++
                }
        }
        return res
}
```

31.19 1071. 字符串的最大公因子 (2)

- 题目

```
对于字符串 S 和 T，只有在 S = T + ... + T（T 与自身连接 1 次或多次）时，我们才认定␣
↪“T 能除尽 S”。
返回最长字符串 X，要求满足 X 能除尽 str1 且 X 能除尽 str2。
示例 1：输入：str1 = "ABCABC", str2 = "ABC" 输出："ABC"
示例 2：输入：str1 = "ABABAB", str2 = "ABAB" 输出："AB"
示例 3：输入：str1 = "LEET", str2 = "CODE" 输出：""
提示：
    1 <= str1.length <= 1000
    1 <= str2.length <= 1000
    str1[i] 和 str2[i] 为大写英文字母
```

- 解题思路

```
func gcdOfStrings(str1 string, str2 string) string {
        if str1+str2 != str2+str1 {
                return ""
        }
        if str1 > str2 {
                str1, str2 = str2, str1
        }
        return str1[:gcd(len(str2), len(str1))]
}

func gcd(a, b int) int {
        if b == 0 {
                return a
        }
        return gcd(b, a%b)
}

#
func gcdOfStrings(str1 string, str2 string) string {
        min := len(str1)
        if min > len(str2) {
                min = len(str2)
        }
        for i := len(str2); i >= 1; i-- {
                if len(str1)%i == 0 && len(str2)%i == 0 && str1[:i] == str2[:i] {
                        a := strings.Repeat(str1[:i], len(str1)/i)
                        b := strings.Repeat(str2[:i], len(str2)/i)
```

(续下页)

(接上页)

```
                        if a == str1 && b == str2 {
                                return str1[:i]
                        }
                }
        }
        return ""
}
```

31.20 1078.Bigram 分词 (1)

- 题目

```
给出第一个词 first 和第二个词 second，
考虑在某些文本 text 中可能以 "first second third" 形式出现的情况，
其中 second 紧随 first 出现，third 紧随 second 出现。
对于每种这样的情况，将第三个词 "third" 添加到答案中，并返回答案。
示例 1：
输入：text = "alice is a good girl she is a good student", first = "a", second = "good
↪"
输出：["girl","student"]
示例 2：
输入：text = "we will we will rock you", first = "we", second = "will"
输出：["we","rock"]
提示：
    1 <= text.length <= 1000
    text 由一些用空格分隔的单词组成，每个单词都由小写英文字母组成
    1 <= first.length, second.length <= 10
    first 和 second 由小写英文字母组成
```

- 解题思路

```
func findOcurrences(text string, first string, second string) []string {
        arr := strings.Fields(text)
        res := make([]string, 0)
        for i := 0; i < len(arr)-2; i++ {
                if arr[i] == first && arr[i+1] == second {
                        res = append(res, arr[i+2])
                }
        }
        return res
}
```

31.21 1089. 复写零 (3)

- 题目

```
给你一个长度固定的整数数组␣
↪arr，请你将该数组中出现的每个零都复写一遍，并将其余的元素向右平移。
注意：请不要在超过该数组长度的位置写入元素。
要求：请对输入的数组 就地 进行上述修改，不要从函数返回任何东西。
示例 1：输入：[1,0,2,3,0,4,5,0] 输出：null
解释：调用函数后，输入的数组将被修改为：[1,0,0,2,3,0,0,4]
示例 2：输入：[1,2,3] 输出：null
解释：调用函数后，输入的数组将被修改为：[1,2,3]
提示：
    1 <= arr.length <= 10000
    0 <= arr[i] <= 9
```

- 解题思路

```
func duplicateZeros(arr []int) {
        count := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] == 0 {
                        count++
                }
        }
        for i := len(arr) - 1; i >= 0; i-- {
                if arr[i] == 0 {
                        count--
                        if i+count < len(arr) {
                                arr[i+count] = 0
                        }
                        if i+count+1 < len(arr) {
                                arr[i+count+1] = 0
                        }
                } else if i+count < len(arr) {
                        arr[i+count] = arr[i]
                }
        }
}

#
func duplicateZeros(arr []int) {
        for i := 0; i < len(arr); i++ {
                if arr[i] == 0 {
```

(续下页)

(接上页)

```
                        for j := len(arr) - 1; j > i; j-- {
                                arr[j] = arr[j-1]
                        }
                        i++
                }
        }
}

#
func duplicateZeros(arr []int) {
        newArr := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                if arr[i] == 0 {
                        newArr = append(newArr, 0)
                }
                newArr = append(newArr, arr[i])
        }
        for i := 0; i < len(arr); i++ {
                arr[i] = newArr[i]
        }
}
```

CHAPTER 32

1001-1100-Medium

32.1 1003. 检查替换后的词是否有效 (2)

- 题目

```
给定有效字符串"abc"。
对于任何有效的字符串 V，我们可以将 V 分成两个部分 X 和 Y，使得 X + Y（X 与 Y␣
↪连接）等于 V。
（X或 Y 可以为空。）那么，X + "abc" + Y 也同样是有效的。
例如，如果 S = "abc"，则有效字符串的示例是："abc"，"aabcbc"，"abcabc"，"abcabcababcc
↪"。
无效字符串的示例是："abccba"，"ab"，"cababc"，"bac"。
如果给定字符串 S 有效，则返回 true；否则，返回 false。
示例 1：输入："aabcbc" 输出：true
解释：从有效字符串 "abc" 开始。
然后我们可以在 "a" 和 "bc" 之间插入另一个 "abc"，产生 "a" + "abc" + "bc"，即 "aabcbc
↪"。
示例 2：输入："abcabcababcc" 输出：true
解释："abcabcabc" 是有效的，它可以视作在原串后连续插入 "abc"。
然后我们可以在最后一个字母之前插入 "abc"，产生 "abcabcab" + "abc" + "c"，即
↪"abcabcababcc"。
示例 3：输入："abccba" 输出：false
示例 4：输入："cababc" 输出：false
提示：1 <= S.length <= 20000
S[i] 为'a'、'b'、或'c'
```

- 解题思路

```
func isValid(s string) bool {
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                stack = append(stack, s[i])
                if len(stack) >= 3 && string(stack[len(stack)-3:]) == "abc" {
                        stack = stack[:len(stack)-3]
                }
        }
        return len(stack) == 0
}

# 2
func isValid(s string) bool {
        for strings.Contains(s, "abc") {
                s = strings.ReplaceAll(s, "abc", "")
        }
        return s == ""
}
```

32.2 1004. 最大连续 1 的个数 III(2)

- 题目

```
给定一个由若干 0 和 1 组成的数组 A，我们最多可以将 K 个值从 0 变成 1 。
返回仅包含 1 的最长（连续）子数组的长度。
示例 1：输入：A = [1,1,1,0,0,0,1,1,1,1,0], K = 2输出：6
解释：  [1,1,1,0,0,1,1,1,1,1,1]
粗体数字从 0 翻转到 1，最长的子数组长度为 6。
示例 2：输入：A = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], K = 3 输出：10
解释：[0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
粗体数字从 0 翻转到 1，最长的子数组长度为 10。
提示：1 <= A.length <= 20000
    0 <= K <= A.length
    A[i] 为 0 或 1
```

- 解题思路

```
func longestOnes(A []int, K int) int {
        res := 0
        left, right := 0, 0
        for right < len(A) {
```

(续下页)

(接上页)

```
                if A[right] == 1 {
                        right++
                } else {
                        if K > 0 {
                                right++
                                K--
                        } else {
                                res = max(res, right-left)
                                if A[left] == 0 {
                                        K++
                                }
                                left++
                        }
                }
        }
        res = max(res, right-left)
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestOnes(A []int, K int) int {
        res := 0
        left, right := 0, 0
        count := 0
        for right = 0; right < len(A); right++ {
                if A[right] == 0 {
                        count++
                }
                for count > K {
                        if A[left] == 0 {
                                count--
                        }
                        left++
                }
                res = max(res, right-left+1)
        }
```

(续下页)

(接上页)

```
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.3 1006. 笨阶乘 (1)

- 题目

```
通常，正整数 n 的阶乘是所有小于或等于 n 的正整数的乘积。
例如，factorial(10) = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1。
相反，我们设计了一个笨阶乘 clumsy：
在整数的递减序列中，我们以一个固定顺序的操作符序列来依次替换原有的乘法操作符：
乘法(*)，除法(/)，加法(+)和减法(-)。
例如，clumsy(10) = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1。
然而，这些运算仍然使用通常的算术运算顺序：
我们在任何加、减步骤之前执行所有的乘法和除法步骤，并且按从左到右处理乘法和除法步骤。
另外，我们使用的除法是地板除法（floor division），所以10 * 9 /␣
↪8等于11。这保证结果是一个整数。
实现上面定义的笨函数：给定一个整数 N，它返回 N 的笨阶乘。
示例 1：输入：4 输出：7
解释：7 = 4 * 3 / 2 + 1
示例 2：输入：10 输出：12
解释：12 = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1
提示：1 <= N <= 10000
-2^31 <= answer <= 2^31 - 1 （答案保证符合 32 位整数。）
```

- 解题思路

```
func clumsy(N int) int {
        res := 0
        sum := 1
        for i := N; i > 0; i = i - 4 {
                if i == 1 {
                        sum = sum * 1
                } else if i == 2 {
                        sum = sum * 2 * 1
```

(续下页)

(接上页)

```
                } else if i == 3 {
                        sum = sum * 3 * 2 / 1
                } else {
                        sum = sum*i*(i-1)/(i-2) + (i - 3)
                }
                res = res + sum
                sum = -1
        }
        return res
}
```

32.4 1007. 行相等的最少多米诺旋转 (2)

- 题目

```
在一排多米诺骨牌中，A[i] 和 B[i]分别代表第 i 个多米诺骨牌的上半部分和下半部分。
（一个多米诺是两个从 1 到 6 的数字同列平铺形成的——该平铺的每一半上都有一个数字。）
我们可以旋转第i张多米诺，使得A[i] 和B[i]的值交换。
返回能使 A 中所有值或者 B 中所有值都相同的最小旋转次数。
如果无法做到，返回-1.
示例 1：输入：A = [2,1,2,4,2,2], B = [5,2,6,2,3,2] 输出：2
解释：图一表示：在我们旋转之前， A 和 B 给出的多米诺牌。
如果我们旋转第二个和第四个多米诺骨牌，我们可以使上面一行中的每个值都等于␣
↪2，如图二所示。
示例 2：输入：A = [3,5,1,2,3], B = [3,6,3,3,4] 输出：-1
解释：在这种情况下，不可能旋转多米诺牌使一行的值相等。
提示：1 <= A[i], B[i] <= 6
2 <= A.length == B.length <= 20000
```

- 解题思路

```
func minDominoRotations(A []int, B []int) int {
        a, b := A[0], B[0]
        resA := check(A, B, a)
        resB := check(A, B, b)
        if resA > 0 && resB > 0 { // 都行选最少
                return min(resA, resB)
        }
        return max(resA, resB) // 不行选最多
}

func check(A []int, B []int, target int) int {
```

(续下页)

(接上页)

```
        a, b := 0, 0
        for i := 0; i < len(A); i++ {
                if A[i] != target && B[i] != target { // 都不满足直接返回-1
                        return -1
                } else if A[i] != target { // A[i]不满足, 需要交换
                        a++
                } else if B[i] != target { // B[i]不满足, 需要交换
                        b++
                }
        }
        return min(a, b)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minDominoRotations(A []int, B []int) int {
        var arr, arrA, arrB [7]int
        for i := 0; i < len(A); i++ {
                if A[i] == B[i] {
                        arr[A[i]]++
                } else {
                        arr[A[i]]++
                        arr[B[i]]++
                        arrA[A[i]]++
                        arrB[B[i]]++
                }
        }
        for i := 0; i < len(arr); i++ {
                if arr[i] == len(A) {
                        return min(arrA[i], arrB[i])
```

(续下页)

(接上页)

```
                }
        }
        return -1
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

32.5 1008. 前序遍历构造二叉搜索树 (2)

- 题目

```
返回与给定前序遍历preorder 相匹配的二叉搜索树（binary search tree）的根结点。
(回想一下，二叉搜索树是二叉树的一种，其每个节点都满足以下规则，
对于node.left的任何后代，值总 < node.val，而 node.right 的任何后代，值总 > node.val。
此外，前序遍历首先显示节点node 的值，然后遍历 node.left，接着遍历 node.right。)
题目保证，对于给定的测试用例，总能找到满足要求的二叉搜索树。
示例：输入：[8,5,1,7,10,12] 输出：[8,5,10,1,7,null,12]
提示：1 <= preorder.length <= 100
1 <= preorder[i]<= 10^8
preorder 中的值互不相同
```

- 解题思路

```
func bstFromPreorder(preorder []int) *TreeNode {
        var root *TreeNode
        for i := 0; i < len(preorder); i++ {
                root = insert(root, &TreeNode{
                        Val:   preorder[i],
                        Left:  nil,
                        Right: nil,
                })
        }
        return root
}

func insert(root *TreeNode, node *TreeNode) *TreeNode {
        if root == nil {
```

(续下页)

(接上页)

```
                return node
        }
        if root.Val < node.Val {
                root.Right = insert(root.Right, node)
        } else if root.Val > node.Val {
                root.Left = insert(root.Left, node)
        } else {
                root.Val = node.Val
        }
        return root
}

# 2
func bstFromPreorder(preorder []int) *TreeNode {
        length := len(preorder)
        if length == 0 {
                return nil
        }
        index := length
        for i := 1; i < length; i++ {
                if preorder[i] > preorder[0] {
                        index = i
                        break
                }
        }
        return &TreeNode{
                Val:   preorder[0],
                Left:  bstFromPreorder(preorder[1:index]),
                Right: bstFromPreorder(preorder[index:]),
        }
}
```

32.6 1011. 在 D 天内送达包裹的能力 (1)

- 题目

```
传送带上的包裹必须在 D 天内从一个港口运送到另一个港口。
传送带上的第 i 个包裹的重量为␣
↪weights[i]。每一天，我们都会按给出重量的顺序往传送带上装载包裹。
我们装载的重量不会超过船的最大运载重量。
返回能在 D 天内将传送带上的所有包裹送达的船的最低运载能力。
示例 1：输入：weights = [1,2,3,4,5,6,7,8,9,10], D = 5 输出：15
```

(续下页)

(接上页)

```
解释： 船舶最低载重 15 就能够在 5 天内送达所有包裹，如下所示：
第 1 天：1, 2, 3, 4, 5
第 2 天：6, 7
第 3 天：8
第 4 天：9
第 5 天：10
请注意，货物必须按照给定的顺序装运，因此使用载重能力为 14 的船舶并将包装分成
(2, 3, 4, 5), (1, 6, 7), (8), (9), (10) 是不允许的。
示例 2：输入：weights = [3,2,2,4,1,4], D = 3 输出：6
解释：船舶最低载重 6 就能够在 3 天内送达所有包裹，如下所示：
第 1 天：3, 2
第 2 天：2, 4
第 3 天：1, 4
示例 3：输入：weights = [1,2,3,1,1], D = 4 输出：3
解释： 第 1 天：1
第 2 天：2
第 3 天：3
第 4 天：1, 1
提示：1 <= D <= weights.length <= 50000
    1 <= weights[i] <= 500
```

- 解题思路

```
func shipWithinDays(weights []int, D int) int {
        sum := weights[0]
        maxValue := weights[0]
        for i := 1; i < len(weights); i++ {
                sum = sum + weights[i]
                if weights[i] > maxValue {
                        maxValue = weights[i]
                }
        }
        left, right := maxValue, sum
        for left < right {
                mid := left + (right-left)/2
                count := judge(weights, mid)
                if count <= D {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}
```

(续下页)

(接上页)

```
func judge(weights []int, target int) int {
        total := 0
        count := 1
        for i := 0; i < len(weights); i++ {
                if total+weights[i] <= target {
                        total = total + weights[i]
                } else {
                        count++
                        total = weights[i]
                }
        }
        return count
}
```

32.7 1014. 最佳观光组合 (1)

- 题目

```
给定正整数数组 A，A[i] 表示第 i 个观光景点的评分，并且两个景点 i 和 j 之间的距离为 j -
↪ i。
一对景点（i < j）组成的观光组合的得分为（A[i] + A[j] + i -␣
↪j）：景点的评分之和减去它们两者之间的距离。
返回一对观光景点能取得的最高分。
示例：输入：[8,1,5,2,6] 输出：11
解释：i = 0, j = 2, A[i] + A[j] + i - j = 8 + 5 + 0 - 2 = 11
提示：2 <= A.length <= 50000
    1 <= A[i] <= 1000
```

- 解题思路

```
func maxScoreSightseeingPair(A []int) int {
        res := 0
        maxValue := A[0] + 0
        // A[i]+A[j]+i-j=> max(A[i]+i)+(A[j]-j)  (i<j)
        for j := 1; j < len(A); j++ {
                res = max(res, A[j]-j+maxValue)
                maxValue = max(maxValue, A[j]+j)
        }
        return res
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.8 1015. 可被 K 整除的最小整数 (1)

- 题目

```
给定正整数K，你需要找出可以被 K 整除的、仅包含数字 1 的最小正整数 N。
返回N的长度。如果不存在这样的N，就返回 -1。
示例 1：输入：1 输出：1
解释：最小的答案是 N = 1，其长度为 1。
示例 2：输入：2 输出：-1
解释：不存在可被 2 整除的正整数 N 。
示例 3：输入：3 输出：3
解释：最小的答案是 N = 111，其长度为 3。
提示：1 <= K <= 10^5
```

- 解题思路

```
func smallestRepunitDivByK(K int) int {
        if K%2 == 0 || K%5 == 0 {
                return -1
        }
        res := 1
        num := 1
        for {
                if num%K == 0 {
                        return res
                }
                num = num % K // (n*10+1)%K = ((n%K)*10+1)%K
                num = 10*num + 1
                res++
        }
        return -1
}
```

32.9 1016. 子串能表示从 1 到 N 数字的二进制串 (2)

- 题目

```
给定一个二进制字符串S（一个仅由若干'0' 和 '1' 构成的字符串）和一个正整数N，
如果对于从 1 到 N 的每个整数 X，其二进制表示都是S 的子串，就返回 true，否则返回␣
↪false。
示例 1：输入：S = "0110", N = 3 输出：true
示例 2：输入：S = "0110", N = 4 输出：false
提示：1 <= S.length <= 1000
1 <= N <= 10^9
```

- 解题思路

```
func queryString(s string, n int) bool {
        for i := 1; i <= n; i++ {
                target := fmt.Sprintf("%b", i)
                if strings.Contains(s, target) == false {
                        return false
                }
        }
        return true
}

# 2
func queryString(s string, n int) bool {
        for i := n/2 + 1; i <= n; i++ {
                target := fmt.Sprintf("%b", i)
                if strings.Contains(s, target) == false {
                        return false
                }
        }
        return true
}
```

32.10 1017. 负二进制转换 (2)

- 题目

```
给出数字N，返回由若干"0"和"1"组成的字符串，该字符串为 N的负二进制（base -2）表示。
除非字符串就是"0"，否则返回的字符串中不能含有前导零。
示例 1：输入：2 输出："110"
解释：(-2) ^ 2 + (-2) ^ 1 = 2
```

(续下页)

(接上页)

```
示例 2: 输入: 3 输出: "111"
解释: (-2) ^ 2 + (-2) ^ 1 + (-2) ^ 0 = 3
示例 3: 输入: 4 输出: "100"
解释: (-2) ^ 2 = 4
提示: 0 <= N <= 10^9
```

- 解题思路

```
func baseNeg2(n int) string {
        res := ""
        if n == 0 {
                return "0"
        }
        for n != 0 {
                if n%2 == 0 { // 偶数
                        res = "0" + res
                        n = n / -2
                } else { // 奇数
                        res = "1" + res
                        n = (n - 1) / -2
                }
        }
        return res
}

# 2
func baseNeg2(n int) string {
        res := ""
        if n == 0 {
                return "0"
        }
        for n != 0 {
                if n%2 == 0 { // 偶数
                        res = "0" + res
                } else { // 奇数
                        res = "1" + res
                }
                // 3 = 111
                // -2做法: -1 / -2 = 0
                // 位做法: -(-1 >> 1) = 1
                n = -(n >> 1) // 除以-2
        }
        return res
}
```

32.11 1019. 链表中的下一个更大节点 (2)

- 题目

```
给出一个以头节点head作为第一个节点的链表。链表中的节点分别编号为：node_1, node_2,
↪node_3, ... 。
每个节点都可能有下一个更大值（next larger value）：对于node_i，如果其next_larger(node_
↪i)是node_j.val，那么就有j > i且node_j.val > node_i.val，
而j是可能的选项中最小的那个。如果不存在这样的j，那么下一个更大值为0。
返回整数答案数组answer，其中answer[i] = next_larger(node_{i+1})。
注意：在下面的示例中，诸如 [2,1,5]
↪这样的输入（不是输出）是链表的序列化表示，其头节点的值为2，
第二个节点值为 1，第三个节点值为5 。
示例 1：输入：[2,1,5] 输出：[5,5,0]
示例 2：输入：[2,7,4,3,5] 输出：[7,0,5,5,0]
示例 3：输入：[1,7,5,1,9,2,5,1] 输出：[7,9,9,9,0,5,0,0]
提示：对于链表中的每个节点，1 <= node.val<= 10^9
给定列表的长度在 [0, 10000]范围内
```

- 解题思路

```
func nextLargerNodes(head *ListNode) []int {
        arr := make([]int, 0)
        if head == nil {
                return arr
        }
        for head != nil {
                arr = append(arr, head.Val)
                head = head.Next
        }
        res := make([]int, len(arr))
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        if arr[i] < arr[j] {
                                res[i] = arr[j]
                                break
                        }
                }
        }
        return res
}

# 2
func nextLargerNodes(head *ListNode) []int {
```

(续下页)

(接上页)

```
        arr := make([]int, 0)
        if head == nil {
                return arr
        }
        for head != nil {
                arr = append(arr, head.Val)
                head = head.Next
        }
        res := make([]int, len(arr))
        stack := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                for len(stack) > 0 && arr[i] > arr[stack[len(stack)-1]] {
                        last := stack[len(stack)-1]
                        res[last] = arr[i]
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, i)
        }
        return res
}
```

32.12 1020. 飞地的数量 (2)

- 题目

```
给出一个二维数组 A，每个单元格为 0（代表海）或 1（代表陆地）。
移动是指在陆地上从一个地方走到另一个地方（朝四个方向之一）或离开网格的边界。
返回网格中无法在任意次数的移动中离开网格边界的陆地单元格的数量。
示例 1：输入：[[0,0,0,0],[1,0,1,0],[0,1,1,0],[0,0,0,0]] 输出：3
解释：  有三个 1 被 0 包围。一个 1 没有被包围，因为它在边界上。
示例 2：输入：[[0,1,1,0],[0,0,1,0],[0,0,1,0],[0,0,0,0]] 输出：0
解释：所有 1 都在边界上或可以到达边界。
提示：
    1 <= A.length <= 500
    1 <= A[i].length <= 500
    0 <= A[i][j] <= 1
    所有行的大小都相同
```

- 解题思路

```
func numEnclaves(A [][]int) int {
        for i := 0; i < len(A); i++ {
```

(续下页)

(接上页)

```
                dfs(A, i, 0)
                dfs(A, i, len(A[i])-1)
        }
        for i := 0; i < len(A[0]); i++ {
                dfs(A, 0, i)
                dfs(A, len(A)-1, i)
        }
        res := 0
        for i := 0; i < len(A); i++ {
                for j := 0; j < len(A[i]); j++ {
                        if A[i][j] == 1 {
                                res++
                        }
                }
        }
        return res
}

func dfs(A [][]int, i, j int) {
        if i < 0 || i >= len(A) || j < 0 || j >= len(A[i]) {
                return
        }
        if A[i][j] == 0 {
                return
        }
        A[i][j] = 0
        dfs(A, i, j+1)
        dfs(A, i, j-1)
        dfs(A, i+1, j)
        dfs(A, i-1, j)
}

# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func numEnclaves(A [][]int) int {
        queue := make([][2]int, 0)
        for i := 0; i < len(A); i++ {
                if A[i][0] == 1 {
                        A[i][0] = 0
                        queue = append(queue, [2]int{i, 0})
                }
```

(续下页)

(接上页)

```
                if A[i][len(A[i])-1] == 1 {
                        A[i][len(A[i])-1] = 0
                        queue = append(queue, [2]int{i, len(A[i]) - 1})
                }
        }
        for i := 0; i < len(A[0]); i++ {
                if A[0][i] == 1 {
                        A[0][i] = 0
                        queue = append(queue, [2]int{0, i})
                }
                if A[len(A)-1][i] == 1 {
                        A[len(A)-1][i] = 0
                        queue = append(queue, [2]int{len(A) - 1, i})
                }
        }
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                for k := 0; k < 4; k++ {
                        x := dx[k] + node[0]
                        y := dy[k] + node[1]
                        if x < 0 || x >= len(A) || y < 0 || y >= len(A[0]) {
                                continue
                        }
                        if A[x][y] == 0 {
                                continue
                        }
                        queue = append(queue, [2]int{x, y})
                        A[x][y] = 0
                }
        }
        res := 0
        for i := 0; i < len(A); i++ {
                for j := 0; j < len(A[i]); j++ {
                        if A[i][j] == 1 {
                                res++
                        }
                }
        }
        return res
}
```

32.13 1023. 驼峰式匹配 (2)

- 题目

```
如果我们可以将小写字母插入模式串pattern得到待查询项query，那么待查询项与给定模式串匹配。
（我们可以在任何位置插入每个字符，也可以插入 0 个字符。）
给定待查询列表queries，和模式串pattern，返回由布尔值组成的答案列表answer。
只有在待查项queries[i] 与模式串pattern 匹配时，answer[i]才为 true，否则为 false。
示例 1：输入：queries = ["FooBar","FooBarTest","FootBall",
"FrameBuffer","ForceFeedBack"], pattern = "FB"
输出：[true,false,true,true,false]
示例："FooBar" 可以这样生成："F" + "oo" + "B" + "ar"。
"FootBall" 可以这样生成："F" + "oot" + "B" + "all".
"FrameBuffer" 可以这样生成："F" + "rame" + "B" + "uffer".
示例 2：输入：queries = ["FooBar","FooBarTest","FootBall","FrameBuffer","ForceFeedBack
↪"],
pattern = "FoBa" 输出：[true,false,true,false,false]
解释：  "FooBar" 可以这样生成："Fo" + "o" + "Ba" + "r".
"FootBall" 可以这样生成："Fo" + "ot" + "Ba" + "ll".
示例 3：输出：queries = ["FooBar","FooBarTest","FootBall","FrameBuffer","ForceFeedBack
↪"],
pattern = "FoBaT" 输入：[false,true,false,false,false]
解释：   "FooBarTest" 可以这样生成："Fo" + "o" + "Ba" + "r" + "T" + "est".
提示：1 <= queries.length <= 100
1 <= queries[i].length <= 100
1 <= pattern.length <= 100
所有字符串都仅由大写和小写英文字母组成。
```

- 解题思路

```
func camelMatch(queries []string, pattern string) []bool {
        n := len(queries)
        res := make([]bool, n)
        for i := 0; i < n; i++ {
                str := queries[i]
                count := 0
                flag := true
                for j := 0; j < len(str); j++ {
                        if count < len(pattern) && str[j] == pattern[count] {
                                count++
                                continue
                        }
                        if 'A' <= str[j] && str[j] <= 'Z' {
                                flag = false
```

(续下页)

(接上页)

```
                                break
                        }
                }
                if flag == true && count == len(pattern) {
                        res[i] = true
                }
        }
        return res
}

# 2
func camelMatch(queries []string, pattern string) []bool {
        node := Constructor()
        node.Insert(pattern)
        res := make([]bool, len(queries))
        for i := 0; i < len(queries); i++ {
                res[i] = node.Match(queries[i])
        }
        return res
}

type Trie struct {
        next   map[int32]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int             // 次数（可以改为bool）
}

func Constructor() *Trie {
        return &Trie{
                next:   make(map[int32]*Trie),
                ending: 0,
        }
}

// 插入word
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v
                if _, ok := temp.next[value]; ok == false {
                        temp.next[value] = Constructor()
                }
                temp = temp.next[value]
        }
```

(续下页)

(接上页)

```
        temp.ending++
}

// 查找
func (this *Trie) Match(word string) bool {
        temp := this
        for _, v := range word {
                value := v
                if _, ok := temp.next[value]; ok == false {
                        if value < 'a' {
                                return false
                        }
                } else {
                        temp = temp.next[value]
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}
```

32.14 1024. 视频拼接 (2)

- 题目

```
你将会获得一系列视频片段，这些片段来自于一项持续时长为T秒的体育赛事。
这些片段可能有所重叠，也可能长度不一。
视频片段clips[i]都用区间进行表示：开始于clips[i][0]并于clips[i][1]结束。
我们甚至可以对这些片段自由地再剪辑，例如片段[0, 7]可以剪切成[0, 1] +[1, 3] + [3,␣
↪7]三部分。
我们需要将这些片段进行再剪辑，并将剪辑后的内容拼接成覆盖整个运动过程的片段（[0, T]）。
返回所需片段的最小数目，如果无法完成该任务，则返回-1 。
示例 1：输入：clips = [[0,2],[4,6],[8,10],[1,9],[1,5],[5,9]], T = 10 输出：3
解释：我们选中 [0,2], [8,10], [1,9] 这三个片段。
然后，按下面的方案重制比赛片段：
将 [1,9] 再剪辑为 [1,2] + [2,8] + [8,9] 。
现在我们手上有 [0,2] + [2,8] + [8,10]，而这些涵盖了整场比赛 [0, 10]。
示例 2：输入：clips = [[0,1],[1,2]], T = 5 输出：-1
解释：我们无法只用 [0,1] 和 [1,2] 覆盖 [0,5] 的整个过程。
示例 3：输入：clips = [[0,1],[6,8],[0,2],[5,6],[0,4],[0,3],[6,7],[1,3],[4,7],[1,4],[2,
↪5],
```

(续下页)

(接上页)

```
[2,6],[3,4],[4,5],[5,7],[6,9]], T = 9 输出：3
解释： 我们选取片段 [0,4], [4,7] 和 [6,9] 。
示例 4：输入：clips = [[0,4],[2,8]], T = 5 输出：2
解释：注意，你可能录制超过比赛结束时间的视频。
提示：1 <= clips.length <= 100
0 <= clips[i][0] <=clips[i][1] <= 100
0 <= T <= 100
```

- 解题思路

```
func videoStitching(clips [][]int, T int) int {
        dp := make([]int, T+1) //dp[i]表示将区间[0,i)覆盖所需的最少子区间的数量
        for i := 0; i <= T; i++ {
                dp[i] = math.MaxInt32 / 10
        }
        dp[0] = 0
        for i := 1; i <= T; i++ {
                for j := 0; j < len(clips); j++ {
                        a, b := clips[j][0], clips[j][1]
                        if a < i && i <= b && dp[a]+1 < dp[i] {
                                dp[i] = dp[a] + 1
                        }
                }
        }
        if dp[T] == math.MaxInt32/10 {
                return -1
        }
        return dp[T]
}

# 2
func videoStitching(clips [][]int, T int) int {
        arr := make([]int, T)
        for i := 0; i < len(clips); i++ {
                a, b := clips[i][0], clips[i][1]
                if a < T && arr[a] < b {
                        arr[a] = b // 更新当前位置能到达最远的位置
                }
        }
        last := 0
        prev := 0
        res := 0
        // 变成leetcode45.跳跃游戏II的变形
        for i := 0; i < len(arr); i++ {
```

(续下页)

(接上页)

```
                if arr[i] > last {
                        last = arr[i]
                }
                if i == last { // 无法达到目标
                        return -1
                }
                if i == prev {
                        res++
                        prev = last
                }
        }
        return res
}
```

32.15 1026. 节点与其祖先之间的最大差值 (2)

- 题目

```
给定二叉树的根节点root，找出存在于 不同 节点A 和B之间的最大值 V，
其中V = |A.val - B.val|，且A是B的祖先。
（如果 A 的任何子节点之一为 B，或者 A 的任何子节点是 B 的祖先，那么我们认为 A 是 B
↪的祖先）
示例 1：输入：root = [8,3,10,1,6,null,14,null,null,4,7,13] 输出：7
解释：  我们有大量的节点与其祖先的差值，其中一些如下：
|8 - 3| = 5
|3 - 7| = 4
|8 - 1| = 7
|10 - 13| = 3
在所有可能的差值中，最大值 7 由 |8 - 1| = 7 得出。
示例 2：输入：root = [1,null,2,null,0,3] 输出：3
提示：树中的节点数在2到5000之间。
0 <= Node.val <= 10^5
```

- 解题思路

```
var res int

func maxAncestorDiff(root *TreeNode) int {
        res = 0
        if root == nil {
                return 0
        }
```

(续下页)

(接上页)

```
        dfs(root, root.Val, root.Val)
        return res
}

func dfs(root *TreeNode, minValue, maxValue int) {
        if root == nil {
                return
        }
        if root.Val > maxValue {
                maxValue = root.Val
        }
        if root.Val < minValue {
                minValue = root.Val
        }
        if maxValue-minValue > res {
                res = maxValue - minValue
        }
        dfs(root.Left, minValue, maxValue)
        dfs(root.Right, minValue, maxValue)
}

# 2
var res int

func maxAncestorDiff(root *TreeNode) int {
        res = 0
        if root == nil {
                return 0
        }
        dfs(root, []int{})
        return res
}

func dfs(root *TreeNode, arr []int) {
        if root == nil {
                return
        }
        for i := 0; i < len(arr); i++ {
                if abs(arr[i], root.Val) > res {
                        res = abs(arr[i], root.Val)
                }
        }
        arr = append(arr, root.Val)
```

(续下页)

(接上页)

```
        dfs(root.Left, arr)
        dfs(root.Right, arr)
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

32.16 1027. 最长等差数列 (2)

- 题目

```
给定一个整数数组A，返回 A中最长等差子序列的长度。
回想一下，A的子序列是列表A[i_1], A[i_2], ..., A[i_k]
其中0 <= i_1 < i_2 < ... < i_k <= A.length - 1。
并且如果B[i+1] - B[i](0 <= i < B.length - 1) 的值都相同，那么序列B是等差的。
示例 1：输入：[3,6,9,12] 输出：4
解释：  整个数组是公差为 3 的等差数列。
示例 2：输入：[9,4,7,2,10] 输出：3
解释：最长的等差子序列是 [4,7,10]。
示例 3：输入：[20,1,15,3,10,5,8] 输出：4
解释：最长的等差子序列是 [20,15,10,5]。
提示：2 <= A.length <= 2000
0 <= A[i] <= 10000
```

- 解题思路

```
func longestArithSeqLength(A []int) int {
        n := len(A)
        if n < 3 {
                return 0
        }
        res := 0
        dp := make([]map[int]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make(map[int]int)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
```

(续下页)

(接上页)

```
                        diff := A[i] - A[j]
                        if _, ok := dp[j][diff]; ok {
                                dp[i][diff] = dp[j][diff] + 1
                        } else {
                                dp[j][diff] = 1
                                dp[i][diff] = dp[j][diff] + 1
                        }
                        if dp[i][diff] > res {
                                res = dp[i][diff]
                        }
                }
        }
        return res
}

# 2
func longestArithSeqLength(A []int) int {
        n := len(A)
        if n < 3 {
                return 0
        }
        res := 0
        dp := make([][20001]int, n)
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
                        diff := A[i] - A[j] + 10001
                        if dp[j][diff] > 0 {
                                dp[i][diff] = dp[j][diff] + 1
                        } else {
                                dp[j][diff] = 1
                                dp[i][diff] = dp[j][diff] + 1
                        }
                        if dp[i][diff] > res {
                                res = dp[i][diff]
                        }
                }
        }
        return res
}
```

32.17 1031. 两个非重叠子数组的最大和 (3)

- 题目

```
给出非负整数数组 A ，返回两个非重叠（连续）子数组中元素的最大和，子数组的长度分别为 L 和 M。
（这里需要澄清的是，长为 L 的子数组可以出现在长为 M 的子数组之前或之后。）
从形式上看，返回最大的 V，而 V = (A[i] + A[i+1] + ... + A[i+L-1]) + (A[j] + A[j+1] + ... + A[j+M-1]) 并满足下列条件之一：
0 <= i < i + L - 1 < j < j + M - 1 < A.length, 或
0 <= j < j + M - 1 < i < i + L - 1 < A.length.
示例 1：输入：A = [0,6,5,2,2,5,1,9,4], L = 1, M = 2 输出：20
解释：子数组的一种选择中，[9] 长度为 1，[6,5] 长度为 2。
示例 2：输入：A = [3,8,1,3,2,1,8,9,0], L = 3, M = 2 输出：29
解释：子数组的一种选择中，[3,8,1] 长度为 3，[8,9] 长度为 2。
示例 3：输入：A = [2,1,5,6,0,9,5,0,3,8], L = 4, M = 3 输出：31
解释：子数组的一种选择中，[5,6,0,9] 长度为 4，[0,3,8] 长度为 3。
提示：L >= 1
M >= 1
L + M <= A.length <= 1000
0 <= A[i] <= 1000
```

- 解题思路

```
func maxSumTwoNoOverlap(nums []int, firstLen int, secondLen int) int {
        res := 0
        n := len(nums)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + nums[i]
        }
        for i := firstLen; i <= n; i++ { // 枚举L的起始点
                L := arr[i] - arr[i-firstLen]
                M := 0
                for j := secondLen; j <= i-firstLen; j++ { // M在L左边的情况
                        M = max(M, arr[j]-arr[j-secondLen])
                }
                for j := n; j >= i+secondLen; j-- { // M在L右边的情况
                        M = max(M, arr[j]-arr[j-secondLen])
                }
                res = max(res, L+M)
        }
        return res
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxSumTwoNoOverlap(nums []int, firstLen int, secondLen int) int {
        res := 0
        n := len(nums)
        a, b := firstLen, secondLen
        for i := 1; i < n; i++ {
                nums[i] = nums[i] + nums[i-1] // 前缀和
        }
        res = nums[a+b-1]
        L, M := nums[a-1], nums[b-1]
        for i := a + b; i < n; i++ { //␣
↪枚举以i结尾的数组：分为左边（不固定）+右边（固定长度为a/b）
                L = max(L, nums[i-b]-nums[i-b-a])                    // L为i-
↪b结尾的数组中的L最大值
                M = max(M, nums[i-a]-nums[i-a-b])                    // M为i-
↪a结尾的数组中的M最大值
                temp := max(L+nums[i]-nums[i-b], nums[i]-nums[i-a]+M) // L+M 和␣
↪M+L的较大者
                res = max(res, temp)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxSumTwoNoOverlap(nums []int, firstLen int, secondLen int) int {
        res := 0
        n := len(nums)
        a, b := firstLen, secondLen
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                nums[i] = nums[i] + nums[i-1] // 前缀和
        }
        res = nums[a+b-1]
        L, M := nums[a-1], nums[b-1]
        for i := a; i < n-b; i++ { // 枚举L以i结尾+右边（固定长度为b）
                L = max(L, nums[i]-nums[i-a])   // L为结尾的数组中的L最大值
                temp := L + nums[i+b] - nums[i] // L+M
                res = max(res, temp)
        }
        for i := b; i < n-a; i++ { // 枚举M以i结尾+右边（固定长度为a）
                M = max(M, nums[i]-nums[i-b])   // M为i结尾的数组中的M最大值
                temp := nums[i+a] - nums[i] + M //  M+L
                res = max(res, temp)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.18 1034. 边框着色 (3)

- 题目

```
给出一个二维整数网格grid，网格中的每个值表示该位置处的网格块的颜色。
只有当两个网格块的颜色相同，而且在四个方向中任意一个方向上相邻时，它们属于同一连通分量。
连通分量的边界是指连通分量中的所有与不在分量中的正方形相邻（四个方向上）的所有正方形，
或者在网格的边界上（第一行/列或最后一行/列）的所有正方形。
给出位于(r0,␣
↪c0)的网格块和颜色color，使用指定颜色color为所给网格块的连通分量的边界进行着色，并返回最终的网格gri
↪。
示例 1：输入：grid = [[1,1],[1,2]], r0 = 0, c0 = 0, color = 3 输出：[[3, 3], [3, 2]]
示例 2：输入：grid = [[1,2,2],[2,3,2]], r0 = 0, c0 = 1, color = 3 输出：[[1, 3, 3],␣
↪[2, 3, 3]]
示例 3：输入：grid = [[1,1,1],[1,1,1],[1,1,1]], r0 = 1, c0 = 1, color = 2
输出：[[2, 2, 2], [2, 1, 2], [2, 2, 2]]
提示：1 <= grid.length <= 50
```

(续下页)

(接上页)

```
1 <= grid[0].length <= 50
1 <= grid[i][j] <= 1000
0 <= r0 < grid.length
0 <= c0 < grid[0].length
1 <= color <= 1000
```

- 解题思路

```
func colorBorder(grid [][]int, r0 int, c0 int, color int) [][]int {
        dfs(grid, r0, c0, grid[r0][c0])
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] < 0 {
                                grid[i][j] = color // 染色
                        }
                }
        }
        return grid
}

func dfs(grid [][]int, i, j int, color int) {
        if i < 0 || i >= len(grid) || j < 0 || j >= len(grid[0]) ||
                grid[i][j] != color {
                return
        }
        grid[i][j] = -color // 先标记相反，表示访问过
        dfs(grid, i+1, j, color)
        dfs(grid, i-1, j, color)
        dfs(grid, i, j+1, color)
        dfs(grid, i, j-1, color)
        if 0 < i && i < len(grid)-1 && 0 < j && j < len(grid[0])-1 &&
                color == abs(grid[i-1][j]) && color == abs(grid[i+1][j]) &&
                color == abs(grid[i][j-1]) && color == abs(grid[i][j+1]) {
                grid[i][j] = color // 访问完，内部的联通分量还原，不做改变
        }
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

(续下页)

(接上页)

```
# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func colorBorder(grid [][]int, r0 int, c0 int, color int) [][]int {
        n, m := len(grid), len(grid[0])
        targetColor := grid[r0][c0]
        visited := make([]bool, n*m)
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{r0, c0})
        visited[r0*m+c0] = true
        for len(queue) > 0 {
                a, b := queue[0][0], queue[0][1]
                queue = queue[1:]
                for i := 0; i < 4; i++ {
                        x := a + dx[i]
                        y := b + dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m {
                                if visited[x*m+y] == true { // 访问过
                                        continue
                                }
                                if grid[x][y] != targetColor { // 颜色不同，边界
                                        grid[a][b] = color
                                } else { // 颜色相同
                                        visited[x*m+y] = true
                                        queue = append(queue, [2]int{x, y})
                                }
                        } else {
                                grid[a][b] = color // 边界
                        }
                }
        }
        return grid
}

# 3
var visited []bool

func colorBorder(grid [][]int, r0 int, c0 int, color int) [][]int {
        visited = make([]bool, len(grid)*len(grid[0]))
        dfs(grid, r0, c0, grid[r0][c0], color)
        return grid
}
```

(续下页)

(接上页)

```
func dfs(grid [][]int, i, j int, targetColor int, color int) int {
        if i < 0 || i >= len(grid) || j < 0 || j >= len(grid[0]) { // 边界
                return 0
        }
        if visited[i*len(grid[0])+j] == true { // 先判断，因为修改了颜色
                return 1
        }
        if grid[i][j] != targetColor {
                return 0
        }
        visited[i*len(grid[0])+j] = true
        a := dfs(grid, i+1, j, targetColor, color)
        b := dfs(grid, i-1, j, targetColor, color)
        c := dfs(grid, i, j+1, targetColor, color)
        d := dfs(grid, i, j-1, targetColor, color)
        if a+b+c+d < 4 {
                grid[i][j] = color
        }
        return 1
}
```

32.19 1035. 不相交的线 (3)

- 题目

```
我们在两条独立的水平线上按给定的顺序写下 A 和 B 中的整数。
现在，我们可以绘制一些连接两个数字 A[i] 和 B[j] 的直线，只要 A[i] == B[j]，
且我们绘制的直线不与任何其他连线（非水平线）相交。
以这种方法绘制线条，并返回我们可以绘制的最大连线数。
示例 1：输入：A = [1,4,2], B = [1,2,4] 输出：2
解释：我们可以画出两条不交叉的线，如上图所示。
我们无法画出第三条不相交的直线，因为从 A[1]=4 到 B[2]=4 的直线将与从 A[2]=2 到 B[1]=2␣
↪的直线相交。
示例 2：输入：A = [2,5,1,2,5], B = [10,5,2,1,5,2] 输出：3
示例 3：输入：A = [1,3,7,1,7,5], B = [1,9,2,5,1] 输出：2
提示：1 <= A.length <= 500
    1 <= B.length <= 500
    1 <= A[i], B[i] <= 2000
```

- 解题思路

```
func maxUncrossedLines(A []int, B []int) int {
        n, m := len(A), len(B)
        dp := make([][]int, n+1)
        for i := 0; i < n+1; i++ {
                dp[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if A[i-1] == B[j-1] {
                                dp[i][j] = dp[i-1][j-1] + 1
                        } else {
                                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
                        }
                }
        }
        return dp[n][m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxUncrossedLines(A []int, B []int) int {
        n, m := len(A), len(B)
        prev := make([]int, m+1)
        cur := make([]int, m+1)
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if A[i-1] == B[j-1] {
                                cur[j] = prev[j-1] + 1
                        } else {
                                cur[j] = max(prev[j], cur[j-1])
                        }
                }
                copy(prev, cur)
        }
        return cur[m]
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 3
func maxUncrossedLines(A []int, B []int) int {
        n, m := len(A), len(B)
        cur := make([]int, m+1)
        for i := 1; i <= n; i++ {
                pre := cur[0]
                for j := 1; j <= m; j++ {
                        temp := cur[j]
                        if A[i-1] == B[j-1] {
                                cur[j] = pre + 1
                        } else {
                                cur[j] = max(cur[j], cur[j-1])
                        }
                        pre = temp
                }
        }
        return cur[m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.20 1038. 把二叉搜索树转换为累加树 (2)

- 题目

```
给出二叉 搜索 树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum␣
↪Tree），
使每个节点 node的新值等于原树中大于或等于node.val的值之和。
提醒一下，二叉搜索树满足下列约束条件：
节点的左子树仅包含键 小于 节点键的节点。
节点的右子树仅包含键 大于 节点键的节点。
```

(续下页)

(接上页)

```
左右子树也必须是二叉搜索树。
注意：该题目与 538:相同
示例 1：输入：[4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
输出：[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
示例 2：输入：root = [0,null,1] 输出：[1,null,1]
示例 3：输入：root = [1,0,2] 输出：[3,3,2]
示例 4：输入：root = [3,2,4,1] 输出：[7,9,4,10]
提示：树中的节点数介于 1 和 100 之间。
每个节点的值介于0 和100之间。
树中的所有值 互不相同 。
给定的树为二叉搜索树。
```

- 解题思路

```
func bstToGst(root *TreeNode) *TreeNode {
        sum := 0
        dfs(root, &sum)
        return root
}

func dfs(root *TreeNode, sum *int) {
        if root == nil {
                return
        }
        dfs(root.Right, sum)
        *sum = *sum + root.Val
        root.Val = *sum
        dfs(root.Left, sum)
}

# 2
func bstToGst(root *TreeNode) *TreeNode {
        if root == nil {
                return root
        }
        stack := make([]*TreeNode, 0)
        temp := root
        sum := 0
        for {
                if temp != nil {
                        stack = append(stack, temp)
                        temp = temp.Right
                } else if len(stack) != 0 {
                        temp = stack[len(stack)-1]
```

(续下页)

(接上页)

```
                        stack = stack[:len(stack)-1]
                        temp.Val = temp.Val + sum
                        sum = temp.Val
                        temp = temp.Left
                } else {
                        break
                }
        }
        return root
}
```

32.21 1040. 移动石子直到连续 II(1)

- 题目

```
在一个长度无限的数轴上，第 i 颗石子的位置为stones[i]。
如果一颗石子的位置最小/最大，那么该石子被称作端点石子。
每个回合，你可以将一颗端点石子拿起并移动到一个未占用的位置，使得该石子不再是一颗端点石子。
值得注意的是，如果石子像stones = [1,2,5]这样，你将无法移动位于位置 5 的端点石子，
因为无论将它移动到任何位置（例如 0 或 3），该石子都仍然会是端点石子。
当你无法进行任何移动时，即，这些石子的位置连续时，游戏结束。
要使游戏结束，你可以执行的最小和最大移动次数分别是多少？
以长度为 2 的数组形式返回答案：answer = [minimum_moves, maximum_moves] 。
示例 1：输入：[7,4,9] 输出：[1,2]
解释：我们可以移动一次，4 -> 8，游戏结束。
或者，我们可以移动两次 9 -> 5，4 -> 6，游戏结束。
示例2：输入：[6,5,4,3,10] 输出：[2,3]
解释：我们可以移动 3 -> 8，接着是 10 -> 7，游戏结束。
或者，我们可以移动 3 -> 7，4 -> 8，5 -> 9，游戏结束。
注意，我们无法进行 10 -> 2 这样的移动来结束游戏，因为这是不合要求的移动。
示例 3：输入：[100,101,104,102,103] 输出：[0,0]
提示：3 <= stones.length <= 10^4
1 <= stones[i] <= 10^9
stones[i]的值各不相同。
```

- 解题思路

```
func numMovesStonesII(stones []int) []int {
        n := len(stones)
        sort.Ints(stones)
        maxRes := 0
        // 最大移动次数：可以移动stones[0]和stones[n-1] 2种情况
```

(续下页)

(接上页)

```
        // 以移动stones[0]为例：
        //␣
↪1、移动stones[0]到stones[1]之后的空位，使得stones[1]、、stones[k]、、stones[0]连续
        //␣
↪2、然后移动stones[1]到stones[0]之后的空位，使得stones[2]、、stones[k]、、stones[0]、stones[k+1]、、
        //␣
↪3、重复上述步骤：把连续数组第1个数移动到最后一个数后面得空位上，形成新的连续数组
        length := (stones[n-1] - stones[0] + 1) - n // 最小值与最大值直接的空格数
        a := stones[1] - stones[0] - 1              // 第1次移动stones[0] 浪费的空间
        b := stones[n-1] - stones[n-2] - 1          // 第1次移动stones[n-1] 浪费的空间
        maxRes = length - min(a, b)
        // 最小移动次数：数组在范围n最多有多少数字
        minRes := maxRes
        j := 0
        for i := 0; i < n; i++ {
                for j < n-1 && stones[j+1]-stones[i]+1 <= n {
                        j++
                }
                total := j - i + 1 // 连续数字的长度
                // 特例：1, 2, 3, 4, 7 => 长度为n-1且连续
                if total == n-1 && stones[j]-stones[i]+1 == n-1 {
                        minRes = min(minRes, 2)
                } else {
                        minRes = min(minRes, n-total)
                }
        }
        return []int{minRes, maxRes}
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

32.22 1041. 困于环中的机器人 (1)

- 题目

```
在无限的平面上，机器人最初位于(0, 0)处，面朝北方。机器人可以接受下列三条指令之一：
"G"：直走 1 个单位
"L"：左转 90 度
"R"：右转 90 度
机器人按顺序执行指令instructions，并一直重复它们。
只有在平面中存在环使得机器人永远无法离开时，返回true。否则，返回 false。
示例 1：输入："GGLLGG" 输出：true
解释：机器人从 (0,0) 移动到 (0,2)，转 180 度，然后回到 (0,0)。
重复这些指令，机器人将保持在以原点为中心，2 为半径的环中进行移动。
示例 2：输入："GG" 输出：false
解释：机器人无限向北移动。
示例 3：输入："GL" 输出：true
解释：机器人按 (0, 0) -> (0, 1) -> (-1, 1) -> (-1, 0) -> (0, 0) -> ... 进行移动。
提示：1 <= instructions.length <= 100
instructions[i] 在{'G', 'L', 'R'}中
```

- 解题思路

```
func isRobotBounded(instructions string) bool {
        var dx = []int{1, 0, -1, 0}
        var dy = []int{0, 1, 0, -1}
        dir := 0
        x, y := 0, 0
        for i := 0; i < len(instructions); i++ {
                if instructions[i] == 'L' {
                        dir = (dir + 3) % 4
                } else if instructions[i] == 'R' {
                        dir = (dir + 1) % 4
                } else {
                        x = x + dx[dir]
                        y = y + dy[dir]
                }
        }
        return (x == 0 && y == 0) || dir != 0
}
```

32.23 1043. 分隔数组以得到最大和 (3)

- 题目

```
给你一个整数数组 arr，请你将该数组分隔为长度最多为 k 的一些（连续）子数组。
分隔完成后，每个子数组的中的所有值都会变为该子数组中的最大值。
返回将数组分隔变换后能够得到的元素最大和。
注意，原数组和分隔后的数组对应顺序应当一致，也就是说，你只能选择分隔数组的位置而不能调整数组中的顺序
示例 1：输入：arr = [1,15,7,9,2,5,10]，k = 3 输出：84
解释：因为 k=3 可以分隔成 [1,15,7] [9] [2,5,10]，结果为 [15,15,15,9,10,10,10]，
和为 84，是该数组所有分隔变换后元素总和最大的。
若是分隔成 [1] [15,7,9] [2,5,10]，
结果就是 [1, 15, 15, 15, 10, 10, 10] 但这种分隔方式的元素总和（76）小于上一种。
示例 2：输入：arr = [1,4,1,5,7,3,6,1,9,9,3]，k = 4 输出：83
示例 3：输入：arr = [1]，k = 1 输出：1
提示：1 <= arr.length <= 500
0 <= arr[i] <= 109
1 <= k <= arr.length
```

- 解题思路

```
func maxSumAfterPartitioning(arr []int, k int) int {
        n := len(arr)
        dp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                start := max(0, i-k)
                maxValue := math.MinInt32
                for j := i; j > start; j-- {
                        maxValue = max(maxValue, arr[j-1])
                        dp[i] = max(dp[i], dp[j-1]+maxValue*(i-j+1))
                }
        }
        return dp[n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxSumAfterPartitioning(arr []int, k int) int {
```

(续下页)

(接上页)

```
        n := len(arr)
        dp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                start := max(0, i-k)
                maxValue := math.MinInt32
                for j := i - 1; j >= start; j-- {
                        maxValue = max(maxValue, arr[j])
                        dp[i] = max(dp[i], dp[j]+maxValue*(i-j))
                }
        }
        return dp[n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxSumAfterPartitioning(arr []int, k int) int {
        n := len(arr)
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                maxValue := arr[i]
                for j := i; j >= 0 && j > i-k; j-- {
                        maxValue = max(maxValue, arr[j])
                        if j > 0 {
                                dp[i] = max(dp[i], dp[j-1]+maxValue*(i-j+1))
                        } else {
                                dp[i] = max(dp[i], maxValue*(i+1))
                        }
                }
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.24 1048. 最长字符串链 (2)

- 题目

```
给出一个单词列表，其中每个单词都由小写英文字母组成。
如果我们可以在word1的任何地方添加一个字母使其变成word2，那么我们认为word1是word2的前身。
例如，"abc"是"abac"的前身。
词链是单词[word_1, word_2, ..., word_k]组成的序列，
k >= 1，其中word_1是word_2的前身，word_2是word_3的前身，依此类推。
从给定单词列表 words 中选择单词组成词链，返回词链的最长可能长度。
示例：输入：["a","b","ba","bca","bda","bdca"] 输出：4
解释：最长单词链之一为 "a","ba","bda","bdca"。
提示：1 <= words.length <= 1000
1 <= words[i].length <= 16
words[i]仅由小写英文字母组成。
```

- 解题思路

```
func longestStrChain(words []string) int {
        sort.Slice(words, func(i, j int) bool {
                return len(words[i]) < len(words[j])
        })
        res := 0
        dp := make(map[string]int)
        for i := 0; i < len(words); i++ {
                str := words[i]
                for j := 0; j < len(words[i]); j++ {
                        target := words[i][:j] + words[i][j+1:]
                        if dp[target]+1 > dp[str] {
                                dp[str] = dp[target] + 1
                        }
                }
                if dp[str] > res {
                        res = dp[str]
                }
        }
        return res
}

# 2
func longestStrChain(words []string) int {
        sort.Slice(words, func(i, j int) bool {
                return len(words[i]) < len(words[j])
        })
```

(续下页)

(接上页)

```
        res := 0
        dp := make([]int, len(words))
        for i := 0; i < len(words); i++ {
                dp[i] = 1
                for j := 0; j < i; j++ {
                        if judge(words[i], words[j]) == true && dp[j]+1 > dp[i] {
                                dp[i] = dp[j] + 1
                        }
                }
                if dp[i] > res {
                        res = dp[i]
                }
        }
        return res
}

func judge(a, b string) bool {
        if len(a)-len(b) != 1 {
                return false
        }
        i, j := 0, 0
        for i < len(a) && j < len(b) {
                if a[i] == b[j] {
                        j++
                }
                i++
        }
        return j == len(b)
}
```

32.25 1049. 最后一块石头的重量 II(2)

- 题目

```
有一堆石头，每块石头的重量都是正整数。
每一回合，从中选出任意两块石头，然后将它们一起粉碎。假设石头的重量分别为x 和y，且x <=
↪y。
那么粉碎的可能结果如下：
如果x == y，那么两块石头都会被完全粉碎；
如果x != y，那么重量为x的石头将会完全粉碎，而重量为y的石头新重量为y-x。
最后，最多只会剩下一块石头。返回此石头最小的可能重量。如果没有石头剩下，就返回 0。
示例：输入：[2,7,4,1,8,1] 输出：1
```

(续下页)

(接上页)

```
解释：组合 2 和 4，得到 2，所以数组转化为 [2,7,1,8,1]，
组合 7 和 8，得到 1，所以数组转化为 [2,1,1,1]，
组合 2 和 1，得到 1，所以数组转化为 [1,1,1]，
组合 1 和 1，得到 0，所以数组转化为 [1]，这就是最优值。
提示：1 <= stones.length <= 30
1 <= stones[i] <= 1000
```

- 解题思路

```
func lastStoneWeightII(stones []int) int {
        sum := 0
        for i := 0; i < len(stones); i++ {
                sum = sum + stones[i]
        }
    // 求最后1个最小，把石头分2堆，求差值
        // 题目转换为0-1背包问题，容量为sum/2，能装多大体积
        target := sum / 2
        dp := make([]int, sum/2+1)
        for i := 0; i < len(stones); i++ {
                for j := target; j >= 0; j-- {
                        if j-stones[i] >= 0 {
                                dp[j] = max(dp[j], dp[j-stones[i]]+stones[i])
                        }
                }
        }
        return sum - 2*dp[sum/2]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func lastStoneWeightII(stones []int) int {
        n := len(stones)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + stones[i]
        }
    // 求最后1个最小，把石头分2堆，求差值
        // 题目转换为0-1背包问题，容量为sum/2，能装多大体积
```

(续下页)

(接上页)

```
        target := sum / 2
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, target+1)
                dp[i][0] = 0
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= target; j++ {
                        if j-stones[i-1] < 0 {
                                dp[i][j] = dp[i-1][j]
                        } else {
                                dp[i][j] = max(dp[i-1][j], dp[i-1][j-stones[i-
↪1]]+stones[i-1])
                        }
                }
        }
        return sum - 2*dp[n][sum/2]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.26 1052. 爱生气的书店老板 (1)

- 题目

```
今天，书店老板有一家店打算试营业 customers.length 分钟。
每分钟都有一些顾客 (customers[i]) 会进入书店，所有这些顾客都会在那一分钟结束后离开。
在某些时候，书店老板会生气。 如果书店老板在第 i 分钟生气，那么 grumpy[i] = 1，否则␣
↪grumpy[i] = 0。
当书店老板生气时，那一分钟的顾客就会不满意，不生气则他们是满意的。
书店老板知道一个秘密技巧，能抑制自己的情绪，可以让自己连续 X␣
↪分钟不生气，但却只能使用一次。
请你返回这一天营业下来，最多有多少客户能够感到满意的数量。
示例：输入：customers = [1,0,1,2,1,1,7,5], grumpy = [0,1,0,1,0,1,0,1], X = 3 输出：16
解释： 书店老板在最后 3 分钟保持冷静。
感到满意的最大客户数量 = 1 + 1 + 1 + 1 + 7 + 5 = 16.
提示： 1 <= X <= customers.length == grumpy.length <= 20000
```

(续下页)

(接上页)

```
    0 <= customers[i] <= 1000
    0 <= grumpy[i] <= 1
```

- 解题思路

```
func maxSatisfied(customers []int, grumpy []int, X int) int {
        n := len(customers)
        total := 0
        res := 0
        for i := 0; i < n; i++ {
                if grumpy[i] == 0 {
                        total = total + customers[i]
                }
        }
        window := 0
        for i := 0; i < X; i++ {
                if grumpy[i] == 1 {
                        window = window + customers[i]
                }
        }
        res = max(res, total+window)
        for i := X; i < n; i++ {
                window = window + customers[i]*grumpy[i] - customers[i-X]*grumpy[i-X]
                res = max(res, total+window)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.27 1053. 交换一次的先前排列 (2)

- 题目

```
给你一个正整数的数组 A（其中的元素不一定完全不同），
请你返回可在一次交换（交换两数字 A[i] 和 A[j] 的位置）后得到的、按字典序排列小于 A␣
↪的最大可能排列。
```

(续下页)

(接上页)

```
如果无法这么操作，就请返回原数组。
示例 1：输入：arr = [3,2,1] 输出：[3,1,2]
解释：交换 2 和 1
示例 2：输入：arr = [1,1,5] 输出：[1,1,5]
解释：已经是最小排列
示例 3：输入：arr = [1,9,4,6,7] 输出：[1,7,4,6,9]
解释：交换 9 和 7
示例 4：输入：arr = [3,1,1,3] 输出：[1,3,1,3]
解释：交换 1 和 3
提示：1 <= arr.length <= 104
1 <= arr[i] <= 104
```

- 解题思路

```
func prevPermOpt1(arr []int) []int {
        for i := len(arr) - 2; i >= 0; i-- {
                if arr[i] > arr[i+1] { // 找到第一个降序的位置a
                        b := -1
                        maxValue := -1
                        for j := i + 1; j < len(arr); j++ { //␣
↪往后找到小于a的最大数第1次出现位置b
                                if arr[i] > arr[j] {
                                        if arr[j] > maxValue {
                                                maxValue = arr[j]
                                                b = j
                                        }
                                }
                        }
                        if b != -1 {
                                arr[i], arr[b] = arr[b], arr[i]
                                return arr
                        }
                }
        }
        return arr
}

# 2
func prevPermOpt1(arr []int) []int {
        a := -1
        b := -1
        var i int
        for i = len(arr) - 2; i >= 0; i-- {
                if arr[i] > arr[i+1] { // 找到第一个降序的位置a
```

(续下页)

(接上页)

```
                        a = i
                        break
                }
        }
        if a == -1 {
                return arr
        }
        maxValue := -1
        for j := i + 1; j < len(arr); j++ { // 往后找到小于a的最大数第1次出现位置b
                if arr[a] > arr[j] {
                        if arr[j] > maxValue {
                                maxValue = arr[j]
                                b = j
                        }
                }
        }
        if a != -1 && b != -1 {
                arr[a], arr[b] = arr[b], arr[a]
        }
        return arr
}
```

32.28 1054. 距离相等的条形码 (2)

- 题目

```
在一个仓库里，有一排条形码，其中第 i 个条形码为barcodes[i]。
请你重新排列这些条形码，使其中两个相邻的条形码 不能 相等。
你可以返回任何满足该要求的答案，此题保证存在答案。
示例 1：输入：[1,1,1,2,2,2] 输出：[2,1,2,1,2,1]
示例 2：输入：[1,1,1,1,2,2,3,3] 输出：[1,3,1,3,2,1,2,1]
提示：1 <= barcodes.length <= 10000
1 <= barcodes[i] <= 10000
```

- 解题思路

```
func rearrangeBarcodes(barcodes []int) []int {
        m := make(map[int]int)
        for i := 0; i < len(barcodes); i++ {
                m[barcodes[i]]++
        }
        arr := make([]Node, 0)
```

(续下页)

(接上页)

```
        for k, v := range m {
                arr = append(arr, Node{
                        k,
                        v,
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].num > arr[j].num
        })
        res := make([]int, len(barcodes))
        index := 0
        // 先偶后奇
        for i := 0; i < 2; i++ {
                for j := i; j < len(barcodes); j = j + 2 {
                        if arr[index].num == 0 {
                                index++
                        }
                        res[j] = arr[index].value
                        arr[index].num--
                }
        }
        return res
}

# 2
func rearrangeBarcodes(barcodes []int) []int {
        n := len(barcodes)
        if n <= 1 {
                return barcodes
        }
        res := make([]int, 0)
        m := make(map[int]int)
        for _, v := range barcodes {
                m[v]++
        }
        nodeHeap := &Heap{}
        heap.Init(nodeHeap)
        for k, v := range m {
                heap.Push(nodeHeap, Node{
                        value: k,
                        num:   v,
                })
        }
```

(续下页)

(接上页)

```
        for nodeHeap.Len() >= 2 {
                node1 := heap.Pop(nodeHeap).(Node)
                node2 := heap.Pop(nodeHeap).(Node)
                res = append(res, node1.value, node2.value)
                node1.num--
                node2.num--
                if node1.num > 0 {
                        heap.Push(nodeHeap, node1)
                }
                if node2.num > 0 {
                        heap.Push(nodeHeap, node2)
                }
        }
        if nodeHeap.Len() > 0 {
                t := heap.Pop(nodeHeap).(Node)
                res = append(res, t.value)
        }
        return res
}

type Node struct {
        value int
        num   int
}

type Heap []Node

func (h Heap) Len() int {
        return len(h)
}

func (h Heap) Less(i, j int) bool {
        return h[i].num > h[j].num
}

func (h Heap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *Heap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}
```

(续下页)

(接上页)

```
func (h *Heap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

32.29 1072. 按列翻转得到最大值等行数 (3)

- 题目

```
给定由若干 0 和 1 组成的矩阵matrix，从中选出任意数量的列并翻转其上的每个单元格。
翻转后，单元格的值从 0 变成 1，或者从 1 变为 0 。
回经过一些翻转后，行与行之间所有值都相等的最大行数。
示例 1：输入：[[0,1],[1,1]] 输出：1
解释：不进行翻转，有 1 行所有值都相等。
示例 2：输入：[[0,1],[1,0]] 输出：2
解释：翻转第一列的值之后，这两行都由相等的值组成。
示例 3：输入：[[0,0,0],[0,0,1],[1,1,0]] 输出：2
解释：翻转前两列的值之后，后两行由相等的值组成。
提示：1 <= matrix.length <= 300
1 <= matrix[i].length <= 300
所有 matrix[i].length都相等
matrix[i][j] 为0 或1
```

- 解题思路

```
func maxEqualRowsAfterFlips(matrix [][]int) int {
        res := 0
        n := len(matrix)
        m := len(matrix[0])
        for i := 0; i < n; i++ {
                count := 0            //␣
↪统计与当前行完全一样的行或者完全不同的行的个数，划分为同一组
                arr := make([]int, m) // 翻转当前行的结果
                for j := 0; j < m; j++ {
                        arr[j] = 1 - matrix[i][j]
                }
                for k := 0; k < n; k++ {
                        if compare(matrix[k], matrix[i]) || compare(matrix[k], arr) {
                                count++
                        }
                }
```

(续下页)

(接上页)

```
                res = max(res, count) //␣
↪翻转最大一组的任意一行中的所有0或者1所在列即可
        }
        return res
}

func compare(a, b []int) bool {
        for i := 0; i < len(a); i++ {
                if a[i] != b[i] {
                        return false
                }
        }
        return true
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxEqualRowsAfterFlips(matrix [][]int) int {
        res := 0
        n := len(matrix)
        m := len(matrix[0])
        M := make(map[string]int)
        for i := 0; i < n; i++ {
                a := make([]byte, 0)
                b := make([]byte, 0)
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 0 {
                                a = append(a, '0')
                                b = append(b, '1')
                        } else {
                                a = append(a, '1')
                                b = append(b, '0')
                        }
                }
                M[string(a)]++
                M[string(b)]++
        }
```

(续下页)

(接上页)

```
        for _, v := range M {
                res = max(res, v)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxEqualRowsAfterFlips(matrix [][]int) int {
        res := 0
        n := len(matrix)
        m := len(matrix[0])
        M := make(map[string]int)
        for i := 0; i < n; i++ {
                a := make([]byte, 0)
                if matrix[i][0] == 0 {
                        for j := 0; j < m; j++ {
                                if matrix[i][j] == 0 {
                                        a = append(a, '0')
                                } else {
                                        a = append(a, '1')
                                }
                        }
                } else {
                        for j := 0; j < m; j++ {
                                if matrix[i][j] == 0 {
                                        a = append(a, '1')
                                } else {
                                        a = append(a, '0')
                                }
                        }
                }
                M[string(a)]++
        }
        for _, v := range M {
                res = max(res, v)
        }
```

(续下页)

(接上页)

```
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

32.30 1073. 负二进制数相加 (1)

- 题目

```
给出基数为 -2的两个数arr1 和arr2，返回两数相加的结果。
数字以数组形式给出：数组由若干 0 和 1 组成，按最高有效位到最低有效位的顺序排列。
例如，arr= [1,1,0,1]表示数字(-2)^3+ (-2)^2 + (-2)^0 = -3。
数组形式的数字也同样不含前导零：以 arr 为例，这意味着要么arr == [0]，要么arr[0] == 1。
返回相同表示形式的 arr1 和 arr2 相加的结果。两数的表示形式为：不含前导零、由若干 0 和␣
↪1 组成的数组。
示例：输入：arr1 = [1,1,1,1,1], arr2 = [1,0,1] 输出：[1,0,0,0,0]
解释：arr1 表示 11，arr2 表示 5，输出表示 16 。
提示：1 <= arr1.length <= 1000
1 <= arr2.length <= 1000
arr1 和arr2都不含前导零
arr1[i] 为0或1
arr2[i]为0 或1
```

- 解题思路

```
func addNegabinary(arr1 []int, arr2 []int) []int {
        res := make([]int, 1005)
        last := 1004
        i := len(arr1) - 1
        j := len(arr2) - 1
        carry := 0
        for i >= 0 || j >= 0 || carry != 0 {
                if i >= 0 {
                        carry = carry + arr1[i]
                        i--
                }
                if j >= 0 {
```

(续下页)

(接上页)

```
                        carry = carry + arr2[j]
                        j--
                }
                // 进位处理:
                // 进位可能: -1 (0+0-1) 、0、1、2、3 (1+1+1)
                // 进位计算: -1 => 1; 0、1 => 0; 2、3=>-1
                res[last] = abs(carry) % 2
                if carry >= 0 {
                        carry = -carry / 2
                } else {
                        carry = 1
                }
                last--
        }
        for last < len(res)-1 && res[last] == 0 { // 消除多余的0, 最后1个0不消除
                last++
        }
        return res[last:]
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

32.31 1079. 活字印刷 (1)

- 题目

```
你有一套活字字模tiles, 其中每个字模上都刻有一个字母tiles[i]。返回你可以印出的非空字母序列的数目。
注意: 本题中, 每个活字字模只能使用一次。
示例 1: 输入: "AAB" 输出: 8
解释: 可能的序列为 "A", "B", "AA", "AB", "BA", "AAB", "ABA", "BAA"。
示例 2: 输入: "AAABBC" 输出: 188
提示: 1 <= tiles.length <= 7
tiles 由大写英文字母组成
```

- 解题思路

```
var res [][]byte

func numTilePossibilities(tiles string) int {
        res = make([][]byte, 0)
        arr := []byte(tiles)
        sort.Slice(arr, func(i, j int) bool {
                return arr[i] < arr[j]
        })
        dfs(arr, 0, make([]int, len(arr)), make([]byte, 0))
        return len(res)
}

func dfs(nums []byte, index int, visited []int, arr []byte) {
        if len(arr) > 0 {
                temp := make([]byte, len(arr))
                copy(temp, arr)
                res = append(res, temp)
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == 1 {
                        continue
                }
                if i > 0 && nums[i] == nums[i-1] && visited[i-1] == 0 {
                        continue
                }
                arr = append(arr, nums[i])
                visited[i] = 1
                dfs(nums, index+1, visited, arr)
                visited[i] = 0
                arr = arr[:len(arr)-1]
        }
}
```

32.32 1080. 根到叶路径上的不足节点 (2)

- 题目

```
给定一棵二叉树的根 root，请你考虑它所有从根到叶的路径：从根到任何叶的路径。
（所谓一个叶子节点，就是一个没有子节点的节点）
假如通过节点 node 的每种可能的 “根-叶” 路径上值的总和全都小于给定的␣
↪limit，则该节点被称之为「不足节点」，需要被删除。
请你删除所有不足节点，并返回生成的二叉树的根。
```

(续下页)

(接上页)

```
示例 1：输入：root = [1,2,3,4,-99,-99,7,8,9,-99,-99,12,13,-99,14], limit = 1
输出：[1,2,3,4,null,null,7,8,9,null,14]
示例 2：输入：root = [5,4,8,11,null,17,4,7,1,null,null,5,3], limit = 22
输出：[5,4,8,11,null,17,4,7,null,null,null,5]
示例 3：输入：root = [5,-6,-6], limit = 0 输出：[]
提示：给定的树有1到5000个节点
-10^5<= node.val <= 10^5
-10^9 <= limit<= 10^9
```

- 解题思路

```
func sufficientSubset(root *TreeNode, limit int) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Left == nil && root.Right == nil {
                if root.Val < limit { // 需要删除
                        return nil
                }
                return root
        }
        left := sufficientSubset(root.Left, limit-root.Val)
        right := sufficientSubset(root.Right, limit-root.Val)
        if left == nil && right == nil { // 都为nil直接返回
                return nil
        }
        root.Left = left
        root.Right = right
        return root
}

# 2
func sufficientSubset(root *TreeNode, limit int) *TreeNode {
        return dfs(root, limit, 0)
}

func dfs(root *TreeNode, limit, sum int) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Left == nil && root.Right == nil {
                if root.Val+sum < limit { // 需要删除
                        return nil
                }
```

(续下页)

(接上页)

```
                return root
        }
        left := dfs(root.Left, limit, sum+root.Val)
        right := dfs(root.Right, limit, sum+root.Val)
        if left == nil && right == nil { // 都为nil直接返回
                return nil
        }
        root.Left = left
        root.Right = right
        return root
}
```

32.33 1081. 不同字符的最小子序列 (2)

- 题目

```
返回字符串 text 中按字典序排列最小的子序列，该子序列包含 text 中所有不同字符一次。
示例 1：输入："cdadabcc" 输出："adbc"
示例 2：输入："abcd" 输出："abcd"
示例 3：输入："ecbacba" 输出："eacb"
示例 4：输入："leetcode" 输出："letcod"
提示：
    1 <= text.length <= 1000
    text 由小写英文字母组成
注意：本题目与 316 https://leetcode.cn/problems/remove-duplicate-letters/ 相同
```

- 解题思路

```
func smallestSubsequence(text string) string {
        stack := make([]byte, 0)
        arr := [256]byte{}
        m := make(map[byte]bool)
        for i := 0; i < len(text); i++ {
                arr[text[i]]++
        }
        for i := 0; i < len(text); i++ {
                if m[text[i]] == true {
                        arr[text[i]]--
                        continue
                }
                // arr[栈顶]说明有重复元素
                // 栈顶>s[i]:说明字典序不满足
```

(续下页)

(接上页)

```
                for len(stack) > 0 && stack[len(stack)-1] > text[i] &&␣
↪arr[stack[len(stack)-1]] > 0 {
                        m[stack[len(stack)-1]] = false
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, text[i])
                arr[text[i]]--
                m[text[i]] = true
        }
        return string(stack)
}

# 2
func smallestSubsequence(text string) string {
        arr := [26]int{}
        pos := 0
        for i := 0; i < len(text); i++ {
                arr[text[i]-'a']++
        }
        for i := 0; i < len(text); i++ {
                if text[i] < text[pos] {
                        pos = i
                }
                arr[text[i]-'a']--
                if arr[text[i]-'a'] == 0 {
                        break
                }
        }
        if len(text) == 0 {
                return ""
        }
        newStr := strings.ReplaceAll(text[pos+1:], string(text[pos]), "")
        return string(text[pos]) + smallestSubsequence(newStr)
}
```

32.34 1090. 受标签影响的最大值 (1)

- 题目

```
我们有一个项的集合，其中第i项的值为values[i]，标签为labels[i]。
我们从这些项中选出一个子集S，这样一来：
|S| <= num_wanted
对于任意的标签 L，子集 S 中标签为 L的项的数目总满足<= use_limit。
返回子集S的最大可能的和。
示例 1：输入：values = [5,4,3,2,1], labels = [1,1,2,2,3], num_wanted = 3, use_limit =␣
↪1
输出：9
解释：选出的子集是第一项，第三项和第五项。
示例 2：输入：values = [5,4,3,2,1], labels = [1,3,3,3,2], num_wanted = 3, use_limit =␣
↪2
输出：12
解释：选出的子集是第一项，第二项和第三项。
示例 3：输入：values = [9,8,8,7,6], labels = [0,0,0,1,1], num_wanted = 3, use_limit =␣
↪1
输出：16
解释：选出的子集是第一项和第四项。
示例 4：输入：values = [9,8,8,7,6], labels = [0,0,0,1,1], num_wanted = 3, use_limit =␣
↪2
输出：24
解释：选出的子集是第一项，第二项和第四项。
提示：1 <= values.length == labels.length <= 20000
0 <= values[i], labels[i]<= 20000
1 <= num_wanted, use_limit<= values.length
```

- 解题思路

```
func largestValsFromLabels(values []int, labels []int, num_wanted int, use_limit int)␣
↪int {
        arr := make([][2]int, 0)
        for i := 0; i < len(values); i++ {
                arr = append(arr, [2]int{values[i], labels[i]})
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][0] > arr[j][0]
        })
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                if m[arr[i][1]] < use_limit {
```

(续下页)

(接上页)

```
                        res = res + arr[i][0]
                        m[arr[i][1]]++
                        num_wanted--
                }
                if num_wanted <= 0 {
                        break
                }
        }
        return res
}
```

32.35 1091. 二进制矩阵中的最短路径 (2)

- 题目

```
在一个N ×N 的方形网格中，每个单元格有两种状态：空（0）或者阻塞（1）。
一条从左上角到右下角、长度为 k 的畅通路径，由满足下述条件的单元格C_1, C_2, ..., C_
↪k组成：
相邻单元格C_i 和C_{i+1}在八个方向之一上连通（此时，C_i 和C_{i+1}不同且共享边或角）
C_1 位于(0, 0)（即，值为grid[0][0]）
C_k位于(N-1, N-1)（即，值为grid[N-1][N-1]）
如果 C_i 位于(r, c)，则 grid[r][c]为空（即，grid[r][c] ==0）
返回这条从左上角到右下角的最短畅通路径的长度。如果不存在这样的路径，返回 -1 。
示例 1：输入：[[0,1],[1,0]] 输出：2
示例 2：输入：[[0,0,0],[1,1,0],[1,1,0]] 输出：4
提示：1 <= grid.length == grid[0].length <= 100
grid[i][j] 为0 或1
```

- 解题思路

```
var dx = []int{-1, -1, -1, 0, 0, 1, 1, 1}
var dy = []int{-1, 0, 1, -1, 1, -1, 0, 1}

func shortestPathBinaryMatrix(grid [][]int) int {
        if grid[0][0] == 1 {
                return -1
        }
        n, m := len(grid), len(grid[0])
        if grid[n-1][m-1] == 1 {
                return -1
        }
        if n == 1 && m == 1 {
```

(续下页)

(接上页)

```
                return 1
        }
        visited := make(map[[2]int]bool)
        visited[[2]int{0, 0}] = true
        queue := make([][3]int, 0)
        queue = append(queue, [3]int{0, 0, 1})
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                x := node[0]
                y := node[1]
                v := node[2]
                for i := 0; i < 8; i++ {
                        newX := x + dx[i]
                        newY := y + dy[i]
                        if 0 <= newX && newX < n && 0 <= newY && newY < m &&
                                grid[newX][newY] == 0 && visited[[2]int{newX, newY}]
↪== false {
                                queue = append(queue, [3]int{newX, newY, v + 1})
                                visited[[2]int{newX, newY}] = true
                                if newX == n-1 && newY == m-1 {
                                        return v + 1
                                }
                        }
                }
        }
        return -1
}

# 2
var dx = []int{-1, -1, -1, 0, 0, 1, 1, 1}
var dy = []int{-1, 0, 1, -1, 1, -1, 0, 1}

func shortestPathBinaryMatrix(grid [][]int) int {
        if grid[0][0] == 1 {
                return -1
        }
        n, m := len(grid), len(grid[0])
        if grid[n-1][m-1] == 1 {
                return -1
        }
        if n == 1 && m == 1 {
                return 1
```

(续下页)

(接上页)

```
        }
        queue := make([]int, 0)
        queue = append(queue, 0)
        grid[0][0] = 1
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                x := node / m
                y := node % m
                for i := 0; i < 8; i++ {
                        newX := x + dx[i]
                        newY := y + dy[i]
                        if 0 <= newX && newX < n && 0 <= newY && newY < m &&␣
↪grid[newX][newY] == 0 {
                                queue = append(queue, newX*m+newY)
                                grid[newX][newY] = grid[x][y] + 1
                                if newX == n-1 && newY == m-1 {
                                        return grid[n-1][m-1]
                                }
                        }
                }
        }
        return -1
}
```

32.36 1093. 大样本统计 (1)

- 题目

```
我们对0到255之间的整数进行采样，并将结果存储在数组count中：count[k]就是整数k␣
↪的采样个数。
我们以浮点数数组的形式，分别返回样本的最小值、最大值、平均值、中位数和众数。其中，众数是保证唯一的。
我们先来回顾一下中位数的知识：
如果样本中的元素有序，并且元素数量为奇数时，中位数为最中间的那个元素；
如果样本中的元素有序，并且元素数量为偶数时，中位数为中间的两个元素的平均值。
示例 1：输入：count = [0,1,3,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

(续下页)

(接上页)

```
输出: [1.00000,3.00000,2.37500,2.50000,3.00000]
示例 2: 输入: count = [0,4,3,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
输出: [1.00000,4.00000,2.18182,2.00000,1.00000]
提示: count.length == 256
1 <= sum(count) <= 10^9
计数表示的众数是唯一的
答案与真实值误差在10^-5以内就会被视为正确答案
```

- 解题思路

```go
func sampleStats(count []int) []float64 {
	maxValue := math.MinInt32
	minValue := math.MaxInt32
	maxTime := 0
	maxTimeValue := 0
	sum := 0
	total := 0
	for i := 0; i < len(count); i++ {
		if count[i] > 0 {
			minValue = i
			break
		}
	}
	for i := len(count) - 1; i >= 0; i-- {
		if count[i] > 0 {
			maxValue = i
			break
		}
	}
	for i := 0; i < len(count); i++ {
		total = total + count[i]
		sum = sum + count[i]*i
		if count[i] > maxTime {
			maxTime = count[i]
			maxTimeValue = i
		}
	}
	a, b := 0, 0
```

(续下页)

(接上页)

```
        temp := 0
        for i := 0; i < len(count); i++ {
                if temp <= (total-1)/2 && (total-1)/2 < temp+count[i] {
                        a = i
                }
                if temp <= total/2 && total/2 < temp+count[i] {
                        b = i
                        break
                }
                temp = temp + count[i]
        }
        return []float64{
                float64(minValue),
                float64(maxValue),
                float64(sum) / float64(total),
                float64(a+b) / 2,
                float64(maxTimeValue),
        }
}
```

32.37 1094. 拼车 (1)

- 题目

```
假设你是一位顺风车司机，车上最初有 capacity 个空座位可以用来载客。由于道路的限制，
车 只能 向一个方向行驶（也就是说，不允许掉头或改变方向，你可以将其想象为一个向量）。
这儿有一份乘客行程计划表 trips[][]，其中 trips[i] =
[num_passengers, start_location, end_location] 包含了第 i 组乘客的行程信息：
    必须接送的乘客数量；
    乘客的上车地点；
    以及乘客的下车地点。
这些给出的地点位置是从你的 初始␣
↪出发位置向前行驶到这些地点所需的距离（它们一定在你的行驶方向上）。
请你根据给出的行程计划表和车子的座位数，来判断你的车是否可以顺利完成接送所有乘客的任务
（当且仅当你可以在所有给定的行程中接送所有乘客时，返回 true，否则请返回 false）。
示例 1：输入：trips = [[2,1,5],[3,3,7]], capacity = 4 输出：false
示例 2：输入：trips = [[2,1,5],[3,3,7]], capacity = 5 输出：true
示例 3：输入：trips = [[2,1,5],[3,5,7]], capacity = 3 输出：true
示例 4：输入：trips = [[3,2,7],[3,7,9],[8,3,9]], capacity = 11 输出：true
提示：
    你可以假设乘客会自觉遵守 “先下后上” 的良好素质
    trips.length <= 1000
```

(续下页)

(接上页)

```
    trips[i].length == 3
    1 <= trips[i][0] <= 100
    0 <= trips[i][1] < trips[i][2] <= 1000
    1 <= capacity <= 100000
```

- 解题思路

```
func carPooling(trips [][]int, capacity int) bool {
        arr := make([]int, 1001)
        for i := 0; i < len(trips); i++ {
                start := trips[i][1]
                end := trips[i][2]
                count := trips[i][0]
                arr[start] = arr[start] + count
                arr[end] = arr[end] - count
        }
        total := 0
        for i := 0; i < len(arr); i++ {
                total = total + arr[i]
                if total > capacity {
                        return false
                }
        }
        return true
}
```

CHAPTER 33

1001-1100-Hard

33.1 1028. 从先序遍历还原二叉树

33.1.1 题目

```
我们从二叉树的根节点 root 开始进行深度优先搜索。
在遍历中的每个节点处，我们输出 D 条短划线（其中 D␣
↪是该节点的深度），然后输出该节点的值。
（如果节点的深度为 D，则其直接子节点的深度为 D + 1。根节点的深度为 0）。
如果节点只有一个子节点，那么保证该子节点为左子节点。
给出遍历输出 S，还原树并返回其根节点 root。
示例 1：输入："1-2--3--4-5--6--7" 输出：[1,2,5,3,4,6,7]
示例 2：输入："1-2--3---4-5--6---7" 输出：[1,2,5,3,null,6,null,4,null,7]
示例 3：输入："1-401--349---90--88" 输出：[1,401,null,349,88,90]
提示：原始树中的节点数介于 1 和 1000 之间。
    每个节点的值介于 1 和 10 ^ 9 之间。
```

33.1.2 解题思路

33.2 1074. 元素和为目标值的子矩阵数量 (3)

- 题目

```
给出矩阵matrix和目标值target，返回元素总和等于目标值的非空子矩阵的数量。
子矩阵x1, y1, x2, y2是满足 x1 <= x <= x2且y1 <= y <= y2的所有单元matrix[x][y]的集合。
如果(x1, y1, x2, y2) 和(x1', y1', x2', y2')两个子矩阵中部分坐标不同（如：x1 != x1
↪'），那么这两个子矩阵也不同。
示例 1：输入：matrix = [[0,1,0],[1,1,1],[0,1,0]], target = 0 输出：4
解释：四个只含 0 的 1x1 子矩阵。
示例 2：输入：matrix = [[1,-1],[-1,1]], target = 0 输出：5
解释：两个 1x2 子矩阵，加上两个 2x1 子矩阵，再加上一个 2x2 子矩阵。
示例 3：输入：matrix = [[904]], target = 0 输出：0
提示：1 <= matrix.length <= 100
1 <= matrix[0].length <= 100
-1000 <= matrix[i] <= 1000
-10^8 <= target <= 10^8
```

- 解题思路

```
func numSubmatrixSumTarget(matrix [][]int, target int) int {
        res := 0
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        arr[i][j] = matrix[i-1][j-1] + arr[i-1][j] + arr[i][j-1] -␣
↪arr[i-1][j-1]
                }
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        for x := i; x <= n; x++ {
                                for y := j; y <= m; y++ {
                                        if arr[x][y]-arr[i-1][y]-arr[x][j-1]+arr[i-
↪1][j-1] == target {
```

(续下页)

(接上页)

```
                                                res++
                                        }
                                }
                        }
                }
        }
        return res
}

# 2
func numSubmatrixSumTarget(matrix [][]int, target int) int {
        res := 0
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        arr[i][j] = matrix[i-1][j-1] + arr[i-1][j] + arr[i][j-1] -␣
↪arr[i-1][j-1]
                }
        }
        for a := 1; a <= n; a++ { // 上边界
                for b := a; b <= n; b++ { // 下边界
                        temp := make(map[int]int)
                        temp[0] = 1
                        for j := 1; j <= m; j++ {
                                value := arr[b][j] - arr[a-1][j]
                                res = res + temp[value-target]
                                temp[value]++
                        }
                }
        }
        return res
}

# 3
func numSubmatrixSumTarget(matrix [][]int, target int) int {
        res := 0
        n, m := len(matrix), len(matrix[0])
        for a := 0; a < n; a++ { // 上边界
                arr := make([]int, m)    // 每列的和
```

(续下页)

(接上页)

```
                for b := a; b < n; b++ { // 下边界
                        for j := 0; j < m; j++ {
                                arr[j] = arr[j] + matrix[b][j]
                        }
                        temp := make(map[int]int)
                        temp[0] = 1
                        sum := 0
                        for j := 0; j < m; j++ {
                                sum = sum + arr[j]
                                res = res + temp[sum-target]
                                temp[sum]++
                        }
                }
        }
        return res
}
```

33.3 1095. 山脉数组中查找目标值

33.3.1 题目

```
（这是一个 交互式问题）
给你一个 山脉数组mountainArr，
请你返回能够使得mountainArr.get(index)等于target最小的下标 index值。
如果不存在这样的下标 index，就请返回-1。
何为山脉数组？如果数组A 是一个山脉数组的话，那它满足如下条件：
首先，A.length >= 3
其次，在0 < i< A.length - 1条件下，存在 i 使得：
A[0] < A[1] < ... A[i-1] < A[i]
A[i] > A[i+1] > ... > A[A.length - 1]
你将不能直接访问该山脉数组，必须通过MountainArray接口来获取数据：
MountainArray.get(k)- 会返回数组中索引为k的元素（下标从 0 开始）
MountainArray.length()- 会返回该数组的长度
注意：对MountainArray.get发起超过 100 次调用的提交将被视为错误答案。
此外，任何试图规避判题系统的解决方案都将会导致比赛资格被取消。
为了帮助大家更好地理解交互式问题，我们准备了一个样例 “答案”：
https://leetcode.cn/playground/RKhe3ave，请注意这 不是一个正确答案。
示例 1：输入：array = [1,2,3,4,5,3,1]，target = 3 输出：2
解释：3 在数组中出现了两次，下标分别为 2 和 5，我们返回最小的下标 2。
示例 2：输入：array = [0,1,2,4,2,1]，target = 3 输出：-1
解释：3 在数组中没有出现，返回 -1。
```

(续下页)

(接上页)

```
提示：3 <= mountain_arr.length() <= 10000
0 <= target <= 10^9
0 <= mountain_arr.get(index) <=10^9
```

33.3.2 解题思路

CHAPTER 34

1101-1200-Easy

34.1 1103. 分糖果 II(3)

- 题目

```
排排坐，分糖果。
我们买了一些糖果 candies，打算把它们分给排好队的 n = num_people 个小朋友。
给第一个小朋友 1 颗糖果，第二个小朋友 2 颗，依此类推，直到给最后一个小朋友 n 颗糖果。
然后，我们再回到队伍的起点，给第一个小朋友 n + 1 颗糖果，第二个小朋友 n + 2 颗，
依此类推，直到给最后一个小朋友 2 * n 颗糖果。
重复上述过程（每次都比上一次多给出一颗糖果，当到达队伍终点后再次从队伍起点开始），直到我们分完所有的
注意，就算我们手中的剩下糖果数不够（不比前一次发出的糖果多），这些糖果也会全部发给当前的小朋友。
返回一个长度为 num_people、元素之和为 candies 的数组，
以表示糖果的最终分发情况（即 ans[i] 表示第 i 个小朋友分到的糖果数）。
示例 1：输入：candies = 7, num_people = 4 输出：[1,2,3,1]
解释：
第一次，ans[0] += 1，数组变为 [1,0,0,0]。
第二次，ans[1] += 2，数组变为 [1,2,0,0]。
第三次，ans[2] += 3，数组变为 [1,2,3,0]。
第四次，ans[3] += 1（因为此时只剩下 1 颗糖果），最终数组变为 [1,2,3,1]。

示例 2：输入：candies = 10, num_people = 3 输出：[5,2,3]
解释：
第一次，ans[0] += 1，数组变为 [1,0,0]。
第二次，ans[1] += 2，数组变为 [1,2,0]。
```

(续下页)

(接上页)

```
第三次，ans[2] += 3，数组变为 [1,2,3]。
第四次，ans[0] += 4，最终数组变为 [5,2,3]。
提示：
    1 <= candies <= 10^9
    1 <= num_people <= 1000
```

- 解题思路

```
func distributeCandies(candies int, num_people int) []int {
        res := make([]int, num_people)
        i := 0
        count := 0
        for candies > 0 {
                count++
                if candies >= count {
                        res[i%num_people] += count
                } else {
                        res[i%num_people] += candies
                }
                i++
                candies = candies - count
        }
        return res
}

#
func distributeCandies(candies int, num_people int) []int {
        res := make([]int, num_people)
        count := 1
        for candies > 0 {
                for i := 0; i < num_people; i++ {
                        if candies >= count {
                                res[i] = res[i] + count
                                candies = candies - count
                        } else {
                                res[i] = res[i] + candies
                                candies = 0
                        }
                        count++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func distributeCandies(candies int, num_people int) []int {
        res := make([]int, num_people)
        times := 1
        for times*(times+1)/2 <= candies {
                times++
        }
        // 计算出当前糖果最多可以发给多少个人,剩下最后一个人多少颗糖
        times--
        last := candies - times*(times+1)/2
        for i := 0; i < num_people; i++ {
                n := times / num_people
                if times%num_people > i {
                        n = n + 1
                }
                // 等差数列{an}的通项公式为：an=a1+(n-1)d。
                // 前n项和公式为：Sn=n*a1+n(n-1)d/2或Sn=n(a1+an)/2
                // Sn=n(a1+a1+(n-1)d)/2=n(2a1+(n-1)d)/2
                // (i+1)为首项，num_people为公差，n为数列长度，的等差数列的和
                res[i] = n * (2*(i+1) + (n-1)*num_people) / 2
                if times%num_people == i {
                        res[i] = res[i] + last
                }
        }
        return res
}
```

34.2 1108.IP 地址无效化 (2)

- 题目

```
给你一个有效的 IPv4 地址 address，返回这个 IP 地址的无效化版本。
所谓无效化 IP 地址，其实就是用 "[.]" 代替了每个 "."。
示例 1：输入：address = "1.1.1.1" 输出："1[.]1[.]1[.]1"
示例 2：输入：address = "255.100.50.0" 输出："255[.]100[.]50[.]0"
提示：
    给出的 address 是一个有效的 IPv4 地址
```

- 解题思路

```
func defangIPaddr(address string) string {
        return strings.ReplaceAll(address, ".", "[.]")
```

(续下页)

(接上页)

```
}

#
func defangIPaddr(address string) string {
        res := ""
        for i := range address {
                if address[i] == '.' {
                        res = res + "[.]"
                } else {
                        res = res + string(address[i])
                }
        }
        return res
}
```

34.3 1122. 数组的相对排序 (3)

- 题目

```
给你两个数组，arr1 和 arr2，
    arr2 中的元素各不相同
    arr2 中的每个元素都出现在 arr1 中
对 arr1 中的元素进行排序，使 arr1 中项的相对顺序和 arr2 中的相对顺序相同。
未在 arr2 中出现过的元素需要按照升序放在 arr1 的末尾。
示例：
输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
输出：[2,2,2,1,4,3,3,9,6,7,19]
提示：
    arr1.length, arr2.length <= 1000
    0 <= arr1[i], arr2[i] <= 1000
    arr2 中的元素 arr2[i] 各不相同
    arr2 中的每个元素 arr2[i] 都出现在 arr1 中
```

- 解题思路

```
func relativeSortArray(arr1 []int, arr2 []int) []int {
        if len(arr2) == 0 {
                sort.Ints(arr1)
                return arr1
        }
        res := make([]int, 0)
        m := make(map[int]int)
```

(续下页)

(接上页)

```
        for i := range arr1 {
                m[arr1[i]]++
        }
        for i := 0; i < len(arr2); i++ {
                for j := 0; j < m[arr2[i]]; j++ {
                        res = append(res, arr2[i])
                }
                m[arr2[i]] = 0
        }
        tempArr := make([]int, 0)
        for key, value := range m {
                for value > 0 {
                        tempArr = append(tempArr, key)
                        value--
                }
        }
        sort.Ints(tempArr)
        res = append(res, tempArr...)
        return res
}

#
func relativeSortArray(arr1 []int, arr2 []int) []int {
        count := 0
        for i := 0; i < len(arr2); i++ {
                for j := count; j < len(arr1); j++ {
                        if arr2[i] == arr1[j] {
                                arr1[count], arr1[j] = arr1[j], arr1[count]
                                count++
                        }
                }
        }
        sort.Ints(arr1[count:])
        return arr1
}

#
func relativeSortArray(arr1 []int, arr2 []int) []int {
        temp := make([]int, 1001)
        for i := range arr1 {
                temp[arr1[i]]++
        }
        count := 0
```

(续下页)

(接上页)

```
        for i := range arr2 {
                for temp[arr2[i]] > 0 {
                        arr1[count] = arr2[i]
                        temp[arr2[i]]--
                        count++
                }
        }
        for i := 0; i < len(temp); i++ {
                for temp[i] > 0 {
                        arr1[count] = i
                        temp[i]--
                        count++
                }
        }
        return arr1
}
```

34.4 1128. 等价多米诺骨牌对的数量 (2)

- 题目

```
给你一个由一些多米诺骨牌组成的列表 dominoes。
如果其中某一张多米诺骨牌可以通过旋转 0 度或 180␣
↪度得到另一张多米诺骨牌，我们就认为这两张牌是等价的。
形式上，dominoes[i] = [a, b] 和 dominoes[j] = [c, d]
等价的前提是 a==c 且 b==d，或是 a==d 且 b==c。
在 0 <= i < j < dominoes.length 的前提下，
找出满足 dominoes[i] 和 dominoes[j] 等价的骨牌对 (i, j) 的数量。

示例：输入：dominoes = [[1,2],[2,1],[3,4],[5,6]] 输出：1
提示：
    1 <= dominoes.length <= 40000
    1 <= dominoes[i][j] <= 9
```

- 解题思路

```
func numEquivDominoPairs(dominoes [][]int) int {
        m := make(map[string]int)
        for i := 0; i < len(dominoes); i++ {
                a := dominoes[i][0]
                b := dominoes[i][1]
                if a > b {
```

(续下页)

(接上页)

```
                        a, b = b, a
                }
                m[fmt.Sprintf("%d,%d", a, b)]++
        }
        res := 0
        for _, v := range m {
                res = res + v*(v-1)/2
        }
        return res
}

#
func numEquivDominoPairs(dominoes [][]int) int {
        res := 0
        arr := make([]int, 101)
        for i := 0; i < len(dominoes); i++ {
                a := dominoes[i][0]
                b := dominoes[i][1]
                if a > b {
                        a, b = b, a
                }
                res = res + arr[a*10+b]
                arr[a*10+b]++
        }
        return res
}
```

34.5 1137. 第 N 个泰波那契数 (3)

- 题目

```
泰波那契序列 Tn 定义如下：
T0 = 0, T1 = 1, T2 = 1, 且在 n >= 0 的条件下 Tn+3 = Tn + Tn+1 + Tn+2
给你整数 n，请返回第 n 个泰波那契数 Tn 的值。

示例 1：输入：n = 4 输出：4
解释：
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4
示例 2：
输入：n = 25
输出：1389537
```

(续下页)

(接上页)

```
提示：
    0 <= n <= 37
    答案保证是一个 32 位整数，即 answer <= 2^31 - 1。
```

- 解题思路

```
func tribonacci(n int) int {
        arr := make([]int, n+3)
        arr[0] = 0
        arr[1] = 1
        arr[2] = 1
        for i := 3; i <= n; i++ {
                arr[i] = arr[i-1] + arr[i-2] + arr[i-3]
        }
        return arr[n]
}

#
func tribonacci(n int) int {
        if n == 0 {
                return 0
        }
        if n == 1 || n == 2 {
                return 1
        }
        a := 0
        b := 1
        c := 1
        for i := 3; i <= n; i++ {
                c, b, a = a+b+c, c, b
        }
        return c
}

#
var m map[int]int

func tribonacci(n int) int {
        if m == nil {
                m = make(map[int]int)
        }
        if n == 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        if n == 1 || n == 2 {
                return 1
        }
        if value, ok := m[n]; ok {
                return value
        } else {
                value := tribonacci(n-1) + tribonacci(n-2) + tribonacci(n-3)
                m[n] = value
        }
        return m[n]
}
```

34.6 1154. 一年中的第几天 (2)

- 题目

```
给你一个按 YYYY-MM-DD 格式表示日期的字符串 date，请你计算并返回该日期是当年的第几天。
通常情况下，我们认为 1 月 1 日是每年的第 1 天，1 月 2 日是每年的第 2 天，依此类推。
每个月的天数与现行公元纪年法（格里高利历）一致。
示例 1：输入：date = "2019-01-09" 输出：9
示例 2：输入：date = "2019-02-10" 输出：41
示例 3：输入：date = "2003-03-01" 输出：60
示例 4：输入：date = "2004-03-01" 输出：61
提示：
    date.length == 10
    date[4] == date[7] == '-'，其他的 date[i] 都是数字。
    date 表示的范围从 1900 年 1 月 1 日至 2019 年 12 月 31 日。
```

- 解题思路

```
func dayOfYear(date string) int {
        arr := strings.Split(date, "-")
        year, _ := strconv.Atoi(arr[0])
        month, _ := strconv.Atoi(arr[1])
        day, _ := strconv.Atoi(arr[2])
        res := 0
        for i := 0; i < month; i++ {
                switch i {
                case 1, 3, 5, 7, 8, 10, 12:
                        res = res + 31
                case 4, 6, 9, 11:
                        res = res + 30
```

(续下页)

(接上页)

```
                case 2:
                        res = res + 28
                        if year%400 == 0 || (year%4 == 0 && year%100 != 0) {
                                res = res + 1
                        }
                }
        }
        res = res + day
        return res
}

#
func dayOfYear(date string) int {
        format := "2006-01-02"
        t, _ := time.Parse(format, date)
        return t.YearDay()
}
```

34.7 1160. 拼写单词 (3)

- 题目

```
给你一份『词汇表』（字符串数组） words 和一张『字母表』（字符串） chars。
假如你可以用 chars 中的『字母』（字符）拼写出 words 中的某个『单词』（字符串），
那么我们就认为你掌握了这个单词。
注意：每次拼写（指拼写词汇表中的一个单词）时，chars 中的每个字母都只能用一次。
返回词汇表 words 中你掌握的所有单词的 长度之和。
示例 1：输入：words = ["cat","bt","hat","tree"], chars = "atach" 输出：6
解释：  可以形成字符串 "cat" 和 "hat"，所以答案是 3 + 3 = 6。
示例 2：输入：words = ["hello","world","leetcode"], chars = "welldonehoneyr" 输出：10
解释：  可以形成字符串 "hello" 和 "world"，所以答案是 5 + 5 = 10。
提示：
    1 <= words.length <= 1000
    1 <= words[i].length, chars.length <= 100
    所有字符串中都仅包含小写英文字母
```

- 解题思路

```
func countCharacters(words []string, chars string) int {
        m := make(map[byte]int)
        for i := range chars {
                m[chars[i]]++
```

(续下页)

(接上页)

```
        }
        res := 0
        for i := 0; i < len(words); i++ {
                temp := make(map[byte]int)
                flag := true
                for j := range words[i] {
                        temp[words[i][j]]++
                }
                if len(temp) > len(m) {
                        continue
                }
                for k, v := range temp {
                        if v > m[k] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = res + len(words[i])
                }
        }
        return res
}

#
func countCharacters(words []string, chars string) int {
        res := 0
        for i := 0; i < len(words); i++ {
                flag := true
                for _, v := range words[i] {
                        if strings.Count(words[i], string(v)) > strings.Count(chars,␣
↪string(v)) {
                                flag = false
                                continue
                        }
                }
                if flag == true {
                        res = res + len(words[i])
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func countCharacters(words []string, chars string) int {
        m := make([]int, 26)
        for i := range chars {
                m[chars[i]-'a']++
        }
        res := 0
        for i := 0; i < len(words); i++ {
                temp := make([]int, 26)
                flag := true
                for j := range words[i] {
                        temp[words[i][j]-'a']++
                }
                if len(temp) > len(m) {
                        continue
                }
                for k, v := range temp {
                        if v > m[k] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = res + len(words[i])
                }
        }
        return res
}
```

34.8 1170. 比较字符串最小字母出现频次 (2)

- 题目

```
我们来定义一个函数 f(s)，其中传入参数 s 是一个非空字符串；
该函数的功能是统计 s 中（按字典序比较）最小字母的出现频次。
例如，若 s = "dcce"，那么 f(s) = 2，因为最小的字母是 "c"，它出现了 2 次。
现在，给你两个字符串数组待查表 queries 和词汇表 words，请你返回一个整数数组 answer␣
↪作为答案，
其中每个 answer[i] 是满足 f(queries[i]) < f(W) 的词的数目，W 是词汇表 words 中的词。
示例 1：输入：queries = ["cbd"], words = ["zaaaz"] 输出：[1]
解释：查询 f("cbd") = 1，而 f("zaaaz") = 3 所以 f("cbd") < f("zaaaz")。
示例 2：输入：queries = ["bbb","cc"], words = ["a","aa","aaa","aaaa"] 输出：[1,2]
```

(续下页)

(接上页)

```
解释：第一个查询 f("bbb") < f("aaaa")，第二个查询 f("aaa") 和 f("aaaa") 都 > f("cc")。
提示：
    1 <= queries.length <= 2000
    1 <= words.length <= 2000
    1 <= queries[i].length, words[i].length <= 10
    queries[i][j], words[i][j] 都是小写英文字母
```

- 解题思路

```
func numSmallerByFrequency(queries []string, words []string) []int {
        queriesArr := make([]int, len(queries))
        wordsArr := make([]int, len(words))
        res := make([]int, 0)
        for i := 0; i < len(words); i++ {
                wordsArr[i] = f(words[i])
        }
        for i := 0; i < len(queries); i++ {
                queriesArr[i] = f(queries[i])
                count := 0
                for j := 0; j < len(wordsArr); j++ {
                        if queriesArr[i] < wordsArr[j] {
                                count++
                        }
                }
                res = append(res, count)
        }
        return res
}

func f(str string) int {
        min := str[0]
        count := 1
        for i := 1; i < len(str); i++ {
                if str[i] < min {
                        min = str[i]
                        count = 1
                } else if str[i] == min{
                        count++
                }
        }
        return count
}

#
```

(续下页)

(接上页)

```
func numSmallerByFrequency(queries []string, words []string) []int {
        wordsArr := make([]int, len(words))
        res := make([]int, 0)
        for i := 0; i < len(words); i++ {
                wordsArr[i] = f(words[i])
        }
        sort.Ints(wordsArr)
        for i := 0; i < len(queries); i++ {
                value := f(queries[i])
                count := binarySearch(value, wordsArr)
                res = append(res, count)
        }
        return res
}

func binarySearch(value int, target []int) int {
        left := 0
        right := len(target) - 1
        for left < right {
                mid := left + (right-left)/2
                if target[mid] > value {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        if target[left] <= value {
                return 0
        }
        return len(target) - left
}

func f(str string) int {
        min := str[0]
        count := 1
        for i := 1; i < len(str); i++ {
                if str[i] < min {
                        min = str[i]
                        count = 1
                } else if str[i] == min {
                        count++
                }
        }
```

(续下页)

(接上页)

```
        return count
}
```

34.9 1175. 质数排列 (1)

- 题目

```
请你帮忙给从 1 到 n␣
↪的数设计排列方案，使得所有的「质数」都应该被放在「质数索引」（索引从 1 开始）上；
你需要返回可能的方案总数。
让我们一起来回顾一下「质数」：质数一定是大于 1␣
↪的，并且不能用两个小于它的正整数的乘积来表示。
由于答案可能会很大，所以请你返回答案 模 mod 10^9 + 7 之后的结果即可。
示例 1：输入：n = 5 输出：12
解释：举个例子，[1,2,5,4,3] 是一个有效的排列，但 [5,2,3,4,1] 不是，
因为在第二种情况里质数 5 被错误地放在索引为 1 的位置上。
示例 2：输入：n = 100 输出：682289015
提示：
    1 <= n <= 100
```

- 解题思路

```
func numPrimeArrangements(n int) int {
        primeNum := 0
        for i := 2; i <= n; i++ {
                if isPrime(i) {
                        primeNum++
                }
        }
        a := 1
        for i := 2; i <= primeNum; i++ {
                a = a * i % 1000000007
        }
        for i := 2; i <= n-primeNum; i++ {
                a = a * i % 1000000007
        }
        return a
}

func isPrime(n int) bool {
        if n == 2 || n == 3 {
                return true
```

(续下页)

(接上页)

```
        }
        for i := 2; i*i <= n; i++ {
                if n%i == 0 {
                        return false
                }
        }
        return true
}
```

34.10 1184. 公交站间的距离 (2)

- 题目

```
环形公交路线上有 n 个站，按次序从 0 到 n - 1 进行编号。
我们已知每一对相邻公交站之间的距离，
distance[i] 表示编号为 i 的车站和编号为 (i + 1) % n 的车站之间的距离。
环线上的公交车都可以按顺时针和逆时针的方向行驶。
返回乘客从出发点 start 到目的地 destination 之间的最短距离。
示例 1：输入：distance = [1,2,3,4], start = 0, destination = 1 输出：1
解释：公交站 0 和 1 之间的距离是 1 或 9，最小值是 1。
示例 2：输入：distance = [1,2,3,4], start = 0, destination = 2 输出：3
解释：公交站 0 和 2 之间的距离是 3 或 7，最小值是 3。
示例 3：输入：distance = [1,2,3,4], start = 0, destination = 3 输出：4
解释：公交站 0 和 3 之间的距离是 6 或 4，最小值是 4。
提示：
    1 <= n <= 10^4
    distance.length == n
    0 <= start, destination < n
    0 <= distance[i] <= 10^4
```

- 解题思路

```
func distanceBetweenBusStops(distance []int, start int, destination int) int {
        x := 0
        y := 0
        for i := start; i != destination; i = (i + 1) % len(distance) {
                x = x + distance[i]
        }
        for i := destination; i != start; i = (i + 1) % len(distance) {
                y = y + distance[i]
        }
        if x > y {
```

(续下页)

(接上页)

```
                return y
        }
        return x
}

#
func distanceBetweenBusStops(distance []int, start int, destination int) int {
        x := 0
        sum := 0
        for i := 0; i < len(distance); i++ {
                sum = sum + distance[i]
                if start < destination {
                        if i >= start && i < destination {
                                x = x + distance[i]
                        }
                } else {
                        if i >= destination && i < start {
                                x = x + distance[i]
                        }
                }
        }
        if sum-x > x {
                return x
        }
        return sum - x
}
```

34.11 1185. 一周中的第几天 (3)

- 题目

```
给你一个日期，请你设计一个算法来判断它是对应一周中的哪一天。
输入为三个整数：day、month 和 year，分别表示日、月、年。
您返回的结果必须是这几个值中的一个
{"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}。
示例 1：输入：day = 31, month = 8, year = 2019 输出："Saturday"
示例 2：输入：day = 18, month = 7, year = 1999 输出："Sunday"
示例 3：输入：day = 15, month = 8, year = 1993 输出："Sunday"
提示：
    给出的日期一定是在 1971 到 2100 年之间的有效日期。
```

- 解题思路

```
func dayOfTheWeek(day int, month int, year int) string {
        t, _ := time.Parse("2006-01-02", fmt.Sprintf("%04d-%02d-%02d", year, month,␣
↪day))
        return t.Weekday().String()
}

#
// 蔡勒公式
// 基姆拉尔森计算公式
// https://baike.baidu.com/item/%E8%94%A1%E5%8B%92%E5%85%AC%E5%BC%8F
// https://www.cnblogs.com/SeekHit/p/7498408.html
// Week = (y+y/4-y/100+y/400+2*m+3*(m+1)/5+d) mod 7;
func dayOfTheWeek(day int, month int, year int) string {
        arr := []string{"Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
↪"Saturday", "Sunday"}
        if month == 1 || month == 2 {
                month = month + 12
                year--
        }
        week := (year + year/4 - year/100 + year/400 + 2*month + 3*(month+1)/5 + day)
↪% 7
        return arr[week]
}

#
var arr = []string{"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
↪ "Sunday"}
var monthDate = []int{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

func dayOfTheWeek(day int, month int, year int) string {
        day1 := totalDay(1993, 8, 15)
        day2 := totalDay(year, month, day)
        diff := 6 - day1%7
        return arr[(day2+diff)%7]
}

func totalDay(year, month, day int) int {
        total := 0
        for i := 1971; i < year; i++ {
                total = total + 365
                if isLeap(i) {
                        total = total + 1
                }
        }
```

(续下页)

(接上页)

```
        for i := 0; i < month-1; i++ {
                total = total + monthDate[i]
                if i == 1 && isLeap(year) {
                        total = total + 1
                }
        }
        total = total + day
        return total
}

func isLeap(year int) bool {
        return year%400 == 0 || (year%4 == 0 && year%100 != 0)
}
```

34.12 1189. “气球” 的最大数量 (3)

- 题目

```
给你一个字符串 text，你需要使用 text 中的字母来拼凑尽可能多的单词 "balloon"（气球）。
字符串 text 中的每个字母最多只能被使用一次。请你返回最多可以拼凑出多少个单词 "balloon
↪"。
示例 1：输入：text = "nlaebolko" 输出：1
示例 2：输入：text = "loonbalxballpoon" 输出：2
示例 3：输入：text = "leetcode" 输出：0
提示：
    1 <= text.length <= 10^4
    text 全部由小写英文字母组成
```

- 解题思路

```
func maxNumberOfBalloons(text string) int {
        m := make([]int, 26)
        str := "ablon"
        for i := 0; i < len(str); i++ {
                m[str[i]-'a']++
        }
        for i := 0; i < len(text); i++ {
                if m[text[i]-'a'] > 0 {
                        m[text[i]-'a']++
                }
        }
        min := math.MaxInt32
```

(续下页)

(接上页)

```
        for k, v := range m {
                if v == 0 {
                        continue
                }
                if k+'a' == 'l' || k+'a' == 'o' {
                        v = (v - 1) / 2
                } else {
                        v = v - 1
                }
                if v < min {
                        min = v
                }
        }
        return min
}

#
func maxNumberOfBalloons(text string) int {
        m := make([]int, 26)
        for i := 0; i < len(text); i++ {
                m[text[i]-'a']++
        }
        res := 0
        str := "balloon"
        for {
                for i := 0; i < len(str); i++ {
                        m[str[i]-'a']--
                        if m[str[i]-'a'] < 0 {
                                return res
                        }
                }
                res++
        }
}

#
func maxNumberOfBalloons(text string) int {
        arr := make([]int, 0)
        str := "ablon"
        for i := 0; i < len(str); i++ {
                if str[i] == 'l' || str[i] == 'o' {
                        arr = append(arr, strings.Count(text, string(str[i]))/2)
                } else {
```

(续下页)

(接上页)

```
                        arr = append(arr, strings.Count(text, string(str[i])))
                }
        }
        min := arr[0]
        for i := 1; i < len(arr); i++ {
                if arr[i] < min {
                        min = arr[i]
                }
        }
        return min
}
```

34.13 1200. 最小绝对差 (2)

- 题目

```
给你个整数数组 arr，其中每个元素都 不相同。
请你找到所有具有最小绝对差的元素对，并且按升序的顺序返回。
示例 1：输入：arr = [4,2,1,3] 输出：[[1,2],[2,3],[3,4]]
示例 2：输入：arr = [1,3,6,10,15] 输出：[[1,3]]
示例 3：输入：arr = [3,8,-10,23,19,-4,-14,27] 输出：[[-14,-10],[19,23],[23,27]]
提示：
    2 <= arr.length <= 10^5
    -10^6 <= arr[i] <= 10^6
```

- 解题思路

```
func minimumAbsDifference(arr []int) [][]int {
        sort.Ints(arr)
        result := make([][]int, 0)
        min := arr[1] - arr[0]
        result = append(result, []int{arr[0], arr[1]})
        for i := 2; i < len(arr); i++ {
                value := arr[i] - arr[i-1]
                if value < min {
                        min = value
                        result = make([][]int, 0)
                        result = append(result, []int{arr[i-1], arr[i]})
                } else if value == min {
                        result = append(result, []int{arr[i-1], arr[i]})
                }
        }
```

(续下页)

(接上页)

```
        return result
}

#
func minimumAbsDifference(arr []int) [][]int {
        sort.Ints(arr)
        result := make([][]int, 0)
        min := arr[1] - arr[0]
        for i := 2; i < len(arr); i++ {
                if min > arr[i]-arr[i-1] {
                        min = arr[i] - arr[i-1]
                }
        }
        for i := 1; i < len(arr); i++ {
                if min == arr[i]-arr[i-1] {
                        result = append(result, []int{arr[i-1], arr[i]})
                }
        }
        return result
}
```

CHAPTER 35

1101-1200-Medium

35.1 1104. 二叉树寻路 (3)

- 题目

```
在一棵无限的二叉树上，每个节点都有两个子节点，树中的节点 逐行 依次按“之”␣
↪字形进行标记。
如下图所示，在奇数行（即，第一行、第三行、第五行……）中，按从左到右的顺序进行标记；
而偶数行（即，第二行、第四行、第六行……）中，按从右到左的顺序进行标记。
给你树上某一个节点的标号 label，请你返回从根节点到该标号为 label 节点的路径，
该路径是由途经的节点标号所组成的。
示例 1：输入：label = 14 输出：[1,3,4,14]
示例 2：输入：label = 26 输出：[1,2,6,10,26]
提示：1 <= label <= 10^6
```

- 解题思路

```
func pathInZigZagTree(label int) []int {
        res := make([]int, 0)
        for label > 0 {
                res = append(res, label)
                label = label / 2
        }
        for i := 0; i < len(res)/2; i++ {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
```

(续下页)

(接上页)

```
        }
        i := 1
        if len(res)%2 == 0 {
                i = 2
        }
        for ; i < len(res); i = i + 2 {
                res[i] = (1<<i)*3 - 1 - res[i]
        }
        return res
}

# 2
func pathInZigZagTree(label int) []int {
        length := int(math.Log2(float64(label)))
        res := make([]int, length+1)
        res[length] = label
        length--
        i := 1
        for length >= 0 {
                target := int(math.Pow(2, float64(length+1))) + int(math.Pow(2,␣
↪float64(length))) - 1
                if i%2 == 1 {
                        res[length] = target - label/2
                } else {
                        res[length] = label / 2
                }
                i++
                length--
                label = label / 2
        }
        return res
}

# 3
func pathInZigZagTree(label int) []int {
        res := make([]int, 0)
        for label > 1 {
                res = append([]int{label}, res...)
                label = label / 2
                length := bits.Len32(uint32(label)) - 1
                label = label ^ ((1 << length) - 1) // 7^3=4 => 111^11=100
        }
        res = append([]int{1}, res...)
```

(续下页)

(接上页)

```
        return res
}
```

35.2 1105. 填充书架 (1)

- 题目

```
附近的家居城促销，你买回了一直心仪的可调节书架，打算把自己的书都整理到新的书架上。
你把要摆放的书 books都整理好，叠成一摞：
从上往下，第 i本书的厚度为 books[i][0]，高度为 books[i][1]。
按顺序将这些书摆放到总宽度为shelf_width 的书架上。
先选几本书放在书架上（它们的厚度之和小于等于书架的宽度 shelf_
↪width），然后再建一层书架。
重复这个过程，直到把所有的书都放在书架上。
需要注意的是，在上述过程的每个步骤中，摆放书的顺序与你整理好的顺序相同。
例如，如果这里有 5 本书，那么可能的一种摆放情况是：
第一和第二本书放在第一层书架上，第三本书放在第二层书架上，第四和第五本书放在最后一层书架上。
每一层所摆放的书的最大高度就是这一层书架的层高，书架整体的高度为各层高之和。
以这种方式布置书架，返回书架整体可能的最小高度。
示例：输入：books = [[1,1],[2,3],[2,3],[1,1],[1,1],[1,1],[1,2]], shelf_width = 4␣
↪输出：6
解释：3 层书架的高度和为 1 + 3 + 2 = 6 。
第 2 本书不必放在第一层书架上。
提示：1 <= books.length <= 1000
1 <= books[i][0] <= shelf_width <= 1000
1 <= books[i][1] <= 1000
```

- 解题思路

```
func minHeightShelves(books [][]int, shelf_width int) int {
        n := len(books)
        dp := make([]int, n+1) // 以第i本书作为结尾的总高度
        for i := 1; i <= n; i++ {
                w, h := books[i-1][0], books[i-1][1]
                dp[i] = dp[i-1] + h // 当前这本书单独一层的高度
                for j := i - 1; j > 0; j-- {
                        if w+books[j-1][0] <= shelf_width {
                                w = w + books[j-1][0]
                                h = max(h, books[j-1][1])
                                dp[i] = min(dp[i], dp[j-1]+h)
                        } else {
                                break
```

(续下页)

(接上页)

```
                        }
                }
        }
        return dp[n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

35.3 1109. 航班预订统计 (2)

- 题目

```
这里有 n 个航班，它们分别从 1 到 n 进行编号。
我们这儿有一份航班预订表，
表中第 i 条预订记录 bookings[i] = [i, j, k] 意味着我们在从 i 到 j 的每个航班上预订了␣
↪k 个座位。
请你返回一个长度为 n 的数组 answer，按航班编号顺序返回每个航班上预订的座位数。
示例：输入：bookings = [[1,2,10],[2,3,20],[2,5,25]], n = 5 输出：[10,55,45,25,25]
提示：1 <= bookings.length <= 20000
    1 <= bookings[i][0] <= bookings[i][1] <= n <= 20000
    1 <= bookings[i][2] <= 10000
```

- 解题思路

```
func corpFlightBookings(bookings [][]int, n int) []int {
        arr := make([]int, n+1)
        for i := 0; i < len(bookings); i++ {
                start := bookings[i][0] - 1
                end := bookings[i][1] - 1
                count := bookings[i][2]
```

(续下页)

(接上页)

```
                arr[start] = arr[start] + count
                arr[end+1] = arr[end+1] - count
        }
        res := make([]int, 0)
        total := 0
        for i := 0; i < n; i++ {
                total = total + arr[i]
                res = append(res, total)
        }
        return res
}

# 2
func corpFlightBookings(bookings [][]int, n int) []int {
        arr := make([]int, n)
        for i := 0; i < len(bookings); i++ {
                start := bookings[i][0]
                end := bookings[i][1]
                count := bookings[i][2]
                arr[start-1] = arr[start-1] + count
                if end < n {
                        arr[end] = arr[end] - count
                }
        }
        for i := 1; i < n; i++ {
                arr[i] = arr[i] + arr[i-1]
        }
        return arr
}
```

35.4 1110. 删点成林 (1)

- 题目

```
给出二叉树的根节点root，树上每个节点都有一个不同的值。
如果节点值在to_
↪delete中出现，我们就把该节点从树上删去，最后得到一个森林（一些不相交的树构成的集合）。
返回森林中的每棵树。你可以按任意顺序组织答案。
示例：输入：root = [1,2,3,4,5,6,7], to_delete = [3,5] 输出：[[1,2,null,4],[6],[7]]
提示：树中的节点数最大为1000。
每个节点都有一个介于1 到1000之间的值，且各不相同。
to_delete.length <= 1000
```

(续下页)

(接上页)

```
to_delete 包含一些从1 到1000、各不相同的值。
```

- 解题思路

```
var res []*TreeNode
var m map[int]bool

func delNodes(root *TreeNode, to_delete []int) []*TreeNode {
        res = make([]*TreeNode, 0)
        m = make(map[int]bool)
        for i := 0; i < len(to_delete); i++ {
                m[to_delete[i]] = true
        }
        root = dfs(root)
        if root != nil {
                res = append(res, root)
        }
        return res
}

func dfs(root *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        root.Left = dfs(root.Left)
        root.Right = dfs(root.Right)
        if m[root.Val] == true {
                if root.Left != nil {
                        res = append(res, root.Left)
                }
                if root.Right != nil {
                        res = append(res, root.Right)
                }
                return nil
        }
        return root
}
```

35.5 1111. 有效括号的嵌套深度 (3)

- 题目

```
有效括号字符串 定义：对于每个左括号，都能找到与之对应的右括号，反之亦然。
详情参见题末「有效括号字符串」部分。
嵌套深度 depth 定义：即有效括号字符串嵌套的层数，depth(A) 表示有效括号字符串 A␣
↪的嵌套深度。
详情参见题末「嵌套深度」部分。
有效括号字符串类型与对应的嵌套深度计算方法如下图所示：
给你一个「有效括号字符串」 seq，请你将其分成两个不相交的有效括号字符串，A 和 B，
并使这两个字符串的深度最小。
    不相交：每个 seq[i] 只能分给 A 和 B 二者中的一个，不能既属于 A 也属于 B 。
    A 或 B 中的元素在原字符串中可以不连续。
    A.length + B.length = seq.length
    深度最小：max(depth(A), depth(B)) 的可能取值最小。
划分方案用一个长度为 seq.length 的答案数组 answer 表示，编码规则如下：
    answer[i] = 0，seq[i] 分给 A 。
    answer[i] = 1，seq[i] 分给 B 。
如果存在多个满足要求的答案，只需返回其中任意 一个 即可。
示例 1：输入：seq = "(()())" 输出：[0,1,1,1,1,0]
示例 2：输入：seq = "()(())()" 输出：[0,0,0,1,1,0,1,1]
解释：本示例答案不唯一。
按此输出 A = "()()"，B = "()()"，max(depth(A), depth(B)) = 1，它们的深度最小。
像 [1,1,1,0,0,1,1,1]，也是正确结果，其中 A = "()()()"，B = "()"，max(depth(A),␣
↪depth(B)) = 1 。
提示：1 < seq.size <= 10000
有效括号字符串：仅由 "(" 和 ")"␣
↪构成的字符串，对于每个左括号，都能找到与之对应的右括号，反之亦然。
下述几种情况同样属于有效括号字符串：
  1. 空字符串
  2. 连接，可以记作 AB（A 与 B 连接），其中 A 和 B 都是有效括号字符串
  3. 嵌套，可以记作 (A)，其中 A 是有效括号字符串
嵌套深度：类似地，我们可以定义任意有效括号字符串 s 的 嵌套深度 depth(S)：
  1. s 为空时，depth("") = 0
  2. s 为 A 与 B 连接时，depth(A + B) = max(depth(A), depth(B))，其中 A 和 B␣
↪都是有效括号字符串
  3. s 为嵌套情况，depth("(" + A + ")") = 1 + depth(A)，其中 A 是有效括号字符串
例如："", "()()"，和 "()(()())" 都是有效括号字符串，
嵌套深度分别为 0，1，2，而 ")(" 和 "(()" 都不是有效括号字符串。
```

- 解题思路

```
func maxDepthAfterSplit(seq string) []int {
```

(续下页)

(接上页)

```
        res := make([]int, 0)
        level := 0
        for i := 0; i < len(seq); i++ {
                if seq[i] == '(' {
                        level++
                        res = append(res, level%2)
                } else {
                        res = append(res, level%2)
                        level--
                }
        }
        return res
}

# 2
func maxDepthAfterSplit(seq string) []int {
        res := make([]int, 0)
        for i := 0; i < len(seq); i++ {
                if seq[i] == '(' {
                        res = append(res, i%2)
                } else {
                        res = append(res, 1-i%2)
                }
        }
        return res
}

# 3
func maxDepthAfterSplit(seq string) []int {
        res := make([]int, 0)
        a, b := 0, 0
        for i := 0; i < len(seq); i++ {
                if seq[i] == '(' {
                        // 谁少给谁
                        if a <= b {
                                a++
                                res = append(res, 0)
                        } else {
                                b++
                                res = append(res, 1)
                        }
                } else {
                        // 谁多减谁
```

(续下页)

(接上页)

```
                        if a > b {
                                a--
                                res = append(res, 0)
                        } else {
                                b--
                                res = append(res, 1)
                        }
                }
        }
        return res
}
```

35.6 1123. 最深叶节点的最近公共祖先 (2)

- 题目

```
给你一个有根节点的二叉树，找到它最深的叶节点的最近公共祖先。
回想一下：
叶节点 是二叉树中没有子节点的节点
树的根节点的深度为0，如果某一节点的深度为d，那它的子节点的深度就是d+1
如果我们假定 A 是一组节点S的 最近公共祖先，S中的每个节点都在以 A 为根节点的子树中，
且 A的深度达到此条件下可能的最大值。
注意：本题与力扣 865 重复：
示例 1：输入：root = [3,5,1,6,2,0,8,null,null,7,4] 输出：[2,7,4]
解释：我们返回值为 2 的节点，在图中用黄色标记。
在图中用蓝色标记的是树的最深的节点。
注意，节点 6、0 和 8 也是叶节点，但是它们的深度是 2 ，而节点 7 和 4 的深度是 3 。
示例 2：输入：root = [1] 输出：[1]
解释：根节点是树中最深的节点，它是它本身的最近公共祖先。
示例 3：输入：root = [0,1,3,null,2] 输出：[2]
解释：树中最深的叶节点是 2 ，最近公共祖先是它自己。
提示：给你的树中将有1 到 1000 个节点。
树中每个节点的值都在 1 到 1000 之间。
每个节点的值都是独一无二的。
```

- 解题思路

```
func lcaDeepestLeaves(root *TreeNode) *TreeNode {
        res, _ := dfs(root, 0)
        return res
}
```

(续下页)

(接上页)

```
func dfs(root *TreeNode, level int) (*TreeNode, int) {
        if root == nil {
                return root, level
        }
        leftNode, left := dfs(root.Left, level+1)
        rightNode, right := dfs(root.Right, level+1)
        if left == right {
                return root, left + 1
        } else if left > right {
                return leftNode, left + 1
        }
        return rightNode, right + 1
}

# 2
func lcaDeepestLeaves(root *TreeNode) *TreeNode {
        if root == nil{
                return nil
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        if left == right{
                return root
        }else if left > right{
                return lcaDeepestLeaves(root.Left)
        }
        return lcaDeepestLeaves(root.Right)
}

func dfs(root *TreeNode) (int) {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        return 1+max(left,right)
}

func max(a, b int)int  {
        if a > b{
                return a
        }
        return b
}
```

35.7 1124. 表现良好的最长时间段 (3)

- 题目

```
给你一份工作时间表hours，上面记录着某一位员工每天的工作小时数。
我们认为当员工一天中的工作小时数大于8 小时的时候，那么这一天就是「劳累的一天」。
所谓「表现良好的时间段」，意味在这段时间内，「劳累的天数」是严格␣
↪大于「不劳累的天数」。
请你返回「表现良好时间段」的最大长度。
示例 1：输入：hours = [9,9,6,0,6,6,9] 输出：3
解释：最长的表现良好时间段是 [9,9,6]。
提示：1 <= hours.length <= 10000
0 <= hours[i] <= 16
```

- 解题思路

```
func longestWPI(hours []int) int {
        arr := make([]int, 0)
        for i := 0; i < len(hours); i++ {
                if hours[i] > 8 {
                        arr = append(arr, 1)
                } else {
                        arr = append(arr, -1)
                }
        }
        temp := make([]int, len(hours)+1)
        for i := 1; i <= len(hours); i++ {
                temp[i] = temp[i-1] + arr[i-1]
        }
        stack := make([]int, 0)
        // 单调栈
        for i := 0; i < len(temp); i++ {
                if len(stack) == 0 || temp[stack[len(stack)-1]] > temp[i] {
                        stack = append(stack, i)
                }
        }
        res := 0
        for i := len(temp) - 1; i >= 0; i-- {
                if len(stack) == 0 {
                        break
                }
                for len(stack) > 0 && temp[i] > temp[stack[len(stack)-1]] {
                        if i-stack[len(stack)-1] > res {
                                res = i - stack[len(stack)-1]
```

(续下页)

(接上页)

```
                        }
                        stack = stack[:len(stack)-1]
                }
        }
        return res
}

# 2
func longestWPI(hours []int) int {
        arr := make([]int, 0)
        for i := 0; i < len(hours); i++ {
                if hours[i] > 8 {
                        arr = append(arr, 1)
                } else {
                        arr = append(arr, -1)
                }
        }
        temp := make([]int, len(hours)+1)
        for i := 1; i <= len(hours); i++ {
                temp[i] = temp[i-1] + arr[i-1]
        }
        res := 0
        for i := 0; i < len(hours); i++ {
                for j := i; j < len(hours); j++ {
                        count := temp[j+1] - temp[i]
                        if count > 0 {
                                res = max(res, j-i+1)
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func longestWPI(hours []int) int {
        m := make(map[int]int)
```

(续下页)

(接上页)

```
        count := 0
        res := 0
        for i := 0; i < len(hours); i++ {
                if hours[i] > 8 {
                        count++
                } else {
                        count--
                }
                if count > 0 {
                        res = i + 1
                } else {
                        if _, ok := m[count]; ok == false {
                                m[count] = i
                        }
                        // (count-(count-1))=1>0
                        if _, ok := m[count-1]; ok == true {
                                res = max(res, i-m[count-1])
                        }

                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

35.8 1129. 颜色交替的最短路径 (2)

- 题目

```
在一个有向图中，节点分别标记为0，1，...，n-
↪1。这个图中的每条边不是红色就是蓝色，且存在自环或平行边。
red_edges中的每一个[i，j]对表示从节点 i 到节点 j 的红色有向边。
类似地，blue_edges中的每一个[i，j]对表示从节点 i 到节点 j 的蓝色有向边。
返回长度为 n␣
↪的数组answer，其中answer[X]是从节点0到节点X的红色边和蓝色边交替出现的最短路径的长度。
如果不存在这样的路径，那么 answer[x] = -1。
```

(续下页)

(接上页)

```
示例 1：输入：n = 3, red_edges = [[0,1],[1,2]], blue_edges = [] 输出：[0,1,-1]
示例 2：输入：n = 3, red_edges = [[0,1]], blue_edges = [[2,1]] 输出：[0,1,-1]
示例 3：输入：n = 3, red_edges = [[1,0]], blue_edges = [[2,1]] 输出：[0,-1,-1]
示例 4：输入：n = 3, red_edges = [[0,1]], blue_edges = [[1,2]] 输出：[0,1,2]
示例 5：输入：n = 3, red_edges = [[0,1],[0,2]], blue_edges = [[1,0]] 输出：[0,1,1]
提示：1 <= n <= 100
red_edges.length <= 400
blue_edges.length <= 400
red_edges[i].length == blue_edges[i].length == 2
0 <= red_edges[i][j], blue_edges[i][j] < n
```

- 解题思路

```
var redArr [][]int
var blueArr [][]int
var res []int

func shortestAlternatingPaths(n int, red_edges [][]int, blue_edges [][]int) []int {
        redArr = make([][]int, n)
        blueArr = make([][]int, n)
        for i := 0; i < len(red_edges); i++ {
                a, b := red_edges[i][0], red_edges[i][1]
                redArr[a] = append(redArr[a], b)
        }
        for i := 0; i < len(blue_edges); i++ {
                a, b := blue_edges[i][0], blue_edges[i][1]
                blueArr[a] = append(blueArr[a], b)
        }
        res = make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = -1
        }
        res[0] = 0
        bfs(n, 0) // 红
        bfs(n, 1) // 蓝
        return res
}

func bfs(n int, color int) {
        visited := make([][2]bool, n)
        visited[0][color] = true
        queue := make([]int, 0)
        queue = append(queue, 0)
        count := 0
```

(续下页)

(接上页)

```
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        node := queue[i]
                        targetColor := count % 2
                        if targetColor == color { //偶数次和当前颜色一致
                                for i := 0; i < len(redArr[node]); i++ {
                                        next := redArr[node][i]
                                        if visited[next][targetColor] == false {
                                                visited[next][targetColor] = true
                                                if res[next] == -1 || res[next] >␣
↪count+1 {
                                                        res[next] = count + 1
                                                }
                                                queue = append(queue, next)
                                        }
                                }
                        } else {
                                for i := 0; i < len(blueArr[node]); i++ {
                                        next := blueArr[node][i]
                                        if visited[next][targetColor] == false {
                                                visited[next][targetColor] = true
                                                if res[next] == -1 || res[next] >␣
↪count+1 {
                                                        res[next] = count + 1
                                                }
                                                queue = append(queue, next)
                                        }
                                }
                        }
                }
                queue = queue[length:]
                count++
        }
}

# 2
func shortestAlternatingPaths(n int, red_edges [][]int, blue_edges [][]int) []int {
        redArr, blueArr := make([][]int, n), make([][]int, n)
        for i := 0; i < len(red_edges); i++ {
                a, b := red_edges[i][0], red_edges[i][1]
                redArr[a] = append(redArr[a], b)
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(blue_edges); i++ {
                a, b := blue_edges[i][0], blue_edges[i][1]
                blueArr[a] = append(blueArr[a], b)
        }
        res := make([]int, n) // res[0] = 0
        for i := 1; i < n; i++ {
                res[i] = -1
        }
        queue, visited := make([][2]int, 0), make([][2]bool, n)
        for i := 0; i < len(redArr[0]); i++ {
                queue = append(queue, [2]int{redArr[0][i], 0})
        }
        for i := 0; i < len(blueArr[0]); i++ {
                queue = append(queue, [2]int{blueArr[0][i], 1})
        }
        count := 1
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        node, targetColor := queue[i][0], queue[i][1]
                        if res[node] == -1 {
                                res[node] = count
                        }
                        if targetColor == 0 && visited[node][targetColor] == false {
                                visited[node][targetColor] = true
                                for j := 0; j < len(blueArr[node]); j++ {
                                        queue = append(queue, [2]int{blueArr[node][j],
↪ 1})
                                }
                        }
                        if targetColor == 1 && visited[node][targetColor] == false {
                                visited[node][targetColor] = true
                                for j := 0; j < len(redArr[node]); j++ {
                                        queue = append(queue, [2]int{redArr[node][j],␣
↪0})
                                }
                        }
                }
                queue = queue[length:]
                count++
        }
        return res
}
```

35.9 1130. 叶值的最小代价生成树 (3)

- 题目

```
给你一个正整数数组arr，考虑所有满足以下条件的二叉树：
每个节点都有 0 个或是 2 个子节点。
数组arr中的值与树的中序遍历中每个叶节点的值一一对应。
（知识回顾：如果一个节点有 0 个子节点，那么该节点为叶节点。）
每个非叶节点的值等于其左子树和右子树中叶节点的最大值的乘积。
在所有这样的二叉树中，返回每个非叶节点的值的最小可能总和。这个和的值是一个32 位整数。
示例：输入：arr = [6,2,4] 输出：32
解释：  有两种可能的树，第一种的非叶节点的总和为 36，第二种非叶节点的总和为 32。
    24            24
   /  \          /  \
  12   4        6    8
 /  \               / \
6    2             2   4
提示：2 <= arr.length <= 40
1 <= arr[i] <= 15
答案保证是一个 32 位带符号整数，即小于2^31。
```

- 解题思路

```
func mctFromLeafValues(arr []int) int {
        res := 0
        stack := make([]int, 0) // 单调递减栈
        stack = append(stack, math.MaxInt32)
        for i := 0; i < len(arr); i++ {
                for len(stack) > 0 && arr[i] >= stack[len(stack)-1] { // 大于栈顶
                        middle := stack[len(stack)-1] // 中间
                        stack = stack[:len(stack)-1]
                        left := stack[len(stack)-1]
                        right := arr[i]
                        res = res + middle*min(left, right)
                }
                stack = append(stack, arr[i])
        }
        for len(stack) > 2 {
                a := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                b := stack[len(stack)-1]
                res = res + a*b
        }
        return res
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func mctFromLeafValues(arr []int) int {
        n := len(arr)
        maxArr := make([][]int, n)
        for i := 0; i < n; i++ {
                maxArr[i] = make([]int, n)
                maxValue := arr[i]
                for j := i; j < n; j++ {
                        maxValue = max(maxValue, arr[j])
                        maxArr[i][j] = maxValue // i到j之间的最大值
                }
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for j := 0; j < n; j++ {
                for i := j - 1; i >= 0; i-- {
                        dp[i][j] = math.MaxInt32
                        for k := i; k+1 <= j; k++ {
                                // dp[i][j]代表从i到j之间的最小代价
                                dp[i][j] = min(dp[i][j],␣
↪dp[i][k]+dp[k+1][j]+maxArr[i][k]*maxArr[k+1][j])
                        }
                }
        }
        return dp[0][n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func mctFromLeafValues(arr []int) int {
        res := 0
        stack := make([]int, 0) // 单调递减栈
        for i := 0; i < len(arr); i++ {
                for len(stack) > 0 && arr[i] >= stack[len(stack)-1] { // 大于栈顶
                        middle := stack[len(stack)-1] // 中间
                        stack = stack[:len(stack)-1]
                        right := arr[i]
                        var left int
                        if len(stack) == 0 {
                                left = math.MaxInt32
                        } else {
                                left = stack[len(stack)-1]
                        }
                        res = res + middle*min(left, right)
                }
                stack = append(stack, arr[i])
        }
        for len(stack) >= 2 {
                a := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                b := stack[len(stack)-1]
                res = res + a*b
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

35.10 1131. 绝对值表达式的最大值 (2)

- 题目

```
给你两个长度相等的整数数组，返回下面表达式的最大值：
|arr1[i] - arr1[j]| + |arr2[i] - arr2[j]| + |i - j|
其中下标 i，j 满足0 <= i, j < arr1.length。
示例 1：输入：arr1 = [1,2,3,4], arr2 = [-1,4,5,6] 输出：13
示例 2：输入：arr1 = [1,-2,-5,0,10], arr2 = [0,-2,-1,-7,-4] 输出：20
提示：2 <= arr1.length == arr2.length <= 40000
-10^6 <= arr1[i], arr2[i] <= 10^6
```

- 解题思路

```
func maxAbsValExpr(arr1 []int, arr2 []int) int {
        /*
                |arr1[i] - arr1[j]| + |arr2[i] - arr2[j]| + |i - j|
                =  (arr1[i] + arr2[i] + i) - (arr1[j] + arr2[j] + j)
                =  (arr1[i] + arr2[i] - i) - (arr1[j] + arr2[j] - j)
                =  (arr1[i] - arr2[i] + i) - (arr1[j] - arr2[j] + j)
                =  (arr1[i] - arr2[i] - i) - (arr1[j] - arr2[j] - j)
                = -(arr1[i] + arr2[i] + i) + (arr1[j] + arr2[j] + j)
                = -(arr1[i] + arr2[i] - i) + (arr1[j] + arr2[j] - j)
                = -(arr1[i] - arr2[i] + i) + (arr1[j] - arr2[j] + j)
                = -(arr1[i] - arr2[i] - i) + (arr1[j] - arr2[j] - j)
                其中：
                A = arr1[i] + arr2[i] + i
                B = arr1[i] + arr2[i] - i
                C = arr1[i] - arr2[i] + i
                D = arr1[i] - arr2[i] - i
                结果：
                max( |arr1[i] - arr1[j]| + |arr2[i] - arr2[j]| + |i - j|)
                = max(max(A) - min(A),max(B) - min(B),max(C) - min(C), max(D) -␣
↪min(D))
        */
        arr := make([][]int, 4)
        for i := 0; i < len(arr1); i++ {
                a, b := arr1[i], arr2[i]
                arr[0] = append(arr[0], a+b+i)
                arr[1] = append(arr[1], a+b-i)
                arr[2] = append(arr[2], a-b+i)
                arr[3] = append(arr[3], a-b-i)
        }
        a, b, c, d := getValue(arr[0]), getValue(arr[1]), getValue(arr[2]),␣
```

(续下页)

(接上页)

```
↪getValue(arr[3])
        return max(a, max(b, max(c, d)))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func getValue(arr []int) int {
        minValue, maxValue := arr[0], arr[0]
        for i := 0; i < len(arr); i++ {
                if arr[i] > maxValue {
                        maxValue = arr[i]
                }
                if arr[i] < minValue {
                        minValue = arr[i]
                }
        }
        return maxValue - minValue
}

# 2
func maxAbsValExpr(arr1 []int, arr2 []int) int {
        aMaxValue, bMaxValue, cMaxValue, dMaxValue := math.MinInt32, math.MinInt32,␣
↪math.MinInt32, math.MinInt32
        aMinValue, bMinValue, cMinValue, dMinValue := math.MaxInt32, math.MaxInt32,␣
↪math.MaxInt32, math.MaxInt32
        for i := 0; i < len(arr1); i++ {
                aMaxValue = max(aMaxValue, arr1[i]+arr2[i]+i)
                aMinValue = min(aMinValue, arr1[i]+arr2[i]+i)
                bMaxValue = max(bMaxValue, arr1[i]+arr2[i]-i)
                bMinValue = min(bMinValue, arr1[i]+arr2[i]-i)
                cMaxValue = max(cMaxValue, arr1[i]-arr2[i]+i)
                cMinValue = min(cMinValue, arr1[i]-arr2[i]+i)
                dMaxValue = max(dMaxValue, arr1[i]-arr2[i]-i)
                dMinValue = min(dMinValue, arr1[i]-arr2[i]-i)
        }
        a, b := aMaxValue-aMinValue, bMaxValue-bMinValue
        c, d := cMaxValue-cMinValue, dMaxValue-dMinValue
        return max(a, max(b, max(c, d)))
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

35.11 1138. 字母板上的路径 (1)

- 题目

```
我们从一块字母板上的位置(0, 0)出发，该坐标对应的字符为board[0][0]。
在本题里，字母板为board = ["abcde", "fghij", "klmno", "pqrst", "uvwxy", "z
↪"]，如下所示。
我们可以按下面的指令规则行动：
如果方格存在，'U'意味着将我们的位置上移一行；
如果方格存在，'D'意味着将我们的位置下移一行；
如果方格存在，'L'意味着将我们的位置左移一列；
如果方格存在，'R'意味着将我们的位置右移一列；
'!'会把在我们当前位置 (r, c) 的字符board[r][c]添加到答案中。
（注意，字母板上只存在有字母的位置。）
返回指令序列，用最小的行动次数让答案和目标target相同。你可以返回任何达成目标的路径。
示例 1：输入：target = "leet" 输出："DDR!UURRR!!DDD!"
示例 2：输入：target = "code" 输出："RR!DDRR!UUL!R!"
提示：1 <= target.length <= 100
target仅含有小写英文字母。
```

- 解题思路

```
func alphabetBoardPath(target string) string {
        res := ""
        x, y := 0, 0
        for i := 0; i < len(target); i++ {
```

(续下页)

(接上页)

```
                newX := int(target[i]-'a') / 5
                newY := int(target[i]-'a') % 5
                if x > newX {
                        res = res + strings.Repeat("U", x-newX) // 优先向上：从Z往上
                }
                if y > newY {
                        res = res + strings.Repeat("L", y-newY) // 优先向左：往Z走
                }
                if y < newY {
                        res = res + strings.Repeat("R", newY-y) // 向右
                }
                if x < newX {
                        res = res + strings.Repeat("D", newX-x) // 向下
                }
                res = res + "!"
                x, y = newX, newY
        }
        return res
}
```

35.12 1139. 最大的以 1 为边界的正方形 (1)

- 题目

```
给你一个由若干 0 和 1 组成的二维网格grid，请你找出边界全部由 1 组成的最大 正方形子网格，并返回该子网格中的元素数量。
如果不存在，则返回 0。
示例 1：输入：grid = [[1,1,1],[1,0,1],[1,1,1]] 输出：9
示例 2：输入：grid = [[1,1,0,0]] 输出：1
提示：1 <= grid.length <= 100
1 <= grid[0].length <= 100
grid[i][j] 为0或1
```

- 解题思路

```
func largest1BorderedSquare(grid [][]int) int {
        res := 0
        arr := [100][100][2]int{}
        n, m := len(grid), len(grid[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if grid[i][j] == 0 {
```

(续下页)

(接上页)

```
                                continue
                        }
                        if i == 0 {
                                arr[i][j][0] = 1
                        } else {
                                arr[i][j][0] = arr[i-1][j][0] + 1 //␣
↪当前坐标上边1的长度
                        }
                        if j == 0 {
                                arr[i][j][1] = 1
                        } else {
                                arr[i][j][1] = arr[i][j-1][1] + 1 //␣
↪当前坐标左边1的长度
                        }
                        minValue := min(arr[i][j][0], arr[i][j][1]) //␣
↪左边、上边连续1选短的
                        for k := minValue; k > res; k-- {
                                if arr[i][j-k+1][0] >= k && arr[i-k+1][j][1] >= k { //
↪ 判断另外2条边
                                        res = k
                                        break
                                }
                        }
                }
        }
        return res * res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

35.13 1140. 石子游戏 II(2)

- 题目

```
亚历克斯和李继续他们的石子游戏。许多堆石子排成一行，每堆都有正整数颗石子piles[i]。
游戏以谁手中的石子最多来决出胜负。
亚历克斯和李轮流进行，亚历克斯先开始。最初，M = 1。
在每个玩家的回合中，该玩家可以拿走剩下的前X堆的所有石子，其中1 <= X <= 2M。
然后，令M = max(M, X)。
游戏一直持续到所有石子都被拿走。
假设亚历克斯和李都发挥出最佳水平，返回亚历克斯可以得到的最大数量的石头。
示例：输入：piles = [2,7,9,4,4] 输出：10
解释：如果亚历克斯在开始时拿走一堆石子，李拿走两堆，接着亚历克斯也拿走两堆。
在这种情况下，亚历克斯可以拿到 2 + 4 + 4 = 10 颗石子。
如果亚历克斯在开始时拿走两堆石子，那么李就可以拿走剩下全部三堆石子。
在这种情况下，亚历克斯可以拿到 2 + 7 = 9 颗石子。
所以我们返回更大的 10。
提示：1 <= piles.length <= 100
1 <= piles[i]<= 10 ^ 4
```

- 解题思路

```
func stoneGameII(piles []int) int {
        n := len(piles)
        dp := make([][]int, n+1) // dp[i][j]=>有piles[i:], M=j的情况下得分
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
        }
        sum := 0
        for i := n - 1; i >= 0; i-- {
                sum = sum + piles[i]
                for M := 1; M <= n; M++ {
                        if i+2*M >= n { // 可以全部拿走
                                dp[i][M] = sum
                        } else {
                                for j := 1; j <= 2*M; j++ { //␣
↪尝试不同拿法，dp[i+j][max(j, M)]其中M=max(j,M)
                                        dp[i][M] = max(dp[i][M], sum-dp[i+j][max(j,␣
↪M)])
                                }
                        }
                }
        }
        return dp[0][1]
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var dp [][]int

func stoneGameII(piles []int) int {
        n := len(piles)
        dp = make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
        }
        for i := n - 2; i >= 0; i-- {
                piles[i] = piles[i] + piles[i+1]
        }
        return dfs(piles, 0, 1)
}

func dfs(piles []int, index, M int) int {
        if index >= len(piles) {
                return 0
        }
        if index+2*M >= len(piles) { // 可以全部拿走
                return piles[index]
        }
        if dp[index][M] > 0 {
                return dp[index][M]
        }
        res := 0
        for i := 1; i <= 2*M; i++ { // 尝试不同拿法
                res = max(res, piles[index]-dfs(piles, index+i, max(M, i)))
        }
        dp[index][M] = res
        return dp[index][M]
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}
```

35.14 1143. 最长公共子序列 (3)

- 题目

```
给定两个字符串 text1 和 text2，返回这两个字符串的最长公共子序列的长度。
一个字符串的 子序列 是指这样一个新的字符串：
它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。
例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。
两个字符串的「公共子序列」是这两个字符串所共同拥有的子序列。
若这两个字符串没有公共子序列，则返回 0。
示例 1:输入：text1 = "abcde", text2 = "ace"  输出：3
解释：最长公共子序列是 "ace"，它的长度为 3。
示例 2:输入：text1 = "abc", text2 = "abc" 输出：3
解释：最长公共子序列是 "abc"，它的长度为 3。
示例 3:输入：text1 = "abc", text2 = "def" 输出：0
解释：两个字符串没有公共子序列，返回 0。
提示：
    1 <= text1.length <= 1000
    1 <= text2.length <= 1000
    输入的字符串只含有小写英文字符。
```

- 解题思路

```
func longestCommonSubsequence(text1 string, text2 string) int {
        n, m := len(text1), len(text2)
        dp := make([][]int, n+1)
        for i := 0; i < n+1; i++ {
                dp[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if text1[i-1] == text2[j-1] {
                                dp[i][j] = dp[i-1][j-1] + 1
                        } else {
                                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
                        }
```

(续下页)

(接上页)

```
                }
        }
        return dp[n][m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestCommonSubsequence(text1 string, text2 string) int {
        n, m := len(text1), len(text2)
        prev := make([]int, m+1)
        cur := make([]int, m+1)
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if text1[i-1] == text2[j-1] {
                                cur[j] = prev[j-1] + 1
                        } else {
                                cur[j] = max(prev[j], cur[j-1])
                        }
                }
                copy(prev, cur)
        }
        return cur[m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func longestCommonSubsequence(text1 string, text2 string) int {
        n, m := len(text1), len(text2)
        cur := make([]int, m+1)
        for i := 1; i <= n; i++ {
                pre := cur[0]
```

(续下页)

(接上页)

```
                for j := 1; j <= m; j++ {
                        temp := cur[j]
                        if text1[i-1] == text2[j-1] {
                                cur[j] = pre + 1
                        } else {
                                cur[j] = max(cur[j], cur[j-1])
                        }
                        pre = temp
                }
        }
        return cur[m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

35.15 1144. 递减元素使数组呈锯齿状 (2)

- 题目

```
给你一个整数数组nums，每次 操作会从中选择一个元素并 将该元素的值减少1。
如果符合下列情况之一，则数组A就是 锯齿数组：
每个偶数索引对应的元素都大于相邻的元素，即A[0] > A[1] < A[2] > A[3] < A[4] > ...
或者，每个奇数索引对应的元素都大于相邻的元素，即A[0] < A[1] > A[2] < A[3] > A[4] < ...
返回将数组nums转换为锯齿数组所需的最小操作次数。
示例 1：输入：nums = [1,2,3] 输出：2
解释：我们可以把 2 递减到 0，或把 3 递减到 1。
示例 2：输入：nums = [9,6,1,6,2] 输出：4
提示：1 <= nums.length <= 1000
1 <= nums[i] <= 1000
```

- 解题思路

```
func movesToMakeZigzag(nums []int) int {
        n := len(nums)
        a, b := 0, 0
        for i := 0; i < n; i++ {
                if i%2 == 0 { // 偶数下标小，如果大于左右两边数，需要减去
```

(续下页)

(接上页)

```
                        left, right := 0, 0
                        if i > 0 && nums[i] >= nums[i-1] {
                                left = nums[i] - nums[i-1] + 1
                        }
                        if i < n-1 && nums[i] >= nums[i+1] {
                                right = nums[i] - nums[i+1] + 1
                        }
                        a = a + max(left, right)
                } else { // 奇数下标小，如果大于左右两边数，需要减去
                        left, right := 0, 0
                        if nums[i] >= nums[i-1] {
                                left = nums[i] - nums[i-1] + 1
                        }
                        if i < n-1 && nums[i] >= nums[i+1] {
                                right = nums[i] - nums[i+1] + 1
                        }
                        b = b + max(left, right)
                }
        }
        return min(a, b)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func movesToMakeZigzag(nums []int) int {
        n := len(nums)
        a, b := 0, 0
        for i := 0; i < n; i = i + 2 { // 偶数下标小，如果大于左右两边数，需要减去
                left, right := 0, 0
                if i > 0 && nums[i] >= nums[i-1] {
```

(续下页)

(接上页)

```
                        left = nums[i] - nums[i-1] + 1
                }
                if i < n-1 && nums[i] >= nums[i+1] {
                        right = nums[i] - nums[i+1] + 1
                }
                a = a + max(left, right)
        }
        for i := 1; i < n; i = i + 2 { // 奇数下标小，如果大于左右两边数，需要减去
                left, right := 0, 0
                if nums[i] >= nums[i-1] {
                        left = nums[i] - nums[i-1] + 1
                }
                if i < n-1 && nums[i] >= nums[i+1] {
                        right = nums[i] - nums[i+1] + 1
                }
                b = b + max(left, right)
        }
        return min(a, b)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

35.16 1145. 二叉树着色游戏 (2)

- 题目

```
有两位极客玩家参与了一场「二叉树着色」的游戏。游戏中，
给出二叉树的根节点root，树上总共有 n 个节点，且 n 为奇数，其中每个节点上的值从1␣
↪到n各不相同。
游戏从「一号」玩家开始（「一号」玩家为红色，「二号」玩家为蓝色），最开始时，
```

(续下页)

(接上页)

```
「一号」玩家从 [1, n]中取一个值x (1 <= x <= n) ;
「二号」玩家也从[1, n]中取一个值y (1 <= y <= n) 且y != x。
「一号」玩家给值为x的节点染上红色，而「二号」玩家给值为y的节点染上蓝色。
之后两位玩家轮流进行操作，每一回合，玩家选择一个他之前涂好颜色的节点，
将所选节点一个 未着色 的邻节点（即左右子节点、或父节点）进行染色。
如果当前玩家无法找到这样的节点来染色时，他的回合就会被跳过。
若两个玩家都没有可以染色的节点时，游戏结束。着色节点最多的那位玩家获得胜利 。
现在，假设你是「二号」玩家，根据所给出的输入，假如存在一个y值可以确保你赢得这场游戏，则返回true;
若无法获胜，就请返回 false。
示例：输入：root = [1,2,3,4,5,6,7,8,9,10,11], n = 11, x = 3 输出：True
解释：第二个玩家可以选择值为 2 的节点。
提示：二叉树的根节点为root，树上由 n 个节点，节点上的值从 1 到 n 各不相同。
n 为奇数。
1 <= x <= n<= 100
```

- 解题思路

```
var targetNode *TreeNode

func btreeGameWinningMove(root *TreeNode, n int, x int) bool {
        dfsTarget(root, x)
        // 统计根节点、目标节点左子树、目标节点右子树
        return dfs(root) > n/2 || dfs(targetNode.Left) > n/2 || dfs(targetNode.Right)␣
↪> n/2
}

func dfsTarget(root *TreeNode, target int) {
        if root != nil {
                if root.Val == target {
                        targetNode = root
                        return
                }
                dfsTarget(root.Left, target)
                dfsTarget(root.Right, target)
        }
}

func dfs(root *TreeNode) int {
        if root == nil || root == targetNode {
                return 0
        }
        return 1 + dfs(root.Left) + dfs(root.Right)
}
```

(续下页)

(接上页)

```
# 2
var leftSum, rightSum int

func btreeGameWinningMove(root *TreeNode, n int, x int) bool {
        leftSum = 0
        rightSum = 0
        total := dfs(root, x)
        return leftSum > n/2 || rightSum > n/2 || (total-1-leftSum-rightSum) > n/2
}

func dfs(root *TreeNode, x int) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left, x)
        right := dfs(root.Right, x)
        if root.Val == x {
                leftSum = left
                rightSum = right
        }
        return 1 + left + right
}
```

35.17 1146. 快照数组 (3)

- 题目

```
实现支持下列接口的「快照数组」-SnapshotArray:
SnapshotArray(int length)- 初始化一个与指定长度相等的 类数组␣
↪的数据结构。初始时，每个元素都等于0。
void set(index, val)- 会将指定索引index处的元素设置为val。
int snap()- 获取该数组的快照，并返回快照的编号snap_
↪id（快照号是调用snap()的总次数减去1）。
int get(index, snap_id)- 根据指定的snap_id选择快照，并返回该快照指定索引 index的值。
示例：输入：["SnapshotArray","set","snap","set","get"]
     [[3],[0,5],[],[0,6],[0,0]]
输出：[null,null,0,null,5]
解释：SnapshotArray snapshotArr = new SnapshotArray(3); // 初始化一个长度为 3␣
↪的快照数组
snapshotArr.set(0,5);  // 令 array[0] = 5
snapshotArr.snap();  // 获取快照，返回 snap_id = 0
snapshotArr.set(0,6);
```

(续下页)

(接上页)

```
snapshotArr.get(0,0);  // 获取 snap_id = 0 的快照中 array[0] 的值，返回 5
提示：1 <= length<= 50000
题目最多进行50000 次set，snap，和get的调用 。
0 <= index<length
0 <=snap_id <我们调用snap()的总次数
0 <=val <= 10^9
```

- 解题思路

```
type SnapshotArray struct {
        id  int
        arr [][]Snap
}

type Snap struct {
        id    int
        value int
}

func Constructor(length int) SnapshotArray {
        return SnapshotArray{
                id:  0,
                arr: make([][]Snap, length),
        }
}

func (this *SnapshotArray) Set(index int, val int) {
        // 保存的是index的操作记录
        this.arr[index] = append(this.arr[index], Snap{
                id:    this.id,
                value: val,
        })
}

func (this *SnapshotArray) Snap() int {
        id := this.id
        this.id++
        return id
}

func (this *SnapshotArray) Get(index int, snap_id int) int {
        n := len(this.arr[index])
        i := 0
        // 根据index的历史记录，找到最后1个id为snap_id的记录
```

(续下页)

(接上页)

```
        // 其中id为snap_id的记录可能有多个
        for i < n && this.arr[index][i].id <= snap_id {
                i++
        }
        if i == 0 {
                return 0
        }
        i--
        return this.arr[index][i].value
}

# 2
type SnapshotArray struct {
        id  int
        arr []map[int]int
}

func Constructor(length int) SnapshotArray {
        return SnapshotArray{
                id:  0,
                arr: make([]map[int]int, length),
        }
}

func (this *SnapshotArray) Set(index int, val int) {
        if this.arr[index] == nil {
                this.arr[index] = make(map[int]int)
        }
        this.arr[index][this.id] = val
}

func (this *SnapshotArray) Snap() int {
        id := this.id
        this.id++
        return id
}

func (this *SnapshotArray) Get(index int, snap_id int) int {
        if this.arr[index] == nil {
                return 0
        }
        for ; snap_id >= 0; snap_id-- {
                if v, ok := this.arr[index][snap_id]; ok {
```

(续下页)

(接上页)

```
                        return v
                }
        }
        return 0
}

# 3
type SnapshotArray struct {
        id  int
        arr [][]Snap
}

type Snap struct {
        id    int
        value int
}

func Constructor(length int) SnapshotArray {
        nums := make([][]Snap, length)
        for i := 0; i < length; i++ {
                nums[i] = []Snap{{ // 填充0
                        id:    0,
                        value: 0,
                }}
        }
        return SnapshotArray{
                id:  0,
                arr: nums,
        }
}

func (this *SnapshotArray) Set(index int, val int) {
        n := len(this.arr[index])
        if this.arr[index][n-1].id == this.id {
                this.arr[index][n-1].value = val
                return
        }
        this.arr[index] = append(this.arr[index], Snap{
                id:    this.id,
                value: val,
        })
}
```

(续下页)

(接上页)

```
func (this *SnapshotArray) Snap() int {
        id := this.id
        this.id++
        return id
}

func (this *SnapshotArray) Get(index int, snap_id int) int {
        n := len(this.arr[index])
        arr := this.arr[index]
        left, right := 0, n-1
        for left < right {
                mid := left + (right-left)/2
                if snap_id <= arr[mid].id {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        if left == n || arr[left].id > snap_id {
                return arr[left-1].value
        }
        return arr[left].value
}
```

35.18 1155. 掷骰子的 N 种方法 (2)

- 题目

```
这里有d个一样的骰子，每个骰子上都有f个面，分别标号为1, 2, ..., f。
我们约定：掷骰子的得到总点数为各骰子面朝上的数字的总和。
如果需要掷出的总点数为target，请你计算出有多少种不同的组合情况（所有的组合情况总共有␣
↪f^d 种），
模10^9 + 7后返回。
示例 1：输入：d = 1, f = 6, target = 3 输出：1
示例 2：输入：d = 2, f = 6, target = 7 输出：6
示例 3：输入：d = 2, f = 5, target = 10 输出：1
示例 4：输入：d = 1, f = 2, target = 3 输出：0
示例 5：输入：d = 30, f = 30, target = 500 输出：222616187
提示：1 <= d, f <= 30
1 <= target <= 1000
```

- 解题思路

```
func numRollsToTarget(d int, f int, target int) int {
        dp := make([]int, target+1)
        dp[0] = 1
        for i := 0; i < d; i++ {
                next := make([]int, target+1)
                for j := 1; j <= f; j++ {
                        for k := 0; k <= target-j; k++ {
                                next[k+j] = (next[k+j] + dp[k]) % 1000000007
                        }
                }
                dp = next
        }
        return dp[target]
}

# 2
func numRollsToTarget(d int, f int, target int) int {
        dp := make([][]int, d+1)
        for i := 0; i <= d; i++ {
                dp[i] = make([]int, target+1)
        }
        dp[0][0] = 1
        for i := 1; i <= d; i++ {
                for j := i; j <= target; j++ {
                        for k := 1; k <= f; k++ {
                                if j >= k {
                                        dp[i][j] = (dp[i][j] + dp[i-1][j-k]) %␣
↪1000000007
                                }
                        }
                }
        }
        return dp[d][target]
}
```

35.19 1156. 单字符重复子串的最大长度 (1)

- 题目

```
如果字符串中的所有字符都相同，那么这个字符串是单字符重复的字符串。
给你一个字符串text，你只能交换其中两个字符一次或者什么都不做，然后得到一些单字符重复的子串。返回其中
示例 1：输入：text = "ababa" 输出：3
```

(续下页)

(接上页)

```
示例 2: 输入: text = "aaabaaa" 输出: 6
示例 3: 输入: text = "aaabbaaa" 输出: 4
示例 4: 输入: text = "aaaaa" 输出: 5
示例 5: 输入: text = "abcdef" 输出: 1
提示: 1 <= text.length <= 20000
text 仅由小写英文字母组成。
```

- 解题思路

```
func maxRepOpt1(text string) int {
        res := 0
        n := len(text)
        arr := [26]int{}
        for i := 0; i < n; i++ {
                v := int(text[i] - 'a')
                arr[v]++ // 统计每个字母出现的次数
        }
        for i := 0; i < n; {
                v := int(text[i] - 'a')
                countA := 0 // 统计相同的个数
                for i+countA < n && text[i] == text[i+countA] {
                        countA++
                }
                j := i + countA + 1
                countB := 0 // 统计相隔1个不同字符往后连续的次数
                for j+countB < n && text[i] == text[j+countB] {
                        countB++
                }
                // 2种情况
                // 1、a...axa...ay..a 可以拿1个来补齐:+1
                //␣
↪2、没有多余的补齐，可以拿第二段右侧最后一个点或者第一段左侧第一个点来补:+0
                total := min(countA+countB+1, arr[v]) // 取较少的
                res = max(res, total)                 // 更新结果
                i = i + countA                        // 窗口后移

        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

35.20 1161. 最大层内元素和 (2)

- 题目

```
给你一个二叉树的根节点root。设根节点位于二叉树的第 1 层，而根节点的子节点位于第 2层，依此类推。
请你找出层内元素之和 最大 的那几层（可能只有一层）的层号，并返回其中最小 的那个。
示例 1：输入：root = [1,7,0,7,-8,null,null] 输出：2
解释：第 1 层各元素之和为 1，
第 2 层各元素之和为 7 + 0 = 7，
第 3 层各元素之和为 7 + -8 = -1，
所以我们返回第 2 层的层号，它的层内元素之和最大。
示例 2：输入：root = [989,null,10250,98693,-89388,null,null,null,-32127] 输出：2
提示：树中的节点数介于1和10^4之间
-10^5 <= node.val <= 10^5
```

- 解题思路

```
func maxLevelSum(root *TreeNode) int {
        res := 0
        maxValue := math.MinInt32
        if root == nil {
                return 0
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        level := 1
        for len(queue) > 0 {
                length := len(queue)
                sum := 0
                for i := 0; i < length; i++ {
                        node := queue[i]
```

(续下页)

(接上页)

```
                        sum = sum + node.Val
                        if node.Left != nil {
                                queue = append(queue, node.Left)
                        }
                        if node.Right != nil {
                                queue = append(queue, node.Right)
                        }
                }
                if sum > maxValue {
                        maxValue = sum
                        res = level
                }
                queue = queue[length:]
                level++
        }
        return res
}

# 2
var arr [][]int

func maxLevelSum(root *TreeNode) int {
        arr = make([][]int, 0)
        if root == nil {
                return 0
        }
        dfs(root, 0)
        res := 0
        maxValue := math.MinInt32
        for i := 0; i < len(arr); i++ {
                sum := 0
                for j := 0; j < len(arr[i]); j++ {
                        sum = sum + arr[i][j]
                }
                if sum > maxValue {
                        maxValue = sum
                        res = i + 1
                }
        }
        return res
}

func dfs(root *TreeNode, level int) {
```

(续下页)

(接上页)

```
        if root == nil {
                return
        }
        if level == len(arr) {
                arr = append(arr, []int{})
        }
        arr[level] = append(arr[level], root.Val)
        dfs(root.Left, level+1)
        dfs(root.Right, level+1)
}
```

35.21 1162. 地图分析 (2)

- 题目

```
你现在手里有一份大小为 N x N 的 网格 grid，上面的每个 单元格 都用 0 和 1 标记好了。
其中 0 代表海洋，1␣
↪代表陆地，请你找出一个海洋单元格，这个海洋单元格到离它最近的陆地单元格的距离是最大的。
我们这里说的距离是「曼哈顿距离」（ Manhattan Distance）：
(x0, y0) 和 (x1, y1) 这两个单元格之间的距离是 |x0 - x1| + |y0 - y1| 。
如果网格上只有陆地或者海洋，请返回 -1。
示例 1：输入：[[1,0,1],[0,0,0],[1,0,1]] 输出：2
解释：海洋单元格 (1, 1) 和所有陆地单元格之间的距离都达到最大，最大距离为 2。
示例 2：输入：[[1,0,0],[0,0,0],[0,0,0]] 输出：4
解释：  海洋单元格 (2, 2) 和所有陆地单元格之间的距离都达到最大，最大距离为 4。
提示：  1 <= grid.length == grid[0].length <= 100
    grid[i][j] 不是 0 就是 1。
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func maxDistance(grid [][]int) int {
        queue := make([][2]int, 0)
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 1 {
                                queue = append(queue, [2]int{i, j})
                        }
                }
        }
```

(续下页)

(接上页)

```
        if len(queue) == 0 || len(queue) == len(grid)*len(grid[0]) {
                return -1
        }
        res := -1
        for len(queue) > 0 {
                res++
                length := len(queue)
                for i := 0; i < length; i++ {
                        x1 := queue[i][0]
                        y1 := queue[i][1]
                        for i := 0; i < 4; i++ {
                                x := x1 + dx[i]
                                y := y1 + dy[i]
                                if 0 <= x && x < len(grid) && 0 <= y && y <␣
↪len(grid[0]) && grid[x][y] == 0 {
                                        queue = append(queue, [2]int{x, y})
                                        grid[x][y] = 2
                                }
                        }
                }
                queue = queue[length:]
        }
        return res
}

# 2
func maxDistance(grid [][]int) int {
        if len(grid) == 0 || len(grid[0]) == 0 {
                return -1
        }
        n := len(grid)
        m := len(grid[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if grid[i][j] == 1 {
                                dp[i][j] = 0
                        } else {
                                dp[i][j] = math.MaxInt32
                        }
```

(续下页)

(接上页)

```
                }
        }
        // 从上往下
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if grid[i][j] == 1 {
                                continue
                        }
                        if i >= 1 {
                                dp[i][j] = min(dp[i][j], dp[i-1][j]+1)
                        }
                        if j >= 1 {
                                dp[i][j] = min(dp[i][j], dp[i][j-1]+1)
                        }
                }
        }
        // 从下往上
        for i := n - 1; i >= 0; i-- {
                for j := m - 1; j >= 0; j-- {
                        if grid[i][j] == 1 {
                                continue
                        }
                        if i < n-1 {
                                dp[i][j] = min(dp[i][j], dp[i+1][j]+1)
                        }
                        if j < m-1 {
                                dp[i][j] = min(dp[i][j], dp[i][j+1]+1)
                        }
                }
        }
        res := -1
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if grid[i][j] == 1 {
                                continue
                        }
                        res = max(res, dp[i][j])
                }
        }
        if res == math.MaxInt32 {
                return -1
        }
        return res
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

35.22 1169. 查询无效交易 (2)

- 题目

```
如果出现下述两种情况，交易 可能无效：
交易金额超过 ¥1000
或者，它和另一个城市中同名的另一笔交易相隔不超过 60 分钟（包含 60 分钟整）
每个交易字符串transactions[i]由一些用逗号分隔的值组成，这些值分别表示交易的名称，时间（以分钟计），金
给你一份交易清单transactions，返回可能无效的交易列表。你可以按任何顺序返回答案。
示例 1：输入：transactions = ["alice,20,800,mtv","alice,50,100,beijing"]
输出：["alice,20,800,mtv","alice,50,100,beijing"]
解释：第一笔交易是无效的，因为第二笔交易和它间隔不超过 60␣
↪分钟、名称相同且发生在不同的城市。同样，第二笔交易也是无效的。
示例 2：输入：transactions = ["alice,20,800,mtv","alice,50,1200,mtv"] 输出：["alice,
↪50,1200,mtv"]
示例 3：输入：transactions = ["alice,20,800,mtv","bob,50,1200,mtv"] 输出：["bob,50,
↪1200,mtv"]
提示：transactions.length <= 1000
每笔交易transactions[i]按"{name},{time},{amount},{city}"的格式进行记录
每个交易名称{name}和城市{city}都由小写英文字母组成，长度在1到10之间
每个交易时间{time}由一些数字组成，表示一个0到1000之间的整数
每笔交易金额{amount}由一些数字组成，表示一个0 到2000之间的整数
```

- 解题思路

```
type Node struct {
        Name   string
        Time   int
        Amount int
        City   string
        Index  int
        Flag   bool
}

func invalidTransactions(transactions []string) []string {
        res := make([]string, 0)
        m := make(map[string][]Node)
        n := len(transactions)
        for i := 0; i < n; i++ {
                arr := strings.Split(transactions[i], ",")
                name, city := arr[0], arr[3]
                t, _ := strconv.Atoi(arr[1])
                amount, _ := strconv.Atoi(arr[2])
                m[name] = append(m[name], Node{Name: name, Time: t,
                        Amount: amount, City: city, Index: i,
                })
        }
        for _, v := range m {
                sort.Slice(v, func(i, j int) bool {
                        return v[i].Time < v[j].Time
                })
                for i := 0; i < len(v); i++ {
                        if v[i].Amount > 1000 {
                                v[i].Flag = true
                        }
                        for j := i + 1; j < len(v); j++ {
                                if v[j].Time-v[i].Time <= 60 {
                                        if v[i].City != v[j].City {
                                                v[i].Flag = true
                                                v[j].Flag = true
                                        }
                                } else {
                                        break
                                }
                        }
                }
        }
        for _, v := range m {
                for i := 0; i < len(v); i++ {
```

(续下页)

(接上页)

```
                        if v[i].Flag == true {
                                res = append(res, transactions[v[i].Index])
                        }
                }
        }
        return res
}

# 2
type Node struct {
        Name   string
        Time   int
        Amount int
        City   string
}

func invalidTransactions(transactions []string) []string {
        res := make([]string, 0)
        arr := make([]Node, 0)
        n := len(transactions)
        for i := 0; i < n; i++ {
                temp := strings.Split(transactions[i], ",")
                name, city := temp[0], temp[3]
                t, _ := strconv.Atoi(temp[1])
                amount, _ := strconv.Atoi(temp[2])
                arr = append(arr, Node{Name: name, Time: t, Amount: amount, City:␣
↪city})
        }
        for i := 0; i < n; i++ {
                if arr[i].Amount > 1000 {
                        res = append(res, transactions[i])
                        continue
                }
                for j := 0; j < n; j++ {
                        if i == j {
                                continue
                        }
                        if arr[i].Name == arr[j].Name && arr[i].City != arr[j].City &&
                                abs(arr[i].Time-arr[j].Time) <= 60 {
                                res = append(res, transactions[i])
                                break
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

35.23 1171. 从链表中删去总和值为零的连续节点 (4)

- 题目

```
给你一个链表的头节点head，请你编写代码，反复删去链表中由 总和值为 0␣
↪的连续节点组成的序列，
直到不存在这样的序列为止。
删除完毕后，请你返回最终结果链表的头节点。
你可以返回任何满足题目要求的答案。
（注意，下面示例中的所有序列，都是对ListNode对象序列化的表示。）
示例 1：输入：head = [1,2,-3,3,1] 输出：[3,1]
提示：答案 [1,2,1] 也是正确的。
示例 2：输入：head = [1,2,3,-3,4] 输出：[1,2,4]
示例 3：输入：head = [1,2,3,-3,-2] 输出：[1]
提示：给你的链表中可能有 1 到1000个节点。
对于链表中的每个节点，节点的值：-1000 <= node.val <= 1000.
```

- 解题思路

```
func removeZeroSumSublists(head *ListNode) *ListNode {
        m := make(map[int]*ListNode)
        res := head
        sum := 0
        for cur := head; cur != nil; cur = cur.Next {
                sum = sum + cur.Val
                if sum == 0 { // 当前都为0
                        res = cur.Next
                        m = make(map[int]*ListNode)
                        continue
                }
                if _, ok := m[sum]; ok == false {
```

(续下页)

(接上页)

```
                        m[sum] = cur
                        continue
                }
                // 出现重复
                first := m[sum]
                tempSum := sum
                for temp := first.Next; temp != cur; temp = temp.Next {
                        tempSum = tempSum + temp.Val
                        delete(m, tempSum)
                }
                first.Next = cur.Next
        }
        return res
}

# 2
func removeZeroSumSublists(head *ListNode) *ListNode {
        res := &ListNode{Next: head}
        for cur := res; cur != nil; {
                sum := 0
                for temp := cur.Next; temp != nil; temp = temp.Next {
                        sum = sum + temp.Val
                        if sum == 0 {
                                cur.Next = temp.Next // 调整指针
                        }
                }
                cur = cur.Next
        }
        return res.Next
}

# 3
func removeZeroSumSublists(head *ListNode) *ListNode {
        res := &ListNode{Next: head}
        m := make(map[int]*ListNode)
        sum := 0
        for cur := res; cur != nil; cur = cur.Next {
                sum = sum + cur.Val
                m[sum] = cur
        }
        sum = 0
        for cur := res; cur != nil; cur = cur.Next {
                sum = sum + cur.Val
```

(续下页)

(接上页)

```
                cur.Next = m[sum].Next
        }
        return res.Next
}

# 4
func removeZeroSumSublists(head *ListNode) *ListNode {
        if head == nil {
                return nil
        }
        sum := 0
        for cur := head; cur != nil; cur = cur.Next {
                sum = sum + cur.Val
                if sum == 0 {
                        return removeZeroSumSublists(cur.Next)
                }
        }
        head.Next = removeZeroSumSublists(head.Next)
        return head
}
```

35.24 1177. 构建回文串检测 (1)

- 题目

```
给你一个字符串s，请你对s的子串进行检测。
每次检测，待检子串都可以表示为queries[i] = [left, right, k]。
我们可以 重新排列 子串s[left], ..., s[right]，并从中选择 最多␣
↪k项替换成任何小写英文字母。
如果在上述检测过程中，子串可以变成回文形式的字符串，那么检测结果为true，否则结果为false。
返回答案数组answer[]，其中answer[i]是第i个待检子串queries[i]的检测结果。
注意：在替换时，子串中的每个字母都必须作为 独立的 项进行计数，
也就是说，如果s[left..right] = "aaa"且k = 2，我们只能替换其中的两个字母。
（另外，任何检测都不会修改原始字符串 s，可以认为每次检测都是独立的）
示例：输入：s = "abcda", queries = [[3,3,0],[1,2,0],[0,3,1],[0,3,2],[0,4,1]]␣
↪输出：[true,false,false,true,true]
解释：queries[0] : 子串 = "d"，回文。
queries[1] :子串 = "bc"，不是回文。
queries[2] :子串 = "abcd"，只替换 1 个字符是变不成回文串的。
queries[3] :子串 = "abcd"，可以变成回文的 "abba"。 也可以变成 "baab"，先重新排序变成
↪"bacd"，然后把 "cd" 替换为 "ab"。
queries[4] :子串 = "abcda"，可以变成回文的 "abcba"。
```

(续下页)

(接上页)

```
提示：1 <= s.length,queries.length<= 10^5
0 <= queries[i][0] <= queries[i][1] <s.length
0 <= queries[i][2] <= s.length
s 中只有小写英文字母
```

- 解题思路

```
func canMakePaliQueries(s string, queries [][]int) []bool {
        n := len(s)
        arr := make([]int, n+1)
        cur := 0
        for i := 0; i < n; i++ {
                value := int(s[i] - 'a')
                cur = cur ^ (1 << value)
                arr[i+1] = cur
        }
        res := make([]bool, len(queries))
        for i := 0; i < len(queries); i++ {
                a, b, c := queries[i][0], queries[i][1], queries[i][2]
                status := arr[b+1] ^ arr[a]
                if bits.OnesCount(uint(status)) <= 2*c+1 { //␣
↪奇数次字母的个数，重新排列后最多允许2*c+1个
                        res[i] = true
                }
        }
        return res
}
```

35.25 1186. 删除一次得到子数组最大和 (2)

- 题目

```
给你一个整数数组，返回它的某个非空␣
↪子数组（连续元素）在执行一次可选的删除操作后，所能得到的最大元素总和。
换句话说，你可以从原数组中选出一个子数组，并可以决定要不要从中删除一个元素（只能删一次哦），
（删除后）子数组中至少应当有一个元素，然后该子数组（剩下）的元素总和是所有子数组之中最大的。
注意，删除一个元素后，子数组 不能为空。
请看示例：
示例 1：输入：arr = [1,-2,0,3] 输出：4
解释：我们可以选出 [1, -2, 0, 3]，然后删掉 -2，这样得到 [1, 0, 3]，和最大。
示例 2：输入：arr = [1,-2,-2,3] 输出：3
解释：我们直接选出 [3]，这就是最大和。
```

(续下页)

(接上页)

```
示例 3: 输入: arr = [-1,-1,-1,-1] 输出: -1
解释: 最后得到的子数组不能为空, 所以我们不能选择 [-1] 并从中删去 -1 来得到 0。
     我们应该直接选择 [-1], 或者选择 [-1, -1] 再从中删去一个 -1。
提示: 1 <= arr.length <= 10^5
-10^4 <= arr[i] <= 10^4
```

- 解题思路

```
func maximumSum(arr []int) int {
        n := len(arr)
        dp := make([][2]int, n)
        dp[0][0] = arr[0] // 不删
        dp[0][1] = 0      // 删除
        res := dp[0][0]   // 长度为1, 不删除
        for i := 1; i < n; i++ {
                dp[i][0] = max(dp[i-1][0]+arr[i], arr[i])
                dp[i][1] = max(dp[i-1][1]+arr[i], dp[i-1][0])
                res = max(res, max(dp[i][0], dp[i][1]))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maximumSum(arr []int) int {
        n := len(arr)
        a := arr[0] // 不删
        b := 0      // 删除
        res := a    // 长度为1, 不删除
        for i := 1; i < n; i++ {
                a, b = max(a+arr[i], arr[i]), max(b+arr[i], a)
                res = max(res, max(a, b))
        }
        return res
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}
```

35.26 1190. 反转每对括号间的子串 (2)

- 题目

```
给出一个字符串s（仅含有小写英文字母和括号）。
请你按照从括号内到外的顺序，逐层反转每对匹配括号中的字符串，并返回最终的结果。
注意，您的结果中 不应 包含任何括号。
示例 1：输入：s = "(abcd)" 输出："dcba"
示例 2：输入：s = "(u(love)i)" 输出："iloveu"
示例 3：输入：s = "(ed(et(oc))el)" 输出："leetcode"
示例 4：输入：s = "a(bcdefghijkl(mno)p)q" 输出："apmnolkjihgfedcbq"
提示：0 <= s.length <= 2000
s 中只有小写英文字母和括号
我们确保所有括号都是成对出现的
```

- 解题思路

```
func reverseParentheses(s string) string {
        arr := []byte(s)
        stack := make([]int, 0)
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        stack = append(stack, i)
                } else if s[i] == ')' {
                        reverse(arr, stack[len(stack)-1], i)
                        stack = stack[:len(stack)-1]
                }
        }
        s = string(arr)
        s = strings.ReplaceAll(s, "(", "")
        s = strings.ReplaceAll(s, ")", "")
        return s
}

func reverse(arr []byte, i, j int) {
        for i < j {
                arr[i], arr[j] = arr[j], arr[i]
```

(续下页)

(接上页)

```
                i++
                j--
        }
}

# 2
func reverseParentheses(s string) string {
        stack := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        stack = append(stack, i)
                } else if s[i] == ')' {
                        last := stack[len(stack)-1]
                        m[i] = last
                        m[last] = i
                        stack = stack[:len(stack)-1]
                }
        }
        dir := 1
        res := make([]byte, 0)
        for i := 0; i < len(s); i = i + dir {
                if s[i] == '(' || s[i] == ')' {
                        i = m[i]   // 从反方向遍历
                        dir = -dir // 变换方向, +1/-1
                } else {
                        res = append(res, s[i])
                }
        }
        return string(res)
}
```

35.27 1191.K 次串联后最大子数组之和 (1)

- 题目

```
给你一个整数数组 arr 和一个整数 k。
首先，我们要对该数组进行修改，即把原数组 arr 重复 k 次。
    举个例子，如果 arr = [1, 2] 且 k = 3，那么修改后的数组就是 [1, 2, 1, 2, 1, 2]。
然后，请你返回修改后的数组中的最大的子数组之和。
注意，子数组长度可以是 0，在这种情况下它的总和也是 0。
由于 结果可能会很大，所以需要 模（mod） 10^9 + 7 后再返回。
```

(续下页)

(接上页)

```
示例 1: 输入: arr = [1,2], k = 3 输出: 9
示例 2: 输入: arr = [1,-2,1], k = 5 输出: 2
示例 3: 输入: arr = [-1,-2], k = 7 输出: 0
提示:
    1 <= arr.length <= 10^5
    1 <= k <= 10^5
    -10^4 <= arr[i] <= 10^4
```

- 解题思路

```
func kConcatenationMaxSum(arr []int, k int) int {
        cur := 0
        sum := 0
        res := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i]
                cur = max(cur+arr[i], arr[i])
                res = max(res, cur)
        }
        for i := 0; i < len(arr); i++ {
                cur = max(cur+arr[i], arr[i])
                res = max(res, cur)
        }
        if sum <= 0 {
                return res % 1000000007
        }
        return (res + (k-2)*sum) % 1000000007
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

CHAPTER 36

1101-1200-Hard

36.1 1147. 段式回文 (2)

- 题目

```
段式回文 其实与 一般回文 类似，只不过是最小的单位是 一段字符而不是 单个字母。
举个例子，对于一般回文 "abcba" 是回文，而 "volvo" 不是，但如果我们把"volvo" 分为 "vo
↪"、"l"、"vo" 三段，
则可以认为 “(vo)(l)(vo)” 是段式回文（分为 3 段）。
给你一个字符串text，在确保它满足段式回文的前提下，请你返回 段 的最大数量k。
如果段的最大数量为k，那么存在满足以下条件的a_1, a_2, ..., a_k：
每个a_i都是一个非空字符串；
将这些字符串首位相连的结果a_1 + a_2 + ... + a_k和原始字符串text相同；
对于所有1 <= i <= k，都有a_i = a_{k+1 - i}。
示例 1：输入：text = "ghiabcdefhelloadamhelloabcdefghi" 输出：7
解释：我们可以把字符串拆分成 "(ghi)(abcdef)(hello)(adam)(hello)(abcdef)(ghi)"。
示例 2：输入：text = "merchant" 输出：1
解释：我们可以把字符串拆分成 "(merchant)"。
示例 3：输入：text = "antaprezatepzapreanta" 输出：11
解释：我们可以把字符串拆分成 "(a)(nt)(a)(pre)(za)(tpe)(za)(pre)(a)(nt)(a)"。
示例 4：输入：text = "aaa" 输出：3
解释：我们可以把字符串拆分成 "(a)(a)(a)"。
提示：text仅由小写英文字符组成。
1 <= text.length <= 1000
```

- 解题思路

```
func longestDecomposition(text string) int {
        n := len(text)
        if n <= 1 {
                return n
        }
        for i := 1; i <= n/2; i++ {
                if text[:i] == text[n-i:] {
                        return 2 + longestDecomposition(text[i:n-i])
                }
        }
        return 1
}

# 2
func longestDecomposition(text string) int {
        n := len(text)
        if n <= 1 {
                return n
        }
        res := 0
        left, right := 0, n
        for left < right {
                for k := 1; ; k++ {
                        if k > (right-left)/2 { // 长度超过剩下的1/2
                                res++
                                return res
                        }
                        if text[left:left+k] == text[right-k:right] {
                                res = res + 2 // 可以切割，长度+2
                                left = left + k
                                right = right - k
                                break
                        }
                }
        }
        return res
}
```

36.2 1187. 使数组严格递增

36.2.1 题目

```
给你两个整数数组arr1 和 arr2，返回使arr1严格递增所需要的最小「操作」数（可能为 0）。
每一步「操作」中，你可以分别从 arr1 和 arr2 中各选出一个索引，
分别为i 和j，0 <=i < arr1.length和0 <= j < arr2.length，然后进行赋值运算arr1[i] =␣
↪arr2[j]。
如果无法让arr1严格递增，请返回-1。
示例 1：输入：arr1 = [1,5,3,6,7], arr2 = [1,3,2,4] 输出：1
解释：用 2 来替换 5，之后 arr1 = [1, 2, 3, 6, 7]。
示例 2：输入：arr1 = [1,5,3,6,7], arr2 = [4,3,1] 输出：2
解释：用 3 来替换 5，然后用 4 来替换 3，得到 arr1 = [1, 3, 4, 6, 7]。
示例3：输入：arr1 = [1,5,3,6,7], arr2 = [1,6,3,3] 输出：-1
解释：无法使 arr1 严格递增。
提示：1 <= arr1.length, arr2.length <= 2000
0 <= arr1[i], arr2[i] <= 10^9
```

36.2.2 解题思路

CHAPTER 37

1201-1300-Easy

37.1 1207. 独一无二的出现次数 (2)

- 题目

```
给你一个整数数组 arr，请你帮忙统计数组中每个数的出现次数。
如果每个数的出现次数都是独一无二的，就返回 true；否则返回 false。
示例 1：输入：arr = [1,2,2,1,1,3] 输出：true
解释：在该数组中，1 出现了 3 次，2 出现了 2 次，3 只出现了 1次。没有两个数的出现次数相同。
示例 2：输入：arr = [1,2] 输出：false
示例 3：输入：arr = [-3,0,1,-3,1,1,1,-3,10,0] 输出：true
提示：
    1 <= arr.length <= 1000
    -1000 <= arr[i] <= 1000
```

- 解题思路

```
func uniqueOccurrences(arr []int) bool {
        m := make(map[int]int)
        for _, v := range arr {
                m[v]++
        }
        temp := make(map[int]bool)
        for _, v := range m {
```

(续下页)

(接上页)

```
                if temp[v] == true {
                        return false
                }
                temp[v] = true
        }
        return true
}

#
func uniqueOccurrences(arr []int) bool {
        tempArr := make([]int,2001)
        for _, v := range arr {
                tempArr[v+1000]++
        }
        temp := make(map[int]bool)
        for _, v := range tempArr {
                if v == 0{
                        continue
                }
                if temp[v] == true {
                        return false
                }
                temp[v] = true
        }
        return true
}
```

37.2 1217. 玩筹码 (1)

- 题目

```
数轴上放置了一些筹码，每个筹码的位置存在数组 chips 当中。
你可以对 任何筹码 执行下面两种操作之一（不限操作次数，0 次也可以）：
    将第 i 个筹码向左或者右移动 2 个单位，代价为 0。
    将第 i 个筹码向左或者右移动 1 个单位，代价为 1。
最开始的时候，同一位置上也可能放着两个或者更多的筹码。
返回将所有筹码移动到同一位置（任意位置）上所需要的最小代价。
示例 1：输入：chips = [1,2,3] 输出：1
解释：第二个筹码移动到位置三的代价是 1，第一个筹码移动到位置三的代价是 0，总代价为 1。
示例 2：输入：chips = [2,2,2,3,3] 输出：2
解释：第四和第五个筹码移动到位置二的代价都是 1，所以最小总代价为 2。
提示：
```

(续下页)

(接上页)

```
    1 <= chips.length <= 100
    1 <= chips[i] <= 10^9
```

- 解题思路

```
/*
1、所有偶数移动到同一偶数位置，花费0
2、所有奇数移动到同一奇数位置，花费0
3、将较小移动到较多的位置。
*/
func minCostToMoveChips(chips []int) int {
        odd := 0
        even := 0
        for i := 0; i < len(chips); i++ {
                if chips[i]%2 == 1 {
                        odd++
                } else {
                        even++
                }
        }
        if odd > even {
                return even
        }
        return odd
}
```

37.3 1221. 分割平衡字符串 (3)

- 题目

```
在一个「平衡字符串」中，'L' 和 'R' 字符的数量是相同的。
给出一个平衡字符串 s，请你将它分割成尽可能多的平衡字符串。
返回可以通过分割得到的平衡字符串的最大数量。
示例 1：输入：s = "RLRRLLRLRL" 输出：4
解释：s 可以分割为 "RL", "RRLL", "RL", "RL", 每个子字符串中都包含相同数量的 'L' 和 'R
↪'。
示例 2：输入：s = "RLLLLRRRLR" 输出：3
解释：s 可以分割为 "RL", "LLLRRR", "LR", 每个子字符串中都包含相同数量的 'L' 和 'R'。
示例 3：输入：s = "LLLLRRRR" 输出：1
解释：s 只能保持原样 "LLLLRRRR".
提示：
    1 <= s.length <= 1000
```

(续下页)

(接上页)

```
    s[i] = 'L' 或 'R'
    分割得到的每个字符串都必须是平衡字符串。
```

- 解题思路

```
func balancedStringSplit(s string) int {
        res := 0
        if len(s) == 0 {
                return res
        }
        stack := make([]byte, 0)
        stack = append(stack, s[0])
        for i := 1; i < len(s); i++ {
                if len(stack) > 0 &&
                        ((s[i] == 'L' && stack[len(stack)-1] == 'R') ||
                                (s[i] == 'R' && stack[len(stack)-1] == 'L')) {
                        stack = stack[:len(stack)-1]
                        if len(stack) == 0 {
                                res++
                        }
                } else {
                        stack = append(stack, s[i])
                }
        }
        return res
}

#
func balancedStringSplit(s string) int {
        res := 0
        if len(s) == 0 {
                return res
        }
        count := 0
        if s[0] == 'L' {
                count++
        } else {
                count--
        }
        for i := 1; i < len(s); i++ {
                if s[i] == 'L' {
                        count++
                } else {
                        count--
```

(续下页)

(接上页)

```
                }
                if count == 0 {
                        res++
                }
        }
        return res
}

#
func balancedStringSplit(s string) int {
        res := 0
        if len(s) == 0 {
                return res
        }
        left := 0
        right := 0
        for i := 0; i < len(s); i++ {
                if s[i] == 'L' {
                        left++
                } else {
                        right++
                }
                if left == right {
                        res++
                }
        }
        return res
}
```

37.4 1232. 缀点成线 (3)

- 题目

```
在一个 XY 坐标系中有一些点，我们用数组 coordinates 来分别记录它们的坐标，
其中 coordinates[i] = [x, y] 表示横坐标为 x、纵坐标为 y 的点。
请你来判断，这些点是否在该坐标系中属于同一条直线上，是则返回 true，否则请返回 false。
示例 1：输入：coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]] 输出：true
示例 2：输入：coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]] 输出：false
提示：
    2 <= coordinates.length <= 1000
    coordinates[i].length == 2
    -10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4
```

(续下页)

(接上页)

```
    coordinates 中不含重复的点
```

- 解题思路

```
// k=y/x k1=y1/x1 => xy1=x1y
func checkStraightLine(coordinates [][]int) bool {
        x, y := coordinates[1][0]-coordinates[0][0], coordinates[1][1]-
↪coordinates[0][1]
        for i := 2; i < len(coordinates); i++ {
                x1, y1 := coordinates[i][0]-coordinates[i-1][0], coordinates[i][1]-
↪coordinates[i-1][1]
                if x*y1 != x1*y {
                        return false
                }
        }
        return true
}

#
// 鞋带公式
// S=|(x1 * y2 + x2 * y3 + x3 * y1 - y1 * x2 - y2 * x3 - y3 * x1)|/2
// S==0组成一条直线
func checkStraightLine(coordinates [][]int) bool {
        for i := 2; i < len(coordinates); i++ {
                x1 := coordinates[i-2][0]*coordinates[i-1][1] +
                        coordinates[i-1][0]*coordinates[i][1] +␣
↪coordinates[i][0]*coordinates[i-2][1]
                x2 := coordinates[i-2][1]*coordinates[i-1][0] +
                        coordinates[i-1][1]*coordinates[i][0] +␣
↪coordinates[i][1]*coordinates[i-2][0]
                if x1 != x2 {
                        return false
                }
        }
        return true
}

#
func checkStraightLine(coordinates [][]int) bool {
        for i := 2; i < len(coordinates); i++ {
                side1 := side(coordinates[i], coordinates[i-1])
                side2 := side(coordinates[i-1], coordinates[i-2])
                side3 := side(coordinates[i], coordinates[i-2])
                arr := []float64{side1, side2, side3}
```

(续下页)

(接上页)

```
                sort.Float64s(arr)
                if arr[2]-arr[1]-arr[0] > 0.00000005 || arr[2]-arr[1]-arr[0] < -0.
↪00000005 {
                        return false
                }
        }
        return true
}

func side(arr1, arr2 []int) float64 {
        res := (arr1[0]-arr2[0])*(arr1[0]-arr2[0]) +
                (arr1[1]-arr2[1])*(arr1[1]-arr2[1])
        return math.Sqrt(float64(res))
}
```

37.5 1237. 找出给定方程的正整数解 (3)

- 题目

```
给出一个函数 f(x, y) 和一个目标结果 z，请你计算方程 f(x,y) == z 所有可能的正整数␣
↪数对 x 和 y。
给定函数是严格单调的，也就是说：
    f(x, y) < f(x + 1, y)
    f(x, y) < f(x, y + 1)
函数接口定义如下：
interface CustomFunction {
public:
  // Returns positive integer f(x, y) for any given positive integer x and y.
  int f(int x, int y);
};
如果你想自定义测试，你可以输入整数 function_id 和一个目标结果 z 作为输入，
其中 function_id 表示一个隐藏函数列表中的一个函数编号，题目只会告诉你列表中的 2␣
↪个函数。
你可以将满足条件的 结果数对 按任意顺序返回。
示例 1：
输入：function_id = 1, z = 5
输出：[[1,4],[2,3],[3,2],[4,1]]
解释：function_id = 1 表示 f(x, y) = x + y
示例 2：
输入：function_id = 2, z = 5
输出：[[1,5],[5,1]]
解释：function_id = 2 表示 f(x, y) = x * y
```

(续下页)

(接上页)

```
提示：
    1 <= function_id <= 9
    1 <= z <= 100
    题目保证 f(x, y) == z 的解处于 1 <= x, y <= 1000 的范围内。
    在 1 <= x, y <= 1000 的前提下，题目保证 f(x, y) 是一个 32 位有符号整数。
```

- 解题思路

```
func findSolution(customFunction func(int, int) int, z int) [][]int {
        res := make([][]int, 0)
        for i := 1; i <= 1000; i++ {
                left := 1
                right := 1000
                v1, v2 := customFunction(i, left), customFunction(i, right)
                if z < v1 || z > v2 {
                        continue
                }
                for left <= right {
                        mid := left + (right-left)/2
                        v := customFunction(i, mid)
                        if z == v {
                                res = append(res, []int{i, mid})
                                break
                        } else if z > v {
                                left = mid + 1
                        } else {
                                right = mid - 1
                        }
                }
        }
        return res
}

#
func findSolution(customFunction func(int, int) int, z int) [][]int {
        res := make([][]int, 0)
        i := 1
        j := 1000
        for i <= 1000 && j >= 1 {
                if customFunction(i, j) == z {
                        res = append(res, []int{i, j})
                        i++
                        j--
                } else if customFunction(i, j) > z {
```

(续下页)

(接上页)

```
                        j--
                } else {
                        i++
                }
        }
        return res
}

#
func findSolution(customFunction func(int, int) int, z int) [][]int {
        res := make([][]int, 0)
        for i := 1; i <= 1000; i++ {
                for j := 1; j < 1000; j++ {
                        if customFunction(i, j) == z {
                                res = append(res, []int{i, j})
                        }
                }
        }
        return res
}
```

37.6 1252. 奇数值单元格的数目 (3)

- 题目

```
给你一个 n 行 m 列的矩阵，最开始的时候，每个单元格中的值都是 0。
另有一个索引数组 indices，indices[i] = [ri, ci] 中的
ri 和 ci 分别表示指定的行和列（从 0 开始编号）。
你需要将每对 [ri, ci] 指定的行和列上的所有单元格的值加 1。
请你在执行完所有 indices 指定的增量操作后，返回矩阵中 「奇数值单元格」 的数目。
示例 1：输入：n = 2, m = 3, indices = [[0,1],[1,1]] 输出：6
解释：最开始的矩阵是 [[0,0,0],[0,0,0]]。
第一次增量操作后得到 [[1,2,1],[0,1,0]]。
最后的矩阵是 [[1,3,1],[1,3,1]]，里面有 6 个奇数。
示例 2：输入：n = 2, m = 2, indices = [[1,1],[0,0]] 输出：0
解释：最后的矩阵是 [[2,2],[2,2]]，里面没有奇数。
提示：
    1 <= n <= 50
    1 <= m <= 50
    1 <= indices.length <= 100
    0 <= indices[i][0] < n
    0 <= indices[i][1] < m
```

- 解题思路

```
func oddCells(n int, m int, indices [][]int) int {
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        for i := 0; i < len(indices); i++ {
                r := indices[i][0]
                c := indices[i][1]
                for j := 0; j < m; j++ {
                        arr[r][j]++
                }
                for j := 0; j < n; j++ {
                        arr[j][c]++
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if arr[i][j]%2 == 1 {
                                res++
                        }
                }
        }
        return res
}

#
func oddCells(n int, m int, indices [][]int) int {
        rows := make([]int, n)
        cols := make([]int, m)
        for i := 0; i < len(indices); i++ {
                rows[indices[i][0]]++
                cols[indices[i][1]]++
        }
        numRows := 0
        for i := 0; i < n; i++ {
                if rows[i]%2 == 0 {
                        numRows++
                }
        }
        res := 0
        for i := 0; i < m; i++ {
                if cols[i]%2 == 0 {
```

(续下页)

(接上页)

```
                        res = res + n - numRows
                } else {
                        res = res + numRows
                }
        }
        return res
}

#
func oddCells(n int, m int, indices [][]int) int {
        rows := make([]int, n)
        cols := make([]int, m)
        for i := 0; i < len(indices); i++ {
                rows[indices[i][0]]++
                cols[indices[i][1]]++
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if (rows[i]+cols[j])%2 == 1 {
                                res++
                        }
                }
        }
        return res
}
```

37.7 1260. 二维网格迁移 (2)

- 题目

```
给你一个 m 行 n 列的二维网格 grid 和一个整数 k。你需要将 grid 迁移 k 次。
每次「迁移」操作将会引发下述活动：
    位于 grid[i][j] 的元素将会移动到 grid[i][j + 1]。
    位于 grid[i][n - 1] 的元素将会移动到 grid[i + 1][0]。
    位于 grid[m - 1][n - 1] 的元素将会移动到 grid[0][0]。
请你返回 k 次迁移操作后最终得到的 二维网格。
示例 1：输入：grid = [[1,2,3],[4,5,6],[7,8,9]], k = 1 输出：[[9,1,2],[3,4,5],[6,7,8]]
示例 2：输入：grid = [[3,8,1,9],[19,7,2,5],[4,6,11,10],[12,0,21,13]], k = 4
输出：[[12,0,21,13],[3,8,1,9],[19,7,2,5],[4,6,11,10]]
示例 3：输入：grid = [[1,2,3],[4,5,6],[7,8,9]], k = 9 输出：[[1,2,3],[4,5,6],[7,8,9]]
提示：
```

(续下页)

(接上页)

```
    1 <= grid.length <= 50
    1 <= grid[i].length <= 50
    -1000 <= grid[i][j] <= 1000
    0 <= k <= 100
```

- 解题思路

```
func shiftGrid(grid [][]int, k int) [][]int {
        for i := 0; i < k; i++ {
                temp := make([][]int, len(grid))
                for i := 0; i < len(temp); i++ {
                        temp[i] = make([]int, len(grid[i]))
                }
                for i := 0; i < len(grid); i++ {
                        for j := 0; j < len(grid[i])-1; j++ {
                                temp[i][j+1] = grid[i][j]
                        }
                }
                for i := 0; i < len(grid)-1; i++ {
                        temp[i+1][0] = grid[i][len(grid[0])-1]
                }
                temp[0][0] = grid[len(grid)-1][len(grid[0])-1]
                grid = temp
        }
        return grid
}

#
func shiftGrid(grid [][]int, k int) [][]int {
        n := len(grid)
        m := len(grid[0])
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                res[i] = make([]int, m)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        x := ((i*m + j) + k) % (n * m) / m
                        y := ((i*m + j) + k) % (n * m) % m
                        // x := (i + (j+k)/m) % n
                        // y := (j + k) % m
                        res[x][y] = grid[i][j]
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

37.8 1266. 访问所有点的最小时间 (1)

- 题目

```
平面上有 n 个点，点的位置用整数坐标表示 points[i] = [xi, yi]。
请你计算访问所有这些点需要的最小时间（以秒为单位）。
你可以按照下面的规则在平面上移动：
    每一秒沿水平或者竖直方向移动一个单位长度，
    或者跨过对角线（可以看作在一秒内向水平和竖直方向各移动一个单位长度）。
    必须按照数组中出现的顺序来访问这些点。
示例 1：输入：points = [[1,1],[3,4],[-1,0]] 输出：7
解释：一条最佳的访问路径是：
[1,1] -> [2,2] -> [3,3] -> [3,4] -> [2,3] -> [1,2] -> [0,1] -> [-1,0]
从 [1,1] 到 [3,4] 需要 3 秒
从 [3,4] 到 [-1,0] 需要 4 秒
一共需要 7 秒
示例 2：输入：points = [[3,2],[-2,2]] 输出：5
提示：
    points.length == n
    1 <= n <= 100
    points[i].length == 2
    -1000 <= points[i][0], points[i][1] <= 1000
```

- 解题思路

```
func minTimeToVisitAllPoints(points [][]int) int {
        res := 0
        for i := 1; i < len(points); i++ {
                x := length(points[i][0], points[i-1][0])
                y := length(points[i][1], points[i-1][1])
                if x > y {
                        res = res + x
                } else {
                        res = res + y
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func length(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

37.9 1275. 找出井字棋的获胜者 (2)

- 题目

```
A 和 B 在一个 3 x 3 的网格上玩井字棋。
井字棋游戏的规则如下：
    玩家轮流将棋子放在空方格 (" ") 上。
    第一个玩家 A 总是用 "X" 作为棋子，而第二个玩家 B 总是用 "O" 作为棋子。
    "X" 和 "O" 只能放在空方格中，而不能放在已经被占用的方格上。
    只要有 3 个相同的（非空）棋子排成一条直线（行、列、对角线）时，游戏结束。
    如果所有方块都放满棋子（不为空），游戏也会结束。
    游戏结束后，棋子无法再进行任何移动。
给你一个数组 moves，其中每个元素是大小为 2 的另一个数组（元素分别对应网格的行和列），
它按照 A 和 B 的行动顺序（先 A 后 B）记录了两人各自的棋子位置。

如果游戏存在获胜者（A 或 B），就返回该游戏的获胜者；
如果游戏以平局结束，则返回 "Draw"；如果仍会有行动（游戏未结束），则返回 "Pending"。
你可以假设 moves 都 有效（遵循井字棋规则），网格最初是空的，A 将先行动。
示例 1：输入：moves = [[0,0],[2,0],[1,1],[2,1],[2,2]] 输出："A"
解释："A" 获胜，他总是先走。
"X  "    "X  "    "X  "    "X  "    "X  "
"   " -> "   " -> " X " -> " X " -> " X "
"   "    "O  "    "O  "    "OO "    "OOX"
示例 2：输入：moves = [[0,0],[1,1],[0,1],[0,2],[1,0],[2,0]] 输出："B"
解释："B" 获胜。
"X  "    "X  "    "XX "    "XXO"    "XXO"    "XXO"
"   " -> " O " -> " O " -> " O " -> "XO " -> "XO "
"   "    "   "    "   "    "   "    "   "    "O  "
示例 3：输入：moves = [[0,0],[1,1],[2,0],[1,0],[1,2],[2,1],[0,1],[0,2],[2,2]] 输出：
↪"Draw"
输出：由于没有办法再行动，游戏以平局结束。
"XXO"
"OOX"
"XOX"
示例 4：输入：moves = [[0,0],[1,1]] 输出："Pending"
```

(续下页)

(接上页)

```
解释：游戏还没有结束。
"X  "
" O "
"   "
提示：
    1 <= moves.length <= 9
    moves[i].length == 2
    0 <= moves[i][j] <= 2
    moves 里没有重复的元素。
    moves 遵循井字棋的规则。
```

- 解题思路

```
func tictactoe(moves [][]int) string {
        arrA := make([]int, 0)
        arrB := make([]int, 0)
        for i := 0; i < len(moves); i++ {
                value := moves[i][0]*3 + moves[i][1] + 1
                if i%2 == 0 {
                        arrA = append(arrA, value)
                        if check(arrA) {
                                return "A"
                        }
                } else {
                        arrB = append(arrB, value)
                        if check(arrB) {
                                return "B"
                        }
                }
        }
        if len(moves) == 9 {
                return "Draw"
        }
        return "Pending"
}

var state = [][]int{
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9},
        {1, 4, 7},
        {2, 5, 8},
        {3, 6, 9},
        {1, 5, 9},
```

(续下页)

(接上页)

```
        {3, 5, 7},
}

func check(arr []int) bool {
        if len(arr) < 3 {
                return false
        }
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        for k := j + 1; k < len(arr); k++ {
                                temp := []int{arr[i], arr[j], arr[k]}
                                sort.Ints(temp)
                                for n := 0; n < len(state); n++ {
                                        if temp[0] == state[n][0] &&
                                                temp[1] == state[n][1] &&
                                                temp[2] == state[n][2] {
                                                return true
                                        }
                                }
                        }
                }
        }
        return false
}

#
func tictactoe(moves [][]int) string {
        arr := make([][]int, 4)
        for i := 0; i < len(arr); i++ {
                arr[i] = make([]int, 4)
        }
        for i := 0; i < len(moves); i++ {
                x := moves[i][0] + 1
                y := moves[i][1] + 1
                if i%2 == 0 {
                        arr[x][y]++
                } else {
                        arr[x][y]--
                }
        }
        sum1 := arr[1][1] + arr[2][2] + arr[3][3]
        sum2 := arr[1][3] + arr[2][2] + arr[3][1]
        if sum1 == 3 || sum2 == 3 {
```

(续下页)

(接上页)

```
                return "A"
        } else if sum1 == -3 || sum2 == -3 {
                return "B"
        }
        for i := 1; i <= 3; i++ {
                sum1 := arr[i][1] + arr[i][2] + arr[i][3]
                sum2 := arr[1][i] + arr[2][i] + arr[3][i]
                if sum1 == 3 || sum2 == 3 {
                        return "A"
                } else if sum1 == -3 || sum2 == -3 {
                        return "B"
                }
        }
        if len(moves) == 9 {
                return "Draw"
        }
        return "Pending"
}
```

37.10 1281. 整数的各位积和之差 (2)

- 题目

```
给你一个整数 n，请你帮忙计算并返回该整数「各位数字之积」与「各位数字之和」的差。
示例 1：输入：n = 234 输出：15
解释：
各位数之积 = 2 * 3 * 4 = 24
各位数之和 = 2 + 3 + 4 = 9
结果 = 24 - 9 = 15
示例 2：输入：n = 4421 输出：21
解释：
各位数之积 = 4 * 4 * 2 * 1 = 32
各位数之和 = 4 + 4 + 2 + 1 = 11
结果 = 32 - 11 = 21
提示：
    1 <= n <= 10^5
```

- 解题思路

```
func subtractProductAndSum(n int) int {
        sum := 0
        mul := 1
```

(续下页)

(接上页)

```
        for n > 0 {
                value := n % 10
                n = n / 10
                sum = sum + value
                mul = mul * value
        }
        return mul - sum
}

func subtractProductAndSum(n int) int {
        sum := 0
        mul := 1
        str := strconv.Itoa(n)
        for i := 0; i < len(str); i++ {
                value := int(str[i] - '0')
                sum = sum + value
                mul = mul * value
        }
        return mul - sum
}
```

37.11 1287. 有序数组中出现次数超过 25% 的元素 (4)

- 题目

```
给你一个非递减的 有序␣
↪整数数组，已知这个数组中恰好有一个整数，它的出现次数超过数组元素总数的 25%。
请你找到并返回这个整数
示例：输入：arr = [1,2,2,6,6,6,6,7,10] 输出：6
提示：
    1 <= arr.length <= 10^4
    0 <= arr[i] <= 10^5
```

- 解题思路

```
func findSpecialInteger(arr []int) int {
        count := 1
        res := arr[0]
        for i := 1; i < len(arr); i++ {
                if arr[i] == arr[i-1] {
                        count++
                        if count > len(arr)/4 {
```

(续下页)

(接上页)

```
                                return arr[i]
                        }
                } else {
                        res = arr[i]
                        count = 1
                }
        }
        return res
}

#
func findSpecialInteger(arr []int) int {
        step := len(arr) / 4
        for i := 0; i < len(arr)-step; i++ {
                if arr[i] == arr[i+step] {
                        return arr[i]
                }
        }
        return -1
}

#
func findSpecialInteger(arr []int) int {
        length := len(arr) / 4
        for i := 1; i <= 2; i++ {
                value := arr[length*i]
                left := leftSearch(arr, value)
                right := rightSearch(arr, value)
                if right-left+1 > length {
                        return value
                }
        }
        return arr[3*length]
}

func leftSearch(arr []int, value int) int {
        left := 0
        right := len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] >= value {
                        right = mid
                } else {
```

(续下页)

(接上页)

```
                        left = mid + 1
                }
        }
        return left
}

func rightSearch(arr []int, value int) int {
        left := 0
        right := len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] > value {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return right - 1
}

#
func findSpecialInteger(arr []int) int {
        length := len(arr) / 4
        for i := 1; i <= 2; i++ {
                value := arr[length*i]
                left := length * i
                for left > 0 {
                        if arr[left] == arr[left-1] {
                                left--
                        } else {
                                break
                        }
                }
                right := length * i
                for right < len(arr)-1 {
                        if arr[right] == arr[right+1] {
                                right++
                        } else {
                                break
                        }
                }
                if right-left+1 > length {
                        return value
```

(续下页)

(接上页)

```
                }
        }
        return arr[3*length]
}
```

37.12 1290. 二进制链表转整数 (3)

- 题目

```
给你一个单链表的引用结点 head。链表中每个结点的值不是 0 就是 1。
已知此链表是一个整数数字的二进制表示形式。
请你返回该链表所表示数字的 十进制值 。
示例 1：输入：head = [1,0,1] 输出：5
解释：二进制数 (101) 转化为十进制数 (5)
示例 2：输入：head = [0] 输出：0
示例 3：输入：head = [1] 输出：1
示例 4：输入：head = [1,0,0,1,0,0,1,1,1,0,0,0,0,0,0] 输出：18880
示例 5：输入：head = [0,0] 输出：0
提示：
    链表不为空。
    链表的结点总数不超过 30。
    每个结点的值不是 0 就是 1。
```

- 解题思路

```
func getDecimalValue(head *ListNode) int {
        arr := make([]int, 0)
        for head != nil {
                arr = append(arr, head.Val)
                head = head.Next
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                res = res * 2
                res = res + arr[i]
        }
        return res
}

#
func getDecimalValue(head *ListNode) int {
        res := 0
```

(续下页)

(接上页)

```
        for head != nil {
                res = res*2 + head.Val
                head = head.Next
        }
        return res
}

#
var count = 0

func getDecimalValue(head *ListNode) int {
        if head == nil {
                count = 0
                return 0
        }
        value := getDecimalValue(head.Next)
        res := head.Val*int(math.Pow(2, float64(count))) + value
        count++
        return res
}
```

37.13 1295. 统计位数为偶数的数字 (2)

- 题目

```
给你一个整数数组 nums, 请你返回其中位数为 偶数 的数字的个数。
示例 1: 输入: nums = [12,345,2,6,7896] 输出: 2
解释: 12 是 2 位数字 (位数为偶数)
345 是 3 位数字 (位数为奇数)
2 是 1 位数字 (位数为奇数)
6 是 1 位数字 位数为奇数)
7896 是 4 位数字 (位数为偶数)
因此只有 12 和 7896 是位数为偶数的数字
示例 2: 输入: nums = [555,901,482,1771]输出: 1
解释: 只有 1771 是位数为偶数的数字。
提示:
    1 <= nums.length <= 500
    1 <= nums[i] <= 10^5
```

- 解题思路

```
func findNumbers(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                count := 0
                value := nums[i]
                for value > 0 {
                        value = value / 10
                        count++
                }
                if count%2 == 0 {
                        res++
                }
        }
        return res
}

#
func findNumbers(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                value := strconv.Itoa(nums[i])
                if len(value)%2 == 0 {
                        res++
                }
        }
        return res
}
```

37.14 1299. 将每个元素替换为右侧最大元素 (3)

- 题目

```
给你一个数组 arr ，请你将每个元素用它右边最大的元素替换，如果是最后一个元素，用 -1␣
↪替换。
完成所有替换操作后，请你返回这个数组。
示例：输入：arr = [17,18,5,4,6,1] 输出：[18,6,6,6,1,-1]
提示：
    1 <= arr.length <= 10^4
    1 <= arr[i] <= 10^5
```

- 解题思路

```
func replaceElements(arr []int) []int {
        max := -1
        for i := len(arr) - 1; i >= 0; i-- {
                if arr[i] > max {
                        arr[i], max = max, arr[i]
                } else {
                        arr[i] = max
                }
        }
        return arr
}

#
func replaceElements(arr []int) []int {
        for i := 0; i < len(arr); i++ {
                max := -1
                for j := i + 1; j < len(arr); j++ {
                        if arr[j] > max {
                                max = arr[j]
                        }
                }
                arr[i] = max
        }
        return arr
}

#
func replaceElements(arr []int) []int {
        res := make([]int, len(arr))
        res[len(res)-1] = -1
        for i := len(arr) - 2; i >= 0; i-- {
                if arr[i+1] > res[i+1] {
                        res[i] = arr[i+1]
                } else {
                        res[i] = res[i+1]
                }
        }
        return res
}
```

CHAPTER 38

1201-1300-Medium

38.1 1201. 丑数 III(2)

- 题目

```
给你四个整数：n 、a 、b 、c ，请你设计一个算法来找出第n个丑数。
丑数是可以被a或b或 c整除的 正整数 。
示例 1：输入：n = 3, a = 2, b = 3, c = 5 输出：4
解释：丑数序列为 2, 3, 4, 5, 6, 8, 9, 10... 其中第 3 个是 4。
示例 2：输入：n = 4, a = 2, b = 3, c = 4 输出：6
解释：丑数序列为 2, 3, 4, 6, 8, 9, 10, 12... 其中第 4 个是 6。
示例 3：输入：n = 5, a = 2, b = 11, c = 13 输出：10
解释：丑数序列为 2, 4, 6, 8, 10, 11, 12, 13... 其中第 5 个是 10。
示例 4：输入：n = 1000000000, a = 2, b = 217983653, c = 336916467 输出：1999999984
提示：1 <= n, a, b, c <= 10^9
1 <= a * b * c <= 10^18
本题结果在[1,2 * 10^9]的范围内
```

- 解题思路

```
func nthUglyNumber(n int, a int, b int, c int) int {
        ab, ac, bc := lcm(a, b), lcm(a, c), lcm(b, c)
        abc := lcm(ab, c)
        left := 1
        right := 2000000000
```

(续下页)

(接上页)

```
        for left <= right {
                mid := left + (right-left)/2
                total := mid/a + mid/b + mid/c - mid/ab - mid/ac - mid/bc + mid/abc
                if total == n {
                        if mid%a == 0 || mid%b == 0 || mid%c == 0 {
                                return mid
                        }
                        right = mid - 1
                } else if total < n {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return left
}

// 求最小公倍数
func lcm(x, y int) int {
        return x * y / gcd(x, y)
}

// 求最大公约数
func gcd(a, b int) int {
        if a < b {
                a, b = b, a
        }
        if a%b == 0 {
                return b
        }
        return gcd(a%b, b)
}

# 2
func nthUglyNumber(n int, a int, b int, c int) int {
        ab, ac, bc := lcm(a, b), lcm(a, c), lcm(b, c)
        abc := lcm(ab, c)
        left := 0
        right := 2000000000
        for left < right {
                mid := left + (right-left)/2
                total := mid/a + mid/b + mid/c - mid/ab - mid/ac - mid/bc + mid/abc
                if total >= n {
```

(续下页)

(接上页)

```
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

// 求最小公倍数
func lcm(x, y int) int {
        return x * y / gcd(x, y)
}

// 求最大公约数
func gcd(a, b int) int {
        if a < b {
                a, b = b, a
        }
        if a%b == 0 {
                return b
        }
        return gcd(a%b, b)
}
```

38.2 1202. 交换字符串中的元素 (1)

- 题目

```
给你一个字符串s，以及该字符串中的一些「索引对」数组pairs，
其中pairs[i] =[a, b]表示字符串中的两个索引（编号从 0 开始）。
你可以 任意多次交换 在pairs中任意一对索引处的字符。
返回在经过若干次交换后，s可以变成的按字典序最小的字符串。
示例 1:输入：s = "dcab", pairs = [[0,3],[1,2]] 输出："bacd"
解释：  交换 s[0] 和 s[3], s = "bcad"
交换 s[1] 和 s[2], s = "bacd"
示例 2：输入：s = "dcab", pairs = [[0,3],[1,2],[0,2]] 输出："abcd"
解释：交换 s[0] 和 s[3], s = "bcad"
交换 s[0] 和 s[2], s = "acbd"
交换 s[1] 和 s[2], s = "abcd"
示例 3：输入：s = "cba", pairs = [[0,1],[1,2]] 输出："abc"
解释：交换 s[0] 和 s[1], s = "bca"
交换 s[1] 和 s[2], s = "bac"
```

(续下页)

(接上页)

```
交换 s[0] 和 s[1], s = "abc"
提示：1 <= s.length <= 10^5
0 <= pairs.length <= 10^5
0 <= pairs[i][0], pairs[i][1] <s.length
s中只含有小写英文字母
```

- 解题思路

```
func smallestStringWithSwaps(s string, pairs [][]int) string {
        n := len(s)
        fa := make([]int, n)
        for i := 0; i < n; i++ {
                fa[i] = i
        }
        for i := 0; i < len(pairs); i++ {
                a, b := pairs[i][0], pairs[i][1]
                union(fa, a, b)
        }
        m := make(map[int][]int)
        for i := 0; i < len(s); i++ {
                target := find(fa, i)
                m[target] = append(m[target], i)
        }
        res := []byte(s)
        for _, v := range m {
                arr := make([]int, 0)
                for i := 0; i < len(v); i++ {
                        arr = append(arr, int(s[v[i]]))
                }
                sort.Ints(arr)
                for i := 0; i < len(v); i++ {
                        res[v[i]] = byte(arr[i])
                }
        }
        return string(res)
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}

func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
```

(续下页)

(接上页)

```
                a = fa[a]
        }
        return a
}
```

38.3 1208. 尽可能使字符串相等 (4)

- 题目

```
给你两个长度相同的字符串，s 和 t。
将 s中的第i个字符变到t中的第 i 个字符需要|s[i] - t[i]|的开销（开销可能为 0），
也就是两个字符的 ASCII 码值的差的绝对值。
用于变更字符串的最大预算是maxCost。
在转化字符串时，总开销应当小于等于该预算，这也意味着字符串的转化可能是不完全的。
如果你可以将 s 的子字符串转化为它在 t 中对应的子字符串，则返回可以转化的最大长度。
如果 s 中没有子字符串可以转化成 t 中对应的子字符串，则返回 0。
示例 1: 输入：s = "abcd", t = "bcdf", cost = 3 输出：3
解释：s 中的 "abc" 可以变为 "bcd"。开销为 3，所以最大长度为 3。
示例 2: 输入：s = "abcd", t = "cdef", cost = 3 输出：1
解释：s 中的任一字符要想变成 t 中对应的字符，其开销都是 2。因此，最大长度为 1。
示例 3: 输入：s = "abcd", t = "acde", cost = 0 输出：1
解释：你无法作出任何改动，所以最大长度为 1。
提示：1 <= s.length, t.length <= 10^5
0 <= maxCost <= 10^6
s 和t都只含小写英文字母。
```

- 解题思路

```
func equalSubstring(s string, t string, maxCost int) int {
        arr := make([]int, len(s))
        for i := 0; i < len(s); i++ {
                arr[i] = getValue(s[i], t[i])
        }
        left, right := 0, 0
        total := 0
        res := 0
        for right < len(s) {
                for total+arr[right] > maxCost {
                        total = total - arr[left]
                        left++
                }
                total = total + arr[right]
```

(续下页)

(接上页)

```
                if right-left+1 > res {
                        res = right - left + 1
                }
                right++
        }
        return res
}

func getValue(a, b byte) int {
        res := int(a) - int(b)
        if res < 0 {
                return -res
        }
        return res
}

# 2
func equalSubstring(s string, t string, maxCost int) int {
        left := 0
        total := 0
        for right := 0; right < len(s); right++ {
                total = total + getValue(s[right], t[right])
                if total > maxCost {
                        total = total - getValue(s[left], t[left])
                        left++
                }
        }
        return len(s) - left
}

func getValue(a, b byte) int {
        res := int(a) - int(b)
        if res < 0 {
                return -res
        }
        return res
}

# 3
func equalSubstring(s string, t string, maxCost int) int {
        res := 0
        for i := 0; i < len(s); i++ {
                total := 0
```

(续下页)

(接上页)

```
                count := 0
                for j := i; j < len(s); j++ {
                        total = total + getValue(s[j], t[j])
                        if total > maxCost {
                                break
                        }
                        count++
                }
                if count > res {
                        res = count
                }
        }
        return res
}

func getValue(a, b byte) int {
        res := int(a) - int(b)
        if res < 0 {
                return -res
        }
        return res
}

# 4
func equalSubstring(s string, t string, maxCost int) int {
        res := 0
        arr := make([]int, len(s)+1)
        for i := 0; i < len(s); i++ {
                arr[i+1] = arr[i] + getValue(s[i], t[i])
        }
        for i := 1; i <= len(s); i++ {
                index := sort.SearchInts(arr[:i], arr[i]-maxCost)
                if i-index > res {
                        res = i - index
                }
        }
        return res
}

func getValue(a, b byte) int {
        res := int(a) - int(b)
        if res < 0 {
                return -res
```

(续下页)

(接上页)

```
        }
        return res
}
```

38.4 1209. 删除字符串中的所有相邻重复项 II(2)

- 题目

```
给你一个字符串s，「k 倍重复项删除操作」将会从 s中选择k个相邻且相等的字母，
并删除它们，使被删去的字符串的左侧和右侧连在一起。
你需要对s重复进行无限次这样的删除操作，直到无法继续为止。
在执行完所有删除操作后，返回最终得到的字符串。
本题答案保证唯一。
示例 1：输入：s = "abcd", k = 2 输出："abcd"
解释：没有要删除的内容。
示例 2：输入：s = "deeedbbcccbdaa", k = 3 输出："aa"
解释： 先删除 "eee" 和 "ccc"，得到 "ddbbbdaa"
再删除 "bbb"，得到 "dddaa"
最后删除 "ddd"，得到 "aa"
示例 3：输入：s = "pbbcggttciiippooaais", k = 2 输出："ps"
提示：1 <= s.length <= 10^5
2 <= k <= 10^4
s中只含有小写英文字母。
```

- 解题思路

```
func removeDuplicates(s string, k int) string {
        if len(s) < k {
                return s
        }
        res := ""
        for i := 0; i < len(s); i++ {
                res = res + string(s[i])
                if len(res) >= k && res[len(res)-k:] == strings.Repeat(string(s[i]),␣
↪k) {
                        res = res[:len(res)-k]
                }
        }
        return res
}

# 2
```

(续下页)

(接上页)

```
func removeDuplicates(s string, k int) string {
        if len(s) < k {
                return s
        }
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                stack = append(stack, s[i])
                if len(stack) >= k {
                        a := string(stack[len(stack)-k:])
                        b := strings.Repeat(string(s[i]), k)
                        if a == b {
                                stack = stack[:len(stack)-k]
                        }
                }
        }
        return string(stack)
}
```

38.5 1218. 最长定差子序列 (2)

- 题目

```
给你一个整数数组arr和一个整数difference，请你找出并返回 arr中最长等差子序列的长度，
该子序列中相邻元素之间的差等于 difference 。
子序列 是指在不改变其余元素顺序的情况下，通过删除一些元素或不删除任何元素而从 arr
↪派生出来的序列。
示例 1：输入：arr = [1,2,3,4], difference = 1 输出：4
解释：最长的等差子序列是 [1,2,3,4]。
示例2：输入：arr = [1,3,5,7], difference = 1 输出：1
解释：最长的等差子序列是任意单个元素。
示例 3：输入：arr = [1,5,7,8,5,3,4,2,1], difference = -2 输出：4
解释：最长的等差子序列是 [7,5,3,1]。
提示：1 <= arr.length <= 105
-104 <= arr[i], difference <= 104
```

- 解题思路

```
func longestSubsequence(arr []int, difference int) int {
        res := 0
        dp := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                dp[arr[i]] = dp[arr[i]-difference] + 1
```

(续下页)

(接上页)

```
                if dp[arr[i]] > res {
                        res = dp[arr[i]]
                }
        }
        return res
}

# 2
func longestSubsequence(arr []int, difference int) int {
        res := 0
        dp := make([]int, 40001)
        for i := 0; i < len(arr); i++ {
                index := arr[i] + 20000
                dp[index] = dp[index-difference] + 1
                if dp[index] > res {
                        res = dp[index]
                }
        }
        return res
}
```

38.6 1219. 黄金矿工 (1)

- 题目

```
你要开发一座金矿，地质勘测学家已经探明了这座金矿中的资源分布，并用大小为m * n 的网格␣
↪grid 进行了标注。
每个单元格中的整数就表示这一单元格中的黄金数量；如果该单元格是空的，那么就是 0。
为了使收益最大化，矿工需要按以下规则来开采黄金：
每当矿工进入一个单元，就会收集该单元格中的所有黄金。
矿工每次可以从当前位置向上下左右四个方向走。
每个单元格只能被开采（进入）一次。
不得开采（进入）黄金数目为 0 的单元格。
矿工可以从网格中 任意一个 有黄金的单元格出发或者是停止。
示例 1：输入：grid = [[0,6,0],[5,8,7],[0,9,0]] 输出：24
解释：[[0,6,0],
 [5,8,7],
 [0,9,0]]
一种收集最多黄金的路线是：9 -> 8 -> 7。
示例 2：

输入：grid = [[1,0,7],[2,0,6],[3,4,5],[0,3,0],[9,0,20]]
```

(续下页)

(接上页)

```
输出：28
解释：
[[1,0,7],
 [2,0,6],
 [3,4,5],
 [0,3,0],
 [9,0,20]]
一种收集最多黄金的路线是：1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7。
提示：1 <= grid.length,grid[i].length <= 15
0 <= grid[i][j] <= 100
最多 25 个单元格中有黄金。
```

- 解题思路

```
var res int

func getMaximumGold(grid [][]int) int {
        res = 0
        visited := make([][]bool, len(grid))
        for i := 0; i < len(grid); i++ {
                visited[i] = make([]bool, len(grid[i]))
        }
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] > 0 {
                                dfs(grid, i, j, 0, visited)
                        }
                }
        }
        return res
}

func dfs(grid [][]int, i, j int, sum int, visited [][]bool) {
        if i < 0 || i >= len(grid) || j < 0 || j >= len(grid[0]) ||
                grid[i][j] == 0 || visited[i][j] == true {
                return
        }
        visited[i][j] = true
        sum = sum + grid[i][j]
        if sum > res {
                res = sum
        }
        dfs(grid, i+1, j, sum, visited)
        dfs(grid, i-1, j, sum, visited)
```

(续下页)

(接上页)

```
        dfs(grid, i, j+1, sum, visited)
        dfs(grid, i, j-1, sum, visited)
        visited[i][j] = false
}
```

38.7 1222. 可以攻击国王的皇后 (1)

- 题目

```
在一个8x8的棋盘上，放置着若干「黑皇后」和一个「白国王」。
「黑皇后」在棋盘上的位置分布用整数坐标数组queens表示，「白国王」的坐标用数组 king表示。
「黑皇后」的行棋规定是：横、直、斜都可以走，步数不受限制，但是，不能越子行棋。
请你返回可以直接攻击到「白国王」的所有「黑皇后」的坐标（任意顺序）。
示例 1：输入：queens = [[0,1],[1,0],[4,0],[0,4],[3,3],[2,4]], king = [0,0]
输出：[[0,1],[1,0],[3,3]]
解释：  [0,1] 的皇后可以攻击到国王，因为他们在同一行上。
[1,0] 的皇后可以攻击到国王，因为他们在同一列上。
[3,3] 的皇后可以攻击到国王，因为他们在同一条对角线上。
[0,4] 的皇后无法攻击到国王，因为她被位于 [0,1] 的皇后挡住了。
[4,0] 的皇后无法攻击到国王，因为她被位于 [1,0] 的皇后挡住了。
[2,4] 的皇后无法攻击到国王，因为她和国王不在同一行/列/对角线上。
示例 2：输入：queens = [[0,0],[1,1],[2,2],[3,4],[3,5],[4,4],[4,5]], king = [3,3]
输出：[[2,2],[3,4],[4,4]]
示例 3：输入：queens = [[5,6],[7,7],[2,1],[0,7],[1,6],[5,1],[3,7],[0,3],[4,0],[1,2],
[6,3],[5,0],[0,4],[2,2],[1,1],[6,4],[5,4],[0,0],[2,6],[4,5],[5,2],[1,4],[7,5],
[2,3],[0,5],[4,2],[1,0],[2,7],[0,1],[4,6],[6,1],[0,6],[4,3],[1,7]], king = [3,4]
输出：[[2,3],[1,4],[1,6],[3,7],[4,3],[5,4],[4,5]]
提示：1 <= queens.length<= 63
queens[i].length == 2
0 <= queens[i][j] <8
king.length == 2
0 <= king[0], king[1] < 8
一个棋盘格上最多只能放置一枚棋子。
```

- 解题思路

```
var dir = [][]int{{-1, -1}, {-1, 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}
↪}

func queensAttacktheKing(queens [][]int, king []int) [][]int {
        arr := make([][]int, 8)
```

(续下页)

(接上页)

```
        for i := 0; i < 8; i++ {
                arr[i] = make([]int, 8)
        }
        for i := 0; i < len(queens); i++ {
                a, b := queens[i][0], queens[i][1]
                arr[a][b] = 1
        }
        res := make([][]int, 0)
        for i := 0; i < len(dir); i++ {
                x := dir[i][0] + king[0]
                y := dir[i][1] + king[1]
                for 0 <= x && x < 8 && 0 <= y && y < 8 {
                        if arr[x][y] == 1 {
                                res = append(res, []int{x, y})
                                break
                        }
                        x = x + dir[i][0]
                        y = y + dir[i][1]
                }
        }
        return res
}
```

38.8 1223. 掷骰子模拟 (2)

- 题目

```
有一个骰子模拟器会每次投掷的时候生成一个 1 到 6 的随机数。
不过我们在使用它时有个约束，就是使得投掷骰子时，
连续 掷出数字i的次数不能超过rollMax[i]（i从 1 开始编号）。
现在，给你一个整数数组rollMax和一个整数n，请你来计算掷n次骰子可得到的不同点数序列的数量。
假如两个序列中至少存在一个元素不同，就认为这两个序列是不同的。
由于答案可能很大，所以请返回 模10^9 + 7之后的结果。
示例 1：输入：n = 2, rollMax = [1,1,2,2,2,3] 输出：34
解释：我们掷 2 次骰子，如果没有约束的话，共有 6 * 6 = 36 种可能的组合。
但是根据 rollMax 数组，数字 1 和 2 最多连续出现一次，所以不会出现序列 (1,1) 和 (2,2)。
因此，最终答案是 36-2 = 34。
示例 2：输入：n = 2, rollMax = [1,1,1,1,1,1] 输出：30
示例 3：输入：n = 3, rollMax = [1,1,1,2,2,3] 输出：181
提示：1 <= n <= 5000
rollMax.length == 6
1 <= rollMax[i] <= 15
```

- 解题思路

```
func dieSimulator(n int, rollMax []int) int {
        dp := make([][7][16]int, n+1) // 第i轮,以j结尾,出现k次
        for j := 1; j <= 6; j++ {
                dp[1][j][1] = 1
        }
        for i := 2; i <= n; i++ {
                for j := 1; j <= 6; j++ {
                        for k := 1; k <= rollMax[j-1] && k <= i; k++ {
                                if k == 1 { // 不同情况
                                        for a := 1; a <= 6; a++ {
                                                if a != j { // 考虑不同的情况
                                                        for b := 1; b <= rollMax[a-1]␣
↪&& b <= i-1; b++ {
                                                                dp[i][j][k] =␣
↪(dp[i][j][k] + dp[i-1][a][b]) % 1000000007
                                                        }
                                                }
                                        }
                                } else { // 直接同上一个
                                        dp[i][j][k] = dp[i-1][j][k-1]
                                }
                        }
                }
        }
        res := 0
        for j := 1; j <= 6; j++ {
                for k := 1; k <= 15; k++ {
                        res = (res + dp[n][j][k]) % 1000000007
                }
        }
        return res
}

# 2
func dieSimulator(n int, rollMax []int) int {
        dp := [7][16]int{} // 以j结尾,出现k次
        for j := 1; j <= 6; j++ {
                dp[j][1] = 1
        }
        for i := 2; i <= n; i++ {
                temp := [7][16]int{}
                for j := 1; j <= 6; j++ {
                        for k := 1; k <= rollMax[j-1] && k <= i; k++ {
```

(续下页)

(接上页)

```
                                if k == 1 { // 不同情况
                                        for a := 1; a <= 6; a++ {
                                                if a != j { // 考虑不同的情况
                                                        for b := 1; b <= rollMax[a-1]␣
↪&& b <= i-1; b++ {
                                                                temp[j][k] =␣
↪(temp[j][k] + dp[a][b]) % 1000000007
                                                        }
                                                }
                                        }
                                } else { // 直接同上一个
                                        temp[j][k] = dp[j][k-1]
                                }
                        }
                }
                dp = temp
        }
        res := 0
        for j := 1; j <= 6; j++ {
                for k := 1; k <= 15; k++ {
                        res = (res + dp[j][k]) % 1000000007
                }
        }
        return res
}
```

38.9 1227. 飞机座位分配概率 (2)

- 题目

```
有 n 位乘客即将登机，飞机正好有 n 个座位。第一位乘客的票丢了，他随便选了一个座位坐下。
剩下的乘客将会：如果他们自己的座位还空着，就坐到自己的座位上，
    当他们自己的座位被占用时，随机选择其他座位
第 n 位乘客坐在自己的座位上的概率是多少？
示例 1：输入：n = 1 输出：1.00000
解释：第一个人只会坐在自己的位置上。
示例 2：输入：n = 2 输出：0.50000
解释：在第一个人选好座位坐下后，第二个人坐在自己的座位上的概率是 0.5。
提示：1 <= n <= 10^5
```

- 解题思路

```
func nthPersonGetsNthSeat(n int) float64 {
        if n == 1 {
                return 1
        }
        return 0.5
}

# 2
func nthPersonGetsNthSeat(n int) float64 {
        res := 1.0
        sum := 1.0
        // f(n) = 1/n * (f(n-1)+f(n-2)+f(n-3)+...+f(2)+1)
        for i := 2; i <= n; i++ {
                nth := 1.0 / float64(i)
                res = nth * sum
                sum += res
        }
        return res
}
```

38.10 1233. 删除子文件夹 (2)

- 题目

```
你是一位系统管理员，手里有一份文件夹列表 folder，你的任务是要删除该列表中的所有␣
↪子文件夹，
并以 任意顺序 返回剩下的文件夹。
我们这样定义「子文件夹」：
如果文件夹folder[i]位于另一个文件夹folder[j]下，那么folder[i]就是folder[j]的子文件夹。
文件夹的「路径」是由一个或多个按以下格式串联形成的字符串：
/后跟一个或者多个小写英文字母。
例如，/leetcode和/leetcode/problems都是有效的路径，而空字符串和/不是。
示例 1：输入：folder = ["/a","/a/b","/c/d","/c/d/e","/c/f"]
输出：["/a","/c/d","/c/f"]
解释："/a/b/" 是 "/a" 的子文件夹，而 "/c/d/e" 是 "/c/d" 的子文件夹。
示例 2：输入：folder = ["/a","/a/b/c","/a/b/d"] 输出：["/a"]
解释：文件夹 "/a/b/c" 和 "/a/b/d/" 都会被删除，因为它们都是 "/a" 的子文件夹。
示例 3：输入：folder = ["/a/b/c","/a/b/d","/a/b/ca"]
输出：["/a/b/c","/a/b/ca","/a/b/d"]
提示：1 <= folder.length<= 4 * 10^4
2 <= folder[i].length <= 100
folder[i]只包含小写字母和 /
```

(续下页)

(接上页)

```
folder[i]总是以字符 /起始
每个文件夹名都是唯一的
```

- 解题思路

```
func removeSubfolders(folder []string) []string {
        sort.Strings(folder)
        res := make([]string, 0)
        res = append(res, folder[0])
        prev := folder[0]
        for i := 1; i < len(folder); i++ {
                // 加'/'避免/a/b => /a/bb的情况
                if strings.HasPrefix(folder[i], prev+"/") == false {
                        res = append(res, folder[i])
                        prev = folder[i]
                }
        }
        return res
}

# 2
func removeSubfolders(folder []string) []string {
        res := make([]string, 0)
        m := make(map[string]bool)
        for i := 0; i < len(folder); i++ {
                m[folder[i]] = true
        }
        arr := make([]int, len(folder))
        for i := 0; i < len(folder); i++ {
                for j := 0; j < len(folder[i]); j++ {
                        if folder[i][j] == '/' && j > 0 && m[folder[i][:j]] == true {
                                arr[i] = 1
                                break
                        }
                }
        }
        for i := 0; i < len(arr); i++ {
                if arr[i] == 0 {
                        res = append(res, folder[i])
                }
        }
        return res
}
```

38.11 1234. 替换子串得到平衡字符串 (2)

- 题目

```
有一个只含有'Q', 'W', 'E','R'四种字符，且长度为 n的字符串。
假如在该字符串中，这四个字符都恰好出现n/4次，那么它就是一个「平衡字符串」。
给你一个这样的字符串 s，请通过「替换一个子串」的方式，使原字符串 s␣
↪变成一个「平衡字符串」。
你可以用和「待替换子串」长度相同的任何 其他字符串来完成替换。
请返回待替换子串的最小可能长度。
如果原字符串自身就是一个平衡字符串，则返回 0。
示例 1：输入：s = "QWER" 输出：0
解释：s 已经是平衡的了。
示例 2：输入：s = "QQWE" 输出：1
解释：我们需要把一个 'Q' 替换成 'R'，这样得到的 "RQWE" (或 "QRWE") 是平衡的。
示例 3：输入：s = "QQQW" 输出：2
解释：我们可以把前面的 "QQ" 替换成 "ER"。
示例 4：输入：s = "QQQQ" 输出：3
解释：我们可以替换后 3 个 'Q'，使 s = "QWER"。
提示：1 <= s.length <= 10^5
s.length是4的倍数
s中只含有'Q', 'W', 'E','R'四种字符
```

- 解题思路

```
func balancedString(s string) int {
        n := len(s)
        target := n / 4
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
        }
        if m['Q'] == target && m['W'] == target && m['E'] == target && m['R'] ==␣
↪target {
                return 0
        }
        res := n
        left := 0
        for right := 0; right < n; right++ {
                m[s[right]]--
                for left < n && left <= right && // [left,
↪right]之间子串都是需要替换的（减去），使得m[x]<=target
                        m['Q'] <= target && m['W'] <= target &&
                        m['E'] <= target && m['R'] <= target {
```

(续下页)

(接上页)

```
                        res = min(res, right-left+1)
                        m[s[left]]++
                        left++
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func balancedString(s string) int {
        n := len(s)
        target := n / 4
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
        }
        if m['Q'] == target && m['W'] == target && m['E'] == target && m['R'] ==␣
↪target {
                return 0
        }
        res := n
        left := 0
        right := 0
        for left < n {
                if m['Q'] > target || m['W'] > target || m['E'] > target || m['R'] >␣
↪target {
                        if right < n {
                                m[s[right]]--
                                right++
                        } else {
                                break
                        }
                } else {
                        res = min(res, right-left)
                        m[s[left]]++
                        left++
```

(续下页)

(接上页)

```
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

38.12 1238. 循环码排列 (2)

- 题目

```
给你两个整数n 和 start。你的任务是返回任意 (0,1,2,,...,2^n-1) 的排列 p，并且满足：
p[0] = start
p[i] 和 p[i+1]的二进制表示形式只有一位不同
p[0] 和 p[2^n -1]的二进制表示形式也只有一位不同
示例 1：输入：n = 2, start = 3 输出：[3,2,0,1]
解释：这个排列的二进制表示是 (11,10,00,01)
     所有的相邻元素都有一位是不同的，另一个有效的排列是 [3,1,0,2]
示例 2：输出：n = 3, start = 2 输出：[2,6,7,5,4,0,1,3]
解释：这个排列的二进制表示是 (010,110,111,101,100,000,001,011)
提示：1 <= n <= 16
0 <= start<2^n
```

- 解题思路

```
func circularPermutation(n int, start int) []int {
        total := 1 << n
        res := make([]int, total)
        for i := 0; i < total; i++ {
                // 计算格雷码：leetcode89.格雷编码
                res[i] = i ^ (i >> 1) ^ start // 在计算格雷码的基础上，再与start异或
        }
        return res
}

# 2
func circularPermutation(n int, start int) []int {
```

(续下页)

(接上页)

```
        total := 1 << n
        res := []int{0, 1}
        for i := 2; i <= n; i++ {
                // 计算格雷码：leetcode89.格雷编码
                for j := len(res) - 1; j >= 0; j-- {
                        res = append(res, res[j]+(1<<(i-1)))
                }
        }
        index := 0
        for i := 0; i < total; i++ {
                if res[i] == start {
                        index = i
                        break
                }
        }
        return append(res[index:], res[:index]...)
}
```

38.13 1239. 串联字符串的最大长度 (3)

- 题目

```
给定一个字符串数组 arr，字符串 s 是将 arr 某一子序列字符串连接所得的字符串，
如果 s 中的每一个字符都只出现过一次，那么它就是一个可行解。
请返回所有可行解 s 中最长长度。
示例 1：输入：arr = ["un","iq","ue"] 输出：4
解释：所有可能的串联组合是 "","un","iq","ue","uniq" 和 "ique"，最大长度为 4。
示例 2：输入：arr = ["cha","r","act","ers"] 输出：6
解释：可能的解答有 "chaers" 和 "acters"。
示例 3：输入：arr = ["abcdefghijklmnopqrstuvwxyz"] 输出：26
提示：1 <= arr.length <= 16
1 <= arr[i].length <= 26
arr[i]中只含有小写英文字母
```

- 解题思路

```
var res []string

func maxLength(arr []string) int {
        res = make([]string, 0)
        dfs(arr, "", 0)
        maxValue := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(res); i++ {
                if check(res[i]) == true && len(res[i]) > maxValue {
                        maxValue = len(res[i])
                }
        }
        return maxValue
}

func dfs(arr []string, path string, index int) {
        res = append(res, path)
        for i := index; i < len(arr); i++ {
                dfs(arr, path+arr[i], i+1)
        }
}

func check(s string) bool {
        arr := [26]int{}
        for i := 0; i < len(s); i++ {
                value := int(s[i] - 'a')
                arr[value]++
                if arr[value] > 1 {
                        return false
                }
        }
        return true
}

# 2
var res int

func maxLength(arr []string) int {
        res = 0
        dfs(arr, "", 0)
        return res
}

func dfs(arr []string, path string, index int) {
        if check(path) == true && len(path) > res {
                res = len(path)
        }
        for i := index; i < len(arr); i++ {
                dfs(arr, path+arr[i], i+1)
        }
```

(续下页)

(接上页)

```
}

func check(s string) bool {
        arr := [26]int{}
        for i := 0; i < len(s); i++ {
                value := int(s[i] - 'a')
                arr[value]++
                if arr[value] > 1 {
                        return false
                }
        }
        return true
}

# 3
var res int

func maxLength(arr []string) int {
        res = 0
        dfs(arr, "", 0)
        return res
}

func dfs(arr []string, path string, index int) {
        for i := index; i < len(arr); i++ {
                newStr := path + arr[i]
                if check(newStr) == true {
                        if len(newStr) > res {
                                res = len(newStr)
                        }
                        dfs(arr, path+arr[i], i+1)
                }
        }
}

func check(s string) bool {
        arr := [26]int{}
        for i := 0; i < len(s); i++ {
                value := int(s[i] - 'a')
                arr[value]++
                if arr[value] > 1 {
                        return false
                }
```

(续下页)

(接上页)

```
        }
        return true
}
```

38.14 1247. 交换字符使得字符串相同 (1)

- 题目

```
有两个长度相同的字符串s1 和s2，且它们其中只含有字符"x" 和"y"，
你需要通过「交换字符」的方式使这两个字符串相同。
每次「交换字符」的时候，你都可以在两个字符串中各选一个字符进行交换。
交换只能发生在两个不同的字符串之间，绝对不能发生在同一个字符串内部。
也就是说，我们可以交换s1[i] 和s2[j]，但不能交换s1[i] 和s1[j]。
最后，请你返回使 s1 和 s2␣
↪相同的最小交换次数，如果没有方法能够使得这两个字符串相同，则返回-1 。
示例 1：输入：s1 = "xx", s2 = "yy" 输出：1
解释：交换 s1[0] 和 s2[1]，得到 s1 = "yx", s2 = "yx"。
示例 2：输入：s1 = "xy", s2 = "yx" 输出：2
解释：交换 s1[0] 和 s2[0]，得到 s1 = "yy", s2 = "xx" 。
交换 s1[0] 和 s2[1]，得到 s1 = "xy", s2 = "xy" 。
注意，你不能交换 s1[0] 和 s1[1] 使得 s1 变成 "yx
↪"，因为我们只能交换属于两个不同字符串的字符。
示例 3：输入：s1 = "xx", s2 = "xy" 输出：-1
示例 4：输入：s1 = "xxyyxyxyxx", s2 = "xyyxyxxxyx" 输出：4
提示：1 <= s1.length, s2.length <= 1000
s1, s2只包含'x'或'y'。
```

- 解题思路

```
func minimumSwap(s1 string, s2 string) int {
        a, b := 0, 0
        for i := 0; i < len(s1); i++ {
                if s1[i] == s2[i] { // 相同不需要交换
                        continue
                }
                if s1[i] == 'x' {
                        a++
                } else {
                        b++
                }
        }
        // a => x y
```

(续下页)

(接上页)

```
        // b => y x
        if a%2+b%2 == 1 {
                return -1
        }
        return a/2 + b/2 + 2*(a%2)
}
```

38.15 1248. 统计「优美子数组」(4)

- 题目

```
给你一个整数数组 nums 和一个整数 k。
如果某个 连续 子数组中恰好有 k 个奇数数字，我们就认为这个子数组是「优美子数组」。
请返回这个数组中「优美子数组」的数目。
示例 1: 输入: nums = [1,1,2,1,1], k = 3 输出: 2
解释: 包含 3 个奇数的子数组是 [1,1,2,1] 和 [1,2,1,1] 。
示例 2: 输入: nums = [2,4,6], k = 1 输出: 0
解释: 数列中不包含任何奇数，所以不存在优美子数组。
示例 3: 输入: nums = [2,2,2,1,2,2,1,2,2,2], k = 2 输出: 16
提示: 1 <= nums.length <= 50000
    1 <= nums[i] <= 10^5
    1 <= k <= nums.length
```

- 解题思路

```
func numberOfSubarrays(nums []int, k int) int {
        res := 0
        arr := make([]int, 0)
        arr = append(arr, -1)
        for i := 0; i < len(nums); i++ {
                if nums[i]%2 == 1 {
                        arr = append(arr, i)
                }
        }
        arr = append(arr, len(nums))
        for i := 1; i+k < len(arr); i++ {
                res = res + (arr[i]-arr[i-1])*(arr[i+k]-arr[i+k-1])
        }
        return res
}

# 2
```

(续下页)

(接上页)

```
func numberOfSubarrays(nums []int, k int) int {
        res := 0
        arr := make([]int, len(nums)+1)
        arr[0] = 1
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]%2
                arr[sum]++
                if sum >= k {
                        res = res + arr[sum-k]
                }
        }
        return res
}

# 3
func numberOfSubarrays(nums []int, k int) int {
        res := 0
        left, right := 0, 0
        count := 0
        for right < len(nums) {
                if nums[right]%2 == 1 {
                        count++
                }
                right++
                if count == k {
                        temp := right
                        for right < len(nums) && nums[right]%2 == 0 {
                                right++
                        }
                        totalRight := right - temp + 1
                        totalLeft := 1
                        for nums[left]%2 == 0 {
                                left++
                                totalLeft++
                        }
                        res = res + totalLeft*totalRight
                        count--
                        left++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 4
func numberOfSubarrays(nums []int, k int) int {
        res := 0
        dp := make([]int, 0)
        count := 0
        for i := 0; i < len(nums); i++ {
                count++
                if nums[i]%2 == 1 {
                        dp = append(dp, count)
                        count = 0
                }
                if len(dp) >= k {
                        res = res + dp[len(dp)-k]
                }
        }
        return res
}
```

38.16 1249. 移除无效的括号 (2)

- 题目

```
给你一个由 '('、')' 和小写字母组成的字符串 s。
你需要从字符串中删除最少数目的 '(' 或者 ')'␣
↪(可以删除任意位置的括号)，使得剩下的「括号字符串」有效。
请返回任意一个合法字符串。
有效「括号字符串」应当符合以下 任意一条 要求：
    空字符串或只包含小写字母的字符串
    可以被写作 AB（A 连接 B）的字符串，其中 A 和 B 都是有效「括号字符串」
    可以被写作 (A) 的字符串，其中 A 是一个有效的「括号字符串」
示例 1：输入：s = "lee(t(c)o)de)" 输出："lee(t(c)o)de"
解释："lee(t(co)de)" , "lee(t(c)ode)" 也是一个可行答案。
示例 2：输入：s = "a)b(c)d" 输出："ab(c)d"
示例 3：输入：s = "))((" 输出：""
解释：空字符串也是有效的
示例 4：输入：s = "(a(b(c)d)" 输出："a(b(c)d)"
提示：1 <= s.length <= 10^5
    s[i] 可能是 '('、')' 或英文小写字母
```

- 解题思路

```
func minRemoveToMakeValid(s string) string {
        arr := []byte(s)
        sum := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] == '(' {
                        sum++
                } else if arr[i] == ')' {
                        sum--
                        if sum < 0 {
                                arr[i] = ' '
                                sum = 0
                        }
                }
        }
        sum = 0
        for i := len(arr) - 1; i >= 0; i-- {
                if arr[i] == ')' {
                        sum++
                } else if arr[i] == '(' {
                        sum--
                        if sum < 0 {
                                arr[i] = ' '
                                sum = 0
                        }
                }
        }
        return strings.ReplaceAll(string(arr), " ", "")
}

# 2
func minRemoveToMakeValid(s string) string {
        arr := []byte(s)
        stack := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                if arr[i] == '(' {
                        stack = append(stack, i)
                } else if arr[i] == ')' {
                        if len(stack) == 0 {
                                arr[i] = ' '
                        } else {
                                stack = stack[:len(stack)-1]
                        }
                }
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(stack); i++ {
                arr[stack[i]] = ' '
        }
        return strings.ReplaceAll(string(arr), " ", "")
}
```

38.17 1253. 重构 2 行二进制矩阵 (2)

- 题目

```
给你一个2行 n 列的二进制数组：
矩阵是一个二进制矩阵，这意味着矩阵中的每个元素不是0就是1。
第 0 行的元素之和为upper。
第 1 行的元素之和为 lower。
第 i 列（从 0 开始编号）的元素之和为colsum[i]，colsum是一个长度为n的整数数组。
你需要利用upper，lower和colsum来重构这个矩阵，并以二维整数数组的形式返回它。
如果有多个不同的答案，那么任意一个都可以通过本题。
如果不存在符合要求的答案，就请返回一个空的二维数组。
示例 1：输入：upper = 2, lower = 1, colsum = [1,1,1] 输出：[[1,1,0],[0,0,1]]
解释：[[1,0,1],[0,1,0]] 和 [[0,1,1],[1,0,0]] 也是正确答案。
示例 2：输入：upper = 2, lower = 3, colsum = [2,2,1,1] 输出：[]
示例 3：输入：upper = 5, lower = 5, colsum = [2,1,2,0,1,0,1,2,0,1]
输出：[[1,1,1,0,1,0,0,1,0,0],[1,0,1,0,0,0,1,1,0,1]]
提示：1 <= colsum.length <= 10^5
0 <= upper, lower <= colsum.length
0 <= colsum[i] <= 2
```

- 解题思路

```
func reconstructMatrix(upper int, lower int, colsum []int) [][]int {
        res := make([][]int, 0)
        total := 0
        two := 0
        for i := 0; i < len(colsum); i++ {
                total = total + colsum[i]
                if colsum[i] == 2 {
                        two++
                }
        }
        if total != upper+lower || two > upper || two > lower {
                return nil
        }
```

(续下页)

(接上页)

```
        up := make([]int, len(colsum))
        down := make([]int, len(colsum))
        upper = upper - two // 上面数组单独1的个数
        for i := 0; i < len(colsum); i++ {
                if colsum[i] == 2 { // 2=>各填充1
                        up[i] = 1
                        down[i] = 1
                        lower--
                } else if colsum[i] == 1 {
                        if upper > 0 { // 先填充上面数组
                                up[i] = 1
                                upper--
                        } else {
                                down[i] = 1
                        }
                }
        }
        res = append(res, up, down)
        return res
}

# 2
func reconstructMatrix(upper int, lower int, colsum []int) [][]int {
        res := make([][]int, 0)
        up := make([]int, len(colsum))
        down := make([]int, len(colsum))
        upSum := 0
        lowSum := 0
        total := 0
        for i := 0; i < len(colsum); i++ {
                total = total + colsum[i]
                if colsum[i] == 2 { // 2=>各填充1
                        up[i] = 1
                        down[i] = 1
                        upSum++
                        lowSum++
                }
        }
        if upSum > upper || lowSum > lower || total != upper+lower {
                return nil
        }
        for i := 0; i < len(colsum); i++ {
                if colsum[i] == 1 {
```

(续下页)

(接上页)

```
                        if upSum < upper {
                                up[i] = 1
                                upSum++
                        } else {
                                down[i] = 1
                        }
                }
        }
        res = append(res, up, down)
        return res
}
```

38.18 1254. 统计封闭岛屿的数目 (2)

- 题目

```
有一个二维矩阵 grid ，每个位置要么是陆地（记号为 0 ）要么是水域（记号为 1 ）。
我们从一块陆地出发，每次可以往上下左右 4␣
↪个方向相邻区域走，能走到的所有陆地区域，我们将其称为一座「岛屿」。
如果一座岛屿 完全␣
↪由水域包围，即陆地边缘上下左右所有相邻区域都是水域，那么我们将其称为 「封闭岛屿」。
请返回封闭岛屿的数目。
示例 1：输入：grid = [[1,1,1,1,1,1,1,0],[1,0,0,0,0,1,1,0],[1,0,1,0,1,1,1,0],
[1,0,0,0,0,1,0,1],[1,1,1,1,1,1,1,0]] 输出：2
解释：灰色区域的岛屿是封闭岛屿，因为这座岛屿完全被水域包围（即被 1 区域包围）。
示例 2：输入：grid = [[0,0,1,0,0],[0,1,0,1,0],[0,1,1,1,0]] 输出：1
示例 3：输入：grid = [[1,1,1,1,1,1,1],
             [1,0,0,0,0,0,1],
             [1,0,1,1,1,0,1],
             [1,0,1,0,1,0,1],
             [1,0,1,1,1,0,1],
             [1,0,0,0,0,0,1],
             [1,1,1,1,1,1,1]]
输出：2
提示：  1 <= grid.length, grid[0].length <= 100
    0 <= grid[i][j] <=1
```

- 解题思路

```
func closedIsland(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 0 && dfs(grid, i, j) == true {
                                res++
                        }
                }
        }
        return res
}

func dfs(grid [][]int, i, j int) bool {
        if i < 0 || i >= len(grid) || j < 0 || j >= len(grid[0]) {
                return false
        }
        if grid[i][j] == 1 {
                return true
        }
        grid[i][j] = 1
        up := dfs(grid, i, j+1)
        down := dfs(grid, i, j-1)
        left := dfs(grid, i-1, j)
        right := dfs(grid, i+1, j)
        return up && down && left && right
}

# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func closedIsland(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 0 {
                                flag := true
                                queue := make([][2]int, 0)
                                queue = append(queue, [2]int{i, j})
                                if i == 0 || j == 0 || i == len(grid)-1 || j ==␣
↪len(grid[i])-1 {
                                        flag = false
                                }
                                for len(queue) > 0 {
                                        node := queue[0]
                                        queue = queue[1:]
```

(续下页)

(接上页)

```
                                        for k := 0; k < 4; k++ {
                                                x := dx[k] + node[0]
                                                y := dy[k] + node[1]
                                                if x < 0 || x >= len(grid) || y < 0␣
↪|| y >= len(grid[i]) {
                                                        continue
                                                }
                                                if grid[x][y] == 1 {
                                                        continue
                                                }
                                                if x == 0 || y == 0 || x == len(grid)-
↪1 || y == len(grid[i])-1 {
                                                        flag = false
                                                }
                                                queue = append(queue, [2]int{x, y})
                                                grid[x][y] = 1
                                        }
                                }
                                if flag == true {
                                        res++
                                }
                        }
                }
        }
        return res
}
```

38.19 1261. 在受污染的二叉树中查找元素 (2)

- 题目

```
给出一个满足下述规则的二叉树：
root.val == 0
如果 treeNode.val == x 且treeNode.left != null，那么treeNode.left.val == 2 * x + 1
如果 treeNode.val == x 且 treeNode.right != null，那么treeNode.right.val == 2 * x + 2
现在这个二叉树受到「污染」，所有的treeNode.val都变成了-1。
请你先还原二叉树，然后实现FindElements类：
FindElements(TreeNode* root)用受污染的二叉树初始化对象，你需要先把它还原。
bool find(int target)判断目标值target是否存在于还原后的二叉树中并返回结果。
示例 1：输入：  ["FindElements","find","find"]
[[[-1,null,-1]],[1],[2]]
输出：[null,false,true]
```

(续下页)

(接上页)

```
解释：FindElements findElements = new FindElements([-1,null,-1]);
findElements.find(1); // return False
findElements.find(2); // return True
示例 2：输入：  ["FindElements","find","find","find"]
[[[-1,-1,-1,-1,-1]],[1],[3],[5]]
输出：[null,true,true,false]
解释：FindElements findElements = new FindElements([-1,-1,-1,-1,-1]);
findElements.find(1); // return True
findElements.find(3); // return True
findElements.find(5); // return False
示例 3：输入：  ["FindElements","find","find","find","find"]
[[[-1,null,-1,-1,null,-1]],[2],[3],[4],[5]]
输出：[null,true,false,false,true]
解释：FindElements findElements = new FindElements([-1,null,-1,-1,null,-1]);
findElements.find(2); // return True
findElements.find(3); // return False
findElements.find(4); // return False
findElements.find(5); // return True
提示：TreeNode.val == -1
二叉树的高度不超过20
节点的总数在[1,10^4]之间
调用find()的总次数在[1,10^4]之间
0 <= target <= 10^6
```

- 解题思路

```
type FindElements struct {
        m map[int]bool
}

type Node struct {
        root  *TreeNode
        index int
}

func Constructor(root *TreeNode) FindElements {
        if root == nil {
                return FindElements{m: map[int]bool{}}
        }
        m := make(map[int]bool)
        m[0] = true
        queue := make([]Node, 0)
        queue = append(queue, Node{root: root, index: 0})
        for len(queue) > 0 {
```

(续下页)

(接上页)

```
                node := queue[0]
                queue = queue[1:]
                var temp Node
                if node.root.Left != nil {
                        temp = Node{
                                root:  node.root.Left,
                                index: node.index*2 + 1,
                        }
                        m[node.index*2+1] = true
                }
                if node.root.Right != nil {
                        temp = Node{
                                root:  node.root.Right,
                                index: node.index*2 + 2,
                        }
                        m[node.index*2+2] = true
                }
                queue = append(queue, temp)
        }
        return FindElements{m: m}
}

func (this *FindElements) Find(target int) bool {
        return this.m[target]
}

# 2
type FindElements struct {
        num []int
}

type Node struct {
        root  *TreeNode
        index int
}

func Constructor(root *TreeNode) FindElements {
        if root == nil {
                return FindElements{num: []int{}}
        }
        num := make([]int, 0)
        num = append(num, 0)
        queue := make([]Node, 0)
```

(续下页)

(接上页)

```
        queue = append(queue, Node{
                root:  root,
                index: 0,
        })
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.root.Left != nil {
                        temp := Node{
                                root:  node.root.Left,
                                index: node.index*2 + 1,
                        }
                        num = append(num, node.index*2+1)
                        queue = append(queue, temp)
                }
                if node.root.Right != nil {
                        temp := Node{
                                root:  node.root.Right,
                                index: node.index*2 + 2,
                        }
                        num = append(num, node.index*2+2)
                        queue = append(queue, temp)
                }
        }
        return FindElements{num: num}
}

func (this *FindElements) Find(target int) bool {
        for i := 0; i < len(this.num); i++ {
                if this.num[i] == target {
                        return true
                }
        }
        return false
}
```

38.20 1262. 可被三整除的最大和 (1)

- 题目

```
给你一个整数数组nums，请你找出并返回能被三整除的元素最大和。
示例 1：输入：nums = [3,6,5,1,8] 输出：18
解释：选出数字 3, 6, 1 和 8，它们的和是 18（可被 3 整除的最大和）。
示例 2：输入：nums = [4] 输出：0
解释：4 不能被 3 整除，所以无法选出数字，返回 0。
示例 3：输入：nums = [1,2,3,4,4] 输出：12
解释：选出数字 1, 3, 4 以及 4，它们的和是 12（可被 3 整除的最大和）。
提示：1 <= nums.length <= 4 * 10^4
1 <= nums[i] <= 10^4
```

- 解题思路

```
func maxSumDivThree(nums []int) int {
        dp := [3]int{}
        for i := 0; i < len(nums); i++ {
                for _, v := range dp {
                        value := v + nums[i]
                        dp[value%3] = max(dp[value%3], value)
                }
        }
        return dp[0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

38.21 1267. 统计参与通信的服务器 (1)

- 题目

```
这里有一幅服务器分布图，服务器的位置标识在m * n的整数矩阵网格grid中，
1 表示单元格上有服务器，0 表示没有。
如果两台服务器位于同一行或者同一列，我们就认为它们之间可以进行通信。
请你统计并返回能够与至少一台其他服务器进行通信的服务器的数量。
示例 1：输入：grid = [[1,0],[0,1]] 输出：0
```

(续下页)

(接上页)

```
解释：没有一台服务器能与其他服务器进行通信。
示例 2：输入：grid = [[1,0],[1,1]] 输出：3
解释：所有这些服务器都至少可以与一台别的服务器进行通信。
示例 3：输入：grid = [[1,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1]] 输出：4
解释：第一行的两台服务器互相通信，第三列的两台服务器互相通信，但右下角的服务器无法与其他服务器通信。
提示：m == grid.length
n == grid[i].length
1 <= m <= 250
1 <= n <= 250
grid[i][j] == 0 or 1
```

- 解题思路

```
func countServers(grid [][]int) int {
        a := make(map[int]int) // 行
        b := make(map[int]int) // 列
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 1 {
                                a[i]++
                                b[j]++
                        }
                }
        }
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 1 && (a[i] > 1 || b[j] > 1) {
                                res++
                        }
                }
        }
        return res
}
```

38.22 1268. 搜索推荐系统 (2)

- 题目

```
给你一个产品数组products和一个字符串searchWord，products␣
↪数组中每个产品都是一个字符串。
请你设计一个推荐系统，在依次输入单词searchWord 的每一个字母后，推荐products␣
```

(续下页)

(接上页)

```
↪数组中前缀与searchWord 相同的最多三个产品。
如果前缀相同的可推荐产品超过三个，请按字典序返回最小的三个。
请你以二维列表的形式，返回在输入searchWord每个字母后相应的推荐产品的列表。
示例 1：输入：products = ["mobile","mouse","moneypot","monitor","mousepad"],␣
↪searchWord = "mouse"
输出：[
["mobile","moneypot","monitor"],
["mobile","moneypot","monitor"],
["mouse","mousepad"],
["mouse","mousepad"],
["mouse","mousepad"]
]
解释：按字典序排序后的产品列表是 ["mobile","moneypot","monitor","mouse","mousepad"]
输入 m 和 mo，由于所有产品的前缀都相同，所以系统返回字典序最小的三个产品 ["mobile",
↪"moneypot","monitor"]
输入 mou， mous 和 mouse 后系统都返回 ["mouse","mousepad"]
示例 2：输入：products = ["havana"], searchWord = "havana"
输出：[["havana"],["havana"],["havana"],["havana"],["havana"],["havana"]]
示例 3：输入：products = ["bags","baggage","banner","box","cloths"], searchWord =
↪"bags"
输出：[["baggage","bags","banner"],["baggage","bags","banner"],["baggage","bags"],[
↪"bags"]]
示例 4：输入：products = ["havana"], searchWord = "tatiana" 输出：[[],[],[],[],[],[],
↪[]]
提示：1 <= products.length <= 1000
1 <= Σ products[i].length <= 2 * 10^4
products[i]中所有的字符都是小写英文字母。
1 <= searchWord.length <= 1000
searchWord中所有字符都是小写英文字母。
```

- 解题思路

```
func suggestedProducts(products []string, searchWord string) [][]string {
        sort.Strings(products)
        res := make([][]string, 0)
        for i := 0; i < len(searchWord); i++ {
                target := searchWord[:i+1]
                arr := make([]string, 0)
                for j := 0; j < len(products); j++ {
                        if len(products[j]) >= len(target) && products[j][:i+1] ==␣
↪target {
                                arr = append(arr, products[j])
                        }
                        if len(arr) == 3 {
```

(续下页)

(接上页)

```
                                break
                        }
                }
                res = append(res, arr)
        }
        return res
}

# 2
func suggestedProducts(products []string, searchWord string) [][]string {
        res := make([][]string, len(searchWord))
        t := &Trie{}
        for i := 0; i < len(products); i++ {
                t.Insert(products[i])
        }
        for i := 0; i < len(searchWord); i++ {
                res[i] = t.StartsWith(searchWord[:i+1])
        }
        return res
}

type Trie struct {
        next [26]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        str  string
}

// 插入word
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next: [26]*Trie{},
                        }
                }
                temp = temp.next[value]
        }
        temp.str = word
}

// 查找前缀
func (this *Trie) StartsWith(prefix string) []string {
```

(续下页)

(接上页)

```
        temp := this
        for _, v := range prefix {
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return nil
                }
        }
        res := make([]string, 0)
        temp.dfs(&res)
        return res
}

func (this *Trie) dfs(res *[]string) {
        if len(*res) == 3 {
                return
        }
        if this.str != "" {
                *res = append(*res, this.str)
        }
        for _, v := range this.next {
                if len(*res) == 3 {
                        return
                }
                if v == nil {
                        continue
                }
                v.dfs(res)
        }
}
```

38.23 1276. 不浪费原料的汉堡制作方案 (1)

- 题目

```
圣诞活动预热开始啦，汉堡店推出了全新的汉堡套餐。为了避免浪费原料，请你帮他们制定合适的制作计划。
给你两个整数tomatoSlices和cheeseSlices，分别表示番茄片和奶酪片的数目。不同汉堡的原料搭配如下：
巨无霸汉堡：4 片番茄和 1 片奶酪
小皇堡：2 片番茄和1 片奶酪
请你以[total_jumbo, total_
↪small]（[巨无霸汉堡总数，小皇堡总数]）的格式返回恰当的制作方案，
使得剩下的番茄片tomatoSlices和奶酪片cheeseSlices的数量都是0。
如果无法使剩下的番茄片tomatoSlices和奶酪片cheeseSlices的数量为0，就请返回[]。
```

(续下页)

(接上页)

```
示例 1: 输入: tomatoSlices = 16, cheeseSlices = 7 输出: [1,6]
解释: 制作 1 个巨无霸汉堡和 6 个小皇堡需要 4*1 + 2*6 = 16 片番茄和 1 + 6 = 7␣
↪片奶酪。不会剩下原料。
示例 2: 输入: tomatoSlices = 17, cheeseSlices = 4 输出: []
解释: 只制作小皇堡和巨无霸汉堡无法用光全部原料。
示例 3: 输入: tomatoSlices = 4, cheeseSlices = 17 输出: []
解释: 制作 1 个巨无霸汉堡会剩下 16 片奶酪, 制作 2 个小皇堡会剩下 15 片奶酪。
示例 4: 输入: tomatoSlices = 0, cheeseSlices = 0 输出: [0,0]
示例 5: 输入: tomatoSlices = 2, cheeseSlices = 1 输出: [0,1]
提示: 0 <= tomatoSlices <= 10^7
0 <= cheeseSlices <= 10^7
```

- 解题思路

```
func numOfBurgers(tomatoSlices int, cheeseSlices int) []int {
        a, b := tomatoSlices, cheeseSlices
        c := a - 2*b
        if c%2 == 0 && c/2 <= b && 0 <= c {
                return []int{c / 2, b - c/2}
        }
        return nil
}
```

38.24 1277. 统计全为 1 的正方形子矩阵 (3)

- 题目

```
给你一个m * n的矩阵, 矩阵中的元素不是 0 就是 1, 请你统计并返回其中完全由 1 组成的␣
↪正方形 子矩阵的个数。
示例 1: 输入: matrix =
[
 [0,1,1,1],
 [1,1,1,1],
 [0,1,1,1]
]
输出: 15
解释:
边长为 1 的正方形有 10 个。
边长为 2 的正方形有 4 个。
边长为 3 的正方形有 1 个。
正方形的总数 = 10 + 4 + 1 = 15.
示例 2: 输入: matrix =
```

(续下页)

(接上页)

```
[
  [1,0,1],
  [1,1,0],
  [1,1,0]
]
输出：7
解释：边长为 1 的正方形有 6 个。
边长为 2 的正方形有 1 个。
正方形的总数 = 6 + 1 = 7.
提示：1 <= arr.length<= 300
1 <= arr[0].length<= 300
0 <= arr[i][j] <= 1
```

- 解题思路

```
func countSquares(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i == 0 || j == 0 {
                                dp[i][j] = matrix[i][j]
                        } else if matrix[i][j] == 0 {
                                dp[i][j] = 0
                        } else {
                                dp[i][j] = min(min(dp[i][j-1], dp[i-1][j]), dp[i-1][j-
↪1]) + 1
                        }
                        res = res + dp[i][j]
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
# 2
func countSquares(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        dp := make([]int, m)
        res := 0
        for i := 0; i < n; i++ {
                temp := make([]int, m)
                for j := 0; j < m; j++ {
                        if i == 0 || j == 0 {
                                temp[j] = matrix[i][j]
                        } else if matrix[i][j] == 0 {
                                temp[j] = 0
                        } else {
                                temp[j] = min(min(temp[j-1], dp[j]), dp[j-1]) + 1
                        }
                        res = res + temp[j]
                }
                copy(dp, temp)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func countSquares(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i > 0 && j > 0 && matrix[i][j] > 0 {
                                matrix[i][j] = min(min(matrix[i][j-1], matrix[i-
↪1][j]), matrix[i-1][j-1]) + 1
                        }
                        res = res + matrix[i][j]
                }
        }
```

(续下页)

(接上页)

```
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

38.25 1282. 用户分组 (2)

- 题目

```
有n位用户参加活动，他们的ID从 0 到 n - 1，每位用户都 恰好 属于某一用户组。
给你一个长度为 n 的数组groupSizes，其中包含每位用户所处的用户组的大小，
请你返回用户分组情况（存在的用户组以及每个组中用户的 ID）。
你可以任何顺序返回解决方案，ID␣
↪的顺序也不受限制。此外，题目给出的数据保证至少存在一种解决方案。
示例 1：输入：groupSizes = [3,3,3,3,3,1,3] 输出：[[5],[0,1,2],[3,4,6]]
解释：  其他可能的解决方案有 [[2,1,6],[5],[0,4,3]] 和 [[5],[0,6,2],[4,3,1]]。
示例 2：输入：groupSizes = [2,1,3,3,3,2] 输出：[[1],[0,5],[2,3,4]]
提示：groupSizes.length == n
1 <= n<= 500
1 <=groupSizes[i] <= n
```

- 解题思路

```
func groupThePeople(groupSizes []int) [][]int {
        res := make([][]int, 0)
        m := make(map[int][]int)
        for i := 0; i < len(groupSizes); i++ {
                m[groupSizes[i]] = append(m[groupSizes[i]], i)
        }
        for k, v := range m {
                for i := 0; i < len(v); i = i + k {
                        res = append(res, v[i:i+k])
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func groupThePeople(groupSizes []int) [][]int {
        res := make([][]int, 0)
        m := make(map[int][]int)
        for i := 0; i < len(groupSizes); i++ {
                m[groupSizes[i]] = append(m[groupSizes[i]], i)
                if groupSizes[i] == len(m[groupSizes[i]]) {
                        res = append(res, m[groupSizes[i]])
                        m[groupSizes[i]] = []int{}
                }
        }
        return res
}
```

38.26 1283. 使结果不超过阈值的最小除数 (1)

- 题目

```
给你一个整数数组nums 和一个正整数threshold␣
↪，你需要选择一个正整数作为除数，然后将数组里每个数都除以它，并对除法结果求和。
请你找出能够使上述结果小于等于阈值threshold的除数中 最小 的那个。
每个数除以除数后都向上取整，比方说 7/3 = 3 ， 10/2 = 5 。
题目保证一定有解。
示例 1：输入：nums = [1,2,5,9], threshold = 6 输出：5
解释：如果除数为 1 ，我们可以得到和为 17 (1+2+5+9) 。
如果除数为 4 ，我们可以得到和为 7 (1+1+2+3) 。如果除数为 5 ，和为 5 (1+1+1+2)。
示例 2：输入：nums = [2,3,5,7,11], threshold = 11 输出：3
示例 3：输入：nums = [19], threshold = 5 输出：4
提示：1 <= nums.length <= 5 * 10^4
1 <= nums[i] <= 10^6
nums.length <=threshold <= 10^6
```

- 解题思路

```
func smallestDivisor(nums []int, threshold int) int {
        left, right := 1, math.MaxInt32/10
        res := 0
        for left <= right {
                mid := left + (right-left)/2
                count := getValue(nums, mid)
                if count > threshold {
                        left = mid + 1
```

(续下页)

(接上页)

```
                } else {
                        right = mid - 1
                        res = mid
                }
        }
        return res
}

func getValue(nums []int, target int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                res = res + (nums[i]+target-1)/target // 向上取整
        }
        return res
}
```

38.27 1286. 字母组合迭代器 (2)

- 题目

```
请你设计一个迭代器类，包括以下内容：
一个构造函数，输入参数包括：一个有序且字符唯一的字符串characters（该字符串只包含小写英文字母）和一个
函数next()，按字典序返回长度为combinationLength 的下一个字母组合。
函数hasNext()，只有存在长度为combinationLength␣
↪的下一个字母组合时，才返回True；否则，返回 False。
示例：CombinationIterator iterator = new CombinationIterator("abc", 2); // 创建迭代器␣
↪iterator
iterator.next(); // 返回 "ab"
iterator.hasNext(); // 返回 true
iterator.next(); // 返回 "ac"
iterator.hasNext(); // 返回 true
iterator.next(); // 返回 "bc"
iterator.hasNext(); // 返回 false
提示：1 <= combinationLength <=characters.length <= 15
每组测试数据最多包含10^4次函数调用。
题目保证每次调用函数next时都存在下一个字母组合。
```

- 解题思路

```
type CombinationIterator struct {
        flag bool
        s    string
```

(续下页)

(接上页)

```
        arr  []int
}

func Constructor(characters string, combinationLength int) CombinationIterator {
        arr := make([]int, combinationLength)
        for i := 0; i < combinationLength; i++ {
                arr[i] = i
        }
        return CombinationIterator{
                flag: false,
                s:    characters,
                arr:  arr,
        }
}

func (this *CombinationIterator) Next() string {
        res := ""
        for i := 0; i < len(this.arr); i++ {
                res = res + string(this.s[this.arr[i]])
        }
        index := -1
        for i := len(this.arr) - 1; i >= 0; i-- {
                // 正常情况下：以abcdef 3 为例子
                // 0 1 5 => abf  下一个 acd
                // 6-3+2 = 5
                // 6-3+1 != 1 => index = 1
                // 然后：arr[index]+1，后面递增
                target := len(this.s) - len(this.arr) + i
                if this.arr[i] != target { // 从右到左边：找到
                        index = i
                        break
                }
        }
        if index == -1 { // 没有更大的
                this.flag = true
        } else {
                this.arr[index]++
                for i := index + 1; i < len(this.arr); i++ {
                        this.arr[i] = this.arr[i-1] + 1
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func (this *CombinationIterator) HasNext() bool {
        return this.flag == false
}

# 2
type CombinationIterator struct {
        arr   []string
        index int
}

func dfs(str string, k int, index int, cur string, res *[]string) {
        if len(cur) == k {
                *res = append(*res, cur)
                return
        }
        for i := index; i < len(str); i++ {
                dfs(str, k, i+1, cur+string(str[i]), res)
        }
}
func Constructor(characters string, combinationLength int) CombinationIterator {
        res := make([]string, 0)
        dfs(characters, combinationLength, 0, "", &res)
        return CombinationIterator{
                arr:   res,
                index: 0,
        }
}

func (this *CombinationIterator) Next() string {
        if this.index < len(this.arr) {
                this.index++
                return this.arr[this.index-1]
        }
        return ""
}

func (this *CombinationIterator) HasNext() bool {
        return this.index < len(this.arr)
}
```

38.28 1288. 删除被覆盖区间 (4)

- 题目

```
给你一个区间列表，请你删除列表中被其他区间所覆盖的区间。
只有当c <= a且b <= d时，我们才认为区间[a,b) 被区间[c,d) 覆盖。
在完成所有删除操作后，请你返回列表中剩余区间的数目。
示例：输入：intervals = [[1,4],[3,6],[2,8]] 输出：2
解释：区间 [3,6] 被区间 [2,8] 覆盖，所以它被删除了。
提示：1 <= intervals.length <= 1000
0 <= intervals[i][0] <intervals[i][1] <= 10^5
对于所有的i != j：intervals[i] != intervals[j]
```

- 解题思路

```
func removeCoveredIntervals(intervals [][]int) int {
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][0] == intervals[j][0] {
                        return intervals[i][1] > intervals[j][1]
                }
                return intervals[i][0] < intervals[j][0]
        })
        res := 0
        maxValue := intervals[0][1]
        for i := 1; i < len(intervals); i++ {
                if intervals[i][0] == intervals[i-1][0] { // 合并
                        res++
                        continue
                }
                if intervals[i][1] > maxValue {
                        maxValue = intervals[i][1] // 更新
                } else {
                        res++ // 合并
                }
        }
        return len(intervals) - res
}

# 2
func removeCoveredIntervals(intervals [][]int) int {
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][0] == intervals[j][0] {
                        return intervals[i][1] > intervals[j][1]
                }
```

(续下页)

(接上页)

```
                return intervals[i][0] < intervals[j][0]
        })
        res := 0
        left := intervals[0][0]
        right := intervals[0][1]
        for i := 1; i < len(intervals); i++ {
                l, r := intervals[i][0], intervals[i][1]
                if left <= l && r <= right { // 合并
                        res++
                }
                if right < r { // 更新
                        left = l
                        right = r
                }
        }
        return len(intervals) - res
}

# 3
func removeCoveredIntervals(intervals [][]int) int {
        sort.Slice(intervals, func(i, j int) bool {
                if intervals[i][0] == intervals[j][0] {
                        return intervals[i][1] > intervals[j][1]
                }
                return intervals[i][0] < intervals[j][0]
        })
        res := 0
        maxValue := 0
        for i := 0; i < len(intervals); i++ {
                if maxValue < intervals[i][1] {
                        res++
                        maxValue = intervals[i][1]
                }
        }
        return res
}

# 4
func removeCoveredIntervals(intervals [][]int) int {
        res := len(intervals)
        for i := 0; i < len(intervals); i++ {
                for j := 0; j < len(intervals); j++ {
                        if i != j && intervals[j][0] <= intervals[i][0] &&
```

(续下页)

(接上页)

```
                                intervals[i][1] <= intervals[j][1] {
                                res--
                                break
                        }
                }
        }
        return res
}
```

38.29 1291. 顺次数 (2)

- 题目

```
我们定义「顺次数」为：每一位上的数字都比前一位上的数字大 1 的整数。
请你返回由 [low, high] 范围内所有顺次数组成的 有序 列表（从小到大排序）。
示例 1：输出：low = 100, high = 300输出：[123,234]
示例 2：输出：low = 1000, high = 13000 输出：[1234,2345,3456,4567,5678,6789,12345]
提示：  10 <= low <= high <= 10^9
```

- 解题思路

```
func sequentialDigits(low int, high int) []int {
        res := make([]int, 0)
        for i := 1; i <= 9; i++ {
                num := i
                for j := i + 1; j <= 9; j++ {
                        num = num*10 + j
                        if num >= low && num <= high {
                                res = append(res, num)
                        }
                }
        }
        sort.Ints(res)
        return res
}

# 2
func sequentialDigits(low int, high int) []int {
        res := make([]int, 0)
        str := "123456789"
        for i := 0; i <= 9; i++ {
                for j := i + 1; j <= 9; j++ {
```

(续下页)

(接上页)

```
                        num, _ := strconv.Atoi(str[i:j])
                        if num >= low && num <= high {
                                res = append(res, num)
                        }
                }
        }
        sort.Ints(res)
        return res
}
```

38.30 1292. 元素和小于等于阈值的正方形的最大边长 (3)

- 题目

```
给你一个大小为m x n的矩阵mat和一个整数阈值threshold。
请你返回元素总和小于或等于阈值的正方形区域的最大边长；如果没有这样的正方形区域，则返回␣
↪0。
示例 1：输入：mat = [[1,1,3,2,4,3,2],[1,1,3,2,4,3,2],[1,1,3,2,4,3,2]], threshold = 4␣
↪输出：2
解释：总和小于或等于 4 的正方形的最大边长为 2，如图所示。
示例 2：输入：mat = [[2,2,2,2,2],[2,2,2,2,2],[2,2,2,2,2],[2,2,2,2,2],[2,2,2,2,2]],␣
↪threshold = 1 输出：0
示例 3：输入：mat = [[1,1,1,1],[1,0,0,0],[1,0,0,0],[1,0,0,0]], threshold = 6 输出：3
示例 4：输入：mat = [[18,70],[61,1],[25,85],[14,40],[11,96],[97,96],[63,45]],␣
↪threshold = 40184 输出：2
提示：1 <= m, n <= 300
m == mat.length
n == mat[i].length
0 <= mat[i][j] <= 10000
0 <= threshold<= 10^5
```

- 解题思路

```
func maxSideLength(mat [][]int, threshold int) int {
        n, m := len(mat), len(mat[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr[i+1][j+1] = mat[i][j] + arr[i+1][j] + arr[i][j+1] -␣
```

(续下页)

(接上页)

```
↪arr[i][j]
                }
        }
        res := min(n, m)
        left, right := 0, res // res可以为0
        for left <= right {
                mid := left + (right-left)/2
                if check(arr, mid, threshold) == true {
                        left = mid + 1
                        res = mid
                } else {
                        right = mid - 1
                }
        }
        return res
}

func check(arr [][]int, mid int, threshold int) bool {
        for i := 1; i+mid <= len(arr); i++ {
                for j := 1; j+mid <= len(arr[i]); j++ {
                        sum := arr[i+mid-1][j+mid-1] - arr[i+mid-1][j-1] - arr[i-
↪1][j+mid-1] + arr[i-1][j-1]
                        if sum <= threshold {
                                return true
                        }
                }
        }
        return false
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func maxSideLength(mat [][]int, threshold int) int {
        n, m := len(mat), len(mat[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
```

(续下页)

(接上页)

```
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr[i+1][j+1] = mat[i][j] + arr[i+1][j] + arr[i][j+1] -␣
↪arr[i][j]
                }
        }
        res := 0
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        for k := 0; i+k <= n && j+k <= m; k++ {
                                value := arr[i+k][j+k] - arr[i-1][j+k] - arr[i+k][j-
↪1] + arr[i-1][j-1]
                                if value <= threshold && res < k+1 {
                                        res = k + 1
                                }
                                if value > threshold {
                                        break
                                }
                        }
                }
        }
        return res
}

# 3
func maxSideLength(mat [][]int, threshold int) int {
        n, m := len(mat), len(mat[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr[i+1][j+1] = mat[i][j] + arr[i+1][j] + arr[i][j+1] -␣
↪arr[i][j]
                }
        }
        res := 0
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        for k := min(i, j); k > res; k-- {
                                value := arr[i][j] - arr[i-k][j] - arr[i][j-k] +␣
```

(续下页)

(接上页)

```
↪arr[i-k][j-k]
                                if value <= threshold && res < k {
                                        res = k
                                        break
                                }
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

38.31 1296. 划分数组为连续数字的集合 (3)

- 题目

```
给你一个整数数组nums和一个正整数k，请你判断是否可以把这个数组划分成一些由k个连续数字组成的集合。
如果可以，请返回True；否则，返回False。
注意：此题目与 846 重复
示例 1：输入：nums = [1,2,3,3,4,4,5,6], k = 4 输出：true
解释：数组可以分成 [1,2,3,4] 和 [3,4,5,6]。
示例 2：输入：nums = [3,2,1,2,3,4,3,4,5,9,10,11], k = 3 输出：true
解释：数组可以分成 [1,2,3] , [2,3,4] , [3,4,5] 和 [9,10,11]。
示例 3：输入：nums = [3,3,2,2,1,1], k = 3 输出：true
示例 4：输入：nums = [1,2,3,4], k = 3 输出：false
解释：数组不能分成几个大小为 3 的子数组。
提示：1 <= nums.length <= 10^5
1 <= nums[i] <= 10^9
1 <= k <= nums.length。
```

- 解题思路

```
func isPossibleDivide(nums []int, k int) bool {
        n := len(nums)
        if n%k != 0 {
                return false
```

(续下页)

(接上页)

```
        }
        if k == 1 {
                return true
        }
        sort.Ints(nums)
        for i := 0; i < n; i++ {
                if nums[i] >= 0 {
                        count := 1
                        for j := i + 1; j < n; j++ {
                                if nums[j] > nums[i]+count {
                                        break
                                }
                                if nums[j] >= 0 && nums[j] == nums[i]+count {
                                        nums[j] = -1
                                        count++
                                        if count == k {
                                                break
                                        }
                                }
                        }
                        if count != k {
                                return false
                        }
                        nums[i] = -1
                }
        }
        return true
}

# 2
func isPossibleDivide(nums []int, k int) bool {
        n := len(nums)
        if n%k != 0 {
                return false
        }
        if k == 1 {
                return true
        }
        arr := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] == 0 {
                        arr = append(arr,nums[i])
```

(续下页)

(接上页)

```
                }
                m[nums[i]]++
        }
        sort.Ints(arr)
        for i := 0; i < len(arr); i++ {
                if m[arr[i]] > 0 {
                        for j := 1; j < k; j++ {
                                value := arr[i] + j
                                m[value] = m[value] - m[arr[i]]
                                if m[value] < 0 {
                                        return false
                                }
                        }
                }
        }
        return true
}

# 3
func isPossibleDivide(nums []int, k int) bool {
        n := len(nums)
        if n%k != 0 {
                return false
        }
        if k == 1 {
                return true
        }
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        sort.Ints(nums)
        for i := 0; i < len(nums); i++ {
                value := m[nums[i]]
                if value > 0 {
                        for j := 0; j < k; j++ {
                                if m[nums[i]+j] < value {
                                        return false
                                }
                                m[nums[i]+j] = m[nums[i]+j] - value
                        }
                }
        }
```

(续下页)

(接上页)

```
        return true
}
```

38.32 1297. 子串的最大出现次数 (2)

- 题目

```
给你一个字符串s ，请你返回满足以下条件且出现次数最大的任意子串的出现次数：
子串中不同字母的数目必须小于等于 maxLetters 。
子串的长度必须大于等于minSize 且小于等于maxSize 。
示例 1：输入：s = "aababcaab", maxLetters = 2, minSize = 3, maxSize = 4 输出：2
解释：子串 "aab" 在原字符串中出现了 2 次。
它满足所有的要求：2 个不同的字母，长度为 3 （在 minSize 和 maxSize 范围内）。
示例 2：输入：s = "aaaa", maxLetters = 1, minSize = 3, maxSize = 3 输出：2
解释：子串 "aaa" 在原字符串中出现了 2 次，且它们有重叠部分。
示例 3：输入：s = "aabcabcab", maxLetters = 2, minSize = 2, maxSize = 3 输出：3
示例 4：输入：s = "abcde", maxLetters = 2, minSize = 3, maxSize = 3 输出：0
提示：1 <= s.length <= 10^5
1 <= maxLetters <= 26
1 <= minSize <= maxSize <= min(26, s.length)
s只包含小写英文字母。
```

- 解题思路

```
func maxFreq(s string, maxLetters int, minSize int, maxSize int) int {
        res := 0
        m := make(map[string]int)
        window := make(map[byte]int)
        count := 0
        left := 0
        for i := 0; i < len(s); i++ {
                window[s[i]]++
                if window[s[i]] == 1 {
                        count++
                }
                length := i - left + 1
                if length < minSize {
                        continue
                }
                if length > minSize { // 滑动窗口左边=>右移
                        window[s[left]]--
                        if window[s[left]] == 0 {
```

(续下页)

(接上页)

```
                                count--
                        }
                        left++
                        length--
                }
                // 只考虑最小值minSize，例如：minSize=2, maxSize=3
                // 如果abc出现3次，那么代表ab至少出现>=3次
                if count <= maxLetters {
                        m[s[left:i+1]]++
                }
        }
        for _, v := range m {
                res = max(res, v)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxFreq(s string, maxLetters int, minSize int, maxSize int) int {
        res := 0
        m := make(map[string]int)
        for i := 0; i <= len(s)-minSize; i++ {
                str := s[i : i+minSize]
                count := getCount(str)
                // 只考虑最小值minSize，例如：minSize=2, maxSize=3
                // 如果abc出现3次，那么代表ab至少出现>=3次
                if count <= maxLetters {
                        m[str]++
                }
        }
        for _, v := range m {
                res = max(res, v)
        }
        return res
}
```

(续下页)

(接上页)

```
func getCount(str string) int {
        m := make(map[byte]int)
        for i := 0; i < len(str); i++ {
                m[str[i]]++
        }
        return len(m)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

38.33 1300. 转变数组后最接近目标值的数组和 (3)

- 题目

```
给你一个整数数组 arr 和一个目标值 target ，请你返回一个整数 value ，
使得将数组中所有大于 value 的值变成 value 后，
数组的和最接近 target （最接近表示两者之差的绝对值最小）。
如果有多种使得和最接近 target 的方案，请你返回这些整数中的最小值。
请注意，答案不一定是 arr 中的数字。
示例 1：输入：arr = [4,9,3], target = 10 输出：3
解释：当选择 value 为 3 时，数组会变成 [3, 3, 3]，和为 9 ，这是最接近 target 的方案。
示例 2：输入：arr = [2,3,5], target = 10 输出：5
示例 3：输入：arr = [60864,25176,27249,21296,20204], target = 56803 输出：11361
提示：
    1 <= arr.length <= 10^4
    1 <= arr[i], target <= 10^5
```

- 解题思路

```
func findBestValue(arr []int, target int) int {
        sort.Ints(arr)
        n := len(arr)
        temp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                temp[i] = temp[i-1] + arr[i-1]
        }
        right := arr[n-1]
```

(续下页)

(接上页)

```
        res := 0
        diff := target
        for i := 1; i <= right; i++ {
                index := sort.SearchInts(arr, i)
                if index < 0 {
                        index = abs(index) - 1
                }
                total := temp[index] + (n-index)*i
                if abs(total-target) < diff {
                        diff = abs(total - target)
                        res = i
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func findBestValue(arr []int, target int) int {
        sort.Ints(arr)
        n := len(arr)
        temp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                temp[i] = temp[i-1] + arr[i-1]
        }
        left, right := 0, arr[n-1]
        res := 0
        for left <= right {
                mid := left + (right-left)/2
                index := sort.SearchInts(arr, mid)
                if index < 0 {
                        index = abs(index) - 1
                }
                total := temp[index] + (n-index)*mid
                if total <= target {
                        res = mid
                        left = mid + 1
```

(续下页)

(接上页)

```
                } else {
                        right = mid - 1
                }
        }
        a := getSum(arr, res)
        b := getSum(arr, res+1)
        if abs(a-target) > abs(b-target) {
                return res + 1
        }
        return res
}

func getSum(nums []int, target int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] <= target {
                        res = res + nums[i]
                } else {
                        res = res + target
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 3
func findBestValue(arr []int, target int) int {
        sort.Ints(arr)
        n := len(arr)
        res := target / n
        diff := target + 1
        for {
                a := getSum(arr, res)
                if diff <= abs(target-a) {
                        return res - 1
                }
                diff = abs(target - a)
```

(续下页)

(接上页)

```
                res = res + 1
        }
        return res
}

func getSum(nums []int, target int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] <= target {
                        res = res + nums[i]
                } else {
                        res = res + target
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

CHAPTER 39

1201-1300-Hard

39.1 1220. 统计元音字母序列的数目 (1)

- 题目

```
给你一个整数n，请你帮忙统计一下我们可以按下述规则形成多少个长度为n的字符串：
字符串中的每个字符都应当是小写元音字母 ('a', 'e', 'i', 'o', 'u')
每个元音'a'后面都只能跟着'e'
每个元音'e'后面只能跟着'a'或者是'i'
每个元音'i'后面不能 再跟着另一个'i'
每个元音'o'后面只能跟着'i'或者是'u'
每个元音'u'后面只能跟着'a'
由于答案可能会很大，所以请你返回 模10^9 + 7之后的结果。
示例 1：输入：n = 1 输出：5
解释：所有可能的字符串分别是："a", "e", "i" , "o" 和 "u"。
示例 2：输入：n = 2 输出：10
解释：所有可能的字符串分别是："ae", "ea", "ei", "ia", "ie", "io", "iu", "oi", "ou" 和
↪"ua"。
示例 3：输入：n = 5 输出：68
提示：1 <= n <= 2 * 10^4
```

- 解题思路

```
var mod = 1000000007
```

(续下页)

(接上页)

```
func countVowelPermutation(n int) int {
        a, e, i, o, u := 1, 1, 1, 1, 1
        for k := 2; k <= n; k++ {
                tempA := (e + i + u) % mod
                tempE := (a + i) % mod
                tempI := (e + o) % mod
                tempO := i % mod
                tempU := (i + o) % mod
                a, e, i, o, u = tempA, tempE, tempI, tempO, tempU
        }
        return (a + e + i + o + u) % mod
}
```

39.2 1235. 规划兼职工作 (4)

- 题目

```
你打算利用空闲时间来做兼职工作赚些零花钱。
这里有n份兼职工作，每份工作预计从startTime[i]开始到endTime[i]结束，报酬为profit[i]。
给你一份兼职工作表，包含开始时间startTime，
结束时间endTime和预计报酬profit三个数组，请你计算并返回可以获得的最大报酬。
注意，时间上出现重叠的 2 份工作不能同时进行。
如果你选择的工作在时间X结束，那么你可以立刻进行在时间X开始的下一份工作。
示例 1：输入：startTime = [1,2,3,3], endTime = [3,4,5,6], profit = [50,10,40,70]␣
↪输出：120
解释：我们选出第 1 份和第 4 份工作，
时间范围是 [1-3]+[3-6]，共获得报酬 120 = 50 + 70。
示例 2：输入：startTime = [1,2,3,4,6], endTime = [3,5,10,6,9], profit = [20,20,100,70,
↪60]
输出：150
解释：我们选择第 1，4，5 份工作。
共获得报酬 150 = 20 + 70 + 60。
示例 3：输入：startTime = [1,1,1], endTime = [2,3,4], profit = [5,6,4] 输出：6
提示：1 <= startTime.length == endTime.length ==profit.length<= 5 * 10^4
1 <=startTime[i] <endTime[i] <= 10^9
1 <=profit[i] <= 10^4
```

- 解题思路

```
type Node struct {
        startTime int
        endTime   int
```

(续下页)

(接上页)

```
        profit    int
}

func jobScheduling(startTime []int, endTime []int, profit []int) int {
        n := len(startTime)
        arr := make([]Node, 0)
        for i := 0; i < n; i++ {
                arr = append(arr, Node{
                        startTime: startTime[i],
                        endTime:   endTime[i],
                        profit:    profit[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i].endTime == arr[j].endTime {
                        return arr[i].startTime < arr[j].startTime
                }
                return arr[i].endTime < arr[j].endTime
        })
        dp := make([]int, n)
        maxValue := 0
        for i := 0; i < n; i++ {
                dp[i] = arr[i].profit
                for j := i - 1; j >= 0; j-- {
                        if arr[j].endTime <= arr[i].startTime {
                                dp[i] = max(dp[i], dp[j]+arr[i].profit)
                                break
                        }
                }
                dp[i] = max(dp[i], maxValue)
                maxValue = max(maxValue, dp[i])
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
type Node struct {
        startTime int
        endTime   int
        profit    int
}

func jobScheduling(startTime []int, endTime []int, profit []int) int {
        n := len(startTime)
        arr := make([]Node, 0)
        for i := 0; i < n; i++ {
                arr = append(arr, Node{
                        startTime: startTime[i],
                        endTime:   endTime[i],
                        profit:    profit[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i].endTime == arr[j].endTime {
                        return arr[i].startTime < arr[j].startTime
                }
                return arr[i].endTime < arr[j].endTime
        })
        for i := 1; i < n; i++ {
                target := sort.Search(i, func(j int) bool {
                        return arr[j].endTime > arr[i].startTime
                })
                if target == 0 {
                        arr[i].profit = max(arr[i].profit, arr[i-1].profit)
                } else {
                        arr[i].profit = max(arr[i].profit+arr[target-1].profit, arr[i-
↪1].profit)
                }
        }
        return arr[n-1].profit
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 3
type Node struct {
        startTime int
        endTime   int
        profit    int
}

func jobScheduling(startTime []int, endTime []int, profit []int) int {
        n := len(startTime)
        arr := make([]Node, 0)
        for i := 0; i < n; i++ {
                arr = append(arr, Node{
                        startTime: startTime[i],
                        endTime:   endTime[i],
                        profit:    profit[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i].endTime == arr[j].endTime {
                        return arr[i].startTime < arr[j].startTime
                }
                return arr[i].endTime < arr[j].endTime
        })
        dp := make([]int, n)
        dp[0] = arr[0].profit
        for i := 1; i < n; i++ {
                dp[i] = dp[i-1]
                dp[i] = max(dp[i], arr[i].profit)
                for j := i - 1; j >= 0; j-- {
                        if arr[j].endTime <= arr[i].startTime {
                                dp[i] = max(dp[i], dp[j]+arr[i].profit)
                                break
                        }
                }
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

# 4
type Node struct {
        startTime int
        endTime   int
        profit    int
}

func jobScheduling(startTime []int, endTime []int, profit []int) int {
        n := len(startTime)
        arr := make([]Node, 0)
        for i := 0; i < n; i++ {
                arr = append(arr, Node{
                        startTime: startTime[i],
                        endTime:   endTime[i],
                        profit:    profit[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i].endTime == arr[j].endTime {
                        return arr[i].startTime < arr[j].startTime
                }
                return arr[i].endTime < arr[j].endTime
        })
        dp := make([]int, n)
        dp[0] = arr[0].profit
        for i := 1; i < n; i++ {
                left, right := 0, i-1
                for left < right {
                        mid := left + (right-left)/2
                        if arr[mid+1].endTime <= arr[i].startTime {
                                left = mid + 1
                        } else {
                                right = mid
                        }
                }
                if arr[left].endTime <= arr[i].startTime {
                        dp[i] = max(dp[i-1], dp[left]+arr[i].profit)
                } else {
                        dp[i] = max(dp[i-1], arr[i].profit)
                }
        }
```

(续下页)

(接上页)

```
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

39.3 1255. 得分最高的单词集合 (3)

- 题目

```
你将会得到一份单词表words，一个字母表letters（可能会有重复字母），以及每个字母对应的得分情况表score。
请你帮忙计算玩家在单词拼写游戏中所能获得的「最高得分」：
能够由letters里的字母拼写出的任意属于 words单词子集中，分数最高的单词集合的得分。
单词拼写游戏的规则概述如下：
玩家需要用字母表letters 里的字母来拼写单词表words中的单词。
可以只使用字母表letters 中的部分字母，但是每个字母最多被使用一次。
单词表 words中每个单词只能计分（使用）一次。
根据字母得分情况表score，字母 'a','b','c', ... ,'z' 对应的得分分别为 score[0],
↪score[1],...,score[25]。
本场游戏的「得分」是指：玩家所拼写出的单词集合里包含的所有字母的得分之和。
示例 1：输入：words = ["dog","cat","dad","good"], letters = ["a","a","c","d","d","d",
↪"g","o","o"],
score = [1,0,9,5,0,0,3,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0]
输出：23
解释：字母得分为  a=1, c=9, d=5, g=3, o=2
使用给定的字母表 letters，我们可以拼写单词 "dad" (5+1+5)和 "good" (3+2+2+5)，得分为
↪23 。
而单词 "dad" 和 "dog" 只能得到 21 分。
示例 2：输入：words = ["xxxz","ax","bx","cx"], letters = ["z","a","b","c","x","x","x
↪"],
score = [4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,10] 输出：27
解释：字母得分为  a=4, b=4, c=4, x=5, z=10
使用给定的字母表 letters，我们可以组成单词 "ax" (4+5)， "bx" (4+5) 和 "cx" (4+5)
↪，总得分为 27 。
单词 "xxxz" 的得分仅为 25 。
示例 3：输入：words = ["leetcode"], letters = ["l","e","t","c","o","d"],
score = [0,0,1,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0] 输出：0
解释：字母 "e" 在字母表 letters 中只出现了一次，所以无法组成单词表 words 中的单词。
```

(续下页)

(接上页)

```
提示：1 <= words.length <= 14
1 <= words[i].length <= 15
1 <= letters.length <= 100
letters[i].length == 1
score.length ==26
0 <= score[i] <= 10
words[i]和letters[i]只包含小写的英文字母。
```

- 解题思路

```
func maxScoreWords(words []string, letters []byte, score []int) int {
        count := [26]int{}
        for i := 0; i < len(letters); i++ {
                count[int(letters[i]-'a')]++ // 统计字符出现的次数
        }
        n := len(words)
        total := 1 << n // 总状态数
        res := 0
        for i := 0; i < total; i++ { // 枚举所有状态
                arr := getStatus(words, i)                    //␣
↪统计每种状态所需字符个数
                res = max(res, getScore(score, count, arr)) // 计算得分
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

// 统计该状态每个字符多少个
func getStatus(words []string, status int) [26]int {
        arr := [26]int{}
        for i := 0; i < len(words); i++ {
                if status&(1<<i) > 0 {
                        for j := 0; j < len(words[i]); j++ {
                                arr[int(words[i][j]-'a')]++
                        }
                }
        }
        return arr
```

(续下页)

(接上页)

```
}

func getScore(score []int, count, arr [26]int) int {
        res := 0
        for i := 0; i < 26; i++ {
                if count[i] < arr[i] {
                        return 0
                }
                res = res + score[i]*arr[i]
        }
        return res
}

# 2
var res int
var count [26]int

func maxScoreWords(words []string, letters []byte, score []int) int {
        res = 0
        count = [26]int{}
        for i := 0; i < len(letters); i++ {
                count[int(letters[i]-'a')]++ // 统计字符出现的次数
        }
        dfs(words, score, 0, [26]int{})
        return res
}

func dfs(words []string, score []int, index int, arr [26]int) {
        sum := 0
        for i := 0; i < 26; i++ {
                if arr[i] > count[i] {
                        return
                }
                sum = sum + arr[i]*score[i]
        }
        res = max(res, sum)
        for i := index; i < len(words); i++ {
                for j := 0; j < len(words[i]); j++ {
                        arr[int(words[i][j]-'a')]++
                }
                dfs(words, score, i+1, arr)
                for j := 0; j < len(words[i]); j++ {
                        arr[int(words[i][j]-'a')]--
```

(续下页)

(接上页)

```
                }
        }
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
var res int
var count [26]int

func maxScoreWords(words []string, letters []byte, score []int) int {
        res = 0
        count = [26]int{}
        for i := 0; i < len(letters); i++ {
                count[int(letters[i]-'a')]++ // 统计字符出现的次数
        }
        dfs(words, score, 0, [26]int{})
        return res
}

func dfs(words []string, score []int, index int, arr [26]int) {
        sum := 0
        for i := 0; i < 26; i++ {
                if arr[i] > count[i] {
                        return
                }
                sum = sum + arr[i]*score[i]
        }
        res = max(res, sum)
        if index >= len(words) {
                return
        }
        dfs(words, score, index+1, arr) // 不选
        for j := 0; j < len(words[index]); j++ {
                arr[int(words[index][j]-'a')]++
        }
        dfs(words, score, index+1, arr) // 选
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

39.4 1269. 停在原地的方案数 (2)

- 题目

```
有一个长度为arrLen的数组，开始有一个指针在索引0 处。
每一步操作中，你可以将指针向左或向右移动 1␣
↪步，或者停在原地（指针不能被移动到数组范围外）。
给你两个整数steps 和arrLen␣
↪，请你计算并返回：在恰好执行steps次操作以后，指针仍然指向索引0 处的方案数。
由于答案可能会很大，请返回方案数 模10^9 + 7 后的结果。
示例 1：输入：steps = 3, arrLen = 2 输出：4
解释：3 步后，总共有 4 种不同的方法可以停在索引 0 处。
向右，向左，不动
不动，向右，向左
向右，不动，向左
不动，不动，不动
示例 2：输入：steps = 2, arrLen = 4 输出：2
解释：2 步后，总共有 2 种不同的方法可以停在索引 0 处。
向右，向左
不动，不动
示例 3：输入：steps = 4, arrLen = 2 输出：8
提示：1 <= steps <= 500
1 <= arrLen<= 10^6
```

- 解题思路

```
var mod = 1000000007

func numWays(steps int, arrLen int) int {
        length := min(arrLen-1, steps)
        dp := make([][]int, steps+1) // dp[i][j] =>␣
↪在i步操作之后，指针位于下标j的方案数
        for i := 0; i <= steps; i++ {
                dp[i] = make([]int, length+1)
```

(续下页)

(接上页)

```
        }
        dp[0][0] = 1
        for i := 1; i <= steps; i++ {
                for j := 0; j <= length; j++ {
                        dp[i][j] = dp[i-1][j] // 不动
                        if j >= 1 {
                                dp[i][j] = (dp[i][j] + dp[i-1][j-1]) % mod // 右移
                        }
                        if j <= length-1 {
                                dp[i][j] = (dp[i][j] + dp[i-1][j+1]) % mod // 左移
                        }
                }
        }
        return dp[steps][0]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var mod = 1000000007

func numWays(steps int, arrLen int) int {
        length := min(arrLen-1, steps)
        dp := make([]int, length+1) // dp[j] => 指针位于下标j的方案数
        dp[0] = 1
        for i := 1; i <= steps; i++ {
                temp := make([]int, length+1)
                for j := 0; j <= length; j++ {
                        temp[j] = dp[j] // 不动
                        if j >= 1 {
                                temp[j] = (temp[j] + dp[j-1]) % mod // 右移
                        }
                        if j <= length-1 {
                                temp[j] = (temp[j] + dp[j+1]) % mod // 左移
                        }
                }
                dp = temp
        }
```

(续下页)

(接上页)

```
        return dp[0]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

39.5 1289. 下降路径最小和 II(3)

- 题目

```
给你一个整数方阵arr，定义「非零偏移下降路径」为：
从arr 数组中的每一行选择一个数字，且按顺序选出来的数字中，相邻数字不在原数组的同一列。
请你返回非零偏移下降路径数字和的最小值。
示例 1：输入：arr = [[1,2,3],[4,5,6],[7,8,9]] 输出：13
解释：所有非零偏移下降路径包括：
[1,5,9], [1,5,7], [1,6,7], [1,6,8],
[2,4,8], [2,4,9], [2,6,7], [2,6,8],
[3,4,8], [3,4,9], [3,5,7], [3,5,9]
下降路径中数字和最小的是[1,5,7] ，所以答案是13 。
提示：1 <= arr.length == arr[i].length <= 200
-99 <= arr[i][j] <= 99
```

- 解题思路

```
func minFallingPathSum(arr [][]int) int {
        n := len(arr)
        firstMin, secondMin := 0, 0
        firstIndex := -1
        for i := 0; i < n; i++ {
                fMin, sMin := math.MaxInt32, math.MaxInt32
                fIndex := -1
                for j := 0; j < n; j++ {
                        sum := 0
                        if firstIndex != j { // 不等于最小值所在行，就+最小值
                                sum = firstMin + arr[i][j]
                        } else { // 等于最小值所在行，就+次小值
                                sum = secondMin + arr[i][j]
                        }
```

(续下页)

(接上页)

```
                        if sum < fMin {
                                sMin = fMin
                                fMin = sum
                                fIndex = j
                        } else if sum < sMin {
                                sMin = sum
                        }
                }
                firstMin = fMin
                secondMin = sMin
                firstIndex = fIndex
        }
        return firstMin
}

# 2
func minFallingPathSum(arr [][]int) int {
        n := len(arr)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for j := 0; j < n; j++ {
                dp[0][j] = arr[0][j]
        }
        for i := 1; i < n; i++ {
                for j := 0; j < n; j++ {
                        dp[i][j] = math.MaxInt32
                        for k := 0; k < n; k++ {
                                if j != k {
                                        dp[i][j] = min(dp[i][j], dp[i-1][k]+arr[i][j])
                                }
                        }
                }
        }
        res := math.MaxInt32
        for j := 0; j < n; j++ {
                res = min(res, dp[n-1][j])
        }
        return res
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}

# 3
func minFallingPathSum(arr [][]int) int {
        n := len(arr)
        for i := 1; i < n; i++ {
                temp := make([][2]int, n)
                for j := 0; j < n; j++ {
                        temp[j] = [2]int{arr[i-1][j], j}
                }
                sort.Slice(temp, func(i, j int) bool {
                        return temp[i][0] < temp[j][0]
                })
                for j := 0; j < n; j++ {
                        if temp[0][1] != j {
                                arr[i][j] = arr[i][j] + temp[0][0]
                        } else {
                                arr[i][j] = arr[i][j] + temp[1][0]
                        }
                }
        }
        res := math.MaxInt32
        for j := 0; j < n; j++ {
                res = min(res, arr[n-1][j])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

39.6 1293. 网格中的最短路径 (1)

- 题目

```
给你一个m *␣
↪n的网格，其中每个单元格不是0（空）就是1（障碍物）。每一步，您都可以在空白单元格中上、下、左、右移动
如果您最多可以消除k个障碍物，请找出从左上角 (0, 0) 到右下角 (m-1, n-1)␣
↪的最短路径，并返回通过该路径所需的步数。
如果找不到这样的路径，则返回 -1。
示例 1：输入： grid =
[[0,0,0],
[1,1,0],
 [0,0,0],
[0,1,1],
 [0,0,0]],
k = 1
输出：6
解释：不消除任何障碍的最短路径是 10。
消除位置 (3,2) 处的障碍后，最短路径是 6 。该路径是 (0,0) -> (0,1) -> (0,2) -> (1,2) ->
↪ (2,2) -> (3,2) -> (4,2).
示例 2：输入：grid =
[[0,1,1],
[1,1,1],
[1,0,0]],
k = 1
输出：-1
解释：我们至少需要消除两个障碍才能找到这样的路径。
提示：grid.length== m
grid[0].length== n
1 <= m, n <= 40
1 <= k <= m*n
grid[i][j] == 0 or 1
grid[0][0] == grid[m-1][n-1] == 0
```

解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func shortestPath(grid [][]int, k int) int {
        n, m := len(grid), len(grid[0])
        if n == 1 && m == 1 {
                return 0
        }
```

(续下页)

(接上页)

```
        k = min(k, n+m-3) // 缩小k的范围
        visited := make([][][]bool, n)
        for i := 0; i < n; i++ {
                visited[i] = make([][]bool, m)
                for j := 0; j < m; j++ {
                        visited[i][j] = make([]bool, k+1)
                }
        }
        count := 1
        queue := make([][3]int, 0)
        queue = append(queue, [3]int{0, 0, k})
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        a, b, c := queue[i][0], queue[i][1], queue[i][2]
                        for j := 0; j < 4; j++ {
                                x, y := a+dx[j], b+dy[j]
                                if 0 <= x && x < n && 0 <= y && y < m {
                                        if grid[x][y] == 0 && visited[x][y][c] ==␣
↪false {
                                                if x == n-1 && y == m-1 { // 走到终点
                                                        return count
                                                }
                                                queue = append(queue, [3]int{x, y, c})
                                                visited[x][y][c] = true
                                        } else if grid[x][y] == 1 && c > 0 &&␣
↪visited[x][y][c-1] == false {
                                                queue = append(queue, [3]int{x, y, c -
↪ 1})
                                                visited[x][y][c-1] = true
                                        }
                                }
                        }
                }
                queue = queue[length:]
                count++
        }
        return -1
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}
```

39.7 1298. 你能从盒子里获得的最大糖果数

39.7.1 题目

```
给你n个盒子，每个盒子的格式为[status, candies, keys, containedBoxes]，其中：
状态字status[i]：整数，如果box[i]是开的，那么是 1，否则是 0。
糖果数candies[i]: 整数，表示box[i] 中糖果的数目。
钥匙keys[i]：数组，表示你打开box[i]后，可以得到一些盒子的钥匙，每个元素分别为该钥匙对应盒子的下标。
内含的盒子containedBoxes[i]：整数，表示放在box[i]里的盒子所对应的下标。
给你一个initialBoxes 数组，表示你现在得到的盒子，你可以获得里面的糖果，
也可以用盒子里的钥匙打开新的盒子，还可以继续探索从这个盒子里找到的其他盒子。
请你按照上述规则，返回可以获得糖果的 最大数目。
示例 1：输入：status = [1,0,1,0], candies = [7,5,4,100], keys = [[],[],[1],[]],
containedBoxes = [[1,2],[3],[],[]], initialBoxes = [0]
输出：16
解释：一开始你有盒子 0 。你将获得它里面的 7 个糖果和盒子 1 和 2。
盒子 1 目前状态是关闭的，而且你还没有对应它的钥匙。所以你将会打开盒子 2␣
↪，并得到里面的 4 个糖果和盒子 1 的钥匙。
在盒子 1 中，你会获得 5 个糖果和盒子 3 ，但是你没法获得盒子 3 的钥匙所以盒子 3␣
↪会保持关闭状态。
你总共可以获得的糖果数目 = 7 + 4 + 5 = 16 个。
示例 2：输入：status = [1,0,0,0,0,0], candies = [1,1,1,1,1,1],
keys = [[1,2,3,4,5],[],[],[],[],[]], containedBoxes = [[1,2,3,4,5],[],[],[],[],[]],␣
↪initialBoxes = [0]
输出：6
解释：你一开始拥有盒子 0 。打开它你可以找到盒子 1,2,3,4,5 和它们对应的钥匙。
打开这些盒子，你将获得所有盒子的糖果，所以总糖果数为 6 个。
示例 3：输入：status = [1,1,1], candies = [100,1,100], keys = [[],[0,2],[]],
containedBoxes = [[],[],[]], initialBoxes = [1] 输出：1
示例 4：输入：status = [1], candies = [100], keys = [[]], containedBoxes = [[]],␣
↪initialBoxes = [] 输出：0
示例 5：输入：status = [1,1,1], candies = [2,3,2], keys = [[],[],[]],
containedBoxes = [[],[],[]], initialBoxes = [2,1,0] 输出：7
提示：1 <= status.length <= 1000
status.length == candies.length == keys.length == containedBoxes.length == n
status[i] 要么是0要么是1 。
1 <= candies[i] <= 1000
```

(续下页)

(接上页)

```
0 <= keys[i].length <= status.length
0 <= keys[i][j] < status.length
keys[i]中的值都是互不相同的。
0 <= containedBoxes[i].length <= status.length
0 <= containedBoxes[i][j] < status.length
containedBoxes[i]中的值都是互不相同的。
每个盒子最多被一个盒子包含。
0 <= initialBoxes.length<= status.length
0 <= initialBoxes[i] < status.length
```

39.7.2 解题思路

CHAPTER 40

1301-1400-Easy

40.1 1304. 和为零的 N 个唯一整数 (2)

- 题目

```
给你一个整数 n，请你返回 任意 一个由 n 个 各不相同 的整数组成的数组，并且这 n␣
↪个数相加和为 0 。
示例 1：输入：n = 5 输出：[-7,-1,1,3,4]
解释：这些数组也是正确的 [-5,-1,1,2,3]，[-3,-1,2,-2,4]。
示例 2：输入：n = 3 输出：[-1,0,1]
示例 3：输入：n = 1 输出：[0]
提示：
    1 <= n <= 1000
```

- 解题思路

```
func sumZero(n int) []int {
        res := make([]int, n)
        sum := 0
        for i := 0; i < n-1; i++ {
                res[i] = i + 1
                sum = sum + i + 1
        }
        res[n-1] = -sum
        return res
```

(续下页)

(接上页)

```
}

#
func sumZero(n int) []int {
        res := make([]int, 0)
        if n%2 == 1 {
                res = append(res, 0)
        }
        for i := 1; i <= n/2; i++ {
                res = append(res, i)
                res = append(res, -i)
        }
        return res
}
```

40.2 1309. 解码字母到整数映射 (3)

- 题目

```
给你一个字符串 s，它由数字（'0' - '9'）和 '#' 组成。我们希望按下述规则将 s␣
↪映射为一些小写英文字符：
    字符（'a' - 'i'）分别用（'1' - '9'）表示。
    字符（'j' - 'z'）分别用（'10#' - '26#'）表示。
返回映射之后形成的新字符串。
题目数据保证映射始终唯一。
示例 1：输入：s = "10#11#12" 输出："jkab"
解释："j" -> "10#" , "k" -> "11#" , "a" -> "1" , "b" -> "2".
示例 2：输入：s = "1326#" 输出："acz"
示例 3：输入：s = "25#" 输出："y"
示例 4：输入：s = "12345678910#11#12#13#14#15#16#17#18#19#20#21#22#23#24#25#26#"
输出："abcdefghijklmnopqrstuvwxyz"
提示：
    1 <= s.length <= 1000
    s[i] 只包含数字（'0'-'9'）和 '#' 字符。
    s 是映射始终存在的有效字符串。
```

- 解题思路

```
func freqAlphabets(s string) string {
        res := ""
        for i := len(s) - 1; i >= 0; {
                if s[i] == '#' {
```

(续下页)

(接上页)

```
                        value, _ := strconv.Atoi(string(s[i-2 : i]))
                        res = string('a'+value-1) + res
                        i = i - 3
                } else {
                        value, _ := strconv.Atoi(string(s[i]))
                        res = string('a'+value-1) + res
                        i = i - 1
                }
        }
        return res
}

#
func freqAlphabets(s string) string {
        res := ""
        for i := 0; i < len(s); {
                if i+2 < len(s) && s[i+2] == '#' {
                        value, _ := strconv.Atoi(string(s[i : i+2]))
                        res = res + string('a'+value-1)
                        i = i + 3
                } else {
                        value, _ := strconv.Atoi(string(s[i]))
                        res = res + string('a'+value-1)
                        i = i + 1
                }
        }
        return res
}

#
func freqAlphabets(s string) string {
        m := make(map[string]string)
        for i := 10; i <= 26; i++ {
                m[strconv.Itoa(i)+"#"] = string('j' + i - 10)
        }
        m2 := make(map[string]string)
        for i := 1; i <= 9; i++ {
                m2[strconv.Itoa(i)] = string('a' + i - 1)
        }
        for k, v := range m {
                s = strings.ReplaceAll(s, k, v)
        }
        for k, v := range m2 {
```

(续下页)

(接上页)

```
                s = strings.ReplaceAll(s, k, v)
        }
        return s
}
```

40.3 1313. 解压缩编码列表 (1)

- 题目

```
给你一个以行程长度编码压缩的整数列表 nums 。
考虑每对相邻的两个元素 [freq, val] = [nums[2*i], nums[2*i+1]] （其中 i >= 0 ），
每一对都表示解压后子列表中有 freq 个值为 val␣
↪的元素，你需要从左到右连接所有子列表以生成解压后的列表。
请你返回解压后的列表。
示例：输入：nums = [1,2,3,4] 输出：[2,4,4,4]
解释：第一对 [1,2] 代表着 2 的出现频次为 1，所以生成数组 [2]。
第二对 [3,4] 代表着 4 的出现频次为 3，所以生成数组 [4,4,4]。
最后将它们串联到一起 [2] + [4,4,4] = [2,4,4,4]。
示例 2：输入：nums = [1,1,2,3] 输出：[1,3,3]
提示：
    2 <= nums.length <= 100
    nums.length % 2 == 0
    1 <= nums[i] <= 100
```

- 解题思路

```
func decompressRLElist(nums []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(nums); i = i + 2 {
                for j := 0; j < nums[i]; j++ {
                        res = append(res, nums[i+1])
                }
        }
        return res
}
```

40.4 1317. 将整数转换为两个无零整数的和 (2)

- 题目

```
「无零整数」是十进制表示中 不含任何 0 的正整数。
给你一个整数 n，请你返回一个 由两个整数组成的列表 [A, B]，满足：
    A 和 B 都是无零整数
    A + B = n
题目数据保证至少有一个有效的解决方案。
如果存在多个有效解决方案，你可以返回其中任意一个。
示例 1：输入：n = 2 输出：[1,1]
解释：A = 1, B = 1. A + B = n 并且 A 和 B 的十进制表示形式都不包含任何 0 。
示例 2：输入：n = 11 输出：[2,9]
示例 3：输入：n = 10000 输出：[1,9999]
示例 4：输入：n = 69 输出：[1,68]
示例 5：输入：n = 1010 输出：[11,999]
提示：2 <= n <= 10^4
```

- 解题思路

```
func getNoZeroIntegers(n int) []int {
        for i := 1; i < n; i++ {
                if strings.ContainsAny(strconv.Itoa(i), "0") ||
                        strings.ContainsAny(strconv.Itoa(n-i), "0") {
                        continue
                }
                return []int{i, n - i}
        }
        return nil
}

#
func getNoZeroIntegers(n int) []int {
        for i := 1; i < n; i++ {
                if contains(i) || contains(n-i) {
                        continue
                }
                return []int{i, n - i}
        }
        return nil
}

func contains(num int) bool {
        for num > 0 {
```

(续下页)

(接上页)

```
                if num%10 == 0 {
                        return false
                }
                num = num / 10
        }
        return true
}
```

40.5 1323.6 和 9 组成的最大数字 (3)

- 题目

```
给你一个仅由数字 6 和 9 组成的正整数 num。
你最多只能翻转一位数字，将 6 变成 9，或者把 9 变成 6 。
请返回你可以得到的最大数字。
示例 1：输入：num = 9669 输出：9969
解释：
改变第一位数字可以得到 6669 。
改变第二位数字可以得到 9969 。
改变第三位数字可以得到 9699 。
改变第四位数字可以得到 9666 。
其中最大的数字是 9969 。
示例 2：输入：num = 9996 输出：9999
解释：将最后一位从 6 变到 9，其结果 9999 是最大的数。
示例 3：输入：num = 9999 输出：9999
解释：无需改变就已经是最大的数字了。
提示：
    1 <= num <= 10^4
    num 每一位上的数字都是 6 或者 9 。
```

- 解题思路

```
func maximum69Number(num int) int {
        arr := []byte(strconv.Itoa(num))
        for i := 0; i < len(arr); i++ {
                if arr[i] == '6' {
                        arr[i] = '9'
                        break
                }
        }
        res, _ := strconv.Atoi(string(arr))
        return res
```

(续下页)

(接上页)

```
}

#
func maximum69Number(num int) int {
        arr := make([]int, 0)
        for num > 0 {
                arr = append(arr, num%10)
                num = num / 10
        }
        res := 0
        flag := true
        for i := len(arr) - 1; i >= 0; i-- {
                if arr[i] == 6 && flag == true {
                        res = res*10 + 9
                        flag = false
                } else {
                        res = res*10 + arr[i]
                }
        }
        return res
}

#
func maximum69Number(num int) int {
        str := strconv.Itoa(num)
        str = strings.Replace(str, "6", "9", 1)
        res, _ := strconv.Atoi(string(str))
        return res
}
```

40.6 1331. 数组序号转换 (2)

- 题目

```
给你一个整数数组 arr ，请你将数组中的每个元素替换为它们排序后的序号。
序号代表了一个元素有多大。序号编号的规则如下：
    序号从 1 开始编号。
    一个元素越大，那么序号越大。如果两个元素相等，那么它们的序号相同。
    每个数字的序号都应该尽可能地小。
示例 1：输入：arr = [40,10,20,30] 输出：[4,1,2,3]
解释：40 是最大的元素。 10 是最小的元素。 20 是第二小的数字。 30 是第三小的数字。
示例 2：输入：arr = [100,100,100] 输出：[1,1,1]
```

(续下页)

(接上页)

```
解释：所有元素有相同的序号。
示例 3：输入：arr = [37,12,28,9,100,56,80,5,12] 输出：[5,3,4,2,8,6,7,1,3]
提示：
    0 <= arr.length <= 10^5
    -10^9 <= arr[i] <= 10^9
```

- 解题思路

```
func arrayRankTransform(arr []int) []int {
        temp := make([]int, len(arr))
        copy(temp, arr)
        sort.Ints(temp)
        m := make(map[int]int)
        count := 1
        for i := 0; i < len(temp); i++ {
                if m[temp[i]] > 0 {
                        continue
                }
                m[temp[i]] = count
                count++
        }
        res := make([]int, len(arr))
        for i := 0; i < len(arr); i++ {
                res[i] = m[arr[i]]
        }
        return res
}

#
func arrayRankTransform(arr []int) []int {
        if len(arr) == 0 {
                return arr
        }
        min := math.MaxInt32
        max := math.MinInt32
        for i := 0; i < len(arr); i++ {
                if arr[i] <= min {
                        min = arr[i]
                }
                if arr[i] >= max {
                        max = arr[i]
                }
        }
        length := max - min + 1
```

(续下页)

(接上页)

```
        temp := make([]int, length)
        for i := 0; i < length; i++ {
                temp[i] = math.MinInt32
        }
        for i := 0; i < len(arr); i++ {
                temp[arr[i]-min] = -1
        }
        count := 0
        for i := 0; i < length; i++ {
                if temp[i] == -1 {
                        temp[i] = count
                        count++
                }
        }
        for i := 0; i < len(arr); i++ {
                arr[i] = temp[arr[i]-min] + 1
        }
        return arr
}
```

40.7 1332. 删除回文子序列 (2)

- 题目

```
给你一个字符串 s，它仅由字母 'a' 和 'b' 组成。每一次删除操作都可以从 s␣
↪中删除一个回文子序列。
返回删除给定字符串中所有字符（字符串为空）的最小删除次数。
「子序列」定义：如果一个字符串可以通过删除原字符串某些字符而不改变原字符顺序得到，
那么这个字符串就是原字符串的一个子序列。
「回文」定义：如果一个字符串向后和向前读是一致的，那么这个字符串就是一个回文。
示例 1：输入：s = "ababa" 输出：1
解释：字符串本身就是回文序列，只需要删除一次。
示例 2：输入：s = "abb" 输出：2
解释："abb" -> "bb" -> "". 先删除回文子序列 "a"，然后再删除 "bb"。
示例 3：输入：s = "baabb" 输出：2
解释："baabb" -> "b" -> "". 先删除回文子序列 "baab"，然后再删除 "b"。
示例 4：输入：s = "" 输出：0
提示：
    0 <= s.length <= 1000
    s 仅包含字母 'a' 和 'b'
```

- 解题思路

```
/*
1.长度为0，返回0
2.字符串为回文子序列，返回1
3.字符串不为回文子序列，返回2，因为可以把a或者b一次都去除，题目没有要求去除的是连续的
*/
func removePalindromeSub(s string) int {
        if len(s) == 0 {
                return 0
        }
        for i, j := 0, len(s)-1; i < j; {
                if s[i] != s[j] {
                        return 2
                }
                i++
                j--
        }
        return 1
}

#
func removePalindromeSub(s string) int {
        if len(s) == 0 {
                return 0
        }
        temp := ""
        for i := len(s) - 1; i >= 0; i-- {
                temp = temp + string(s[i])
        }
        if temp == s {
                return 1
        }
        return 2
}
```

40.8 1337. 方阵中战斗力最弱的 K 行 (2)

- 题目

```
给你一个大小为 m * n 的方阵 mat，方阵由若干军人和平民组成，分别用 1 和 0 表示。
请你返回方阵中战斗力最弱的 k 行的索引，按从最弱到最强排序。
如果第 i 行的军人数量少于第 j 行，或者两行军人数量相同但 i 小于 j，
那么我们认为第 i 行的战斗力比第 j 行弱。
```

(续下页)

(接上页)

```
军人 总是 排在一行中的靠前位置，也就是说 1 总是出现在 0 之前。
示例 1：输入：mat =
[[1,1,0,0,0],
 [1,1,1,1,0],
 [1,0,0,0,0],
 [1,1,0,0,0],
 [1,1,1,1,1]],
k = 3
输出：[2,0,3]
解释：
每行中的军人数目：
行 0 -> 2
行 1 -> 4
行 2 -> 1
行 3 -> 2
行 4 -> 5
从最弱到最强对这些行排序后得到 [2,0,3,1,4]
示例 2：输入：mat =
[[1,0,0,0],
 [1,1,1,1],
 [1,0,0,0],
 [1,0,0,0]],
k = 2
输出：[0,2]
解释：
每行中的军人数目：
行 0 -> 1
行 1 -> 4
行 2 -> 1
行 3 -> 1
从最弱到最强对这些行排序后得到 [0,2,3,1]

提示：
    m == mat.length
    n == mat[i].length
    2 <= n, m <= 100
    1 <= k <= m
    matrix[i][j] 不是 0 就是 1
```

- 解题思路

```
func kWeakestRows(mat [][]int, k int) []int {
        arr := make([]int, 0)
        for i := 0; i < len(mat); i++ {
```

(续下页)

(接上页)

```
                sum := 0
                for j := 0; j < len(mat[i]); j++ {
                        if mat[i][j] == 1 {
                                sum++
                        }
                }
                arr = append(arr, sum*100+i)
        }
        sort.Ints(arr)
        for i := 0; i < k; i++ {
                arr[i] = arr[i] % 100
        }
        return arr[:k]
}

#
func kWeakestRows(mat [][]int, k int) []int {
        arr := make([][]int, 0)
        for i := 0; i < len(mat); i++ {
                sum := 0
                for j := 0; j < len(mat[i]); j++ {
                        if mat[i][j] == 1 {
                                sum++
                        }
                }
                arr = append(arr, []int{sum, i})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][0] == arr[j][0] {
                        return arr[i][1] < arr[j][1]
                }
                return arr[i][0] < arr[j][0]
        })
        res := make([]int, 0)
        for i := 0; i < k; i++ {
                res = append(res, arr[i][1])
        }
        return res
}
```

40.9 1342. 将数字变成 0 的操作次数 (3)

- 题目

```
给你一个非负整数 num ，请你返回将它变成 0 所需要的步数。
如果当前数字是偶数，你需要把它除以 2 ；否则，减去 1 。
示例 1：输入：num = 14 输出：6
解释：
步骤 1) 14 是偶数，除以 2 得到 7 。
步骤 2) 7 是奇数，减 1 得到 6 。
步骤 3) 6 是偶数，除以 2 得到 3 。
步骤 4) 3 是奇数，减 1 得到 2 。
步骤 5) 2 是偶数，除以 2 得到 1 。
步骤 6) 1 是奇数，减 1 得到 0 。
示例 2：输入：num = 8 输出：4
解释：
步骤 1) 8 是偶数，除以 2 得到 4 。
步骤 2) 4 是偶数，除以 2 得到 2 。
步骤 3) 2 是偶数，除以 2 得到 1 。
步骤 4) 1 是奇数，减 1 得到 0 。
示例 3：输入：num = 123 输出：12
提示： 0 <= num <= 10^6
```

- 解题思路

```
func numberOfSteps(num int) int {
        res := 0
        for num > 0 {
                if num%2 == 1 {
                        num = num - 1
                } else {
                        num = num / 2
                }
                res++
        }
        return res
}

#
var res int

func numberOfSteps(num int) int {
        res = 0
        dfs(num)
```

(续下页)

(接上页)

```
        return res
}

func dfs(num int) {
        if num != 0 {
                res++
                if num%2 == 1 {
                        dfs(num - 1)
                } else {
                        dfs(num / 2)
                }
        }
}

#
func numberOfSteps(num int) int {
        if num == 0 {
                return 0
        } else if num%2 == 1 {
                return 1 + numberOfSteps(num-1)
        }
        return 1 + numberOfSteps(num/2)
}
```

40.10 1346. 检查整数及其两倍数是否存在 (3)

- 题目

```
给你一个整数数组 arr，请你检查是否存在两个整数 N 和 M，满足 N 是 M 的两倍（即，N = 2␣
↪* M）。
更正式地，检查是否存在两个下标 i 和 j 满足：
    i != j
    0 <= i, j < arr.length
    arr[i] == 2 * arr[j]
示例 1：输入：arr = [10,2,5,3] 输出：true
解释：N = 10 是 M = 5 的两倍，即 10 = 2 * 5 。
示例 2：输入：arr = [7,1,14,11] 输出：true
解释：N = 14 是 M = 7 的两倍，即 14 = 2 * 7 。
示例 3：输入：arr = [3,1,7,11] 输出：false
解释：在该情况下不存在 N 和 M 满足 N = 2 * M 。
提示：
    2 <= arr.length <= 500
```

(续下页)

(接上页)

```
    -10^3 <= arr[i] <= 10^3
```

- 解题思路

```
func checkIfExist(arr []int) bool {
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        if arr[i]*2 == arr[j] || arr[j]*2 == arr[i] {
                                return true
                        }
                }
        }
        return false
}

#
func checkIfExist(arr []int) bool {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                if m[arr[i]*2] > 0 || (i%2 == 0 && m[arr[i]/2] > 0) {
                        return true
                }
                m[arr[i]] = 1
        }
        return false
}

#
func checkIfExist(arr []int) bool {
        var target int
        sort.Ints(arr)
        for i := 0; i < len(arr); i++ {
                left := i + 1
                right := len(arr) - 1
                if arr[i] >= 0 {
                        target = 2 * arr[i]
                } else {
                        if arr[i]%2 == -1 {
                                continue
                        }
                        target = arr[i] / 2
                }
                for left <= right {
                        mid := left + (right-left)/2
```

(续下页)

(接上页)

```
                        if arr[mid] == target {
                                return true
                        } else if arr[mid] > target {
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                }
        }
        return false
}
```

40.11 1351. 统计有序矩阵中的负数 (4)

- 题目

```
给你一个 m * n 的矩阵 grid，矩阵中的元素无论是按行还是按列，都以非递增顺序排列。
请你统计并返回 grid 中 负数 的数目。
示例 1：输入：grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]] 输出：8
解释：矩阵中共有 8 个负数。
示例 2：输入：grid = [[3,2],[1,0]] 输出：0
示例 3：输入：grid = [[1,-1],[-1,-1]] 输出：3
示例 4：输入：grid = [[-1]] 输出：1
提示：
    m == grid.length
    n == grid[i].length
    1 <= m, n <= 100
    -100 <= grid[i][j] <= 100
```

- 解题思路

```
func countNegatives(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] < 0 {
                                res++
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func countNegatives(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                res = res + (len(grid[i]) - search(grid[i]))
        }
        return res
}

func search(arr []int) int {
        left := 0
        right := len(arr) - 1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] >= 0 {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return left
}

#
func countNegatives(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := len(grid[i]) - 1; j >= -1; j-- {
                        if j == -1 {
                                res = res + len(grid[i])
                                break
                        }
                        if grid[i][j] >= 0 {
                                count := len(grid[i]) - 1 - j
                                res = res + count
                                break
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
#
func countNegatives(grid [][]int) int {
        res := 0
        i := 0
        j := len(grid[0])-1
        for  i < len(grid) && j >= 0 {
                if grid[i][j] >=0{
                        res = res + len(grid[0])-j-1
                        i++
                }else {
                        j--
                }
        }
        if j < 0{
                res = res + (len(grid)-i)*len(grid[0])
        }
        return res
}
```

40.12 1356. 根据数字二进制下 1 的数目排序 (3)

- 题目

```
给你一个整数数组 arr 。请你将数组中的元素按照其二进制表示中数字 1 的数目升序排序。
如果存在多个数字二进制中 1 的数目相同，则必须将它们按照数值大小升序排列。
请你返回排序后的数组。
示例 1：输入：arr = [0,1,2,3,4,5,6,7,8] 输出：[0,1,2,4,8,3,5,6,7]
解释：[0] 是唯一一个有 0 个 1 的数。
[1,2,4,8] 都有 1 个 1 。
[3,5,6] 有 2 个 1 。
[7] 有 3 个 1 。
按照 1 的个数排序得到的结果数组为 [0,1,2,4,8,3,5,6,7]
示例 2：输入：arr = [1024,512,256,128,64,32,16,8,4,2,1]
输出：[1,2,4,8,16,32,64,128,256,512,1024]
解释：数组中所有整数二进制下都只有 1 个 1 ，所以你需要按照数值大小将它们排序。
示例 3：输入：arr = [10000,10000] 输出：[10000,10000]
示例 4：输入：arr = [2,3,5,7,11,13,17,19] 输出：[2,3,5,17,7,11,13,19]
示例 5：输入：arr = [10,100,1000,10000] 输出：[10,100,10000,1000]
提示：
    1 <= arr.length <= 500
    0 <= arr[i] <= 10^4
```

- 解题思路

```
func sortByBits(arr []int) []int {
        sort.Slice(arr, func(i, j int) bool {
                if countBit(arr[i]) == countBit(arr[j]) {
                        return arr[i] < arr[j]
                }
                return countBit(arr[i]) < countBit(arr[j])
        })
        return arr
}

func countBit(num int) int {
        res := 0
        for num > 0 {
                if num%2 == 1 {
                        res++
                }
                num = num / 2
        }
        return res
}

#
func sortByBits(arr []int) []int {
        sort.Ints(arr)
        m := make(map[int][]int, 0)
        for i := 0; i < len(arr); i++ {
                num := countBit(arr[i])
                m[num] = append(m[num], arr[i])
        }
        res := make([]int, 0)
        for i := 0; i < 32; i++ {
                for _, value := range m[i] {
                        res = append(res, value)
                }
        }
        return res
}

func countBit(num int) int {
        res := 0
        for num > 0 {
                if num%2 == 1 {
                        res++
                }
```

(续下页)

(接上页)

```
                num = num / 2
        }
        return res
}

#
func sortByBits(arr []int) []int {
        sort.Slice(arr, func(i, j int) bool {
                if bits.OnesCount32(uint32(arr[i])) == bits.
↪OnesCount32(uint32(arr[j])) {
                        return arr[i] < arr[j]
                }
                return bits.OnesCount32(uint32(arr[i])) < bits.
↪OnesCount32(uint32(arr[j]))
        })
        return arr
}
```

40.13 1360. 日期之间隔几天 (2)

- 题目

```
请你编写一个程序来计算两个日期之间隔了多少天。
日期以字符串形式给出，格式为 YYYY-MM-DD，如示例所示。
示例 1：输入：date1 = "2019-06-29", date2 = "2019-06-30" 输出：1
示例 2：输入：date1 = "2020-01-15", date2 = "2019-12-31" 输出：15
提示：
    给定的日期是 1971 年到 2100 年之间的有效日期。
```

- 解题思路

```
func daysBetweenDates(date1 string, date2 string) int {
        v1 := totalDay(date1)
        v2 := totalDay(date2)
        if v1 > v2 {
                return v1 - v2
        }
        return v2 - v1
}

var monthDate = []int{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
```

(续下页)

(接上页)

```
func totalDay(date string) int {
        var year, month, day int
        arr := strings.Split(date, "-")
        year, _ = strconv.Atoi(arr[0])
        month, _ = strconv.Atoi(arr[1])
        day, _ = strconv.Atoi(arr[2])
        total := 0
        for i := 1971; i < year; i++ {
                total = total + 365
                if isLeap(i) {
                        total = total + 1
                }
        }
        for i := 0; i < month-1; i++ {
                total = total + monthDate[i]
                if i == 1 && isLeap(year) {
                        total = total + 1
                }
        }
        total = total + day
        return total
}

func isLeap(year int) bool {
        return year%400 == 0 || (year%4 == 0 && year%100 != 0)
}

#
func daysBetweenDates(date1 string, date2 string) int {
        t1, _ := time.Parse("2006-01-02", date1)
        t2, _ := time.Parse("2006-01-02", date2)
        value := int(t1.Sub(t2).Hours() / 24)
        if value > 0 {
                return value
        }
        return -value
}
```

40.14 1365. 有多少小于当前数字的数字 (3)

- 题目

```
给你一个数组 nums，对于其中每个元素 nums[i]，请你统计数组中比它小的所有数字的数目。
换而言之，对于每个 nums[i] 你必须计算出有效的 j 的数量，其中 j 满足 j != i 且 nums[j]
↪< nums[i] 。
以数组形式返回答案。
示例 1：输入：nums = [8,1,2,2,3] 输出：[4,0,1,1,3]
解释：   对于 nums[0]=8 存在四个比它小的数字：（1，2，2 和 3）。
对于 nums[1]=1 不存在比它小的数字。
对于 nums[2]=2 存在一个比它小的数字：（1）。
对于 nums[3]=2 存在一个比它小的数字：（1）。
对于 nums[4]=3 存在三个比它小的数字：（1，2 和 2）。
示例 2：输入：nums = [6,5,4,8] 输出：[2,1,0,3]
示例 3：输入：nums = [7,7,7,7] 输出：[0,0,0,0]
提示：
    2 <= nums.length <= 500
    0 <= nums[i] <= 100
```

- 解题思路

```
func smallerNumbersThanCurrent(nums []int) []int {
        arr := make([]int, 101)
        res := make([]int, len(nums))
        for i := 0; i < len(nums); i++ {
                arr[nums[i]]++
        }
        for i := 1; i < len(arr); i++ {
                arr[i] = arr[i] + arr[i-1]
        }
        for i := 0; i < len(nums); i++ {
                if nums[i] != 0 {
                        res[i] = arr[nums[i]-1]
                }
        }
        return res
}

#
func smallerNumbersThanCurrent(nums []int) []int {
        res := make([]int, len(nums))
        for i := 0; i < len(nums); i++ {
                for j := 0; j < len(nums); j++ {
```

(续下页)

(接上页)

```
                        if nums[i] > nums[j] {
                                res[i]++
                        }
                }
        }
        return res
}

#
func smallerNumbersThanCurrent(nums []int) []int {
        temp := make([]int, len(nums))
        copy(temp, nums)
        sort.Ints(temp)
        m := make(map[int]int)
        count := 0
        m[temp[0]] = count

        for i := 1; i < len(temp); i++ {
                count++
                if temp[i-1] != temp[i] {
                        m[temp[i]] = count
                } else {
                        m[temp[i]] = m[temp[i-1]]
                }
        }
        res := make([]int, len(nums))
        for i := 0; i < len(nums); i++ {
                res[i] = m[nums[i]]
        }
        return res
}
```

40.15 1370. 上升下降字符串 (2)

- 题目

```
给你一个字符串 s ，请你根据下面的算法重新构造字符串：
    从 s 中选出 最小 的字符，将它 接在 结果字符串的后面。
    从 s 剩余字符中选出 最小 的字符，且该字符比上一个添加的字符大，将它 接在␣
↪结果字符串后面。
    重复步骤 2 ，直到你没法从 s 中选择字符。
    从 s 中选出 最大 的字符，将它 接在 结果字符串的后面。
```

(续下页)

(接上页)

```
    从 s 剩余字符中选出 最大 的字符，且该字符比上一个添加的字符小，将它 接在␣
↪结果字符串后面。
    重复步骤 5 ，直到你没法从 s 中选择字符。
    重复步骤 1 到 6 ，直到 s 中所有字符都已经被选过。
在任何一步中，如果最小或者最大字符不止一个␣
↪，你可以选择其中任意一个，并将其添加到结果字符串。
请你返回将 s 中字符重新排序后的 结果字符串 。
示例 1：输入：s = "aaaabbbbcccc" 输出："abccbaabccba"
解释：第一轮的步骤 1，2，3 后，结果字符串为 result = "abc"
第一轮的步骤 4，5，6 后，结果字符串为 result = "abccba"
第一轮结束，现在 s = "aabbcc" ，我们再次回到步骤 1
第二轮的步骤 1，2，3 后，结果字符串为 result = "abccbaabc"
第二轮的步骤 4，5，6 后，结果字符串为 result = "abccbaabccba"
示例 2：输入：s = "rat" 输出："art"
解释：单词 "rat" 在上述算法重排序以后变成 "art"
示例 3：输入：s = "leetcode" 输出："cdelotee"
示例 4：输入：s = "ggggggg" 输出："ggggggg"
示例 5：输入：s = "spo" 输出："ops"
提示：
    1 <= s.length <= 500
    s 只包含小写英文字母。
```

- 解题思路

```
func sortString(s string) string {
        arr := make([]int, 26)
        for i := 0; i < len(s); i++ {
                arr[s[i]-'a']++
        }
        res := ""
        for len(res) < len(s) {
                for i := 0; i < 26; i++ {
                        if arr[i] > 0 {
                                res = res + string(i+'a')
                                arr[i]--
                        }
                }
                for i := 25; i >= 0; i-- {
                        if arr[i] > 0 {
                                res = res + string(i+'a')
                                arr[i]--
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}

#
func sortString(s string) string {
        m := make(map[int]int, 26)
        for i := 0; i < len(s); i++ {
                m[int(s[i]-'a')]++
        }
        res := ""
        for len(res) < len(s) {
                for i := 0; i < 26; i++ {
                        if m[i] > 0 {
                                res = res + string(i+'a')
                                m[i]--
                        }
                }
                for i := 25; i >= 0; i-- {
                        if m[i] > 0 {
                                res = res + string(i+'a')
                                m[i]--
                        }
                }
        }
        return res
}
```

40.16 1374. 生成每种字符都是奇数个的字符串 (2)

- 题目

```
给你一个整数 n，请你返回一个含 n 个字符的字符串，其中每种字符在该字符串中都恰好出现␣
↪奇数次 。
返回的字符串必须只含小写英文字母。如果存在多个满足题目要求的字符串，则返回其中任意一个即可。
示例 1: 输入: n = 4 输出: "pppz"
解释: "pppz" 是一个满足题目要求的字符串，因为 'p' 出现 3 次，且 'z' 出现 1 次。
当然，还有很多其他字符串也满足题目要求，比如: "ohhh" 和 "love"。
示例 2: 输入: n = 2 输出: "xy"
解释: "xy" 是一个满足题目要求的字符串，因为 'x' 和 'y' 各出现 1 次。
当然，还有很多其他字符串也满足题目要求，比如: "ag" 和 "ur"。
示例 3: 输入: n = 7 输出: "holasss"
提示: 1 <= n <= 500
```

- 解题思路

```
func generateTheString(n int) string {
        if n % 2 == 0 {
                return strings.Repeat("a", n-1)+"b"
        }
        return strings.Repeat("a", n)
}

#
func generateTheString(n int) string {
        res := ""
        if n%2 == 0 {
                res = "a"
                for i := 0; i < n-1; i++ {
                        res = res + "b"
                }
        } else {
                for i := 0; i < n; i++ {
                        res = res + "a"
                }
        }
        return res
}
```

40.17 1379. 找出克隆二叉树中的相同节点

40.17.1 题目

```
给你两棵二叉树，原始树 original 和克隆树 cloned，以及一个位于原始树␣
↪original 中的目标节点 target。
其中，克隆树 cloned 是原始树 original 的一个 副本 。
请找出在树 cloned 中，与 target 相同 的节点，
并返回对该节点的引用（在 C/C++ 等有指针的语言中返回 节点指针，其他语言返回节点本身）。
注意：你 不能 对两棵二叉树，以及 target 节点进行更改。只能␣
↪返回对克隆树 cloned 中已有的节点的引用。
示例 1:输入：tree = [7,4,3,null,null,6,19], target = 3 输出：3
解释：上图画出了树 original 和 cloned。target 节点在树 original 中，用绿色标记。
答案是树 cloned 中的黄颜色的节点（其他示例类似）。
示例 2:输入：tree = [7], target =  7 输出：7
示例 3:输入：tree = [8,null,6,null,5,null,4,null,3,null,2,null,1], target = 4 输出：4
提示：树中节点的数量范围为 [1, 104] 。
```

(续下页)

(接上页)

```
同一棵树中，没有值相同的节点。
target 节点是树 original 中的一个节点，并且不会是 null 。
进阶：如果树中允许出现值相同的节点，将如何解答？
```

40.17.2 解题思路

40.18 1380. 矩阵中的幸运数 (2)

- 题目

```
给你一个 m * n 的矩阵，矩阵中的数字 各不相同 。请你按 任意␣
↪顺序返回矩阵中的所有幸运数。
幸运数是指矩阵中满足同时下列两个条件的元素：
    在同一行的所有元素中最小
    在同一列的所有元素中最大
示例 1：输入：matrix = [[3,7,8],[9,11,13],[15,16,17]] 输出：[15]
解释：15 是唯一的幸运数，因为它是其所在行中的最小值，也是所在列中的最大值。
示例 2：输入：matrix = [[1,10,4,2],[9,3,8,7],[15,16,17,12]] 输出：[12]
解释：12 是唯一的幸运数，因为它是其所在行中的最小值，也是所在列中的最大值。
示例 3：输入：matrix = [[7,8],[1,2]] 输出：[7]
提示：
    m == mat.length
    n == mat[i].length
    1 <= n, m <= 50
    1 <= matrix[i][j] <= 10^5
    矩阵中的所有元素都是不同的
```

- 解题思路

```
func luckyNumbers(matrix [][]int) []int {
        res := make([]int, 0)
        for i := 0; i < len(matrix); i++ {
                min := matrix[i][0]
                minIndex := 0
                for j := 1; j < len(matrix[i]); j++ {
                        if min > matrix[i][j] {
                                min = matrix[i][j]
                                minIndex = j
```

(续下页)

(接上页)

```
                        }
                }
                flag := true
                for j := 0; j < len(matrix); j++ {
                        if matrix[j][minIndex] > min {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = append(res, min)
                }
        }
        return res
}

#
func luckyNumbers(matrix [][]int) []int {
        res := make([]int, 0)
        minArr := make([]int, 0)
        maxArr := make([]int, 0)
        for i := 0; i < len(matrix); i++ {
                min := matrix[i][0]
                for j := 1; j < len(matrix[i]); j++ {
                        if min > matrix[i][j] {
                                min = matrix[i][j]
                        }
                }
                minArr = append(minArr, min)
        }
        for i := 0; i < len(matrix[0]); i++ {
                max := matrix[0][i]
                for j := 1; j < len(matrix); j++ {
                        if max < matrix[j][i] {
                                max = matrix[j][i]
                        }
                }
                maxArr = append(maxArr, max)
        }
        for i := 0; i < len(minArr); i++ {
                for j := 0; j < len(maxArr); j++ {
                        if minArr[i] == maxArr[j] {
                                res = append(res, minArr[i])
```

(续下页)

(接上页)

```
                        }
                }
        }
        return res
}
```

40.19 1385. 两个数组间的距离值 (2)

- 题目

```
给你两个整数数组 arr1 ， arr2 和一个整数 d ，请你返回两个数组之间的 距离值 。
「距离值」 定义为符合此描述的元素数目：
对于元素 arr1[i] ，不存在任何元素 arr2[j] 满足 |arr1[i]-arr2[j]| <= d 。
示例 1：输入：arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2 输出：2
解释：
对于 arr1[0]=4 我们有：
|4-10|=6 > d=2
|4-9|=5 > d=2
|4-1|=3 > d=2
|4-8|=4 > d=2
对于 arr1[1]=5 我们有：
|5-10|=5 > d=2
|5-9|=4 > d=2
|5-1|=4 > d=2
|5-8|=3 > d=2
对于 arr1[2]=8 我们有：
|8-10|=2 <= d=2
|8-9|=1 <= d=2
|8-1|=7 > d=2
|8-8|=0 <= d=2
示例 2：输入：arr1 = [1,4,2,3], arr2 = [-4,-3,6,10,20,30], d = 3 输出：2
示例 3：输入：arr1 = [2,1,100,3], arr2 = [-5,-2,10,-3,7], d = 6 输出：1
提示：
    1 <= arr1.length, arr2.length <= 500
    -10^3 <= arr1[i], arr2[j] <= 10^3
    0 <= d <= 100
```

- 解题思路

```
func findTheDistanceValue(arr1 []int, arr2 []int, d int) int {
        res := 0
        for i := 0; i < len(arr1); i++ {
```

(续下页)

(接上页)

```
                flag := true
                for j := 0; j < len(arr2); j++ {
                        if abs(arr1[i], arr2[j]) <= d {
                                flag = false
                        }
                }
                if flag == true {
                        res++
                }
        }
        return res
}

#
func findTheDistanceValue(arr1 []int, arr2 []int, d int) int {
        res := 0
        sort.Ints(arr2)
        for i := 0; i < len(arr1); i++ {
                if search(arr1[i], arr2, d) {
                        res++
                }
        }
        return res
}

func search(target int, arr []int, d int) bool {
        left := 0
        right := len(arr) - 1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] < target {
                        if target-arr[mid] <= d {
                                return false
                        }
                        left = mid + 1
                } else {
                        if arr[mid]-target <= d {
                                return false
                        }
                        right = mid - 1
                }
        }
        return true
}
```

40.20 1389. 按既定顺序创建目标数组 (3)

- 题目

```
给你两个整数数组 nums 和 index。你需要按照以下规则创建目标数组：
    目标数组 target 最初为空。
    按从左到右的顺序依次读取 nums[i] 和 index[i]，
    在 target 数组中的下标 index[i] 处插入值 nums[i] 。
    重复上一步，直到在 nums 和 index 中都没有要读取的元素。
请你返回目标数组。
题目保证数字插入位置总是存在。
示例 1：输入：nums = [0,1,2,3,4], index = [0,1,2,2,1] 输出：[0,4,1,3,2]
解释：
nums        index     target
0             0         [0]
1             1         [0,1]
2             2         [0,1,2]
3             2         [0,1,3,2]
4             1         [0,4,1,3,2]
示例 2：输入：nums = [1,2,3,4,0], index = [0,1,2,3,0] 输出：[0,1,2,3,4]
解释：
nums        index     target
1             0         [1]
2             1         [1,2]
3             2         [1,2,3]
4             3         [1,2,3,4]
0             0         [0,1,2,3,4]
示例 3：输入：nums = [1], index = [0] 输出：[1]
提示：
    1 <= nums.length, index.length <= 100
    nums.length == index.length
    0 <= nums[i] <= 100
    0 <= index[i] <= i
```

- 解题思路

```
func createTargetArray(nums []int, index []int) []int {
        res := make([]int, len(nums))
        for i := 0; i < len(index); i++ {
                for j := len(res) - 1; j > index[i]; j-- {
                        res[j] = res[j-1]
                }
                res[index[i]] = nums[i]
        }
```

(续下页)

(接上页)

```
        return res
}

#
func createTargetArray(nums []int, index []int) []int {
        res := make([]int, len(nums))
        for i := 0; i < len(index); i++ {
                copy(res[index[i]+1:], res[index[i]:])
                res[index[i]] = nums[i]
        }
        return res
}

#
func createTargetArray(nums []int, index []int) []int {
        res := make([]int, len(nums))
        for i := 0; i < len(index); i++ {
                for j := 0; j < i; j++ {
                        if index[j] >= index[i] {
                                index[j]++
                        }
                }
        }
        for i := 0; i < len(index); i++ {
                res[index[i]] = nums[i]
        }
        return res
}
```

40.21 1394. 找出数组中的幸运数 (2)

- 题目

```
在整数数组中，如果一个整数的出现频次和它的数值大小相等，我们就称这个整数为「幸运数」。
给你一个整数数组 arr，请你从中找出并返回一个幸运数。
    如果数组中存在多个幸运数，只需返回 最大 的那个。
    如果数组中不含幸运数，则返回 -1 。
示例 1：输入：arr = [2,2,3,4] 输出：2
解释：数组中唯一的幸运数是 2 ，因为数值 2 的出现频次也是 2 。
示例 2：输入：arr = [1,2,2,3,3,3] 输出：3
解释：1、2 以及 3 都是幸运数，只需要返回其中最大的 3 。
示例 3：输入：arr = [2,2,2,3,3] 输出：-1
```

(续下页)

(接上页)

```
解释：数组中不存在幸运数。
示例 4：输入：arr = [5] 输出：-1
示例 5：输入：arr = [7,7,7,7,7,7,7] 输出：7
提示：
    1 <= arr.length <= 500
    1 <= arr[i] <= 500
```

- 解题思路

```
func findLucky(arr []int) int {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]]++
        }
        max := -1
        for i := range m {
                if i == m[i] && max < i {
                        max = i
                }
        }
        return max
}

#
func findLucky(arr []int) int {
        res := make([]int, 501)
        for i := 0; i < len(arr); i++ {
                res[arr[i]]++
        }
        for i := 500; i >= 1; i-- {
                if res[i] == i {
                        return i
                }
        }
        return -1
}
```

40.22 1399. 统计最大组的数目 (2)

- 题目

```
给你一个整数 n 。请你先求出从 1 到 n 的每个整数 10␣
↪进制表示下的数位和（每一位上的数字相加），
然后把数位和相等的数字放到同一个组中。
请你统计每个组中的数字数目，并返回数字数目并列最多的组有多少个。
示例 1：输入：n = 13 输出：4
解释：总共有 9 个组，将 1 到 13 按数位求和后这些组分别是：
[1,10]，[2,11]，[3,12]，[4,13]，[5]，[6]，[7]，[8]，[9]。总共有 4␣
↪个组拥有的数字并列最多。
示例 2：输入：n = 2 输出：2
解释：总共有 2 个大小为 1 的组 [1]，[2]。
示例 3：输入：n = 15 输出：6
示例 4：输入：n = 24 输出：5
提示：1 <= n <= 10^4
```

- 解题思路

```
func countLargestGroup(n int) int {
        if n < 10 {
                return n
        }
        m := make(map[int]int,50)
        max := 0
        for i := 1; i <= n; i++ {
                value := sum(i)
                m[value]++
                if m[value] > max {
                        max = m[value]
                }
        }
        res := 0
        for i := range m {
                if m[i] == max {
                        res++
                }
        }
        return res
}

func sum(n int) int {
        res := 0
```

(续下页)

(接上页)

```
        for n > 0 {
                res = res + n%10
                n = n / 10
        }
        return res
}

#
func countLargestGroup(n int) int {
        if n < 10 {
                return n
        }
        arr := make([]int, 50)
        max := 0
        for i := 1; i <= n; i++ {
                value := sum(i)
                arr[value]++
                if arr[value] > max {
                        max = arr[value]
                }
        }
        res := 0
        for i := range arr {
                if arr[i] == max {
                        res++
                }
        }
        return res
}

func sum(n int) int {
        res := 0
        for n > 0 {
                res = res + n%10
                n = n / 10
        }
        return res
}
```

CHAPTER 41

1301-1400-Medium

41.1 1302. 层数最深叶子节点的和 (2)

- 题目

```
给你一棵二叉树，请你返回层数最深的叶子节点的和。
示例：输入：root = [1,2,3,4,5,null,6,7,null,null,null,null,8] 输出：15
提示：树中节点数目在 1 到 10^4 之间。
    每个节点的值在 1 到 100 之间。
```

- 解题思路

```
var maxLevel, sum int

func deepestLeavesSum(root *TreeNode) int {
        maxLevel, sum = 0, 0
        dfs(root, 0)
        return sum
}

func dfs(root *TreeNode, level int) {
        if root != nil {
                if level > maxLevel {
                        maxLevel = level
                        sum = root.Val
```

(续下页)

(接上页)

```
                } else if level == maxLevel {
                        sum = sum + root.Val
                }
                dfs(root.Left, level+1)
                dfs(root.Right, level+1)
        }
}

# 2
func deepestLeavesSum(root *TreeNode) int {
        if root == nil {
                return 0
        }
        res := 0
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                res = 0
                for i := 0; i < length; i++ {
                        res = res + queue[i].Val
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                }
                queue = queue[length:]
        }
        return res
}
```

41.2 1305. 两棵二叉搜索树中的所有元素 (3)

- 题目

```
给你 root1 和 root2 这两棵二叉搜索树。
请你返回一个列表，其中包含 两棵树 中的所有整数并按 升序 排序。.
示例 1：输入：root1 = [2,1,4], root2 = [1,0,3] 输出：[0,1,1,2,3,4]
示例 2：输入：root1 = [0,-10,10], root2 = [5,1,7,0,2] 输出：[-10,0,0,1,2,5,7,10]
示例 3：输入：root1 = [], root2 = [5,1,7,0,2] 输出：[0,1,2,5,7]
```

(续下页)

(接上页)

```
示例 4：输入：root1 = [0,-10,10], root2 = [] 输出：[-10,0,10]
示例 5：输入：root1 = [1,null,8], root2 = [8,1] 输出：[1,1,8,8]
提示：每棵树最多有 5000 个节点。
    每个节点的值在 [-10^5, 10^5] 之间。
```

- 解题思路

```
var res []int

func getAllElements(root1 *TreeNode, root2 *TreeNode) []int {
        res = make([]int, 0)
        dfs(root1)
        dfs(root2)
        sort.Ints(res)
        return res
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        res = append(res, root.Val)
        dfs(root.Left)
        dfs(root.Right)
}

# 2
func getAllElements(root1 *TreeNode, root2 *TreeNode) []int {
        res := make([]int, 0)
        arr1, arr2 := make([]int, 0), make([]int, 0)
        dfs(root1, &arr1)
        dfs(root2, &arr2)
        i, j := 0, 0
        for i < len(arr1) || j < len(arr2) {
                if i < len(arr1) && (j == len(arr2) || arr1[i] < arr2[j]) {
                        res = append(res, arr1[i])
                        i++
                } else {
                        res = append(res, arr2[j])
                        j++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func dfs(root *TreeNode, arr *[]int) {
        if root == nil {
                return
        }
        dfs(root.Left, arr)
        *arr = append(*arr, root.Val)
        dfs(root.Right, arr)
}

# 3
func getAllElements(root1 *TreeNode, root2 *TreeNode) []int {
        res := make([]int, 0)
        arr1 := inorderTraversal(root1)
        arr2 := inorderTraversal(root2)
        i, j := 0, 0
        for i < len(arr1) || j < len(arr2) {
                if i < len(arr1) && (j == len(arr2) || arr1[i] < arr2[j]) {
                        res = append(res, arr1[i])
                        i++
                } else {
                        res = append(res, arr2[j])
                        j++
                }
        }
        return res
}

func inorderTraversal(root *TreeNode) []int {
        if root == nil {
                return nil
        }
        stack := make([]*TreeNode, 0)
        res := make([]int, 0)
        for len(stack) > 0 || root != nil {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                last := len(stack) - 1
                res = append(res, stack[last].Val)
                root = stack[last].Right
                stack = stack[:last]
```

(续下页)

(接上页)

```
        }
        return res
}
```

41.3 1306. 跳跃游戏 III(2)

- 题目

```
这里有一个非负整数数组 arr，你最开始位于该数组的起始下标 start 处。
当你位于下标 i 处时，你可以跳到 i + arr[i] 或者 i - arr[i]。
请你判断自己是否能够跳到对应元素值为 0 的 任一 下标处。
注意，不管是什么情况下，你都无法跳到数组之外。
示例 1：输入：arr = [4,2,3,0,3,1,2], start = 5 输出：true
解释：到达值为 0 的下标 3 有以下可能方案：
下标 5 -> 下标 4 -> 下标 1 -> 下标 3
下标 5 -> 下标 6 -> 下标 4 -> 下标 1 -> 下标 3
示例 2：输入：arr = [4,2,3,0,3,1,2], start = 0 输出：true
解释：到达值为 0 的下标 3 有以下可能方案：
下标 0 -> 下标 4 -> 下标 1 -> 下标 3
示例 3：输入：arr = [3,0,2,1,2], start = 2 输出：false
解释：无法到达值为 0 的下标 1 处。
提示：
    1 <= arr.length <= 5 * 10^4
    0 <= arr[i] < arr.length
    0 <= start < arr.length
```

- 解题思路

```
func canReach(arr []int, start int) bool {
        m := make(map[int]bool)
        queue := make([]int, 0)
        queue = append(queue, start)
        for len(queue) > 0 {
                length := len(queue)
                for j := 0; j < length; j++ {
                        i := queue[j]
                        if m[i] == false {
                                m[i] = true
                                if i+arr[i] < len(arr) {
                                        if arr[i+arr[i]] == 0 {
                                                return true
                                        }
```

(续下页)

(接上页)

```
                                        queue = append(queue, i+arr[i])
                                }
                                if i-arr[i] >= 0 {
                                        if arr[i-arr[i]] == 0 {
                                                return true
                                        }
                                        queue = append(queue, i-arr[i])
                                }
                        }
                }
                queue = queue[length:]
        }
        return false
}

# 2
var m map[int]bool

func canReach(arr []int, start int) bool {
        m = make(map[int]bool)
        return dfs(arr, start)
}

func dfs(arr []int, i int) bool {
        if i < 0 || i > len(arr)-1 || m[i] == true {
                return false
        }
        m[i] = true
        return arr[i] == 0 || dfs(arr, i+arr[i]) || dfs(arr, i-arr[i])
}
```

41.4 1310. 子数组异或查询 (1)

- 题目

```
有一个正整数数组arr，现给你一个对应的查询数组queries，其中queries[i] = [Li,Ri]。
对于每个查询i，请你计算从Li到Ri的XOR值
（即arr[Li] xor arr[Li+1] xor ... xor arr[Ri]）作为本次查询的结果。
并返回一个包含给定查询queries所有结果的数组。
示例 1：输入：arr = [1,3,4,8], queries = [[0,1],[1,2],[0,3],[3,3]] 输出：[2,7,14,8]
解释：数组中元素的二进制表示形式是：
1 = 0001
```

(续下页)

(接上页)

```
3 = 0011
4 = 0100
8 = 1000
查询的 XOR 值为：
[0,1] = 1 xor 3 = 2
[1,2] = 3 xor 4 = 7
[0,3] = 1 xor 3 xor 4 xor 8 = 14
[3,3] = 8
示例 2：输入：arr = [4,8,2,10], queries = [[2,3],[1,3],[0,0],[0,3]] 输出：[8,0,4,4]
提示：1 <= arr.length <= 3 *10^4
1 <= arr[i] <= 10^9
1 <= queries.length <= 3 * 10^4
queries[i].length == 2
0 <= queries[i][0] <= queries[i][1] < arr.length
```

- 解题思路

```
func xorQueries(arr []int, queries [][]int) []int {
        temp := make([]int, len(arr)+1)
        for i := 0; i < len(arr); i++ {
                temp[i+1] = temp[i] ^ arr[i]
        }
        res := make([]int, 0)
        for i := 0; i < len(queries); i++ {
                a, b := queries[i][0], queries[i][1]+1
                res = append(res, temp[a]^temp[b])
        }
        return res
}
```

41.5 1311. 获取你好友已观看的视频 (1)

- 题目

```
有n 个人，每个人都有一个 0到n-1的唯一id。
给你数组 watchedVideos 和friends，
其中watchedVideos[i] 和friends[i]分别表示id = i的人观看过的视频列表和他的好友列表。
Level1的视频包含所有你好友观看过的视频，level2的视频包含所有你好友的好友观看过的视频，以此类推。
一般的，Level 为 k的视频包含所有从你出发，最短距离为k的好友观看过的视频。
给定你的id 和一个level值，请你找出所有指定 level 的视频，并将它们按观看频率升序返回。
如果有频率相同的视频，请将它们按字母顺序从小到大排列。
示例 1：输入：watchedVideos = [["A","B"],["C"],["B","C"],["D"]],
```

(续下页)

(接上页)

```
friends = [[1,2],[0,3],[0,3],[1,2]], id = 0, level = 1
输出：["B","C"]
解释：你的 id 为 0（绿色），你的朋友包括（黄色）：
id 为 1 -> watchedVideos = ["C"]
id 为 2 -> watchedVideos = ["B","C"]
你朋友观看过视频的频率为：
B -> 1
C -> 2
示例 2：输入：watchedVideos = [["A","B"],["C"],["B","C"],["D"]],
friends = [[1,2],[0,3],[0,3],[1,2]], id = 0, level = 2
输出：["D"]
解释：你的 id 为 0（绿色），你朋友的朋友只有一个人，他的 id 为 3（黄色）。
提示：
n == watchedVideos.length ==friends.length
2 <= n<= 100
1 <=watchedVideos[i].length <= 100
1 <=watchedVideos[i][j].length <= 8
0 <= friends[i].length < n
0 <= friends[i][j]< n
0 <= id < n
1 <= level < n
如果friends[i] 包含j，那么friends[j] 包含i
```

- 解题思路

```
func watchedVideosByFriends(watchedVideos [][]string, friends [][]int, id int, level␣
↪int) []string {
        m := make(map[string]int)
        visited := make(map[int]bool)
        visited[id] = true
        queue := make([]int, 0)
        queue = append(queue, id)
        for len(queue) > 0 {
                level--
                length := len(queue)
                for i := 0; i < length; i++ {
                        node := queue[i]
                        for j := 0; j < len(friends[node]); j++ {
                                ID := friends[node][j]
                                if visited[ID] == false {
                                        visited[ID] = true
                                        queue = append(queue, ID)
                                        if level == 0 {
                                                for _, v := range watchedVideos[ID] {
```

(续下页)

(接上页)

```
                                                        m[v]++
                                                }
                                        }
                                }
                        }
                }
                if level == 0 {
                        break
                }
                queue = queue[length:]
        }
        res := make([]string, 0)
        for k := range m {
                res = append(res, k)
        }
        sort.Slice(res, func(i, j int) bool {
                if m[res[i]] == m[res[j]] {
                        return res[i] < res[j]
                }
                return m[res[i]] < m[res[j]]
        })
        return res
}
```

41.6 1314. 矩阵区域和 (2)

- 题目

```
给你一个m * n的矩阵mat和一个整数K ，请你返回一个矩阵answer，
其中每个answer[i][j]是所有满足下述条件的元素mat[r][c] 的和：
i - K <= r <= i + K, j - K <= c <= j + K
(r, c)在矩阵内。
示例 1：输入：mat = [[1,2,3],[4,5,6],[7,8,9]], K = 1
输出：[[12,21,16],[27,45,33],[24,39,28]]
示例 2：输入：mat = [[1,2,3],[4,5,6],[7,8,9]], K = 2
输出：[[45,45,45],[45,45,45],[45,45,45]]
提示：m ==mat.length
n ==mat[i].length
1 <= m, n, K <= 100
1 <= mat[i][j] <= 100
```

- 解题思路

```
func matrixBlockSum(mat [][]int, K int) [][]int {
    n, m := len(mat), len(mat[0])
    arr := make([][]int, n+1)
    for i := 0; i < n+1; i++ {
            arr[i] = make([]int, m+1)
    }
    for i := 1; i <= n; i++ {
            for j := 1; j <= m; j++ {
                    arr[i][j] = arr[i][j-1] + arr[i-1][j] - arr[i-1][j-1] + mat[i-1][j-
↪1]
            }
    }
    res := make([][]int, n)
    for i := 0; i < n; i++ {
            res[i] = make([]int, m)
            for j := 0; j < m; j++ {
                    a1, a2 := getIndex(n, m, i+K+1, j+K+1)
                    b1, b2 := getIndex(n, m, i-K, j+K+1)
                    c1, c2 := getIndex(n, m, i+K+1, j-K)
                    d1, d2 := getIndex(n, m, i-K, j-K)
                    res[i][j] = arr[a1][a2] - arr[b1][b2] - arr[c1][c2] + arr[d1][d2]
            }
    }
    return res
}

func getIndex(a, b, x, y int) (int, int) {
    x = max(min(a, x), 0)
    y = max(min(b, y), 0)
    return x, y
}

func max(a, b int) int {
    if a > b {
            return a
    }
    return b
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
```

(续下页)

(接上页)

```
}

# 2
func matrixBlockSum(mat [][]int, K int) [][]int {
    n, m := len(mat), len(mat[0])
    arr := make([][]int, n+1)
    for i := 0; i < n+1; i++ {
            arr[i] = make([]int, m+1)
    }
    for i := 1; i <= n; i++ {
            for j := 1; j <= m; j++ {
                    arr[i][j] = arr[i][j-1] + arr[i-1][j] - arr[i-1][j-1] + mat[i-1][j-
↪1]
            }
    }
    res := make([][]int, n)
    for i := 0; i < n; i++ {
            res[i] = make([]int, m)
            for j := 0; j < m; j++ {
                    left, right := max(0, j-K), min(m, j+K+1)
                    up, down := max(0, i-K), min(n, i+K+1)
                    res[i][j] = arr[down][right] - arr[down][left] - arr[up][right] +␣
↪arr[up][left]
            }
    }
    return res
}

func max(a, b int) int {
    if a > b {
            return a
    }
    return b
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
}
```

41.7 1315. 祖父节点值为偶数的节点和 (3)

- 题目

```
给你一棵二叉树，请你返回满足以下条件的所有节点的值之和：
该节点的祖父节点的值为偶数。（一个节点的祖父节点是指该节点的父节点的父节点。）
如果不存在祖父节点值为偶数的节点，那么返回0 。
示例：输入：root = [6,7,8,2,7,1,3,9,null,1,4,null,null,null,5] 输出：18
解释：图中红色节点的祖父节点的值为偶数，蓝色节点为这些红色节点的祖父节点。
提示：树中节点的数目在1 到10^4之间。
每个节点的值在1 到100 之间。
```

- 解题思路

```
func sumEvenGrandparent(root *TreeNode) int {
        res := 0
        if root == nil {
                return res
        }
        if root.Val%2 == 0 {
                if root.Left != nil && root.Left.Left != nil {
                        res = res + root.Left.Left.Val
                }
                if root.Left != nil && root.Left.Right != nil {
                        res = res + root.Left.Right.Val
                }
                if root.Right != nil && root.Right.Left != nil {
                        res = res + root.Right.Left.Val
                }
                if root.Right != nil && root.Right.Right != nil {
                        res = res + root.Right.Right.Val
                }
        }
        res = res + sumEvenGrandparent(root.Left)
        res = res + sumEvenGrandparent(root.Right)
        return res
}

# 2
var res int

func sumEvenGrandparent(root *TreeNode) int {
        res = 0
        if root == nil {
```

(续下页)

(接上页)

```
                return res
        }
        dfs(root, false, false)
        return res
}

func dfs(root *TreeNode, grandfather, father bool) {
        if root == nil {
                return
        }
        if grandfather == true {
                res = res + root.Val
        }
        flag := true
        if root.Val%2 == 1 {
                flag = false
        }
        dfs(root.Left, father, flag)
        dfs(root.Right, father, flag)
}

# 3
func sumEvenGrandparent(root *TreeNode) int {
        res := 0
        if root == nil {
                return res
        }
        quque := make([]*TreeNode, 0)
        quque = append(quque, root)
        for len(quque) > 0 {
                length := len(quque)
                for i := 0; i < length; i++ {
                        node := quque[i]
                        if node.Val%2 == 0 {
                                if node.Left != nil && node.Left.Left != nil {
                                        res = res + node.Left.Left.Val
                                }
                                if node.Left != nil && node.Left.Right != nil {
                                        res = res + node.Left.Right.Val
                                }
                                if node.Right != nil && node.Right.Left != nil {
                                        res = res + node.Right.Left.Val
                                }
```

(续下页)

(接上页)

```
                                if node.Right != nil && node.Right.Right != nil {
                                        res = res + node.Right.Right.Val
                                }
                        }
                        if node.Left != nil {
                                quque = append(quque, node.Left)
                        }
                        if node.Right != nil {
                                quque = append(quque, node.Right)
                        }
                }
                quque = quque[length:]
        }
        return res
}
```

41.8 1318. 或运算的最小翻转次数 (2)

- 题目

```
给你三个正整数a、b 和 c。
你可以对 a 和 b的二进制表示进行位翻转操作，返回能够使按位或运算 a OR b ==␣
↪c成立的最小翻转次数。
「位翻转操作」是指将一个数的二进制表示任何单个位上的 1 变成 0 或者 0 变成 1 。
示例 1：输入：a = 2, b = 6, c = 5 输出：3
解释：翻转后 a = 1 , b = 4 , c = 5 使得 a OR b == c
示例 2：输入：a = 4, b = 2, c = 7 输出：1
示例 3：输入：a = 1, b = 2, c = 3 输出：0
提示：1 <= a <= 10^9
1 <= b<= 10^9
1 <= c<= 10^9
```

- 解题思路

```
func minFlips(a int, b int, c int) int {
        res := 0
        for i := 0; i < 31; i++ {
                A := (a >> i) & 1
                B := (b >> i) & 1
                C := (c >> i) & 1
                if C == 0 {
                        res = res + A + B
```

(续下页)

(接上页)

```
                } else {
                        if A+B == 0 {
                                res = res + 1
                        }
                }
        }
        return res
}

# 2
func minFlips(a int, b int, c int) int {
        res := 0
        for i := 0; i < 31; i++ {
                A := a & 1
                B := b & 1
                C := c & 1
                if C == 0 {
                        res = res + A + B
                } else {
                        if A+B == 0 {
                                res = res + 1
                        }
                }
                a, b, c = a>>1, b>>1, c>>1
        }
        return res
}
```

41.9 1319. 连通网络的操作次数 (2)

- 题目

```
用以太网线缆将n台计算机连接成一个网络，计算机的编号从0到n-1。
线缆用connections表示，其中connections[i] = [a, b]连接了计算机a和b。
网络中的任何一台计算机都可以通过网络直接或者间接访问同一个网络中其他任意一台计算机。
给你这个计算机网络的初始布线connections，你可以拔开任意两台直连计算机之间的线缆，
并用它连接一对未直连的计算机。
请你计算并返回使所有计算机都连通所需的最少操作次数。如果不可能，则返回-1 。
示例 1：输入：n = 4, connections = [[0,1],[0,2],[1,2]] 输出：1
解释：拔下计算机 1 和 2 之间的线缆，并将它插到计算机 1 和 3 上。
示例 2：输入：n = 6, connections = [[0,1],[0,2],[0,3],[1,2],[1,3]] 输出：2
示例 3：输入：n = 6, connections = [[0,1],[0,2],[0,3],[1,2]] 输出：-1
```

(续下页)

(接上页)

```
解释：线缆数量不足。
示例 4：输入：n = 5, connections = [[0,1],[0,2],[3,4],[2,3]] 输出：0
提示：1 <= n <= 10^5
1 <= connections.length <= min(n*(n-1)/2, 10^5)
connections[i].length == 2
0 <= connections[i][0], connections[i][1]< n
connections[i][0] != connections[i][1]
没有重复的连接。
两台计算机不会通过多条线缆连接。
```

- 解题思路

```
func makeConnected(n int, connections [][]int) int {
        if len(connections) < n-1 {
                return -1
        }
        res := n - 1
        fa := make([]int, n)
        for i := 0; i < n; i++ {
                fa[i] = i
        }
        for i := 0; i < len(connections); i++ {
                a, b := connections[i][0], connections[i][1]
                if find(fa, a) != find(fa, b) {
                        union(fa, a, b)
                        res--
                }
        }
        return res
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}

func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
                a = fa[a]
        }
        return a
}

# 2
```

(续下页)

(接上页)

```
var m map[int][]int
var visited []bool

func makeConnected(n int, connections [][]int) int {
        if len(connections) < n-1 {
                return -1
        }
        m = make(map[int][]int)
        visited = make([]bool, n)
        for i := 0; i < len(connections); i++ {
                a, b := connections[i][0], connections[i][1]
                m[a] = append(m[a], b)
                m[b] = append(m[b], a)
    }
        res := 0
        for i := 0; i < n; i++ {
                if visited[i] == false {
                        dfs(i)
                        res++
                }
        }
    // 连通子图数量-1
        return res - 1
}

func dfs(i int) {
        if visited[i] == true {
                return
        }
        visited[i] = true
        for j := 0; j < len(m[i]); j++ {
                dfs(m[i][j])
        }
}
```

41.10 1324. 竖直打印单词 (2)

- 题目

```
给你一个字符串s。请你按照单词在 s 中的出现顺序将它们全部竖直返回。
单词应该以字符串列表的形式返回，必要时用空格补位，但输出尾部的空格需要删除（不允许尾随空格）。
每个单词只能放在一列上，每一列中也只能有一个单词。
示例 1：输入：s = "HOW ARE YOU" 输出：["HAY","ORO","WEU"]
解释：每个单词都应该竖直打印。
 "HAY"
"ORO"
"WEU"
示例 2：输入：s = "TO BE OR NOT TO BE" 输出：["TBONTB","OEROOE","   T"]
解释：题目允许使用空格补位，但不允许输出末尾出现空格。
"TBONTB"
"OEROOE"
"   T"
示例 3：输入：s = "CONTEST IS COMING" 输出：["CIC","OSO","N M","T I","E N","S G","T"]
提示：1 <= s.length <= 200
s仅含大写英文字母。
题目数据保证两个单词之间只有一个空格。
```

- 解题思路

```
func printVertically(s string) []string {
        arr := strings.Split(s, " ")
        n := len(arr)
        maxLength := 0
        for i := 0; i < n; i++ {
                if len(arr[i]) > maxLength {
                        maxLength = len(arr[i])
                }
        }
        temp := make([][]byte, maxLength)
        for i := 0; i < maxLength; i++ {
                temp[i] = make([]byte, n)
                for j := 0; j < n; j++ {
                        temp[i][j] = ' '
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < len(arr[i]); j++ {
                        temp[j][i] = arr[i][j]
                }
```

(续下页)

(接上页)

```
        }
        res := make([]string, 0)
        for i := 0; i < len(temp); i++ {
                res = append(res, strings.TrimRight(string(temp[i]), " "))
        }
        return res
}

# 2
func printVertically(s string) []string {
        arr := strings.Split(s, " ")
        n := len(arr)
        maxLength := 0
        for i := 0; i < n; i++ {
                if len(arr[i]) > maxLength {
                        maxLength = len(arr[i])
                }
        }
        res := make([]string, 0)
        for i := 0; i < maxLength; i++ {
                temp := make([]byte, 0)
                for j := 0; j < len(arr); j++ {
                        if len(res) < len(arr[j]) {
                                temp = append(temp, arr[j][i])
                        } else {
                                temp = append(temp, ' ')
                        }
                }
                res = append(res, strings.TrimRight(string(temp), " "))
        }
        return res
}
```

41.11 1325. 删除给定值的叶子节点 (1)

- 题目

```
给你一棵以root为根的二叉树和一个整数target，请你删除所有值为target 的叶子节点 。
注意，一旦删除值为target的叶子节点，它的父节点就可能变成叶子节点；
如果新叶子节点的值恰好也是target ，那么这个节点也应该被删除。
也就是说，你需要重复此过程直到不能继续删除。
示例 1：输入：root = [1,2,3,2,null,2,4], target = 2 输出：[1,null,3,null,4]
```

(续下页)

(接上页)

```
解释：上面左边的图中，绿色节点为叶子节点，且它们的值与 target 相同（同为 2），
它们会被删除，得到中间的图。
有一个新的节点变成了叶子节点且它的值与 target␣
↪相同，所以将再次进行删除，从而得到最右边的图。
示例 2：输入：root = [1,3,3,3,2], target = 3 输出：[1,3,null,null,2]
示例 3：输入：root = [1,2,null,2,null,2], target = 2 输出：[1]
解释：每一步都删除一个绿色的叶子节点（值为 2）。
示例 4：输入：root = [1,1,1], target = 1 输出：[]
示例 5：输入：root = [1,2,3], target = 1 输出：[1,2,3]
提示：1 <= target<= 1000
每一棵树最多有 3000 个节点。
每一个节点值的范围是[1, 1000]。
```

- 解题思路

```
func removeLeafNodes(root *TreeNode, target int) *TreeNode {
        if root == nil {
                return root
        }
        root.Left = removeLeafNodes(root.Left, target)
        root.Right = removeLeafNodes(root.Right, target)
        if root.Left == nil && root.Right == nil && root.Val == target {
                return nil
        }
        return root
}
```

41.12 1328. 破坏回文串 (1)

- 题目

```
给你一个回文字符串palindrome ，请你将其中一个 字符用任意小写英文字母替换，
使得结果字符串的字典序最小，且不是回文串。
请你返回结果字符串。如果无法做到，则返回一个空串。
示例 1：输入：palindrome = "abccba" 输出："aaccba"
示例 2：输入：palindrome = "a" 输出：""
提示：1 <= palindrome.length <= 1000
palindrome只包含小写英文字母。
```

- 解题思路

```
func breakPalindrome(palindrome string) string {
        if len(palindrome) < 2 {
                return ""
        }
        for i := 0; i < len(palindrome)/2; i++ {
                if palindrome[i] != 'a' {
                        return palindrome[:i] + "a" + palindrome[i+1:]
                }
        }
        return palindrome[:len(palindrome)-1] + "b"
}
```

41.13 1329. 将矩阵按对角线排序 (2)

- 题目

```
给你一个m * n的整数矩阵mat，请你将同一条对角线上的元素（从左上到右下）按升序排序后，
返回排好序的矩阵。
示例 1：输入：mat = [[3,3,1,1],[2,2,1,2],[1,1,1,2]] 输出：[[1,1,1,1],[1,2,2,2],[1,2,3,
↪3]]
提示：m ==mat.length
n ==mat[i].length
1 <= m, n<= 100
1 <= mat[i][j] <= 100
```

- 解题思路

```
func diagonalSort(mat [][]int) [][]int {
        m := make(map[int][]int)
        for i := 0; i < len(mat); i++ {
                for j := 0; j < len(mat[i]); j++ {
                        m[i-j] = append(m[i-j], mat[i][j])
                }
        }
        for _, v := range m {
                sort.Ints(v)
        }
        for i := 0; i < len(mat); i++ {
                for j := 0; j < len(mat[i]); j++ {
                        mat[i][j] = m[i-j][0]
                        m[i-j] = m[i-j][1:]
                }
        }
```

(续下页)

(接上页)

```
        return mat
}

# 2
func diagonalSort(mat [][]int) [][]int {
        m, n := len(mat), len(mat[0])
        for k := 0; k < min(m, n); k++ {
                for i := 0; i < m-1; i++ {
                        for j := 0; j < n-1; j++ {
                                if mat[i][j] > mat[i+1][j+1] {
                                        mat[i][j], mat[i+1][j+1] = mat[i+1][j+1],␣
↪mat[i][j]
                                }
                        }
                }
        }
        return mat
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

41.14 1333. 餐厅过滤器 (1)

- 题目

```
给你一个餐馆信息数组 restaurants，其中 restaurants[i] =
[idi, ratingi, veganFriendlyi, pricei, distancei]。
你必须使用以下三个过滤器来过滤这些餐馆信息。
其中素食者友好过滤器 veganFriendly 的值可以为 true 或者 false，
如果为 true 就意味着你应该只包括 veganFriendlyi 为 true 的餐馆，为 false␣
↪则意味着可以包括任何餐馆。
此外，我们还有最大价格 maxPrice 和最大距离 maxDistance 两个过滤器，
它们分别考虑餐厅的价格因素和距离因素的最大值。
过滤后返回餐馆的 id，按照 rating 从高到低排序。如果 rating 相同，那么按 id␣
↪从高到低排序。
简单起见， veganFriendlyi 和 veganFriendly 为 true 时取值为 1，为 false 时，取值为 0␣
↪。
```

(续下页)

(接上页)

```
示例 1：输入：restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],
[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 1, maxPrice = 50, maxDistance = 10
输出：[3,1,5]
解释：  这些餐馆为：
餐馆 1 [id=1, rating=4, veganFriendly=1, price=40, distance=10]
餐馆 2 [id=2, rating=8, veganFriendly=0, price=50, distance=5]
餐馆 3 [id=3, rating=8, veganFriendly=1, price=30, distance=4]
餐馆 4 [id=4, rating=10, veganFriendly=0, price=10, distance=3]
餐馆 5 [id=5, rating=1, veganFriendly=1, price=15, distance=1]
在按照 veganFriendly = 1, maxPrice = 50 和 maxDistance = 10 进行过滤后，
我们得到了餐馆 3, 餐馆 1 和 餐馆 5（按评分从高到低排序）。
示例 2：输入：restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],
[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 0, maxPrice = 50, maxDistance = 10
输出：[4,3,2,1,5]
解释：餐馆与示例 1 相同，但在 veganFriendly = 0 的过滤条件下，应该考虑所有餐馆。
示例 3：输入：restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],
[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 0, maxPrice = 30, maxDistance = 3
输出：[4,5]
提示：1 <= restaurants.length <= 10^4
    restaurants[i].length == 5
    1 <= idi, ratingi, pricei, distancei <= 10^5
    1 <= maxPrice, maxDistance <= 10^5
    veganFriendlyi 和 veganFriendly 的值为 0 或 1 。
    所有 idi 各不相同。
```

- 解题思路

```
func filterRestaurants(restaurants [][]int, veganFriendly int, maxPrice int,␣
↪maxDistance int) []int {
        res := make([]int, 0)
        arr := make([][2]int, 0)
        for i := 0; i < len(restaurants); i++ {
                if restaurants[i][3] > maxPrice || restaurants[i][4] > maxDistance {
                        continue
                }
                if veganFriendly == 1 {
                        if restaurants[i][2] == 0 {
                                continue
                        }
                }
                arr = append(arr, [2]int{restaurants[i][0], restaurants[i][1]})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][1] == arr[j][1] {
```

(续下页)

(接上页)

```
                        return arr[i][0] > arr[j][0]
                }
                return arr[i][1] > arr[j][1]
        })
        for i := 0; i < len(arr); i++ {
                res = append(res, arr[i][0])
        }
        return res
}
```

41.15 1334. 阈值距离内邻居最少的城市

41.15.1 题目

```
有 n个城市，按从 0 到 n-1编号。
给你一个边数组edges，其中 edges[i] = [fromi, toi,␣
↪weighti]代表fromi和toi两个城市之间的双向加权边，
距离阈值是一个整数distanceThreshold。
返回能通过某些路径到达其他城市数目最少、且路径距离 最大 为distanceThreshold的城市。
如果有多个这样的城市，则返回编号最大的城市。
注意，连接城市 i 和 j 的路径的距离等于沿该路径的所有边的权重之和。
示例 1：输入：n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4␣
↪输出：3
解释：城市分布图如上。
每个城市阈值距离 distanceThreshold = 4 内的邻居城市分别是：
城市 0 -> [城市 1, 城市 2]
城市 1 -> [城市 0, 城市 2, 城市 3]
城市 2 -> [城市 0, 城市 1, 城市 3]
城市 3 -> [城市 1, 城市 2]
城市 0 和 3 在阈值距离 4 以内都有 2 个邻居城市，但是我们必须返回城市␣
↪3，因为它的编号最大。
示例 2：输入：n = 5, edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]],␣
↪distanceThreshold = 2 输出：0
解释：城市分布图如上。
每个城市阈值距离 distanceThreshold = 2 内的邻居城市分别是：
城市 0 -> [城市 1]
城市 1 -> [城市 0, 城市 4]
城市 2 -> [城市 3, 城市 4]
城市 3 -> [城市 2, 城市 4]
城市 4 -> [城市 1, 城市 2, 城市 3]
城市 0 在阈值距离 2 以内只有 1 个邻居城市。
```

(续下页)

(接上页)

```
提示：2 <= n <= 100
1 <= edges.length <= n * (n - 1) / 2
edges[i].length == 3
0 <= fromi < toi < n
1 <= weighti,distanceThreshold <= 10^4
所有 (fromi, toi)都是不同的。
```

41.15.2 解题思路

```
func findTheCity(n int, edges [][]int, distanceThreshold int) int {
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = math.MaxInt32
                }
                arr[i][i] = 0
        }
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1], edges[i][2] // a<=>b
                arr[a][b], arr[b][a] = c, c
        }
        for k := 0; k < n; k++ { // floyd
                for i := 0; i < n; i++ {
                        for j := 0; j < n; j++ {
                                if arr[i][k] != math.MaxInt32 && arr[k][j] != math.
↪MaxInt32 &&
                                        arr[i][j] >= arr[i][k]+arr[k][j] {
                                        arr[i][j] = arr[i][k] + arr[k][j]
                                }
                        }
                }
        }
        res := -1
        minCount := math.MaxInt32
        for i := 0; i < n; i++ {
                count := 0
                for j := 0; j < n; j++ {
                        if arr[i][j] <= distanceThreshold && i != j {
                                count++
                        }
                }
```

(续下页)

(接上页)

```
                if count <= minCount {
                        minCount = count
                        res = i // 等于的时候更新为较大的编号
                }
        }
        return res
}
```

41.16 1338. 数组大小减半 (2)

- 题目

```
给你一个整数数组arr。你可以从中选出一个整数集合，并删除这些整数在数组中的每次出现。
返回至少能删除数组中的一半整数的整数集合的最小大小。
示例 1：输入：arr = [3,3,3,3,5,5,5,2,2,7] 输出：2
解释：选择 {3,7} 使得结果数组为 [5,5,5,2,2]、长度为 5（原数组长度的一半）。
大小为 2 的可行集合有 {3,5},{3,2},{5,2}。
选择 {2,7} 是不可行的，它的结果数组为 [3,3,3,3,5,5,
↪5]，新数组长度大于原数组的二分之一。
示例 2：输入：arr = [7,7,7,7,7,7] 输出：1
解释：我们只能选择集合 {7}，结果数组为空。
示例 3：输入：arr = [1,9]  输出：1
示例 4：输入：arr = [1000,1000,3,7] 输出：1
示例 5：输入：arr = [1,2,3,4,5,6,7,8,9,10] 输出：5
提示：1 <= arr.length <= 10^5
arr.length为偶数
1 <= arr[i] <= 10^5
```

- 解题思路

```
func minSetSize(arr []int) int {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]]++
        }
        temp := make([][2]int, 0)
        for k, v := range m {
                temp = append(temp, [2]int{k, v})
        }
        sort.Slice(temp, func(i, j int) bool {
                return temp[i][1] > temp[j][1]
        })
```

(续下页)

(接上页)

```
        res := 0
        total := 0
        for i := 0; i < len(temp); i++ {
                total = total + temp[i][1]
                res++
                if total*2 >= len(arr) {
                        return res
                }
        }
        return res
}

# 2
func minSetSize(arr []int) int {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]]++
        }
        temp := make([]int, 0)
        for _, v := range m {
                temp = append(temp, v)
        }
        sort.Ints(temp)
        res := 0
        total := 0
        for i := len(temp) - 1; i >= 0; i-- {
                total = total + temp[i]
                res++
                if total*2 >= len(arr) {
                        return res
                }
        }
        return res
}
```

41.17 1339. 分裂二叉树的最大乘积 (2)

- 题目

```
给你一棵二叉树，它的根为root 。请你删除 1
↪条边，使二叉树分裂成两棵子树，且它们子树和的乘积尽可能大。
由于答案可能会很大，请你将结果对 10^9 + 7 取模后再返回。
示例 1：输入：root = [1,2,3,4,5,6] 输出：110
解释：删除红色的边，得到 2 棵子树，和分别为 11 和 10 。它们的乘积是 110 （11*10）
示例 2：输入：root = [1,null,2,3,4,null,null,5,6] 输出：90
解释：移除红色的边，得到 2 棵子树，和分别是 15 和 6 。它们的乘积为 90 （15*6）
示例 3：输入：root = [2,3,9,10,7,8,6,5,4,11,1] 输出：1025
示例 4：输入：root = [1,1] 输出：1
提示：每棵树最多有50000个节点，且至少有2个节点。
每个节点的值在[1, 10000]之间。
```

- 解题思路

```
var sum int
var res int

func maxProduct(root *TreeNode) int {
        sum = 0
        res = 0
        dfsSum(root)
        dfs(root)
        return res % 1000000007
}

func dfsSum(root *TreeNode) {
        if root == nil {
                return
        }
        sum = sum + root.Val
        dfsSum(root.Left)
        dfsSum(root.Right)
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
```

(续下页)

(接上页)

```
        res = max(res, left*(sum-left))
        res = max(res, right*(sum-right))
        return left + right + root.Val
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var sum int
var target int

func maxProduct(root *TreeNode) int {
        sum = 0
        target = 0
        dfsSum(root)
        dfs(root)
        return target * (sum - target) % 1000000007
}

func dfsSum(root *TreeNode) {
        if root == nil {
                return
        }
        sum = sum + root.Val
        dfsSum(root.Left)
        dfsSum(root.Right)
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        total := left + right + root.Val
        if abs(sum-2*total) < abs(sum-2*target) {
                target = total
        }
```

(续下页)

(接上页)

```
        return total
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

41.18 1343. 大小为 K 且平均值大于等于阈值的子数组数目 (2)

- 题目

```
给你一个整数数组arr和两个整数 k和 threshold。
请你返回长度为 k且平均值大于等于threshold的子数组数目。
示例 1：输入：arr = [2,2,2,2,5,5,5,8], k = 3, threshold = 4 输出：3
解释：子数组 [2,5,5],[5,5,5] 和 [5,5,8] 的平均值分别为 4，5 和 6 。
其他长度为 3 的子数组的平均值都小于 4 （threshold 的值）。
示例 2：输入：arr = [1,1,1,1,1], k = 1, threshold = 0 输出：5
示例 3：输入：arr = [11,13,17,23,29,31,7,5,2,3], k = 3, threshold = 5 输出：6
解释：前 6 个长度为 3 的子数组平均值都大于 5 。注意平均值不是整数。
示例 4：输入：arr = [7,7,7,7,7,7,7], k = 7, threshold = 7 输出：1
示例 5：输入：arr = [4,4,4,4], k = 4, threshold = 1 输出：1
提示：1 <= arr.length <= 10^5
1 <= arr[i] <= 10^4
1 <= k <= arr.length
0 <= threshold <= 10^4
```

- 解题思路

```
func numOfSubarrays(arr []int, k int, threshold int) int {
        n := len(arr)
        temp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                temp[i] = temp[i-1] + arr[i-1]
        }
        res := 0
        target := k * threshold
        for i := k; i <= n; i++ {
                if temp[i]-temp[i-k] >= target {
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 2
func numOfSubarrays(arr []int, k int, threshold int) int {
        n := len(arr)
        sum := 0
        for i := 0; i < k; i++ {
                sum = sum + arr[i]
        }
        res := 0
        target := k * threshold
        if sum >= target {
                res++
        }
        for i := k; i < n; i++ {
                sum = sum + arr[i] - arr[i-k]
                if sum >= target {
                        res++
                }
        }
        return res
}
```

41.19 1344. 时钟指针的夹角 (1)

- 题目

```
给你两个数hour和minutes。请你返回在时钟上，由给定时间的时针和分针组成的较小角的角度（60
↪单位制）。
示例 1：输入：hour = 12, minutes = 30 输出：165
示例 2：输入：hour = 3, minutes = 30 输出；75
示例 3：输入：hour = 3, minutes = 15 输出：7.5
示例 4：输入：hour = 4, minutes = 50 输出：155
示例 5：输入：hour = 12, minutes = 0 输出：0
提示：1 <= hour <= 12
0 <= minutes <= 59
与标准答案误差在10^-5以内的结果都被视为正确结果。
```

- 解题思路

```
func angleClock(hour int, minutes int) float64 {
        m := float64(minutes) * 6
        h := float64(hour)*30 + float64(minutes)*0.5
        res := math.Abs(m - h)
        if res > 180 {
                return 360 - res
        }
        return res
}
```

41.20 1347. 制造字母异位词的最小步骤数 (3)

- 题目

```
给你两个长度相等的字符串s 和 t。每一个步骤中，你可以选择将t中的 任一字符 替换为另一个字符。
返回使t成为s的字母异位词的最小步骤数。
字母异位词 指字母相同，但排列不同（也可能相同）的字符串。
示例 1：输出：s = "bab", t = "aba" 输出：1
提示：用 'b' 替换 t 中的第一个 'a'，t = "bba" 是 s 的一个字母异位词。
示例 2：输出：s = "leetcode", t = "practice" 输出：5
提示：用合适的字符替换 t 中的 'p', 'r', 'a', 'i' 和 'c'，使 t 变成 s 的字母异位词。
示例 3：输出：s = "anagram", t = "mangaar" 输出：0
提示："anagram" 和 "mangaar" 本身就是一组字母异位词。
示例 4：输出：s = "xxyyzz", t = "xxyyzz" 输出：0
示例 5：输出：s = "friend", t = "family" 输出：4
提示：1 <= s.length <= 50000
s.length == t.length
s 和 t只包含小写英文字母
```

- 解题思路

```
func minSteps(s string, t string) int {
        res := 0
        m := make(map[uint8]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
        }
        for i := 0; i < len(t); i++ {
                if _, ok := m[t[i]]; ok {
                        m[t[i]]--
                } else {
                        m[t[i]] = -1
```

(续下页)

(接上页)

```
                }
        }
        for _, v := range m {
                if v > 0 {
                        res = res + v
                }
        }
        return res
}

# 2
func minSteps(s string, t string) int {
        res := 0
        m := make(map[uint8]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
                m[t[i]]--
        }
        for _, v := range m {
                if v > 0 {
                        res = res + v
                }
        }
        return res
}

# 3
func minSteps(s string, t string) int {
        res := 0
        a := make(map[int]int)
        b := make(map[int]int)
        for i := 0; i < len(s); i++ {
                a[int(s[i]-'a')]++
                b[int(t[i]-'a')]++
        }
        for i := 0; i < 26; i++ {
                if a[i] < b[i] {
                        res = res + b[i] - a[i]
                }
        }
        return res
}
```

41.21 1348. 推文计数 (1)

- 题目

```
请你实现一个能够支持以下两种方法的推文计数类TweetCounts:
1. recordTweet(string tweetName, int time)
记录推文发布情况: 用户tweetName在time (以 秒为单位) 时刻发布了一条推文。
2. getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int␣
↪endTime)
返回从开始时间startTime (以 秒 为单位) 到结束时间endTime (以 秒 为单位) 内,
每 分minute, 时hour 或者 日day (取决于freq) 内指定用户tweetName发布的推文总数。
freq的值始终为 分minute, 时 hour或者日␣
↪day之一, 表示获取指定用户tweetName发布推文次数的时间间隔。
第一个时间间隔始终从 startTime 开始, 因此时间间隔为
[startTime, startTime + delta*1>, [startTime + delta*1, startTime + delta*2>,
[startTime + delta*2, startTime + delta*3>, ... ,
[startTime + delta*i,min(startTime + delta*(i+1), endTime + 1)>,
其中 i 和 delta (取决于 freq) 都是非负整数。
示例: 输入: ["TweetCounts","recordTweet","recordTweet","recordTweet",
"getTweetCountsPerFrequency","getTweetCountsPerFrequency","recordTweet",
"getTweetCountsPerFrequency"]
[[],["tweet3",0],["tweet3",60],["tweet3",10],["minute","tweet3",0,59],
["minute","tweet3",0,60],["tweet3",120],["hour","tweet3",0,210]]
输出: [null,null,null,null,[2],[2,1],null,[4]]
解释:
TweetCounts tweetCounts = new TweetCounts();
tweetCounts.recordTweet("tweet3", 0);
tweetCounts.recordTweet("tweet3", 60);
tweetCounts.recordTweet("tweet3", 10);
//"tweet3"发布推文的时间分别是0,10和60 。
tweetCounts.getTweetCountsPerFrequency("minute", "tweet3", 0, 59);
//返回[2]。统计频率是每分钟 (60 秒) , 因此只有一个有效时间间隔 [0,60>->2条推文。
tweetCounts.getTweetCountsPerFrequency("minute", "tweet3", 0, 60);
//返回[2,1]。统计频率是每分钟 (60 秒) , 因此有两个有效时间间隔1)[0,60>->
↪2条推文, 和2)[60,61>->1条推文。
tweetCounts.recordTweet("tweet3", 120);
// "tweet3"发布推文的时间分别是 0, 10, 60 和 120 。
tweetCounts.getTweetCountsPerFrequency("hour", "tweet3", 0, 210);
//返回[4]。统计频率是每小时 (3600 秒) , 因此只有一个有效时间间隔 [0,211>->4条推文。
提示: 同时考虑recordTweet和getTweetCountsPerFrequency, 最多有 10000 次操作。
0 <= time, startTime, endTime <=10^9
0 <= endTime - startTime <= 10^4
```

- 解题思路

```
type TweetCounts struct {
        m map[string][]int
}

func Constructor() TweetCounts {
        return TweetCounts{m: make(map[string][]int)}
}

func (this *TweetCounts) RecordTweet(tweetName string, time int) {
        this.m[tweetName] = append(this.m[tweetName], time)
}

func (this *TweetCounts) GetTweetCountsPerFrequency(freq string, tweetName string,␣
↪startTime int, endTime int) []int {
        per := 0
        if freq == "minute" {
                per = 60
        } else if freq == "hour" {
                per = 60 * 60
        } else if freq == "day" {
                per = 60 * 60 * 24
        }
        n := (endTime-startTime)/per + 1
        res := make([]int, n)
        for i := 0; i < len(this.m[tweetName]); i++ {
                t := this.m[tweetName][i]
                if startTime <= t && t <= endTime {
                        index := (t - startTime) / per
                        res[index]++
                }
        }
        return res
}
```

41.22 1352. 最后 K 个数的乘积 (2)

- 题目

```
请你实现一个「数字乘积类」ProductOfNumbers，要求支持下述两种方法：
1.add(int num)
将数字num添加到当前数字列表的最后面。
2. getProduct(int k)
```

(续下页)

(接上页)

```
返回当前数字列表中，最后k个数字的乘积。
你可以假设当前列表中始终 至少 包含 k 个数字。
题目数据保证：任何时候，任一连续数字序列的乘积都在 32-bit 整数范围内，不会溢出。
示例：输入：
["ProductOfNumbers","add","add","add","add","add",
"getProduct","getProduct","getProduct","add","getProduct"]
[[],[3],[0],[2],[5],[4],[2],[3],[4],[8],[2]]
输出：  [null,null,null,null,null,null,20,40,0,null,32]
解释：ProductOfNumbers productOfNumbers = new ProductOfNumbers();
productOfNumbers.add(3);        // [3]
productOfNumbers.add(0);        // [3,0]
productOfNumbers.add(2);        // [3,0,2]
productOfNumbers.add(5);        // [3,0,2,5]
productOfNumbers.add(4);        // [3,0,2,5,4]
productOfNumbers.getProduct(2); // 返回 20 。最后 2 个数字的乘积是 5 * 4 = 20
productOfNumbers.getProduct(3); // 返回 40 。最后 3 个数字的乘积是 2 * 5 * 4 = 40
productOfNumbers.getProduct(4); // 返回  0 。最后 4 个数字的乘积是 0 * 2 * 5 * 4 = 0
productOfNumbers.add(8);        // [3,0,2,5,4,8]
productOfNumbers.getProduct(2); // 返回 32 。最后 2 个数字的乘积是 4 * 8 = 32
提示：add 和 getProduct两种操作加起来总共不会超过40000次。
0 <= num<=100
1 <= k <= 40000
```

- 解题思路

```
type ProductOfNumbers struct {
        arr []int
}

func Constructor() ProductOfNumbers {
        return ProductOfNumbers{arr: make([]int, 0)}
}

func (this *ProductOfNumbers) Add(num int) {
        this.arr = append(this.arr, num)
}

func (this *ProductOfNumbers) GetProduct(k int) int {
        res := 1
        n := len(this.arr)
        for i := n - 1; i >= n-k && i >= 0; i-- {
                res = res * this.arr[i]
        }
        return res
```

(续下页)

(接上页)

```
}

# 2
type ProductOfNumbers struct {
        arr []int
}

func Constructor() ProductOfNumbers {
        return ProductOfNumbers{arr: []int{1}}
}

func (this *ProductOfNumbers) Add(num int) {
        if num == 0 {
                this.arr = []int{1}
                return
        }
        num = num * this.arr[len(this.arr)-1]
        this.arr = append(this.arr, num)
}

func (this *ProductOfNumbers) GetProduct(k int) int {
        if k > len(this.arr)-1 {
                return 0
        }
        return this.arr[len(this.arr)-1] / this.arr[len(this.arr)-1-k]
}
```

41.23 1353. 最多可以参加的会议数目

41.23.1 题目

```
给你一个数组events，其中events[i] = [startDayi, endDayi]，
表示会议i开始于startDayi，结束于endDayi。
你可以在满足startDayi<= d <=␣
↪endDayi中的任意一天d参加会议i。注意，一天只能参加一个会议。
请你返回你可以参加的最大会议数目。
示例 1：输入：events = [[1,2],[2,3],[3,4]] 输出：3
解释：你可以参加所有的三个会议。
安排会议的一种方案如上图。
第 1 天参加第一个会议。
第 2 天参加第二个会议。
```

(续下页)

(接上页)

```
第 3 天参加第三个会议。
示例 2: 输入: events= [[1,2],[2,3],[3,4],[1,2]] 输出: 4
示例 3: 输入: events = [[1,4],[4,4],[2,2],[3,4],[1,1]] 输出: 4
示例 4: 输入: events = [[1,100000]] 输出: 1
示例 5: 输入: events = [[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[1,7]] 输出: 7
提示: 1 <= events.length <= 10^5
events[i].length == 2
1 <= events[i][0] <= events[i][1] <= 10^5
```

41.23.2 解题思路

```

```

41.24 1357. 每隔 n 个顾客打折 (1)

- 题目

```
超市里正在举行打折活动，每隔n个顾客会得到 discount的折扣。
超市里有一些商品，第i种商品为products[i]且每件单品的价格为prices[i]。
结账系统会统计顾客的数目，每隔n个顾客结账时，该顾客的账单都会打折，
折扣为discount（也就是如果原本账单为x，
那么实际金额会变成x - (discount * x) / 100），然后系统会重新开始计数。
顾客会购买一些商品，product[i]是顾客购买的第i种商品，amount[i]是对应的购买该种商品的数目。
请你实现Cashier类：
Cashier(int n, int discount, int[] products, int[] prices)初始化实例对象，
参数分别为打折频率n，折扣大小 discount，超市里的商品列表 products和它们的价格 prices。
doublegetBill(int[] product, int[]␣
↪amount)返回账单的实际金额（如果有打折，请返回打折后的结果）。
返回结果与标准答案误差在10^-5以内都视为正确结果。
示例 1: 输入
["Cashier","getBill","getBill","getBill","getBill","getBill","getBill","getBill"]
[[3,50,[1,2,3,4,5,6,7],[100,200,300,400,300,200,100]],[[1,2],[1,2]],[[3,7],[10,10]],
[[1,2,3,4,5,6,7],[1,1,1,1,1,1,1]],[[4],[10]],[[7,3],[10,10]],[[7,5,3,1,6,4,2],
[10,10,10,9,9,9,7]],[[2,3,5],[5,3,2]]]
输出 [null,500.0,4000.0,800.0,4000.0,4000.0,7350.0,2500.0]
解释Cashier cashier = new Cashier(3,50,[1,2,3,4,5,6,7],[100,200,300,400,300,200,100]);
cashier.getBill([1,2],[1,2]);                        // 返回 500.0, 账单金额为 = 1 *␣
↪100 + 2 * 200 = 500.
cashier.getBill([3,7],[10,10]);                      // 返回 4000.0
cashier.getBill([1,2,3,4,5,6,7],[1,1,1,1,1,1,1]);    // 返回 800.0 , 账单原本为 1600.
```

(续下页)

(接上页)

```
↪0 ，但由于该顾客是第三位顾客，他将得到 50% 的折扣，所以实际金额为 1600 - 1600 * (50␣
↪/ 100) = 800 。
cashier.getBill([4],[10]);                              // 返回 4000.0
cashier.getBill([7,3],[10,10]);                         // 返回 4000.0
cashier.getBill([7,5,3,1,6,4,2],[10,10,10,9,9,9,7]); // 返回 7350.0 ，账单原本为␣
↪14700.0 ，但由于系统计数再次达到三，该顾客将得到 50% 的折扣，实际金额为 7350.0 。
cashier.getBill([2,3,5],[5,3,2]);                       // 返回 2500.0
提示：1 <= n <= 10^4
0 <= discount <= 100
1 <= products.length <= 200
1 <= products[i] <= 200
products列表中不会有重复的元素。
prices.length == products.length
1 <= prices[i] <= 1000
1 <= product.length <= products.length
product[i]在products出现过。
amount.length == product.length
1 <= amount[i] <= 1000
最多有1000 次对getBill函数的调用。
返回结果与标准答案误差在10^-5以内都视为正确结果。
```

- 解题思路

```
type Cashier struct {
        n        int
        discount int
        count    int
        m        map[int]int
}

func Constructor(n int, discount int, products []int, prices []int) Cashier {
        res := Cashier{
                n:        n,
                discount: discount,
                count:    0,
                m:        make(map[int]int),
        }
        for i := 0; i < len(products); i++ {
                res.m[products[i]] = prices[i]
        }
        return res
}

func (this *Cashier) GetBill(product []int, amount []int) float64 {
```

(续下页)

(接上页)

```
        var res int
        for i := 0; i < len(product); i++ {
                res = res + amount[i]*this.m[product[i]]
        }
        this.count++
        if this.count%this.n == 0 {
                this.count = 0
                return float64(res*(100-this.discount)) / 100.0
        }
        return float64(res)
}
```

41.25 1358. 包含所有三种字符的子字符串数目 (4)

- 题目

```
给你一个字符串 s，它只包含三种字符 a，b 和 c 。
请你返回 a，b 和 c 都至少出现过一次的子字符串数目。
示例 1：输入：s = "abcabc" 输出：10
解释：包含 a，b 和 c 各至少一次的子字符串为
"abc", "abca", "abcab", "abcabc", "bca", "bcab", "bcabc", "cab", "cabc" 和 "abc"␣
↪(相同字符串算多次)。
示例 2：输入：s = "aaacb" 输出：3
解释：包含 a，b 和 c 各至少一次的子字符串为 "aaacb", "aacb" 和 "acb" 。
示例 3：输入：s = "abc" 输出：1
提示：3 <= s.length <= 5 x 10^4
s只包含字符 a，b 和 c 。
```

- 解题思路

```
func numberOfSubstrings(s string) int {
        res := 0
        n := len(s)
        arr := make([][3]int, n+1)
        for i := 0; i < n; i++ {
                for j := 0; j < 3; j++ {
                        arr[i+1][j] = arr[i][j]
                }
                value := int(s[i] - 'a')
                arr[i+1][value]++
        }
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                left := i + 1
                right := n
                index := -1
                for left <= right {
                        mid := left + (right-left)/2
                        if arr[mid][0] > arr[i][0] &&
                                arr[mid][1] > arr[i][1] &&
                                arr[mid][2] > arr[i][2] {
                                right = mid - 1
                                index = mid
                        } else {
                                left = mid + 1
                        }
                }
                if index != -1 {
                        res = res + n - index + 1
                }
        }
        return res
}

# 2
func numberOfSubstrings(s string) int {
        res := 0
        n := len(s)
        arr := [3]int{}
        left := 0
        right := -1
        var value int
        for left = 0; left < n; left++ {
                for right < n && (arr[0] == 0 || arr[1] == 0 || arr[2] == 0) {
                        right++
                        if right == n {
                                break
                        }
                        value = int(s[right] - 'a')
                        arr[value]++
                }
                res = res + n - right
                value = int(s[left] - 'a')
                arr[value]--
        }
        return res
```

(续下页)

(接上页)

```
}

# 3
func numberOfSubstrings(s string) int {
        res := 0
        n := len(s)
        arr := [3]int{}
        var value int
        i := 0
        for j := 0; j < n; j++ {
                value = int(s[j] - 'a')
                arr[value]++
                for arr[0] > 0 && arr[1] > 0 && arr[2] > 0 {
                        res = res + n - j
                        value = int(s[i] - 'a')
                        arr[value]--
                        i++
                }
        }
        return res
}

# 4
func numberOfSubstrings(s string) int {
        res := 0
        n := len(s)
        m := make(map[int]int)
        count := 0
        var value int
        i := 0
        for j := 0; j < n; j++ {
                value = int(s[j] - 'a')
                if m[value] == 0 {
                        count++
                }
                m[value]++
                for count == 3 {
                        res = res + n - j
                        value = int(s[i] - 'a')
                        m[value]--
                        i++
                        if m[value] == 0 {
                                count--
```

(续下页)

(接上页)

```
                        }
                }
        }
        return res
}
```

41.26 1361. 验证二叉树 (2)

- 题目

```
二叉树上有 n个节点，按从0到 n -␣
↪1编号，其中节点i的两个子节点分别是leftChild[i]和rightChild[i]。
只有 所有 节点能够形成且 只 形成 一颗有效的二叉树时，返回true；否则返回 false。
如果节点i没有左子节点，那么leftChild[i]就等于-1。右子节点也符合该规则。
注意：节点没有值，本问题中仅仅使用节点编号。
示例 1：输入：n = 4, leftChild = [1,-1,3,-1], rightChild = [2,-1,-1,-1] 输出：true
示例 2：输入：n = 4, leftChild = [1,-1,3,-1], rightChild = [2,3,-1,-1] 输出：false
示例 3：输入：n = 2, leftChild = [1,0], rightChild = [-1,-1] 输出：false
示例 4：输入：n = 6, leftChild = [1,-1,-1,4,-1,-1], rightChild = [2,-1,-1,5,-1,-1]␣
↪输出：false
提示：1 <= n <= 10^4
leftChild.length == rightChild.length == n
-1 <= leftChild[i], rightChild[i] <= n - 1
```

- 解题思路

```
func validateBinaryTreeNodes(n int, leftChild []int, rightChild []int) bool {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                if leftChild[i] != -1 {
                        arr[leftChild[i]]++
                }
                if rightChild[i] != -1 {
                        arr[rightChild[i]]++
                }
        }
        root := -1 // 寻找一个根节点
        for i := 0; i < n; i++ {
                if arr[i] == 0 {
                        root = i
                        break
                }
```

(续下页)

(接上页)

```
        }
        if root == -1 {
                return false
        }
        visited := make(map[int]bool)
        visited[root] = true
        queue := make([]int, 0)
        queue = append(queue, root)
        for len(queue) > 0 { // 层序遍历
                node := queue[0]
                queue = queue[1:]
                a, b := leftChild[node], rightChild[node]
                if a != -1 {
                        if visited[a] == true {
                                return false
                        }
                        visited[a] = true
                        queue = append(queue, a)
                }
                if b != -1 {
                        if visited[b] == true {
                                return false
                        }
                        visited[b] = true
                        queue = append(queue, b)
                }
        }
        return len(visited) == n
}

# 2
func validateBinaryTreeNodes(n int, leftChild []int, rightChild []int) bool {
        fa = Init(n)
        for i := 0; i < n; i++ {
                a, b := leftChild[i], rightChild[i]
                if a != -1 {
                        if find(a) != a || query(i, a) == true {
                                return false
                        }
                        union(a, i) // 注意顺序
                }
                if b != -1 {
                        if find(b) != b || query(i, b) == true {
```

(续下页)

(接上页)

```
                                return false
                        }
                        union(b, i) // 注意顺序
                }
        }
        return getCount() == 1
}

var fa []int
var count int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        count = n
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
        }
}

func query(i, j int) bool {
        return find(i) == find(j)
}

func getCount() int {
```

(续下页)

(接上页)

```
        return count
}
```

41.27 1362. 最接近的因数 (2)

- 题目

```
给你一个整数num，请你找出同时满足下面全部要求的两个整数：
两数乘积等于 num + 1或num + 2
以绝对差进行度量，两数大小最接近
你可以按任意顺序返回这两个整数。
示例 1：输入：num = 8 输出：[3,3]
解释：对于 num + 1 = 9，最接近的两个因数是 3 & 3；对于 num + 2 = 10，
最接近的两个因数是 2 & 5，因此返回 3 & 3 。
示例 2：输入：num = 123 输出：[5,25]
示例 3：输入：num = 999 输出：[40,25]
提示：1 <= num <= 10^9
```

- 解题思路

```
func closestDivisors(num int) []int {
        res := []int{1, num + 1}
        for i := num + 1; i <= num+2; i++ {
                temp := divide(i)
                if abs(temp[0]-temp[1]) < abs(res[0]-res[1]) {
                        res = temp
                }
        }
        return res
}

func divide(n int) []int {
        for i := int(math.Sqrt(float64(n))); i >= 0; i-- {
                if n%i == 0 {
                        return []int{i, n / i}
                }
        }
        return nil
}

func abs(a int) int {
        if a < 0 {
```

(续下页)

(接上页)

```
                return -a
        }
        return a
}


# 2
func closestDivisors(num int) []int {
        res := []int{1, num + 1}
        for i := int(math.Sqrt(float64(num + 2))); i >= 0; i-- {
                if (num+1)%i == 0 {
                        return []int{i, (num + 1) / i}
                }
                if (num+2)%i == 0 {
                        return []int{i, (num + 2) / i}
                }
        }
        return res
}
```

41.28 1366. 通过投票对团队排名 (1)

- 题目

```
现在有一个特殊的排名系统，依据参赛团队在投票人心中的次序进行排名，
每个投票者都需要按从高到低的顺序对参与排名的所有团队进行排位。
排名规则如下：
    参赛团队的排名次序依照其所获「排位第一」的票的多少决定。
    ␣
↪如果存在多个团队并列的情况，将继续考虑其「排位第二」的票的数量。以此类推，直到不再存在并列的情况。
    如果在考虑完所有投票情况后仍然出现并列现象，则根据团队字母的字母顺序进行排名。
给你一个字符串数组 votes␣
↪代表全体投票者给出的排位情况，请你根据上述排名规则对所有参赛团队进行排名。
请你返回能表示按排名系统 排序后 的所有团队排名的字符串。
示例 1：输入：votes = ["ABC","ACB","ABC","ACB","ACB"] 输出："ACB"
解释：A 队获得五票「排位第一」，没有其他队获得「排位第一」，所以 A 队排名第一。
B 队获得两票「排位第二」，三票「排位第三」。
C 队获得三票「排位第二」，两票「排位第三」。
由于 C 队「排位第二」的票数较多，所以 C 队排第二，B 队排第三。
示例 2：输入：votes = ["WXYZ","XYZW"] 输出："XWYZ"
解释：X 队在并列僵局打破后成为排名第一的团队。
X 队和 W 队的「排位第一」票数一样，但是 X 队有一票「排位第二」，而 W␣
```

(续下页)

(接上页)

```
↪没有获得「排位第二」。
示例 3: 输入: votes = ["ZMNAGUEDSJYLBOPHRQICWFXTVK"]
输出: "ZMNAGUEDSJYLBOPHRQICWFXTVK"
解释: 只有一个投票者，所以排名完全按照他的意愿。
示例 4: 输入: votes = ["BCA","CAB","CBA","ABC","ACB","BAC"] 输出: "ABC"
解释: A 队获得两票「排位第一」，两票「排位第二」，两票「排位第三」。
B 队获得两票「排位第一」，两票「排位第二」，两票「排位第三」。
C 队获得两票「排位第一」，两票「排位第二」，两票「排位第三」。
完全并列，所以我们需要按照字母升序排名。
示例 5: 输入: votes = ["M","M","M","M"] 输出: "M"
解释: 只有 M 队参赛，所以它排名第一。
提示: 1 <= votes.length <= 1000
    1 <= votes[i].length <= 26
    votes[i].length == votes[j].length for 0 <= i, j < votes.length
    votes[i][j] 是英文 大写 字母
    votes[i] 中的所有字母都是唯一的
    votes[0] 中出现的所有字母 同样也 出现在 votes[j] 中，其中 1 <= j < votes.length
```

- 解题思路

```
type Node struct {
        rank     []int
        teamName byte
}

func rankTeams(votes []string) string {
        m := make(map[byte][]int, 0)
        for i := 0; i < len(votes[0]); i++ {
                team := votes[0][i]
                m[team] = make([]int, len(votes[0]))
        }
        for _, vote := range votes {
                for i := 0; i < len(vote); i++ {
                        m[vote[i]][i]++
                }
        }
        arr := make([]Node, 0)
        for k, v := range m {
                arr = append(arr, Node{
                        rank:     v,
                        teamName: k,
                })
        }
        sort.Slice(arr, func(i, j int) bool {
```

(续下页)

(接上页)

```
                for k := 0; k < len(arr[i].rank); k++ {
                        if arr[i].rank[k] != arr[j].rank[k] {
                                return arr[i].rank[k] > arr[j].rank[k]
                        }
                }
                return arr[i].teamName < arr[j].teamName
        })
        res := ""
        for i := 0; i < len(arr); i++ {
                res = res + string(arr[i].teamName)
        }
        return res
}
```

41.29 1367. 二叉树中的列表 (2)

- 题目

```
给你一棵以 root 为根的二叉树和一个 head 为第一个节点的链表。
如果在二叉树中，存在一条一直向下的路径，且每个点的数值恰好一一对应以 head␣
↪为首的链表中每个节点的值，
那么请你返回 True ，否则返回 False 。
一直向下的路径的意思是：从树中某个节点开始，一直连续向下的路径。
示例 1：
输入：head = [4,2,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,3]
输出：true
解释：树中蓝色的节点构成了与链表对应的子路径。
示例 2：
输入：head = [1,4,2,6], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,
↪3]
输出：true
示例 3：
输入：head = [1,4,2,6,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,
↪1,3]
输出：false
解释：二叉树中不存在一一对应链表的路径。
提示：
    二叉树和链表中的每个节点的值都满足 1 <= node.val <= 100 。
    链表包含的节点数目在 1 到 100 之间。
    二叉树包含的节点数目在 1 到 2500 之间。
```

- 解题思路

```
func isSubPath(head *ListNode, root *TreeNode) bool {
        if root == nil {
                return false
        }
        return dfs(head, root) || isSubPath(head, root.Left) || isSubPath(head, root.
↪Right)
}

func dfs(head *ListNode, root *TreeNode) bool {
        if head == nil {
                return true
        }
        if root == nil || root.Val != head.Val {
                return false
        }
        return dfs(head.Next, root.Left) || dfs(head.Next, root.Right)
}

# 2
func isSubPath(head *ListNode, root *TreeNode) bool {
        if root == nil {
                return false
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.Val == head.Val {
                        if dfs(head, node) == true {
                                return true
                        }
                }
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return false
}

func dfs(head *ListNode, root *TreeNode) bool {
```

(续下页)

(接上页)

```
        if head == nil {
                return true
        }
        if root == nil || root.Val != head.Val {
                return false
        }
        return dfs(head.Next, root.Left) || dfs(head.Next, root.Right)
}
```

41.30 1371. 每个元音包含偶数次的最长子字符串 (2)

- 题目

```
给你一个字符串s，请你返回满足以下条件的最长子字符串的长度：
每个元音字母，即'a'，'e'，'i'，'o'，'u' ，在子字符串中都恰好出现了偶数次。
示例 1：输入：s = "eleetminicoworoep" 输出：13
解释：最长子字符串是 "leetminicowor" ，它包含 e，i，o各 2 个，以及 0 个 a，u 。
示例 2：输入：s = "leetcodeisgreat" 输出：5
解释：最长子字符串是 "leetc" ，其中包含 2 个 e 。
示例 3：  输入：s = "bcbcbc" 输出：6
解释：这个示例中，字符串 "bcbcbc" 本身就是最长的，因为所有的元音 a，e，i，o，u
↪都出现了 0 次。
提示：1 <= s.length <= 5 x 10^5
s只包含小写英文字母。
```

- 解题思路

```
func findTheLongestSubstring(s string) int {
        res := 0
        m := make(map[int]int)
        m[0] = -1
        status := 0
        for i := 0; i < len(s); i++ {
                switch s[i] {
                case 'a':
                        status = status ^ 0b00001
                case 'e':
                        status = status ^ 0b00010
                case 'i':
                        status = status ^ 0b00100
                case 'o':
                        status = status ^ 0b01000
```

(续下页)

(接上页)

```
                case 'u':
                        status = status ^ 0b10000
                }
                if v, ok := m[status]; ok {
                        res = max(res, i-v)
                } else {
                        m[status] = i // 记录第一次出现的位置
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func findTheLongestSubstring(s string) int {
        res := 0
        m := make(map[[5]int]int)
        status := [5]int{}
        m[status] = -1
        for i := 0; i < len(s); i++ {
                switch s[i] {
                case 'a':
                        status[0] = 1 - status[0]
                case 'e':
                        status[1] = 1 - status[1]
                case 'i':
                        status[2] = 1 - status[2]
                case 'o':
                        status[3] = 1 - status[3]
                case 'u':
                        status[4] = 1 - status[4]
                }
                if v, ok := m[status]; ok {
                        res = max(res, i-v)
                } else {
                        m[status] = i // 记录第一次出现的位置
                }
```

(续下页)

(接上页)

```
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

41.31 1372. 二叉树中的最长交错路径 (1)

- 题目

```
给你一棵以root为根的二叉树，二叉树中的交错路径定义如下：
选择二叉树中 任意节点和一个方向（左或者右）。
如果前进方向为右，那么移动到当前节点的的右子节点，否则移动到它的左子节点。
改变前进方向：左变右或者右变左。
重复第二步和第三步，直到你在树中无法继续移动。
交错路径的长度定义为：访问过的节点数目 - 1（单个节点的路径长度为 0 ）。
请你返回给定树中最长 交错路径的长度。
示例 1：输入：root = [1,null,1,1,1,null,null,1,1,null,1,null,null,null,1,null,1]␣
↪输出：3
解释：蓝色节点为树中最长交错路径（右 -> 左 -> 右）。
示例 2：输入：root = [1,1,1,null,1,null,null,1,1,null,1] 输出：4
解释：蓝色节点为树中最长交错路径（左 -> 右 -> 左 -> 右）。
示例 3：输入：root = [1] 输出：0
提示：每棵树最多有50000个节点。
每个节点的值在[1, 100] 之间。
```

- 解题思路

```
var res int

func longestZigZag(root *TreeNode) int {
        res = 0
        dfs(root, 0, 0)
        return res
}

func dfs(root *TreeNode, left, right int) {
```

(续下页)

(接上页)

```
        if root != nil {
                res = max(res, left)
                res = max(res, right)
                dfs(root.Left, right+1, 0)
                dfs(root.Right, 0, left+1)
        }
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

41.32 1375. 灯泡开关 III(2)

- 题目

```
房间中有 n 枚灯泡，编号从 1 到 n，自左向右排成一排。最初，所有的灯都是关着的。
在 k 时刻（k 的取值范围是 0 到 n - 1），我们打开 light[k] 这个灯。
灯的颜色要想 变成蓝色 就必须同时满足下面两个条件：
    灯处于打开状态。
    排在它之前（左侧）的所有灯也都处于打开状态。
请返回能够让 所有开着的 灯都 变成蓝色 的时刻 数目 。
示例 1：输入：light = [2,1,3,5,4] 输出：3
解释：所有开着的灯都变蓝的时刻分别是 1，2 和 4 。
示例 2：输入：light = [3,2,4,1,5] 输出：2
解释：所有开着的灯都变蓝的时刻分别是 3 和 4（index-0）。
示例 3：输入：light = [4,1,2,3] 输出：1
解释：所有开着的灯都变蓝的时刻是 3（index-0）。
第 4 个灯在时刻 3 变蓝。
示例 4：输入：light = [2,1,4,3,6,5] 输出：3
示例 5：输入：light = [1,2,3,4,5,6] 输出：6
提示：
    n == light.length
    1 <= n <= 5 * 10^4
    light 是 [1, 2, ..., n] 的一个排列。
```

- 解题思路

```
func numTimesAllBlue(light []int) int {
        res := 0
        sum := 0
        for i := 0; i < len(light); i++ {
                sum = sum + light[i]
                if (i+1)*(i+2)/2 == sum {
                        res++
                }
        }
        return res
}

# 2
func numTimesAllBlue(light []int) int {
        res := 0
        maxValue := 0
        for i := 0; i < len(light); i++ {
                // 最大亮起来的灯等于前面灯的数量，那么说明前面灯都亮了
                maxValue = max(maxValue, light[i])
                if maxValue == i+1 {
                        res++
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

41.33 1376. 通知所有员工所需的时间 (3)

- 题目

```
公司里有 n 名员工，每个员工的 ID 都是独一无二的，编号从 0 到 n - 1。
公司的总负责人通过 headID 进行标识。
在 manager 数组中，每个员工都有一个直属负责人，其中 manager[i] 是第 i␣
↪名员工的直属负责人。
对于总负责人，manager[headID] = -1。题目保证从属关系可以用树结构显示。
```

(续下页)

(接上页)

```
公司总负责人想要向公司所有员工通告一条紧急消息。
他将会首先通知他的直属下属们，然后由这些下属通知他们的下属，直到所有的员工都得知这条紧急消息。
第 i 名员工需要 informTime[i] 分钟来通知它的所有直属下属（也就是说在 informTime[i]␣
↪分钟后，
他的所有直属下属都可以开始传播这一消息）。
返回通知所有员工这一紧急消息所需要的 分钟数 。
示例 1：输入：n = 1, headID = 0, manager = [-1], informTime = [0] 输出：0
解释：公司总负责人是该公司的唯一一名员工。
示例 2：输入：n = 6, headID = 2, manager = [2,2,-1,2,2,2], informTime = [0,0,1,0,0,0]
输出：1
解释：id = 2 的员工是公司的总负责人，也是其他所有员工的直属负责人，他需要 1␣
↪分钟来通知所有员工。
上图显示了公司员工的树结构。
示例 3：输入：n = 7, headID = 6, manager = [1,2,3,4,5,6,-1], informTime = [0,6,5,4,3,
↪2,1]
输出：21
解释：总负责人 id = 6。他将在 1 分钟内通知 id = 5 的员工。
id = 5 的员工将在 2 分钟内通知 id = 4 的员工。
id = 4 的员工将在 3 分钟内通知 id = 3 的员工。
id = 3 的员工将在 4 分钟内通知 id = 2 的员工。
id = 2 的员工将在 5 分钟内通知 id = 1 的员工。
id = 1 的员工将在 6 分钟内通知 id = 0 的员工。
所需时间 = 1 + 2 + 3 + 4 + 5 + 6 = 21 。
示例 4：输入：n = 15, headID = 0, manager = [-1,0,0,1,1,2,2,3,3,4,4,5,5,6,6],
informTime = [1,1,1,1,1,1,1,0,0,0,0,0,0,0,0] 输出：3
解释：第一分钟总负责人通知员工 1 和 2 。
第二分钟他们将会通知员工 3, 4, 5 和 6 。
第三分钟他们将会通知剩下的员工。
示例 5：输入：n = 4, headID = 2, manager = [3,3,-1,2], informTime = [0,0,162,914]␣
↪输出：1076
提示：
    1 <= n <= 10^5
    0 <= headID < n
    manager.length == n
    0 <= manager[i] < n
    manager[headID] == -1
    informTime.length == n
    0 <= informTime[i] <= 1000
    如果员工 i 没有下属，informTime[i] == 0 。
    题目 保证 所有员工都可以收到通知。
```

- 解题思路

```
var res int
var m map[int][]int

func numOfMinutes(n int, headID int, manager []int, informTime []int) int {
        m = make(map[int][]int)
        for i := 0; i < len(manager); i++ {
                if _, ok := m[manager[i]]; ok {
                        m[manager[i]] = append(m[manager[i]], i)
                } else {
                        m[manager[i]] = []int{i}
                }
        }
        res = 0
        dfs(headID, 0, informTime)
        return res
}

func dfs(headID int, cost int, informTime []int) {
        arr, ok := m[headID]
        if !ok {
                if cost > res {
                        res = cost
                }
                return
        }
        cost = cost + informTime[headID]
        for i := 0; i < len(arr); i++ {
                dfs(arr[i], cost, informTime)
        }
}

# 2
func numOfMinutes(n int, headID int, manager []int, informTime []int) int {
        res := 0
        for i := 0; i < len(manager); i++ {
                // 没有下属
                if informTime[i] == 0 {
                        count := 0
                        index := i
                        for index != -1 {
                                count = count + informTime[index]
                                index = manager[index]
                        }
                        res = max(res, count)
```

(续下页)

(接上页)

```
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func numOfMinutes(n int, headID int, manager []int, informTime []int) int {
        res := 0
        m := make(map[int][]int)
        for i := 0; i < len(manager); i++ {
                if _, ok := m[manager[i]]; ok {
                        m[manager[i]] = append(m[manager[i]], i)
                } else {
                        m[manager[i]] = []int{i}
                }
        }
        queue := make([]int, 0)
        queue = append(queue, headID)
        costM := make(map[int]int)
        costM[headID] = 0
        for len(queue) > 0 {
                id := queue[0]
                queue = queue[1:]
                res = max(res, costM[id])
                for i := 0; i < len(m[id]); i++ {
                        costM[m[id][i]] = informTime[id] + costM[id]
                        queue = append(queue, m[id][i])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

41.34 1381. 设计一个支持增量操作的栈 (2)

- 题目

```
请你设计一个支持下述操作的栈。
实现自定义栈类 CustomStack ：
CustomStack(int maxSize)：用 maxSize 初始化对象，maxSize 是栈中最多能容纳的元素数量，
栈在增长到 maxSize 之后则不支持 push 操作。
void push(int x)：如果栈还未增长到 maxSize ，就将 x 添加到栈顶。
int pop()：弹出栈顶元素，并返回栈顶的值，或栈为空时返回 -1 。
void inc(int k, int val)：栈底的 k 个元素的值都增加 val 。
如果栈中元素总数小于 k ，则栈中的所有元素都增加 val 。
示例：输入：
["CustomStack","push","push","pop","push","push","push",
"increment","increment","pop","pop","pop","pop"]
[[3],[1],[2],[],[2],[3],[4],[5,100],[2,100],[],[],[],[]]
输出：[null,null,null,2,null,null,null,null,null,103,202,201,-1]
解释：CustomStack customStack = new CustomStack(3); // 栈是空的 []
customStack.push(1);                          // 栈变为 [1]
customStack.push(2);                          // 栈变为 [1, 2]
customStack.pop();                            // 返回 2 --> 返回栈顶值 2，栈变为 [1]
customStack.push(2);                          // 栈变为 [1, 2]
customStack.push(3);                          // 栈变为 [1, 2, 3]
customStack.push(4);
// 栈仍然是 [1, 2, 3]，不能添加其他元素使栈大小变为 4
customStack.increment(5, 100);                // 栈变为 [101, 102, 103]
customStack.increment(2, 100);                // 栈变为 [201, 202, 103]
customStack.pop();
// 返回 103 --> 返回栈顶值 103，栈变为 [201, 202]
customStack.pop();                            // 返回 202 --> 返回栈顶值 202，栈变为␣
↪[201]
customStack.pop();                            // 返回 201 --> 返回栈顶值 201，栈变为␣
↪[]
customStack.pop();                            // 返回 -1 --> 栈为空，返回 -1
提示：1 <= maxSize <= 1000
1 <= x <= 1000
1 <= k <= 1000
0 <= val <= 100
每种方法 increment，push 以及 pop 分别最多调用 1000 次
```

- 解题思路

```
type CustomStack struct {
        stack []int
        size  int
}

func Constructor(maxSize int) CustomStack {
        return CustomStack{
                stack: make([]int, 0),
                size:  maxSize,
        }
}

func (this *CustomStack) Push(x int) {
        if len(this.stack) < this.size {
                this.stack = append(this.stack, x)
        }
}

func (this *CustomStack) Pop() int {
        if len(this.stack) > 0 {
                res := this.stack[len(this.stack)-1]
                this.stack = this.stack[:len(this.stack)-1]
                return res
        }
        return -1
}

func (this *CustomStack) Increment(k int, val int) {
        if k > len(this.stack) {
                k = len(this.stack)
        }
        for i := 0; i < k; i++ {
                this.stack[i] = this.stack[i] + val
        }
}

# 2
type CustomStack struct {
        stack []int
        add   []int
        top   int
}
```

(续下页)

(接上页)

```
func Constructor(maxSize int) CustomStack {
        return CustomStack{
                stack: make([]int, maxSize),
                add:   make([]int, maxSize),
                top:   -1,
        }
}

func (this *CustomStack) Push(x int) {
        if this.top != len(this.stack)-1 {
                this.top++
                this.stack[this.top] = x
        }
}

func (this *CustomStack) Pop() int {
        if this.top == -1 {
                return -1
        }
        res := this.stack[this.top] + this.add[this.top]
        if this.top != 0 {
                this.add[this.top-1] = this.add[this.top-1] + this.add[this.top]
        }
        this.add[this.top] = 0
        this.top--
        return res
}

func (this *CustomStack) Increment(k int, val int) {
        index := int(math.Min(float64(k-1), float64(this.top)))
        if index >= 0{
                this.add[index] = this.add[index] + val
        }
}
```

41.35 1382. 将二叉搜索树变平衡 (1)

- 题目

```
给你一棵二叉搜索树，请你返回一棵平衡后的二叉搜索树，新生成的树应该与原来的树有着相同的节点值。
如果一棵二叉搜索树中，每个节点的两棵子树高度差不超过 1␣
↪，我们就称这棵二叉搜索树是平衡的 。
如果有多种构造方法，请你返回任意一种。
示例：输入：root = [1,null,2,null,3,null,4,null,null] 输出：[2,1,3,null,null,null,4]
解释：这不是唯一的正确答案，[3,1,4,null,2,null,null] 也是一个可行的构造方案。
提示：树节点的数目在1到10^4之间。
树节点的值互不相同，且在1到10^5 之间。
```

- 解题思路

```go
type TreeNode struct {
        Val   int
        Left  *TreeNode
        Right *TreeNode
}

var arr []int

func balanceBST(root *TreeNode) *TreeNode {
        arr = make([]int, 0)
        dfs(root) // 先转换成数组
        return build(0, len(arr)-1)
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                arr = append(arr, root.Val)
                dfs(root.Right)
        }
}

// leetcode108.将有序数组转换为二叉搜索树
func build(left, right int) *TreeNode {
        if left > right {
                return nil
        }
        mid := left + (right-left)/2
        return &TreeNode{
```

(续下页)

(接上页)

```
                Val:   arr[mid],
                Left:  build(left, mid-1),
                Right: build(mid+1, right),
        }
}
```

41.36 1386. 安排电影院座位 (2)

- 题目

```
如上图所示，电影院的观影厅中有 n行座位，行编号从 1到 n，
且每一行内总共有 10 个座位，列编号从 1 到 10 。
给你数组reservedSeats，包含所有已经被预约了的座位。
比如说，researvedSeats[i]=[3,8]，它表示第3行第8个座位被预约了。
请你返回最多能安排多少个 4 人家庭。4 人家庭要占据同一行内连续的 4 个座位。
隔着过道的座位（比方说 [3,3] 和 [3,4]）不是连续的座位，
但是如果你可以将 4 人家庭拆成过道两边各坐 2 人，这样子是允许的。
示例 1：输入：n = 3, reservedSeats = [[1,2],[1,3],[1,8],[2,6],[3,1],[3,10]] 输出：4
解释：上图所示是最优的安排方案，总共可以安排 4 个家庭。
蓝色的叉表示被预约的座位，橙色的连续座位表示一个 4 人家庭。
示例 2：输入：n = 2, reservedSeats = [[2,1],[1,8],[2,6]] 输出：2
示例 3：输入：n = 4, reservedSeats = [[4,3],[1,4],[4,6],[1,7]] 输出：4
提示：1 <= n <= 10^9
1 <=reservedSeats.length <= min(10*n, 10^4)
reservedSeats[i].length == 2
1<=reservedSeats[i][0] <= n
1 <=reservedSeats[i][1] <= 10
所有reservedSeats[i] 都是互不相同的。
```

- 解题思路

```
func maxNumberOfFamilies(n int, reservedSeats [][]int) int {
        m := make(map[int]map[int]bool)
        for i := 0; i < len(reservedSeats); i++ {
                a, b := reservedSeats[i][0], reservedSeats[i][1]
                if b == 1 || b == 10 { // 1和10不影响
                        continue
                }
                if m[a] == nil {
                        m[a] = make(map[int]bool)
                }
                m[a][b] = true
```

(续下页)

(接上页)

```
        }
        res := 0
        for _, v := range m {
                flag := false
                // 左边
                if v[2] == false && v[3] == false && v[4] == false && v[5] == false {
                        res++
                        flag = true
                }
                // 右边
                if v[6] == false && v[7] == false && v[8] == false && v[9] == false {
                        res++
                        flag = true
                }
                // 中间
                if flag == false &&
                        v[4] == false && v[5] == false && v[6] == false && v[7] ==␣
↪false {
                        res++
                }
        }
        res = res + 2*(n-len(m))
        return res
}

# 2
func maxNumberOfFamilies(n int, reservedSeats [][]int) int {
        m := make(map[int]int)
        for i := 0; i < len(reservedSeats); i++ {
                a, b := reservedSeats[i][0], reservedSeats[i][1]
                if b == 1 || b == 10 { // 1和10不影响
                        continue
                }
                m[a] = m[a] | (1 << (b - 2))
        }
        left := 0b11110000
        middle := 0b11000011
        right := 0b00001111
        res := (n - len(m)) * 2
        for _, v := range m {
                // v一定有1个为1
                if v|left == left || v|middle == middle || v|right == right {
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}
```

41.37 1387. 将整数按权重排序 (2)

- 题目

```
我们将整数 x 的 权重 定义为按照下述规则将 x 变成 1 所需要的步数：
    如果 x 是偶数，那么 x = x / 2
    如果 x 是奇数，那么 x = 3 * x + 1
比方说，x=3 的权重为 7 。因为 3 需要 7 步变成 1
（3 --> 10 --> 5 --> 16 --> 8 --> 4 --> 2 --> 1）。
给你三个整数 lo， hi 和 k 。你的任务是将区间 [lo, hi] 之间的整数按照它们的权重␣
↪升序排序 ，
如果大于等于 2 个整数有 相同 的权重，那么按照数字自身的数值 升序排序 。
请你返回区间 [lo, hi] 之间的整数按权重排序后的第 k 个数。
注意，题目保证对于任意整数 x （lo <= x <= hi） ，它变成 1 所需要的步数是一个 32␣
↪位有符号整数。
示例 1：输入：lo = 12, hi = 15, k = 2 输出：13
解释：12 的权重为 9（12 --> 6 --> 3 --> 10 --> 5 --> 16 --> 8 --> 4 --> 2 --> 1）
13 的权重为 9
14 的权重为 17
15 的权重为 17
区间内的数按权重排序以后的结果为 [12,13,14,15] 。对于 k = 2 ，答案是第二个整数也就是␣
↪13 。
注意，12 和 13 有相同的权重，所以我们按照它们本身升序排序。14 和 15 同理。
示例 2：输入：lo = 1, hi = 1, k = 1 输出：1
示例 3：输入：lo = 7, hi = 11, k = 4 输出：7
解释：区间内整数 [7, 8, 9, 10, 11] 对应的权重为 [16, 3, 19, 6, 14] 。
按权重排序后得到的结果为 [8, 10, 11, 7, 9] 。
排序后数组中第 4 个数字为 7 。
示例 4：输入：lo = 10, hi = 20, k = 5 输出：13
示例 5：输入：lo = 1, hi = 1000, k = 777 输出：570
提示：1 <= lo <= hi <= 1000
    1 <= k <= hi - lo + 1
```

- 解题思路

```
var m map[int]int

```

(续下页)

(接上页)

```
func getKth(lo int, hi int, k int) int {
        m = make(map[int]int)
        arr := make([][2]int, 0)
        for i := lo; i <= hi; i++ {
                arr = append(arr, [2]int{i, getCount(i)})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][1] == arr[j][1] {
                        return arr[i][0] < arr[j][0]
                }
                return arr[i][1] < arr[j][1]
        })
        return arr[k-1][0]
}

func getCount(i int) int {
        res := 0
        temp := i
        for temp != 1 {
                if temp%2 == 1 {
                        temp = temp*3 + 1
                } else {
                        temp = temp / 2
                }
                res++
                if value, ok := m[temp]; ok {
                        res = res + value
                        break
                }
        }
        m[i] = res
        return res
}

# 2
func getKth(lo int, hi int, k int) int {
        arr := make([][2]int, 0)
        for i := lo; i <= hi; i++ {
                arr = append(arr, [2]int{i, getCount(i)})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][1] == arr[j][1] {
                        return arr[i][0] < arr[j][0]
```

(续下页)

(接上页)

```
                }
                return arr[i][1] < arr[j][1]
        })
        return arr[k-1][0]
}

func getCount(i int) int {
        if i == 1 {
                return 0
        }
        if i%2 == 1 {
                return getCount(i*3+1) + 1
        }
        return getCount(i/2) + 1
}
```

41.38 1390. 四因数 (1)

- 题目

```
给你一个整数数组 nums，请你返回该数组中恰有四个因数的这些整数的各因数之和。
如果数组中不存在满足题意的整数，则返回 0 。
示例：输入：nums = [21,4,7]输出：32
解释：21 有 4 个因数：1, 3, 7, 21
4 有 3 个因数：1, 2, 4
7 有 2 个因数：1, 7
答案仅为 21 的所有因数的和。
提示：1 <= nums.length <= 10^4
1 <= nums[i] <= 10^5
```

- 解题思路

```
func sumFourDivisors(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                sum := 0
                count := 0
                for j := 1; j*j <= nums[i]; j++ {
                        if nums[i]%j == 0 {
                                count++
                                sum = sum + j
                                if j*j != nums[i] {
```

(续下页)

(接上页)

```
                                        count++
                                        sum = sum + nums[i]/j
                                }
                        }
                }
                if count == 4 {
                        res = res + sum
                }
        }
        return res
}
```

41.39 1391. 检查网格中是否存在有效路径 (2)

- 题目

```
给你一个 m x n 的网格 grid。网格里的每个单元都代表一条街道。grid[i][j] 的街道可以是：
1 表示连接左单元格和右单元格的街道。
2 表示连接上单元格和下单元格的街道。
3表示连接左单元格和下单元格的街道。
4 表示连接右单元格和下单元格的街道。
5 表示连接左单元格和上单元格的街道。
6 表示连接右单元格和上单元格的街道。
你最开始从左上角的单元格 (0,0) 开始出发，网格中的「有效路径」是指从左上方的单元格 (0,
↪0) 开始、
一直到右下方的 (m-1,n-1) 结束的路径。该路径必须只沿着街道走。
注意：你 不能 变更街道。
如果网格中存在有效的路径，则返回 true，否则返回 false 。
示例 1：输入：grid = [[2,4,3],[6,5,2]] 输出：true
解释：如图所示，你可以从 (0, 0) 开始，访问网格中的所有单元格并到达 (m - 1, n - 1) 。
示例 2：输入：grid = [[1,2,1],[1,2,1]] 输出：false
解释：如图所示，单元格 (0, 0) 上的街道没有与任何其他单元格上的街道相连，你只会停在 (0,
↪ 0) 处。
示例 3：输入：grid = [[1,1,2]] 输出：false
解释：你会停在 (0, 1)，而且无法到达 (0, 2) 。
示例 4：输入：grid = [[1,1,1,1,1,1,3]] 输出：true
示例 5：输入：grid = [[2],[2],[2],[2],[2],[2],[6]] 输出：true
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 300
1 <= grid[i][j] <= 6
```

- 解题思路

```
func hasValidPath(grid [][]int) bool {
        n, m := len(grid), len(grid[0])
        arr := make([][]bool, 3*n)
        for i := 0; i < 3*n; i++ {
                arr[i] = make([]bool, 3*m)
        }
        for i := 0; i < n; i++ { // 放大处理
                for j := 0; j < m; j++ {
                        x, y := 3*i, 3*j     // 9宫格的左上角坐标
                        arr[x+1][y+1] = true // 换成9宫格后的中心点
                        switch grid[i][j] {
                        case 1:
                                arr[x+1][y], arr[x+1][y+2] = true, true
                        case 2:
                                arr[x][y+1], arr[x+2][y+1] = true, true
                        case 3:
                                arr[x+1][y], arr[x+2][y+1] = true, true
                        case 4:
                                arr[x+1][y+2], arr[x+2][y+1] = true, true
                        case 5:
                                arr[x+1][y], arr[x][y+1] = true, true
                        case 6:
                                arr[x][y+1], arr[x+1][y+2] = true, true
                        }
                }
        }
        return dfs(arr, 1, 1, 3*n-2, 3*m-2) // 从0,0的中心，走到n-1,m-1的中心
}

func dfs(arr [][]bool, i, j int, x, y int) bool {
        if i == x && j == y {
                return true
        }
        if i < 0 || i >= len(arr) || j < 0 || j >= len(arr[0]) || arr[i][j] == false {
                return false
        }
        arr[i][j] = false
        return dfs(arr, i-1, j, x, y) || dfs(arr, i+1, j, x, y) ||
                dfs(arr, i, j-1, x, y) || dfs(arr, i, j+1, x, y)
}

# 2
func hasValidPath(grid [][]int) bool {
        n, m := len(grid), len(grid[0])
```

(续下页)

(接上页)

```
        fa = Init(n * m)
        for i := 0; i < n; i++ { // 放大处理
                for j := 0; j < m; j++ {
                        index := i*m + j
                        if grid[i][j] == 1 || grid[i][j] == 3 || grid[i][j] == 5 {
                                if j > 0 && (grid[i][j-1] == 1 || grid[i][j-1] == 4
↪|| grid[i][j-1] == 6) {
                                        union(index, index-1) // 跟左边相连
                                }
                        }
                        if grid[i][j] == 2 || grid[i][j] == 5 || grid[i][j] == 6 {
                                if i > 0 && (grid[i-1][j] == 2 || grid[i-1][j] == 3
↪|| grid[i-1][j] == 4) {
                                        union(index, index-m) // 跟上边相连
                                }
                        }
                }
        }
        return query(0, n*m-1)
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
```

(续下页)

(接上页)

```
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

41.40 1395. 统计作战单位数 (2)

- 题目

```
n 名士兵站成一排。每个士兵都有一个 独一无二 的评分 rating 。
每 3 个士兵可以组成一个作战单位，分组规则如下：
    从队伍中选出下标分别为 i、j、k 的 3 名士兵，
    他们的评分分别为 rating[i]、rating[j]、rating[k]
    作战单位需满足： rating[i] < rating[j] < rating[k]
    或者 rating[i] > rating[j] > rating[k] ，其中  0 <= i < j < k < n
请你返回按上述条件可以组建的作战单位数量。每个士兵都可以是多个作战单位的一部分。
示例 1：输入：rating = [2,5,3,4,1] 输出：3
解释：我们可以组建三个作战单位 (2,3,4)、(5,4,1)、(5,3,1) 。
示例 2：输入：rating = [2,1,3] 输出：0
解释：根据题目条件，我们无法组建作战单位。
示例 3：输入：rating = [1,2,3,4] 输出：4
提示：n == rating.length
    1 <= n <= 200
    1 <= rating[i] <= 10^5
```

- 解题思路

```
func numTeams(rating []int) int {
        res := 0
        n := len(rating)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        for k := j + 1; k < n; k++ {
                                if (rating[i] < rating[j] && rating[j] < rating[k]) ||
                                        (rating[i] > rating[j] && rating[j] >␣
↪rating[k]) {
                                        res++
                                }
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func numTeams(rating []int) int {
        res := 0
        n := len(rating)
        for i := 0; i < n; i++ {
                leftMax, leftMin := 0, 0
                rightMax, rightMin := 0, 0
                for j := 0; j < i; j++ {
                        if rating[j] > rating[i] {
                                leftMax++
                        }
                        if rating[j] < rating[i] {
                                leftMin++
                        }
                }
                for j := i + 1; j < len(rating); j++ {
                        if rating[j] > rating[i] {
                                rightMin++
                        }
                        if rating[j] < rating[i] {
                                rightMax++
                        }
                }
                res = res + leftMin*rightMin + leftMax*rightMax
        }
        return res
}
```

41.41 1396. 设计地铁系统 (1)

- 题目

```
请你实现一个类UndergroundSystem，它支持以下 3 种方法：
1.checkIn(int id, string stationName, int t)
编号为id的乘客在 t时刻进入地铁站stationName。
一个乘客在同一时间只能在一个地铁站进入或者离开。
2.checkOut(int id, string stationName, int t)
编号为id的乘客在 t时刻离开地铁站 stationName。
3.getAverageTime(string startStation, string endStation)
```

(续下页)

(接上页)

```
返回从地铁站startStation到地铁站endStation的平均花费时间。
平均时间计算的行程包括当前为止所有从startStation直接到达endStation的行程。
调用getAverageTime时，询问的路线至少包含一趟行程。
你可以假设所有对checkIn和checkOut的调用都是符合逻辑的。
也就是说，如果一个顾客在 t1时刻到达某个地铁站，那么他离开的时间t2一定满足t2 >␣
↪t1。所有的事件都按时间顺序给出。
示例：输入：["UndergroundSystem","checkIn","checkIn","checkIn","checkOut","checkOut",
"checkOut","getAverageTime","getAverageTime","checkIn","getAverageTime","checkOut",
↪"getAverageTime"]
[[],[45,"Leyton",3],[32,"Paradise",8],[27,"Leyton",10],[45,"Waterloo",15],[27,
↪"Waterloo",20],
[32,"Cambridge",22],["Paradise","Cambridge"],["Leyton","Waterloo"],
[10,"Leyton",24],["Leyton","Waterloo"],[10,"Waterloo",38],["Leyton","Waterloo"]]
输出：  [null,null,null,null,null,null,null,14.0,11.0,null,11.0,null,12.0]
解释：
UndergroundSystem undergroundSystem = new UndergroundSystem();
undergroundSystem.checkIn(45, "Leyton", 3);
undergroundSystem.checkIn(32, "Paradise", 8);
undergroundSystem.checkIn(27, "Leyton", 10);
undergroundSystem.checkOut(45, "Waterloo", 15);
undergroundSystem.checkOut(27, "Waterloo", 20);
undergroundSystem.checkOut(32, "Cambridge", 22);
undergroundSystem.getAverageTime("Paradise", "Cambridge");
// 返回 14.0。从 "Paradise"（时刻 8）到 "Cambridge"(时刻 22)的行程只有一趟
undergroundSystem.getAverageTime("Leyton", "Waterloo");
// 返回 11.0。总共有 2 躺从 "Leyton" 到 "Waterloo" 的行程，
编号为 id=45 的乘客出发于 time=3 到达于 time=15，编号为 id=27 的乘客于 time=10 出发于␣
↪time=20 到达。
所以平均时间为 ( (15-3) + (20-10) ) / 2 = 11.0
undergroundSystem.checkIn(10, "Leyton", 24);
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // 返回 11.0
undergroundSystem.checkOut(10, "Waterloo", 38);
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // 返回 12.0
提示：总共最多有20000次操作。
1 <= id, t <= 10^6
所有的字符串包含大写字母，小写字母和数字。
1 <=stationName.length <= 10
与标准答案误差在10^-5以内的结果都视为正确结果。
```

- 解题思路

```
type Node struct {
        startName string
        startTime int
```

(续下页)

(接上页)

```
}

type SumInfo struct {
        count int // 总次数
        total int // 总时常
}

type UndergroundSystem struct {
        record map[int]Node
        sum    map[string]SumInfo
}

func Constructor() UndergroundSystem {
        return UndergroundSystem{
                record: make(map[int]Node),
                sum:    make(map[string]SumInfo),
        }
}

func (this *UndergroundSystem) CheckIn(id int, stationName string, t int) {
        this.record[id] = Node{
                startName: stationName,
                startTime: t,
        }
}

func (this *UndergroundSystem) CheckOut(id int, stationName string, t int) {
        key := this.record[id].startName + "=>" + stationName
        node := this.sum[key]
        node.count++
        node.total = node.total + t - this.record[id].startTime
        this.sum[key] = node
}

func (this *UndergroundSystem) GetAverageTime(startStation string, endStation string)␣
↪float64 {
        key := startStation + "=>" + endStation
        return float64(this.sum[key].total) / float64(this.sum[key].count)
}
```

41.42 1400. 构造 K 个回文字符串 (1)

- 题目

```
给你一个字符串 s和一个整数 k。请你用 s字符串中 所有字符构造 k个非空 回文串。
如果你可以用s中所有字符构造k个回文字符串，那么请你返回 True，否则返回False。
示例 1：输入：s = "annabelle", k = 2 输出：true
解释：可以用 s 中所有字符构造 2 个回文字符串。
一些可行的构造方案包括："anna" + "elble"，"anbna" + "elle"，"anellena" + "b"
示例 2：输入：s = "leetcode", k = 3 输出：false
解释：无法用 s 中所有字符构造 3 个回文串。
示例 3：输入：s = "true", k = 4 输出：true
解释：唯一可行的方案是让 s 中每个字符单独构成一个字符串。
示例 4：输入：s = "yzyzyzyzyzyzyzy", k = 2 输出：true
解释：你只需要将所有的 z 放在一个字符串中，所有的 y␣
↪放在另一个字符串中。那么两个字符串都是回文串。
示例 5：输入：s = "cr", k = 7 输出：false
解释：我们没有足够的字符去构造 7 个回文串。
提示：1 <= s.length <= 10^5
s中所有字符都是小写英文字母。
1 <= k <= 10^5
```

- 解题思路

```
func canConstruct(s string, k int) bool {
        if k > len(s) {
                return false
        }
        arr := [26]int{}
        for i := 0; i < len(s); i++ {
                arr[s[i]-'a']++
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                if arr[i]%2 == 1 {
                        res++
                }
        }
        return res <= k
}
```

CHAPTER 42

1301-1400-Hard

42.1 1301. 最大得分的路径数目 (1)

- 题目

```
给你一个正方形字符数组board，你从数组最右下方的字符'S'出发。
你的目标是到达数组最左上角的字符'E' ，数组剩余的部分为数字字符1，2，...，9或者障碍 'X
↪'。
在每一步移动中，你可以向上、向左或者左上方移动，可以移动的前提是到达的格子没有障碍。
一条路径的 「得分」 定义为：路径上所有数字的和。
请你返回一个列表，包含两个整数：第一个整数是 「得分」␣
↪的最大值，第二个整数是得到最大得分的方案数，请把结果对10^9 + 7 取余。
如果没有任何路径可以到达终点，请返回[0，0] 。
示例 1：输入：board = ["E23","2X2","12S"] 输出：[7,1]
示例 2：输入：board = ["E12","1X1","21S"] 输出：[4,2]
示例 3：输入：board = ["E11","XXX","11S"] 输出：[0,0]
提示：2 <= board.length == board[i].length <= 100
```

- 解题思路

```
var mod = 1000000007
var dx = []int{1, 0, 1}
var dy = []int{0, 1, 1}

func pathsWithMaxScore(board []string) []int {
```

(续下页)

(接上页)

```
        n := len(board)
        dp := make([][][2]int, n) // dp[i][j] 达到坐标i,j的 和最大值 以及 方案数
        for i := 0; i < n; i++ {
                dp[i] = make([][2]int, n)
        }
        dp[n-1][n-1][1] = 1 // 方案数
        for i := n - 1; i >= 0; i-- {
                for j := n - 1; j >= 0; j-- {
                        if board[i][j] == 'X' {
                                continue
                        }
                        var value int
                        if board[i][j] == 'E' || board[i][j] == 'S' {
                                value = 0
                        } else {
                                value = int(board[i][j] - '0')
                        }
                        for k := 0; k < 3; k++ { // 枚举当前节点下、右、下右节点
                                x := i + dx[k]
                                y := j + dy[k]
                                if x < n && y < n && dp[x][y][1] > 0 {
                                        sum := dp[x][y][0] + value
                                        if sum > dp[i][j][0] {
                                                dp[i][j][0] = sum
                                                dp[i][j][1] = dp[x][y][1]
                                        } else if sum == dp[i][j][0] {
                                                dp[i][j][1] = (dp[i][j][1] +␣
↪dp[x][y][1]) % mod
                                        }
                                }
                        }
                }
        }
        return []int{dp[0][0][0], dp[0][0][1] % mod}
}
```

42.2 1312. 让字符串成为回文串的最少插入次数 (4)

- 题目

```
给你一个字符串s，每一次操作你都可以在字符串的任意位置插入任意字符。
请你返回让s成为回文串的最少操作次数。
「回文串」是正读和反读都相同的字符串。
示例 1：输入：s = "zzazz" 输出：0
解释：字符串 "zzazz" 已经是回文串了，所以不需要做任何插入操作。
示例 2：输入：s = "mbadm" 输出：2
解释：字符串可变为 "mbdadbm" 或者 "mdbabdm" 。
示例 3：输入：s = "leetcode" 输出：5
解释：插入 5 个字符后字符串变为 "leetcodocteel" 。
示例 4：输入：s = "g" 输出：0
示例 5：输入：s = "no" 输出：1
提示：1 <= s.length <= 500
s中所有字符都是小写字母。
```

- 解题思路

```
func minInsertions(s string) int {
        n := len(s)
        t := reverse(s)
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
        }
        // 求s和它的反转字符串t的 最长公共子序列的长度
        // leetcode 1143.最长公共子序列
        for i := 1; i <= n; i++ {
                for j := 1; j <= n; j++ {
                        if s[i-1] == t[j-1] {
                                dp[i][j] = dp[i-1][j-1] + 1
                        } else {
                                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
                        }
                }
        }
        return n - dp[n][n]
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

func reverse(str string) string {
        arr := []byte(str)
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
        return string(arr)
}

# 2
func minInsertions(s string) int {
        n := len(s)
        dp := make([][]int, n) // dp[i][j]表示字符串s[i:j]成为回文的添加次数
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for k := 2; k <= n; k++ { // 枚举长度
                for i := 0; i <= n-k; i++ { // 枚举起始点
                        j := i + k - 1
                        dp[i][j] = min(dp[i+1][j], dp[i][j-1]) + 1 // 需要添加一个
                        if s[i] == s[j] {
                                dp[i][j] = min(dp[i][j], dp[i+1][j-1]) //␣
↪相等就不需要添加
                        }
                }
        }
        return dp[0][n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func minInsertions(s string) int {
        n := len(s)
        dp := make([][]int, n) // dp[i][j]表示字符串s[i:j]成为回文的添加次数
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        dp[i][j] = min(dp[i+1][j], dp[i][j-1]) + 1 // 需要添加一个
                        if s[i] == s[j] {
                                dp[i][j] = min(dp[i][j], dp[i+1][j-1]) //␣
↪相等就不需要添加
                        }
                }
        }
        return dp[0][n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func minInsertions(s string) int {
        n := len(s)
        // leetcode 516.最长回文子序列
        dp := make([][]int, n) // 最长回文子序列
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                dp[i][i] = 1
        }
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        if s[i] == s[j] {
                                dp[i][j] = dp[i+1][j-1] + 2 // 内层+2
                        } else {
                                dp[i][j] = max(dp[i+1][j], dp[i][j-1])
                        }
                }
        }
        return n - dp[0][n-1]
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

42.3 1326. 灌溉花园的最少水龙头数目 (3)

- 题目

```
在 x 轴上有一个一维的花园。花园长度为n，从点0开始，到点n结束。
花园里总共有n + 1 个水龙头，分别位于[0, 1, ..., n] 。
给你一个整数n和一个长度为n + 1 的整数数组ranges，其中ranges[i] （下标从 0 开始）表示：
如果打开点i处的水龙头，可以灌溉的区域为[i - ranges[i], i + ranges[i]]。
请你返回可以灌溉整个花园的最少水龙头数目。如果花园始终存在无法灌溉到的地方，请你返回-
↪1。
示例 1：输入：n = 5, ranges = [3,4,1,1,0,0] 输出：1
解释：点 0 处的水龙头可以灌溉区间 [-3,3]
点 1 处的水龙头可以灌溉区间 [-3,5]
点 2 处的水龙头可以灌溉区间 [1,3]
点 3 处的水龙头可以灌溉区间 [2,4]
点 4 处的水龙头可以灌溉区间 [4,4]
点 5 处的水龙头可以灌溉区间 [5,5]
只需要打开点 1 处的水龙头即可灌溉整个花园 [0,5] 。
示例 2：输入：n = 3, ranges = [0,0,0,0] 输出：-1
解释：即使打开所有水龙头，你也无法灌溉整个花园。
示例 3：输入：n = 7, ranges = [1,2,1,0,2,1,0,1] 输出：3
示例 4：输入：n = 8, ranges = [4,0,0,0,0,0,0,0,4] 输出：2
示例 5：输入：n = 8, ranges = [4,0,0,0,4,0,0,0,4] 输出：1
提示：1 <= n <= 10^4
ranges.length == n + 1
0 <= ranges[i] <= 100
```

- 解题思路

```
func minTaps(n int, ranges []int) int {
        // 处理成一组start-end的数组
        // 题目变成 leetcode 1024.视频拼接
        arr := make([][2]int, n+1)
        for i := 0; i <= n; i++ {
                l := max(0, i-ranges[i])
```

(续下页)

(接上页)

```
                r := min(n, i+ranges[i])
                arr = append(arr, [2]int{l, r})
        }
        dp := make([]int, n+1) //dp[i]表示将区间[0,i)覆盖所需的最少子区间的数量
        for i := 0; i <= n; i++ {
                dp[i] = math.MaxInt32 / 10
        }
        dp[0] = 0
        for i := 1; i <= n; i++ {
                for j := 0; j < len(arr); j++ {
                        a, b := arr[j][0], arr[j][1]
                        if a < i && i <= b && dp[a]+1 < dp[i] {
                                dp[i] = dp[a] + 1
                        }
                }
        }
        if dp[n] == math.MaxInt32/10 {
                return -1
        }
        return dp[n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minTaps(n int, ranges []int) int {
        // 题目变成leetcode45.跳跃游戏II
        arr := make([]int, n+1)
        for i := 0; i <= n; i++ {
                l := max(0, i-ranges[i])
                r := min(n, i+ranges[i])
```

(续下页)

(接上页)

```
                arr[l] = max(arr[l], r) // 更新当前位置能到达最远的位置
        }
        last := 0
        prev := 0
        res := 0
        // 变成leetcode45.跳跃游戏II的变形
        for i := 0; i < len(arr); i++ {
                if arr[i] > last {
                        last = arr[i]
                }
                if i == last && last < n { // 无法达到目标
                        return -1
                }
                if i == prev && i < n {
                        res++
                        prev = last
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func minTaps(n int, ranges []int) int {
        // 处理成一组start-end的数组
        // 题目变成leetcode45.跳跃游戏II
        arr := make([]int, n+1)
        for i := 0; i <= n; i++ {
                l := max(0, i-ranges[i])
                r := min(n, i+ranges[i])
```

(续下页)

(接上页)

```
                for j := l; j < r; j++ {
                        arr[j] = max(arr[j], r) // 更新当前位置能到达最远的位置
                }
        }
        res := 0
        cur := 0
        for cur < n {
                if arr[cur] == 0 {
                        return -1
                }
                cur = arr[cur]
                res++
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

42.4 1340. 跳跃游戏 V(2)

- 题目

```
给你一个整数数组arr 和一个整数d 。每一步你可以从下标i跳到：
i + x，其中i + x < arr.length且0 < x <= d。
i - x，其中i - x >= 0且0 < x <= d。
除此以外，你从下标i 跳到下标 j需要满足：arr[i] > arr[j]且 arr[i] > arr[k]，
其中下标k是所有 i到 j之间的数字（更正式的，min(i, j) < k < max(i, j)）。
你可以选择数组的任意下标开始跳跃。请你返回你 最多可以访问多少个下标。
请注意，任何时刻你都不能跳到数组的外面。
示例 1：输入：arr = [6,4,14,6,8,13,9,7,10,6,12], d = 2 输出：4
```

(续下页)

(接上页)

```
解释：你可以从下标 10 出发，然后如上图依次经过 10 --> 8 --> 6 --> 7 。
注意，如果你从下标 6 开始，你只能跳到下标 7 处。你不能跳到下标 5 处因为 13 > 9 。
你也不能跳到下标 4 处，因为下标 5 在下标 4 和 6 之间且 13 > 9 。
类似的，你不能从下标 3 处跳到下标 2 或者下标 1 处。
示例 2：输入：arr = [3,3,3,3,3], d = 3 输出：1
解释：你可以从任意下标处开始且你永远无法跳到任何其他坐标。
示例 3：输入：arr = [7,6,5,4,3,2,1], d = 1 输出：7
解释：从下标 0 处开始，你可以按照数值从大到小，访问所有的下标。
示例 4：输入：arr = [7,1,7,1,7,1], d = 2 输出：2
示例 5：输入：arr = [66], d = 1 输出：1
提示：1 <= arr.length <= 1000
1 <= arr[i] <= 10^5
1 <= d <= arr.length
```

- 解题思路

```
var dp []int

func maxJumps(arr []int, d int) int {
        n := len(arr)
        dp = make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = -1
        }
        for i := 0; i < n; i++ {
                dfs(arr, d, i)
        }
        res := 1
        for i := 0; i < n; i++ {
                res = max(res, dp[i])
        }
        return res
}

func dfs(arr []int, d int, index int) {
        if dp[index] != -1 {
                return
        }
        dp[index] = 1
        for i := index - 1; i >= 0 && index-i <= d && arr[index] > arr[i]; i-- {
                dfs(arr, d, i)
                dp[index] = max(dp[index], dp[i]+1)
        }
        for i := index + 1; i < len(arr) && i-index <= d && arr[index] > arr[i]; i++ {
```

(续下页)

(接上页)

```
                dfs(arr, d, i)
                dp[index] = max(dp[index], dp[i]+1)
        }
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
type Node struct {
        index int
        value int
}

func maxJumps(arr []int, d int) int {
        n := len(arr)
        dp := make([]int, n)
        temp := make([]Node, 0)
        for i := 0; i < n; i++ {
                temp = append(temp, Node{index: i, value: arr[i]})
        }
        sort.Slice(temp, func(i, j int) bool {
                if temp[i].value == temp[j].value {
                        return temp[i].index < temp[j].index
                }
                return temp[i].value < temp[j].value
        })
        res := 1
        for i := 0; i < n; i++ {
                index := temp[i].index
                dp[index] = 1
                for j := index - 1; j >= 0 && index-j <= d && arr[index] > arr[j]; j--
↪ {
                        if dp[j] != 0 {
                                dp[index] = max(dp[index], dp[j]+1)
                        }
                }
                for j := index + 1; j < len(arr) && j-index <= d && arr[index] >␣
↪arr[j]; j++ {
```

(续下页)

(接上页)

```
                        if dp[j] != 0 {
                                dp[index] = max(dp[index], dp[j]+1)
                        }
                }
                res = max(res, dp[index])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

42.5 1354. 多次求和构造目标数组 (1)

- 题目

```
给你一个整数数组target 。一开始，你有一个数组A ，它的所有元素均为 1␣
↪，你可以执行以下操作：
令x为你数组里所有元素的和
选择满足0 <= i < target.size的任意下标i，并让A数组里下标为i处的值为x。
你可以重复该过程任意次。如果能从A开始构造出目标数组target，请你返回 True ，否则返回␣
↪False 。
示例 1：输入：target = [9,3,5] 输出：true
解释：从 [1, 1, 1] 开始
[1, 1, 1]，和为 3 ，选择下标 1
[1, 3, 1]，和为 5， 选择下标 2
[1, 3, 5]，和为 9， 选择下标 0
[9, 3, 5] 完成
示例 2：输入：target = [1,1,1,2] 输出：false
解释：不可能从 [1,1,1,1] 出发构造目标数组。
示例 3：输入：target = [8,5] 输出：true
提示：N == target.length
1 <= target.length<= 5 * 10^4
1 <= target[i] <= 10^9
```

- 解题思路

```
func isPossible(target []int) bool {
        sum := 0
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < len(target); i++ {
                sum = sum + target[i]
                heap.Push(&intHeap, target[i])
        }
        for {
                curMax := heap.Pop(&intHeap).(int) // 当前轮最大值（从大到小替换）
                if curMax == 1 {                   // 全是1的情况
                        break
                }
                otherSum := sum - curMax
                // [5,8] 8 13
                // [3,5] 5 8
                // [2,3] 3 5
                // [1,2] 2 3
                if curMax <= otherSum || otherSum == 0 { // 例如[1,1,2]=>curMax=2,␣
↪otherSum=2的情况
                        return false
                }
                temp := curMax % otherSum
                heap.Push(&intHeap, temp)
                sum = sum - curMax + temp
        }
        return true
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}
```

(续下页)

(接上页)

```
func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

42.6 1359. 有效的快递序列数目 (1)

- 题目

```
给你n笔订单，每笔订单都需要快递服务。
请你统计所有有效的 收件/配送 序列的数目，
确保第 i 个物品的配送服务delivery(i) 总是在其收件服务pickup(i) 之后。
由于答案可能很大，请返回答案对 10^9 + 7 取余的结果。
示例 1：输入：n = 1 输出：1
解释：只有一种序列 (P1, D1)，物品 1 的配送服务 (D1) 在物品 1 的收件服务 (P1) 后。
示例 2：输入：n = 2 输出：6
解释：所有可能的序列包括：
(P1,P2,D1,D2)，(P1,P2,D2,D1)，(P1,D1,P2,D2)，(P2,P1,D1,D2)，
(P2,P1,D2,D1) 和 (P2,D2,P1,D1)。
(P1,D2,P2,D1) 是一个无效的序列，因为物品 2 的收件服务 (P2) 不应在物品 2␣
↪的配送服务 (D2) 之后。
示例 3：输入：n = 3 输出：90
提示：1 <= n <= 500
```

- 解题思路

```
var mod = 1000000007

func countOrders(n int) int {
        if n == 1 {
                return 1
        }
        res := 1
    // 前面有2(i-1)个数
        // 把Pi和Di当成1个插入，有2(i-1)+1=2i-1种插法
        // 把Pi和Di当前2个依次插入，有(2i-1)*(2i-2)/2=(2i-1)*(i-1)
        // 加起来 => (2*i-1)*i
```

(续下页)

(接上页)

```
        for i := 2; i <= n; i++ {
                res = res * (i*2 - 1) * i % mod
        }
        return res
}
```

42.7 1363. 形成三的最大倍数 (1)

- 题目

```
给你一个整数数组digits，你可以通过按任意顺序连接其中某些数字来形成 3 的倍数，
请你返回所能得到的最大的 3 的倍数。
由于答案可能不在整数数据类型范围内，请以字符串形式返回答案。
如果无法得到答案，请返回一个空字符串。
示例 1：输入：digits = [8,1,9] 输出："981"
示例 2：输入：digits = [8,6,7,1,0] 输出："8760"
示例 3：输入：digits = [1] 输出：""
示例 4：输入：digits = [0,0,0,0,0,0] 输出："0"
提示：1 <= digits.length <= 10^4
0 <= digits[i] <= 9
返回的结果不应包含不必要的前导零。
```

- 解题思路

```
func largestMultipleOfThree(digits []int) string {
        arr, arr3 := [10]int{}, [3]int{}
        var sum, index, count int // 需要移除的数字和个数
        for i := 0; i < len(digits); i++ {
                arr[digits[i]]++
                arr3[digits[i]%3]++
                sum = sum + digits[i]
        }
        if sum%3 == 1 { // 多个1
                if arr3[1] >= 1 {
                        index = 1
                        count = 1
                } else { // 移除2个2
                        index = 2
                        count = 2
                }
        } else if sum%3 == 2 {
                if arr3[2] >= 1 {
```

(续下页)

(接上页)

```
                        index = 2
                        count = 1
                } else { // 移除2个1
                        index = 1
                        count = 2
                }
        }
        res := make([]byte, 0)
        for i := 0; i <= 9; i++ {
                for j := 0; j < arr[i]; j++ {
                        if count > 0 && i%3 == index {
                                count--
                        } else {
                                res = append(res, byte('0'+i))
                        }
                }
        }
        sort.Slice(res, func(i, j int) bool {
                return res[i] > res[j]
        })
        if len(res) > 0 && res[0] == '0' {
                return "0"
        }
        return string(res)
}
```

42.8 1368. 使网格图至少有一条有效路径的最小代价 (3)

- 题目

```
给你一个 m x n␣
↪的网格图grid。grid中每个格子都有一个数字，对应着从该格子出发下一步走的方向。grid[i][j]中的数字可能

1，下一步往右走，也就是你会从 grid[i][j]走到 grid[i][j + 1]
2，下一步往左走，也就是你会从 grid[i][j]走到 grid[i][j - 1]
3，下一步往下走，也就是你会从 grid[i][j]走到 grid[i + 1][j]
4，下一步往上走，也就是你会从 grid[i][j]走到 grid[i - 1][j]
注意网格图中可能会有无效数字，因为它们可能指向grid以外的区域。
一开始，你会从最左上角的格子(0,0)出发。我们定义一条有效路径为从格子(0,0)出发，
每一步都顺着数字对应方向走，最终在最右下角的格子(m - 1, n -␣
↪1)结束的路径。有效路径不需要是最短路径。
你可以花费cost = 1的代价修改一个格子中的数字，但每个格子中的数字只能修改一次。
```

(续下页)

(接上页)

```
请你返回让网格图至少有一条有效路径的最小代价。
示例 1：输入：grid = [[1,1,1,1],[2,2,2,2],[1,1,1,1],[2,2,2,2]] 输出：3
解释：你将从点 (0, 0) 出发。
到达 (3, 3) 的路径为： (0, 0) --> (0, 1) --> (0, 2) --> (0, 3)
花费代价 cost = 1 使方向向下 --> (1, 3) --> (1, 2) --> (1, 1) --> (1, 0)
花费代价 cost = 1 使方向向下 --> (2, 0) --> (2, 1) --> (2, 2) --> (2, 3)
花费代价 cost = 1 使方向向下 --> (3, 3)
总花费为 cost = 3.
示例 2：输入：grid = [[1,1,3],[3,2,2],[1,1,4]] 输出：0
解释：不修改任何数字你就可以从 (0, 0) 到达 (2, 2) 。
示例 3：输入：grid = [[1,2],[4,3]] 输出：1
示例 4：输入：grid = [[2,2,2],[2,2,2]] 输出：3
示例 5：输入：grid = [[4]] 输出：0
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 100
```

- 解题思路

```
// 右左下上
var dx = []int{0, 0, 1, -1}
var dy = []int{1, -1, 0, 0}

func minCost(grid [][]int) int {
        n, m := len(grid), len(grid[0])
        total := n * m
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        arr[i][j] = total
                }
        }
        arr[0][0] = 0
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, []int{0, 0, 0})
        visited := make(map[[2]int]bool)
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([]int)
                v, a, b := node[0], node[1], node[2]
                if visited[[2]int{a, b}] == true {
                        continue
                }
```

(续下页)

(接上页)

```
                visited[[2]int{a, b}] = true
                for i := 0; i < 4; i++ {
                        x, y := a+dx[i], b+dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m {
                                dis := v
                                if i+1 != grid[a][b] {
                                        dis++ // 变换方向+1
                                }
                                if dis < arr[x][y] {
                                        arr[x][y] = dis
                                        heap.Push(&intHeap, []int{dis, x, y})
                                }
                        }
                }
        }
        return arr[n-1][m-1]
}

type IntHeap [][]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][0] < h[j][0] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 2
// 右左下上
var dx = []int{0, 0, 1, -1}
var dy = []int{1, -1, 0, 0}

func minCost(grid [][]int) int {
        n, m := len(grid), len(grid[0])
        total := n * m
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
```

(续下页)

(接上页)

```
                for j := 0; j < m; j++ {
                        arr[i][j] = total
                }
        }
        arr[0][0] = 0
        visited := make(map[[2]int]bool)
        deque := make([][]int, 0)
        deque = append(deque, []int{0, 0})
        for len(deque) > 0 {
                node := deque[0]
                deque = deque[1:]
                a, b := node[0], node[1]
                if visited[[2]int{a, b}] == true {
                        continue
                }
                visited[[2]int{a, b}] = true
                for i := 0; i < 4; i++ {
                        x, y := a+dx[i], b+dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m {
                                dis := arr[a][b]
                                if i+1 != grid[a][b] {
                                        dis = dis + 1
                                }
                                if dis < arr[x][y] {
                                        arr[x][y] = dis
                                        if i+1 == grid[a][b] { // 相同方向
                                                deque = append([][]int{{x, y}}, deque.
↪..) // 插入到前面
                                        } else {
                                                deque = append(deque, []int{x, y}) //␣
↪插入到后面
                                        }
                                }
                        }
                }
        }
        return arr[n-1][m-1]
}

# 3
// 右左下上
var dx = []int{0, 0, 1, -1}
var dy = []int{1, -1, 0, 0}
```

(续下页)

(接上页)

```
func minCost(grid [][]int) int {
        n, m := len(grid), len(grid[0])
        total := n * m
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        arr[i][j] = total
                }
        }
        arr[0][0] = 0
        queue := make([][]int, 0)
        queue = append(queue, []int{0, 0})
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                a, b := node[0], node[1]
                for i := 0; i < 4; i++ {
                        x, y := a+dx[i], b+dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m {
                                dis := arr[a][b]
                                if i+1 != grid[a][b] {
                                        dis = dis + 1
                                }
                                if dis < arr[x][y] {
                                        arr[x][y] = dis
                                        queue = append(queue, []int{x, y})
                                }
                        }
                }
        }
        return arr[n-1][m-1]
}
```

42.9 1373. 二叉搜索子树的最大键值和 (1)

- 题目

```
给你一棵以root为根的二叉树，请你返回 任意二叉搜索子树的最大键值和。
二叉搜索树的定义如下：
任意节点的左子树中的键值都小于此节点的键值。
任意节点的右子树中的键值都 大于此节点的键值。
任意节点的左子树和右子树都是二叉搜索树。
示例 1：输入：root = [1,4,3,2,4,2,5,null,null,null,null,null,null,4,6] 输出：20
解释：键值为 3 的子树是和最大的二叉搜索树。
示例 2：输入：root = [4,3,null,1,2] 输出：2
解释：键值为 2 的单节点子树是和最大的二叉搜索树。
示例 3：输入：root = [-4,-2,-5] 输出：0
解释：所有节点键值都为负数，和最大的二叉搜索树为空。
示例 4：输入：root = [2,1,3] 输出：6
示例 5：输入：root = [5,4,8,3,null,6,3] 输出：7
提示：每棵树有 1 到 40000个节点。
每个节点的键值在[-4 * 10^4, 4 * 10^4] 之间。
```

- 解题思路

```
var res int

func maxSumBST(root *TreeNode) int {
        res = 0
        dfs(root)
        return res
}

func dfs(root *TreeNode) (bool, int, int, int) {
        if root == nil {
                return true, 0, math.MaxInt32, math.MinInt32
        }

        leftOk, leftValue, leftMin, leftMax := dfs(root.Left)
        rightOk, rightValue, rightMin, rightMax := dfs(root.Right)
        if leftOk == false || rightOk == false || root.Val <= leftMax || root.Val >=␣
↪rightMin {
                return false, 0, 0, 0
        }
        sum := root.Val + leftValue + rightValue
        res = max(res, sum)
        return true, sum, min(root.Val, leftMin), max(root.Val, rightMax)
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

42.10 1377.T 秒后青蛙的位置 (2)

- 题目

```
给你一棵由 n 个顶点组成的无向树，顶点编号从 1 到 n。青蛙从 顶点 1 开始起跳。规则如下：
在一秒内，青蛙从它所在的当前顶点跳到另一个 未访问 过的顶点（如果它们直接相连）。
青蛙无法跳回已经访问过的顶点。
如果青蛙可以跳到多个不同顶点，那么它跳到其中任意一个顶点上的机率都相同。
如果青蛙不能跳到任何未访问过的顶点上，那么它每次跳跃都会停留在原地。
无向树的边用数组 edges 描述，其中 edges[i] = [fromi, toi] 意味着存在一条直接连通␣
↪fromi 和 toi 两个顶点的边
返回青蛙在 t 秒后位于目标顶点 target 上的概率。
示例 1：输入：n = 7, edges = [[1,2],[1,3],[1,7],[2,4],[2,6],[3,5]], t = 2, target = 4
输出：0.16666666666666666
解释：上图显示了青蛙的跳跃路径。青蛙从顶点 1 起跳，第 1 秒 有 1/3 的概率跳到顶点 2 ，
然后第 2 秒 有 1/2 的概率跳到顶点 4，因此青蛙在 2 秒后位于顶点 4 的概率是 1/3 * 1/2 =␣
↪1/6 = 0.16666666666666666 。
示例 2：输入：n = 7, edges = [[1,2],[1,3],[1,7],[2,4],[2,6],[3,5]], t = 1, target = 7␣
↪输出：0.3333333333333333
解释：上图显示了青蛙的跳跃路径。青蛙从顶点 1 起跳，有 1/3 = 0.3333333333333333␣
↪的概率能够 1 秒 后跳到顶点 7 。
示例 3：输入：n = 7, edges = [[1,2],[1,3],[1,7],[2,4],[2,6],[3,5]], t = 20, target = 6
输出：0.16666666666666666
提示：1 <= n <= 100
edges.length == n-1
edges[i].length == 2
```

(续下页)

(接上页)

```
1 <= edges[i][0], edges[i][1] <= n
1 <= t<= 50
1 <= target<= n
与准确值误差在 10^-5 之内的结果将被判定为正确。
```

- 解题思路

```
var arr [][]int
var res []float64

func frogPosition(n int, edges [][]int, t int, target int) float64 {
        arr = make([][]int, n+1)
        res = make([]float64, n+1)
        res[1] = 1
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        visited := make([]bool, n+1) // 已经访问过的
        visited[1] = true
        dfs(1, t, visited)
        return res[target]
}

func dfs(start int, t int, visited []bool) {
        if t <= 0 {
                return
        }
        count := 0
        for i := 0; i < len(arr[start]); i++ {
                next := arr[start][i]
                if visited[next] == false {
                        count++
                }
        }
        if count == 0 {
                return
        }
        per := res[start] / float64(count) // 每一跳的概率
        for i := 0; i < len(arr[start]); i++ {
                next := arr[start][i]
                if visited[next] == false {
                        visited[next] = true
```

(续下页)

(接上页)

```
                        res[start] = res[start] - per // start-per
                        res[next] = res[next] + per   // next+per
                        dfs(next, t-1, visited)
                        visited[next] = false
                }
        }
}

# 2
func frogPosition(n int, edges [][]int, t int, target int) float64 {
        arr := make([][]int, n+1)
        res := make([]float64, n+1)
        res[1] = 1
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        visited := make([]bool, n+1) // 已经访问过的
        queue := make([]int, 0)
        queue = append(queue, 1)
        count := 0
        for len(queue) > 0 {
                length := len(queue)
                if count == t {
                        break
                }
                for i := 0; i < length; i++ {
                        start := queue[i]
                        visited[start] = true
                        count := 0
                        for j := 0; j < len(arr[start]); j++ {
                                next := arr[start][j]
                                if visited[next] == false {
                                        count++
                                }
                        }
                        if count == 0 {
                                continue
                        }
                        per := res[start] / float64(count) // 每一跳的概率
                        for j := 0; j < len(arr[start]); j++ {
                                next := arr[start][j]
```

(续下页)

(接上页)

```
                                if visited[next] == false {
                                        res[start] = res[start] - per // start-per
                                        res[next] = res[next] + per   // next+per
                                        queue = append(queue, next)
                                }
                        }
                }
                queue = queue[length:]
                count++
        }
        return res[target]
}
```

42.11 1383. 最大的团队表现值 (1)

- 题目

```
公司有编号为 1到 n的 n个工程师，给你两个数组 speed和 efficiency，
其中 speed[i]和 efficiency[i]分别代表第 i位工程师的速度和效率。
请你返回由最多k个工程师组成的最大团队表现值，由于答案可能很大，请你返回结果对 10^9 +␣
↪7 取余后的结果。
团队表现值的定义为：一个团队中「所有工程师速度的和」乘以他们「效率值中的最小值」。
示例 1：输入：n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 2␣
↪输出：60
解释：我们选择工程师 2（speed=10 且 efficiency=4）和工程师 5（speed=5 且␣
↪efficiency=7）。
他们的团队表现值为 performance = (10 + 5) * min(4, 7) = 60 。
示例 2：输入：n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 3␣
↪输出：68
解释：此示例与第一个示例相同，除了 k = 3 。
我们可以选择工程师 1 ，工程师 2 和工程师 5 得到最大的团队表现值。
表现值为 performance = (2 + 10 + 5) * min(5, 4, 7) = 68 。
示例 3：输入：n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 4␣
↪输出：72
提示：1 <= n <= 10^5
speed.length == n
efficiency.length == n
1 <= speed[i] <= 10^5
1 <= efficiency[i] <= 10^8
1 <= k <= n
```

- 解题思路

```
func maxPerformance(n int, speed []int, efficiency []int, k int) int {
        arr := make([]Node, 0)
        for i := 0; i < len(speed); i++ {
                arr = append(arr, Node{
                        speed:      speed[i],
                        efficiency: efficiency[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i].efficiency == arr[j].efficiency {
                        return arr[i].speed > arr[j].speed
                }
                return arr[i].efficiency > arr[j].efficiency // 效率递减
        })
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        res := 0
        sum := 0
        for i := 0; i < len(arr); i++ {
                s := arr[i].speed
                e := arr[i].efficiency
                sum = sum + s           // 速度和
                heap.Push(&intHeap, s) // 速度
                if intHeap.Len() > k {
                        value := heap.Pop(&intHeap).(int)
                        sum = sum - value
                }
                res = max(res, sum*e)
        }
        return res % 1000000007
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type Node struct {
        speed      int
        efficiency int
}
```

(续下页)

(接上页)

```
type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

42.12 1388.3n 块披萨 (2)

- 题目

```
给你一个披萨，它由 3n␣
↪块不同大小的部分组成，现在你和你的朋友们需要按照如下规则来分披萨：
你挑选 任意一块披萨。
Alice 将会挑选你所选择的披萨逆时针方向的下一块披萨。
Bob 将会挑选你所选择的披萨顺时针方向的下一块披萨。
重复上述过程直到没有披萨剩下。
每一块披萨的大小按顺时针方向由循环数组 slices表示。
请你返回你可以获得的披萨大小总和的最大值。
示例 1: 输入: slices = [1,2,3,4,5,6] 输出: 10
解释: 选择大小为 4 的披萨，Alice 和 Bob 分别挑选大小为 3 和 5 的披萨。
然后你选择大小为 6 的披萨，Alice 和 Bob 分别挑选大小为 2 和 1␣
↪的披萨。你获得的披萨总大小为 4 + 6 = 10 。
示例 2: 输入: slices = [8,9,8,6,1,1] 输出: 16
```

(续下页)

(接上页)

```
解释：两轮都选大小为 8 的披萨。如果你选择大小为 9 的披萨，你的朋友们就会选择大小为 8␣
↪的披萨，这种情况下你的总和不是最大的。
示例 3：输入：slices = [4,1,2,5,8,3,1,9,7] 输出：21
示例 4：输入：slices = [3,1,2] 输出：3
提示：1 <= slices.length <= 500
slices.length % 3 == 0
1 <= slices[i] <= 1000
```

- 解题思路

```
func maxSizeSlices(slices []int) int {
        a := calculate(slices[1:])              // 去除第一个数
        b := calculate(slices[:len(slices)-1]) // 去除最后一个数
        return max(a, b)
}

func calculate(slices []int) int {
        n := len(slices)
        target := (n + 1) / 3
        dp := make([][]int, n+1) // dp[i][j] => 在前i个数，选择j个不相邻的数的最大和
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, target+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= target; j++ {
                        var a, b int
                        if 2 <= i {
                                a = dp[i-2][j-1]
                        }
                        a = a + slices[i-1] // 选择第i-1个数
                        b = dp[i-1][j]      // 不选第i-1个数
                        dp[i][j] = max(a, b)
                }
        }
        return dp[n][target]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
var m map[int][2]int

func maxSizeSlices(slices []int) int {
        n := len(slices)
        target := n / 3
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        m = make(map[int][2]int)
        for i := 0; i < n; i++ {
                left := (i - 1 + n) % n
                right := (i + 1) % n
                m[i] = [2]int{left, right} // 第i个数左右两边位置
                heap.Push(&intHeap, [2]int{slices[i], i})
        }
        res := 0
        visited := make([]bool, n)
        for i := 0; i < target; {
                top := heap.Pop(&intHeap).([2]int)
                value, index := top[0], top[1]
                if visited[index] == true { // 当前序号不可用
                        continue
                }
                i++
                left, right := m[index][0], m[index][1]
                visited[left], visited[right] = true, true
                res = res + value
                slices[index] = slices[left] + slices[right] - value //␣
↪更新当前序号的值为反悔值
                heap.Push(&intHeap, [2]int{slices[index], index})     // 重新赋值放回堆
                reconnect(left)                                       //␣
↪更改左边数的指针
                reconnect(right)                                      //␣
↪更改右边数的指针
        }
        return res
}

func reconnect(index int) {
        left, right := m[index][0], m[index][1]
        m[right] = [2]int{left, m[right][1]}
        m[left] = [2]int{m[left][0], right}
}
```

(续下页)

(接上页)

```
type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][0] > h[j][0]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

CHAPTER 43

1401-1500-Easy

43.1 1403. 非递增顺序的最小子序列 (2)

- 题目

```
给你一个数组 nums，请你从中抽取一个子序列，满足该子序列的元素之和 严格␣
↪大于未包含在该子序列中的各元素之和。
如果存在多个解决方案，只需返回 长度最小 的子序列。如果仍然有多个解决方案，则返回␣
↪元素之和最大 的子序列。
与子数组不同的地方在于，「数组的子序列」不强调元素在原数组中的连续性，
也就是说，它可以通过从数组中分离一些（也可能不分离）元素得到。
注意，题目数据保证满足所有约束条件的解决方案是 唯一 的。同时，返回的答案应当按␣
↪非递增顺序 排列。
示例 1：输入：nums = [4,3,10,9,8] 输出：[10,9]
解释：子序列 [10,9] 和 [10,8] 是最小的、满足元素之和大于其他各元素之和的子序列。
但是 [10,9] 的元素之和最大。
示例 2：输入：nums = [4,4,7,6,7] 输出：[7,7,6]
解释：子序列 [7,7] 的和为 14 ，不严格大于剩下的其他元素之和（14 = 4 + 4 + 6）。
因此，[7,6,7] 是满足题意的最小子序列。注意，元素按非递增顺序返回。
示例 3：输入：nums = [6] 输出：[6]
提示：
    1 <= nums.length <= 500
    1 <= nums[i] <= 100
```

- 解题思路

```
func minSubsequence(nums []int) []int {
        sort.Ints(nums)
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        target := sum / 2
        sum = 0
        res := make([]int, 0)
        for i := len(nums) - 1; i >= 0; i-- {
                if sum <= target {
                        res = append(res, nums[i])
                        sum = sum + nums[i]
                }
        }
        return res
}

#
func minSubsequence(nums []int) []int {
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] > nums[j]
        })
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        target := sum / 2
        sum = 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if sum > target {
                        return nums[:i+1]
                }
        }
        return nil
}
```

43.2 1408. 数组中的字符串匹配 (3)

- 题目

```
给你一个字符串数组 words ，数组中的每个字符串都可以看作是一个单词。
请你按 任意 顺序返回 words 中是其他单词的子字符串的所有单词。
如果你可以删除 words[j] 最左侧和/或最右侧的若干字符得到 word[i] ，
那么字符串 words[i] 就是 words[j] 的一个子字符串。
示例 1：输入：words = ["mass","as","hero","superhero"] 输出：["as","hero"]
解释："as" 是 "mass" 的子字符串，"hero" 是 "superhero" 的子字符串。
["hero","as"] 也是有效的答案。
示例 2：输入：words = ["leetcode","et","code"] 输出：["et","code"]
解释："et" 和 "code" 都是 "leetcode" 的子字符串。
示例 3：输入：words = ["blue","green","bu"] 输出：[]
提示：
    1 <= words.length <= 100
    1 <= words[i].length <= 30
    words[i] 仅包含小写英文字母。
    题目数据 保证 每个 words[i] 都是独一无二的。
```

- 解题思路

```
func stringMatching(words []string) []string {
        res := make([]string, 0)
        m := make(map[string]bool)
        for i := 0; i < len(words); i++ {
                for j := i + 1; j < len(words); j++ {
                        if strings.Contains(words[i], words[j]) {
                                if _, ok := m[words[j]]; !ok {
                                        res = append(res, words[j])
                                        m[words[j]] = true
                                }
                        } else if strings.Contains(words[j], words[i]) {
                                if _, ok := m[words[i]]; !ok {
                                        res = append(res, words[i])
                                        m[words[i]] = true
                                }
                        }
                }
        }
        return res
}

#
```

(续下页)

(接上页)

```
func stringMatching(words []string) []string {
        res := make([]string, 0)
        for i := 0; i < len(words); i++ {
                for j := 0; j < len(words); j++ {
                        if i != j && strings.Contains(words[j], words[i]) {
                                res = append(res, words[i])
                                break
                        }
                }
        }
        return res
}

#
func stringMatching(words []string) []string {
        sort.Slice(words, func(i, j int) bool {
                return len(words[i]) < len(words[j])
        })
        res := make([]string, 0)
        for i := 0; i < len(words); i++ {
                for j := i + 1; j < len(words); j++ {
                        if strings.Contains(words[j], words[i]) {
                                res = append(res, words[i])
                                break
                        }
                }
        }
        return res
}
```

43.3 1413. 逐步求和得到正数的最小值 (2)

- 题目

```
给你一个整数数组 nums 。你可以选定任意的 正数 startValue 作为初始值。
你需要从左到右遍历 nums 数组，并将 startValue 依次累加上 nums 数组中的值。
请你在确保累加和始终大于等于 1 的前提下，选出一个最小的 正数 作为 startValue 。
示例 1：输入：nums = [-3,2,-3,4,2] 输出：5
解释：如果你选择 startValue = 4，在第三次累加时，和小于 1 。
                累加求和
                startValue = 4 | startValue = 5 | nums
                  (4 -3 ) = 1  | (5 -3 ) = 2    |  -3
```

(续下页)

(接上页)

```
                  (1 +2 ) = 3  | (2 +2 ) = 4    |   2
                  (3 -3 ) = 0  | (4 -3 ) = 1    |  -3
                  (0 +4 ) = 4  | (1 +4 ) = 5    |   4
                  (4 +2 ) = 6  | (5 +2 ) = 7    |   2
示例 2: 输入: nums = [1,2] 输出: 1
解释: 最小的 startValue 需要是正数。
示例 3: 输入: nums = [1,-2,-3] 输出: 5
提示:
    1 <= nums.length <= 100
    -100 <= nums[i] <= 100
```

- 解题思路

```
func minStartValue(nums []int) int {
        min := nums[0]
        sum := nums[0]
        for i := 1; i < len(nums); i++ {
                sum = sum + nums[i]
                if sum < min {
                        min = sum
                }
        }
        if min >= 0 {
                return 1
        }
        return 1 - min
}

#
func minStartValue(nums []int) int {
        res := 1
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if sum+res < 1 {
                        res = 1 - sum
                }
        }
        return res
}
```

43.4 1417. 重新格式化字符串 (2)

- 题目

```
给你一个混合了数字和字母的字符串 s，其中的字母均为小写英文字母。
请你将该字符串重新格式化，使得任意两个相邻字符的类型都不同。
也就是说，字母后面应该跟着数字，而数字后面应该跟着字母。
请你返回 重新格式化后 的字符串；如果无法按要求重新格式化，则返回一个 空字符串 。
示例 1：输入：s = "a0b1c2" 输出："0a1b2c"
解释："0a1b2c" 中任意两个相邻字符的类型都不同。
"a0b1c2", "0a1b2c", "0c2a1b" 也是满足题目要求的答案。
示例 2：输入：s = "leetcode" 输出：""
解释："leetcode" 中只有字母，所以无法满足重新格式化的条件。
示例 3：输入：s = "1229857369" 输出：""
解释："1229857369" 中只有数字，所以无法满足重新格式化的条件。
示例 4：输入：s = "covid2019" 输出："c2o0v1i9d"
示例 5：输入：s = "ab123" 输出："1a2b3"
提示：
    1 <= s.length <= 500
    s 仅由小写英文字母和/或数字组成。
```

- 解题思路

```
func reformat(s string) string {
        arr := make([]byte, 0)
        str := make([]byte, 0)
        res := ""
        for i := 0; i < len(s); i++ {
                if s[i] >= '0' && s[i] <= '9' {
                        arr = append(arr, s[i])
                } else {
                        str = append(str, s[i])
                }
        }
        if abs(len(arr), len(str)) > 1 {
                return res
        } else {
                length := len(arr)
                if len(str) < length {
                        length = len(str)
                }
                for i := 0; i < length; i++ {
                        res = res + string(arr[i])
                        res = res + string(str[i])
```

(续下页)

(接上页)

```
                }
                if length == len(str) && length < len(arr) {
                        res = res + string(arr[length])
                } else if length == len(arr) && length < len(str) {
                        res = string(str[length]) + res
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func reformat(s string) string {
        res := make([]byte, 0)
        m1 := make([]byte, 0)
        m2 := make([]byte, 0)
        for i := range s {
                if s[i] >= '0' && s[i] <= '9' {
                        m1 = append(m1, s[i])
                } else {
                        m2 = append(m2, s[i])
                }
        }
        if len(m1)-len(m2) == 1 {
                for i := 0; i < len(m2); i++ {
                        res = append(res, m1[i])
                        res = append(res, m2[i])
                }
                res = append(res, m1[len(m1)-1])
                return string(res)
        } else if len(m2)-len(m1) == 1 {
                for i := 0; i < len(m1); i++ {
                        res = append(res, m2[i])
                        res = append(res, m1[i])
                }
                res = append(res, m2[len(m2)-1])
                return string(res)
```

(续下页)

(接上页)

```
        } else if len(m2) == len(m1) {
                for i := 0; i < len(m1); i++ {
                        res = append(res, m2[i])
                        res = append(res, m1[i])
                }
                return string(res)
        } else {
                return ""
        }
}
```

43.5 1422. 分割字符串的最大得分 (3)

- 题目

```
给你一个由若干 0 和 1 组成的字符串 s ，
请你计算并返回将该字符串分割成两个 非空 子字符串（即 左 子字符串和 右␣
↪子字符串）所能获得的最大得分。
「分割字符串的得分」为 左 子字符串中 0 的数量加上 右 子字符串中 1 的数量。
示例 1：输入：s = "011101" 输出：5
解释：
将字符串 s 划分为两个非空子字符串的可行方案有：
左子字符串 = "0" 且 右子字符串 = "11101"，得分 = 1 + 4 = 5
左子字符串 = "01" 且 右子字符串 = "1101"，得分 = 1 + 3 = 4
左子字符串 = "011" 且 右子字符串 = "101"，得分 = 1 + 2 = 3
左子字符串 = "0111" 且 右子字符串 = "01"，得分 = 1 + 1 = 2
左子字符串 = "01110" 且 右子字符串 = "1"，得分 = 2 + 1 = 3
示例 2：输入：s = "00111" 输出：5
解释：当 左子字符串 = "00" 且 右子字符串 = "111" 时，我们得到最大得分 = 2 + 3 = 5
示例 3：输入：s = "1111" 输出：3
提示：
2 <= s.length <= 500
字符串 s 仅由字符 '0' 和 '1' 组成。
```

- 解题思路

```
func maxScore(s string) int {
        one := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '1' {
                        one++
                }
```

(续下页)

(接上页)

```
        }
        max := 0
        zero := 0
        for i := 0; i < len(s)-1; i++ {
                if s[i] == '1' {
                        one--
                } else {
                        zero++
                }
                if one+zero > max {
                        max = one + zero
                }
        }
        return max
}

#
func maxScore(s string) int {
        max := 0
        for i := 0; i < len(s)-1; i++ {
                zero := 0
                one := 0
                for j := 0; j <= i; j++ {
                        if s[j] == '0' {
                                zero++
                        }
                }
                for j := i + 1; j < len(s); j++ {
                        if s[j] == '1' {
                                one++
                        }
                }
                if zero+one > max {
                        max = zero + one
                }
        }
        return max
}

#
func maxScore(s string) int {
        max := 0
        arr := make([]int, len(s)+1)
```

(续下页)

(接上页)

```
        for i := 0; i < len(s); i++ {
                if s[i] == '1' {
                        arr[i+1] = arr[i] + 1
                } else {
                        arr[i+1] = arr[i]
                }
        }
        for i := 1; i < len(s); i++ {
                zero := i - arr[i]
                one := arr[len(s)] - arr[i]
                v := zero + one
                if v > max {
                        max = v
                }
        }
        return max
}
```

43.6 1431. 拥有最多糖果的孩子 (2)

- 题目

```
给你一个数组 candies 和一个整数 extraCandies ，其中 candies[i] 代表第 i␣
↪个孩子拥有的糖果数目。
对每一个孩子，检查是否存在一种方案，将额外的 extraCandies␣
↪个糖果分配给孩子们之后，此孩子有 最多 的糖果。
注意，允许有多个孩子同时拥有 最多 的糖果数目。
示例 1：输入：candies = [2,3,5,1,3], extraCandies = 3 输出：[true,true,true,false,
↪true]
解释：
孩子 1 有 2 个糖果，如果他得到所有额外的糖果（3个），那么他总共有 5␣
↪个糖果，他将成为拥有最多糖果的孩子。
孩子 2 有 3 个糖果，如果他得到至少 2 个额外糖果，那么他将成为拥有最多糖果的孩子。
孩子 3 有 5 个糖果，他已经是拥有最多糖果的孩子。
孩子 4 有 1 个糖果，即使他得到所有额外的糖果，他也只有 4␣
↪个糖果，无法成为拥有糖果最多的孩子。
孩子 5 有 3 个糖果，如果他得到至少 2 个额外糖果，那么他将成为拥有最多糖果的孩子。
示例 2：输入：candies = [4,2,1,1,2], extraCandies = 1 输出：[true,false,false,false,
↪false]
解释：只有 1 个额外糖果，所以不管额外糖果给谁，只有孩子 1 可以成为拥有糖果最多的孩子。
示例 3：输入：candies = [12,1,12], extraCandies = 10 输出：[true,false,true]
提示：
```

(续下页)

(接上页)

```
    2 <= candies.length <= 100
    1 <= candies[i] <= 100
    1 <= extraCandies <= 50
```

- 解题思路

```
func kidsWithCandies(candies []int, extraCandies int) []bool {
        res := make([]bool, len(candies))
        max := 0
        for i := 0; i < len(candies); i++ {
                if candies[i] > max {
                        max = candies[i]
                }
        }
        for i := 0; i < len(candies); i++ {
                if candies[i]+extraCandies >= max {
                        res[i] = true
                }
        }
        return res
}

#
func kidsWithCandies(candies []int, extraCandies int) []bool {
        res := make([]bool, len(candies))
        for i := 0; i < len(candies); i++ {
                flag := true
                for j := 0; j < len(candies); j++ {
                        if candies[i]+extraCandies < candies[j] {
                                flag = false
                                res[i] = false
                                break
                        }
                }
                if flag == true {
                        res[i] = true
                }
        }
        return res
}
```

43.7 1436. 旅行终点站 (4)

- 题目

```
给你一份旅游线路图，该线路图中的旅行线路用数组 paths 表示，
其中 paths[i] = [cityAi, cityBi] 表示该线路将会从 cityAi 直接前往 cityBi 。
请你找出这次旅行的终点站，即没有任何可以通往其他城市的线路的城市。
题目数据保证线路图会形成一条不存在循环的线路，因此只会有一个旅行终点站。
示例 1:
输入：paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]
输出："Sao Paulo"
解释：从 "London" 出发，最后抵达终点站 "Sao Paulo" 。
本次旅行的路线是 "London" -> "New York" -> "Lima" -> "Sao Paulo" 。
示例 2：输入：paths = [["B","C"],["D","B"],["C","A"]] 输出："A"
解释：所有可能的线路是：
"D" -> "B" -> "C" -> "A".
"B" -> "C" -> "A".
"C" -> "A".
"A".
显然，旅行终点站是 "A" 。
示例 3：输入：paths = [["A","Z"]] 输出："Z"
提示：

    1 <= paths.length <= 100
    paths[i].length == 2
    1 <= cityAi.length, cityBi.length <= 10
    cityAi != cityBi
    所有字符串均由大小写英文字母和空格字符组成。
```

- 解题思路

```
func destCity(paths [][]string) string {
        m := make(map[string]string)
        for i := 0; i < len(paths); i++ {
                m[paths[i][0]] = paths[i][1]
        }
        for _, v := range m {
                if _, ok := m[v]; !ok {
                        return v
                }
        }
        return ""
}
```

(续下页)

(接上页)

```
#
func destCity(paths [][]string) string {
        m := make(map[string]bool)
        for i := 0; i < len(paths); i++ {
                m[paths[i][1]] = true
        }
        for i := 0; i < len(paths); i++ {
                m[paths[i][0]] = false
        }
        for key, value := range m {
                if value == true {
                        return key
                }
        }
        return ""
}

#
func destCity(paths [][]string) string {
        m := make(map[string]int)
        for i := 0; i < len(paths); i++ {
                m[paths[i][1]] -= 1
                m[paths[i][0]] += 1

        }
        for key, value := range m {
                if value == -1 {
                        return key
                }
        }
        return ""
}

#
func destCity(paths [][]string) string {
        for i := 0; i < len(paths); i++{
                flag := false
                for j := 0; j < len(paths); j++{
                        if j == i {
                                continue
                        }
                        if paths[i][1] == paths[j][0]{
                                flag = true
```

(续下页)

(接上页)

```
                                break
                        }
                }
                if flag == false{
                        return paths[i][1]
                }
        }
        return ""
}
```

43.8 1441. 用栈操作构建数组 (2)

- 题目

```
给你一个目标数组 target 和一个整数 n。每次迭代，需要从 list = {1,2,3..., n}
↪中依序读取一个数字。
请使用下述操作来构建目标数组 target ：
    Push：从 list 中读取一个新元素， 并将其推入数组中。
    Pop：删除数组中的最后一个元素。
    如果目标数组构建完成，就停止读取更多元素。
题目数据保证目标数组严格递增，并且只包含 1 到 n 之间的数字。
请返回构建目标数组所用的操作序列。
题目数据保证答案是唯一的。
示例 1：输入：target = [1,3], n = 3 输出：["Push","Push","Pop","Push"]
解释：
读取 1 并自动推入数组 -> [1]
读取 2 并自动推入数组，然后删除它 -> [1]
读取 3 并自动推入数组 -> [1,3]
示例 2：输入：target = [1,2,3], n = 3 输出：["Push","Push","Push"]
示例 3：输入：target = [1,2], n = 4 输出：["Push","Push"]
解释：只需要读取前 2 个数字就可以停止。
示例 4：输入：target = [2,3,4], n = 4 输出：["Push","Pop","Push","Push","Push"]
提示：
    1 <= target.length <= 100
    1 <= target[i] <= 100
    1 <= n <= 100
    target 是严格递增的
```

- 解题思路

```
func buildArray(target []int, n int) []string {
        res := make([]string, 0)
```

(续下页)

(接上页)

```
        j := 0
        for i := 1; i <= n; i++ {
                if j >= len(target) {
                        break
                }
                if target[j] != i {
                        res = append(res, "Push")
                        res = append(res, "Pop")
                } else {
                        res = append(res, "Push")
                        j++
                }
        }
        return res
}

#
func buildArray(target []int, n int) []string {
        res := make([]string, 0)
        j := 1
        for i := 0; i < len(target); i++ {
                for ; j < target[i]; j++ {
                        res = append(res, "Push")
                        res = append(res, "Pop")
                }
                res = append(res, "Push")
                j++
        }
        return res
}
```

43.9 1446. 连续字符 (2)

- 题目

```
给你一个字符串 s ，字符串的「能量」定义为：只包含一种字符的最长非空子字符串的长度。
请你返回字符串的能量。
示例 1：输入：s = "leetcode" 输出：2
解释：子字符串 "ee" 长度为 2 ，只包含字符 'e' 。
示例 2：输入：s = "abbcccddddeeeeedcba" 输出：5
解释：子字符串 "eeeee" 长度为 5 ，只包含字符 'e' 。
示例 3：输入：s = "triplepilloooooow" 输出：5
```

(续下页)

(接上页)

```
示例 4: 输入: s = "hooraaaaaaaaaaay" 输出: 11
示例 5: 输入: s = "tourist" 输出: 1
提示:
    1 <= s.length <= 500
    s 只包含小写英文字母。
```

- 解题思路

```
func maxPower(s string) int {
        max := 1
        count := 1
        for i := 1; i < len(s); i++ {
                if s[i] == s[i-1] {
                        count++
                } else {
                        count = 1
                }
                if count > max {
                        max = count
                }
        }
        return max
}

#
func maxPower(s string) int {
        max := 1
        left := 0
        right := 1
        for right < len(s) {
                if s[left] != s[right] {
                        if right-left > max {
                                max = right - left
                        }
                        left = right
                }
                right++
        }
        if right-left > max {
                return right - left
        }
        return max
}
```

43.10 1450. 在既定时间做作业的学生人数 (1)

- 题目

```
给你两个整数数组 startTime（开始时间）和 endTime（结束时间），并指定一个整数␣
↪queryTime 作为查询时间。
已知，第 i 名学生在 startTime[i] 时开始写作业并于 endTime[i] 时完成作业。
请返回在查询时间 queryTime 时正在做作业的学生人数。
形式上，返回能够使 queryTime 处于区间 [startTime[i], endTime[i]]（含）的学生人数。
示例 1：输入：startTime = [1,2,3], endTime = [3,2,7], queryTime = 4 输出：1
解释：一共有 3 名学生。
第一名学生在时间 1 开始写作业，并于时间 3 完成作业，在时间 4 没有处于做作业的状态。
第二名学生在时间 2 开始写作业，并于时间 2 完成作业，在时间 4 没有处于做作业的状态。
第三名学生在时间 3 开始写作业，预计于时间 7 完成作业，这是是唯一一名在时间 4␣
↪时正在做作业的学生。
示例 2：输入：startTime = [4], endTime = [4], queryTime = 4 输出：1
解释：在查询时间只有一名学生在做作业。
示例 3：输入：startTime = [4], endTime = [4], queryTime = 5 输出：0
示例 4：输入：startTime = [1,1,1,1], endTime = [1,3,2,4], queryTime = 7 输出：0
示例 5：输入：startTime = [9,8,7,6,5,4,3,2,1],
endTime = [10,10,10,10,10,10,10,10,10], queryTime = 5
输出：5
提示：

    startTime.length == endTime.length
    1 <= startTime.length <= 100
    1 <= startTime[i] <= endTime[i] <= 1000
    1 <= queryTime <= 1000
```

- 解题思路

```
func busyStudent(startTime []int, endTime []int, queryTime int) int {
        res := 0
        for i := 0; i < len(startTime); i++ {
                if queryTime >= startTime[i] && queryTime <= endTime[i] {
                        res++
                }
        }
        return res
}
```

43.11 1455. 检查单词是否为句中其他单词的前缀 (2)

- 题目

```
给你一个字符串 sentence 作为句子并指定检索词为 searchWord ，其中句子由若干用 单个空格␣
↪分隔的单词组成。
请你检查检索词 searchWord 是否为句子 sentence 中任意单词的前缀。
    如果 searchWord 是某一个单词的前缀，则返回句子 sentence␣
↪中该单词所对应的下标（下标从 1 开始）。
    如果 searchWord 是多个单词的前缀，则返回匹配的第一个单词的下标（最小下标）。
    如果 searchWord 不是任何单词的前缀，则返回 -1 。
字符串 S 的 「前缀」是 S 的任何前导连续子字符串。
示例 1：输入：sentence = "i love eating burger", searchWord = "burg" 输出：4
解释："burg" 是 "burger" 的前缀，而 "burger" 是句子中第 4 个单词。
示例 2：输入：sentence = "this problem is an easy problem", searchWord = "pro" 输出：2
解释："pro" 是 "problem" 的前缀，而 "problem" 是句子中第 2 个也是第 6 个单词，
但是应该返回最小下标 2 。
示例 3：输入：sentence = "i am tired", searchWord = "you" 输出：-1
解释："you" 不是句子中任何单词的前缀。
示例 4：输入：sentence = "i use triple pillow", searchWord = "pill" 输出：4
示例 5：输入：sentence = "hello from the other side", searchWord = "they" 输出：-1
提示：

    1 <= sentence.length <= 100
    1 <= searchWord.length <= 10
    sentence 由小写英文字母和空格组成。
    searchWord 由小写英文字母组成。
    前缀就是紧密附着于词根的语素，中间不能插入其它成分，并且它的位置是固定的——-
↪位于词根之前。
```

- 解题思路

```
func isPrefixOfWord(sentence string, searchWord string) int {
        arr := strings.Split(sentence, " ")
        for k, v := range arr {
                if strings.HasPrefix(v, searchWord) {
                        return k + 1
                }
        }
        return -1
}

#
func isPrefixOfWord(sentence string, searchWord string) int {
```

(续下页)

(接上页)

```
        arr := strings.Fields(sentence)
        for k, v := range arr {
                if len(v) >= len(searchWord) {
                        if v[:len(searchWord)] == searchWord {
                                return k + 1
                        }
                }
        }
        return -1
}
```

43.12 1460. 通过翻转子数组使两个数组相等 (3)

- 题目

```
给你两个长度相同的整数数组 target 和 arr 。
每一步中，你可以选择 arr 的任意 非空子数组 并将它翻转。你可以执行此过程任意次。
如果你能让 arr 变得与 target 相同，返回 True；否则，返回 False 。
示例 1：输入：target = [1,2,3,4], arr = [2,4,1,3] 输出：true
解释：你可以按照如下步骤使 arr 变成 target：
1- 翻转子数组 [2,4,1] ，arr 变成 [1,4,2,3]
2- 翻转子数组 [4,2] ，arr 变成 [1,2,4,3]
3- 翻转子数组 [4,3] ，arr 变成 [1,2,3,4]
上述方法并不是唯一的，还存在多种将 arr 变成 target 的方法。
示例 2：输入：target = [7], arr = [7] 输出：true
解释：arr 不需要做任何翻转已经与 target 相等。
示例 3：输入：target = [1,12], arr = [12,1] 输出：true
示例 4：输入：target = [3,7,9], arr = [3,7,11] 输出：false
解释：arr 没有数字 9 ，所以无论如何也无法变成 target 。
示例 5：输入：target = [1,1,1,1,1], arr = [1,1,1,1,1] 输出：true
提示：
    target.length == arr.length
    1 <= target.length <= 1000
    1 <= target[i] <= 1000
    1 <= arr[i] <= 1000
```

- 解题思路

```
func canBeEqual(target []int, arr []int) bool {
        sort.Ints(target)
        sort.Ints(arr)
        for i := 0; i < len(target); i++ {
```

(续下页)

(接上页)

```
                if target[i] != arr[i] {
                        return false
                }
        }
        return true
}

#
func canBeEqual(target []int, arr []int) bool {
        temp := make([]int, 1001)
        for i := 0; i < len(target); i++ {
                temp[target[i]]++
                temp[arr[i]]--
        }
        for i := 0; i < len(temp); i++ {
                if temp[i] != 0 {
                        return false
                }
        }
        return true
}
```

43.13 1464. 数组中两元素的最大乘积 (3)

- 题目

```
给你一个整数数组 nums，请你选择数组的两个不同下标 i 和 j，使 (nums[i]-1)*(nums[j]-1)␣
↪取得最大值。
请你计算并返回该式的最大值。
示例 1：输入：nums = [3,4,5,2] 输出：12
解释：如果选择下标 i=1 和 j=2（下标从 0 开始），则可以获得最大值，
(nums[1]-1)*(nums[2]-1) = (4-1)*(5-1) = 3*4 = 12 。
示例 2：输入：nums = [1,5,4,5] 输出：16
解释：选择下标 i=1 和 j=3（下标从 0 开始），则可以获得最大值 (5-1)*(5-1) = 16 。
示例 3：输入：nums = [3,7] 输出：12
提示：
    2 <= nums.length <= 500
    1 <= nums[i] <= 10^3
```

- 解题思路

```
func maxProduct(nums []int) int {
        sort.Ints(nums)
        return (nums[len(nums)-1] - 1) * (nums[len(nums)-2] - 1)
}

#
func maxProduct(nums []int) int {
        max := math.MinInt32
        next := math.MinInt32
        for i := 0; i < len(nums); i++ {
                if nums[i] > max {
                        next, max = max, nums[i]
                } else if nums[i] > next {
                        next = nums[i]
                }
        }
        return (max - 1) * (next - 1)
}

#
func maxProduct(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if (nums[i]-1)*(nums[j]-1) > res {
                                res = (nums[i] - 1) * (nums[j] - 1)
                        }
                }
        }
        return res
}
```

43.14 1470. 重新排列数组 (2)

- 题目

```
给你一个数组 nums ，数组中有 2n 个元素，按 [x1,x2,...,xn,y1,y2,...,yn] 的格式排列。
请你将数组按 [x1,y1,x2,y2,...,xn,yn] 格式重新排列，返回重排后的数组。
示例 1：输入：nums = [2,5,1,3,4,7], n = 3 输出：[2,3,5,4,1,7]
解释：由于 x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 ，所以答案为 [2,3,5,4,1,7]
示例 2：输入：nums = [1,2,3,4,4,3,2,1], n = 4 输出：[1,4,2,3,3,2,4,1]
示例 3：输入：nums = [1,1,2,2], n = 2 输出：[1,2,1,2]
```

(续下页)

(接上页)

```
提示：
    1 <= n <= 500
    nums.length == 2n
    1 <= nums[i] <= 10^3
```

- 解题思路

```
func shuffle(nums []int, n int) []int {
        res := make([]int,0)
        for i := 0; i < n; i++{
                res = append(res, nums[i], nums[i+n])
        }
        return res
}

#
func shuffle(nums []int, n int) []int {
        for i := n; i < 2*n; i++ {
                temp := i
                for j := 0; j < 2*n-1-i; j++ {
                        nums[temp], nums[temp-1] = nums[temp-1], nums[temp]
                        temp--
                }
        }
        return nums
}
```

43.15 1475. 商品折扣后的最终价格 (2)

- 题目

```
给你一个数组 prices ，其中 prices[i] 是商店里第 i 件商品的价格。
商店里正在进行促销活动，如果你要买第 i 件商品，那么你可以得到与 prices[j] 相等的折扣，
其中 j 是满足 j > i 且 prices[j] <= prices[i] 的 最小下标 ，
如果没有满足条件的 j ，你将没有任何折扣。
请你返回一个数组，数组中第 i 个元素是折扣后你购买商品 i 最终需要支付的价格。
示例 1：输入：prices = [8,4,6,2,3] 输出：[4,2,4,2,3]
解释：
商品 0 的价格为 price[0]=8 ，你将得到 prices[1]=4 的折扣，所以最终价格为 8 - 4 = 4 。
商品 1 的价格为 price[1]=4 ，你将得到 prices[3]=2 的折扣，所以最终价格为 4 - 2 = 2 。
商品 2 的价格为 price[2]=6 ，你将得到 prices[3]=2 的折扣，所以最终价格为 6 - 2 = 4 。
商品 3 和 4 都没有折扣。
```

(续下页)

(接上页)

```
示例 2：输入：prices = [1,2,3,4,5] 输出：[1,2,3,4,5]
解释：在这个例子中，所有商品都没有折扣。
示例 3：输入：prices = [10,1,1,6] 输出：[9,0,1,6]
提示：
    1 <= prices.length <= 500
    1 <= prices[i] <= 10^3
```

- 解题思路

```
func finalPrices(prices []int) []int {
        for i := 0; i < len(prices); i++ {
                for j := i + 1; j < len(prices); j++ {
                        if prices[j] <= prices[i] {
                                prices[i] = prices[i] - prices[j]
                                break
                        }
                }
        }
        return prices
}

#
func finalPrices(prices []int) []int {
        stack := make([]int, 0)
        for i := 0; i < len(prices); i++ {
                for len(stack) > 0 {
                        index := stack[len(stack)-1]
                        if prices[i] > prices[index] {
                                break
                        }
                        prices[index] = prices[index] - prices[i]
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, i)
        }
        return prices
}
```

43.16 1480. 一维数组的动态和 (2)

- 题目

```
给你一个数组 nums 。数组「动态和」的计算公式为：runningSum[i] = sum(nums[0]…nums[i])。
请返回 nums 的动态和。
示例 1：输入：nums = [1,2,3,4] 输出：[1,3,6,10]
解释：动态和计算过程为 [1, 1+2, 1+2+3, 1+2+3+4] 。
示例 2：输入：nums = [1,1,1,1,1] 输出：[1,2,3,4,5]
解释：动态和计算过程为 [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1] 。
示例 3：输入：nums = [3,1,2,10,1] 输出：[3,4,6,16,17]
提示：
    1 <= nums.length <= 1000
    -10^6 <= nums[i] <= 10^6
```

- 解题思路

```
func runningSum(nums []int) []int {
        for i := 1; i < len(nums); i++{
                nums[i] = nums[i-1]+nums[i]
        }
        return nums
}

#
func runningSum(nums []int) []int {
        res := make([]int, len(nums))
        res[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                res[i] = res[i-1] + nums[i]
        }
        return res
}
```

43.17 1486. 数组异或操作 (1)

- 题目

```
给你两个整数，n 和 start 。
数组 nums 定义为：nums[i] = start + 2*i（下标从 0 开始）且 n == nums.length 。
请返回 nums 中所有元素按位异或（XOR）后得到的结果。
示例 1：输入：n = 5, start = 0 输出：8
```

(续下页)

(接上页)

```
解释：数组 nums 为 [0, 2, 4, 6, 8]，其中 (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8 。
     "^" 为按位异或 XOR 运算符。
示例 2：输入：n = 4, start = 3 输出：8
解释：数组 nums 为 [3, 5, 7, 9]，其中 (3 ^ 5 ^ 7 ^ 9) = 8.
示例 3：输入：n = 1, start = 7 输出：7
示例 4：输入：n = 10, start = 5 输出：2
提示：
    1 <= n <= 1000
    0 <= start <= 1000
    n == nums.length
```

- 解题思路

```
func xorOperation(n int, start int) int {
        res := 0
        for i := 0; i < n; i++ {
                res = res ^ (start + 2*i)
        }
        return res
}
```

43.18 1491. 去掉最低工资和最高工资后的工资平均值 (2)

- 题目

```
给你一个整数数组 salary ，数组里每个数都是 唯一 的，其中 salary[i] 是第 i␣
↪个员工的工资。
请你返回去掉最低工资和最高工资以后，剩下员工工资的平均值。
示例 1：输入：salary = [4000,3000,1000,2000] 输出：2500.00000
解释：最低工资和最高工资分别是 1000 和 4000 。
去掉最低工资和最高工资以后的平均工资是 (2000+3000)/2= 2500
示例 2：输入：salary = [1000,2000,3000] 输出：2000.00000
解释：最低工资和最高工资分别是 1000 和 3000 。
去掉最低工资和最高工资以后的平均工资是 (2000)/1= 2000
示例 3：输入：salary = [6000,5000,4000,3000,2000,1000] 输出：3500.00000
示例 4：输入：salary = [8000,9000,2000,3000,6000,1000] 输出：4750.00000
提示：
    3 <= salary.length <= 100
    10^3 <= salary[i] <= 10^6
    salary[i] 是唯一的。
    与真实值误差在 10^-5 以内的结果都将视为正确答案。
```

- 解题思路

```
func average(salary []int) float64 {
        sort.Ints(salary)
        sum := 0
        for i := 1; i < len(salary)-1; i++ {
                sum = sum + salary[i]
        }
        return float64(sum) / float64(len(salary)-2)
}

#
func average(salary []int) float64 {
        sum := salary[0]
        max := salary[0]
        min := salary[0]
        for i := 1; i < len(salary); i++ {
                sum = sum + salary[i]
                if salary[i] > max {
                        max = salary[i]
                }
                if salary[i] < min {
                        min = salary[i]
                }
        }
        return float64(sum-max-min) / float64(len(salary)-2)
}
```

43.19 1496. 判断路径是否相交 (1)

- 题目

```
给你一个字符串 path，其中 path[i] 的值可以是 'N'、'S'、'E' 或者 'W'，
分别表示向北、向南、向东、向西移动一个单位。
机器人从二维平面上的原点 (0, 0) 处开始出发，按 path 所指示的路径行走。
如果路径在任何位置上出现相交的情况，也就是走到之前已经走过的位置，请返回 True␣
↪；否则，返回 False 。
示例 1：输入：path = "NES" 输出：false
解释：该路径没有在任何位置相交。
示例 2：输入：path = "NESWW" 输出：true
解释：该路径经过原点两次。
提示：
    1 <= path.length <= 10^4
    path 仅由 {'N', 'S', 'E', 'W} 中的字符组成
```

- 解题思路

```
func isPathCrossing(path string) bool {
        m := make(map[string]bool)
        m["0,0"] = true
        x := 0
        y := 0
        for i := 0; i < len(path); i++ {
                switch path[i] {
                case 'N':
                        y = y + 1
                case 'S':
                        y = y - 1
                case 'E':
                        x = x + 1
                case 'W':
                        x = x - 1
                }
                if m[fmt.Sprintf("%d,%d", x, y)] {
                        return true
                }
                m[fmt.Sprintf("%d,%d", x, y)] = true
        }
        return false
}
```

CHAPTER 44

1401-1500-Medium

44.1 1401. 圆和矩形是否有重叠 (2)

- 题目

```
给你一个以 (radius, x_center, y_center) 表示的圆和一个与坐标轴平行的矩形 (x1, y1, x2,␣
↪y2)，
其中(x1, y1) 是矩形左下角的坐标，(x2, y2) 是右上角的坐标。
如果圆和矩形有重叠的部分，请你返回 True ，否则返回 False。
换句话说，请你检测是否 存在 点(xi, yi)␣
↪，它既在圆上也在矩形上（两者都包括点落在边界上的情况）。
示例 1：输入：radius = 1, x_center = 0, y_center = 0, x1 = 1, y1 = -1, x2 = 3, y2 = 1
输出：true
解释：圆和矩形有公共点 (1,0)
示例 2：输入：radius = 1, x_center = 0, y_center = 0, x1 = -1, y1 = 0, x2 = 0, y2 = 1
输出：true
示例 3：输入：radius = 1, x_center = 1, y_center = 1, x1 = -3, y1 = -3, x2 = 3, y2 = 3
输出：true
示例 4：输入：radius = 1, x_center = 1, y_center = 1, x1 = 1, y1 = -3, x2 = 2, y2 = -1
输出：false
提示：1 <= radius <= 2000
-10^4 <= x_center, y_center, x1, y1, x2, y2 <= 10^4
x1 < x2
y1 < y2
```

- 解题思路

```
func checkOverlap(radius int, x_center int, y_center int, x1 int, y1 int, x2 int, y2␣
↪int) bool {
        if x1 <= x_center && x_center <= x2 && // 在里面
                y1 <= y_center && y_center <= y2 {
                return true
        } else if x_center > x2 && y1 <= y_center && y_center <= y2 { // 右边
                return x_center-x2 <= radius
        } else if x_center < x1 && y1 <= y_center && y_center <= y2 { // 左边
                return x1-x_center <= radius
        } else if y_center < y1 && x1 <= x_center && x_center <= x2 { // 下边
                return y1-y_center <= radius
        } else if y_center > y2 && x1 <= x_center && x_center <= x2 { // 上边
                return y_center-y2 <= radius
        }
        // 4个顶点判断
        minValue := (x1-x_center)*(x1-x_center) + (y1-y_center)*(y1-y_center)
        minValue = min(minValue, (x1-x_center)*(x1-x_center)+(y2-y_center)*(y2-y_
↪center))
        minValue = min(minValue, (x2-x_center)*(x2-x_center)+(y1-y_center)*(y1-y_
↪center))
        minValue = min(minValue, (x2-x_center)*(x2-x_center)+(y2-y_center)*(y2-y_
↪center))
        return minValue <= radius*radius
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func checkOverlap(radius int, x_center int, y_center int, x1 int, y1 int, x2 int, y2␣
↪int) bool {
        minValue := 0
        // minValue = (x-x_center)*(x-x_center)+(y-y_center)*(y-y_center)
        if x_center < x1 || x_center > x2 {
                minValue = minValue + min((x1-x_center)*(x1-x_center), (x2-x_
↪center)*(x2-x_center))
        }
        if y_center < y1 || y_center > y2 {
                minValue = minValue + min((y1-y_center)*(y1-y_center), (y2-y_
```

(续下页)

(接上页)

```
↪center)*(y2-y_center))
        }
        return minValue <= radius*radius
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

44.2 1404. 将二进制表示减到 1 的步骤数 (2)

- 题目

```
给你一个以二进制形式表示的数字 s 。请你返回按下述规则将其减少到 1 所需要的步骤数：
    如果当前数字为偶数，则将其除以 2 。
    如果当前数字为奇数，则将其加上 1 。
题目保证你总是可以按上述规则将测试用例变为 1 。
示例 1：输入：s = "1101" 输出：6
解释："1101" 表示十进制数 13 。
Step 1) 13 是奇数，加 1 得到 14
Step 2) 14 是偶数，除 2 得到 7
Step 3) 7  是奇数，加 1 得到 8
Step 4) 8  是偶数，除 2 得到 4
Step 5) 4  是偶数，除 2 得到 2
Step 6) 2  是偶数，除 2 得到 1
示例 2：输入：s = "10" 输出：1
解释："10" 表示十进制数 2 。
Step 1) 2 是偶数，除 2 得到 1
示例 3：输入：s = "1" 输出：0
提示：1 <= s.length <= 500
    s 由字符 '0' 或 '1' 组成。
    s[0] == '1'
```

- 解题思路

```
func numSteps(s string) int {
        res := 0
        for s != "1" {
                n := len(s)
```

(续下页)

(接上页)

```
                if s[n-1] == '0' {
                        s = s[:n-1]
                } else {
                        s = add(s)
                }
                res++
        }
        return res
}

func add(s string) string {
        arr := []byte(s)
        flag := true
        for i := len(arr) - 1; i >= 0; i-- {
                if arr[i] == '0' {
                        arr[i] = '1'
                        flag = false
                } else {
                        arr[i] = '0'
                }
                if flag == false {
                        return string(arr)
                }
        }
        return "1" + string(arr)
}

# 2
func numSteps(s string) int {
        res := 0
        flag := false
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '0' {
                        if flag == true {
                                res = res + 2
                        } else {
                                res = res + 1 // 没有进位，遇0加1
                        }
                } else {
                        if flag == true {
                                res++
                        } else {
                                if i != 0 {
```

(续下页)

(接上页)

```
                                        res = res + 2
                                }
                                flag = true
                        }
                }
        }
        return res
}
```

44.3 1405. 最长快乐字符串 (1)

- 题目

```
如果字符串中不含有任何 'aaa', 'bbb' 或 'ccc'␣
↪这样的字符串作为子串，那么该字符串就是一个「快乐字符串」。
给你三个整数 a, b ，c, 请你返回 任意一个 满足下列全部条件的字符串 s:
s 是一个尽可能长的快乐字符串。
s 中 最多 有a 个字母 'a'、b个字母 'b'、c 个字母 'c' 。
s 中只含有 'a'、'b' 、'c' 三种字母。
如果不存在这样的字符串 s ，请返回一个空字符串 ""。
示例 1: 输入: a = 1, b = 1, c = 7 输出: "ccaccbcc"
解释: "ccbccacc" 也是一种正确答案。
示例 2: 输入: a = 2, b = 2, c = 1 输出: "aabbc"
示例 3: 输入: a = 7, b = 1, c = 0 输出: "aabaa"
解释: 这是该测试用例的唯一正确答案。
提示: 0 <= a, b, c <= 100
a + b + c > 0
```

- 解题思路

```
func longestDiverseString(a int, b int, c int) string {
        arr := [][2]int{
                {0, a},
                {1, b},
                {2, c},
        }
        res := make([]byte, 0)
        for {
                // 按照次数排序
                sort.Slice(arr, func(i, j int) bool {
                        return arr[i][1] < arr[j][1]
                })
```

(续下页)

(接上页)

```
                // 每次放1个，如果最后2个相同，则使用次多的
                if len(res) >= 2 &&
                        res[len(res)-1] == byte(arr[2][0]+'a') &&
                        res[len(res)-2] == byte(arr[2][0]+'a') {
                        if arr[1][1] > 0 { // 使用次多的
                                res = append(res, byte(arr[1][0]+'a'))
                                arr[1][1]--
                        } else {
                                break
                        }
                } else {
                        if arr[2][1] > 0 { // 使用最多的
                                res = append(res, byte(arr[2][0]+'a'))
                                arr[2][1]--
                        } else {
                                break
                        }
                }
        }
        return string(res)
}
```

44.4 1409. 查询带键的排列 (4)

- 题目

```
给你一个待查数组 queries ，数组中的元素为 1 到 m 之间的正整数。
请你根据以下规则处理所有待查项 queries[i] (从 i=0 到 i=queries.length-1) :
一开始，排列 P=[1,2,3,...,m]。
对于当前的 i ，请你找出待查项 queries[i] 在排列 P 中的位置（下标从 0 开始），
然后将其从原位置移动到排列 P 的起始位置（即下标为 0 处）。
注意， queries[i] 在 P 中的位置就是 queries[i] 的查询结果。
请你以数组形式返回待查数组 queries 的查询结果。
示例 1: 输入: queries = [3,1,2,1], m = 5 输出: [2,1,2,1]
解释: 待查数组 queries 处理如下:
对于 i=0: queries[i]=3, P=[1,2,3,4,5], 3 在 P 中的位置是 2，接着我们把 3 移动到 P␣
↪的起始位置，得到 P=[3,1,2,4,5] 。
对于 i=1: queries[i]=1, P=[3,1,2,4,5], 1 在 P 中的位置是 1，接着我们把 1 移动到 P␣
↪的起始位置，得到 P=[1,3,2,4,5] 。
对于 i=2: queries[i]=2, P=[1,3,2,4,5], 2 在 P 中的位置是 2，接着我们把 2 移动到 P␣
↪的起始位置，得到 P=[2,1,3,4,5] 。
对于 i=3: queries[i]=1, P=[2,1,3,4,5], 1 在 P 中的位置是 1，接着我们把 1 移动到 P␣
```

(续下页)

(接上页)

```
↪的起始位置，得到 P=[1,2,3,4,5] 。
因此，返回的结果数组为 [2,1,2,1] 。
示例 2：输入：queries = [4,1,2,2], m = 4 输出：[3,1,2,0]
示例 3：输入：queries = [7,5,5,8,3], m = 8 输出：[6,5,0,7,5]
提示：1 <= m <= 10^3
1 <= queries.length <= m
1 <= queries[i] <= m
```

- 解题思路

```
func processQueries(queries []int, m int) []int {
        n := len(queries)
        res := make([]int, n)
        arr := make([]int, m)
        for i := 0; i < m; i++ {
                arr[i] = i + 1
        }
        for i := 0; i < n; i++ {
                index := 0
                for j := 0; j < m; j++ {
                        if arr[j] == queries[i] {
                                index = j
                                break
                        }
                }
                res[i] = index
                arr = append(append([]int{arr[index]}, arr[:index]...), arr[index+1:].
↪..)
        }
        return res
}

# 2
func processQueries(queries []int, m int) []int {
        n := len(queries)
        res := make([]int, n)
        arr := make([]int, m)
        for i := 0; i < m; i++ {
                arr[i] = i + 1
        }
        for i := 0; i < n; i++ {
                index := 0
                for j := 0; j < m; j++ {
                        if arr[j] == queries[i] {
```

(续下页)

(接上页)

```
                                index = j
                                // 交换位置
                                for k := 0; k < index; k++ {
                                        arr[k], arr[index] = arr[index], arr[k]
                                }
                                break
                        }
                }
                res[i] = index
        }
        return res
}

# 3
func processQueries(queries []int, m int) []int {
        n := len(queries)
        res := make([]int, n)
        arr := make([]int, m+1) // 存放下标
        for i := 1; i <= m; i++ {
                arr[i] = i - 1
        }
        for i := 0; i < n; i++ {
                x := queries[i]
                index := arr[x]
                res[i] = index
                for j := 1; j <= m; j++ {
                        if arr[j] < index { // 下标小于目标值
                                arr[j]++ // 前面的后移
                        }
                }
                arr[x] = 0 // 移到第一位
        }
        return res
}

# 4
func processQueries(queries []int, m int) []int {
        n := len(queries)
        res := make([]int, n)
        // 使用树状数组
        c = make([]int, n+m+1)
        length = n + m
        arr := make([]int, m+1)
```

(续下页)

(接上页)

```
        for i := 1; i <= m; i++ {
                arr[i] = n + i
                upData(n+i, 1) // 数组长度n+m+1, 从后面n开始存
        }

        for i := 0; i < n; i++ {
                cur := arr[queries[i]]
                upData(cur, -1)      // cur位置-1
                res[i] = getSum(cur) // cur之前的个数
                cur = n - i
                arr[queries[i]] = cur // 移到到n-i, 每次往前移动1位
                upData(cur, 1)        // cur位置+1
        }
        return res
}

var length int
var c []int // 树状数组

func lowBit(x int) int {
        return x & (-x)
}

// 单点修改
func upData(i, k int) { // 在i位置加上k
        for i <= length {
                c[i] = c[i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(i int) int {
        res := 0
        for i > 0 {
                res = res + c[i]
                i = i - lowBit(i)
        }
        return res
}
```

44.5 1410.HTML 实体解析器 (3)

- 题目

```
「HTML实体解析器」 是一种特殊的解析器，它将 HTML 代码作为输入，
并用字符本身替换掉所有这些特殊的字符实体。
HTML 里这些特殊字符和它们对应的字符实体包括：
双引号：字符实体为&quot;，对应的字符是"。
单引号：字符实体为&apos;，对应的字符是'。
与符号：字符实体为&amp;，对应对的字符是&。
大于号：字符实体为&gt;，对应的字符是>。
小于号：字符实体为&lt;，对应的字符是<。
斜线号：字符实体为&frasl;，对应的字符是/。
给你输入字符串text，请你实现一个 HTML实体解析器，返回解析器解析后的结果。
示例 1：输入：text = "&amp; is an HTML entity but &ambassador; is not."
输出："& is an HTML entity but &ambassador; is not."
解释：解析器把字符实体 &amp; 用 & 替换
示例2：输入：text = "and I quote: &quot;...&quot;"
输出："and I quote: \"...\""
示例 3：输入：text = "Stay home! Practice on Leetcode :)"
输出："Stay home! Practice on Leetcode :)"
示例 4：输入：text = "x &gt; y &amp;&amp; x &lt; y is always false"
输出："x > y && x < y is always false"
示例 5：输入：text = "leetcode.com&frasl;problemset&frasl;all"
输出："leetcode.com/problemset/all"
提示：1 <= text.length <= 10^5
字符串可能包含 256 个ASCII 字符中的任意字符。
```

- 解题思路

```
func entityParser(text string) string {
        text = html.UnescapeString(text)
        return strings.ReplaceAll(text, "⁄", "/")
}

# 2
func entityParser(text string) string {
        text = strings.ReplaceAll(text, "&quot;", "\"")
        text = strings.ReplaceAll(text, "&apos;", "'")
        text = strings.ReplaceAll(text, "&gt;", ">")
        text = strings.ReplaceAll(text, "&lt;", "<")
        text = strings.ReplaceAll(text, "&frasl;", "/")
        text = strings.ReplaceAll(text, "&amp;", "&")
        return text
```

(续下页)

(接上页)

```
}

# 3
func entityParser(text string) string {
        res := make([]byte, 0)
        temp := make([]byte, 0)
        for i := 0; i < len(text); i++ {
                if text[i] == '&' {
                        if len(temp) == 0 {
                                temp = append(temp, text[i])
                        } else {
                                res = append(res, temp...)
                                temp = make([]byte, 0)
                                temp = append(temp, text[i])
                        }
                } else if text[i] == ';' {
                        temp = append(temp, text[i])
                        switch string(temp) {
                        case "&gt;":
                                res = append(res, '>')
                        case "&lt;":
                                res = append(res, '<')
                        case "&quot;":
                                res = append(res, '"')
                        case "&apos;":
                                res = append(res, '\'')
                        case "&frasl;":
                                res = append(res, '/')
                        case "&amp;":
                                res = append(res, '&')
                        default:
                                res = append(res, temp...)
                        }
                        temp = make([]byte, 0)
                } else if len(temp) == 0 {
                        res = append(res, text[i])
                } else {
                        temp = append(temp, text[i])
                }
        }
        if len(temp) > 0 {
                res = append(res, temp...)
        }
```

(续下页)

(接上页)

```
        return string(res)
}
```

44.6 1414. 和为 K 的最少斐波那契数字数目 (2)

- 题目

```
给你数字␣
↪k，请你返回和为k的斐波那契数字的最少数目，其中，每个斐波那契数字都可以被使用多次。
斐波那契数字定义为：
F1 = 1
F2 = 1
Fn = Fn-1 + Fn-2，  其中 n > 2 。
数据保证对于给定的 k，一定能找到可行解。
示例 1：输入：k = 7 输出：2
解释：斐波那契数字为：1，1，2，3，5，8，13，……
对于 k = 7 ，我们可以得到 2 + 5 = 7 。
示例 2：输入：k = 10 输出：2
解释：对于 k = 10 ，我们可以得到 2 + 8 = 10 。
示例 3：输入：k = 19 输出：3
解释：对于 k = 19 ，我们可以得到 1 + 5 + 13 = 19 。
提示：1 <= k <= 10^9
```

- 解题思路

```
func findMinFibonacciNumbers(k int) int {
        arr := make([]int, 2)
        arr[0] = 1
        arr[1] = 1
        for arr[len(arr)-2]+arr[len(arr)-1] <= k {
                arr = append(arr, arr[len(arr)-2]+arr[len(arr)-1])
        }
        res := 0
        for i := len(arr) - 1; i >= 0; i-- {
                for arr[i] <= k {
                        k = k - arr[i]
                        res++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func findMinFibonacciNumbers(k int) int {
        res := 0
        for {
                target := get(k)
                k = k - target
                res++
                if k == 0 {
                        break
                }
        }
        return res
}

func get(k int) int {
        a, b := 1, 1
        for {
                a, b = b, a+b
                if b > k {
                        return a
                } else if b == k {
                        return b
                }
        }
}
```

44.7 1415. 长度为 n 的开心字符串中字典序第 k 小的字符串 (2)

- 题目

```
一个 「开心字符串」定义为：
仅包含小写字母['a', 'b', 'c'].
对所有在1到s.length - 1之间的i，满足s[i] != s[i + 1]（字符串的下标从 1 开始）。
比方说，字符串"abc"，"ac"，"b" 和"abcbabcbcb"都是开心字符串，
但是"aa"，"baa"和"ababbc"都不是开心字符串。
给你两个整数 n和 k，你需要将长度为 n的所有开心字符串按字典序排序。
请你返回排序后的第 k 个开心字符串，如果长度为 n的开心字符串少于 k个，那么请你返回␣
↪空字符串。
示例 1：输入：n = 1，k = 3 输出："c"
解释：列表 ["a"，"b"，"c"] 包含了所有长度为 1␣
↪的开心字符串。按照字典序排序后第三个字符串为 "c" 。
示例 2：输入：n = 1，k = 4 输出：""
```

(续下页)

(接上页)

```
解释：长度为 1 的开心字符串只有 3 个。
示例 3：输入：n = 3, k = 9 输出："cab"
解释：长度为 3 的开心字符串总共有 12 个 ["aba", "abc", "aca", "acb", "bab", "bac",
"bca", "bcb", "cab", "cac", "cba", "cbc"] 。
第 9 个字符串为 "cab"
示例 4：输入：n = 2, k = 7 输出：""
示例 5：输入：n = 10, k = 100 输出："abacbabacb"
提示：1 <= n <= 10
1 <= k <= 100
```

- 解题思路

```
var arr []string

func getHappyString(n int, k int) string {
        arr = make([]string, 0)
        dfs(n, "")
        if len(arr) < k {
                return ""
        }
        return arr[k-1]
}

func dfs(n int, str string) {
        if len(str) > 1 && str[len(str)-1] == str[len(str)-2] {
                return
        }
        if len(str) == n {
                arr = append(arr, str)
                return
        }
        for i := 0; i < 3; i++ {
                char := 'a' + i
                dfs(n, str+string(char))
        }
}

# 2
func getHappyString(n int, k int) string {
        per := 1 << (n - 1)
        if k > 3*per {
                return ""
        }
        res := make([]byte, n)
```

(续下页)

(接上页)

```
        next := [][]int{{1, 2}, {0, 2}, {0, 1}} // 3个分支
        k = k - 1
        var status int
        for i := 0; i < n; i++ {
                if i == 0 { // 确定第1个分支
                        status = k / per
                } else {
                        per = per / 2
                        status = next[status][k/per] // 确定后面的分支
                }
                k = k % per
                res[i] = byte(status + 'a')
        }
        return string(res)
}
```

44.8 1418. 点菜展示表 (1)

- 题目

```
给你一个数组 orders，表示客户在餐厅中完成的订单，确切地说，
orders[i]=[customerNamei,tableNumberi,foodItemi] ，其中 customerNamei 是客户的姓名，
tableNumberi 是客户所在餐桌的桌号，而 foodItemi 是客户点的餐品名称。
请你返回该餐厅的 点菜展示表 。在这张表中，表中第一行为标题，其第一列为餐桌桌号␣
↪“Table” ，
后面每一列都是按字母顺序排列的餐品名称。
接下来每一行中的项则表示每张餐桌订购的相应餐品数量，第一列应当填对应的桌号，后面依次填写下单的餐品数
注意：客户姓名不是点菜展示表的一部分。此外，表中的数据行应该按餐桌桌号升序排列。
示例 1：
输入：orders = [["David","3","Ceviche"],["Corina","10","Beef Burrito"],
["David","3","Fried Chicken"],["Carla","5","Water"],
["Carla","5","Ceviche"],["Rous","3","Ceviche"]]
输出：[["Table","Beef Burrito","Ceviche","Fried Chicken","Water"],
["3","0","2","1","0"],["5","0","1","0","1"],["10","1","0","0","0"]]
解释：点菜展示表如下所示：
Table,Beef Burrito,Ceviche,Fried Chicken,Water
3    ,0           ,2      ,1            ,0
5    ,0           ,1      ,0            ,1
10   ,1           ,0      ,0            ,0
对于餐桌 3：David 点了 "Ceviche" 和 "Fried Chicken"，而 Rous 点了 "Ceviche"
而餐桌 5：Carla 点了 "Water" 和 "Ceviche"
餐桌 10：Corina 点了 "Beef Burrito"
```

(续下页)

(接上页)

```
示例 2: 输入: orders = [["James","12","Fried Chicken"],["Ratesh","12","Fried Chicken
↪"],
["Amadeus","12","Fried Chicken"],["Adam","1","Canadian Waffles"],
["Brianna","1","Canadian Waffles"]]
输出: [["Table","Canadian Waffles","Fried Chicken"],["1","2","0"],["12","0","3"]]
解释: 对于餐桌 1: Adam 和 Brianna 都点了 "Canadian Waffles"
而餐桌 12: James, Ratesh 和 Amadeus 都点了 "Fried Chicken"
示例 3:
输入: orders = [["Laura","2","Bean Burrito"],["Jhon","2","Beef Burrito"],["Melissa","2
↪","Soda"]]
输出: [["Table","Bean Burrito","Beef Burrito","Soda"],["2","1","1","1"]]
提示:
    1 <= orders.length <= 5 * 10^4
    orders[i].length == 3
    1 <= customerNamei.length, foodItemi.length <= 20
    customerNamei 和 foodItemi 由大小写英文字母及空格字符 ' ' 组成。
    tableNumberi 是 1 到 500 范围内的整数。
```

- 解题思路

```
func displayTable(orders [][]string) [][]string {
        res := make([][]string, 0)
        titles := make([]string, 0)
        idArr := make([]int, 0)
        m := make(map[string]bool)
        m2 := make(map[string]map[string]int)
        for i := 0; i < len(orders); i++ {
                m[orders[i][2]] = true
                if m2[orders[i][1]] == nil {
                        m2[orders[i][1]] = make(map[string]int)
                }
                m2[orders[i][1]][orders[i][2]]++
        }
        for k := range m {
                titles = append(titles, k)
        }
        for k := range m2 {
                tableID, _ := strconv.Atoi(k)
                idArr = append(idArr, tableID)
        }
        sort.Strings(titles)
        sort.Ints(idArr)
        res = append(res, append([]string{"Table"}, titles...))
        for i := 0; i < len(idArr); i++ {
```

(续下页)

(接上页)

```
                tableStr := strconv.Itoa(idArr[i])
                temp := make([]string, 0)
                temp = append(temp, tableStr)
                for j := 0; j < len(titles); j++ {
                        temp = append(temp, strconv.Itoa(m2[tableStr][titles[j]]))
                }
                res = append(res, temp)
        }
        return res
}
```

44.9 1419. 数青蛙 (2)

- 题目

```
给你一个字符串 croakOfFrogs，它表示不同青蛙发出的蛙鸣声（字符串 "croak" ）的组合。
由于同一时间可以有多只青蛙呱呱作响，所以 croakOfFrogs 中会混合多个 “croak” 。
请你返回模拟字符串中所有蛙鸣所需不同青蛙的最少数目。
注意：要想发出蛙鸣 "croak"，青蛙必须 依序 输出 ‘c’, ’r’, ’o’, ’a’, ’k’ 这 5␣
↪个字母。
如果没有输出全部五个字母，那么它就不会发出声音。
如果字符串 croakOfFrogs 不是由若干有效的 "croak" 字符混合而成，请返回 -1 。
示例 1：输入：croakOfFrogs = "croakcroak" 输出：1
解释：一只青蛙 “呱呱” 两次
示例 2：输入：croakOfFrogs = "crcoakroak" 输出：2
解释：最少需要两只青蛙，“呱呱” 声用黑体标注
第一只青蛙 "crcoakroak"
第二只青蛙 "crcoakroak"
示例 3：输入：croakOfFrogs = "croakcrook" 输出：-1
解释：给出的字符串不是 "croak" 的有效组合。
示例 4：输入：croakOfFrogs = "croakcroa" 输出：-1
提示：1 <= croakOfFrogs.length <= 10^5
    字符串中的字符只有 'c', 'r', 'o', 'a' 或者 'k'
```

- 解题思路

```
func minNumberOfFrogs(croakOfFrogs string) int {
        res := 0
        var c, r, o, a, k int
        temp := 0
        for _, char := range croakOfFrogs {
                if char == 'c' {
```

(续下页)

(接上页)

```
                        c++
                        if temp > 0 {
                                temp-- // 有空闲的青蛙
                        } else {
                                res++ // 没有空闲的青蛙
                        }
                } else if r < c && char == 'r' {
                        r++
                } else if o < r && char == 'o' {
                        o++
                } else if a < o && char == 'a' {
                        a++
                } else if k < a && char == 'k' {
                        k++
                        temp++ // 结束有空闲
                } else {
                        return -1
                }
        }
        if temp != res { // 避免出现"croakcroa"的情况
                return -1
        }
        return res
}

# 2
func minNumberOfFrogs(croakOfFrogs string) int {
        res := 0
        var c, r, o, a, k int
        for _, char := range croakOfFrogs {
                if char == 'c' {
                        c++
                } else if char == 'r' {
                        r++
                } else if char == 'o' {
                        o++
                } else if char == 'a' {
                        a++
                } else if char == 'k' {
                        k++
                } else {
                        return -1
                }
```

(续下页)

(接上页)

```
                res = max(res, c)
                if c < r || r < o || o < a || a < k {
                        return -1
                }
                if k == 1 {
                        c--
                        r--
                        o--
                        a--
                        k--
                }
        }
        if c == 0 && r == 0 && o == 0 && a == 0 && k == 0 {
                return res
        }
        return -1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

44.10 1423. 可获得的最大点数 (2)

- 题目

```
几张卡牌 排成一行，每张卡牌都有一个对应的点数。点数由整数数组 cardPoints 给出。
每次行动，你可以从行的开头或者末尾拿一张卡牌，最终你必须正好拿 k 张卡牌。
你的点数就是你拿到手中的所有卡牌的点数之和。
给你一个整数数组 cardPoints 和整数 k，请你返回可以获得的最大点数。
示例 1：输入：cardPoints = [1,2,3,4,5,6,1], k = 3 输出：12
解释：第一次行动，不管拿哪张牌，你的点数总是 1␣
↪。但是，先拿最右边的卡牌将会最大化你的可获得点数。
最优策略是拿右边的三张牌，最终点数为 1 + 6 + 5 = 12 。
示例 2：输入：cardPoints = [2,2,2], k = 2 输出：4
解释：无论你拿起哪两张卡牌，可获得的点数总是 4 。
示例 3：输入：cardPoints = [9,7,7,9,7,7,9], k = 7 输出：55
解释：你必须拿起所有卡牌，可以获得的点数为所有卡牌的点数之和。
示例 4：输入：cardPoints = [1,1000,1], k = 1 输出：1
```

(续下页)

(接上页)

```
解释：你无法拿到中间那张卡牌，所以可以获得的最大点数为 1 。
示例 5：输入：cardPoints = [1,79,80,1,1,1,200,1], k = 3 输出：202
提示：1 <= cardPoints.length <= 10^5
    1 <= cardPoints[i] <= 10^4
    1 <= k <= cardPoints.length
```

- 解题思路

```
func maxScore(cardPoints []int, k int) int {
        res := 0
        left := 0
        for i := 0; i < k; i++ {
                left = left + cardPoints[i]
        }
        res = left
        right := 0
        count := k
        for i := len(cardPoints) - 1; i >= len(cardPoints)-k; i-- {
                right = right + cardPoints[i]
                left = left - cardPoints[count-1]
                res = max(res, left+right)
                count--
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxScore(cardPoints []int, k int) int {
        res := 0
        n := len(cardPoints)
        window := 0
        sum := 0
        for i := 0; i < n-k; i++ {
                sum = sum + cardPoints[i]
                window = window + cardPoints[i]
        }
        res = window
```

(续下页)

(接上页)

```
        count := 0
        for i := n - k; i < n; i++ {
                sum = sum + cardPoints[i]
                window = window + cardPoints[i] - cardPoints[count]
                res = min(res, window)
                count++
        }
        return sum - res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

44.11 1424. 对角线遍历 II(2)

- 题目

```
给你一个列表 nums ，里面每一个元素都是一个整数列表。
请你依照下面各图的规则，按顺序返回 nums 中对角线上的整数。
示例 1：输入：nums = [[1,2,3],[4,5,6],[7,8,9]] 输出：[1,4,2,7,5,3,8,6,9]
示例 2：输入：nums = [[1,2,3,4,5],[6,7],[8],[9,10,11],[12,13,14,15,16]]
输出：[1,6,2,8,7,3,9,4,12,10,5,13,11,14,15,16]
示例 3：输入：nums = [[1,2,3],[4],[5,6,7],[8],[9,10,11]] 输出：[1,4,2,5,3,8,6,9,7,10,
↪11]
示例 4：输入：nums = [[1,2,3,4,5,6]] 输出：[1,2,3,4,5,6]
提示：  1 <= nums.length <= 10^5
    1 <= nums[i].length <= 10^5
    1 <= nums[i][j] <= 10^9
    nums 中最多有 10^5 个数字。
```

- 解题思路

```
func findDiagonalOrder(nums [][]int) []int {
        arr := make([][2]int, 0)
        for i := 0; i < len(nums); i++ {
                for j := 0; j < len(nums[i]); j++ {
                        arr = append(arr, [2]int{i, j})
                }
```

(续下页)

(接上页)

```
        }
        sort.Slice(arr, func(i, j int) bool {
                a := arr[i][0] + arr[i][1]
                b := arr[j][0] + arr[j][1]
                if a == b {
                        return arr[i][1] < arr[j][1]
                }
                return a < b
        })
        res := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                a, b := arr[i][0], arr[i][1]
                res = append(res, nums[a][b])
        }
        return res
}

# 2
func findDiagonalOrder(nums [][]int) []int {
        maxValue := 0
        m := make(map[int][]int)
        for i := 0; i < len(nums); i++ {
                for j := 0; j < len(nums[i]); j++ {
                        m[i+j] = append(m[i+j], nums[i][j])
                        if i+j > maxValue {
                                maxValue = i + j
                        }
                }
        }
        res := make([]int, 0)
        for i := 0; i <= maxValue; i++ {
                for j := len(m[i]) - 1; j >= 0; j-- {
                        res = append(res, m[i][j])
                }
        }
        return res
}
```

44.12 1432. 改变一个整数能得到的最大差值 (2)

- 题目

```
给你一个整数num。你可以对它进行如下步骤恰好 两次：
选择一个数字x (0<= x <= 9).
选择另一个数字y (0<= y <= 9)。数字y可以等于x。
将 num中所有出现 x的数位都用 y替换。
得到的新的整数 不能有前导 0 ，得到的新整数也 不能是 0。
令两次对 num的操作得到的结果分别为a和b。
请你返回a 和b的 最大差值 。
示例 1：输入：num = 555 输出：888
解释：第一次选择 x = 5 且 y = 9 ，并把得到的新数字保存在 a 中。
第二次选择 x = 5 且 y = 1 ，并把得到的新数字保存在 b 中。
现在，我们有 a = 999 和 b = 111 ，最大差值为 888
示例 2：输入：num = 9 输出：8
解释：第一次选择 x = 9 且 y = 9 ，并把得到的新数字保存在 a 中。
第二次选择 x = 9 且 y = 1 ，并把得到的新数字保存在 b 中。
现在，我们有 a = 9 和 b = 1 ，最大差值为 8
示例 3：输入：num = 123456 输出：820000
示例 4： 输入：num = 10000 输出：80000
示例 5：输入：num = 9288 输出：8700
提示：1 <= num <= 10^8
```

- 解题思路

```
func maxDiff(num int) int {
        maxValue, minValue := num, num
        str := strconv.Itoa(num)
        for i := 0; i < len(str); i++ {
                if str[i] < '9' {
                        maxValue, _ = strconv.Atoi(strings.ReplaceAll(str,␣
↪string(str[i]), "9"))
                        break
                }
        }
        if str[0] > '1' {
                minValue, _ = strconv.Atoi(strings.ReplaceAll(str, string(str[0]), "1
↪"))
        } else {
                for i := 1; i < len(str); i++ {
                        if str[i] > '1' && str[0] != str[i] {
                                minValue, _ = strconv.Atoi(strings.ReplaceAll(str,␣
↪string(str[i]), "0"))
```

(续下页)

(接上页)

```
                                break
                        }
                }
        }
        return maxValue - minValue
}

# 2
func maxDiff(num int) int {
        maxValue, minValue := num, num
        str := strconv.Itoa(num)
        for x := 0; x < 10; x++ {
                for y := 0; y < 10; y++ {
                        newStr := strings.ReplaceAll(str, string('0'+x), string('0
↪'+y))
                        if newStr[0] == '0' {
                                continue
                        }
                        value, _ := strconv.Atoi(newStr)
                        if value > maxValue {
                                maxValue = value
                        }
                        if value < minValue {
                                minValue = value
                        }
                }
        }
        return maxValue - minValue
}
```

44.13 1433. 检查一个字符串是否可以打破另一个字符串 (2)

- 题目

```
给你两个字符串 s1 和 s2 ，它们长度相等，请你检查是否存在一个 s1 的排列可以打破 s2␣
↪的一个排列，
或者是否存在一个 s2 的排列可以打破 s1 的一个排列。
字符串 x 可以打破字符串 y （两者长度都为 n ）需满足对于所有 i（在 0 到 n - 1 之间）
都有 x[i] >= y[i]（字典序意义下的顺序）。
示例 1：输入：s1 = "abc", s2 = "xya" 输出：true
解释："ayx" 是 s2="xya" 的一个排列，"abc" 是字符串 s1="abc" 的一个排列，且 "ayx"␣
↪可以打破 "abc" 。
```

(续下页)

(接上页)

```
示例 2: 输入: s1 = "abe", s2 = "acd" 输出: false
解释: s1="abe" 的所有排列包括: "abe", "aeb", "bae", "bea", "eab" 和 "eba" ,
s2="acd" 的所有排列包括: "acd", "adc", "cad", "cda", "dac" 和 "dca"。
然而没有任何 s1 的排列可以打破 s2 的排列。也没有 s2 的排列能打破 s1 的排列。
示例 3: 输入: s1 = "leetcodee", s2 = "interview" 输出: true
提示:  s1.length == n
    s2.length == n
    1 <= n <= 10^5
    所有字符串都只包含小写英文字母。
```

- 解题思路

```
func checkIfCanBreak(s1 string, s2 string) bool {
        arr1 := []byte(s1)
        sort.Slice(arr1, func(i, j int) bool {
                return arr1[i] < arr1[j]
        })
        arr2 := []byte(s2)
        sort.Slice(arr2, func(i, j int) bool {
                return arr2[i] < arr2[j]
        })
        s1 = string(arr1)
        s2 = string(arr2)
        return compare(s1, s2) || compare(s2, s1)
}

func compare(s1 string, s2 string) bool {
        for i := 0; i < len(s1); i++ {
                if s1[i] < s2[i] {
                        return false
                }
        }
        return true
}

# 2
func checkIfCanBreak(s1 string, s2 string) bool {
        arr1 := [26]int{}
        arr2 := [26]int{}
        for i := 0; i < len(s1); i++ {
                arr1[int(s1[i]-'a')]++
                arr2[int(s2[i]-'a')]++
        }
        a, b := 0, 0
```

(续下页)

(接上页)

```
        totalA, totalB := 0, 0
        for i := 0; i < 26; i++ {
                totalA = totalA + arr1[i]
                totalB = totalB + arr2[i]
                if totalA >= totalB {
                        a++
                }
                if totalB >= totalA {
                        b++
                }
        }
        return a == 26 || b == 26
}
```

44.14 1437. 是否所有 1 都至少相隔 k 个元素 (2)

- 题目

```
给你一个由若干 0 和 1 组成的数组 nums 以及整数 k。如果所有 1 都至少相隔 k␣
↪个元素，则返回 True ；
否则，返回 False 。
示例 1：输入：nums = [1,0,0,0,1,0,0,1], k = 2 输出：true
解释：每个 1 都至少相隔 2 个元素。
示例 2：输入：nums = [1,0,0,1,0,1], k = 2 输出：false
解释：第二个 1 和第三个 1 之间只隔了 1 个元素。
示例 3：输入：nums = [1,1,1,1,1], k = 0 输出：true
示例 4：输入：nums = [0,1,0,1], k = 1 输出：true
提示：1 <= nums.length <= 10^5
    0 <= k <= nums.length
    nums[i] 的值为 0 或 1
```

- 解题思路

```
func kLengthApart(nums []int, k int) bool {
        last := -(k + 1) // 兼容第0个元素是1
        for i := 0; i < len(nums); i++ {
                if nums[i] == 1 {
                        if i-last <= k {
                                return false
                        }
                        last = i
                }
```

(续下页)

(接上页)

```
        }
        return true
}

# 2
func kLengthApart(nums []int, k int) bool {
        last := -1
        for i := 0; i < len(nums); i++ {
                if nums[i] == 1 {
                        if last != -1 && i-last <= k {
                                return false
                        }
                        last = i
                }
        }
        return true
}
```

44.15 1438. 绝对差不超过限制的最长连续子数组 (3)

- 题目

```
给你一个整数数组 nums ，和一个表示限制的整数 limit，
请你返回最长连续子数组的长度，该子数组中的任意两个元素之间的绝对差必须小于或者等于␣
↪limit 。
如果不存在满足条件的子数组，则返回 0 。
示例 1：输入：nums = [8,2,4,7], limit = 4 输出：2
解释：所有子数组如下：
[8] 最大绝对差 |8-8| = 0 <= 4.
[8,2] 最大绝对差 |8-2| = 6 > 4.
[8,2,4] 最大绝对差 |8-2| = 6 > 4.
[8,2,4,7] 最大绝对差 |8-2| = 6 > 4.
[2] 最大绝对差 |2-2| = 0 <= 4.
[2,4] 最大绝对差 |2-4| = 2 <= 4.
[2,4,7] 最大绝对差 |2-7| = 5 > 4.
[4] 最大绝对差 |4-4| = 0 <= 4.
[4,7] 最大绝对差 |4-7| = 3 <= 4.
[7] 最大绝对差 |7-7| = 0 <= 4.
因此，满足题意的最长子数组的长度为 2 。
示例 2：输入：nums = [10,1,2,4,7,2], limit = 5 输出：4
解释：满足题意的最长子数组是 [2,4,7,2]，其最大绝对差 |2-7| = 5 <= 5 。
示例 3：输入：nums = [4,2,2,2,4,4,2,2], limit = 0 输出：3
```

(续下页)

(接上页)

```
提示：1 <= nums.length <= 10^5
1 <= nums[i] <= 10^9
0 <= limit <= 10^9
```

- 解题思路

```
func longestSubarray(nums []int, limit int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                maxValue, minValue := nums[i], nums[i]
                for j := i; j < len(nums); j++ {
                        maxValue = max(maxValue, nums[j])
                        minValue = min(minValue, nums[j])
                        if maxValue-minValue <= limit {
                                res = max(res, j-i+1)
                        } else {
                                break
                        }
                }
                if res >= len(nums)-i {
                        break
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func longestSubarray(nums []int, limit int) int {
        res := 0
        maxStack, minStack := make([]int, 0), make([]int, 0)
```

(续下页)

(接上页)

```
        var i int
        for j := 0; j < len(nums); j++ {
                for len(minStack) > 0 && minStack[len(minStack)-1] > nums[j] {
                        minStack = minStack[:len(minStack)-1]
                }
                minStack = append(minStack, nums[j])
                for len(maxStack) > 0 && maxStack[len(maxStack)-1] < nums[j] {
                        maxStack = maxStack[:len(maxStack)-1]
                }
                maxStack = append(maxStack, nums[j])
                for maxStack[0]-minStack[0] > limit { // 移除左边
                        if nums[i] == maxStack[0] {
                                maxStack = maxStack[1:]
                        }
                        if nums[i] == minStack[0] {
                                minStack = minStack[1:]
                        }
                        i++
                }
                res = max(res, j-i+1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
import (
        "math/rand"
)

func longestSubarray(nums []int, limit int) int {
        res := 0
        t := &treap{}
        var i int
        for j := 0; j < len(nums); j++ {
                t.insert(nums[j])
                for t.max().key-t.min().key > limit {
```

(续下页)

(接上页)

```
                        t.delete(nums[i])
                        i++
                }
                res = max(res, j-i+1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type Node struct {
        ch       [2]*Node
        priority int
        key      int
        cnt      int
}

func (n *Node) cmp(b int) int {
        switch {
        case b < n.key:
                return 0
        case b > n.key:
                return 1
        default:
                return -1
        }
}

func (n *Node) rotate(b int) *Node {
        target := n.ch[b^1]
        n.ch[b^1] = target.ch[b]
        target.ch[b] = n
        return target
}

type treap struct {
        root *Node
}
```

(续下页)

(接上页)

```
func (t *treap) ins(n *Node, key int) *Node {
        if n == nil {
                return &Node{
                        ch:       [2]*Node{}, // 0左边，1右边
                        priority: rand.Int(),
                        key:      key,
                        cnt:      1,
                }
        }
        if b := n.cmp(key); b >= 0 {
                n.ch[b] = t.ins(n.ch[b], key)
                if n.ch[b].priority > n.priority {
                        n = n.rotate(b ^ 1)
                }
        } else {
                n.cnt++
        }
        return n
}

func (t *treap) del(n *Node, key int) *Node {
        if n == nil {
                return nil
        }
        if b := n.cmp(key); b >= 0 {
                n.ch[b] = t.del(n.ch[b], key)
        } else {
                if n.cnt > 1 {
                        n.cnt--
                } else {
                        if n.ch[1] == nil {
                                return n.ch[0]
                        }
                        if n.ch[0] == nil {
                                return n.ch[1]
                        }
                        b = 0
                        if n.ch[0].priority > n.ch[1].priority {
                                b = 1
                        }
                        n = n.rotate(b)
                        n.ch[b] = t.del(n.ch[b], key)
```

(续下页)

(接上页)

```
                }
        }
        return n
}

func (t *treap) insert(key int) {
        t.root = t.ins(t.root, key)
}

func (t *treap) delete(key int) {
        t.root = t.del(t.root, key)
}

func (t *treap) min() (min *Node) {
        for temp := t.root; temp != nil; temp = temp.ch[0] {
                min = temp
        }
        return min
}

func (t *treap) max() (max *Node) {
        for temp := t.root; temp != nil; temp = temp.ch[1] {
                max = temp
        }
        return max
}
```

44.16 1442. 形成两个异或相等数组的三元组数目 (3)

- 题目

```
给你一个整数数组 arr 。
现需要从数组中取三个下标 i、j 和 k ，其中 (0 <= i < j <= k < arr.length) 。
a 和 b 定义如下：
    a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]
    b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]
注意：^ 表示 按位异或 操作。
请返回能够令 a == b 成立的三元组 (i, j , k) 的数目。
示例 1：输入：arr = [2,3,1,6,7] 输出：4
解释：满足题意的三元组分别是 (0,1,2), (0,2,2), (2,3,4) 以及 (2,4,4)
示例 2：输入：arr = [1,1,1,1,1] 输出：10
示例 3：输入：arr = [2,3] 输出：0
```

(续下页)

(接上页)

```
示例 4: 输入: arr = [1,3,5,7,9] 输出: 3
示例 5: 输入: arr = [7,11,12,9,5,2,7,17,22] 输出: 8
提示:
    1 <= arr.length <= 300
    1 <= arr[i] <= 10^8
```

- 解题思路

```
func countTriplets(arr []int) int {
        res := 0
        for i := 1; i < len(arr); i++ {
                arr[i] = arr[i] ^ arr[i-1]
        }
        for i := 0; i < len(arr)-1; i++ {
                for j := i + 1; j < len(arr); j++ {
                        for k := j; k < len(arr); k++ {
                                var a, b int
                                if i == 0 {
                                        a = arr[j-1]
                                } else {
                                        a = arr[j-1] ^ arr[i-1]
                                }
                                b = arr[k] ^ arr[j-1]
                                if a == b {
                                        res++
                                }
                        }
                }
        }
        return res
}

#
// a[i]^...a[j-1]^a[j]^...a[k] = 0, 则j可以取i+1、i+2、...、..k
func countTriplets(arr []int) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                temp := arr[i]
                for k := i + 1; k < len(arr); k++ {
                        temp = temp ^ arr[k]
                        if temp == 0 {
                                res = res + k - i
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

#
func countTriplets(arr []int) int {
        res := 0
        sumM := make(map[int]int)
        countM := make(map[int]int)
        countM[0] = 1
        temp := 0
        for i := 0; i < len(arr); i++ {
                temp = temp ^ arr[i]
                if countM[temp] > 0 {
                        // 相同异或结果，分别出现在下标[a,b,c,d]
                        // 则[a,d]有d-a-1个满足条件的
                        // sum = (d-a-1)+(d-b-1)+(d-c-1)
                        // ==> nd - [(a+1) + (b+1) + (c+1)]
                        // 同理得[a,b], [a,c]
                        res = res + i*countM[temp] - sumM[temp]
                }
                countM[temp]++
                sumM[temp] = sumM[temp] + (i + 1)
        }
        return res
}
```

44.17 1443. 收集树上所有苹果的最少时间 (2)

- 题目

```
给你一棵有n个节点的无向树，节点编号为0到n-1，它们中有一些节点有苹果。
通过树上的一条边，需要花费 1 秒钟。
你从节点0出发，请你返回最少需要多少秒，可以收集到所有苹果，并回到节点 0 。
无向树的边由edges给出，其中edges[i] = [fromi, toi]，表示有一条边连接from和toi 。
除此以外，还有一个布尔数组hasApple ，其中hasApple[i] = true代表节点i有一个苹果，
否则，节点i没有苹果。
示例 1：输入：n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]],
hasApple = [false,false,true,false,true,true,false] 输出：8
解释：上图展示了给定的树，其中红色节点表示有苹果。一个能收集到所有苹果的最优方案由绿色箭头表示。
示例 2：输入：n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]],
hasApple = [false,false,true,false,false,true,false] 输出：6
```

(续下页)

(接上页)

```
解释：上图展示了给定的树，其中红色节点表示有苹果。一个能收集到所有苹果的最优方案由绿色箭头表示。
示例 3：输入：n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]],
hasApple = [false,false,false,false,false,false,false] 输出：0
提示：1 <= n <= 10^5
edges.length == n-1
edges[i].length == 2
0 <= fromi, toi <= n-1
fromi< toi
hasApple.length == n
```

- 解题思路

```
func minTime(n int, edges [][]int, hasApple []bool) int {
        arr := make([][]int, n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        visited := make([]bool, n)
        res, _ := dfs(arr, hasApple, visited, 0)
        if res >= 2 {
                return res - 2 // 遍历N个点，长度：2N-2
        }
        return 0
}

func dfs(arr [][]int, hasApple, visited []bool, index int) (int, bool) {
        visited[index] = true
        res := 0
        has := false
        if hasApple[index] == true {
                has = true
        }
        for i := 0; i < len(arr[index]); i++ {
                next := arr[index][i]
                if visited[next] == true {
                        continue
                }
                total, isExist := dfs(arr, hasApple, visited, next)
                if isExist {
                        has = true
                        res = res + total
                }
```

(续下页)

(接上页)

```
        }
        if has == true {
                return res + 2, true
        }
        return 0, false
}

# 2
func minTime(n int, edges [][]int, hasApple []bool) int {
        ans := 0
        m := make([]bool, n)
        m[0] = true
        for i := 0; i < len(edges); i++ {
                from, to := edges[i][0], edges[i][1]
                if m[from] {
                        m[to] = true
                } else {
                        m[from] = true
                        // 改变数据
                        // [[0 2] [0 3] [1 2]] => [[0 2] [0 3] [2 1]]
                        edges[i][0], edges[i][1] = edges[i][1], edges[i][0]
                }
        }
        for i := len(edges) - 1; i >= 0; i-- {
                from, to := edges[i][0], edges[i][1]
                if hasApple[to] {
                        hasApple[from] = true
                        ans += 2
                }
        }
        return ans
}
```

44.18 1447. 最简分数 (2)

- 题目

```
给你一个整数 n ，请你返回所有 0 到 1 之间（不包括 0 和 1）满足分母小于等于 n 的 最简␣
↪分数 。
分数可以以 任意 顺序返回。
示例 1：输入：n = 2 输出：["1/2"]
解释："1/2" 是唯一一个分母小于等于 2 的最简分数。
```

(续下页)

(接上页)

```
示例 2: 输入: n = 3 输出: ["1/2","1/3","2/3"]
示例 3: 输入: n = 4 输出: ["1/2","1/3","1/4","2/3","3/4"]
解释: "2/4" 不是最简分数, 因为它可以化简为 "1/2" 。
示例 4: 输入: n = 1 输出: []
提示: 1 <= n <= 100
```

- 解题思路

```
func simplifiedFractions(n int) []string {
        res := make([]string, 0)
        for i := 2; i <= n; i++ {
                for j := 1; j < i; j++ {
                        if gcd(i, j) == 1 {
                                res = append(res, fmt.Sprintf("%d/%d", j, i))
                        }
                }
        }
        return res
}

func gcd(a, b int) int {
        if a%b == 0 {
                return b
        }
        return gcd(b, a%b)
}

# 2
func simplifiedFractions(n int) []string {
        res := make([]string, 0)
        m := make(map[string]bool)
        for i := 2; i <= n; i++ {
                for j := 1; j < i; j++ {
                        str := fmt.Sprintf("%d/%d", j, i)
                        if _, ok := m[str]; ok{
                                continue
                        }
                        res = append(res, str)
                        for k := 1; i *k <= n; k++{
                                m[fmt.Sprintf("%d/%d", j*k, i*k)] = true
                        }
                }
        }
        return res
```

(续下页)

(接上页)

```
}
```

44.19 1448. 统计二叉树中好节点的数目 (1)

- 题目

```
给你一棵根为root的二叉树，请你返回二叉树中好节点的数目。
「好节点」X 定义为：从根到该节点 X 所经过的节点中，没有任何节点的值大于 X 的值。
示例 1：输入：root = [3,1,4,3,null,1,5] 输出：4
解释：图中蓝色节点为好节点。
根节点 (3) 永远是个好节点。
节点 4 -> (3,4) 是路径中的最大值。
节点 5 -> (3,4,5) 是路径中的最大值。
节点 3 -> (3,1,3) 是路径中的最大值。
示例 2：输入：root = [3,3,null,4,2] 输出：3
解释：节点 2 -> (3, 3, 2) 不是好节点，因为 "3" 比它大。
示例 3：输入：root = [1] 输出：1
解释：根节点是好节点。
提示：二叉树中节点数目范围是[1, 10^5]。
每个节点权值的范围是[-10^4, 10^4]。
```

- 解题思路

```
func goodNodes(root *TreeNode) int {
        maxValue := math.MinInt32
        return dfs(root, maxValue)
}

func dfs(root *TreeNode, maxValue int) int {
        if root == nil {
                return 0
        }
        if root.Val >= maxValue {
                return dfs(root.Left, root.Val) + dfs(root.Right, root.Val) + 1
        }
        return dfs(root.Left, maxValue) + dfs(root.Right, maxValue)
}
```

44.20 1451. 重新排列句子中的单词 (1)

- 题目

```
「句子」是一个用空格分隔单词的字符串。给你一个满足下述格式的句子 text :
    句子的首字母大写
    text 中的每个单词都用单个空格分隔。
请你重新排列 text 中的单词，使所有单词按其长度的升序排列。
如果两个单词的长度相同，则保留其在原句子中的相对顺序。
请同样按上述格式返回新的句子。
示例 1：输入：text = "Leetcode is cool" 输出："Is cool leetcode"
解释：句子中共有 3 个单词，长度为 8 的 "Leetcode" ，长度为 2 的 "is" 以及长度为 4 的
↪"cool" 。
输出需要按单词的长度升序排列，新句子中的第一个单词首字母需要大写。
示例 2：输入：text = "Keep calm and code on" 输出："On and keep calm code"
解释：输出的排序情况如下：
"On" 2 个字母。
"and" 3 个字母。
"keep" 4␣
↪个字母，因为存在长度相同的其他单词，所以它们之间需要保留在原句子中的相对顺序。
"calm" 4 个字母。
"code" 4 个字母。
示例 3：输入：text = "To be or not to be" 输出："To be or to be not"
提示：  text 以大写字母开头，然后包含若干小写字母以及单词间的单个空格。
    1 <= text.length <= 10^5
```

- 解题思路

```
func arrangeWords(text string) string {
        text = strings.ToLower(text)
        arr := strings.Fields(text)
        sort.SliceStable(arr, func(i, j int) bool {
                return len(arr[i]) < len(arr[j])
        })
        arr[0] = strings.Title(arr[0])
        return strings.Join(arr, " ")
}
```

44.21 1452. 收藏清单 (1)

- 题目

```
给你一个数组 favoriteCcompanies ，
其中 favoriteCcompanies[i] 是第 i 名用户收藏的公司清单（下标从 0 开始）。
请找出不是其他任何人收藏的公司清单的子集的收藏清单，并返回该清单下标。下标需要按升序排列。
示例 1：输入：favoriteCcompanies = [["leetcode","google","facebook"],
["google","microsoft"],["google","facebook"],["google"],["amazon"]]
输出：[0,1,4]
解释：favoriteCcompanies[2]=["google","facebook"] 是
favoriteCompanies[0]=["leetcode","google","facebook"] 的子集。
favoriteCompanies[3]=["google"] 是 favoriteCompanies[0]=
["leetcode","google","facebook"] 和 favoriteCompanies[1]=["google","microsoft"]␣
↪的子集。
其余的收藏清单均不是其他任何人收藏的公司清单的子集，因此，答案为 [0,1,4] 。
示例 2：输入：favoriteCompanies = [["leetcode","google","facebook"],
["leetcode","amazon"],["facebook","google"]]
输出：[0,1]
解释：favoriteCompanies[2]=["facebook","google"] 是 favoriteCompanies[0]=["leetcode",
↪"google","facebook"] 的子集，因此，答案为 [0,1] 。
示例 3：输入：favoriteCompanies = [["leetcode"],["google"],["facebook"],["amazon"]]
输出：[0,1,2,3]
提示：1 <=favoriteCompanies.length <= 100
1 <=favoriteCompanies[i].length <= 500
1 <=favoriteCompanies[i][j].length <= 20
favoriteCompanies[i] 中的所有字符串 各不相同 。
用户收藏的公司清单也 各不相同 ，
也就是说，即便我们按字母顺序排序每个清单， favoriteCompanies[i] !=␣
↪favoriteCompanies[j] 仍然成立。
所有字符串仅包含小写英文字母。
```

- 解题思路

```
type Node struct {
        index int
        str   []string
}

func peopleIndexes(favoriteCcompanies [][]string) []int {
        n := len(favoriteCcompanies)
        arr := make([]Node, 0)
        for i := 0; i < len(favoriteCompanies); i++ {
                arr = append(arr, Node{
```

(续下页)

(接上页)

```
                        index: i,
                        str:   favoriteCompanies[i],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return len(arr[i].str) < len(arr[j].str)
        })
        res := make([]int, 0)
        for i := 0; i < n; i++ {
                flag := true
                for j := i + 1; j < n; j++ {
                        if judge(arr[i].str, arr[j].str) == true {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res = append(res, arr[i].index)
                }
        }
        sort.Ints(res)
        return res
}

func judge(a, b []string) bool {
        m := make(map[string]bool)
        for i := 0; i < len(a); i++ {
                m[a[i]] = true
        }
        for i := 0; i < len(b); i++ {
                if _, ok := m[b[i]]; ok {
                        delete(m, b[i])
                }
        }
        return len(m) == 0
}
```

44.22 1456. 定长子串中元音的最大数目 (2)

- 题目

```
给你字符串 s 和整数 k 。
请返回字符串 s 中长度为 k 的单个子字符串中可能包含的最大元音字母数。
英文中的 元音字母 为 (a, e, i, o, u) 。
示例 1: 输入: s = "abciiidef", k = 3 输出: 3
解释: 子字符串 "iii" 包含 3 个元音字母。
示例 2: 输入: s = "aeiou", k = 2 输出: 2
解释: 任意长度为 2 的子字符串都包含 2 个元音字母。
示例 3: 输入: s = "leetcode", k = 3 输出: 2
解释: "lee"、"eet" 和 "ode" 都包含 2 个元音字母。
示例 4: 输入: s = "rhythms", k = 4 输出: 0
解释: 字符串 s 中不含任何元音字母。
示例 5: 输入: s = "tryhard", k = 4 输出: 1
提示: 1 <= s.length <= 10^5
s 由小写英文字母组成
1 <= k <= s.length
```

- 解题思路

```
func maxVowels(s string, k int) int {
        res := 0
        total := 0
        for i := 0; i < len(s); i++ {
                if isVowel(s[i]) == true {
                        total++
                }
                if i >= k {
                        if isVowel(s[i-k]) == true {
                                total--
                        }
                }
                res = max(res, total)
        }
        return res
}

func isVowel(b byte) bool {
        return b == 'a' || b == 'e' ||
                b == 'i' || b == 'o' || b == 'u'
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxVowels(s string, k int) int {
        res := 0
        total := 0
        arr := make([]int, len(s)+1)
        for i := 0; i < len(s); i++ {
                if isVowel(s[i]) == true {
                        total++
                }
                arr[i+1] = total
                if i >= k-1 {
                        res = max(res, arr[i+1]-arr[i-k+1])
                }
        }
        return res
}

func isVowel(b byte) bool {
        return b == 'a' || b == 'e' ||
                b == 'i' || b == 'o' || b == 'u'
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

44.23 1457. 二叉树中的伪回文路径 (3)

- 题目

```
给你一棵二叉树，每个节点的值为 1 到 9 。我们称二叉树中的一条路径是 「伪回文」的，
当它满足：路径经过的所有节点值的排列中，存在一个回文序列。
请你返回从根到叶子节点的所有路径中伪回文路径的数目。
示例 1：输入：root = [2,3,1,3,1,null,1] 输出：2
解释：上图为给定的二叉树。总共有 3 条从根到叶子的路径：红色路径 [2,3,3] ，
绿色路径 [2,1,1] 和路径 [2,3,1] 。
     在这些路径中，只有红色和绿色的路径是伪回文路径，因为红色路径 [2,3,3]␣
↪存在回文排列 [3,2,3] ，
     绿色路径 [2,1,1] 存在回文排列 [1,2,1] 。
示例 2：输入：root = [2,1,1,1,3,null,null,null,null,null,1] 输出：1
解释：上图为给定二叉树。总共有 3 条从根到叶子的路径：
绿色路径 [2,1,1] ，路径 [2,1,3,1] 和路径 [2,1] 。
这些路径中只有绿色路径是伪回文路径，因为 [2,1,1] 存在回文排列 [1,2,1] 。
示例 3：输入：root = [9] 输出：1
提示：给定二叉树的节点数目在1到10^5之间。
节点值在1 到9之间。
```

- 解题思路

```
var res int

func pseudoPalindromicPaths(root *TreeNode) int {
        res = 0
        dfs(root, [10]int{})
        return res
}

func dfs(root *TreeNode, arr [10]int) {
        arr[root.Val]++
        if root.Left == nil && root.Right == nil {
                count := 0
                for i := 1; i <= 9; i++ {
                        if arr[i]%2 == 1 {
                                count++
                        }
                }
                if count <= 1 {
                        res++
                }
                return
```

(续下页)

(接上页)

```
        }
        if root.Left != nil {
                dfs(root.Left, arr)
        }
        if root.Right != nil {
                dfs(root.Right, arr)
        }
}

# 2
var res int

func pseudoPalindromicPaths(root *TreeNode) int {
        res = 0
        dfs(root, make([]int, 10))
        return res
}

func dfs(root *TreeNode, arr []int) {
        if root == nil {
                return
        }
        arr[root.Val]++
        if root.Left == nil && root.Right == nil {
                count := 0
                for i := 1; i <= 9; i++ {
                        if arr[i]%2 == 1 {
                                count++
                        }
                }
                if count <= 1 {
                        res++
                }
                return
        }
        tempLeft := make([]int, 10)
        copy(tempLeft, arr)
        dfs(root.Left, tempLeft)

        tempRight := make([]int, 10)
        copy(tempRight, arr)
        dfs(root.Right, tempRight)
}
```

(续下页)

(接上页)

```
# 3
var res int

func pseudoPalindromicPaths(root *TreeNode) int {
        res = 0
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, value int) {
        if root == nil {
                return
        }
        temp := value ^ (1 << root.Val)
        if root.Left == nil && root.Right == nil {
                if temp == 0 || (temp&(temp-1)) == 0 {
                        res++
                }
                return
        }
        dfs(root.Left, temp)
        dfs(root.Right, temp)
}
```

44.24 1461. 检查一个字符串是否包含所有长度为 K 的二进制子串 (2)

- 题目

```
给你一个二进制字符串s和一个整数k。
如果所有长度为 k的二进制字符串都是 s的子串，请返回 True ，否则请返回 False 。
示例 1：输入：s = "00110110", k = 2 输出：true
解释：长度为 2 的二进制串包括 "00", "01", "10" 和 "11"。
它们分别是 s 中下标为 0, 1, 3, 2 开始的长度为 2 的子串。
示例 2：输入：s = "00110", k = 2 输出：true
示例 3：输入：s = "0110", k = 1 输出：true
解释：长度为 1 的二进制串包括 "0" 和 "1", 显然它们都是 s 的子串。
示例 4：输入：s = "0110", k = 2 输出：false
解释：长度为 2 的二进制串 "00" 没有出现在 s 中。
示例 5：输入：s = "0000000001011100", k = 4 输出：false
提示：1 <= s.length <= 5 * 10^5
s 中只含 0 和 1 。
```

(续下页)

(接上页)

```
1 <= k <= 20
```

- 解题思路

```
func hasAllCodes(s string, k int) bool {
        m := make(map[string]bool)
        for i := 0; i <= len(s)-k; i++ {
                m[s[i:i+k]] = true
        }
        return len(m) == 1<<k
}

# 2
func hasAllCodes(s string, k int) bool {
        length := 1 << k
        arr := make([]bool, length)
        cur := 0
        for i := 0; i < len(s); i++ {
                num := int(s[i] - '0')
                cur = cur<<1 + num
                if i >= k-1 {
                        cur = cur & (length - 1)
                        arr[cur] = true
                }
        }
        for i := 0; i < len(arr); i++ {
                if arr[i] == false {
                        return false
                }
        }
        return true
}
```

44.25 1462. 课程表 IV(2)

- 题目

```
你总共需要上 n门课，课程编号依次为 0到 n-1。
有的课会有直接的先修课程，比如如果想上课程0 ，你必须先上课程 1 ，那么会以 [1,
↪0]数对的形式给出先修课程数对。
给你课程总数 n和一个直接先修课程数对列表prerequisite 和一个查询对列表queries。
对于每个查询对 queries[i]，请判断queries[i][0]是否是queries[i][1]的先修课程。
```

(续下页)

(接上页)

```
请返回一个布尔值列表，列表中每个元素依次分别对应 queries每个查询对的判断结果。
注意：如果课程a是课程b的先修课程且课程b是课程c的先修课程，那么课程a也是课程c的先修课程。
示例 1：输入：n = 2, prerequisites = [[1,0]], queries = [[0,1],[1,0]] 输出：[false,
↪true]
解释：课程 0 不是课程 1 的先修课程，但课程 1 是课程 0 的先修课程。
示例 2：输入：n = 2, prerequisites = [], queries = [[1,0],[0,1]] 输出：[false,false]
解释：没有先修课程对，所以每门课程之间是独立的。
示例 3：输入：n = 3, prerequisites = [[1,2],[1,0],[2,0]], queries = [[1,0],[1,2]]␣
↪输出：[true,true]
示例 4：输入：n = 3, prerequisites = [[1,0],[2,0]], queries = [[0,1],[2,0]]␣
↪输出：[false,true]
示例 5：输入：n = 5, prerequisites = [[0,1],[1,2],[2,3],[3,4]], queries = [[0,4],[4,
↪0],[1,3],[3,0]]
输出：[true,false,true,false]
提示：2 <= n <= 100
0 <= prerequisite.length <= (n * (n - 1) / 2)
0 <= prerequisite[i][0], prerequisite[i][1] < n
prerequisite[i][0] != prerequisite[i][1]
先修课程图中没有环。
先修课程图中没有重复的边。
1 <= queries.length <= 10^4
queries[i][0] != queries[i][1]
```

- 解题思路

```
var arr []map[int]bool
var m map[string]bool

func checkIfPrerequisite(numCourses int, prerequisites [][]int, queries [][]int)␣
↪[]bool {
        res := make([]bool, len(queries))
        m = make(map[string]bool)
        arr = make([]map[int]bool, numCourses)
        for i := 0; i < len(prerequisites); i++ {
                a, b := prerequisites[i][0], prerequisites[i][1]
                if arr[a] == nil {
                        arr[a] = make(map[int]bool)
                }
                arr[a][b] = true // a=>b
        }
        for i := 0; i < len(queries); i++ {
                a, b := queries[i][0], queries[i][1]
                res[i] = dfs(a, b) // a=>b
        }
```

(续下页)

(接上页)

```
        return res
}

func dfs(i, target int) bool {
        status := fmt.Sprintf("%d,%d", i, target)
        if value, ok := m[status]; ok {
                return value
        }
        res := false
        if arr[i][target] == true {
                res = true
        } else {
                for k := range arr[i] {
                        if dfs(k, target) == true {
                                res = true
                                break
                        }
                }
        }
        m[status] = res
        return res
}

# 2
func checkIfPrerequisite(numCourses int, prerequisites [][]int, queries [][]int)␣
↪[]bool {
        res := make([]bool, len(queries))
        m := make(map[int]map[int]bool)
        for i := 0; i < numCourses; i++ {
                m[i] = make(map[int]bool)
        }
        for i := 0; i < len(prerequisites); i++ {
                a, b := prerequisites[i][0], prerequisites[i][1]
                m[a][b] = true // a=>b
        }
        for k := 0; k < numCourses; k++ {
                for i := 0; i < numCourses; i++ {
                        for j := 0; j < numCourses; j++ {
                                // ik + kj =>ij
                                if m[i][k] == true && m[k][j] == true {
                                        m[i][j] = true
                                }
                        }
```

(续下页)

(接上页)

```
                }
        }
        for i := 0; i < len(queries); i++ {
                a, b := queries[i][0], queries[i][1]
                res[i] = m[a][b]
        }
        return res
}
```

44.26 1465. 切割后面积最大的蛋糕 (1)

- 题目

```
矩形蛋糕的高度为 h 且宽度为 w，给你两个整数数组 horizontalCuts 和 verticalCuts，
其中 horizontalCuts[i] 是从矩形蛋糕顶部到第 i 个水平切口的距离，
类似地， verticalCuts[j] 是从矩形蛋糕的左侧到第 j 个竖直切口的距离。
请你按数组 horizontalCuts 和 verticalCuts 中提供的水平和竖直位置切割后，
请你找出 面积最大 的那份蛋糕，并返回其 面积 。
由于答案可能是一个很大的数字，因此需要将结果对 10^9 + 7 取余后返回。
示例 1：输入：h = 5, w = 4, horizontalCuts = [1,2,4], verticalCuts = [1,3] 输出：4
解释：上图所示的矩阵蛋糕中，红色线表示水平和竖直方向上的切口。切割蛋糕后，绿色的那份蛋糕面积最大。
示例 2：输入：h = 5, w = 4, horizontalCuts = [3,1], verticalCuts = [1] 输出：6
解释：上图所示的矩阵蛋糕中，红色线表示水平和竖直方向上的切口。切割蛋糕后，绿色和黄色的两份蛋糕面积最
示例 3：输入：h = 5, w = 4, horizontalCuts = [3], verticalCuts = [3] 输出：9
提示：2 <= h,w <= 10^9
1 <=horizontalCuts.length <min(h, 10^5)
1 <=verticalCuts.length < min(w, 10^5)
1 <=horizontalCuts[i] < h
1 <=verticalCuts[i] < w
题目数据保证 horizontalCuts 中的所有元素各不相同
题目数据保证 verticalCuts中的所有元素各不相同
```

- 解题思路

```
func maxArea(h int, w int, horizontalCuts []int, verticalCuts []int) int {
        hArr := append(horizontalCuts, h)
        wArr := append(verticalCuts, w)
        sort.Ints(hArr)
        sort.Ints(wArr)
        maxHeight := hArr[0]
        maxWeight := wArr[0]
        for i := 1; i < len(hArr); i++ {
```

(续下页)

(接上页)

```
                if hArr[i]-hArr[i-1] > maxHeight {
                        maxHeight = hArr[i] - hArr[i-1]
                }
        }
        for i := 1; i < len(wArr); i++ {
                if wArr[i]-wArr[i-1] > maxWeight {
                        maxWeight = wArr[i] - wArr[i-1]
                }
        }
        return maxHeight * maxWeight % 1000000007
}
```

44.27 1466. 重新规划路线 (2)

- 题目

```
n 座城市，从 0 到 n-1 编号，其间共有 n-1 条路线。
因此，要想在两座不同城市之间旅行只有唯一一条路线可供选择（路线网形成一颗树）。
去年，交通运输部决定重新规划路线，以改变交通拥堵的状况。
路线用 connections 表示，其中 connections[i] = [a, b] 表示从城市 a 到 b␣
↪的一条有向路线。
今年，城市 0 将会举办一场大型比赛，很多游客都想前往城市 0 。
请你帮助重新规划路线方向，使每个城市都可以访问城市 0 。返回需要变更方向的最小路线数。
题目数据 保证 每个城市在重新规划路线方向后都能到达城市 0 。
示例 1：输入：n = 6, connections = [[0,1],[1,3],[2,3],[4,0],[4,5]] 输出：3
解释：更改以红色显示的路线的方向，使每个城市都可以到达城市 0 。
示例 2：输入：n = 5, connections = [[1,0],[1,2],[3,2],[3,4]] 输出：2
解释：更改以红色显示的路线的方向，使每个城市都可以到达城市 0 。
示例 3：输入：n = 3, connections = [[1,0],[2,0]] 输出：0
提示：2 <= n <= 5 * 10^4
connections.length == n-1
connections[i].length == 2
0 <= connections[i][0], connections[i][1] <= n-1
connections[i][0] != connections[i][1]
```

- 解题思路

```
func minReorder(n int, connections [][]int) int {
        m := make(map[int]map[int]int)
        for i := 0; i < len(connections); i++ {
                a, b := connections[i][0], connections[i][1]
                if _, ok := m[a]; ok == false {
```

(续下页)

(接上页)

```
                        m[a] = make(map[int]int)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = make(map[int]int)
                }
                m[a][b] = 1  // a->b
                m[b][a] = -1 // b->a
        }
        res := 0
        visited := make(map[int]bool)
        visited[0] = true
        queue := make([]int, 0)
        queue = append(queue, 0)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                for k, v := range m[node] {
                        if visited[k] == true {
                                continue
                        }
                        visited[k] = true
                        if v == 1 {
                                res++
                        }
                        queue = append(queue, k)
                }
        }
        return res
}

# 2
var res int
var m map[int]map[int]int

func minReorder(n int, connections [][]int) int {
        m = make(map[int]map[int]int)
        for i := 0; i < len(connections); i++ {
                a, b := connections[i][0], connections[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = make(map[int]int)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = make(map[int]int)
```

(续下页)

(接上页)

```
                }
                m[a][b] = 1  // a->b
                m[b][a] = -1 // b->a
        }
        res = 0
        visited := make(map[int]bool)
        visited[0] = true
        dfs(0, visited)
        return res
}

func dfs(start int, visited map[int]bool) {
        for k, v := range m[start] {
                if visited[k] == true {
                        continue
                }
                visited[k] = true
                if v == 1 {
                        res++
                }
                dfs(k, visited)
        }
}
```

44.28 1471. 数组中的 k 个最强值 (2)

- 题目

```
给你一个整数数组 arr 和一个整数 k 。
设 m 为数组的中位数，只要满足下述两个前提之一，就可以判定 arr[i] 的值比 arr[j]␣
↪的值更强：
|arr[i] - m| > |arr[j]- m|
|arr[i] - m| == |arr[j] - m|，且 arr[i] > arr[j]
请返回由数组中最强的 k 个值组成的列表。答案可以以 任意顺序 返回。
中位数 是一个有序整数列表中处于中间位置的值。
形式上，如果列表的长度为 n ，那么中位数就是该有序列表（下标从 0 开始）中位于 ((n - 1)␣
↪/ 2) 的元素。
例如 arr =[6, -3, 7, 2, 11]，n = 5：数组排序后得到 arr = [-3, 2, 6, 7, 11] ，
数组的中间位置为 m = ((5 - 1) / 2) = 2 ，中位数 arr[m] 的值为 6 。
例如 arr =[-7, 22, 17, 3]，n = 4：数组排序后得到arr = [-7, 3, 17, 22] ，
数组的中间位置为m = ((4 - 1) / 2) = 1 ，中位数 arr[m] 的值为 3 。
示例 1：输入：arr = [1,2,3,4,5]，k = 2 输出：[5,1]
```

(续下页)

(接上页)

```
解释：中位数为 3，按从强到弱顺序排序后，数组变为 [5,1,4,2,3]。
最强的两个元素是 [5, 1]。[1, 5] 也是正确答案。
注意，尽管 |5 - 3| == |1 - 3| ，但是 5 比 1 更强，因为 5 > 1 。
示例 2：输入：arr = [1,1,3,5,5], k = 2 输出：[5,5]
解释：中位数为 3，按从强到弱顺序排序后，数组变为 [5,5,1,1,3]。最强的两个元素是 [5,␣
↪5]。
示例 3：输入：arr = [6,7,11,7,6,8], k = 5 输出：[11,8,6,6,7]
解释：中位数为 7，按从强到弱顺序排序后，数组变为 [11,8,6,6,7,7]。
[11,8,6,6,7] 的任何排列都是正确答案。
示例 4：输入：arr = [6,-3,7,2,11], k = 3 输出：[-3,11,2]
示例 5：输入：arr = [-7,22,17,3], k = 2 输出：[22,17]
提示：1 <= arr.length <= 10^5
-10^5 <= arr[i] <= 10^5
1 <= k <= arr.length
```

- 解题思路

```
func getStrongest(arr []int, k int) []int {
        sort.Ints(arr)
        mid := arr[(len(arr)-1)/2]
        sort.Slice(arr, func(i, j int) bool {
                if abs(arr[i]-mid) == abs(arr[j]-mid) {
                        return arr[i] > arr[j]
                }
                return abs(arr[i]-mid) > abs(arr[j]-mid)
        })
        return arr[:k]
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func getStrongest(arr []int, k int) []int {
        sort.Ints(arr)
        mid := arr[(len(arr)-1)/2]
        res := make([]int, 0)
        left, right := 0, len(arr)-1
        for k > 0 {
                if arr[right]-mid >= mid-arr[left] {
```

(续下页)

(接上页)

```
                        res = append(res, arr[right])
                        right--
                } else {
                        res = append(res, arr[left])
                        left++
                }
                k--
        }
        return res
}
```

44.29 1472. 设计浏览器历史记录 (1)

- 题目

```
你有一个只支持单个标签页的 浏览器，最开始你浏览的网页是homepage，
你可以访问其他的网站url，也可以在浏览历史中后退steps步或前进steps步。
请你实现BrowserHistory 类：
BrowserHistory(string homepage)，用homepage初始化浏览器类。
void visit(string url)从当前页跳转访问 url 对应的页面。
执行此操作会把浏览历史前进的记录全部删除。
string back(int steps)在浏览历史中后退steps步。
如果你只能在浏览历史中后退至多x 步且steps > x，那么你只后退x步。
请返回后退 至多 steps步以后的url。
string forward(int steps)在浏览历史中前进steps步。
如果你只能在浏览历史中前进至多x步且steps > x，那么你只前进 x步。
请返回前进至多steps步以后的 url。
示例：输入：["BrowserHistory","visit","visit","visit","back","back","forward",
"visit","forward","back","back"]
[["leetcode.com"],["google.com"],["facebook.com"],
["youtube.com"],[1],[1],[1],["linkedin.com"],[2],[2],[7]]
输出：[null,null,null,null,"facebook.com","google.com","facebook.com",null,
"linkedin.com","google.com","leetcode.com"]
解释：
BrowserHistory browserHistory = new BrowserHistory("leetcode.com");
browserHistory.visit("google.com");       // 你原本在浏览 "leetcode.com" 。访问
↪"google.com"
browserHistory.visit("facebook.com");     // 你原本在浏览 "google.com" 。访问
↪"facebook.com"
browserHistory.visit("youtube.com");
// 你原本在浏览 "facebook.com" 。访问 "youtube.com"
browserHistory.back(1);
```

(续下页)

(接上页)

```
// 你原本在浏览 "youtube.com" ，后退到 "facebook.com" 并返回 "facebook.com"
browserHistory.back(1);
// 你原本在浏览 "facebook.com" ，后退到 "google.com" 并返回 "google.com"
browserHistory.forward(1);
// 你原本在浏览 "google.com" ，前进到 "facebook.com" 并返回 "facebook.com"
browserHistory.visit("linkedin.com");
// 你原本在浏览 "facebook.com" 。 访问 "linkedin.com"
browserHistory.forward(2);
// 你原本在浏览 "linkedin.com" ，你无法前进任何步数。
browserHistory.back(2);
// 你原本在浏览 "linkedin.com" ，后退两步依次先到 "facebook.com" ，然后到 "google.com
↪" ，并返回 "google.com"
browserHistory.back(7);
// 你原本在浏览 "google.com"， 你只能后退一步到 "leetcode.com" ，并返回 "leetcode.com"
提示：1 <= homepage.length <= 20
1 <= url.length <= 20
1 <= steps <= 100
homepage 和url都只包含'.' 或者小写英文字母。
最多调用5000次visit，back和forward函数。
```

- 解题思路

```
type BrowserHistory struct {
        arr   []string
        index int
}

func Constructor(homepage string) BrowserHistory {
        return BrowserHistory{
                arr:   []string{homepage},
                index: 0,
        }
}

func (this *BrowserHistory) Visit(url string) {
        length := len(this.arr)
        if this.index == length-1 {
                this.arr = append(this.arr, url)
        } else if this.index < length-1 {
                this.arr = this.arr[:this.index+1]
                this.arr = append(this.arr, url)
        }
        this.index++
}
```

(续下页)

(接上页)

```
func (this *BrowserHistory) Back(steps int) string {
        if steps > this.index {
                this.index = 0
                return this.arr[0]
        }
        this.index = this.index - steps
        return this.arr[this.index]
}

func (this *BrowserHistory) Forward(steps int) string {
        length := len(this.arr)
        if this.index == length-1 {
        } else if this.index+steps > length-1 {
                this.index = length - 1
        } else {
                this.index = this.index + steps
        }
        return this.arr[this.index]
}
```

44.30 1476. 子矩形查询 (2)

- 题目

```
请你实现一个类SubrectangleQueries，
它的构造函数的参数是一个 rows x cols的矩形（这里用整数矩阵表示），并支持以下两种操作：
1.updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)
用newValue更新以(row1,col1)为左上角且以(row2,col2)为右下角的子矩形。
2.getValue(int row, int col)
返回矩形中坐标 (row,col) 的当前值。
示例 1： 输入：["SubrectangleQueries","getValue","updateSubrectangle",
"getValue","getValue","updateSubrectangle","getValue","getValue"]
[[[[1,2,1],[4,3,4],[3,2,1],[1,1,1]]],[0,2],[0,0,3,2,5],[0,2],[3,1],[3,0,3,2,10],[3,1],
↪[0,2]]
输出： [null,1,null,5,5,null,10,5]
解释：
SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,2,1],[4,3,4],[3,
↪2,1],[1,1,1]]);
// 初始的 (4x3) 矩形如下：
// 1 2 1
// 4 3 4
```

(续下页)

(接上页)

```
// 3 2 1
// 1 1 1
subrectangleQueries.getValue(0, 2); // 返回 1
subrectangleQueries.updateSubrectangle(0, 0, 3, 2, 5);
// 此次更新后矩形变为：
// 5 5 5
// 5 5 5
// 5 5 5
// 5 5 5
subrectangleQueries.getValue(0, 2); // 返回 5
subrectangleQueries.getValue(3, 1); // 返回 5
subrectangleQueries.updateSubrectangle(3, 0, 3, 2, 10);
// 此次更新后矩形变为：
// 5   5   5
// 5   5   5
// 5   5   5
// 10  10  10
subrectangleQueries.getValue(3, 1); // 返回 10
subrectangleQueries.getValue(0, 2); // 返回 5
示例 2：输入：["SubrectangleQueries","getValue","updateSubrectangle","getValue",
"getValue","updateSubrectangle","getValue"]
[[[[1,1,1],[2,2,2],[3,3,3]]],[0,0],[0,0,2,2,100],[0,0],[2,2],[1,1,2,2,20],[2,2]]
输出：[null,1,null,100,100,null,20]
解释：SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,1,1],[2,2,
↪2],[3,3,3]]);
subrectangleQueries.getValue(0, 0); // 返回 1
subrectangleQueries.updateSubrectangle(0, 0, 2, 2, 100);
subrectangleQueries.getValue(0, 0); // 返回 100
subrectangleQueries.getValue(2, 2); // 返回 100
subrectangleQueries.updateSubrectangle(1, 1, 2, 2, 20);
subrectangleQueries.getValue(2, 2); // 返回 20
提示：最多有500次updateSubrectangle 和getValue操作。
1 <= rows, cols <= 100
rows ==rectangle.length
cols == rectangle[i].length
0 <= row1 <= row2 < rows
0 <= col1 <= col2 < cols
1 <= newValue, rectangle[i][j] <= 10^9
0 <= row < rows
0 <= col < cols
```

- 解题思路

```
type SubrectangleQueries struct {
        arr [][]int
}

func Constructor(rectangle [][]int) SubrectangleQueries {
        return SubrectangleQueries{arr: rectangle}
}

func (this *SubrectangleQueries) UpdateSubrectangle(row1 int, col1 int, row2 int,␣
↪col2 int, newValue int) {
        for i := row1; i <= row2; i++ {
                for j := col1; j <= col2; j++ {
                        this.arr[i][j] = newValue
                }
        }
}

func (this *SubrectangleQueries) GetValue(row int, col int) int {
        return this.arr[row][col]
}

# 2
type SubrectangleQueries struct {
        arr    [][]int
        record [][]int // 更改记录
}

func Constructor(rectangle [][]int) SubrectangleQueries {
        return SubrectangleQueries{arr: rectangle}
}

func (this *SubrectangleQueries) UpdateSubrectangle(row1 int, col1 int, row2 int,␣
↪col2 int, newValue int) {
        this.record = append(this.record, []int{row1, col1, row2, col2, newValue})
}

func (this *SubrectangleQueries) GetValue(row int, col int) int {
        for i := len(this.record) - 1; i >= 0; i-- {
                row1, col1 := this.record[i][0], this.record[i][1]
                row2, col2 := this.record[i][2], this.record[i][3]
                newValue := this.record[i][4]
                if row1 <= row && row <= row2 &&
                        col1 <= col && col <= col2 {
                        return newValue
```

(续下页)

(接上页)

```
                }
        }
        return this.arr[row][col]
}
```

44.31 1477. 找两个和为目标值且不重叠的子数组 (1)

- 题目

```
给你一个整数数组arr 和一个整数值target。
请你在 arr中找 两个互不重叠的子数组且它们的和都等于target。
可能会有多种方案，请你返回满足要求的两个子数组长度和的 最小值 。
请返回满足要求的最小长度和，如果无法找到这样的两个子数组，请返回 -1。
示例 1：输入：arr = [3,2,2,4,3], target = 3 输出：2
解释：只有两个子数组和为 3 （[3] 和 [3]）。它们的长度和为 2 。
示例 2：输入：arr = [7,3,4,7], target = 7 输出：2
解释：尽管我们有 3 个互不重叠的子数组和为 7 （[7], [3,4] 和 [7]），
但我们会选择第一个和第三个子数组，因为它们的长度和 2 是最小值。
示例 3：输入：arr = [4,3,2,6,2,3,4], target = 6 输出：-1
解释：我们只有一个和为 6 的子数组。
示例 4：输入：arr = [5,5,4,4,5], target = 3 输出：-1
解释：我们无法找到和为 3 的子数组。
示例 5：输入：arr = [3,1,1,1,5,1,2,1], target = 3 输出：3
解释：注意子数组 [1,2] 和 [2,1] 不能成为一个方案因为它们重叠了。
提示：1 <= arr.length <= 10^5
1 <= arr[i] <= 1000
1 <= target <= 10^8
```

- 解题思路

```
func minSumOfLengths(arr []int, target int) int {
        res := math.MaxInt32
        n := len(arr)
        sum := 0
        left := 0
        temp := make([]int, n) // 保存每个位置之前的最小值
        for i := 0; i < n; i++ {
                temp[i] = math.MaxInt32 // 默认为最大值
        }
        for right := 0; right < n; right++ {
                sum = sum + arr[right]
                for sum > target {
```

(续下页)

(接上页)

```
                        sum = sum - arr[left]
                        left++
                }
                if right >= 1 {
                        temp[right] = temp[right-1] // 默认同之前最小值
                }
                if sum == target { // 找到目标值
                        temp[right] = min(temp[right], right-left+1) // 更新最小值
                        if left >= 1 && temp[left-1] != math.MaxInt32 {
                                res = min(res, temp[left-1]+right-left+1) //取left之前(即left-1)的最小值，加上当前长度
                        }
                }
        }
        if res == math.MaxInt32 {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

44.32 1481. 不同整数的最少数目 (1)

- 题目

```
给你一个整数数组 arr 和一个整数 k 。
现需要从数组中恰好移除 k 个元素，请找出移除后数组中不同整数的最少数目。
示例 1: 输入：arr = [5,5,4], k = 1 输出：1
解释：移除 1 个 4 ，数组中只剩下 5 一种整数。
示例 2: 输入：arr = [4,3,1,1,3,3,2], k = 3 输出：2
解释：先移除 4、2 ，然后再移除两个 1 中的任意 1 个或者三个 3 中的任意 1 个，最后剩下1 和 3 两种整数。
提示：1 <= arr.length <= 10^5
    1 <= arr[i] <= 10^9
    0 <= k <= arr.length
```

- 解题思路

```
func findLeastNumOfUniqueInts(arr []int, k int) int {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]]++
        }
        temp := make([]int, 0)
        for _, v := range m {
                temp = append(temp, v)
        }
        sort.Ints(temp)
        res := len(temp)
        for i := 0; i < len(temp); i++ {
                if k >= temp[i] {
                        res--
                        k = k - temp[i]
                } else {
                        break
                }
        }
        return res
}
```

44.33 1482. 制作 m 束花所需的最少天数 (1)

- 题目

```
给你一个整数数组 bloomDay，以及两个整数 m 和 k 。
现需要制作 m 束花。制作花束时，需要使用花园中 相邻的 k 朵花 。
花园中有 n 朵花，第 i 朵花会在 bloomDay[i] 时盛开，恰好 可以用于 一束 花中。
请你返回从花园中摘 m 束花需要等待的最少的天数。如果不能摘到 m 束花则返回 -1 。
示例 1：输入：bloomDay = [1,10,3,10,2], m = 3, k = 1 输出：3
解释：让我们一起观察这三天的花开过程，x 表示花开，而 _ 表示花还未开。
现在需要制作 3 束花，每束只需要 1 朵。
1 天后：[x, _, _, _, _]   // 只能制作 1 束花
2 天后：[x, _, _, _, x]   // 只能制作 2 束花
3 天后：[x, _, x, _, x]   // 可以制作 3 束花，答案为 3
示例 2：输入：bloomDay = [1,10,3,10,2], m = 3, k = 2 输出：-1
解释：要制作 3 束花，每束需要 2 朵花，也就是一共需要 6 朵花。
而花园中只有 5 朵花，无法满足制作要求，返回 -1 。
示例 3：输入：bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3 输出：12
解释：要制作 2 束花，每束需要 3 朵。
花园在 7 天后和 12 天后的情况如下：
```

(续下页)

(接上页)

```
7 天后：[x, x, x, x, _, x, x]
可以用前 3 朵盛开的花制作第一束花。但不能使用后 3 朵盛开的花，因为它们不相邻。
12 天后：[x, x, x, x, x, x, x]
显然，我们可以用不同的方式制作两束花。
示例 4：输入：bloomDay = [1000000000,1000000000], m = 1, k = 1 输出：1000000000
解释：需要等 1000000000 天才能采到花来制作花束
示例 5：输入：bloomDay = [1,10,2,9,3,8,4,7,5,6], m = 4, k = 2 输出：9
提示：bloomDay.length == n
    1 <= n <= 10^5
    1 <= bloomDay[i] <= 10^9
    1 <= m <= 10^6
    1 <= k <= n
```

- 解题思路

```
func minDays(bloomDay []int, m int, k int) int {
        if m*k > len(bloomDay) {
                return -1
        }
        minValue, maxValue := bloomDay[0], bloomDay[0]
        for i := 1; i < len(bloomDay); i++ {
                if bloomDay[i] > maxValue {
                        maxValue = bloomDay[i]
                }
                if bloomDay[i] < minValue {
                        minValue = bloomDay[i]
                }
        }
        left, right := minValue, maxValue
        for left < right {
                mid := left + (right-left)/2
                count := judge(bloomDay, mid, k)
                if count >= m {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

func judge(bloomDay []int, mid int, k int) int {
        total := 0
        count := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(bloomDay); i++ {
                if bloomDay[i] <= mid {
                        total++
                } else {
                        total = 0
                }
                if total == k {
                        count++
                        total = 0
                }
        }
        return count
}
```

44.34 1487. 保证文件名唯一 (2)

- 题目

```
给你一个长度为 n 的字符串数组 names 。
你将会在文件系统中创建 n 个文件夹：在第 i 分钟，新建名为 names[i] 的文件夹。
由于两个文件 不能 共享相同的文件名，因此如果新建文件夹使用的文件名已经被占用，
系统会以 (k) 的形式为新文件夹的文件名添加后缀，其中 k 是能保证文件名唯一的 最小正整数␣
↪。
返回长度为 n 的字符串数组，其中 ans[i] 是创建第 i␣
↪个文件夹时系统分配给该文件夹的实际名称。
示例 1：
输入：names = ["pes","fifa","gta","pes(2019)"]
输出：["pes","fifa","gta","pes(2019)"]
解释：文件系统将会这样创建文件名：
"pes" --> 之前未分配，仍为 "pes"
"fifa" --> 之前未分配，仍为 "fifa"
"gta" --> 之前未分配，仍为 "gta"
"pes(2019)" --> 之前未分配，仍为 "pes(2019)"
示例 2：
输入：names = ["gta","gta(1)","gta","avalon"]
输出：["gta","gta(1)","gta(2)","avalon"]
解释：文件系统将会这样创建文件名：
"gta" --> 之前未分配，仍为 "gta"
"gta(1)" --> 之前未分配，仍为 "gta(1)"
"gta" --> 文件名被占用，系统为该名称添加后缀 (k)，由于 "gta(1)" 也被占用，所以 k = 2␣
↪。
实际创建的文件名为 "gta(2)" 。
```

(续下页)

(接上页)

```
"avalon" --> 之前未分配，仍为 "avalon"
示例 3:
输入：names = ["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece"]
输出：["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece(4)"]
解释：当创建最后一个文件夹时，最小的正有效 k 为 4 ，文件名变为 "onepiece(4)"。
示例 4:
输入：names = ["wano","wano","wano","wano"]
输出：["wano","wano(1)","wano(2)","wano(3)"]
解释：每次创建文件夹 "wano" 时，只需增加后缀中 k 的值即可。
示例 5:
输入：names = ["kaido","kaido(1)","kaido","kaido(1)"]
输出：["kaido","kaido(1)","kaido(2)","kaido(1)(1)"]
解释：注意，如果含后缀文件名被占用，那么系统也会按规则在名称后添加新的后缀 (k) 。
提示：
    1 <= names.length <= 5 * 10^4
    1 <= names[i].length <= 20
    names[i] 由小写英文字母、数字和/或圆括号组成。
```

- 解题思路

```
func getFolderNames(names []string) []string {
        m := make(map[string]int)
        for i, name := range names {
                if value, ok := m[name]; ok {
                        names[i] = getName(m, name, value)
                        m[names[i]] = 1
                } else {
                        m[name] = 1
                }
        }
        return names
}

func getName(m map[string]int, name string, n int) string {
        newName := name + fmt.Sprintf("(%d)", n)
        if _, ok := m[newName]; ok {
                return getName(m, name, n+1)
        }
        m[name] = n + 1
        return newName
}


#
```

(续下页)

(接上页)

```
func getFolderNames(names []string) []string {
        m := make(map[string]int)
        res := make([]string, 0)
        for _, name := range names {
                if value, ok := m[name]; ok {
                        for {
                                newName := name + fmt.Sprintf("(%d)", value)
                                if _, ok2 := m[newName]; ok2 {
                                        value++
                                        continue
                                }
                                res = append(res, newName)
                                m[newName] = 1
                                m[name] = value
                                break
                        }
                } else {
                        res = append(res, name)
                        m[name] = 1
                }
        }
        return res
}
```

44.35 1488. 避免洪水泛滥

44.35.1 题目

```
你的国家有无数个湖泊，所有湖泊一开始都是空的。
当第 n个湖泊下雨的时候，如果第␣
↪n个湖泊是空的，那么它就会装满水，否则这个湖泊会发生洪水。
你的目标是避免任意一个湖泊发生洪水。
给你一个整数数组rains，其中：
rains[i] > 0表示第 i天时，第 rains[i]个湖泊会下雨。
rains[i] == 0表示第 i天没有湖泊会下雨，你可以选择 一个湖泊并 抽干这个湖泊的水。
请返回一个数组ans，满足：ans.length == rains.length
如果rains[i] > 0 ，那么ans[i] == -1。
如果rains[i] == 0，ans[i]是你第i天选择抽干的湖泊。
如果有多种可行解，请返回它们中的 任意一个。如果没办法阻止洪水，请返回一个 空的数组。
请注意，如果你选择抽干一个装满水的湖泊，它会变成一个空的湖泊。
但如果你选择抽干一个空的湖泊，那么将无事发生（详情请看示例 4）。
```

(续下页)

(接上页)

```
示例 1: 输入: rains = [1,2,3,4] 输出: [-1,-1,-1,-1]
解释: 第一天后, 装满水的湖泊包括 [1]
第二天后, 装满水的湖泊包括 [1,2]
第三天后, 装满水的湖泊包括 [1,2,3]
第四天后, 装满水的湖泊包括 [1,2,3,4]
没有哪一天你可以抽干任何湖泊的水, 也没有湖泊会发生洪水。
示例 2: 输入: rains = [1,2,0,0,2,1] 输出: [-1,-1,2,1,-1,-1]
解释: 第一天后, 装满水的湖泊包括 [1]
第二天后, 装满水的湖泊包括 [1,2]
第三天后, 我们抽干湖泊 2 。所以剩下装满水的湖泊包括 [1]
第四天后, 我们抽干湖泊 1 。所以暂时没有装满水的湖泊了。
第五天后, 装满水的湖泊包括 [2]。
第六天后, 装满水的湖泊包括 [1,2]。
可以看出, 这个方案下不会有洪水发生。同时, [-1,-1,1,2,-1,-1]␣
↪也是另一个可行的没有洪水的方案。
示例 3: 输入: rains = [1,2,0,1,2] 输出: []
解释: 第二天后, 装满水的湖泊包括 [1,2]。我们可以在第三天抽干一个湖泊的水。
但第三天后, 湖泊 1 和 2␣
↪都会再次下雨, 所以不管我们第三天抽干哪个湖泊的水, 另一个湖泊都会发生洪水。
示例 4: 输入: rains = [69,0,0,0,69] 输出: [-1,69,1,1,-1]
解释: 任何形如 [-1,69,x,y,-1], [-1,x,69,y,-1] 或者 [-1,x,y,69,-1] 都是可行的解,
其中 1 <= x,y <= 10^9
示例 5: 输入: rains = [10,20,20] 输出: []
解释: 由于湖泊 20 会连续下 2 天的雨, 所以没有没有办法阻止洪水。
提示: 1 <= rains.length <= 10^5
0 <= rains[i] <= 10^9
```

44.35.2 解题思路

44.36 1492.n 的第 k 个因子 (2)

- 题目

```
给你两个正整数 n 和 k 。
如果正整数 i 满足 n % i == 0 , 那么我们就说正整数 i 是整数 n 的因子。
考虑整数 n 的所有因子, 将它们 升序排列 。请你返回第 k 个因子。如果 n 的因子数少于 k␣
↪, 请你返回 -1 。
示例 1: 输入: n = 12, k = 3输出: 3
```

(续下页)

(接上页)

```
解释：因子列表包括 [1, 2, 3, 4, 6, 12]，第 3 个因子是 3 。
示例 2：输入：n = 7, k = 2 输出：7
解释：因子列表包括 [1, 7] ，第 2 个因子是 7 。
示例 3：输入：n = 4, k = 4 输出：-1
解释：因子列表包括 [1, 2, 4] ，只有 3 个因子，所以我们应该返回 -1 。
示例 4：输入：n = 1, k = 1 输出：1
解释：因子列表包括 [1] ，第 1 个因子为 1 。
示例 5：输入：n = 1000, k = 3 输出：4
解释：因子列表包括 [1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000]␣
↪。
提示：
    1 <= k <= n <= 1000
```

- 解题思路

```
func kthFactor(n int, k int) int {
        count := 0
        for i := 1; i <= n; i++ {
                if n%i == 0 {
                        count++
                        if count == k {
                                return i
                        }
                }
        }
        return -1
}

#
func kthFactor(n int, k int) int {
        count := 0
        i := 1
        for i = 1; i*i <= n; i++ {
                if n%i == 0 {
                        count++
                        if count == k {
                                return i
                        }
                }
        }
        i--
        if i*i == n {
                i--
        }
```

(续下页)

(接上页)

```
        for ; i > 0; i-- {
                if n%i == 0 {
                        count++
                        if count == k {
                                return n / i
                        }
                }
        }
        return -1
}
```

44.37 1493. 删掉一个元素以后全为 1 的最长子数组 (3)

- 题目

```
给你一个二进制数组 nums ，你需要从中删掉一个元素。
请你在删掉元素的结果数组中，返回最长的且只包含 1 的非空子数组的长度。
如果不存在这样的子数组，请返回 0 。
提示 1: 输入：nums = [1,1,0,1] 输出：3
解释：删掉位置 2 的数后，[1,1,1] 包含 3 个 1 。
示例 2: 输入：nums = [0,1,1,1,0,1,1,0,1] 输出：5
解释：删掉位置 4 的数字后，[0,1,1,1,1,1,0,1] 的最长全 1 子数组为 [1,1,1,1,1] 。
示例 3: 输入：nums = [1,1,1] 输出：2
解释：你必须要删除一个元素。
示例 4: 输入：nums = [1,1,0,0,1,1,1,0,1] 输出：4
示例 5: 输入：nums = [0,0,0] 输出：0
提示：
    1 <= nums.length <= 10^5
    nums[i] 要么是 0 要么是 1 。
```

- 解题思路

```
func longestSubarray(nums []int) int {
        n := len(nums)
        pre := make([]int, n)
        suf := make([]int, n)
        pre[0] = nums[0]
        for i := 1; i < n; i++ {
                if nums[i] == 1 {
                        pre[i] = pre[i-1] + 1
                } else {
                        pre[i] = 0
```

(续下页)

(接上页)

```
                }
        }
        suf[n-1] = nums[n-1]
        for i := n - 2; i >= 0; i-- {
                if nums[i] == 1 {
                        suf[i] = suf[i+1] + 1
                } else {
                        suf[i] = 0
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                var p, s int
                if i == 0 {
                        p = 0
                } else {
                        p = pre[i-1]
                }
                if i == n-1 {
                        s = 0
                } else {
                        s = suf[i+1]
                }
                if p+s > res {
                        res = p + s
                }
        }
        return res
}

#
func longestSubarray(nums []int) int {
        res := 0
        p, q := 0, 0 // q=>中间有一个“非1”的和, p=>连续1的和
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        q = p
                        p = 0
                } else {
                        p++
                        q++
                }
                if q > res {
```

(续下页)

(接上页)

```
                        res = q
                }
        }
        if res == len(nums) {
                return res - 1
        }
        return res
}

#
func longestSubarray(nums []int) int {
        arr := make([]int, 0)
        count := 0
        for _, v := range nums {
                if v == 0 {
                        arr = append(arr, count)
                        count = 0
                        continue
                }
                count++
        }
        arr = append(arr, count)
        if len(arr) == 1 {
                return arr[0] - 1
        }
        res := 0
        for i := 0; i < len(arr)-1; i++ {
                if arr[i]+arr[i+1] > res {
                        res = arr[i] + arr[i+1]
                }
        }
        return res
}
```

44.38 1497. 检查数组对是否可以被 k 整除 (2)

- 题目

```
给你一个整数数组 arr 和一个整数 k ，其中数组长度是偶数，值为 n 。
现在需要把数组恰好分成 n / 2 对，以使每对数字的和都能够被 k 整除。
如果存在这样的分法，请返回 True ；否则，返回 False 。
示例 1：输入：arr = [1,2,3,4,5,10,6,7,8,9], k = 5 输出：true
```

(续下页)

(接上页)

```
解释：划分后的数字对为 (1,9),(2,8),(3,7),(4,6) 以及 (5,10) 。
示例 2：输入：arr = [1,2,3,4,5,6], k = 7 输出：true
解释：划分后的数字对为 (1,6),(2,5) 以及 (3,4) 。
示例 3：输入：arr = [1,2,3,4,5,6], k = 10 输出：false
解释：无法在将数组中的数字分为三对的同时满足每对数字和能够被 10 整除的条件。
示例 4：输入：arr = [-10,10], k = 2 输出：true
示例 5：输入：arr = [-1,1,-2,2,-3,3,-4,4], k = 3 输出：true
提示：arr.length == n
    1 <= n <= 10^5
    n 为偶数
    -10^9 <= arr[i] <= 10^9
    1 <= k <= 10^5
```

- 解题思路

```
func canArrange(arr []int, k int) bool {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                value := ((arr[i] % k) + k) % k
                m[value]++
        }
        for key, value := range m {
                if key == 0 && value%2 != 0 {
                        return false
                }
                target := (k - key) % k // 避免key=0, k-0=k的情况
                if m[target] != value {
                        return false
                }
        }
        return true
}

# 2
func canArrange(arr []int, k int) bool {
        temp := make([]int, k)
        for i := 0; i < len(arr); i++ {
                value := ((arr[i] % k) + k) % k
                temp[value]++
        }
        for i := 1; i < k; i++ {
                if temp[i] != temp[k-i] {
                        return false
                }
```

(续下页)

(接上页)

```
        }
        return temp[0]%2 == 0
}
```

44.39 1498. 满足条件的子序列数目 (2)

- 题目

```
给你一个整数数组 nums 和一个整数 target 。
请你统计并返回 nums 中能满足其最小元素与最大元素的 和 小于或等于 target 的 非空␣
↪子序列的数目。
由于答案可能很大，请将结果对 10^9 + 7 取余后返回。
示例 1：输入：nums = [3,5,6,7], target = 9 输出：4
解释：有 4 个子序列满足该条件。
[3] -> 最小元素 + 最大元素 <= target (3 + 3 <= 9)
[3,5] -> (3 + 5 <= 9)
[3,5,6] -> (3 + 6 <= 9)
[3,6] -> (3 + 6 <= 9)
示例 2：输入：nums = [3,3,6,8], target = 10 输出：6
解释：有 6 个子序列满足该条件。（nums 中可以有重复数字）
[3] , [3] , [3,3], [3,6] , [3,6] , [3,3,6]
示例 3：输入：nums = [2,3,3,4,6,7], target = 12 输出：61
解释：共有 63 个非空子序列，其中 2 个不满足条件（[6,7], [7]）
有效序列总数为（63 - 2 = 61）
示例 4：输入：nums = [5,2,4,1,7,6,8], target = 16 输出：127
解释：所有非空子序列都满足条件 (2^7 - 1) = 127
提示：1 <= nums.length <= 10^5
1 <= nums[i] <= 10^6
1 <= target <= 10^6
```

- 解题思路

```
func numSubseq(nums []int, target int) int {
        sort.Ints(nums)
        // 计算长度为length满足条件的非空子序列的数目
        // 如1、2、3、4，长度为4，1必选，其他3个数可选可不选，组合数：2^3=8
        m := make(map[int]int)
        m[1] = 1
        for i := 2; i <= len(nums); i++ {
                m[i] = (m[i-1] * 2) % 1000000007
        }
        res := 0
```

(续下页)

(接上页)

```
        left, right := 0, len(nums)-1
        for left <= right {
                if nums[left]+nums[right] <= target {
                        length := right - left + 1
                        res = res + m[length]
                        left++
                } else {
                        right--
                }
        }
        return res % 1000000007
}

# 2
func numSubseq(nums []int, target int) int {
        sort.Ints(nums)
        // 计算长度为length满足条件的非空子序列的数目
        // 如1、2、3、4，长度为4，1必选，其他3个数可选可不选，组合数：2^3=8
        m := make(map[int]int)
        m[1] = 1
        for i := 2; i <= len(nums); i++ {
                m[i] = (m[i-1] * 2) % 1000000007
        }
        res := 0
        for i := 0; i < len(nums); i++ {
                left, right := i, len(nums)
                for left+1 < right {
                        mid := left + (right-left)/2
                        if nums[mid]+nums[i] <= target {
                                left = mid
                        } else {
                                right = mid
                        }
                }
                if nums[left]+nums[i] <= target {
                        length := left - i + 1
                        res = res + m[length]
                }
        }
        return res % 1000000007
}
```

CHAPTER 45

1401-1500-Hard

45.1 1402. 做菜顺序 (3)

- 题目

```
一个厨师收集了他n道菜的满意程度satisfaction，这个厨师做出每道菜的时间都是 1 单位时间。
一道菜的␣
↪「喜爱时间」系数定义为烹饪这道菜以及之前每道菜所花费的时间乘以这道菜的满意程度，
也就是time[i]*satisfaction[i]。
请你返回做完所有菜 「喜爱时间」总和的最大值为多少。
你可以按任意顺序安排做菜的顺序，你也可以选择放弃做某些菜来获得更大的总和。
示例 1：输入：satisfaction = [-1,-8,0,5,-9] 输出：14
解释：去掉第二道和最后一道菜，最大的喜爱时间系数和为 (-1*1 + 0*2 + 5*3 = 14) 。
每道菜都需要花费 1 单位时间完成。
示例 2：输入：satisfaction = [4,3,2] 输出：20
解释：按照原来顺序相反的时间做菜 (2*1 + 3*2 + 4*3 = 20)
示例 3：输入：satisfaction = [-1,-4,-5] 输出：0
解释：大家都不喜欢这些菜，所以不做任何菜可以获得最大的喜爱时间系数。
示例 4：输入：satisfaction = [-2,5,-1,0,3,-3] 输出：35
提示：n == satisfaction.length
1 <= n <= 500
-10^3 <=satisfaction[i] <= 10^3
```

- 解题思路

```
func maxSatisfaction(satisfaction []int) int {
        res := 0
        sort.Slice(satisfaction, func(i, j int) bool {
                return satisfaction[i] > satisfaction[j]
        })
        sum := 0
        for i := 0; i < len(satisfaction); i++ {
                if sum+satisfaction[i] <= 0 {
                        break
                }
                sum = sum + satisfaction[i]
                res = res + sum //␣
↪每多遍历1次，前后相邻的2个数，较大数相对较小数多+1次
        }
        return res
}

# 2
func maxSatisfaction(satisfaction []int) int {
        res := 0
        sort.Slice(satisfaction, func(i, j int) bool {
                return satisfaction[i] > satisfaction[j]
        })
        sum := 0
        temp := 0
        for i := 0; i < len(satisfaction); i++ {
                sum = sum + satisfaction[i]
                temp = temp + sum //␣
↪每多遍历1次，前后相邻的2个数，较大数相对较小数多+1次
                if temp > res {
                        res = temp
                }
        }
        return res
}

# 3
func maxSatisfaction(satisfaction []int) int {
        sort.Slice(satisfaction, func(i, j int) bool {
                return satisfaction[i] > satisfaction[j]
        })
        if satisfaction[0] <= 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        n := len(satisfaction)
        dp := make([]int, n)
        dp[0] = satisfaction[0]
        sum := satisfaction[0]
        for i := 1; i < n; i++ {
                sum = sum + satisfaction[i]
                dp[i] = max(dp[i-1], dp[i-1]+sum)
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

45.2 1411. 给 Nx3 网格图涂色的方案数 (2)

- 题目

```
你有一个 n x 3的网格图 grid，你需要用 红，黄，绿三种颜色之一给每一个格子上色，
且确保相邻格子颜色不同（也就是有相同水平边或者垂直边的格子颜色不同）。
给你网格图的行数 n。
请你返回给grid涂色的方案数。由于答案可能会非常大，请你返回答案对10^9 + 7取余的结果。
示例 1：输入：n = 1 输出：12
解释：总共有 12 种可行的方法：
示例 2：输入：n = 2 输出：54
示例 3：输入：n = 3 输出：246
示例 4：输入：n = 7 输出：106494
示例 5：输入：n = 5000 输出：30228214
提示：n == grid.length
grid[i].length == 3
1 <= n <= 5000
```

- 解题思路

```
var mod = 1000000007

func numOfWays(n int) int {
        arr := make([]int, 0) // 保存所有满足条件的排列
```

(续下页)

(接上页)

```
        for i := 0; i < 3; i++ {
                for j := 0; j < 3; j++ {
                        for k := 0; k < 3; k++ {
                                if i != j && j != k {
                                        arr = append(arr, i*100+j*10+k)
                                }
                        }
                }
        }
        length := len(arr)
        judgeArr := make([][]int, length) // 相邻关系判断：1代表相邻行可以相邻
        for i := 0; i < length; i++ {
                judgeArr[i] = make([]int, length)
                for j := 0; j < length; j++ {
                        if arr[i]/100 != arr[j]/100 &&
                                arr[i]/10%10 != arr[j]/10%10 &&
                                arr[i]%10 != arr[j]%10 {
                                judgeArr[i][j] = 1
                        }
                }
        }
        // 上面是预处理
        dp := make([][]int, n+1) // dp[i][j]表示：i个x3网格，最后一行是呈现的是j的方案数
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, length)
        }
        for j := 0; j < length; j++ {
                dp[1][j] = 1 // 第一行可以使用任何类型
        }
        for i := 2; i <= n; i++ {
                for j := 0; j < length; j++ {
                        for k := 0; k < length; k++ {
                                if judgeArr[j][k] == 1 {
                                        dp[i][j] = (dp[i][j] + dp[i-1][k]) % mod
                                }
                        }
                }
        }
        res := 0
        for j := 0; j < length; j++ {
                res = (res + dp[n][j]) % mod
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
var mod = 1000000007

func numOfWays(n int) int {
        // 满足要求的组合12种
        // 6种互不相同：012, 021, 102, 120, 201, 210
        // 6种带有相同：010, 020, 101, 121, 202, 212
        a := 6 // 第1行是互不相同的
        b := 6 // 第1行带相同的
        for i := 2; i <= n; i++ {
                x := (2*a + 2*b) % mod // 当前行是互不相同的
                y := (2*a + 3*b) % mod // 当前行是带相同的
                a = x
                b = y
        }
        return (a + b) % mod
}
```

45.3 1420. 生成数组 (2)

- 题目

```
给你三个整数 n、m 和 k 。下图描述的算法用于找出正整数数组中最大的元素。
请你生成一个具有下述属性的数组 arr ：
arr 中有 n 个整数。
1 <= arr[i] <= m 其中 (0 <= i < n) 。
将上面提到的算法应用于 arr ，search_cost 的值等于 k 。
返回上述条件下生成数组 arr 的 方法数 ，由于答案可能会很大，所以 必须 对 10^9 + 7␣
↪取余。
示例 1：输入：n = 2, m = 3, k = 1 输出：6
解释：可能的数组分别为 [1, 1], [2, 1], [2, 2], [3, 1], [3, 2] [3, 3]
示例 2：输入：n = 5, m = 2, k = 3 输出：0
解释：没有数组可以满足上述条件
示例 3：输入：n = 9, m = 1, k = 1 输出：1
解释：可能的数组只有 [1, 1, 1, 1, 1, 1, 1, 1, 1]
示例 4：输入：n = 50, m = 100, k = 25 输出：34549172
解释：不要忘了对 1000000007 取余
示例 5：输入：n = 37, m = 17, k = 7 输出：418930126
提示：1 <= n <= 50
```

(续下页)

(接上页)

```
1 <= m <= 100
0 <= k <= n
```

- 解题思路

```
var mod = 1000000007

func numOfArrays(n int, m int, k int) int {
        if k == 0 {
                return 0
        }
        dp := make([][][]int, n+1) // 数组第i位最大值为j，比较次数为k的结果
        for i := 0; i <= n; i++ {
                dp[i] = make([][]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = make([]int, k+1)
                }
        }
        for i := 1; i <= m; i++ {
                dp[1][i][1] = 1
        }
        for i := 2; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        for a := 1; a <= k; a++ {
                                for b := 1; b < j; b++ { // 比j小,才可以次数+1
                                        dp[i][j][a] = (dp[i][j][a] + dp[i-1][b][a-1])
↪% mod
                                }
                                dp[i][j][a] = (dp[i][j][a] + dp[i-1][j][a]*j) % mod //
↪ 跟j相同，可选择[1,j]共j个数
                        }
                }
        }
        res := 0
        for i := 1; i <= m; i++ {
                res = (res + dp[n][i][k]) % mod
        }
        return res
}

# 2
var mod = 1000000007

func numOfArrays(n int, m int, k int) int {
```

(续下页)

(接上页)

```
        if k == 0 {
                return 0
        }
        dp := make([][][]int, n+1) // 数组第i位最大值为j, 比较次数为k的结果
        for i := 0; i <= n; i++ {
                dp[i] = make([][]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = make([]int, k+1)
                }
        }
        for i := 1; i <= m; i++ {
                dp[1][i][1] = 1
        }
        for i := 2; i <= n; i++ {
                for a := 1; a <= k; a++ {
                        tempSum := 0
                        for j := 1; j <= m; j++ {
                                dp[i][j][a] = (dp[i][j][a] + dp[i-1][j][a]*j) % mod //
↪ 跟j相同, 可选择[1,j]共j个数
                                dp[i][j][a] = (dp[i][j][a] + tempSum) % mod         //
↪ 跟j不同
                                tempSum = (tempSum + dp[i-1][j][a-1]) % mod
                        }
                }
        }
        res := 0
        for i := 1; i <= m; i++ {
                res = (res + dp[n][i][k]) % mod
        }
        return res
}
```

45.4 1425. 带限制的子序列和 (4)

- 题目

```
给你一个整数数组nums和一个整数k, 请你返回 非空子序列元素和的最大值, 子序列需要满足:
子序列中每两个 相邻的整数nums[i]和nums[j],
它们在原数组中的下标i和j满足i < j且 j - i <= k 。
数组的子序列定义为: 将数组中的若干个数字删除 (可以删除 0␣
↪个数字), 剩下的数字按照原本的顺序排布。
示例 1: 输入: nums = [10,2,-10,5,20], k = 2 输出: 37
```

(续下页)

(接上页)

```
解释：子序列为 [10, 2, 5, 20] 。
示例 2：输入：nums = [-1,-2,-3], k = 1 输出：-1
解释：子序列必须是非空的，所以我们选择最大的数字。
示例 3：输入：nums = [10,-2,-10,-5,20], k = 2 输出：23
解释：子序列为 [10, -2, -5, 20] 。
提示：1 <= k <= nums.length <= 10^5
-10^4<= nums[i] <= 10^4
```

- 解题思路

```
func constrainedSubsetSum(nums []int, k int) int {
        n := len(nums)
        dp := make([]int, n)
        if k > n {
                k = n
        }
        res := nums[0]
        dp[0] = nums[0]
        maxValue := nums[0]
        for i := 1; i < len(nums); i++ {
                if i <= k { // 在前k个，dp[i] = maxValue + nums[i]
                        if maxValue < 0 {
                                dp[i] = nums[i]
                        } else {
                                dp[i] = maxValue + nums[i]
                        }
                        maxValue = max(maxValue, dp[i])
                } else {
                        if maxValue == dp[i-1-k] { // 需要重新找maxValue
                                maxValue = getMaxValue(dp[i-k : i])
                        }
                        if maxValue < 0 {
                                dp[i] = nums[i]
                        } else {
                                dp[i] = maxValue + nums[i]
                        }
                        maxValue = max(maxValue, dp[i])
                }
                res = max(res, dp[i])
        }
        return res
}

func getMaxValue(arr []int) int {
```

(续下页)

(接上页)

```
        maxValue := arr[0]
        for i := 0; i < len(arr); i++ {
                if arr[i] > maxValue {
                        maxValue = arr[i]
                }
        }
        return maxValue
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func constrainedSubsetSum(nums []int, k int) int {
        n := len(nums)
        dp := make([]int, n)
        if k > n {
                k = n
        }
        dp[0] = nums[0]
        maxValue := nums[0]
        maxIndex := 0
        res := nums[0]
        for i := 1; i < len(nums); i++ {
                if i <= k { // 在前k个, dp[i] = maxValue + nums[i]
                        if maxValue < 0 {
                                dp[i] = nums[i]
                        } else {
                                dp[i] = maxValue + nums[i]
                        }
                        if dp[i] >= maxValue {
                                maxValue = dp[i]
                                maxIndex = i
                        }
                } else {
                        if i-k > maxIndex {
                                maxValue = dp[maxIndex+1]
                                for j := maxIndex + 1; j < i; j++ {
                                        if dp[j] >= maxValue {
```

(续下页)

(接上页)

```
                                                maxValue = dp[j]
                                                maxIndex = j
                                        }
                                }
                        }
                        if maxValue < 0 {
                                dp[i] = nums[i]
                        } else {
                                dp[i] = maxValue + nums[i]
                        }
                        if dp[i] >= maxValue {
                                maxValue = dp[i]
                                maxIndex = i
                        }
                }
                if dp[i] > res {
                        res = dp[i]
                }
        }
        return res
}

# 3
func constrainedSubsetSum(nums []int, k int) int {
        n := len(nums)
        if k > n {
                k = n
        }
        temp := nums[0]
        res := nums[0]
        stack := make([][2]int, 0)
        stack = append(stack, [2]int{0, nums[0]})
        for i := 1; i < len(nums); i++ {
                if stack[0][0] < i-k {
                        stack = stack[1:]
                }
                if stack[0][1] < 0 {
                        temp = nums[i]
                } else {
                        temp = stack[0][1] + nums[i]
                }
                for len(stack) > 0 && stack[len(stack)-1][1] < temp {
                        stack = stack[:len(stack)-1]
```

(续下页)

(接上页)

```
                }
                stack = append(stack, [2]int{i, temp})
                if temp > res {
                        res = temp
                }
        }
        return res
}

# 4
func constrainedSubsetSum(nums []int, k int) int {
        n := len(nums)
        if k > n {
                k = n
        }
        res := nums[0]
        temp := nums[0]
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [2]int{0, nums[0]})
        for i := 1; i < len(nums); i++ {
                for i-intHeap[0][0] > k { // 不满足删除
                        heap.Pop(&intHeap)
                }
                if intHeap[0][1] < 0 {
                        temp = nums[i]
                } else {
                        temp = intHeap[0][1] + nums[i]
                }
                if temp > res {
                        res = temp
                }
                heap.Push(&intHeap, [2]int{i, temp})
        }
        return res
}

type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}
```

(续下页)

(接上页)

```
// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][1] > h[j][1]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

45.5 1434. 每个人戴不同帽子的方案数 (3)

- 题目

```
总共有 n个人和 40 种不同的帽子，帽子编号从 1 到 40 。
给你一个整数列表的列表hats，其中hats[i]是第 i个人所有喜欢帽子的列表。
请你给每个人安排一顶他喜欢的帽子，确保每个人戴的帽子跟别人都不一样，并返回方案数。
由于答案可能很大，请返回它对10^9 + 7取余后的结果。
示例 1：输入：hats = [[3,4],[4,5],[5]] 输出：1
解释：给定条件下只有一种方法选择帽子。
第一个人选择帽子 3，第二个人选择帽子 4，最后一个人选择帽子 5。
示例 2：输入：hats = [[3,5,1],[3,5]] 输出：4
解释：总共有 4 种安排帽子的方法：
(3,5)，(5,3)，(1,3) 和 (1,5)
示例 3：输入：hats = [[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]] 输出：24
解释：每个人都可以从编号为 1 到 4 的帽子中选。
(1,2,3,4) 4 个帽子的排列方案数为 24 。
示例 4：输入：hats = [[1,2,3],[2,3,5,6],[1,3,7,9],[1,8,9],[2,5,7]] 输出：111
提示：n == hats.length
1 <= n <= 10
1 <= hats[i].length <= 40
1 <= hats[i][j] <= 40
hats[i]包含一个数字互不相同的整数列表。
```

- 解题思路

```
var mod = 1000000007

func numberWays(hats [][]int) int {
        n := len(hats)                  // n个人
        maxValue := 0                   // 最大的帽子编号，从1开始
        m := make(map[int]map[int]bool) // 帽子对应人的喜欢关系
        for i := 0; i < n; i++ {
                for j := 0; j < len(hats[i]); j++ {
                        id := hats[i][j]
                        maxValue = max(maxValue, id)
                        if m[id] == nil {
                                m[id] = make(map[int]bool)
                        }
                        m[id][i] = true
                }
        }
        dp := make([][]int, maxValue+1) // dp[i][j] 表示第i个帽子状态为j的方案数
        for i := 0; i <= maxValue; i++ {
                dp[i] = make([]int, 1<<n)
        }
        dp[0][0] = 1
        target := (1 << n) - 1 // n个1
        for i := 1; i <= maxValue; i++ {
                for j := 0; j <= target; j++ { // 状态为0~2^n-1
                        dp[i][j] = dp[i-1][j]
                        for k := range m[i] { // 对第i个帽子喜欢
                                if (j>>k)&1 == 1 { // 判断状态j的第k位是否是1
                                        // j ^ (1<<k) : 第k个人没有分配帽子
                                        dp[i][j] = (dp[i][j] + dp[i-1][j^(1<<k)]) %␣
↪mod
                                }
                        }
                }
        }
        return dp[maxValue][target]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
var mod = 1000000007

func numberWays(hats [][]int) int {
        n := len(hats)                    // n个人
        m := make(map[int]map[int]bool) // 帽子对应人的喜欢关系
        for i := 0; i < n; i++ {
                for j := 0; j < len(hats[i]); j++ {
                        id := hats[i][j]
                        if m[id] == nil {
                                m[id] = make(map[int]bool)
                        }
                        m[id][i] = true
                }
        }
        target := (1 << n) - 1      // n个1
        dp := make([]int, target+1) // dp[i][j] 表示第i个帽子状态为j的方案数
        dp[0] = 1
        for i := 1; i <= 40; i++ {
                for j := target; j >= 0; j-- { // 状态为0~2^n-1
                        for k := range m[i] { // 对第i个帽子喜欢
                                if (j>>k)&1 == 0 { // 判断状态j的第k位是否是0
                                        // j+1<<k : 第k个人没有分配帽子，然后分配帽子
                                        next := j + 1<<k
                                        dp[next] = (dp[j] + dp[next]) % mod
                                }
                        }
                }
        }
        return dp[target]
}

# 3
var mod = 1000000007

func numberWays(hats [][]int) int {
        n := len(hats)
        target := (1 << n) - 1
        dp := make([]int, target+1)
        dp[0] = 1
        arr := make([][]int, 41)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := range hats[i] {
                        arr[hats[i][j]] = append(arr[hats[i][j]], i)
                }
        }
        for i := 1; i <= 40; i++ {
                for j := target - 1; j >= 0; j-- {
                        for _, k := range arr[i] {
                                if j&(1<<k) == 0 {
                                        dp[j|(1<<k)] = dp[j|(1<<k)] + dp[j]
                                }
                        }
                }
        }
        return dp[target] % mod
}
```

45.6 1439. 有序矩阵中的第 k 个最小数组和

45.6.1 题目

```
给你一个 m* n 的矩阵 mat，以及一个整数 k ，矩阵中的每一行都以非递减的顺序排列。
你可以从每一行中选出 1 个元素形成一个数组。返回所有可能数组中的第 k 个 最小 数组和。
示例 1: 输入: mat = [[1,3,11],[2,4,6]], k = 5输出: 7
解释: 从每一行中选出一个元素，前 k 个和最小的数组分别是:
[1,2], [1,4], [3,2], [3,4], [1,6]。其中第 5 个的和是 7 。
示例 2: 输入: mat = [[1,3,11],[2,4,6]], k = 9输出: 17
示例 3: 输入: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7 输出: 9
解释: 从每一行中选出一个元素，前 k 个和最小的数组分别是:
[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]。其中第 7 个的和是 9 。
示例 4: 输入: mat = [[1,1,10],[2,2,9]], k = 7 输出: 12
提示: m == mat.length
n == mat.length[i]
1 <= m, n <= 40
1 <= k <= min(200, n ^m)
1 <= mat[i][j] <= 5000
mat[i] 是一个非递减数组
```

45.6.2 解题思路

45.7 1449. 数位成本和为目标值的最大数字 (3)

- 题目

```
给你一个整数数组cost和一个整数target。请你返回满足如下规则可以得到的最大整数:
给当前结果添加一个数位 (i + 1) 的成本为cost[i] (cost数组下标从 0 开始) 。
总成本必须恰好等于target。
添加的数位中没有数字 0 。
由于答案可能会很大, 请你以字符串形式返回。
如果按照上述要求无法得到任何整数, 请你返回 "0" 。
示例 1: 输入: cost = [4,3,2,5,6,7,2,5,5], target = 9 输出: "7772"
解释: 添加数位 '7' 的成本为 2 , 添加数位 '2' 的成本为 3 。
所以 "7772" 的代价为 2*3+ 3*1 = 9 。 "977" 也是满足要求的数字, 但 "7772"␣
↪是较大的数字。
 数字      成本
  1  ->   4
  2  ->   3
  3  ->   2
  4  ->   5
  5  ->   6
  6  ->   7
  7  ->   2
  8  ->   5
  9  ->   5
示例 2: 输入: cost = [7,6,5,5,5,6,8,7,8], target = 12 输出: "85"
解释: 添加数位 '8' 的成本是 7 , 添加数位 '5' 的成本是 5 。"85" 的成本为 7 + 5 = 12 。
示例 3: 输入: cost = [2,4,6,2,4,6,4,4,4], target = 5 输出: "0"
解释: 总成本是 target 的条件下, 无法生成任何整数。
示例 4: 输入: cost = [6,10,15,40,40,40,40,40,40], target = 47 输出: "32211"
提示: cost.length == 9
1 <= cost[i] <= 5000
1 <= target <= 5000
```

- 解题思路

```
func largestNumber(cost []int, target int) string {
        dp, path := make([][]int, 10), make([][]int, 10)
        for i := 0; i < 10; i++ {
                dp[i], path[i] = make([]int, target+1), make([]int, target+1)
```

(续下页)

(接上页)

```
                for j := 0; j <= target; j++ {
                        dp[i][j] = math.MinInt32
                }
        }
        dp[0][0] = 0 // dp[i+1][j] 表示使用前i个数且花费总代价恰好为j时的最大长度
        for i := 0; i < len(cost); i++ {
                for j := 0; j <= target; j++ {
                        if j < cost[i] { // 不够
                                dp[i+1][j] = dp[i][j]
                                path[i+1][j] = j
                        } else {
                                // 不选:dp[i][j]、选:dp[i+1][j-cost[i]]
                                if dp[i][j] > dp[i+1][j-cost[i]]+1 { // 不选
                                        dp[i+1][j] = dp[i][j]
                                        path[i+1][j] = j
                                } else { // 选：相等时，也要更新，cost[i]相同取更大i
                                        dp[i+1][j] = dp[i+1][j-cost[i]] + 1
                                        path[i+1][j] = j - cost[i]
                                }
                        }
                }
        }
        if dp[9][target] < 0 {
                return "0"
        }
        res := ""
        i, j := 9, target
        for i > 0 {
                if j == path[i][j] {
                        i--
                } else {
                        res = res + string('0'+i)
                        j = path[i][j]
                }
        }
        return res
}

# 2
func largestNumber(cost []int, target int) string {
        dp := make([]int, target+1)
        for i := 0; i <= target; i++ {
                dp[i] = math.MinInt32
```

(续下页)

(接上页)

```
        }
        dp[0] = 0 // dp[i]表示花费总代价恰好为i时的最大长度
        for i := 0; i < len(cost); i++ {
                for j := cost[i]; j <= target; j++ {
                        dp[j] = max(dp[j], dp[j-cost[i]]+1)
                }
        }
        if dp[target] < 0 {
                return "0"
        }
        res := ""
        j := target
        for i := 8; i >= 0; i-- {
                for c := cost[i]; j >= c && dp[j] == dp[j-c]+1; j = j - c {
                        res = res + string('1'+i)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
// 有x个物品，求给定费用target的前提下，花光所有费用所能选择的最大物品个数为多少
func largestNumber(cost []int, target int) string {
        dp := make([]string, target+1)
        for i := 1; i <= target; i++ {
                dp[i] = "0"
                for j := 8; j >= 0; j-- {
                        if cost[j] > i || (i-cost[j] != 0 && dp[i-cost[j]] == "0") {
                                continue
                        }
                        a, b := dp[i], dp[i-cost[j]]+fmt.Sprintf("%d", j+1)
                        if len(a) < len(b) {
                                dp[i] = b
                        } else if len(a) == len(b) && a < b {
                                dp[i] = b
                        }
```

(续下页)

(接上页)

```
                }
        }
        return dp[target]
}
```

45.8 1458. 两个子序列的最大点积 (2)

- 题目

```
给你两个数组nums1和nums2。
请你返回 nums1 和 nums2 中两个长度相同的 非空 子序列的最大点积。
数组的非空子序列是通过删除原数组中某些元素（可能一个也不删除）后剩余数字组成的序列，
但不能改变数字间相对顺序。比方说，[2,3,5]是[1,2,3,4,5]的一个子序列而[1,5,3]不是。
示例 1：输入：nums1 = [2,1,-2,5], nums2 = [3,0,-6] 输出：18
解释：从 nums1 中得到子序列 [2,-2] ，从 nums2 中得到子序列 [3,-6] 。
它们的点积为 (2*3 + (-2)*(-6)) = 18 。
示例 2：输入：nums1 = [3,-2], nums2 = [2,-6,7] 输出：21
解释：从 nums1 中得到子序列 [3] ，从 nums2 中得到子序列 [7] 。
它们的点积为 (3*7) = 21 。
示例 3：输入：nums1 = [-1,-1], nums2 = [1,1] 输出：-1
解释：从 nums1 中得到子序列 [-1] ，从 nums2 中得到子序列 [1] 。
它们的点积为 -1 。
提示：1 <= nums1.length, nums2.length <= 500
-1000 <= nums1[i], nums2[i] <= 100
点积：定义 a= [a1,a2,…,an] 和 b= [b1,b2,…,bn] 的点积为：
这里的 Σ 指示总和符号。
```

- 解题思路

```
func maxDotProduct(nums1 []int, nums2 []int) int {
        n, m := len(nums1), len(nums2)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        value := nums1[i] * nums2[j] // 单独选对应的i,j 乘积
                        dp[i][j] = value             // 至少选一对
                        if i > 0 {
                                dp[i][j] = max(dp[i][j], dp[i-1][j])
                        }
```

(续下页)

(接上页)

```
                        if j > 0 {
                                dp[i][j] = max(dp[i][j], dp[i][j-1])
                        }
                        if i > 0 && j > 0 {
                                dp[i][j] = max(dp[i][j], dp[i-1][j-1]+value)
                        }
                }
        }
        return dp[n-1][m-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxDotProduct(nums1 []int, nums2 []int) int {
        n, m := len(nums1), len(nums2)
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = math.MinInt32
                }
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        value := nums1[i-1] * nums2[j-1] // 单独选对应的i,j 乘积
                        dp[i][j] = max(dp[i][j], value)  // 至少选一对
                        dp[i][j] = max(dp[i][j], dp[i-1][j-1]+value)
                        dp[i][j] = max(dp[i][j], dp[i-1][j])
                        dp[i][j] = max(dp[i][j], dp[i][j-1])
                }
        }
        return dp[n][m]
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}
```

45.9 1478. 安排邮筒

45.9.1 题目

```
给你一个房屋数组houses和一个整数k，其中houses[i]是第
↪i栋房子在一条街上的位置，现需要在这条街上安排 k个邮筒。
请你返回每栋房子与离它最近的邮筒之间的距离的 最小 总和。
答案保证在 32 位有符号整数范围以内。
示例 1：输入：houses = [1,4,8,10,20], k = 3 输出：5
解释：将邮筒分别安放在位置 3， 9 和 20 处。
每个房子到最近邮筒的距离和为 |3-1| + |4-3| + |9-8| + |10-9| + |20-20| = 5 。
示例 2：输入：houses = [2,3,5,12,18], k = 2 输出：9
解释：将邮筒分别安放在位置 3 和 14 处。
每个房子到最近邮筒距离和为 |2-3| + |3-3| + |5-3| + |12-14| + |18-14| = 9 。
示例 3：输入：houses = [7,4,6,1], k = 1 输出：8
示例 4：输入：houses = [3,6,14,10], k = 4 输出：0
提示：n == houses.length
1 <= n<= 100
1 <= houses[i] <= 10^4
1 <= k <= n
数组houses中的整数互不相同。
```

45.9.2 解题思路

```

```

45.10 1483. 树节点的第 K 个祖先 (1)

- 题目

```
给你一棵树，树上有 n 个节点，按从 0 到 n-1 编号。
树以父节点数组的形式给出，其中 parent[i] 是节点 i 的父节点。树的根节点是编号为 0
↪的节点。
请你设计并实现 getKthAncestor(int node, int k) 函数，函数返回节点 node 的第 k
```

(续下页)

(接上页)

```
↪个祖先节点。
如果不存在这样的祖先节点，返回 -1 。
树节点的第 k 个祖先节点是从该节点到根节点路径上的第 k 个节点。
示例：输入：["TreeAncestor","getKthAncestor","getKthAncestor","getKthAncestor"]
[[7,[-1,0,0,1,1,2,2]],[3,1],[5,2],[6,3]]
输出：[null,1,0,-1]
解释：TreeAncestor treeAncestor = new TreeAncestor(7, [-1, 0, 0, 1, 1, 2, 2]);
treeAncestor.getKthAncestor(3, 1);  // 返回 1 ，它是 3 的父节点
treeAncestor.getKthAncestor(5, 2);  // 返回 0 ，它是 5 的祖父节点
treeAncestor.getKthAncestor(6, 3);  // 返回 -1 因为不存在满足要求的祖先节点
提示：1 <= k <=n <= 5*10^4
parent[0] == -1 表示编号为 0 的节点是根节点。
对于所有的 0 <i < n ，0 <= parent[i] < n 总成立
0 <= node < n
至多查询 5*10^4 次
```

- 解题思路

```
type TreeAncestor struct {
        dp [][]int
}

func Constructor(n int, parent []int) TreeAncestor {
        m := 0
        for i := 1; i <= n; i = i * 2 {
                m++
        }
        dp := make([][]int, n) // dp[i][j] => i的第2^j个父节点
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m+1)
                for j := 0; j <= m; j++ {
                        dp[i][j] = -1
                }
        }
        for i := 0; i < n; i++ {
                dp[i][0] = parent[i]
        }
        for i := 0; i < n; i++ {
                for j := 1; j < m; j++ {
                        if dp[i][j-1] == -1 { // 没有父节点
                                continue
                        }
                        prev := dp[i][j-1] // 状态转移方程：dp[i][j] = dp[prev][j-1]
                        dp[i][j] = dp[prev][j-1]
```

(续下页)

(接上页)

```
                }
        }
        return TreeAncestor{dp: dp}
}

func (this *TreeAncestor) GetKthAncestor(node int, k int) int {
        for i := 0; i < 16; i++ {
                if k&(1<<i) > 0 {
                        node = this.dp[node][i]
                }
                if node == -1 {
                        break
                }
        }
        return node
}
```

45.11 1494. 并行课程 II

45.11.1 题目

```
给你一个整数n表示某所大学里课程的数目，编号为1到n，数组dependencies中，
dependencies[i] = [xi, yi]␣
↪表示一个先修课的关系，也就是课程xi必须在课程yi之前上。同时你还有一个整数k。
在一个学期中，你 最多可以同时上 k门课，前提是这些课的先修课在之前的学期里已经上过了。
请你返回上完所有课最少需要多少个学期。题目保证一定存在一种上完所有课的方式。
示例 1: 输入: n = 4, dependencies = [[2,1],[3,1],[1,4]], k = 2 输出: 3
解释: 上图展示了题目输入的图。在第一个学期中，我们可以上课程 2 和课程 3␣
↪。然后第二个学期上课程 1 ，第三个学期上课程 4 。
示例 2: 输入: n = 5, dependencies = [[2,1],[3,1],[4,1],[1,5]], k = 2 输出: 4
解释: 上图展示了题目输入的图。一个最优方案是: 第一学期上课程 2 和 3，第二学期上课程 4␣
↪，第三学期上课程 1 ，第四学期上课程 5 。
示例 3: 输入: n = 11, dependencies = [], k = 2 输出: 6
提示: 1 <= n <= 15
1 <= k <= n
0 <=dependencies.length <= n * (n-1) / 2
dependencies[i].length == 2
1 <= xi, yi<= n
xi != yi
所有先修关系都是不同的，也就是说dependencies[i] != dependencies[j]。
题目输入的图是个有向无环图。
```

45.11.2 解题思路

CHAPTER 46

1501-1600-Easy

46.1 1502. 判断能否形成等差数列 (2)

- 题目

```
给你一个数字数组 arr 。
如果一个数列中，任意相邻两项的差总等于同一个常数，那么这个数列就称为 等差数列 。
如果可以重新排列数组形成等差数列，请返回 true ；否则，返回 false 。
示例 1：输入：arr = [3,5,1] 输出：true
解释：对数组重新排序得到 [1,3,5] 或者 [5,3,1] ，任意相邻两项的差分别为 2 或 -2␣
↪，可以形成等差数列。
示例 2：输入：arr = [1,2,4] 输出：false
解释：无法通过重新排序得到等差数列。
提示：
    2 <= arr.length <= 1000
    -10^6 <= arr[i] <= 10^6
```

- 解题思路

```
func canMakeArithmeticProgression(arr []int) bool {
        sort.Ints(arr)
        diff := arr[1] - arr[0]
        for i := 2; i < len(arr); i++ {
                if arr[i]-arr[i-1] != diff {
                        return false
```

(续下页)

(接上页)

```
                }
        }
        return true
}

#
func canMakeArithmeticProgression(arr []int) bool {
        min, max := arr[0], arr[0]
        m := make(map[int]bool)
        for i := 0; i < len(arr); i++ {
                if arr[i] <= min {
                        min = arr[i]
                }
                if arr[i] >= max {
                        max = arr[i]
                }
                m[arr[i]] = true
        }
        diff := (max - min) / (len(arr) - 1)
        for i := 0; i < len(m); i++ {
                value := min + diff*i
                if _, ok := m[value]; !ok {
                        return false
                }
        }
        return true
}
```

46.2 1507. 转变日期格式 (1)

- 题目

```
给你一个字符串 date ，它的格式为 Day Month Year ，其中：
    Day 是集合 {"1st", "2nd", "3rd", "4th", ..., "30th", "31st"} 中的一个元素。
    Month 是集合 {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
    "Aug", "Sep", "Oct", "Nov", "Dec"} 中的一个元素。
    Year 的范围在 [1900, 2100] 之间。
请你将字符串转变为 YYYY-MM-DD 的格式，其中：
    YYYY 表示 4 位的年份。
    MM 表示 2 位的月份。
    DD 表示 2 位的天数。
示例 1: 输入: date = "20th Oct 2052" 输出: "2052-10-20"
```

(续下页)

(接上页)

```
示例 2: 输入: date = "6th Jun 1933" 输出: "1933-06-06"
示例 3: 输入: date = "26th May 1960" 输出: "1960-05-26"
提示: 给定日期保证是合法的, 所以不需要处理异常输入。
```

- 解题思路

```
func reformatDate(date string) string {
        month := []string{"", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
↪"Sep", "Oct", "Nov", "Dec"}
        arr := strings.Split(date, " ")
        res := arr[2] + "-"
        for i := 1; i < len(month); i++ {
                if month[i] == arr[1] {
                        res = res + fmt.Sprintf("%02d", i) + "-"
                }
        }
        /*
                if arr[0][1] >= '0' && arr[0][1] <= '9' {
                        res = res + arr[0][:2]
                } else {
                        res = res + "0" + arr[0][:1]
                }
        */
        if len(arr[0]) == 3 {
                res = res + "0" + arr[0][:1]
        } else {
                res = res + arr[0][:2]
        }
        return res
}
```

46.3 1512. 好数对的数目 (3)

- 题目

```
给你一个整数数组 nums 。
如果一组数字 (i,j) 满足 nums[i] == nums[j] 且 i < j , 就可以认为这是一组 好数对 。
返回好数对的数目。
示例 1: 输入: nums = [1,2,3,1,1,3] 输出: 4
解释: 有 4 组好数对, 分别是 (0,3), (0,4), (3,4), (2,5) , 下标从 0 开始
示例 2: 输入: nums = [1,1,1,1] 输出: 6
解释: 数组中的每组数字都是好数对
```

(续下页)

(接上页)

```
示例 3: 输入: nums = [1,2,3] 输出: 0
提示:
    1 <= nums.length <= 100
    1 <= nums[i] <= 100
```

- 解题思路

```
func numIdenticalPairs(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i] == nums[j] {
                                res++
                        }
                }
        }
        return res
}

#
func numIdenticalPairs(nums []int) int {
        res := 0
        arr := make([]int, 101)
        for i := 0; i < len(nums); i++ {
                res = res + arr[nums[i]]
                arr[nums[i]]++
        }
        return res
}

#
func numIdenticalPairs(nums []int) int {
        res := 0
        m := make(map[int]int, 101)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for _, v := range m {
                if v > 0 {
                        res = res + v*(v-1)/2
                }
        }
        return res
}
```

46.4 1518. 换酒问题 (2)

- 题目

```
小区便利店正在促销，用 numExchange 个空酒瓶可以兑换一瓶新酒。你购入了 numBottles␣
↪瓶酒。
如果喝掉了酒瓶中的酒，那么酒瓶就会变成空的。
请你计算 最多 能喝到多少瓶酒。
示例 1：输入：numBottles = 9, numExchange = 3 输出：13
解释：你可以用 3 个空酒瓶兑换 1 瓶酒。
所以最多能喝到 9 + 3 + 1 = 13 瓶酒。
示例 2：输入：numBottles = 15, numExchange = 4 输出：19
解释：你可以用 4 个空酒瓶兑换 1 瓶酒。
所以最多能喝到 15 + 3 + 1 = 19 瓶酒。
示例 3：输入：numBottles = 5, numExchange = 5 输出：6
示例 4：输入：numBottles = 2, numExchange = 3 输出：2
提示：
    1 <= numBottles <= 100
    2 <= numExchange <= 100
```

- 解题思路

```
func numWaterBottles(numBottles int, numExchange int) int {
        res := numBottles
        for numBottles > 0 {
                times := numBottles / numExchange
                res = res + times
                numBottles = numBottles%numExchange + times
                if numBottles < numExchange {
                        break
                }
        }
        return res
}

#
func numWaterBottles(numBottles int, numExchange int) int {
        return numBottles + (numBottles-1)/(numExchange-1)
}
```

46.5 1523. 在区间范围内统计奇数数目 (2)

- 题目

```
给你两个非负整数 low 和 high 。请你返回 low 和 high 之间（包括二者）奇数的数目。
示例 1：输入：low = 3, high = 7 输出：3
解释：3 到 7 之间奇数数字为 [3,5,7] 。
示例 2：输入：low = 8, high = 10输出：1
解释：8 到 10 之间奇数数字为 [9] 。
提示：0 <= low <= high <= 10^9
```

- 解题思路

```
func countOdds(low int, high int) int {
        if low%2 == 1 {
                low = low - 1
        }
        if high%2 == 1 {
                high = high + 1
        }
        return (high - low) / 2
}

#
func countOdds(low int, high int) int {
        if low%2 == 0 && high%2 == 0 {
                return (high - low) / 2
        }
        return (high-low)/2 + 1
}
```

46.6 1528. 重新排列字符串 (1)

- 题目

```
给你一个字符串 s 和一个 长度相同 的整数数组 indices 。
请你重新排列字符串 s ，其中第 i 个字符需要移动到 indices[i] 指示的位置。
返回重新排列后的字符串。
示例 1：输入：s = "codeleet", indices = [4,5,6,7,0,2,1,3] 输出："leetcode"
解释：如图所示，"codeleet" 重新排列后变为 "leetcode" 。
示例 2：输入：s = "abc", indices = [0,1,2] 输出："abc"
解释：重新排列后，每个字符都还留在原来的位置上。
示例 3：输入：s = "aiohn", indices = [3,1,4,2,0] 输出："nihao"
```

(续下页)

(接上页)

```
示例 4: 输入: s = "aaiougrt", indices = [4,0,2,6,7,3,1,5]输出: "arigatou"
示例 5: 输入: s = "art", indices = [1,0,2] 输出: "rat"
提示:
    s.length == indices.length == n
    1 <= n <= 100
    s 仅包含小写英文字母。
    0 <= indices[i] < n
    indices 的所有的值都是唯一的 (也就是说, indices 是整数 0 到 n - 1␣
↪形成的一组排列) 。
```

- 解题思路

```
func restoreString(s string, indices []int) string {
        arr := []byte(s)
        res := make([]byte, len(s))
        for i := 0; i < len(indices); i++ {
                res[indices[i]] = arr[i]
        }
        return string(res)
}
```

46.7 1534. 统计好三元组 (2)

- 题目

```
给你一个整数数组 arr , 以及 a、b 、c 三个整数。请你统计其中好三元组的数量。
如果三元组 (arr[i], arr[j], arr[k]) 满足下列全部条件, 则认为它是一个 好三元组 。
    0 <= i < j < k < arr.length
    |arr[i] - arr[j]| <= a
    |arr[j] - arr[k]| <= b
    |arr[i] - arr[k]| <= c
其中 |x| 表示 x 的绝对值。
返回 好三元组的数量 。
示例 1: 输入: arr = [3,0,1,1,9,7], a = 7, b = 2, c = 3 输出: 4
解释: 一共有 4 个好三元组: [(3,0,1), (3,0,1), (3,1,1), (0,1,1)] 。
示例 2: 输入: arr = [1,1,2,2,3], a = 0, b = 0, c = 1 输出: 0
解释: 不存在满足所有条件的三元组。
提示:
    3 <= arr.length <= 100
    0 <= arr[i] <= 1000
    0 <= a, b, c <= 1000
```

- 解题思路

```
func countGoodTriplets(arr []int, a int, b int, c int) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        for k := j + 1; k < len(arr); k++ {
                                if abs(arr[i], arr[j]) <= a &&
                                        abs(arr[j], arr[k]) <= b &&
                                        abs(arr[i], arr[k]) <= c {
                                        res++
                                }
                        }
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

#
func countGoodTriplets(arr []int, a int, b int, c int) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        if abs(arr[i], arr[j]) > a {
                                continue
                        }
                        for k := j + 1; k < len(arr); k++ {
                                if abs(arr[j], arr[k]) <= b &&
                                        abs(arr[i], arr[k]) <= c {
                                        res++
                                }
                        }
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a - b
        }
        return b - a
}
```

46.8 1539. 第 k 个缺失的正整数 (3)

- 题目

```
给你一个 严格升序排列 的正整数数组 arr 和一个整数 k 。
请你找到这个数组里第 k 个缺失的正整数。
示例 1: 输入: arr = [2,3,4,7,11], k = 5 输出: 9
解释: 缺失的正整数包括 [1,5,6,8,9,10,12,13,...] 。第 5 个缺失的正整数为 9 。
示例 2: 输入: arr = [1,2,3,4], k = 2 输出: 6
解释: 缺失的正整数包括 [5,6,7,...] 。第 2 个缺失的正整数为 6 。
提示: 1 <= arr.length <= 1000
    1 <= arr[i] <= 1000
    1 <= k <= 1000
    对于所有 1 <= i < j <= arr.length 的 i 和 j 满足 arr[i] < arr[j]
```

- 解题思路

```
func findKthPositive(arr []int, k int) int {
        start := 1
        i := 0
        for {
                if i < len(arr) && arr[i] == start {
                        i++
                } else {
                        k--
                        if k == 0 {
                                return start
                        }
                }
                start++
        }
}

# 2
func findKthPositive(arr []int, k int) int {
        for i := 0; i < len(arr); i++ {
                if arr[i]-(i+1) >= k {
```

(续下页)

(接上页)

```
                        return k + i
                }
        }
        return k + len(arr)
}

# 3
func findKthPositive(arr []int, k int) int {
        left := 0
        right := len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid]-(mid+1) >= k {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return k + left
}
```

46.9 1544. 整理字符串 (1)

- 题目

```
给你一个由大小写英文字母组成的字符串 s 。
一个整理好的字符串中，两个相邻字符 s[i] 和 s[i + 1] 不会同时满足下述条件：
    0 <= i <= s.length - 2
    s[i] 是小写字符，但 s[i + 1] 是相同的大写字符；反之亦然 。
请你将字符串整理好，每次你都可以从字符串中选出满足上述条件的 两个相邻␣
↪字符并删除，直到字符串整理好为止。
请返回整理好的 字符串 。题目保证在给出的约束条件下，测试样例对应的答案是唯一的。
注意：空字符串也属于整理好的字符串，尽管其中没有任何字符。
示例 1：输入：s = "leEeetcode" 输出："leetcode"
解释：无论你第一次选的是 i = 1 还是 i = 2，都会使 "leEeetcode" 缩减为 "leetcode" 。
示例 2：输入：s = "abBAcC" 输出：""
解释：存在多种不同情况，但所有的情况都会导致相同的结果。例如：
"abBAcC" --> "aAcC" --> "cC" --> ""
"abBAcC" --> "abBA" --> "aA" --> ""
示例 3：输入：s = "s"输出："s"
提示：  1 <= s.length <= 100
    s 只包含小写和大写英文字母
```

- 解题思路

```
func makeGood(s string) string {
        if len(s) <= 1 {
                return s
        }
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                stack = append(stack, s[i])
                if len(stack) > 1 {
                        length := len(stack)
                        if stack[length-1]-'A'+'a' == stack[length-2] ||
                                stack[length-1]-'a'+'A' == stack[length-2] {
                                stack = stack[:length-2]
                        }
                }
        }
        return string(stack)
}
```

46.10 1550. 存在连续三个奇数的数组 (2)

- 题目

```
给你一个整数数组 arr，请你判断数组中是否存在连续三个元素都是奇数的情况：
如果存在，请返回 true ；否则，返回 false 。
示例 1：输入：arr = [2,6,4,1] 输出：false
解释：不存在连续三个元素都是奇数的情况。
示例 2：输入：arr = [1,2,34,3,4,5,7,23,12] 输出：true
解释：存在连续三个元素都是奇数的情况，即 [5,7,23] 。
提示：  1 <= arr.length <= 1000
    1 <= arr[i] <= 1000
```

- 解题思路

```
func threeConsecutiveOdds(arr []int) bool {
        for i := 0; i < len(arr)-2; i++ {
                if arr[i]%2 == 1 && arr[i+1]%2 == 1 && arr[i+2]%2 == 1 {
                        return true
                }
        }
        return false
}
```

(续下页)

(接上页)

```
# 2
func threeConsecutiveOdds(arr []int) bool {
        count := 0
        for i := 0; i < len(arr); i++ {
                if arr[i]%2 == 1 {
                        count++
                } else {
                        count = 0
                }
                if count == 3 {
                        return true
                }
        }
        return false
}
```

46.11 1556. 千位分隔数 (1)

- 题目

```
给你一个整数 n，请你每隔三位添加点（即 "."␣
↪符号）作为千位分隔符，并将结果以字符串格式返回。
示例 1：输入：n = 987 输出："987"
示例 2：输入：n = 1234 输出："1.234"
示例 3：输入：n = 123456789 输出："123.456.789"
示例 4：输入：n = 0 输出："0"
提示：  0 <= n < 2^31
```

- 解题思路

```
func thousandSeparator(n int) string {
        res := ""
        if n == 0 {
                return "0"
        }
        count := 0
        for n != 0 {
                count++
                value := n % 10
                if count%3 == 1 {
                        res = strconv.Itoa(value) + "." + res
```

(续下页)

(接上页)

```
                } else {
                        res = strconv.Itoa(value) + res
                }
                n = n / 10
        }
        return strings.Trim(res, ".")
}
```

46.12 1560. 圆形赛道上经过次数最多的扇区 (2)

- 题目

```
给你一个整数 n 和一个整数数组 rounds 。有一条圆形赛道由 n 个扇区组成，扇区编号从 1 到␣
↪n 。
现将在这条赛道上举办一场马拉松比赛，该马拉松全程由 m 个阶段组成。
其中，第 i 个阶段将会从扇区 rounds[i - 1] 开始，到扇区 rounds[i] 结束。
举例来说，第 1 阶段从 rounds[0] 开始，到 rounds[1] 结束。
请你以数组形式返回经过次数最多的那几个扇区，按扇区编号 升序 排列。
注意，赛道按扇区编号升序逆时针形成一个圆（请参见第一个示例）。
示例 1：输入：n = 4, rounds = [1,3,1,2] 输出：[1,2]
解释：本场马拉松比赛从扇区 1 开始。经过各个扇区的次序如下所示：
1 --> 2 --> 3（阶段 1 结束）--> 4 --> 1（阶段 2 结束）--> 2
（阶段 3 结束，即本场马拉松结束）其中，扇区 1 和 2␣
↪都经过了两次，它们是经过次数最多的两个扇区。
扇区 3 和 4 都只经过了一次。
示例 2：输入：n = 2, rounds = [2,1,2,1,2,1,2,1,2] 出：[2]
示例 3：输入：n = 7, rounds = [1,3,5,7] 出：[1,2,3,4,5,6,7]
提示：
    2 <= n <= 100
    1 <= m <= 100
    rounds.length == m + 1
    1 <= rounds[i] <= n
    rounds[i] != rounds[i + 1] , 其中 0 <= i < m
```

- 解题思路

```
func mostVisited(n int, rounds []int) []int {
        arr := make([]int, n+1)
        res := make([]int, 0)
        max := 0
        arr[rounds[0]]++
        for i := 0; i < len(rounds)-1; i++ {
```

(续下页)

(接上页)

```
                start := rounds[i]
                end := rounds[i+1]
                if start < end {
                        for j := start + 1; j <= end; j++ {
                                arr[j]++
                        }
                } else {
                        for j := start + 1; j <= n; j++ {
                                arr[j]++
                        }
                        for j := 1; j <= end; j++ {
                                arr[j]++
                        }
                }
        }
        for i := 1; i < len(arr); i++ {
                if arr[i] > max {
                        max = arr[i]
                        res = make([]int, 0)
                        res = append(res, i)
                } else if arr[i] == max {
                        res = append(res, i)
                }
        }
        return res
}

# 2
func mostVisited(n int, rounds []int) []int {
        res := make([]int, 0)
        start := rounds[0]
        end := rounds[len(rounds)-1]
        if start <= end {
                for i := start; i <= end; i++ {
                        res = append(res, i)
                }
        } else {
                for i := 1; i <= end; i++ {
                        res = append(res, i)
                }
                for i := start; i <= n; i++ {
                        res = append(res, i)
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

46.13 1566. 重复至少 K 次且长度为 M 的模式 (2)

- 题目

```
给你一个正整数数组 arr，请你找出一个长度为 m 且在数组中至少重复 k 次的模式。
模式 是由一个或多个值组成的子数组（连续的子序列），连续 重复多次但 不重叠 。
↪模式由其长度和重复次数定义。
如果数组中存在至少重复 k 次且长度为 m 的模式，则返回 true ，否则返回  false 。
示例 1：输入：arr = [1,2,4,4,4,4], m = 1, k = 3 输出：true
解释：模式 (4) 的长度为 1 ，且连续重复 4 次。注意，模式可以重复 k
↪次或更多次，但不能少于 k 次。
示例 2：输入：arr = [1,2,1,2,1,1,1,3], m = 2, k = 2 输出：true
解释：模式 (1,2) 长度为 2 ，且连续重复 2 次。另一个符合题意的模式是 (2,1) ，同样重复
↪2 次。
示例 3：输入：arr = [1,2,1,2,1,3], m = 2, k = 3 输出：false
解释：模式 (1,2) 长度为 2 ，但是只连续重复 2 次。不存在长度为 2 且至少重复 3
↪次的模式。
示例 4：输入：arr = [1,2,3,1,2], m = 2, k = 2 输出：false
解释：模式 (1,2) 出现 2 次但并不连续，所以不能算作连续重复 2 次。
示例 5：输入：arr = [2,2,2,2], m = 2, k = 3 输出：false
解释：长度为 2 的模式只有 (2,2) ，但是只连续重复 2 次。注意，不能计算重叠的重复次数。
提示：  2 <= arr.length <= 100
    1 <= arr[i] <= 100
    1 <= m <= 100
    2 <= k <= 100
```

- 解题思路

```
func containsPattern(arr []int, m int, k int) bool {
        for i := 0; i < len(arr)-m; i++ {
                count := 0
                j := i
                for j+m <= len(arr) {
                        flag := true
                        for l := 0; l < m; l++ {
                                if arr[i+l] != arr[j+l] {
                                        flag = false
                                        break
```

(续下页)

(接上页)

```
                                }
                        }
                        if flag == true {
                                count++
                                j = j + m
                        } else {
                                break
                        }
                }
                if count >= k {
                        return true
                }
        }
        return false
}

# 2
func containsPattern(arr []int, m int, k int) bool {
        n := len(arr)
        if n < m*k {
                return false
        }
        i, j := 0, 0
        for i = 0; i <= n-m*k; i++ {
                for j = i + m; j < i+m*k; j++ {
                        if arr[j] != arr[j-m] {
                                break
                        }
                }
                if j == i+m*k {
                        return true
                }
        }
        return false
}
```

46.14 1572. 矩阵对角线元素的和 (2)

- 题目

```
给你一个正方形矩阵 mat，请你返回矩阵对角线元素的和。
请你返回在矩阵主对角线上的元素和副对角线上且不在主对角线上元素的和。
示例 1：输入：mat = [[1,2,3],
             [4,5,6],
             [7,8,9]]
输出：25
解释：对角线的和为：1 + 5 + 9 + 3 + 7 = 25
请注意，元素 mat[1][1] = 5 只会被计算一次。
示例 2：输入：mat = [[1,1,1,1],
             [1,1,1,1],
             [1,1,1,1],
             [1,1,1,1]]
输出：8
示例 3：输入：mat = [[5]] 输出：5
提示：
    n == mat.length == mat[i].length
    1 <= n <= 100
    1 <= mat[i][j] <= 100
```

- 解题思路

```
func diagonalSum(mat [][]int) int {
        res := 0
        for i := 0; i < len(mat); i++ {
                a, b := i, len(mat)-1-i
                if a == b {
                        res = res + mat[a][b]
                } else {
                        res = res + mat[a][a]
                        res = res + mat[a][b]
                }
        }
        return res
}

# 2
func diagonalSum(mat [][]int) int {
        res := 0
        for i := 0; i < len(mat); i++ {
                a, b := i, len(mat)-1-i
```

(续下页)

(接上页)

```
                res = res + mat[a][a]
                res = res + mat[a][b]

        }
        if len(mat)%2 == 1 {
                res = res - mat[len(mat)/2][len(mat)/2]
        }
        return res
}
```

46.15 1576. 替换所有的问号 (1)

- 题目

```
给你一个仅包含小写英文字母和 '?' 字符的字符串 s，
请你将所有的 '?' 转换为若干小写字母，使最终的字符串不包含任何 连续重复 的字符。
注意：你 不能 修改非 '?' 字符。
题目测试用例保证 除 '?' 字符 之外，不存在连续重复的字符。
在完成所有转换（可能无需转换）后返回最终的字符串。
如果有多个解决方案，请返回其中任何一个。可以证明，在给定的约束条件下，答案总是存在的。
示例 1：输入：s = "?zs" 输出："azs"
解释：该示例共有 25 种解决方案，从 "azs" 到 "yzs" 都是符合题目要求的。
只有 "z" 是无效的修改，因为字符串 "zzs" 中有连续重复的两个 'z' 。
示例 2：输入：s = "ubv?w"输出："ubvaw"
解释：该示例共有 24 种解决方案，只有替换成 "v" 和 "w" 不符合题目要求。
因为 "ubvvw" 和 "ubvww" 都包含连续重复的字符。
示例 3：输入：s = "j?qg??b" 输出："jaqgacb"
示例 4：输入：s = "??yw?ipkj?" 输出："acywaipkja"
提示：1 <= s.length <= 100
    s 仅包含小写英文字母和 '?' 字符
```

- 解题思路

```
func modifyString(s string) string {
        res := []byte(s)
        for i := 0; i < len(s); i++ {
                if s[i] == '?' {
                        for j := byte('a'); j <= 'z'; j++ {
                                if (i == 0 || res[i-1] != j) && (i == len(s)-1 ||␣
↪s[i+1] != j) {
                                        res[i] = j
                                        break
```

(续下页)

(接上页)

```
                                    }
                                }
                            }
                        }
                        return string(res)
}
```

46.16 1582. 二进制矩阵中的特殊位置 (2)

- 题目

```
给你一个大小为 rows x cols 的矩阵 mat，其中 mat[i][j] 是 0 或 1，
请返回 矩阵 mat 中特殊位置的数目 。
特殊位置 定义：如果 mat[i][j] == 1 并且第 i 行和第 j 列中的所有其他元素均为 0
（行和列的下标均 从 0 开始 ），则位置 (i, j) 被称为特殊位置。
示例 1：输入：mat = [[1,0,0],
             [0,0,1],
             [1,0,0]]
输出：1
解释：(1,2) 是一个特殊位置，因为 mat[1][2] == 1 且所处的行和列上所有其他元素都是 0
示例 2：输入：mat = [[1,0,0],
             [0,1,0],
             [0,0,1]]
输出：3
解释：(0,0), (1,1) 和 (2,2) 都是特殊位置
示例 3：输入：mat = [[0,0,0,1],
             [1,0,0,0],
             [0,1,1,0],
             [0,0,0,0]]
输出：2
示例 4：输入：mat = [[0,0,0,0,0],
             [1,0,0,0,0],
             [0,1,0,0,0],
             [0,0,1,0,0],
             [0,0,0,1,1]]
输出：3
提示：
    rows == mat.length
    cols == mat[i].length
    1 <= rows, cols <= 100
    mat[i][j] 是 0 或 1
```

- 解题思路

```
func numSpecial(mat [][]int) int {
        res := 0
        for i := 0; i < len(mat); i++ {
                for j := 0; j < len(mat[i]); j++ {
                        if mat[i][j] == 1 && judge(mat, i, j) == true {
                                res++
                        }
                }
        }
        return res
}

func judge(mat [][]int, i, j int) bool {
        for a := 0; a < len(mat[i]); a++ {
                if mat[i][a] == 1 && a != j {
                        return false
                }
        }
        for a := 0; a < len(mat); a++ {
                if mat[a][j] == 1 && a != i {
                        return false
                }
        }
        return true
}

# 2
func numSpecial(mat [][]int) int {
        res := 0
        rows, cols := make([]int, len(mat)), make([]int, len(mat[0]))
        for i := 0; i < len(mat); i++ {
                for j := 0; j < len(mat[i]); j++ {
                        rows[i] = rows[i] + mat[i][j]
                        cols[j] = cols[j] + mat[i][j]
                }
        }
        for i := 0; i < len(mat); i++ {
                for j := 0; j < len(mat[i]); j++ {
                        if mat[i][j] == 1 && rows[i] == 1 && cols[j] == 1 {
                                res++
                        }
                }
        }
        return res
```

(续下页)

(接上页)

```
}
```

46.17 1588. 所有奇数长度子数组的和 (3)

- 题目

```
给你一个正整数数组 arr ，请你计算所有可能的奇数长度子数组的和。
子数组 定义为原数组中的一个连续子序列。
请你返回 arr 中 所有奇数长度子数组的和 。
示例 1：输入：arr = [1,4,2,5,3] 输出：58
解释：所有奇数长度子数组和它们的和为：
[1] = 1
[4] = 4
[2] = 2
[5] = 5
[3] = 3
[1,4,2] = 7
[4,2,5] = 11
[2,5,3] = 10
[1,4,2,5,3] = 15
我们将所有值求和得到 1 + 4 + 2 + 5 + 3 + 7 + 11 + 10 + 15 = 58
示例 2：输入：arr = [1,2] 输出：3
解释：总共只有 2 个长度为奇数的子数组，[1] 和 [2]。它们的和为 3 。
示例 3：输入：arr = [10,11,12] 输出：66
提示：
    1 <= arr.length <= 100
    1 <= arr[i] <= 1000
```

- 解题思路

```
func sumOddLengthSubarrays(arr []int) int {
        sum := make([]int, len(arr)+1)
        sum[0] = 0
        for i := 1; i <= len(arr); i++ {
                sum[i] = sum[i-1] + arr[i-1]
        }
        res := 0
        for i := 1; i <= len(arr); i = i + 2 {
                for j := 0; j < len(arr); j++ {
                        k := j + i
                        if k <= len(arr) {
                                total := sum[k] - sum[j]
```

(续下页)

(接上页)

```
                                res = res + total
                        }
                }
        }
        return res
}

# 2
func sumOddLengthSubarrays(arr []int) int {
        res := 0
        for i := 1; i <= len(arr); i = i + 2 {
                for j := 0; j < len(arr); j++ {
                        k := j + i
                        for start := j; start < k && k <= len(arr); start++ {
                                res = res + arr[start]
                        }
                }
        }
        return res
}

# 3
# https://leetcode.cn/problems/sum-of-all-odd-length-subarrays/solution/cong-on3-dao-
↪on-de-jie-fa-by-liuyubobobo/
func sumOddLengthSubarrays(arr []int) int {
        res := 0
        n := len(arr)
        for i := 0; i < n; i++ {
                left := i + 1
                right := n - i
                leftEven := (left + 1) / 2
                rightEven := (right + 1) / 2
                leftOdd := left / 2
                rightOdd := right / 2
                res = res + arr[i]*(leftEven*rightEven+leftOdd*rightOdd)
        }
        return res
}
```

46.18 1592. 重新排列单词间的空格 (1)

- 题目

```
给你一个字符串 text␣
↪，该字符串由若干被空格包围的单词组成。每个单词由一个或者多个小写英文字母组成，
并且两个单词之间至少存在一个空格。题目测试用例保证 text 至少包含一个单词 。
请你重新排列空格，使每对相邻单词之间的空格数目都 相等 ，并尽可能 最大化 该数目。
如果不能重新平均分配所有空格，请 将多余的空格放置在字符串末尾 ，
这也意味着返回的字符串应当与原 text 字符串的长度相等。
返回 重新排列空格后的字符串 。
示例 1：输入：text = "  this   is  a sentence " 输出："this   is   a   sentence"
解释：总共有 9 个空格和 4 个单词。可以将 9␣
↪个空格平均分配到相邻单词之间，相邻单词间空格数为：
9 / (4-1) = 3 个。
示例 2：输入：text = " practice   makes   perfect" 输出："practice   makes   perfect "
解释：总共有 7 个空格和 3 个单词。7 / (3-1) = 3 个空格加上 1 个多余的空格。
多余的空格需要放在字符串的末尾。
示例 3：输入：text = "hello   world" 输出："hello   world"
示例 4：输入：text = "  walks  udp package   into  bar a"
输出："walks  udp  package  into  bar  a "
示例 5：输入：text = "a" 输出："a"
提示：  1 <= text.length <= 100
    text 由小写英文字母和 ' ' 组成
    text 中至少包含一个单词
```

- 解题思路

```go
func reorderSpaces(text string) string {
        count := strings.Count(text, " ")
        arr := strings.Fields(text)
        if len(arr) == 1 {
                return arr[0] + strings.Repeat(" ", count)
        }
        value := count / (len(arr) - 1)
        left := count % (len(arr) - 1)
        res := strings.Join(arr, strings.Repeat(" ", value))
        res = res + strings.Repeat(" ", left)
        return res
}
```

46.19 1598. 文件夹操作日志搜集器 (1)

- 题目

```
每当用户执行变更文件夹操作时，LeetCode 文件系统都会保存一条日志记录。
下面给出对变更操作的说明：
    "../" ：移动到当前文件夹的父文件夹。如果已经在主文件夹下，则 继续停留在当前文件夹。
↪。
    "./" ：继续停留在当前文件夹。
    "x/" ：移动到名为 x 的子文件夹中。题目数据 保证总是存在文件夹 x 。
给你一个字符串列表 logs ，其中 logs[i] 是用户在 ith 步执行的操作。
文件系统启动时位于主文件夹，然后执行 logs 中的操作。
执行完所有变更文件夹操作后，请你找出 返回主文件夹所需的最小步数 。
示例 1：输入：logs = ["d1/","d2/","../","d21/","./"] 输出：2
解释：执行 "../" 操作变更文件夹 2 次，即可回到主文件夹
示例 2：输入：logs = ["d1/","d2/","./","d3/","../","d31/"] 输出：3
示例 3：输入：logs = ["d1/","../","../","../"] 输出：0
提示：  1 <= logs.length <= 103
    2 <= logs[i].length <= 10
    logs[i] 包含小写英文字母，数字，'.' 和 '/'
    logs[i] 符合语句中描述的格式
    文件夹名称由小写英文字母和数字组成
```

- 解题思路

```
func minOperations(logs []string) int {
        stack := make([]string, 0)
        for i := 0; i < len(logs); i++ {
                if logs[i] == "../" {
                        if len(stack) > 0 {
                                stack = stack[:len(stack)-1]
                        }
                } else if logs[i] != "./" {
                        stack = append(stack, logs[i])
                }
        }
        return len(stack)
}
```

CHAPTER 47

1501-1600-Medium

47.1 1503. 所有蚂蚁掉下来前的最后一刻 (2)

- 题目

```
有一块木板，长度为 n 个 单位 。一些蚂蚁在木板上移动，每只蚂蚁都以 每秒一个单位␣
↪的速度移动。
其中，一部分蚂蚁向 左 移动，其他蚂蚁向 右 移动。
当两只向 不同 方向移动的蚂蚁在某个点相遇时，它们会同时改变移动方向并继续移动。
假设更改方向不会花费任何额外时间。
而当蚂蚁在某一时刻 t 到达木板的一端时，它立即从木板上掉下来。
给你一个整数 n 和两个整数数组 left 以及 right 。
两个数组分别标识向左或者向右移动的蚂蚁在 t = 0␣
↪时的位置。请你返回最后一只蚂蚁从木板上掉下来的时刻。
示例 1：输入：n = 4, left = [4,3], right = [0,1] 输出：4
解释：如上图所示：
-下标 0 处的蚂蚁命名为 A 并向右移动。
-下标 1 处的蚂蚁命名为 B 并向右移动。
-下标 3 处的蚂蚁命名为 C 并向左移动。
-下标 4 处的蚂蚁命名为 D 并向左移动。
请注意，蚂蚁在木板上的最后时刻是 t = 4 秒，之后蚂蚁立即从木板上掉下来。
（也就是说在 t = 4.0000000001 时，木板上没有蚂蚁）。
示例 2：输入：n = 7, left = [], right = [0,1,2,3,4,5,6,7] 输出：7
解释：所有蚂蚁都向右移动，下标为 0 的蚂蚁需要 7 秒才能从木板上掉落。
示例 3：输入：n = 7, left = [0,1,2,3,4,5,6,7], right = [] 输出：7
```

(续下页)

(接上页)

```
解释：所有蚂蚁都向左移动，下标为 7 的蚂蚁需要 7 秒才能从木板上掉落。
示例 4：输入：n = 9, left = [5], right = [4] 输出：5
解释：t = 1 秒时，两只蚂蚁将回到初始位置，但移动方向与之前相反。
示例 5：输入：n = 6, left = [6], right = [0] 输出：6
提示：
    1 <= n <= 10^4
    0 <= left.length <= n + 1
    0 <= left[i] <= n
    0 <= right.length <= n + 1
    0 <= right[i] <= n
    1 <= left.length + right.length <= n + 1
    left 和 right 中的所有值都是唯一的，并且每个值 只能出现在二者之一 中。
```

- 解题思路

```
// 2只蚂蚁相遇=>两只蚂蚁都不改变移动方向=>求离终点最远距离
func getLastMoment(n int, left []int, right []int) int {
        max := 0
        for i := 0; i < len(left); i++ {
                if left[i] > max {
                        max = left[i]
                }
        }
        for i := 0; i < len(right); i++ {
                if n-right[i] > max {
                        max = n - right[i]
                }
        }
        return max
}

# 2
func getLastMoment(n int, left []int, right []int) int {
        sort.Ints(left)
        sort.Ints(right)
        if len(left) == 0 {
                return n - right[0]
        }
        if len(right) == 0 {
                return left[len(left)-1]
        }
        if n-right[0] > left[len(left)-1] {
                return n - right[0]
        }
```

(续下页)

(接上页)

```
        return left[len(left)-1]
}
```

47.2 1504. 统计全 1 子矩形 (2)

- 题目

```
给你一个只包含 0 和 1 的rows * columns矩阵mat，请你返回有多少个子矩形的元素全部都是 1
↪。
示例 1：输入：mat = [[1,0,1],
            [1,1,0],
            [1,1,0]]
输出：13
解释：有 6个 1x1 的矩形。
有 2 个 1x2 的矩形。
有 3 个 2x1 的矩形。
有 1 个 2x2 的矩形。
有 1 个 3x1 的矩形。
矩形数目总共 = 6 + 2 + 3 + 1 + 1 = 13。
示例 2：输入：mat = [[0,1,1,0],
            [0,1,1,1],
            [1,1,1,0]]
输出：24
解释：有 8 个 1x1 的子矩形。
有 5 个 1x2 的子矩形。
有 2 个 1x3 的子矩形。
有 4 个 2x1 的子矩形。
有 2 个 2x2 的子矩形。
有 2 个 3x1 的子矩形。
有 1 个 3x2 的子矩形。
矩形数目总共 = 8 + 5 + 2 + 4 + 2 + 2 + 1 = 24 。
示例 3：输入：mat = [[1,1,1,1,1,1]] 输出：21
示例 4：输入：mat = [[1,0,1],[0,1,0],[1,0,1]] 输出：5
提示：1 <= rows<= 150
1 <= columns<= 150
0 <= mat[i][j] <= 1
```

- 解题思路

```
func numSubmat(mat [][]int) int {
        res := 0
        n, m := len(mat), len(mat[0])
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if j > 0 && mat[i][j] == 1 {
                                mat[i][j] = mat[i][j-1] + 1 // 到左边连续1的个数
                        }
                        target := mat[i][j]       //␣
↪底：当前行以mat[i][j]=1结尾的连续1的个数
                        for k := i; k >= 0; k-- { // 遍历高
                                if target > mat[k][j] { // 上一层连续1的个数小
                                        target = mat[k][j] // 缩小底
                                }
                                // 示例1：右下角一行3个1
                                // 1(1) 1(2) 1(3) => 有3种组成矩形的情况
                                // 第1种情况.1(3)
                                // 第2种情况.1(2)+1(3)
                                // 第3种情况.1(1)+1(2)+1(3)
                                res = res + target // 加上该高度的个数
                        }
                }
        }
        return res
}

# 2
func numSubmat(mat [][]int) int {
        res := 0
        n, m := len(mat), len(mat[0])
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if j == 0 {
                                arr[i][j] = mat[i][j]
                        } else if j > 0 && mat[i][j] == 1 {
                                arr[i][j] = arr[i][j-1] + 1 // 到左边连续1的个数
                        }
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        target := arr[i][j]
```

(续下页)

(接上页)

```
                        for k := i; k >= 0; k-- {
                                if target > arr[k][j] {
                                        target = arr[k][j]
                                }
                                res = res + target
                        }
                }
        }
        return res
}
```

47.3 1508. 子数组和排序后的区间和 (1)

- 题目

```
给你一个数组 nums ，它包含 n␣
↪个正整数。你需要计算所有非空连续子数组的和，并将它们按升序排序，
得到一个新的包含 n * (n + 1) / 2 个数字的数组。
请你返回在新数组中下标为 left 到 right （下标从 1 开始）的所有数字和（包括左右端点）。
由于答案可能很大，请你将它对 10^9 + 7 取模后返回。
示例 1：输入：nums = [1,2,3,4], n = 4, left = 1, right = 5 输出：13
解释：所有的子数组和为 1, 3, 6, 10, 2, 5, 9, 3, 7, 4 。
将它们升序排序后，我们得到新的数组 [1, 2, 3, 3, 4, 5, 6, 7, 9, 10] 。
下标从 le = 1 到 ri = 5 的和为 1 + 2 + 3 + 3 + 4 = 13 。
示例 2：输入：nums = [1,2,3,4], n = 4, left = 3, right = 4 输出：6
解释：给定数组与示例 1 一样，所以新数组为 [1, 2, 3, 3, 4, 5, 6, 7, 9, 10] 。
下标从 le = 3 到 ri = 4 的和为 3 + 3 = 6 。
示例 3：输入：nums = [1,2,3,4], n = 4, left = 1, right = 10 输出：50
提示：
    1 <= nums.length <= 10^3
    nums.length == n
    1 <= nums[i] <= 100
    1 <= left <= right <= n * (n + 1) / 2
```

- 解题思路

```
func rangeSum(nums []int, n int, left int, right int) int {
        arr := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum = sum + nums[j]
```

(续下页)

(接上页)

```
                        arr = append(arr, sum)
                }
        }
        sort.Ints(arr)
        res := 0
        for i := left - 1; i <= right-1; i++ {
                res = (res + arr[i]) % 1000000007
        }
        return res
}
```

47.4 1509. 三次操作后最大值与最小值的最小差 (2)

- 题目

```
给你一个数组 nums ，每次操作你可以选择 nums 中的任意一个元素并将它改成任意值。
请你返回三次操作后， nums 中最大值与最小值的差的最小值。
示例 1：输入：nums = [5,3,2,4] 输出：0
解释：将数组 [5,3,2,4] 变成 [2,2,2,2].
最大值与最小值的差为 2-2 = 0 。
示例 2：输入：nums = [1,5,0,10,14] 输出：1
解释：将数组 [1,5,0,10,14] 变成 [1,1,0,1,1] 。
最大值与最小值的差为 1-0 = 1 。
示例 3：输入：nums = [6,6,0,1,1,4,6] 输出：2
示例 4：输入：nums = [1,5,6,14,15] 输出：1
提示：
    1 <= nums.length <= 10^5
    -10^9 <= nums[i] <= 10^9
```

- 解题思路

```
func minDifference(nums []int) int {
        if len(nums) < 5 {
                return 0
        }
        sort.Ints(nums)
        res := math.MaxInt32
        for i := 0; i <= 3; i++ {
                res = min(res, nums[len(nums)-1-(3-i)]-nums[i])
        }
        return res
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

#
func minDifference(nums []int) int {
        if len(nums) < 5 {
                return 0
        }
        sort.Ints(nums)
        res := math.MaxInt32
        res = min(res, nums[len(nums)-1]-nums[3])
        res = min(res, nums[len(nums)-2]-nums[2])
        res = min(res, nums[len(nums)-3]-nums[1])
        res = min(res, nums[len(nums)-4]-nums[0])
        return res
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}
```

47.5 1513. 仅含 1 的子串数 (2)

- 题目

```
给你一个二进制字符串 s（仅由 '0' 和 '1' 组成的字符串）。
返回所有字符都为 1 的子字符串的数目。
由于答案可能很大，请你将它对 10^9 + 7 取模后返回。

示例 1：输入：s = "0110111" 输出：9
解释：共有 9 个子字符串仅由 '1' 组成
"1" -> 5 次
"11" -> 3 次
"111" -> 1 次
```

(续下页)

(接上页)

```
示例 2: 输入: s = "101" 输出: 2
解释: 子字符串 "1" 在 s 中共出现 2 次
示例 3: 输入: s = "111111" 输出: 21
解释: 每个子字符串都仅由 '1' 组成
示例 4: 输入: s = "000" 输出: 0
提示:
    s[i] == '0' 或 s[i] == '1'
    1 <= s.length <= 10^5
```

- 解题思路

```
func numSub(s string) int {
        res := 0
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '1' {
                        count++
                } else {
                        res = (res + (count+1)*count/2) % 1000000007
                        count = 0
                }
        }
        if count > 0 {
                res = (res + (count+1)*count/2) % 1000000007
        }
        return res
}

#
func numSub(s string) int {
        res := 0
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '1' {
                        count++
                } else {
                        count = 0
                }
                res = (res + count) % 1000000007
        }
        return res
}
```

47.6 1514. 概率最大的路径 (2)

- 题目

```
给你一个由 n 个节点（下标从 0 开始）组成的无向加权图，该图由一个描述边的列表组成，
其中 edges[i] = [a, b] 表示连接节点 a 和 b 的一条无向边，且该边遍历成功的概率为␣
↪succProb[i] 。
指定两个节点分别作为起点 start 和终点 end␣
↪，请你找出从起点到终点成功概率最大的路径，并返回其成功概率。
如果不存在从 start 到 end 的路径，请 返回 0 。只要答案与标准答案的误差不超过 1e-5 ，
就会被视作正确答案。
示例 1：
输入：n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2
输出：0.25000
解释：从起点到终点有两条路径，其中一条的成功概率为 0.2 ，而另一条为 0.5 * 0.5 = 0.25
示例 2：
输入：n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.3], start = 0, end = 2
输出：0.30000
示例 3：输入：n = 3, edges = [[0,1]], succProb = [0.5], start = 0, end = 2 输出：0.
↪00000
解释：节点 0 和 节点 2 之间不存在路径
提示：2 <= n <= 10^4
0 <= start, end < n
start != end
0 <= a, b < n
a != b
0 <= succProb.length == edges.length <= 2*10^4
0 <= succProb[i] <= 1
每两个节点之间最多有一条边
```

- 解题思路

```
func maxProbability(n int, edges [][]int, succProb []float64, start int, end int)␣
↪float64 {
        arr := make([][]Node, n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], Node{index: b, Value: succProb[i]})
                arr[b] = append(arr[b], Node{index: a, Value: succProb[i]})
        }
        visited := make([]bool, n)
        maxValue := make([]float64, n)
        nodeHeap := make(NodeHeap, 0)
        heap.Init(&nodeHeap)
```

(续下页)

(接上页)

```
        heap.Push(&nodeHeap, Node{
                Value: 1,
                index: start,
        })
        for nodeHeap.Len() > 0 {
                node := heap.Pop(&nodeHeap).(Node)
                visited[node.index] = true
                if node.index == end {
                        return node.Value
                }
                for i := 0; i < len(arr[node.index]); i++ {
                        next := arr[node.index][i]
                        if visited[next.index] == true || node.Value*next.Value <␣
↪maxValue[next.index] {
                                continue
                        }
                        maxValue[next.index] = node.Value * next.Value
                        heap.Push(&nodeHeap, Node{
                                Value: maxValue[next.index],
                                index: next.index,
                        })
                }
        }
        return 0
}

type Node struct {
        Value float64
        index int
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h NodeHeap) Less(i, j int) bool {
        return h[i].Value > h[j].Value
}

func (h NodeHeap) Swap(i, j int) {
```

(续下页)

(接上页)

```
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func maxProbability(n int, edges [][]int, succProb []float64, start int, end int)␣
↪float64 {
        arr := make([][]Node, n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], Node{index: b, Value: succProb[i]})
                arr[b] = append(arr[b], Node{index: a, Value: succProb[i]})
        }
        maxValue := make([]float64, n)
        maxValue[start] = 1.0
        queue := make([]Node, 0)
        queue = append(queue, Node{
                Value: 1.0,
                index: start,
        })
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if maxValue[node.index] > node.Value {
                        continue
                }
                for i := 0; i < len(arr[node.index]); i++ {
                        next := arr[node.index][i]
                        // 注意使用<=，即=号要过滤，不然容易超时
                        if maxValue[node.index]*next.Value <= maxValue[next.index] {
                                continue
                        }
                        maxValue[next.index] = maxValue[node.index] * next.Value
                        queue = append(queue, Node{
```

(续下页)

(接上页)

```
                                Value: maxValue[next.index],
                                index: next.index,
                        })
                }
        }
        return maxValue[end]
}

type Node struct {
        Value float64
        index int
}
```

47.7 1519. 子树中标签相同的节点数 (2)

- 题目

```
给你一棵树（即，一个连通的无环无向图），这棵树由编号从 0 到 n - 1 的 n 个节点组成，
且恰好有 n - 1 条 edges 。
树的根节点为节点 0 ，树上的每一个节点都有一个标签，
也就是字符串 labels 中的一个小写字符（编号为 i 的 节点的标签就是 labels[i] ）
边数组 edges 以 edges[i] = [ai, bi] 的形式给出，该格式表示节点 ai 和 bi␣
↪之间存在一条边。
返回一个大小为 n 的数组，其中 ans[i] 表示第 i 个节点的子树中与节点 i␣
↪标签相同的节点数。
树 T 中的子树是由 T 中的某个节点及其所有后代节点组成的树。
示例 1：输入：n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], labels = "abaedcd"
输出：[2,1,1,1,1,1,1]
解释：节点 0 的标签为 'a' ，以 'a' 为根节点的子树中，节点 2 的标签也是 'a'␣
↪，因此答案为 2 。
注意树中的每个节点都是这棵子树的一部分。
节点 1 的标签为 'b' ，节点 1 的子树包含节点 1、4 和 5，
但是节点 4、5 的标签与节点 1 不同，故而答案为 1（即，该节点本身）。
示例 2：输入：n = 4, edges = [[0,1],[1,2],[0,3]], labels = "bbbb" 输出：[4,2,1,1]
解释：节点 2 的子树中只有节点 2 ，所以答案为 1 。
节点 3 的子树中只有节点 3 ，所以答案为 1 。
节点 1 的子树中包含节点 1 和 2 ，标签都是 'b' ，因此答案为 2 。
节点 0 的子树中包含节点 0、1、2 和 3，标签都是 'b'，因此答案为 4 。
示例 3：输入：n = 5, edges = [[0,1],[0,2],[1,3],[0,4]], labels = "aabab" 输出：[3,2,1,
↪1,1]
示例 4：输入：n = 6, edges = [[0,1],[0,2],[1,3],[3,4],[4,5]], labels = "cbabaa"
输出：[1,2,1,1,2,1]
```

(续下页)

(接上页)

```
示例 5: 输入: n = 7, edges = [[0,1],[1,2],[2,3],[3,4],[4,5],[5,6]], labels = "aaabaaa"
输出: [6,5,4,1,3,2,1]
提示: 1 <= n <= 10^5
edges.length == n - 1
edges[i].length == 2
0 <= ai,bi < n
ai !=bi
labels.length == n
labels 仅由小写英文字母组成
```

- 解题思路

```
var arr [][]int // 邻接表
var res []int   // 结果

func countSubTrees(n int, edges [][]int, labels string) []int {
        res = make([]int, n)
        arr = make([][]int, n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        dfs(labels, 0, 0)
        return res
}

// 后序遍历
func dfs(s string, cur int, prev int) [26]int {
        count := [26]int{}
        for i := 0; i < len(arr[cur]); i++ {
                next := arr[cur][i]
                if next == prev { // 注意避免重复往回走
                        continue
                }
                res := dfs(s, next, cur)
                for j := 0; j < len(res); j++ {
                        count[j] = count[j] + res[j]
                }
        }
        value := int(s[cur] - 'a')
        count[value]++
        res[cur] = count[value]
        return count
```

(续下页)

(接上页)

```
}

# 2
var arr [][]int   // 邻接表
var res []int     // 结果
var m map[int]int // 临时结果
func countSubTrees(n int, edges [][]int, labels string) []int {
        res = make([]int, n)
        arr = make([][]int, n)
        m = make(map[int]int)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        dfs(labels, 0, 0)
        for i := 0; i < n; i++ {
                value := int(labels[i] - 'a')
                res[i] = m[i*26+value]
        }
        return res
}

// 后序遍历
func dfs(s string, cur int, prev int) {
        m[cur*26+int(s[cur]-'a')]++
        for i := 0; i < len(arr[cur]); i++ {
                next := arr[cur][i]
                if next == prev { // 注意避免重复往回走
                        continue
                }
                dfs(s, next, cur)
                for j := 0; j < 26; j++ { // 子节点的结果
                        m[cur*26+j] = m[cur*26+j] + m[next*26+j]
                }
        }
}
```

47.8 1524. 和为奇数的子数组数目 (2)

- 题目

```
给你一个整数数组 arr 。请你返回和为 奇数 的子数组数目。
由于答案可能会很大，请你将结果对 10^9 + 7 取余后返回。
示例 1：输入：arr = [1,3,5] 输出：4
解释：所有的子数组为 [[1],[1,3],[1,3,5],[3],[3,5],[5]] 。
所有子数组的和为 [1,4,9,3,8,5].
奇数和包括 [1,9,3,5] ，所以答案为 4 。
示例 2 ：输入：arr = [2,4,6] 输出：0
解释：所有子数组为 [[2],[2,4],[2,4,6],[4],[4,6],[6]] 。
所有子数组和为 [2,6,12,4,10,6] 。
所有子数组和都是偶数，所以答案为 0 。
示例 3：输入：arr = [1,2,3,4,5,6,7] 输出：16
示例 4：输入：arr = [100,100,99,99] 输出：4
示例 5：输入：arr = [7] 输出：1
提示：1 <= arr.length <= 10^5
    1 <= arr[i] <= 100
```

- 解题思路

```
func numOfSubarrays(arr []int) int {
        odd, even := 0, 1
        res := 0
        total := 0
        for i := 0; i < len(arr); i++ {
                total = total + arr[i]
                if total%2 == 0 {
                        res = res + odd
                        even = even + 1
                } else {
                        res = res + even
                        odd = odd + 1
                }
        }
        return res % 1000000007
}

# 2
func numOfSubarrays(arr []int) int {
        res := 0
        dp := make([][2]int, len(arr))
        // dp[i][0]子数组和为奇数的个数
```

(续下页)

(接上页)

```
        // dp[i][1]子数组和为偶数的个数
        if arr[0]%2 == 1 {
                dp[0][0] = 1
                res++
        } else {
                dp[0][1] = 1
        }
        for i := 1; i < len(arr); i++ {
                if arr[i]%2 == 1 {
                        dp[i][0] = dp[i-1][1] + 1
                        dp[i][1] = dp[i-1][0]
                } else {
                        dp[i][0] = dp[i-1][0]
                        dp[i][1] = dp[i-1][1] + 1
                }
                res = res + dp[i][0]
        }
        return res % 1000000007
}
```

47.9 1525. 字符串的好分割数目 (2)

- 题目

```
给你一个字符串 s ，一个分割被称为 「好分割」 当它满足：
将 s 分割成 2 个字符串 p 和 q ，它们连接起来等于 s 且 p 和 q 中不同字符的数目相同。
请你返回 s 中好分割的数目。
示例 1：输入：s = "aacaba" 输出：2
解释：总共有 5 种分割字符串 "aacaba" 的方法，其中 2 种是好分割。
("a", "acaba") 左边字符串和右边字符串分别包含 1 个和 3 个不同的字符。
("aa", "caba") 左边字符串和右边字符串分别包含 1 个和 3 个不同的字符。
("aac", "aba") 左边字符串和右边字符串分别包含 2 个和 2 个不同的字符。这是一个好分割。
("aaca", "ba") 左边字符串和右边字符串分别包含 2 个和 2 个不同的字符。这是一个好分割。
("aacab", "a") 左边字符串和右边字符串分别包含 3 个和 1 个不同的字符。
示例 2：输入：s = "abcd"输出：1 解释：好分割为将字符串分割成 ("ab", "cd") 。
示例 3：输入：s = "aaaaa"输出：4 解释：所有分割都是好分割。
示例 4：输入：s = "acbadbaada"输出：2
提示：s 只包含小写英文字母。 1 <= s.length <= 10^5
```

- 解题思路

```
func numSplits(s string) int {
        left := make(map[byte]int)
        right := make(map[byte]int)
        for i := 0; i < len(s); i++{
                left[s[i]]++
        }
        res := 0
        for i := 0; i < len(s); i++{
                left[s[i]]--
                right[s[i]]++
                if left[s[i]] == 0{
                        delete(left,s[i])
                }
                if len(left) == len(right){
                        res++
                }
        }
        return res
}

#
func numSplits(s string) int {
        left := [256]int{}
        right := [256]int{}
        leftCount := 0
        rightCount := 0
        for i := 0; i < len(s); i++ {
                left[s[i]]++
                if left[s[i]] == 1 {
                        leftCount++
                }
        }
        res := 0
        for i := 0; i < len(s); i++ {
                left[s[i]]--
                right[s[i]]++
                if left[s[i]] == 0 {
                        leftCount--
                }
                if right[s[i]] == 1 {
                        rightCount++
                }
                if leftCount == rightCount {
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}
```

47.10 1529. 灯泡开关 IV(2)

- 题目

```
房间中有 n 个灯泡，编号从 0 到 n-1 ，自左向右排成一行。最开始的时候，所有的灯泡都是关着的。
请你设法使得灯泡的开关状态和 target 描述的状态一致，其中 target[i] 等于 1 第 i 个灯泡是开着的，
等于 0 意味着第 i 个灯是关着的。
有一个开关可以用于翻转灯泡的状态，翻转操作定义如下：
    选择当前配置下的任意一个灯泡（下标为 i ）
    翻转下标从 i 到 n-1 的每个灯泡
翻转时，如果灯泡的状态为 0 就变为 1，为 1 就变为 0 。
返回达成 target 描述的状态所需的 最少 翻转次数。
示例 1：输入：target = "10111" 输出：3 解释：初始配置 "00000".
从第 3 个灯泡（下标为 2）开始翻转 "00000" -> "00111"
从第 1 个灯泡（下标为 0）开始翻转 "00111" -> "11000"
从第 2 个灯泡（下标为 1）开始翻转 "11000" -> "10111"
至少需要翻转 3 次才能达成 target 描述的状态
示例 2：输入：target = "101" 输出：3 解释："000" -> "111" -> "100" -> "101".
示例 3：输入：target = "00000" 输出：0
示例 4：输入：target = "001011101" 输出：5
提示：
    1 <= target.length <= 10^5
    target[i] == '0' 或者 target[i] == '1'
```

- 解题思路

```
func minFlips(target string) int {
        res := 0
        prev := uint8('0')
        for i := 0; i < len(target); i++ {
                // 只要后一个与前一个不相等，都会额外增加一次翻转
                if prev != target[i] {
                        res++
                        prev = target[i]
                }
```

(续下页)

(接上页)

```
        }
        return res
}

# 2
func minFlips(target string) int {
        res := 0
        target = "0" + target
        for i := 1; i < len(target); i++ {
                // 只要后一个与前一个不相等, 都会额外增加一次翻转
                // 从前往后, 每次翻转一样的
                // 如: "10111" => 00000=>(1)1111=>(10)000=>(10111)
                if target[i] != target[i-1] {
                        res++
                }
        }
        return res
}
```

47.11 1530. 好叶子节点对的数量 (2)

- 题目

```
给你二叉树的根节点 root 和一个整数 distance 。
如果二叉树中两个 叶 节点之间的 最短路径长度 小于或者等于 distance␣
↪, 那它们就可以构成一组 好叶子节点对 。
返回树中 好叶子节点对的数量 。
示例 1: 输入: root = [1,2,3,null,4], distance = 3 输出: 1
解释: 树的叶节点是 3 和 4 , 它们之间的最短路径的长度是 3 。这是唯一的好叶子节点对。
示例 2: 输入: root = [1,2,3,4,5,6,7], distance = 3 输出: 2
解释: 好叶子节点对为 [4,5] 和 [6,7] , 最短路径长度都是 2 。
但是叶子节点对 [4,6] 不满足要求, 因为它们之间的最短路径长度为 4 。
示例 3: 输入: root = [7,1,4,6,null,5,3,null,null,null,null,null,2], distance = 3␣
↪输出: 1
解释: 唯一的好叶子节点对是 [2,5] 。
示例 4: 输入: root = [100], distance = 1 输出: 0
示例 5: 输入: root = [1,1,1], distance = 2 输出: 1
提示: tree 的节点数在 [1, 2^10] 范围内。
每个节点的值都在 [1, 100] 之间。
1 <= distance <= 10
```

- 解题思路

```
func countPairs(root *TreeNode, distance int) int {
        _, res := dfs(root, distance)
        return res
}

func dfs(root *TreeNode, distance int) (arr []int, count int) {
        arr = make([]int, distance+2)
        if root == nil {
                return arr, 0
        }
        if root.Left == nil && root.Right == nil {
                arr[1] = 1
                return arr, 0
        }
        leftArr, rightArr := make([]int, distance+1), make([]int, distance+1)
        leftCount, rightCount := 0, 0
        if root.Left != nil {
                leftArr, leftCount = dfs(root.Left, distance)
        }
        if root.Right != nil {
                rightArr, rightCount = dfs(root.Right, distance)
        }
        for i := 1; i <= distance; i++ {
                arr[i+1] = leftArr[i] + rightArr[i]
        }
        count = 0
        for i := 0; i <= distance; i++ {
                for j := 0; j <= distance-i; j++ {
                        count = count + leftArr[i]*rightArr[j]
                }
        }
        return arr, count + leftCount + rightCount
}

# 2
var res int

func countPairs(root *TreeNode, distance int) int {
        res = 0
        dfs(root, distance)
        return res
}

func dfs(root *TreeNode, distance int) (arr []int) {
```

(续下页)

(接上页)

```
        arr = make([]int, distance+2)
        if root == nil {
                return arr
        }
        if root.Left == nil && root.Right == nil {
                arr[1] = 1
                return arr
        }
        leftArr, rightArr := make([]int, distance+1), make([]int, distance+1)
        if root.Left != nil {
                leftArr = dfs(root.Left, distance)
        }
        if root.Right != nil {
                rightArr = dfs(root.Right, distance)
        }
        for i := 1; i <= distance; i++ {
                arr[i+1] = leftArr[i] + rightArr[i]
        }
        for i := 0; i <= distance; i++ {
                for j := 0; j <= distance-i; j++ {
                        res = res + leftArr[i]*rightArr[j]
                }
        }
        return arr
}
```

47.12 1535. 找出数组游戏的赢家 (1)

- 题目

```
给你一个由 不同 整数组成的整数数组 arr 和一个整数 k 。
每回合游戏都在数组的前两个元素（即 arr[0] 和 arr[1] ）之间进行。
比较 arr[0] 与 arr[1] 的大小，较大的整数将会取得这一回合的胜利并保留在位置 0 ，
较小的整数移至数组的末尾。
当一个整数赢得 k 个连续回合时，游戏结束，该整数就是比赛的 赢家 。
返回赢得比赛的整数。
题目数据 保证 游戏存在赢家。
示例 1：输入：arr = [2,1,3,5,4,6,7], k = 2 输出：5
解释：一起看一下本场游戏每回合的情况：
因此将进行 4 回合比赛，其中 5 是赢家，因为它连胜 2 回合。
示例 2：输入：arr = [3,2,1], k = 10 输出：3
解释：3 将会在前 10 个回合中连续获胜。
```

(续下页)

(接上页)

```
示例 3: 输入: arr = [1,9,8,2,3,7,6,4,5], k = 7 输出: 9
示例 4: 输入: arr = [1,11,22,33,44,55,66,77,88,99], k = 1000000000 输出: 99
提示: 2 <= arr.length <= 10^5
    1 <= arr[i] <= 10^6
    arr 所含的整数 各不相同 。
    1 <= k <= 10^9
```

- 解题思路

```
func getWinner(arr []int, k int) int {
        res := arr[0]
        count := 0
        for i := 1; i < len(arr); i++ {
                if arr[i] > res {
                        res = arr[i]
                        count = 1
                } else {
                        count++
                }
                if count == k {
                        break
                }
        }
        return res
}
```

47.13 1536. 排布二进制网格的最少交换次数 (1)

- 题目

```
给你一个nx n的二进制网格grid，每一次操作中，你可以选择网格的相邻两行进行交换。
一个符合要求的网格需要满足主对角线以上的格子全部都是 0。
请你返回使网格满足要求的最少操作次数，如果无法使网格符合要求，请你返回 -1。
主对角线指的是从(1, 1)到(n, n)的这些格子。
示例 1: 输入: grid = [[0,0,1],[1,1,0],[1,0,0]] 输出: 3
示例 2: 输入: grid = [[0,1,1,0],[0,1,1,0],[0,1,1,0],[0,1,1,0]] 输出: -1
解释: 所有行都是一样的，交换相邻行无法使网格符合要求。
示例 3: 输入: grid = [[1,0,0],[1,1,0],[1,1,1]] 输出: 0
提示: n == grid.length
n == grid[i].length
1 <= n<= 200
grid[i][j]要么是0要么是1。
```

- 解题思路

```
func minSwaps(grid [][]int) int {
        n := len(grid)
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                count := 0
                for j := n - 1; j >= 0; j-- {
                        if grid[i][j] == 0 {
                                count++
                        } else {
                                break
                        }
                }
                arr[i] = count
        }
        res := 0
        for i := 0; i < n-1; i++ {
                if arr[i] >= n-1-i { // 满足条件
                        continue
                }
                j := i
                for ; j < n; j++ { // 往后找
                        if arr[j] >= n-1-i {
                                break
                        }
                }
                if j == n { // 找不到
                        return -1
                }
                for ; j > i; j-- { // 前移
                        arr[j], arr[j-1] = arr[j-1], arr[j]
                        res++
                }
        }
        return res
}
```

47.14 1540.K 次操作转变字符串 (2)

- 题目

```
给你两个字符串 s 和 t ，你的目标是在 k 次操作以内把字符串 s 转变成 t 。
在第 i 次操作时 (1 <= i <= k) ，你可以选择进行如下操作：
    选择字符串 s 中满足 1 <= j <= s.length 且之前未被选过的任意下标 j （下标从 1␣
↪开始），
    并将此位置的字符切换 i 次。
    不进行任何操作。
切换 1 次字符的意思是用字母表中该字母的下一个字母替换它（字母表环状接起来，所以 'z'␣
↪切换后会变成 'a'）。
请记住任意一个下标 j 最多只能被操作 1 次。
如果在不超过 k 次操作内可以把字符串 s 转变成 t ，那么请你返回 true ，否则请你返回␣
↪false 。
示例 1：输入：s = "input", t = "ouput", k = 9 输出：true
解释：第 6 次操作时，我们将 'i' 切换 6 次得到 'o' 。第 7 次操作时，我们将 'n' 切换 7␣
↪次得到 'u' 。
示例 2：输入：s = "abc", t = "bcd", k = 10 输出：false
解释：我们需要将每个字符切换 1 次才能得到 t 。我们可以在第 1 次操作时将 'a' 切换成 'b
↪' ，
但另外 2 个字母在剩余操作中无法再转变为 t 中对应字母。
示例 3：输入：s = "aab", t = "bbb", k = 27 输出：true
解释：第 1 次操作时，我们将第一个 'a' 切换 1 次得到 'b' 。
在第 27 次操作时，我们将第二个字母 'a' 切换 27 次得到 'b' 。
提示：1 <= s.length, t.length <= 10^5
    0 <= k <= 10^9
    s 和 t 只包含小写英文字母。
```

- 解题思路

```
func canConvertString(s string, t string, k int) bool {
        if len(s) != len(t) {
                return false
        }
        m := make(map[int]int)
        maxValue := 0
        for i := 0; i < len(s); i++ {
                if s[i] != t[i] {
                        count := (int(t[i]) - int(s[i]) + 26) % 26 // 最少操作的次数
                        // 1 <= i <=␣
↪k，其中i只能出现一次，如果切换的次数一样，需要到下N一圈(i+26xN)才行
                        maxValue = 26*m[count] + count
                        if maxValue > k {
```

(续下页)

(接上页)

```
                                return false
                        }
                        m[count]++
                }
        }
        return true
}

# 2
func canConvertString(s string, t string, k int) bool {
        if len(s) != len(t) {
                return false
        }
        next := [26]int{}
        for i := 0; i < 26; i++ {
                next[i] = i
        }
        for i := 0; i < len(s); i++ {
                if s[i] != t[i] {
                        count := (int(t[i]) - int(s[i]) + 26) % 26 // 最少操作的次数
                        // 1 <= i <=␣
↪k，其中i只能出现一次，如果切换的次数一样，需要到下N一圈(i+26xN)才行
                        if next[count] > k {
                                return false
                        }
                        next[count] = next[count] + 26
                }
        }
        return true
}
```

47.15 1541. 平衡括号字符串的最少插入次数 (2)

- 题目

```
给你一个括号字符串 s ，它只包含字符 '(' 和 ')' 。一个括号字符串被称为平衡的当它满足：
    任何左括号 '(' 必须对应两个连续的右括号 '))' 。
    左括号 '(' 必须在对应的连续两个右括号 '))' 之前。
比方说 "())"， "())(())))" 和 "(())())))" 都是平衡的， ")()"， "()))" 和 "(()))"␣
↪都是不平衡的。
你可以在任意位置插入字符 '(' 和 ')' 使字符串平衡。
请你返回让 s 平衡的最少插入次数。
```

(续下页)

(接上页)

```
示例 1: 输入: s = "(()))" 输出: 1
解释: 第二个左括号有与之匹配的两个右括号, 但是第一个左括号只有一个右括号。
我们需要在字符串结尾额外增加一个 ')' 使字符串变成平衡字符串 "(())))" 。
示例 2: 输入: s = "())" 输出: 0
解释: 字符串已经平衡了。
示例 3: 输入: s = "))())(" 输出: 3
解释: 添加 '(' 去匹配最开头的 '))' , 然后添加 '))' 去匹配最后一个 '(' 。
示例 4: 输入: s = "((((((" 输出: 12
解释: 添加 12 个 ')' 得到平衡字符串。
示例 5: 输入: s = ")))))))" 输出: 5
解释: 在字符串开头添加 4 个 '(' 并在结尾添加 1 个 ')' , 字符串变成平衡字符串
↪"(((()))))))))" 。
提示:
    1 <= s.length <= 10^5
    s 只包含 '(' 和 ')' 。
```

- 解题思路

```
func minInsertions(s string) int {
        res := 0
        left := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        left++
                } else if s[i] == ')' {
                        if i+1 < len(s) && s[i+1] == ')' {
                                i++
                        } else {
                                res++ // 少一个')'补一个')'到2个')'
                        }
                        if left > 0 {
                                left--
                        } else {
                                res++ // 左边无'('补一个
                        }
                }
        }
        res = res + left*2
        return res
}

# 2
func minInsertions(s string) int {
        res := 0
```

(续下页)

(接上页)

```
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        stack = append(stack, '(')
                } else if s[i] == ')' {
                        if i+1 < len(s) && s[i+1] == ')' {
                                if len(stack) == 0 {
                                        // '))'的情况，补'('
                                        res++
                                } else {
                                        stack = stack[:len(stack)-1]
                                }
                                i++
                        } else {
                                if len(stack) == 0 {
                                        // ')('的情况，')'补2'()'
                                        res = res + 2
                                } else {
                                        // '()('的情况，'()'补')'
                                        res++
                                        stack = stack[:len(stack)-1]
                                }
                        }
                }
        }
        res = res + len(stack)*2
        return res
}
```

47.16 1545. 找出第 N 个二进制字符串中的第 K 位 (2)

- 题目

```
给你两个正整数 n 和 k，二进制字符串  Sn 的形成规则如下：
    S1 = "0"
    当 i > 1 时，Si = Si-1 + "1" + reverse(invert(Si-1))
其中 + 表示串联操作，reverse(x) 返回反转 x 后得到的字符串，
而 invert(x) 则会翻转 x 中的每一位（0 变为 1，而 1 变为 0）
例如，符合上述描述的序列的前 4 个字符串依次是：
    S1 = "0"
    S2 = "011"
    S3 = "0111001"
```

(续下页)

(接上页)

```
    S4 = "0111001101110001"
请你返回 Sn 的 第 k 位字符 ，题目数据保证 k 一定在 Sn 长度范围以内。
示例 1: 输入: n = 3, k = 1 输出: "0"
解释: S3 为 "0111001", 其第 1 位为 "0" 。
示例 2: 输入: n = 4, k = 11 输出: "1"
解释: S4 为 "0111001101110001", 其第 11 位为 "1" 。
示例 3: 输入: n = 1, k = 1 输出: "0"
示例 4: 输入: n = 2, k = 3 输出: "1"
提示:
    1 <= n <= 20
    1 <= k <= 2n - 1
```

- 解题思路

```
func findKthBit(n int, k int) byte {
        if n == 1 {
                return '0'
        }
        mid := 1 << (n - 1)
        if k == mid {
                return '1'
        } else if k < mid {
                return findKthBit(n-1, k)
        }
        // 下标从1开始，如n=4,k=15=>3,1(2^4-15=16-15=1)
        return change(findKthBit(n-1, (1<<n)-k))
}

func change(char byte) byte {
        if char == '0' {
                return '1'
        }
        return '0'
}

# 2
func findKthBit(n int, k int) byte {
        arr := generate(n)
        return arr[k-1]
}

func generate(n int) []byte {
        arr := make([][]byte, n)
        arr[0] = []byte{'0'}
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                arr[i] = append(arr[i], arr[i-1]...)
                arr[i] = append(arr[i], '1')
                arr[i] = append(arr[i], reverse(invert(arr[i-1]))...)
        }
        return arr[n-1]
}

func reverse(arr []byte) []byte {
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
        return arr
}

func invert(arr []byte) []byte {
        for i := 0; i < len(arr); i++ {
                arr[i] = '1' - arr[i] + '0'
        }
        return arr
}

# 3
func findKthBit(n int, k int) byte {
        if n == 1 {
                return '0'
        }
        flag := false
        mid := 1 << (n - 1)
        for k > 1 {
                if k == mid {
                        if flag == true {
                                return '0'
                        }
                        return '1'
                } else if k > mid {
                        if flag == true {
                                flag = false
                        } else {
                                flag = true
                        }
                        k = (mid << 1) - k
                }
```

(续下页)

(接上页)

```
                mid = mid >> 1
        }
        if flag == true {
                return '1'
        }
        return '0'
}
```

47.17 1546. 和为目标值的最大数目不重叠非空子数组数目 (2)

- 题目

```
给你一个数组 nums 和一个整数 target 。
请你返回 非空不重叠 子数组的最大数目，且每个子数组中数字和都为 target 。
示例 1：输入：nums = [1,1,1,1,1], target = 2 输出：2
解释：总共有 2 个不重叠子数组（加粗数字表示） [1,1,1,1,1] ，它们的和为目标值 2 。
示例 2：输入：nums = [-1,3,5,1,4,2,-9], target = 6 输出：2
解释：总共有 3 个子数组和为 6 。([5,1], [4,2], [3,5,1,4,2,-9]) 但只有前 2␣
↪个是不重叠的。
示例 3：输入：nums = [-2,6,6,3,5,4,1,2,8], target = 10 输出：3
示例 4：输入：nums = [0,0,0], target = 0 输出：3
提示： 1 <= nums.length <= 10^5
    -10^4 <= nums[i] <= 10^4
    0 <= target <= 10^6
```

- 解题思路

```
func maxNonOverlapping(nums []int, target int) int {
        res := 0
        m := make(map[int]int)
        m[0] = -1
        sum := 0
        prev := -1
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if index, ok := m[sum-target]; ok && index >= prev {
                        res++
                        prev = i
                }
                m[sum] = i
        }
        return res
```

(续下页)

(接上页)

```
}

# 2
func maxNonOverlapping(nums []int, target int) int {
        res := 0
        m := make(map[int]int)
        m[0] = -1
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if _, ok := m[sum-target]; ok {
                        m = make(map[int]int)
                        sum = 0
                        res++
                }
                m[sum] = i
        }
        return res
}
```

47.18 1551. 使数组中所有元素相等的最小操作数 (3)

- 题目

```
存在一个长度为 n 的数组 arr ，其中 arr[i] = (2 * i) + 1 （ 0 <= i < n ）。
一次操作中，你可以选出两个下标，记作 x 和 y （ 0 <= x, y < n ）并使 arr[x] 减去 1 、
arr[y] 加上 1 （即 arr[x] -=1 且 arr[y] += 1 ）。
最终的目标是使数组中的所有元素都 相等 。
题目测试用例将会 保证 ：在执行若干步操作后，数组中的所有元素最终可以全部相等。
给你一个整数 n，即数组的长度。请你返回使数组 arr 中所有元素相等所需的 最小操作数 。
示例 1：输入：n = 3 输出：2
解释：arr = [1, 3, 5]
第一次操作选出 x = 2 和 y = 0，使数组变为 [2, 3, 4]
第二次操作继续选出 x = 2 和 y = 0，数组将会变成 [3, 3, 3]
示例 2：输入：n = 6 输出：9
提示： 1 <= n <= 10^4
```

- 解题思路

```
func minOperations(n int) int {
        res := 0
        for i := 1; i < n; i = i + 2 {
```

(续下页)

(接上页)

```
                res = res + n - i
        }
        return res
}

# 2
func minOperations(n int) int {
        res := 0
        if n == 1 {
                return res
        }
        if n%2 == 1 {
                value := n / 2
                res = value * (value + 1)
        } else {
                value := n / 2
                res = value * value
        }
        return res
}

# 3
func minOperations(n int) int {
        return n*n/4
}
```

47.19 1552. 两球之间的磁力 (2)

- 题目

```
在代号为 C-137 的地球上，Rick␣
↪发现如果他将两个球放在他新发明的篮子里，它们之间会形成特殊形式的磁力
。Rick 有 n 个空的篮子，第 i 个篮子的位置在 position[i] ，Morty 想把 m␣
↪个球放到这些篮子里，
使得任意两球间 最小磁力 最大。
已知两个球如果分别位于 x 和 y ，那么它们之间的磁力为 |x - y| 。
给你一个整数数组 position 和一个整数 m ，请你返回最大化的最小磁力。
示例 1: 输入: position = [1,2,3,4,7], m = 3 输出: 3
解释: 将 3 个球分别放入位于 1，4 和 7 的三个篮子，两球间的磁力分别为 [3, 3,␣
↪6]。最小磁力为 3 。
我们没办法让最小磁力大于 3 。
示例 2: 输入: position = [5,4,3,2,1,1000000000], m = 2 输出: 999999999
```

(续下页)

(接上页)

```
解释：我们使用位于 1 和 1000000000 的篮子时最小磁力最大。
提示：n == position.length
    2 <= n <= 10^5
    1 <= position[i] <= 10^9
    所有 position 中的整数 互不相同 。
    2 <= m <= position.length
```

- 解题思路

```
func maxDistance(position []int, m int) int {
        sort.Ints(position)
        n := len(position)
        maxValue := position[n-1] - position[0] // 最大值
        minValue := position[n-1]               // 求最小值
        for i := 1; i < n; i++ {
                if minValue > position[i]-position[i-1] {
                        minValue = position[i] - position[i-1]
                }
        }
        if m == 2 {
                return maxValue
        }
        left, right := minValue, maxValue
        for left <= right {
                mid := left + (right-left)/2
                if check(position, mid, m) {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return left - 1
}

func check(arr []int, value int, m int) bool {
        count := 1
        target := arr[0] + value
        for i := 0; i < len(arr); i++ {
                if arr[i] >= target {
                        count++
                        target = arr[i] + value
                }
        }
        return count >= m
```

(续下页)

(接上页)

```
}

# 2
func maxDistance(position []int, m int) int {
        sort.Ints(position)
        n := len(position)
        maxValue := (position[n-1] - position[0]) / (m - 1) // 最大值
        minValue := 1                                       // 最小值
        left, right := minValue, maxValue
        res := 1
        for left <= right {
                mid := left + (right-left)/2
                // 满足条件，left=mid+1尝试最大
                if check(position, mid, m) {
                        res = mid
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return res
}

func check(arr []int, value int, m int) bool {
        count := 1
        prev := 0
        for i := 1; i < len(arr); i++ {
                if arr[i]-arr[prev] >= value {
                        count++
                        prev = i
                }
        }
        return count >= m
}
```

47.20 1557. 可以到达所有点的最少点数目 (2)

- 题目

```
给你一个 有向无环图 ， n 个节点编号为 0 到 n-1 ，以及一个边数组 edges ，
其中 edges[i] = [fromi, toi] 表示一条从点  fromi 到点 toi 的有向边。
找到最小的点集使得从这些点出发能到达图中所有点。题目保证解存在且唯一。
你可以以任意顺序返回这些节点编号。
示例 1：输入：n = 6, edges = [[0,1],[0,2],[2,5],[3,4],[4,2]] 输出：[0,3]
解释：从单个节点出发无法到达所有节点。从 0 出发我们可以到达 [0,1,2,5] 。
从 3 出发我们可以到达 [3,4,2,5] 。所以我们输出 [0,3] 。
示例 2：输入：n = 5, edges = [[0,1],[2,1],[3,1],[1,4],[2,4]] 输出：[0,2,3]
解释：注意到节点 0，3 和 2 无法从其他节点到达，所以我们必须将它们包含在结果点集中，
这些点都能到达节点 1 和 4 。
提示：
    2 <= n <= 10^5
    1 <= edges.length <= min(10^5, n * (n - 1) / 2)
    edges[i].length == 2
    0 <= fromi, toi < n
    所有点对 (fromi, toi) 互不相同。
```

- 解题思路

```
func findSmallestSetOfVertices(n int, edges [][]int) []int {
        res := make([]int, 0)
        inEdges := make([]int, n)
        for i := 0; i < len(edges); i++ {
                // a->b
                b := edges[i][1]
                inEdges[b]++ // 入度
        }
        for i := 0; i < len(inEdges); i++ {
                if inEdges[i] == 0 {
                        res = append(res, i)
                }
        }
        return res
}

# 2
func findSmallestSetOfVertices(n int, edges [][]int) []int {
        res := make([]int, 0)
        m := make(map[int]bool)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                m[i] = true
        }
        for i := 0; i < len(edges); i++ {
                b := edges[i][1]
                delete(m, b)
        }
        for k := range m {
                res = append(res, k)
        }
        return res
}
```

47.21 1558. 得到目标数组的最少函数调用次数 (2)

- 题目

```
给你一个与 nums 大小相同且初始值全为 0 的数组 arr ，请你调用以上函数得到整数数组 nums␣
↪。
请你返回将 arr 变成 nums 的最少函数调用次数。
答案保证在 32 位有符号整数以内。
示例 1: 输入: nums = [1,5] 输出: 5
解释: 给第二个数加 1 : [0, 0] 变成 [0, 1] (1 次操作) 。
将所有数字乘以 2 : [0, 1] -> [0, 2] -> [0, 4] (2 次操作) 。
给两个数字都加 1 : [0, 4] -> [1, 4] -> [1, 5] (2 次操作) 。
总操作次数为: 1 + 2 + 2 = 5 。
示例 2: 输入: nums = [2,2] 输出: 3
解释: 给两个数字都加 1 : [0, 0] -> [0, 1] -> [1, 1] (2 次操作) 。
将所有数字乘以 2 : [1, 1] -> [2, 2] (1 次操作) 。
总操作次数为: 2 + 1 = 3 。
示例 3: 输入: nums = [4,2,5] 输出: 6
解释: (初始) [0,0,0] -> [1,0,0] -> [1,0,1] -> [2,0,2] -> [2,1,2] ->
[4,2,4] -> [4,2,5] (nums 数组) 。
示例 4: 输入: nums = [3,2,2,4] 输出: 7
示例 5: 输入: nums = [2,4,8,16] 输出: 8
提示: 1 <= nums.length <= 10^5
    0 <= nums[i] <= 10^9
```

- 解题思路

```
func minOperations(nums []int) int {
        res := 0
        for judge(nums) != true {
```

(续下页)

(接上页)

```
                res++ // 最后一次循环会多算一次,最后要减掉
                res = res + div(nums)
        }
        return res - 1
}

func judge(arr []int) bool {
        for i := 0; i < len(arr); i++ {
                if arr[i] != 0 {
                        return false
                }
        }
        return true
}

func div(arr []int) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] == 0 {
                        continue
                }
                if arr[i]%2 == 1 {
                        res++
                }
                arr[i] = arr[i] / 2
        }
        return res
}

# 2
func minOperations(nums []int) int {
        res := 0
        maxValue := 0 // 保存x2的次数
        for i := 0; i < len(nums); i++ {
                count := 0
                n := nums[i]
                for n != 0 {
                        if n%2 == 0 {
                                n = n / 2
                                count++
                        } else {
                                n = n - 1
                                res = res + 1 // 单独加1
```

(续下页)

(接上页)

```
                        }
                }
                maxValue = max(maxValue, count)
        }
        return res + maxValue
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

47.22 1561. 你可以获得的最大硬币数目 (3)

- 题目

```
有 3n 堆数目不一的硬币，你和你的朋友们打算按以下方式分硬币：
    每一轮中，你将会选出 任意 3 堆硬币（不一定连续）。
    Alice 将会取走硬币数量最多的那一堆。
    你将会取走硬币数量第二多的那一堆。
    Bob 将会取走最后一堆。
    重复这个过程，直到没有更多硬币。
给你一个整数数组 piles ，其中 piles[i] 是第 i 堆中硬币的数目。
返回你可以获得的最大硬币数目。
示例 1：输入：piles = [2,4,1,2,7,8] 输出：9
解释：选出 (2, 7, 8) ，Alice 取走 8 枚硬币的那堆，你取走 7 枚硬币的那堆，Bob␣
↪取走最后一堆。
选出 (1, 2, 4) ，Alice 取走 4 枚硬币的那堆，你取走 2 枚硬币的那堆，Bob 取走最后一堆。
你可以获得的最大硬币数目：7 + 2 = 9.
考虑另外一种情况，如果选出的是 (1, 2, 8) 和 (2, 4, 7) ，你就只能得到 2 + 4 = 6␣
↪枚硬币，这不是最优解。
示例 2：输入：piles = [2,4,5] 输出：4
示例 3：输入：piles = [9,8,7,6,5,1,2,3,4] 输出：18
提示：
    3 <= piles.length <= 10^5
    piles.length % 3 == 0
    1 <= piles[i] <= 10^4
```

- 解题思路

```
func maxCoins(piles []int) int {
        res := 0
        sort.Ints(piles)
        count := len(piles) / 3
        for i := 0; i < count; i++ {
                index := len(piles) - 1 - i*2 - 1
                res = res + piles[index]
        }
        return res
}

# 2
func maxCoins(piles []int) int {
        res := 0
        sort.Slice(piles, func(i, j int) bool {
                return piles[i] > piles[j]
        })
        for i := 0; i < len(piles)/3; i++ {
                res = res + piles[i*2+1]
        }
        return res
}

# 3
func maxCoins(piles []int) int {
        res := 0
        sort.Ints(piles)
        left := 0
        right := len(piles)-1
        for left < right{
                left++
                right--
                res = res + piles[right]
                right--
        }
        return res
}
```

47.23 1562. 查找大小为 M 的最新分组 (2)

- 题目

```
给你一个数组 arr ，该数组表示一个从 1 到 n 的数字排列。
有一个长度为 n 的二进制字符串，该字符串上的所有位最初都设置为 0 。
在从 1 到 n 的每个步骤 i 中（假设二进制字符串和 arr 都是从 1 开始索引的情况下），
二进制字符串上位于位置 arr[i] 的位将会设为 1 。
给你一个整数 m ，请你找出二进制字符串上存在长度为 m 的一组 1 的最后步骤。
一组 1 是一个连续的、由 1 组成的子串，且左右两边不再有可以延伸的 1 。
返回存在长度 恰好 为 m 的 一组 1  的最后步骤。如果不存在这样的步骤，请返回 -1 。
示例 1：输入：arr = [3,5,1,2,4], m = 1 输出：4
解释：步骤 1："00100", 由 1 构成的组：["1"]
步骤 2："00101", 由 1 构成的组：["1", "1"]
步骤 3："10101", 由 1 构成的组：["1", "1", "1"]
步骤 4："11101", 由 1 构成的组：["111", "1"]
步骤 5："11111", 由 1 构成的组：["11111"]
存在长度为 1 的一组 1 的最后步骤是步骤 4 。
示例 2：输入：arr = [3,1,5,4,2], m = 2 输出：-1
解释：步骤 1："00100", 由 1 构成的组：["1"]
步骤 2："10100", 由 1 构成的组：["1", "1"]
步骤 3："10101", 由 1 构成的组：["1", "1", "1"]
步骤 4："10111", 由 1 构成的组：["1", "111"]
步骤 5："11111", 由 1 构成的组：["11111"]
不管是哪一步骤都无法形成长度为 2 的一组 1 。
示例 3：输入：arr = [1], m = 1 输出：1
示例 4：输入：arr = [2,1], m = 2 输出：2
提示：n == arr.length
    1 <= n <= 10^5
    1 <= arr[i] <= n
    arr 中的所有整数 互不相同
    1 <= m <= arr.length
```

- 解题思路

```
func findLatestStep(arr []int, m int) int {
        res := -1
        temp := make([]int, len(arr)+2)
        M := make(map[int]bool)
        for i := 0; i < len(arr); i++ {
                index := arr[i]
                temp[index] = 1
                left := temp[index-1]
                right := temp[index+1]
```

(续下页)

(接上页)

```
                total := 1 + left + right
                temp[index] = total
                temp[index-left] = total
                temp[index+right] = total
                for k := range M {
                        if index-left <= k && k < index {
                                delete(M, k)
                        }
                        if index < k && k <= index+right {
                                delete(M, k)
                        }
                }
                if total == m {
                        M[index] = true
                }
                if len(M) > 0 {
                        res = i + 1
                }
        }
        return res
}

# 2
func findLatestStep(arr []int, m int) int {
        if len(arr) == m {
                return len(arr)
        }
        res := -1
        temp := make([]int, len(arr)+2)
        for i := 0; i < len(arr); i++ {
                index := arr[i]
                total := 1 + temp[index-1] + temp[index+1]
                // 上一步一定存在m
                if temp[index-1] == m || temp[index+1] == m {
                        res = i
                }
                temp[index-temp[index-1]] = total
                temp[index+temp[index+1]] = total
        }
        return res
}
```

47.24 1567. 乘积为正数的最长子数组长度 (2)

- 题目

```
给你一个整数数组 nums ，请你求出乘积为正数的最长子数组的长度。
一个数组的子数组是由原数组中零个或者更多个连续数字组成的数组。
请你返回乘积为正数的最长子数组长度。
示例 1：输入：nums = [1,-2,-3,4] 输出：4
解释：数组本身乘积就是正数，值为 24 。
示例 2：输入：nums = [0,1,-2,-3,-4] 输出：3
解释：最长乘积为正数的子数组为 [1,-2,-3] ，乘积为 6 。
注意，我们不能把 0 也包括到子数组中，因为这样乘积为 0 ，不是正数。
示例 3：输入：nums = [-1,-2,-3,0,1] 输出：2
解释：乘积为正数的最长子数组是 [-1,-2] 或者 [-2,-3] 。
示例 4：输入：nums = [-1,2] 输出：1
示例 5：输入：nums = [1,2,3,5,-6,4,0,10] 输出：4
提示：1 <= nums.length <= 10^5
    -10^9 <= nums[i] <= 10^9
```

- 解题思路

```
func getMaxLen(nums []int) int {
        arr := make([][]int, 0)
        temp := make([]int, 0)
        res := 0
        total := 1
        for i := 0; i < len(nums); i++ {
                if nums[i] > 0 {
                        total = total * 1
                } else if nums[i] < 0 {
                        total = total * -1
                } else {
                        total = 0
                }
                if total > 0 {
                        temp = append(temp, 1)
                } else if nums[i] == 0 {
                        if len(temp) > 0 {
                                arr = append(arr, temp)
                                temp = make([]int, 0)
                        }
                        total = 1
                } else if total < 0 {
                        temp = append(temp, -1)
```

(续下页)

(接上页)

```
                }
        }
        if len(temp) > 0 {
                arr = append(arr, temp)
        }
        for i := 0; i < len(arr); i++ {
                tempArr := arr[i]
                // 1、寻找最后一个1
                left, right := 0, len(tempArr)-1
                for 0 <= right && tempArr[right] != 1 {
                        right--
                }
                res = max(res, right-left+1)
                // 2、寻找前后2个-1
                left, right = 0, len(tempArr)-1
                for left < len(tempArr) && tempArr[left] != -1 {
                        left++
                }
                for 0 <= right && tempArr[right] != -1 {
                        right--
                }
                res = max(res, right-left)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func getMaxLen(nums []int) int {
        dp := make([][2]int, len(nums)+1)
        res := 0
        for i := 1; i <= len(nums); i++ {
                if nums[i-1] == 0 {
                        dp[i][0] = 0
                        dp[i][1] = 0
                } else if nums[i-1] > 0 {
                        dp[i][0] = dp[i-1][0] + 1
```

(续下页)

(接上页)

```
                        if dp[i-1][1] != 0 {
                                dp[i][1] = dp[i-1][1] + 1
                        } else {
                                dp[i][1] = 0
                        }
                } else {
                        if dp[i-1][1] != 0 {
                                dp[i][0] = dp[i-1][1] + 1
                        } else {
                                dp[i][0] = 0
                        }
                        dp[i][1] = dp[i-1][0] + 1
                }
                res = max(res, dp[i][0])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

47.25 1568. 使陆地分离的最少天数 (1)

- 题目

```
给你一个由若干 0 和 1 组成的二维网格 grid ，其中 0 表示水，而 1 表示陆地。
岛屿由水平方向或竖直方向上相邻的 1 （陆地）连接形成。
如果 恰好只有一座岛屿 ，则认为陆地是 连通的 ；否则，陆地就是 分离的 。
一天内，可以将任何单个陆地单元（1）更改为水单元（0）。
返回使陆地分离的最少天数。
示例 1：输入：grid = [[0,1,1,0],[0,1,1,0],[0,0,0,0]] 输出：2
解释：至少需要 2 天才能得到分离的陆地。
将陆地 grid[1][1] 和 grid[0][2] 更改为水，得到两个分离的岛屿。
示例 2：输入：grid = [[1,1]] 输出：2
解释：如果网格中都是水，也认为是分离的 ([[1,1]] -> [[0,0]])，0 岛屿。
示例 3：输入：grid = [[1,0,1,0]] 输出：0
示例 4：输入：grid = [[1,1,0,1,1],
             [1,1,1,1,1],
```

(续下页)

(接上页)

```
            [1,1,0,1,1],
            [1,1,0,1,1]]
            输出：1
示例 5：输入：grid = [[1,1,0,1,1],
            [1,1,1,1,1],
            [1,1,0,1,1],
            [1,1,1,1,1]]
输出：2
提示：1 <= grid.length, grid[i].length <= 30
    grid[i][j] 为 0 或 1
```

- 解题思路

```
func minDays(grid [][]int) int {
        temp := copyArr(grid)
        nums := numIslands(temp)
        if nums >= 2 {
                return 0
        }
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[0]); j++ {
                        if grid[i][j] == 0 {
                                continue
                        }
                        temp = copyArr(grid)
                        temp[i][j] = 0
                        if numIslands(temp) == 2 {
                                return 1
                        }
                }
        }
        return 2
}

func numIslands(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 1 {
                                dfs(grid, i, j)
                                res++
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}

func dfs(grid [][]int, i, j int) {
        if i < 0 || j < 0 || i >= len(grid) || j >= len(grid[0]) ||
                grid[i][j] == 0 {
                return
        }
        grid[i][j] = 0
        dfs(grid, i+1, j)
        dfs(grid, i-1, j)
        dfs(grid, i, j+1)
        dfs(grid, i, j-1)
}

func copyArr(grid [][]int) [][]int {
        temp := make([][]int, len(grid))
        for i := 0; i < len(grid); i++ {
                temp[i] = make([]int, len(grid[i]))
                for j := 0; j < len(grid[i]); j++ {
                        temp[i][j] = grid[i][j]
                }
        }
        return temp
}
```

47.26 1573. 分割字符串的方案数 (2)

- 题目

```
给你一个二进制串 s （一个只包含 0 和 1 的字符串），
我们可以将 s 分割成 3 个 非空 字符串 s1, s2, s3 （s1 + s2 + s3 = s）。
请你返回分割 s 的方案数，满足 s1, s2 和 s3 中字符 '1' 的数目相同。
由于答案可能很大，请将它对 10^9 + 7 取余后返回。
示例 1: 输入：s = "10101" 输出：4
解释：总共有 4 种方法将 s 分割成含有 '1' 数目相同的三个子字符串。
"1|010|1"
"1|01|01"
"10|10|1"
"10|1|01"
示例 2: 输入：s = "1001" 输出：0
示例 3: 输入：s = "0000" 输出：3
```

(续下页)

(接上页)

```
解释：总共有 3 种分割 s 的方法。
"0|0|00"
"0|00|0"
"00|0|0"
示例 4：输入：s = "100100010100110" 输出：12
提示：
    s[i] == '0' 或者 s[i] == '1'
    3 <= s.length <= 10^5
```

- 解题思路

```
func numWays(s string) int {
        total := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '1' {
                        total++
                }
        }
        if total%3 != 0 {
                return 0
        }
        if total == 0 {
                return ((len(s) - 2) * (len(s) - 1) / 2) % 1000000007
        }
        single := total / 3
        first, second := single*1, single*2
        var start, left, right int
        for i := 0; i < len(s); i++ {
                if s[i] == '1' {
                        start++
                } else {
                        continue
                }
                if start == first {
                        left = i + 1
                } else if start == first+1 {
                        left = i - left
                }
                if start == second {
                        right = i + 1
                } else if start == second+1 {
                        right = i - right
                }
        }
```

(续下页)

(接上页)

```
        return (left + 1) * (right + 1) % 1000000007
}
```

47.27 1574. 删除最短的子数组使剩余数组有序 (1)

- 题目

```
给你一个整数数组 arr ，请你删除一个子数组（可以为空），使得 arr 中剩下的元素是 非递减
↪的。
一个子数组指的是原数组中连续的一个子序列。
请你返回满足题目要求的最短子数组的长度。
示例 1：输入：arr = [1,2,3,10,4,2,3,5] 输出：3
解释：我们需要删除的最短子数组是 [10,4,2] ，长度为 3 。剩余元素形成非递减数组 [1,2,3,
↪3,5] 。
另一个正确的解为删除子数组 [3,10,4] 。
示例 2：输入：arr = [5,4,3,2,1] 输出：4
解释：由于数组是严格递减的，我们只能保留一个元素。所以我们需要删除长度为 4 的子数组，
要么删除 [5,4,3,2]，要么删除 [4,3,2,1]。
示例 3：输入：arr = [1,2,3] 输出：0
解释：数组已经是非递减的了，我们不需要删除任何元素。
示例 4：输入：arr = [1] 输出：0
提示：
    1 <= arr.length <= 10^5
    0 <= arr[i] <= 10^9
```

- 解题思路

```
func findLengthOfShortestSubarray(arr []int) int {
        if len(arr) < 2 {
                return 0
        }
        flag := true
        left := 0
        for i := 1; i < len(arr); i++ {
                if arr[i-1] > arr[i] {
                        flag = false
                        break
                } else {
                        left = i
                }
        }
        if flag == true {
```

(续下页)

(接上页)

```
                return 0
        }
        right := len(arr) - 1
        for i := len(arr) - 1; i >= 1; i-- {
                if arr[i-1] <= arr[i] {
                        right = i - 1
                } else {
                        break
                }
        }
        leftC, rightC := 0, 0
        for i := left; i >= 0 && arr[i] > arr[right]; i-- {
                leftC++
        }
        for i := right; i < len(arr) && arr[i] < arr[left]; i++ {
                rightC++
        }
        res := 0
        res = max(res, (left+1)+(len(arr)-right)-leftC)
        res = max(res, (left+1)+(len(arr)-right)-rightC)
        return len(arr) - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

47.28 1577. 数的平方等于两数乘积的方法数 (1)

- 题目

```
给你两个整数数组 nums1 和 nums2 ，请你返回根据以下规则形成的三元组的数目（类型 1
↪和类型 2 ）：
    类型 1：三元组 (i, j, k) ，如果 nums1[i]2 == nums2[j] * nums2[k]
    其中 0 <= i < nums1.length 且 0 <= j < k < nums2.length
    类型 2：三元组 (i, j, k) ，如果 nums2[i]2 == nums1[j] * nums1[k]
    其中 0 <= i < nums2.length 且 0 <= j < k < nums1.length
示例 1：输入：nums1 = [7,4], nums2 = [5,2,8,9] 输出：1
解释：类型 1：(1,1,2), nums1[1]^2 = nums2[1] * nums2[2] (4^2 = 2 * 8)
```

(续下页)

(接上页)

```
示例 2: 输入: nums1 = [1,1], nums2 = [1,1,1] 输出: 9
解释: 所有三元组都符合题目要求, 因为 1^2 = 1 * 1
类型 1: (0,0,1), (0,0,2), (0,1,2), (1,0,1), (1,0,2), (1,1,2),
nums1[i]^2 = nums2[j] * nums2[k]
类型 2: (0,0,1), (1,0,1), (2,0,1), nums2[i]^2 = nums1[j] * nums1[k]
示例 3: 输入: nums1 = [7,7,8,3], nums2 = [1,2,9,7] 输出: 2
解释: 有两个符合题目要求的三元组
类型 1: (3,0,2), nums1[3]^2 = nums2[0] * nums2[2]
类型 2: (3,0,1), nums2[3]^2 = nums1[0] * nums1[1]
示例 4: 输入: nums1 = [4,7,9,11,23], nums2 = [3,5,1024,12,18] 输出: 0
解释: 不存在符合题目要求的三元组
提示: 1 <= nums1.length, nums2.length <= 1000
    1 <= nums1[i], nums2[i] <= 10^5
```

- 解题思路

```
func numTriplets(nums1 []int, nums2 []int) int {
        res := 0
        res = res + getCount(nums1, nums2)
        res = res + getCount(nums2, nums1)
        return res
}

func getCount(nums1 []int, nums2 []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(nums1); i++ {
                m[nums1[i]*nums1[i]]++
        }
        for i := 0; i < len(nums2); i++ {
                for j := i + 1; j < len(nums2); j++ {
                        res = res + m[nums2[i]*nums2[j]]
                }
        }
        return res
}
```

47.29 1578. 避免重复字母的最小删除成本 (2)

- 题目

```
给你一个字符串 s 和一个整数数组 cost ，其中 cost[i] 是从 s 中删除字符 i 的代价。
返回使字符串任意相邻两个字母不相同的最小删除成本。
请注意，删除一个字符后，删除其他字符的成本不会改变。
示例 1：输入：s = "abaac", cost = [1,2,3,4,5] 输出：3
解释：删除字母 "a" 的成本为 3，然后得到 "abac"（字符串中相邻两个字母不相同）。
示例 2：输入：s = "abc", cost = [1,2,3] 输出：0
解释：无需删除任何字母，因为字符串中不存在相邻两个字母相同的情况。
示例 3：输入：s = "aabaa", cost = [1,2,3,4,1] 输出：2
解释：删除第一个和最后一个字母，得到字符串 ("aba") 。
提示：  s.length == cost.length
    1 <= s.length, cost.length <= 10^5
    1 <= cost[i] <= 10^4
    s 中只含有小写英文字母
```

- 解题思路

```
func minCost(s string, cost []int) int {
        res := 0
        cur := 0
        for i := 0; i < len(s)-1; i++ {
                if s[cur] == s[i+1] {
                        res = res + min(cost[cur], cost[i+1])
                        if cost[cur] < cost[i+1] {
                                cur = i + 1 // 相同，保存较大的
                        }
                } else {
                        cur = i + 1
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
```

(续下页)

(接上页)

```
func minCost(s string, cost []int) int {
        res := 0
        for i := 0; i < len(s)-1; i++ {
                if s[i] == s[i+1] {
                        res = res + min(cost[i], cost[i+1])
                        if cost[i] > cost[i+1] {
                                cost[i], cost[i+1] = cost[i+1], cost[i] //␣
↪相同，把较大的存在后面
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

47.30 1583. 统计不开心的朋友 (1)

- 题目

```
给你一份 n 位朋友的亲近程度列表，其中 n 总是 偶数 。
对每位朋友 i，preferences[i] 包含一份 按亲近程度从高到低排列 的朋友列表。
换句话说，排在列表前面的朋友与 i 的亲近程度比排在列表后面的朋友更高。
每个列表中的朋友均以 0 到 n-1 之间的整数表示。
所有的朋友被分成几对，配对情况以列表 pairs 给出，其中 pairs[i] = [xi, yi] 表示 xi 与␣
↪yi 配对，
且 yi 与 xi 配对。
但是，这样的配对情况可能会是其中部分朋友感到不开心。在 x 与 y 配对且 u 与 v␣
↪配对的情况下，
如果同时满足下述两个条件，x 就会不开心：
    x 与 u 的亲近程度胜过 x 与 y，且
    u 与 x 的亲近程度胜过 u 与 v
返回 不开心的朋友的数目 。
示例 1：输入：n = 4, preferences = [[1, 2, 3], [3, 2, 0], [3, 1, 0], [1, 2, 0]],
pairs = [[0, 1], [2, 3]] 输出：2
解释：朋友 1 不开心，因为：
- 1 与 0 配对，但 1 与 3 的亲近程度比 1 与 0 高，且
```

(续下页)

(接上页)

```
- 3 与 1 的亲近程度比 3 与 2 高。
朋友 3 不开心，因为：
- 3 与 2 配对，但 3 与 1 的亲近程度比 3 与 2 高，且
- 1 与 3 的亲近程度比 1 与 0 高。
朋友 0 和 2 都是开心的。
示例 2：输入：n = 2, preferences = [[1], [0]], pairs = [[1, 0]] 输出：0
解释：朋友 0 和 1 都开心。
示例 3：输入：n = 4, preferences = [[1, 3, 2], [2, 3, 0], [1, 3, 0], [0, 2, 1]],
pairs = [[1, 3], [0, 2]] 输出：4
提示：2 <= n <= 500
    n 是偶数
    preferences.length == n
    preferences[i].length == n - 1
    0 <= preferences[i][j] <= n - 1
    preferences[i] 不包含 i
    preferences[i] 中的所有值都是独一无二的
    pairs.length == n/2
    pairs[i].length == 2
    xi != yi
    0 <= xi, yi <= n - 1
    每位朋友都 恰好 被包含在一对中
```

- 解题思路

```go
func unhappyFriends(n int, preferences [][]int, pairs [][]int) int {
        arr := make(map[int]map[int]int)
        for i := 0; i < n; i++ {
                arr[i] = make(map[int]int)
        }
        for i := 0; i < len(preferences); i++ {
                total := n
                for j := 0; j < len(preferences[i]); j++ {
                        arr[i][preferences[i][j]] = total
                        total = total - 1
                }
        }
        m := make(map[int]bool)
        for i := 0; i < len(pairs); i++ {
                x, y := pairs[i][0], pairs[i][1]
                x1, y1 := pairs[i][1], pairs[i][0]
                for j := 0; j < len(pairs); j++ {
                        if i == j {
                                continue
                        }
```

(续下页)

(接上页)

```
                        u, v := pairs[j][0], pairs[j][1]
                        u1, v1 := pairs[j][1], pairs[j][0]
                        if arr[x][u] > arr[x][y] && arr[u][x] > arr[u][v] {
                                m[x] = true
                        }
                        if arr[x1][u] > arr[x1][y1] && arr[u][x1] > arr[u][v] {
                                m[x1] = true
                        }
                        if arr[x][u1] > arr[x][y] && arr[u1][x] > arr[u1][v1] {
                                m[x] = true
                        }
                        if arr[x1][u1] > arr[x1][y1] && arr[u1][x1] > arr[u1][v1] {
                                m[x1] = true
                        }
                }
        }
        return len(m)
}
```

47.31 1584. 连接所有点的最小费用 (3)

- 题目

```
给你一个points数组，表示 2D 平面上的一些点，其中points[i] = [xi, yi]。
连接点[xi, yi] 和点[xj, yj]的费用为它们之间的 曼哈顿距离：|xi - xj| + |yi -␣
↪yj|，其中|val|表示val的绝对值。
请你返回将所有点连接的最小总费用。只有任意两点之间␣
↪有且仅有一条简单路径时，才认为所有点都已连接。
示例 1：输入：points = [[0,0],[2,2],[3,10],[5,2],[7,0]] 输出：20
解释：我们可以按照上图所示连接所有点得到最小总费用，总费用为 20 。
注意到任意两个点之间只有唯一一条路径互相到达。
示例 2：输入：points = [[3,12],[-2,5],[-4,1]] 输出：18
示例 3：输入：points = [[0,0],[1,1],[1,0],[-1,1]] 输出：4
示例 4：输入：points = [[-1000000,-1000000],[1000000,1000000]] 输出：4000000
示例 5：输入：points = [[0,0]] 输出：0
提示：1 <= points.length <= 1000
-106<= xi, yi <= 106
所有点(xi, yi)两两不同。
```

- 解题思路

```
func minCostConnectPoints(points [][]int) int {
        n := len(points)
        arr := make([][3]int, 0)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        arr = append(arr, [3]int{i, j, dis(points[i], points[j])}) //␣
↪a=>b c
                }
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][2] < arr[j][2]
        })
        return Kruskal(n, arr)
}

func Kruskal(n int, arr [][3]int) int {
        res := 0
        fa = Init(n)
        count := 0
        for i := 0; i < len(arr); i++ {
                a, b, c := arr[i][0], arr[i][1], arr[i][2]
                if query(a, b) == false {
                        union(a, b)
                        res = res + c
                        count++
                        if count == n-1 {
                                break
                        }
                }
        }
        return res
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}
```

(续下页)

(接上页)

```
// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}

func dis(a, b []int) int {
        return abs(a[0]-b[0]) + abs(a[1]-b[1])
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func minCostConnectPoints(points [][]int) int {
        n := len(points)
        arr := make([][]int, n) // 邻接矩阵
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        c := dis(points[i], points[j])
                        arr[i][j] = c
                        arr[j][i] = c
                }
        }
        return Prime(arr)
```

(续下页)

(接上页)

```
}

// Prime: 传入邻接矩阵
func Prime(arr [][]int) int {
        res := 0
        n := len(arr)
        visited := make([]bool, n)
        target := 0
        visited[target] = true // 任选1个即可
        dis := make([]int, n)  // 任意选择的节点: 到其它点的距离
        for i := 0; i < n; i++ {
                dis[i] = arr[target][i]
        }
        for i := 1; i < n; i++ { // 执行n-1次: n-1条边
                var index int
                minValue := math.MaxInt32
                for j := 0; j < n; j++ { // 寻找: 未访问过的最短边
                        if visited[j] == false && dis[j] < minValue {
                                minValue = dis[j]
                                index = j
                        }
                }
                visited[index] = true    // 标记为访问过的点
                res = res + minValue     // 加上最短边
                for j := 0; j < n; j++ { //␣
↪更新距离: 以index为起点, 更新生成树到每一个非树顶点的距离
                        if visited[j] == false && dis[j] > arr[index][j] {
                                dis[j] = arr[index][j]
                        }
                }
        }
        return res
}

func dis(a, b []int) int {
        return abs(a[0]-b[0]) + abs(a[1]-b[1])
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
```

(续下页)

(接上页)

```
}

# 3
func minCostConnectPoints(points [][]int) int {
        n := len(points)
        arr := make([][]int, n) // 邻接矩阵
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        c := dis(points[i], points[j])
                        arr[i][j] = c
                        arr[j][i] = c
                }
        }
        return Prime(arr)
}

// Prime-堆优化-邻接表
func Prime(arr [][]int) int {
        res := 0
        n := len(arr)
        visited := make([]bool, n)
        target := 0
        dis := make([]int, n) // 任意选择的节点：到其它点的距离
        for i := 0; i < n; i++ {
                dis[i] = math.MaxInt32
        }
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [2]int{0, target}) // [2]int{距离，下标}
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([2]int)
                minValue, index := node[0], node[1]
                if visited[index] == true {
                        continue
                }
                visited[index] = true
                res = res + minValue
                for j := 0; j < len(arr[index]); j++ {
                        if visited[j] == false && dis[j] > arr[index][j] {
                                dis[j] = arr[index][j]
```

(续下页)

(接上页)

```
                                heap.Push(&intHeap, [2]int{arr[index][j], j})
                        }
                }
        }
        return res
}

type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][0] < h[j][0]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

func dis(a, b []int) int {
        return abs(a[0]-b[0]) + abs(a[1]-b[1])
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

47.32 1589. 所有排列中的最大和 (1)

- 题目

```
有一个整数数组 nums ，和一个查询数组 requests ，其中 requests[i] = [starti, endi] 。
第 i 个查询求 nums[starti] + nums[starti + 1] + ... + nums[endi - 1] + nums[endi]
↪的结果 ，
starti 和 endi 数组索引都是 从 0 开始 的。
你可以任意排列 nums 中的数字，请你返回所有查询结果之和的最大值。
由于答案可能会很大，请你将它对 109 + 7 取余 后返回。
示例 1：输入：nums = [1,2,3,4,5], requests = [[1,3],[0,1]] 输出：19
解释：一个可行的 nums 排列为 [2,1,3,4,5]，并有如下结果：
requests[0] -> nums[1] + nums[2] + nums[3] = 1 + 3 + 4 = 8
requests[1] -> nums[0] + nums[1] = 2 + 1 = 3
总和为：8 + 3 = 11。
一个总和更大的排列为 [3,5,4,2,1]，并有如下结果：
requests[0] -> nums[1] + nums[2] + nums[3] = 5 + 4 + 2 = 11
requests[1] -> nums[0] + nums[1] = 3 + 5  = 8
总和为： 11 + 8 = 19，这个方案是所有排列中查询之和最大的结果。
示例 2：输入：nums = [1,2,3,4,5,6], requests = [[0,1]] 输出：11
解释：一个总和最大的排列为 [6,5,4,3,2,1] ，查询和为 [11]。
示例 3：输入：nums = [1,2,3,4,5,10], requests = [[0,2],[1,3],[1,1]] 输出：47
解释：一个和最大的排列为 [4,10,5,3,2,1] ，查询结果分别为 [19,18,10]。
提示：n == nums.length
    1 <= n <= 105
    0 <= nums[i] <= 105
    1 <= requests.length <= 105
    requests[i].length == 2
    0 <= starti <= endi < n
```

- 解题思路

```
func maxSumRangeQuery(nums []int, requests [][]int) int {
        d := make([]int, len(nums)+1)
        arr := make([]int, len(nums))
        for i := 0; i < len(requests); i++ {
                start := requests[i][0]
                end := requests[i][1]
                d[start]++
                d[end+1]--
        }
        arr[0] = d[0]
        for i := 1; i < len(nums); i++ {
                arr[i] = d[i] + arr[i-1]
```

(续下页)

(接上页)

```
        }
        sort.Ints(arr)
        sort.Ints(nums)
        res := 0
        for i := len(arr) - 1; i >= 0; i-- {
                res = (res + arr[i]*nums[i]) % 1000000007
        }
        return res
}
```

47.33 1590. 使数组和能被 P 整除 (2)

- 题目

```
给你一个正整数数组 nums，请你移除 最短 子数组（可以为 空），使得剩余元素的 和 能被 p 整除。
不允许 将整个数组都移除。
请你返回你需要移除的最短子数组的长度，如果无法满足题目要求，返回 -1 。
子数组 定义为原数组中连续的一组元素。
示例 1：输入：nums = [3,1,4,2], p = 6 输出：1
解释：nums 中元素和为 10，不能被 p 整除。我们可以移除子数组 [4] ，剩余元素的和为 6 。
示例 2：输入：nums = [6,3,5,2], p = 9 输出：2
解释：我们无法移除任何一个元素使得和被 9 整除，最优方案是移除子数组 [5,2]，剩余元素为 [6,3]，和为 9 。
示例 3：输入：nums = [1,2,3], p = 3 输出：0
解释：和恰好为 6 ，已经能被 3 整除了。所以我们不需要移除任何元素。
示例 4：输入：nums = [1,2,3], p = 7 输出：-1
解释：没有任何方案使得移除子数组后剩余元素的和被 7 整除。
示例 5：输入：nums = [1000000000,1000000000,1000000000], p = 3 输出：0
提示：  1 <= nums.length <= 105
    1 <= nums[i] <= 109
    1 <= p <= 109
```

- 解题思路

```
func minSubarray(nums []int, p int) int {
        n := len(nums)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = (arr[i] + nums[i]) % p
        }
        if arr[n] == 0 {
```

(续下页)

(接上页)

```
                return 0
        }
        targetValue := arr[n]
        res := n
        m := make(map[int]int)
        m[0] = -1
        for i := 0; i < n; i++ {
                target := (arr[i+1] - targetValue + p) % p
                if value, ok := m[target]; ok {
                        if res > i-value {
                                res = i - value
                        }
                }
                m[arr[i+1]] = i
        }
        if res == n {
                return -1
        }
        return res
}

# 2
func minSubarray(nums []int, p int) int {
        n := len(nums)
        targetValue := 0
        for i := 0; i < n; i++ {
                targetValue = (targetValue + nums[i]) % p
        }
        if targetValue == 0 {
                return 0
        }
        res := n
        m := make(map[int]int)
        m[0] = -1
        total := 0
        for i := 0; i < n; i++ {
                total = (total + nums[i] + p) % p
                target := (total - targetValue + p) % p
                if value, ok := m[target]; ok {
                        if res > i-value {
                                res = i - value
                        }
                }
```

(续下页)

(接上页)

```
                m[total] = i
        }
        if res == n {
                return -1
        }
        return res
}
```

47.34 1593. 拆分字符串使唯一子字符串的数目最大 (1)

- 题目

```
给你一个字符串 s ，请你拆分该字符串，并返回拆分后唯一子字符串的最大数目。
字符串 s 拆分后可以得到若干 非空子字符串 ，这些子字符串连接后应当能够还原为原字符串。
但是拆分出来的每个子字符串都必须是 唯一的 。
注意：子字符串 是字符串中的一个连续字符序列。
示例 1：输入：s = "ababccc" 输出：5
解释：一种最大拆分方法为 ['a', 'b', 'ab', 'c', 'cc'] 。
像 ['a', 'b', 'a', 'b', 'c', 'cc'] 这样拆分不满足题目要求，
因为其中的 'a' 和 'b' 都出现了不止一次。
示例 2：输入：s = "aba" 输出：2
解释：一种最大拆分方法为 ['a', 'ba'] 。
示例 3：输入：s = "aa" 输出：1
解释：无法进一步拆分字符串。
提示：1 <= s.length <= 16
    s 仅包含小写英文字母。
```

- 解题思路

```
var res int

func maxUniqueSplit(s string) int {
        res = 1
        dfs(s, make(map[string]bool), make([]string, 0))
        return res
}

func dfs(s string, m map[string]bool, arr []string) {
        if len(s) == 0 {
                if len(arr) > res {
                        res = len(arr)
                }
```

(续下页)

(接上页)

```
                return
        }
        for i := 0; i < len(s); i++ {
                newStr := s[:i+1]
                if m[newStr] == false {
                        m[newStr] = true
                        dfs(s[i+1:], m, append(arr, newStr))
                        m[newStr] = false
                }
        }
}
```

47.35 1594. 矩阵的最大非负积 (2)

- 题目

```
给你一个大小为 rows x cols 的矩阵 grid 。最初，你位于左上角 (0, 0) ，
每一步，你可以在矩阵中 向右 或 向下 移动。
在从左上角 (0, 0) 开始到右下角 (rows - 1, cols - 1) 结束的所有路径中，
找出具有 最大非负积 的路径。路径的积是沿路径访问的单元格中所有整数的乘积。
返回 最大非负积 对 109 + 7 取余 的结果。如果最大积为负数，则返回 -1 。
注意，取余是在得到最大积之后执行的。
示例 1：输入：grid = [[-1,-2,-3],
              [-2,-3,-3],
              [-3,-3,-2]]
输出：-1
解释：从 (0, 0) 到 (2, 2) 的路径中无法得到非负积，所以返回 -1
示例 2：输入：grid = [[1,-2,1],
              [1,-2,1],
              [3,-4,1]]
输出：8
解释：最大非负积对应的路径已经用粗体标出 (1 * 1 * -2 * -4 * 1 = 8)
示例 3：输入：grid = [[1, 3],
              [0,-4]]
输出：0
解释：最大非负积对应的路径已经用粗体标出 (1 * 0 * -4 = 0)
示例 4：输入：grid = [[ 1, 4,4,0],
              [-2, 0,0,1],
              [ 1,-1,1,1]]
输出：2
解释：最大非负积对应的路径已经用粗体标出 (1 * -2 * 1 * -1 * 1 * 1 = 2)
提示：1 <= rows, cols <= 15
```

(续下页)

(接上页)

```
    -4 <= grid[i][j] <= 4
```

- 解题思路

```
func maxProductPath(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        dp := make([][][2]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([][2]int, m)
        }
        dp[0][0][0] = grid[0][0] // 负数
        dp[0][0][1] = grid[0][0] // 正数
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i == 0 && j != 0 {
                                dp[i][j][0] = min(dp[i][j-1][0]*grid[i][j], dp[i][j-
↪1][1]*grid[i][j])
                                dp[i][j][1] = max(dp[i][j-1][0]*grid[i][j], dp[i][j-
↪1][1]*grid[i][j])
                        } else if i != 0 && j == 0 {
                                dp[i][j][0] = min(dp[i-1][j][0]*grid[i][j], dp[i-
↪1][j][1]*grid[i][j])
                                dp[i][j][1] = max(dp[i-1][j][0]*grid[i][j], dp[i-
↪1][j][1]*grid[i][j])
                        } else if i != 0 && j != 0 {
                                if grid[i][j] > 0 {
                                        dp[i][j][0] = min(min(dp[i-1][j][0], dp[i][j-
↪1][0]), min(dp[i-1][j][1], dp[i][j-1][1])) * grid[i][j]
                                        dp[i][j][1] = max(max(dp[i-1][j][0], dp[i][j-
↪1][0]), max(dp[i-1][j][1], dp[i][j-1][1])) * grid[i][j]
                                } else {
                                        dp[i][j][0] = max(max(dp[i-1][j][0], dp[i][j-
↪1][0]), max(dp[i-1][j][1], dp[i][j-1][1])) * grid[i][j]
                                        dp[i][j][1] = min(min(dp[i-1][j][0], dp[i][j-
↪1][0]), min(dp[i-1][j][1], dp[i][j-1][1])) * grid[i][j]
                                }
                        }
                }
        }
        a, b := dp[n-1][m-1][0], dp[n-1][m-1][1]
```

(续下页)

(接上页)

```
        if a == 0 || b == 0 {
                return max(0, max(a, b))
        }
        if a > 0 && b > 0 {
                return max(a, b) % 1000000007
        }
        if b > 0 {
                return b % 1000000007
        }
        if a > 0 {
                return a % 1000000007
        }
        return -1
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var res int

func maxProductPath(grid [][]int) int {
        if len(grid) == 0 {
                return 0
        }
        res = -1
        dfs(grid, 0, 0, 1)
        if res < 0 {
                return -1
        }
        return res % 1000000007
}
```

(续下页)

(接上页)

```
func dfs(grid [][]int, i, j int, value int) {
        value = value * grid[i][j]
        if grid[i][j] == 0 {
                if value > res {
                        res = value
                }
                return
        }
        if i == len(grid)-1 && j == len(grid[0])-1 {
                if value > res {
                        res = value
                }
        }
        if i < len(grid)-1 {
                dfs(grid, i+1, j, value)
        }
        if j < len(grid[0])-1 {
                dfs(grid, i, j+1, value)
        }
}
```

47.36 1599. 经营摩天轮的最大利润 (2)

- 题目

```
你正在经营一座摩天轮，该摩天轮共有 4 个座舱 ，每个座舱 最多可以容纳 4 位游客 。
你可以 逆时针 轮转座舱，但每次轮转都需要支付一定的运行成本 runningCost 。
摩天轮每次轮转都恰好转动 1 / 4 周。
给你一个长度为 n 的数组 customers ，
customers[i] 是在第 i 次轮转（下标从 0 开始）之前到达的新游客的数量。
这也意味着你必须在新游客到来前轮转 i 次。
每位游客在登上离地面最近的座舱前都会支付登舱成本 boardingCost␣
↪，一旦该座舱再次抵达地面，
他们就会离开座舱结束游玩。
你可以随时停下摩天轮，即便是 在服务所有游客之前 。
如果你决定停止运营摩天轮，为了保证所有游客安全着陆，将免费进行所有后续轮转 。
注意，如果有超过 4 位游客在等摩天轮，那么只有 4 位游客可以登上摩天轮，其余的需要等待␣
↪下一次轮转 。
返回最大化利润所需执行的 最小轮转次数 。 如果不存在利润为正的方案，则返回 -1 。
示例 1：输入：customers = [8,3], boardingCost = 5, runningCost = 6 输出：3
解释：座舱上标注的数字是该座舱的当前游客数。
```

(续下页)

(接上页)

```
1. 8 位游客抵达，4 位登舱，4 位等待下一舱，摩天轮轮转。当前利润为 4 * $5 - 1 * $6 =
↪$14 。
2. 3 位游客抵达，4 位在等待的游客登舱，其他 3 位等待，摩天轮轮转。当前利润为 8 * $5 -␣
↪2 * $6 = $28 。
3. 最后 3 位游客登舱，摩天轮轮转。当前利润为 11 * $5 - 3 * $6 = $37 。
轮转 3 次得到最大利润，最大利润为 $37 。
示例 2：输入：customers = [10,9,6], boardingCost = 6, runningCost = 4 输出：7
解释：1. 10 位游客抵达，4 位登舱，6 位等待下一舱，摩天轮轮转。当前利润为 4 * $6 - 1 *
↪$4 = $20 。
2. 9 位游客抵达，4 位登舱，11 位等待（2 位是先前就在等待的，9␣
↪位新加入等待的），摩天轮轮转。
当前利润为 8 * $6 - 2 * $4 = $40 。
3. 最后 6 位游客抵达，4 位登舱，13 位等待，摩天轮轮转。当前利润为 12 * $6 - 3 * $4 =
↪$60 。
4. 4 位登舱，9 位等待，摩天轮轮转。当前利润为 * $6 - 4 * $4 = $80 。
5. 4 位登舱，5 位等待，摩天轮轮转。当前利润为 20 * $6 - 5 * $4 = $100 。
6. 4 位登舱，1 位等待，摩天轮轮转。当前利润为 24 * $6 - 6 * $4 = $120 。
7. 1 位登舱，摩天轮轮转。当前利润为 25 * $6 - 7 * $4 = $122 。
轮转 7 次得到最大利润，最大利润为$122 。
示例 3：输入：customers = [3,4,0,5,1], boardingCost = 1, runningCost = 92 输出：-1
解释：1. 3 位游客抵达，3 位登舱，0 位等待，摩天轮轮转。当前利润为 3 * $1 - 1 * $92 = -
↪$89 。
2. 4 位游客抵达，4 位登舱，0 位等待，摩天轮轮转。当前利润为 is 7 * $1 - 2 * $92 = -
↪$177 。
3. 0 位游客抵达，0 位登舱，0 位等待，摩天轮轮转。当前利润为 7 * $1 - 3 * $92 = -$269␣
↪。
4. 5 位游客抵达，4 位登舱，1 位等待，摩天轮轮转。当前利润为 12 * $1 - 4 * $92 = -$356␣
↪。
5. 1 位游客抵达，2 位登舱，0 位等待，摩天轮轮转。当前利润为 13 * $1 - 5 * $92 = -$447␣
↪。
利润永不为正，所以返回 -1 。
示例 4：输入：customers = [10,10,6,4,7], boardingCost = 3, runningCost = 8 输出：9
解释：1. 10 位游客抵达，4 位登舱，6 位等待，摩天轮轮转。当前利润为 4 * $3 - 1 * $8 =
↪$4 。
2. 10 位游客抵达，4 位登舱，12 位等待，摩天轮轮转。当前利润为 8 * $3 - 2 * $8 = $8 。
3. 6 位游客抵达，4 位登舱，14 位等待，摩天轮轮转。当前利润为 12 * $3 - 3 * $8 = $12 。
4. 4 位游客抵达，4 位登舱，14 位等待，摩天轮轮转。当前利润为 16 * $3 - 4 * $8 = $16 。
5. 7 位游客抵达，4 位登舱，17 位等待，摩天轮轮转。当前利润为 20 * $3 - 5 * $8 = $20 。
6. 4 位登舱，13 位等待，摩天轮轮转。当前利润为 24 * $3 - 6 * $8 = $24 。
7. 4 位登舱，9 位等待，摩天轮轮转。当前利润为 28 * $3 - 7 * $8 = $28 。
8. 4 位登舱，5 位等待，摩天轮轮转。当前利润为 32 * $3 - 8 * $8 = $32 。
9. 4 位登舱，1 位等待，摩天轮轮转。当前利润为 36 * $3 - 9 * $8 = $36 。
10. 1 位登舱，0 位等待，摩天轮轮转。当前利润为 37 * $3 - 10 * $8 = $31 。
```

(续下页)

(接上页)

```
轮转 9 次得到最大利润，最大利润为 $36 。
提示：n == customers.length
    1 <= n <= 105
    0 <= customers[i] <= 50
    1 <= boardingCost, runningCost <= 100
```

- 解题思路

```
func minOperationsMaxProfit(customers []int, boardingCost int, runningCost int) int {
        n := len(customers)
        arr := make([]int, 0)
        total := 0
        for i := 0; i < len(customers)-1; i++ {
                total = total + customers[i]
                if total > 4 {
                        arr = append(arr, 4)
                        total = total - 4
                        customers[i+1] = customers[i+1] + total
                        total = 0
                } else {
                        arr = append(arr, total)
                        total = 0
                }
        }
        if customers[n-1] > 0 {
                for customers[n-1] > 4 {
                        arr = append(arr, 4)
                        customers[n-1] = customers[n-1] - 4
                }
                arr = append(arr, customers[n-1])
        }
        maxValue := 0
        res := -1
        total = 0
        for i := 0; i < len(arr); i++ {
                total = total + arr[i]
                profit := total*boardingCost - (i+1)*runningCost
                if profit > 0 {
                        if profit > maxValue {
                                maxValue = profit
                                res = i + 1
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func minOperationsMaxProfit(customers []int, boardingCost int, runningCost int) int {
        maxValue := 0
        res := -1
        total := 0
        i := 0
        profit := 0
        for total > 0 || i < len(customers) {
                if i < len(customers) {
                        total = total + customers[i]
                }
                count := min(total, 4)
                total = total - count
                profit = profit + count*boardingCost - runningCost
                if profit > maxValue {
                        maxValue = profit
                        res = i + 1
                }
                i++
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

47.37 1600. 皇位继承顺序 (2)

- 题目

```
一个王国里住着国王、他的孩子们、他的孙子们等等。每一个时间点，这个家庭里有人出生也有人死亡。
这个王国有一个明确规定的皇位继承顺序，第一继承人总是国王自己。
我们定义递归函数 Successor(x, curOrder) ，给定一个人 x 和当前的继承顺序，该函数返回 x␣
↪的下一继承人。
Successor(x, curOrder):
```

(续下页)

(接上页)

```
    如果 x 没有孩子或者所有 x 的孩子都在 curOrder 中：
        如果 x 是国王，那么返回 null
        否则，返回 Successor(x 的父亲, curOrder)
    否则，返回 x 不在 curOrder 中最年长的孩子
比方说，假设王国由国王，他的孩子 Alice 和 Bob （Alice 比 Bob 年长）和 Alice 的孩子Jack 组成。
    一开始， curOrder 为 ["king"].
    调用 Successor(king, curOrder) ，返回 Alice ，
    所以我们将 Alice 放入 curOrder 中，得到 ["king", "Alice"] 。
    调用 Successor(Alice, curOrder) ，返回 Jack ，
    所以我们将 Jack 放入 curOrder 中，得到 ["king", "Alice", "Jack"] 。
    调用 Successor(Jack, curOrder) ，返回 Bob ，
    所以我们将 Bob 放入 curOrder 中，得到 ["king", "Alice", "Jack", "Bob"] 。
    调用 Successor(Bob, curOrder) ，返回 null 。
    最终得到继承顺序为 ["king", "Alice", "Jack", "Bob"] 。
通过以上的函数，我们总是能得到一个唯一的继承顺序。
请你实现 ThroneInheritance 类：
    ThroneInheritance(string kingName) 初始化一个 ThroneInheritance 类的对象。
    国王的名字作为构造函数的参数传入。
    void birth(string parentName,
    string childName) 表示 parentName 新拥有了一个名为 childName 的孩子。
    void death(string name) 表示名为 name 的人死亡。一个人的死亡不会影响 Successor函数，
    也不会影响当前的继承顺序。你可以只将这个人标记为死亡状态。
    string[] getInheritanceOrder() 返回 除去 死亡人员的当前继承顺序列表。
示例：输入：
["ThroneInheritance", "birth", "birth", "birth", "birth", "birth", "birth",
"getInheritanceOrder", "death", "getInheritanceOrder"]
[["king"], ["king", "andy"], ["king", "bob"],
["king", "catherine"], ["andy", "matthew"], ["bob", "alex"], ["bob", "asha"],
[null], ["bob"], [null]]
输出：[null, null, null, null, null, null, null, ["king", "andy", "matthew", "bob",
"alex", "asha", "catherine"], null, ["king", "andy", "matthew", "alex", "asha",
"catherine"]]
解释：
ThroneInheritance t= new ThroneInheritance("king"); // 继承顺序：king
t.birth("king", "andy"); // 继承顺序：king > andy
t.birth("king", "bob"); // 继承顺序：king > andy > bob
t.birth("king", "catherine"); // 继承顺序：king > andy > bob > catherine
t.birth("andy", "matthew"); // 继承顺序：king > andy > matthew > bob > catherine
t.birth("bob", "alex"); // 继承顺序：king > andy > matthew > bob > alex > catherine
t.birth("bob", "asha"); // 继承顺序：king > andy > matthew > bob > alex > asha >catherine
```

(续下页)

(接上页)

```
t.getInheritanceOrder(); // 返回 ["king", "andy", "matthew", "bob", "alex",
"asha", "catherine"]
t.death("bob"); // 继承顺序：king > andy > matthew > bob（已经去世）> alex >
asha > catherine
t.getInheritanceOrder();
// 返回 ["king", "andy", "matthew", "alex", "asha", "catherine"]
提示：
    1 <= kingName.length, parentName.length, childName.length, name.length <= 15
    kingName, parentName, childName 和 name 仅包含小写英文字母。
    所有的参数 childName 和 kingName 互不相同。
    所有 death 函数中的死亡名字 name 要么是国王，要么是已经出生了的人员名字。
    每次调用 birth(parentName, childName) 时，测试用例都保证 parentName␣
↪对应的人员是活着的。
    最多调用 105 次birth 和 death 。
    最多调用 10 次 getInheritanceOrder 。
```

- 解题思路

```
type Node struct {
        Name  string
        Child []*Node
}

type ThroneInheritance struct {
        isDead map[string]bool
        m      map[string]*Node
        king   *Node
}

func Constructor(kingName string) ThroneInheritance {
        node := &Node{
                Name:  kingName,
                Child: make([]*Node, 0),
        }
        res := ThroneInheritance{
                king:   node,
                m:      map[string]*Node{},
                isDead: map[string]bool{},
        }
        res.m[kingName] = node
        return res
}

func (this *ThroneInheritance) Birth(parentName string, childName string) {
```

(续下页)

(接上页)

```
        node := this.m[parentName]
        child := &Node{
                Name:  childName,
                Child: make([]*Node, 0),
        }
        this.m[childName] = child
        node.Child = append(node.Child, child)
}

func (this *ThroneInheritance) Death(name string) {
        this.isDead[name] = true
}

func (this *ThroneInheritance) GetInheritanceOrder() []string {
        res := make([]string, 0)
        root := this.king
        stack := make([]*Node, 0)
        stack = append(stack, root)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if this.isDead[node.Name] == false {
                        res = append(res, node.Name)
                }
                if len(node.Child) > 0 {
                        for i := len(node.Child) - 1; i >= 0; i-- {
                                stack = append(stack, node.Child[i])
                        }
                }
        }
        return res
}

# 2
type ThroneInheritance struct {
        isDead   map[string]bool
        children map[string][]string
        king     string
}

func Constructor(kingName string) ThroneInheritance {
        return ThroneInheritance{
                isDead:   make(map[string]bool),
```

(续下页)

(接上页)

```
                children: make(map[string][]string),
                king:     kingName,
        }
}

func (this *ThroneInheritance) Birth(parentName string, childName string) {
        this.children[parentName] = append(this.children[parentName], childName)
}

func (this *ThroneInheritance) Death(name string) {
        this.isDead[name] = true
}

func (this *ThroneInheritance) GetInheritanceOrder() []string {
        return dfs(this, this.king)
}

func dfs(this *ThroneInheritance, name string) []string {
        res := make([]string, 0)
        if this.isDead[name] == false {
                res = append(res, name)
        }
        for _, child := range this.children[name] {
                res = append(res, dfs(this, child)...)
        }
        return res
}
```

CHAPTER 48

1501-1600-Hard

48.1 1510. 石子游戏 IV(1)

- 题目

```
Alice 和 Bob 两个人轮流玩一个游戏，Alice 先手。
一开始，有 n 个石子堆在一起。每个人轮流操作，正在操作的玩家可以从石子堆里拿走 任意非零 平方数 个石子。
如果石子堆里没有石子了，则无法操作的玩家输掉游戏。
给你正整数 n ，且已知两个人都采取最优策略。如果 Alice 会赢得比赛，那么返回 True，否则返回 False 。
示例 1：输入：n = 1 输出：true
解释：Alice 拿走 1 个石子并赢得胜利，因为 Bob 无法进行任何操作。
示例 2：输入：n = 2 输出：false
解释：Alice 只能拿走 1 个石子，然后 Bob 拿走最后一个石子并赢得胜利（2 -> 1 -> 0）。
示例 3：输入：n = 4 输出：true
解释：n 已经是一个平方数，Alice 可以一次全拿掉 4 个石子并赢得胜利（4 -> 0）。
示例 4：输入：n = 7 输出：false
解释：当 Bob 采取最优策略时，Alice 无法赢得比赛。
如果 Alice 一开始拿走 4 个石子， Bob 会拿走 1 个石子，然后 Alice 只能拿走 1 个石子，
Bob 拿走最后一个石子并赢得胜利（7 -> 3 -> 2 -> 1 -> 0）。
如果 Alice 一开始拿走 1 个石子， Bob 会拿走 4 个石子，然后 Alice 只能拿走 1 个石子，
Bob 拿走最后一个石子并赢得胜利（7 -> 6 -> 2 -> 1 -> 0）。
示例 5：输入：n = 17 输出：false
解释：如果 Bob 采取最优策略，Alice 无法赢得胜利。
```

(续下页)

(接上页)

```
提示：  1 <= n <= 10^5
```

- 解题思路

```
func winnerSquareGame(n int) bool {
        dp := make([]bool, n+1)
        count := 1
        for i := 1; i <= n; i++ {
                if count*count == i {
                        dp[i] = true
                        count++
                        continue
                }
                for j := 1; j*j < i; j++ {
                        if dp[i-j*j] == false {
                                dp[i] = true
                                break
                        }
                }
        }
        return dp[n]
}
```

48.2 1526. 形成目标数组的子数组最少增加次数 (1)

- 题目

```
给你一个整数数组 target 和一个数组 initial ，initial 数组与 target  数组有同样的维度，
且一开始全部为 0 。
请你返回从 initial 得到  target 的最少操作次数，每次操作需遵循以下规则：
    在 initial 中选择 任意 子数组，并将子数组中每个元素增加 1 。
答案保证在 32 位有符号整数以内。
示例 1：输入：target = [1,2,3,2,1] 输出：3
解释：我们需要至少 3 次操作从 intial 数组得到 target 数组。
[0,0,0,0,0] 将下标为 0 到 4 的元素（包含二者）加 1 。
[1,1,1,1,1] 将下标为 1 到 3 的元素（包含二者）加 1 。
[1,2,2,2,1] 将下表为 2 的元素增加 1 。
[1,2,3,2,1] 得到了目标数组。
示例 2：输入：target = [3,1,1,2] 输出：4
解释：(initial)[0,0,0,0] -> [1,1,1,1] -> [1,1,1,2] -> [2,1,1,2] -> [3,1,1,2] (target)␣
↪。
示例 3：输入：target = [3,1,5,4,2] 输出：7
```

(续下页)

(接上页)

```
解释：(initial)[0,0,0,0,0] -> [1,1,1,1,1] -> [2,1,1,1,1] -> [3,1,1,1,1]
-> [3,1,2,2,2] -> [3,1,3,3,2] -> [3,1,4,4,2] -> [3,1,5,4,2] (target)。
示例 4：输入：target = [1,1,1,1] 输出：1
提示：
    1 <= target.length <= 10^5
    1 <= target[i] <= 10^5
```

- 解题思路

```
func minNumberOperations(target []int) int {
        res := target[0]
        for i := 1; i < len(target); i++ {
                res = res + max(target[i]-target[i-1], 0)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

48.3 1537. 最大得分 (1)

- 题目

```
你有两个 有序且数组内元素互不相同的数组nums1 和nums2。
一条合法路径定义如下：
选择数组 nums1 或者 nums2 开始遍历（从下标 0 处开始）。
从左到右遍历当前数组。
如果你遇到了 nums1和 nums2中都存在的值，
那么你可以切换路径到另一个数组对应数字处继续遍历（但在合法路径中重复数字只会被统计一次）。
得分定义为合法路径中不同数字的和。
请你返回所有可能合法路径中的最大得分。
由于答案可能很大，请你将它对 10^9 + 7 取余后返回。
示例 1：输入：nums1 = [2,4,5,8,10], nums2 = [4,6,8,9] 输出：30
解释：合法路径包括：
[2,4,5,8,10], [2,4,5,8,9], [2,4,6,8,9], [2,4,6,8,10], (从 nums1 开始遍历)
[4,6,8,9], [4,5,8,10], [4,5,8,9], [4,6,8,10]  (从 nums2 开始遍历)
最大得分为上图中的绿色路径 [2,4,6,8,10]。
```

(续下页)

(接上页)

```
示例 2: 输入: nums1 = [1,3,5,7,9], nums2 = [3,5,100] 输出: 109
解释: 最大得分由路径 [1,3,5,100] 得到。
示例 3: 输入: nums1 = [1,2,3,4,5], nums2 = [6,7,8,9,10] 输出: 40
解释: nums1 和 nums2 之间无相同数字。
最大得分由路径 [6,7,8,9,10] 得到。
示例 4: 输入: nums1 = [1,4,5,8,9,11,19], nums2 = [2,3,4,11,12] 输出: 61
提示: 1 <= nums1.length <= 10^5
1 <= nums2.length <= 10^5
1 <= nums1[i], nums2[i] <= 10^7
nums1 和nums2都是严格递增的数组。
```

- 解题思路

```
func maxSum(nums1 []int, nums2 []int) int {
        n := len(nums1)
        m := len(nums2)
        a, b := 0, 0
        i, j := 0, 0
        for i < n || j < m {
                if i < n && j < m {
                        if nums1[i] < nums2[j] {
                                a = a + nums1[i]
                                i++
                        } else if nums1[i] > nums2[j] {
                                b = b + nums2[j]
                                j++
                        } else {
                                temp := max(a, b) + nums1[i] // 遇到相同值，取较大值
                                a = temp
                                b = temp
                                i++
                                j++
                        }
                } else if i < n {
                        a = a + nums1[i]
                        i++
                } else if j < m {
                        b = b + nums2[j]
                        j++
                }
        }
        return max(a, b) % 1000000007
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

48.4 1542. 找出最长的超赞子字符串 (1)

- 题目

```
给你一个字符串 s 。请返回 s 中最长的 超赞子字符串 的长度。
「超赞子字符串」需满足满足下述两个条件：
该字符串是 s 的一个非空子字符串
进行任意次数的字符交换后，该字符串可以变成一个回文字符串
示例 1：输入：s = "3242415" 输出：5
解释："24241" 是最长的超赞子字符串，交换其中的字符后，可以得到回文 "24142"
示例 2：输入：s = "12345678" 输出：1
示例 3：输入：s = "213123" 输出：6
解释："213123" 是最长的超赞子字符串，交换其中的字符后，可以得到回文 "231132"
示例 4：输入：s = "00" 输出：2
提示：1 <= s.length <= 10^5
s 仅由数字组成
```

- 解题思路

```
func longestAwesome(s string) int {
        res := 0
        n := len(s)
        m := make(map[int]int) // 保存每个状态第1次出现的下标
        m[0] = -1              // 0对应的下标
        cur := 0
        for i := 0; i < n; i++ {
                value := int(s[i] - '0')
                cur = cur ^ (1 << value)
                if index, ok := m[cur]; ok { // 相同的情况
                        res = max(res, i-index)
                } else {
                        m[cur] = i
                }
                for j := 0; j < 10; j++ { // 相差1位的情况
                        if index, ok := m[cur^(1<<j)]; ok {
```

(续下页)

(接上页)

```
                                res = max(res, i-index)
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

48.5 1547. 切棍子的最小成本 (3)

- 题目

```
有一根长度为 n 个单位的木棍，棍上从 0 到 n 标记了若干位置。例如，长度为 6␣
↪的棍子可以标记如下：
给你一个整数数组 cuts ，其中 cuts[i] 表示你需要将棍子切开的位置。
你可以按顺序完成切割，也可以根据需要更改切割的顺序。
每次切割的成本都是当前要切割的棍子的长度，切棍子的总成本是历次切割成本的总和。
对棍子进行切割将会把一根木棍分成两根较小的木棍（这两根木棍的长度和就是切割前木棍的长度）。
请参阅第一个示例以获得更直观的解释。
返回切棍子的 最小总成本 。
示例 1：输入：n = 7, cuts = [1,3,4,5] 输出：16
解释：按 [1, 3, 4, 5] 的顺序切割的情况如下所示：
第一次切割长度为 7 的棍子，成本为 7 。
第二次切割长度为 6 的棍子（即第一次切割得到的第二根棍子），第三次切割为长度 4 的棍子，
最后切割长度为 3 的棍子。总成本为 7 + 6 + 4 + 3 = 20 。
而将切割顺序重新排列为 [3, 5, 1, 4] 后，总成本 = 16（如示例图中 7 + 4 + 3 + 2 = 16）。
示例 2：输入：n = 9, cuts = [5,6,1,4,2] 输出：22
解释：如果按给定的顺序切割，则总成本为 25 。
总成本 <= 25 的切割顺序很多，例如，[4, 6, 5, 2, 1] 的总成本 =␣
↪22，是所有可能方案中成本最小的。
提示：2 <= n <= 10^6
1 <= cuts.length <= min(n - 1, 100)
1 <= cuts[i] <= n - 1
cuts 数组中的所有整数都 互不相同
```

- 解题思路

```
func minCost(n int, cuts []int) int {
        m := len(cuts)
        cuts = append(cuts, 0, n)
        sort.Ints(cuts)
        dp := make([][]int, m+2) // dp[L][R]为切割以L,R为左右端点的最小成本
        for i := 0; i < m+2; i++ {
                dp[i] = make([]int, m+2)
        }
        for i := m; i >= 1; i-- {
                for j := i; j <= m; j++ {
                        dp[i][j] = math.MaxInt32
                        for k := i; k <= j; k++ { // 枚举切割点
                                dp[i][j] = min(dp[i][j], dp[i][k-1]+dp[k+1][j])
                        }
                        dp[i][j] = dp[i][j] + cuts[j+1] - cuts[i-1]
                }
        }
        return dp[1][m]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minCost(n int, cuts []int) int {
        m := len(cuts)
        cuts = append(cuts, 0, n)
        sort.Ints(cuts)
        dp := make([][]int, m+2) // dp[L][R]为切割以L,R为左右端点的最小成本
        for i := 0; i < m+2; i++ {
                dp[i] = make([]int, m+2)
        }
        for length := 2; length <= m+1; length++ { // 枚举长度
                for i := 1; i <= m; i++ {
                        j := i + length - 2
                        if j > m {
                                break
                        }
                        dp[i][j] = math.MaxInt32
                        for k := i; k <= j; k++ { // 枚举切割点
```

(续下页)

(接上页)

```
                                dp[i][j] = min(dp[i][j], dp[i][k-1]+dp[k+1][j])
                        }
                        dp[i][j] = dp[i][j] + cuts[j+1] - cuts[i-1]
                }
        }
        return dp[1][m]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var dp [][]int

func minCost(n int, cuts []int) int {
        m := len(cuts)
        cuts = append(cuts, 0, n)
        sort.Ints(cuts)
        dp = make([][]int, m+2) // dp[L][R]为切割以L,R为左右端点的最小成本
        for i := 0; i < m+2; i++ {
                dp[i] = make([]int, m+2)
        }
        return dfs(cuts, 1, m)
}

func dfs(cuts []int, i, j int) int {
        if dp[i][j] != 0 {
                return dp[i][j]
        }
        if i > j {
                return 0
        }
        if i == j {
                return cuts[j+1] - cuts[i-1]
        }
        dp[i][j] = math.MaxInt32
        for k := i; k <= j; k++ {
                dp[i][j] = min(dp[i][j], dfs(cuts, i, k-1)+dfs(cuts, k+1, j))
        }
```

(续下页)

(接上页)

```
        dp[i][j] = dp[i][j] + cuts[j+1] - cuts[i-1]
        return dp[i][j]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

48.6 1553. 吃掉 N 个橘子的最少天数 (2)

- 题目

```
厨房里总共有 n 个橘子，你决定每一天选择如下方式之一吃这些橘子：
    吃掉一个橘子。
    如果剩余橘子数 n 能被 2 整除，那么你可以吃掉 n/2 个橘子。
    如果剩余橘子数 n 能被 3 整除，那么你可以吃掉 2*(n/3) 个橘子。
每天你只能从以上 3 种方案中选择一种方案。
请你返回吃掉所有 n 个橘子的最少天数。
示例 1：输入：n = 10 输出：4
解释：你总共有 10 个橘子。
第 1 天：吃 1 个橘子，剩余橘子数 10 - 1 = 9。
第 2 天：吃 6 个橘子，剩余橘子数 9 - 2*(9/3) = 9 - 6 = 3。（9 可以被 3 整除）
第 3 天：吃 2 个橘子，剩余橘子数 3 - 2*(3/3) = 3 - 2 = 1。
第 4 天：吃掉最后 1 个橘子，剩余橘子数 1 - 1 = 0。
你需要至少 4 天吃掉 10 个橘子。
示例 2：输入：n = 6 输出：3
解释：你总共有 6 个橘子。
第 1 天：吃 3 个橘子，剩余橘子数 6 - 6/2 = 6 - 3 = 3。（6 可以被 2 整除）
第 2 天：吃 2 个橘子，剩余橘子数 3 - 2*(3/3) = 3 - 2 = 1。（3 可以被 3 整除）
第 3 天：吃掉剩余 1 个橘子，剩余橘子数 1 - 1 = 0。
你至少需要 3 天吃掉 6 个橘子。
示例 3：输入：n = 1 输出：1
示例 4：输入：n = 56 输出：6
提示：  1 <= n <= 2*10^9
```

- 解题思路

```
var dp map[int]int
```

(续下页)

(接上页)

```
func minDays(n int) int {
        dp = make(map[int]int)
        dp[0] = 0
        dp[1] = 1
        return dfs(n)
}

func dfs(n int) int {
        if value, ok := dp[n]; ok {
                return value
        }
        // 吃n/2的情况，先吃掉n%2，然后就剩下dfs(n/2)+1
        // 吃n/3的情况，先吃点n%3，然后就剩下dfs(n/3)+1
        dp[n] = min(dfs(n/2)+n%2+1, dfs(n/3)+n%3+1)
        return dp[n]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minDays(n int) int {
        m := make(map[int]bool)
        queue := make([]int, 0)
        queue = append(queue, n)
        res := 0
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        day := queue[i]
                        if day == 0 {
                                return res
                        }
                        if day%3 == 0 && m[day/3] == false {
                                queue = append(queue, day/3)
                                m[day/3] = true
                        }
                        if day%2 == 0 && m[day/2] == false {
                                queue = append(queue, day/2)
```

(续下页)

(接上页)

```
                                m[day/2] = true
                        }
                        if m[day-1] == false {
                                queue = append(queue, day-1)
                                m[day-1] = true
                        }
                }
                res++
                queue = queue[length:]
        }
        return res
}
```

48.7 1559. 二维网格图中探测环 (1)

- 题目

```
给你一个二维字符网格数组 grid ，大小为 m x n ，你需要检查 grid 中是否存在 相同值形成的环。
一个环是一条开始和结束于同一个格子的长度 大于等于 4 的路径。对于一个给定的格子，
你可以移动到它上、下、左、右四个方向相邻的格子之一，可以移动的前提是这两个格子有相同的值 。
同时，你也不能回到上一次移动时所在的格子。比方说，环 (1, 1) -> (1, 2) -> (1, 1)是不合法的，
因为从 (1, 2) 移动到 (1, 1) 回到了上一次移动时的格子。
如果 grid 中有相同值形成的环，请你返回 true ，否则返回 false 。
示例 1:
输入：grid = [["a","a","a","a"],["a","b","b","a"],["a","b","b","a"],["a","a","a","a"]]
输出：true
解释：如下图所示，有 2 个用不同颜色标出来的环:
示例 2:
输入：grid = [["c","c","c","a"],["c","d","c","c"],["c","c","e","c"],["f","c","c","c"]]
输出：true
解释：如下图所示，只有高亮所示的一个合法环:
示例 3: 输入：grid = [["a","b","b"],["b","z","b"],["b","b","a"]] 输出：false
提示： m == grid.length
    n == grid[i].length
    1 <= m <= 500
    1 <= n <= 500
    grid 只包含小写英文字母。
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

var visited map[[2]int]bool

func containsCycle(grid [][]byte) bool {
        n, m := len(grid), len(grid[0])
        visited = make(map[[2]int]bool)
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if visited[[2]int{i, j}] == true {
                                continue
                        }
                        start := grid[i][j]
                        if dfs(grid, start, i, j, -1, -1) == true {
                                return true
                        }
                }
        }
        return false
}

// x,y 当前节点， pX,pY 父节点
func dfs(grid [][]byte, start byte, x, y int, pX, pY int) bool {
        visited[[2]int{x, y}] = true
        for i := 0; i < 4; i++ {
                newX := x + dx[i]
                newY := y + dy[i]
                if newX < 0 || newX >= len(grid) || newY < 0 || newY >= len(grid[0])␣
↪||
                        (newX == pX && newY == pY) {
                        continue
                }
                if start == grid[newX][newY] {
                        if visited[[2]int{newX, newY}] == true {
                                return true
                        }
                        result := dfs(grid, start, newX, newY, x, y)
                        if result == true {
                                return true
                        }
                }
        }
        return false
```

(续下页)

(接上页)

```
}
```

48.8 1563. 石子游戏 V(2)

- 题目

```
几块石子 排成一行 ，每块石子都有一个关联值，关联值为整数，由数组 stoneValue 给出。
游戏中的每一轮：Alice 会将这行石子分成两个 非空行（即，左侧行和右侧行）；
Bob 负责计算每一行的值，即此行中所有石子的值的总和。
Bob 会丢弃值最大的行，Alice 的得分为剩下那行的值（每轮累加）。
如果两行的值相等，Bob 让 Alice 决定丢弃哪一行。下一轮从剩下的那一行开始。
只剩下一块石子 时，游戏结束。Alice 的分数最初为 0 。
返回 Alice 能够获得的最大分数 。
示例 1：输入：stoneValue = [6,2,3,4,5,5] 输出：18
解释：在第一轮中，Alice 将行划分为 [6, 2, 3], [4, 5, 5] 。
左行的值是 11 ，右行的值是 14 。Bob 丢弃了右行，Alice 的分数现在是 11 。
在第二轮中，Alice 将行分成 [6], [2, 3] 。这一次 Bob 扔掉了左行，Alice 的分数变成了␣
↪16（11 + 5）。
最后一轮 Alice 只能将行分成 [2], [3] 。Bob 扔掉右行，Alice 的分数现在是 18（16 + 2）。
游戏结束，因为这行只剩下一块石头了。
示例 2：输入：stoneValue = [7,7,7,7,7,7,7] 输出：28
示例 3：输入：stoneValue = [4] 输出：0
提示：

    1 <= stoneValue.length <= 500
    1 <= stoneValue[i] <= 10^6
```

- 解题思路

```
func stoneGameV(stoneValue []int) int {
        n := len(stoneValue)
        sum := make([]int, n+1)
        sum[0] = stoneValue[0]
        for i := 1; i < n; i++ {
                sum[i] = sum[i-1] + stoneValue[i]
        }
    // dp[i][j]：代表从i到j区间,Alice分数最大值
        dp := make([][]int, n+1)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n+1)
        }
```

(续下页)

(接上页)

```
        // 长度从2开始到N依次枚举
        for length := 2; length <= n; length++ {
                for i := 0; i+length-1 < n; i++ {
                        j := i + length - 1
                        for k := i; k <= j; k++ {
                                if i > k || k+1 > j {
                                        continue
                                }
                                left := dp[i][k]
                                right := dp[k+1][j]
                                leftSum := sum[k]
                                if i > 0 {
                                        leftSum = sum[k] - sum[i-1]
                                }
                                rightSum := sum[j] - sum[k]
                                if leftSum == rightSum {
                                        dp[i][j] = max(dp[i][j], max(left,␣
↪right)+leftSum)
                                } else if leftSum > rightSum {
                                        dp[i][j] = max(dp[i][j], right+rightSum)
                                } else if leftSum < rightSum {
                                        dp[i][j] = max(dp[i][j], left+leftSum)
                                }
                        }
                }
        }
        return dp[0][n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var dp [][]int
var sum []int

func stoneGameV(stoneValue []int) int {
        n := len(stoneValue)
        sum = make([]int, n+1)
```

(续下页)

(接上页)

```
        sum[0] = 0
        for i := 0; i < n; i++ {
                sum[i+1] = sum[i] + stoneValue[i]
        }
        dp = make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, n+1)
                for j := 0; j <= n; j++ {
                        dp[i][j] = -1
                }
        }
        return dfs(1, n)
}

func dfs(left, right int) int {
        if dp[left][right] != -1 {
                return dp[left][right]
        }
        if left == right {
                dp[left][right] = 0
        } else {
                value := 0
                for i := left; i < right; i++ {
                        leftSum := sum[i] - sum[left-1]
                        rightSum := sum[right] - sum[i]
                        if leftSum < rightSum {
                                value = max(value, leftSum+dfs(left, i))
                        } else if leftSum > rightSum {
                                value = max(value, rightSum+dfs(i+1, right))
                        } else {
                                value = max(value, max(dfs(left, i), dfs(i+1,␣
↪right))+leftSum)
                        }
                }
                dp[left][right] = value
        }
        return dp[left][right]
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

48.9 1579. 保证图可完全遍历 (1)

- 题目

```
Alice 和 Bob 共有一个无向图，其中包含 n 个节点和 3 种类型的边：
类型 1：只能由 Alice 遍历。
类型 2：只能由 Bob 遍历。
类型 3：Alice 和 Bob 都可以遍历。
给你一个数组 edges ，其中 edges[i] = [typei, ui, vi]表示节点 ui 和 vi 之间存在类型为␣
↪typei 的双向边。
请你在保证图仍能够被 Alice和 Bob 完全遍历的前提下，找出可以删除的最大边数。
如果从任何节点开始，Alice 和 Bob 都可以到达所有其他节点，则认为图是可以完全遍历的。
返回可以删除的最大边数，如果 Alice 和 Bob 无法完全遍历图，则返回 -1 。
示例 1：输入：n = 4, edges = [[3,1,2],[3,2,3],[1,1,3],[1,2,4],[1,1,2],[2,3,4]] 输出：2
解释：如果删除 [1,1,2] 和 [1,1,3] 这两条边，Alice 和 Bob 仍然可以完全遍历这个图。
再删除任何其他的边都无法保证图可以完全遍历。所以可以删除的最大边数是 2 。
示例 2：输入：n = 4, edges = [[3,1,2],[3,2,3],[1,1,4],[2,1,4]] 输出：0
解释：注意，删除任何一条边都会使 Alice 和 Bob 无法完全遍历这个图。
示例 3：输入：n = 4, edges = [[3,2,3],[1,1,2],[2,3,4]] 输出：-1
解释：在当前图中，Alice 无法从其他节点到达节点 4 。类似地，Bob 也不能达到节点 1␣
↪。因此，图无法完全遍历。
提示：1 <= n <= 10^5
1 <= edges.length <= min(10^5, 3 * n * (n-1) / 2)
edges[i].length == 3
1 <= edges[i][0] <= 3
1 <= edges[i][1] < edges[i][2] <= n
所有元组 (typei, ui, vi) 互不相同
```

- 解题思路

```
func maxNumEdgesToRemove(n int, edges [][]int) int {
        res := len(edges)
        alice, bob := &UnionFind{}, &UnionFind{}
        alice.Init(n)
        bob.Init(n)
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1]-1, edges[i][2]-1
                if a == 3 && (alice.query(b, c) == false || bob.query(b, c) == false)
↪{ // 公共边不在一个集合
```

(续下页)

(接上页)

```
                        alice.union(b, c)
                        bob.union(b, c)
                        res--
                }
        }
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1]-1, edges[i][2]-1
                if a == 1 && alice.query(b, c) == false { // 单边不在一个集合
                        alice.union(b, c)
                        res--
                }
                if a == 2 && bob.query(b, c) == false { // 单边不在一个集合
                        bob.union(b, c)
                        res--
                }
        }
        if alice.count > 1 || bob.count > 1 {
                return -1
        }
        return res
}

type UnionFind struct {
        fa    []int
        count int
}

// 初始化
func (u *UnionFind) Init(n int) {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        u.count = n
        u.fa = arr
}

// 查询
func (u UnionFind) find(x int) int {
        if u.fa[x] == x {
                return x
        }
        // 路径压缩
```

(续下页)

(接上页)

```
        u.fa[x] = u.find(u.fa[x])
        return u.fa[x]
}

// 合并
func (u *UnionFind) union(i, j int) {
        x, y := u.find(i), u.find(j)
        if x != y {
                u.fa[x] = y
                u.count--
        }
}

func (u UnionFind) query(i, j int) bool {
        return u.find(i) == u.find(j)
}
```

48.10 1585. 检查字符串是否可以通过排序子字符串得到另一个字符串 (1)

- 题目

```
给你两个字符串s 和t，请你通过若干次以下操作将字符串s转化成字符串t：
选择 s中一个 非空子字符串并将它包含的字符就地 升序排序。
比方说，对下划线所示的子字符串进行操作可以由"14234"得到"12344"。
如果可以将字符串 s变成 t，返回 true。否则，返回 false。
一个 子字符串定义为一个字符串中连续的若干字符。
示例 1：输入：s = "84532", t = "34852" 输出：true
解释：你可以按以下操作将 s 转变为 t ：
"84532" （从下标 2 到下标 3）-> "84352"
"84352" （从下标 0 到下标 2） -> "34852"
示例 2：输入：s = "34521", t = "23415" 输出：true
解释：你可以按以下操作将 s 转变为 t ：
"34521" -> "23451"
"23451" -> "23415"
示例 3：输入：s = "12345", t = "12435" 输出：false
示例 4：输入：s = "1", t = "2" 输出：false
提示：s.length == t.length
1 <= s.length <= 105
s 和t都只包含数字字符，即'0'到'9' 。
```

- 解题思路

```
func isTransformable(s string, t string) bool {
        n := len(s)
        arr := [10][]int{}
        for i := 0; i < n; i++ {
                arr[int(s[i]-'0')] = append(arr[int(s[i]-'0')], i)
        }
        for i := 0; i < n; i++ {
                index := int(t[i] - '0')
                if len(arr[index]) == 0 {
                        return false
                }
                // 看当前位置s中等于t[i]的第x个元素，能不能移动到前面
                // 一次交换前后2个，目标值往前移（冒泡排序）
                for j := 0; j < index; j++ {
                        // 前面不能存在比当前值小的，因为2个数排序后当前值会在后面
                        if len(arr[j]) > 0 && arr[j][0] < arr[index][0] {
                                return false
                        }
                }
                arr[index] = arr[index][1:] // 当前数字匹配到了，往后移
        }
        return true
}
```

CHAPTER 49

1601-1700-Easy

49.1 1603. 设计停车系统 (2)

- 题目

```
请你给一个停车场设计一个停车系统。停车场总共有三种不同大小的车位：大，中和小，每种尺寸分别有固定数目
请你实现 ParkingSystem 类：
    ParkingSystem(int big, int medium, int small) 初始化 ParkingSystem 类，
    三个参数分别对应每种停车位的数目。
    bool addCar(int carType) 检车是否有 carType 对应的停车位。
    carType 有三种类型：大，中，小，分别用数字 1， 2 和 3 表示。
    一辆车只能停在  carType 对应尺寸的停车位中。
    如果没有空车位，请返回 false ，否则将该车停入车位并返回 true 。
示例 1：输入：["ParkingSystem", "addCar", "addCar", "addCar", "addCar"]
[[1, 1, 0], [1], [2], [3], [1]]
输出：[null, true, true, false, false]
解释：ParkingSystem parkingSystem = new ParkingSystem(1, 1, 0);
parkingSystem.addCar(1); // 返回 true ，因为有 1 个空的大车位
parkingSystem.addCar(2); // 返回 true ，因为有 1 个空的中车位
parkingSystem.addCar(3); // 返回 false ，因为没有空的小车位
parkingSystem.addCar(1); // 返回 false␣
↪，因为没有空的大车位，唯一一个大车位已经被占据了
提示：  0 <= big, medium, small <= 1000
    carType 取值为 1， 2 或 3
    最多会调用 addCar 函数 1000 次
```

- 解题思路

```
type ParkingSystem struct {
        arr   [3]int
        total [3]int
}

func Constructor(big int, medium int, small int) ParkingSystem {
        return ParkingSystem{
                arr:   [3]int{big, medium, small},
                total: [3]int{0, 0, 0},
        }
}

func (this *ParkingSystem) AddCar(carType int) bool {
        index := carType - 1
        if this.total[index] < this.arr[index] {
                this.total[index]++
                return true
        }
        return false
}

# 2
type ParkingSystem struct {
        arr [3]int
}

func Constructor(big int, medium int, small int) ParkingSystem {
        return ParkingSystem{
                arr: [3]int{big, medium, small},
        }
}

func (this *ParkingSystem) AddCar(carType int) bool {
        if this.arr[carType-1] > 0 {
                this.arr[carType-1]--
                return true
        }
        return false
}
```

49.2 1608. 特殊数组的特征值 (3)

- 题目

```
给你一个非负整数数组 nums 。
如果存在一个数 x ，使得 nums 中恰好有 x 个元素 大于或者等于 x ，
那么就称 nums 是一个 特殊数组 ，而 x 是该数组的 特征值 。
注意： x 不必 是 nums 的中的元素。
如果数组 nums 是一个 特殊数组 ，请返回它的特征值 x 。
否则，返回 -1 。可以证明的是，如果 nums 是特殊数组，那么其特征值 x 是 唯一的 。
示例 1：输入：nums = [3,5] 输出：2
解释：有 2 个元素 (3 和 5) 大于或等于 2 。
示例 2：输入：nums = [0,0] 输出：-1
解释：没有满足题目要求的特殊数组，故而也不存在特征值 x 。
如果 x = 0，应该有 0 个元素 >= x，但实际有 2 个。
如果 x = 1，应该有 1 个元素 >= x，但实际有 0 个。
如果 x = 2，应该有 2 个元素 >= x，但实际有 0 个。
x 不能取更大的值，因为 nums 中只有两个元素。
示例 3：输入：nums = [0,4,3,0,4] 输出：3
解释：有 3 个元素大于或等于 3 。
示例 4：输入：nums = [3,6,7,7,0] 输出：-1
提示：1 <= nums.length <= 100
    0 <= nums[i] <= 1000
```

- 解题思路

```
func specialArray(nums []int) int {
        arr := make([]int, 1001)
        for i := 0; i < len(nums); i++ {
                arr[nums[i]]++
        }
        for i := len(arr) - 2; i >= 0; i-- {
                arr[i] = arr[i] + arr[i+1]
        }
        for i := 0; i <= len(nums); i++ {
                if arr[i] == i {
                        return i
                }
        }
        return -1
}

# 2
func specialArray(nums []int) int {
```

(续下页)

(接上页)

```
        for i := 0; i <= len(nums); i++ {
                count := 0
                for j := 0; j < len(nums); j++ {
                        if nums[j] >= i {
                                count++
                        }
                }
                if count == i {
                        return i
                }
        }
        return -1
}

# 3
func specialArray(nums []int) int {
        sort.Ints(nums)
        n := len(nums)
        if nums[0] >= n {
                return n
        }
        for i := 1; i < n; i++ {
                target := n - i
                if nums[i] >= target && target > nums[i-1] {
                        return target
                }
        }
        return -1
}
```

49.3 1614. 括号的最大嵌套深度 (2)

- 题目

```
如果字符串满足一下条件之一，则可以称之为 有效括号字符串 (valid parentheses␣
↪string，可以简写为 VPS)：
    字符串是一个空字符串 ""，或者是一个不为 "(" 或 ")" 的单字符。
    字符串可以写为 AB (A 与 B 字符串连接)，其中 A 和 B 都是 有效括号字符串 。
    字符串可以写为 (A)，其中 A 是一个 有效括号字符串 。
类似地，可以定义任何有效括号字符串 S 的 嵌套深度 depth(S)：
    depth("") = 0
    depth(A + B) = max(depth(A), depth(B))，其中 A 和 B 都是 有效括号字符串
```

(续下页)

(接上页)

```
    depth("(" + A + ")") = 1 + depth(A), 其中 A 是一个 有效括号字符串
例如："""、"()()"、"()(()())" 都是 有效括号字符串（嵌套深度分别为 0、1、2），
而 ")(" 、"(()" 都不是 有效括号字符串 。
给你一个 有效括号字符串 s，返回该字符串的 s 嵌套深度 。
示例 1：输入：s = "(1+(2*3)+((8)/4))+1" 输出：3
解释：数字 8 在嵌套的 3 层括号中。
示例 2：输入：s = "(1)+((2))+(((3)))" 输出：3
示例 3：输入：s = "1+(2*3)/(2-1)" 输出：1
示例 4：输入：s = "1" 输出：0
提示： 1 <= s.length <= 100
    s 由数字 0-9 和字符 '+'、'-'、'*'、'/'、'('、')' 组成
    题目数据保证括号表达式 s 是 有效的括号表达式
```

- 解题思路

```
func maxDepth(s string) int {
        res := 0
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        count++
                } else if s[i] == ')' {
                        if count > res {
                                res = count
                        }
                        count--
                }
        }
        return res
}

# 2
func maxDepth(s string) int {
        res := 0
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        stack = append(stack, s[i])
                } else if s[i] == ')' {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) > res {
                        res = len(stack)
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

49.4 1619. 删除某些元素后的数组均值 (2)

- 题目

```
给你一个整数数组arr，请你删除最小5%的数字和最大 5%的数字后，剩余数字的平均值。
与 标准答案误差在10-5的结果都被视为正确结果。
示例 1：输入：arr = [1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3] 输出：2.00000
解释：删除数组中最大和最小的元素后，所有元素都等于 2，所以平均值为 2 。
示例 2：输入：arr = [6,2,7,5,1,2,0,3,10,2,5,0,5,5,0,8,7,6,8,0] 输出：4.00000
示例 3：输入：arr =[6,0,7,0,7,5,7,8,3,4,0,7,8,1,6,8,1,1,2,4,8,1,9,5,4,
3,8,5,10,8,6,6,1,0,6,10,8,2,3,4]
输出：4.77778
示例 4：输入：arr =[9,7,8,7,7,8,4,4,6,8,8,7,6,8,8,9,2,6,0,0,1,10,8,6,3,3,5,1,10,9,0,
7,10,0,10,4,1,10,6,9,3,6,0,0,2,7,0,6,7,2,9,7,7,3,0,1,6,1,10,3]
输出：5.27778
示例 5：输入：arr = [4,8,4,10,0,7,1,3,7,8,8,3,4,1,6,2,1,1,8,0,9,8,0,3,9,10,3,10,1,10,
↪7,3,2,1,4,
9,10,7,6,4,0,8,5,1,2,1,6,2,5,0,7,10,9,10,3,7,10,5,8,5,7,6,7,6,10,9,5,10,5,5,
7,2,10,7,7,8,2,0,1,1]
输出：5.29167
提示：20 <= arr.length <= 1000
arr.length是20的倍数
0 <= arr[i] <= 105
```

- 解题思路

```
func trimMean(arr []int) float64 {
        temp := make([]float64, 0)
        for i := 0; i < len(arr); i++ {
                temp = append(temp, float64(arr[i]))
        }
        sort.Float64s(temp)
        var sum float64
        count := int(float64(len(arr)) * 0.05)
        for i := count; i < len(temp)-count; i++ {
                sum = sum + temp[i]
        }
        return sum / float64(len(arr)-2*count)
```

(续下页)

(接上页)

```
}

# 2
func trimMean(arr []int) float64 {
        n := len(arr)
        count := n / 20
        sort.Ints(arr)
        sum := 0
        for i := count; i < n-count; i++ {
                sum += arr[i]
        }
        return float64(sum) / float64(n-2*count)
}
```

49.5 1624. 两个相同字符之间的最长子字符串 (2)

- 题目

```
给你一个字符串 s，请你返回 两个相同字符之间的最长子字符串的长度␣
↪，计算长度时不含这两个字符。
如果不存在这样的子字符串，返回 -1 。
子字符串 是字符串中的一个连续字符序列。
示例 1：输入：s = "aa" 输出：0
解释：最优的子字符串是两个 'a' 之间的空子字符串。
示例 2：输入：s = "abca" 输出：2
解释：最优的子字符串是 "bc" 。
示例 3：输入：s = "cbzxy" 输出：-1
解释：s 中不存在出现出现两次的字符，所以返回 -1 。
示例 4：输入：s = "cabbac" 输出：4
解释：最优的子字符串是 "abba" ，其他的非最优解包括 "bb" 和 "" 。
提示：1 <= s.length <= 300  s 只含小写英文字母
```

- 解题思路

```
func maxLengthBetweenEqualCharacters(s string) int {
        m := make(map[byte]int)
        res := -1
        for i := 0; i < len(s); i++ {
                if value, ok := m[s[i]]; ok {
                        res = max(res, i-value-1)
                } else {
                        m[s[i]] = i
```

(续下页)

(接上页)

```
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxLengthBetweenEqualCharacters(s string) int {
        res := -1
        for i := 0; i < len(s); i++ {
                for j := i + 1; j < len(s); j++ {
                        if s[i] == s[j] {
                                res = max(res, j-i-1)
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

49.6 1629. 按键持续时间最长的键 (1)

- 题目

```
LeetCode 设计了一款新式键盘，正在测试其可用性。测试人员将会点击一系列键（总计 n␣
↪个），每次一个。
给你一个长度为 n 的字符串 keysPressed ，其中 keysPressed[i] 表示测试序列中第 i␣
↪个被按下的键。
releaseTimes 是一个升序排列的列表，其中 releaseTimes[i] 表示松开第 i 个键的时间。
字符串和数组的 下标都从 0 开始 。
```

(续下页)

(接上页)

```
第 0 个键在时间为 0 时被按下，接下来每个键都 恰好 在前一个键松开时被按下。
测试人员想要找出按键 持续时间最长 的键。
第 i 次按键的持续时间为 releaseTimes[i] - releaseTimes[i - 1] ，
第 0 次按键的持续时间为 releaseTimes[0] 。
注意，测试期间，同一个键可以在不同时刻被多次按下，而每次的持续时间都可能不同。
请返回按键 持续时间最长 的键，如果有多个这样的键，则返回 按字母顺序排列最大 的那个键。
示例 1：输入：releaseTimes = [9,29,49,50], keysPressed = "cbcd" 输出："c"
解释：按键顺序和持续时间如下：
按下 'c' ，持续时间 9（时间 0 按下，时间 9 松开）
按下 'b' ，持续时间 29 - 9 = 20（松开上一个键的时间 9 按下，时间 29 松开）
按下 'c' ，持续时间 49 - 29 = 20（松开上一个键的时间 29 按下，时间 49 松开）
按下 'd' ，持续时间 50 - 49 = 1（松开上一个键的时间 49 按下，时间 50 松开）
按键持续时间最长的键是 'b' 和 'c'（第二次按下时），持续时间都是 20
'c' 按字母顺序排列比 'b' 大，所以答案是 'c'
示例 2：输入：releaseTimes = [12,23,36,46,62], keysPressed = "spuda" 输出："a"
解释：按键顺序和持续时间如下：
按下 's' ，持续时间 12
按下 'p' ，持续时间 23 - 12 = 11
按下 'u' ，持续时间 36 - 23 = 13
按下 'd' ，持续时间 46 - 36 = 10
按下 'a' ，持续时间 62 - 46 = 16
按键持续时间最长的键是 'a' ，持续时间 16
提示：  releaseTimes.length == n
    keysPressed.length == n
    2 <= n <= 1000
    0 <= releaseTimes[i] <= 109
    releaseTimes[i] < releaseTimes[i+1]
    keysPressed 仅由小写英文字母组成
```

- 解题思路

```
func slowestKey(releaseTimes []int, keysPressed string) byte {
        res := 0
        maxValue := releaseTimes[0]
        for i := 1; i < len(releaseTimes); i++ {
                if releaseTimes[i]-releaseTimes[i-1] > maxValue {
                        maxValue = releaseTimes[i] - releaseTimes[i-1]
                        res = i
                } else if releaseTimes[i]-releaseTimes[i-1] == maxValue &&
                        keysPressed[i] > keysPressed[res] {
                        res = i
                }
        }
        return keysPressed[res]
```

(续下页)

(接上页)

```
}
```

49.7 1636. 按照频率将数组升序排序 (1)

- 题目

```
给你一个整数数组 nums ，请你将数组按照每个值的频率 升序 排序。
如果有多个值的频率相同，请你按照数值本身将它们 降序 排序。
请你返回排序后的数组。
示例 1：输入：nums = [1,1,2,2,2,3] 输出：[3,1,1,2,2,2]
解释：'3' 频率为 1，'1' 频率为 2，'2' 频率为 3 。
示例 2：输入：nums = [2,3,1,3,2] 输出：[1,3,3,2,2]
解释：'2' 和 '3' 频率都为 2 ，所以它们之间按照数值本身降序排序。
示例 3：输入：nums = [-1,1,-6,4,5,-6,1,4,1] 输出：[5,-1,4,4,-6,-6,1,1,1]
提示：1 <= nums.length <= 100
    -100 <= nums[i] <= 100
```

- 解题思路

```
func frequencySort(nums []int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        arr := make([][2]int, 0)
        for k, v := range m {
                arr = append(arr, [2]int{k, v})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][1] == arr[j][1] {
                        return arr[i][0] > arr[j][0]
                }
                return arr[i][1] < arr[j][1]
        })
        res := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                for j := 0; j < arr[i][1]; j++ {
                        res = append(res, arr[i][0])
                }
        }
        return res
}
```

49.8 1640. 能否连接形成数组 (2)

- 题目

```
给你一个整数数组 arr ，数组中的每个整数 互不相同 。
另有一个由整数数组构成的数组 pieces，其中的整数也 互不相同 。
请你以 任意顺序 连接 pieces 中的数组以形成 arr 。但是，不允许 对每个数组 pieces[i]␣
↪中的整数重新排序。
如果可以连接 pieces 中的数组形成 arr ，返回 true ；否则，返回 false 。
示例 1：输入：arr = [85], pieces = [[85]]输出：true
示例 2：输入：arr = [15,88], pieces = [[88],[15]] 输出：true
解释：依次连接 [15] 和 [88]
示例 3：输入：arr = [49,18,16], pieces = [[16,18,49]] 输出：false
解释：即便数字相符，也不能重新排列 pieces[0]
示例 4：输入：arr = [91,4,64,78], pieces = [[78],[4,64],[91]] 输出：true
解释：依次连接 [91]、[4,64] 和 [78]
示例 5：输入：arr = [1,3,5,7], pieces = [[2,4,6,8]] 输出：false
提示：1 <= pieces.length <= arr.length <= 100
    sum(pieces[i].length) == arr.length
    1 <= pieces[i].length <= arr.length
    1 <= arr[i], pieces[i][j] <= 100
    arr 中的整数 互不相同
    pieces 中的整数 互不相同（也就是说，如果将 pieces␣
↪扁平化成一维数组，数组中的所有整数互不相同）
```

- 解题思路

```
func canFormArray(arr []int, pieces [][]int) bool {
        for i := 0; i < len(pieces); i++ {
                value := pieces[i]
                length := len(value)
                flag := false
                for j := 0; j < len(arr); j++ {
                        if arr[j] == value[0] {
                                flag = true
                                for k := j; k < j+length && k < len(arr); k++ {
                                        if arr[k] != value[k-j] {
                                                return false
                                        } else {
                                                arr[k] = 0
                                        }
                                }
                                break
                        }
```

(续下页)

(接上页)

```
                }
                if flag == false {
                        return false
                }
        }
        for i := 0; i < len(arr); i++ {
                if arr[i] != 0 {
                        return false
                }
        }
        return true
}

# 2
func canFormArray(arr []int, pieces [][]int) bool {
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]] = i
        }
        for i := 0; i < len(pieces); i++ {
                if _, ok := m[pieces[i][0]]; !ok {
                        return false
                }
                for k := 0; k < len(pieces[i])-1; k++ {
                        if m[pieces[i][k+1]]-m[pieces[i][k]] != 1 {
                                return false
                        }
                }
        }
        return true
}
```

49.9 1646. 获取生成数组中的最大值 (1)

- 题目

```
给你一个整数 n 。按下述规则生成一个长度为 n + 1 的数组 nums ：
nums[0] = 0
nums[1] = 1
当 2 <= 2 * i <= n 时，nums[2 * i] = nums[i]
当 2 <= 2 * i + 1 <= n 时，nums[2 * i + 1] = nums[i] + nums[i + 1]
返回生成数组 nums 中的 最大 值。
```

(续下页)

(接上页)

```
示例 1: 输入: n = 7 输出: 3
解释: 根据规则:
  nums[0] = 0
  nums[1] = 1
  nums[(1 * 2) = 2] = nums[1] = 1
  nums[(1 * 2) + 1 = 3] = nums[1] + nums[2] = 1 + 1 = 2
  nums[(2 * 2) = 4] = nums[2] = 1
  nums[(2 * 2) + 1 = 5] = nums[2] + nums[3] = 1 + 2 = 3
  nums[(3 * 2) = 6] = nums[3] = 2
  nums[(3 * 2) + 1 = 7] = nums[3] + nums[4] = 2 + 1 = 3
因此, nums = [0,1,1,2,1,3,2,3], 最大值 3
示例 2: 输入: n = 2 输出: 1
解释: 根据规则, nums[0]、nums[1] 和 nums[2] 之中的最大值是 1
示例 3: 输入: n = 3 输出: 2
解释: 根据规则, nums[0]、nums[1]、nums[2] 和 nums[3] 之中的最大值是 2
提示: 0 <= n <= 100
```

- 解题思路

```
func getMaximumGenerated(n int) int {
        arr := make([]int, n+3)
        arr[0] = 0
        arr[1] = 1
        res := 0
        for i := 0; i <= n; i++ {
                if i%2 == 0 {
                        arr[i] = arr[i/2]
                } else {
                        arr[i] = arr[i/2] + arr[(i+1)/2]
                }
                if arr[i] > res {
                        res = arr[i]
                }
        }
        return res
}
```

49.10 1652. 拆炸弹 (1)

- 题目

```
你有一个炸弹需要拆除，时间紧迫！你的情报员会给你一个长度为n的循环数组code以及一个密钥k。
为了获得正确的密码，你需要替换掉每一个数字。所有数字会同时被替换。
如果k > 0，将第i个数字用 接下来k个数字之和替换。
如果k < 0，将第i个数字用 之前k个数字之和替换。
如果k == 0，将第i个数字用0替换。
由于code是循环的，code[n-1]下一个元素是code[0]，且code[0]前一个元素是code[n-1]。
给你 循环数组code和整数密钥k，请你返回解密后的结果来拆除炸弹！
示例 1：输入：code = [5,7,1,4]，k = 3 输出：[12,10,16,13]
解释：每个数字都被接下来 3 个数字之和替换。
解密后的密码为 [7+1+4，1+4+5，4+5+7，5+7+1]。注意到数组是循环连接的。
示例 2：输入：code = [1,2,3,4]，k = 0 输出：[0,0,0,0]
解释：当 k 为 0 时，所有数字都被 0 替换。
示例 3：输入：code = [2,4,9,3]，k = -2 输出：[12,5,6,13]
解释：解密后的密码为 [3+9，2+3，4+2，9+4] 。注意到数组是循环连接的。
如果 k 是负数，那么和为 之前 的数字。
提示：n == code.length
1 <= n<= 100
1 <= code[i] <= 100
-(n - 1) <= k <= n - 1
```

- 解题思路

```
func decrypt(code []int, k int) []int {
        n := len(code)
        res := make([]int, n)
        if k == 0 {
                return res
        }
        for i := 0; i < n; i++ {
                sum := 0
                for j := 1; j <= abs(k); j++ {
                        if k > 0 {
                                sum = sum + code[(i+j+n)%n]
                        } else {
                                sum = sum + code[(i-j+n)%n]
                        }
                }
                res[i] = sum
        }
        return res
```

(续下页)

(接上页)

```
}

func abs(x int) int {
        if x < 0 {
                return -x
        }
        return x
}
```

49.11 1656. 设计有序流 (1)

- 题目

```
有 n 个 (id, value) 对，其中 id 是 1 到 n 之间的一个整数，value 是一个字符串。
不存在 id 相同的两个(id, value) 对。
设计一个流，以 任意 顺序获取 n个(id, value)对，并在多次调用时 按 id 递增的顺序
↪返回一些值。
实现 OrderedStream 类：
OrderedStream(int n) 构造一个能接收 n 个值的流，并将当前指针 ptr 设为 1 。
String[] insert(int id, String value) 向流中存储新的 (id, value) 对。存储后：
如果流存储有 id = ptr 的 (id, value) 对，则找出从 id = ptr 开始的 最长 id
↪连续递增序列 ，
并 按顺序 返回与这些 id 关联的值的列表。然后，将 ptr 更新为最后那个 id + 1。
否则，返回一个空列表。
示例：输入["OrderedStream", "insert", "insert", "insert", "insert", "insert"]
[[5], [3, "ccccc"], [1, "aaaaa"], [2, "bbbbb"], [5, "eeeee"], [4, "ddddd"]]
输出[null, [], ["aaaaa"], ["bbbbb", "ccccc"], [], ["ddddd", "eeeee"]]
解释：OrderedStream os= new OrderedStream(5);
os.insert(3, "ccccc"); // 插入 (3, "ccccc")，返回 []
os.insert(1, "aaaaa"); // 插入 (1, "aaaaa")，返回 ["aaaaa"]
os.insert(2, "bbbbb"); // 插入 (2, "bbbbb")，返回 ["bbbbb", "ccccc"]
os.insert(5, "eeeee"); // 插入 (5, "eeeee")，返回 []
os.insert(4, "ddddd"); // 插入 (4, "ddddd")，返回 ["ddddd", "eeeee"]
提示：1 <= n <= 1000
1 <= id <= n
value.length == 5
value 仅由小写字母组成
每次调用 insert 都会使用一个唯一的 id
恰好调用 n 次 insert
```

- 解题思路

```
type OrderedStream struct {
        arr []string
        ptr int
}

func Constructor(n int) OrderedStream {
        return OrderedStream{
                arr: make([]string, n+1),
                ptr: 1,
        }
}

func (this *OrderedStream) Insert(id int, value string) []string {
        res := make([]string, 0)
        this.arr[id] = value
        if this.ptr < id-1 {
                return res
        }
        for i := this.ptr; i < len(this.arr); i++ {
                if this.arr[i] == "" {
                        break
                }
                res = append(res, this.arr[i])
                this.ptr++
        }
        return res
}
```

49.12 1662. 检查两个字符串数组是否相等 (2)

- 题目

```
给你两个字符串数组 word1 和 word2 。如果两个数组表示的字符串相同，返回 true␣
↪；否则，返回 false 。
数组表示的字符串是由数组中的所有元素 按顺序 连接形成的字符串。
示例 1：输入：word1 = ["ab", "c"], word2 = ["a", "bc"] 输出：true
解释：word1 表示的字符串为 "ab" + "c" -> "abc"
word2 表示的字符串为 "a" + "bc" -> "abc"
两个字符串相同，返回 true
示例 2：输入：word1 = ["a", "cb"], word2 = ["ab", "c"] 输出：false
示例 3：输入：word1  = ["abc", "d", "defg"], word2 = ["abcddefg"] 输出：true
提示：1 <= word1.length, word2.length <= 103
```

(续下页)

(接上页)

```
1 <= word1[i].length, word2[i].length <= 103
1 <= sum(word1[i].length), sum(word2[i].length) <= 103
word1[i] 和 word2[i] 由小写字母组成
```

- 解题思路

```
func arrayStringsAreEqual(word1 []string, word2 []string) bool {
        return strings.Join(word1,"") == strings.Join(word2,"")
}

# 2
func arrayStringsAreEqual(word1 []string, word2 []string) bool {
        str1 := ""
        str2 := ""
        for i := 0; i < len(word1); i++ {
                str1 = str1 + word1[i]
        }
        for i := 0; i < len(word2); i++ {
                str2 = str2 + word2[i]
        }
        return str1 == str2
}
```

49.13 1668. 最大重复子字符串 (1)

- 题目

```
给你一个字符串sequence，如果字符串␣
↪word连续重复k次形成的字符串是sequence的一个子字符串，
那么单词word 的 重复值为 k 。单词 word的␣
↪最大重复值是单词word在sequence中最大的重复值。
如果word不是sequence的子串，那么重复值k为 0 。
给你一个字符串 sequence和 word，请你返回 最大重复值k 。
示例 1：输入：sequence = "ababc", word = "ab" 输出：2
解释："abab" 是 "ababc" 的子字符串。
示例 2：输入：sequence = "ababc", word = "ba" 输出：1
解释："ba" 是 "ababc" 的子字符串，但 "baba" 不是 "ababc" 的子字符串。
示例 3：输入：sequence = "ababc", word = "ac" 输出：0
解释："ac" 不是 "ababc" 的子字符串。
提示：1 <= sequence.length <= 100
1 <= word.length <= 100
sequence 和word都只包含小写英文字母。
```

- 解题思路

```
func maxRepeating(sequence string, word string) int {
        res := 0
        for i := 1; ; i++ {
                if strings.Contains(sequence, strings.Repeat(word, i)) == false {
                        break
                } else {
                        res = i
                }
        }
        return res
}
```

49.14 1672. 最富有客户的资产总量 (1)

- 题目

```
给你一个 m x n 的整数网格 accounts ，其中 accounts[i][j] 是第 i位客户在第 j家银行托管的资产数量。
返回最富有客户所拥有的 资产总量 。
客户的 资产总量 就是他们在各家银行托管的资产数量之和。最富有客户就是 资产总量最大的客户。
示例 1：输入：accounts = [[1,2,3],[3,2,1]] 输出：6
解释：第 1 位客户的资产总量 = 1 + 2 + 3 = 6
第 2 位客户的资产总量 = 3 + 2 + 1 = 6
两位客户都是最富有的，资产总量都是 6 ，所以返回 6 。
示例 2：输入：accounts = [[1,5],[7,3],[3,5]] 输出：10
解释：第 1 位客户的资产总量 = 6
第 2 位客户的资产总量 = 10
第 3 位客户的资产总量 = 8
第 2 位客户是最富有的，资产总量是 10
示例 3：输入：accounts = [[2,8,7],[7,1,3],[1,9,5]] 输出：17
提示：m ==accounts.length
n ==accounts[i].length
1 <= m, n <= 50
1 <= accounts[i][j] <= 100
```

- 解题思路

```
func maximumWealth(accounts [][]int) int {
        res := 0
        for i := 0; i < len(accounts); i++ {
```

(续下页)

(接上页)

```
                sum := 0
                for j := 0; j < len(accounts[i]); j++ {
                        sum = sum + accounts[i][j]
                }
                if sum > res {
                        res = sum
                }
        }
        return res
}
```

49.15 1678. 设计 Goal 解析器 (2)

- 题目

```
请你设计一个可以解释字符串 command 的 Goal 解析器 。
command 由 "G"、"()" 和/或 "(al)" 按某种顺序组成。
Goal 解析器会将 "G" 解释为字符串 "G"、"()" 解释为字符串 "o" ，"(al)" 解释为字符串 "al
↪" 。
然后，按原顺序将经解释得到的字符串连接成一个字符串。
给你字符串 command ，返回 Goal 解析器 对 command 的解释结果。
示例 1：输入：command = "G()(al)" 输出："Goal"
解释：Goal 解析器解释命令的步骤如下所示：
G -> G
() -> o
(al) -> al
最后连接得到的结果是 "Goal"
示例 2：输入：command = "G()()()()(al)" 输出："Gooooal"
示例 3：输入：command = "(al)G(al)()()G" 输出："alGalooG"
提示：1 <= command.length <= 100
command 由 "G"、"()" 和/或 "(al)" 按某种顺序组成
```

- 解题思路

```
func interpret(command string) string {
        command = strings.ReplaceAll(command, "(al)", "al")
        command = strings.ReplaceAll(command, "()", "o")
        return command
}

# 2
func interpret(command string) string {
```

(续下页)

(接上页)

```
        res := ""
        for i := 0; i < len(command); {
                if command[i] == 'G' {
                        res = res + "G"
                        i = i + 1
                } else if command[i] == '(' {
                        if command[i+1] == ')' {
                                res = res + "o"
                                i = i + 2
                        } else {
                                res = res + "al"
                                i = i + 4
                        }
                }
        }
        return res
}
```

49.16 1684. 统计一致字符串的数目 (1)

- 题目

```
给你一个由不同字符组成的字符串allowed和一个字符串数组words。
如果一个字符串的每一个字符都在 allowed中，就称这个字符串是 一致字符串。
请你返回words数组中一致字符串的数目。
示例 1：输入：allowed = "ab", words = ["ad","bd","aaab","baa","badab"] 输出：2
解释：字符串 "aaab" 和 "baa" 都是一致字符串，因为它们只包含字符 'a' 和 'b' 。
示例 2：输入：allowed = "abc", words = ["a","b","c","ab","ac","bc","abc"] 输出：7
解释：所有字符串都是一致的。
示例 3：输入：allowed = "cad", words = ["cc","acd","b","ba","bac","bad","ac","d"]␣
↪输出：4
解释：字符串 "cc", "acd", "ac" 和 "d" 是一致字符串。
提示：1 <= words.length <= 104
1 <= allowed.length <= 26
1 <= words[i].length <= 10
allowed中的字符 互不相同。
words[i] 和allowed只包含小写英文字母。
```

- 解题思路

```
func countConsistentStrings(allowed string, words []string) int {
        res := 0
```

(续下页)

(接上页)

```
        m := make(map[byte]bool)
        for i := 0; i < len(allowed); i++ {
                m[allowed[i]] = true
        }

        for _, word := range words {
                flag := true
                for i := 0; i < len(word); i++ {
                        if _, ok := m[word[i]]; !ok {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res++
                }
        }
        return res
}
```

49.17 1688. 比赛中的配对次数 (2)

- 题目

```
给你一个整数 n ，表示比赛中的队伍数。比赛遵循一种独特的赛制：
如果当前队伍数是 偶数 ，那么每支队伍都会与另一支队伍配对。
总共进行 n / 2 场比赛，且产生 n / 2 支队伍进入下一轮。
如果当前队伍数为 奇数 ，那么将会随机轮空并晋级一支队伍，其余的队伍配对。
总共进行 (n - 1) / 2 场比赛，且产生 (n - 1) / 2 + 1 支队伍进入下一轮。
返回在比赛中进行的配对次数，直到决出获胜队伍为止。
示例 1：输入：n = 7 输出：6
解释：比赛详情：
- 第 1 轮：队伍数 = 7 ，配对次数 = 3 ，4 支队伍晋级。
- 第 2 轮：队伍数 = 4 ，配对次数 = 2 ，2 支队伍晋级。
- 第 3 轮：队伍数 = 2 ，配对次数 = 1 ，决出 1 支获胜队伍。
总配对次数 = 3 + 2 + 1 = 6
示例 2：输入：n = 14 输出：13
解释：比赛详情：
- 第 1 轮：队伍数 = 14 ，配对次数 = 7 ，7 支队伍晋级。
- 第 2 轮：队伍数 = 7 ，配对次数 = 3 ，4 支队伍晋级。
- 第 3 轮：队伍数 = 4 ，配对次数 = 2 ，2 支队伍晋级。
- 第 4 轮：队伍数 = 2 ，配对次数 = 1 ，决出 1 支获胜队伍。
```

(续下页)

(接上页)

```
总配对次数 = 7 + 3 + 2 + 1 = 13
提示：1 <= n <= 200
```

- 解题思路

```
func numberOfMatches(n int) int {
        res := 0
        for n > 1 {
                res = res + n/2
                n = n/2 + n%2
        }
        return res
}

# 2
func numberOfMatches(n int) int {
        // 共有n个队伍，一个冠军，需要淘汰n-1个 队伍。
        // 每一场比赛淘汰一个队伍，因此进行了n-1场比赛。
        return n - 1
}
```

49.18 1694. 重新格式化电话号码 (1)

- 题目

```
给你一个字符串形式的电话号码 number 。number 由数字、空格 ' '、和破折号 '-' 组成。
请你按下述方式重新格式化电话号码。
首先，删除 所有的空格和破折号。
其次，将数组从左到右 每 3 个一组 分块，直到 剩下 4
↪个或更少数字。剩下的数字将按下述规定再分块：
2 个数字：单个含 2 个数字的块。
3 个数字：单个含 3 个数字的块。
4 个数字：两个分别含 2 个数字的块。
最后用破折号将这些块连接起来。
注意，重新格式化过程中 不应该 生成仅含 1 个数字的块，并且 最多 生成两个含 2
↪个数字的块。
返回格式化后的电话号码。
示例 1：输入：number = "1-23-45 6" 输出："123-456"
解释：数字是 "123456"
步骤 1：共有超过 4 个数字，所以先取 3 个数字分为一组。第 1 个块是 "123" 。
步骤 2：剩下 3 个数字，将它们放入单个含 3 个数字的块。第 2 个块是 "456" 。
连接这些块后得到 "123-456" 。
```

(续下页)

(接上页)

```
示例 2：输入：number = "123 4-567" 输出："123-45-67"
解释：数字是 "1234567".
步骤 1：共有超过 4 个数字，所以先取 3 个数字分为一组。第 1 个块是 "123" 。
步骤 2：剩下 4 个数字，所以将它们分成两个含 2 个数字的块。这 2 块分别是 "45" 和 "67"␣
↪。
连接这些块后得到 "123-45-67" 。
示例 3：输入：number = "123 4-5678" 输出："123-456-78"
解释：数字是 "12345678" 。
步骤 1：第 1 个块 "123" 。
步骤 2：第 2 个块 "456" 。
步骤 3：剩下 2 个数字，将它们放入单个含 2 个数字的块。第 3 个块是 "78" 。
连接这些块后得到 "123-456-78" 。
示例 4：输入：number = "12" 输出："12"
示例 5：输入：number = "--17-5 229 35-39475 " 输出："175-229-353-94-75"
提示：2 <= number.length <= 100
number 由数字和字符 '-' 及 ' ' 组成。
number 中至少含 2 个数字。
```

- 解题思路

```
func reformatNumber(number string) string {
        str := strings.ReplaceAll(number, "-", "")
        str = strings.ReplaceAll(str, " ", "")
        res := ""
        for len(str) > 4 {
                res = res + str[:3] + "-"
                str = str[3:]
        }
        if len(str) == 4 {
                res = res + str[:2] + "-" + str[2:]
        } else {
                res = res + str
        }
        return res
}
```

49.19 1700. 无法吃午餐的学生数量 (1)

- 题目

```
学校的自助午餐提供圆形和方形的三明治，分别用数字0和1表示。
所有学生站在一个队列里，每个学生要么喜欢圆形的要么喜欢方形的。
餐厅里三明治的数量与学生的数量相同。所有三明治都放在一个栈里，每一轮：
如果队列最前面的学生喜欢栈顶的三明治，那么会拿走它并离开队列。
否则，这名学生会放弃这个三明治并回到队列的尾部。
这个过程会一直持续到队列里所有学生都不喜欢栈顶的三明治为止。
给你两个整数数组students 和sandwiches，
其中sandwiches[i]是栈里面第i个三明治的类型（i = 0是栈的顶部），
students[j]是初始队列里第j名学生对三明治的喜好（j = 0是队列的最开始位置）。
请你返回无法吃午餐的学生数量。
示例 1：输入：students = [1,1,0,0], sandwiches = [0,1,0,1] 输出：0
解释：- 最前面的学生放弃最顶上的三明治，并回到队列的末尾，学生队列变为 students = [1,
↪0,0,1]。
- 最前面的学生放弃最顶上的三明治，并回到队列的末尾，学生队列变为 students = [0,0,1,
↪1]。
- 最前面的学生拿走最顶上的三明治，剩余学生队列为 students = [0,1,1],
三明治栈为 sandwiches = [1,0,1]。
- 最前面的学生放弃最顶上的三明治，并回到队列的末尾，学生队列变为 students = [1,1,0]。
- 最前面的学生拿走最顶上的三明治，剩余学生队列为 students = [1,0]，三明治栈为␣
↪sandwiches = [0,1]。
- 最前面的学生放弃最顶上的三明治，并回到队列的末尾，学生队列变为 students = [0,1]。
- 最前面的学生拿走最顶上的三明治，剩余学生队列为 students = [1]，三明治栈为␣
↪sandwiches = [1]。
- 最前面的学生拿走最顶上的三明治，剩余学生队列为 students = []，三明治栈为 sandwiches␣
↪= []。
所以所有学生都有三明治吃。
示例 2：输入：students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1] 输出：3
提示：1 <= students.length, sandwiches.length <= 100
students.length == sandwiches.length
sandwiches[i]要么是0，要么是1。
students[i]要么是0，要么是1。
```

- 解题思路

```
func countStudents(students []int, sandwiches []int) int {
        a, b := 0, 0
        for i := 0; i < len(students); i++ {
                if students[i] == 0 {
                        a++
                } else {
```

(续下页)

(接上页)

```
                        b++
                }
        }
        for i := 0; i < len(sandwiches); i++ {
                if sandwiches[i] == 0 && a > 0 {
                        a--
                } else if sandwiches[i] == 1 && b > 0 {
                        b--
                } else {
                        break
                }
        }
        return a + b
}
```

CHAPTER 50

1601-1700-Medium

50.1 1604. 警告一小时内使用相同员工卡大于等于三次的人 (2)

- 题目

```
力扣公司的员工都使用员工卡来开办公室的门。
每当一个员工使用一次他的员工卡，安保系统会记录下员工的名字和使用时间。
如果一个员工在一小时时间内使用员工卡的次数大于等于三次，这个系统会自动发布一个 警告 。
给你字符串数组 keyName 和 keyTime ，
其中 [keyName[i], keyTime[i]] 对应一个人的名字和他在 某一天 内使用员工卡的时间。
使用时间的格式是 24小时制 ，形如 "HH:MM" ，比方说 "23:51" 和 "09:49" 。
请你返回去重后的收到系统警告的员工名字，将它们按 字典序升序 排序后返回。
请注意 "10:00" - "11:00" 视为一个小时时间范围内，
而 "23:51" - "00:10" 不被视为一小时内，因为系统记录的是某一天内的使用情况。
示例 1：输入：keyName = ["daniel","daniel","daniel","luis","luis","luis","luis"],
keyTime = ["10:00","10:40","11:00","09:00","11:00","13:00","15:00"]
输出：["daniel"]
解释："daniel" 在一小时内使用了 3 次员工卡 ("10:00", "10:40", "11:00") 。
示例 2：输入：keyName = ["alice","alice","alice","bob","bob","bob","bob"],
keyTime = ["12:01","12:00","18:00","21:00","21:20","21:30","23:00"]
输出：["bob"]
解释："bob" 在一小时内使用了 3 次员工卡 ("21:00", "21:20", "21:30") 。
示例 3：输入：keyName = ["john","john","john"], keyTime = ["23:58","23:59","00:01"]
输出：[]
示例 4：输入：keyName = ["leslie","leslie","leslie","clare","clare","clare","clare"],
```

(续下页)

(接上页)

```
keyTime = ["13:00","13:20","14:00","18:00","18:51","19:30","19:49"]
输出：["clare","leslie"]
提示：1 <= keyName.length, keyTime.length <= 105
    keyName.length == keyTime.length
    keyTime 格式为 "HH:MM" 。
    保证 [keyName[i], keyTime[i]] 形成的二元对 互不相同 。
    1 <= keyName[i].length <= 10
    keyName[i] 只包含小写英文字母。
```

- 解题思路

```
func alertNames(keyName []string, keyTime []string) []string {
        m := make(map[string][]int)
        for i := 0; i < len(keyName); i++ {
                m[keyName[i]] = append(m[keyName[i]], strToInt(keyTime[i]))
        }
        res := make([]string, 0)
        for k, v := range m {
                sort.Ints(v)
                first := v[0]
                second := v[0]
                count := 1
                for i := 1; i < len(v); i++ {
                        if v[i] > first && v[i]-first <= 60 {
                                second = v[i]
                                count++
                        } else {
                                first = second
                                second = v[i]
                                count = 2
                        }
                        if count >= 3 {
                                res = append(res, k)
                                break
                        }
                }
        }
        sort.Strings(res)
        return res
}

func strToInt(str string) int {
        arr := strings.Split(str, ":")
        hour, _ := strconv.Atoi(arr[0])
```

(续下页)

(接上页)

```
        minute, _ := strconv.Atoi(arr[1])
        return hour*60 + minute
}

# 2
func alertNames(keyName []string, keyTime []string) []string {
        m := make(map[string][]int)
        for i := 0; i < len(keyName); i++ {
                m[keyName[i]] = append(m[keyName[i]], strToInt(keyTime[i]))
        }
        res := make([]string, 0)
        for k, v := range m {
                sort.Ints(v)
                for i := 0; i < len(v)-2; i++ {
                        if v[i+2]-v[i] <= 60 {
                                res = append(res, k)
                                break
                        }
                }
        }
        sort.Strings(res)
        return res
}

func strToInt(str string) int {
        arr := strings.Split(str, ":")
        hour, _ := strconv.Atoi(arr[0])
        minute, _ := strconv.Atoi(arr[1])
        return hour*60 + minute
}
```

50.2 1605. 给定行和列的和求可行矩阵 (1)

- 题目

```
给你两个非负整数数组 rowSum 和 colSum ，其中 rowSum[i] 是二维矩阵中第 i 行元素的和，
colSum[j] 是第 j␣
↪列元素的和。换言之你不知道矩阵里的每个元素，但是你知道每一行和每一列的和。
请找到大小为 rowSum.length x colSum.length 的任意 非负整数 矩阵，
且该矩阵满足 rowSum 和 colSum 的要求。
请你返回任意一个满足题目要求的二维矩阵，题目保证存在 至少一个 可行矩阵。
示例 1：输入：rowSum = [3,8], colSum = [4,7] 输出：[[3,0],
```

(续下页)

(接上页)

```
      [1,7]]
解释：第 0 行：3 + 0 = 0 == rowSum[0]
第 1 行：1 + 7 = 8 == rowSum[1]
第 0 列：3 + 1 = 4 == colSum[0]
第 1 列：0 + 7 = 7 == colSum[1]
行和列的和都满足题目要求，且所有矩阵元素都是非负的。
另一个可行的矩阵为：[[1,2],
                  [3,5]]
示例 2：输入：rowSum = [5,7,10], colSum = [8,6,8]
输出：[[0,5,0],
      [6,1,0],
      [2,0,8]]
示例 3：输入：rowSum = [14,9], colSum = [6,9,8]
输出：[[0,9,5],
      [6,0,3]]
示例 4：输入：rowSum = [1,0], colSum = [1]
输出：[[1],
      [0]]
示例 5：输入：rowSum = [0], colSum = [0]
输出：[[0]]
提示：1 <= rowSum.length, colSum.length <= 500
    0 <= rowSum[i], colSum[i] <= 108
    sum(rows) == sum(columns)
```

- 解题思路

```
func restoreMatrix(rowSum []int, colSum []int) [][]int {
        n := len(rowSum)
        m := len(colSum)
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                res[i] = make([]int, m)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        value := min(rowSum[i], colSum[j])
                        res[i][j] = value
                        rowSum[i] = rowSum[i] - value
                        colSum[j] = colSum[j] - value
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

50.3 1609. 奇偶树 (1)

- 题目

```
如果一棵二叉树满足下述几个条件，则可以称为 奇偶树 ：
    二叉树根节点所在层下标为 0 ，根的子节点所在层下标为 1 ，根的孙节点所在层下标为 2␣
↪，依此类推。
    偶数下标 层上的所有节点的值都是 奇 整数，从左到右按顺序 严格递增
    奇数下标 层上的所有节点的值都是 偶 整数，从左到右按顺序 严格递减
给你二叉树的根节点，如果二叉树为 奇偶树 ，则返回 true ，否则返回 false 。
示例 1：输入：root = [1,10,4,3,null,7,9,12,8,6,null,null,2]
输出：true
解释：每一层的节点值分别是：
0 层：[1]
1 层：[10,4]
2 层：[3,7,9]
3 层：[12,8,6,2]
由于 0 层和 2 层上的节点值都是奇数且严格递增，而 1 层和 3␣
↪层上的节点值都是偶数且严格递减，
因此这是一棵奇偶树。
示例 2：输入：root = [5,4,2,3,3,7] 输出：false
解释：每一层的节点值分别是：
0 层：[5]
1 层：[4,2]
2 层：[3,3,7]
2 层上的节点值不满足严格递增的条件，所以这不是一棵奇偶树。
示例 3：输入：root = [5,9,1,3,5,7] 输出：false
解释：1 层上的节点值应为偶数。
示例 4：输入：root = [1]输出：true
示例 5：输入：root = [11,8,6,1,3,9,11,30,20,18,16,12,10,4,2,17] 输出：true
提示：树中节点数在范围 [1, 105] 内
    1 <= Node.val <= 106
```

- 解题思路

```
func isEvenOddTree(root *TreeNode) bool {
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        level := 1
        for len(queue) > 0 {
                length := len(queue)
                temp := make([]int, 0)
                for i := 0; i < length; i++ {
                        temp = append(temp, queue[i].Val)
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                }
                if level%2 == 0 {
                        for j := 0; j < len(temp)/2; j++ {
                                temp[j], temp[len(temp)-1-j] = temp[len(temp)-1-j],␣
↪temp[j]
                        }
                }
                for i := 0; i < len(temp); i++ {
                        if i < len(temp)-1 && temp[i] >= temp[i+1] {
                                return false
                        }
                        if temp[i]%2 != level%2 {
                                return false
                        }
                }
                queue = queue[length:]
                level++
        }
        return true
}
```

50.4 1615. 最大网络秩 (1)

- 题目

```
n 座城市和一些连接这些城市的道路 roads 共同组成一个基础设施网络。
每个 roads[i] = [ai, bi] 都表示在城市 ai 和 bi 之间有一条双向道路。
两座不同城市构成的 城市对 的 网络秩 定义为：与这两座城市 直接 相连的道路总数。
如果存在一条道路直接连接这两座城市，则这条道路只计算 一次 。
整个基础设施网络的 最大网络秩 是所有不同城市对中的 最大网络秩 。
给你整数 n 和数组 roads，返回整个基础设施网络的 最大网络秩 。
示例 1：输入：n = 4, roads = [[0,1],[0,3],[1,2],[1,3]] 输出：4
解释：城市 0 和 1 的网络秩是 4，因为共有 4 条道路与城市 0 或 1 相连。
位于 0 和 1 之间的道路只计算一次。
示例 2：输入：n = 5, roads = [[0,1],[0,3],[1,2],[1,3],[2,3],[2,4]] 输出：5
解释：共有 5 条道路与城市 1 或 2 相连。
示例 3：输入：n = 8, roads = [[0,1],[1,2],[2,3],[2,4],[5,6],[5,7]] 输出：5
解释：2 和 5 的网络秩为 5，注意并非所有的城市都需要连接起来。
提示：2 <= n <= 100
0 <= roads.length <= n * (n - 1) / 2
roads[i].length == 2
0 <= ai, bi<= n-1
ai!=bi
每对城市之间 最多只有一条道路相连
```

- 解题思路

```
func maximalNetworkRank(n int, roads [][]int) int {
        arr := make([]int, n)
        m := make(map[int]int)
        for i := 0; i < len(roads); i++ {
                a, b := roads[i][0], roads[i][1]
                arr[a]++
                arr[b]++
                m[a*100+b]++
                m[b*100+a]++
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        value := arr[i] + arr[j] - m[i*100+j]
                        if value > res {
                                res = value
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

50.5 1616. 分割两个字符串得到回文串 (2)

- 题目

```
给你两个字符串 a 和 b ，它们长度相同。请你选择一个下标，将两个字符串都在 相同的下标分割开。
由 a 可以得到两个字符串： aprefix 和 asuffix ，满足 a = aprefix + asuffix ，
同理，由 b 可以得到两个字符串 bprefix 和 bsuffix ，满足 b = bprefix + bsuffix 。
请你判断 aprefix + bsuffix 或者 bprefix + asuffix 能否构成回文串。
当你将一个字符串 s 分割成 sprefix 和 ssuffix 时， ssuffix 或者 sprefix 可以为空。
比方说， s = "abc" 那么 "" + "abc" ， "a" + "bc" ， "ab" + "c" 和 "abc" + "" 都是合法分割。
如果 能构成回文字符串 ，那么请返回 true，否则返回 false 。
请注意， x + y 表示连接字符串 x 和 y 。
示例 1：输入：a = "x", b = "y" 输出：true
解释：如果 a 或者 b 是回文串，那么答案一定为 true ，因为你可以如下分割：
aprefix = "", asuffix = "x"
bprefix = "", bsuffix = "y"
那么 aprefix + bsuffix = "" + "y" = "y" 是回文串。
示例 2：输入：a = "ulacfd", b = "jizalu" 输出：true
解释：在下标为 3 处分割：
aprefix = "ula", asuffix = "cfd"
bprefix = "jiz", bsuffix = "alu"
那么 aprefix + bsuffix = "ula" + "alu" = "ulaalu" 是回文串。
提示：1 <= a.length, b.length <= 105
    a.length == b.length
    a 和 b 都只包含小写英文字母
```

- 解题思路

```
func checkPalindromeFormation(a string, b string) bool {
        if judge(a, b) == true || judge(b, a) == true {
                return true
        }
        return false
}

// 判断prefixA+suffixB是否是回文
```

(续下页)

(接上页)

```
func judge(a, b string) bool {
        left, right := 0, len(a)-1
        for left < right {
                if a[left] == b[right] {
                        left++
                        right--
                } else {
                        break
                }
        }
        var i, j int
        // 2种判断
        // A1+A2+A3
        // B1+B2+B3
        // 1.reverse(A1)=B3, 判断B2是否回文
        i, j = left, right
        for i < j {
                if b[i] == b[j] {
                        i++
                        j--
                } else {
                        break
                }
        }
        if i >= j {
                return true
        }
        // 2.reverse(A1)=B3, 判断A2是否回文
        i, j = left, right
        for i < j {
                if a[i] == a[j] {
                        i++
                        j--
                } else {
                        break
                }
        }
        if i >= j {
                return true
        }
        return false
}
```

(续下页)

(接上页)

```
# 2
func checkPalindromeFormation(a string, b string) bool {
        start := len(a)/2 - 1 // 中间靠左坐标
        c := check(a, a, start)
        if check(a, b, c) == -1 || check(b, a, c) == -1 {
                return true
        }
        c = check(b, b, start)
        if check(a, b, c) == -1 || check(b, a, c) == -1 {
                return true
        }
        return false
}

// 从start往左边走，从n-1-start往右边走
func check(a, b string, start int) int {
        left := start
        right := len(a) - 1 - start
        for left >= 0 {
                if a[left] != b[right] {
                        break
                }
                left--
                right++
        }
        return left
}
```

50.6 1620. 网络信号最好的坐标 (1)

- 题目

```
给你一个数组 towers和一个整数 radius，数组中包含一些网络信号塔，
其中towers[i] = [xi, yi, qi]表示第i个网络信号塔的坐标是(xi, yi)且信号强度参数为qi。
所有坐标都是在 X-Y 坐标系内的整数坐标。两个坐标之间的距离用 欧几里得距离计算。
整数radius表示一个塔 能到达的 最远距离。
如果一个坐标跟塔的距离在 radius以内，那么该塔的信号可以到达该坐标。
在这个范围以外信号会很微弱，所以 radius以外的距离该塔是 不能到达的。
如果第 i个塔能到达 (x, y)，那么该塔在此处的信号为⌊qi / (1 +␣
↪d)⌋，其中d是塔跟此坐标的距离。
一个坐标的 网络信号是所有 能到达该坐标的塔的信号强度之和。
请你返回␣
```

(续下页)

(接上页)

```
↪网络信号最大的整数坐标点。如果有多个坐标网络信号一样大，请你返回字典序最小的一个坐标。
注意：坐标(x1, y1)字典序比另一个坐标(x2, y2)小：要么x1 < x2，要么x1 == x2 且y1 < y2。
⌊val⌋表示小于等于val的最大整数（向下取整函数）。
示例 1：输入：towers = [[1,2,5],[2,1,7],[3,1,9]], radius = 2 输出：[2,1]
解释：坐标 (2, 1) 信号强度之和为 13
- 塔 (2, 1) 强度参数为 7 ，在该点强度为 ⌊7 / (1 + sqrt(0))⌋ = ⌊7⌋ = 7
- 塔 (1, 2) 强度参数为 5 ，在该点强度为 ⌊5 / (1 + sqrt(2))⌋ = ⌊2.07⌋ = 2
- 塔 (3, 1) 强度参数为 9 ，在该点强度为 ⌊9 / (1 + sqrt(1))⌋ = ⌊4.5⌋ = 4
没有别的坐标有更大的信号强度。
示例 2：输入：towers = [[23,11,21]], radius = 9 输出：[23,11]
示例 3：输入：towers = [[1,2,13],[2,1,7],[0,1,9]], radius = 2 输出：[1,2]
示例 4：输入：towers = [[2,1,9],[0,1,9]], radius = 2 输出：[0,1]
解释：坐标 (0, 1) 和坐标 (2, 1) 都是强度最大的位置，但是 (0, 1) 字典序更小。
提示：1 <= towers.length <= 50
towers[i].length == 3
0 <= xi, yi, qi <= 50
1 <= radius <= 50
```

- 解题思路

```
func bestCoordinate(towers [][]int, radius int) []int {
        res := []int{0, 0}
        maxValue := 0
        for i := 0; i <= 50; i++ {
                for j := 0; j <= 50; j++ {
                        sum := 0
                        for k := 0; k < len(towers); k++ {
                                a, b, c := towers[k][0], towers[k][1], towers[k][2]
                                d := (a-i)*(a-i) + (b-j)*(b-j)
                                if d <= radius*radius {
                                        value := float64(c) / (1 + math.
↪Sqrt(float64(d)))
                                        sum = sum + int(math.Floor(value))
                                }
                        }
                        if sum > maxValue {
                                maxValue = sum
                                res = []int{i, j}
                        }
                }
        }
        return res
}
```

50.7 1621. 大小为 K 的不重叠线段的数目 (2)

- 题目

```
给你一维空间的n个点，其中第i个点（编号从0 到n-1）位于x = i处，
请你找到恰好k个不重叠线段且每个线段至少覆盖两个点的方案数。
线段的两个端点必须都是整数坐标。
这k个线段不需要全部覆盖全部n个点，且它们的端点可以重合。
请你返回 k个不重叠线段的方案数。由于答案可能很大，请将结果对109 + 7取余 后返回。
示例 1：输入：n = 4, k = 2 输出：5
解释：如图所示，两个线段分别用红色和蓝色标出。
上图展示了 5 种不同的方案 {(0,2),(2,3)}, {(0,1),(1,3)}, {(0,1),(2,3)},
{(1,2),(2,3)}, {(0,1),(1,2)} 。
示例 2：输入：n = 3, k = 1 输出：3
解释：总共有 3 种不同的方案 {(0,1)}, {(0,2)}, {(1,2)} 。
示例 3：输入：n = 30, k = 7 输出：796297179
解释：画 7 条线段的总方案数为 3796297200 种。将这个数对 109 + 7 取余得到 796297179 。
示例 4：输入：n = 5, k = 3 输出：7
示例 5：输入：n = 3, k = 2 输出：1
提示：2 <= n <= 1000
1 <= k <= n-1
```

- 解题思路

```
var mod = 1000000007

func numberOfSets(n int, k int) int {
        dp := make([][][2]int, n)
        // dp[i][j][0] =>0到i的点，构造成j条线段的方案数，第j条线段的右端点没有使用i
        // dp[i][j][1] =>0到i的点，构造成j条线段的方案数，第j条线段的右端点使用i
        for i := 0; i < n; i++ {
                dp[i] = make([][2]int, n)
        }
        dp[0][0][0] = 1
        for i := 1; i < n; i++ {
                for j := 0; j <= k; j++ {
                        dp[i][j][0] = (dp[i-1][j][0] + dp[i-1][j][1]) % mod //␣
↪没有使用i
                        dp[i][j][1] = dp[i-1][j][1]                          //␣
↪使用i：扩展右侧
                        if j > 0 {
                                dp[i][j][1] = (dp[i][j][1] + dp[i-1][j-1][0] + dp[i-
↪1][j-1][1]) % mod // 使用i：新开一个
                        }
```

(续下页)

(接上页)

```
                }
        }
        return (dp[n-1][k][0] + dp[n-1][k][1]) % mod
}

# 2

# 2
var mod = 1000000007

func numberOfSets(n int, k int) int {
        // 共享k-1个点+n个点 => n+k-1个点
        // 共 n+k-1 个数中选择 2k 个
        return C(n+k-1, 2*k)
}

func C(n, m int) int {
        a := multiMod(n, n-m+1)
        b := multiMod(m, 1)
        return a * powMod(b, mod-2) % mod
}

func multiMod(n, m int) int {
        res := 1
        for i := m; i <= n; i++ {
                res = (res * i) % mod
        }
        return res
}

func powMod(a, b int) int {
        res := 1
        for b > 0 {
                if b%2 == 1 {
                        res = (res * a) % mod
                }
                a = a * a % mod
                b = b / 2
        }
        return res
}
```

50.8 1625. 执行操作后字典序最小的字符串 (2)

- 题目

```
给你一个字符串 s 以及两个整数 a 和 b 。其中，字符串 s 的长度为偶数，且仅由数字 0 到 9 组成。
你可以在 s 上按任意顺序多次执行下面两个操作之一：
累加：将 a 加到 s 中所有下标为奇数的元素上（下标从 0 开始）。数字一旦超过 9 就会变成 0，如此循环往复。
例如，s = "3456" 且 a = 5，则执行此操作后 s 变成 "3951"。
轮转：将 s 向右轮转 b 位。例如，s = "3456" 且 b = 1，则执行此操作后 s 变成 "6345"。
请你返回在 s 上执行上述操作任意次后可以得到的 字典序最小 的字符串。
如果两个字符串长度相同，那么字符串 a 字典序比字符串 b 小可以这样定义：
在 a 和 b 出现不同的第一个位置上，字符串 a 中的字符出现在字母表中的时间早于 b 中的对应字符。
例如，"0158” 字典序比 "0190" 小，因为不同的第一个位置是在第三个字符，显然 '5' 出现在 '9' 之前。
示例 1：输入：s = "5525", a = 9, b = 2 输出："2050"
解释：执行操作如下：
初态："5525"
轮转："2555"
累加："2454"
累加："2353"
轮转："5323"
累加："5222"
累加："5121"
轮转："2151"
累加："2050"
无法获得字典序小于 "2050" 的字符串。
示例 2：输入：s = "74", a = 5, b = 1 输出："24"
解释：执行操作如下：
初态："74"
轮转："47"
累加："42"
轮转："24"
无法获得字典序小于 "24" 的字符串。
示例 3：输入：s = "0011", a = 4, b = 2 输出："0011"
解释：无法获得字典序小于 "0011" 的字符串。
示例 4：输入：s = "43987654", a = 7, b = 3 输出："00553311"
提示：2 <= s.length <= 100
s.length 是偶数
s 仅由数字 0 到 9 组成
1 <= a <= 9
1 <= b <= s.length - 1
```

- 解题思路

```
var m map[string]bool
var res string

func findLexSmallestString(s string, a int, b int) string {
        res = s
        m = make(map[string]bool)
        dfs(s, a, b)
        return res
}

func dfs(s string, a, b int) {
        if m[s] == true {
                return
        }
        m[s] = true
        if s < res {
                res = s
        }
        dfs(add(s, a), a, b)
        dfs(s[b:]+s[:b], a, b)
}

func add(s string, a int) string {
        res := []byte(s)
        for i := 1; i < len(s); i = i + 2 {
                res[i] = byte('0' + (int(s[i]-'0')+a)%10)
        }
        return string(res)
}

# 2
func findLexSmallestString(s string, a int, b int) string {
        res := s
        m := make(map[string]bool)
        queue := make([]string, 0)
        queue = append(queue, s)
        for len(queue) > 0 {
                str := queue[0]
                queue = queue[1:]
                if m[str] == true {
                        continue
                }
                m[str] = true
```

(续下页)

(接上页)

```
                if str < res {
                        res = str
                }
                queue = append(queue, str[b:]+str[:b])
                queue = append(queue, add(str, a))
        }
        return res
}

func add(s string, a int) string {
        res := []byte(s)
        for i := 1; i < len(s); i = i + 2 {
                res[i] = byte('0' + (int(s[i]-'0')+a)%10)
        }
        return string(res)
}
```

50.9 1626. 无矛盾的最佳球队 (1)

- 题目

```
假设你是球队的经理。对于即将到来的锦标赛，你想组合一支总体得分最高的球队。
球队的得分是球队中所有球员的分数 总和 。
然而，球队中的矛盾会限制球员的发挥，所以必须选出一支 没有矛盾 的球队。
如果一名年龄较小球员的分数 严格大于␣
↪一名年龄较大的球员，则存在矛盾。同龄球员之间不会发生矛盾。
给你两个列表 scores 和 ages，其中每组 scores[i] 和 ages[i] 表示第 i␣
↪名球员的分数和年龄。
请你返回 所有可能的无矛盾球队中得分最高那支的分数 。
示例 1：输入：scores = [1,3,5,10,15], ages = [1,2,3,4,5] 输出：34
解释：你可以选中所有球员。
示例 2：输入：scores = [4,5,6,5], ages = [2,1,2,1] 输出：16
解释：最佳的选择是后 3 名球员。注意，你可以选中多个同龄球员。
示例 3：输入：scores = [1,2,3,5], ages = [8,9,10,1] 输出：6
解释：最佳的选择是前 3 名球员。
提示： 1 <= scores.length, ages.length <= 1000
    scores.length == ages.length
    1 <= scores[i] <= 10^6
    1 <= ages[i] <= 1000
```

- 解题思路

```
func bestTeamScore(scores []int, ages []int) int {
        arr := make([][2]int, 0)
        for i := 0; i < len(ages); i++ {
                arr = append(arr, [2]int{ages[i], scores[i]})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][0] == arr[j][0] {
                        return arr[i][1] < arr[j][1]
                }
                return arr[i][0] < arr[j][0]
        })
        dp := make([]int, len(arr))
        res := 0
        for i := 0; i < len(arr); i++ {
                dp[i] = arr[i][1]
                for j := 0; j < i; j++ {
                        if arr[j][1] <= arr[i][1] {
                                dp[i] = max(dp[i], dp[j]+arr[i][1])
                        }
                }
                res = max(res, dp[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

50.10 1630. 等差子数组 (1)

- 题目

```
如果一个数列由至少两个元素组成，且每两个连续元素之间的差值都相同，那么这个序列就是␣
↪等差数列 。
更正式地，数列 s 是等差数列，只需要满足：对于每个有效的 i ， s[i+1] - s[i] == s[1] -␣
↪s[0] 都成立。
例如，下面这些都是 等差数列 ：
1, 3, 5, 7, 9
```

(续下页)

(接上页)

```
7, 7, 7, 7
3, -1, -5, -9
下面的数列 不是等差数列 :
1, 1, 2, 5, 7
给你一个由 n 个整数组成的数组 nums, 和两个由 m 个整数组成的数组 l 和r,
后两个数组表示 m 组范围查询,
其中第 i 个查询对应范围 [l[i], r[i]] 。所有数组的下标都是 从 0 开始 的。
返回 boolean 元素构成的答案列表 answer 。如果子数组 nums[l[i]], nums[l[i]+1], ... ,
nums[r[i]] 可以 重新排列 形成 等差数列 , answer[i] 的值就是 true; 否则answer[i]
的值就是 false 。
示例 1: 输入: nums = [4,6,5,9,3,7], l = [0,0,2], r = [2,3,5] 输出: [true,false,true]
解释: 第 0 个查询, 对应子数组 [4,6,5] 。可以重新排列为等差数列 [6,5,4] 。
第 1 个查询, 对应子数组 [4,6,5,9] 。无法重新排列形成等差数列。
第 2 个查询, 对应子数组 [5,9,3,7] 。可以重新排列为等差数列 [3,5,7,9] 。
示例 2: 输入: nums = [-12,-9,-3,-12,-6,15,20,-25,-20,-15,-10],
l = [0,1,6,4,8,7], r = [4,4,9,7,9,10]
输出: [false,true,false,false,true,true]
提示: n == nums.length
    m == l.length
    m == r.length
    2 <= n <= 500
    1 <= m <= 500
    0 <= l[i] < r[i] < n
    -105 <= nums[i] <= 105
```

- 解题思路

```
func checkArithmeticSubarrays(nums []int, l []int, r []int) []bool {
        res := make([]bool, 0)
        for i := 0; i < len(l); i++ {
                flag := true
                arr := make([]int, 0)
                arr = append(arr, nums[l[i]:r[i]+1]...)
                sort.Ints(arr)
                for j := 2; j < len(arr); j++ {
                        if arr[j]-arr[j-1] != arr[1]-arr[0] {
                                flag = false
                                break
                        }
                }
                res = append(res, flag)
        }
        return res
}
```

50.11 1631. 最小体力消耗路径 (4)

- 题目

```
你准备参加一场远足活动。给你一个二维rows x columns的地图heights，
其中heights[row][col]表示格子(row, col)的高度。
一开始你在最左上角的格子(0, 0)，且你希望去最右下角的格子(rows-1, columns-
↪1)（注意下标从 0 开始编号）。
你每次可以往 上，下，左，右四个方向之一移动，你想要找到耗费 体力 最小的一条路径。
一条路径耗费的 体力值是路径上相邻格子之间 高度差绝对值的 最大值决定的。
请你返回从左上角走到右下角的最小体力消耗值。
示例 1：输入：heights = [[1,2,2],[3,8,2],[5,3,5]] 输出：2
解释：路径 [1,3,5,3,5] 连续格子的差值绝对值最大为 2 。
这条路径比路径 [1,2,2,2,5] 更优，因为另一条路径差值最大值为 3 。
示例 2：输入：heights = [[1,2,3],[3,8,4],[5,3,5]] 输出：1
解释：路径 [1,2,3,4,5] 的相邻格子差值绝对值最大为 1 ，比路径 [1,3,5,3,5] 更优。
示例 3：输入：heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]␣
↪输出：0
解释：上图所示路径不需要消耗任何体力。
提示：rows == heights.length
columns == heights[i].length
1 <= rows, columns <= 100
1 <= heights[i][j] <= 106
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func minimumEffortPath(heights [][]int) int {
        n, m := len(heights), len(heights[0])
        left, right := 0, 1000000
        res := 0
        for left <= right {
                mid := left + (right-left)/2 // 二分枚举最大值
                queue := make([][2]int, 0)
                queue = append(queue, [2]int{0, 0})
                visited := make([][]bool, n)
                for i := 0; i < n; i++ {
                        visited[i] = make([]bool, m)
                }
                for len(queue) > 0 {
                        a, b := queue[0][0], queue[0][1]
                        queue = queue[1:]
```

(续下页)

(接上页)

```
                        for i := 0; i < 4; i++ {
                                x, y := a+dx[i], b+dy[i]
                                if 0 <= x && x < n && 0 <= y && y < m &&
                                        visited[x][y] == false && abs(heights[a][b]-
↪heights[x][y]) <= mid {
                                        queue = append(queue, [2]int{x, y})
                                        visited[x][y] = true
                                }
                        }
                }
                if visited[n-1][m-1] == true { // 缩小范围
                        res = mid
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func minimumEffortPath(heights [][]int) int {
        n, m := len(heights), len(heights[0])
        arr := make([][3]int, 0) // 相邻格子可以连一条边，高度差绝对值最为边的权值
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        index := i*m + j // 转化为一维坐标
                        if i > 0 {
                                value := abs(heights[i][j] - heights[i-1][j])      //␣
↪上到下
                                arr = append(arr, [3]int{index - m, index, value}) //␣
↪之前坐标，当前坐标，绝对值
                        }
                        if j > 0 {
                                value := abs(heights[i][j] - heights[i][j-1])      //␣
↪左到右
```

(续下页)

(接上页)

```
                                arr = append(arr, [3]int{index - 1, index, value}) //之前坐标，当前坐标，绝对值
                        }
                }
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][2] < arr[j][2]
        })
        fa = Init(n * m)
        for i := 0; i < len(arr); i++ { // 从小到大枚举高度差绝对值
                a, b, c := arr[i][0], arr[i][1], arr[i][2]
                union(a, b)
                if query(0, n*m-1) == true {
                        return c
                }
        }
        return 0
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
```

(续下页)

(接上页)

```
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}

# 3
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func minimumEffortPath(heights [][]int) int {
        n, m := len(heights), len(heights[0])
        arr := make([][]int, n) //  高度差绝对值的最大值
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        arr[i][j] = math.MaxInt32 // 初始化为最大值
                }
        }
        arr[0][0] = 0
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [3]int{0, 0, 0})
        for {
                node := heap.Pop(&intHeap).([3]int)
                a, b, c := node[0], node[1], node[2]
                if a == n-1 && b == m-1 { // 返回结果
                        return c
                }
                if arr[a][b] < c { // 大于 高度差绝对值的最大值，跳过
                        continue
                }
                for i := 0; i < 4; i++ {
                        x, y := a+dx[i], b+dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m {
                                diff := max(c, abs(heights[x][y]-heights[a][b])) //␣
↪更新：去 高度差绝对值的最大值 的较大值
                                if diff < arr[x][y] {                            //␣
```

(续下页)

(接上页)

```
↪更新：加入堆，点会重复，通过更新arr[x][y]来过滤
                                        arr[x][y] = diff
                                        heap.Push(&intHeap, [3]int{x, y, diff})
                                }
                        }
                }
        }
        return 0
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type IntHeap [][3]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][2] < h[j][2]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([3]int))
}
```

(续下页)

(接上页)

```
func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 4
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func minimumEffortPath(heights [][]int) int {
        n, m := len(heights), len(heights[0])
        return sort.Search(1000000, func(target int) bool {
                queue := make([][2]int, 0)
                queue = append(queue, [2]int{0, 0})
                visited := make([][]bool, n)
                for i := 0; i < n; i++ {
                        visited[i] = make([]bool, m)
                }
                for len(queue) > 0 {
                        a, b := queue[0][0], queue[0][1]
                        queue = queue[1:]
                        if a == n-1 && b == m-1 {
                                return true
                        }
                        for i := 0; i < 4; i++ {
                                x, y := a+dx[i], b+dy[i]
                                if 0 <= x && x < n && 0 <= y && y < m &&
                                        visited[x][y] == false && abs(heights[a][b]-
↪heights[x][y]) <= target {
                                        queue = append(queue, [2]int{x, y})
                                        visited[x][y] = true
                                }
                        }
                }
                return false
        })
        return 0
}

func abs(a int) int {
        if a < 0 {
                return -a
```

(续下页)

(接上页)

```
        }
        return a
}
```

50.12 1637. 两点之间不包含任何点的最宽垂直面积 (1)

- 题目

```
给你 n 个二维平面上的点 points ，其中 points[i] = [xi, yi] ，
请你返回两点之间内部不包含任何点的 最宽垂直面积 的宽度。
垂直面积 的定义是固定宽度，而 y 轴上无限延伸的一块区域（也就是高度为无穷大）。
最宽垂直面积 为宽度最大的一个垂直面积。
请注意，垂直区域 边上 的点 不在 区域内。
示例 1：输入：points = [[8,7],[9,9],[7,4],[9,7]] 输出：1
解释：红色区域和蓝色区域都是最优区域。
示例 2：输入：points = [[3,1],[9,0],[1,0],[1,4],[5,3],[8,8]] 输出：3
提示：  n == points.length
    2 <= n <= 105
    points[i].length == 2
    0 <= xi, yi <= 109
```

- 解题思路

```
func maxWidthOfVerticalArea(points [][]int) int {
        sort.Slice(points, func(i, j int) bool {
                return points[i][0] < points[j][0]
        })
        res := 0
        for i := 1; i < len(points); i++ {
                res = max(res, points[i][0]-points[i-1][0])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

50.13 1638. 统计只差一个字符的子串数目 (3)

- 题目

```
给你两个字符串 s 和 t ，请你找出 s 中的非空子串的数目，这些子串满足替换 一个不同字符以后，
是 t 串的子串。换言之，请你找到 s 和 t 串中 恰好 只有一个字符不同的子字符串对的数目。
比方说， "computer" 和 "computation" 加粗部分只有一个字符不同：
'e'/'a' ，所以这一对子字符串会给答案加 1 。
请你返回满足上述条件的不同子字符串对数目。
一个 子字符串 是一个字符串中连续的字符。
示例 1：输入：s = "aba", t = "baba" 输出：6
解释：以下为只相差 1 个字符的 s 和 t 串的子字符串对：
("aba", "baba")
("aba", "baba")
("aba", "baba")
("aba", "baba")
("aba", "baba")
("aba", "baba")
加粗部分分别表示 s 和 t 串选出来的子字符串。
示例 2：输入：s = "ab", t = "bb" 输出：3
解释：以下为只相差 1 个字符的 s 和 t 串的子字符串对：
("ab", "bb")
("ab", "bb")
("ab", "bb")
加粗部分分别表示 s 和 t 串选出来的子字符串。
示例 3：输入：s = "a", t = "a" 输出：0
示例 4：输入：s = "abe", t = "bbc" 输出：10
提示：1 <= s.length, t.length <= 100
    s 和 t 都只包含小写英文字母。
```

- 解题思路

```
func countSubstrings(s string, t string) int {
        res := 0
        for i := 0; i < len(s); i++ {
                for j := 0; j < len(t); j++ {
                        diff := 0
                        for k := 0; i+k < len(s) && j+k < len(t); k++ {
                                if s[i+k] != t[j+k] {
                                        diff++
                                }
                                if diff > 1 {
                                        break
```

(续下页)

(接上页)

```
                                }
                                if diff == 1 {
                                        res++
                                }
                        }
                }
        }
        return res
}

# 2
func countSubstrings(s string, t string) int {
        res := 0
        for i := 0; i < len(s); i++ {
                for j := i + 1; j <= len(s); j++ {
                        length := j - i
                        newStr := s[i:j]
                        for k := 0; k <= len(t)-length; k++ {
                                b := t[k : k+length]
                                if compare(newStr, b) {
                                        res++
                                }
                        }
                }
        }
        return res
}

func compare(a, b string) bool {
        count := 0
        for i := 0; i < len(a); i++ {
                if a[i] != b[i] {
                        count++
                }
                if count >= 2 {
                        return false
                }
        }
        if count == 0 {
                return false
        }
        return true
}
```

(续下页)

(接上页)

```
# 3
func countSubstrings(s, t string) int {
        res := 0
        m, n := len(s), len(t)
        // dp以s[i]和t[j]结尾的所有子串对中，满足恰好只有一个字符不同的字符串对的数目
        dp := make([][]int, m+1)
        for i := 0; i <= m; i++ {
                dp[i] = make([]int, n+1)
        }
        // 以s[i]和t[j]为结尾的子串，最多有多少个连续相同的字符
        same := make([][]int, m+1)
        for i := 0; i <= m; i++ {
                same[i] = make([]int, n+1)
        }
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if s[i] == t[j] {
                                dp[i+1][j+1] = dp[i+1][j+1] + dp[i][j]
                                same[i+1][j+1] = same[i][j] + 1
                        } else {
                                dp[i+1][j+1] = same[i][j] + 1
                        }
                        res = res + dp[i+1][j+1]
                }
        }
        return res
}
```

50.14 1641. 统计字典序元音字符串的数目 (3)

- 题目

```
给你一个整数 n，请返回长度为 n 、仅由元音 (a, e, i, o, u) 组成且按 字典序排列␣
↪的字符串数量。
字符串 s 按 字典序排列 需要满足：对于所有有效的 i，
s[i] 在字母表中的位置总是与 s[i+1] 相同或在 s[i+1] 之前。
示例 1：输入：n = 1 输出：5
解释：仅由元音组成的 5 个字典序字符串为 ["a","e","i","o","u"]
示例 2：输入：n = 2 输出：15
解释：仅由元音组成的 15 个字典序字符串为
["aa","ae","ai","ao","au","ee","ei","eo","eu","ii","io","iu","oo","ou","uu"]
```

(续下页)

(接上页)

```
注意，"ea" 不是符合题意的字符串，因为 'e' 在字母表中的位置比 'a' 靠后
示例 3：输入：n = 33 输出：66045
提示：1 <= n <= 50
```

- 解题思路

```
func countVowelStrings(n int) int {
        dp := make([][5]int, n+1)
        dp[0][0], dp[0][1], dp[0][2], dp[0][3], dp[0][4] = 1, 1, 1, 1, 1
        for i := 1; i <= n; i++ {
                for j := 0; j < 5; j++ {
                        for k := 0; k <= j; k++ {
                                dp[i][j] = dp[i][j] + dp[i-1][k]
                        }
                }
        }
        res := 0
        for i := 0; i < 5; i++ {
                res = res + dp[n-1][i]
        }
        return res
}

# 2
func countVowelStrings(n int) int {
        dp := make([]int, 5)
        dp[0] = 1
        for i := 0; i < n; i++ {
                for j := 1; j < 5; j++ {
                        dp[j] = dp[j] + dp[j-1]
                }
        }
        res := 0
        for i := 0; i < 5; i++ {
                res = res + dp[i]
        }
        return res
}

# 3
// 将n个小球放到m个盒子里，盒子可以空：C(n+m-1, m-1) m=5 => C(n+4,4)
// 在n+4中选择4个整数(4*3*2)
func countVowelStrings(n int) int {
        return (n + 4) * (n + 3) * (n + 2) * (n + 1) / 24
```

(续下页)

(接上页)

```
}
```

50.15 1642. 可以到达的最远建筑 (2)

- 题目

```
给你一个整数数组 heights ，表示建筑物的高度。另有一些砖块 bricks 和梯子 ladders 。
你从建筑物 0 开始旅程，不断向后面的建筑物移动，期间可能会用到砖块或梯子。
当从建筑物 i 移动到建筑物 i+1（下标 从 0 开始 ）时：
如果当前建筑物的高度 大于或等于 下一建筑物的高度，则不需要梯子或砖块
如果当前建筑的高度 小于 下一个建筑的高度，您可以使用 一架梯子 或 (h[i+1] - h[i])␣
↪个砖块
如果以最佳方式使用给定的梯子和砖块，返回你可以到达的最远建筑物的下标（下标 从 0 开始␣
↪）。
示例 1：输入：heights = [4,2,7,6,9,14,12], bricks = 5, ladders = 1 输出：4
解释：从建筑物 0 出发，你可以按此方案完成旅程：
- 不使用砖块或梯子到达建筑物 1 ，因为 4 >= 2
- 使用 5 个砖块到达建筑物 2 。你必须使用砖块或梯子，因为 2 < 7
- 不使用砖块或梯子到达建筑物 3 ，因为 7 >= 6
- 使用唯一的梯子到达建筑物 4 。你必须使用砖块或梯子，因为 6 < 9
无法越过建筑物 4 ，因为没有更多砖块或梯子。
示例 2：输入：heights = [4,12,2,7,3,18,20,3,19], bricks = 10, ladders = 2 输出：7
示例 3：输入：heights = [14,3,19,3], bricks = 17, ladders = 0 输出：3
提示：1 <= heights.length <= 105
1 <= heights[i] <= 106
0 <= bricks <= 109
0 <= ladders <= heights.length
```

- 解题思路

```
func furthestBuilding(heights []int, bricks int, ladders int) int {
        if len(heights) <= 1 {
                return 0
        }
        intHeap := &IntHeap{}
        heap.Init(intHeap)
        need := 0
        for i := 1; i < len(heights); i++ {
                need = heights[i] - heights[i-1]
                if need <= 0 {
                        continue
                }
```

(续下页)

(接上页)

```
                heap.Push(intHeap, need)
                if intHeap.Len() > ladders {
                        need = heap.Pop(intHeap).(int)
                        if need > bricks {
                                return i - 1
                        }
                        bricks = bricks - need
                }
        }
        return len(heights) - 1
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func furthestBuilding(heights []int, bricks int, ladders int) int {
        left, right := 0, len(heights)
        res := 0
        for left <= right {
                mid := left + (right-left)/2
                if check(heights[0:mid], bricks, ladders) {
```

(续下页)

(接上页)

```
                        left = mid + 1
                        res = mid
                } else {
                        right = mid - 1
                }
        }
        return res - 1
}

func check(heights []int, bricks int, ladders int) bool {
        arr := make([]int, 0)
        for i := 1; i < len(heights); i++ {
                need := heights[i] - heights[i-1]
                if need > 0 {
                        arr = append(arr, need)
                }
        }
        sort.Ints(arr)
        i := 0
        for ; i < len(arr); i++ {
                if bricks-arr[i] >= 0 {
                        bricks = bricks - arr[i]
                        continue
                }
                if ladders > 0 {
                        ladders--
                        continue
                }
                break
        }
        return i == len(arr)
}
```

50.16 1647. 字符频次唯一的最小删除次数 (1)

- 题目

```
如果字符串 s 中 不存在 两个不同字符 频次 相同的情况，就称 s 是 优质字符串 。
给你一个字符串 s，返回使 s 成为 优质字符串 需要删除的 最小 字符数。
字符串中字符的 频次 是该字符在字符串中的出现次数。
例如，在字符串 "aab" 中，'a' 的频次是 2，而 'b' 的频次是 1 。
示例 1：输入：s = "aab" 输出：0
```

(续下页)

(接上页)

```
解释：s 已经是优质字符串。
示例 2：输入：s = "aaabbbcc" 输出：2
解释：可以删除两个 'b'，得到优质字符串 "aaabcc" 。
另一种方式是删除一个 'b' 和一个 'c'，得到优质字符串 "aaabbc" 。
示例 3：输入：s = "ceabaacb" 输出：2
解释：可以删除两个 'c' 得到优质字符串 "eabaab" 。
注意，只需要关注结果字符串中仍然存在的字符。（即，频次为 0 的字符会忽略不计。）
提示：1 <= s.length <= 10^5  s 仅含小写英文字母
```

- 解题思路

```
func minDeletions(s string) int {
        m := make(map[int]int)
        for i := 0; i < len(s); i++ {
                m[int(s[i])]++
        }
        arr := make([]int, 0)
        for _, v := range m {
                arr = append(arr, v)
        }
        sort.Ints(arr)
        M := make(map[int]bool)
        res := 0
        for i := len(arr) - 1; i >= 0; i-- {
                if M[arr[i]] == false {
                        M[arr[i]] = true
                        continue
                }
                j := arr[i]
                for j >= 0 {
                        if M[j] == false {
                                M[j] = true
                                res = res + arr[i] - j
                                break
                        }
                        j--
                }
                if j == -1 {
                        res = res + arr[i]
                }
        }
        return res
}
```

50.17 1648. 销售价值减少的颜色球 (2)

- 题目

```
你有一些球的库存inventory，里面包含着不同颜色的球。一个顾客想要任意颜色␣
↪总数为orders的球。
这位顾客有一种特殊的方式衡量球的价值：每个球的价值是目前剩下的同色球的数目。
比方说还剩下6个黄球，那么顾客买第一个黄球的时候该黄球的价值为6。
这笔交易以后，只剩下5个黄球了，所以下一个黄球的价值为5（也就是球的价值随着顾客购买同色球是递减的）
给你整数数组inventory，其中inventory[i]表示第i种颜色球一开始的数目。
同时给你整数orders，表示顾客总共想买的球数目。你可以按照 任意顺序卖球。
请你返回卖了 orders个球以后 最大总价值之和。
由于答案可能会很大，请你返回答案对 109+ 7取余数的结果。
示例 1：输入：inventory = [2,5], orders = 4 输出：14
解释：卖 1 个第一种颜色的球（价值为 2 ），卖 3 个第二种颜色的球（价值为 5 + 4 + 3）。
最大总和为 2 + 5 + 4 + 3 = 14 。
示例 2：输入：inventory = [3,5], orders = 6 输出：19
解释：卖 2 个第一种颜色的球（价值为 3 + 2），卖 4 个第二种颜色的球（价值为 5 + 4 + 3␣
↪+ 2）。
最大总和为 3 + 2 + 5 + 4 + 3 + 2 = 19 。
示例 3：输入：inventory = [2,8,4,10,6], orders = 20 输出：110
示例 4：输入：inventory = [1000000000], orders = 1000000000 输出：21
解释：卖 1000000000 次第一种颜色的球，总价值为 500000000500000000 。
500000000500000000 对 109 + 7 取余为 21 。
提示：1 <= inventory.length <= 105
1 <= inventory[i] <= 109
1 <= orders <= min(sum(inventory[i]), 109)
```

- 解题思路

```
func maxProfit(inventory []int, orders int) int {
        inventory = append(inventory, 0) // 避免第一个数特判
        sort.Ints(inventory)
        n := len(inventory)
        res := 0
        // 每次把当前数减到前一个数
        for i := n - 1; i >= 1; i-- {
                if orders <= 0 {
                        break
                }
                total := (n - i) * (inventory[i] - inventory[i-1])
                if total <= orders { // 够用
                        sum := (inventory[i-1] + 1 + inventory[i]) * (inventory[i] -␣
↪inventory[i-1]) / 2 * (n - i)
```

(续下页)

(接上页)

```
                        res = (res + sum) % 1000000007
                        orders = orders - total
                } else { // 不够用
                        a, b := orders/(n-i), orders%(n-i)
                        start := inventory[i] - a + 1
                        sum := (start+inventory[i])*a/2*(n-i) + b*(start-1)
                        res = (res + sum) % 1000000007
                        orders = 0
                }
        }
        return res
}

# 2
func maxProfit(inventory []int, orders int) int {
        left, right := 0, math.MaxInt32
        target := 0
        for left <= right {
                target = left + (right-left)/2
                sum := 0
                count := 0
                for i := 0; i < len(inventory); i++ {
                        if inventory[i] >= target {
                                count++
                                sum = sum + (inventory[i] - target)
                        }
                }
                if sum > orders { // 过小
                        left = target + 1
                } else if sum+count <= orders { // 过小
                        right = target - 1
                } else {
                        break
                }
        }
        res := 0
        temp := 0
        for i := 0; i < len(inventory); i++ {
                if inventory[i] > target {
                        res = (res + getCount(target+1, inventory[i])) % 1000000007
                        temp = temp + inventory[i] - target
                }
        }
```

(续下页)

(接上页)

```
        return (res + (orders-temp)*target) % 1000000007
}

func getCount(a, b int) int {
        return (a + b) * (b - a + 1) / 2
}
```

50.18 1653. 使字符串平衡的最少删除次数 (4)

- 题目

```
给你一个字符串s，它仅包含字符'a' 和'b'。
你可以删除s中任意数目的字符，使得s 平衡。
我们称s平衡的当不存在下标对(i,j)满足i < j 且s[i] = 'b'同时s[j]= 'a'。
请你返回使 s平衡的 最少删除次数。
示例 1：输入：s = "aababbab" 输出：2
解释：你可以选择以下任意一种方案：
下标从 0 开始，删除第 2 和第 6 个字符 ("aababbab" -> "aaabbb") ，
下标从 0 开始，删除第 3 和第 6 个字符 ("aababbab" -> "aabbbb") 。
示例 2：输入：s = "bbaaaaabb" 输出：2
解释：唯一的最优解是删除最前面两个字符。
提示：1 <= s.length <= 105
s[i]要么是'a' 要么是'b'。
```

- 解题思路

```
func minimumDeletions(s string) int {
        n := len(s)
        dpA := make([]int, n)
        dpB := make([]int, n)
        if s[0] == 'a' {
                dpA[0] = 1
        }
        for i := 1; i < n; i++ {
                if s[i] == 'a' {
                        dpA[i] = dpA[i-1] + 1
                } else {
                        dpA[i] = dpA[i-1]
                }
        }
        if s[n-1] == 'b' {
                dpB[n-1] = 1
```

(续下页)

(接上页)

```
        }
        for i := n - 2; i >= 0; i-- {
                if s[i] == 'b' {
                        dpB[i] = dpB[i+1] + 1
                } else {
                        dpB[i] = dpB[i+1]
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                res = max(res, dpA[i]+dpB[i])
        }
        return n - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minimumDeletions(s string) int {
        res := 0
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                if s[i] == 'b' {
                        stack = append(stack, 'b')
                } else {
                        // 遇到'ba'则删掉，并次数+1
                        if len(stack) > 0 {
                                res++
                                stack = stack[:len(stack)-1]
                        }
                }
        }
        return res
}

# 3
func minimumDeletions(s string) int {
        aCount := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(s); i++ {
                if s[i] == 'a' {
                        aCount = aCount + 1
                }
        }
        res := aCount
        bCount := 0
        for i := 0; i < len(s); i++ {
                if s[i] == 'a' {
                        aCount--
                } else {
                        bCount++
                }
                // 要删除：前面b的个数+后面a的个数
                res = min(res, aCount+bCount)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func minimumDeletions(s string) int {
        n := len(s)
        dp := make([][2]int, n+1)
        // dp[n][0]以a结尾需要删除的次数
        // dp[n][1]以b结尾需要删除的次数
        for i := 0; i < n; i++ {
                if s[i] == 'a' {
                        dp[i+1][0] = dp[i][0]
                        dp[i+1][1] = dp[i][1] + 1
                } else {
                        dp[i+1][0] = dp[i][0] + 1
                        dp[i+1][1] = min(dp[i][0], dp[i][1])
                }
        }
        return min(dp[n][0], dp[n][1])
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

50.19 1654. 到家的最少跳跃次数 (2)

- 题目

```
有一只跳蚤的家在数轴上的位置x处。请你帮助它从位置0出发，到达它的家。
跳蚤跳跃的规则如下：
它可以 往前 跳恰好 a个位置（即往右跳）。
它可以 往后跳恰好 b个位置（即往左跳）。
它不能 连续 往后跳 2 次。
它不能跳到任何forbidden数组中的位置。
跳蚤可以往前跳 超过它的家的位置，但是它 不能跳到负整数的位置。
给你一个整数数组forbidden，其中forbidden[i]是跳蚤不能跳到的位置，同时给你整数a，b和x，
请你返回跳蚤到家的最少跳跃次数。
如果没有恰好到达 x的可行方案，请你返回 -1 。
示例 1：输入：forbidden = [14,4,18,1,15], a = 3, b = 15, x = 9 输出：3
解释：往前跳 3 次（0 -> 3 -> 6 -> 9），跳蚤就到家了。
示例 2：输入：forbidden = [8,3,16,6,12,20], a = 15, b = 13, x = 11 输出：-1
示例 3：输入：forbidden = [1,6,2,14,5,17,4], a = 16, b = 9, x = 7 输出：2
解释：往前跳一次（0 -> 16），然后往回跳一次（16 -> 7），跳蚤就到家了。
提示：1 <= forbidden.length <= 1000
1 <= a, b, forbidden[i] <= 2000
0 <= x <= 2000
forbidden中所有位置互不相同。
位置x不在 forbidden中。
```

- 解题思路

```
func minimumJumps(forbidden []int, a int, b int, x int) int {
        m := make([]bool, 6001)
        for i := 0; i < len(forbidden); i++ {
                m[forbidden[i]] = true
        }
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{0, 0})
```

(续下页)

(接上页)

```
        res := -1
        for len(queue) > 0 {
                length := len(queue)
                res++
                for i := 0; i < length; i++ {
                        index, dir := queue[i][0], queue[i][1]
                        if index == x {
                                return res
                        }
                        if dir == 0 && index-b > 0 && m[index-b] == false { // 向后跳-
↪b
                                m[index-b] = true
                                queue = append(queue, [2]int{index - b, 1})
                        }
                        if index+a < len(m) && m[index+a] == false { // 向前跳+a
                                m[index+a] = true
                                queue = append(queue, [2]int{index + a, 0})
                        }
                }
                queue = queue[length:]
        }
        return -1
}

# 2
var m []bool

func minimumJumps(forbidden []int, a int, b int, x int) int {
        m = make([]bool, 6001)
        for i := 0; i < len(forbidden); i++ {
                m[forbidden[i]] = true
        }
        return dfs(0, 0, a, b, x)
}

func dfs(index int, dir int, a int, b int, x int) int {
        if index == x {
                return 0
        }
        res := -1
        if index+a < len(m) && m[index+a] == false { // 向前跳+a
                m[index+a] = true
                res = dfs(index+a, 0, a, b, x)
```

(续下页)

(接上页)

```
                if res != -1 {
                        return res + 1
                }
        }
        if dir == 0 && index-b > 0 && m[index-b] == false { // 向后跳-b
                res = dfs(index-b, 1, a, b, x)
                if res != -1 {
                        return res + 1
                }
        }
        return res
}
```

50.20 1657. 确定两个字符串是否接近 (1)

- 题目

```
如果可以使用以下操作从一个字符串得到另一个字符串，则认为两个字符串 接近 ：
操作 1：交换任意两个 现有 字符。
例如，abcde -> aecdb
操作 2：将一个 现有 字符的每次出现转换为另一个 现有␣
↪字符，并对另一个字符执行相同的操作。
例如，aacabb -> bbcbaa（所有 a 转化为 b ，而所有的 b 转换为 a ）
你可以根据需要对任意一个字符串多次使用这两种操作。
给你两个字符串，word1 和 word2 。如果 word1 和 word2 接近 ，就返回 true ；否则，返回␣
↪false 。
示例 1：输入：word1 = "abc", word2 = "bca" 输出：true
解释：2 次操作从 word1 获得 word2 。
执行操作 1："abc" -> "acb"
执行操作 1："acb" -> "bca"
示例 2：输入：word1 = "a", word2 = "aa" 输出：false
解释：不管执行多少次操作，都无法从 word1 得到 word2 ，反之亦然。
示例 3：输入：word1 = "cabbba", word2 = "abbccc" 输出：true
解释：3 次操作从 word1 获得 word2 。
执行操作 1："cabbba" -> "caabbb"
执行操作 2："caabbb" -> "baaccc"
执行操作 2："baaccc" -> "abbccc"
示例 4：输入：word1 = "cabbba", word2 = "aabbss" 输出：false
解释：不管执行多少次操作，都无法从 word1 得到 word2 ，反之亦然。
提示：1 <= word1.length, word2.length <= 105
word1 和 word2 仅包含小写英文字母
```

- 解题思路

```
func closeStrings(word1 string, word2 string) bool {
        if len(word1) != len(word2) {
                return false
        }
        arr1 := make([]int, 26)
        arr2 := make([]int, 26)
        m1 := make(map[uint8]bool)
        m2 := make(map[uint8]bool)
        for i := 0; i < len(word1); i++ {
                arr1[word1[i]-'a']++
                arr2[word2[i]-'a']++
                m1[word1[i]-'a'] = true
                m2[word2[i]-'a'] = true
        }
        for key := range m1 {
                if m2[key] != true {
                        return false
                }
        }
        sort.Ints(arr1)
        sort.Ints(arr2)
        for i := 0; i < 26; i++ {
                if arr1[i] != arr2[i] {
                        return false
                }
        }
        return true
}
```

50.21 1658. 将 x 减到 0 的最小操作数 (2)

- 题目

```
给你一个整数数组 nums 和一个整数 x 。每一次操作时，你应当移除数组 nums␣
↪最左边或最右边的元素，
然后从 x 中减去该元素的值。请注意，需要 修改 数组以供接下来的操作使用。
如果可以将 x恰好 减到0 ，返回 最小操作数 ；否则，返回 -1 。
示例 1：输入：nums = [1,1,4,2,3], x = 5 输出：2
解释：最佳解决方案是移除后两个元素，将 x 减到 0 。
示例 2：输入：nums = [5,6,7,8,9], x = 4 输出：-1
示例 3：输入：nums = [3,2,20,1,1,3], x = 10 输出：5
解释：最佳解决方案是移除后三个元素和前两个元素（总共 5 次操作），将 x 减到 0 。
```

(续下页)

(接上页)

```
提示：1 <= nums.length <= 105
1 <= nums[i] <= 104
1 <= x <= 109
```

- 解题思路

```
func minOperations(nums []int, x int) int {
        n := len(nums)
        res := n + 1
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
        }
        if sum < x {
                return -1
        }
        if sum == x {
                return n
        }
        left := make([]int, n)
        right := make([]int, n)
        mLeft := make(map[int]int)
        mRight := make(map[int]int)
        left[0] = nums[0]
        mLeft[nums[0]] = 0
        right[n-1] = nums[n-1]
        mRight[nums[n-1]] = n - 1
        if left[0] == x || right[n-1] == x {
                return 1
        }
        for i := 1; i < n; i++ {
                left[i] = left[i-1] + nums[i]
                mLeft[left[i]] = i
                if left[i] == x {
                        res = min(res, i+1)
                }
        }
        for i := n - 2; i >= 0; i-- {
                right[i] = right[i+1] + nums[i]
                mRight[right[i]] = i
                if right[i] == x {
                        res = min(res, n-i)
                }
        }
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                left := left[i]
                if index, ok := mRight[x-left]; ok && i < index {
                        target := n - (index - i - 1)
                        res = min(res, target)
                }
        }
        if res == n+1 {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minOperations(nums []int, x int) int {
        n := len(nums)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
        }
        target := sum - x
        left := 0
        right := 0
        res := -1
        cur := 0
        // 和为sum-x的最长连续子数组
        for left < n {
                if right < n {
                        cur = cur + nums[right]
                        right++
                }
                for cur > target && left < n {
                        cur = cur - nums[left]
                        left++
                }
                if cur == target {
```

(续下页)

(接上页)

```
                        res = max(res, right-left)
                }
                if right == n {
                        left++
                }
        }
        if res == -1 {
                return -1
        }
        return n - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

50.22 1663. 具有给定数值的最小字符串 (2)

- 题目

```
小写字符 的 数值 是它在字母表中的位置（从 1 开始），因此 a 的数值为 1 ，b 的数值为 2␣
↪，
c 的数值为 3 ，以此类推。
字符串由若干小写字符组成，字符串的数值 为各字符的数值之和。
例如，字符串 "abe" 的数值等于 1 + 2 + 5 = 8 。
给你两个整数 n 和 k 。返回 长度 等于 n 且 数值 等于 k 的 字典序最小 的字符串。
注意，如果字符串 x 在字典排序中位于 y 之前，就认为 x 字典序比 y 小，有以下两种情况：
x 是 y 的一个前缀；
如果 i 是x[i] != y[i] 的第一个位置，且 x[i]在字母表中的位置比y[i]靠前。
示例 1：输入：n = 3, k = 27 输出："aay"
解释：字符串的数值为 1 + 1 + 25 = 27，它是数值满足要求且长度等于 3␣
↪字典序最小的字符串。
示例 2：输入：n = 5, k = 73 输出："aaszz"
提示：1 <= n <= 105
n <= k <= 26 * n
```

- 解题思路

```
func getSmallestString(n int, k int) string {
        res := ""
        k = k - n
        a := k / 25
        b := k % 25
        right := a
        var left, middle int
        if b == 0 {
                left = n - right
                middle = 0
        } else {
                left = n - right - 1
                middle = b
        }
        res = res + strings.Repeat("a", left)
        if middle > 0 {
                res = res + string('a'+middle)
        }
        res = res + strings.Repeat("z", right)
        return res
}

# 2
func getSmallestString(n int, k int) string {
        arr := make([]byte, n)
        k = k - n
        for i := n - 1; i >= 0; i-- {
                if k > 25 {
                        arr[i] = 'z'
                        k = k - 25
                } else {
                        arr[i] = byte('a' + k)
                        k = 0
                }
        }
        return string(arr)
}
```

50.23 1664. 生成平衡数组的方案数 (2)

- 题目

```
给你一个整数数组nums。你需要选择 恰好一个下标（下标从 0开始）并删除对应的元素。
请注意剩下元素的下标可能会因为删除操作而发生改变。
比方说，如果nums = [6,1,7,4,1]，那么：
选择删除下标 1 ，剩下的数组为nums = [6,7,4,1]。
选择删除下标2，剩下的数组为nums = [6,1,4,1]。
选择删除下标4，剩下的数组为nums = [6,1,7,4]。
如果一个数组满足奇数下标元素的和与偶数下标元素的和相等，该数组就是一个 平衡数组 。
请你返回删除操作后，剩下的数组nums是平衡数组 的方案数。
示例 1：输入：nums = [2,1,6,4] 输出：1
解释：删除下标 0 ：[1,6,4] -> 偶数元素下标为：1 + 4 = 5 。奇数元素下标为：6 。不平衡。
删除下标 1 ：[2,6,4] -> 偶数元素下标为：2 + 4 = 6 。奇数元素下标为：6 。平衡。
删除下标 2 ：[2,1,4] -> 偶数元素下标为：2 + 4 = 6 。奇数元素下标为：1 。不平衡。
删除下标 3 ：[2,1,6] -> 偶数元素下标为：2 + 6 = 8 。奇数元素下标为：1 。不平衡。
只有一种让剩余数组成为平衡数组的方案。
示例 2：输入：nums = [1,1,1] 输出：3
解释：你可以删除任意元素，剩余数组都是平衡数组。
示例 3：输入：nums = [1,2,3] 输出：0
解释：不管删除哪个元素，剩下数组都不是平衡数组。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 104
```

- 解题思路

```
func waysToMakeFair(nums []int) int {
        res := 0
        a := 0
        b := 0
        for i := 0; i < len(nums); i++ {
                if i%2 == 0 {
                        a = a + nums[i]
                } else {
                        b = b + nums[i]
                }
        }
        x, y := 0, 0
        for i := 0; i < len(nums); i++ {
                if i%2 == 0 {
                        a = a - nums[i]
                } else {
                        b = b - nums[i]
```

(续下页)

(接上页)

```
                }
                even := x + b
                odd := y + a
                if even == odd {
                        res++
                }
                if i%2 == 0 {
                        x = x + nums[i]
                } else {
                        y = y + nums[i]
                }
        }
        return res
}

# 2
func waysToMakeFair(nums []int) int {
        n := len(nums)
        dp := make([]int, n+1)
        for i := 0; i < n; i++ {
                if i%2 == 0 {
                        dp[i+1] = dp[i] + nums[i]
                } else {
                        dp[i+1] = dp[i] - nums[i]
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                // dp[i]表示索引i左边部分奇偶元素差值,
                // dp[n] - dp[i+1]表示索引i右边部分奇偶元素差值
                if dp[i] == dp[n]-dp[i+1] {
                        res++
                }
        }
        return res
}
```

50.24 1669. 合并两个链表 (2)

- 题目

```
给你两个链表list1 和list2，它们包含的元素分别为n 个和m 个。
请你将list1中第a个节点到第b个节点删除，并将list2接在被删除节点的位置。
下图中蓝色边和节点展示了操作后的结果：
请你返回结果链表的头指针。
示例 1：输入：list1 = [0,1,2,3,4,5], a = 3, b = 4, list2 = [1000000,1000001,1000002]
输出：[0,1,2,1000000,1000001,1000002,5]
解释：我们删除 list1 中第三和第四个节点，并将 list2␣
↪接在该位置。上图中蓝色的边和节点为答案链表。
示例 2：输入：list1 = [0,1,2,3,4,5,6], a = 2, b = 5, list2 = [1000000,1000001,1000002,
↪1000003,1000004]
输出：[0,1,1000000,1000001,1000002,1000003,1000004,6]
解释：上图中蓝色的边和节点为答案链表。
提示：3 <= list1.length <= 104
1 <= a <= b < list1.length - 1
1 <= list2.length <= 104
```

- 解题思路

```
func mergeInBetween(list1 *ListNode, a int, b int, list2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for i := 0; i < a; i++{
                temp.Next = list1
                temp = temp.Next
                list1 = list1.Next
        }
        for list2 != nil{
                temp.Next = list2
                temp = temp.Next
                list2 = list2.Next
        }
        for i := a; i <= b; i++{
                list1 = list1.Next
        }
        for list1 != nil{
                temp.Next = list1
                temp = temp.Next
                list1 = list1.Next
        }
        return  res.Next
```

(续下页)

(接上页)

```
}

# 2
func mergeInBetween(list1 *ListNode, a int, b int, list2 *ListNode) *ListNode {
        cur := list1
        for i := 0; i < a-1; i++ {
                cur = cur.Next
        }
        temp := cur.Next
        for i := 0; i < (b - a + 1); i++ {
                temp = temp.Next
        }
        cur.Next = list2
        for cur.Next != nil {
                cur = cur.Next
        }
        cur.Next = temp
        return list1
}
```

50.25 1670. 设计前中后队列 (1)

- 题目

```
请你设计一个队列，支持在前，中，后三个位置的 push和 pop操作。
请你完成FrontMiddleBack类：
FrontMiddleBack()初始化队列。
void pushFront(int val) 将val添加到队列的 最前面。
void pushMiddle(int val) 将val添加到队列的 正中间。
void pushBack(int val)将val添加到队里的 最后面。
int popFront()将 最前面 的元素从队列中删除并返回值，如果删除之前队列为空，那么返回 -
↪1。
int popMiddle() 将 正中间的元素从队列中删除并返回值，如果删除之前队列为空，那么返回 -
↪1。
int popBack() 将 最后面 的元素从队列中删除并返回值，如果删除之前队列为空，那么返回 -
↪1。
请注意当有两个中间位置的时候，选择靠前面的位置进行操作。比方说：
将 6添加到[1, 2, 3, 4, 5]的中间位置，结果数组为[1, 2, 6, 3, 4, 5]。
从[1, 2, 3, 4, 5, 6]的中间位置弹出元素，返回3，数组变为[1, 2, 4, 5, 6]。
示例 1：输入：
["FrontMiddleBackQueue", "pushFront", "pushBack", "pushMiddle",
"pushMiddle", "popFront", "popMiddle", "popMiddle", "popBack", "popFront"]
```

(续下页)

(接上页)

```
[[], [1], [2], [3], [4], [], [], [], [], []]
输出：[null, null, null, null, null, 1, 3, 4, 2, -1]
解释：FrontMiddleBackQueue q = new FrontMiddleBackQueue();
q.pushFront(1);   // [1]
q.pushBack(2);    // [1, 2]
q.pushMiddle(3);  // [1, 3, 2]
q.pushMiddle(4);  // [1, 4, 3, 2]
q.popFront();     // 返回 1 -> [4, 3, 2]
q.popMiddle();    // 返回 3 -> [4, 2]
q.popMiddle();    // 返回 4 -> [2]
q.popBack();      // 返回 2 -> []
q.popFront();     // 返回 -1 -> []（队列为空）
提示：1 <= val <= 109
最多调用1000次pushFront，pushMiddle，pushBack，popFront，popMiddle和popBack 。
```

- 解题思路

```go
type FrontMiddleBackQueue struct {
        arr []int
}

func Constructor() FrontMiddleBackQueue {
        return FrontMiddleBackQueue{}
}

func (this *FrontMiddleBackQueue) PushFront(val int) {
        this.arr = append([]int{val}, this.arr...)
}

func (this *FrontMiddleBackQueue) PushMiddle(val int) {
        mid := len(this.arr) / 2
        this.arr = append(this.arr[:mid], append([]int{val}, this.arr[mid:]...)...)
}

func (this *FrontMiddleBackQueue) PushBack(val int) {
        this.arr = append(this.arr, val)
}

func (this *FrontMiddleBackQueue) PopFront() int {
        var res int
        if len(this.arr) == 0 {
                return -1
        }
        res = this.arr[0]
```

(续下页)

(接上页)

```
        this.arr = this.arr[1:]
        return res
}

func (this *FrontMiddleBackQueue) PopMiddle() int {
        var res, mid int
        if len(this.arr) == 0 {
                return -1
        }
        if len(this.arr)%2 == 1 {
                mid = len(this.arr) / 2
        } else {
                mid = len(this.arr)/2 - 1
        }
        res = this.arr[mid]
        this.arr = append(this.arr[:mid], this.arr[mid+1:]...)
        return res
}

func (this *FrontMiddleBackQueue) PopBack() int {
        var res int
        if len(this.arr) == 0 {
                return -1
        }
        res = this.arr[len(this.arr)-1]
        this.arr = this.arr[:len(this.arr)-1]
        return res
}
```

50.26 1673. 找出最具竞争力的子序列 (1)

- 题目

```
给你一个整数数组 nums 和一个正整数 k ，返回长度为 k 且最具 竞争力 的 nums 子序列。
数组的子序列是从数组中删除一些元素（可能不删除元素）得到的序列。
在子序列a 和子序列b 第一个不相同的位置上，如果a中的数字小于 b 中对应的数字，
那么我们称子序列 a 比子序列 b（相同长度下）更具 竞争力 。
例如，[1,3,4] 比 [1,3,5] 更具竞争力，在第一个不相同的位置，也就是最后一个位置上，4␣
↪小于 5 。
示例 1：输入：nums = [3,5,2,6], k = 2 输出：[2,6]
解释：在所有可能的子序列集合 {[3,5], [3,2], [3,6], [5,2], [5,6], [2,6]} 中，
[2,6] 最具竞争力。
```

(续下页)

(接上页)

```
示例 2: 输入: nums = [2,4,3,3,5,4,9,6], k = 4 输出: [2,3,3,4]
提示: 1 <= nums.length <= 105
0 <= nums[i] <= 109
1 <= k <= nums.length
```

- 解题思路

```
func mostCompetitive(nums []int, k int) []int {
        stack := make([]int, 0)
        k = len(nums) - k
        for i := 0; i < len(nums); i++ {
                value := nums[i]
                //␣
↪栈顶元素打大于后面的元素，摘除栈顶元素（因为前面的更大，需要删除了才能变的最小）
                for len(stack) > 0 && stack[len(stack)-1] > value && k > 0 {
                        stack = stack[:len(stack)-1]
                        k--
                }
                stack = append(stack, value)
        }
        stack = stack[:len(stack)-k]
        return stack
}
```

50.27 1674. 使数组互补的最少操作次数 (1)

- 题目

```
给你一个长度为 偶数 n 的整数数组 nums 和一个整数 limit 。
每一次操作，你可以将 nums 中的任何整数替换为1到limit 之间的另一个整数。
如果对于所有下标 i（下标从 0 开始），nums[i] + nums[n - 1 - i]都等于同一个数，
则数组 nums 是 互补的 。
例如，数组 [1,2,3,4] 是互补的，因为对于所有下标i ，nums[i] + nums[n - 1 - i] = 5 。
返回使数组 互补 的 最少操作次数。
示例 1: 输入: nums = [1,2,4,3], limit = 4 输出: 1
解释: 经过 1 次操作，你可以将数组 nums 变成 [1,2,2,3]（加粗元素是变更的数字）:
nums[0] + nums[3] = 1 + 3 = 4.
nums[1] + nums[2] = 2 + 2 = 4.
nums[2] + nums[1] = 2 + 2 = 4.
nums[3] + nums[0] = 3 + 1 = 4.
对于每个 i ，nums[i] + nums[n-1-i] = 4 ，所以 nums 是互补的。
示例 2: 输入: nums = [1,2,2,1], limit = 2 输出: 2
```

(续下页)

(接上页)

```
解释：经过 2 次操作，你可以将数组 nums 变成 [2,2,2,2] 。你不能将任何数字变更为 3
↪，因为 3 > limit 。
示例 3：输入：nums = [1,2,1,2], limit = 2 输出：0
解释：nums 已经是互补的。
提示：n == nums.length
2 <= n<=105
1 <= nums[i]<= limit <=105
n 是偶数。
```

- 解题思路

```
func minMoves(nums []int, limit int) int {
        n := len(nums)
        arr := make([]int, 2*limit+2)
        for i := 0; i < n/2; i++ {
                a, b := nums[i], nums[n-1-i]
                // 1、首先[2,2*limit]都加2=>操作2次
                arr[2] = arr[2] + 2
                arr[2*limit+1] = arr[2*limit+1] - 2
                // 2、将[1+min(a,b),limit+max(a,b)] 减1=>操作1次
                arr[1+min(a, b)] = arr[1+min(a, b)] - 1
                arr[limit+max(a, b)+1] = arr[limit+max(a, b)+1] + 1
                // 3、将[a+b]减1，目标值=>操作0次
                arr[a+b] = arr[a+b] - 1
                arr[a+b+1] = arr[a+b+1] + 1
        }
        res := n
        sum := 0
        for i := 2; i <= 2*limit; i++ {
                sum = sum + arr[i]
                if res > sum {
                        res = sum
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}
```

50.28 1679.K 和数对的最大数目 (3)

- 题目

```
给你一个整数数组 nums 和一个整数 k 。
每一步操作中，你需要从数组中选出和为 k 的两个整数，并将它们移出数组。
返回你可以对数组执行的最大操作数。
示例 1：输入：nums = [1,2,3,4], k = 5 输出：2
解释：开始时 nums = [1,2,3,4]：
- 移出 1 和 4 ，之后 nums = [2,3]
- 移出 2 和 3 ，之后 nums = []
不再有和为 5 的数对，因此最多执行 2 次操作。
示例 2：输入：nums = [3,1,3,4,3], k = 6 输出：1
解释：开始时 nums = [3,1,3,4,3]：
- 移出前两个 3 ，之后nums = [1,4,3]
不再有和为 6 的数对，因此最多执行 1 次操作。
提示：1 <= nums.length <= 10^5
1 <= nums[i] <= 10^9
1 <= k <= 10^9
```

- 解题思路

```
func maxOperations(nums []int, k int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                if m[k-nums[i]] > 0 {
                        res++
                        m[k-nums[i]]--
                } else {
                        m[nums[i]]++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func maxOperations(nums []int, k int) int {
        sort.Ints(nums)
        res := 0
        left := 0
        right := len(nums) - 1
        for left < right {
                sum := nums[left] + nums[right]
                if sum == k {
                        res++
                        left++
                        right--
                } else if sum > k {
                        right--
                } else {
                        left++
                }
        }
        return res
}

# 3
func maxOperations(nums []int, k int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for key := range m {
                res = res + min(m[key], m[k-key])
        }
        return res / 2
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

50.29 1680. 连接连续二进制数字 (2)

- 题目

```
给你一个整数n，请你将1到 n的二进制表示连接起来，
并返回连接结果对应的 十进制数字对 109+ 7取余的结果。
示例 1：输入：n = 1 输出：1
解释：二进制的 "1" 对应着十进制的 1 。
示例 2：输入：n = 3 输出：27
解释：二进制下，1，2 和 3 分别对应 "1" ，"10" 和 "11" 。
将它们依次连接，我们得到 "11011" ，对应着十进制的 27 。
示例 3：输入：n = 12 输出：505379714
解释：连接结果为 "110111001011101111000100110101011110O" 。
对应的十进制数字为 118505380540 。
对 10^9 + 7 取余后，结果为 505379714 。
提示：1 <= n <= 10^5
```

- 解题思路

```
package main

import "math/bits"

func main() {

}

func concatenatedBinary(n int) int {
        res := 0
        for i := 1; i <= n; i++ {
                length := bits.Len(uint(i))
                res = (res<<length + i) % 1000000007
        }
        return res
}

# 2
func concatenatedBinary(n int) int {
        res := 0
        length := 0
        for i := 1; i <= n; i++ {
                if i&(i-1) == 0 {
                        length++
                }
```

(续下页)

(接上页)

```
                res = (res<<length + i) % 1000000007
        }
        return res
}
```

50.30 1685. 有序数组中差绝对值之和 (2)

- 题目

```
给你一个 非递减有序整数数组nums。
请你建立并返回一个整数数组result，它跟nums长度相同，
且result[i]等于nums[i]与数组中所有其他元素差的绝对值之和。
换句话说，result[i]等于sum(|nums[i]-nums[j]|) ，
其中0 <= j < nums.length 且j != i（下标从 0 开始）。
示例 1：输入：nums = [2,3,5] 输出：[4,3,5]
解释：假设数组下标从 0 开始，那么
result[0] = |2-2| + |2-3| + |2-5| = 0 + 1 + 3 = 4,
result[1] = |3-2| + |3-3| + |3-5| = 1 + 0 + 2 = 3,
result[2] = |5-2| + |5-3| + |5-5| = 3 + 2 + 0 = 5。
示例 2：输入：nums = [1,4,6,8,10] 输出：[24,15,13,15,21]
提示：2 <= nums.length <= 105
1 <= nums[i] <= nums[i + 1] <= 104
```

- 解题思路

```
func getSumAbsoluteDifferences(nums []int) []int {
        n := len(nums)
        res := make([]int, 0)
        right := 0 // 右边和
        left := 0  // 左边和
        for i := 1; i < n; i++ {
                right = right + (nums[i] - nums[0])
        }
        res = append(res, right)
        for i := 1; i < n; i++ {
                diff := nums[i] - nums[i-1]
                left = left + diff*i
                right = right - diff*(n-i)
                res = append(res, left+right)
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func getSumAbsoluteDifferences(nums []int) []int {
        n := len(nums)
        res := make([]int, 0)
        arr := make([]int, 0)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
                arr = append(arr, sum)
        }
        res = append(res, sum-n*nums[0])
        for i := 1; i < n; i++ {
                left := nums[i]*i - arr[i-1]              // 左边和
                right := (sum - arr[i]) - nums[i]*(n-1-i) // 右边和
                res = append(res, left+right)
        }
        return res
}
```

50.31 1686. 石子游戏 VI(1)

- 题目

```
Alice 和Bob 轮流玩一个游戏，Alice 先手。
一堆石子里总共有n个石子，轮到某个玩家时，他可以移出一个石子并得到这个石子的价值。
Alice 和 Bob 对石子价值有不一样的的评判标准。双方都知道对方的评判标准。
给你两个长度为 n的整数数组aliceValues 和bobValues。
aliceValues[i] 和bobValues[i]分别表示 Alice 和 Bob 认为第i个石子的价值。
所有石子都被取完后，得分较高的人为胜者。
如果两个玩家得分相同，那么为平局。两位玩家都会采用 最优策略进行游戏。
请你推断游戏的结果，用如下的方式表示：
如果 Alice 赢，返回1。
如果 Bob 赢，返回-1。
如果游戏平局，返回0。
示例 1：输入：aliceValues = [1,3], bobValues = [2,1] 输出：1
解释：如果 Alice 拿石子 1 （下标从 0开始），那么 Alice 可以得到 3 分。
Bob 只能选择石子 0 ，得到 2 分。
Alice 获胜。
示例 2：输入：aliceValues = [1,2], bobValues = [3,1] 输出：0
解释：Alice 拿石子 0 ， Bob 拿石子 1 ，他们得分都为 1 分。
打平。
```

(续下页)

(接上页)

```
示例 3：输入：aliceValues = [2,4,3], bobValues = [1,6,7] 输出：-1
解释：不管 Alice 怎么操作，Bob 都可以得到比 Alice 更高的得分。
比方说，Alice 拿石子 1 ，Bob 拿石子 2 ， Alice 拿石子 0 ，Alice 会得到 6 分而 Bob
↪得分为 7 分。
Bob 会获胜。
提示：n == aliceValues.length == bobValues.length
1 <= n <= 105
1 <= aliceValues[i], bobValues[i] <= 100
```

- 解题思路

```
func stoneGameVI(aliceValues []int, bobValues []int) int {
        arr := make([][2]int, len(aliceValues))
        for i := 0; i < len(arr); i++ {
                arr[i] = [2]int{i, aliceValues[i] + bobValues[i]}
        }
        // 贪心策略：将两组石头的价值相加，每次取价值最大的那一组
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] > arr[j][1]
        })
        a, b := 0, 0
        for i := 0; i < len(arr); i++ {
                if i%2 == 0 {
                        a = a + aliceValues[arr[i][0]]
                } else {
                        b = b + bobValues[arr[i][0]]
                }
        }
        if a == b {
                return 0
        } else if a > b {
                return 1
        }
        return -1
}
```

50.32 1689. 十-二进制数的最少数目 (1)

- 题目

```
如果一个十进制数字不含任何前导零，且每一位上的数字不是 0 就是 1 ，那么该数字就是一个␣
↪十-二进制数 。
例如，101 和 1100 都是 十-二进制数，而 112 和 3001 不是。
给你一个表示十进制整数的字符串 n ，返回和为 n 的 十-二进制数 的最少数目。
示例 1：输入：n = "32" 输出：3
解释：10 + 11 + 11 = 32
示例 2：输入：n = "82734" 输出：8
示例 3：输入：n = "27346209830709182346" 输出：9
提示：1 <= n.length <= 105
n 仅由数字组成
n 不含任何前导零并总是表示正整数
```

- 解题思路

```
func minPartitions(n string) int {
        res := 0
        for i := 0; i < len(n); i++ {
                value := int(n[i] - '0')
                if value > res {
                        res = value
                }
        }
        return res
}
```

50.33 1690. 石子游戏 VII(4)

- 题目

```
石子游戏中，爱丽丝和鲍勃轮流进行自己的回合，爱丽丝先开始 。
有 n 块石子排成一排。每个玩家的回合中，可以从行中 移除 最左边的石头或最右边的石头，
并获得与该行中剩余石头值之 和 相等的得分。当没有石头可移除时，得分较高者获胜。
鲍勃发现他总是输掉游戏（可怜的鲍勃，他总是输），所以他决定尽力 减小得分的差值 。
爱丽丝的目标是最大限度地 扩大得分的差值 。
给你一个整数数组stones ，其中 stones[i] 表示 从左边开始 的第 i 个石头的值，
如果爱丽丝和鲍勃都 发挥出最佳水平 ，请返回他们 得分的差值 。
示例 1：输入：stones = [5,3,1,4,2] 输出：6
解释：
- 爱丽丝移除 2 ，得分 5 + 3 + 1 + 4 = 13 。游戏情况：爱丽丝 = 13 ，鲍勃 = 0 ，石子 =␣
```

(续下页)

(接上页)

```
↪[5,3,1,4] 。
- 鲍勃移除 5 ，得分 3 + 1 + 4 = 8 。游戏情况：爱丽丝 = 13 ，鲍勃 = 8 ，石子 = [3,1,4]␣
↪。
- 爱丽丝移除 3 ，得分 1 + 4 = 5 。游戏情况：爱丽丝 = 18 ，鲍勃 = 8 ，石子 = [1,4] 。
- 鲍勃移除 1 ，得分 4 。游戏情况：爱丽丝 = 18 ，鲍勃 = 12 ，石子 = [4] 。
- 爱丽丝移除 4 ，得分 0 。游戏情况：爱丽丝 = 18 ，鲍勃 = 12 ，石子 = [] 。
得分的差值 18 - 12 = 6 。
示例 2：输入：stones = [7,90,5,1,100,10,10,2] 输出：122
提示：n == stones.length
2 <= n <= 1000
1 <= stones[i] <= 1000
```

- 解题思路

```
func stoneGameVII(stones []int) int {
        n := len(stones)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + stones[i]
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for j := 2; j <= n; j++ {
                for i := 0; i+j <= n; i++ {
                        left := arr[i+j] - arr[i+1] - dp[i+1][i+j-1] // 左边得分-得分
                        right := arr[i+j-1] - arr[i] - dp[i][i+j-2]  // 右边得分-得分
                        dp[i][i+j-1] = max(left, right)
                }
        }
        return dp[0][n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var dp [][]int
var arr []int
```

(续下页)

(接上页)

```
func stoneGameVII(stones []int) int {
        n := len(stones)
        arr = make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + stones[i]
        }
        dp = make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        if i == j {
                                continue
                        }
                        dp[i][j] = -1
                }
        }
        dfs(0, n-1)
        return dp[0][n-1]
}

func dfs(i, j int) int {
        if dp[i][j] != -1 {
                return dp[i][j]
        }
        left := arr[j+1] - arr[i+1] - dfs(i+1, j)
        right := arr[j] - arr[i] - dfs(i, j-1)
        dp[i][j] = max(left, right)
        return dp[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func stoneGameVII(stones []int) int {
        n := len(stones)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                arr[i+1] = arr[i] + stones[i]
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        left := arr[j+1] - arr[i+1] - dp[i+1][j] // 左边得分-得分
                        right := arr[j] - arr[i] - dp[i][j-1]    // 右边得分-得分
                        dp[i][j] = max(left, right)
                }
        }
        return dp[0][n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func stoneGameVII(stones []int) int {
        n := len(stones)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + stones[i]
        }
        dp := make([]int, n)
        for i := n - 2; i >= 0; i-- {
                sum := stones[i]
                for j := i + 1; j < n; j++ {
                        sum = sum + stones[j]
                        left := sum - stones[i] - dp[j]    // 左边得分-得分
                        right := sum - stones[j] - dp[j-1] // 右边得分-得分
                        dp[j] = max(left, right)
                }
        }
        return dp[n-1]
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

50.34 1695. 删除子数组的最大得分 (1)

- 题目

```
给你一个正整数数组 nums ，请你从中删除一个含有 若干不同元素 的子数组。
删除子数组的 得分 就是子数组各元素之 和 。
返回 只删除一个 子数组可获得的 最大得分 。
如果数组 b 是数组 a 的一个连续子序列，即如果它等于 a[l],a[l+1],...,a[r] ，那么它就是a
↪的一个子数组。
示例 1：输入：nums = [4,2,4,5,6] 输出：17
解释：最优子数组是 [2,4,5,6]
示例 2：输入：nums = [5,2,1,2,5,2,1,2,5] 输出：8
解释：最优子数组是 [5,2,1] 或 [1,2,5]
提示：1 <= nums.length <= 105
1 <= nums[i] <= 104
```

- 解题思路

```
func maximumUniqueSubarray(nums []int) int {
        res := 0
        sum := 0
        m := make(map[int]int)
        left := 0
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
                sum = sum + nums[i]
                for m[nums[i]] > 1 {
                        m[nums[left]]--
                        sum = sum - nums[left]
                        left++
                }
                if sum > res {
                        res = sum
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

50.35 1696. 跳跃游戏 VI(4)

- 题目

```
给你一个下标从 0 开始的整数数组 nums和一个整数 k。
一开始你在下标0处。每一步，你最多可以往前跳k步，但你不能跳出数组的边界。
也就是说，你可以从下标i跳到[i + 1， min(n - 1, i + k)]包含 两个端点的任意位置。
你的目标是到达数组最后一个位置（下标为 n - 1），你的 得分为经过的所有数字之和。
请你返回你能得到的 最大得分。
示例 1：输入：nums = [1,-1,-2,4,-7,3], k = 2 输出：7
解释：你可以选择子序列 [1,-1,4,3] （上面加粗的数字），和为 7 。
示例 2：输入：nums = [10,-5,-2,4,0,3], k = 3 输出：17
解释：你可以选择子序列 [10,4,3] （上面加粗数字），和为 17 。
示例 3：输入：nums = [1,-5,-20,4,-1,3,-6,-3], k = 2 输出：0
提示：1 <= nums.length, k <= 105
-104<= nums[i]<= 104
```

- 解题思路

```
func maxResult(nums []int, k int) int {
        n := len(nums)
        dp := make([]int, n)
        if k > n {
                k = n
        }
        dp[0] = nums[0]
        maxValue := nums[0]
        for i := 1; i < len(nums); i++ {
                if i <= k { // 在前k个，dp[i] = maxValue + nums[i]
                        dp[i] = maxValue + nums[i]
                        maxValue = max(maxValue, dp[i])
                } else {
                        if maxValue == dp[i-1-k] { // 需要重新找maxValue
                                maxValue = getMaxValue(dp[i-k : i])
                        }
                        dp[i] = maxValue + nums[i]
                        maxValue = max(maxValue, dp[i])
                }
        }
```

(续下页)

(接上页)

```
        return dp[n-1]
}

func getMaxValue(arr []int) int {
        maxValue := arr[0]
        for i := 0; i < len(arr); i++ {
                if arr[i] > maxValue {
                        maxValue = arr[i]
                }
        }
        return maxValue
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxResult(nums []int, k int) int {
        n := len(nums)
        dp := make([]int, n)
        if k > n {
                k = n
        }
        dp[0] = nums[0]
        maxValue := nums[0]
        maxIndex := 0
        for i := 1; i < len(nums); i++ {
                if i <= k { // 在前k个, dp[i] = maxValue + nums[i]
                        dp[i] = maxValue + nums[i]
                        if dp[i] >= maxValue {
                                maxValue = dp[i]
                                maxIndex = i
                        }
                } else {
                        if i-k > maxIndex {
                                maxValue = dp[maxIndex+1]
                                for j := maxIndex + 1; j < i; j++ {
                                        if dp[j] >= maxValue {
                                                maxValue = dp[j]
```

(续下页)

(接上页)

```
                                                maxIndex = j
                                        }
                                }
                        }
                        dp[i] = maxValue + nums[i]
                        if dp[i] >= maxValue {
                                maxValue = dp[i]
                                maxIndex = i
                        }
                }
        }
        return dp[n-1]
}

# 3
func maxResult(nums []int, k int) int {
        n := len(nums)
        if k > n {
                k = n
        }
        res := nums[0]
        stack := make([][2]int, 0)
        stack = append(stack, [2]int{0, nums[0]})
        for i := 1; i < len(nums); i++ {
                if stack[0][0] < i-k {
                        stack = stack[1:]
                }
                res = stack[0][1] + nums[i]
                for len(stack) > 0 && stack[len(stack)-1][1] < res {
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, [2]int{i, res})
        }
        return res
}

# 4
func maxResult(nums []int, k int) int {
        n := len(nums)
        if k > n {
                k = n
        }
        res := nums[0]
```

(续下页)

(接上页)

```
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [2]int{0, nums[0]})
        for i := 1; i < len(nums); i++ {
                for i-intHeap[0][0] > k { // 不满足删除
                        heap.Pop(&intHeap)
                }
                res = intHeap[0][1] + nums[i]
                heap.Push(&intHeap, [2]int{i, res})
        }
        return res
}

type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][1] > h[j][1]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

CHAPTER 51

1601-1700-Hard

51.1 1606. 找到处理最多请求的服务器 (2)

- 题目

```
你有 k个服务器，编号为 0到 k-1，它们可以同时处理多个请求组。
每个服务器有无穷的计算能力但是 不能同时处理超过一个请求。请求分配到服务器的规则如下：
第i（序号从 0 开始）个请求到达。
如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。
如果第(i % k)个服务器空闲，那么对应服务器会处理该请求。
否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0␣
↪个服务器开始继续找下一个空闲的服务器）。
比方说，如果第 i个服务器在忙，那么会查看第 (i+1)个服务器，第 (i+2)个服务器等等。
给你一个 严格递增的正整数数组arrival，表示第i个任务的到达时间，和另一个数组load，
其中load[i]表示第i个请求的工作量（也就是服务器完成它所需要的时间）。
你的任务是找到␣
↪最繁忙的服务器。最繁忙定义为一个服务器处理的请求数是所有服务器里最多的。
请你返回包含所有最繁忙服务器序号的列表，你可以以任意顺序返回这个列表。
示例 1：输入：k = 3, arrival = [1,2,3,4,5], load = [5,2,3,3,3]  输出：[1]
解释：所有服务器一开始都是空闲的。
前 3 个请求分别由前 3 台服务器依次处理。
请求 3 进来的时候，服务器 0 被占据，所以它呗安排到下一台空闲的服务器，也就是服务器 1␣
↪。
请求 4 进来的时候，由于所有服务器都被占据，该请求被舍弃。
服务器 0 和 2 分别都处理了一个请求，服务器 1 处理了两个请求。所以服务器 1␣
```

(续下页)

(接上页)

```
↪是最忙的服务器。
示例 2：输入：k = 3, arrival = [1,2,3,4], load = [1,2,1,2] 输出：[0]
解释：前 3 个请求分别被前 3 个服务器处理。
请求 3 进来，由于服务器 0 空闲，它被服务器 0 处理。
服务器 0 处理了两个请求，服务器 1 和 2 分别处理了一个请求。所以服务器 0␣
↪是最忙的服务器。
示例 3：输入：k = 3, arrival = [1,2,3], load = [10,12,11] 输出：[0,1,2]
解释：每个服务器分别处理了一个请求，所以它们都是最忙的服务器。
示例 4：输入：k = 3, arrival = [1,2,3,4,8,9,10], load = [5,2,10,3,1,2,2] 输出：[1]
示例 5：输入：k = 1, arrival = [1], load = [1] 输出：[0]
提示：1 <= k <= 105
1 <= arrival.length, load.length <= 105
arrival.length == load.length
1 <= arrival[i], load[i] <= 109
arrival保证 严格递增。
```

- 解题思路

```
func busiestServers(k int, arrival []int, load []int) []int {
        n := len(arrival)
        res := make([]int, 0)
        freeHeap := &mixHeap{isBig: false} // 小根堆：下标小的优先处理
        busyHeap := &mixHeap{isBig: false} // 小根堆：  空闲时间小，早结束
        for i := 0; i < k; i++ {
                freeHeap.push([]int{i})
        }
        arr := make([]int, k)
        for i := 0; i < n; i++ {
                start := arrival[i]
                // busy堆执行完出队列
                for busyHeap.Len() > 0 && busyHeap.Top()[0] <= start {
                        id := busyHeap.pop()[1]
                        // i+((id-i)%k+k)%k 表示下一个循环的编号
                        freeHeap.push([]int{i + ((id-i)%k+k)%k}) //␣
↪入free堆：注意时间处理，负数取模后要取正
                }
                // 选择一个执行，如果为空舍弃：条件2
                if freeHeap.Len() > 0 {
                        id := freeHeap.pop()[0] % k
                        busyHeap.push([]int{start + load[i], id}) // 结束时间+id
                        arr[id]++
                }
        }
        var maxValue int
```

(续下页)

(接上页)

```
        for i := 0; i < len(arr); i++ {
                if arr[i] > maxValue {
                        maxValue = arr[i]
                        res = []int{i}
                } else if arr[i] == maxValue {
                        res = append(res, i)
                }
        }
        return res
}

type mixHeap struct {
        arr   [][]int
        isBig bool
}

func (m *mixHeap) Len() int {
        return len(m.arr)
}

func (m *mixHeap) Swap(i, j int) {
        m.arr[i], m.arr[j] = m.arr[j], m.arr[i]
}

func (m *mixHeap) Less(i, j int) bool {
        if m.isBig {
                return m.arr[i][0] > m.arr[j][0] // 大根堆
        }
        return m.arr[i][0] < m.arr[j][0] // 小根堆
}

func (m *mixHeap) Push(x interface{}) {
        m.arr = append(m.arr, x.([]int))
}

func (m *mixHeap) Pop() interface{} {
        value := (m.arr)[len(m.arr)-1]
        m.arr = (m.arr)[:len(m.arr)-1]
        return value
}

func (m *mixHeap) push(x []int) {
        heap.Push(m, x)
```

(续下页)

(接上页)

```
}

func (m *mixHeap) pop() []int {
        return heap.Pop(m).([]int)
}

func (m *mixHeap) Top() []int {
        if m.Len() > 0 {
                return m.arr[0]
        }
        return nil
}

# 2
func busiestServers(k int, arrival []int, load []int) []int {
        n := len(arrival)
        res := make([]int, 0)
        arr := make([]int, k)
        free := make([]int, k)
        for i := 0; i < k; i++ {
                free[i] = i
        }
        busyHeap := make(IntHeap, 0)
        heap.Init(&busyHeap)
        for i := 0; i < n; i++ {
                start := arrival[i]
                // busy堆执行完出队列
                for busyHeap.Len() > 0 && busyHeap[0][0] <= start {
                        // free插入该下标：插入有序数组后有序
                        id := busyHeap[0][1]
                        index := sort.SearchInts(free, id)
                        free = append(free[:index], append([]int{id}, free[index:]...
↪)...)
                        heap.Pop(&busyHeap)
                }
                if len(free) == 0 {
                        continue
                }
                id := sort.SearchInts(free, i%k)
                if id == len(free) {
                        id = 0
                }
                arr[free[id]]++
```

(续下页)

(接上页)

```
                heap.Push(&busyHeap, []int{start + load[i], free[id]})
                free = append(free[:id], free[id+1:]...) // 删除该下标
        }
        var maxValue int
        for i := 0; i < len(arr); i++ {
                if arr[i] > maxValue {
                        maxValue = arr[i]
                        res = []int{i}
                } else if arr[i] == maxValue {
                        res = append(res, i)
                }
        }
        return res
}

type IntHeap [][]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][0] < h[j][0] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}
```

51.2 1611. 使整数变为 0 的最少操作次数 (3)

- 题目

```
给你一个整数 n，你需要重复执行多次下述操作将其转换为 0 ：
翻转 n 的二进制表示中最右侧位（第 0 位）。
如果第 (i-1) 位为 1 且从第 (i-2) 位到第 0 位都为 0，则翻转 n 的二进制表示中的第 i 位。
返回将 n 转换为 0 的最小操作次数。
示例 1：输入：n = 0 输出：0
示例 2：输入：n = 3 输出：2
解释：3 的二进制表示为 "11"
"11" -> "01" ，执行的是第 2 种操作，因为第 0 位为 1 。
"01" -> "00" ，执行的是第 1 种操作。
```

(续下页)

(接上页)

```
示例 3: 输入: n = 6 输出: 4
解释: 6 的二进制表示为 "110".
"110" -> "010" , 执行的是第 2 种操作, 因为第 1 位为 1 , 第 0 到 0 位为 0 。
"010" -> "011" , 执行的是第 1 种操作。
"011" -> "001" , 执行的是第 2 种操作, 因为第 0 位为 1 。
"001" -> "000" , 执行的是第 1 种操作。
示例 4: 输入: n = 9 输出: 14
示例 5: 输入: n = 333 输出: 393
提示: 0 <= n <= 109
```

- 解题思路

```
func minimumOneBitOperations(n int) int {
        // 依次将高位的1翻转为0
        // 操作2: 00..110000..00 => 00..010000..00
        // 操作2+操作1: 00..010000..00 => 00..011000..00
        // 操作2: 00..011000..00 => 00..001000..00
        res := 0
        if n == 0 {
                return 0
        }
        for i := 0; (1 << i) <= n; i++ {
                if (1<<i)&n > 0 { // 第i位>0
                        total := 1<<(1+i) - 1 // 把(1+i)位数变为100..00的次数
                        res = total - res     // 交替加减
                }
        }
        return res
}

# 2
func minimumOneBitOperations(n int) int {
        // 在一组数的编码中, 若任意两个相邻的代码只有一位二进制数不同
        // 则称这种编码为格雷码 (Gray Code)
        res := 0
        for n > 0 {
                res = res ^ n
                n = n / 2
        }
        return res
}

# 3
func minimumOneBitOperations(n int) int {
```

(续下页)

(接上页)

```
        // 依次将高位的1翻转为0
        // 操作2: 00..110000..00 => 00..010000..00
        // 操作2+操作1: 00..010000..00 => 00..011000..00
        // 操作2: 00..011000..00 => 00..001000..00
        res := 0
        if n == 0 {
                return 0
        }
        length := bits.Len(uint(n))
        flag := 1
        for i := 0; (1 << i) <= n; i++ {
                if (1<<(length-1-i))&n > 0 { // 第length-1-i位>0{
                        total := 1<<(length-i) - 1
                        res = res + total*flag
                        flag = -1 * flag
                }
        }
        return res
}
```

51.3 1649. 通过指令创建有序数组 (2)

- 题目

```
给你一个整数数组instructions，你需要根据instructions中的元素创建一个有序数组。
一开始你有一个空的数组nums，你需要从左到右遍历instructions中的元素，将它们依次插入nums数组中。
每一次插入操作的代价是以下两者的 较小值：
nums中 严格小于instructions[i]的数字数目。
nums中 严格大于instructions[i]的数字数目。
比方说，如果要将3 插入到nums = [1,2,3,5]，
那么插入操作的代价为min(2, 1) (元素1和2小于3，元素5大于3) ，插入后nums 变成[1,2,3,3,
↪5]。
请你返回将instructions中所有元素依次插入nums后的␣
↪总最小代价。由于答案会很大，请将它对109 + 7取余后返回。
示例 1: 输入: instructions = [1,5,6,2] 输出: 1
解释: 一开始 nums = [] 。
插入 1 ，代价为 min(0, 0) = 0 ，现在 nums = [1] 。
插入 5 ，代价为 min(1, 0) = 0 ，现在 nums = [1,5] 。
插入 6 ，代价为 min(2, 0) = 0 ，现在 nums = [1,5,6] 。
插入 2 ，代价为 min(1, 2) = 1 ，现在 nums = [1,2,5,6] 。
总代价为 0 + 0 + 0 + 1 = 1 。
示例 2:输入: instructions = [1,2,3,6,5,4] 输出: 3
```

(续下页)

(接上页)

```
解释：一开始 nums = [] 。
插入 1 ，代价为 min(0, 0) = 0 ，现在 nums = [1] 。
插入 2 ，代价为 min(1, 0) = 0 ，现在 nums = [1,2] 。
插入 3 ，代价为 min(2, 0) = 0 ，现在 nums = [1,2,3] 。
插入 6 ，代价为 min(3, 0) = 0 ，现在 nums = [1,2,3,6] 。
插入 5 ，代价为 min(3, 1) = 1 ，现在 nums = [1,2,3,5,6] 。
插入 4 ，代价为 min(3, 2) = 2 ，现在 nums = [1,2,3,4,5,6] 。
总代价为 0 + 0 + 0 + 0 + 1 + 2 = 3 。
示例 3：输入：instructions = [1,3,3,3,2,4,2,1,2] 输出：4
解释：一开始 nums = [] 。
插入 1 ，代价为 min(0, 0) = 0 ，现在 nums = [1] 。
插入 3 ，代价为 min(1, 0) = 0 ，现在 nums = [1,3] 。
插入 3 ，代价为 min(1, 0) = 0 ，现在 nums = [1,3,3] 。
插入 3 ，代价为 min(1, 0) = 0 ，现在 nums = [1,3,3,3] 。
插入 2 ，代价为 min(1, 3) = 1 ，现在 nums = [1,2,3,3,3] 。
插入 4 ，代价为 min(5, 0) = 0 ，现在 nums = [1,2,3,3,3,4] 。
插入 2 ，代价为 min(1, 4) = 1 ，现在 nums = [1,2,2,3,3,3,4] 。
插入 1 ，代价为 min(0, 6) = 0 ，现在 nums = [1,1,2,2,3,3,3,4] 。
插入 2 ，代价为 min(2, 4) = 2 ，现在 nums = [1,1,2,2,2,3,3,3,4] 。
总代价为 0 + 0 + 0 + 0 + 1 + 0 + 1 + 0 + 2 = 4 。
提示：1 <= instructions.length <= 105
1 <= instructions[i] <= 105
```

- 解题思路

```
var mod = 1000000007

func createSortedArray(instructions []int) int {
        res := 0
        c = make([]int, 100002)
        length = 100001
        for i := 0; i < len(instructions); i++ {
                value := instructions[i]
                upData(value, 1) // 次数+1
                left := getSum(value - 1)
                right := getSum(100000) - getSum(value)
                res = (res + min(left, right)) % mod
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}

var length int
var c []int // 树状数组

func lowBit(x int) int {
        return x & (-x)
}

// 单点修改
func upData(i, k int) { // 在i位置加上k
        for i <= length {
                c[i] = c[i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(i int) int {
        res := 0
        for i > 0 {
                res = res + c[i]
                i = i - lowBit(i)
        }
        return res
}

# 2
var mod = 1000000007

func createSortedArray(instructions []int) int {
        res := 0
        n := 100001
        arr = make([]int, n*4+1)
        for i := 0; i < len(instructions); i++ {
                x := instructions[i]
                left := query(0, 1, n, 1, x-1)  // 查询1~x-1
                right := query(0, 1, n, x+1, n) // 查询 x+1~n
                res = (res + min(left, right)) % mod
                update(0, 1, n, x) // 添加x
        }
```

(续下页)

(接上页)

```
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

var arr []int // 线段树

func update(id int, left, right, x int) {
        if left > x || right < x {
                return
        }
        arr[id]++
        if left == right {
                return
        }
        mid := left + (right-left)/2
        update(2*id+1, left, mid, x)    // 左节点
        update(2*id+2, mid+1, right, x) // 右节点
}

func query(id int, left, right, queryLeft, queryRight int) int {
        if left > queryRight || right < queryLeft {
                return 0
        }
        if queryLeft <= left && right <= queryRight {
                return arr[id]
        }
        mid := left + (right-left)/2
        return query(id*2+1, left, mid, queryLeft, queryRight) +
                query(id*2+2, mid+1, right, queryLeft, queryRight)
}
```

51.4 1655. 分配重复整数

51.4.1 题目

```
给你一个长度为n的整数数组nums，这个数组中至多有50个不同的值。
同时你有 m个顾客的订单 quantity，其中，整数quantity[i]是第i位顾客订单的数目。
请你判断是否能将 nums中的整数分配给这些顾客，且满足：
第i位顾客 恰好有quantity[i]个整数。
第i位顾客拿到的整数都是 相同的。
每位顾客都满足上述两个要求。
如果你可以分配 nums中的整数满足上面的要求，那么请返回true，否则返回 false。
示例 1：输入：nums = [1,2,3,4]，quantity = [2] 输出：false
解释：第 0 位顾客没办法得到两个相同的整数。
示例 2：输入：nums = [1,2,3,3]，quantity = [2] 输出：true
解释：第 0 位顾客得到 [3,3] 。整数 [1,2] 都没有被使用。
示例 3：输入：nums = [1,1,2,2]，quantity = [2,2] 输出：true
解释：第 0 位顾客得到 [1,1] ，第 1 位顾客得到 [2,2] 。
示例 4：输入：nums = [1,1,2,3]，quantity = [2,2] 输出：false
解释：尽管第 0 位顾客可以得到 [1,1] ，第 1 位顾客没法得到 2 个一样的整数。
示例 5：输入：nums = [1,1,1,1,1]，quantity = [2,3] 输出：true
解释：第 0 位顾客得到 [1,1] ，第 1 位顾客得到 [1,1,1] 。
提示：n == nums.length
1 <= n <= 105
1 <= nums[i] <= 1000
m == quantity.length
1 <= m <= 10
1 <= quantity[i] <= 105
nums中至多有50个不同的数字。
```

51.4.2 解题思路

51.5 1665. 完成所有任务的最少初始能量 (1)

- 题目

```
给你一个任务数组tasks ，其中tasks[i] = [actuali，minimumi]：
actuali是完成第 i个任务 需要耗费的实际能量。
minimumi是开始第 i个任务前需要达到的最低能量。
```

(续下页)

(接上页)

```
比方说，如果任务为[10, 12]且你当前的能量为11，那么你不能开始这个任务。
如果你当前的能量为13，你可以完成这个任务，且完成它后剩余能量为 3。
你可以按照 任意顺序完成任务。
请你返回完成所有任务的 最少初始能量。
示例 1：输入：tasks = [[1,2],[2,4],[4,8]] 输出：8
解释：一开始有 8 能量，我们按照如下顺序完成任务：
    - 完成第 3 个任务，剩余能量为 8 - 4 = 4 。
    - 完成第 2 个任务，剩余能量为 4 - 2 = 2 。
    - 完成第 1 个任务，剩余能量为 2 - 1 = 1 。
注意到尽管我们有能量剩余，但是如果一开始只有 7␣
↪能量是不能完成所有任务的，因为我们无法开始第 3 个任务。
示例 2：输入：tasks = [[1,3],[2,4],[10,11],[10,12],[8,9]] 输出：32
解释：一开始有 32 能量，我们按照如下顺序完成任务：
    - 完成第 1 个任务，剩余能量为 32 - 1 = 31 。
    - 完成第 2 个任务，剩余能量为 31 - 2 = 29 。
    - 完成第 3 个任务，剩余能量为 29 - 10 = 19 。
    - 完成第 4 个任务，剩余能量为 19 - 10 = 9 。
    - 完成第 5 个任务，剩余能量为 9 - 8 = 1 。
示例 3：输入：tasks = [[1,7],[2,8],[3,9],[4,10],[5,11],[6,12]] 输出：27
解释：一开始有 27 能量，我们按照如下顺序完成任务：
    - 完成第 5 个任务，剩余能量为 27 - 5 = 22 。
    - 完成第 2 个任务，剩余能量为 22 - 2 = 20 。
    - 完成第 3 个任务，剩余能量为 20 - 3 = 17 。
    - 完成第 1 个任务，剩余能量为 17 - 1 = 16 。
    - 完成第 4 个任务，剩余能量为 16 - 4 = 12 。
    - 完成第 6 个任务，剩余能量为 12 - 6 = 6 。
提示：1 <= tasks.length <= 105
1 <= actuali<= minimumi<= 104
```

- 解题思路

```
func minimumEffort(tasks [][]int) int {
        sort.Slice(tasks, func(i, j int) bool {
                return tasks[i][1]-tasks[i][0] > tasks[j][1]-tasks[j][0]
        })
        total := 0
        res := 0
        for i := 0; i < len(tasks); i++ {
                res = max(res, total+tasks[i][1])
                total = total + tasks[i][0]
        }
        return res
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

51.6 1671. 得到山形数组的最少删除次数 (1)

- 题目

```
我们定义arr是 山形数组当且仅当它满足:
arr.length >= 3
存在某个下标i (从 0 开始) 满足0 < i < arr.length - 1且:
arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
arr[i] > arr[i + 1] > ... > arr[arr.length - 1]
给你整数数组nums, 请你返回将 nums变成 山形状数组的最少删除次数。
示例 1: 输入: nums = [1,3,1] 输出: 0
解释: 数组本身就是山形数组, 所以我们不需要删除任何元素。
示例 2: 输入: nums = [2,1,1,5,6,2,3,1]输出: 3
解释: 一种方法是将下标为 0, 1 和 5 的元素删除, 剩余元素为 [1,5,6,3,1] , 是山形数组。
示例 3: 输入: nums = [4,3,2,1,1,2,3,1]输出: 4
提示: 输入: nums = [1,2,3,4,4,3,2,1]输出: 1
提示: 3 <= nums.length <= 1000
1 <= nums[i] <= 109
题目保证nums 删除一些元素后一定能得到山形数组。
```

- 解题思路

```
func minimumMountainRemovals(nums []int) int {
        n := len(nums)
        res := 0
        left := make([]int, n)
        right := make([]int, n)
        for i := 0; i < n; i++ {
                left[i] = 0
                for j := 0; j < i; j++ {
                        if nums[j] < nums[i] {
                                left[i] = max(left[j]+1, left[i])
                        }
                }
        }
```

(续下页)

(接上页)

```
        for i := n - 1; i >= 0; i-- {
                right[i] = 0
                for j := n - 1; j > i; j-- {
                        if nums[j] < nums[i] {
                                right[i] = max(right[j]+1, right[i])
                        }
                }
        }
        for i := 1; i < n-1; i++ {
                res = max(res, left[i]+right[i]+1)
        }
        return n - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

51.7 1675. 数组的最小偏移量 (1)

- 题目

```
给你一个由 n 个正整数组成的数组 nums 。
你可以对数组的任意元素执行任意次数的两类操作：
如果元素是 偶数 ，除以 2
例如，如果数组是 [1,2,3,4] ，那么你可以对最后一个元素执行此操作，使其变成 [1,2,3,2]
如果元素是 奇数 ，乘上 2
例如，如果数组是 [1,2,3,4] ，那么你可以对第一个元素执行此操作，使其变成 [2,2,3,4]
数组的 偏移量 是数组中任意两个元素之间的 最大差值 。
返回数组在执行某些操作之后可以拥有的 最小偏移量 。
示例 1：输入：nums = [1,2,3,4] 输出：1
解释：你可以将数组转换为 [1,2,3,2]，然后转换成 [2,2,3,2]，偏移量是 3 - 2 = 1
示例 2：输入：nums = [4,1,5,20,3] 输出：3
解释：两次操作后，你可以将数组转换为 [4,2,5,5,3]，偏移量是 5 - 2 = 3
示例 3：输入：nums = [2,10,8] 输出：3
提示：n == nums.length
2 <= n <= 105
1 <= nums[i] <= 109
```

- 解题思路

```
func minimumDeviation(nums []int) int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        minValue := math.MaxInt32
        for i := 0; i < len(nums); i++ {
                if nums[i]%2 == 1 { // 奇数x2=>变为偶数放入堆，统一处理为偶数
                        nums[i] = nums[i] * 2
                }
                heap.Push(&intHeap, nums[i])
                minValue = min(minValue, nums[i]) // 记录最小值
        }
        res := intHeap[0] - minValue
        for intHeap.Len() > 0 && intHeap[0]%2 == 0 { //  把最大偶数处理/2
                node := heap.Pop(&intHeap).(int)
                minValue = min(minValue, node/2)
                heap.Push(&intHeap, node/2)
                res = min(res, intHeap[0]-minValue) //␣
↪目标结果：将最大值除以2，用最大值减去最小值
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}
```

(续下页)

(接上页)

```
func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

51.8 1691. 堆叠长方体的最大高度 (1)

- 题目

```
给你 n 个长方体 cuboids ，
其中第 i 个长方体的长宽高表示为 cuboids[i] = [widthi, lengthi, heighti] （下标从 0␣
↪开始）。
请你从 cuboids 选出一个 子集 ，并将它们堆叠起来。
如果 widthi <= widthj 且 lengthi <= lengthj 且 heighti <= heightj ，
你就可以将长方体 i 堆叠在长方体 j␣
↪上。你可以通过旋转把长方体的长宽高重新排列，以将它放在另一个长方体上。
返回 堆叠长方体cuboids 可以得到的 最大高度 。
示例 1：输入：cuboids = [[50,45,20],[95,37,53],[45,23,12]] 输出：190
解释：第 1 个长方体放在底部，53x37 的一面朝下，高度为 95 。
第 0 个长方体放在中间，45x20 的一面朝下，高度为 50 。
第 2 个长方体放在上面，23x12 的一面朝下，高度为 45 。
总高度是 95 + 50 + 45 = 190 。
示例 2：输入：cuboids = [[38,25,45],[76,35,3]] 输出：76
解释：无法将任何长方体放在另一个上面。
选择第 1 个长方体然后旋转它，使 35x3 的一面朝下，其高度为 76 。
示例 3：输入：cuboids = [[7,11,17],[7,17,11],[11,7,17],[11,17,7],[17,7,11],[17,11,7]]
输出：102
解释：重新排列长方体后，可以看到所有长方体的尺寸都相同。
你可以把 11x7 的一面朝下，这样它们的高度就是 17 。
堆叠长方体的最大高度为 6 * 17 = 102 。
提示：n == cuboids.length
1 <= n <= 100
1 <= widthi, lengthi, heighti <= 100
```

- 解题思路

```
func maxHeight(cuboids [][]int) int {
        for i := 0; i < len(cuboids); i++ {
                arr := cuboids[i]
                sort.Ints(arr)
                cuboids[i] = arr
        }
        sort.Slice(cuboids, func(i, j int) bool {
                if cuboids[i][0] == cuboids[j][0] {
                        if cuboids[i][1] == cuboids[j][1] {
                                return cuboids[i][2] < cuboids[j][2]
                        }
                        return cuboids[i][1] < cuboids[j][1]
                }
                return cuboids[i][0] < cuboids[j][0]
        })
        n := len(cuboids)
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = cuboids[i][2]
        }
        res := dp[0]
        for i := 1; i < n; i++ {
                for j := 0; j < i; j++ {
                        if cuboids[i][0] >= cuboids[j][0] &&
                                cuboids[i][1] >= cuboids[j][1] &&
                                cuboids[i][2] >= cuboids[j][2] {
                                dp[i] = max(dp[i], dp[j]+cuboids[i][2])
                        }
                }
                res = max(res, dp[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

CHAPTER 52

1701-1800-Easy

52.1 1704. 判断字符串的两半是否相似 (1)

- 题目

```
给你一个偶数长度的字符串 s 。将其拆分成长度相同的两半，前一半为 a ，后一半为 b 。
两个字符串 相似 的前提是它们都含有相同数目的元音（'a', 'e', 'i', 'o', 'u', 'A', 'E',
↪'I', 'O', 'U'）。注意，s 可能同时含有大写和小写字母。
如果 a 和 b 相似，返回 true ；否则，返回 false 。
示例 1：输入：s = "book" 输出：true
解释：a = "bo" 且 b = "ok" 。a 中有 1 个元音，b 也有 1 个元音。所以，a 和 b 相似。
示例 2：输入：s = "textbook" 输出：false
解释：a = "text" 且 b = "book" 。a 中有 1 个元音，b 中有 2 个元音。因此，a 和 b␣
↪不相似。
注意，元音 o 在 b 中出现两次，记为 2 个。
示例 3：输入：s = "MerryChristmas" 输出：false
示例 4：输入：s = "AbCdEfGh" 输出：true
提示：2 <= s.length <= 1000
s.length 是偶数
s 由 大写和小写 字母组成
```

- 解题思路

```
func halvesAreAlike(s string) bool {
        s = strings.ToLower(s)
```

(续下页)

(接上页)

```
        total := 0
        for i := 0; i < len(s); i++ {
                if isVowel(s[i]) == true {
                        if i < len(s)/2 {
                                total++
                        } else {
                                total--
                        }
                }
        }
        return total == 0
}

func isVowel(b byte) bool {
        return b == 'a' || b == 'e' ||
                b == 'i' || b == 'o' || b == 'u'
}
```

52.2 1710. 卡车上的最大单元数 (1)

- 题目

```
请你将一些箱子装在 一辆卡车 上。
给你一个二维数组 boxTypes ，其中 boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]␣
↪:
numberOfBoxesi 是类型 i 的箱子的数量。
numberOfUnitsPerBoxi 是类型 i每个箱子可以装载的单元数量。
整数 truckSize 表示卡车上可以装载 箱子 的 最大数量 。
只要箱子数量不超过 truckSize ，你就可以选择任意箱子装到卡车上。
返回卡车可以装载单元 的 最大 总数。
示例 1: 输入: boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4 输出: 8
解释: 箱子的情况如下:
- 1 个第一类的箱子，里面含 3 个单元。
- 2 个第二类的箱子，每个里面含 2 个单元。
- 3 个第三类的箱子，每个里面含 1 个单元。
可以选择第一类和第二类的所有箱子，以及第三类的一个箱子。
单元总数 = (1 * 3) + (2 * 2) + (1 * 1) = 8
示例 2: 输入: boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10 输出: 91
提示: 1 <= boxTypes.length <= 1000
1 <= numberOfBoxesi, numberOfUnitsPerBoxi <= 1000
1 <= truckSize <= 106
```

- 解题思路

```
func maximumUnits(boxTypes [][]int, truckSize int) int {
        sort.Slice(boxTypes, func(i, j int) bool {
                return boxTypes[i][1] > boxTypes[j][1]
        })
        res := 0
        for i := 0; i < len(boxTypes); i++ {
                if boxTypes[i][0] <= truckSize {
                        res = res + boxTypes[i][0]*boxTypes[i][1]
                } else {
                        res = res + truckSize*boxTypes[i][1]
                        break
                }
                truckSize = truckSize - boxTypes[i][0]
        }
        return res
}
```

52.3 1716. 计算力扣银行的钱 (2)

- 题目

```
Hercy 想要为购买第一辆车存钱。他 每天 都往力扣银行里存钱。
最开始，他在周一的时候存入 1块钱。从周二到周日，他每天都比前一天多存入 1块钱。
在接下来每一个周一，他都会比 前一个周一 多存入 1块钱。
给你n，请你返回在第 n天结束的时候他在力扣银行总共存了多少块钱。
示例 1：输入：n = 4 输出：10
解释：第 4 天后，总额为 1 + 2 + 3 + 4 = 10 。
示例 2：输入：n = 10 输出：37
解释：第 10 天后，总额为 (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4) = 37 。
注意到第二个星期一，Hercy 存入 2 块钱。
示例 3：输入：n = 20 输出：96
解释：第 20 天后，总额为 (1 + 2 + 3 + 4 + 5 + 6 + 7) +
(2 + 3 + 4 + 5 + 6 + 7 + 8) + (3 + 4 + 5 + 6 + 7 + 8) = 96 。
提示：1 <= n <= 1000
```

- 解题思路

```
func totalMoney(n int) int {
        res := 0
        for i := 0; i < n; i++ {
                a, b := i/7, i%7+1
                res = res + a + b
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func totalMoney(n int) int {
        res := 0
        a, b := n/7, n%7
        res = res + (28+(a-1)*7+28)*a/2
        res = res + (a+1+a+b)*b/2
        return res
}
```

52.4 1720. 解码异或后的数组 (1)

- 题目

```
未知 整数数组 arr 由 n 个非负整数组成。
经编码后变为长度为 n - 1 的另一个整数数组 encoded ，其中 encoded[i] = arr[i] XOR
↪arr[i + 1] 。
例如，arr = [1,0,2,1] 经编码后得到 encoded = [1,2,3] 。
给你编码后的数组 encoded 和原数组 arr 的第一个元素 first（arr[0]）。
请解码返回原数组 arr 。可以证明答案存在并且是唯一的。
示例 1：输入：encoded = [1,2,3], first = 1 输出：[1,0,2,1]
解释：若 arr = [1,0,2,1] ，
那么 first = 1 且 encoded = [1 XOR 0, 0 XOR 2, 2 XOR 1] = [1,2,3]
示例 2：输入：encoded = [6,2,7,3], first = 4 输出：[4,2,0,7,4]
提示：2 <= n <= 104
encoded.length == n - 1
0 <= encoded[i] <= 105
0 <= first <= 105
```

- 解题思路

```
func decode(encoded []int, first int) []int {
        res := make([]int, 0)
        res = append(res, first)
        for i := 0; i < len(encoded); i++ {
                res = append(res, encoded[i]^res[len(res)-1])
        }
        return res
}
```

52.5 1725. 可以形成最大正方形的矩形数目 (1)

- 题目

```
给你一个数组 rectangles ，其中 rectangles[i] = [li, wi] 表示第 i 个矩形的长度为 li
↪、宽度为 wi 。
如果存在 k 同时满足 k <= li 和 k <= wi ，就可以将第 i 个矩形切成边长为 k 的正方形。
例如，矩形 [4,6] 可以切成边长最大为 4 的正方形。
设 maxLen 为可以从矩形数组rectangles 切分得到的 最大正方形 的边长。
返回可以切出边长为 maxLen 的正方形的矩形 数目 。
示例 1：输入：rectangles = [[5,8],[3,9],[5,12],[16,5]] 输出：3
解释：能从每个矩形中切出的最大正方形边长分别是 [5,3,5,5] 。
最大正方形的边长为 5 ，可以由 3 个矩形切分得到。
示例 2：输入：rectangles = [[2,3],[3,7],[4,3],[3,7]] 输出：3
提示：1 <= rectangles.length <= 1000
rectangles[i].length == 2
1 <= li, wi <= 109
li != wi
```

- 解题思路

```
func countGoodRectangles(rectangles [][]int) int {
        res := 0
        maxValue := 0
        for i := 0; i < len(rectangles); i++ {
                minValue := min(rectangles[i][0], rectangles[i][1])
                if minValue > maxValue {
                        res = 1
                        maxValue = minValue
                } else if minValue == maxValue {
                        res++
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}
```

52.6 1732. 找到最高海拔 (1)

- 题目

```
有一个自行车手打算进行一场公路骑行，这条路线总共由n + 1个不同海拔的点组成。
自行车手从海拔为 0的点0开始骑行。
给你一个长度为 n的整数数组gain,
其中 gain[i]是点 i和点 i + 1的 净海拔高度差 (0 <= i < n) 。请你返回 最高点的海拔 。
示例 1: 输入: gain = [-5,1,5,0,-7] 输出: 1
解释: 海拔高度依次为 [0,-5,-4,1,1,-6] 。最高海拔为 1 。
示例 2: 输入: gain = [-4,-3,-2,-1,4,3,2] 输出: 0
解释: 海拔高度依次为 [0,-4,-7,-9,-10,-6,-3,-1] 。最高海拔为 0 。
提示: n == gain.length
1 <= n <= 100
-100 <= gain[i] <= 100
```

- 解题思路

```
func largestAltitude(gain []int) int {
        res := 0
        sum := 0
        for i := 0; i < len(gain); i++ {
                sum = sum + gain[i]
                res = max(res, sum)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

52.7 1736. 替换隐藏数字得到的最晚时间 (1)

- 题目

```
给你一个字符串 time ，格式为 hh:mm（小时：分钟），其中某几位数字被隐藏（用 ? 表示）。
有效的时间为 00:00 到 23:59 之间的所有时间，包括 00:00 和 23:59 。
替换time 中隐藏的数字，返回你可以得到的最晚有效时间。
示例 1：输入：time = "2?:?0" 输出："23:50"
解释：以数字 '2' 开头的最晚一小时是 23 ，以 '0' 结尾的最晚一分钟是 50 。
示例 2：输入：time = "0?:3?" 输出："09:39"
示例 3：输入：time = "1?:22" 输出："19:22"
提示：time 的格式为 hh:mm
题目数据保证你可以由输入的字符串生成有效的时间
```

- 解题思路

```
func maximumTime(time string) string {
   res := []byte(time)
   for i := 0; i < len(res); i++ {
           if res[i] == '?' {
                   if i == 0 && (('0' <= time[1] && time[1] <= '3') || time[1] == '?
↪') {
                           res[i] = '2'
                   } else {
                           res[i] = '1'
                   }
                   if i == 1 {
                           if time[0] == '2' || res[0] == '2' {
                                   res[i] = '3'
                           } else {
                                   res[i] = '9'
                           }
                   }
                   if i == 3 {
                           res[i] = '5'
                   }
                   if i == 4 {
                           res[i] = '9'
                   }
           }
   }
   return string(res)
}
```

52.8 1742. 盒子中小球的最大数量 (1)

- 题目

```
你在一家生产小球的玩具厂工作，有 n 个小球，编号从 lowLimit 开始，
到 highLimit 结束（包括 lowLimit 和highLimit ，即n == highLimit - lowLimit + 1）。
另有无限数量的盒子，编号从 1 到 infinity 。
你的工作是将每个小球放入盒子中，其中盒子的编号应当等于小球编号上每位数字的和。
例如，编号 321 的小球应当放入编号 3 + 2 + 1 = 6 的盒子，
而编号 10 的小球应当放入编号 1 + 0 = 1 的盒子。
给你两个整数 lowLimit 和 highLimit ，返回放有最多小球的盒子中的小球数量。
如果有多个盒子都满足放有最多小球，只需返回其中任一盒子的小球数量。
示例 1：输入：lowLimit = 1, highLimit = 10 输出：2
解释：盒子编号：1 2 3 4 5 6 7 8 9 10 11 ...
小球数量：2 1 1 1 1 1 1 1 1 0  0  ...
编号 1 的盒子放有最多小球，小球数量为 2 。
示例 2：输入：lowLimit = 5, highLimit = 15 输出：2
解释：盒子编号：1 2 3 4 5 6 7 8 9 10 11 ...
小球数量：1 1 1 1 2 2 1 1 1 0  0  ...
编号 5 和 6 的盒子放有最多小球，每个盒子中的小球数量都是 2 。
示例 3：输入：lowLimit = 19, highLimit = 28 输出：2
解释：盒子编号：1 2 3 4 5 6 7 8 9 10 11 12 ...
小球数量：0 1 1 1 1 1 1 1 1 2  0  0  ...
编号 10 的盒子放有最多小球，小球数量为 2 。
提示：1 <= lowLimit <= highLimit <= 105
```

- 解题思路

```
func countBalls(lowLimit int, highLimit int) int {
        res := 0
        m := make(map[int]int)
        for i := lowLimit; i <= highLimit; i++ {
                sum := getSum(i)
                m[sum]++
                if m[sum] > res {
                        res = m[sum]
                }
        }
        return res
}

func getSum(i int) int {
        sum := 0
        for i > 0 {
```

(续下页)

(接上页)

```
                sum = sum + i%10
                i = i / 10
        }
        return sum
}
```

52.9 1748. 唯一元素的和 (1)

- 题目

```
给你一个整数数组nums。数组中唯一元素是那些只出现恰好一次的元素。
请你返回 nums中唯一元素的 和。
示例 1: 输入: nums = [1,2,3,2] 输出: 4
解释: 唯一元素为 [1,3] ，和为 4 。
示例 2: 输入: nums = [1,1,1,1,1] 输出: 0
解释: 没有唯一元素，和为 0 。
示例 3 : 输入: nums = [1,2,3,4,5] 输出: 15
解释: 唯一元素为 [1,2,3,4,5] ，和为 15 。
提示: 1 <= nums.length <= 100
1 <= nums[i] <= 100
```

- 解题思路

```
func sumOfUnique(nums []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for k, v := range m {
                if v == 1 {
                        res = res + k
                }
        }
        return res
}
```

52.10 1752. 检查数组是否经排序和轮转得到 (2)

- 题目

```
给你一个数组 nums 。nums 的源数组中，所有元素与 nums 相同，但按非递减顺序排列。
如果nums 能够由源数组轮转若干位置（包括 0 个位置）得到，则返回 true ；否则，返回␣
↪false 。
源数组中可能存在 重复项 。
注意：我们称数组 A 在轮转 x 个位置后得到长度相同的数组 B ，
当它们满足 A[i] == B[(i+x) % A.length] ，其中 % 为取余运算。
示例 1：输入：nums = [3,4,5,1,2] 输出：true
解释：[1,2,3,4,5] 为有序的源数组。
可以轮转 x = 3 个位置，使新数组从值为 3 的元素开始：[3,4,5,1,2] 。
示例 2：输入：nums = [2,1,3,4] 输出：false
解释：源数组无法经轮转得到 nums 。
示例 3：输入：nums = [1,2,3] 输出：true
解释：[1,2,3] 为有序的源数组。
可以轮转 x = 0 个位置（即不轮转）得到 nums 。
示例 4：输入：nums = [1,1,1] 输出：true
解释：[1,1,1] 为有序的源数组。
轮转任意个位置都可以得到 nums 。
示例 5：输入：nums = [2,1] 输出：true
解释：[1,2] 为有序的源数组。
可以轮转 x = 5 个位置，使新数组从值为 2 的元素开始：[2,1] 。
提示：1 <= nums.length <= 100
1 <= nums[i] <= 100
```

- 解题思路

```
func check(nums []int) bool {
        temp := make([]int, len(nums))
        copy(temp, nums)
        sort.Ints(temp)
        nums = append(nums, nums...)
        a := change(temp)
        b := change(nums)
        if strings.Contains(b, a) {
                return true
        }
        return false
}

func change(arr []int) string {
        res := ""
```

(续下页)

(接上页)

```
        for i := 0; i < len(arr); i++ {
                res = res + strconv.Itoa(arr[i]) + ","
        }
        res = strings.TrimRight(res, ",")
        return res
}

# 2
func check(nums []int) bool {
        count := 0
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] > nums[i+1] {
                        count++
                        if count > 1 {
                                return false
                        }
                        if nums[0] < nums[len(nums)-1] {
                                return false
                        }
                }
        }
        return true
}
```

52.11 1758. 生成交替二进制字符串的最少操作数 (2)

- 题目

```
给你一个仅由字符 '0' 和 '1' 组成的字符串 s 。
一步操作中，你可以将任一 '0' 变成 '1' ，或者将 '1' 变成 '0' 。
交替字符串␣
↪定义为：如果字符串中不存在相邻两个字符相等的情况，那么该字符串就是交替字符串。
例如，字符串 "010" 是交替字符串，而字符串 "0100" 不是。
返回使 s 变成 交替字符串 所需的 最少 操作数。
示例 1：输入：s = "0100" 输出：1
解释：如果将最后一个字符变为 '1' ，s 就变成 "0101" ，即符合交替字符串定义。
示例 2：输入：s = "10" 输出：0
解释：s 已经是交替字符串。
示例 3：输入：s = "1111" 输出：2
解释：需要 2 步操作得到 "0101" 或 "1010" 。
提示：1 <= s.length <= 104
s[i] 是 '0' 或 '1'
```

- 解题思路

```
func minOperations(s string) int {
        a, b := 0, 0
        for i := 0; i < len(s); i++ {
                if i%2 == 0 && s[i] == '1' { // a以偶0, 奇1
                        a++
                } else if i%2 == 1 && s[i] == '0' {
                        a++
                }
                if i%2 == 0 && s[i] == '0' { // b以偶1, 奇0
                        b++
                } else if i%2 == 1 && s[i] == '1' {
                        b++
                }
        }
        if a > b {
                return b
        }
        return a
}

# 2
func minOperations(s string) int {
        a, b := 0, 0
        for i := 0; i < len(s); i++ {
                if int(s[i]-'0')%2 != i%2 {
                        a++
                } else {
                        b++
                }
        }
        if a > b {
                return b
        }
        return a
}
```

52.12 1763. 最长的美好子字符串 (1)

- 题目

```
当一个字符串 s包含的每一种字母的大写和小写形式 同时出现在 s中，就称这个字符串s是 美好字符串。
比方说，"abABB"是美好字符串，因为'A' 和'a'同时出现了，且'B' 和'b'也同时出现了。
然而，"abA"不是美好字符串因为'b'出现了，而'B'没有出现。
给你一个字符串s，请你返回s最长的美好子字符串。
如果有多个答案，请你返回最早出现的一个。如果不存在美好子字符串，请你返回一个空字符串。
示例 1：输入：s = "YazaAay" 输出："aAa"
解释："aAa" 是一个美好字符串，因为这个子串中仅含一种字母，其小写形式 'a' 和大写形式 'A' 也同时出现了。
"aAa" 是最长的美好子字符串。
示例 2：输入：s = "Bb" 输出："Bb"
解释："Bb" 是美好字符串，因为 'B' 和 'b' 都出现了。整个字符串也是原字符串的子字符串。
示例 3：输入：s = "c" 输出：""
解释：没有美好子字符串。
示例 4：输入：s = "dDzeE" 输出："dD"
解释："dD" 和 "eE" 都是最长美好子字符串。
由于有多个美好子字符串，返回 "dD" ，因为它出现得最早。
提示：1 <= s.length <= 100
s只包含大写和小写英文字母。
```

- 解题思路

```go
func longestNiceSubstring(s string) string {
        res := ""
        for i := 0; i < len(s); i++ {
                for j := i + 1; j <= len(s); j++ {
                        if judge(s[i:j]) == true && len(s[i:j]) > len(res) {
                                res = s[i:j]
                        }
                }
        }
        return res
}

func judge(str string) bool {
        a := [26]int{}
        A := [26]int{}
        for i := 0; i < len(str); i++ {
                if 'a' <= str[i] && str[i] <= 'z' {
                        a[str[i]-'a']++
```

(续下页)

(接上页)

```
                } else {
                        A[str[i]-'A']++
                }
        }
        for i := 0; i < 26; i++ {
                if (a[i] > 0 && A[i] == 0) || (a[i] == 0 && A[i] > 0) {
                        return false
                }
        }
        return true
}
```

52.13 1768. 交替合并字符串 (2)

- 题目

```
给你两个字符串 word1 和 word2 。请你从 word1 开始，通过交替添加字母来合并字符串。
如果一个字符串比另一个字符串长，就将多出来的字母追加到合并后字符串的末尾。
返回 合并后的字符串 。
示例 1：输入：word1 = "abc", word2 = "pqr" 输出："apbqcr"
解释：字符串合并情况如下所示：
word1：  a   b   c
word2：    p   q   r
合并后：  a p b q c r
示例 2：输入：word1 = "ab", word2 = "pqrs" 输出："apbqrs"
解释：注意，word2 比 word1 长，"rs" 需要追加到合并后字符串的末尾。
word1：  a   b
word2：    p   q   r   s
合并后：  a p b q   r   s
示例 3：输入：word1 = "abcd", word2 = "pq" 输出："apbqcd"
解释：注意，word1 比 word2 长，"cd" 需要追加到合并后字符串的末尾。
word1：  a   b   c   d
word2：    p   q
合并后：  a p b q c   d
提示：1 <= word1.length, word2.length <= 100
word1 和 word2 由小写英文字母组成
```

- 解题思路

```
func mergeAlternately(word1 string, word2 string) string {
        res := ""
        i, j := 0, 0
```

(续下页)

(接上页)

```
        for i < len(word1) && j < len(word2) {
                res = res + string(word1[i])
                res = res + string(word2[j])
                i++
                j++
        }
        if i < len(word1) {
                res = res + string(word1[i:])
        }
        if j < len(word2) {
                res = res + string(word2[j:])
        }
        return res
}

# 2
func mergeAlternately(word1 string, word2 string) string {
        res := ""
        for i := 0; i < len(word1) || i < len(word2); i++ {
                if i < len(word1) {
                        res = res + string(word1[i])
                }
                if i < len(word2) {
                        res = res + string(word2[i])
                }
        }
        return res
}
```

52.14 1773. 统计匹配检索规则的物品数量 (1)

- 题目

```
给你一个数组 items ，其中items[i] = [typei, colori, namei] ，
描述第 i 件物品的类型、颜色以及名称。
另给你一条由两个字符串ruleKey 和 ruleValue 表示的检索规则。
如果第 i 件物品能满足下述条件之一，则认为该物品与给定的检索规则 匹配 ：
ruleKey == "type" 且 ruleValue == typei 。
ruleKey == "color" 且 ruleValue == colori 。
ruleKey == "name" 且 ruleValue == namei 。
统计并返回 匹配检索规则的物品数量 。
示例 1：输入：items = [["phone","blue","pixel"],["computer","silver","lenovo"],
```

(续下页)

(接上页)

```
["phone","gold","iphone"]], ruleKey = "color", ruleValue = "silver"
输出：1 解释：只有一件物品匹配检索规则，这件物品是 ["computer","silver","lenovo"] 。
示例 2：输入：items = [["phone","blue","pixel"],["computer","silver","phone"],
["phone","gold","iphone"]], ruleKey = "type", ruleValue = "phone"
输出：2 解释：只有两件物品匹配检索规则，这两件物品分别是 ["phone","blue","pixel"] 和 [
↪"phone","gold","iphone"] 。
注意，["computer","silver","phone"] 未匹配检索规则。
提示：1 <= items.length <= 104
1 <= typei.length, colori.length, namei.length, ruleValue.length <= 10
ruleKey 等于 "type"、"color" 或 "name"
所有字符串仅由小写字母组成
```

- 解题思路

```
func countMatches(items [][]string, ruleKey string, ruleValue string) int {
        res := 0
        for i := 0; i < len(items); i++ {
                if ruleKey == "type" && items[i][0] == ruleValue {
                        res++
                } else if ruleKey == "color" && items[i][1] == ruleValue {
                        res++
                } else if ruleKey == "name" && items[i][2] == ruleValue {
                        res++
                }
        }
        return res
}
```

52.15 1779. 找到最近的有相同 X 或 Y 坐标的点 (1)

- 题目

```
给你两个整数x 和y，表示你在一个笛卡尔坐标系下的(x, y)处。
同时，在同一个坐标系下给你一个数组points，其中points[i] = [ai, bi]表示在(ai,␣
↪bi)处有一个点。
当一个点与你所在的位置有相同的 x 坐标或者相同的 y 坐标时，我们称这个点是 有效的。
请返回距离你当前位置曼哈顿距离最近的有效点的下标（下标从 0 开始）。
如果有多个最近的有效点，请返回下标最小的一个。如果没有有效点，请返回-1。
两个点 (x1, y1)和 (x2, y2)之间的 曼哈顿距离为abs(x1 - x2) + abs(y1 - y2)。
示例 1：输入：x = 3, y = 4, points = [[1,2],[3,1],[2,4],[2,3],[4,4]] 输出：2
解释：所有点中，[3,1]，[2,4] 和 [4,4] 是有效点。
有效点中，[2,4] 和 [4,4] 距离你当前位置的曼哈顿距离最小，都为 1 。[2,4]␣
```

(续下页)

(接上页)

```
↪的下标最小，所以返回 2 。
示例 2：输入：x = 3, y = 4, points = [[3,4]] 输出：0
提示：答案可以与你当前所在位置坐标相同。
示例 3：输入：x = 3, y = 4, points = [[2,3]] 输出：-1
解释：没有有效点。
提示：1 <= points.length <= 104
points[i].length == 2
1 <= x, y, ai, bi <= 104
```

- 解题思路

```
func nearestValidPoint(x int, y int, points [][]int) int {
        res := -1
        maxValue := math.MaxInt32
        for i := 0; i < len(points); i++ {
                a, b := points[i][0], points[i][1]
                if a == x || b == y {
                        value := abs(a-x) + abs(b-y)
                        if value < maxValue {
                                res = i
                                maxValue = value
                        }
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

52.16 1784. 检查二进制字符串字段 (3)

- 题目

```
给你一个二进制字符串 s ，该字符串 不含前导零 。
如果 s 最多包含 一个由连续的 '1' 组成的字段 ，返回 true 。否则，返回 false 。
示例 1：输入：s = "1001" 输出：false
解释：字符串中的 1 没有形成一个连续字段。
```

(续下页)

(接上页)

```
示例 2: 输入: s = "110" 输出: true
提示: 1 <= s.length <= 100
s[i]为 '0' 或 '1'
s[0] 为 '1'
```

- 解题思路

```
func checkOnesSegment(s string) bool {
        flag := true
        for i := 0; i < len(s); i++ {
                if s[i] == '0' {
                        flag = false
                }
                if flag == false && s[i] == '1' {
                        return false
                }
        }
        return true
}

# 2
func checkOnesSegment(s string) bool {
        if strings.Contains(s, "01") {
                return false
        }
        return true
}

# 3
func checkOnesSegment(s string) bool {
        arr := strings.Split(s, "0")
        return len(arr) == 1
}
```

52.17 1790. 仅执行一次字符串交换能否使两个字符串相等 (1)

- 题目

```
给你长度相等的两个字符串 s1 和 s2 。
一次 字符串交换␣
↪操作的步骤如下: 选出某个字符串中的两个下标 (不必不同) , 并交换这两个下标所对应的字符。
如果对 其中一个字符串 执行 最多一次字符串交换 就可以使两个字符串相等, 返回 true␣
```

(续下页)

(接上页)

```
↪; 否则，返回 false 。
示例 1: 输入: s1 = "bank", s2 = "kanb" 输出: true
解释: 例如，交换 s2 中的第一个和最后一个字符可以得到 "bank"
示例 2: 输入: s1 = "attack", s2 = "defend" 输出: false
解释: 一次字符串交换无法使两个字符串相等
示例 3: 输入: s1 = "kelb", s2 = "kelb" 输出: true
解释: 两个字符串已经相等，所以不需要进行字符串交换
示例 4: 输入: s1 = "abcd", s2 = "dcba" 输出: false
提示: 1 <= s1.length, s2.length <= 100
s1.length == s2.length
s1 和 s2 仅由小写英文字母组成
```

- 解题思路

```
func areAlmostEqual(s1 string, s2 string) bool {
        if s1 == s2 {
                return true
        }
        if len(s1) != len(s2) {
                return false
        }
        arr := make([]int, 0)
        count := 0
        for i := 0; i < len(s1); i++ {
                if s1[i] != s2[i] {
                        arr = append(arr, i)
                        count++
                }
        }
        if count != 2 {
                return false
        }
        if s1[arr[0]] == s2[arr[1]] && s1[arr[1]] == s2[arr[0]] {
                return true
        }
        return false
}
```

52.18 1796. 字符串中第二大的数字 (1)

- 题目

```
给你一个混合字符串s，请你返回 s中 第二大 的数字，如果不存在第二大的数字，请你返回 -1。
混合字符串 由小写英文字母和数字组成。
示例 1：输入：s = "dfa12321afd" 输出：2
解释：出现在 s 中的数字包括 [1, 2, 3] 。第二大的数字是 2 。
示例 2：输入：s = "abc1111" 输出：-1
解释：出现在 s 中的数字只包含 [1] 。没有第二大的数字。
提示：1 <= s.length <= 500
s只包含小写英文字母和（或）数字。
```

- 解题思路

```
func secondHighest(s string) int {
        m := make(map[int]int)
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        m[int(s[i]-'0')]++
                }
        }
        count := 0
        for i := 9; i >= 0; i-- {
                if m[i] > 0 {
                        count++
                        if count == 2 {
                                return i
                        }
                }
        }
        return -1
}
```

52.19 1800. 最大升序子数组和 (2)

- 题目

```
给你一个正整数组成的数组 nums ，返回 nums 中一个 升序 子数组的最大可能元素和。
子数组是数组中的一个连续数字序列。
已知子数组 [numsl, numsl+1, ..., numsr-1, numsr] ，若对所有 i（l <= i < r），
numsi < numsi+1 都成立，则称这一子数组为 升序 子数组。注意，大小为 1 的子数组也视作␣
↪升序 子数组。
```

(续下页)

(接上页)

```
示例 1: 输入: nums = [10,20,30,5,10,50] 输出: 65
解释: [5,10,50] 是元素和最大的升序子数组, 最大元素和为 65 。
示例 2: 输入: nums = [10,20,30,40,50] 输出: 150
解释: [10,20,30,40,50] 是元素和最大的升序子数组, 最大元素和为 150 。
示例 3: 输入: nums = [12,17,15,13,10,11,12] 输出: 33
解释: [10,11,12] 是元素和最大的升序子数组, 最大元素和为 33 。
示例 4: 输入: nums = [100,10,1] 输出: 100
提示: 1 <= nums.length <= 100
1 <= nums[i] <= 100
```

- 解题思路

```
func maxAscendingSum(nums []int) int {
        res, sum := nums[0], nums[0]
        for i := 1; i < len(nums); i++ {
                if nums[i-1] < nums[i] {
                        sum = sum + nums[i]
                } else {
                        sum = nums[i]
                }
                res = max(res, sum)
        }
        return res
}

func max(x, y int) int {
        if x > y {
                return x
        }
        return y
}

# 2
func maxAscendingSum(nums []int) int {
        res := nums[0]
        dp := make([]int, len(nums))
        dp[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                if nums[i-1] < nums[i] {
                        dp[i] = dp[i-1] + nums[i]
                } else {
                        dp[i] = nums[i]
                }
                res = max(res, dp[i])
```

(续下页)

(接上页)

```
        }
        return res
}

func max(x, y int) int {
        if x > y {
                return x
        }
        return y
}
```

CHAPTER 53

1701-1800-Medium

53.1 1701. 平均等待时间 (1)

- 题目

```
有一个餐厅，只有一位厨师。你有一个顾客数组customers，其中customers[i] = [arrivali,
↪timei]：
arrivali是第i位顾客到达的时间，到达时间按 非递减 顺序排列。
timei是给第 i位顾客做菜需要的时间。
当一位顾客到达时，他将他的订单给厨师，厨师一旦空闲的时候就开始做这位顾客的菜。
每位顾客会一直等待到厨师完成他的订单。
厨师同时只能做一个人的订单。厨师会严格按照 订单给他的顺序做菜。
请你返回所有顾客需要等待的 平均时间。与标准答案误差在10-5范围以内，都视为正确结果。
示例 1：输入：customers = [[1,2],[2,5],[4,3]] 输出：5.00000
解释：1) 第一位顾客在时刻 1 到达，厨师拿到他的订单并在时刻 1 立马开始做菜，
并在时刻 3 完成，第一位顾客等待时间为 3 - 1 = 2 。
2) 第二位顾客在时刻 2 到达，厨师在时刻 3 开始为他做菜，
并在时刻 8 完成，第二位顾客等待时间为 8 - 2 = 6 。
3) 第三位顾客在时刻 4 到达，厨师在时刻 8 开始为他做菜，并在时刻 11 完成，
第三位顾客等待时间为 11 - 4 = 7 。
平均等待时间为 (2 + 6 + 7) / 3 = 5 。
示例 2：输入：customers = [[5,2],[5,4],[10,3],[20,1]] 输出：3.25000
解释：1) 第一位顾客在时刻 5 到达，厨师拿到他的订单并在时刻 5 立马开始做菜，
并在时刻 7 完成，第一位顾客等待时间为 7 - 5 = 2 。
2) 第二位顾客在时刻 5 到达，厨师在时刻 7 开始为他做菜，并在时刻 11 完成，
```

(续下页)

(接上页)

```
第二位顾客等待时间为 11 - 5 = 6 。
3) 第三位顾客在时刻 10 到达，厨师在时刻 11 开始为他做菜，并在时刻 14 完成，
第三位顾客等待时间为 14 - 10 = 4 。
4) 第四位顾客在时刻 20 到达，厨师拿到他的订单并在时刻 20 立马开始做菜，
并在时刻 21 完成，第四位顾客等待时间为 21 - 20 = 1 。
平均等待时间为 (2 + 6 + 4 + 1) / 4 = 3.25 。
提示：1 <= customers.length <= 105
1 <= arrivali, timei <= 104
arrivali<= arrivali+1
```

- 解题思路

```
func averageWaitingTime(customers [][]int) float64 {
        sum := 0
        cur := customers[0][0]
        for i := 0; i < len(customers); i++ {
                if cur < customers[i][0] {
                        cur = customers[i][0] + customers[i][1]
                } else {
                        cur = cur + customers[i][1] // 做菜时间
                }
                wait := cur - customers[i][0] // 等待时间
                sum = sum + wait
        }
        return float64(sum) / float64(len(customers))
}
```

53.2 1702. 修改后的最大二进制字符串 (2)

- 题目

```
给你一个二进制字符串binary，它仅有0或者1组成。你可以使用下面的操作任意次对它进行修改：
操作 1 ：如果二进制串包含子字符串"00"，你可以用"10"将其替换。
比方说，"00010" -> "10010"
操作 2 ：如果二进制串包含子字符串"10"，你可以用"01"将其替换。
比方说，"00010" -> "00001"
请你返回执行上述操作任意次以后能得到的 最大二进制字符串。
如果二进制字符串 x对应的十进制数字大于二进制字符串 y对应的十进制数字，
那么我们称二进制字符串x大于二进制字符串y。
示例 1：输入：binary = "000110" 输出："111011"
解释：一个可行的转换为：
"000110" -> "000101"
```

(续下页)

(接上页)

```
"000101" -> "100101"
"100101" -> "110101"
"110101" -> "110011"
"110011" -> "111011"
示例 2: 输入: binary = "01" 输出: "01"
解释: "01" 没办法进行任何转换。
提示: 1 <= binary.length <= 105
binary 仅包含'0' 和'1' 。
```

- 解题思路

```
func maximumBinaryString(binary string) string {
        flag := true
        rightOne := 0 // 记录第1个0后面1的数量
        for i := 0; i < len(binary); i++ {
                if binary[i] == '0' {
                        flag = false
                } else {
                        if flag == false {
                                rightOne++
                        }
                }
        }
        if flag == true { // 全是1, 直接返回
                return binary
        }
        // 首先: 第1个0之前的1不需要移动。
        // 然后: 把第1个0之后的1移到后面, 后移: 10=>01。
        // 最后: 然后把中间的000, 都变成110, 00=>10。
        arr := make([]byte, len(binary))
        for i := 0; i < len(arr); i++ {
                arr[i] = '1'
        }
        arr[len(arr)-1-rightOne] = '0'
        return string(arr)
}

# 2
func maximumBinaryString(binary string) string {
        n := len(binary)
        count := strings.Count(binary, "1")
        if count >= n-1 {
                return binary
        }
```

(续下页)

(接上页)

```
        indexZero := strings.IndexByte(binary, '0')
        count = count - indexZero
        return strings.Repeat("1", n-1-count) + "0" + strings.Repeat("1", count)
}
```

53.3 1705. 吃苹果的最大数目 (2)

- 题目

```
有一棵特殊的苹果树，一连 n 天，每天都可以长出若干个苹果。
在第 i 天，树上会长出 apples[i] 个苹果，
这些苹果将会在 days[i] 天后（也就是说，第 i + days[i] 天时）腐烂，变得无法食用。
也可能有那么几天，树上不会长出新的苹果，此时用 apples[i] == 0 且 days[i] == 0 表示。
你打算每天 最多 吃一个苹果来保证营养均衡。注意，你可以在这 n 天之后继续吃苹果。
给你两个长度为 n 的整数数组 days 和 apples ，返回你可以吃掉的苹果的最大数目。
示例 1：输入：apples = [1,2,3,5,2], days = [3,2,1,4,2] 输出：7
解释：你可以吃掉 7 个苹果：
- 第一天，你吃掉第一天长出来的苹果。
- 第二天，你吃掉一个第二天长出来的苹果。
- 第三天，你吃掉一个第二天长出来的苹果。过了这一天，第三天长出来的苹果就已经腐烂了。
- 第四天到第七天，你吃的都是第四天长出来的苹果。
示例 2：输入：apples = [3,0,0,0,0,2], days = [3,0,0,0,0,2] 输出：5
解释：你可以吃掉 5 个苹果：
- 第一天到第三天，你吃的都是第一天长出来的苹果。
- 第四天和第五天不吃苹果。
- 第六天和第七天，你吃的都是第六天长出来的苹果。
提示：apples.length == n
days.length == n
1 <= n <= 2 * 104
0 <= apples[i], days[i] <= 2 * 104
只有在 apples[i] = 0 时，days[i] = 0 才成立
```

- 解题思路

```
func eatenApples(apples []int, days []int) int {
        res := 0
        nodeHeap := make(NodeHeap, 0)
        heap.Init(&nodeHeap)
        for i := 0; i < len(apples) || nodeHeap.Len() > 0; i++ {
                if i < len(apples) && apples[i] > 0 {
                        heap.Push(&nodeHeap, Node{
                                date: days[i] + i,
```

(续下页)

(接上页)

```
                                num:  apples[i],
                        })
                }
                for nodeHeap.Len() > 0 && nodeHeap[0].date == i {
                        heap.Pop(&nodeHeap)
                }
                if nodeHeap.Len() > 0 && nodeHeap[0].num > 0 {
                        res++
                        nodeHeap[0].num--
                        if nodeHeap[0].num == 0 {
                                heap.Pop(&nodeHeap)
                        }
                }
        }
        return res
}

type Node struct {
        date int
        num  int
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h NodeHeap) Less(i, j int) bool {
        return h[i].date < h[j].date
}

func (h NodeHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
```

(续下页)

(接上页)

```
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func eatenApples(apples []int, days []int) int {
        arr := make([]int, 40000)
        res := 0
        left, right := 0, 0
        for i := 0; ; i++ {
                if i < len(apples) {
                        target := i + days[i] // 保质期内
                        arr[target] = arr[target] + apples[i]
                        if target > right {
                                right = target
                        }
                        if left > target {
                                left = target
                        }
                } else {
                        if left > right {
                                break
                        }
                }
                // 吃苹果
                for left = i + 1; left <= right; left++ {
                        if arr[left] > 0 {
                                res++
                                arr[left]--
                                break
                        }
                }
        }
        return res
}
```

53.4 1706. 球会落何处 (3)

- 题目

```
用一个大小为 m x n 的二维网格 grid 表示一个箱子。你有 n
↪颗球。箱子的顶部和底部都是开着的。
箱子中的每个单元格都有一个对角线挡板，跨过单元格的两个角，可以将球导向左侧或者右侧。
将球导向右侧的挡板跨过左上角和右下角，在网格中用 1 表示。
将球导向左侧的挡板跨过右上角和左下角，在网格中用 -1 表示。
在箱子每一列的顶端各放一颗球。每颗球都可能卡在箱子里或从底部掉出来。
如果球恰好卡在两块挡板之间的 "V"
↪形图案，或者被一块挡导向到箱子的任意一侧边上，就会卡住。
返回一个大小为 n 的数组 answer ，
其中 answer[i] 是球放在顶部的第 i
↪列后从底部掉出来的那一列对应的下标，如果球卡在盒子里，则返回 -1 。
示例 1:
输入: grid = [[1,1,1,-1,-1],[1,1,1,-1,-1],[-1,-1,-1,1,1],[1,1,1,1,-1],[-1,-1,-1,-1,-
↪1]]
输出: [1,-1,-1,-1,-1]
解释: 示例如图:
b0 球开始放在第 0 列上，最终从箱子底部第 1 列掉出。
b1 球开始放在第 1 列上，会卡在第 2、3 列和第 1 行之间的 "V" 形里。
b2 球开始放在第 2 列上，会卡在第 2、3 列和第 0 行之间的 "V" 形里。
b3 球开始放在第 3 列上，会卡在第 2、3 列和第 0 行之间的 "V" 形里。
b4 球开始放在第 4 列上，会卡在第 2、3 列和第 1 行之间的 "V" 形里。
示例 2: 输入: grid = [[-1]] 输出: [-1]
解释: 球被卡在箱子左侧边上。
示例 3: 输入: grid = [[1,1,1,1,1,1],[-1,-1,-1,-1,-1,-1],[1,1,1,1,1,1],[-1,-1,-1,-1,-1,
↪-1]]
输出: [0,1,2,3,4,-1]
提示: m == grid.length
n == grid[i].length
1 <= m, n <= 100
grid[i][j] 为 1 或 -1
```

- 解题思路

```
func findBall(grid [][]int) []int {
        n, m := len(grid), len(grid[0])
        res := make([]int, m)
        for i := 0; i < m; i++ {
                res[i] = i
        }
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < m; j++ {
                        if res[j] != -1 {
                                if res[j] != m-1 && grid[i][res[j]] == 1 &&␣
↪grid[i][res[j]+1] == 1 { // 向右侧 \
                                        res[j] = res[j] + 1 // 坐标向右+1
                                } else if res[j] != 0 && grid[i][res[j]] == -1 &&␣
↪grid[i][res[j]-1] == -1 { // 向左侧 /
                                        res[j] = res[j] - 1 // 坐标向左-1
                                } else {
                                        res[j] = -1
                                }
                        }
                }
        }
        return res
}

# 2
func findBall(grid [][]int) []int {
        n, m := len(grid), len(grid[0])
        res := make([]int, 0)
        for j := 0; j < m; j++ { // 每列模拟
                index := j
                for i := 0; i < n; i++ {
                        if (grid[i][index] == 1 && (index == m-1 || grid[i][index+1]␣
↪== -1)) ||
                                (grid[i][index] == -1 && (index == 0 || grid[i][index-
↪1] == 1)) {
                                index = -1
                                break
                        }
                        index = index + grid[i][index]
                }
                res = append(res, index)
        }
        return res
}

# 3
func findBall(grid [][]int) []int {
        n, m := len(grid), len(grid[0])
        res := make([]int, m)
        for i := 0; i < m; i++ {
```

(续下页)

(接上页)

```
                res[i] = i
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if res[j] == -1 {
                                continue
                        }
                        if (grid[i][res[j]] == 1 && (res[j] == m-1 ||␣
↪grid[i][res[j]+1] == -1)) ||
                                (grid[i][res[j]] == -1 && (res[j] == 0 ||␣
↪grid[i][res[j]-1] == 1)) {
                                res[j] = -1
                                continue
                        }
                        res[j] = res[j] + grid[i][res[j]]
                }
        }
        return res
}
```

53.5 1711. 大餐计数 (1)

- 题目

```
大餐 是指 恰好包含两道不同餐品 的一餐，其美味程度之和等于 2 的幂。
你可以搭配 任意 两道餐品做一顿大餐。
给你一个整数数组 deliciousness ，其中 deliciousness[i] 是第 i道餐品的美味程度，
返回你可以用数组中的餐品做出的不同大餐的数量。
结果需要对 109 + 7 取余。
注意，只要餐品下标不同，就可以认为是不同的餐品，即便它们的美味程度相同。
示例 1：输入：deliciousness = [1,3,5,7,9] 输出：4
解释：大餐的美味程度组合为 (1,3) 、(1,7) 、(3,5) 和 (7,9) 。
它们各自的美味程度之和分别为 4 、8 、8 和 16 ，都是 2 的幂。
示例 2：输入：deliciousness = [1,1,1,3,3,3,7] 输出：15
解释：大餐的美味程度组合为 3 种 (1,1) ，9 种 (1,3) ，和 3 种 (1,7) 。
提示：1 <= deliciousness.length <= 105
0 <= deliciousness[i] <= 220
```

- 解题思路

```
func countPairs(deliciousness []int) int {
        res := 0
```

(续下页)

(接上页)

```
        m := make(map[int]int)
        for i := 0; i < len(deliciousness); i++ {
                total := 1
                for j := 0; j <= 21; j++ {
                        if m[total-deliciousness[i]] > 0 {
                                res = res + m[total-deliciousness[i]]
                        }
                        total = total * 2
                }
                m[deliciousness[i]]++
        }
        return res % 1000000007
}
```

53.6 1712. 将数组分成三个子数组的方案数 (2)

- 题目

```
我们称一个分割整数数组的方案是 好的，当它满足：
数组被分成三个 非空连续子数组，从左至右分别命名为left，mid，right。
left中元素和小于等于mid中元素和，mid中元素和小于等于right中元素和。
给你一个 非负 整数数组nums，请你返回好的 分割 nums方案数目。
由于答案可能会很大，请你将结果对 109+ 7取余后返回。
示例 1：输入：nums = [1,1,1] 输出：1
解释：唯一一种好的分割方案是将 nums 分成 [1] [1] [1] 。
示例 2：输入：nums = [1,2,2,2,5,0] 输出：3
解释：nums 总共有 3 种好的分割方案：
[1] [2] [2,2,5,0]
[1] [2,2] [2,5,0]
[1,2] [2,2] [5,0]
示例 3：输入：nums = [3,2,1] 输出：0
解释：没有好的分割方案。
提示：3 <= nums.length <= 105
0 <= nums[i] <= 104
```

- 解题思路

```
func waysToSplit(nums []int) int {
        res := 0
        n := len(nums)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                arr[i+1] = arr[i] + nums[i]
        }
        left, right := 2, 2
        for i := 1; i <= n-1; i++ {
                first := arr[i] // 第1个数
                left = max(left, i+1)
                right = max(right, i+1)
                for left <= n-1 && arr[left]-first < first { //找到第一个满足的中间数组左边坐标
                        left++
                }
                for right <= n-1 && arr[right]-first <= arr[n]-arr[right] {
                        right++
                }
                if left <= right {
                        res = (res + right - left) % 1000000007
                }
        }
        return res % 1000000007
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func waysToSplit(nums []int) int {
        res := 0
        n := len(nums)
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + nums[i]
        }
        for i := 1; i <= n-1; i++ {
                first := arr[i] // 第1个数
                if first*3 > arr[n] {
                        break
                }
                left, right := i+1, n-1
                for left <= right {
```

(续下页)

(接上页)

```
                        mid := left + (right-left)/2
                        if arr[n]-arr[mid] < arr[mid]-first {
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                }
                b := left
                left, right = i+1, n-1
                for left <= right {
                        mid := left + (right-left)/2
                        if arr[mid]-first < first {
                                left = mid + 1
                        } else {
                                right = mid - 1
                        }
                }
                a := left
                res = (res + b - a) % 1000000007
        }
        return res % 1000000007
}
```

53.7 1717. 删除子字符串的最大得分 (2)

- 题目

```
给你一个字符串s和两个整数x 和y。你可以执行下面两种操作任意次。
删除子字符串"ab"并得到x分。
比方说，从"cabxbae"删除 ab，得到"cxbae"。
删除子字符串"ba"并得到y分。
比方说，从"cabxbae"删除 ba，得到"cabxe"。
请返回对 s字符串执行上面操作若干次能得到的最大得分。
示例 1：输入：s = "cdbcbbaaabab", x = 4, y = 5 输出：19
解释：- 删除 "cdbcbbaaabab" 中加粗的 "ba" ，得到 s = "cdbcbbaaab" ，加 5 分。
- 删除 "cdbcbbaaab" 中加粗的 "ab" ，得到 s = "cdbcbbaa" ，加 4 分。
- 删除 "cdbcbbaa" 中加粗的 "ba" ，得到 s = "cdbcba" ，加 5 分。
- 删除 "cdbcba" 中加粗的 "ba" ，得到 s = "cdbc" ，加 5 分。
总得分为 5 + 4 + 5 + 5 = 19 。
示例 2：输入：s = "aabbaaxybbaabb", x = 5, y = 4 输出：20
提示：1 <= s.length <= 105
1 <= x, y <= 104
```

(续下页)

(接上页)

s只包含小写英文字母。

- 解题思路

```
func maximumGain(s string, x int, y int) int {
        if x > y {
                x, y = y, x
                s = reverse(s)
        }
        res := 0
        stack := make([]byte, 0)
        // 先处理ba,y分多
        for i := 0; i < len(s); i++ {
                if s[i] != 'a' {
                        stack = append(stack, s[i])
                } else {
                        if len(stack) > 0 && stack[len(stack)-1] == 'b' {
                                stack = stack[:len(stack)-1]
                                res = res + y
                        } else {
                                stack = append(stack, s[i])
                        }
                }
        }
        // 处理ab
        temp := make([]byte, 0)
        for len(stack) > 0 {
                c := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                if c != 'a' {
                        temp = append(temp, c)
                } else {
                        if len(temp) > 0 && temp[len(temp)-1] == 'b' {
                                temp = temp[:len(temp)-1]
                                res = res + x
                        } else {
                                temp = append(temp, c)
                        }
                }
        }
        return res
}

func reverse(s string) string {
```

(续下页)

(接上页)

```
        arr := []byte(s)
        for i := 0; i < len(s)/2; i++ {
                arr[i], arr[len(s)-1-i] = arr[len(s)-1-i], arr[i]
        }
        return string(arr)
}

# 2
func maximumGain(s string, x int, y int) int {
        if x > y {
                x, y = y, x
                s = reverse(s)
        }
        res := 0
        a, b := 0, 0
        for i := 0; i < len(s); i++ {
                if s[i] == 'a' || s[i] == 'b' {
                        if s[i] == 'a' {
                                a++
                        } else {
                                b++
                        }
                        // 处理ba
                        if s[i] == 'a' && b > 0 {
                                a--
                                b--
                                res = res + y
                        }
                } else {
                        res = res + x*min(a, b) // 处理ab
                        a, b = 0, 0
                }
        }
        res = res + x*min(a, b) // 处理ab
        return res
}

func reverse(s string) string {
        arr := []byte(s)
        for i := 0; i < len(s)/2; i++ {
                arr[i], arr[len(s)-1-i] = arr[len(s)-1-i], arr[i]
        }
        return string(arr)
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

53.8 1718. 构建字典序最大的可行序列 (1)

- 题目

```
给你一个整数n，请你找到满足下面条件的一个序列：
整数1在序列中只出现一次。
2到n之间每个整数都恰好出现两次。
对于每个2到n之间的整数i，两个i之间出现的距离恰好为i。
序列里面两个数 a[i]和 a[j]之间的 距离，我们定义为它们下标绝对值之差|j - i|。
请你返回满足上述条件中字典序最大的序列。题目保证在给定限制条件下，一定存在解。
一个序列a被认为比序列b（两者长度相同）字典序更大的条件是：
a 和b中第一个不一样的数字处，a序列的数字比b序列的数字大。
比方说，[0,1,9,0]比[0,1,5,6]字典序更大，因为第一个不同的位置是第三个数字，且9比5大。
示例 1：输入：n = 3 输出：[3,1,2,3,2]
解释：[2,3,2,1,3] 也是一个可行的序列，但是 [3,1,2,3,2] 是字典序最大的序列。
示例 2：输入：n = 5 输出：[5,3,1,4,3,5,2,4,2]
提示：1 <= n <= 20
```

- 解题思路

```
var res []int

func constructDistancedSequence(n int) []int {
        path := make([]int, 2*n-1)
        res = make([]int, 2*n-1)
        visited := make([]bool, n+1)
        dfs(n, 0, &path, &visited)
        return res
}

func dfs(n int, cur int, path *[]int, visited *[]bool) bool {
        if cur == 2*n-1 {
                copy(res, *path)
```

(续下页)

(接上页)

```
                return true
        }
        if (*path)[cur] > 0 {
                return dfs(n, cur+1, path, visited)
        }
        for i := n; i > 0; i-- { // 从大的开始尝试
                next := cur + i
                if (*visited)[i] == true {
                        continue
                }
                if i > 1 && (next >= 2*n-1 || (*path)[next] > 0) {
                        continue
                }
                a, b := cur, next
                if i == 1 { // 1特判
                        a, b = cur, cur
                }
                (*visited)[i] = true
                (*path)[a] = i
                (*path)[b] = i
                if dfs(n, cur+1, path, visited) == true {
                        return true
                }
                (*path)[a] = 0
                (*path)[b] = 0
                (*visited)[i] = false
        }
        return false
}
```

53.9 1721. 交换链表中的节点 (2)

- 题目

```
给你链表的头节点 head 和一个整数 k 。
交换 链表正数第 k 个节点和倒数第 k 个节点的值后，返回链表的头节点（链表 从 1␣
↪开始索引）。
示例 1：输入：head = [1,2,3,4,5], k = 2 输出：[1,4,3,2,5]
示例 2：输入：head = [7,9,6,6,7,8,3,0,9,5], k = 5 输出：[7,9,6,6,8,7,3,0,9,5]
示例 3：输入：head = [1], k = 1 输出：[1]
示例 4：输入：head = [1,2], k = 1 输出：[2,1]
示例 5：输入：head = [1,2,3], k = 2 输出：[1,2,3]
```

(续下页)

(接上页)

```
提示：链表中节点的数目是 n
1 <= k <= n <= 105
0 <= Node.val <= 100
```

- 解题思路

```
func swapNodes(head *ListNode, k int) *ListNode {
        slow, fast := head, head
        for i := 0; i < k-1; i++ {
                fast = fast.Next
        }
        a := fast
        for fast != nil && fast.Next != nil {
                fast = fast.Next
                slow = slow.Next
        }
        slow.Val, a.Val = a.Val, slow.Val
        return head
}

# 2
func swapNodes(head *ListNode, k int) *ListNode {
        res := make([]*ListNode, 0)
        cur := head
        for cur != nil {
                res = append(res, cur)
                cur = cur.Next
        }
        res[k-1].Val, res[len(res)-k].Val = res[len(res)-k].Val, res[k-1].Val
        return head
}
```

53.10 1722. 执行交换操作后的最小汉明距离 (1)

- 题目

```
给你两个整数数组 source 和 target ，长度都是 n 。
还有一个数组 allowedSwaps ，其中每个 allowedSwaps[i] = [ai, bi]
表示你可以交换数组 source 中下标为 ai 和 bi（下标从 0 开始）的两个元素。
注意，你可以按 任意 顺序 多次 交换一对特定下标指向的元素。
相同长度的两个数组source 和 target 间的 汉明距离 是元素不同的下标数量。
形式上，其值等于满足source[i] != target[i] （下标从 0 开始）的下标 i（0 <= i <= n-
```

(续下页)

(接上页)

```
↪1）的数量。
在对数组 source 执行 任意 数量的交换操作后，返回 source 和 target 间的 最小汉明距离 。
示例 1：输入：source = [1,2,3,4], target = [2,1,4,5], allowedSwaps = [[0,1],[2,3]]␣
↪输出：1
解释：source 可以按下述方式转换：
- 交换下标 0 和 1 指向的元素：source = [2,1,3,4]
- 交换下标 2 和 3 指向的元素：source = [2,1,4,3]
source 和 target 间的汉明距离是 1 ，二者有 1 处元素不同，在下标 3 。
示例 2：输入：source = [1,2,3,4], target = [1,3,2,4], allowedSwaps = [] 输出：2
解释：不能对 source 执行交换操作。
source 和 target 间的汉明距离是 2 ，二者有 2 处元素不同，在下标 1 和下标 2 。
示例 3：输入：source = [5,1,2,4,3], target = [1,5,4,2,3],
allowedSwaps = [[0,4],[4,2],[1,3],[1,4]] 输出：0
提示：n == source.length == target.length
1 <= n <= 105
1 <= source[i], target[i] <= 105
0 <= allowedSwaps.length <= 105
allowedSwaps[i].length == 2
0 <= ai, bi <= n - 1
ai != bi
```

- 解题思路

```
func minimumHammingDistance(source []int, target []int, allowedSwaps [][]int) int {
        n := len(source)
        fa = Init(n)
        for i := 0; i < len(allowedSwaps); i++ {
                a, b := allowedSwaps[i][0], allowedSwaps[i][1]
                union(a, b)
        }
        m := make(map[int]map[int]int)
        res := 0
        for i := 0; i < len(source); i++ {
                v := find(i)
                if m[v] == nil {
                        m[v] = make(map[int]int)
                }
                m[v][source[i]]++
        }
        for i := 0; i < len(target); i++ {
                v := find(i)
                if m[v][target[i]] == 0 {
                        res++
                } else {
```

(续下页)

(接上页)

```
                        m[v][target[i]]--
                }
        }
        return res
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}
```

53.11 1726. 同积元组 (1)

- 题目

```
给你一个由 不同 正整数组成的数组 nums ，请你返回满足a * b = c * d 的元组 (a, b, c, d)␣
↪的数量。
其中 a、b、c 和 d 都是 nums 中的元素，且 a != b != c != d 。
示例 1：输入：nums = [2,3,4,6] 输出：8
解释：存在 8 个满足题意的元组：
(2,6,3,4) , (2,6,4,3) , (6,2,3,4) , (6,2,4,3)
(3,4,2,6) , (3,4,2,6) , (3,4,6,2) , (4,3,6,2)
示例 2：输入：nums = [1,2,4,5,10] 输出：16
```

(续下页)

(接上页)

```
解释：存在 16 个满足题意的元组：
(1,10,2,5) , (1,10,5,2) , (10,1,2,5) , (10,1,5,2)
(2,5,1,10) , (2,5,10,1) , (5,2,1,10) , (5,2,10,1)
(2,10,4,5) , (2,10,5,4) , (10,2,4,5) , (10,2,4,5)
(4,5,2,10) , (4,5,10,2) , (5,4,2,10) , (5,4,10,2)
示例 3：输入：nums = [2,3,4,6,8,12] 输出：40
示例 4：输入：nums = [2,3,5,7] 输出：0
提示：1 <= nums.length <= 1000
1 <= nums[i] <= 104
nums 中的所有元素 互不相同
```

- 解题思路

```
func tupleSameProduct(nums []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        total := nums[i] * nums[j]
                        m[total]++
                }
        }
        for _, v := range m {
                if v >= 2 {
                        res = res + v*(v-1)/2*8
                }
        }
        return res
}
```

53.12 1727. 重新排列后的最大子矩阵 (1)

- 题目

```
给你一个二进制矩阵matrix，它的大小为m x n，你可以将 matrix中的 列按任意顺序重新排列。
请你返回最优方案下将 matrix重新排列后，全是 1的子矩阵面积。
示例 1：输入：matrix = [[0,0,1],[1,1,1],[1,0,1]] 输出：4
解释：你可以按照上图方式重新排列矩阵的每一列。
最大的全 1 子矩阵是上图中加粗的部分，面积为 4 。
示例 2：输入：matrix = [[1,0,1,0,1]] 输出：3
解释：你可以按照上图方式重新排列矩阵的每一列。
最大的全 1 子矩阵是上图中加粗的部分，面积为 3 。
```

(续下页)

(接上页)

```
示例 3：输入：matrix = [[1,1,0],[1,0,1]] 输出：2
解释：由于你只能整列整列重新排布，所以没有比面积为 2 更大的全 1 子矩形。
示例 4：输入：matrix = [[0,0],[0,0]] 输出：0
解释：由于矩阵中没有 1 ，没有任何全 1 的子矩阵，所以面积为 0 。
提示：m == matrix.length
n == matrix[i].length
1 <= m * n <= 105
matrix[i][j]要么是0，要么是1 。
```

- 解题思路

```
func largestSubmatrix(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        for i := 1; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 1 { // 统计以该行为基连续1的个数
                                matrix[i][j] = matrix[i][j] + matrix[i-1][j]
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                sort.Ints(matrix[i])
                for j := m - 1; j >= 0; j-- {
                        height := matrix[i][j] // 高度有序，右边最高=>高度取决于左边
                        width := m - j
                        if height*width > res {
                                res = height * width
                        }
                }
        }
        return res
}
```

53.13 1733. 需要教语言的最少人数 (1)

- 题目

```
在一个由m个用户组成的社交网络里，我们获取到一些用户之间的好友关系。
两个用户之间可以相互沟通的条件是他们都掌握同一门语言。
给你一个整数n，数组languages和数组friendships，它们的含义如下：
总共有n种语言，编号从1 到n。
```

(续下页)

(接上页)

```
languages[i]是第 i位用户掌握的语言集合。
friendships[i] = [ui, vi]表示ui和vi为好友关系。
你可以选择 一门语言并教会一些用户，使得所有好友之间都可以相互沟通。请返回你␣
↪最少需要教会多少名用户。
请注意，好友关系没有传递性，也就是说如果x 和y是好友，且y和z是好友，x 和z不一定是好友。
示例 1: 输入: n = 2, languages = [[1],[2],[1,2]], friendships = [[1,2],[1,3],[2,3]]␣
↪输出: 1
解释: 你可以选择教用户 1 第二门语言，也可以选择教用户 2 第一门语言。
示例 2:
输入: n = 3, languages = [[2],[1,3],[1,2],[3]], friendships = [[1,4],[1,2],[3,4],[2,
↪3]]
输出: 2
解释: 教用户 1 和用户 3 第三门语言，需要教 2 名用户。
提示: 2 <= n <= 500
languages.length == m
1 <= m <= 500
1 <= languages[i].length <= n
1 <= languages[i][j] <= n
1 <= ui < vi <= languages.length
1 <= friendships.length <= 500
所有的好友关系(ui, vi)都是唯一的。
languages[i]中包含的值互不相同。
```

- 解题思路

```
func minimumTeachings(n int, languages [][]int, friendships [][]int) int {
        m := make(map[int]map[int]int)
        for i := 0; i < len(languages); i++ {
                a := i + 1
                m[a] = make(map[int]int)
                for j := 0; j < len(languages[i]); j++ {
                        b := languages[i][j]
                        m[a][b] = 1
                }
        }
        need := make(map[int]bool) // 需要沟通的人列表
        for i := 0; i < len(friendships); i++ {
                a, b := friendships[i][0], friendships[i][1]
                flag := false
                for j := 1; j <= n; j++ {
                        if m[a][j] == 1 && m[b][j] == 1 {
                                flag = true
                                break
                        }
```

(续下页)

(接上页)

```
                }
                if flag == false {
                        need[a] = true
                        need[b] = true
                }
        }
        res := 0
        for i := 1; i <= n; i++ {
                count := 0
                for k := range need {
                        if m[k][i] == 1 {
                                count++
                        }
                }
                if count > res {
                        res = count
                }
        }
        return len(need) - res
}
```

53.14 1734. 解码异或后的排列 (2)

- 题目

```
给你一个整数数组perm，它是前n个正整数的排列，且n是个 奇数。
它被加密成另一个长度为 n - 1的整数数组encoded，满足encoded[i] = perm[i] XOR perm[i +␣
↪1]。
比方说，如果perm = [1,3,2]，那么encoded = [2,1]。
给你encoded数组，请你返回原始数组perm。题目保证答案存在且唯一。
示例 1：输入：encoded = [3,1] 输出：[1,2,3]
解释：如果 perm = [1,2,3] ，那么 encoded = [1 XOR 2,2 XOR 3] = [3,1]
示例 2：输入：encoded = [6,5,4,6] 输出：[2,4,1,5,3]
提示：3 <= n <105
n是奇数。
encoded.length == n - 1
```

- 解题思路

```
func decode(encoded []int) []int {
        n := len(encoded)
        temp := make([]int, n)
```

(续下页)

(接上页)

```
        copy(temp, encoded)
        first := encoded[0]
        for i := 1; i < n; i++ {
                temp[i] = encoded[i] ^ temp[i-1]
                first = first ^ temp[i]
        }
        for i := 1; i <= n+1; i++ {
                first = first ^ i
        }

        res := make([]int, n+1)
        res[0] = first
        for i := 0; i < n; i++ {
                res[i+1] = encoded[i] ^ res[i]
        }
        return res
}

# 2
func decode(encoded []int) []int {
        n := len(encoded)
        first := 0
        for i := 1; i <= n+1; i++ {
                first = first ^ i
        }
        for i := 1; i <= n; i = i + 2 {
                first = first ^ encoded[i]
        }
        res := make([]int, n+1)
        res[0] = first
        for i := 0; i < n; i++ {
                res[i+1] = encoded[i] ^ res[i]
        }
        return res
}
```

53.15 1737. 满足三条件之一需改变的最少字符数 (1)

- 题目

```
给你两个字符串 a 和 b ，二者均由小写字母组成。
一步操作中，你可以将 a 或 b 中的 任一字符 改变为 任一小写字母 。
操作的最终目标是满足下列三个条件 之一 ：
a 中的 每个字母 在字母表中 严格小于 b 中的 每个字母 。
b 中的 每个字母 在字母表中 严格小于 a 中的 每个字母 。
a 和 b 都 由 同一个 字母组成。
返回达成目标所需的 最少 操作数。
示例 1：输入：a = "aba", b = "caa" 输出：2
解释：满足每个条件的最佳方案分别是：
1) 将 b 变为 "ccc"，2 次操作，满足 a 中的每个字母都小于 b 中的每个字母；
2) 将 a 变为 "bbb" 并将 b 变为 "aaa"，3 次操作，满足 b 中的每个字母都小于 a
↪中的每个字母；
3) 将 a 变为 "aaa" 并将 b 变为 "aaa"，2 次操作，满足 a 和 b 由同一个字母组成。
最佳的方案只需要 2 次操作（满足条件 1 或者条件 3）。
示例 2：输入：a = "dabadd", b = "cda" 输出：3
解释：满足条件 1 的最佳方案是将 b 变为 "eee" 。
提示：1 <= a.length, b.length <= 105
a 和 b 只由小写字母组成
```

- 解题思路

```
func minCharacters(a string, b string) int {
        arrA, arrB := [26]int{}, [26]int{}
        for i := 0; i < len(a); i++ {
                arrA[a[i]-'a']++
        }
        for i := 0; i < len(b); i++ {
                arrB[b[i]-'a']++
        }
        res := math.MaxInt32
        lengthA, lengthB := len(a), len(b)
        sumA, sumB := 0, 0
        // a-y的情况
        for i := 0; i < 25; i++ {
                sumA = sumA + arrA[i]
                sumB = sumB + arrB[i]
                res = min(res, lengthA-sumA+sumB)                // 条件1: a<b
                res = min(res, sumA+lengthB-sumB)                // 条件2: b<a
                res = min(res, lengthA-arrA[i]+lengthB-arrB[i]) // 条件3: 全都相同
        }
```

(续下页)

(接上页)

```
        res = min(res, lengthA-arrA[25]+lengthB-arrB[25]) // z全都相同
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

53.16 1738. 找出第 K 大的异或坐标值 (2)

- 题目

```
给你一个二维矩阵 matrix 和一个整数 k ，矩阵大小为m x n 由非负整数组成。
矩阵中坐标 (a, b) 的 值 可由对所有满足 0 <= i <= a < m 且 0 <= j <= b < n 的元素
matrix[i][j] (下标从 0 开始计数) 执行异或运算得到。
请你找出matrix 的所有坐标中第 k 大的值 (k 的值从 1 开始计数) 。
示例 1: 输入: matrix = [[5,2],[1,6]], k = 1 输出: 7
解释: 坐标 (0,1) 的值是 5 XOR 2 = 7 ，为最大的值。
示例 2: 输入: matrix = [[5,2],[1,6]], k = 2 输出: 5
解释: 坐标 (0,0) 的值是 5 = 5 ，为第 2 大的值。
示例 3: 输入: matrix = [[5,2],[1,6]], k = 3 输出: 4
解释: 坐标 (1,0) 的值是 5 XOR 1 = 4 ，为第 3 大的值。
示例 4: 输入: matrix = [[5,2],[1,6]], k = 4 输出: 0
解释: 坐标 (1,1) 的值是 5 XOR 2 XOR 1 XOR 6 = 0 ，为第 4 大的值。
提示: m == matrix.length
n == matrix[i].length
1 <= m, n <= 1000
0 <= matrix[i][j] <= 106
1 <= k <= m * n
```

- 解题思路

```
func kthLargestValue(matrix [][]int, k int) int {
        m := len(matrix)
        n := len(matrix[0])
        arr := make([]int, 0)
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if i == 0 && j > 0 {
```

(续下页)

(接上页)

```
                                matrix[i][j] = matrix[i][j-1] ^ matrix[i][j]
                        } else if i > 0 && j == 0 {
                                matrix[i][j] = matrix[i-1][j] ^ matrix[i][j]
                        } else if i > 0 && j > 0 {
                                matrix[i][j] = matrix[i-1][j] ^ matrix[i][j-1] ^␣
↪matrix[i][j] ^ matrix[i-1][j-1]
                        }
                        arr = append(arr, matrix[i][j])
                }
        }
        sort.Ints(arr)
        return arr[len(arr)-k]
}

# 2
func kthLargestValue(matrix [][]int, k int) int {
        m := len(matrix)
        n := len(matrix[0])
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if i == 0 && j > 0 {
                                matrix[i][j] = matrix[i][j-1] ^ matrix[i][j]
                        } else if i > 0 && j == 0 {
                                matrix[i][j] = matrix[i-1][j] ^ matrix[i][j]
                        } else if i > 0 && j > 0 {
                                matrix[i][j] = matrix[i-1][j] ^ matrix[i][j-1] ^␣
↪matrix[i][j] ^ matrix[i-1][j-1]
                        }
                        heap.Push(&intHeap, matrix[i][j])
                        if intHeap.Len() > k {
                                heap.Pop(&intHeap)
                        }
                }
        }
        return intHeap[0]
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
```

(续下页)

(接上页)

```
}

func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

53.17 1743. 从相邻元素对还原数组 (2)

- 题目

```
存在一个由 n 个不同元素组成的整数数组 nums ，但你已经记不清具体内容。
好在你还记得 nums 中的每一对相邻元素。
给你一个二维整数数组 adjacentPairs ，大小为 n - 1 ，
其中每个 adjacentPairs[i] = [ui, vi] 表示元素 ui 和 vi 在 nums 中相邻。
题目数据保证所有由元素 nums[i] 和 nums[i+1] 组成的相邻元素对都存在于 adjacentPairs␣
↪中，
存在形式可能是 [nums[i], nums[i+1]] ，也可能是 [nums[i+1], nums[i]] 。
这些相邻元素对可以 按任意顺序 出现。
返回 原始数组 nums 。如果存在多种解答，返回 其中任意一个 即可。
示例 1：输入：adjacentPairs = [[2,1],[3,4],[3,2]] 输出：[1,2,3,4]
解释：数组的所有相邻元素对都在 adjacentPairs 中。
特别要注意的是，adjacentPairs[i] 只表示两个元素相邻，并不保证其 左-右 顺序。
示例 2：输入：adjacentPairs = [[4,-2],[1,4],[-3,1]] 输出：[-2,4,1,-3]
解释：数组中可能存在负数。
另一种解答是 [-3,1,4,-2] ，也会被视作正确答案。
示例 3：输入：adjacentPairs = [[100000,-100000]] 输出：[100000,-100000]
提示：nums.length == n
adjacentPairs.length == n - 1
```

(续下页)

(接上页)

```
adjacentPairs[i].length == 2
2 <= n <= 105
-105 <= nums[i], ui, vi <= 105
题目数据保证存在一些以adjacentPairs 作为元素对的数组 nums
```

- 解题思路

```
var res []int
var m map[int][]int

func restoreArray(adjacentPairs [][]int) []int {
        m = make(map[int][]int)
        for i := 0; i < len(adjacentPairs); i++ {
                a, b := adjacentPairs[i][0], adjacentPairs[i][1]
                m[a] = append(m[a], b)
                m[b] = append(m[b], a)
        }
        arr := make([]int, 0)
        for k, v := range m {
                if len(v) == 1 {
                        arr = append(arr, k)
                }
        }
        res = make([]int, 0)
        dfs(arr[0], make(map[int]bool))
        return res
}

func dfs(cur int, visited map[int]bool) {
        res = append(res, cur)
        visited[cur] = true
        for i := range m[cur] {
                if visited[m[cur][i]] == false {
                        dfs(m[cur][i], visited)
                }
        }
}

# 2
func restoreArray(adjacentPairs [][]int) []int {
        m := make(map[int][]int)
        for i := 0; i < len(adjacentPairs); i++ {
                a, b := adjacentPairs[i][0], adjacentPairs[i][1]
                m[a] = append(m[a], b)
```

(续下页)

(接上页)

```
                m[b] = append(m[b], a)
        }
        prev := 0
        cur := 0
        for k, v := range m {
                if len(v) == 1 {
                        prev = k
                        cur = v[0]
                        break
                }
        }
        res := make([]int, 0)
        res = append(res, prev)
        n := len(adjacentPairs) + 1
        for n > len(res) {
                res = append(res, cur)
                for i := 0; i < len(m[cur]); i++ {
                        if m[cur][i] != prev {
                                prev = cur
                                cur = m[cur][i]
                                break
                        }
                }
        }
        return res
}
```

53.18 1744. 你能在你最喜欢的那天吃到你最喜欢的糖果吗？(1)

- 题目

```
给你一个下标从 0 开始的正整数数组candiesCount，
其中candiesCount[i]表示你拥有的第i类糖果的数目。
同时给你一个二维数组queries，其中queries[i] = [favoriteTypei, favoriteDayi,␣
↪dailyCapi]。
你按照如下规则进行一场游戏：
你从第0天开始吃糖果。
你在吃完 所有第 i - 1类糖果之前，不能吃任何一颗第 i类糖果。
在吃完所有糖果之前，你必须每天 至少吃 一颗糖果。
请你构建一个布尔型数组answer，满足answer.length == queries.length 。
answer[i]为true的条件是：在每天吃 不超过 dailyCapi颗糖果的前提下，
你可以在第favoriteDayi天吃到第favoriteTypei类糖果；否则 answer[i]为 false。
```

(续下页)

(接上页)

```
注意，只要满足上面 3 条规则中的第二条规则，你就可以在同一天吃不同类型的糖果。
请你返回得到的数组answer。
示例 1：输入：candiesCount = [7,4,5,3,8], queries = [[0,2,2],[4,2,4],[2,13,
↪1000000000]]
输出：[true,false,true]
提示：1- 在第 0 天吃 2 颗糖果(类型 0)，第 1 天吃 2 颗糖果（类型 0），第 2␣
↪天你可以吃到类型 0 的糖果。
2- 每天你最多吃 4 颗糖果。即使第 0 天吃 4 颗糖果（类型 0），第 1 天吃 4 颗糖果（类型␣
↪0 和类型 1），
你也没办法在第 2 天吃到类型 4 的糖果。
换言之，你没法在每天吃 4 颗糖果的限制下在第 2 天吃到第 4 类糖果。
3- 如果你每天吃 1 颗糖果，你可以在第 13 天吃到类型 2 的糖果。
示例 2：输入：candiesCount = [5,2,6,4,1],
queries = [[3,1,2],[4,10,3],[3,10,100],[4,100,30],[1,3,1]]
输出：[false,true,true,false,false]
提示：1 <= candiesCount.length <= 105
1 <= candiesCount[i] <= 105
1 <= queries.length <= 105
queries[i].length == 3
0 <= favoriteTypei < candiesCount.length
0 <= favoriteDayi <= 109
1 <= dailyCapi <= 109
```

- 解题思路

```go
func canEat(candiesCount []int, queries [][]int) []bool {
        arr := make([]int, len(candiesCount)+1)
        for i := 1; i <= len(candiesCount); i++ {
                arr[i] = arr[i-1] + candiesCount[i-1]
        }
        res := make([]bool, len(queries))
        for i := 0; i < len(queries); i++ {
                a := queries[i][0] // 第几类
                b := queries[i][1] // 第几天，到第n天共n+1天
                c := queries[i][2] // 每天最多吃c颗
                total := arr[a+1]
                if total <= b { // 最少一天一颗
                        continue
                }
                if c*(b+1) <= arr[a] { // 最多每天吃c颗
                        continue
                }
                res[i] = true
        }
```

(续下页)

(接上页)

```
        return res
}
```

53.19 1749. 任意子数组和的绝对值的最大值 (3)

- 题目

```
给你一个整数数组nums。一个子数组[numsl, numsl+1, ..., numsr-1, numsr]
的 和的绝对值为abs(numsl + numsl+1 + ... + numsr-1 + numsr)。
请你找出 nums中 和的绝对值 最大的任意子数组（可能为空），并返回该 最大值。
abs(x)定义如下：
如果x是负整数，那么abs(x) = -x。
如果x是非负整数，那么abs(x) = x。
示例 1：输入：nums = [1,-3,2,3,-4] 输出：5
解释：子数组 [2,3] 和的绝对值最大，为 abs(2+3) = abs(5) = 5 。
示例 2：输入：nums = [2,-5,1,-4,3,-2] 输出：8
解释：子数组 [-5,1,-4] 和的绝对值最大，为 abs(-5+1-4) = abs(-8) = 8 。
提示：1 <= nums.length <= 105
-104 <= nums[i] <= 104
```

- 解题思路

```
func maxAbsoluteSum(nums []int) int {
        arr := make([]int, len(nums)+1)
        for i := 1; i <= len(nums); i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        sort.Ints(arr)
        return arr[len(nums)] - arr[0]
}

# 2
func maxAbsoluteSum(nums []int) int {
        arr := make([]int, len(nums)+1)
        for i := 1; i <= len(nums); i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        maxValue, minValue := 0, 0
        res := 0
        for i := 1; i < len(arr); i++ {
                res = max(res, abs(arr[i]-maxValue))
                res = max(res, abs(arr[i]-minValue))
```

(续下页)

(接上页)

```
                maxValue = max(maxValue, arr[i])
                minValue = min(minValue, arr[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 3
func maxAbsoluteSum(nums []int) int {
        maxValue, minValue := 0, 0
        res := 0
        for i := 0; i < len(nums); i++ {
                if maxValue >= 0 {
                        maxValue = maxValue + nums[i]
                } else {
                        maxValue = nums[i]
                }
                if minValue <= 0 {
                        minValue = minValue + nums[i]
                } else {
                        minValue = nums[i]
                }
                res = max(res, abs(maxValue))
```

(续下页)

(接上页)

```
                res = max(res, abs(minValue))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

53.20 1750. 删除字符串两端相同字符后的最短长度 (2)

- 题目

```
给你一个只包含字符 'a', 'b'和 'c'的字符串s, 你可以执行下面这个操作 (5 个步骤) 任意次:
选择字符串 s一个 非空 的前缀, 这个前缀的所有字符都相同。
选择字符串 s一个 非空 的后缀, 这个后缀的所有字符都相同。
前缀和后缀在字符串中任意位置都不能有交集。
前缀和后缀包含的所有字符都要相同。
同时删除前缀和后缀。
请你返回对字符串 s执行上面操作任意次以后 (可能 0 次), 能得到的 最短长度。
示例 1: 输入: s = "ca" 输出: 2
解释: 你没法删除任何一个字符, 所以字符串长度仍然保持不变。
示例 2: 输入: s = "cabaabac" 输出: 0
解释: 最优操作序列为:
- 选择前缀 "c" 和后缀 "c" 并删除它们, 得到 s = "abaaba" 。
- 选择前缀 "a" 和后缀 "a" 并删除它们, 得到 s = "baab" 。
- 选择前缀 "b" 和后缀 "b" 并删除它们, 得到 s = "aa" 。
- 选择前缀 "a" 和后缀 "a" 并删除它们, 得到 s = "" 。
示例 3: 输入: s = "aabccabba" 输出: 3
解释: 最优操作序列为:
- 选择前缀 "aa" 和后缀 "a" 并删除它们, 得到 s = "bccabb" 。
- 选择前缀 "b" 和后缀 "bb" 并删除它们, 得到 s = "cca" 。
```

(续下页)

(接上页)

```
提示：1 <= s.length <= 105
s只包含字符'a', 'b'和'c'。
```

- 解题思路

```
func minimumLength(s string) int {
        for len(s) > 0 {
                if s[0] == s[len(s)-1] && len(s) != 1 {
                        temp := string(s[0])
                        s = strings.TrimLeft(s, temp)
                        s = strings.TrimRight(s, temp)
                } else {
                        break
                }
        }
        return len(s)
}

# 2
func minimumLength(s string) int {
        left, right := 0, len(s)-1
        for left < right {
                if s[left] != s[right] {
                        break
                }
                temp := s[left]
                for left <= right && s[left] == temp {
                        left++
                }
                for left <= right && s[right] == temp {
                        right--
                }
        }
        return right - left + 1
}
```

53.21 1753. 移除石子的最大得分 (2)

- 题目

```
你正在玩一个单人游戏，面前放置着大小分别为 a、b 和 c的 三堆 石子。
每回合你都要从两个 不同的非空堆 中取出一颗石子，并在得分上加 1 分。当存在 两个或更多的空堆时，游戏停止。
给你三个整数 a 、b 和 c ，返回可以得到的 最大分数 。
示例 1：输入：a = 2, b = 4, c = 6 输出：6
解释：石子起始状态是 (2, 4, 6) ，最优的一组操作是：
- 从第一和第三堆取，石子状态现在是 (1, 4, 5)
- 从第一和第三堆取，石子状态现在是 (0, 4, 4)
- 从第二和第三堆取，石子状态现在是 (0, 3, 3)
- 从第二和第三堆取，石子状态现在是 (0, 2, 2)
- 从第二和第三堆取，石子状态现在是 (0, 1, 1)
- 从第二和第三堆取，石子状态现在是 (0, 0, 0)
总分：6 分 。
示例 2：输入：a = 4, b = 4, c = 6 输出：7
解释：石子起始状态是 (4, 4, 6) ，最优的一组操作是：
- 从第一和第二堆取，石子状态现在是 (3, 3, 6)
- 从第一和第三堆取，石子状态现在是 (2, 3, 5)
- 从第一和第三堆取，石子状态现在是 (1, 3, 4)
- 从第一和第三堆取，石子状态现在是 (0, 3, 3)
- 从第二和第三堆取，石子状态现在是 (0, 2, 2)
- 从第二和第三堆取，石子状态现在是 (0, 1, 1)
- 从第二和第三堆取，石子状态现在是 (0, 0, 0)
总分：7 分 。
示例 3：输入：a = 1, b = 8, c = 8 输出：8
解释：最优的一组操作是连续从第二和第三堆取 8 回合，直到将它们取空。
注意，由于第二和第三堆已经空了，游戏结束，不能继续从第一堆中取石子。
提示：1 <= a, b, c <= 105
```

- 解题思路

```
func maximumScore(a int, b int, c int) int {
        res := 0
        arr := []int{a, b, c}
        for {
                sort.Ints(arr)
                if arr[2] > 0 && arr[1] > 0 {
                        arr[2]--
                        arr[1]--
                        res++
                } else {
```

(续下页)

(接上页)

```
                        break
                }
        }
        return res
}

# 2
func maximumScore(a int, b int, c int) int {
        arr := []int{a, b, c}
        sort.Ints(arr)
        if arr[0]+arr[1] <= arr[2] {
                return arr[0] + arr[1]
        }
        return (arr[0] + arr[1] + arr[2]) / 2
}
```

53.22 1754. 构造字典序最大的合并字符串 (1)

- 题目

```
给你两个字符串 word1 和 word2 。
你需要按下述方式构造一个新字符串 merge ：如果 word1 或 word2 非空，选择 下面选项之一继续操作：
如果 word1 非空，将 word1 中的第一个字符附加到 merge 的末尾，并将其从 word1 中移除。
例如，word1 = "abc" 且 merge = "dv" ，在执行此选项操作之后，word1 = "bc" ，同时 merge = "dva" 。
如果 word2 非空，将 word2 中的第一个字符附加到 merge 的末尾，并将其从 word2 中移除。
例如，word2 = "abc" 且 merge = "" ，在执行此选项操作之后，word2 = "bc" ，同时 merge = "a" 。
返回你可以构造的字典序 最大 的合并字符串 merge 。
长度相同的两个字符串 a 和 b 比较字典序大小，如果在 a 和 b 出现不同的第一个位置，
a 中字符在字母表中的出现顺序位于 b 中相应字符之后，就认为字符串 a 按字典序比字符串 b 更大。
例如，"abcd" 按字典序比 "abcc" 更大，因为两个字符串出现不同的第一个位置是第四个字符，
而 d 在字母表中的出现顺序位于 c 之后。
示例 1：输入：word1 = "cabaa", word2 = "bcaaa" 输出："cbcabaaaaa"
解释：构造字典序最大的合并字符串，可行的一种方法如下所示：
- 从 word1 中取第一个字符：merge = "c", word1 = "abaa", word2 = "bcaaa"
- 从 word2 中取第一个字符：merge = "cb", word1 = "abaa", word2 = "caaa"
- 从 word2 中取第一个字符：merge = "cbc", word1 = "abaa", word2 = "aaa"
- 从 word1 中取第一个字符：merge = "cbca", word1 = "baa", word2 = "aaa"
- 从 word1 中取第一个字符：merge = "cbcab", word1 = "aa", word2 = "aaa"
```

(续下页)

(接上页)

```
- 将 word1 和 word2 中剩下的 5 个 a 附加到 merge 的末尾。
示例 2: 输入: word1 = "abcabc", word2 = "abdcaba" 输出: "abdcabcabcaba"
提示: 1 <= word1.length, word2.length <= 3000
word1 和 word2 仅由小写英文组成
```

- 解题思路

```
func largestMerge(word1 string, word2 string) string {
        res := make([]byte, len(word1)+len(word2))
        for i := 0; i < len(res); i++ {
                if word1 < word2 {
                        res[i], word2 = word2[0], word2[1:]
                } else {
                        res[i], word1 = word1[0], word1[1:]
                }
        }
        return string(res)
}
```

53.23 1759. 统计同构子字符串的数目 (3)

- 题目

```
给你一个字符串 s ，返回 s 中 同构子字符串 的数目。由于答案可能很大，只需返回对 109 +␣
↪7 取余 后的结果。
同构字符串 的定义为：如果一个字符串中的所有字符都相同，那么该字符串就是同构字符串。
子字符串 是字符串中的一个连续字符序列。
示例 1: 输入: s = "abbcccaa" 输出: 13
解释: 同构子字符串如下所列:
"a"   出现 3 次。
"aa"  出现 1 次。
"b"   出现 2 次。
"bb"  出现 1 次。
"c"   出现 3 次。
"cc"  出现 2 次。
"ccc" 出现 1 次。
3 + 1 + 2 + 1 + 3 + 2 + 1 = 13
示例 2: 输入: s = "xy" 输出: 2
解释: 同构子字符串是 "x" 和 "y" 。
示例 3: 输入: s = "zzzzz" 输出: 15
提示: 1 <= s.length <= 105
s 由小写字符串组成
```

- 解题思路

```
func countHomogenous(s string) int {
        res := 0
        left, right := 0, 0
        for right < len(s) {
                if s[left] == s[right] {
                        right++
                } else {
                        length := right - left
                        res = res + length*(length+1)/2
                        left = right
                        right++
                }
        }
        length := right - left
        res = res + length*(length+1)/2
        return res % 1000000007
}

# 2
func countHomogenous(s string) int {
        res := 0
        left, right := 0, 0
        for right < len(s) {
                if s[left] != s[right] {
                        left = right
                } else {
                        res = res + (right-left+1)%1000000007
                        right++
                }
        }
        return res % 1000000007
}

# 3
func countHomogenous(s string) int {
        res := 1
        count := 1
        for i := 1; i < len(s); i++ {
                if s[i] == s[i-1] {
                        count++
                } else {
                        count = 1
                }
```

(续下页)

(接上页)

```
                res = res + count
        }
        return res % 1000000007
}
```

53.24 1760. 袋子里最少数目的球 (3)

- 题目

```
给你一个整数数组nums，其中nums[i]表示第i个袋子里球的数目。同时给你一个整数maxOperations。
你可以进行如下操作至多maxOperations次：
选择任意一个袋子，并将袋子里的球分到2 个新的袋子中，每个袋子里都有 正整数个球。
比方说，一个袋子里有5个球，你可以把它们分到两个新袋子里，
分别有 1个和 4个球，或者分别有 2个和 3个球。
你的开销是单个袋子里球数目的 最大值，你想要 最小化开销。
请你返回进行上述操作后的最小开销。
示例 1：输入：nums = [9], maxOperations = 2 输出：3
解释：- 将装有 9 个球的袋子分成装有 6 个和 3 个球的袋子。[9] -> [6,3] 。
- 将装有 6 个球的袋子分成装有 3 个和 3 个球的袋子。[6,3] -> [3,3,3] 。
装有最多球的袋子里装有 3 个球，所以开销为 3 并返回 3 。
示例 2：输入：nums = [2,4,8,2], maxOperations = 4 输出：2
解释：- 将装有 8 个球的袋子分成装有 4 个和 4 个球的袋子。[2,4,8,2] -> [2,4,4,4,2] 。
- 将装有 4 个球的袋子分成装有 2 个和 2 个球的袋子。[2,4,4,4,2] -> [2,2,2,4,4,2] 。
- 将装有 4 个球的袋子分成装有 2 个和 2 个球的袋子。[2,2,2,4,4,2] -> [2,2,2,2,2,4,2] 。
- 将装有 4 个球的袋子分成装有 2 个和 2 个球的袋子。[2,2,2,2,2,4,2] -> [2,2,2,2,2,2,2,
↪2] 。
装有最多球的袋子里装有 2 个球，所以开销为 2 并返回 2 。
示例 3：输入：nums = [7,17], maxOperations = 2 输出：7
提示：1 <= nums.length <= 105
1 <= maxOperations, nums[i] <= 109
```

- 解题思路

```
func minimumSize(nums []int, maxOperations int) int {
        sort.Ints(nums)
        left := 1
        right := nums[len(nums)-1]
        res := 0
        for left <= right {
                mid := (left + right) / 2
                count := getCount(nums, mid)
                if count <= maxOperations {
```

(续下页)

(接上页)

```
                        res = mid
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res
}

func getCount(nums []int, per int) int {
        count := 0
        for i := 0; i < len(nums); i++ {
                if nums[i]%per == 0 {
                        count = count + nums[i]/per - 1
                } else {
                        count = count + nums[i]/per
                }
        }
        return count
}

# 2
func minimumSize(nums []int, maxOperations int) int {
        sort.Ints(nums)
        left := 1
        right := nums[len(nums)-1]
        res := 0
        for left <= right {
                mid := (left + right) / 2
                count := getCount(nums, mid)
                if count <= maxOperations {
                        res = mid
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res
}

func getCount(nums []int, per int) int {
        count := 0
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                count = count + (nums[i]-1)/per
        }
        return count
}

# 3
func minimumSize(nums []int, maxOperations int) int {
        left := 1
        right := nums[0]
        for i := 1; i < len(nums); i++ {
                if nums[i] > right {
                        right = nums[i]
                }
        }
        res := 0
        for left <= right {
                mid := (left + right) / 2
                count := getCount(nums, mid)
                if count <= maxOperations {
                        res = mid
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res
}

func getCount(nums []int, per int) int {
        count := 0
        for i := 0; i < len(nums); i++ {
                count = count + (nums[i]-1)/per
        }
        return count
}
```

53.25 1764. 通过连接另一个数组的子数组得到一个数组 (1)

- 题目

```
给你一个长度为 n的二维整数数组groups，同时给你一个整数数组nums。
你是否可以从 nums中选出 n个 不相交 的子数组，
使得第 i个子数组与 groups[i]（下标从 0开始）完全相同，且如果i > 0，
那么第(i-1)个子数组在 nums中出现的位置在第 i个子数组前面。
（也就是说，这些子数组在 nums中出现的顺序需要与 groups 顺序相同）
如果你可以找出这样的 n 个子数组，请你返回true ，否则返回false。
如果不存在下标为 k的元素 nums[k]属于不止一个子数组，就称这些子数组是 不相交 的。
子数组指的是原数组中连续元素组成的一个序列。
示例 1：输入：groups = [[1,-1,-1],[3,-2,0]], nums = [1,-1,0,1,-1,-1,3,-2,0] 输出：true
解释：你可以分别在 nums 中选出第 0 个子数组 [1,-1,0,1,-1,-1,3,-2,0]
和第 1 个子数组 [1,-1,0,1,-1,-1,3,-2,0] 。
这两个子数组是不相交的，因为它们没有任何共同的元素。
示例 2：输入：groups = [[10,-2],[1,2,3,4]], nums = [1,2,3,4,10,-2] 输出：false
解释：选择子数组 [1,2,3,4,10,-2] 和 [1,2,3,4,10,-2] 是不正确的，
因为它们出现的顺序与 groups 中顺序不同。
[10,-2] 必须出现在 [1,2,3,4] 之前。
示例 3：输入：groups = [[1,2,3],[3,4]], nums = [7,7,1,2,3,4,7,7] 输出：false
解释：选择子数组 [7,7,1,2,3,4,7,7] 和 [7,7,1,2,3,4,7,7]␣
↪是不正确的，因为它们不是不相交子数组。
它们有一个共同的元素 nums[4] （下标从 0 开始）。
提示：groups.length == n
1 <= n <= 103
1 <= groups[i].length, sum(groups[i].length) <= 103
1 <= nums.length <= 103
```

- 解题思路

```
func canChoose(groups [][]int, nums []int) bool {
        count := 0
        for i := 0; i < len(groups); i++ {
                arr := groups[i]
                flag := false
                for count+len(arr) <= len(nums) {
                        if judge(arr, nums[count:count+len(arr)]) == true {
                                count = count + len(arr)
                                flag = true
                                break
                        } else {
                                count = count + 1
                        }
```

(续下页)

(接上页)

```
                }
                if flag == false {
                        return false
                }
        }
        return true
}

func judge(a []int, b []int) bool {
        for i := 0; i < len(a); i++ {
                if a[i] != b[i] {
                        return false
                }
        }
        return true
}
```

53.26 1765. 地图中的最高点 (1)

- 题目

```
给你一个大小为m x n的整数矩阵isWater，它代表了一个由 陆地和 水域单元格组成的地图。
如果isWater[i][j] == 0，格子(i, j)是一个 陆地格子。
如果isWater[i][j] == 1，格子(i, j)是一个 水域格子。
你需要按照如下规则给每个单元格安排高度：
每个格子的高度都必须是非负的。
如果一个格子是是 水域，那么它的高度必须为 0。
任意相邻的格子高度差 至多为␣
↪1。当两个格子在正东、南、西、北方向上相互紧挨着，就称它们为相邻的格子。
（也就是说它们有一条公共边）
找到一种安排高度的方案，使得矩阵中的最高高度值最大。
请你返回一个大小为m x n的整数矩阵 height，其中 height[i][j]是格子 (i, j)的高度。
如果有多种解法，请返回任意一个。
示例 1：输入：isWater = [[0,1],[0,0]] 输出：[[1,0],[2,1]]
解释：上图展示了给各个格子安排的高度。
蓝色格子是水域格，绿色格子是陆地格。
示例 2：输入：isWater = [[0,0,1],[1,0,0],[0,0,0]] 输出：[[1,1,0],[0,1,1],[1,2,2]]
解释：所有安排方案中，最高可行高度为 2 。
任意安排方案中，只要最高高度为 2 且符合上述规则的，都为可行方案。
提示：m == isWater.length
n == isWater[i].length
1 <= m, n <= 1000
```

(续下页)

(接上页)

```
isWater[i][j]要么是0，要么是1。
至少有 1个水域格子。
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func highestPeak(isWater [][]int) [][]int {
    n := len(isWater)
    m := len(isWater[0])
    res := make([][]int, n)
    for i := 0; i < n; i++ {
            res[i] = make([]int, m)
    }
    queue := make([][2]int, 0)
    for i := 0; i < n; i++ {
            for j := 0; j < m; j++ {
                    if isWater[i][j] == 1 {
                            res[i][j] = -1
                            queue = append(queue, [2]int{i, j})
                    }
            }
    }
    level := 0
    for len(queue) > 0 {
            level++
            length := len(queue)
            for i := 0; i < length; i++ {
                    a, b := queue[i][0], queue[i][1]
                    for j := 0; j < 4; j++ {
                            x := a + dx[j]
                            y := b + dy[j]
                            if 0 <= x && x < n && 0 <= y && y < m && res[x][y] == 0 {
                                    res[x][y] = level
                                    queue = append(queue, [2]int{x, y})
                            }
                    }
            }
            queue = queue[length:]
    }
    for i := 0; i < n; i++ {
            for j := 0; j < m; j++ {
                    if res[i][j] == -1 {
```

(续下页)

(接上页)

```
                            res[i][j] = 0
                    }
            }
    }
    return res
}
```

53.27 1769. 移动所有球到每个盒子所需的最小操作数 (3)

- 题目

```
有 n 个盒子。给你一个长度为 n 的二进制字符串 boxes ，
其中 boxes[i] 的值为 '0' 表示第 i 个盒子是 空 的，而 boxes[i] 的值为 '1' 表示盒子里有␣
↪一个 小球。
在一步操作中，你可以将 一个 小球从某个盒子移动到一个与之相邻的盒子中。
第 i 个盒子和第 j 个盒子相邻需满足 abs(i - j) == 1 。
注意，操作执行后，某些盒子中可能会存在不止一个小球。
返回一个长度为 n 的数组 answer ，其中 answer[i] 是将所有小球移动到第 i 个盒子所需的␣
↪最小 操作数。
每个 answer[i] 都需要根据盒子的 初始状态 进行计算。
示例 1：输入：boxes = "110" 输出：[1,1,3]
解释：每个盒子对应的最小操作数如下：
1) 第 1 个盒子：将一个小球从第 2 个盒子移动到第 1 个盒子，需要 1 步操作。
2) 第 2 个盒子：将一个小球从第 1 个盒子移动到第 2 个盒子，需要 1 步操作。
3) 第 3 个盒子：将一个小球从第 1 个盒子移动到第 3 个盒子，需要 2␣
↪步操作。将一个小球从第 2 个盒子移动到第 3 个盒子，需要 1 步操作。共计 3 步操作。
示例 2：输入：boxes = "001011" 输出：[11,8,5,4,3,4]
提示：n == boxes.length
1 <= n <= 2000
boxes[i] 为 '0' 或 '1'
```

- 解题思路

```
func minOperations(boxes string) []int {
        n := len(boxes)
        res := make([]int, n)
        right, rightCount := 0, 0
        for i := 0; i < n; i++ {
                if boxes[i] == '1' {
                        right = right + i
                        rightCount++
                }
```

(续下页)

(接上页)

```
        }
        left, leftCount := 0, 0
        for i := 0; i < n; i++ {
                res[i] = left + right
                if boxes[i] == '1' {
                        leftCount++
                        rightCount--
                }
                left = left + leftCount
                right = right - rightCount
        }
        return res
}

# 2
func minOperations(boxes string) []int {
        n := len(boxes)
        res := make([]int, n)
        pre := make([]int, n)
        count, sum := 0, 0
        for i := 0; i < n; i++ {
                pre[i] = sum
                if boxes[i] == '1' {
                        count++
                }
                sum = sum + count
        }
        suf := make([]int, n)
        count, sum = 0, 0
        for i := n - 1; i >= 0; i-- {
                suf[i] = sum
                if boxes[i] == '1' {
                        count++
                }
                sum = sum + count
        }
        for i := 0; i < n; i++ {
                res[i] = pre[i] + suf[i]
        }
        return res
}

# 3
```

(续下页)

(接上页)

```
func minOperations(boxes string) []int {
        n := len(boxes)
        res := make([]int, n)
        arr := make([]int, 0)
        for i := 0; i < n; i++ {
                if boxes[i] == '1' {
                        arr = append(arr, i)
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < len(arr); j++ {
                        res[i] = res[i] + abs(arr[j]-i)
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

53.28 1770. 执行乘法运算的最大分数 (3)

- 题目

```
给你两个长度分别 n 和 m 的整数数组 nums 和 multipliers ，其中 n >= m ，数组下标 从 1␣
↪开始 计数。
初始时，你的分数为 0 。你需要执行恰好 m 步操作。在第 i 步操作（从 1 开始␣
↪计数）中，需要：
选择数组 nums 开头处或者末尾处 的整数 x 。
你获得 multipliers[i] * x 分，并累加到你的分数中。
将 x 从数组 nums 中移除。
在执行 m 步操作后，返回 最大 分数。
示例 1：输入：nums = [1,2,3], multipliers = [3,2,1] 输出：14
解释：一种最优解决方案如下：
- 选择末尾处的整数 3 ，[1,2,3] ，得 3 * 3 = 9 分，累加到分数中。
- 选择末尾处的整数 2 ，[1,2] ，得 2 * 2 = 4 分，累加到分数中。
- 选择末尾处的整数 1 ，[1] ，得 1 * 1 = 1 分，累加到分数中。
总分数为 9 + 4 + 1 = 14 。
```

(续下页)

(接上页)

```
示例 2：输入：nums = [-5,-3,-3,-2,7,1], multipliers = [-10,-5,3,4,6] 输出：102
解释：一种最优解决方案如下：
- 选择开头处的整数 -5 ，[-5,-3,-3,-2,7,1] ，得 -5 * -10 = 50 分，累加到分数中。
- 选择开头处的整数 -3 ，[-3,-3,-2,7,1] ，得 -3 * -5 = 15 分，累加到分数中。
- 选择开头处的整数 -3 ，[-3,-2,7,1] ，得 -3 * 3 = -9 分，累加到分数中。
- 选择末尾处的整数 1 ，[-2,7,1] ，得 1 * 4 = 4 分，累加到分数中。
- 选择末尾处的整数 7 ，[-2,7] ，得 7 * 6 = 42 分，累加到分数中。
总分数为 50 + 15 - 9 + 4 + 42 = 102 。
提示：n == nums.length
m == multipliers.length
1 <= m <= 103
m <= n <= 105
-1000 <= nums[i], multipliers[i] <= 1000
```

- 解题思路

```
func maximumScore(nums []int, multipliers []int) int {
        n, m := len(nums), len(multipliers)
        dp := make([][]int, m+1) // dp[i][j] 左i右j的值
        for i := 0; i <= m; i++ {
                dp[i] = make([]int, m+1)
        }
        dp[0][0] = 0
        for i := 1; i <= m; i++ {
                dp[i][0] = dp[i-1][0] + nums[i-1]*multipliers[i-1] // 左i右0
                dp[0][i] = dp[0][i-1] + nums[n-i]*multipliers[i-1] // 左0右i
        }
        for i := 1; i <= m; i++ {
                for j := 1; i+j <= m; j++ {
                        left := dp[i-1][j] + nums[i-1]*multipliers[i+j-1]
                        right := dp[i][j-1] + nums[n-j]*multipliers[i+j-1]
                        dp[i][j] = max(left, right)
                }
        }
        res := math.MinInt32
        for i := 0; i <= m; i++ {
                res = max(res, dp[i][m-i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

# 2
func maximumScore(nums []int, multipliers []int) int {
        n, m := len(nums), len(multipliers)
        dp := make([][]int, m+1) // dp[i][j] 左i右j的值
        for i := 0; i <= m; i++ {
                dp[i] = make([]int, m+1)
        }
        res := math.MinInt32
        for k := 1; k <= m; k++ {
                for i := 0; i <= k; i++ {
                        left := i
                        right := k - i
                        if i == 0 {
                                dp[left][right] = dp[left][right-1] + nums[n-
↪right]*multipliers[k-1]
                        } else if i == k {
                                dp[left][right] = dp[left-1][right] + nums[left-
↪1]*multipliers[k-1]
                        } else {
                                l := dp[left][right-1] + nums[n-right]*multipliers[k-
↪1]
                                r := dp[left-1][right] + nums[left-1]*multipliers[k-1]
                                dp[left][right] = max(l, r)
                        }
                        if k == m {
                                res = max(res, dp[left][right])
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 3
var dp [][]int

func maximumScore(nums []int, multipliers []int) int {
        n, m := len(nums), len(multipliers)
        dp = make([][]int, m) // dp[i][j] 左i右j的值
        for i := 0; i < m; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        dp[i][j] = math.MinInt32
                }
        }
        res := dfs(nums, multipliers, 0, n-1)
        return res
}

func dfs(nums []int, multipliers []int, i, j int) int {
        n, m := len(nums), len(multipliers)
        if dp[i][j-n+m] != math.MinInt32 {
                return dp[i][j-n+m]
        }
        if j-i == n-m {
                dp[i][j-n+m] = max(dp[i][j-n+m], nums[i]*multipliers[m-1])
                dp[i][j-n+m] = max(dp[i][j-n+m], nums[j]*multipliers[m-1])
                return dp[i][j-n+m]
        }
        k := i + n - 1 - j
        left := dfs(nums, multipliers, i+1, j) + nums[i]*multipliers[k]
        right := dfs(nums, multipliers, i, j-1) + nums[j]*multipliers[k]
        dp[i][j-n+m] = max(left, right)
        return dp[i][j-n+m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

53.29 1774. 最接近目标价格的甜点成本 (2)

- 题目

```
你打算做甜点，现在需要购买配料。目前共有 n 种冰激凌基料和 m 种配料可供选购。
而制作甜点需要遵循以下几条规则：
必须选择 一种 冰激凌基料。
可以添加 一种或多种 配料，也可以不添加任何配料。
每种类型的配料 最多两份 。
给你以下三个输入：
baseCosts ，一个长度为 n 的整数数组，其中每个 baseCosts[i] 表示第 i␣
↪种冰激凌基料的价格。
toppingCosts，一个长度为 m 的整数数组，其中每个 toppingCosts[i] 表示 一份 第 i␣
↪种冰激凌配料的价格。
target ，一个整数，表示你制作甜点的目标价格。
你希望自己做的甜点总成本尽可能接近目标价格 target 。
返回最接近 target 的甜点成本。如果有多种方案，返回成本相对较低 的一种。
示例 1：输入：baseCosts = [1,7], toppingCosts = [3,4], target = 10 输出：10
解释：考虑下面的方案组合（所有下标均从 0 开始）：
- 选择 1 号基料：成本 7
- 选择 1 份 0 号配料：成本 1 x 3 = 3
- 选择 0 份 1 号配料：成本 0 x 4 = 0
总成本：7 + 3 + 0 = 10 。
示例 2：输入：baseCosts = [2,3], toppingCosts = [4,5,100], target = 18 输出：17
解释：考虑下面的方案组合（所有下标均从 0 开始）：
- 选择 1 号基料：成本 3
- 选择 1 份 0 号配料：成本 1 x 4 = 4
- 选择 2 份 1 号配料：成本 2 x 5 = 10
- 选择 0 份 2 号配料：成本 0 x 100 = 0
总成本：3 + 4 + 10 + 0 = 17 。不存在总成本为 18 的甜点制作方案。
示例 3：输入：baseCosts = [3,10], toppingCosts = [2,5], target = 9 输出：8
解释：可以制作总成本为 8 和 10 的甜点。返回 8 ，因为这是成本更低的方案。
示例 4：输入：baseCosts = [10], toppingCosts = [1], target = 1 输出：10
解释：注意，你可以选择不添加任何配料，但你必须选择一种基料。
提示：n == baseCosts.length
m == toppingCosts.length
1 <= n, m <= 10
1 <= baseCosts[i], toppingCosts[i] <= 104
1 <= target <= 104
```

- 解题思路

```
var res int
```

(续下页)

(接上页)

```
func closestCost(baseCosts []int, toppingCosts []int, target int) int {
        res = math.MaxInt32
        for i := 0; i < len(baseCosts); i++ {
                dfs(toppingCosts, target, baseCosts[i], 0)
        }
        return res
}

func dfs(toppingCosts []int, target int, sum int, index int) {
        if abs(res-target) > abs(sum-target) {
                res = sum
        } else if abs(res-target) == abs(sum-target) && sum < res {
                res = sum
        }
        if sum > target {
                return
        }
        if index == len(toppingCosts) {
                return
        }
        dfs(toppingCosts, target, sum, index+1)
        dfs(toppingCosts, target, sum+toppingCosts[index], index+1)
        dfs(toppingCosts, target, sum+2*toppingCosts[index], index+1)
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func closestCost(baseCosts []int, toppingCosts []int, target int) int {
        res := math.MaxInt32
        n, m := len(baseCosts), len(toppingCosts)
        for i := 0; i < n; i++ {
                for j := 0; j < (1 << m); j++ { // 选择第1次
                        for k := j; k < (1 << m); k++ { // 选择第2次
                                total := baseCosts[i]
                                for l := 0; l < m; l++ {
                                        if j&(1<<l) != 0 {
                                                total = total + toppingCosts[l]
```

(续下页)

(接上页)

```
                                        }
                                        if k&(1<<l) != 0 {
                                                total = total + toppingCosts[l]
                                        }
                                }
                                if abs(res-target) > abs(total-target) {
                                        res = total
                                } else if abs(res-target) == abs(total-target) &&␣
↪total < res {
                                        res = total
                                }
                        }
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

53.30 1775. 通过最少操作次数使数组的和相等 (2)

- 题目

```
给你两个长度可能不等的整数数组nums1␣
↪和nums2。两个数组中的所有值都在1到6之间（包含1和6）。
每次操作中，你可以选择 任意数组中的任意一个整数，将它变成 1到 6之间 任意的值（包含␣
↪1和 6）。
请你返回使 nums1中所有数的和与nums2中所有数的和相等的最少操作次数。
如果无法使两个数组的和相等，请返回 -1。
示例 1：输入：nums1 = [1,2,3,4,5,6], nums2 = [1,1,2,2,2,2] 输出：3
解释：你可以通过 3 次操作使 nums1 中所有数的和与 nums2␣
↪中所有数的和相等。以下数组下标都从 0 开始。
- 将 nums2[0] 变为 6 。 nums1 = [1,2,3,4,5,6], nums2 = [6,1,2,2,2,2] 。
- 将 nums1[5] 变为 1 。 nums1 = [1,2,3,4,5,1], nums2 = [6,1,2,2,2,2] 。
- 将 nums1[2] 变为 2 。 nums1 = [1,2,2,4,5,1], nums2 = [6,1,2,2,2,2] 。
示例 2：输入：nums1 = [1,1,1,1,1,1,1], nums2 = [6] 输出：-1
解释：没有办法减少 nums1 的和或者增加 nums2 的和使二者相等。
```

(续下页)

(接上页)

```
示例 3：输入：nums1 = [6,6], nums2 = [1] 输出：3
解释：你可以通过 3 次操作使 nums1 中所有数的和与 nums2␣
↪中所有数的和相等。以下数组下标都从 0 开始。
- 将 nums1[0] 变为 2 。 nums1 = [2,6], nums2 = [1] 。
- 将 nums1[1] 变为 2 。 nums1 = [2,2], nums2 = [1] 。
- 将 nums2[0] 变为 4 。 nums1 = [2,2], nums2 = [4] 。
提示：1 <= nums1.length, nums2.length <= 105
1 <= nums1[i], nums2[i] <= 6
```

- 解题思路

```
func minOperations(nums1 []int, nums2 []int) int {
        aMin, bMin := len(nums1), len(nums2)
        aMax, bMax := aMin*6, bMin*6
        if aMin > bMax || aMax < bMin {
                return -1
        }
        a, b := 0, 0
        arrA, arrB := [7]int{}, [7]int{}
        for i := 0; i < len(nums1); i++ {
                a = a + nums1[i]
                arrA[nums1[i]]++
        }
        for i := 0; i < len(nums2); i++ {
                b = b + nums2[i]
                arrB[nums2[i]]++
        }
        if a == b {
                return 0
        }
        if b > a {
                arrA, arrB = arrB, arrA
                a, b = b, a
        }
        arr := make([]int, 0) // 存储
        for i := 1; i < 6; i++ {
                if arrA[7-i] > arrB[i] {
                        arr = append(arr, arrA[7-i], arrB[i])
                } else {
                        arr = append(arr, arrB[i], arrA[7-i])
                }
        }
        res := 0
        total := a - b
```

(续下页)

(接上页)

```
        for i := 0; i < len(arr); i++ {
                diff := 6 - (i+2)/2
                if total-diff*arr[i] > 0 {
                        total = total - diff*arr[i]
                        res = res + arr[i]
                } else {
                        if total%diff == 0 {
                                res = res + total/diff
                        } else {
                                res = res + total/diff + 1
                        }
                        return res
                }
        }
        return res
}

# 2
func minOperations(nums1 []int, nums2 []int) int {
        aMin, bMin := len(nums1), len(nums2)
        aMax, bMax := aMin*6, bMin*6
        if aMin > bMax || aMax < bMin {
                return -1
        }
        a, b := 0, 0
        arrA, arrB := [7]int{}, [7]int{}
        for i := 0; i < len(nums1); i++ {
                a = a + nums1[i]
                arrA[nums1[i]]++
        }
        for i := 0; i < len(nums2); i++ {
                b = b + nums2[i]
                arrB[nums2[i]]++
        }
        if a == b {
                return 0
        }
        if b > a {
                arrA, arrB = arrB, arrA
                a, b = b, a
        }
        arr := make([]int, 6) // 存储
        for i := 1; i < 6; i++ {
```

(续下页)

(接上页)

```
                arr[i-1] = arr[i-1] + arrA[7-i] + arrB[i]
        }
        res, total := 0, a-b
        for i := 0; i < len(arr); i++ {
                diff := 5 - i
                if total-diff*arr[i] > 0 {
                        total = total - diff*arr[i]
                        res = res + arr[i]
                } else {
                        if total%diff == 0 {
                                res = res + total/diff
                        } else {
                                res = res + total/diff + 1
                        }
                        return res
                }
        }
        return res
}
```

53.31 1780. 判断一个数字是否可以表示成三的幂的和 (2)

- 题目

```
给你一个整数n，如果你可以将n表示成若干个不同的三的幂之和，请你返回true，否则请返回␣
↪false。
对于一个整数 y，如果存在整数 x满足 y == 3x，我们称这个整数 y是三的幂。
示例 1：输入：n = 12 输出：true
解释：12 = 31 + 32
示例 2：输入：n = 91 输出：true
解释：91 = 30 + 32 + 34
示例 3：输入：n = 21 输出：false
提示：1 <= n <= 107
```

- 解题思路

```
func checkPowersOfThree(n int) bool {
        arr := make([]int, 0)
        arr = append(arr, 1)
        sum := 1
        for i := 0; i < 15; i++ {
                sum = sum * 3
```

(续下页)

(接上页)

```
                arr = append(arr, sum)
        }
        for i := len(arr) - 1; i >= 0; i-- {
                if n > arr[i] {
                        n = n - arr[i]
                } else if n == arr[i] {
                        return true
                }
        }
        return false
}

# 2
func checkPowersOfThree(n int) bool {
        for n > 0 {
                if n%3 == 2 { // 转换为3进制，n*3^x，其中n为0或者1，不会为2
                        return false
                }
                n = n / 3
        }
        return true
}
```

53.32 1781. 所有子字符串美丽值之和 (1)

- 题目

```
一个字符串的 美丽值定义为：出现频率最高字符与出现频率最低字符的出现次数之差。
比方说，"abaacc"的美丽值为3 - 1 = 2。
给你一个字符串s，请你返回它所有子字符串的美丽值之和。
示例 1：输入：s = "aabcb" 输出：5
解释：美丽值不为零的字符串包括 ["aab","aabc","aabcb","abcb","bcb"]␣
↪，每一个字符串的美丽值都为 1 。
示例 2：输入：s = "aabcbaa" 输出：17
提示：1 <= s.length <= 500
s只包含小写英文字母。
```

- 解题思路

```
func beautySum(s string) int {
        res := 0
        for i := 0; i < len(s); i++ {
```

(续下页)

(接上页)

```
                arr := [26]int{}
                arr[s[i]-'a']++
                for j := i + 2; j <= len(s); j++ {
                        arr[s[j-1]-'a']++
                        res = res + getCount(arr)
                }
        }
        return res
}

func getCount(arr [26]int) int {
        maxValue, minValue := math.MinInt32, math.MaxInt32
        for i := 0; i < 26; i++ {
                if arr[i] > 0 {
                        if arr[i] > maxValue {
                                maxValue = arr[i]
                        }
                        if arr[i] < minValue {
                                minValue = arr[i]
                        }
                }
        }
        return maxValue - minValue
}
```

53.33 1785. 构成特定和需要添加的最少元素 (2)

- 题目

```
给你一个整数数组 nums ，和两个整数 limit 与 goal 。数组 nums 有一条重要属性：
abs(nums[i]) <= limit 。
返回使数组元素总和等于 goal 所需要向数组中添加的 最少元素数量 ，
添加元素 不应改变 数组中 abs(nums[i]) <= limit 这一属性。
注意，如果 x >= 0 ，那么 abs(x) 等于 x ；否则，等于 -x 。
示例 1：输入：nums = [1,-1,1], limit = 3, goal = -4 输出：2
解释：可以将 -2 和 -3 添加到数组中，数组的元素总和变为 1 - 1 + 1 - 2 - 3 = -4 。
示例 2：输入：nums = [1,-10,9,1], limit = 100, goal = 0 输出：1
提示：1 <= nums.length <= 105
1 <= limit <= 106
-limit <= nums[i] <= limit
-109 <= goal <= 109
```

- 解题思路

```
func minElements(nums []int, limit int, goal int) int {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        left := abs(goal - sum)
        a := left / limit
        b := left % limit
        if b != 0 {
                a = a + 1
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func minElements(nums []int, limit int, goal int) int {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        res := abs(goal - sum)
        return (res - 1 + limit) / limit
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

53.34 1786. 从第一个节点出发到最后一个节点的受限路径数

53.34.1 题目

```
现有一个加权无向连通图。给你一个正整数 n ，表示图中有 n 个节点，并按从 1 到 n␣
↪给节点编号；
另给你一个数组 edges ，其中每个 edges[i] = [ui, vi, weighti] 表示存在一条位于节点 ui␣
↪和 vi 之间的边，
这条边的权重为 weighti 。
从节点 start 出发到节点 end 的路径是一个形如 [z0, z1, z2, ..., zk] 的节点序列，
满足 z0 = start 、zk = end 且在所有符合 0 <= i <= k-1 的节点 zi 和 zi+1␣
↪之间存在一条边。
路径的距离定义为这条路径上所有边的权重总和。用 distanceToLastNode(x) 表示节点 n 和 x␣
↪之间路径的最短距离。
受限路径 为满足 distanceToLastNode(zi) > distanceToLastNode(zi+1) 的一条路径，其中 0
↪<= i <= k-1 。
返回从节点 1 出发到节点 n 的 受限路径数 。由于数字可能很大，请返回对 109 + 7 取余␣
↪的结果。
示例 1：输入：n = 5, edges = [[1,2,3],[1,3,3],[2,3,1],[1,4,2],[5,2,2],[3,5,1],[5,4,
↪10]] 输出：3
解释：每个圆包含黑色的节点编号和蓝色的 distanceToLastNode 值。三条受限路径分别是：
1) 1 --> 2 --> 5
2) 1 --> 2 --> 3 --> 5
3) 1 --> 3 --> 5
示例 2：输入：n = 7, edges = [[1,3,1],[4,1,2],[7,3,4],[2,5,3],[5,6,1],[6,7,2],[7,5,3],
↪[2,6,4]] 输出：1
解释：每个圆包含黑色的节点编号和蓝色的 distanceToLastNode 值。唯一一条受限路径是：1 --
↪> 3 --> 7 。
提示：1 <= n <= 2 * 104
n - 1 <= edges.length <= 4 * 104
edges[i].length == 3
1 <= ui, vi <= n
ui != vi
1 <= weighti <= 105
任意两个节点之间至多存在一条边
任意两个节点之间至少存在一条路径
```

53.34.2 解题思路

53.35 1791. 找出星型图的中心节点 (2)

- 题目

```
有一个无向的 星型 图，由 n 个编号从 1 到 n 的节点组成。
星型图有一个 中心 节点，并且恰有 n - 1 条边将中心节点与其他每个节点连接起来。
给你一个二维整数数组 edges ，其中edges[i] = [ui, vi] 表示在节点 ui 和 vi␣
↪之间存在一条边。
请你找出并返回edges 所表示星型图的中心节点。
示例 1：输入：edges = [[1,2],[2,3],[4,2]] 输出：2
解释：如上图所示，节点 2 与其他每个节点都相连，所以节点 2 是中心节点。
示例 2：输入：edges = [[1,2],[5,1],[1,3],[1,4]] 输出：1
提示：3 <= n <= 105
edges.length == n - 1
edges[i].length == 2
1 <= ui, vi <= n
ui != vi
题目数据给出的 edges 表示一个有效的星型图
```

- 解题思路

```
func findCenter(edges [][]int) int {
        m := make(map[int]int)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                m[a]++
                m[b]++
        }
        for k, v := range m {
                if v == len(edges) {
                        return k
                }
        }
        return -1
}

# 2
func findCenter(edges [][]int) int {
        if edges[0][0] == edges[1][0] || edges[0][0] == edges[1][1] {
```

(续下页)

(接上页)

```
                return edges[0][0]
        }
        return edges[0][1]
}
```

53.36 1792. 最大平均通过率 (1)

- 题目

```
一所学校里有一些班级，每个班级里有一些学生，现在每个班都会进行一场期末考试。
给你一个二维数组 classes，其中classes[i] = [passi, totali]，
表示你提前知道了第i个班级总共有totali个学生，其中只有passi个学生可以通过考试。
给你一个整数extraStudents，表示额外有extraStudents个聪明的学生，
他们 一定能通过任何班级的期末考。
你需要给这extraStudents个学生每人都安排一个班级，使得 所有班级的 平均通过率 最大。
一个班级的通过率等于这个班级通过考试的学生人数除以这个班级的总人数。
平均通过率是所有班级的通过率之和除以班级数目。
请你返回在安排这 extraStudents 个学生去对应班级后的 最大平均通过率。
与标准答案误差范围在10-5以内的结果都会视为正确结果。
示例 1：输入：classes = [[1,2],[3,5],[2,2]], extraStudents = 2 输出：0.78333
解释：你可以将额外的两个学生都安排到第一个班级，平均通过率为 (3/4 + 3/5 + 2/2) / 3 =␣
↪0.78333 。
示例 2：输入：classes = [[2,4],[3,9],[4,5],[2,10]], extraStudents = 4 输出：0.53485
提示：1 <= classes.length <= 105
classes[i].length == 2
1 <= passi <= totali <= 105
1 <= extraStudents <= 105
```

- 解题思路

```
func maxAverageRatio(classes [][]int, extraStudents int) float64 {
        nodeHeap := make(NodeHeap, 0)
        heap.Init(&nodeHeap)
        for i := 0; i < len(classes); i++ {
                x, y := float64(classes[i][0]), float64(classes[i][1])
                a := x / y
                b := (x + 1) / (y + 1)
                heap.Push(&nodeHeap, Node{
                        id:    i,
                        ratio: b - a,
                })
        }
```

(续下页)

(接上页)

```
        for i := 0; i < extraStudents; i++ {
                node := heap.Pop(&nodeHeap).(Node)
                id := node.id
                classes[id][0]++
                classes[id][1]++
                x, y := float64(classes[id][0]), float64(classes[id][1])
                a := x / y
                b := (x + 1) / (y + 1)
                heap.Push(&nodeHeap, Node{
                        id:    id,
                        ratio: b - a,
                })
        }
        sum := float64(0)
        for i := 0; i < len(classes); i++ {
                x, y := float64(classes[i][0]), float64(classes[i][1])
                sum = sum + x/y
        }
        return sum / float64(len(classes))
}

type Node struct {
        id    int
        ratio float64
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h NodeHeap) Less(i, j int) bool {
        return h[i].ratio > h[j].ratio
}

func (h NodeHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
```

(续下页)

(接上页)

```
}

func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

53.37 1797. 设计一个验证系统 (2)

- 题目

```
你需要设计一个包含验证码的验证系统。每一次验证中，用户会收到一个新的验证码，
这个验证码在 currentTime时刻之后 timeToLive秒过期。
如果验证码被更新了，那么它会在 currentTime（可能与之前的␣
↪currentTime不同）时刻延长timeToLive秒。
请你实现AuthenticationManager类：
AuthenticationManager(int timeToLive)构造AuthenticationManager并设置timeToLive参数。
generate(string tokenId, int currentTime)给定 tokenId，
在当前时间currentTime 生成一个新的验证码。
renew(string tokenId, int currentTime)将给定 tokenId
且 未过期的验证码在 currentTime时刻更新。
如果给定tokenId对应的验证码不存在或已过期，请你忽略该操作，不会有任何更新操作发生。
countUnexpiredTokens(int currentTime)请返回在给定currentTime时刻，未过期的验证码数目。
如果一个验证码在时刻t过期，且另一个操作恰好在时刻t发生（renew或者countUnexpiredTokens操作），
过期事件优先于其他操作。
示例 1：输入：["AuthenticationManager", "renew", "generate",
"countUnexpiredTokens", "generate", "renew", "renew", "countUnexpiredTokens"]
[[5], ["aaa", 1], ["aaa", 2], [6], ["bbb", 7], ["aaa", 8], ["bbb", 10], [15]]
输出：[null, null, null, 1, null, null, null, 0]
解释：AuthenticationManager authenticationManager = new AuthenticationManager(5);
// 构造 AuthenticationManager ，设置 timeToLive = 5 秒。
authenticationManager.renew("aaa", 1);
// 时刻 1 时，没有验证码的 tokenId 为 "aaa" ，没有验证码被更新。
authenticationManager.generate("aaa", 2);
// 时刻 2 时，生成一个 tokenId 为 "aaa" 的新验证码。
authenticationManager.countUnexpiredTokens(6);
// 时刻 6 时，只有 tokenId 为 "aaa" 的验证码未过期，所以返回 1 。
authenticationManager.generate("bbb", 7);
// 时刻 7 时，生成一个 tokenId 为 "bbb" 的新验证码。
authenticationManager.renew("aaa", 8);
// tokenId 为 "aaa" 的验证码在时刻 7 过期，
```

(续下页)

(接上页)

```
且 8 >= 7 ，所以时刻 8 的renew 操作被忽略，没有验证码被更新。
authenticationManager.renew("bbb", 10);
// tokenId 为 "bbb" 的验证码在时刻 10 没有过期，所以 renew 操作会执行，该 token␣
↪将在时刻 15 过期。
authenticationManager.countUnexpiredTokens(15);
// tokenId 为 "bbb" 的验证码在时刻 15 过期，
tokenId 为 "aaa" 的验证码在时刻 7 过期，所有验证码均已过期，所以返回 0 。
提示：1 <= timeToLive <= 108
1 <= currentTime <= 108
1 <= tokenId.length <= 5
tokenId只包含小写英文字母。
所有generate函数的调用都会包含独一无二的tokenId值。
所有函数调用中，currentTime的值 严格递增。
所有函数的调用次数总共不超过2000次。
```

- 解题思路

```
type AuthenticationManager struct {
        m          map[string]int
        timeToLive int
}

func Constructor(timeToLive int) AuthenticationManager {
        return AuthenticationManager{
                m:          make(map[string]int),
                timeToLive: timeToLive,
        }
}

func (this *AuthenticationManager) Generate(tokenId string, currentTime int) {
        this.m[tokenId] = currentTime + this.timeToLive
}

func (this *AuthenticationManager) Renew(tokenId string, currentTime int) {
        if v, ok := this.m[tokenId]; ok {
                if v <= currentTime {
                        delete(this.m, tokenId)
                } else {
                        this.m[tokenId] = currentTime + this.timeToLive
                }
        }
}

func (this *AuthenticationManager) CountUnexpiredTokens(currentTime int) int {
```

(续下页)

(接上页)

```
        count := 0
        arr := make([]string, 0)
        for k, v := range this.m {
                if v > currentTime {
                        count++
                } else {
                        arr = append(arr, k)
                }
        }
        for i := 0; i < len(arr); i++ {
                delete(this.m, arr[i])
        }
        return count
}

# 2
type AuthenticationManager struct {
        m          map[string]int
        timeToLive int
}

func Constructor(timeToLive int) AuthenticationManager {
        return AuthenticationManager{
                m:          make(map[string]int),
                timeToLive: timeToLive,
        }
}

func (this *AuthenticationManager) Generate(tokenId string, currentTime int) {
        this.m[tokenId] = currentTime + this.timeToLive
}

func (this *AuthenticationManager) Renew(tokenId string, currentTime int) {
        if v, ok := this.m[tokenId]; ok {
                if v <= currentTime {
                        delete(this.m, tokenId)
                } else {
                        this.m[tokenId] = currentTime + this.timeToLive
                }
        }
}

func (this *AuthenticationManager) CountUnexpiredTokens(currentTime int) int {
```

(续下页)

(接上页)

```
        count := 0
        for _, v := range this.m {
                if v > currentTime {
                        count++
                }
        }
        return count
}
```

53.38 1798. 你能构造出连续值的最大数目 (2)

- 题目

```
给你一个长度为 n的整数数组coins，它代表你拥有的n个硬币。第i个硬币的值为coins[i]。
如果你从这些硬币中选出一部分硬币，它们的和为x，那么称，你可以构造出x。
请返回从 0开始（包括0），你最多能构造出多少个连续整数。
你可能有多个相同值的硬币。
示例 1：输入：coins = [1,3] 输出：2
解释：你可以得到以下这些值：
- 0：什么都不取 []
- 1：取 [1]
从 0 开始，你可以构造出 2 个连续整数。
示例 2：输入：coins = [1,1,1,4] 输出：8
解释：你可以得到以下这些值：
- 0：什么都不取 []
- 1：取 [1]
- 2：取 [1,1]
- 3：取 [1,1,1]
- 4：取 [4]
- 5：取 [4,1]
- 6：取 [4,1,1]
- 7：取 [4,1,1,1]
从 0 开始，你可以构造出 8 个连续整数。
示例 3：输入：nums = [1,4,10,3,1] 输出：20
提示：coins.length == n
1 <= n <= 4 * 104
1 <= coins[i] <= 4 * 104
```

53.38.1 解题思路

```
func getMaximumConsecutive(coins []int) int {
        sort.Ints(coins)
        res := 1
        target := 1
        sum := 0
        for i := 0; i < len(coins); i++ {
                sum = sum + coins[i]
                if coins[i] <= target {
                        target = sum + 1
                        res = target
                } else {
                        break
                }
        }
        return res
}

# 2
func getMaximumConsecutive(coins []int) int {
        sort.Ints(coins)
        res := 0
        for i := 0; i < len(coins); i++ {
                if res >= coins[i]-1 {
                        res = res + coins[i]
                }else {
                        break
                }
        }
        return res + 1
}
```

CHAPTER 54

1701-1800-Hard

54.1 1703. 得到连续 K 个 1 的最少相邻交换次数

54.1.1 题目

```
给你一个整数数组nums和一个整数k。nums 仅包含0和1。每一次移动，你可以选择␣
↪相邻两个数字并将它们交换。
请你返回使nums中包含k个 连续1的 最少交换次数。
示例 1：输入：nums = [1,0,0,1,0,1], k = 2 输出：1
解释：在第一次操作时，nums 可以变成 [1,0,0,0,1,1] 得到连续两个 1 。
示例 2：输入：nums = [1,0,0,0,0,0,1,1], k = 3 输出：5
解释：通过 5 次操作，最左边的 1 可以移到右边直到 nums 变为 [0,0,0,0,0,1,1,1] 。
示例 3：输入：nums = [1,1,0,1], k = 2 输出：0
解释：nums 已经有连续 2 个 1 了。
提示：1 <= nums.length <= 105
nums[i] 要么是0，要么是1。
1 <= k <= sum(nums)
```

54.1.2 解题思路

54.2 1707. 与数组中元素的最大异或值 (2)

- 题目

```
给你一个由非负整数组成的数组 nums 。另有一个查询数组 queries ，其中 queries[i] = [xi,␣
↪mi] 。
第 i 个查询的答案是 xi 和任何 nums 数组中不超过 mi 的元素按位异或 (XOR) 得到的最大值。
换句话说，答案是 max(nums[j] XOR xi) ，其中所有 j 均满足 nums[j] <= mi 。
如果 nums 中的所有元素都大于 mi，最终答案就是 -1 。
返回一个整数数组 answer 作为查询的答案，其中 answer.length == queries.length 且␣
↪answer[i] 是第 i 个查询的答案。
示例 1：输入：nums = [0,1,2,3,4], queries = [[3,1],[1,3],[5,6]] 输出：[3,3,7]
解释：1) 0 和 1 是仅有的两个不超过 1 的整数。0 XOR 3 = 3 而 1 XOR 3 = 2␣
↪。二者中的更大值是 3 。
2) 1 XOR 2 = 3.
3) 5 XOR 2 = 7.
示例 2：输入：nums = [5,2,4,6,6,3], queries = [[12,4],[8,1],[6,3]] 输出：[15,-1,5]
提示：1 <= nums.length, queries.length <= 105
queries[i].length == 2
0 <= nums[j], xi, mi <= 109
```

- 解题思路

```
func maximizeXor(nums []int, queries [][]int) []int {
        sort.Ints(nums)
        n := len(queries)
        for i := 0; i < n; i++ {
                queries[i] = append(queries[i], i) //␣
↪每个查询添加1个下标方便排序后找到原来的位置
        }
        sort.Slice(queries, func(i, j int) bool {
                return queries[i][1] < queries[j][1] // 按照mi从小到大排序
        })
        root := &Trie{next: make([]*Trie, 2)}
        res := make([]int, n)
        var j int
        for i := 0; i < n; i++ {
                target, mi, index := queries[i][0], queries[i][1], queries[i][2]
                for j < len(nums) && nums[j] <= mi { // 插入小于等于mi的数
```

(续下页)

(接上页)

```
                        root.Insert(nums[j])
                        j++
                }
                if j == 0 {
                        res[index] = -1
                } else {
                        res[index] = root.getMaxValue(target)
                }
        }
        return res
}

type Trie struct {
        next []*Trie // 0或者1
}

// 插入num
func (t *Trie) Insert(num int) {
        temp := t
        for i := 31; i >= 0; i-- {
                value := (num >> i) & 1
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next: make([]*Trie, 2),
                        }
                }
                temp = temp.next[value]
        }
}

// 查找异或对应的最大值
func (t *Trie) getMaxValue(num int) int {
        res := 0
        temp := t
        for i := 31; i >= 0; i-- {
                value := (num >> i) & 1
                if temp.next[1-value] != nil { // 能取到1
                        res = res | (1 << i) // 结果在该位可以为1，该为置为1
                        temp = temp.next[1-value]
                } else {
                        temp = temp.next[value]
                }
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
type Trie struct {
        next     []*Trie // 0或者1
        minValue int     // 小于当前节点的最小值
}

// 插入num
func (t *Trie) Insert(num int) {
        temp := t
        if num < temp.minValue {
                temp.minValue = num
        }
        for i := 31; i >= 0; i-- {
                value := (num >> i) & 1
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:     make([]*Trie, 2),
                                minValue: num,
                        }
                }
                temp = temp.next[value]
                if num < temp.minValue {
                        temp.minValue = num
                }
        }
}

func (t *Trie) getMaxValueWithLimit(num int, limit int) int {
        res := 0
        temp := t
        if temp.minValue > limit {
                return -1
        }
        for i := 31; i >= 0; i-- {
                value := (num >> i) & 1
                if temp.next[1-value] != nil && temp.next[1-value].minValue <= limit
↪{ // 能取到1
                        res = res | (1 << i) // 结果在该位可以为1，该为置为1
                        temp = temp.next[1-value]
                } else {
```

(续下页)

(接上页)

```
                        temp = temp.next[value]
                }
        }
        return res
}
```

54.3 1713. 得到子序列的最少操作次数 (1)

- 题目

```
给你一个数组target，包含若干 互不相同的整数，以及另一个整数数组arr，arr可能
↪包含重复元素。
每一次操作中，你可以在 arr的任意位置插入任一整数。比方说，如果arr = [1,4,1,2]，
那么你可以在中间添加 3得到[1,4,3,1,2]。你可以在数组最开始或最后面添加整数。
请你返回 最少操作次数，使得target成为arr的一个子序列。
一个数组的 子序列指的是删除原数组的某些元素（可能一个元素都不删除），
同时不改变其余元素的相对顺序得到的数组。比方说，[2,7,4]是[4,2,3,7,2,1,
↪4]的子序列（加粗元素），
但[2,4,2]不是子序列。
示例 1：输入：target = [5,1,3], arr = [9,4,2,3,4] 输出：2
解释：你可以添加 5 和 1 ，使得 arr 变为 [5,9,4,1,2,3,4] ，target 为 arr 的子序列。
示例 2：输入：target = [6,4,8,1,3,2], arr = [4,7,6,2,3,8,6,1] 输出：3
提示：1 <= target.length, arr.length <= 105
1 <= target[i], arr[i] <= 109
target不包含任何重复元素。
```

- 解题思路

```
func minOperations(target []int, arr []int) int {
        m := make(map[int]int)
        for i := 0; i < len(target); i++ {
                m[target[i]] = i
        }
        nums := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                if v, ok := m[arr[i]]; ok {
                        nums = append(nums, v)
                }
        }
        res := lengthOfLIS(nums) // leetcode 300.最长上升子序列
        return len(target) - res
}
```

(续下页)

(接上页)

```
func lengthOfLIS(nums []int) int {
        if len(nums) < 2 {
                return len(nums)
        }
        arr := make([]int, len(nums)+1)
        arr[1] = nums[0]
        res := 1
        for i := 1; i < len(nums); i++ {
                if arr[res] < nums[i] {
                        res++
                        arr[res] = nums[i]
                } else {
                        left, right := 1, res
                        index := 0
                        for left <= right {
                                mid := left + (right-left)/2
                                if arr[mid] < nums[i] {
                                        index = mid
                                        left = mid + 1
                                } else {
                                        right = mid - 1
                                }
                        }
                        arr[index+1] = nums[i]
                }
        }
        return res
}
```

54.4 1723. 完成所有工作的最短时间 (4)

- 题目

```
给你一个整数数组 jobs ，其中 jobs[i] 是完成第 i 项工作要花费的时间。
请你将这些工作分配给 k␣
↪位工人。所有工作都应该分配给工人，且每项工作只能分配给一位工人。
工人的 工作时间␣
↪是完成分配给他们的所有工作花费时间的总和。请你设计一套最佳的工作分配方案，使工人的␣
↪最大工作时间 得以 最小化 。
返回分配方案中尽可能 最小 的 最大工作时间 。
示例 1：输入：jobs = [3,2,3]，k = 3 输出：3
```

(续下页)

(接上页)

```
解释：给每位工人分配一项工作，最大工作时间是 3 。
示例 2：输入：jobs = [1,2,4,7,8], k = 2 输出：11
解释：按下述方式分配工作：
1 号工人：1、2、8（工作时间 = 1 + 2 + 8 = 11）
2 号工人：4、7（工作时间 = 4 + 7 = 11）
最大工作时间是 11 。
提示：1 <= k <= jobs.length <= 12
1 <= jobs[i] <= 107
```

- 解题思路

```
func minimumTimeRequired(jobs []int, k int) int {
        sort.Slice(jobs, func(i, j int) bool {
                return jobs[i] > jobs[j]
        })
        sum := 0
        for i := 0; i < len(jobs); i++ {
                sum = sum + jobs[i]
        }
        left, right := jobs[0], sum
        for left < right {
                mid := left + (right-left)/2
                if dfs(jobs, make([]int, k), mid, 0) {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

func dfs(jobs []int, arr []int, limit int, index int) bool {
        if index >= len(jobs) {
                return true
        }
        for i := 0; i < len(arr); i++ {
                if arr[i]+jobs[index] <= limit {
                        arr[i] = arr[i] + jobs[index]
                        if dfs(jobs, arr, limit, index+1) == true {
                                return true
                        }
                        arr[i] = arr[i] - jobs[index]
                }
                // 当前没有被分配 | 当前分配达到上线 => 不需要尝试继续分配
```

(续下页)

(接上页)

```
                // 剪枝:去除也可以过
                if arr[i] == 0 || arr[i]+jobs[index] == limit {
                        break
                }
        }
        return false
}

# 2
func minimumTimeRequired(jobs []int, k int) int {
        sort.Slice(jobs, func(i, j int) bool {
                return jobs[i] > jobs[j]
        })
        sum := 0
        for i := 0; i < len(jobs); i++ {
                sum = sum + jobs[i]
        }
        left, right := jobs[0], sum
        return left + sort.Search(right-left, func(limit int) bool {
                return dfs(jobs, make([]int, k), limit+left, 0)
        })
}

func dfs(jobs []int, arr []int, limit int, index int) bool {
        if index >= len(jobs) {
                return true
        }
        for i := 0; i < len(arr); i++ {
                if arr[i]+jobs[index] <= limit {
                        arr[i] = arr[i] + jobs[index]
                        if dfs(jobs, arr, limit, index+1) == true {
                                return true
                        }
                        arr[i] = arr[i] - jobs[index]
                }
                // 当前没有被分配 | 当前分配达到上线 => 不需要尝试继续分配
                // 剪枝:去除也可以过
                if arr[i] == 0 || arr[i]+jobs[index] == limit {
                        break
                }
        }
        return false
}
```

(续下页)

(接上页)

```
# 3
func minimumTimeRequired(jobs []int, k int) int {
        n := len(jobs)
        total := 1 << n
        sum := make([]int, total)
        for i := 0; i < n; i++ { // 预处理：任务的状态和，分配给某一个人的和
                count := 1 << i
                for j := 0; j < count; j++ {
                        sum[count|j] = sum[j] + jobs[i] // 按位或运算：j前面补1=>
↪子集和加上tasks[i]
                }
        }
        dp := make([][]int, k) // f[i][j]=>
↪给前i个人分配工作，工作的分配情况为j时，完成所有工作的最短时间
        for i := 0; i < k; i++ {
                dp[i] = make([]int, total)
        }
        for i := 0; i < total; i++ { // 第0个人的时候
                dp[0][i] = sum[i]
        }
        for i := 1; i < k; i++ {
                for j := 0; j < total; j++ {
                        minValue := math.MaxInt32            // dp[i][j]未赋值，为0
                        for a := j; a > 0; a = (a - 1) & j { //␣
↪遍历得到比较小的子集：数字j二进制为1位置上的非0子集
                                // 取子集的补集：j-a 或者 使用异或j^a
                                // minValue = min(minValue, max(dp[i-1][j-a], sum[a]))
                                minValue = min(minValue, max(dp[i-1][j^a], sum[a]))
                        }
                        dp[i][j] = minValue
                }
        }
        return dp[k-1][total-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
var res int

func minimumTimeRequired(jobs []int, k int) int {
        res = math.MaxInt32
        dfs(jobs, make([]int, k), 0, 0, 0)
        return res
}

// index => job的下标; count => 已经分配工人数组的个数
func dfs(jobs []int, arr []int, index int, maxValue int, count int) {
        if maxValue > res { // 剪枝
                return
        }
        if index == len(jobs) {
                res = maxValue
                return
        }
        if count < len(arr) { // 分配给空闲的
                arr[count] = jobs[index]
                dfs(jobs, arr, index+1, max(maxValue, arr[count]), count+1)
                arr[count] = 0
        }
        for i := 0; i < count; i++ { // 分配给非空闲的
                arr[i] = arr[i] + jobs[index]
                dfs(jobs, arr, index+1, max(maxValue, arr[i]), count)
                arr[i] = arr[i] - jobs[index]
        }
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

54.5 1739. 放置盒子 (2)

- 题目

```
有一个立方体房间，其长度、宽度和高度都等于 n 个单位。请你在房间里放置 n
↪个盒子，每个盒子都是一个单位边长的立方体。放置规则如下：
你可以把盒子放在地板上的任何地方。
如果盒子 x 需要放置在盒子 y 的顶部，那么盒子 y 竖直的四个侧面都 必须
↪与另一个盒子或墙相邻。
给你一个整数 n ，返回接触地面的盒子的 最少 可能数量。
示例 1：输入：n = 3 输出：3
解释：上图是 3 个盒子的摆放位置。
这些盒子放在房间的一角，对应左侧位置。
示例 2：输入：n = 4 输出：3
解释：上图是 3 个盒子的摆放位置。
这些盒子放在房间的一角，对应左侧位置。
示例 3：输入：n = 10 输出：6
解释：上图是 10 个盒子的摆放位置。
这些盒子放在房间的一角，对应后方位置。
提示：1 <= n <= 109
```

- 解题思路

```
func minimumBoxes(n int) int {
        res := 0
        k := 1 // 第几层
        total := 0
        for {
                count := k * (k + 1) / 2 // 第几层数量
                if total+count > n {     // 大于n就不加了
                        break
                }
                total = total + count
                k++
        }
        k--
        res = k * (k + 1) / 2 // 最底层
        k = 1
        for total < n { // 底层再从一条边上往墙角叠加
                total = total + k
                k++
                res++
        }
        return res
```

(续下页)

(接上页)

```
}

# 2
func minimumBoxes(n int) int {
        res := 0
        left, right := 1, 3000
        for left < right {
                mid := left + (right-left)/2
                if mid*(mid+1)*(mid+2)/6 < n {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        left = left - 1
        res = (1 + left) * left / 2
        n = n - left*(left+1)*(left+2)/6
        left, right = 0, n
        for left < right {
                mid := left + (right-left)/2
                target := mid * (mid + 1) / 2
                if target < n {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return res + left
}
```

54.6 1745. 回文串分割 IV(2)

- 题目

```
给你一个字符串s，如果可以将它分割成三个非空回文子字符串，那么返回true，否则返回false。
当一个字符串正着读和反着读是一模一样的，就称其为 回文字符串 。
示例 1：输入：s = "abcbdd" 输出：true
解释："abcbdd" = "a" + "bcb" + "dd"，三个子字符串都是回文的。
示例 2：输入：s = "bcbddxy" 输出：false
解释：s 没办法被分割成 3 个回文子字符串。
提示：3 <= s.length <= 2000
s只包含小写英文字母。
```

- 解题思路

```
func checkPartitioning(s string) bool {
        for i := 1; i < len(s)-1; i++ {
                a := s[0:i]
                if check(a) == false {
                        continue
                }
                for j := len(s) - 1; j > i; j-- {
                        c := s[j:]
                        if check(c) == false {
                                continue
                        }
                        b := s[i:j]
                        if check(b) == true {
                                return true
                        }
                }
        }
        return false
}

func check(s string) bool {
        left, right := 0, len(s)-1
        for left < right {
                if s[left] != s[right] {
                        return false
                }
                left++
                right--
        }
        return true
}

# 2
func checkPartitioning(s string) bool {
        n := len(s)
        dp := make([][]bool, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]bool, n)
        }
        for i := n - 1; i >= 0; i-- {
                for j := i; j < n; j++ {
                        if i == j {
                                dp[i][j] = true
```

(续下页)

(接上页)

```
                        } else if i+1 == j && s[i] == s[j] {
                                dp[i][j] = true
                        } else {
                                if s[i] == s[j] && dp[i+1][j-1] == true {
                                        dp[i][j] = true
                                }
                        }
                }
        }
        for i := 0; i < n-2; i++ {
                if dp[0][i] == false {
                        continue
                }
                for j := i + 1; j < n-1; j++ {
                        if dp[i+1][j] && dp[j+1][n-1] {
                                return true
                        }
                }
        }
        return false
}
```

54.7 1751. 最多可以参加的会议数目 II(2)

- 题目

```
给你一个events数组，其中events[i] = [startDayi, endDayi,␣
↪valuei]，表示第i个会议在startDayi天开始，
第endDayi天结束，如果你参加这个会议，你能得到价值valuei。同时给你一个整数k表示你能参加的最多会议数目
你同一时间只能参加一个会议。如果你选择参加某个会议，那么你必须 完整地参加完这个会议。
会议结束日期是包含在会议内的，也就是说你不能同时参加一个开始日期与另一个结束日期相同的两个会议。
请你返回能得到的会议价值最大和。
示例 1： 输入：events = [[1,2,4],[3,4,3],[2,3,1]], k = 2 输出：7
解释：选择绿色的活动会议 0 和 1，得到总价值和为 4 + 3 = 7 。
示例 2：输入：events = [[1,2,4],[3,4,3],[2,3,10]], k = 2 输出：10
解释：参加会议 2 ，得到价值和为 10 。
你没法再参加别的会议了，因为跟会议 2 有重叠。你 不需要参加满 k 个会议。
示例 3：输入：events = [[1,1,1],[2,2,2],[3,3,3],[4,4,4]], k = 3 输出：9
解释：尽管会议互不重叠，你只能参加 3 个会议，所以选择价值最大的 3 个会议。
提示：1 <= k <= events.length
1 <= k * events.length <= 106
1 <= startDayi <= endDayi <= 109
```

(续下页)

(接上页)

```
1 <= valuei <= 106
```

- 解题思路

```
func maxValue(events [][]int, k int) int {
        n := len(events)
        sort.Slice(events, func(i, j int) bool {
                if events[i][1] == events[j][1] {
                        return events[i][2] < events[j][2]
                }
                return events[i][1] < events[j][1]
        })
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= k; j++ {
                        dp[i][j] = dp[i-1][j]
                        left, right := 0, i-1
                        for left < right {
                                mid := left + (right-left)/2
                                if events[mid][1] < events[i-1][0] {
                                        left = mid + 1
                                } else {
                                        right = mid
                                }
                        }
                        dp[i][j] = max(dp[i][j], dp[right][j-1]+events[i-1][2])
                }
        }
        res := 0
        for i := 1; i <= k; i++ {
                res = max(res, dp[n][i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
func maxValue(events [][]int, k int) int {
        n := len(events)
        sort.Slice(events, func(i, j int) bool {
                if events[i][1] == events[j][1] {
                        return events[i][2] < events[j][2]
                }
                return events[i][1] < events[j][1]
        })
        dp := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
        }
        for i := 1; i <= n; i++ {
                index := 0
                for j := i - 1; j >= 1; j-- {
                        if events[j-1][1] < events[i-1][0] {
                                index = j
                                break
                        }
                }
                for j := 1; j <= k; j++ {
                        dp[i][j] = max(dp[i-1][j], dp[index][j-1]+events[i-1][2])
                }
        }
        return dp[n][k]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

54.8 1755. 最接近目标值的子序列和

54.8.1 题目

```
给你一个整数数组 nums 和一个目标值 goal 。
你需要从 nums 中选出一个子序列，使子序列元素总和最接近 goal 。
也就是说，如果子序列元素和为 sum ，你需要 最小化绝对差 abs(sum - goal) 。
返回 abs(sum - goal) 可能的 最小值 。
注意，数组的子序列是通过移除原始数组中的某些元素（可能全部或无）而形成的数组。
示例 1: 输入: nums = [5,-7,3,5], goal = 6 输出: 0
解释: 选择整个数组作为选出的子序列，元素和为 6 。
子序列和与目标值相等，所以绝对差为 0 。
示例 2: 输入: nums = [7,-9,15,-2], goal = -5 输出: 1
解释: 选出子序列 [7,-9,-2] ，元素和为 -4 。
绝对差为 abs(-4 - (-5)) = abs(1) = 1 ，是可能的最小值。
示例 3: 输入: nums = [1,2,3], goal = -7 输出: 7
提示: 1 <= nums.length <= 40
-107 <= nums[i] <= 107
-109 <= goal <= 109
```

54.8.2 解题思路

```

```

54.9 1761. 一个图中连通三元组的最小度数 (1)

- 题目

```
给你一个无向图，整数 n表示图中节点的数目，edges数组表示图中的边，
其中edges[i] = [ui, vi]，表示ui 和vi之间有一条无向边。
一个 连通三元组指的是 三个节点组成的集合且这三个点之间 两两有边。
连通三元组的度数是所有满足此条件的边的数目: 一个顶点在三元组内，而另一个顶点不在三元组内。
请你返回所有连通三元组中度数的 最小值，如果图中没有连通三元组，那么返回 -1。
示例 1: 输入: n = 6, edges = [[1,2],[1,3],[3,2],[4,1],[5,2],[3,6]] 输出: 3
解释: 只有一个三元组 [1,2,3] 。构成度数的边在上图中已被加粗。
示例 2: 输入: n = 7, edges = [[1,3],[4,1],[4,3],[2,5],[5,6],[6,7],[7,5],[2,6]] 输出: 0
解释: 有 3 个三元组:
1) [1,4,3]，度数为 0 。
2) [2,5,6]，度数为 2 。
3) [5,6,7]，度数为 2 。
```

(续下页)

(接上页)

```
提示：2 <= n <= 400
edges[i].length == 2
1 <= edges.length <= n * (n-1) / 2
1 <= ui, vi <= n
ui != vi
图中没有重复的边。
```

- 解题思路

```
func minTrioDegree(n int, edges [][]int) int {
        arr := make([]int, 401)
        m := make(map[int]bool)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a]++
                arr[b]++
                m[getValue(a, b)] = true
        }
        res := math.MaxInt32
        for i := 1; i < n-1; i++ {
                for j := i + 1; j < n; j++ {
                        if m[getValue(i, j)] == true {
                                for k := j + 1; k <= n; k++ {
                                        if m[getValue(i, k)] == true && m[getValue(j,␣
↪k)] == true {
                                                if arr[i]+arr[j]+arr[k]-6 < res {
                                                        res = arr[i] + arr[j] +␣
↪arr[k] - 6
                                                }
                                        }
                                }
                        }
                }
        }
        if res == math.MaxInt32 {
                return -1
        }
        return res
}

func getValue(a, b int) int {
        if a > b {
                a, b = b, a
        }
```

(续下页)

(接上页)

```
        return a*1000 + b
}
```

54.10 1766. 互质树 (1)

- 题目

```
给你一个 n个节点的树（也就是一个无环连通无向图），节点编号从 0到 n - 1，且恰好有 n -␣
↪1条边，每个节点有一个值。
树的 根节点为 0 号点。
给你一个整数数组nums和一个二维数组edges来表示这棵树。nums[i]表示第i个点的值，
edges[j] = [uj, vj]表示节点uj和节点vj在树中有一条边。
当gcd(x, y) == 1，我们称两个数x 和y是 互质的，其中gcd(x, y)是 x和 y的 最大公约数。
从节点i到 根最短路径上的点都是节点 i的祖先节点。一个节点 不是 它自己的祖先节点。
请你返回一个大小为 n的数组 ans，其中ans[i]是离节点i最近的祖先节点且满足nums[i]␣
↪和nums[ans[i]]是 互质的，
如果不存在这样的祖先节点，ans[i]为 -1。
示例 1：输入：nums = [2,3,3,2], edges = [[0,1],[1,2],[1,3]] 输出：[-1,0,0,1]
解释：上图中，每个节点的值在括号中表示。
- 节点 0 没有互质祖先。
- 节点 1 只有一个祖先节点 0 。它们的值是互质的（gcd(2,3) == 1）。
- 节点 2 有两个祖先节点，分别是节点 1 和节点 0 。节点 1 的值与它的值不是互质的（gcd(3,
↪3) == 3）但节点 0 的值是互质的(gcd(2,3) == 1)，所以节点 0␣
↪是最近的符合要求的祖先节点。
- 节点 3 有两个祖先节点，分别是节点 1 和节点 0 。
它与节点 1 互质（gcd(3,2) == 1），所以节点 1 是离它最近的符合要求的祖先节点。
示例 2：输入：nums = [5,6,10,2,3,6,15], edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]]␣
↪输出：[-1,0,-1,0,0,0,-1]
提示：nums.length == n
1 <= nums[i] <= 50
1 <= n <= 105
edges.length == n - 1
edges[j].length == 2
0 <= uj, vj < n
uj != vj
```

- 解题思路

```
var res []int              // 结果
var arr [][]int            // 邻接表
var path map[int][][2]int //␣
↪路径：以每个值做区分来记录path，每个值取最新的path来比较即可
```

(续下页)

(接上页)

```
func getCoprimes(nums []int, edges [][]int) []int {
        n := len(nums)
        res = make([]int, n)
        arr = make([][]int, n)
        path = make(map[int][][2]int)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        dfs(nums, 0, -1, 0)
        return res
}

func dfs(nums []int, cur int, prev int, level int) {
        index, depth := -1, -1
        for i := 1; i <= 50; i++ {
                if len(path[i]) > 0 {
                        a, b := path[i][len(path[i])-1][0], path[i][len(path[i])-
↪1][1] // 遍历每个值最近
                        if a > depth && gcd(i, nums[cur]) == 1 {
↪ // 更新：更近+互质
                                depth = a
                                index = b
                        }
                }
        }
        res[cur] = index
        for i := 0; i < len(arr[cur]); i++ {
                value := nums[cur]
                next := arr[cur][i]
                if next != prev {
                        path[value] = append(path[value], [2]int{level, cur})
                        dfs(nums, next, cur, level+1)
                        path[value] = path[value][:len(path[value])-1]
                }
        }
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
```

(续下页)

(接上页)

```
        }
        return b
}
```

54.11 1771. 由子序列构造的最长回文串的长度 (3)

- 题目

```
给你两个字符串 word1 和 word2 ，请你按下述方法构造一个字符串：
从 word1 中选出某个 非空 子序列 subsequence1 。
从 word2 中选出某个 非空 子序列 subsequence2 。
连接两个子序列 subsequence1 + subsequence2 ，得到字符串。
返回可按上述方法构造的最长 回文串 的 长度 。如果无法构造回文串，返回 0 。
字符串 s 的一个 子序列 是通过从 s␣
↪中删除一些（也可能不删除）字符而不更改其余字符的顺序生成的字符串。
回文串 是正着读和反着读结果一致的字符串。
示例 1：输入：word1 = "cacb", word2 = "cbba" 输出：5
解释：从 word1 中选出 "ab" ，从 word2 中选出 "cba" ，得到回文串 "abcba" 。
示例 2：输入：word1 = "ab", word2 = "ab" 输出：3
解释：从 word1 中选出 "ab" ，从 word2 中选出 "a" ，得到回文串 "aba" 。
示例 3：输入：word1 = "aa", word2 = "bb" 输出：0
解释：无法按题面所述方法构造回文串，所以返回 0 。
提示：1 <= word1.length, word2.length <= 1000
word1 和 word2 由小写英文字母组成
```

- 解题思路

```
func longestPalindrome(word1 string, word2 string) int {
        s := word1 + word2
        a := len(word1)
        n := len(s)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
                dp[i][i] = 1
        }
        res := 0
        for i := n - 2; i >= 0; i-- {
                for j := i + 1; j < n; j++ {
                        if s[i] == s[j] {
                                dp[i][j] = dp[i+1][j-1] + 2 // 内层+2
                                if i < a && j >= a {
```

(续下页)

(接上页)

```
                                        res = max(res, dp[i][j])
                                }
                        } else {
                                dp[i][j] = max(dp[i+1][j], dp[i][j-1])
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestPalindrome(word1 string, word2 string) int {
        s := word1 + word2
        a := len(word1)
        n := len(s)
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = 1
        }
        res := 0
        for i := n - 1; i >= 0; i-- {
                prev := 0
                for j := i + 1; j < n; j++ {
                        temp := dp[j]
                        if s[i] == s[j] {
                                dp[j] = prev + 2 // 内层+2
                                if i < a && j >= a {
                                        res = max(res, dp[j])
                                }
                        } else {
                                dp[j] = max(dp[j], dp[j-1])
                        }
                        prev = temp
                }
        }
        return res
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
var dp [][]int
var a int
var res int

func longestPalindrome(word1 string, word2 string) int {
        s := word1 + word2
        a = len(word1)
        res = 0
        n := len(s)
        dp = make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        dfs(s, 0, n-1)
        return res
}

func dfs(s string, i, j int) int {
        if i == j {
                return 1
        }
        if i > j {
                return 0
        }
        if dp[i][j] > 0 {
                return dp[i][j]
        }
        if s[i] == s[j] {
                dp[i][j] = dfs(s, i+1, j-1) + 2
                if i < a && j >= a {
                        res = max(res, dp[i][j])
                }
        } else {
```

(续下页)

(接上页)

```
                dp[i][j] = max(dfs(s, i+1, j), dfs(s, i, j-1))
        }
        return dp[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

54.12 1776. 车队 II(2)

- 题目

```
在一条单车道上有 n辆车，它们朝着同样的方向行驶。
给你一个长度为 n的数组 cars，其中cars[i] = [positioni, speedi]，它表示：
positioni是第 i辆车和道路起点之间的距离（单位：米）。题目保证positioni < positioni+1。
speedi是第 i辆车的初始速度（单位：米/秒）。
简单起见，所有车子可以视为在数轴上移动的点。当两辆车占据同一个位置时，我们称它们相遇了。
一旦两辆车相遇，它们会合并成一个车队，这个车队里的车有着同样的位置和相同的速度，速度为这个车队里最慢
请你返回一个数组answer，其中answer[i]是第 i辆车与下一辆车相遇的时间（单位：秒），
如果这辆车不会与下一辆车相遇，则 answer[i]为 -1。答案精度误差需在 10-5以内。
示例 1：输入：cars = [[1,2],[2,1],[4,3],[7,2]] 输出：[1.00000,-1.00000,3.00000,-1.
↪00000]
解释：经过恰好 1 秒以后，第一辆车会与第二辆车相遇，并形成一个 1 m/s 的车队。
经过恰好 3 秒以后，第三辆车会与第四辆车相遇，并形成一个 2 m/s 的车队。
示例 2：输入：cars = [[3,4],[5,4],[6,3],[9,1]] 输出：[2.00000,1.00000,1.50000,-1.
↪00000]
提示：1 <= cars.length <= 105
1 <= positioni, speedi <= 106
positioni < positioni+1
```

- 解题思路

```
func getCollisionTimes(cars [][]int) []float64 {
        n := len(cars)
        res := make([]float64, n)
        stack := make([]int, 0) // 单调栈
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 {
```

(续下页)

(接上页)

```
                        top := stack[len(stack)-1]
                        a := float64(cars[top][0] - cars[i][0]) // 距离差
                        b := float64(cars[i][1] - cars[top][1]) // 速度差
                        if (cars[i][1] <= cars[top][1]) ||      // 1、永远追不上栈顶车
                                (res[top] > 0 && a/b > res[top]) { //␣
↪2、可以在栈顶车追上更前面车之前追上栈顶车
                                stack = stack[:len(stack)-1]
                        } else {
                                break
                        }
                }
                if len(stack) == 0 {
                        res[i] = -1
                } else { // 可以追上
                        top := stack[len(stack)-1]
                        a := float64(cars[top][0] - cars[i][0]) // 距离差
                        b := float64(cars[i][1] - cars[top][1]) // 速度差
                        res[i] = a / b
                }
                stack = append(stack, i)
        }
        return res
}

# 2
func getCollisionTimes(cars [][]int) []float64 {
        n := len(cars)
        res := make([]float64, n)
        stack := make([]int, 0) // 单调栈
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 {
                        top := stack[len(stack)-1]
                        if cars[i][1] <= cars[top][1] { // 1、永远追不上栈顶车
                                stack = stack[:len(stack)-1]
                        } else {
                                a := float64(cars[top][0] - cars[i][0]) // 距离差
                                b := float64(cars[i][1] - cars[top][1]) // 速度差
                                if res[top] < 0 {
                                        break
                                } else if res[top] > 0 && a/b > res[top] { //␣
↪2、可以在栈顶车追上更前面车之前追上栈顶车
                                        stack = stack[:len(stack)-1]
                                } else {
```

(续下页)

(接上页)

```
                                        break
                                }
                        }
                }
                if len(stack) == 0 {
                        res[i] = -1
                } else { // 可以追上
                        top := stack[len(stack)-1]
                        a := float64(cars[top][0] - cars[i][0]) // 距离差
                        b := float64(cars[i][1] - cars[top][1]) // 速度差
                        res[i] = a / b
                }
                stack = append(stack, i)
        }
        return res
}
```

54.13 1782. 统计点对的数目 (1)

- 题目

```
给你一个无向图，无向图由整数n，表示图中节点的数目，和edges组成，
其中edges[i] = [ui, vi]表示ui
↪和vi之间有一条无向边。同时给你一个代表查询的整数数组queries。
第 j 个查询的答案是满足如下条件的点对 (a, b) 的数目：
a < b
cnt是与 a或者b相连的边的数目，且 cnt严格大于queries[j]。
请你返回一个数组answers，其中answers.length == queries.length 且answers[j]是第
↪j个查询的答案。
请注意，图中可能会有 重复边。
示例 1：输入：n = 4, edges = [[1,2],[2,4],[1,3],[2,3],[2,1]], queries = [2,3]
↪输出：[6,5]
解释：每个点对中，与至少一个点相连的边的数目如上图所示。
示例 2：输入：n = 5, edges = [[1,5],[1,5],[3,4],[2,5],[1,3],[5,1],[2,3],[2,5]],
↪queries = [1,2,3,4,5]
输出：[10,10,9,8,6]
提示：2 <= n <= 2 * 104
1 <= edges.length <= 105
1 <= ui, vi <= n
ui != vi
1 <= queries.length <= 20
0 <= queries[j] < edges.length
```

- 解题思路

```
func countPairs(n int, edges [][]int, queries []int) []int {
        degree := make([]int, n+1) // 点=>相连边的次数
        m := make(map[[2]int]int)  // 边的出现次数
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                if a > b { // 调整为a<b
                        a, b = b, a
                }
                m[[2]int{a, b}]++
                degree[a]++
                degree[b]++
        }
        arr := make([]int, n+1) // 使用临时数组来排序
        copy(arr, degree)
        sort.Ints(arr)
        res := make([]int, len(queries))
        for i := 0; i < len(queries); i++ {
                target := queries[i]
                left, right := 1, n // 双指针=>2数之和
                for left < right {
                        if target < arr[left]+arr[right] {
                                res[i] = res[i] + right - left // 计算对数
                                right--
                        } else {
                                left++
                        }
                }
                for k, v := range m { //␣
↪遍历边：减去2点之间的重复边的数量后不满足要求，数目-1
                        a, b := k[0], k[1]
                        if target < degree[a]+degree[b] && target >=␣
↪degree[a]+degree[b]-v {
                                res[i]--
                        }
                }
        }
        return res
}
```

54.14 1787. 使所有区间的异或结果为零

54.14.1 题目

```
给你一个整数数组 nums和一个整数 k 。区间 [left, right] (left <= right) 的 异或结果
是对下标位于left 和 right (包括 left 和 right ) 之间所有元素进行 XOR 运算的结果:
nums[left] XOR nums[left+1] XOR ... XOR nums[right] 。
返回数组中 要更改的最小元素数 , 以使所有长度为 k 的区间异或结果等于零。
示例 1: 输入: nums = [1,2,0,3,0], k = 1 输出: 3
解释: 将数组 [1,2,0,3,0] 修改为 [0,0,0,0,0]
示例 2: 输入: nums = [3,4,5,2,1,7,3,4,7], k = 3 输出: 3
解释: 将数组 [3,4,5,2,1,7,3,4,7] 修改为 [3,4,7,3,4,7,3,4,7]
示例 3: 输入: nums = [1,2,4,1,2,5,1,2,6], k = 3 输出: 3
解释: 将数组[1,2,4,1,2,5,1,2,6] 修改为 [1,2,3,1,2,3,1,2,3]
提示: 1 <= k <= nums.length <= 2000
0 <= nums[i] < 210
```

54.14.2 解题思路

54.15 1793. 好子数组的最大分数 (2)

- 题目

```
给你一个整数数组nums (下标从 0 开始) 和一个整数k。
一个子数组 (i, j)的 分数定义为min(nums[i], nums[i+1], ..., nums[j]) * (j - i + 1)。
一个好子数组的两个端点下标需要满足i <= k <= j。
请你返回 好子数组的最大可能 分数。
示例 1: 输入: nums = [1,4,3,7,4,5], k = 3 输出: 15
解释: 最优子数组的左右端点下标是 (1, 5) , 分数为 min(4,3,7,4,5) * (5-1+1) = 3 * 5 =␣
↪15 。
示例 2: 输入: nums = [5,5,4,5,4,1,1,1], k = 0 输出: 20
解释: 最优子数组的左右端点下标是 (0, 4) , 分数为 min(5,5,4,5,4) * (4-0+1) = 4 * 5 =␣
↪20 。
提示: 1 <= nums.length <= 105
1 <= nums[i] <= 2 * 104
0 <= k < nums.length
```

- 解题思路

```
func maximumScore(nums []int, k int) int {
        left, right := k, k
        res := 0
        for minValue := nums[k]; minValue >= 1; minValue-- {
                for left >= 0 && nums[left] >= minValue {
                        left--
                }
                for right < len(nums) && nums[right] >= minValue {
                        right++
                }
                left++  // left注意+1
                right-- // right注意-1
                res = max(res, minValue*(right-left+1))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maximumScore(nums []int, k int) int {
        res := 0
        n := len(nums)
        left := make([]int, n)
        right := make([]int, n)
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                for len(stack) > 0 && nums[stack[len(stack)-1]] >= nums[i] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        left[i] = -1
                } else {
                        left[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }
        stack = make([]int, 0)
        for i := n - 1; i >= 0; i-- {
```

(续下页)

(接上页)

```
                for len(stack) > 0 && nums[stack[len(stack)-1]] >= nums[i] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        right[i] = n
                } else {
                        right[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }
        for i := 0; i < n; i++ {
                if left[i] < k && right[i] > k {
                        res = max(res, nums[i]*(right[i]-left[i]-1))
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

54.16 1799.N 次操作后的最大分数和 (3)

- 题目

```
给你nums，它是一个大小为2 * n的正整数数组。你必须对这个数组执行 n次操作。
在第i次操作时（操作编号从 1开始），你需要：
选择两个元素x 和y。
获得分数i * gcd(x, y)。
将x和y 从nums中删除。
请你返回 n次操作后你能获得的分数和最大为多少。
函数gcd(x, y)是x 和y的最大公约数。
示例 1：输入：nums = [1,2] 输出：1
解释：最优操作是：(1 * gcd(1, 2)) = 1
示例 2：输入：nums = [3,4,6,8] 输出：11
解释：最优操作是：(1 * gcd(3, 6)) + (2 * gcd(4, 8)) = 3 + 8 = 11
示例 3：输入：nums = [1,2,3,4,5,6] 输出：14
解释：最优操作是：(1 * gcd(1, 5)) + (2 * gcd(2, 4)) + (3 * gcd(3, 6)) = 1 + 4 + 9 = 14
```

(续下页)

(接上页)

```
提示：1 <= n <= 7
nums.length == 2 * n
1 <= nums[i] <= 106
```

- 解题思路

```
func maxScore(nums []int) int {
        n := len(nums)
        total := 1 << n
        dp := make([]int, total)
        arr := [14][14]int{}
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        value := gcd(nums[i], nums[j])
                        arr[i][j] = value
                        arr[j][i] = value
                }
        }
        for i := 0; i < total; i++ { // 枚举状态：以当前状态推出后面的状态
                count := bits.OnesCount(uint(i)) // i对应二进制1个数
                if count%2 == 1 {                // 可去除；剪枝：1的个数为奇数个
                        continue
                }
                count = count / 2 // 第count次操作
                for j := 0; j < n; j++ {
                        for k := j + 1; k < n; k++ {
                                if i&(1<<j) == 0 && i&(1<<k) == 0 { //␣
↪i的第j位和第k位都为0
                                        next := i | (1 << j) | (1 << k) //␣
↪按位或运算：把i的第j位和第k位补1
                                        value := dp[i] + (count+1)*arr[j][k]
                                        if value > dp[next] {
                                                dp[next] = value
                                        }
                                }
                        }
                }
        }
        return dp[total-1]
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
```

(续下页)

(接上页)

```
        }
        return b
}

# 2
func maxScore(nums []int) int {
        n := len(nums)
        total := 1 << n
        dp := make([]int, total)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        status := (1 << i) | (1 << j) // 把第i位和第j位置1
                        dp[status] = gcd(nums[i], nums[j])
                }
        }
        for i := 0; i < total; i++ { // 枚举状态：当前状态依赖之前的
                a := bits.OnesCount(uint(i))
                if a%2 == 1 { // 可去除；剪枝：1的个数为奇数个
                        continue
                }
                for j := i; j > 0; j = (j - 1) & i { // 遍历i二进制中位置1的子集
                        b := bits.OnesCount(uint(j))
                        if a-b == 2 { // 1的个数相差2
                                dp[i] = max(dp[i], dp[j]+a/2*dp[i^j]) // 取补集-
↪异或操作：i^j
                        }
                }
        }
        return dp[total-1]
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

# 3
var dp []int
var temp []int

func maxScore(nums []int) int {
        n := len(nums)
        total := 1 << n
        dp = make([]int, total)
        temp = make([]int, total)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        status := (1 << i) | (1 << j) // 把第i位和第j位置1
                        temp[status] = gcd(nums[i], nums[j])
                }
        }
        return dfs(nums, total-1, 0) // 从目标往前推
}

func dfs(nums []int, status int, count int) int {
        if dp[status] > 0 {
                return dp[status]
        }
        n := len(nums)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if (status>>i)&1 == 0 || (status>>j)&1 == 0 { //␣
↪status的第i或者第j位为0
                                continue
                        }
                        cur := (1 << i) | (1 << j)
                        prev := status ^ (1 << i) ^ (1 << j) //␣
↪异或操作：把status都为1的第i,j位变为0
                        dp[status] = max(dp[status], dfs(nums, prev,␣
↪count+1)+(count+1)*temp[cur])
                }
        }
        return dp[status]
}

func gcd(a, b int) int {
        for a != 0 {
```

(续下页)

(接上页)

```
                a, b = b%a, a
        }
        return b
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

CHAPTER 55

1801-1900-Easy

55.1 1805. 字符串中不同整数的数目 (2)

- 题目

```
给你一个字符串 word ，该字符串由数字和小写英文字母组成。
请你用空格替换每个不是数字的字符。例如，"a123bc34d8ef34" 将会变成 " 123 34 8 34" 。
注意，剩下的这些整数为（相邻彼此至少有一个空格隔开）："123"、"34"、"8" 和 "34" 。
返回对 word 完成替换后形成的 不同 整数的数目。
只有当两个整数的 不含前导零 的十进制表示不同， 才认为这两个整数也不同。
示例 1：输入：word = "a123bc34d8ef34" 输出：3
解释：不同的整数有 "123"、"34" 和 "8" 。注意，"34" 只计数一次。
示例 2：输入：word = "leet1234code234" 输出：2
示例 3：输入：word = "a1b01c001" 输出：1
解释："1"、"01" 和 "001"␣
↪视为同一个整数的十进制表示，因为在比较十进制值时会忽略前导零的存在。
提示：1 <= word.length <= 1000
word 由数字和小写英文字母组成
```

- 解题思路

```
func numDifferentIntegers(word string) int {
        m := make(map[string]bool)
        arr := strings.FieldsFunc(word, func(r rune) bool {
                return 'a' <= r && r <= 'z'
```

(续下页)

(接上页)

```
        })
        for i := 0; i < len(arr); i++ {
                s := strings.Trim(arr[i], " ")
                for len(s) > 0 && s[0] == '0' {
                        s = s[1:]
                }
                m[s] = true
        }
        return len(m)
}

# 2
func numDifferentIntegers(word string) int {
        m := make(map[int]bool)
        for i := 0; i < len(word); i++ {
                if '0' <= word[i] && word[i] <= '9' {
                        value := int(word[i] - '0')
                        for i+1 < len(word) && '0' <= word[i+1] && word[i+1] <= '9' {
                                i++
                                value = value*10 + int(word[i]-'0')
                        }
                        m[value] = true
                }
        }
        return len(m)
}
```

55.2 1812. 判断国际象棋棋盘中一个格子的颜色 (2)

- 题目

```
给你一个坐标coordinates，它是一个字符串，表示国际象棋棋盘中一个格子的坐标。下图是国际象棋棋盘示意图。
如果所给格子的颜色是白色，请你返回true，如果是黑色，请返回false。
给定坐标一定代表国际象棋棋盘上一个存在的格子。坐标第一个字符是字母，第二个字符是数字。
示例 1：输入：coordinates = "a1" 输出：false
解释：如上图棋盘所示，"a1" 坐标的格子是黑色的，所以返回 false 。
示例 2：输入：coordinates = "h3" 输出：true
解释：如上图棋盘所示，"h3" 坐标的格子是白色的，所以返回 true 。
示例 3：输入：coordinates = "c7" 输出：false
提示：coordinates.length == 2
'a' <= coordinates[0] <= 'h'
'1' <= coordinates[1] <= '8'
```

- 解题思路

```
func squareIsWhite(coordinates string) bool {
        a := int(coordinates[0] - 'a')
        b := int(coordinates[1] - '1')
        return (a+b)%2 != 0
}

# 2
func squareIsWhite(coordinates string) bool {
        // a => 97  1 => 49
        return (coordinates[0]+coordinates[1])%2 != 0
}
```

55.3 1816. 截断句子 (2)

- 题目

```
句子 是一个单词列表，列表中的单词之间用单个空格隔开，且不存在前导或尾随空格。
每个单词仅由大小写英文字母组成（不含标点符号）。
例如，"Hello World"、"HELLO" 和 "hello world hello world" 都是句子。
给你一个句子 s和一个整数 k，请你将 s截断 ，使截断后的句子仅含 前 k个单词。
返回 截断 s后得到的句子。
示例 1：输入：s = "Hello how are you Contestant", k = 4 输出："Hello how are you"
解释：s 中的单词为 ["Hello", "how" "are", "you", "Contestant"]
前 4 个单词为 ["Hello", "how", "are", "you"]
因此，应当返回 "Hello how are you"
示例 2：输入：s = "What is the solution to this problem", k = 4
输出："What is the solution"
解释：s 中的单词为 ["What", "is" "the", "solution", "to", "this", "problem"]
前 4 个单词为 ["What", "is", "the", "solution"]
因此，应当返回 "What is the solution"
示例 3：输入：s = "chopper is not a tanuki", k = 5 输出："chopper is not a tanuki"
提示：1 <= s.length <= 500
k 的取值范围是 [1, s 中单词的数目]
s 仅由大小写英文字母和空格组成
s 中的单词之间由单个空格隔开
不存在前导或尾随空格
```

- 解题思路

```
func truncateSentence(s string, k int) string {
        arr := strings.Split(s, " ")
```

(续下页)

(接上页)

```
        if k < len(arr) {
                return strings.Join(arr[:k], " ")
        }
        return s
}

# 2
func truncateSentence(s string, k int) string {
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == ' ' {
                        count++
                        if count == k {
                                return s[:i]
                        }
                }
        }
        return s
}
```

55.4 1822. 数组元素积的符号 (1)

- 题目

```
已知函数signFunc(x) 将会根据 x 的正负返回特定值:
如果 x 是正数，返回 1 。
如果 x 是负数，返回 -1 。
如果 x 是等于 0 ，返回 0 。
给你一个整数数组 nums 。令 product 为数组 nums 中所有元素值的乘积。
返回 signFunc(product) 。
示例 1: 输入: nums = [-1,-2,-3,-4,3,2,1] 输出: 1
解释: 数组中所有值的乘积是 144 ，且 signFunc(144) = 1
示例 2: 输入: nums = [1,5,0,2,-3] 输出: 0
解释: 数组中所有值的乘积是 0 ，且 signFunc(0) = 0
示例 3: 输入: nums = [-1,1,-1,1,-1] 输出: -1
解释: 数组中所有值的乘积是 -1 ，且 signFunc(-1) = -1
提示: 1 <= nums.length <= 1000
-100 <= nums[i] <= 100
```

- 解题思路

```
func arraySign(nums []int) int {
        res := 1
        for i := 0; i < len(nums); i++ {
                if nums[i] < 0 {
                        res = -res
                } else if nums[i] == 0 {
                        return 0
                }
        }
        return res
}
```

55.5 1827. 最少操作使数组递增 (2)

- 题目

```
给你一个整数数组nums（下标从 0␣
↪开始）。每一次操作中，你可以选择数组中一个元素，并将它增加1。
比方说，如果nums = [1,2,3]，你可以选择增加nums[1]得到nums = [1,3,3]。
请你返回使 nums严格递增的 最少操作次数。
我们称数组nums是 严格递增的，当它满足对于所有的0 <= i < nums.length - 1都有
nums[i] < nums[i+1]。一个长度为 1的数组是严格递增的一种特殊情况。
示例 1：输入：nums = [1,1,1] 输出：3
解释：你可以进行如下操作：
1) 增加 nums[2] ，数组变为 [1,1,2] 。
2) 增加 nums[1] ，数组变为 [1,2,2] 。
3) 增加 nums[2] ，数组变为 [1,2,3] 。
示例 2：输入：nums = [1,5,2,4,1] 输出：14
示例 3：输入：nums = [8] 输出：0
提示：1 <= nums.length <= 5000
1 <= nums[i] <= 104
```

- 解题思路

```
func minOperations(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        res := 0
        for i := 1; i < n; i++ {
                if nums[i-1] >= nums[i] {
                        res = res + nums[i-1] + 1 - nums[i]
```

(续下页)

(接上页)

```
                        nums[i] = nums[i-1] + 1
                }
        }
        return res
}

# 2
func minOperations(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        res := 0
        target := nums[0]
        for i := 1; i < n; i++ {
                if target >= nums[i] {
                        res = res + target + 1 - nums[i]
                        target = target + 1
                } else {
                        target = nums[i]
                }
        }
        return res
}
```

55.6 1832. 判断句子是否为全字母句 (3)

- 题目

```
全字母句 指包含英语字母表中每个字母至少一次的句子。
给你一个仅由小写英文字母组成的字符串 sentence ，请你判断sentence 是否为 全字母句 。
如果是，返回 true ；否则，返回 false 。
示例 1：输入：sentence = "thequickbrownfoxjumpsoverthelazydog" 输出：true
解释：sentence 包含英语字母表中每个字母至少一次。
示例 2：输入：sentence = "leetcode" 输出：false
提示：1 <= sentence.length <= 1000
sentence 由小写英语字母组成
```

- 解题思路

```
func checkIfPangram(sentence string) bool {
        arr := [26]int{}
```

(续下页)

(接上页)

```
        for i := 0; i < len(sentence); i++ {
                arr[int(sentence[i]-'a')]++
        }
        for i := 0; i < 26; i++ {
                if arr[i] == 0 {
                        return false
                }
        }
        return true
}

# 2
func checkIfPangram(sentence string) bool {
        res := 0
        target := 1 << 26 -1
        for i := 0; i < len(sentence); i++{
                res = res |  1 << (sentence[i]-'a')
        }
        return res == target
}

# 3
func checkIfPangram(sentence string) bool {
        m := make(map[byte]int)
        for i := 0; i < len(sentence); i++ {
                m[sentence[i]-'a']++
        }
        return len(m) == 26
}
```

55.7 1837.K 进制表示下的各位数字总和 (1)

- 题目

```
给你一个整数 n（10 进制）和一个基数 k ，请你将 n 从 10 进制表示转换为 k 进制表示，
计算并返回转换后各位数字的 总和 。
转换后，各位数字应当视作是 10 进制数字，且它们的总和也应当按 10 进制表示返回。
示例 1：输入：n = 34, k = 6 输出：9
解释：34 (10 进制) 在 6 进制下表示为 54 。5 + 4 = 9 。
示例 2：输入：n = 10, k = 10 输出：1
解释：n 本身就是 10 进制。 1 + 0 = 1 。
提示：1 <= n <= 100
```

(续下页)

(接上页)

```
2 <= k <= 10
```

- 解题思路

```
func sumBase(n int, k int) int {
        res := 0
        for n > 0 {
                res = res + n%k
                n = n / k
        }
        return res
}
```

55.8 1844. 将所有数字用字符替换 (1)

- 题目

```
给你一个下标从 0开始的字符串 s，它的 偶数 下标处为小写英文字母，奇数下标处为数字。
定义一个函数shift(c, x)，其中c是一个字符且x是一个数字，函数返回字母表中c后面第␣
↪x个字符。
比方说，shift('a', 5) = 'f'和shift('x', 0) = 'x'。
对于每个 奇数下标i，你需要将数字s[i] 用shift(s[i-1], s[i])替换。
请你替换所有数字以后，将字符串 s返回。题目 保证shift(s[i-1], s[i])不会超过 'z'。
示例 1：输入：s = "a1c1e1" 输出："abcdef"
解释：数字被替换结果如下：
- s[1] -> shift('a',1) = 'b'
- s[3] -> shift('c',1) = 'd'
- s[5] -> shift('e',1) = 'f'
示例 2：输入：s = "a1b2c3d4e" 输出："abbdcfdhe"
解释：数字被替换结果如下：
- s[1] -> shift('a',1) = 'b'
- s[3] -> shift('b',2) = 'd'
- s[5] -> shift('c',3) = 'f'
- s[7] -> shift('d',4) = 'h'
提示：1 <= s.length <= 100
s只包含小写英文字母和数字。
对所有 奇数 下标处的i，满足shift(s[i-1], s[i]) <= 'z'。
```

- 解题思路

```
func replaceDigits(s string) string {
        res := []byte(s)
```

(续下页)

(接上页)

```
        for i := 1; i < len(s); i = i + 2 {
                res[i] = s[i-1] + s[i] - '0'
        }
        return string(res)
}
```

55.9 1848. 到目标元素的最小距离 (1)

- 题目

```
给你一个整数数组 nums （下标 从 0 开始 计数）以及两个整数 target 和 start␣
↪, 请你找出一个下标 i ,
满足 nums[i] == target 且 abs(i - start) 最小化 。注意：abs(x) 表示 x 的绝对值。
返回 abs(i - start) 。
题目数据保证 target 存在于 nums 中。
示例 1：输入：nums = [1,2,3,4,5], target = 5, start = 3 输出：1
解释：nums[4] = 5 是唯一一个等于 target 的值，所以答案是 abs(4 - 3) = 1 。
示例 2：输入：nums = [1], target = 1, start = 0 输出：0
解释：nums[0] = 1 是唯一一个等于 target 的值，所以答案是 abs(0 - 0) = 1 。
示例 3：输入：nums = [1,1,1,1,1,1,1,1,1,1], target = 1, start = 0 输出：0
解释：nums 中的每个值都是 1 ，但 nums[0] 使 abs(i - start)␣
↪的结果得以最小化，所以答案是 abs(0 - 0) = 0 。
提示：1 <= nums.length <= 1000
1 <= nums[i] <= 104
0 <= start < nums.length
target 存在于 nums 中
```

- 解题思路

```
func getMinDistance(nums []int, target int, start int) int {
        res := math.MaxInt32
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        res = min(res, abs(i-start))
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

55.10 1854. 人口最多的年份 (2)

- 题目

```
给你一个二维整数数组 logs ，其中每个 logs[i] = [birthi, deathi] 表示第 i␣
↪个人的出生和死亡年份。
年份 x 的 人口 定义为这一年期间活着的人的数目。第 i 个人被计入年份 x 的人口需要满足：
x 在闭区间 [birthi, deathi - 1] 内。注意，人不应当计入他们死亡当年的人口中。
返回 人口最多 且 最早 的年份。
示例 1：输入：logs = [[1993,1999],[2000,2010]] 输出：1993
解释：人口最多为 1 ，而 1993 是人口为 1 的最早年份。
示例 2：输入：logs = [[1950,1961],[1960,1971],[1970,1981]] 输出：1960
解释： 人口最多为 2 ，分别出现在 1960 和 1970 。
其中最早年份是 1960 。
提示：1 <= logs.length <= 100
1950 <= birthi < deathi <= 2050
```

- 解题思路

```
func maximumPopulation(logs [][]int) int {
        arr := [101]int{}
        for i := 0; i < len(logs); i++ {
                a, b := logs[i][0], logs[i][1]
                arr[a-1950]++
                arr[b-1950]--
        }
        res := 0
        sum := 0
        count := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i]
```

(续下页)

(接上页)

```
                if sum > count {
                        count = sum
                        res = i + 1950
                }
        }
        return res
}

# 2
func maximumPopulation(logs [][]int) int {
        arr := [101]int{}
        for i := 0; i < len(logs); i++ {
                a, b := logs[i][0], logs[i][1]
                for j := a; j < b; j++ {
                        arr[j-1950]++
                }
        }
        res := 0
        count := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] > count {
                        count = arr[i]
                        res = i + 1950
                }
        }
        return res
}
```

55.11 1859. 将句子排序 (2)

- 题目

```
一个␣
↪句子指的是一个序列的单词用单个空格连接起来，且开头和结尾没有任何空格。每个单词都只包含小写或大写英
我们可以给一个句子添加 从 1 开始的单词位置索引 ，并且将句子中所有单词打乱顺序。
比方说，句子"This is a sentence"可以被打乱顺序得到"sentence4 a3 is2 This1"或者"is2␣
↪sentence4 This1 a3"。
给你一个 打乱顺序的句子s，它包含的单词不超过9个，请你重新构造并得到原本顺序的句子。
示例 1：输入：s = "is2 sentence4 This1 a3"
输出："This is a sentence"
解释：将 s 中的单词按照初始位置排序，得到 "This1 is2 a3 sentence4" ，然后删除数字。
示例 2：输入：s = "Myself2 Me1 I4 and3"
```

(续下页)

(接上页)

```
输出：“Me Myself and I”
解释：将 s 中的单词按照初始位置排序，得到 “Me1 Myself2 and3 I4” ，然后删除数字。
提示：2 <= s.length <= 200
s只包含小写和大写英文字母、空格以及从1到9的数字。
s中单词数目为1到9个。
s中的单词由单个空格分隔。
s不包含任何前导或者后缀空格。
```

- 解题思路

```
func sortSentence(s string) string {
        arr := strings.Split(s, " ")
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][len(arr[i])-1] < arr[j][len(arr[j])-1]
        })
        for i := 0; i < len(arr); i++ {
                arr[i] = arr[i][:len(arr[i])-1]
        }
        return strings.Join(arr, " ")
}

# 2
func sortSentence(s string) string {
        arr := strings.Split(s, " ")
        temp := make([]string, len(arr)+1)
        for i := 0; i < len(arr); i++ {
                index, _ := strconv.Atoi(string(arr[i][len(arr[i])-1]))
                temp[index] = arr[i][:len(arr[i])-1]
        }
        return strings.Join(temp[1:], " ")
}
```

55.12 1863. 找出所有子集的异或总和再求和 (4)

- 题目

```
一个数组的 异或总和 定义为数组中所有元素按位 XOR 的结果；如果数组为 空 ，则异或总和为␣
↪0 。
例如，数组[2,5,6] 的 异或总和 为 2 XOR 5 XOR 6 = 1 。
给你一个数组 nums ，请你求出 nums 中每个 子集 的 异或总和 ，计算并返回这些值相加之 和␣
↪。
注意：在本题中，元素 相同 的不同子集应 多次 计数。
```

(续下页)

(接上页)

```
数组 a 是数组 b 的一个 子集 的前提条件是：从 b 删除几个（也可能不删除）元素能够得到 a␣
↪。
示例 1：输入：nums = [1,3] 输出：6
解释：[1,3] 共有 4 个子集：
- 空子集的异或总和是 0 。
- [1] 的异或总和为 1 。
- [3] 的异或总和为 3 。
- [1,3] 的异或总和为 1 XOR 3 = 2 。
0 + 1 + 3 + 2 = 6
示例 2：输入：nums = [5,1,6] 输出：28
解释：[5,1,6] 共有 8 个子集：
- 空子集的异或总和是 0 。
- [5] 的异或总和为 5 。
- [1] 的异或总和为 1 。
- [6] 的异或总和为 6 。
- [5,1] 的异或总和为 5 XOR 1 = 4 。
- [5,6] 的异或总和为 5 XOR 6 = 3 。
- [1,6] 的异或总和为 1 XOR 6 = 7 。
- [5,1,6] 的异或总和为 5 XOR 1 XOR 6 = 2 。
0 + 5 + 1 + 6 + 4 + 3 + 7 + 2 = 28
示例 3：输入：nums = [3,4,5,6,7,8] 输出：480
解释：每个子集的全部异或总和值之和为 480 。
提示：1 <= nums.length <= 12
1 <= nums[i] <= 20
```

- 解题思路

```
var res int

func subsetXORSum(nums []int) int {
        res = 0
        dfs(nums, 0, 0)
        return res
}

func dfs(nums []int, sum int, index int) {
        if index >= len(nums) {
                res = res + sum
                return
        }
        dfs(nums, sum, index+1)
        dfs(nums, sum^nums[index], index+1)
}
```

(续下页)

(接上页)

```
# 2
func subsetXORSum(nums []int) int {
        res := 0
        n := len(nums)
        left := 1 << n
        right := 1 << (n + 1)
        for i := left; i < right; i++ {
                sum := 0
                for j := 0; j < n; j++ {
                        if i&(1<<j) != 0 {
                                sum = sum ^ nums[j]
                        }
                }
                res = res + sum
        }
        return res
}

# 3
func subsetXORSum(nums []int) int {
        res := 0
        n := len(nums)
        total := 1 << n
        for i := 0; i < total; i++ {
                sum := 0
                for j := 0; j < n; j++ {
                        if (i>>j)&1 == 1 {
                                sum = sum ^ nums[j]
                        }
                }
                res = res + sum
        }
        return res
}

# 4
func subsetXORSum(nums []int) int {
        // 对于任意一位，2^n个子集异或结果中
        //␣
↪如果nums中任何一个数字在这一位上都是0，则任何一个子集的异或结果在这一位上都是0
        //␣
↪如果nums中有一个数字在这一位上是1，则所有子集异或结果中在这一位上，一半是0，一半是1
        n := len(nums)
```

(续下页)

(接上页)

```
        temp := 0
        for i := 0; i < n; i++ {
                temp = temp | nums[i]
        }
        return temp << (n - 1)
}
```

55.13 1869. 哪种连续子字符串更长 (3)

- 题目

```
给你一个二进制字符串 s 。如果字符串中由 1 组成的 最长 连续子字符串 严格长于 由 0␣
↪组成的 最长 连续子字符串，返回 true ；
否则，返回 false 。
例如，s = "110100010" 中，由 1 组成的最长连续子字符串的长度是 2 ，由 0␣
↪组成的最长连续子字符串的长度是 3 。
注意，如果字符串中不存在 0 ，此时认为由 0 组成的最长连续子字符串的长度是 0␣
↪。字符串中不存在 1 的情况也适用此规则。
示例 1：输入：s = "1101" 输出：true
解释：由 1 组成的最长连续子字符串的长度是 2："1101"
由 0 组成的最长连续子字符串的长度是 1："1101"
由 1 组成的子字符串更长，故返回 true 。
示例 2：输入：s = "111000" 输出：false
解释：由 1 组成的最长连续子字符串的长度是 3："111000"
由 0 组成的最长连续子字符串的长度是 3："111000"
由 1 组成的子字符串不比由 0 组成的子字符串长，故返回 false 。
示例 3：输入：s = "110100010" 输出：false
解释：由 1 组成的最长连续子字符串的长度是 2："110100010"
由 0 组成的最长连续子字符串的长度是 3："110100010"
由 1 组成的子字符串不比由 0 组成的子字符串长，故返回 false 。
提示：1 <= s.length <= 100
s[i] 不是 '0' 就是 '1'
```

- 解题思路

```
func checkZeroOnes(s string) bool {
        a, b := 0, 0
        prev := uint8(' ')
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == prev {
                        count++
```

(续下页)

(接上页)

```
                } else {
                        if prev == '0' {
                                a = max(a, count)
                        } else if prev == '1' {
                                b = max(b, count)
                        }
                        count = 1
                }
                prev = s[i]
        }
        if prev == '0' {
                a = max(a, count)
        } else if prev == '1' {
                b = max(b, count)
        }
        return b > a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func checkZeroOnes(s string) bool {
        a, b := 0, 0
        aMax, bMax := 0, 0
        for i := 0; i < len(s); i++ {
                if s[i] == '0' {
                        a++
                        b = 0
                } else if s[i] == '1' {
                        a = 0
                        b++
                }
                aMax = max(aMax, a)
                bMax = max(bMax, b)
        }
        return bMax > aMax
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func checkZeroOnes(s string) bool {
        a, b := 0, 0
        for _, v := range strings.Split(s, "1") {
                b = max(b, len(v))
        }
        for _, v := range strings.Split(s, "0") {
                a = max(a, len(v))
        }
        return b > a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

55.14 1876. 长度为三且各字符不同的子字符串 (1)

- 题目

```
如果一个字符串不含有任何重复字符，我们称这个字符串为 好字符串。
给你一个字符串 s，请你返回 s中长度为 3的 好子字符串 的数量。
注意，如果相同的好子字符串出现多次，每一次都应该被记入答案之中。
子字符串 是一个字符串中连续的字符序列。
示例 1：输入：s = "xyzzaz" 输出：1
解释：总共有 4 个长度为 3 的子字符串："xyz"，"yzz"，"zza" 和 "zaz" 。
唯一的长度为 3 的好子字符串是 "xyz" 。
示例 2：输入：s = "aababcabc" 输出：4
解释：总共有 7 个长度为 3 的子字符串："aab"，"aba"，"bab"，"abc"，"bca"，"cab" 和 "abc
↪" 。
好子字符串包括 "abc"，"bca"，"cab" 和 "abc" 。
提示：1 <= s.length <= 100
```

(续下页)

(接上页)

```
s只包含小写英文字母。
```

- 解题思路

```
func countGoodSubstrings(s string) int {
        res := 0
        for i := 2; i < len(s); i++ {
                if s[i-2] != s[i-1] && s[i-2] != s[i] && s[i-1] != s[i] {
                        res++
                }
        }
        return res
}
```

55.15 1880. 检查某单词是否等于两单词之和 (2)

- 题目

```
字母的 字母值 取决于字母在字母表中的位置，从 0 开始 计数。即，'a' -> 0、'b' -> 1、'c'␣
↪-> 2，以此类推。
对某个由小写字母组成的字符串s 而言，其 数值 就等于将 s 中每个字母的 字母值 按顺序␣
↪连接 并 转换 成对应整数。
例如，s = "acb" ，依次连接每个字母的字母值可以得到 "021" ，转换为整数得到 21 。
给你三个字符串 firstWord、secondWord 和 targetWord ，每个字符串都由从 'a' 到 'j' （含
↪'a' 和 'j' ）的小写英文字母组成。
如果firstWord 和 secondWord 的 数值之和 等于 targetWord 的数值，返回 true␣
↪；否则，返回 false 。
示例 1：输入：firstWord = "acb", secondWord = "cba", targetWord = "cdb" 输出：true
解释：firstWord 的数值为 "acb" -> "021" -> 21
secondWord 的数值为 "cba" -> "210" -> 210
targetWord 的数值为 "cdb" -> "231" -> 231
由于 21 + 210 == 231 ，返回 true
示例 2：输入：firstWord = "aaa", secondWord = "a", targetWord = "aab" 输出：false
解释：firstWord 的数值为 "aaa" -> "000" -> 0
secondWord 的数值为 "a" -> "0" -> 0
targetWord 的数值为 "aab" -> "001" -> 1
由于 0 + 0 != 1 ，返回 false
示例 3：输入：firstWord = "aaa", secondWord = "a", targetWord = "aaaa" 输出：true
解释：firstWord 的数值为 "aaa" -> "000" -> 0
secondWord 的数值为 "a" -> "0" -> 0
targetWord 的数值为 "aaaa" -> "0000" -> 0
由于 0 + 0 == 0 ，返回 true
```

(续下页)

(接上页)

```
提示：1 <= firstWord.length, secondWord.length, targetWord.length <= 8
firstWord、secondWord 和 targetWord 仅由从 'a' 到 'j' （含'a' 和 'j'␣
↪）的小写英文字母组成。
```

- 解题思路

```
func isSumEqual(firstWord string, secondWord string, targetWord string) bool {
        a := 0
        b := 0
        c := 0
        for i := 0; i < len(firstWord); i++ {
                a = a*10 + int(firstWord[i]-'a')
        }
        for i := 0; i < len(secondWord); i++ {
                b = b*10 + int(secondWord[i]-'a')
        }
        for i := 0; i < len(targetWord); i++ {
                c = c*10 + int(targetWord[i]-'a')
        }
        return a+b == c
}

# 2
func isSumEqual(firstWord string, secondWord string, targetWord string) bool {
        return getValue(firstWord)+getValue(secondWord) == getValue(targetWord)
}

func getValue(str string) int {
        res := 0
        for i := 0; i < len(str); i++ {
                res = res*10 + int(str[i]-'a')
        }
        return res
}
```

55.16 1886. 判断矩阵经轮转后是否一致 (1)

- 题目

```
给你两个大小为 n x n 的二进制矩阵 mat 和 target 。
现 以 90 度顺时针轮转 矩阵 mat 中的元素 若干次 ，如果能够使 mat 与target 一致，返回␣
↪true ；否则，返回 false 。
```

(续下页)

(接上页)

```
示例 1：输入：mat = [[0,1],[1,0]], target = [[1,0],[0,1]] 输出：true
解释：顺时针轮转 90 度一次可以使 mat 和 target 一致。
示例 2：输入：mat = [[0,1],[1,1]], target = [[1,0],[0,1]] 输出：false
解释：无法通过轮转矩阵中的元素使 equal 与 target 一致。
示例 3：输入：mat = [[0,0,0],[0,1,0],[1,1,1]], target = [[1,1,1],[0,1,0],[0,0,0]]␣
↪输出：true
解释：顺时针轮转 90 度两次可以使 mat 和 target 一致。
提示：n == mat.length == target.length
n == mat[i].length == target[i].length
1 <= n <= 10
mat[i][j] 和 target[i][j] 不是 0 就是 1
```

- 解题思路

```go
func findRotation(mat [][]int, target [][]int) bool {
        for i := 0; i < 4; i++ {
                rotate(mat)
                if compare(mat, target) == true {
                        return true
                }
        }
        return false
}

// leetcode 48.旋转图像
func rotate(matrix [][]int) {
        n := len(matrix)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        arr[j][n-1-i] = matrix[i][j]
                }
        }
        copy(matrix, arr)
}

func compare(a, b [][]int) bool {
        n := len(a)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if a[i][j] != b[i][j] {
```

(续下页)

(接上页)

```
                                return false
                        }
                }
        }
        return true
}
```

55.17 1893. 检查是否区域内所有整数都被覆盖 (3)

- 题目

```
给你一个二维整数数组ranges和两个整数left和right。
每个ranges[i] = [starti, endi]表示一个从starti到endi的闭区间。
如果闭区间[left,␣
↪right]内每个整数都被ranges中至少一个区间覆盖，那么请你返回true，否则返回false。
已知区间 ranges[i] = [starti, endi] ，如果整数 x 满足 starti <= x <=␣
↪endi，那么我们称整数x被覆盖了。
示例 1：输入：ranges = [[1,2],[3,4],[5,6]], left = 2, right = 5 输出：true
解释：2 到 5 的每个整数都被覆盖了：
- 2 被第一个区间覆盖。
- 3 和 4 被第二个区间覆盖。
- 5 被第三个区间覆盖。
示例 2：输入：ranges = [[1,10],[10,20]], left = 21, right = 21 输出：false
解释：21 没有被任何一个区间覆盖。
提示：1 <= ranges.length <= 50
1 <= starti <= endi <= 50
1 <= left <= right <= 50
```

- 解题思路

```
func isCovered(ranges [][]int, left int, right int) bool {
        arr := make([]bool, 51)
        for i := 0; i < len(ranges); i++ {
                a, b := ranges[i][0], ranges[i][1]
                for j := a; j <= b; j++ {
                        arr[j] = true
                }
        }
        for i := left; i <= right; i++ {
                if arr[i] == false {
                        return false
                }
```

(续下页)

(接上页)

```
        }
        return true
}

# 2
func isCovered(ranges [][]int, left int, right int) bool {
        arr := make([]int, 52)
        for i := 0; i < len(ranges); i++ {
                a, b := ranges[i][0], ranges[i][1]
                arr[a]++
                arr[b+1]--
        }
        sum := 0
        for i := 0; i <= 50; i++ {
                sum = sum + arr[i]
                if left <= i && i <= right && sum <= 0 {
                        return false
                }
        }
        return true
}

# 3
func isCovered(ranges [][]int, left int, right int) bool {
        for i := 0; i < len(ranges); i++ {
                a, b := ranges[i][0], ranges[i][1]
                if a <= left { // a到left已经覆盖：left移动到b后面
                        left = b + 1
                }
                if b >= right { // right到b已经覆盖：right移动到a前面
                        right = a - 1
                }
                if left > right {
                        return true
                }
        }
        return false
}
```

55.18 1897. 重新分配字符使所有字符串都相等 (1)

- 题目

```
给你一个字符串数组 words（下标 从 0 开始 计数）。
在一步操作中，需先选出两个 不同 下标 i 和 j，其中 words[i] 是一个非空字符串，
接着将 words[i] 中的 任一 字符移动到 words[j] 中的 任一 位置上。
如果执行任意步操作可以使 words 中的每个字符串都相等，返回 true ；否则，返回 false 。
示例 1：输入：words = ["abc","aabc","bc"] 输出：true
解释：将 words[1] 中的第一个 'a' 移动到 words[2] 的最前面。
使 words[1] = "abc" 且 words[2] = "abc" 。
所有字符串都等于 "abc" ，所以返回 true 。
示例 2：输入：words = ["ab","a"] 输出：false
解释：执行操作无法使所有字符串都相等。
提示：1 <= words.length <= 100
1 <= words[i].length <= 100
words[i] 由小写英文字母组成
```

- 解题思路

```
func makeEqual(words []string) bool {
        arr := [26]int{}
        n := len(words)
        for i := 0; i < n; i++ {
                for j := 0; j < len(words[i]); j++ {
                        arr[int(words[i][j]-'a')]++
                }
        }
        for i := 0; i < 26; i++ {
                if arr[i]%n != 0 {
                        return false
                }
        }
        return true
}
```

CHAPTER 56

1801-1900-Medium

56.1 1801. 积压订单中的订单总数 (1)

- 题目

```
给你一个二维整数数组 orders ，其中每个 orders[i] = [pricei, amounti, orderTypei]
表示有 amounti 笔类型为orderTypei 、价格为pricei 的订单。
订单类型 orderTypei 可以分为两种：
0 表示这是一批采购订单 buy
1 表示这是一批销售订单 sell
注意，orders[i] 表示一批共计 amounti 笔的独立订单，这些订单的价格和类型相同。
对于所有有效的 i ，由 orders[i] 表示的所有订单提交时间均早于 orders[i+1]␣
↪表示的所有订单。
存在由未执行订单组成的 积压订单 。积压订单最初是空的。提交订单时，会发生以下情况：
如果该订单是一笔采购订单 buy ，则可以查看积压订单中价格 最低 的销售订单 sell 。
如果该销售订单 sell 的价格 低于或等于 当前采购订单 buy 的价格，
则匹配并执行这两笔订单，并将销售订单 sell 从积压订单中删除。否则，采购订单 buy␣
↪将会添加到积压订单中。
反之亦然，如果该订单是一笔销售订单 sell ，则可以查看积压订单中价格 最高 的采购订单␣
↪buy 。
如果该采购订单 buy 的价格 高于或等于 当前销售订单 sell 的价格，
则匹配并执行这两笔订单，并将采购订单 buy 从积压订单中删除。否则，销售订单 sell␣
↪将会添加到积压订单中。
输入所有订单后，返回积压订单中的 订单总数 。由于数字可能很大，所以需要返回对 109 + 7␣
↪取余的结果。
```

(续下页)

(接上页)

```
示例 1：输入：orders = [[10,5,0],[15,2,1],[25,1,1],[30,4,0]] 输出：6
解释：输入订单后会发生下述情况：
- 提交 5 笔采购订单，价格为 10 。没有销售订单，所以这 5 笔订单添加到积压订单中。
- 提交 2 笔销售订单，价格为 15 。没有采购订单的价格大于或等于 15 ，所以这 2
↪笔订单添加到积压订单中。
- 提交 1 笔销售订单，价格为 25 。没有采购订单的价格大于或等于 25 ，所以这 1
↪笔订单添加到积压订单中。
- 提交 4 笔采购订单，价格为 30 。前 2 笔采购订单与价格最低（价格为 15）的 2
↪笔销售订单匹配，
从积压订单中删除这 2 笔销售订单。第 3 笔采购订单与价格最低的 1
↪笔销售订单匹配，销售订单价格为 25 ，
从积压订单中删除这 1 笔销售订单。积压订单中不存在更多销售订单，所以第 4
↪笔采购订单需要添加到积压订单中。
最终，积压订单中有 5 笔价格为 10 的采购订单，和 1 笔价格为 30 的采购订单。
所以积压订单中的订单总数为 6 。
示例 2：输入：orders = [[7,1000000000,1],[15,3,0],[5,999999995,0],[5,1,1]]
↪输出：999999984
解释：输入订单后会发生下述情况：
- 提交 109 笔销售订单，价格为 7 。没有采购订单，所以这 109 笔订单添加到积压订单中。
- 提交 3 笔采购订单，价格为 15 。这些采购订单与价格最低（价格为 7 ）的 3
↪笔销售订单匹配，
从积压订单中删除这 3 笔销售订单。
- 提交 999999995 笔采购订单，价格为 5 。销售订单的最低价为 7 ，
所以这 999999995 笔订单添加到积压订单中。
- 提交 1 笔销售订单，价格为 5 。这笔销售订单与价格最高（价格为 5 ）的 1
↪笔采购订单匹配，
从积压订单中删除这 1 笔采购订单。
最终，积压订单中有 (1000000000-3) 笔价格为 7 的销售订单，和 (999999995-1) 笔价格为 5
↪的采购订单。
所以积压订单中的订单总数为 1999999991 ，等于 999999984 % (109 + 7) 。
提示：1 <= orders.length <= 105
orders[i].length == 3
1 <= pricei, amounti <= 109
orderTypei 为 0 或 1
```

- 解题思路

```
func getNumberOfBacklogOrders(orders [][]int) int {
        res := 0
        buyHeap := make(BuyHeap, 0)
        sellHeap := make(SellHeap, 0)
        heap.Init(&buyHeap)
        heap.Init(&sellHeap)
        for i := 0; i < len(orders); i++ {
```

(续下页)

(接上页)

```
                price := orders[i][0]
                count := orders[i][1]
                typeInt := orders[i][2]
                if typeInt == 1 { // sell
                        for buyHeap.Len() > 0 {
                                node := heap.Pop(&buyHeap).(Node)
                                if node.price < price {
                                        heap.Push(&buyHeap, node)
                                        break
                                }
                                if node.count > count { // 数量大于
                                        node.count = node.count - count
                                        count = 0
                                        heap.Push(&buyHeap, node)
                                        break
                                }
                                count = count - node.count
                        }
                        if count > 0 {
                                heap.Push(&sellHeap, Node{
                                        count: count,
                                        price: price,
                                })
                        }
                } else { // buy
                        for sellHeap.Len() > 0 {
                                node := heap.Pop(&sellHeap).(Node)
                                if node.price > price {
                                        heap.Push(&sellHeap, node)
                                        break
                                }
                                if node.count > count { // 数量小于
                                        node.count = node.count - count
                                        count = 0
                                        heap.Push(&sellHeap, node)
                                        break
                                }
                                count = count - node.count
                        }
                        if count > 0 {
                                heap.Push(&buyHeap, Node{
                                        count: count,
                                        price: price,
```

(续下页)

(接上页)

```
                                })
                        }
                }
        }
        for buyHeap.Len() > 0 {
                node := heap.Pop(&buyHeap).(Node)
                res = res + node.count
        }
        for sellHeap.Len() > 0 {
                node := heap.Pop(&sellHeap).(Node)
                res = res + node.count
        }
        return res % 1000000007
}

type Node struct {
        count int
        price int
}
type SellHeap []Node

func (h SellHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h SellHeap) Less(i, j int) bool {
        return h[i].price < h[j].price
}

func (h SellHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *SellHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *SellHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

(续下页)

(接上页)

```
type BuyHeap []Node

func (h BuyHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h BuyHeap) Less(i, j int) bool {
        return h[i].price > h[j].price
}

func (h BuyHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *BuyHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *BuyHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

56.2 1802. 有界数组中指定下标处的最大值 (2)

- 题目

```
给你三个正整数 n、index 和 maxSum 。
你需要构造一个同时满足下述所有条件的数组 nums（下标 从 0 开始 计数）：
nums.length == n
nums[i] 是 正整数 ，其中 0 <= i < n
abs(nums[i] - nums[i+1]) <= 1 ，其中 0 <= i < n-1
nums 中所有元素之和不超过 maxSum
nums[index] 的值被 最大化
返回你所构造的数组中的 nums[index] 。
注意：abs(x) 等于 x 的前提是 x >= 0 ；否则，abs(x) 等于 -x 。
示例 1：输入：n = 4, index = 2,  maxSum = 6 输出：2
解释：数组 [1,1,2,1] 和 [1,2,2,1]␣
↪满足所有条件。不存在其他在指定下标处具有更大值的有效数组。
```

(续下页)

(接上页)

```
示例 2: 输入: n = 6, index = 1,  maxSum = 10 输出: 3
提示: 1 <= n <= maxSum <= 109
0 <= index < n
```

- 解题思路

```
func maxValue(n int, index int, maxSum int) int {
        if n == 1 {
                return maxSum
        }
        res := 1
        leftTotal, rightTotal := index, n-index-1
        left, right := 1, maxSum
        for left < right {
                mid := left + (right-left)/2
                l := getTotal(mid, leftTotal)
                r := getTotal(mid, rightTotal)
                if l+r+mid <= maxSum {
                        left = mid + 1
                        res = mid
                } else {
                        right = mid
                }
        }
        return res
}

func getTotal(high int, total int) int {
        need := high - 1
        if need >= total {
                return total * (need + high - total) / 2
        }
        return need*(1+need)/2 + total - need
}

# 2
func maxValue(n int, index int, maxSum int) int {
        res := 1
        leftTotal, rightTotal := index, n-index-1
        left, right := 1, maxSum+1
        for left < right {
                mid := left + (right-left)/2
                l := getTotal(mid, leftTotal)
                r := getTotal(mid, rightTotal)
```

(续下页)

(接上页)

```
                if l+r+mid <= maxSum {
                        left = mid + 1
                        res = mid
                } else {
                        right = mid
                }
        }
        return res
}

func getTotal(high int, total int) int {
        need := high - 1
        if need >= total {
                return total * (need + high - total) / 2
        }
        return need*(1+need)/2 + total - need
}
```

56.3 1806. 还原排列的最少操作步数 (3)

- 题目

```
给你一个偶数 n，已知存在一个长度为 n 的排列 perm ，其中 perm[i] == i（下标 从 0 开始␣
↪计数）。
一步操作中，你将创建一个新数组 arr ，对于每个 i ：
如果 i % 2 == 0 ，那么 arr[i] = perm[i / 2]
如果 i % 2 == 1 ，那么 arr[i] = perm[n / 2 + (i - 1) / 2]
然后将 arr赋值给 perm 。
要想使 perm 回到排列初始值，至少需要执行多少步操作？返回最小的 非零 操作步数。
示例 1：输入：n = 2 输出：1
解释：最初，perm = [0,1]
第 1步操作后，perm = [0,1]
所以，仅需执行 1 步操作
示例 2：输入：n = 4 输出：2
解释：最初，perm = [0,1,2,3]
第 1步操作后，perm = [0,2,1,3]
第 2步操作后，perm = [0,1,2,3]
所以，仅需执行 2 步操作
示例 3：输入：n = 6 输出：4
提示：2 <= n <= 1000
n是一个偶数
```

- 解题思路

```
func reinitializePermutation(n int) int {
        res := 0
        target := make([]int, n)
        perm := make([]int, n)
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                perm[i] = i
        }
        copy(target, perm)
        for {
                for i := 0; i < n; i++ {
                        if i%2 == 0 {
                                arr[i] = perm[i/2]
                        } else {
                                arr[i] = perm[n/2+(i-1)/2]
                        }
                }
                res++
                if reflect.DeepEqual(target, arr) {
                        break
                }
                copy(perm, arr)
        }
        return res
}

# 2
func reinitializePermutation(n int) int {
        res := 0
        target := make([]int, n)
        perm := make([]int, n)
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                perm[i] = i
        }
        copy(target, perm)
        for {
                for i := 0; i < n; i++ {
                        if i%2 == 0 {
                                arr[i] = perm[i/2]
                        } else {
                                arr[i] = perm[n/2+(i-1)/2]
                        }
                }
```

(续下页)

(接上页)

```
                res++
                flag := true
                for i := 0; i < n; i++ {
                        if arr[i] != target[i] {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        break
                }
                copy(perm, arr)
        }
        return res
}

# 3
func reinitializePermutation(n int) int {
        res := 0
        target := 1
        // 反向思路，只考虑1的变换
        for {
                if target*2 < n {
                        target = target * 2
                } else {
                        target = target*2 + 1 - n
                }
                res++
                if target == 1 {
                        break
                }
        }
        return res
}
```

56.4 1807. 替换字符串中的括号内容 (1)

- 题目

```
给你一个字符串s，它包含一些括号对，每个括号中包含一个 非空的键。
比方说，字符串"(name)is(age)yearsold"中，有两个括号对，分别包含键"name" 和"age"。
你知道许多键对应的值，这些关系由二维字符串数组knowledge表示，
其中knowledge[i] = [keyi, valuei]，表示键keyi对应的值为valuei。
你需要替换 所有的括号对。当你替换一个括号对，且它包含的键为keyi时，你需要：
将keyi和括号用对应的值valuei替换。
如果从 knowledge中无法得知某个键对应的值，你需要将keyi和括号用问号"?
↪"替换（不需要引号）。
knowledge中每个键最多只会出现一次。s中不会有嵌套的括号。
请你返回替换 所有括号对后的结果字符串。
示例 1：输入：s = "(name)is(age)yearsold", knowledge = [["name","bob"],["age","two"]]
输出："bobistwoyearsold"
解释：键 "name" 对应的值为 "bob" ，所以将 "(name)" 替换为 "bob" 。
键 "age" 对应的值为 "two" ，所以将 "(age)" 替换为 "two" 。
示例 2：输入：s = "hi(name)", knowledge = [["a","b"]] 输出："hi?"
解释：由于不知道键 "name" 对应的值，所以用 "?" 替换 "(name)" 。
示例 3：输入：s = "(a)(a)(a)aaa", knowledge = [["a","yes"]] 输出："yesyesyesaaa"
解释：相同的键在 s 中可能会出现多次。
键 "a" 对应的值为 "yes" ，所以将所有的 "(a)" 替换为 "yes" 。
注意，不在括号里的 "a" 不需要被替换。
示例 4：输入：s = "(a)(b)", knowledge = [["a","b"],["b","a"]] 输出："ba"
提示：1 <= s.length <= 105
0 <= knowledge.length <= 105
knowledge[i].length == 2
1 <= keyi.length, valuei.length <= 10
s只包含小写英文字母和圆括号'('和')'。
s中每一个左圆括号'('都有对应的右圆括号')'。
s中每对括号内的键都不会为空。
s中不会有嵌套括号对。
keyi和valuei只包含小写英文字母。
knowledge中的keyi不会重复。
```

- 解题思路

```
func evaluate(s string, knowledge [][]string) string {
        m := make(map[string]string)
        for i := 0; i < len(knowledge); i++ {
                a, b := knowledge[i][0], knowledge[i][1]
                m[a] = b
        }
```

(续下页)

(接上页)

```
        res := ""
        left := -1
        for i := 0; i < len(s); i++ {
                if s[i] == '(' {
                        left = i
                } else if s[i] == ')' {
                        str := s[left+1 : i]
                        if v, ok := m[str]; ok {
                                res = res + v
                        } else {
                                res = res + "?"
                        }
                        left = -1
                } else if left == -1 {
                        res = res + string(s[i])
                }
        }
        return res
}
```

56.5 1813. 句子相似性 III(1)

- 题目

```
一个句子是由一些单词与它们之间的单个空格组成，且句子的开头和结尾没有多余空格。
比方说，"Hello World", "HELLO", "hello world hello world"都是句子。
每个单词都 只包含大写和小写英文字母。
如果两个句子sentence1 和sentence2，
可以通过往其中一个句子插入一个任意的句子（可以是空句子）而得到另一个句子，那么我们称这两个句子是
↪相似的。
比方说，sentence1 = "Hello my name is Jane" 且sentence2 = "Hello Jane"，
我们可以往 sentence2中"Hello" 和"Jane"之间插入"my name is"得到 sentence1。
给你两个句子sentence1 和sentence2，如果sentence1 和sentence2 是相似的，
请你返回true，否则返回false。
示例 1：输入：sentence1 = "My name is Haley", sentence2 = "My Haley" 输出：true
解释：可以往 sentence2 中 "My" 和 "Haley" 之间插入 "name is" ，得到 sentence1 。
示例 2：输入：sentence1 = "of", sentence2 = "A lot of words" 输出：false
解释：没法往这两个句子中的一个句子只插入一个句子就得到另一个句子。
示例 3：输入：sentence1 = "Eating right now", sentence2 = "Eating" 输出：true
解释：可以往 sentence2 的结尾插入 "right now" 得到 sentence1 。
示例 4：输入：sentence1 = "Luky", sentence2 = "Lucccky" 输出：false
提示：1 <= sentence1.length, sentence2.length <= 100
```

(续下页)

(接上页)

```
sentence1和sentence2都只包含大小写英文字母和空格。
sentence1和sentence2中的单词都只由单个空格隔开。
```

- 解题思路

```
func areSentencesSimilar(sentence1 string, sentence2 string) bool {
        if len(sentence1) > len(sentence2) { // 交换保证 1 < 2
                sentence1, sentence2 = sentence2, sentence1
        }
        // 4种情况:
        // 1、全等:  AB = AB
        // 2、头部等于: AXX = A
        // 3、尾部等于: XXA = A
        // 4、首尾相等: AXXB = AB
        a := strings.Fields(sentence1)
        b := strings.Fields(sentence2)
        i, j := 0, 0
        for i < len(a) && a[i] == b[i] { // 从开头统计
                i++
        }
        for 0 <= len(a)-1-j && a[len(a)-1-j] == b[len(b)-1-j] { // 从结尾统计
                j++
        }
        return i+j >= len(a) // 大于等于较短的长度
}
```

56.6 1814. 统计一个数组中好对子的数目 (1)

- 题目

```
给你一个数组nums，数组中只包含非负整数。定义rev(x)的值为将整数x各个数字位反转得到的结果。
比方说rev(123) = 321, rev(120) = 21。我们称满足下面条件的下标对(i, j)是好的:
0 <= i < j < nums.length
nums[i] + rev(nums[j]) == nums[j] + rev(nums[i])
请你返回好下标对的数目。由于结果可能会很大，请将结果对109 + 7取余后返回。
示例 1: 输入: nums = [42,11,1,97] 输出: 2
解释: 两个坐标对为:
 - (0,3): 42 + rev(97) = 42 + 79 = 121, 97 + rev(42) = 97 + 24 = 121 。
 - (1,2): 11 + rev(1) = 11 + 1 = 12, 1 + rev(11) = 1 + 11 = 12 。
示例 2: 输入: nums = [13,10,35,24,76] 输出: 4
提示: 1 <= nums.length <= 105
0 <= nums[i] <= 109
```

- 解题思路

```
func countNicePairs(nums []int) int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]-reverse(nums[i])]++
        }
        res := 0
        for _, v := range m {
                res = (res + v*(v-1)/2) % 1000000007
        }
        return res
}

func reverse(num int) int {
        res := 0
        for num > 0 {
                res = res*10 + num%10
                num = num / 10
        }
        return res
}
```

56.7 1817. 查找用户活跃分钟数 (1)

- 题目

```
给你用户在 LeetCode 的操作日志，和一个整数 k 。日志用一个二维整数数组 logs 表示，
其中每个 logs[i] = [IDi, timei] 表示 ID 为 IDi 的用户在 timei 分钟时执行了某个操作。
多个用户 可以同时执行操作，单个用户可以在同一分钟内执行 多个操作 。
指定用户的 用户活跃分钟数（user active minutes，UAM） 定义为用户对 LeetCode␣
↪执行操作的 唯一分钟数 。
即使一分钟内执行多个操作，也只能按一分钟计数。
请你统计用户活跃分钟数的分布情况，统计结果是一个长度为 k 且 下标从 1 开始计数 的数组␣
↪answer ，
对于每个 j（1 <= j <= k），answer[j] 表示 用户活跃分钟数 等于 j 的用户数。
返回上面描述的答案数组 answer 。
示例 1：输入：logs = [[0,5],[1,2],[0,2],[0,5],[1,3]], k = 5 输出：[0,2,0,0,0]
解释：
ID=0 的用户执行操作的分钟分别是：5 、2 和 5 。因此，该用户的用户活跃分钟数为 2（分钟␣
↪5 只计数一次）
ID=1 的用户执行操作的分钟分别是：2 和 3 。因此，该用户的用户活跃分钟数为 2
2 个用户的用户活跃分钟数都是 2 ，answer[2] 为 2 ，其余 answer[j] 的值都是 0
```

(续下页)

(接上页)

```
示例 2：输入：logs = [[1,1],[2,2],[2,3]], k = 4 输出：[1,1,0,0]
解释：  ID=1 的用户仅在分钟 1 执行单个操作。因此，该用户的用户活跃分钟数为 1
ID=2 的用户执行操作的分钟分别是：2 和 3 。因此，该用户的用户活跃分钟数为 2
1 个用户的用户活跃分钟数是 1 ，1 个用户的用户活跃分钟数是 2
因此，answer[1] = 1 ，answer[2] = 1 ，其余的值都是 0
提示：1 <= logs.length <= 104
0 <= IDi <= 109
1 <= timei <= 105
k 的取值范围是 [用户的最大用户活跃分钟数，105]
```

- 解题思路

```
func findingUsersActiveMinutes(logs [][]int, k int) []int {
        m := make(map[int]map[int]int)
        for i := 0; i < len(logs); i++{
                a, b := logs[i][0], logs[i][1]
                if _, ok := m[a]; ok == false{
                        m[a] = make(map[int]int)
                }
                m[a][b]++
        }
        res := make([]int, k)
        for _, v := range m{
                value := len(v)
                res[value-1]++
        }
        return res
}
```

56.8 1818. 绝对差值和 (1)

- 题目

```
给你两个正整数数组 nums1 和 nums2 ，数组的长度都是 n 。
数组 nums1 和 nums2 的 绝对差值和 定义为所有 |nums1[i] - nums2[i]| （0 <= i < n）的
总和（下标从 0 开始）。
你可以选用 nums1 中的 任意一个 元素来替换 nums1 中的 至多 一个元素，以 最小化␣
↪绝对差值和。
在替换数组 nums1 中最多一个元素 之后 ，返回最小绝对差值和。
因为答案可能很大，所以需要对 109 + 7 取余 后返回。
|x| 定义为：如果 x >= 0 ，值为 x ，或者
如果 x <= 0 ，值为 -x
```

(续下页)

(接上页)

```
示例 1：输入：nums1 = [1,7,5], nums2 = [2,3,5] 输出：3
解释：有两种可能的最优方案：
- 将第二个元素替换为第一个元素：[1,7,5] => [1,1,5] ，或者
- 将第二个元素替换为第三个元素：[1,7,5] => [1,5,5]
两种方案的绝对差值和都是 |1-2| + (|1-3| 或者 |5-3|) + |5-5| = 3
示例 2：输入：nums1 = [2,4,6,8,10], nums2 = [2,4,6,8,10] 输出：0
解释：nums1 和 nums2 相等，所以不用替换元素。绝对差值和为 0
示例 3：输入：nums1 = [1,10,4,4,2,7], nums2 = [9,3,5,1,7,4] 输出：20
解释：将第一个元素替换为第二个元素：[1,10,4,4,2,7] => [10,10,4,4,2,7]
绝对差值和为 |10-9| + |10-3| + |4-5| + |4-1| + |2-7| + |7-4| = 20
提示：n == nums1.length
n == nums2.length
1 <= n <= 105
1 <= nums1[i], nums2[i] <= 105
```

- 解题思路

```
func minAbsoluteSumDiff(nums1 []int, nums2 []int) int {
        arr := make([]int, len(nums1))
        sum := 0
        for i := 0; i < len(nums1); i++ {
                arr[i] = nums1[i]
                sum = (sum + abs(nums1[i]-nums2[i])) % 1000000007
        }
        sort.Ints(arr)
        maxValue := 0
        for i := 0; i < len(arr); i++ {
                if nums1[i] == nums2[i] {
                        continue
                }
                b := nums2[i]
                target := search(arr, b)
                maxValue = max(maxValue, abs(nums1[i]-b)-abs(target-b))
        }
        return (sum - maxValue+ 1000000007) % 1000000007
}

func search(arr []int, target int) int {
        res := 0
        if arr[0] > target {
                return arr[0]
        }
        if arr[len(arr)-1] < target {
                return arr[len(arr)-1]
```

(续下页)

(接上页)

```
        }
        left, right := 0, len(arr)-1
        for left <= right {
                mid := left + (right-left)/2
                if target < arr[mid] {
                        right = mid - 1
                        if abs(res-target) > abs(target-arr[mid]) {
                                res = arr[mid]
                        }
                } else if target == arr[mid] {
                        return target
                } else if target > arr[mid] {
                        left = mid + 1
                        if abs(res-target) > abs(target-arr[mid]) {
                                res = arr[mid]
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

56.9 1823. 找出游戏的获胜者 (2)

- 题目

```
共有 n 名小伙伴一起做游戏。小伙伴们围成一圈，按 顺时针顺序 从 1 到 n 编号。
确切地说，从第 i 名小伙伴顺时针移动一位会到达第 (i+1) 名小伙伴的位置，其中 1 <= i < n␣
↪，
从第 n 名小伙伴顺时针移动一位会回到第 1 名小伙伴的位置。
游戏遵循如下规则：
从第 1 名小伙伴所在位置 开始 。
沿着顺时针方向数 k 名小伙伴，计数时需要 包含 起始时的那位小伙伴。
逐个绕圈进行计数，一些小伙伴可能会被数过不止一次。
你数到的最后一名小伙伴需要离开圈子，并视作输掉游戏。
如果圈子中仍然有不止一名小伙伴，从刚刚输掉的小伙伴的 顺时针下一位 小伙伴␣
↪开始，回到步骤 2 继续执行。
否则，圈子中最后一名小伙伴赢得游戏。
给你参与游戏的小伙伴总数 n ，和一个整数 k ，返回游戏的获胜者。
示例 1：输入：n = 5, k = 2 输出：3
解释：游戏运行步骤如下：
1) 从小伙伴 1 开始。
2) 顺时针数 2 名小伙伴，也就是小伙伴 1 和 2 。
3) 小伙伴 2 离开圈子。下一次从小伙伴 3 开始。
4) 顺时针数 2 名小伙伴，也就是小伙伴 3 和 4 。
5) 小伙伴 4 离开圈子。下一次从小伙伴 5 开始。
6) 顺时针数 2 名小伙伴，也就是小伙伴 5 和 1 。
7) 小伙伴 1 离开圈子。下一次从小伙伴 3 开始。
8) 顺时针数 2 名小伙伴，也就是小伙伴 3 和 5 。
9) 小伙伴 5 离开圈子。只剩下小伙伴 3 。所以小伙伴 3 是游戏的获胜者。
示例 2：输入：n = 6, k = 5 输出：1
解释：小伙伴离开圈子的顺序：5、4、6、2、3 。小伙伴 1 是游戏的获胜者。
提示：1 <= k <= n <= 500
```

- 解题思路

```
func findTheWinner(n int, k int) int {
        idx := 0
        for i := 2; i <= n; i++ {
                idx = (idx + k) % i
        }
        return idx + 1
}

# 2
func findTheWinner(n int, k int) int {
```

(续下页)

(接上页)

```
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        last := 0
        for len(arr) > 1 {
                index := (last + k - 1) % len(arr)
                arr = remove(arr, index)
                last = index
        }
        return arr[0] + 1
}

func remove(arr []int, index int) []int {
        if index == 0 {
                return arr[1:]
        }
        if index == len(arr)-1 {
                return arr[:len(arr)-1]
        }
        return append(arr[:index], arr[index+1:]...)
}
```

56.10 1824. 最少侧跳次数 (2)

- 题目

```
给你一个长度为n的3 跑道道路，它总共包含n + 1个点，编号为0到n。
一只青蛙从0号点第二条跑道出发，它想要跳到点n处。然而道路上可能有一些障碍。
给你一个长度为 n + 1的数组obstacles，其中obstacles[i]（取值范围从 0 到 3）
表示在点 i处的obstacles[i]跑道上有一个障碍。
如果obstacles[i] == 0，那么点i处没有障碍。任何一个点的三条跑道中最多有一个障碍。
比方说，如果obstacles[2] == 1，那么说明在点 2 处跑道 1 有障碍。
这只青蛙从点 i跳到点 i + 1且跑道不变的前提是点 i + 1的同一跑道上没有障碍。
为了躲避障碍，这只青蛙也可以在同一个点处侧跳到 另外一条跑道（这两条跑道可以不相邻），
但前提是跳过去的跑道该点处没有障碍。
比方说，这只青蛙可以从点 3 处的跑道 3 跳到点 3 处的跑道 1 。
这只青蛙从点 0 处跑道 2出发，并想到达点 n处的 任一跑道 ，请你返回 最少侧跳次数。
注意：点 0处和点 n处的任一跑道都不会有障碍。
示例 1：输入：obstacles = [0,1,2,3,0] 输出：2
解释：最优方案如上图箭头所示。总共有 2 次侧跳（红色箭头）。
注意，这只青蛙只有当侧跳时才可以跳过障碍（如上图点 2 处所示）。
```

(续下页)

(接上页)

```
示例 2：输入：obstacles = [0,1,1,3,3,0] 输出：0
解释：跑道 2 没有任何障碍，所以不需要任何侧跳。
示例 3：输入：obstacles = [0,2,1,0,3,0] 输出：2
解释：最优方案如上图所示。总共有 2 次侧跳。
提示：obstacles.length == n + 1
1 <= n <= 5 * 105
0 <= obstacles[i] <= 3
obstacles[0] == obstacles[n] == 0
```

- 解题思路

```
func minSideJumps(obstacles []int) int {
        n := len(obstacles)
        dp := make([][3]int, n) // dp[i][j] 到达第i，下标为j的跑道最少次数
        for i := 0; i < n; i++ {
                for j := 0; j < 3; j++ {
                        dp[i][j] = math.MaxInt32 / 10
                }
        }
        dp[0][0] = 1
        dp[0][1] = 0
        dp[0][2] = 1
        for i := 1; i < n; i++ {
                // 当前位置无障碍物，继承之前的次数，次数不变
                if obstacles[i] != 1 {
                        dp[i][0] = dp[i-1][0]
                }
                if obstacles[i] != 2 {
                        dp[i][1] = dp[i-1][1]
                }
                if obstacles[i] != 3 {
                        dp[i][2] = dp[i-1][2]
                }
                // 从其它位置跳过来
                if obstacles[i] != 1 {
                        dp[i][0] = min(dp[i][0], min(dp[i][1], dp[i][2])+1)
                }
                if obstacles[i] != 2 {
                        dp[i][1] = min(dp[i][1], min(dp[i][0], dp[i][2])+1)
                }
                if obstacles[i] != 3 {
                        dp[i][2] = min(dp[i][2], min(dp[i][0], dp[i][1])+1)
                }
        }
```

(续下页)

(接上页)

```
        return min(dp[n-1][0], min(dp[n-1][1], dp[n-1][2]))
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minSideJumps(obstacles []int) int {
        n := len(obstacles)
        dp := make([][3]int, n) // dp[i][j] 到达第i, 下标为j的跑道最少次数
        for i := 0; i < n; i++ {
                for j := 0; j < 3; j++ {
                        dp[i][j] = math.MaxInt32 / 10
                }
        }
        dp[0][0] = 1
        dp[0][1] = 0
        dp[0][2] = 1
        for i := 1; i < n; i++ {
                if obstacles[i] == 0 { // 没有障碍物, 从其它2条道跳过来
                        dp[i][0] = min(dp[i-1][0], min(dp[i-1][1], dp[i-1][2])+1)
                        dp[i][1] = min(dp[i-1][1], min(dp[i-1][0], dp[i-1][2])+1)
                        dp[i][2] = min(dp[i-1][2], min(dp[i-1][0], dp[i-1][1])+1)
                } else {
                        a := obstacles[i] - 1
                        b := (obstacles[i]) % 3
                        c := (obstacles[i] + 1) % 3
                        dp[i][a] = math.MaxInt32 / 10 // 不可达
                        dp[i][b] = min(dp[i-1][b], dp[i-1][c]+1)
                        dp[i][c] = min(dp[i-1][c], dp[i-1][b]+1)
                }

        }
        return min(dp[n-1][0], min(dp[n-1][1], dp[n-1][2]))
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}
```

56.11 1828. 统计一个圆中点的数目 (1)

- 题目

```
给你一个数组points，其中points[i] = [xi, yi]，表示第i个点在二维平面上的坐标。
多个点可能会有 相同的坐标。
同时给你一个数组queries，其中queries[j] = [xj, yj, rj]，
表示一个圆心在(xj, yj)且半径为rj的圆。
对于每一个查询queries[j]，计算在第 j个圆 内点的数目。
如果一个点在圆的 边界上，我们同样认为它在圆内。
请你返回一个数组answer，其中answer[j]是第j个查询的答案。
示例 1： 输入：points = [[1,3],[3,3],[5,3],[2,2]], queries = [[2,3,1],[4,3,1],[1,1,2]]
输出：[3,2,2]
解释：所有的点和圆如上图所示。
queries[0] 是绿色的圆，queries[1] 是红色的圆，queries[2] 是蓝色的圆。
示例 2：输入：points = [[1,1],[2,2],[3,3],[4,4],[5,5]],
queries = [[1,2,2],[2,2,2],[4,3,2],[4,3,3]] 输出：[2,3,2,4]
解释：所有的点和圆如上图所示。
queries[0] 是绿色的圆，queries[1] 是红色的圆，queries[2] 是蓝色的圆，queries[3]␣
↪是紫色的圆。
提示：1 <= points.length <= 500
points[i].length == 2
0 <= xi, yi <= 500
1 <= queries.length <= 500
queries[j].length == 3
0 <= xj, yj <= 500
1 <= rj <= 500
所有的坐标都是整数。
```

- 解题思路

```
func countPoints(points [][]int, queries [][]int) []int {
        n := len(queries)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                count := 0
                for j := 0; j < len(points); j++ {
                        if judge(queries[i], points[j]) == true {
```

(续下页)

(接上页)

```
                                count++
                        }
                }
                res[i] = count
        }
        return res
}

func judge(query []int, point []int) bool {
        x, y, r := query[0], query[1], query[2]
        x1, y1 := point[0], point[1]
        return (x-x1)*(x-x1)+(y-y1)*(y-y1) <= r*r
}
```

56.12 1829. 每个查询的最大异或值 (2)

- 题目

```
给你一个 有序数组nums，它由n个非负整数组成，同时给你一个整数maximumBit。
你需要执行以下查询 n次：
找到一个非负整数k < 2maximumBit，
使得nums[0] XOR nums[1] XOR ... XOR nums[nums.length-1] XOR k的结果 最大化。
k是第 i个查询的答案。
从当前数组nums删除最后一个元素。
请你返回一个数组answer，其中answer[i]是第i个查询的结果。
示例 1：输入：nums = [0,1,1,3]，maximumBit = 2 输出：[0,3,2,3]
解释：查询的答案如下：
第一个查询：nums = [0,1,1,3]，k = 0，因为 0 XOR 1 XOR 1 XOR 3 XOR 0 = 3 。
第二个查询：nums = [0,1,1]，k = 3，因为 0 XOR 1 XOR 1 XOR 3 = 3 。
第三个查询：nums = [0,1]，k = 2，因为 0 XOR 1 XOR 2 = 3 。
第四个查询：nums = [0]，k = 3，因为 0 XOR 3 = 3 。
示例 2：输入：nums = [2,3,4,7]，maximumBit = 3 输出：[5,2,6,5]
解释：查询的答案如下：
第一个查询：nums = [2,3,4,7]，k = 5，因为 2 XOR 3 XOR 4 XOR 7 XOR 5 = 7。
第二个查询：nums = [2,3,4]，k = 2，因为 2 XOR 3 XOR 4 XOR 2 = 7 。
第三个查询：nums = [2,3]，k = 6，因为 2 XOR 3 XOR 6 = 7 。
第四个查询：nums = [2]，k = 5，因为 2 XOR 5 = 7 。
示例 3：输入：nums = [0,1,2,2,5,7]，maximumBit = 3 输出：[4,3,6,4,6,7]
提示：nums.length == n
1 <= n <= 105
1 <= maximumBit <= 20
0 <= nums[i] < 2maximumBit
```

(续下页)

(接上页)

nums中的数字已经按升序排好序。

- 解题思路

```
func getMaximumXor(nums []int, maximumBit int) []int {
        n := len(nums)
        res := make([]int, n)
        temp := nums[0]
        res[n-1] = temp
        for i := 1; i < n; i++ {
                temp = temp ^ nums[i]
                res[n-1-i] = temp
        }
        target := 1<<maximumBit - 1
        for i := 0; i < n; i++ {
                res[i] = res[i] ^ target
        }
        return res
}

# 2
func getMaximumXor(nums []int, maximumBit int) []int {
        n := len(nums)
        res := make([]int, 0)
        temp := nums[0]
        for i := 1; i < n; i++ {
                temp = temp ^ nums[i]
        }
        target := 1<<maximumBit - 1
        for i := n - 1; i >= 0; i-- {
                res = append(res, temp^target)
                temp = temp ^ nums[i]
        }
        return res
}
```

56.13 1833. 雪糕的最大数量 (1)

- 题目

```
夏日炎炎，小男孩 Tony 想买一些雪糕消消暑。
商店中新到 n 支雪糕，用长度为 n 的数组 costs 表示雪糕的定价，其中 costs[i] 表示第 i 支雪糕的现金价格。
Tony 一共有 coins 现金可以用于消费，他想要买尽可能多的雪糕。
给你价格数组 costs 和现金量 coins ，请你计算并返回 Tony 用 coins 现金能够买到的雪糕的最大数量 。
注意：Tony 可以按任意顺序购买雪糕。
示例 1：输入：costs = [1,3,2,4,1], coins = 7 输出：4
解释：Tony 可以买下标为 0、1、2、4 的雪糕，总价为 1 + 3 + 2 + 1 = 7
示例 2：输入：costs = [10,6,8,7,7,8], coins = 5 输出：0
解释：Tony 没有足够的钱买任何一支雪糕。
示例 3：输入：costs = [1,6,3,1,2,5], coins = 20 输出：6
解释：Tony 可以买下所有的雪糕，总价为 1 + 6 + 3 + 1 + 2 + 5 = 18 。
提示：costs.length == n
1 <= n <= 105
1 <= costs[i] <= 105
1 <= coins <= 108
```

- 解题思路

```
func maxIceCream(costs []int, coins int) int {
        sort.Ints(costs)
        for i := 0; i < len(costs); i++ {
                if costs[i] <= coins {
                        coins = coins - costs[i]
                } else {
                        return i
                }
        }
        return len(costs)
}
```

56.14 1834. 单线程 CPU(1)

- 题目

```
给你一个二维数组 tasks ，用于表示 n项从 0 到 n - 1 编号的任务。
其中 tasks[i] = [enqueueTimei, processingTimei] 意味着
第 i项任务将会于 enqueueTimei 时进入任务队列，需要 processingTimei 的时长完成执行。
现有一个单线程 CPU ，同一时间只能执行 最多一项 任务，该 CPU 将会按照下述方式运行：
如果 CPU 空闲，且任务队列中没有需要执行的任务，则 CPU 保持空闲状态。
如果 CPU 空闲，但任务队列中有需要执行的任务，则 CPU 将会选择 执行时间最短␣
↪的任务开始执行。
如果多个任务具有同样的最短执行时间，则选择下标最小的任务开始执行。
一旦某项任务开始执行，CPU 在 执行完整个任务 前都不会停止。
CPU 可以在完成一项任务后，立即开始执行一项新任务。
返回 CPU 处理任务的顺序。
示例 1：输入：tasks = [[1,2],[2,4],[3,2],[4,1]] 输出：[0,2,3,1]
解释：事件按下述流程运行：
- time = 1 ，任务 0 进入任务队列，可执行任务项 = {0}
- 同样在 time = 1 ，空闲状态的 CPU 开始执行任务 0 ，可执行任务项 = {}
- time = 2 ，任务 1 进入任务队列，可执行任务项 = {1}
- time = 3 ，任务 2 进入任务队列，可执行任务项 = {1, 2}
- 同样在 time = 3 ，CPU 完成任务 0 并开始执行队列中用时最短的任务 2 ，可执行任务项 =
↪{1}
- time = 4 ，任务 3 进入任务队列，可执行任务项 = {1, 3}
- time = 5 ，CPU 完成任务 2 并开始执行队列中用时最短的任务 3 ，可执行任务项 = {1}
- time = 6 ，CPU 完成任务 3 并开始执行任务 1 ，可执行任务项 = {}
- time = 10 ，CPU 完成任务 1 并进入空闲状态
示例 2：输入：tasks = [[7,10],[7,12],[7,5],[7,4],[7,2]] 输出：[4,3,2,0,1]
解释：事件按下述流程运行：
- time = 7 ，所有任务同时进入任务队列，可执行任务项 = {0,1,2,3,4}
- 同样在 time = 7 ，空闲状态的 CPU 开始执行任务 4 ，可执行任务项 = {0,1,2,3}
- time = 9 ，CPU 完成任务 4 并开始执行任务 3 ，可执行任务项 = {0,1,2}
- time = 13 ，CPU 完成任务 3 并开始执行任务 2 ，可执行任务项 = {0,1}
- time = 18 ，CPU 完成任务 2 并开始执行任务 0 ，可执行任务项 = {1}
- time = 28 ，CPU 完成任务 0 并开始执行任务 1 ，可执行任务项 = {}
- time = 40 ，CPU 完成任务 1 并进入空闲状态
提示：tasks.length == n
1 <= n <= 105
1 <= enqueueTimei, processingTimei <= 109
```

- 解题思路

```
func getOrder(tasks [][]int) []int {
        n := len(tasks)
```

(续下页)

(接上页)

```
        res := make([]int, 0)
        arr := make([]Node, 0)
        for i := 0; i < n; i++ {
                arr = append(arr, Node{
                        Id:             i,
                        StartTime:      tasks[i][0],
                        ProcessingTime: tasks[i][1],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].StartTime < arr[j].StartTime
        })
        nodeHeap := make(NodeHeap, 0)
        heap.Init(&nodeHeap)
        curTime := 0
        cur := 0
        for i := 0; i < n; i++ { // 每次处理1个任务
                if nodeHeap.Len() == 0 { // 空任务
                        curTime = max(curTime, tasks[arr[cur].Id][0]) // 时间移动
                }
                for ; cur < n && arr[cur].StartTime <= curTime; cur++ { //␣
↪加入优先队列：将小于等于时间戳的任务加入堆
                        heap.Push(&nodeHeap, arr[cur])
                }
                node := heap.Pop(&nodeHeap).(Node)       //␣
↪选择任务：选择处理时间小的任务
                curTime = curTime + node.ProcessingTime // 时间处理
                res = append(res, node.Id)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type Node struct {
        Id             int
        StartTime      int
        ProcessingTime int
```

(续下页)

(接上页)

```
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h NodeHeap) Less(i, j int) bool {
        if h[i].ProcessingTime == h[j].ProcessingTime {
                return h[i].Id < h[j].Id
        }
        return h[i].ProcessingTime < h[j].ProcessingTime
}

func (h NodeHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

56.15 1838. 最高频元素的频数 (3)

- 题目

```
元素的 频数 是该元素在一个数组中出现的次数。
给你一个整数数组 nums 和一个整数 k 。在一步操作中，你可以选择 nums 的一个下标，
并将该下标对应元素的值增加 1 。
执行最多 k 次操作后，返回数组中最高频元素的 最大可能频数 。
示例 1：输入：nums = [1,2,4], k = 5 输出：3
解释：对第一个元素执行 3 次递增操作，对第二个元素执 2 次递增操作，此时 nums = [4,4,4]␣
↪。
4 是数组中最高频元素，频数是 3 。
```

(续下页)

(接上页)

```
示例 2: 输入: nums = [1,4,8,13], k = 5 输出: 2
解释: 存在多种最优解决方案:
- 对第一个元素执行 3 次递增操作, 此时 nums = [4,4,8,13] 。4␣
↪是数组中最高频元素, 频数是 2 。
- 对第二个元素执行 4 次递增操作, 此时 nums = [1,8,8,13] 。8␣
↪是数组中最高频元素, 频数是 2 。
- 对第三个元素执行 5 次递增操作, 此时 nums = [1,4,13,13] 。13␣
↪是数组中最高频元素, 频数是 2 。
示例 3: 输入: nums = [3,9,6], k = 2 输出: 1
提示: 1 <= nums.length <= 105
1 <= nums[i] <= 105
1 <= k <= 105
```

- 解题思路

```
func maxFrequency(nums []int, k int) int {
        n := len(nums)
        sort.Ints(nums)
        arr := make([]int, n+1)
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        res := 1
        i := 0
        for j := 0; j < n; j++ {
                for nums[j]*(j-i)-(arr[j]-arr[i]) > k {
                        i++
                }
                if j-i+1 > res {
                        res = j - i + 1
                }
        }
        return res
}

# 2
func maxFrequency(nums []int, k int) int {
        n := len(nums)
        sort.Ints(nums)
        res := 1
        total := 0
        i := 0
        for j := 1; j < n; j++ {
                total = total + (nums[j]-nums[j-1])*(j-i) // 累加
```

(续下页)

(接上页)

```
                for total > k {
                        total = total - (nums[j] - nums[i]) // 不满足，要减去
                        i++
                }
                if j-i+1 > res {
                        res = j - i + 1
                }
        }
        return res
}

# 3
func maxFrequency(nums []int, k int) int {
        n := len(nums)
        sort.Ints(nums)
        arr := make([]int, n+1)
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        res := 1
        for j := 0; j < n; j++ {
                i := sort.Search(j, func(i int) bool {
                        return nums[j]*(j-i)-(arr[j]-arr[i]) <= k
                })
                if j-i+1 > res {
                        res = j - i + 1
                }
        }
        return res
}
```

56.16 1839. 所有元音按顺序排布的最长子字符串 (1)

- 题目

```
当一个字符串满足如下条件时，我们称它是 美丽的：
所有 5 个英文元音字母（'a', 'e', 'i', 'o', 'u'）都必须至少出现一次。
这些元音字母的顺序都必须按照 字典序升序排布（也就是说所有的 'a'都在 'e'前面，
所有的 'e'都在 'i'前面，以此类推）
比方说，字符串"aeiou" 和"aaaaaaeiiiioou"都是 美丽的，
但是"uaeio", "aeoiu"和"aaaeeeooo"不是美丽的。
给你一个只包含英文元音字母的字符串word，请你返回word 中 最长美丽子字符串的长度。
```

(续下页)

(接上页)

```
如果不存在这样的子字符串，请返回 0。
子字符串 是字符串中一个连续的字符序列。
示例 1：输入：word = "aeiaaioaaaaeiiiiouuuooaauuaeiu" 输出：13
解释：最长子字符串是 "aaaaeiiiiouuu" ，长度为 13 。
示例 2：输入：word = "aeeeiiiioooauuuaeiou" 输出：5
解释：最长子字符串是 "aeiou" ，长度为 5 。
示例 3：输入：word = "a" 输出：0
解释：没有美丽子字符串，所以返回 0 。
提示：1 <= word.length <= 5 * 105
word只包含字符'a', 'e', 'i', 'o'和'u'。
```

- 解题思路

```
func longestBeautifulSubstring(word string) int {
        res := 0
        count := 1
        i := 0
        for j := 1; j < len(word); j++ { // 题目保证了全是目标元素，而且顺序是有序的
                if word[j] < word[j-1] { // 大小不对，窗口右移，重新计数
                        i = j
                        count = 1
                } else if word[j] > word[j-1] {
                        count++
                }
                if count == 5 { // 长度为5
                        res = max(res, j-i+1)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

56.17 1845. 座位预约管理系统 (1)

- 题目

```
请你设计一个管理 n个座位预约的系统，座位编号从1到n。
请你实现SeatManager类：
SeatManager(int n)初始化一个SeatManager对象，它管理从 1到n编号的n个座位。所有座位初始都是可预约的。
int reserve()返回可以预约座位的最小编号，此座位变为不可预约。
void unreserve(int seatNumber)将给定编号seatNumber对应的座位变成可以预约。
示例 1：输入：["SeatManager", "reserve", "reserve", "unreserve", "reserve", "reserve", "reserve",
"reserve", "unreserve"]
[[5], [], [], [2], [], [], [], [], [5]]
输出：  [null, 1, 2, null, 2, 3, 4, 5, null]
解释：
SeatManager seatManager = new SeatManager(5); // 初始化 SeatManager ，有 5 个座位。
seatManager.reserve();    // 所有座位都可以预约，所以返回最小编号的座位，也就是 1 。
seatManager.reserve();    // 可以预约的座位为 [2,3,4,5] ，返回最小编号的座位，也就是2 。
seatManager.unreserve(2); // 将座位 2 变为可以预约，现在可预约的座位为 [2,3,4,5] 。
seatManager.reserve();    // 可以预约的座位为 [2,3,4,5] ，返回最小编号的座位，也就是2 。
seatManager.reserve();    // 可以预约的座位为 [3,4,5] ，返回最小编号的座位，也就是 3。
seatManager.reserve();    // 可以预约的座位为 [4,5] ，返回最小编号的座位，也就是 4 。
seatManager.reserve();    // 唯一可以预约的是座位 5 ，所以返回 5 。
seatManager.unreserve(5); // 将座位 5 变为可以预约，现在可预约的座位为 [5] 。
提示：1 <= n <= 105
1 <= seatNumber <= n
每一次对reserve的调用，题目保证至少存在一个可以预约的座位。
每一次对unreserve的调用，题目保证seatNumber在调用函数前都是被预约状态。
对reserve 和unreserve的调用总共不超过105次。
```

- 解题思路

```go
type SeatManager struct {
        intHeap IntHeap
}

func Constructor(n int) SeatManager {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 1; i <= n; i++ {
```

(续下页)

(接上页)

```
                heap.Push(&intHeap, i)
        }
        return SeatManager{intHeap: intHeap}
}

func (this *SeatManager) Reserve() int {
        top := heap.Pop(&this.intHeap).(int)
        return top
}

func (this *SeatManager) Unreserve(seatNumber int) {
        heap.Push(&this.intHeap, seatNumber)
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

56.18 1846. 减小和重新排列数组后的最大元素 (2)

- 题目

```
给你一个正整数数组arr。请你对␣
↪arr执行一些操作（也可以不进行任何操作），使得数组满足以下条件：
arr中 第一个元素必须为1。
任意相邻两个元素的差的绝对值 小于等于1，也就是说，对于任意的 1 <= i < arr.
↪length（数组下标从 0 开始），
都满足abs(arr[i] - arr[i - 1]) <= 1。abs(x)为x的绝对值。
你可以执行以下 2 种操作任意次：
减小 arr中任意元素的值，使其变为一个 更小的正整数。
重新排列arr中的元素，你可以以任意顺序重新排列。
请你返回执行以上操作后，在满足前文所述的条件下，arr中可能的 最大值。
示例 1：输入：arr = [2,2,1,2,1] 输出：2
解释：我们可以重新排列 arr 得到 [1,2,2,2,1] ，该数组满足所有条件。
arr 中最大元素为 2 。
示例 2：输入：arr = [100,1,1000] 输出：3
解释：一个可行的方案如下：
1. 重新排列 arr 得到 [1,100,1000] 。
2. 将第二个元素减小为 2 。
3. 将第三个元素减小为 3 。
现在 arr = [1,2,3] ，满足所有条件。
arr 中最大元素为 3 。
示例 3：输入：arr = [1,2,3,4,5] 输出：5
解释：数组已经满足所有条件，最大元素为 5 。
提示：1 <= arr.length <= 105
1 <= arr[i] <= 109
```

- 解题思路

```
func maximumElementAfterDecrementingAndRearranging(arr []int) int {
        sort.Ints(arr)
        n := len(arr)
        arr[0] = 1
        for i := 1; i < n; i++ {
                arr[i] = min(arr[i], arr[i-1]+1)
        }
        return arr[n-1]
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}

# 2
func maximumElementAfterDecrementingAndRearranging(arr []int) int {
        sort.Ints(arr)
        res := 0
        for i := 0; i < len(arr); i++ {
                if res < arr[i] {
                        res++
                }
        }
        return res
}
```

56.19 1849. 将字符串拆分为递减的连续值 (3)

- 题目

```
给你一个仅由数字组成的字符串 s 。
请你判断能否将 s 拆分成两个或者多个 非空子字符串 ，使子字符串的 数值 按 降序␣
↪排列，且每两个 相邻子字符串 的数值之 差 等于 1 。
例如，字符串 s = "0090089" 可以拆分成 ["0090", "089"] ，数值为 [90,89] 。
这些数值满足按降序排列，且相邻值相差 1 ，这种拆分方法可行。
另一个例子中，字符串 s = "001" 可以拆分成 ["0", "01"]、["00", "1"] 或 ["0", "0", "1"]␣
↪。
然而，所有这些拆分方法都不可行，因为对应数值分别是 [0,1]、[0,1] 和 [0,0,1]␣
↪，都不满足按降序排列的要求。
如果可以按要求拆分 s ，返回 true ；否则，返回 false 。
子字符串 是字符串中的一个连续字符序列。
示例 1：输入：s = "1234" 输出：false
解释：不存在拆分 s 的可行方法。
示例 2：输入：s = "050043" 输出：true
解释：s 可以拆分为 ["05", "004", "3"] ，对应数值为 [5,4,3] 。
满足按降序排列，且相邻值相差 1 。
示例 3：输入：s = "9080701" 输出：false
解释：不存在拆分 s 的可行方法。
示例 4：输入：s = "10009998" 输出：true
解释：s 可以拆分为 ["100", "099", "98"] ，对应数值为 [100,99,98] 。
满足按降序排列，且相邻值相差 1 。
提示：1 <= s.length <= 20
```

(续下页)

(接上页)

```
s 仅由数字组成
```

- 解题思路

```
func splitString(s string) bool {
        for i := 0; i < len(s); i++ {
                value, _ := strconv.Atoi(s[:i+1])
                if s[i+1:] != "" && dfs(s[i+1:], value-1) == true {
                        return true
                }
        }
        return false
}

func dfs(s string, target int) bool {
        value, _ := strconv.Atoi(s)
        if s == "" || value == target {
                return true
        }
        for i := 0; i < len(s); i++ {
                v, _ := strconv.Atoi(s[:i+1])
                if v < target {
                        continue
                }
                if v > target {
                        return false
                }
                if v == target {
                        if dfs(s[i+1:], target-1) == true {
                                return true
                        }
                        return false
                }
        }
        return false
}

# 2
func splitString(s string) bool {
        return dfs([]byte(s), 0, 0, 0)
}

func dfs(arr []byte, index int, count int, target int) bool {
        if index == len(arr) {
```

(续下页)

(接上页)

```
                return count > 1
        }
        value := 0
        for i := index; i < len(arr); i++ {
                value = value*10 + int(arr[i]-'0')
                if count == 0 || value == target-1 {
                        if dfs(arr, i+1, count+1, value) == true {
                                return true
                        }
                }
        }
        return false
}

# 3
func splitString(s string) bool {
        n := len(s)
        for i := 0; i < n-1; i++ {
                a, _ := strconv.Atoi(s[0 : i+1])
                index := i
                for j := i + 1; j < n; j++ {
                        b, _ := strconv.Atoi(s[index+1 : j+1])
                        c, _ := strconv.Atoi(s[index+1 : n])
                        if c == a-1 {
                                return true
                        }
                        if b == a-1 {
                                index = j
                                a = b
                                c, _ := strconv.Atoi(s[index+1 : n])
                                if c == a-1 {
                                        return true
                                }
                        } else if b > a-1 {
                                break
                        }
                }
        }
        return false
}
```

56.20 1850. 邻位交换的最小次数 (1)

- 题目

```
给你一个表示大整数的字符串 num ，和一个整数 k 。
如果某个整数是 num 中各位数字的一个 排列 且它的 值大于 num ，则称这个整数为 妙数 。
可能存在很多妙数，但是只需要关注 值最小 的那些。
例如，num = "5489355142" ：
第 1 个最小妙数是 "5489355214"
第 2 个最小妙数是 "5489355241"
第 3 个最小妙数是 "5489355412"
第 4 个最小妙数是 "5489355421"
返回要得到第 k 个 最小妙数 需要对 num 执行的 相邻位数字交换的最小次数 。
测试用例是按存在第 k 个最小妙数而生成的。
示例 1：输入：num = "5489355142", k = 4 输出：2
解释：第 4 个最小妙数是 "5489355421" ，要想得到这个数字：
- 交换下标 7 和下标 8 对应的位："5489355142" -> "5489355412"
- 交换下标 8 和下标 9 对应的位："5489355412" -> "5489355421"
示例 2：输入：num = "11112", k = 4 输出：4
解释：第 4 个最小妙数是 "21111" ，要想得到这个数字：
- 交换下标 3 和下标 4 对应的位："11112" -> "11121"
- 交换下标 2 和下标 3 对应的位："11121" -> "11211"
- 交换下标 1 和下标 2 对应的位："11211" -> "12111"
- 交换下标 0 和下标 1 对应的位："12111" -> "21111"
示例 3：输入：num = "00123", k = 1 输出：1
解释：第 1 个最小妙数是 "00132" ，要想得到这个数字：
- 交换下标 3 和下标 4 对应的位："00123" -> "00132"
提示：2 <= num.length <= 1000
1 <= k <= 1000
num 仅由数字组成
```

- 解题思路

```
func getMinSwaps(num string, k int) int {
        target := []byte(num)
        n := len(target)
        res := 0
        for i := 1; i <= k; i++ { // 求接下来的第k个排列
                nextPermutation(target)
        }
        // 统计交换次数
        arr := []byte(num)
        for i := 0; i < n; i++ {
                if arr[i] != target[i] {
```

(续下页)

(接上页)

```
                        for j := i + 1; j < n; j++ {
                                if arr[j] == target[i] { // 找到交换
                                        for k := j - 1; k >= i; k-- { //␣
↪把arr[j]交换到前面去
                                                arr[k], arr[k+1] = arr[k+1], arr[k]
                                                res++
                                        }
                                        break
                                }
                        }
                }
        }
        return res
}

// leetcode31.下一个排列
func nextPermutation(nums []byte) {
        n := len(nums)
        left := n - 2
        // 以12385764为例，从后往前找到5<7 的升序情况，目标值为左边的数5
        for left >= 0 && nums[left] >= nums[left+1] {
                left--
        }
        if left >= 0 { // 存在升序的情况
                right := n - 1
                // 从后往前，找到第一个大于目标值的数，如6>5，然后交换
                for right >= 0 && nums[right] <= nums[left] {
                        right--
                }
                nums[left], nums[right] = nums[right], nums[left]
        }
        reverse(nums, left+1, n-1)
}

func reverse(nums []byte, left, right int) {
        for left < right {
                nums[left], nums[right] = nums[right], nums[left]
                left++
                right--
        }
}
```

56.21 1855. 下标对中的最大距离 (3)

- 题目

```
给你两个 非递增 的整数数组 nums1和 nums2，数组下标均 从 0 开始 计数。
下标对 (i, j) 中 0 <= i < nums1.length 且 0 <= j < nums2.length 。
如果该下标对同时满足 i <= j 且 nums1[i] <= nums2[j] ，则称之为 有效␣
↪下标对，该下标对的 距离 为 j - i。
返回所有 有效 下标对 (i, j) 中的 最大距离 。如果不存在有效下标对，返回 0 。
一个数组 arr ，如果每个 1 <= i < arr.length 均有 arr[i-1] >= arr[i]␣
↪成立，那么该数组是一个 非递增 数组。
示例 1：输入：nums1 = [55,30,5,4,2], nums2 = [100,20,10,10,5] 输出：2
解释：有效下标对是 (0,0), (2,2), (2,3), (2,4), (3,3), (3,4) 和 (4,4) 。
最大距离是 2 ，对应下标对 (2,4) 。
示例 2：输入：nums1 = [2,2,2], nums2 = [10,10,1] 输出：1
解释：有效下标对是 (0,0), (0,1) 和 (1,1) 。
最大距离是 1 ，对应下标对 (0,1) 。
示例 3：输入：nums1 = [30,29,19,5], nums2 = [25,25,25,25,25] 输出：2
解释：有效下标对是 (2,2), (2,3), (2,4), (3,3) 和 (3,4) 。
最大距离是 2 ，对应下标对 (2,4) 。
示例 4：输入：nums1 = [5,4], nums2 = [3,2] 输出：0
解释：不存在有效下标对，所以返回 0 。
提示：1 <= nums1.length <= 105
1 <= nums2.length <= 105
1 <= nums1[i], nums2[j] <= 105
nums1 和 nums2 都是 非递增 数组
```

- 解题思路

```
func maxDistance(nums1 []int, nums2 []int) int {
        res := 0
        i := 0
        for j := 0; j < len(nums2); j++ {
                for i < len(nums1) && nums2[j] < nums1[i] {
                        i++
                }
                if i < len(nums1) {
                        if j-i > res {
                                res = j - i
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func maxDistance(nums1 []int, nums2 []int) int {
        res := 0
        for j := 0; j < len(nums2); j++ {
                n := min(j, len(nums1))
                i := sort.Search(n, func(i int) bool {
                        return nums1[i] <= nums2[j]
                })
                if i < n {
                        if j-i > res {
                                res = j - i
                        }
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func maxDistance(nums1 []int, nums2 []int) int {
        res := 0
        j := 0
        for i := 0; i < len(nums1); i++ {
                for j < len(nums2) && nums1[i] <= nums2[j] {
                        j++
                }
                if j-i-1 > res {
                        res = j - i - 1
                }
        }
        return res
}
```

56.22 1856. 子数组最小乘积的最大值 (2)

- 题目

```
一个数组的 最小乘积定义为这个数组中 最小值乘以数组的 和。
比方说，数组[3,2,5]（最小值是2）的最小乘积为2 * (3+2+5) = 2 * 10 = 20。
给你一个正整数数组nums，请你返回nums任意非空子数组的最小乘积的最大值。
由于答案可能很大，请你返回答案对109 + 7取余的结果。
请注意，最小乘积的最大值考虑的是取余操作 之前的结果。
题目保证最小乘积的最大值在 不取余 的情况下可以用 64 位有符号整数保存。
子数组定义为一个数组的 连续部分。
示例 1：输入：nums = [1,2,3,2] 输出：14
解释：最小乘积的最大值由子数组 [2,3,2] （最小值是 2）得到。
2 * (2+3+2) = 2 * 7 = 14 。
示例 2：输入：nums = [2,3,3,1,2] 输出：18
解释：最小乘积的最大值由子数组 [3,3] （最小值是 3）得到。
3 * (3+3) = 3 * 6 = 18 。
示例 3：输入：nums = [3,1,5,6,4,2] 输出：60
解释：最小乘积的最大值由子数组 [5,6,4] （最小值是 4）得到。
4 * (5+6+4) = 4 * 15 = 60 。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 107
```

- 解题思路

```
var mod = 1000000007

func maxSumMinProduct(nums []int) int {
        res := 0
        n := len(nums)
        arr := make([]int, n+1) // 前缀和
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        left := make([]int, n)  // 左侧最近的严格小于nums[i]的元素下标
        right := make([]int, n) // 右侧最近的小于等于nums[i]的元素下标
        for i := 0; i < n; i++ {
                left[i] = 0      // 默认是最左边
                right[i] = n - 1 // 默认是最右边
        }
        stack := make([]int, 0) // 单调递减栈
        for i := 0; i < n; i++ {
                for len(stack) > 0 && nums[stack[len(stack)-1]] >= nums[i] {
                        right[stack[len(stack)-1]] = i - 1
```

(续下页)

(接上页)

```
                        stack = stack[:len(stack)-1]
                }
                if len(stack) > 0 {
                        left[i] = stack[len(stack)-1] + 1
                }
                stack = append(stack, i)
        }
        for i := 0; i < n; i++ {
                target := (arr[right[i]+1] - arr[left[i]]) * nums[i]
                res = max(res, target)
        }
        return res % mod
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var mod = 1000000007

func maxSumMinProduct(nums []int) int {
        res := 0
        n := len(nums)
        arr := make([]int, n+1) // 前缀和
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        left, right := make([]int, n), make([]int, n)
        for i := 0; i < n; i++ {
                left[i] = -1 // 默认是最左边
                right[i] = n // 默认是最右边
        }
        stack := make([]int, 0) // 双栈：leetcode 84.柱状图中最大的矩形
        for i := 0; i < n; i++ {
                for len(stack) > 0 && nums[stack[len(stack)-1]] > nums[i] {
                        right[stack[len(stack)-1]] = i
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, i)
```

(续下页)

(接上页)

```
        }
        stack = make([]int, 0)
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 && nums[stack[len(stack)-1]] > nums[i] {
                        left[stack[len(stack)-1]] = i
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, i)
        }
        for i := 0; i < n; i++ {
                target := (arr[right[i]] - arr[left[i]+1]) * nums[i]
                res = max(res, target)
        }
        return res % mod
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

56.23 1860. 增长的内存泄露 (2)

- 题目

```
给你两个整数memory1␣
↪和memory2分别表示两个内存条剩余可用内存的位数。现在有一个程序每秒递增的速度消耗着内存。
在第i秒（秒数从 1 开始），有␣
↪i位内存被分配到剩余内存较多的内存条（如果两者一样多，则分配到第一个内存条）。
如果两者剩余内存都不足 i位，那么程序将 意外退出。
请你返回一个数组，包含 [crashTime, memory1crash, memory2crash],
其中crashTime是程序意外退出的时间（单位为秒），memory1crash␣
↪和memory2crash分别是两个内存条最后剩余内存的位数。
示例 1：输入：memory1 = 2, memory2 = 2 输出：[3,1,0]
解释：内存分配如下：
- 第 1 秒，内存条 1 被占用 1 位内存。内存条 1 现在有 1 位剩余可用内存。
- 第 2 秒，内存条 2 被占用 2 位内存。内存条 2 现在有 0 位剩余可用内存。
- 第 3 秒，程序意外退出，两个内存条分别有 1 位和 0 位剩余可用内存。
示例 2：输入：memory1 = 8, memory2 = 11 输出：[6,0,4]
解释：内存分配如下：
```

(续下页)

(接上页)

```
- 第 1 秒，内存条 2 被占用 1 位内存，内存条 2 现在有 10 位剩余可用内存。
- 第 2 秒，内存条 2 被占用 2 位内存，内存条 2 现在有 8 位剩余可用内存。
- 第 3 秒，内存条 1 被占用 3 位内存，内存条 1 现在有 5 位剩余可用内存。
- 第 4 秒，内存条 2 被占用 4 位内存，内存条 2 现在有 4 位剩余可用内存。
- 第 5 秒，内存条 1 被占用 5 位内存，内存条 1 现在有 0 位剩余可用内存。
- 第 6 秒，程序意外退出，两个内存条分别有 0 位和 4 位剩余可用内存。
提示：0 <= memory1, memory2 <= 231 - 1
```

- 解题思路

```
func memLeak(memory1 int, memory2 int) []int {
        i := 1
        for {
                if i <= memory1 || i <= memory2 {
                        if memory1 < memory2 {
                                memory2 = memory2 - i
                        } else {
                                memory1 = memory1 - i
                        }
                } else {
                        break
                }
                i++
        }
        return []int{i, memory1, memory2}
}

# 2
func memLeak(memory1 int, memory2 int) []int {
        i := 1
        for ; i <= memory1 || i <= memory2; i++ {
                if memory1 < memory2 {
                        memory2 = memory2 - i
                } else {
                        memory1 = memory1 - i
                }
        }
        return []int{i, memory1, memory2}
}
```

56.24 1861. 旋转盒子 (3)

- 题目

```
给你一个m x n的字符矩阵box，它表示一个箱子的侧视图。箱子的每一个格子可能为：
'#'表示石头
'*'表示固定的障碍物
'.'表示空位置
这个箱子被 顺时针旋转 90 度，由于重力原因，部分石头的位置会发生改变。
每个石头会垂直掉落，直到它遇到障碍物，另一个石头或者箱子的底部。
重力␣
↪不会影响障碍物的位置，同时箱子旋转不会产生惯性，也就是说石头的水平位置不会发生改变。
题目保证初始时box中的石头要么在一个障碍物上，要么在另一个石头上，要么在箱子的底部。
请你返回一个n x m的矩阵，表示按照上述旋转后，箱子内的结果。
示例 1：输入：box = [["#",".","#"]]
输出：[["."],
     ["#"],
     ["#"]]
示例 2：输入：box = [["#",".","*","."],
            ["#","#","*","."]]
输出：[["#","."],
     ["#","#"],
     ["*","*"],
     [".","."]]
示例 3：输入：box = [["#","#","*",".","*","."],
            ["#","#","#","*",".","."],
            ["#","#","#",".","#","."]]
输出：[[".","#","#"],
     [".","#","#"],
     ["#","#","*"],
     ["#","*","."],
     ["#",".","*"],
     ["#",".","."]]
提示：m == box.length
n == box[i].length
1 <= m, n <= 500
box[i][j]只可能是'#', '*'或者'.'。
```

- 解题思路

```
func rotateTheBox(box [][]byte) [][]byte {
        n, m := len(box), len(box[0])
        res := make([][]byte, m)
        for i := 0; i < m; i++ {
```

(续下页)

(接上页)

```
                res[i] = make([]byte, n)
                for j := 0; j < n; j++ {
                        res[i][j] = '.'
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        count := 0 // 统计一行中#的个数
                        for ; j < m && box[i][j] != '*'; j++ {
                                if box[i][j] == '#' {
                                        count++
                                }
                        }
                        if j < m {
                                res[j][n-1-i] = '*' // 填充*
                        }
                        // 移动# => 填充#
                        for k := j - 1; count > 0; k-- {
                                res[k][n-1-i] = '#'
                                count--
                        }
                }
        }
        return res
}

# 2
func rotateTheBox(box [][]byte) [][]byte {
        n, m := len(box), len(box[0])
        for i := 0; i < n; i++ {
                queue := make([]int, 0)
                for j := m - 1; j >= 0; j-- {
                        if box[i][j] == '*' {
                                queue = make([]int, 0) // 维护可移动的y坐标
                        } else if box[i][j] == '#' {
                                if len(queue) > 0 { // 石头落下
                                        first := queue[0]        // 最右边能落下的位置
                                        queue = queue[1:]        // 退出队列
                                        box[i][first] = '#'      // 落下
                                        box[i][j] = '.'          // 原位置为空
                                        queue = append(queue, j) // 置空后进入队列
                                }
                        } else {
```

(续下页)

(接上页)

```
                                queue = append(queue, j)
                        }
                }
        }
        res := make([][]byte, m)
        for i := 0; i < m; i++ {
                res[i] = make([]byte, n)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        res[j][n-1-i] = box[i][j]
                }
        }
        return res
}

# 3
func rotateTheBox(box [][]byte) [][]byte {
        n, m := len(box), len(box[0])
        for i := 0; i < n; i++ {
                last := m - 1 // 最后1个空位
                for j := m - 1; j >= 0; j-- {
                        if box[i][j] == '*' {
                                last = j - 1 // 有障碍，最后空位更新
                        } else if box[i][j] == '#' {
                                if last > j { // 可以移动
                                        box[i][last] = '#'
                                        box[i][j] = '.'
                                        last--
                                } else { // 当前位置不可以移动
                                        last = j - 1
                                }
                        }
                }
        }
        res := make([][]byte, m)
        for i := 0; i < m; i++ {
                res[i] = make([]byte, n)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        res[j][n-1-i] = box[i][j]
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

56.25 1864. 构成交替字符串需要的最小交换次数 (1)

- 题目

```
给你一个二进制字符串 s ，现需要将其转化为一个 交替字符串 。
请你计算并返回转化所需的 最小 字符交换次数，如果无法完成转化，返回 -1 。
交替字符串 是指：相邻字符之间不存在相等情况的字符串。例如，字符串 "010" 和 "1010"␣
↪属于交替字符串，但 "0100" 不是。
任意两个字符都可以进行交换，不必相邻 。
示例 1：输入：s = "111000" 输出：1
解释：交换位置 1 和 4："111000" -> "101010" ，字符串变为交替字符串。
示例 2：输入：s = "010" 输出：0
解释：字符串已经是交替字符串了，不需要交换。
示例 3：输入：s = "1110" 输出：-1
提示：1 <= s.length <= 1000
s[i] 的值为 '0' 或 '1'
```

- 解题思路

```
func minSwaps(s string) int {
        a, b := strings.Count(s, "1"), strings.Count(s, "0")
        if a-b > 1 || b-a > 1 {
                return -1
        }
        n := len(s)
        res := math.MaxInt32
        if a == (n+1)/2 && b == n/2 { // 以1开头：1010xxx
                count := 0
                for i := 0; i < n; i++ {
                        if int(s[i]-'0') == i%2 { // 0在偶数下标位置，1在奇数下标位置
                                count++
                        }
                }
                res = min(res, count/2)
        }
        if b == (n+1)/2 && a == n/2 { // 以0开头：0101xxx
                count := 0
                for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                        if int(s[i]-'0') != i%2 { // 1在偶数下标位置，0在奇数下标位置
                                count++
                        }
                }
                res = min(res, count/2)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

56.26 1865. 找出和为指定值的下标对 (1)

- 题目

```
给你两个整数数组 nums1 和 nums2 ，请你实现一个支持下述两类查询的数据结构：
累加 ，将一个正整数加到 nums2 中指定下标对应元素上。
计数 ，统计满足 nums1[i] + nums2[j] 等于指定值的下标对 (i, j) 数目
（0 <= i < nums1.length 且 0 <= j < nums2.length）。
实现 FindSumPairs 类：
FindSumPairs(int[] nums1, int[] nums2) 使用整数数组nums1 和 nums2 初始化 FindSumPairs␣
↪对象。
void add(int index, int val) 将 val 加到 nums2[index] 上，即，执行 nums2[index] +=␣
↪val 。
int count(int tot) 返回满足nums1[i] + nums2[j] == tot 的下标对 (i, j) 数目。
示例：输入：
["FindSumPairs", "count", "add", "count", "count", "add", "add", "count"]
[[[1, 1, 2, 2, 2, 3], [1, 4, 5, 2, 5, 4]], [7], [3, 2], [8], [4], [0, 1], [1, 1], [7]]
输出：[null, 8, null, 2, 1, null, null, 11]
解释：FindSumPairs findSumPairs = new FindSumPairs([1, 1, 2, 2, 2, 3], [1, 4, 5, 2, 5,
↪ 4]);
findSumPairs.count(7);  // 返回 8 ；下标对 (2,2), (3,2), (4,2), (2,4), (3,4), (4,4)␣
↪满足 2 + 5 = 7 ，
下标对 (5,1), (5,5) 满足 3 + 4 = 7
findSumPairs.add(3, 2); // 此时 nums2 = [1,4,5,4,5,4]
findSumPairs.count(8);  // 返回 2 ；下标对 (5,2), (5,4) 满足 3 + 5 = 8
findSumPairs.count(4);  // 返回 1 ；下标对 (5,0) 满足 3 + 1 = 4
```

(续下页)

(接上页)

```
findSumPairs.add(0, 1); // 此时 nums2 = [2,4,5,4,5,4]
findSumPairs.add(1, 1); // 此时 nums2 = [2,5,5,4,5,4]
findSumPairs.count(7);  // 返回 11 ; 下标对 (2,1), (2,2), (2,4), (3,1), (3,2), (3,4),
↪(4,1), (4,2), (4,4)
满足 2 + 5 = 7 , 下标对 (5,3), (5,5) 满足 3 + 4 = 7
提示: 1 <= nums1.length <= 1000
1 <= nums2.length <= 105
1 <= nums1[i] <= 109
1 <= nums2[i] <= 105
0 <= index < nums2.length
1 <= val <= 105
1 <= tot <= 109
最多调用add 和 count 函数各 1000 次
```

- 解题思路

```
type FindSumPairs struct {
        nums1, nums2 []int
        m            map[int]int
}

func Constructor(nums1 []int, nums2 []int) FindSumPairs {
        m := make(map[int]int)
        for i := 0; i < len(nums2); i++ {
                m[nums2[i]]++
        }
        return FindSumPairs{
                nums1: nums1,
                nums2: nums2,
                m:     m,
        }
}

func (this *FindSumPairs) Add(index int, val int) {
        this.m[this.nums2[index]]--
        this.nums2[index] = this.nums2[index] + val
        this.m[this.nums2[index]]++
}

func (this *FindSumPairs) Count(tot int) int {
        res := 0
        for i := 0; i < len(this.nums1); i++ {
                res = res + this.m[tot-this.nums1[i]]
        }
```

(续下页)

(接上页)

```
        return res
}
```

56.27 1870. 准时到达的列车最小时速 (1)

- 题目

```
给你一个浮点数 hour␣
↪，表示你到达办公室可用的总通勤时间。要到达办公室，你必须按给定次序乘坐 n 趟列车。
另给你一个长度为 n 的整数数组 dist ，其中 dist[i] 表示第 i␣
↪趟列车的行驶距离（单位是千米）。
每趟列车均只能在整点发车，所以你可能需要在两趟列车之间等待一段时间。
例如，第 1 趟列车需要 1.5 小时，那你必须再等待 0.5 小时，搭乘在第 2 小时发车的第 2␣
↪趟列车。
返回能满足你准时到达办公室所要求全部列车的 最小正整数␣
↪时速（单位：千米每小时），如果无法准时到达，则返回 -1 。
生成的测试用例保证答案不超过 107 ，且 hour 的 小数点后最多存在两位数字 。
示例 1：输入：dist = [1,3,2], hour = 6 输出：1
解释：速度为 1 时：
- 第 1 趟列车运行需要 1/1 = 1 小时。
- 由于是在整数时间到达，可以立即换乘在第 1 小时发车的列车。第 2 趟列车运行需要 3/1 =␣
↪3 小时。
- 由于是在整数时间到达，可以立即换乘在第 4 小时发车的列车。第 3 趟列车运行需要 2/1 =␣
↪2 小时。
- 你将会恰好在第 6 小时到达。
示例 2：输入：dist = [1,3,2], hour = 2.7 输出：3
解释：速度为 3 时：
- 第 1 趟列车运行需要 1/3 = 0.33333 小时。
- 由于不是在整数时间到达，故需要等待至第 1 小时才能搭乘列车。第 2 趟列车运行需要 3/3␣
↪= 1 小时。
- 由于是在整数时间到达，可以立即换乘在第 2 小时发车的列车。第 3 趟列车运行需要 2/3 =␣
↪0.66667 小时。
- 你将会在第 2.66667 小时到达。
示例 3：输入：dist = [1,3,2], hour = 1.9 输出：-1
解释：不可能准时到达，因为第 3 趟列车最早是在第 2 小时发车。
提示：n == dist.length
1 <= n <= 105
1 <= dist[i] <= 105
1 <= hour <= 109
hours 中，小数点后最多存在两位数字
```

- 解题思路

```
func minSpeedOnTime(dist []int, hour float64) int {
        n := len(dist)
        if float64(n-1) > hour {
                return -1
        }
        left, right := 1, math.MaxInt32
        for left <= right {
                mid := left + (right-left)/2
                if judge(dist, float64(mid)) <= hour {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return left
}

func judge(arr []int, speed float64) float64 {
        n := len(arr)
        res := float64(0)
        for i := 0; i < n-1; i++ {
                res = res + math.Ceil(float64(arr[i])/speed) // 向上取整
        }
        res = res + float64(arr[n-1])/speed // 注意：最后一个不需要等待
        return res
}
```

56.28 1871. 跳跃游戏 VII(4)

- 题目

```
给你一个下标从 0 开始的二进制字符串s和两个整数minJump 和maxJump。
一开始，你在下标0处，且该位置的值一定为'0'。
当同时满足如下条件时，你可以从下标i移动到下标j处：
i + minJump <= j <= min(i + maxJump, s.length - 1)且 s[j] == '0'.
如果你可以到达 s的下标s.length - 1处，请你返回true，否则返回false。
示例 1：输入：s = "011010", minJump = 2, maxJump = 3 输出：true
解释：
第一步，从下标 0 移动到下标 3 。
第二步，从下标 3 移动到下标 5 。
示例 2：输入：s = "01101110", minJump = 2, maxJump = 3 输出：false
提示：2 <= s.length <= 105
```

(续下页)

(接上页)

```
s[i]要么是'0', 要么是'1'
s[0] == '0'
1 <= minJump <= maxJump < s.length
```

- 解题思路

```
func canReach(s string, minJump int, maxJump int) bool {
        n := len(s)
        if s[n-1] == '1' {
                return false
        }
        minDis, maxDis := 0, 0 // 更新最后可达到的最小+最大坐标的范围
        for i := 0; i < n-1; i++ {
                if s[i] == '0' && minDis <= i && i <= maxDis {
                        minDis = i + minJump
                        maxDis = i + maxJump
                }
                if minDis <= n-1 && n-1 <= maxDis {
                        return true
                }
        }
        return false
}

# 2
func canReach(s string, minJump int, maxJump int) bool {
        n := len(s)
        if s[n-1] == '1' {
                return false
        }
        dp := make([]bool, n) // dp[i] => 下标i是否可达
        dp[0] = true
        count := 1 // 滑动窗口里面可达数量
        for i := minJump; i < n; i++ {
                if s[i] == '0' && count > 0 { // 当前可达
                        dp[i] = true
                }
                if maxJump <= i && dp[i-maxJump] == true { // 滑动窗口左端点:-1
                        count--
                }
                if dp[i-minJump+1] == true { // 滑动窗口右端点: +1
                        count++
                }
        }
```

(续下页)

(接上页)

```
        return dp[n-1]
}

# 3
func canReach(s string, minJump int, maxJump int) bool {
        n := len(s)
        if s[n-1] == '1' {
                return false
        }
        dp := make([]int, n) // dp[i] => 下标i是否可达
        dp[0] = 1
        arr := make([]int, n)
        for i := 0; i < minJump; i++ { // 前缀和从minJump开始
                arr[i] = 1
        }
        for i := minJump; i < n; i++ {
                left := i - maxJump
                right := i - minJump
                if s[i] == '0' {
                        var total int
                        if left <= 0 {
                                total = arr[right]
                        } else {
                                total = arr[right] - arr[left-1]
                        }
                        if total > 0 { // 通过前缀和计算，范围内存在多少满足条件的坐标
                                dp[i] = 1
                        }
                }
                arr[i] = arr[i-1] + dp[i]
        }
        return dp[n-1] > 0
}

# 4
func canReach(s string, minJump int, maxJump int) bool {
        n := len(s)
        if s[n-1] == '1' {
                return false
        }
        arr := make([]int, n+1)
        arr[0] = 1
        arr[1] = -1
```

(续下页)

(接上页)

```
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + arr[i]
                if s[i] == '0' && sum > 0 {
                        arr[min(i+minJump, n)]++
                        arr[min(i+maxJump+1, n)]--
                }
        }
        return sum > 0 // 计算范围内可达到最后有几个坐标满足条件
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

56.29 1877. 数组中最大数对和的最小值 (1)

- 题目

```
一个数对(a,b)的 数对和等于a + b。最大数对和是一个数对数组中最大的数对和。
比方说，如果我们有数对(1,5)，(2,3)和(4,4)，最大数对和为max(1+5, 2+3, 4+4) = max(6, 5,␣
↪8) = 8。
给你一个长度为 偶数n的数组nums，请你将 nums中的元素分成 n / 2个数对，使得：
nums中每个元素恰好在 一个数对中，且
最大数对和的值 最小。
请你在最优数对划分的方案下，返回最小的 最大数对和。
示例 1：输入：nums = [3,5,2,3] 输出：7
解释：数组中的元素可以分为数对 (3,3) 和 (5,2) 。
最大数对和为 max(3+3, 5+2) = max(6, 7) = 7 。
示例 2：输入：nums = [3,5,4,2,4,6] 输出：8
解释：数组中的元素可以分为数对 (3,5)，(4,4) 和 (6,2) 。
最大数对和为 max(3+5, 4+4, 6+2) = max(8, 8, 8) = 8 。
提示：n == nums.length
2 <= n <= 105
n是 偶数。
1 <= nums[i] <= 105
```

- 解题思路

```
func minPairSum(nums []int) int {
        n := len(nums)
        sort.Ints(nums)
        res := 0
        for i := 0; i < n/2; i++ {
                res = max(res, nums[i]+nums[n-1-i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

56.30 1878. 矩阵中最大的三个菱形和 (2)

- 题目

```
给你一个m x n的整数矩阵grid。
菱形和 指的是 grid中一个正菱形 边界上的元素之和。
本题中的菱形必须为正方形旋转45度，且四个角都在一个格子当中。
下图是四个可行的菱形，每个菱形和应该包含的格子都用了相应颜色标注在图中。
注意，菱形可以是一个面积为 0 的区域，如上图中右下角的紫色菱形所示。
请你按照 降序返回␣
↪grid中三个最大的互不相同的菱形和。如果不同的和少于三个，则将它们全部返回。
示例 1: 输入: grid = [[3,4,5,1,3],[3,3,4,2,3],[20,30,200,40,10],[1,5,5,4,1],[4,3,2,2,
↪5]] 输出: [228,216,211]
解释: 最大的三个菱形和如上图所示。
- 蓝色: 20 + 3 + 200 + 5 = 228
- 红色: 200 + 2 + 10 + 4 = 216
- 绿色: 5 + 200 + 4 + 2 = 211
示例 2: 输入: grid = [[1,2,3],[4,5,6],[7,8,9]] 输出: [20,9,8]
解释: 最大的三个菱形和如上图所示。
- 蓝色: 4 + 2 + 6 + 8 = 20
- 红色: 9 (右下角红色的面积为 0 的菱形)
- 绿色: 8 (下方中央面积为 0 的菱形)
示例 3: 输入: grid = [[7,7,7]] 输出: [7]
解释: 所有三个可能的菱形和都相同，所以返回 [7] 。
提示: m == grid.length
```

(续下页)

(接上页)

```
n == grid[i].length
1 <= m, n <= 100
1 <= grid[i][j] <= 105
```

- 解题思路

```
func getBiggestThree(grid [][]int) []int {
        arr := make([]int, 0)
        n, m := len(grid), len(grid[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr = append(arr, grid[i][j]) // 单独一个
                        for k := 1; k < n; k++ {      // 枚举增加的长度
                                left := i - k
                                right := i + k
                                button := j + 2*k
                                mid := j + k
                                if left < 0 || right > n-1 || button > m-1 {
                                        break
                                }
                                sum := 0
                                b := mid
                                for a := left; a < i; a++ { // 左到上
                                        sum = sum + grid[a][b]
                                        b--
                                }
                                for a := i; a < right; a++ { // 上到右
                                        sum = sum + grid[a][b]
                                        b++
                                }
                                for a := right; a > i; a-- { // 右到下
                                        sum = sum + grid[a][b]
                                        b++
                                }
                                for a := i; a > left; a-- { // 下到左
                                        sum = sum + grid[a][b]
                                        b--
                                }
                                arr = append(arr, sum)
                        }
                }
        }
        sort.Ints(arr)
        res := make([]int, 0)
```

(续下页)

(接上页)

```
        res = append(res, arr[len(arr)-1])
        for i := len(arr) - 2; i >= 0; i-- {
                if arr[i] != arr[i+1] && len(res) < 3 {
                        res = append(res, arr[i])
                }
        }
        return res
}

# 2
func getBiggestThree(grid [][]int) []int {
        arr := make([]int, 0)
        n, m := len(grid), len(grid[0])
        sum1, sum2 := make([][]int, n+2), make([][]int, n+2)
        for i := 0; i <= n+1; i++ {
                sum1[i], sum2[i] = make([]int, m+2), make([]int, m+2)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        sum1[i][j] = sum1[i-1][j-1] + grid[i-1][j-1] // 下到右的斜线
                        sum2[i][j] = sum2[i-1][j+1] + grid[i-1][j-1] // 下到左的斜线
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr = append(arr, grid[i][j])       // 单独一个
                        for k := i + 2; k < n; k = k + 2 { // 上下长度
                                aX, aY := i, j                // 上顶点
                                bX, bY := k, j                // 下顶点
                                cX, cY := (i+k)/2, j-(k-i)/2 // 左顶点
                                dX, dY := (i+k)/2, j+(k-i)/2 // 右顶点
                                if cY < 0 || dY >= m {
                                        break
                                }
                                sum := (sum2[cX+1][cY+1] - sum2[aX+1-1][aY+1+1]) + //␣
↪左到上
                                        (sum2[bX+1][bY+1] - sum2[dX+1-1][dY+1+1]) + //
↪ 下到右
                                        (sum1[bX+1][bY+1] - sum1[cX+1-1][cY+1-1]) + //
↪ 下到左
                                        (sum1[dX+1][dY+1] - sum1[aX+1-1][aY+1-1]) - //
↪ 右到上
                                        (grid[aX][aY] + grid[bX][bY] + grid[cX][cY] +␣
```

(续下页)

(接上页)

```
↪grid[dX][dY])
                                arr = append(arr, sum)
                        }
                }
        }
        sort.Ints(arr)
        res := make([]int, 0)
        res = append(res, arr[len(arr)-1])
        for i := len(arr) - 2; i >= 0; i-- {
                if arr[i] != arr[i+1] && len(res) < 3 {
                        res = append(res, arr[i])
                }
        }
        return res
}
```

56.31 1881. 插入后的最大值 (1)

- 题目

```
给你一个非常大的整数 n 和一个整数数字 x ，大整数 n用一个字符串表示。
n 中每一位数字和数字 x 都处于闭区间 [1, 9] 中，且 n 可能表示一个 负数 。
你打算通过在 n 的十进制表示的任意位置插入 x 来 最大化 n 的 数值。但 不能␣
↪在负号的左边插入 x 。
例如，如果 n = 73 且 x = 6 ，那么最佳方案是将 6 插入 7 和 3 之间，使 n = 763 。
如果 n = -55 且 x = 2 ，那么最佳方案是将 2 插在第一个 5 之前，使 n = -255 。
返回插入操作后，用字符串表示的n 的最大值。
示例 1: 输入: n = "99", x = 9 输出: "999"
解释: 不管在哪里插入 9 ，结果都是相同的。
示例 2: 输入: n = "-13", x = 2 输出: "-123"
解释: 向 n 中插入 x 可以得到 -213、-123 或者 -132 ，三者中最大的是 -123 。
提示: 1 <= n.length <= 105
1 <= x <= 9
n中每一位的数字都在闭区间 [1, 9] 中。
n代表一个有效的整数。
当 n 表示负数时，将会以字符 '-' 开始。
```

- 解题思路

```
func maxValue(n string, x int) string {
        if strings.Contains(n, "-") {
                for i := 1; i < len(n); i++ {
```

(续下页)

(接上页)

```
                        if x < int(n[i]-'0') {
                                return n[:i] + string(x+'0') + n[i:]
                        }
                }
        } else {
                for i := 0; i < len(n); i++ {
                        if x > int(n[i]-'0') {
                                return n[:i] + string(x+'0') + n[i:]
                        }
                }
        }
        return n + string(x+'0')
}
```

56.32 1882. 使用服务器处理任务 (1)

- 题目

```
给你两个 下标从 0 开始 的整数数组 servers 和 tasks ，长度分别为 n和 m。
servers[i] 是第i台服务器的 权重 ，而 tasks[j] 是处理第j项任务␣
↪所需要的时间（单位：秒）。
你正在运行一个仿真系统，在处理完所有任务后，该系统将会关闭。每台服务器只能同时处理一项任务。
第 0 项任务在第 0 秒可以开始处理，相应地，第 j 项任务在第 j秒可以开始处理。
处理第 j 项任务时，你需要为它分配一台 权重最小 的空闲服务器。
如果存在多台相同权重的空闲服务器，请选择 下标最小 的服务器。
如果一台空闲服务器在第 t 秒分配到第 j 项任务，那么在 t + tasks[j] 时它将恢复空闲状态。
如果没有空闲服务器，则必须等待，直到出现一台空闲服务器，并 尽可能早地处理剩余任务。
如果有多项任务等待分配，则按照 下标递增 的顺序完成分配。
如果同一时刻存在多台空闲服务器，可以同时将多项任务分别分配给它们。
构建长度为m 的答案数组 ans ，其中 ans[j] 是第 j 项任务分配的服务器的下标。
返回答案数组 ans。
示例 1：输入：servers = [3,3,2], tasks = [1,2,3,2,1,2] 输出：[2,2,0,2,1,2]
解释：事件按时间顺序如下：
- 0 秒时，第 0 项任务加入到任务队列，使用第 2 台服务器处理到 1 秒。
- 1 秒时，第 2 台服务器空闲，第 1 项任务加入到任务队列，使用第 2 台服务器处理到 3 秒。
- 2 秒时，第 2 项任务加入到任务队列，使用第 0 台服务器处理到 5 秒。
- 3 秒时，第 2 台服务器空闲，第 3 项任务加入到任务队列，使用第 2 台服务器处理到 5 秒。
- 4 秒时，第 4 项任务加入到任务队列，使用第 1 台服务器处理到 5 秒。
- 5 秒时，所有服务器都空闲，第 5 项任务加入到任务队列，使用第 2 台服务器处理到 7 秒。
示例 2：输入：servers = [5,1,4,3,2], tasks = [2,1,2,4,5,2,1] 输出：[1,4,1,4,1,3,2]
解释：事件按时间顺序如下：
- 0 秒时，第 0 项任务加入到任务队列，使用第 1 台服务器处理到 2 秒。
```

(续下页)

(接上页)

```
- 1 秒时，第 1 项任务加入到任务队列，使用第 4 台服务器处理到 2 秒。
- 2 秒时，第 1 台和第 4 台服务器空闲，第 2 项任务加入到任务队列，使用第 1␣
↪台服务器处理到 4 秒。
- 3 秒时，第 3 项任务加入到任务队列，使用第 4 台服务器处理到 7 秒。
- 4 秒时，第 1 台服务器空闲，第 4 项任务加入到任务队列，使用第 1 台服务器处理到 9 秒。
- 5 秒时，第 5 项任务加入到任务队列，使用第 3 台服务器处理到 7 秒。
- 6 秒时，第 6 项任务加入到任务队列，使用第 2 台服务器处理到 7 秒。
提示：servers.length == n
tasks.length == m
1 <= n, m <= 2 * 105
1 <= servers[i], tasks[j] <= 2 * 105
```

- 解题思路

```go
func assignTasks(servers []int, tasks []int) []int {
        res := make([]int, 0)
        n := len(servers)
        waitHeap := make(WaitHeap, 0)
        heap.Init(&waitHeap)
        runHeap := make(RunHeap, 0)
        heap.Init(&runHeap)
        for i := 0; i < n; i++ {
                heap.Push(&waitHeap, Node{
                        Id:   i,
                        Rank: servers[i],
                })
        }
        curTime := 0
        for i := 0; i < len(tasks); i++ {
                curTime = max(curTime, i)
                if waitHeap.Len() == 0 { // 无服务器可用，时间移动
                        endTime := runHeap[0].EndTime // 最小的结束时间的出来
                        for runHeap.Len() > 0 && runHeap[0].EndTime == endTime {
                                node := heap.Pop(&runHeap).(Node)
                                heap.Push(&waitHeap, node)
                        }
                        curTime = max(curTime, endTime)
                } else {
                        for runHeap.Len() > 0 && runHeap[0].EndTime <= curTime {
                                node := heap.Pop(&runHeap).(Node)
                                heap.Push(&waitHeap, node)
                        }
                }
                node := heap.Pop(&waitHeap).(Node)
```

(续下页)

(接上页)

```
                res = append(res, node.Id)
                heap.Push(&runHeap, Node{
                        Id:      node.Id,
                        Rank:    node.Rank,
                        EndTime: curTime + tasks[i],
                })
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type Node struct {
        Id      int
        Rank    int // 权重
        EndTime int
}

// 运行堆
type RunHeap []Node

func (h RunHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h RunHeap) Less(i, j int) bool {
        return h[i].EndTime < h[j].EndTime // 按照时间排序
}

func (h RunHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *RunHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}
```

(续下页)

(接上页)

```
func (h *RunHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

// 等待堆
type WaitHeap []Node

func (h WaitHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h WaitHeap) Less(i, j int) bool {
        if h[i].Rank == h[j].Rank {
                return h[i].Id < h[j].Id
        }
        return h[i].Rank < h[j].Rank
}

func (h WaitHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *WaitHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *WaitHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

56.33 1884. 鸡蛋掉落-两枚鸡蛋 (2)

- 题目

```
给你 2枚相同 的鸡蛋，和一栋从第 1层到第 n 层共有 n 层楼的建筑。
已知存在楼层 f ，满足0 <= f <= n ，任何从 高于 f 的楼层落下的鸡蛋都 会碎 ，
从 f 楼层或比它低 的楼层落下的鸡蛋都 不会碎 。
每次操作，你可以取一枚 没有碎 的鸡蛋并把它从任一楼层 x 扔下（满足1 <= x <= n）。
如果鸡蛋碎了，你就不能再次使用它。如果某枚鸡蛋扔下后没有摔碎，则可以在之后的操作中␣
↪重复使用 这枚鸡蛋。
请你计算并返回要确定 f 确切的值 的 最小操作次数 是多少？
示例 1：输入：n = 2 输出：2
解释：我们可以将第一枚鸡蛋从 1 楼扔下，然后将第二枚从 2 楼扔下。
如果第一枚鸡蛋碎了，可知 f = 0；
如果第二枚鸡蛋碎了，但第一枚没碎，可知 f = 1；
否则，当两个鸡蛋都没碎时，可知 f = 2。
示例 2：输入：n = 100 输出：14
解释：一种最优的策略是：
- 将第一枚鸡蛋从 9 楼扔下。如果碎了，那么 f 在 0 和 8 之间。
将第二枚从 1 楼扔下，然后每扔一次上一层楼，在 8 次内找到 f 。总操作次数 = 1 + 8 = 9 。
- 如果第一枚鸡蛋没有碎，那么再把第一枚鸡蛋从 22 层扔下。如果碎了，那么 f 在 9 和 21␣
↪之间。
将第二枚鸡蛋从 10 楼扔下，然后每扔一次上一层楼，在 12 次内找到 f 。总操作次数 = 2 +␣
↪12 = 14 。
- 如果第一枚鸡蛋没有再次碎掉，则按照类似的方法从 34, 45, 55, 64, 72, 79, 85, 90, 94,␣
↪97,
99 和 100 楼分别扔下第一枚鸡蛋。
不管结果如何，最多需要扔 14 次来确定 f 。
提示：1 <= n <= 1000
```

- 解题思路

```
func twoEggDrop(n int) int {
        dp := [3][]int{} // dp[i][j] 有i枚鸡蛋，验证j层楼需要的最少操作次数
        for i := 0; i < 3; i++ {
                dp[i] = make([]int, n+1)
                for j := 1; j < n+1; j++ {
                        dp[i][j] = math.MaxInt32
                        if i == 1 {
                                dp[1][j] = j // 1个鸡蛋，需要j次
                        }
                }
        }
        // 值为0
```

(续下页)

(接上页)

```
        // dp[1][0], dp[2][0] = 0,0
        for j := 1; j <= n; j++ {
                for k := 1; k <= j; k++ { //
                        // a := k               //  假设在k层破碎：剩下1枚鸡蛋，求k-
↪1层结果，(k-1)+1
                        a := dp[1][k-1] + 1 //  假设在k层破碎。剩下1枚鸡蛋 求k-1层结果
                        b := dp[2][j-k] + 1 //  假设在k层没有破碎：剩余2枚鸡蛋求j-
↪k层结果
                        dp[2][j] = min(dp[2][j], max(a, b))
                }
        }
        return dp[2][n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func twoEggDrop(n int) int {
        dp := make([]int, n+1) // dp[j]验证j层楼需要的最少操作次数
        for j := 1; j < n+1; j++ {
                dp[j] = math.MaxInt32
        }
        // 值为0
        // dp[0] = 0
        for j := 1; j <= n; j++ {
                for k := 1; k <= j; k++ {
                        a := k           //  假设在k层破碎：剩下1枚鸡蛋，求k-
↪1层结果，(k-1)+1
                        b := dp[j-k] + 1 //  假设在k层没有破碎：剩余2枚鸡蛋求j-k层结果
                        dp[j] = min(dp[j], max(a, b))
                }
```

(续下页)

(接上页)

```
        }
        return dp[n]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

56.34 1887. 使数组元素相等的减少操作次数 (2)

- 题目

```
给你一个整数数组 nums ，你的目标是令 nums␣
↪中的所有元素相等。完成一次减少操作需要遵照下面的几个步骤：
找出 nums 中的 最大 值。记这个值为 largest 并取其下标 i （下标从 0 开始计数）。
如果有多个元素都是最大值，则取最小的 i 。
找出 nums 中的 下一个最大 值，这个值 严格小于 largest ，记为 nextLargest 。
将 nums[i] 减少到 nextLargest 。
返回使 nums 中的所有元素相等的操作次数。
示例 1：输入：nums = [5,1,3] 输出：3
解释：需要 3 次操作使 nums 中的所有元素相等：
1. largest = 5 下标为 0 。nextLargest = 3 。将 nums[0] 减少到 3 。nums = [3,1,3] 。
2. largest = 3 下标为 0 。nextLargest = 1 。将 nums[0] 减少到 1 。nums = [1,1,3] 。
3. largest = 3 下标为 2 。nextLargest = 1 。将 nums[2] 减少到 1 。nums = [1,1,1] 。
示例 2：输入：nums = [1,1,1] 输出：0
解释：nums 中的所有元素已经是相等的。
示例 3：输入：nums = [1,1,2,2,3] 输出：4
解释：需要 4 次操作使 nums 中的所有元素相等：
1. largest = 3 下标为 4 。nextLargest = 2 。将 nums[4] 减少到 2 。nums = [1,1,2,2,2]␣
↪。
2. largest = 2 下标为 2 。nextLargest = 1 。将 nums[2] 减少到 1 。nums = [1,1,1,2,2]␣
↪。
```

(续下页)

(接上页)

```
3. largest = 2 下标为 3 。nextLargest = 1 。将 nums[3] 减少到 1 。nums = [1,1,1,1,2]␣
↪。
4. largest = 2 下标为 4 。nextLargest = 1 。将 nums[4] 减少到 1 。nums = [1,1,1,1,1]␣
↪。
提示：1 <= nums.length <= 5 * 104
1 <= nums[i] <= 5 * 104
```

- 解题思路

```
func reductionOperations(nums []int) int {
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] > nums[j]
        })
        res := 0
        for i := 1; i < len(nums); i++ {
                if nums[i] != nums[i-1] {
                        res = res + i
                }
        }
        return res
}

# 2

func reductionOperations(nums []int) int {
        sort.Ints(nums)
        res := 0
        count := 0
        for i := 1; i < len(nums); i++ {
                if nums[i] != nums[i-1] {
                        count++
                }
                res = res + count
        }
        return res
}
```

56.35 1888. 使二进制字符串字符交替的最少反转次数

56.35.1 题目

56.35.2 解题思路

56.36 1894. 找到需要补充粉笔的学生编号 (1)

- 题目

```
一个班级里有n个学生，编号为 0到 n - 1。每个学生会依次回答问题，编号为␣
↪0的学生先回答，然后是编号为 1的学生，
以此类推，直到编号为 n - 1的学生，然后老师会重复这个过程，重新从编号为␣
↪0的学生开始回答问题。
给你一个长度为 n且下标从␣
↪0开始的整数数组chalk和一个整数k。一开始粉笔盒里总共有k支粉笔。
当编号为i的学生回答问题时，他会消耗 chalk[i]支粉笔。如果剩余粉笔数量␣
↪严格小于chalk[i]，那么学生 i需要 补充粉笔。
请你返回需要 补充粉笔的学生 编号。
示例 1：输入：chalk = [5,1,5], k = 22
输出：0
解释：学生消耗粉笔情况如下：
- 编号为 0 的学生使用 5 支粉笔，然后 k = 17 。
- 编号为 1 的学生使用 1 支粉笔，然后 k = 16 。
- 编号为 2 的学生使用 5 支粉笔，然后 k = 11 。
- 编号为 0 的学生使用 5 支粉笔，然后 k = 6 。
- 编号为 1 的学生使用 1 支粉笔，然后 k = 5 。
- 编号为 2 的学生使用 5 支粉笔，然后 k = 0 。
编号为 0 的学生没有足够的粉笔，所以他需要补充粉笔。
示例 2：输入：chalk = [3,4,1,2], k = 25 输出：1
解释：学生消耗粉笔情况如下：
- 编号为 0 的学生使用 3 支粉笔，然后 k = 22 。
- 编号为 1 的学生使用 4 支粉笔，然后 k = 18 。
- 编号为 2 的学生使用 1 支粉笔，然后 k = 17 。
- 编号为 3 的学生使用 2 支粉笔，然后 k = 15 。
- 编号为 0 的学生使用 3 支粉笔，然后 k = 12 。
```

(续下页)

(接上页)

```
- 编号为 1 的学生使用 4 支粉笔，然后 k = 8 。
- 编号为 2 的学生使用 1 支粉笔，然后 k = 7 。
- 编号为 3 的学生使用 2 支粉笔，然后 k = 5 。
- 编号为 0 的学生使用 3 支粉笔，然后 k = 2 。
编号为 1 的学生没有足够的粉笔，所以他需要补充粉笔。
提示：chalk.length == n
1 <= n <= 105
1 <= chalk[i] <= 105
1 <= k <= 109
```

- 解题思路

```
func chalkReplacer(chalk []int, k int) int {
        res := 0
        n := len(chalk)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + chalk[i]
        }
        k = k % sum // 求余
        for i := 0; i < n; i++ {
                if k < chalk[i] {
                        return i
                }
                k = k - chalk[i]
        }
        return res
}
```

56.37 1895. 最大的幻方 (2)

- 题目

```
一个k x k的幻方指的是一个k x k填满整数的方格阵，且每一行、每一列以及两条对角线的和
↪全部相等。
幻方中的整数 不需要互不相同。显然，每个1 x 1的方格都是一个幻方。
给你一个m x n的整数矩阵grid，请你返回矩阵中最大幻方的尺寸（即边长 k）。
示例 1：输入：grid = [[7,1,4,5,6],[2,5,1,6,4],[1,5,4,3,2],[1,2,7,3,4]] 输出：3
解释：最大幻方尺寸为 3 。
每一行，每一列以及两条对角线的和都等于 12 。
- 每一行的和：5+1+6 = 5+4+3 = 2+7+3 = 12
- 每一列的和：5+5+2 = 1+4+7 = 6+3+3 = 12
```

(续下页)

(接上页)

```
- 对角线的和：5+4+3 = 6+4+2 = 12
示例 2：输入：grid = [[5,1,3,1],[9,3,3,1],[1,3,3,8]] 输出：2
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 50
1 <= grid[i][j] <= 106
```

- 解题思路

```
func largestMagicSquare(grid [][]int) int {
        n, m := len(grid), len(grid[0])
        rowArr := make([][]int, n) // 行前缀和
        colArr := make([][]int, n) // 列前缀和
        for i := 0; i < n; i++ {
                rowArr[i] = make([]int, m)
                colArr[i] = make([]int, m)
        }
        for i := 0; i < n; i++ {
                rowArr[i][0] = grid[i][0]
                for j := 1; j < m; j++ {
                        rowArr[i][j] = rowArr[i][j-1] + grid[i][j]
                }
        }
        for j := 0; j < m; j++ {
                colArr[0][j] = grid[0][j]
                for i := 1; i < n; i++ {
                        colArr[i][j] = colArr[i-1][j] + grid[i][j]
                }
        }
        for length := min(n, m); length >= 2; length-- { // 枚举边长
                for i := 0; i+length <= n; i++ {
                        for j := 0; j+length <= m; j++ {
                                var target int // 以行目标和
                                if j == 0 {
                                        target = rowArr[i][j+length-1]
                                } else {
                                        target = rowArr[i][j+length-1] - rowArr[i][j-
↪1]

                                }
                                flag := true
                                for k := i + 1; k < i+length; k++ { // 继续检查行
                                        var temp int
                                        if j == 0 {
                                                temp = rowArr[k][j+length-1]
```

(续下页)

(接上页)

```
                                        } else {
                                                temp = rowArr[k][j+length-1] -␣
↪rowArr[k][j-1]
                                        }
                                        if temp != target {
                                                flag = false
                                                break
                                        }
                                }
                                if flag == false {
                                        continue
                                }
                                for k := j; k < j+length; k++ { // 检查列
                                        var temp int
                                        if i == 0 {
                                                temp = colArr[i+length-1][k]
                                        } else {
                                                temp = colArr[i+length-1][k] -␣
↪colArr[i-1][k]
                                        }
                                        if temp != target {
                                                flag = false
                                                break
                                        }
                                }
                                if flag == false {
                                        continue
                                }
                                var left, right int // 左右对角线
                                for k := 0; k < length; k++ {
                                        left = left + grid[i+k][j+k]
                                        right = right + grid[i+k][j+length-1-k]
                                }
                                if target == left && target == right {
                                        return length
                                }
                        }
                }
        }
        return 1
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}

# 2
func largestMagicSquare(grid [][]int) int {
        n, m := len(grid), len(grid[0])
        rowArr := make([][]int, n+1)   // 行前缀和
        colArr := make([][]int, n+1)   // 列前缀和
        leftArr := make([][]int, n+1)  // 对角线
        rightArr := make([][]int, n+1) // 对角线
        for i := 0; i <= n; i++ {
                rowArr[i] = make([]int, m+1)
                colArr[i] = make([]int, m+1)
                leftArr[i] = make([]int, m+1)
                rightArr[i] = make([]int, m+1)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        rowArr[i][j+1] = rowArr[i][j] + grid[i][j]
                        colArr[i+1][j] = colArr[i][j] + grid[i][j]
                        leftArr[i+1][j+1] = leftArr[i][j] + grid[i][j]
                        rightArr[i+1][j] = rightArr[i][j+1] + grid[i][j]
                }
        }
        for length := min(n, m); length >= 2; length-- { // 枚举边长
                for i := 0; i+length <= n; i++ {
                        for j := 0; j+length <= m; j++ {
                                target := leftArr[i+length][j+length] - leftArr[i][j]
                                if rightArr[i+length][j]-rightArr[i][j+length] !=␣
↪target {
                                        continue
                                }
                                flag := true
                                for k := i; k < i+length; k++ { // 检查行
                                        temp := rowArr[k][j+length] - rowArr[k][j]
                                        if temp != target {
                                                flag = false
                                                break
                                        }
                                }
```

(续下页)

(接上页)

```
                                if flag == false {
                                        continue
                                }
                                for k := j; k < j+length; k++ { // 检查列
                                        temp := colArr[i+length][k] - colArr[i][k]
                                        if temp != target {
                                                flag = false
                                                break
                                        }
                                }
                                if flag == true {
                                        return length
                                }
                        }
                }
        }
        return 1
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

56.38 1898. 可移除字符的最大数目 (3)

- 题目

```
给你两个字符串 s 和 p ，其中 p 是 s 的一个 子序列 。
同时，给你一个元素 互不相同 且下标 从 0 开始 计数的整数数组removable ，
该数组是 s 中下标的一个子集（s 的下标也 从 0 开始 计数）。
请你找出一个整数 k（0 <= k <= removable.length），选出removable 中的 前 k 个下标，
然后从 s 中移除这些下标对应的 k 个字符。整数 k 需满足：在执行完上述步骤后， p 仍然是␣
↪s 的一个 子序列 。
更正式的解释是，对于每个 0 <= i < k ，先标记出位于 s[removable[i]] 的字符，
接着移除所有标记过的字符，然后检查 p 是否仍然是 s 的一个子序列。
返回你可以找出的 最大 k ，满足在移除字符后 p 仍然是 s 的一个子序列。
字符串的一个 子序列 是一个由原字符串生成的新字符串，
生成过程中可能会移除原字符串中的一些字符（也可能不移除）但不改变剩余字符之间的相对顺序。
示例 1：输入：s = "abcacb", p = "ab", removable = [3,1,0] 输出：2
```

(续下页)

(接上页)

```
解释：在移除下标 3 和 1 对应的字符后，"abcacb" 变成 "accb" 。
"ab" 是 "accb" 的一个子序列。
如果移除下标 3、1 和 0 对应的字符后，"abcacb" 变成 "ccb" ，那么 "ab" 就不再是 s␣
↪的一个子序列。
因此，最大的 k 是 2 。
示例 2：输入：s = "abcbddddd", p = "abcd", removable = [3,2,1,4,5,6] 输出：1
解释：在移除下标 3 对应的字符后，"abcbddddd" 变成 "abcddddd" 。
"abcd" 是 "abcddddd" 的一个子序列。
示例 3：输入：s = "abcab", p = "abc", removable = [0,1,2,3,4] 输出：0
解释：如果移除数组 removable 的第一个下标，"abc" 就不再是 s 的一个子序列。
提示：1 <= p.length <= s.length <= 105
0 <= removable.length < s.length
0 <= removable[i] < s.length
p 是 s 的一个 子字符串
s 和 p 都由小写英文字母组成
removable 中的元素 互不相同
```

- 解题思路

```
func maximumRemovals(s string, p string, removable []int) int {
        n := len(removable)
        left, right := 0, n
        for left < right {
                mid := left + (right-left)/2
                if judge(s, p, removable, mid) == true {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

// leetcode 392.判断子序列
func judge(s string, p string, removable []int, index int) bool {
        m := make(map[int]bool)
        for i := 0; i < len(removable[:index+1]); i++ {
                m[removable[i]] = true
        }
        i, j := 0, 0
        for i < len(p) && j < len(s) {
                if p[i] == s[j] && m[j] == false {
                        i++
                }
```

(续下页)

(接上页)

```
                j++
        }
        return i == len(p)
}

# 2
func maximumRemovals(s string, p string, removable []int) int {
        n := len(removable)
        left, right := 0, n+1
        for left < right {
                mid := left + (right-left)/2
                if judge(s, p, removable, mid) == true {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left - 1
}

// leetcode 392.判断子序列
func judge(s string, p string, removable []int, index int) bool {
        m := make(map[int]bool)
        for i := 0; i < len(removable[:index]); i++ {
                m[removable[i]] = true
        }
        i, j := 0, 0
        for i < len(p) && j < len(s) {
                if p[i] == s[j] && m[j] == false {
                        i++
                }
                j++
        }
        return i == len(p)
}

# 3
func maximumRemovals(s string, p string, removable []int) int {
        n := len(removable)
        return sort.Search(n, func(index int) bool {
                m := make(map[int]bool)
                for i := 0; i < len(removable[:index+1]); i++ {
                        m[removable[i]] = true
```

(续下页)

(接上页)

```
                }
                i, j := 0, 0
                for i < len(p) && j < len(s) {
                        if p[i] == s[j] && m[j] == false {
                                i++
                        }
                        j++
                }
                return i != len(p)
        })
}
```

56.39 1899. 合并若干三元组以形成目标三元组 (2)

- 题目

```
三元组 是一个由三个整数组成的数组。给你一个二维整数数组 triplets ，其中 triplets[i] =
↪[ai, bi, ci] 表示第 i 个 三元组 。
同时，给你一个整数数组 target = [x, y, z] ，表示你想要得到的 三元组 。
为了得到 target ，你需要对 triplets 执行下面的操作 任意次（可能 零 次）：
选出两个下标（下标 从 0 开始 计数）i 和 j（i != j），
并 更新 triplets[j] 为 [max(ai, aj), max(bi, bj), max(ci, cj)] 。
例如，triplets[i] = [2, 5, 3] 且 triplets[j] = [1, 7, 5]，
triplets[j] 将会更新为 [max(2, 1), max(5, 7), max(3, 5)] = [2, 7, 5] 。
如果通过以上操作我们可以使得目标 三元组target成为triplets 的一个 元素，则返回 true
↪；否则，返回 false 。
示例 1：输入：triplets = [[2,5,3],[1,8,4],[1,7,5]], target = [2,7,5] 输出：true
解释：执行下述操作：
- 选择第一个和最后一个三元组 [[2,5,3],[1,8,4],[1,7,5]] 。
更新最后一个三元组为 [max(2,1), max(5,7), max(3,5)] = [2,7,5] 。triplets = [[2,5,3],
↪[1,8,4],[2,7,5]]
目标三元组 [2,7,5] 现在是 triplets 的一个元素。
示例 2：输入：triplets = [[1,3,4],[2,5,8]], target = [2,5,8] 输出：true
解释：目标三元组 [2,5,8] 已经是 triplets 的一个元素。
示例 3：输入：triplets = [[2,5,3],[2,3,4],[1,2,5],[5,2,3]], target = [5,5,5]
↪输出：true
解释：执行下述操作：
- 选择第一个和第三个三元组 [[2,5,3],[2,3,4],[1,2,5],[5,2,3]] 。
更新第三个三元组为 [max(2,1), max(5,2), max(3,5)] = [2,5,5] 。
triplets = [[2,5,3],[2,3,4],[2,5,5],[5,2,3]] 。
- 选择第三个和第四个三元组 [[2,5,3],[2,3,4],[2,5,5],[5,2,3]] 。
更新第四个三元组为 [max(2,5), max(5,2), max(5,3)] = [5,5,5] 。
```

(续下页)

(接上页)

```
triplets = [[2,5,3],[2,3,4],[2,5,5],[5,5,5]] 。
目标三元组 [5,5,5] 现在是 triplets 的一个元素。
示例 4：输入：triplets = [[3,4,5],[4,5,6]], target = [3,2,5] 输出：false
解释：无法得到 [3,2,5] ，因为 triplets 不含 2 。
提示：1 <= triplets.length <= 105
triplets[i].length == target.length == 3
1 <= ai, bi, ci, x, y, z <= 1000
```

- 解题思路

```
func mergeTriplets(triplets [][]int, target []int) bool {
        x, y, z := target[0], target[1], target[2]
        a, b, c := 0, 0, 0
        for i := 0; i < len(triplets); i++ {
                a1, b1, c1 := triplets[i][0], triplets[i][1], triplets[i][2]
                if a1 <= x && b1 <= y && c1 <= z {
                        a = max(a, a1)
                        b = max(b, b1)
                        c = max(c, c1)
                }
        }
        return a == x && b == y && c == z
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func mergeTriplets(triplets [][]int, target []int) bool {
        x, y, z := target[0], target[1], target[2]
        a, b, c := false, false, false
        for i := 0; i < len(triplets); i++ {
                a1, b1, c1 := triplets[i][0], triplets[i][1], triplets[i][2]
                if a1 <= x && b1 <= y && c1 <= z {
                        if a1 == x {
                                a = true
                        }
                        if b1 == y {
                                b = true
                        }
```

(续下页)

(接上页)

```
                        if c1 == z {
                                c = true
                        }
                }
        }
        return a == true && b == true && c == true
}
```

CHAPTER 57

1801-1900-Hard

57.1 1803. 统计异或值在范围内的数对有多少 (1)

- 题目

```
给你一个整数数组 nums （下标 从 0 开始 计数）以及两个整数：low 和 high ，请返回␣
↪漂亮数对 的数目。
漂亮数对 是一个形如 (i, j) 的数对，其中 0 <= i < j < nums.length 且 low <= (nums[i]␣
↪XOR nums[j]) <= high 。
示例 1：输入：nums = [1,4,2,7], low = 2, high = 6 输出：6
解释：所有漂亮数对 (i, j) 列出如下：
    - (0, 1): nums[0] XOR nums[1] = 5
    - (0, 2): nums[0] XOR nums[2] = 3
    - (0, 3): nums[0] XOR nums[3] = 6
    - (1, 2): nums[1] XOR nums[2] = 6
    - (1, 3): nums[1] XOR nums[3] = 3
    - (2, 3): nums[2] XOR nums[3] = 5
示例 2：输入：nums = [9,8,4,2,1], low = 5, high = 14 输出：8
解释：所有漂亮数对 (i, j) 列出如下：
    - (0, 2): nums[0] XOR nums[2] = 13
   - (0, 3): nums[0] XOR nums[3] = 11
   - (0, 4): nums[0] XOR nums[4] = 8
   - (1, 2): nums[1] XOR nums[2] = 12
   - (1, 3): nums[1] XOR nums[3] = 10
   - (1, 4): nums[1] XOR nums[4] = 9
```

(续下页)

(接上页)

```
   - (2, 3): nums[2] XOR nums[3] = 6
   - (2, 4): nums[2] XOR nums[4] = 5
提示：1 <= nums.length <= 2 * 104
1 <= nums[i] <= 2 * 104
1 <= low <= high <= 2 * 104
```

- 解题思路

```
func countPairs(nums []int, low int, high int) int {
        res := 0
        n := len(nums)
        root := &Trie{next: make([]*Trie, 2)}
        for i := 0; i < n; i++ {
                // 先查询，再插入
                res = res + root.Search(nums[i], high+1) - root.Search(nums[i], low)
                root.Insert(nums[i])
        }
        return res
}

type Trie struct {
        next []*Trie // 0或者1
        size int     // 次数
}

// 插入num
func (t *Trie) Insert(num int) {
        temp := t
        for i := 31; i >= 0; i-- {
                value := (num >> i) & 1
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next: make([]*Trie, 2),
                        }
                }
                temp = temp.next[value]
                temp.size++
        }
}

// 查找小于target的数量
func (t *Trie) Search(num int, target int) int {
        res := 0
        temp := t
```

(续下页)

(接上页)

```
        for i := 31; i >= 0; i-- {
                if temp == nil { // 直接返回
                        return res
                }
                value := (num >> i) & 1
                targetValue := (target >> i) & 1
                if targetValue > 0 { // target该位为1
                        if temp.next[value] != nil {
                                res = res + temp.next[value].size
                        }
                        temp = temp.next[1-value] // value ^ (1-value) = 1 => 往1-
↪value走
                } else {
                        temp = temp.next[value] // value ^ value = 0 // 往value走
                }
        }
        return res
}
```

57.2 1808. 好因子的最大数目 (1)

- 题目

```
给你一个正整数primeFactors。你需要构造一个正整数n，它满足以下条件：
n质因数（质因数需要考虑重复的情况）的数目 不超过primeFactors个。
n好因子的数目最大化。如果 n的一个因子可以被 n的每一个质因数整除，我们称这个因子是␣
↪好因子 。
比方说，如果n = 12，那么它的质因数为[2,2,3]，那么6和12是好因子，但3 和4不是。
请你返回n的好因子的数目。由于答案可能会很大，请返回答案对109 + 7取余的结果。
请注意，一个质数的定义是大于 1，且不能被分解为两个小于该数的自然数相乘。
一个数 n的质因子是将 n分解为若干个质因子，且它们的乘积为 n。
示例 1：输入：primeFactors = 5 输出：6
解释：200 是一个可行的 n 。
它有 5 个质因子：[2,2,2,5,5] ，且有 6 个好因子：[10,20,40,50,100,200] 。
不存在别的 n 有至多 5 个质因子，且同时有更多的好因子。
示例 2：输入：primeFactors = 8 输出：18
提示：1 <= primeFactors <= 109
```

- 解题思路

```
/*
由题意有：n=a1^k1 * a2^k2 *...*an^kn(如12=2^2 * 3^1)
```

(续下页)

(接上页)

```
其中
1、a1,a2,...,an是不同的质数(2,3不重复)
2、k1+k2+...+kn <=primeFactors
3、n的好因子,要能被每一个质因数(a1,a2,a3,...,an)整除，即好因子必须含有a1*a2*...
↪*an作为因数
=>好因子的个数 k = k1*k2*...*kn =>求k最大，其中k1+..kn=primeFactors
等价于343题，整数拆分
*/

var mod = 1000000007

func maxNiceDivisors(primeFactors int) int {
        n := primeFactors
        if n <= 3 {
                return n
        }
        if n%3 == 0 {
                return pow(3, n/3) % mod
        } else if n%3 == 1 {
                return pow(3, (n-4)/3) * 4 % mod
        }
        return pow(3, n/3) * 2 % mod
}

func pow(a, b int) int {
        res := 1
        for b > 0 {
                if b%2 == 1 {
                        res = res * a % mod
                }
                a = a * a % mod
                b = b / 2
        }
        return res
}
```

57.3 1835. 所有数对按位与结果的异或和 (2)

- 题目

```
列表的 异或和 (XOR sum) 指对所有元素进行按位 XOR␣
↪运算的结果。如果列表中仅有一个元素，那么其 异或和 就等于该元素。
例如，[1,2,3,4] 的 异或和 等于 1 XOR 2 XOR 3 XOR 4 = 4 ，而 [3] 的 异或和 等于 3 。
给你两个下标 从 0 开始 计数的数组 arr1 和 arr2 ，两数组均由非负整数组成。
根据每个(i, j) 数对，构造一个由 arr1[i] AND arr2[j] (按位 AND 运算) 结果组成的列表。
其中 0 <= i < arr1.length 且 0 <= j < arr2.length 。
返回上述列表的 异或和 。
示例 1: 输入: arr1 = [1,2,3], arr2 = [6,5] 输出: 0
解释: 列表 = [1 AND 6, 1 AND 5, 2 AND 6, 2 AND 5, 3 AND 6, 3 AND 5] = [0,1,2,0,2,1] ，
异或和 = 0 XOR 1 XOR 2 XOR 0 XOR 2 XOR 1 = 0 。
示例 2: 输入: arr1 = [12], arr2 = [4] 输出: 4
解释: 列表 = [12 AND 4] = [4] ，异或和 = 4 。
提示: 1 <= arr1.length, arr2.length <= 105
0 <= arr1[i], arr2[j] <= 109
```

- 解题思路

```
func getXORSum(arr1 []int, arr2 []int) int {
        res := 0
        n, m := len(arr1), len(arr2)
        for i := 31; i >= 0; i-- {
                a, b := 0, 0
                for j := 0; j < n; j++ {
                        if (arr1[j] & (1 << i)) > 0 {
                                a++
                        }
                }
                for j := 0; j < m; j++ {
                        if (arr2[j] & (1 << i)) > 0 {
                                b++
                        }
                }
                if a%2 == 1 && b%2 == 1 { // 在该位1的次数都为奇数: 奇数x奇数=奇数
                        res = res | (1 << i)
                }
        }
        return res
}

# 2
```

(续下页)

(接上页)

```
func getXORSum(arr1 []int, arr2 []int) int {
        a, b := 0, 0
        for i := 0; i < len(arr1); i++ {
                a = a ^ arr1[i]
        }
        for i := 0; i < len(arr2); i++ {
                b = b ^ arr2[i]
        }
        return a & b // 位与操作：相同位数都是1，则该结果返回1
}
```

57.4 1857. 有向图中最大颜色值 (1)

- 题目

```
给你一个有向图，它含有n个节点和 m条边。节点编号从0 到n - 1。
给你一个字符串colors ，其中colors[i]是小写英文字母，表示图中第 i个节点的 颜色（下标从0开始）。
同时给你一个二维数组edges，其中edges[j] = [aj, bj]表示从节点aj到节点bj有一条有向边。
图中一条有效 路径是一个点序列x1 -> x2 -> x3 -> ... -> xk，对于所有1 <= i < k，从xi到xi+1在图中有一条有向边。
路径的 颜色值是路径中 出现次数最多 颜色的节点数目。
请你返回给定图中有效路径里面的最大颜色值。如果图中含有环，请返回 -1。
示例 1：输入：colors = "abaca", edges = [[0,1],[0,2],[2,3],[3,4]] 输出：3
解释：路径 0 -> 2 -> 3 -> 4 含有 3 个颜色为 "a" 的节点（上图中的红色节点）。
示例 2：输入：colors = "a", edges = [[0,0]] 输出：-1
解释：从 0 到 0 有一个环。
提示：n == colors.length
m == edges.length
1 <= n <= 105
0 <= m <= 105
colors只含有小写英文字母。
0 <= aj, bj< n
```

- 解题思路

```
func largestPathValue(colors string, edges [][]int) int {
        n := len(colors)
        arr := make([][]int, n)  // 邻接表
        degree := make([]int, n) // 入度
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a=>b
```

(续下页)

(接上页)

```
                arr[a] = append(arr[a], b)
                degree[b]++
        }
        queue := make([]int, 0)
        for i := 0; i < n; i++ {
                if degree[i] == 0 { // 入度为0
                        queue = append(queue, i)
                }
        }
        dp := make([][26]int, n) // dp[i][j] => 到节点i颜色j出现的次数
        count := 0
        for len(queue) > 0 {
                count++ // 节点+1
                node := queue[0]
                queue = queue[1:]
                dp[node][int(colors[node]-'a')]++ // 该节点颜色+1
                for i := 0; i < len(arr[node]); i++ {
                        next := arr[node][i]
                        degree[next]--
                        if degree[next] == 0 {
                                queue = append(queue, next)
                        }
                        // 更新次数
                        for j := 0; j < 26; j++ {
                                dp[next][j] = max(dp[next][j], dp[node][j])
                        }
                }
        }
        if count != n { // 判断环
                return -1
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < 26; j++ {
                        res = max(res, dp[i][j])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}
```

57.5 1862. 向下取整数对和 (1)

- 题目

```
给你一个整数数组nums，请你返回所有下标对0 <= i, j < nums.length的floor(nums[i] /␣
↪nums[j])结果之和。
由于答案可能会很大，请你返回答案对109 + 7取余的结果。
函数floor()返回输入数字的整数部分。
示例 1：输入：nums = [2,5,9] 输出：10
解释：floor(2 / 5) = floor(2 / 9) = floor(5 / 9) = 0
floor(2 / 2) = floor(5 / 5) = floor(9 / 9) = 1
floor(5 / 2) = 2
floor(9 / 2) = 4
floor(9 / 5) = 1
我们计算每一个数对商向下取整的结果并求和得到 10 。
示例 2：输入：nums = [7,7,7,7,7,7,7] 输出：49
提示：1 <= nums.length <= 105
1 <= nums[i] <= 105
```

- 解题思路

```
func sumOfFlooredPairs(nums []int) int {
        n := len(nums)
        count := make([]int, 200001)
        for i := 0; i < n; i++ {
                count[nums[i]]++
        }
        arr := make([]int, 200001+1) // 前缀和
        for i := 0; i < len(count); i++ {
                arr[i+1] = arr[i] + count[i]
        }
        res := 0
        // a/b = c
        for i := 0; i < len(count); i++ { // 枚举b
                if count[i] > 0 { // i个数大于0
                        for j := 1; i*j <= 100000; j++ { // 枚举c
                                // b的个数 X c X 目标范围内的数字个数
                                total := count[i] * j * (arr[i*(j+1)] - arr[i*j])
```

(续下页)

(接上页)

```
                                res = (res + total) % 1000000007
                        }
                }
        }
        return res
}
```

57.6 1866. 恰有 K 根木棍可以看到的排列数目 (1)

- 题目

```
有 n 根长度互不相同的木棍，长度为从 1 到 n 的整数。
请你将这些木棍排成一排，并满足从左侧 可以看到恰好 k 根木棍。
从左侧 可以看到 木棍的前提是这个木棍的 左侧 不存在比它 更长的 木棍。
例如，如果木棍排列为 [1,3,2,5,4] ，那么从左侧可以看到的就是长度分别为 1、3 、5␣
↪的木棍。
给你 n 和 k ，返回符合题目要求的排列 数目 。由于答案可能很大，请返回对 109 + 7 取余␣
↪的结果。
示例 1：输入：n = 3, k = 2 输出：3
解释：[1,3,2], [2,3,1] 和 [2,1,3] 是仅有的能满足恰好 2 根木棍可以看到的排列。
可以看到的木棍已经用粗体+斜体标识。
示例 2：输入：n = 5, k = 5 输出：1
解释：[1,2,3,4,5] 是唯一一种能满足全部 5 根木棍可以看到的排列。
可以看到的木棍已经用粗体+斜体标识。
示例 3：输入：n = 20, k = 11 输出：647427950
解释：总共有 647427950 (mod 109 + 7) 种能满足恰好有 11 根木棍可以看到的排列。
提示：1 <= n <= 1000
1 <= k <= n
```

- 解题思路

```
func rearrangeSticks(n int, k int) int {
        dp := make([][]int, n+1) //␣
↪dp[i][j]代表i根棍子能看到j根；假设每次插入1，假设上一轮的数为：2到n
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
        }
        dp[0][0] = 1
        for i := 1; i <= n; i++ {
                for j := 1; j <= k; j++ {
                        a := dp[i-1][j-1]                       // i-
↪1根棍子里面已经看到了j-1根，插入到最前面
```

(续下页)

(接上页)

```
                        b := ((i - 1) * dp[i-1][j]) % 1000000007 // i-
↪1根棍子里面已经看到了j根，插入到除最前面的其它i-1个地方都行
                        dp[i][j] = (a + b) % 1000000007
                }
        }
        return dp[n][k]
}
```

57.7 1872. 石子游戏 VIII

57.7.1 题目

```
Alice 和 Bob 玩一个游戏，两人轮流操作， Alice 先手。
总共有n个石子排成一行。轮到某个玩家的回合时，如果石子的数目 大于 1，他将执行以下操作：
选择一个整数x > 1，并且 移除最左边的x个石子。
将移除的石子价值之 和累加到该玩家的分数中。
将一个 新的石子放在最左边，且新石子的值为被移除石子值之和。
当只剩下 一个石子时，游戏结束。
Alice 和 Bob 的 分数之差为(Alice 的分数- Bob 的分数)。
Alice 的目标是最大化分数差，Bob 的目标是 最小化分数差。
给你一个长度为 n的整数数组stones，其中stones[i]是 从左边起第i个石子的价值。
请你返回在双方都采用 最优 策略的情况下，Alice 和 Bob 的 分数之差 。
示例 1：输入：stones = [-1,2,-3,4,-5] 输出：5
解释：- Alice 移除最左边的 4 个石子，得分增加 (-1) + 2 + (-3) + 4 = 2␣
↪，并且将一个价值为 2 的石子放在最左边。
stones = [2,-5] 。
- Bob 移除最左边的 2 个石子，得分增加 2 + (-5) = -3 ，并且将一个价值为 -3␣
↪的石子放在最左边。stones = [-3] 。
两者分数之差为 2 - (-3) = 5 。
示例 2：输入：stones = [7,-6,5,10,5,-2,-6] 输出：13
解释：- Alice 移除所有石子，得分增加 7 + (-6) + 5 + 10 + 5 + (-2) + (-6) = 13 ，
并且将一个价值为 13 的石子放在最左边。stones = [13] 。
两者分数之差为 13 - 0 = 13 。
示例 3：输入：stones = [-10,-12] 输出：-22
解释：- Alice 只有一种操作，就是移除所有石子。得分增加 (-10) + (-12) = -22␣
↪，并且将一个价值为 -22 的石子放在最左边。
stones = [-22] 。
两者分数之差为 (-22) - 0 = -22 。
提示：n == stones.length
2 <= n <= 105
-104 <= stones[i] <= 104
```

57.7.2 解题思路

57.8 1889. 装包裹的最小浪费空间 (3)

- 题目

```
给你n个包裹，你需要把它们装在箱子里，每个箱子装一个包裹。
总共有m个供应商提供 不同尺寸的箱子（每个规格都有无数个箱子）。
如果一个包裹的尺寸 小于等于一个箱子的尺寸，那么这个包裹就可以放入这个箱子之中。
包裹的尺寸用一个整数数组packages表示，其中packages[i]是第i个包裹的尺寸。
供应商用二维数组boxes表示，其中boxes[j]是第 j个供应商提供的所有箱子尺寸的数组。
你想要选择 一个供应商并只使用该供应商提供的箱子，使得 总浪费空间最小。
对于每个装了包裹的箱子，我们定义 浪费的空间等于 箱子的尺寸 -␣
↪包裹的尺寸。总浪费空间为所有箱子中浪费空间的总和。
比方说，如果你想要用尺寸数组为[4,8]的箱子装下尺寸为[2,3,5]的包裹，
你可以将尺寸为 2和 3的两个包裹装入两个尺寸为 4的箱子中，同时把尺寸为␣
↪5的包裹装入尺寸为 8的箱子中。
总浪费空间为(4-2) + (4-3) + (8-5) = 6。
请你选择 最优箱子供应商，使得 总浪费空间最小。如果 无法␣
↪将所有包裹放入箱子中，请你返回 -1。
由于答案可能会 很大，请返回它对109 + 7取余的结果。
示例 1：输入：packages = [2,3,5], boxes = [[4,8],[2,8]] 输出：6
解释：选择第一个供应商最优，用两个尺寸为 4 的箱子和一个尺寸为 8 的箱子。
总浪费空间为 (4-2) + (4-3) + (8-5) = 6 。
示例 2：输入：packages = [2,3,5], boxes = [[1,4],[2,3],[3,4]] 输出：-1
解释：没有箱子能装下尺寸为 5 的包裹。
示例 3：输入：packages = [3,5,8,10,11,12], boxes = [[12],[11,9],[10,5,14]] 输出：9
解释：选择第三个供应商最优，用两个尺寸为 5 的箱子，两个尺寸为 10 的箱子和两个尺寸为␣
↪14 的箱子。
总浪费空间为 (5-3) + (5-5) + (10-8) + (10-10) + (14-11) + (14-12) = 9 。
提示：n == packages.length
m == boxes.length
1 <= n <= 105
1 <= m <= 105
1 <= packages[i] <= 105
1 <= boxes[j].length <= 105
1 <= boxes[j][k] <= 105
sum(boxes[j].length) <= 105
boxes[j]中的元素 互不相同。
```

- 解题思路

```
var mod = 1000000007

func minWastedSpace(packages []int, boxes [][]int) int {
        sort.Ints(packages)
        n := len(packages)
        res := math.MaxInt64
        for i := 0; i < len(boxes); i++ {
                temp := boxes[i]
                sort.Ints(temp)
                if temp[len(temp)-1] < packages[n-1] { // 最大箱子无法装最大包裹
                        continue
                }
                sum := 0
                left := 0
                for j := 0; j < len(temp); j++ { // 枚举箱子
                        right := binarySearch(packages, temp[j]) //␣
↪选择当前箱子能装下的位置
                        sum = sum + (right-left)*temp[j]         //␣
↪累加：个数*箱子大小
                        left = right
                }
                res = min(res, sum)
        }
        if res == math.MaxInt64 {
                return -1
        }
        for i := 0; i < n; i++ {
                res = res - packages[i]
        }
        return res % mod
}

// 找到第一个大于target的位置
func binarySearch(arr []int, target int) int {
        left, right := 0, len(arr)-1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] <= target {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return left
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var mod = 1000000007

func minWastedSpace(packages []int, boxes [][]int) int {
        sort.Ints(packages)
        n := len(packages)
        res := math.MaxInt64
        for i := 0; i < len(boxes); i++ {
                temp := boxes[i]
                sort.Ints(temp)
                if temp[len(temp)-1] < packages[n-1] { // 最大箱子无法装最大包裹
                        continue
                }
                sum := 0
                left := 0
                for j := 0; j < len(temp); j++ { // 枚举箱子
                        // 找到第一个大于target的位置
                        right := sort.SearchInts(packages, temp[j]+1) //␣
↪选择当前箱子能装下的位置
                        sum = sum + (right-left)*temp[j]              //␣
↪累加：个数*箱子大小
                        left = right
                }
                res = min(res, sum)
        }
        if res == math.MaxInt64 {
                return -1
        }
        for i := 0; i < n; i++ {
                res = res - packages[i]
        }
        return res % mod
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var mod = 1000000007

func minWastedSpace(packages []int, boxes [][]int) int {
        sort.Ints(packages)
        n := len(packages)
        arr := make([]int, n+1)
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1] + packages[i-1]
        }
        res := math.MaxInt64
        for i := 0; i < len(boxes); i++ {
                temp := boxes[i]
                sort.Ints(temp)
                if temp[len(temp)-1] < packages[n-1] { // 最大箱子无法装最大包裹
                        continue
                }
                sum := 0
                left := 0
                for j := 0; j < len(temp); j++ { // 枚举箱子
                        right := binarySearch(packages, temp[j], left) //␣
↪选择当前箱子能装下的位置
                        sum = sum + (right-left)*temp[j] - (arr[right] - arr[left])
                        left = right
                }
                res = min(res, sum)
        }
        if res == math.MaxInt64 {
                return -1
        }
        return res % mod
}

func binarySearch(arr []int, target int, left int) int {
        right := len(arr)
        for left < right {
```

(续下页)

(接上页)

```
                mid := left + (right-left)/2
                if arr[mid] <= target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

CHAPTER 58

1901-2000-Easy

58.1 1903. 字符串中的最大奇数 (1)

- 题目

```
给你一个字符串 num ，表示一个大整数。请你在字符串 num 的所有非空子字符串 中找出值最大的奇数 ，并以字符串形式返回。
如果不存在奇数，则返回一个空字符串 "" 。
子字符串 是字符串中的一个连续的字符序列。
示例 1：输入：num = "52" 输出："5"
解释：非空子字符串仅有 "5"、"2" 和 "52" 。"5" 是其中唯一的奇数。
示例 2：输入：num = "4206" 输出：""
解释：在 "4206" 中不存在奇数。
示例 3：输入：num = "35427" 输出："35427"
解释："35427" 本身就是一个奇数。
提示：1 <= num.length <= 105
num 仅由数字组成且不含前导零
```

- 解题思路

```
func largestOddNumber(num string) string {
        n := len(num)
        for i := n - 1; i >= 0; i-- {
                if int(num[i]-'0')%2 == 1 {
                        return num[:i+1]
```

(续下页)

(接上页)

```
                }
        }
        return ""
}
```

58.2 1909. 删除一个元素使数组严格递增 (3)

- 题目

```
给你一个下标从␣
↪0开始的整数数组nums，如果恰好删除一个元素后，数组严格递增，那么请你返回true，否则返回false。
如果数组本身已经是严格递增的，请你也返回true。
数组nums是 严格递增的定义为：对于任意下标的1 <= i < nums.length都满足nums[i - 1] <␣
↪nums[i]。
示例 1：输入：nums = [1,2,10,5,7] 输出：true
解释：从 nums 中删除下标 2 处的 10 ，得到 [1,2,5,7] 。
[1,2,5,7] 是严格递增的，所以返回 true 。
示例 2：输入：nums = [2,3,1,2] 输出：false
解释：[3,1,2] 是删除下标 0 处元素后得到的结果。
[2,1,2] 是删除下标 1 处元素后得到的结果。
[2,3,2] 是删除下标 2 处元素后得到的结果。
[2,3,1] 是删除下标 3 处元素后得到的结果。
没有任何结果数组是严格递增的，所以返回 false 。
示例 3：输入：nums = [1,1,1] 输出：false
解释：删除任意元素后的结果都是 [1,1] 。
[1,1] 不是严格递增的，所以返回 false 。
示例 4：输入：nums = [1,2,3] 输出：true
解释：[1,2,3] 已经是严格递增的，所以返回 true 。
提示：2 <= nums.length <= 1000
1 <= nums[i] <= 1000
```

- 解题思路

```
func canBeIncreasing(nums []int) bool {
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] >= nums[i+1] {
                        return judge(nums, i) == true || judge(nums, i+1) == true
                }
        }
        return true
}
```

(续下页)

(接上页)

```
func judge(nums []int, index int) bool {
        arr := append([]int{}, nums[:index]...)
        arr = append(arr, nums[index+1:]...)
        for i := 0; i < len(arr)-1; i++ {
                if arr[i] >= arr[i+1] {
                        return false
                }
        }
        return true
}

# 2
func canBeIncreasing(nums []int) bool {
        for i := 0; i < len(nums); i++ {
                if judge(nums, i) == true {
                        return true
                }
        }
        return false
}

func judge(nums []int, index int) bool {
        arr := append([]int{}, nums[:index]...)
        arr = append(arr, nums[index+1:]...)
        for i := 0; i < len(arr)-1; i++ {
                if arr[i] >= arr[i+1] {
                        return false
                }
        }
        return true
}

# 3
// leetcode300.最长上升子序列
func canBeIncreasing(nums []int) bool {
        n := len(nums)
        dp := make([]int, n)
        res := 1
        for i := 0; i < n; i++ {
                dp[i] = 1
                for j := 0; j < i; j++ {
                        if nums[j] < nums[i] {
                                dp[i] = max(dp[j]+1, dp[i])
```

(续下页)

(接上页)

```
                        }
                }
                res = max(res, dp[i])
        }
        return res == n || res == n-1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

58.3 1913. 两个数对之间的最大乘积差 (2)

- 题目

```
两个数对(a, b) 和 (c, d) 之间的 乘积差 定义为 (a * b) - (c * d) 。
例如，(5, 6) 和 (2, 7) 之间的乘积差是 (5 * 6) - (2 * 7) = 16 。
给你一个整数数组 nums ，选出四个 不同的 下标 w、x、y 和 z ，
使数对 (nums[w], nums[x]) 和 (nums[y], nums[z]) 之间的 乘积差 取到 最大值 。
返回以这种方式取得的乘积差中的 最大值 。
示例 1：输入：nums = [5,6,2,7,4] 输出：34
解释：可以选出下标为 1 和 3 的元素构成第一个数对 (6, 7) 以及下标 2 和 4␣
↪构成第二个数对 (2, 4)
乘积差是 (6 * 7) - (2 * 4) = 34
示例 2：输入：nums = [4,2,5,9,7,4,8] 输出：64
解释：可以选出下标为 3 和 6 的元素构成第一个数对 (9, 8) 以及下标 1 和 5␣
↪构成第二个数对 (2, 4)
乘积差是 (9 * 8) - (2 * 4) = 64
提示：4 <= nums.length <= 104
1 <= nums[i] <= 104
```

- 解题思路

```
func maxProductDifference(nums []int) int {
        sort.Ints(nums)
        n := len(nums)
        return nums[n-1]*nums[n-2] - nums[0]*nums[1]
}
```

(续下页)

(接上页)

```
# 2
func maxProductDifference(nums []int) int {
        n := len(nums)
        minA, minB := min(nums[0], nums[1]), max(nums[0], nums[1])
        maxA, maxB := max(nums[0], nums[1]), min(nums[0], nums[1])
        for i := 2; i < n; i++ {
                if nums[i] > maxA {
                        maxA, maxB = nums[i], maxA
                } else if nums[i] > maxB {
                        maxB = nums[i]
                }
                if nums[i] < minA {
                        minA, minB = nums[i], minA
                } else if nums[i] < minB {
                        minB = nums[i]
                }
        }
        return maxA*maxB - minA*minB
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

58.4 1920. 基于排列构建数组 (2)

- 题目

```
给你一个 从 0 开始的排列 nums（下标也从 0 开始）。
请你构建一个 同样长度 的数组 ans ，其中，对于每个 i（0 <= i < nums.length），都满足␣
↪ans[i] = nums[nums[i]] 。
返回构建好的数组 ans 。
```

(续下页)

(接上页)

```
从 0 开始的排列 nums 是一个由 0 到nums.length - 1 (0 和 nums.length - 1
↪也包含在内) 的不同整数组成的数组。
示例 1: 输入: nums = [0,2,1,5,3,4] 输出: [0,1,2,4,5,3]
解释: 数组 ans 构建如下:
ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]],
↪nums[nums[5]]]
    = [nums[0], nums[2], nums[1], nums[5], nums[3], nums[4]]
    = [0,1,2,4,5,3]
示例 2: 输入: nums = [5,0,1,2,3,4] 输出: [4,5,0,1,2,3]
解释: 数组 ans 构建如下:
ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]],
↪nums[nums[5]]]
    = [nums[5], nums[0], nums[1], nums[2], nums[3], nums[4]]
    = [4,5,0,1,2,3]
提示: 1 <= nums.length <= 1000
0 <= nums[i] < nums.length
nums 中的元素 互不相同
```

- 解题思路

```
func buildArray(nums []int) []int {
        n := len(nums)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = nums[nums[i]]
        }
        return res
}

# 2
func buildArray(nums []int) []int {
        n := len(nums)
        for i := 0; i < n; i++ {
                nums[i] = nums[i] + 1000*(nums[nums[i]]%1000) //
↪前3位存储目标值, 后3位存储原来值
        }
        for i := 0; i < n; i++ {
                nums[i] = nums[i] / 1000
        }
        return nums
}
```

58.5 1925. 统计平方和三元组的数目 (3)

- 题目

```
一个 平方和三元组(a,b,c)指的是满足 a2 + b2 = c2的 整数三元组a, b和c。
给你一个整数n, 请你返回满足1 <= a, b, c <= n的 平方和三元组 的数目。
示例 1: 输入: n = 5 输出: 2
解释: 平方和三元组为 (3,4,5) 和 (4,3,5) 。
示例 2: 输入: n = 10 输出: 4
解释: 平方和三元组为 (3,4,5), (4,3,5), (6,8,10) 和 (8,6,10) 。
提示: 1 <= n <= 250
```

- 解题思路

```
func countTriples(n int) int {
        res := 0
        for i := 1; i <= n; i++ {
                for j := 1; j <= n; j++ {
                        for k := 1; k <= n; k++ {
                                if i*i+j*j == k*k {
                                        res++
                                }
                        }
                }
        }
        return res
}

# 2
func countTriples(n int) int {
        res := 0
        m := make(map[int]bool)
        for i := 1; i <= n; i++ {
                m[i*i] = true
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= n; j++ {
                        if m[i*i+j*j] == true {
                                res++
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 3
func countTriples(n int) int {
        res := 0
        for i := 1; i <= n; i++ {
                for j := 1; j <= n; j++ {
                        k := int(math.Sqrt(float64(i*i + j*j)))
                        if k*k == i*i+j*j && k <= n {
                                res++
                        }
                }
        }
        return res
}
```

58.6 1929. 数组串联 (3)

- 题目

```
给你一个长度为 n 的整数数组 nums 。请你构建一个长度为 2n 的答案数组 ans ，
数组下标 从 0 开始计数 ，对于所有0 <= i < n 的 i ，满足下述所有要求：
ans[i] == nums[i]
ans[i + n] == nums[i]
具体而言，ans 由两个 nums 数组 串联 形成。
返回数组 ans 。
示例 1：输入：nums = [1,2,1] 输出：[1,2,1,1,2,1]
解释：数组 ans 按下述方式形成：
- ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]]
- ans = [1,2,1,1,2,1]
示例 2：输入：nums = [1,3,2,1] 输出：[1,3,2,1,1,3,2,1]
解释：数组 ans 按下述方式形成：
- ans = [nums[0],nums[1],nums[2],nums[3],nums[0],nums[1],nums[2],nums[3]]
- ans = [1,3,2,1,1,3,2,1]
提示：n == nums.length
1 <= n <= 1000
1 <= nums[i] <= 1000
```

- 解题思路

```
func getConcatenation(nums []int) []int {
        n := len(nums)
        res := make([]int, 2*n)
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                res[i] = nums[i]
                res[i+n] = nums[i]
        }
        return res
}

# 2
func getConcatenation(nums []int) []int {
        n := len(nums)
        for i := 0; i < n; i++ {
                nums = append(nums, nums[i])
        }
        return nums
}

# 3
func getConcatenation(nums []int) []int {
        return append(nums, nums...)
}
```

58.7 1935. 可以输入的最大单词数 (2)

- 题目

```
键盘出现了一些故障，有些字母键无法正常工作。而键盘上所有其他键都能够正常工作。
给你一个由若干单词组成的字符串 text ，单词间由单个空格组成（不含前导和尾随空格）；
另有一个字符串 brokenLetters␣
↪，由所有已损坏的不同字母键组成，返回你可以使用此键盘完全输入的 text 中单词的数目。
示例 1：输入：text = "hello world", brokenLetters = "ad" 输出：1
解释：无法输入 "world" ，因为字母键 'd' 已损坏。
示例 2：输入：text = "leet code", brokenLetters = "lt" 输出：1
解释：无法输入 "leet" ，因为字母键 'l' 和 't' 已损坏。
示例 3：输入：text = "leet code", brokenLetters = "e" 输出：0
解释：无法输入任何单词，因为字母键 'e' 已损坏。
提示：1 <= text.length <= 104
0 <= brokenLetters.length <= 26
text 由若干用单个空格分隔的单词组成，且不含任何前导和尾随空格
每个单词仅由小写英文字母组成
brokenLetters 由 互不相同 的小写英文字母组成
```

- 解题思路

```
func canBeTypedWords(text string, brokenLetters string) int {
        res := 0
        arr := strings.Split(text, " ")
        for i := 0; i < len(arr); i++ {
                if strings.ContainsAny(arr[i], brokenLetters) == false {
                        res++
                }
        }
        return res
}

# 2
func canBeTypedWords(text string, brokenLetters string) int {
        res := 0
        m := make(map[byte]bool)
        for i := 0; i < len(brokenLetters); i++ {
                m[brokenLetters[i]] = true
        }
        arr := strings.Split(text, " ")
        for i := 0; i < len(arr); i++ {
                flag := true
                for j := 0; j < len(arr[i]); j++ {
                        if m[arr[i][j]] == true {
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res++
                }
        }
        return res
}
```

58.8 1941. 检查是否所有字符出现次数相同 (2)

- 题目

```
给你一个字符串s，如果 s是一个 好字符串，请你返回 true，否则请返回 false。
如果 s中出现过的所有 字符的出现次数 相同，那么我们称字符串 s是 好字符串。
示例 1：输入：s = "abacbc" 输出：true
解释：s 中出现过的字符为 'a'，'b' 和 'c' 。s 中所有字符均出现 2 次。
```

(续下页)

(接上页)

```
示例 2: 输入: s = "aaabb" 输出: false
解释: s 中出现过的字符为 'a' 和 'b' 。
'a' 出现了 3 次, 'b' 出现了 2 次, 两者出现次数不同。
提示: 1 <= s.length <= 1000
s只包含小写英文字母。
```

- 解题思路

```
func areOccurrencesEqual(s string) bool {
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
        }
        count := 0
        for _, v := range m {
                if count == 0 {
                        count = v
                } else {
                        if count != v {
                                return false
                        }
                }
        }
        return true
}

# 2
func areOccurrencesEqual(s string) bool {
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
        }
        count := len(s)/len(m)
        for _, v := range m {
                if v != count{
                        return false
                }
        }
        return true
}
```

58.9 1945. 字符串转化后的各位数字之和 (2)

- 题目

```
给你一个由小写字母组成的字符串 s ，以及一个整数 k 。
首先，用字母在字母表中的位置替换该字母，将 s 转化 为一个整数（也就是，'a' 用 1 替换，
↪'b' 用 2 替换，... 'z' 用 26 替换）。
接着，将整数 转换 为其 各位数字之和 。共重复 转换 操作 k 次 。
例如，如果 s = "zbax" 且 k = 2 ，那么执行下述步骤后得到的结果是整数 8 ：
转化："zbax" ➡ "(26)(2)(1)(24)" ➡ "262124" ➡ 262124
转换 #1：262124➡ 2 + 6 + 2 + 1 + 2 + 4➡ 17
转换 #2：17 ➡ 1 + 7 ➡ 8
返回执行上述操作后得到的结果整数。
示例 1：输入：s = "iiii", k = 1 输出：36
解释：操作如下：- 转化："iiii" ➡ "(9)(9)(9)(9)" ➡ "9999" ➡ 9999
- 转换 #1：9999 ➡ 9 + 9 + 9 + 9 ➡ 36
因此，结果整数为 36 。
示例 2：输入：s = "leetcode", k = 2 输出：6
解释：操作如下：- 转化："leetcode" ➡ "(12)(5)(5)(20)(3)(15)(4)(5)" ➡ "12552031545" ➡␣
↪12552031545
- 转换 #1：12552031545 ➡ 1 + 2 + 5 + 5 + 2 + 0 + 3 + 1 + 5 + 4 + 5 ➡ 33
- 转换 #2：33 ➡ 3 + 3 ➡ 6
因此，结果整数为 6 。
提示：1 <= s.length <= 100
1 <= k <= 10
s 由小写英文字母组成
```

- 解题思路

```
func getLucky(s string, k int) int {
        arr := make([]int, 0)
        for i := 0; i < len(s); i++ {
                value := int(s[i]-'a') + 1
                if value < 10 {
                        arr = append(arr, value)
                } else {
                        arr = append(arr, value/10)
                        arr = append(arr, value%10)
                }
        }
        for i := 1; i <= k; i++ {
                arr = trans(arr)
        }
        res := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(arr); i++ {
                res = 10*res + arr[i]
        }
        return res
}

func trans(arr []int) []int {
        sum := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i]
        }
        res := make([]int, 0)
        for sum > 0 {
                res = append([]int{sum % 10}, res...)
                sum = sum / 10
        }
        return res
}

# 2
func getLucky(s string, k int) int {
        sum := 0
        for i := 0; i < len(s); i++ {
                value := int(s[i]-'a') + 1
                sum = sum + (value/10 + value%10)
        }
        res := sum
        for i := 1; i <= k-1; i++ {
                sum = res
                res = 0
                for sum > 0 {
                        res = res + sum%10
                        sum = sum / 10
                }
        }
        return res
}
```

58.10 1952. 三除数 (3)

- 题目

```
给你一个整数 n 。如果 n 恰好有三个正除数 ，返回 true ；否则，返回 false 。
如果存在整数 k ，满足 n = k * m ，那么整数 m 就是 n 的一个 除数 。
示例 1：输入：n = 2 输出：false
解释：2 只有两个除数：1 和 2 。
示例 2：输入：n = 4 输出：true
解释：4 有三个除数：1、2 和 4 。
提示：1 <= n <= 104
```

- 解题思路

```
func isThree(n int) bool {
        count := 0
        for i := 1; i <= n; i++ {
                if n%i == 0 {
                        count++
                }
        }
        return count == 3
}

# 2
func isThree(n int) bool {
        count := 0
        for i := 1; i*i <= n; i++ {
                if n%i == 0 {
                        if i*i == n {
                                count++
                        } else {
                                count = count + 2
                        }
                }
        }
        return count == 3
}

# 3
func isThree(n int) bool {
        target := int(math.Sqrt(float64(n)))
        return target*target == n && isPrime(target) == true
}
```

(续下页)

(接上页)

```
func isPrime(n int) bool {
        for i := 2; i*i <= n; i++ {
                if n%i == 0 {
                        return false
                }
        }
        return n >= 2
}
```

58.11 1957. 删除字符使字符串变好 (2)

- 题目

```
一个字符串如果没有 三个连续相同字符，那么它就是一个 好字符串。
给你一个字符串s，请你从 s删除最少的字符，使它变成一个 好字符串 。
请你返回删除后的字符串。题目数据保证答案总是 唯一的 。
示例 1：输入：s = "leeetcode" 输出："leetcode"
解释：从第一组 'e' 里面删除一个 'e' ，得到 "leetcode" 。
没有连续三个相同字符，所以返回 "leetcode" 。
示例 2：输入：s = "aaabaaaa" 输出："aabaa"
解释：从第一组 'a' 里面删除一个 'a' ，得到 "aabaaaa" 。
从第二组 'a' 里面删除两个 'a' ，得到 "aabaa" 。
没有连续三个相同字符，所以返回 "aabaa" 。
示例 3：输入：s = "aab" 输出："aab"
解释：没有连续三个相同字符，所以返回 "aab" 。
提示：1 <= s.length <= 105
s只包含小写英文字母。
```

- 解题思路

```
func makeFancyString(s string) string {
        for i := 'a'; i <= 'z'; i++ {
                for strings.Contains(s, strings.Repeat(string(i), 3)) == true {
                        s = strings.ReplaceAll(s, strings.Repeat(string(i), 3),␣
↪strings.Repeat(string(i), 2))
                }
        }
        return s
}

# 2
```

(续下页)

(接上页)

```
func makeFancyString(s string) string {
        res := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                if len(res) >= 2 && res[len(res)-1] == s[i] && res[len(res)-2] ==␣
↪s[i] {
                        continue
                }
                res = append(res, s[i])
        }
        return string(res)
}
```

58.12 1961. 检查字符串是否为数组前缀 (1)

- 题目

```
给你一个字符串 s 和一个字符串数组 words ，请你判断 s 是否为 words 的 前缀字符串 。
字符串 s 要成为 words 的 前缀字符串 ，需要满足：s 可以由 words 中的前 k (k 为 正数␣
↪) 个字符串按顺序相连得到，
且 k 不超过 words.length 。
如果 s 是 words 的 前缀字符串 ，返回 true ；否则，返回 false 。
示例 1：输入：s = "iloveleetcode", words = ["i","love","leetcode","apples"] 输出：true
解释：s 可以由 "i"、"love" 和 "leetcode" 相连得到。
示例 2：输入：s = "iloveleetcode", words = ["apples","i","love","leetcode"]␣
↪输出：false
解释：  数组的前缀相连无法得到 s 。
提示：1 <= words.length <= 100
1 <= words[i].length <= 20
1 <= s.length <= 1000
words[i] 和 s 仅由小写英文字母组成
```

- 解题思路

```
func isPrefixString(s string, words []string) bool {
        temp := ""
        for i := 0; i < len(words); i++ {
                temp = temp + words[i]
                if temp == s {
                        return true
                }
        }
        return false
```

(续下页)

(接上页)

```
}
```

58.13 1967. 作为子字符串出现在单词中的字符串数目 (1)

- 题目

```
给你一个字符串数组 patterns 和一个字符串 word ，统计 patterns 中有多少个字符串是 word␣
↪的子字符串。返回字符串数目。
子字符串 是字符串中的一个连续字符序列。
示例 1：输入：patterns = ["a","abc","bc","d"], word = "abc" 输出：3
解释：- "a" 是 "abc" 的子字符串。
- "abc" 是 "abc" 的子字符串。
- "bc" 是 "abc" 的子字符串。
- "d" 不是 "abc" 的子字符串。
patterns 中有 3 个字符串作为子字符串出现在 word 中。
示例 2：输入：patterns = ["a","b","c"], word = "aaaaabbbbb" 输出：2
解释：- "a" 是 "aaaaabbbbb" 的子字符串。
- "b" 是 "aaaaabbbbb" 的子字符串。
- "c" 不是 "aaaaabbbbb" 的字符串。
patterns 中有 2 个字符串作为子字符串出现在 word 中。
示例 3：输入：patterns = ["a","a","a"], word = "ab" 输出：3
解释：patterns 中的每个字符串都作为子字符串出现在 word "ab" 中。
提示：1 <= patterns.length <= 100
1 <= patterns[i].length <= 100
1 <= word.length <= 100
patterns[i] 和 word 由小写英文字母组成
```

- 解题思路

```
func numOfStrings(patterns []string, word string) int {
        res := 0
        for i := 0; i < len(patterns); i++ {
                if strings.Contains(word, patterns[i]) == true {
                        res++
                }
        }
        return res
}
```

58.14 1974. 使用特殊打字机键入单词的最少时间 (2)

- 题目

```
有一个特殊打字机，它由一个 圆盘 和一个 指针组成， 圆盘上标有小写英文字母'a' 到'z'。
只有当指针指向某个字母时，它才能被键入。指针 初始时指向字符 'a'。
每一秒钟，你可以执行以下操作之一：
将指针 顺时针或者 逆时针移动一个字符。
键入指针 当前指向的字符。
给你一个字符串word，请你返回键入word所表示单词的 最少秒数。
示例 1：输入：word = "abc" 输出：5
解释：单词按如下操作键入：
- 花 1 秒键入字符 'a' in 1 ，因为指针初始指向 'a' ，故不需移动指针。
- 花 1 秒将指针顺时针移到 'b' 。
- 花 1 秒键入字符 'b' 。
- 花 1 秒将指针顺时针移到 'c' 。
- 花 1 秒键入字符 'c' 。
示例 2：输入：word = "bza" 输出：7
解释：单词按如下操作键入：
- 花 1 秒将指针顺时针移到 'b' 。
- 花 1 秒键入字符 'b' 。
- 花 2 秒将指针逆时针移到 'z' 。
- 花 1 秒键入字符 'z' 。
- 花 1 秒将指针顺时针移到 'a' 。
- 花 1 秒键入字符 'a' 。
示例 3：输入：word = "zjpc" 输出：34
解释：单词按如下操作键入：
- 花 1 秒将指针逆时针移到 'z' 。
- 花 1 秒键入字符 'z' 。
- 花 10 秒将指针顺时针移到 'j' 。
- 花 1 秒键入字符 'j' 。
- 花 6 秒将指针顺时针移到 'p' 。
- 花 1 秒键入字符 'p' 。
- 花 13 秒将指针逆时针移到 'c' 。
- 花 1 秒键入字符 'c' 。
提示：1 <= word.length <= 100
word只包含小写英文字母。
```

- 解题思路

```
func minTimeToType(word string) int {
        res := 0
        cur := 0
        for i := 0; i < len(word); i++ {
```

(续下页)

(接上页)

```
                target := int(word[i] - 'a')
                left := (cur - target + 26) % 26
                right := (target - cur + 26) % 26
                res = res + 1 + min(left, right)
                cur = target
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minTimeToType(word string) int {
        res := 0
        cur := 0
        for i := 0; i < len(word); i++ {
                target := abs(int(word[i]-'a') - cur)
                res = res + 1 + min(target, 26-target)
                cur = int(word[i]-'a')
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

58.15 1979. 找出数组的最大公约数 (2)

- 题目

```
给你一个整数数组 nums ，返回数组中最大数和最小数的 最大公约数 。
两个数的最大公约数 是能够被两个数整除的最大正整数。
示例 1：输入：nums = [2,5,6,9,10] 输出：2
解释：nums 中最小的数是 2
nums 中最大的数是 10
2 和 10 的最大公约数是 2
示例 2：输入：nums = [7,5,6,8,3] 输出：1
解释：nums 中最小的数是 3
nums 中最大的数是 8
3 和 8 的最大公约数是 1
示例 3：输入：nums = [3,3] 输出：3
解释：nums 中最小的数是 3
nums 中最大的数是 3
3 和 3 的最大公约数是 3
提示：2 <= nums.length <= 1000
1 <= nums[i] <= 1000
```

- 解题思路

```
func findGCD(nums []int) int {
        sort.Ints(nums)
        a, b := nums[0], nums[len(nums)-1]
        return gcd(a, b)
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}

# 2
func findGCD(nums []int) int {
        a, b := nums[0], nums[0]
        for i := 0; i < len(nums); i++ {
                if nums[i] > a {
                        a = nums[i]
                }
                if nums[i] < b {
```

(续下页)

(接上页)

```
                        b = nums[i]
                }
        }
        return gcd(a, b)
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}
```

58.16 1984. 学生分数的最小差值 (1)

- 题目

```
给你一个 下标从 0 开始 的整数数组 nums ，其中 nums[i] 表示第 i
↪名学生的分数。另给你一个整数 k 。
从数组中选出任意 k 名学生的分数，使这 k 个分数间 最高分 和 最低分 的 差值 达到 最小化
↪。
返回可能的 最小差值 。
示例 1：输入：nums = [90], k = 1 输出：0
解释：选出 1 名学生的分数，仅有 1 种方法：
- [90] 最高分和最低分之间的差值是 90 - 90 = 0
可能的最小差值是 0
示例 2：输入：nums = [9,4,1,7], k = 2 输出：2
解释：选出 2 名学生的分数，有 6 种方法：
- [9,4,1,7] 最高分和最低分之间的差值是 9 - 4 = 5
- [9,4,1,7] 最高分和最低分之间的差值是 9 - 1 = 8
- [9,4,1,7] 最高分和最低分之间的差值是 9 - 7 = 2
- [9,4,1,7] 最高分和最低分之间的差值是 4 - 1 = 3
- [9,4,1,7] 最高分和最低分之间的差值是 7 - 4 = 3
- [9,4,1,7] 最高分和最低分之间的差值是 7 - 1 = 6
可能的最小差值是 2
提示：1 <= k <= nums.length <= 1000
0 <= nums[i] <= 105
```

- 解题思路

```
func minimumDifference(nums []int, k int) int {
        sort.Ints(nums)
```

(续下页)

(接上页)

```
        res := math.MaxInt32
        for i := 0; i <= len(nums)-k; i++ {
                left := nums[i]
                right := nums[i+k-1]
                res = min(res, right-left)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

58.17 1991. 找到数组的中间位置 (1)

- 题目

```
给你一个下标从 0开始的整数数组nums，请你找到最左边的中间位置middleIndex（也就是所有可能中间位置下标最小的一个）。
中间位置middleIndex是满足nums[0] + nums[1] + ... + nums[middleIndex-1] ==
nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1]的数组下标。
如果middleIndex == 0，左边部分的和定义为 0。类似的，如果middleIndex == nums.length -1，右边部分的和定义为0。
请你返回满足上述条件 最左边的middleIndex，如果不存在这样的中间位置，请你返回-1。
示例 1：输入：nums = [2,3,-1,8,4] 输出：3
解释：下标 3 之前的数字和为：2 + 3 + -1 = 4
下标 3 之后的数字和为：4 = 4
示例 2：输入：nums = [1,-1,4] 输出：2
解释：下标 2 之前的数字和为：1 + -1 = 0
下标 2 之后的数字和为：0
示例 3：输入：nums = [2,5]输出：-1
解释：不存在符合要求的 middleIndex 。
示例 4：输入：nums = [1] 输出：0
解释：下标 0 之前的数字和为：0
下标 0 之后的数字和为：0
提示：1 <= nums.length <= 100
-1000 <= nums[i] <= 1000
```

- 解题思路

```
func findMiddleIndex(nums []int) int {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        count := 0
        for i := 0; i < len(nums); i++ {
                if 2*count == sum-nums[i] {
                        return i
                }
                count = count + nums[i]
        }
        return -1
}
```

58.18 1995. 统计特殊四元组 (1)

- 题目

```
给你一个 下标从 0 开始 的整数数组 nums ，返回满足下述条件的 不同 四元组 (a, b, c, d)
↪的 数目 ：
nums[a] + nums[b] + nums[c] == nums[d] ，且 a < b < c < d
示例 1：输入：nums = [1,2,3,6] 输出：1
解释：满足要求的唯一一个四元组是 (0, 1, 2, 3) 因为 1 + 2 + 3 == 6 。
示例 2：输入：nums = [3,3,6,4,5] 输出：0
解释：[3,3,6,4,5] 中不存在满足要求的四元组。
示例 3：输入：nums = [1,1,1,3,5] 输出：4
解释：满足要求的 4 个四元组如下：
- (0, 1, 2, 3): 1 + 1 + 1 == 3
- (0, 1, 3, 4): 1 + 1 + 3 == 5
- (0, 2, 3, 4): 1 + 1 + 3 == 5
- (1, 2, 3, 4): 1 + 1 + 3 == 5
提示：4 <= nums.length <= 50
1 <= nums[i] <= 100
```

- 解题思路

```
func countQuadruplets(nums []int) int {
        res := 0
        n := len(nums)
        for i := 0; i < n-3; i++ {
                for j := i + 1; j < n-2; j++ {
                        for k := j + 1; k < n-1; k++ {
```

(续下页)

(接上页)

```
                                for l := k + 1; l < n; l++ {
                                        if nums[i]+nums[j]+nums[k] == nums[l] {
                                                res++
                                        }
                                }
                        }
                }
        }
        return res
}
```

58.19 2000. 反转单词前缀 (1)

- 题目

```
给你一个下标从 0 开始的字符串 word 和一个字符 ch 。找出 ch 第一次出现的下标 i ，
反转 word 中从下标 0 开始、直到下标 i 结束（含下标 i ）的那段字符。如果 word␣
↪中不存在字符 ch ，则无需进行任何操作。
例如，如果 word = "abcdefd" 且 ch = "d" ，那么你应该 反转 从下标 0 开始、直到下标 3␣
↪结束（含下标 3 ）。
结果字符串将会是 "dcbaefd" 。
返回 结果字符串 。
示例 1：输入：word = "abcdefd", ch = "d" 输出："dcbaefd"
解释："d" 第一次出现在下标 3 。
反转从下标 0 到下标 3（含下标 3）的这段字符，结果字符串是 "dcbaefd" 。
示例 2：输入：word = "xyxzxe", ch = "z" 输出："zxyxxe"
解释："z" 第一次也是唯一一次出现是在下标 3 。
反转从下标 0 到下标 3（含下标 3）的这段字符，结果字符串是 "zxyxxe" 。
示例 3：输入：word = "abcd", ch = "z" 输出："abcd"
解释："z" 不存在于 word 中。
无需执行反转操作，结果字符串是 "abcd" 。
提示：1 <= word.length <= 250
word 由小写英文字母组成
ch 是一个小写英文字母
```

- 解题思路

```
func reversePrefix(word string, ch byte) string {
        index := strings.Index(word, string(ch))
        arr := []byte(word)
        reverse(arr[:index+1])
        return string(arr)
```

(续下页)

(接上页)

```
}

func reverse(arr []byte) []byte {
        start := 0
        end := len(arr) - 1
        for start < end {
                arr[start], arr[end] = arr[end], arr[start]
                start++
                end--
        }
        return arr
}
```

CHAPTER 59

1901-2000-Medium

59.1 1901. 找出顶峰元素 II(3)

- 题目

```
一个 2D 网格中的 顶峰元素 是指那些 严格大于 其相邻格子(上、下、左、右)的元素。
给你一个 从 0 开始编号 的 m x n 矩阵 mat ，其中任意两个相邻格子的值都 不相同 。
找出 任意一个 顶峰元素 mat[i][j] 并 返回其位置 [i,j] 。
你可以假设整个矩阵周边环绕着一圈值为 -1 的格子。
要求必须写出时间复杂度为 O(m log(n)) 或 O(n log(m)) 的算法
示例 1:输入: mat = [[1,4],[3,2]] 输出: [0,1]
解释:3和4都是顶峰元素，所以[1,0]和[0,1]都是可接受的答案。
示例 2:输入: mat = [[10,20,15],[21,30,14],[7,16,32]] 输出: [1,1]
解释:30和32都是顶峰元素，所以[1,1]和[2,2]都是可接受的答案。
提示： m == mat.length
n == mat[i].length
1 <= m, n <= 500
1 <= mat[i][j] <= 105
任意两个相邻元素均不相等.
```

- 解题思路

```
func findPeakGrid(mat [][]int) []int {
        for i := 0; i < len(mat); i++ {
                for j := 0; j < len(mat[i]); j++ {
```

(续下页)

(接上页)

```
                        if (1 <= i && mat[i][j] < mat[i-1][j]) ||
                                (i < len(mat)-1 && mat[i][j] < mat[i+1][j]) ||
                                (1 <= j && mat[i][j] < mat[i][j-1]) ||
                                (j < len(mat[i])-1 && mat[i][j] < mat[i][j+1]) {
                                continue
                        }
                        return []int{i, j}
                }
        }
        return nil
}

# 2
func findPeakGrid(mat [][]int) []int {
        n, m := len(mat), len(mat[0])
        x, y := 0, 0
        for {
                if x < n-1 && mat[x][y] < mat[x+1][y] {
                        x++
                } else if 1 <= x && mat[x][y] < mat[x-1][y] {
                        x--
                } else if y < m-1 && mat[x][y] < mat[x][y+1] {
                        y++
                } else if 1 <= y && mat[x][y] < mat[x][y-1] {
                        y--
                } else {
                        return []int{x, y}
                }
        }
        return nil
}

# 3
func findPeakGrid(mat [][]int) []int {
        n := len(mat)
        left, right := 0, n-1
        for left <= right {
                mid := left + (right-left)/2
                midValue, index := getMax(mat[mid])
                a, b := -1, -1
                if mid >= 1 {
                        a, _ = getMax(mat[mid-1])
                }
```

(续下页)

(接上页)

```
                if mid <= n-2 {
                        b, _ = getMax(mat[mid+1])
                }
                res, _ := getMax([]int{a, b, midValue})
                if res == midValue {
                        return []int{mid, index}
                } else if res == a {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return nil
}

func getMax(arr []int) (int, int) {
        res := arr[0]
        index := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] > res {
                        index = i
                        res = arr[i]
                }
        }
        return res, index
}
```

59.2 1904. 你完成的完整对局数 (1)

- 题目

```
一款新的在线电子游戏在近期发布，在该电子游戏中，以 刻钟 为周期规划若干时长为 15 分钟的游戏对局。
这意味着，在 HH:00、HH:15、HH:30 和 HH:45 ，将会开始一个新的对局，其中 HH 用一个从 00 到 23 的整数表示。
游戏中使用 24 小时制的时钟 ，所以一天中最早的时间是 00:00，最晚的时间是 23:59 。
给你两个字符串 startTime 和 finishTime ，均符合 "HH:MM" 格式，分别表示你 进入 和 退出游戏的确切时间，
请计算在整个游戏会话期间，你完成的 完整对局的对局数 。
例如，如果 startTime = "05:20" 且 finishTime = "05:59" ，这意味着你仅仅完成从 05:30 到 05:45 这一个完整对局。
而你没有完成从 05:15 到 05:30 的完整对局，因为你是在对局开始后进入的游戏；
```

(续下页)

(接上页)

```
同时，你也没有完成从 05:45 到 06:00 的完整对局，因为你是在对局结束前退出的游戏。
如果 finishTime 早于 startTime ，这表示你玩了个通宵（也就是从 startTime␣
↪到午夜，再从午夜到 finishTime）。
假设你是从startTime 进入游戏，并在finishTime 退出游戏，请计算并返回你完成的␣
↪完整对局的对局数 。
示例 1：输入：startTime = "12:01", finishTime = "12:44" 输出：1
解释：你完成了从 12:15 到 12:30 的一个完整对局。
你没有完成从 12:00 到 12:15 的完整对局，因为你是在对局开始后的 12:01 进入的游戏。
你没有完成从 12:30 到 12:45 的完整对局，因为你是在对局结束前的 12:44 退出的游戏。
示例 2：输入：startTime = "20:00", finishTime = "06:00" 输出：40
解释：你完成了从 20:00 到 00:00 的 16 个完整的对局，以及从 00:00 到 06:00 的 24␣
↪个完整的对局。
16 + 24 = 40
示例 3：输入：startTime = "00:00", finishTime = "23:59" 输出：95
解释：除最后一个小时你只完成了 3 个完整对局外，其余每个小时均完成了 4 场完整对局。
提示：startTime 和 finishTime 的格式为 HH:MM
00 <= HH <= 23
00 <= MM <= 59
startTime 和 finishTime 不相等
```

- 解题思路

```
func numberOfRounds(startTime string, finishTime string) int {
        a, _ := strconv.Atoi(startTime[:2])
        b, _ := strconv.Atoi(startTime[3:])
        c, _ := strconv.Atoi(finishTime[:2])
        d, _ := strconv.Atoi(finishTime[3:])
        t1 := 60*a + b
        t2 := 60*c + d
        if t2 < t1 {
                t2 = t2 + 1440
        }
        start := int(math.Ceil(float64(t1) / float64(15)))   // 向上取整
        finish := int(math.Floor(float64(t2) / float64(15))) // 向下取整
        if finish - start >= 0{
                return finish-start
        }
        return 0 // 00:47 ~00:48
}
```

59.3 1905. 统计子岛屿 (2)

- 题目

```
给你两个m x n的二进制矩阵grid1 和grid2，它们只包含0（表示水域）和 1（表示陆地）。
一个 岛屿是由⌴
↪四个方向（水平或者竖直）上相邻的1组成的区域。任何矩阵以外的区域都视为水域。
如果 grid2的一个岛屿，被 grid1的一个岛屿完全 包含，
也就是说 grid2中该岛屿的每一个格子都被 grid1中同一个岛屿完全包含，那么我们称⌴
↪grid2中的这个岛屿为 子岛屿。
请你返回 grid2中 子岛屿的 数目。
示例 1：输入：grid1 = [[1,1,1,0,0],[0,1,1,1,1],[0,0,0,0,0],[1,0,0,0,0],[1,1,0,1,1]],
grid2 = [[1,1,1,0,0],[0,0,1,1,1],[0,1,0,0,0],[1,0,1,1,0],[0,1,0,1,0]]
输出：3
解释：如上图所示，左边为 grid1 ，右边为 grid2 。
grid2 中标红的 1 区域是子岛屿，总共有 3 个子岛屿。
示例 2：输入：grid1 = [[1,0,1,0,1],[1,1,1,1,1],[0,0,0,0,0],[1,1,1,1,1],[1,0,1,0,1]],
grid2 = [[0,0,0,0,0],[1,1,1,1,1],[0,1,0,1,0],[0,1,0,1,0],[1,0,0,0,1]]
输出：2
解释：如上图所示，左边为 grid1 ，右边为 grid2 。
grid2 中标红的 1 区域是子岛屿，总共有 2 个子岛屿。
提示：m == grid1.length == grid2.length
n == grid1[i].length == grid2[i].length
1 <= m, n <= 500
grid1[i][j] 和grid2[i][j]都要么是0要么是1。
```

- 解题思路

```
func countSubIslands(grid1 [][]int, grid2 [][]int) int {
        res := 0
        for i := 0; i < len(grid2); i++ {
                for j := 0; j < len(grid2[i]); j++ {
                        if grid2[i][j] == 1 { // 查找grid2
                                if dfs(grid1, grid2, i, j) == true {
                                        res++
                                }
                        }
                }
        }
        return res
}

// leetcode200.岛屿数量
func dfs(grid1, grid2 [][]int, i, j int) bool {
```

(续下页)

(接上页)

```
        if i < 0 || j < 0 || i >= len(grid2) || j >= len(grid2[0]) || grid2[i][j] ==␣
↪0 {
                return true
        }
        if grid1[i][j] == 0 {
                return false
        }
        grid1[i][j], grid2[i][j] = 0, 0
        res1 := dfs(grid1, grid2, i+1, j)
        res2 := dfs(grid1, grid2, i-1, j)
        res3 := dfs(grid1, grid2, i, j+1)
        res4 := dfs(grid1, grid2, i, j-1)
        if res1 == false || res2 == false || res3 == false || res4 == false {
                return false
        }
        return true
}

# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func countSubIslands(grid1 [][]int, grid2 [][]int) int {
        res := 0
        for i := 0; i < len(grid2); i++ {
                for j := 0; j < len(grid2[i]); j++ {
                        if grid2[i][j] == 1 { // 查找grid2
                                if bfs(grid1, grid2, i, j) == true {
                                        res++
                                }
                        }
                }
        }
        return res
}

// leetcode200.岛屿数量
func bfs(grid1, grid2 [][]int, i, j int) bool {
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{i, j})
        flag := true
        if grid1[i][j] == 0 {
                flag = false
```

(续下页)

(接上页)

```
        }
        grid1[i][j] = 0
        grid2[i][j] = 0
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                for i := 0; i < 4; i++ {
                        x := node[0] + dx[i]
                        y := node[1] + dy[i]
                        if 0 <= x && x < len(grid2) &&
                                0 <= y && y < len(grid2[0]) &&
                                grid2[x][y] == 1 {
                                queue = append(queue, [2]int{x, y})
                                if grid1[x][y] == 0 {
                                        flag = false
                                }
                                grid1[x][y] = 0
                                grid2[x][y] = 0
                        }
                }
        }
        return flag
}
```

59.4 1906. 查询差绝对值的最小值 (2)

- 题目

```
一个数组 a的 差绝对值的最小值定义为：0 <= i < j < a.length且 a[i] != a[j]的 |a[i] -␣
↪a[j]| 的 最小值。
如果 a中所有元素都 相同，那么差绝对值的最小值为 -1。
比方说，数组[5,2,3,7,2]差绝对值的最小值是|2 - 3| = 1。注意答案不为 0，因为a[i]␣
↪和a[j]必须不相等。
给你一个整数数组nums和查询数组queries，其中queries[i] = [li, ri]。
对于每个查询i，计算子数组nums[li...ri]中 差绝对值的最小值 ，
子数组nums[li...ri] 包含 nums数组（下标从 0开始）中下标在 li 和ri␣
↪之间的所有元素（包含li 和ri 在内）。
请你返回ans数组，其中 ans[i]是第 i个查询的答案。
子数组是一个数组中连续的一段元素。
|x|的值定义为：
如果x >= 0，那么值为x。
如果x < 0，那么值为-x。
```

(续下页)

(接上页)

```
示例 1：输入：nums = [1,3,4,8], queries = [[0,1],[1,2],[2,3],[0,3]] 输出：[2,1,4,1]
解释：查询结果如下：
- queries[0] = [0,1]：子数组是 [1,3] ，差绝对值的最小值为 |1-3| = 2 。
- queries[1] = [1,2]：子数组是 [3,4] ，差绝对值的最小值为 |3-4| = 1 。
- queries[2] = [2,3]：子数组是 [4,8] ，差绝对值的最小值为 |4-8| = 4 。
- queries[3] = [0,3]：子数组是 [1,3,4,8] ，差的绝对值的最小值为 |3-4| = 1 。
示例 2：输入：nums = [4,5,2,2,7,10], queries = [[2,3],[0,2],[0,5],[3,5]]
输出：[-1,1,1,3]
解释：查询结果如下：
- queries[0] = [2,3]：子数组是 [2,2] ，差绝对值的最小值为 -1 ，因为所有元素相等。
- queries[1] = [0,2]：子数组是 [4,5,2] ，差绝对值的最小值为 |4-5| = 1 。
- queries[2] = [0,5]：子数组是 [4,5,2,2,7,10] ，差绝对值的最小值为 |4-5| = 1 。
- queries[3] = [3,5]：子数组是 [2,7,10] ，差绝对值的最小值为 |7-10| = 3 。
提示：2 <= nums.length <= 105
1 <= nums[i] <= 100
1 <= queries.length <= 2* 104
0 <= li < ri < nums.length
```

- 解题思路

```
func minDifference(nums []int, queries [][]int) []int {
        n := len(nums)
        arr := make([][101]int, n+1)
        for i := 1; i <= n; i++ {
                arr[i] = arr[i-1]
                arr[i][nums[i-1]]++
        }
        res := make([]int, len(queries))
        for i := 0; i < len(queries); i++ {
                res[i] = -1
                left, right := queries[i][0], queries[i][1]
                prev, minValue := 0, math.MaxInt32
                for j := 1; j <= 100; j++ { // 枚举每个数
                        if arr[right+1][j] != arr[left][j] { // 当j出现
                                if prev != 0 {
                                        if j-prev < minValue { // 保存较小的差值
                                                minValue = j - prev
                                        }
                                }
                                prev = j // 更新上一个出现的数
                        }
                }
                if minValue != math.MaxInt32 {
                        res[i] = minValue
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 2
func minDifference(nums []int, queries [][]int) []int {
        n := len(nums)
        length = n
        c = make([][]int, 101) // 存储每个数对应顺序区间出现的次数
        for i := 0; i < 101; i++ {
                c[i] = make([]int, n+1)
        }
        for i := 1; i <= n; i++ {
                upData(nums[i-1], i, 1)
        }
        res := make([]int, len(queries))
        for i := 0; i < len(queries); i++ {
                res[i] = -1
                left, right := queries[i][0], queries[i][1]
                prev, minValue := 0, math.MaxInt32
                for j := 1; j <= 100; j++ { // 枚举每个数
                        count := getSum(j, right+1) - getSum(j, left)
                        if count > 0 { // 当j出现
                                if prev != 0 {
                                        if j-prev < minValue { // 保存较小的差值
                                                minValue = j - prev
                                        }
                                }
                                prev = j // 更新上一个出现的数
                        }
                }
                if minValue != math.MaxInt32 {
                        res[i] = minValue
                }
        }
        return res
}

var length int
var c [][]int // 树状数组

func lowBit(x int) int {
```

(续下页)

(接上页)

```
        return x & (-x)
}

// 单点修改
func upData(index, i, k int) { // 在i位置加上k
        for i <= length {
                c[index][i] = c[index][i] + k
                i = i + lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func getSum(index, i int) int {
        res := 0
        for i > 0 {
                res = res + c[index][i]
                i = i - lowBit(i)
        }
        return res
}
```

59.5 1910. 删除一个字符串中所有出现的给定子字符串 (2)

- 题目

```
给你两个字符串s和part，请你对s反复执行以下操作直到 所有子字符串part都被删除：
找到 s中 最左边的子字符串 part，并将它从 s中删除。
请你返回从 s中删除所有 part子字符串以后得到的剩余字符串。
一个 子字符串是一个字符串中连续的字符序列。
示例 1：输入：s = "daabcbaabcbc", part = "abc" 输出："dab"
解释：以下操作按顺序执行：
- s = "daabcbaabcbc" ，删除下标从 2 开始的 "abc" ，得到 s = "dabaabcbc" 。
- s = "dabaabcbc" ，删除下标从 4 开始的 "abc" ，得到 s = "dababc" 。
- s = "dababc" ，删除下标从 3 开始的 "abc" ，得到 s = "dab" 。
此时 s 中不再含有子字符串 "abc" 。
示例 2：输入：s = "axxxxyyyyb", part = "xy" 输出："ab"
解释：以下操作按顺序执行：
- s = "axxxxyyyyb" ，删除下标从 4 开始的 "xy" ，得到 s = "axxxyyyb" 。
- s = "axxxyyyb" ，删除下标从 3 开始的 "xy" ，得到 s = "axxyyb" 。
- s = "axxyyb" ，删除下标从 2 开始的 "xy" ，得到 s = "axyb" 。
- s = "axyb" ，删除下标从 1 开始的 "xy" ，得到 s = "ab" 。
此时 s 中不再含有子字符串 "xy" 。
```

(续下页)

(接上页)

```
提示：1 <= s.length <= 1000
1 <= part.length <= 1000
s和part只包小写英文字母。
```

- 解题思路

```
func removeOccurrences(s string, part string) string {
        for strings.Contains(s, part) {
                s = strings.Replace(s, part, "", 1)
        }
        return s
}

# 2
func removeOccurrences(s string, part string) string {
        for {
                index := strings.Index(s, part)
                if index < 0 {
                        return s
                }
                s = s[:index] + s[index+len(part):]
        }
        return s
}
```

59.6 1911. 最大子序列交替和 (3)

- 题目

```
一个下标从 0开始的数组的 交替和定义为 偶数下标处元素之 和减去 奇数下标处元素之 和。
比方说，数组[4,2,5,3]的交替和为(4 + 5) - (2 + 3) = 4。
给你一个数组nums，请你返回nums中任意子序列的最大交替和（子序列的下标 重新从 0␣
↪开始编号）。
一个数组的␣
↪子序列是从原数组中删除一些元素后（也可能一个也不删除）剩余元素不改变顺序组成的数组。
比方说，[2,7,4]是[4,2,3,7,2,1,4]的一个子序列（加粗元素），但是[2,4,2] 不是。
示例 1：输入：nums = [4,2,5,3] 输出：7
解释：最优子序列为 [4,2,5] ，交替和为 (4 + 5) - 2 = 7 。
示例 2：输入：nums = [5,6,7,8] 输出：8
解释：最优子序列为 [8] ，交替和为 8 。
示例 3：输入：nums = [6,2,1,2,4,5] 输出：10
解释：最优子序列为 [6,1,5] ，交替和为 (6 + 5) - 1 = 10 。
```

(续下页)

(接上页)

```
提示：1 <= nums.length <= 105
1 <= nums[i] <= 105
```

- 解题思路

```
func maxAlternatingSum(nums []int) int64 {
        n := len(nums)
        odd, even := 0, nums[0]
        for i := 1; i < n; i++ {
                odd, even = max(even-nums[i], odd), max(odd+nums[i], even)
        }
        return int64(even)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxAlternatingSum(nums []int) int64 {
        n := len(nums)
        dp := make([][2]int, n)
        dp[0][0] = int(nums[0])
        dp[0][1] = 0
        for i := 1; i < n; i++ {
                dp[i][0] = max(dp[i-1][0], dp[i-1][1]+nums[i])
                dp[i][1] = max(dp[i-1][1], dp[i][0]-nums[i])
        }
        return int64(max(dp[n-1][0], dp[n-1][1]))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxAlternatingSum(nums []int) int64 {
        // leetcode122.买卖股票的最佳时机II
```

(续下页)

(接上页)

```
        nums = append([]int{0}, nums...) // 补零
        res := 0
        for i := 1; i < len(nums); i++ {
                if nums[i] > nums[i-1] {
                        res = res + nums[i] - nums[i-1]
                }
        }
        return int64(res)
}
```

59.7 1914. 循环轮转矩阵 (1)

- 题目

```
给你一个大小为 m x n 的整数矩阵 grid，其中 m 和 n 都是 偶数 ；另给你一个整数 k 。
矩阵由若干层组成，如下图所示，每种颜色代表一层：
矩阵的循环轮转是通过分别循环轮转矩阵中的每一层完成的。
在对某一层进行一次循环旋转操作时，层中的每一个元素将会取代其逆时针␣
↪方向的相邻元素。轮转示例如下：
返回执行 k 次循环轮转操作后的矩阵。
示例 1：输入：grid = [[40,10],[30,20]], k = 1 输出：[[10,20],[40,30]]
解释：上图展示了矩阵在执行循环轮转操作时每一步的状态。
示例 2：输入：grid = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]], k = 2
输出：[[3,4,8,12],[2,11,10,16],[1,7,6,15],[5,9,13,14]]
解释：上图展示了矩阵在执行循环轮转操作时每一步的状态。
提示：m == grid.length
n == grid[i].length
2 <= m, n <= 50
m 和 n 都是 偶数
1 <= grid[i][j] <= 5000
1 <= k <= 109
```

- 解题思路

```
func rotateGrid(grid [][]int, k int) [][]int {
        n, m := len(grid), len(grid[0])
        count := min(n/2, m/2)
        for level := 0; level < count; level++ {
                arr := make([][3]int, 0)
                for i := level; i < n-1-level; i++ { // 左上=>左下
                        arr = append(arr, [3]int{i, level, grid[i][level]})
                }
```

(续下页)

(接上页)

```
                for j := level; j < m-1-level; j++ { // 左下=>右下
                        arr = append(arr, [3]int{n - 1 - level, j, grid[n-1-level][j]}
↪)
                }
                for i := n - 1 - level; i > level; i-- { // 右下=>右上
                        arr = append(arr, [3]int{i, m - 1 - level, grid[i][m-1-level]}
↪)
                }
                for j := m - 1 - level; j > level; j-- { // 右上=>左上
                        arr = append(arr, [3]int{level, j, grid[level][j]})
                }
                total := len(arr)
                step := k % total
                for i := 0; i < total; i++ {
                        index := (i + total - step) % total
                        a, b, c := arr[i][0], arr[i][1], arr[index][2]
                        grid[a][b] = c
                }
        }
        return grid
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

59.8 1915. 最美子字符串的数目 (1)

- 题目

```
如果某个字符串中 至多一个 字母出现 奇数 次，则称其为 最美 字符串。
例如，"ccjjc" 和 "abab" 都是最美字符串，但 "ab" 不是。
给你一个字符串 word ，该字符串由前十个小写英文字母组成 ('a' 到 'j') 。
请你返回 word 中 最美非空子字符串 的数目。如果同样的子字符串在 word␣
↪中出现多次，那么应当对 每次出现 分别计数。
子字符串 是字符串中的一个连续字符序列。
示例 1: 输入: word = "aba" 输出: 4
解释: 4 个最美子字符串如下所示:
- "aba" -> "a"
```

(续下页)

(接上页)

```
- "aba" -> "b"
- "aba" -> "a"
- "aba" -> "aba"
示例 2: 输入: word = "aabb" 输出: 9
解释: 9 个最美子字符串如下所示:
- "aabb" -> "a"
- "aabb" -> "aa"
- "aabb" -> "aab"
- "aabb" -> "aabb"
- "aabb" -> "a"
- "aabb" -> "abb"
- "aabb" -> "b"
- "aabb" -> "bb"
- "aabb" -> "b"
示例 3: 输入: word = "he" 输出: 2
解释: 2 个最美子字符串如下所示:
- "he" -> "h"
- "he" -> "e"
提示: 1 <= word.length <= 105
word 由从 'a' 到 'j' 的小写英文字母组成
```

- 解题思路

```
func wonderfulSubstrings(word string) int64 {
        res := 0
        m := make(map[int]int)
        m[0] = 1 // 初始状态没有任何字符，相当于全是偶数
        cur := 0
        for i := 0; i < len(word); i++ {
                value := int(word[i] - 'a')
                cur = cur ^ (1 << value) // 位运算-
↪异或后的状态结果，表示到当前下标各字母出现的奇偶次数
                res = res + m[cur]       // 跟当前状态相同
                for i := 0; i < 10; i++ {
                        count := m[cur^(1<<i)] // 跟当前状态差1位相同
                        res = res + count
                }
                m[cur]++
        }
        return int64(res)
}
```

59.9 1921. 消灭怪物的最大数量 (1)

- 题目

```
你正在玩一款电子游戏，在游戏中你需要保护城市免受怪物侵袭。
给你一个 下标从 0 开始 且长度为 n 的整数数组 dist ，其中 dist[i] 是第 i␣
↪个怪物与城市的 初始距离（单位：米）。
怪物以 恒定 的速度走向城市。
给你一个长度为 n 的整数数组 speed 表示每个怪物的速度，其中 speed[i] 是第 i␣
↪个怪物的速度（单位：米/分）。
怪物从 第 0 分钟 时开始移动。你有一把武器，并可以 选择 在每一分钟的开始时使用，包括第␣
↪0 分钟。
但是你无法在一分钟的中间使用武器。这种武器威力惊人，一次可以消灭任一还活着的怪物。
一旦任一怪物到达城市，你就输掉了这场游戏。
如果某个怪物 恰 在某一分钟开始时到达城市，这会被视为␣
↪输掉游戏，在你可以使用武器之前，游戏就会结束。
返回在你输掉游戏前可以消灭的怪物的 最大␣
↪数量。如果你可以在所有怪物到达城市前将它们全部消灭，返回 n 。
示例 1：输入：dist = [1,3,4], speed = [1,1,1] 输出：3
解释：第 0 分钟开始时，怪物的距离是 [1,3,4]，你消灭了第一个怪物。
第 1 分钟开始时，怪物的距离是 [X,2,3]，你没有消灭任何怪物。
第 2 分钟开始时，怪物的距离是 [X,1,2]，你消灭了第二个怪物。
第 3 分钟开始时，怪物的距离是 [X,X,1]，你消灭了第三个怪物。
所有 3 个怪物都可以被消灭。
示例 2：输入：dist = [1,1,2,3], speed = [1,1,1,1] 输出：1
解释：第 0 分钟开始时，怪物的距离是 [1,1,2,3]，你消灭了第一个怪物。
第 1 分钟开始时，怪物的距离是 [X,0,1,2]，你输掉了游戏。
你只能消灭 1 个怪物。
示例 3：输入：dist = [3,2,4], speed = [5,3,2] 输出：1
解释：第 0 分钟开始时，怪物的距离是 [3,2,4]，你消灭了第一个怪物。
第 1 分钟开始时，怪物的距离是 [X,0,2]，你输掉了游戏。
你只能消灭 1 个怪物。
提示：n == dist.length == speed.length
1 <= n <= 105
1 <= dist[i], speed[i] <= 105
```

- 解题思路

```
func eliminateMaximum(dist []int, speed []int) int {
        res := 0
        temp := make([]float64, len(dist))
        for i := 0; i < len(dist); i++ {
                temp[i] = float64(dist[i]) / float64(speed[i])
        }
```

(续下页)

(接上页)

```
        sort.Slice(temp, func(i, j int) bool {
                return temp[i] < temp[j]
        })
        for i := 0; i < len(dist); i++ {
                if float64(i) < temp[i] {
                        res++
                } else {
                        break
                }
        }
        return res
}
```

59.10 1922. 统计好数字的数目 (2)

- 题目

```
我们称一个数字字符串是 好数字 当它满足（下标从 0开始）偶数 下标处的数字为 偶数且␣
↪奇数下标处的数字为 质数（2，3，5或7）。
比方说，"2582"是好数字，因为偶数下标处的数字（2和8）是偶数且奇数下标处的数字（5␣
↪和2）为质数。
但"3245"不是 好数字，因为3在偶数下标处但不是偶数。
给你一个整数n，请你返回长度为n且为好数字的数字字符串总数。由于答案可能会很大，请你将它对109␣
↪+ 7取余后返回。
一个 数字字符串是每一位都由0到 9组成的字符串，且可能包含前导 0 。
示例 1：输入：n = 1 输出：5
解释：长度为 1 的好数字包括 "0"，"2"，"4"，"6"，"8" 。
示例 2：输入：n = 4 输出：400
示例 3：输入：n = 50 输出：564908303
提示：1 <= n <= 1015
```

- 解题思路

```
var mod = int64(1000000007)

func countGoodNumbers(n int64) int {
        res := mypow(20, n/2)
        if n%2 == 1 {
                res = res * 5 % mod
        }
        return int(res)
}
```

(续下页)

(接上页)

```
func mypow(a int64, n int64) int64 {
        var res = int64(1)
        for n > 0 {
                if n%2 == 1 {
                        res = res * a % mod
                }
                a = a * a % mod
                n = n / 2
        }
        return res
}

# 2
var mod = int64(1000000007)

func countGoodNumbers(n int64) int {
        return int(mypow(5, (n+1)/2) * mypow(4, n/2) % mod)
}

func mypow(a int64, n int64) int64 {
        var res = int64(1)
        for n > 0 {
                if n%2 == 1 {
                        res = res * a % mod
                }
                a = a * a % mod
                n = n / 2
        }
        return res
}
```

59.11 1926. 迷宫中离入口最近的出口 (1)

- 题目

```
给你一个m x n的迷宫矩阵maze（下标从 0 开始），矩阵中有空格子（用'.'表示）和墙（用'+
↪'表示）。
同时给你迷宫的入口entrance，用entrance = [entrancerow,␣
↪entrancecol]表示你一开始所在格子的行和列。
每一步操作，你可以往 上，下，左 或者␣
↪右移动一个格子。你不能进入墙所在的格子，你也不能离开迷宫。
```

(续下页)

(接上页)

```
你的目标是找到离entrance最近的出口。出口的含义是maze边界上的空格子。entrance格子不算出口。
请你返回从 entrance到最近出口的最短路径的 步数，如果不存在这样的路径，请你返回 -1。
示例 1：输入：maze = [["+","+",".","+"],[".",".",".","+"],["+","+","+","."]],
↪entrance = [1,2] 输出：1
解释：总共有 3 个出口，分别位于 (1,0)，(0,2) 和 (2,3) 。
一开始，你在入口格子 (1,2) 处。
- 你可以往左移动 2 步到达 (1,0) 。
- 你可以往上移动 1 步到达 (0,2) 。
从入口处没法到达 (2,3) 。
所以，最近的出口是 (0,2) ，距离为 1 步。
示例 2：输入：maze = [["+","+","+"],[".",".","."],["+","+","+"]], entrance = [1,0]
↪输出：2
解释：迷宫中只有 1 个出口，在 (1,2) 处。
(1,0) 不算出口，因为它是入口格子。
初始时，你在入口与格子 (1,0) 处。
- 你可以往右移动 2 步到达 (1,2) 处。
所以，最近的出口为 (1,2) ，距离为 2 步。
示例 3：输入：maze = [[".","+"]], entrance = [0,0] 输出：-1
解释：这个迷宫中没有出口。
提示：maze.length == m
maze[i].length == n
1 <= m, n <= 100
maze[i][j] 要么是'.'，要么是'+'。
entrance.length == 2
0 <= entrancerow < m
0 <= entrancecol < n
entrance一定是空格子。
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func nearestExit(maze [][]byte, entrance []int) int {
        n, m := len(maze), len(maze[0])
        queue := make([][2]int, 0)
        visited := make(map[[2]int]bool)
        queue = append(queue, [2]int{entrance[0], entrance[1]})
        visited[[2]int{entrance[0], entrance[1]}] = true
        count := 0
        for len(queue) > 0 {
                count++
                length := len(queue)
                for i := 0; i < length; i++ {
```

(续下页)

(接上页)

```
                        a, b := queue[i][0], queue[i][1]
                        for j := 0; j < 4; j++ {
                                x := a + dx[j]
                                y := b + dy[j]
                                if 0 <= x && x < n && 0 <= y && y < m &&
                                        maze[x][y] != '+' && visited[[2]int{x, y}] ==␣
↪false {
                                        if (x == 0 || x == n-1 || y == 0 || y == m-1)␣
↪&& maze[x][y] == '.' {
                                                return count
                                        }
                                        queue = append(queue, [2]int{x, y})
                                        visited[[2]int{x, y}] = true
                                }
                        }
                }
                queue = queue[length:]
        }
        return -1
}
```

59.12 1927. 求和游戏 (1)

- 题目

```
Alice 和 Bob 玩一个游戏，两人轮流行动，Alice 先手。
给你一个 偶数长度的字符串num，每一个字符为数字字符或者'?'。
每一次操作中，如果 num中至少有一个 '?'，那么玩家可以执行以下操作：
选择一个下标 i满足num[i] == '?'。
将num[i]用'0'到'9'之间的一个数字字符替代。
当 num中没有 '?' 时，游戏结束。
Bob 获胜的条件是 num中前一半数字的和 等于后一半数字的和。
Alice 获胜的条件是前一半的和与后一半的和 不相等。
比方说，游戏结束时num = "243801"，那么Bob 获胜，因为2+4+3 = 8+0+1。
如果游戏结束时num = "243803"，那么 Alice 获胜，因为2+4+3 != 8+0+3。
在 Alice 和 Bob 都采取 最优策略的前提下，如果 Alice 获胜，请返回 true，如果 Bob␣
↪获胜，请返回 false。
示例 1：输入：num = "5023" 输出：false
解释：num 中没有 '?' ，没法进行任何操作。
前一半的和等于后一半的和：5 + 0 = 2 + 3 。
示例 2：输入：num = "25??" 输出：true
解释：Alice 可以将两个 '?' 中的一个替换为 '9' ，Bob␣
```

(续下页)

(接上页)

```
↪无论如何都无法使前一半的和等于后一半的和。
示例 3: 输入: num = "?3295???" 输出: false
解释: Bob 总是能赢。一种可能的结果是:
- Alice 将第一个 '?' 用 '9' 替换。num = "93295???" 。
- Bob 将后面一半中的一个 '?' 替换为 '9' 。num = "932959??" 。
- Alice 将后面一半中的一个 '?' 替换为 '2' 。num = "9329592?" 。
- Bob 将后面一半中最后一个 '?' 替换为 '7' 。num = "93295927" 。
Bob 获胜, 因为 9 + 3 + 2 + 9 = 5 + 9 + 2 + 7 。
提示: 2 <= num.length <= 105
num.length是 偶数。
num只包含数字字符和'?'。
```

- 解题思路

```
func sumGame(num string) bool {
        n := len(num)
        leftSum, rightSum := 0, 0
        leftCount, rightCount := 0, 0
        for i := 0; i < n; i++ {
                if i < n/2 {
                        if num[i] == '?' {
                                leftCount++
                        } else {
                                leftSum = leftSum + int(num[i]-'0')
                        }
                } else {
                        if num[i] == '?' {
                                rightCount++
                        } else {
                                rightSum = rightSum + int(num[i]-'0')
                        }
                }
        }
        if (leftCount+rightCount)%2 == 1 { // ?总数为奇数个
                return true
        }
        if leftSum-rightSum != (rightCount-leftCount)*9/2 {
                return true
        }
        return false
}
```

59.13 1930. 长度为 3 的不同回文子序列 (3)

- 题目

```
给你一个字符串 s ，返回 s 中 长度为 3 的不同回文子序列 的个数。
即便存在多种方法来构建相同的子序列，但相同的子序列只计数一次。
回文 是正着读和反着读一样的字符串。
子序列␣
↪是由原字符串删除其中部分字符（也可以不删除）且不改变剩余字符之间相对顺序形成的一个新字符串。
例如，"ace" 是 "abcde" 的一个子序列。
示例 1：输入：s = "aabca"输出：3
解释：长度为 3 的 3 个回文子序列分别是：
- "aba" ("aabca" 的子序列)
- "aaa" ("aabca" 的子序列)
- "aca" ("aabca" 的子序列)
示例 2：输入：s = "adc"输出：0
解释："adc" 不存在长度为 3 的回文子序列。
示例 3：输入：s = "bbcbaba" 输出：4
解释：长度为 3 的 4 个回文子序列分别是：
- "bbb" ("bbcbaba" 的子序列)
- "bcb" ("bbcbaba" 的子序列)
- "bab" ("bbcbaba" 的子序列)
- "aba" ("bbcbaba" 的子序列)
提示：3 <= s.length <= 105
s 仅由小写英文字母组成
```

- 解题思路

```
func countPalindromicSubsequence(s string) int {
        n := len(s)
        arr := [26][]int{}
        for i := 0; i < n; i++ {
                index := int(s[i] - 'a')
                arr[index] = append(arr[index], i)
        }
        m := make(map[string]bool)
        for i := 0; i < 26; i++ { // 枚举2边字符
                if len(arr[i]) <= 1 {
                        continue
                }
                left, right := arr[i][0], arr[i][len(arr[i])-1]
                for j := 0; j < 26; j++ {
                        if len(arr[j]) == 0 {
                                continue
```

(续下页)

(接上页)

```
                        }
                        if i == j && len(arr[i]) > 2 {
                                m[fmt.Sprintf("%d,%d,%d", i, i, i)] = true
                                continue
                        }
                        flag := false
                        for k := 0; k < len(arr[j]); k++ {
                                if left < arr[j][k] && arr[j][k] < right {
                                        flag = true
                                        break
                                }
                        }
                        if flag == true {
                                m[fmt.Sprintf("%d,%d,%d", i, j, i)] = true
                        }
                }
        }
        return len(m)
}

# 2
func countPalindromicSubsequence(s string) int {
        n := len(s)
        res := 0
        for i := 0; i < 26; i++ { // 枚举2边字符
                left, right := 0, n-1
                for left < n && s[left] != byte(i+'a') {
                        left++
                }
                for right >= 0 && s[right] != byte(i+'a') {
                        right--
                }
                if left+2 > right {
                        continue
                }
                m := make(map[byte]int)
                for k := left + 1; k < right; k++ {
                        m[s[k]] = 1
                }
                res = res + len(m)
        }
        return res
}
```

(续下页)

(接上页)

```
# 3
func countPalindromicSubsequence(s string) int {
        n := len(s)
        res := 0
        pre, suf := make([]int, n+1), make([]int, n+1)
        arr := make([]int, 26)
        for i := 0; i < n; i++ {
                pre[i+1] = pre[i] | (1 << int(s[i]-'a'))
        }
        for i := n - 1; i >= 0; i-- {
                suf[i] = suf[i+1] | (1 << int(s[i]-'a'))
        }
        for i := 1; i < n-1; i++ {
                index := int(s[i] - 'a')
                arr[index] = arr[index] | (pre[i] & suf[i+1])
        }

        for i := 0; i < 26; i++ {
                res = res + bits.OnesCount(uint(arr[i]))
        }
        return res
}
```

59.14 1936. 新增的最少台阶数 (1)

- 题目

```
给你一个 严格递增 的整数数组 rungs ，用于表示梯子上每一台阶的 高度 。
当前你正站在高度为 0 的地板上，并打算爬到最后一个台阶。
另给你一个整数 dist␣
↪。每次移动中，你可以到达下一个距离你当前位置（地板或台阶）不超过dist高度的台阶。
当然，你也可以在任何正 整数 高度处插入尚不存在的新台阶。
返回爬到最后一阶时必须添加到梯子上的 最少台阶数。
示例 1：输入：rungs = [1,3,5,10], dist = 2 输出：2
解释：现在无法到达最后一阶。
在高度为 7 和 8 的位置增设新的台阶，以爬上梯子。
梯子在高度为 [1,3,5,7,8,10] 的位置上有台阶。
示例 2：输入：rungs = [3,6,8,10], dist = 3 输出：0
解释：这个梯子无需增设新台阶也可以爬上去。
示例 3：输入：rungs = [3,4,6,7], dist = 2 输出：1
解释：现在无法从地板到达梯子的第一阶。
```

(续下页)

(接上页)

```
在高度为 1 的位置增设新的台阶，以爬上梯子。
梯子在高度为 [1,3,4,6,7] 的位置上有台阶。
示例 4：输入：rungs = [5], dist = 10 输出：0
解释：这个梯子无需增设新台阶也可以爬上去。
提示：1 <= rungs.length <= 105
1 <= rungs[i] <= 109
1 <= dist <= 109
rungs 严格递增
```

- 解题思路

```
func addRungs(rungs []int, dist int) int {
        res := 0
        cur := 0
        for i := 0; i < len(rungs); i++{
                res = res + (rungs[i]-cur-1)/dist
                cur = rungs[i]
        }
        return res
}
```

59.15 1937. 扣分后的最大得分 (3)

- 题目

```
给你一个m x n的整数矩阵points（下标从 0开始）。一开始你的得分为␣
↪0，你想最大化从矩阵中得到的分数。
你的得分方式为：每一行中选取一个格子，选中坐标为(r, c)的格子会给你的总得分␣
↪增加points[r][c]。
然而，相邻行之间被选中的格子如果隔得太远，你会失去一些得分。
对于相邻行r 和r + 1（其中0 <= r < m - 1），选中坐标为(r, c1) 和(r + 1, c2)的格子，
你的总得分减少abs(c1 - c2)。
请你返回你能得到的 最大得分。
abs(x)定义为：如果x >= 0，那么值为x。如果x <0，那么值为 -x。
示例 1：输入：points = [[1,2,3],[1,5,1],[3,1,1]] 输出：9
解释：蓝色格子是最优方案选中的格子，坐标分别为 (0, 2)，(1, 1) 和 (2, 0) 。
你的总得分增加 3 + 5 + 3 = 11 。
但是你的总得分需要扣除 abs(2 - 1) + abs(1 - 0) = 2 。
你的最终得分为 11 - 2 = 9 。
示例 2：输入：points = [[1,5],[2,3],[4,2]] 输出：11
解释：蓝色格子是最优方案选中的格子，坐标分别为 (0, 1)，(1, 1) 和 (2, 0) 。
你的总得分增加 5 + 3 + 4 = 12 。
```

(续下页)

(接上页)

```
但是你的总得分需要扣除 abs(1 - 1) + abs(1 - 0) = 1 。
你的最终得分为 12 - 1 = 11 。
提示：m == points.length
n == points[r].length
1 <= m, n <= 105
1 <= m * n <= 105
0 <= points[r][c] <= 105
```

- 解题思路

```
func maxPoints(points [][]int) int64 {
        n, m := len(points), len(points[0])
        dp := make([][]int, n) // dp[i][j]表示在第i行第j列选择的格子的最大分数
        // dp[i][j]=max{dp[i-1][j']-abs(j-j')} + points[i][j]
        // j > j' => dp[i][j]=max{dp[i-1][j']+j'} + points[i][j] - j
        // j <= j' => dp[i][j]=max{dp[i-1][j']-j'} + points[i][j] + j
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        for j := 0; j < m; j++ {
                dp[0][j] = points[0][j]
        }
        for i := 1; i < n; i++ {
                maxValue := math.MinInt32 // max{dp[i-1][j']+j'}
                for j := 0; j < m; j++ {  // 正序，j越大=>maxValue越大
                        maxValue = max(maxValue, dp[i-1][j]+j)
                        // dp[i][j]=max{dp[i-1][j']+j'} + points[i][j] - j
                        dp[i][j] = max(dp[i][j], maxValue+points[i][j]-j)
                }
                maxValue = math.MinInt32      // max{dp[i-1][j']-j'}
                for j := m - 1; j >= 0; j-- { // 逆序，j越小=>maxValue越大
                        maxValue = max(maxValue, dp[i-1][j]-j)
                        // dp[i][j]=max{dp[i-1][j']-j'} + points[i][j] + j
                        dp[i][j] = max(dp[i][j], maxValue+points[i][j]+j)
                }
        }
        res := 0
        for j := 0; j < m; j++ {
                res = max(res, dp[n-1][j])
        }
        return int64(res)
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 2
func maxPoints(points [][]int) int64 {
        n, m := len(points), len(points[0])
        dp := make([]int, m) // dp[j]表示在第j列选择的格子的最大分数
        // dp[i][j]=max{dp[i-1][j']-abs(j-j')} + points[i][j]
        // j > j' => dp[i][j]=max{dp[i-1][j']+j'} + points[i][j] - j
        // j <= j' => dp[i][j]=max{dp[i-1][j']-j'} + points[i][j] + j
        for i := 0; i < n; i++ {
                temp := make([]int, m)
                maxValue := math.MinInt32 // max{dp[i-1][j']+j'}
                for j := 0; j < m; j++ {  // 正序, j越大=>maxValue越大
                        maxValue = max(maxValue, dp[j]+j)
                        // dp[i][j]=max{dp[i-1][j']+j'} + points[i][j] - j
                        temp[j] = max(temp[j], maxValue+points[i][j]-j)
                }
                maxValue = math.MinInt32      // max{dp[i-1][j']-j'}
                for j := m - 1; j >= 0; j-- { // 逆序, j越小=>maxValue越大
                        maxValue = max(maxValue, dp[j]-j)
                        // dp[i][j]=max{dp[i-1][j']-j'} + points[i][j] + j
                        temp[j] = max(temp[j], maxValue+points[i][j]+j)
                }
                copy(dp, temp)
        }
        res := 0
        for j := 0; j < m; j++ {
                res = max(res, dp[j])
        }
        return int64(res)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 3
func maxPoints(points [][]int) int64 {
        n, m := len(points), len(points[0])
        dp := make([]int, m) // dp[j]表示在第j列选择的格子的最大分数
        // dp[i][j]=max{dp[i-1][j']-abs(j-j')} + points[i][j]
        // j > j' => dp[i][j]=max{dp[i-1][j']+j'} + points[i][j] - j
        // j <= j' => dp[i][j]=max{dp[i-1][j']-j'} + points[i][j] + j
        for i := 0; i < n; i++ {
                temp := make([]int, m)
                leftArr := make([]int, m)  // 左边最大值
                rightArr := make([]int, m) // 右边最大值
                leftArr[0] = dp[0]
                rightArr[m-1] = dp[m-1] - (m - 1)
                for j := 1; j < m; j++ {
                        leftArr[j] = max(leftArr[j-1], dp[j]+j)
                }
                for j := m - 2; j >= 0; j-- {
                        rightArr[j] = max(rightArr[j+1], dp[j]-j)
                }
                for j := 0; j < m; j++ {
                        temp[j] = points[i][j] + max(leftArr[j]-j, rightArr[j]+j)
                }
                copy(dp, temp)
        }
        res := 0
        for j := 0; j < m; j++ {
                res = max(res, dp[j])
        }
        return int64(res)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

59.16 1942. 最小未被占据椅子的编号 (2)

- 题目

```
有 n个朋友在举办一个派对，这些朋友从 0到 n - 1编号。派对里有 无数张椅子，编号为 0到
↪infinity。
当一个朋友到达派对时，他会占据编号最小且未被占据的椅子。
比方说，当一个朋友到达时，如果椅子0，1和5被占据了，那么他会占据2号椅子。
当一个朋友离开派对时，他的椅子会立刻变成未占据状态。如果同一时刻有另一个朋友到达，可以立即占据这张椅
给你一个下标从 0开始的二维整数数组times，其中times[i] = [arrivali, leavingi]
表示第 i个朋友到达和离开的时刻，同时给你一个整数 targetFriend。所有到达时间 互不相同。
请你返回编号为 targetFriend的朋友占据的 椅子编号。
示例 1：输入：times = [[1,4],[2,3],[4,6]], targetFriend = 1 输出：1
解释：- 朋友 0 时刻 1 到达，占据椅子 0 。
- 朋友 1 时刻 2 到达，占据椅子 1 。
- 朋友 1 时刻 3 离开，椅子 1 变成未占据。
- 朋友 0 时刻 4 离开，椅子 0 变成未占据。
- 朋友 2 时刻 4 到达，占据椅子 0 。
朋友 1 占据椅子 1 ，所以返回 1 。
示例 2：输入：times = [[3,10],[1,5],[2,6]], targetFriend = 0 输出：2
解释：- 朋友 1 时刻 1 到达，占据椅子 0 。
- 朋友 2 时刻 2 到达，占据椅子 1 。
- 朋友 0 时刻 3 到达，占据椅子 2 。
- 朋友 1 时刻 5 离开，椅子 0 变成未占据。
- 朋友 2 时刻 6 离开，椅子 1 变成未占据。
- 朋友 0 时刻 10 离开，椅子 2 变成未占据。
朋友 0 占据椅子 2 ，所以返回 2 。
提示：n == times.length
2 <= n <= 104
times[i].length == 2
1 <= arrivali < leavingi <= 105
0 <= targetFriend <= n - 1
每个arrivali时刻互不相同。
```

- 解题思路

```
type Node struct {
        Index      int
        ArriveTime int
        LeaveTime  int
}

func smallestChair(times [][]int, targetFriend int) int {
        n := len(times)
```

(续下页)

(接上页)

```
        arr := make([]Node, 0)
        waitHeap := make(WaitHeap, 0)
        heap.Init(&waitHeap)
        runHeap := make(RunHeap, 0)
        heap.Init(&runHeap)
        for i := 0; i < n; i++ {
                heap.Push(&waitHeap, i)
                arr = append(arr, Node{
                        Index:      i,
                        ArriveTime: times[i][0],
                        LeaveTime:  times[i][1],
                })
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].ArriveTime < arr[j].ArriveTime
        })
        cur := 0
        for i := 0; i < n; i++ {
                cur = arr[i].ArriveTime
                for runHeap.Len() > 0 && runHeap[0].EndTime <= cur { //␣
↪结束时间小于当前时间，出堆
                        node := heap.Pop(&runHeap).(RunNode)
                        heap.Push(&waitHeap, node.Id)
                }
                node := heap.Pop(&waitHeap).(int)
                heap.Push(&runHeap, RunNode{
                        Id:      node,
                        EndTime: arr[i].LeaveTime,
                })
                if arr[i].Index == targetFriend { // 目标值
                        return node
                }
        }
        return -1
}

type WaitHeap []int // 空闲堆

func (h WaitHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
```

(续下页)

(接上页)

```
func (h WaitHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h WaitHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *WaitHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *WaitHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

type RunNode struct {
        Id      int
        EndTime int
}

type RunHeap []RunNode // 运行堆

func (h RunHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h RunHeap) Less(i, j int) bool {
        return h[i].EndTime < h[j].EndTime
}

func (h RunHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *RunHeap) Push(x interface{}) {
        *h = append(*h, x.(RunNode))
}

func (h *RunHeap) Pop() interface{} {
```

(续下页)

(接上页)

```
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func smallestChair(times [][]int, targetFriend int) int {
        n := len(times)
        waitHeap := make(WaitHeap, 0)
        heap.Init(&waitHeap)
        arriveArr := make([][2]int, 0)
        leaveArr := make([][2]int, 0)
        for i := 0; i < n; i++ {
                heap.Push(&waitHeap, i)
                arriveArr = append(arriveArr, [2]int{i, times[i][0]})
                leaveArr = append(leaveArr, [2]int{i, times[i][1]})
        }
        sort.Slice(arriveArr, func(i, j int) bool {
                return arriveArr[i][1] < arriveArr[j][1]
        })
        sort.Slice(leaveArr, func(i, j int) bool {
                return leaveArr[i][1] < leaveArr[j][1]
        })

        j := 0
        m := make(map[int]int) // 人=>座位
        for i := 0; i < n; i++ {
                for j < n && leaveArr[j][1] <= arriveArr[i][1] { // 小于当前时间：出堆
                        heap.Push(&waitHeap, m[leaveArr[j][0]])
                        j++
                }
                target := heap.Pop(&waitHeap).(int)
                m[arriveArr[i][0]] = target
                if arriveArr[i][0] == targetFriend { // 目标值
                        return target
                }
        }
        return -1
}

type WaitHeap []int

func (h WaitHeap) Len() int {
```

(续下页)

(接上页)

```
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h WaitHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h WaitHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *WaitHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *WaitHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

59.17 1943. 描述绘画结果 (3)

- 题目

```
给你一个细长的画，用数轴表示。这幅画由若干有重叠的线段表示，每个线段有␣
↪独一无二的颜色。
给你二维整数数组segments，
其中segments[i] = [starti, endi, colori]表示线段为半开区间[starti, endi)␣
↪且颜色为colori。
线段间重叠部分的颜色会被␣
↪混合。如果有两种或者更多颜色混合时，它们会形成一种新的颜色，用一个␣
↪集合表示这个混合颜色。
比方说，如果颜色2，4和6被混合，那么结果颜色为{2,4,6}。
为了简化题目，你不需要输出整个集合，只需要用集合中所有元素的 和来表示颜色集合。
你想要用 最少数目不重叠 半开区间来␣
↪表示这幅混合颜色的画。这些线段可以用二维数组painting表示，
其中 painting[j] = [leftj, rightj, mixj]表示一个半开区间[leftj, rightj)的颜色␣
↪和为mixj。
比方说，这幅画由segments = [[1,4,5],[1,7,7]]组成，那么它可以表示为painting = [[1,4,
↪12],[4,7,7]]，因为：
```

(续下页)

(接上页)

```
[1,4)由颜色{5,7}组成（和为12），分别来自第一个线段和第二个线段。
[4,7)由颜色{7}组成，来自第二个线段。
请你返回二维数组painting，它表示最终绘画的结果（没有被涂色的部分不出现在结果中）。
你可以按 任意顺序 返回最终数组的结果。
半开区间[a, b)是数轴上点a 和点b之间的部分，包含 点a且 不包含点b。
示例 1：输入：segments = [[1,4,5],[4,7,7],[1,7,9]] 输出：[[1,4,14],[4,7,16]]
解释：绘画借故偶可以表示为：
- [1,4) 颜色为 {5,9} （和为 14），分别来自第一和第二个线段。
- [4,7) 颜色为 {7,9} （和为 16），分别来自第二和第三个线段。
示例 2：输入：segments = [[1,7,9],[6,8,15],[8,10,7]] 输出：[[1,6,9],[6,7,24],[7,8,15],
↪[8,10,7]]
解释：绘画结果可以以表示为：
- [1,6) 颜色为 9 ，来自第一个线段。
- [6,7) 颜色为 {9,15} （和为 24），来自第一和第二个线段。
- [7,8) 颜色为 15 ，来自第二个线段。
- [8,10) 颜色为 7 ，来自第三个线段。
示例 3：输入：segments = [[1,4,5],[1,4,7],[4,7,1],[4,7,11]] 输出：[[1,4,12],[4,7,12]]
解释：绘画结果可以表示为：
- [1,4) 颜色为 {5,7} （和为 12），分别来自第一和第二个线段。
- [4,7) 颜色为 {1,11} （和为 12），分别来自第三和第四个线段。
注意，只返回一个单独的线段 [1,7) 是不正确的，因为混合颜色的集合不相同。
提示：1 <= segments.length <= 2 * 104
segments[i].length == 3
1 <= starti < endi <= 105
1 <= colori <= 109
每种颜色colori互不相同。
```

- 解题思路

```
func splitPainting(segments [][]int) [][]int64 {
        res := make([][]int64, 0)
        arr1 := make([]int64, 100005) // 和
        arr2 := make([]int64, 100005) // 次数
        n := len(segments)
        for i := 0; i < n; i++ {
                start := segments[i][0]
                end := segments[i][1]
                count := segments[i][2]
                arr1[start], arr2[start] = arr1[start]+int64(count), arr1[start]+1
                arr1[end], arr2[end] = arr1[end]-int64(count), arr1[end]-1
        }
        prevIndex := -1
        var prev1, prev2 int64
        var sum1, sum2 int64
```

(续下页)

(接上页)

```
        for i := 1; i < 100005; i++ {
                sum1 = sum1 + arr1[i]
                sum2 = sum2 + arr2[i]
                if sum1 > 0 {
                        if prevIndex == -1 { // 之前不为-1, 开头
                                prevIndex = i
                                prev1, prev2 = sum1, sum2
                        } else if prev1 != sum1 { // 和不同, 加入区间
                                res = append(res, []int64{int64(prevIndex), int64(i),␣
↪prev1})
                                prevIndex = i
                                prev1, prev2 = sum1, sum2
                        } else {
                                if prev2 != sum2 { // 次数不同。加入区间
                                        res = append(res, []int64{int64(prevIndex),␣
↪int64(i), prev1})
                                        prevIndex = i
                                        prev1, prev2 = sum1, sum2
                                }
                        }
                } else if sum1 == 0 {
                        if prevIndex > 0 { // 和为0, 之前不为0, 加入区间
                                res = append(res, []int64{int64(prevIndex), int64(i),␣
↪prev1})
                                prevIndex = -1
                                prev1, prev2 = sum1, sum2
                        }
                }
        }
        return res
}

# 2
func splitPainting(segments [][]int) [][]int64 {
        res := make([][]int64, 0)
        m := make(map[int]int)
        for i := 0; i < len(segments); i++ {
                start := segments[i][0]
                end := segments[i][1]
                count := segments[i][2]
                m[start] = m[start] + count
                m[end] = m[end] - count
        }
```

(续下页)

(接上页)

```
        arr := make([][2]int, 0)
        for k, v := range m {
                arr = append(arr, [2]int{k, v})
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][0] < arr[j][0]
        })
        n := len(arr)
        for i := 1; i < n; i++ { // 前缀和
                arr[i][1] = arr[i][1] + arr[i-1][1]
        }
        for i := 0; i < n-1; i++ {
                if arr[i][1] > 0 { // 和大于0
                        res = append(res, []int64{
                                int64(arr[i][0]),
                                int64(arr[i+1][0]),
                                int64(arr[i][1]),
                        })
                }
        }
        return res
}

# 3
func splitPainting(segments [][]int) [][]int64 {
        res := make([][]int64, 0)
        arr := make([]int64, 100005) // 和
        m := make([]bool, 100005)
        n := len(segments)
        for i := 0; i < n; i++ {
                start := segments[i][0]
                end := segments[i][1]
                count := segments[i][2]
                arr[start] = arr[start] + int64(count)
                arr[end] = arr[end] - int64(count)
                m[start] = true
                m[end] = true
        }
        var sum int64
        var prev int64
        for i := 1; i < 100005; i++ {
                if m[i] == true {
                        if sum > 0 {
```

(续下页)

(接上页)

```
                                res = append(res, []int64{prev, int64(i), sum})
                        }
                        sum = sum + arr[i]
                        prev = int64(i)
                }
        }
        return res
}
```

59.18 1946. 子字符串突变后可能得到的最大整数 (2)

- 题目

```
给你一个字符串 num ，该字符串表示一个大整数。另给你一个长度为 10 且 下标从 0 开始的整数数组 change ，
该数组将 0-9 中的每个数字映射到另一个数字。更规范的说法是，数字 d 映射为数字change[d] 。
你可以选择 突变 num 的任一子字符串。
突变 子字符串意味着将每位数字 num[i] 替换为该数字在 change 中的映射（也就是说，将num[i] 替换为 change[num[i]]）。
请你找出在对 num 的任一子字符串执行突变操作（也可以不执行）后，可能得到的 最大整数，并用字符串表示返回。
子字符串 是字符串中的一个连续序列。
示例 1：输入：num = "132", change = [9,8,5,0,3,6,4,2,6,8] 输出："832"
解释：替换子字符串 "1"：
- 1 映射为 change[1] = 8 。
因此 "132" 变为 "832" 。
"832" 是可以构造的最大整数，所以返回它的字符串表示。
示例 2：输入：num = "021", change = [9,4,3,5,7,2,1,9,0,6] 输出："934"
解释：替换子字符串 "021"：
- 0 映射为 change[0] = 9 。
- 2 映射为 change[2] = 3 。
- 1 映射为 change[1] = 4 。
因此，"021" 变为 "934" 。
"934" 是可以构造的最大整数，所以返回它的字符串表示。
示例 3：输入：num = "5", change = [1,4,7,5,3,2,5,6,9,4] 输出："5"
解释："5" 已经是可以构造的最大整数，所以返回它的字符串表示。
提示：1 <= num.length <= 105
num 仅由数字 0-9 组成
change.length == 10
0 <= change[d] <= 9
```

- 解题思路

```
func maximumNumber(num string, change []int) string {
        res := []byte(num)
        flag := false
        for i := 0; i < len(num); i++ {
                value := int(num[i] - '0')
                if value < change[value] {
                        flag = true
                        res[i] = byte(change[value] + '0')
                } else if value == change[value] {
                        res[i] = byte(change[value] + '0')
                } else {
                        if flag == true {
                                break
                        }
                }
        }
        return string(res)
}

# 2
func maximumNumber(num string, change []int) string {
        res := []byte(num)
        for i := 0; i < len(num); i++ {
                if getValue(num[i]) < change[getValue(num[i])] {
                        for i < len(num) && change[getValue(num[i])] >=␣
↪getValue(num[i]) {
                                res[i] = byte(change[getValue(num[i])] + '0')
                                i++
                        }
                        break
                }
        }
        return string(res)
}

func getValue(c byte) int {
        return int(c - '0')
}
```

59.19 1947. 最大兼容性评分和 (4)

- 题目

```
有一份由 n 个问题组成的调查问卷，每个问题的答案要么是 0（no，否），要么是1（yes，是）。
这份调查问卷被分发给 m 名学生和 m 名导师，学生和导师的编号都是从 0 到 m - 1 。
学生的答案用一个二维整数数组 students 表示，
其中 students[i] 是一个整数数组，包含第 i 名学生对调查问卷给出的答案（下标从 0开始）。
导师的答案用一个二维整数数组 mentors 表示，其中 mentors[j] 是一个整数数组，
包含第 j 名导师对调查问卷给出的答案（下标从 0 开始）。
每个学生都会被分配给 一名 导师，而每位导师也会分配到 一名学生。配对的学生与导师之间的兼容性评分等于学生和导师答案相同的次数。
例如，学生答案为[1, 0, 1] 而导师答案为 [0, 0, 1] ，那么他们的兼容性评分为 2，因为只有第二个和第三个答案相同。
请你找出最优的学生与导师的配对方案，以 最大程度上 提高 兼容性评分和 。
给你 students 和 mentors ，返回可以得到的 最大兼容性评分和 。
示例 1：输入：students = [[1,1,0],[1,0,1],[0,0,1]], mentors = [[1,0,0],[0,0,1],[1,1,0]] 输出：8
解释：按下述方式分配学生和导师：
- 学生 0 分配给导师 2 ，兼容性评分为 3 。
- 学生 1 分配给导师 0 ，兼容性评分为 2 。
- 学生 2 分配给导师 1 ，兼容性评分为 3 。
最大兼容性评分和为 3 + 2 + 3 = 8 。
示例 2：输入：students = [[0,0],[0,0],[0,0]], mentors = [[1,1],[1,1],[1,1]] 输出：0
解释：任意学生与导师配对的兼容性评分都是 0 。
提示：m == students.length == mentors.length
n == students[i].length == mentors[j].length
1 <= m, n <= 8
students[i][k] 为 0 或 1
mentors[j][k] 为 0 或 1
```

- 解题思路

```
var arr [][]int
var res int

func maxCompatibilitySum(students [][]int, mentors [][]int) int {
        n := len(students)
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
```

(续下页)

(接上页)

```
                        arr[i][j] = calculate(students[i], mentors[j])
                }
        }
        res = 0
        for i := 0; i < n; i++ {
                visited1, visited2 := make([]int, n), make([]int, n)
                visited1[0], visited2[i] = 1, 1
                dfs(n, visited1, visited2, arr[0][i])
        }
        return res
}

func dfs(n int, visited1, visited2 []int, sum int) {
        if sum > res {
                res = sum
        }
        index := -1
        for i := 0; i < n; i++ {
                if visited1[i] == 0 {
                        index = i
                        break
                }
        }
        if index == -1 {
                return
        }
        for i := 0; i < n; i++ {
                if visited2[i] == 1 {
                        continue
                }
                temp1 := make([]int, n)
                temp2 := make([]int, n)
                copy(temp1, visited1)
                copy(temp2, visited2)
                temp1[index] = 1
                temp2[i] = 1
                dfs(n, temp1, temp2, sum+arr[index][i])
        }
}

func calculate(a, b []int) int {
        res := 0
        for i := 0; i < len(a); i++ {
```

(续下页)

(接上页)

```
                if a[i] == b[i] {
                        res++
                }
        }
        return res
}

# 2
func maxCompatibilitySum(students [][]int, mentors [][]int) int {
        n := len(students)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = calculate(students[i], mentors[j])
                }
        }
        temp := make([]int, n)
        for i := 0; i < n; i++ {
                temp[i] = i
        }
        res := 0
        for {
                if temp == nil {
                        break
                }
                sum := 0
                for i := 0; i < n; i++ {
                        sum = sum + arr[i][temp[i]]
                }
                if sum > res {
                        res = sum
                }
                temp = nextPermutation(temp)
        }
        return res
}

// leetcode31.下一个排列
func nextPermutation(nums []int) []int {
        n := len(nums)
        left := n - 2
        // 以12385764为例，从后往前找到5<7 的升序情况，目标值为左边的数5
```

(续下页)

(接上页)

```
        for left >= 0 && nums[left] >= nums[left+1] {
                left--
        }
        if left == -1 { // 排完了，下一个返回nil结束
                return nil
        }
        right := n - 1
        // 从后往前，找到第一个大于目标值的数，如6>5，然后交换
        for right >= 0 && nums[right] <= nums[left] {
                right--
        }
        nums[left], nums[right] = nums[right], nums[left]
        count := 0
        // 后面是降序状态，让它变为升序
        for i := left + 1; i <= (left+1+n-1)/2; i++ {
                nums[i], nums[n-1-count] = nums[n-1-count], nums[i]
                count++
        }
        return nums
}

func calculate(a, b []int) int {
        res := 0
        for i := 0; i < len(a); i++ {
                if a[i] == b[i] {
                        res++
                }
        }
        return res
}

# 3
func maxCompatibilitySum(students [][]int, mentors [][]int) int {
        n := len(students)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = calculate(students[i], mentors[j])
                }
        }
        target := 1 << n
        dp := make([]int, target) // 所有的状态
```

(续下页)

(接上页)

```
        for i := 1; i < target; i++ {
                count := bits.OnesCount(uint(i))
                for j := 0; j < n; j++ {
                        if i&(1<<j) != 0 { // 判断第j位是否为1
                                prev := i ^ (1 << j)
                                dp[i] = max(dp[i], dp[prev]+arr[count-1][j])
                        }
                }
        }
        return dp[target-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func calculate(a, b []int) int {
        res := 0
        for i := 0; i < len(a); i++ {
                if a[i] == b[i] {
                        res++
                }
        }
        return res
}

# 4
var arr [][]int
var res int

func maxCompatibilitySum(students [][]int, mentors [][]int) int {
        n := len(students)
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = calculate(students[i], mentors[j])
                }
        }
        res = 0
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                visited1, visited2 := make([]int, n), make([]int, n)
                visited1[0], visited2[i] = 1, 1
                dfs(n, visited1, visited2, arr[0][i])
        }
        return res
}

func dfs(n int, visited1, visited2 []int, sum int) {
        if sum > res {
                res = sum
        }
        index := -1
        for i := 0; i < n; i++ {
                if visited1[i] == 0 {
                        index = i
                        break
                }
        }
        if index == -1 {
                return
        }
        for i := 0; i < n; i++ {
                if visited2[i] == 1 {
                        continue
                }
                visited1[index] = 1
                visited2[i] = 1
                dfs(n, visited1, visited2, sum+arr[index][i])
                visited1[index] = 0
                visited2[i] = 0
        }
}

func calculate(a, b []int) int {
        res := 0
        for i := 0; i < len(a); i++ {
                if a[i] == b[i] {
                        res++
                }
        }
        return res
}
```

59.20 1953. 你可以工作的最大周数 (1)

- 题目

```
给你n 个项目，编号从 0 到 n - 1 。
同时给你一个整数数组 milestones ，其中每个 milestones[i] 表示第 i
↪个项目中的阶段任务数量。
你可以按下面两个规则参与项目中的工作：每周，你将会完成 某一个 项目中的
↪恰好一个阶段任务。你每周都 必须 工作。
在 连续的 两周中，你 不能 参与并完成同一个项目中的两个阶段任务。
一旦所有项目中的全部阶段任务都完成，或者仅剩余一个阶段任务都会导致你违反上面的规则，那么你将停止工作
↪。
注意，由于这些条件的限制，你可能无法完成所有阶段任务。
返回在不违反上面规则的情况下你最多能工作多少周。
示例 1：输入：milestones = [1,2,3] 输出：6
解释：一种可能的情形是：- 第 1 周，你参与并完成项目 0 中的一个阶段任务。
- 第 2 周，你参与并完成项目 2 中的一个阶段任务。
- 第 3 周，你参与并完成项目 1 中的一个阶段任务。
- 第 4 周，你参与并完成项目 2 中的一个阶段任务。
- 第 5 周，你参与并完成项目 1 中的一个阶段任务。
- 第 6 周，你参与并完成项目 2 中的一个阶段任务。
总周数是 6 。
示例 2：输入：milestones = [5,2,1] 输出：7
解释：一种可能的情形是：
- 第 1 周，你参与并完成项目 0 中的一个阶段任务。
- 第 2 周，你参与并完成项目 1 中的一个阶段任务。
- 第 3 周，你参与并完成项目 0 中的一个阶段任务。
- 第 4 周，你参与并完成项目 1 中的一个阶段任务。
- 第 5 周，你参与并完成项目 0 中的一个阶段任务。
- 第 6 周，你参与并完成项目 2 中的一个阶段任务。
- 第 7 周，你参与并完成项目 0 中的一个阶段任务。
总周数是 7 。
注意，你不能在第 8 周参与完成项目 0 中的最后一个阶段任务，因为这会违反规则。
因此，项目 0 中会有一个阶段任务维持未完成状态。
提示：n == milestones.length
1 <= n <= 105
1 <= milestones[i] <= 109
```

- 解题思路

```
func numberOfWeeks(milestones []int) int64 {
        var maxCount int64
        var sum int64
        for i := 0; i < len(milestones); i++ {
```

(续下页)

(接上页)

```
                if int64(milestones[i]) > maxCount {
                        maxCount = int64(milestones[i])
                }
                sum = sum + int64(milestones[i])
        }
        if maxCount > (sum+1)/2 {
                return (sum-maxCount)*2 + 1
        }
        return sum
}
```

59.21 1954. 收集足够苹果的最小花园周长 (3)

- 题目

```
给你一个用无限二维网格表示的花园，每一个整数坐标处都有一棵苹果树。整数坐标(i,␣
↪j)处的苹果树有 |i| + |j|个苹果。
你将会买下正中心坐标是 (0, 0)的一块 正方形土地，且每条边都与两条坐标轴之一平行。
给你一个整数neededApples，请你返回土地的最小周长，使得至少有neededApples个苹果在土地里面或者边缘上。
|x|的值定义为：
如果x >= 0，那么值为x
如果x <0，那么值为-x
示例 1：输入：neededApples = 1 输出：8
解释：边长长度为 1 的正方形不包含任何苹果。
但是边长为 2 的正方形包含 12 个苹果（如上图所示）。
周长为 2 * 4 = 8 。
示例 2：输入：neededApples = 13 输出：16
示例 3：输入：neededApples = 1000000000 输出：5040
提示：1 <= neededApples <= 1015
```

- 解题思路

```
func minimumPerimeter(neededApples int64) int64 {
        var total int64
        var i int64
        var sum int64
        for i = 0; ; i = i + 2 {
                sum = 3 * i * i // 边长为i（偶数）的正方形的外围苹果总个数
                total = total + sum
                if neededApples <= total {
                        return int64(i) * 4
                }
```

(续下页)

(接上页)

```
        }
        return 0
}

# 2
func minimumPerimeter(neededApples int64) int64 {
        var i int64
        for i = 1; ; i = i + 1 {
                total := 2 * i * (i + 1) * (2*i + 1)
                if neededApples <= total {
                        return i * 8
                }
        }
        return 0
}

# 3
func minimumPerimeter(neededApples int64) int64 {
        var res, left, right int64
        left = 1
        right = 100000
        for left <= right {
                mid := left + (right-left)/2
                total := 2 * mid * (mid + 1) * (2*mid + 1)
                if total >= neededApples {
                        res = mid
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res * 8
}
```

59.22 1958. 检查操作是否合法 (1)

- 题目

```
给你一个下标从0开始的8 x 8 网格board，其中board[r][c]表示游戏棋盘上的格子(r, c)。
棋盘上空格用'.'表示，白色格子用'W'表示，黑色格子用'B'表示。
游戏中每次操作步骤为：选择一个空格子，将它变成你正在执行的颜色（要么白色，要么黑色）。
但是，合法 操作必须满足：涂色后这个格子是␣
```

(续下页)

(接上页)

```
↪好线段的一个端点（好线段可以是水平的，竖直的或者是对角线）。
好线段指的是一个包含 三个或者更多格子（包含端点格子）的线段，线段两个端点格子为␣
↪同一种颜色，
且中间剩余格子的颜色都为␣
↪另一种颜色（线段上不能有任何空格子）。你可以在下图找到好线段的例子：
给你两个整数rMove 和cMove以及一个字符color，表示你正在执行操作的颜色（白或者黑），
如果将格子(rMove,␣
↪cMove)变成颜色color后，是一个合法操作，那么返回true，如果不是合法操作返回false。
示例 1：输入：board = [[".",".",".","B",".",".",".",".",
[".",".",".","W",".",".",".",".",],[".",".",".","W",".",".",".",".",
[".",".",".","W",".",".",".",".",],["W","B","B",".","W","W","W","B"],
[".",".",".","B",".",".",".",".",],[".",".",".","B",".",".",".",".",
[".",".",".","W",".",".",".",".",]], rMove = 4, cMove = 3, color = "B"
输出：true
解释：'.', 'W' 和 'B' 分别用颜色蓝色，白色和黑色表示。格子 (rMove, cMove) 用 'X'␣
↪标记。
以选中格子为端点的两个好线段在上图中用红色矩形标注出来了。
示例 2：输入：board = [[".",".",".",".",".",".",".",".",
[".","B",".",".","W",".",".",".",],[".",".","W",".",".",".",".",".",
[".",".",".","W","B",".",".",".",],[".",".",".",".",".",".",".",".",
[".",".",".",".","B","W",".",".",],[".",".",".",".",".",".","W",".",
[".",".",".",".",".",".",".","B"]], rMove = 4, cMove = 4, color = "W"
输出：false
解释：虽然选中格子涂色后，棋盘上产生了好线段，但选中格子是作为中间格子，没有产生以选中格子为端点的好
提示：board.length == board[r].length == 8
0 <= rMove, cMove < 8
board[rMove][cMove] == '.'
color要么是'B' 要么是'W'。
```

- 解题思路

```
var dx = []int{1, 1, 0, -1, -1, -1, 0, 1}
var dy = []int{0, 1, 1, 1, 0, -1, -1, -1}

func checkMove(board [][]byte, rMove int, cMove int, color byte) bool {
        for i := 0; i < 8; i++ {
                if judge(board, rMove, cMove, color, dx[i], dy[i]) == true {
                        return true
                }
        }
        return false
}

func judge(board [][]byte, rMove int, cMove int, color byte, dirX, dirY int) bool {
```

(续下页)

(接上页)

```
        x, y := rMove+dirX, cMove+dirY
        count := 1
        for 0 <= x && x < 8 && 0 <= y && y < 8 {
                if board[x][y] == '.' {
                        return false
                }
                if count == 1 {
                        if board[x][y] == color {
                                return false
                        }
                } else {
                        if board[x][y] == color {
                                return true
                        }
                }
                count++
                x = x + dirX
                y = y + dirY
        }
        return false
}
```

59.23 1959.K 次调整数组大小浪费的最小总空间 (1)

- 题目

```
你正在设计一个动态数组。给你一个下标从␣
↪0开始的整数数组nums，其中nums[i]是i时刻数组中的元素数目。
除此以外，你还有一个整数 k，表示你可以 调整数组大小的 最多次数（每次都可以调整成␣
↪任意大小）。
t时刻数组的大小sizet必须大于等于nums[t]，因为数组需要有足够的空间容纳所有元素。
t时刻 浪费的空间为sizet - nums[t]，总浪费空间为满足0 <= t < nums.
↪length的每一个时刻t浪费的空间之和。
在调整数组大小不超过 k次的前提下，请你返回 最小总浪费空间。
注意：数组最开始时可以为任意大小，且不计入调整大小的操作次数。
示例 1：输入：nums = [10,20]，k = 0 输出：10
解释：size = [20,20].
我们可以让数组初始大小为 20 。
总浪费空间为 (20 - 10) + (20 - 20) = 10 。
示例 2：输入：nums = [10,20,30]，k = 1 输出：10
解释：size = [20,20,30].
我们可以让数组初始大小为 20 ，然后时刻 2 调整大小为 30 。
```

(续下页)

(接上页)

```
总浪费空间为 (20 - 10) + (20 - 20) + (30 - 30) = 10 。
示例 3：输入：nums = [10,20,15,30,20], k = 2 输出：15
解释：size = [10,20,20,30,30].
我们可以让数组初始大小为 10 ，时刻 1 调整大小为 20 ，时刻 3 调整大小为 30 。
总浪费空间为 (10 - 10) + (20 - 20) + (20 - 15) + (30 - 30) + (30 - 20) = 15 。
提示：1 <= nums.length <= 200
1 <= nums[i] <= 106
0 <= k <= nums.length - 1
```

- 解题思路

```
func minSpaceWastedKResizing(nums []int, k int) int {
        n := len(nums)
        arr := make([][]int, n) // arr[i][j]表示nums[i:j]的最小浪费空间
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < n; i++ {
                maxValue, sum := math.MinInt32, 0
                for j := i; j < n; j++ {
                        if nums[j] > maxValue {
                                maxValue = nums[j]
                        }
                        sum = sum + nums[j]
                        arr[i][j] = maxValue*(j-i+1) - sum // 最大值*长度-总和
                }
        }
        dp := make([][]int, n) // dp[i][j]表示将nums[:i]分为j段的最小浪费空间
        for i := 0; i < n; i++ {
                dp[i] = make([]int, k+2)
                for j := 0; j < k+2; j++ {
                        dp[i][j] = math.MaxInt32 / 10
                }
        }
        for i := 0; i < n; i++ {
                for j := 1; j <= k+1; j++ { // 调整k次，最少1段，最多k+1段
                        for l := 0; l <= i; l++ {
                                if l == 0 {
                                        dp[i][j] = arr[0][i]
                                } else {
                                        dp[i][j] = min(dp[i][j], dp[l-1][j-
↪1]+arr[l][i])
                                }
                        }
```

(续下页)

(接上页)

```
                }
        }
        return dp[n-1][k+1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

59.24 1962. 移除石子使总数最小 (1)

- 题目

```
给你一个整数数组 piles ，数组 下标从 0 开始 ，其中 piles[i] 表示第 i␣
↪堆石子中的石子数量。
另给你一个整数 k ，请你执行下述操作 恰好 k 次：
选出任一石子堆 piles[i] ，并从中 移除 floor(piles[i] / 2) 颗石子。
注意：你可以对 同一堆 石子多次执行此操作。
返回执行 k 次操作后，剩下石子的 最小 总数。
floor(x) 为 小于 或 等于 x 的 最大 整数。（即，对 x 向下取整）。
示例 1：输入：piles = [5,4,9], k = 2 输出：12
解释：可能的执行情景如下：
- 对第 2 堆石子执行移除操作，石子分布情况变成 [5,4,5] 。
- 对第 0 堆石子执行移除操作，石子分布情况变成 [3,4,5] 。
剩下石子的总数为 12 。
示例 2：输入：piles = [4,3,6,7], k = 3 输出：12
解释：可能的执行情景如下：
- 对第 2 堆石子执行移除操作，石子分布情况变成 [4,3,3,7] 。
- 对第 3 堆石子执行移除操作，石子分布情况变成 [4,3,3,4] 。
- 对第 0 堆石子执行移除操作，石子分布情况变成 [2,3,3,4] 。
剩下石子的总数为 12 。
提示：1 <= piles.length <= 105
1 <= piles[i] <= 104
1 <= k <= 105
```

- 解题思路

```
func minStoneSum(piles []int, k int) int {
        intHeap := make(IntHeap, 0)
```

(续下页)

(接上页)

```
        heap.Init(&intHeap)
        for i := 0; i < len(piles); i++ {
                heap.Push(&intHeap, piles[i])
        }
        for i := 1; i <= k; i++ {
                node := heap.Pop(&intHeap).(int)
                value := node - node/2
                heap.Push(&intHeap, value)
        }
        res := 0
        for i := 0; i < len(piles); i++ {
                res = res + intHeap[i]
        }
        return res
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] > h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

59.25 1963. 使字符串平衡的最小交换次数 (3)

- 题目

```
给你一个字符串 s ，下标从 0 开始 ，且长度为偶数 n 。字符串 恰好 由 n / 2 个开括号 '['␣
↪和 n / 2 个闭括号 ']' 组成。
只有能满足下述所有条件的字符串才能称为 平衡字符串 ：
字符串是一个空字符串，或者
字符串可以记作 AB ，其中 A 和 B 都是 平衡字符串 ，或者
字符串可以写成 [C] ，其中 C 是一个 平衡字符串 。
你可以交换 任意 两个下标所对应的括号 任意 次数。
返回使 s 变成 平衡字符串 所需要的 最小 交换次数。
示例 1：输入：s = "][][" 输出：1
解释：交换下标 0 和下标 3 对应的括号，可以使字符串变成平衡字符串。
最终字符串变成 "[[]]" 。
示例 2：输入：s = "]]][[[" 输出：2
解释：执行下述操作可以使字符串变成平衡字符串：
- 交换下标 0 和下标 4 对应的括号，s = "[]][[]" 。
- 交换下标 1 和下标 5 对应的括号，s = "[[][]]" 。
最终字符串变成 "[[][]]" 。
示例 3：输入：s = "[]" 输出：0
解释：这个字符串已经是平衡字符串。
提示：n == s.length
2 <= n <= 106
n 为偶数
s[i] 为'[' 或 ']'
开括号 '[' 的数目为 n / 2 ，闭括号 ']' 的数目也是 n / 2
```

- 解题思路

```
func minSwaps(s string) int {
        res := 0
        left, right := 0, 0
        for i := 0; i < len(s); i++ {
                if s[i] == '[' {
                        left++
                } else {
                        right++
                }
                if right > left { // 要交换
                        res++
                        right--
                        left++
                }
```

(续下页)

(接上页)

```
        }
        return res
}

# 2
func minSwaps(s string) int {
        count := 0
        minValue := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '[' {
                        count++
                } else {
                        count--
                        minValue = min(minValue, count)
                }
        }
        return (abs(minValue) + 1) / 2
}
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 3
func minSwaps(s string) int {
        stack := make([]byte, 0)
        for i := 0; i < len(s); i++ {
                if s[i] == '[' {
                        stack = append(stack, '[')
                } else {
                        if len(stack) > 0 && stack[len(stack)-1] == '[' {
                                stack = stack[:len(stack)-1]
                        }
                }
```

(续下页)

(接上页)

```
        }
        return len(stack)/2 + len(stack)%2
}
```

59.26 1968. 构造元素不等于两相邻元素平均值的数组 (2)

- 题目

```
给你一个 下标从 0 开始 的数组 nums ，数组由若干 互不相同的 整数组成。
你打算重新排列数组中的元素以满足：重排后，数组中的每个元素都 不等于 其两侧相邻元素的平均值 。
更公式化的说法是，重新排列的数组应当满足这一属性：
对于范围1 <= i < nums.length - 1 中的每个 i ，(nums[i-1] + nums[i+1]) / 2 不等于nums[i] 均成立 。
返回满足题意的任一重排结果。
示例 1：输入：nums = [1,2,3,4,5] 输出：[1,2,4,5,3]
解释：i=1, nums[i] = 2, 两相邻元素平均值为 (1+4) / 2 = 2.5
i=2, nums[i] = 4, 两相邻元素平均值为 (2+5) / 2 = 3.5
i=3, nums[i] = 5, 两相邻元素平均值为 (4+3) / 2 = 3.5
示例 2：输入：nums = [6,2,0,9,7] 输出：[9,7,6,2,0]
解释：i=1, nums[i] = 7, 两相邻元素平均值为 (9+6) / 2 = 7.5
i=2, nums[i] = 6, 两相邻元素平均值为 (7+2) / 2 = 4.5
i=3, nums[i] = 2, 两相邻元素平均值为 (6+0) / 2 = 3
提示：3 <= nums.length <= 105
0 <= nums[i] <= 105
```

- 解题思路

```
func rearrangeArray(nums []int) []int {
        n := len(nums)
        res := make([]int, n)
        sort.Ints(nums)
        index := 0
    // 小大小大小...
        // 后一半数：前后都小于当前
        // 前一半数：前后都大于当前
        for i := 0; i < n; i++ {
                res[index] = nums[i]
                index = index + 2
                if index >= n {
                        index = 1
                }
```

(续下页)

(接上页)

```
        }
        return res
}

# 2
func rearrangeArray(nums []int) []int {
        n := len(nums)
        res := make([]int, 0)
        sort.Ints(nums)
        m := (n + 1) / 2
    // 小大小大小...
        // 后一半数：前后都小于当前
        // 前一半数：前后都大于当前
        for i := 0; i < m; i++ {
                res = append(res, nums[i])
                if i+m < n {
                        res = append(res, nums[i+m])
                }
        }
        return res
}
```

59.27 1969. 数组元素的最小非零乘积 (1)

- 题目

```
给你一个正整数p。你有一个下标从 1开始的数组nums，这个数组包含范围[1, 2p -␣
↪1]内所有整数的二进制形式（两端都 包含）。
你可以进行以下操作 任意次：
从 nums中选择两个元素x和y 。
选择 x中的一位与 y对应位置的位交换。对应位置指的是两个整数 相同位置的二进制位。
比方说，如果x = 1101且y = 0011，交换右边数起第 2位后，我们得到x = 1111 和y = 0001。
请你算出进行以上操作 任意次以后，nums能得到的 最小非零乘积。将乘积对109 + 7取余␣
↪后返回。
注意：答案应为取余 之前的最小值。
示例 1：输入：p = 1 输出：1
解释：nums = [1] 。
只有一个元素，所以乘积为该元素。
示例 2：输入：p = 2 输出：6
解释：nums = [01, 10, 11] 。
所有交换要么使乘积变为 0 ，要么乘积与初始乘积相同。
所以，数组乘积 1 * 2 * 3 = 6 已经是最小值。
```

(续下页)

(接上页)

```
示例 3: 输入: p = 3 输出: 1512
解释: nums = [001, 010, 011, 100, 101, 110, 111]
- 第一次操作中，我们交换第二个和第五个元素最左边的数位。
- 结果数组为 [001, 110, 011, 100, 001, 110, 111] 。
- 第二次操作中，我们交换第三个和第四个元素中间的数位。
- 结果数组为 [001, 110, 001, 110, 001, 110, 111] 。
数组乘积 1 * 6 * 1 * 6 * 1 * 6 * 7 = 1512 是最小乘积。
提示: 1 <= p <= 60
```

- 解题思路

```
var mod = 1000000007

func minNonZeroProduct(p int) int {
        // 2个数x,y交换后，x+y的和是不变的
        // 和不变，要求非0乘积xy最小 => x=1,y=2^p-1
        // 最后: 1个2^p-1, 2^(p-1)-1个1和2^p-2
        a := (1<<p - 1) % mod
        b := (1<<p - 2) % mod
        c := 1<<(p-1) - 1 // 指数不mod
        return a * mypow(b, c) % mod
}

func mypow(a int, n int) int {
        res := 1
        for n > 0 {
                if n%2 == 1 {
                        res = res * a % mod
                }
                a = a * a % mod
                n = n / 2
        }
        return res
}
```

59.28 1971. 寻找图中是否存在路径 (3)

- 题目

```
有一个具有 n个顶点的 双向 图，其中每个顶点标记从 0 到 n - 1 (包含 0 和 n - 1) 。
图中的边用一个二维整数数组 edges 表示，其中 edges[i] = [ui, vi] 表示顶点 ui 和顶点 vi␣
↪之间的双向边。
```

(续下页)

(接上页)

```
每个顶点对由 最多一条 边连接，并且没有顶点存在与自身相连的边。
请你确定是否存在从顶点 start 开始，到顶点 end 结束的 有效路径 。
给你数组 edges 和整数 n、start和end，如果从 start 到 end 存在 有效路径 ，则返回␣
↪true，否则返回 false 。
示例 1：输入：n = 3, edges = [[0,1],[1,2],[2,0]], start = 0, end = 2 输出：true
解释：存在由顶点 0 到顶点 2 的路径:
- 0 → 1 → 2
- 0 → 2
示例 2：输入：n = 6, edges = [[0,1],[0,2],[3,5],[5,4],[4,3]], start = 0, end = 5␣
↪输出：false
解释：不存在由顶点 0 到顶点 5 的路径.
提示：1 <= n <= 2 * 105
0 <= edges.length <= 2 * 105
edges[i].length == 2
0 <= ui, vi <= n - 1
ui != vi
0 <= start, end <= n - 1
不存在双向边
不存在指向顶点自身的边
```

- 解题思路

```
var m map[int][]int
var visited map[int]bool

func validPath(n int, edges [][]int, source int, destination int) bool {
        m = make(map[int][]int)
        visited = make(map[int]bool)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a<=>b
                m[a] = append(m[a], b)
                m[b] = append(m[b], a)
        }
        return dfs(source, destination)
}

func dfs(source int, destination int) bool {
        visited[source] = true
        if source == destination {
                return true
        }
        for i := 0; i < len(m[source]); i++ {
                next := m[source][i]
                if visited[next] == false && dfs(next, destination) {
```

(续下页)

(接上页)

```
                        return true
                }
        }
        return false
}

# 2
func validPath(n int, edges [][]int, source int, destination int) bool {
        m := make(map[int][]int)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a<=>b
                m[a] = append(m[a], b)
                m[b] = append(m[b], a)
        }
        queue := make([]int, 0)
        queue = append(queue, source)
        visited := make(map[int]bool)
        for len(queue) > 0 {
                cur := queue[0]
                queue = queue[1:]
                if cur == destination {
                        return true
                }
                for i := 0; i < len(m[cur]); i++ {
                        next := m[cur][i]
                        if visited[next] == false {
                                queue = append(queue, next)
                                visited[next] = true
                        }
                }
        }
        return false
}

# 3
func validPath(n int, edges [][]int, source int, destination int) bool {
        fa = Init(n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a<=>b
                union(a, b)
        }
        return query(source, destination)
}
```

(续下页)

(接上页)

```
var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

59.29 1975. 最大方阵和 (1)

- 题目

```
给你一个n x n的整数方阵matrix。你可以执行以下操作任意次：
选择matrix中相邻两个元素，并将它们都 乘以-1。
如果两个元素有 公共边，那么它们就是 相邻的。
你的目的是 最大化方阵元素的和。请你在执行以上操作之后，返回方阵的最大和。
示例 1：输入：matrix = [[1,-1],[-1,1]] 输出：4
解释：我们可以执行以下操作使和等于 4 ：
- 将第一行的 2 个元素乘以 -1 。
- 将第一列的 2 个元素乘以 -1 。
示例2：输入：matrix = [[1,2,3],[-1,-2,-3],[1,2,3]] 输出：16
```

(续下页)

(接上页)

```
解释：我们可以执行以下操作使和等于 16 ：
- 将第二行的最后 2 个元素乘以 -1 。
提示：n == matrix.length == matrix[i].length
2 <= n <= 250
-105 <= matrix[i][j] <= 105
```

- 解题思路

```
func maxMatrixSum(matrix [][]int) int64 {
        res := int64(0)
        minValue := math.MaxInt32
        count := 0
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        res = res + int64(abs(matrix[i][j]))
                        minValue = min(minValue, abs(matrix[i][j]))
                        if matrix[i][j] <= 0 {
                                count++
                        }
                }
        }
        if count%2 == 0 {
                return res
        }
        return res - 2*int64(minValue)
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

59.30 1976. 到达目的地的方案数 (4)

- 题目

```
你在一个城市里，城市由 n个路口组成，路口编号为0到n - 1，某些路口之间有 双向道路。
输入保证你可以从任意路口出发到达其他任意路口，且任意两个路口之间最多有一条路。
给你一个整数n和二维整数数组roads，
其中roads[i] = [ui, vi,␣
↪timei]表示在路口ui和vi之间有一条需要花费timei时间才能通过的道路。
你想知道花费 最少时间从路口0出发到达路口n - 1的方案数。
请返回花费 最少时间到达目的地的 路径数目。由于答案可能很大，将结果对109 +␣
↪7取余后返回。
示例 1：输入：n = 7, roads = [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3,3],[3,5,1],[6,5,1],
↪[2,5,1],[0,4,5],[4,6,2]]
输出：4
解释：从路口 0 出发到路口 6 花费的最少时间是 7 分钟。
四条花费 7 分钟的路径分别为：
- 0 → 6
- 0 → 4 → 6
- 0 → 1 → 2 → 5 → 6
- 0 → 1 → 3 → 5 → 6
示例 2：输入：n = 2, roads = [[1,0,10]] 输出：1
解释：只有一条从路口 0 到路口 1 的路，花费 10 分钟。
提示：1 <= n <= 200
n - 1 <= roads.length <= n * (n - 1) / 2
roads[i].length == 3
0 <= ui, vi <= n - 1
1 <= timei <= 109
ui != vi
任意两个路口之间至多有一条路。
从任意路口出发，你能够到达其他任意路口。
```

- 解题思路

```
var mod = 1000000007

func countPaths(n int, roads [][]int) int {
        maxValue := int(1e11)   // math.MaxInt32=2147483647 才10位
↪<200*1e9，可以使用11位或者更大
        arr := make([][]int, n) // 邻接矩阵：i=>j的最短距离
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = maxValue
```

(续下页)

(接上页)

```
                }
        }
        for i := 0; i < len(roads); i++ {
                a, b, c := roads[i][0], roads[i][1], roads[i][2]
                arr[a][b] = c
                arr[b][a] = c
        }
        dis := make([]int, n)
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[0] = 0
        visited := make([]bool, n)
        for i := 0; i < n; i++ {
                target := -1 // 寻找未访问的距离起点最近点
                for j := 0; j < n; j++ {
                        if visited[j] == false && (target == -1 || dis[j] <␣
↪dis[target]) {
                                target = j
                        }
                }
                visited[target] = true
                for j := 0; j < n; j++ { // 更新距离
                        dis[j] = min(dis[j], dis[target]+arr[target][j])
                }
        }
        // 计算某条边是否在边上
        edge := make([]int, n) // 入度
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if dis[i]+arr[i][j] == dis[j] {
                                edge[j]++ // 入度+1
                        }
                }
        }
        dp := make([]int, n) // dp[i] 表示0=>i的最短路个数
        dp[0] = 1
        queue := make([]int, 0)
        queue = append(queue, 0)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                for j := 0; j < n; j++ {
```

(续下页)

(接上页)

```
                        if dis[node]+arr[node][j] == dis[j] {
                                dp[j] = (dp[j] + dp[node]) % mod
                                edge[j]--          // 入度-1
                                if edge[j] == 0 { // 入队
                                        queue = append(queue, j)
                                }
                        }
                }
        }
        return dp[n-1] % mod
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var mod = 1000000007
var dp []int

func countPaths(n int, roads [][]int) int {
        maxValue := int(1e12)   // math.MaxInt32=2147483647 才10位
↪<200*1e9, 可以使用11位或者更大, 这里使用11不行
        arr := make([][]int, n) // 邻接矩阵: i=>j的最短距离
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = maxValue
                }
                arr[i][i] = 0
        }
        for i := 0; i < len(roads); i++ {
                a, b, c := roads[i][0], roads[i][1], roads[i][2]
                arr[a][b] = c
                arr[b][a] = c
        }
        arr = Floyd(arr)
        path := make([][]int, n)
        dp = make([]int, n)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                path[i] = make([]int, 0)
                dp[i] = -1
        }
        // 构建图，都是从下标0开始
        for i := 0; i < len(roads); i++ {
                a, b, c := roads[i][0], roads[i][1], roads[i][2]
                if arr[0][b]-arr[0][a] == c {
                        path[a] = append(path[a], b)
                } else if arr[0][a]-arr[0][b] == c {
                        path[b] = append(path[b], a)
                }
        }
        return dfs(path, 0)
}

func dfs(path [][]int, index int) int {
        if index == len(path)-1 {
                return 1
        }
        if dp[index] != -1 {
                return dp[index]
        }
        dp[index] = 0
        for i := 0; i < len(path[index]); i++ {
                next := path[index][i]
                dp[index] = (dp[index] + dfs(path, next)) % mod
        }
        return dp[index]
}

func Floyd(arr [][]int) [][]int {
        n := len(arr)
        for k := 0; k < n; k++ {
                for i := 0; i < n; i++ {
                        for j := 0; j < n; j++ {
                                if arr[i][k]+arr[k][j] < arr[i][j] {
                                        arr[i][j] = arr[i][k] + arr[k][j]
                                }
                        }
                }
        }
        return arr
}
```

(续下页)

(接上页)

```
# 3
var mod = 1000000007

func countPaths(n int, roads [][]int) int {
        if n <= 2 {
                return 1
        }
        maxValue := int(1e12)   // math.MaxInt32=2147483647 才10位
↪<200*1e9, 可以使用11位或者更大, 这里使用11不行
        arr := make([][]int, n) // 邻接矩阵: i=>j的最短距离
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
                dp[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = maxValue
                }
                arr[i][i] = 0
        }
        for i := 0; i < len(roads); i++ {
                a, b, c := roads[i][0], roads[i][1], roads[i][2]
                arr[a][b] = c
                arr[b][a] = c
                dp[a][b] = 1
                dp[b][a] = 1
        }
        for k := 0; k < n; k++ {
                for i := 0; i < n; i++ {
                        if arr[i][k] == maxValue { // 不联通
                                continue
                        }
                        for j := 0; j < n; j++ {
                                // 不联通/距离大于当前值
                                if arr[j][k] == maxValue || arr[i][k]+arr[k][j] >␣
↪arr[i][j] {
                                        continue
                                }
                                if arr[i][k]+arr[k][j] < arr[i][j] {
                                        dp[i][j] = (dp[i][k] * dp[k][j]) % mod
                                        arr[i][j] = arr[i][k] + arr[k][j]
                                } else if arr[i][k]+arr[k][j] == arr[i][j] {
                                        dp[i][j] = (dp[i][j] + dp[i][k]*dp[k][j]) %␣
```

(续下页)

(接上页)

```
↪mod
                                }
                        }
                }
        }
        return dp[0][n-1]
}

# 4
var mod = 1000000007

func countPaths(n int, roads [][]int) int {
        mathValue := int(1e12)
        arr := make([][][2]int, n)
        for i := 0; i < len(roads); i++ { // 邻接表
                a, b, c := roads[i][0], roads[i][1], roads[i][2]
                arr[a] = append(arr[a], [2]int{b, c})
                arr[b] = append(arr[b], [2]int{a, c})
        }
        dp := make([]int, n)
        dis := make([]int, n) // 0到其他点的距离
        for i := 0; i < n; i++ {
                dis[i] = mathValue
        }
        dis[0] = 0
        dp[0] = 1
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [2]int{0, 0})

        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([2]int)
                a, d := node[0], node[1]
                if dis[a] < d {
                        continue
                }
                for i := 0; i < len(arr[a]); i++ {
                        b, c := arr[a][i][0], arr[a][i][1]
                        if dis[a]+c < dis[b] {
                                dis[b] = dis[a] + c
                                dp[b] = dp[a]
                                heap.Push(&intHeap, [2]int{b, dis[b]})
                        } else if dis[a]+c == dis[b] {
```

(续下页)

(接上页)

```
                                dp[b] = (dp[b] + dp[a]) % mod
                            }
                        }
                }
        return dp[n-1] % mod
}

type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][1] < h[j][1]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

59.31 1980. 找出不同的二进制字符串 (2)

- 题目

```
给你一个字符串数组 nums ，该数组由 n 个 互不相同␣
↪的二进制字符串组成，且每个字符串长度都是 n 。
请你找出并返回一个长度为n且没有出现 在 nums␣
↪中的二进制字符串。如果存在多种答案，只需返回 任意一个 即可。
示例 1：输入：nums = ["01","10"] 输出："11"
解释："11" 没有出现在 nums 中。"00" 也是正确答案。
```

(续下页)

(接上页)

```
示例 2：输入：nums = ["00","01"] 输出："11"
解释："11" 没有出现在 nums 中。"10" 也是正确答案。
示例 3：输入：nums = ["111","011","001"] 输出："101"
解释："101" 没有出现在 nums 中。"000"、"010"、"100"、"110" 也是正确答案。
提示：n == nums.length
1 <= n <= 16
nums[i].length == n
nums[i] 为 '0' 或 '1'
```

- 解题思路

```
func findDifferentBinaryString(nums []string) string {
        n := len(nums)
        m := make(map[string]bool)
        for i := 0; i < len(nums); i++{
                m[strings.Repeat("0",16-n)+nums[i]] = true
        }
        for i := 0; i < (1 << n); i++{
                if m[fmt.Sprintf("%016b", i)] ==false{
                        res := fmt.Sprintf("%016b", i)
                        return res[16-n:]
                }
        }
        return ""
}

# 2
func findDifferentBinaryString(nums []string) string {
        n := len(nums)
        res := ""
        // 康托对角线
        // 只要和nums[i][i]不同，构造出的串就和所有的串都不同
        for i := 0; i < n; i++ {
                if nums[i][i] == '0' {
                        res = res + "1"
                } else {
                        res = res + "0"
                }
        }
        return res
}
```

59.32 1981. 最小化目标值与所选元素的差 (4)

- 题目

```
给你一个大小为 m x n 的整数矩阵 mat 和一个整数 target 。
从矩阵的 每一行 中选择一个整数，你的目标是最小化所有选中元素之和与目标值 target 的
↪绝对差 。
返回 最小的绝对差 。
a 和 b 两数字的 绝对差 是 a - b 的绝对值。
示例 1：输入：mat = [[1,2,3],[4,5,6],[7,8,9]], target = 13 输出：0
解释：一种可能的最优选择方案是：
- 第一行选出 1
- 第二行选出 5
- 第三行选出 7
所选元素的和是 13 ，等于目标值，所以绝对差是 0 。
示例 2：输入：mat = [[1],[2],[3]], target = 100 输出：94
解释：唯一一种选择方案是：
- 第一行选出 1
- 第二行选出 2
- 第三行选出 3
所选元素的和是 6 ，绝对差是 94 。
示例 3：输入：mat = [[1,2,9,8,7]], target = 6 输出：1
解释：最优的选择方案是选出第一行的 7 。
绝对差是 1 。
提示：m == mat.length
n == mat[i].length
1 <= m, n <= 70
1 <= mat[i][j] <= 70
1 <= target <= 800
```

- 解题思路

```
func minimizeTheDifference(mat [][]int, target int) int {
        n := len(mat)
        m := len(mat[0])
        maxSum := 0
        dp := make([][]bool, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]bool, 5000)
        }
        dp[0][0] = true
        for i := 0; i < n; i++ {
                maxValue := 0
                for j := 0; j < m; j++ {
```

(续下页)

(接上页)

```
                        maxValue = max(maxValue, mat[i][j])
                        // 枚举起点+终点
                        for k := mat[i][j]; k <= maxSum+mat[i][j]; k++ {
                                if dp[i][k-mat[i][j]] == true { // 可以转移
                                        dp[i+1][k] = true
                                }
                        }
                }
                maxSum = maxSum + maxValue
        }
        res := math.MaxInt32
        for j := 0; j <= maxSum; j++ {
                if dp[n][j] == true {
                        res = min(res, abs(j-target))
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func minimizeTheDifference(mat [][]int, target int) int {
        n := len(mat)
```

(续下页)

(接上页)

```
        m := len(mat[0])
        maxSum := 0
        dp := []int{1}
        for i := 0; i < n; i++ {
                maxValue := 0
                for j := 0; j < m; j++ {
                        maxValue = max(maxValue, mat[i][j])
                }
                temp := make([]int, maxSum+maxValue+1)
                for j := 0; j < m; j++ {
                        // 枚举起点+终点
                        for k := mat[i][j]; k <= maxSum+mat[i][j]; k++ {
                                temp[k] = temp[k] | dp[k-mat[i][j]]
                        }
                }
                dp = temp
                maxSum = maxSum + maxValue
        }
        res := math.MaxInt32
        for j := 0; j <= maxSum; j++ {
                if dp[j] > 0 {
                        res = min(res, abs(j-target))
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
```

(续下页)

(接上页)

```
                return -a
        }
        return a
}

# 3
func minimizeTheDifference(mat [][]int, target int) int {
        n := len(mat)
        m := len(mat[0])
        dp := make([]bool, target)
        dp[0] = true
        maxValue := math.MaxInt32 // 存储大于等于target的数
        for i := 0; i < n; i++ {
                temp := make([]bool, target)
                tempMaxValue := math.MaxInt32
                for j := 0; j < m; j++ {
                        // 枚举起点+终点
                        for k := 0; k < target; k++ {
                                if dp[k] == true {
                                        if k+mat[i][j] >= target {
                                                tempMaxValue = min(tempMaxValue,␣
↪k+mat[i][j])
                                        } else {
                                                temp[k+mat[i][j]] = true
                                        }
                                }
                        }
                        if maxValue != math.MaxInt32 {
                                tempMaxValue = min(tempMaxValue, maxValue+mat[i][j])
                        }
                }
                dp = temp
                maxValue = tempMaxValue
        }
        res := abs(maxValue - target)
        for i := target - 1; i >= 0; i-- {
                if dp[i] == true {
                        res = min(res, target-i) // i都小于target
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 4
func minimizeTheDifference(mat [][]int, target int) int {
        n := len(mat)
        m := len(mat[0])
        maxValue := 5000
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, maxValue)
        }
        for j := 0; j < m; j++ {
                dp[0][mat[0][j]] = 1
        }
        for i := 1; i < n; i++ {
                for j := 0; j < m; j++ {
                        for k := mat[i][j]; k < maxValue; k++ {
                                dp[i][k] = dp[i][k] | dp[i-1][k-mat[i][j]]
                        }
                }
        }
        res := math.MaxInt32
        for j := 0; j < maxValue; j++ {
                if dp[n-1][j] == 1 {
                        res = min(res, abs(j-target))
                }
        }
        return res
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

59.33 1985. 找出数组中的第 K 大整数 (1)

- 题目

```
给你一个字符串数组 nums 和一个整数 k 。nums 中的每个字符串都表示一个不含前导零的整数。
返回 nums 中表示第 k 大整数的字符串。
注意：重复的数字在统计时会视为不同元素考虑。例如，如果 nums 是 ["1","2","2"],
那么 "2" 是最大的整数，"2" 是第二大的整数，"1" 是第三大的整数。
示例 1：输入：nums = ["3","6","7","10"], k = 4 输出："3"
解释：nums 中的数字按非递减顺序排列为 ["3","6","7","10"]
其中第 4 大整数是 "3"
示例 2：输入：nums = ["2","21","12","1"], k = 3 输出："2"
解释：nums 中的数字按非递减顺序排列为 ["1","2","12","21"]
其中第 3 大整数是 "2"
示例 3：输入：nums = ["0","0"], k = 2 输出："0"
解释：nums 中的数字按非递减顺序排列为 ["0","0"]
其中第 2 大整数是 "0"
提示：1 <= k <= nums.length <= 104
1 <= nums[i].length <= 100
nums[i] 仅由数字组成
nums[i] 不含任何前导零
```

- 解题思路

```
func kthLargestNumber(nums []string, k int) string {
        sort.Slice(nums, func(i, j int) bool {
                if len(nums[i]) == len(nums[j]) {
                        return nums[i] > nums[j]
                }
```

(续下页)

(接上页)

```
                return len(nums[i]) > len(nums[j])
        })
        return nums[k-1]
}
```

59.34 1986. 完成任务的最少工作时间段 (5)

- 题目

```
你被安排了 n个任务。任务需要花费的时间用长度为 n的整数数组tasks表示，第␣
↪i个任务需要花费tasks[i]小时完成。
一个 工作时间段中，你可以 至多连续工作sessionTime个小时，然后休息一会儿。
你需要按照如下条件完成给定任务：
如果你在某一个时间段开始一个任务，你需要在 同一个时间段完成它。
完成一个任务后，你可以 立马开始一个新的任务。
你可以按 任意顺序完成任务。
给你tasks␣
↪和sessionTime，请你按照上述要求，返回完成所有任务所需要的最少数目的工作时间段。
测试数据保证sessionTime 大于等于tasks[i]中的最大值。
示例 1：输入：tasks = [1,2,3], sessionTime = 3 输出：2
解释：你可以在两个工作时间段内完成所有任务。
- 第一个工作时间段：完成第一和第二个任务，花费 1 + 2 = 3 小时。
- 第二个工作时间段：完成第三个任务，花费 3 小时。
示例 2：输入：tasks = [3,1,3,1,1], sessionTime = 8 输出：2
解释：你可以在两个工作时间段内完成所有任务。
- 第一个工作时间段：完成除了最后一个任务以外的所有任务，花费 3 + 1 + 3 + 1 = 8 小时。
- 第二个工作时间段，完成最后一个任务，花费 1 小时。
示例 3：输入：tasks = [1,2,3,4,5], sessionTime = 15 输出：1
解释：你可以在一个工作时间段以内完成所有任务。
提示：n == tasks.length
1 <= n <= 14
1 <= tasks[i] <= 10
max(tasks[i]) <= sessionTime <= 15
```

- 解题思路

```
func minSessions(tasks []int, sessionTime int) int {
        n := len(tasks)
        total := 1 << n          // 总的状态数
        dp := make([]int, total) // dp[i] => 任务状态为i时的最少工作时间
        for i := 0; i < total; i++ {
                dp[i] = n // 最多n次
```

(续下页)

(接上页)

```
        }
        dp[0] = 0
        sum := make([]int, total) // 枚举任务所有状态的和
        for i := 0; i < n; i++ {
                count := 1 << i
                for j := 0; j < count; j++ {
                        sum[count|j] = sum[j] + tasks[i] // 按位或运算：j前面补1=>
↪子集和加上tasks[i]
                }
        }
        for i := 0; i < total; i++ {
                for j := i; j > 0; j = (j - 1) & i { //␣
↪遍历得到比较小的子集：数字i二进制为1位置上的非0子集
                        if sum[j] <= sessionTime {
                                dp[i] = min(dp[i], dp[i^j]+1) // 取补集-
↪异或操作：取dp[i^j]+1操作最小值
                        }
                }
        }
        return dp[total-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minSessions(tasks []int, sessionTime int) int {
        n := len(tasks)
        total := 1 << n          // 总的状态数
        dp := make([]int, total) // dp[i] => 任务状态为i时的最少工作时间
        for i := 0; i < total; i++ {
                dp[i] = n // 最多n次
        }
        dp[0] = 0
        valid := make([]bool, total) // 状态是否<=sessionTime
        for i := 1; i < total; i++ { // 枚举状态
                sum := 0 // 该状态和
                for j := 0; j < n; j++ {
                        if i&(1<<j) > 0 {
```

(续下页)

(接上页)

```
                                sum = sum + tasks[j]
                        }
                }
                if sum <= sessionTime {
                        valid[i] = true
                }
        }

        for i := 1; i < total; i++ { // 枚举状态
                for j := i; j > 0; j = (j - 1) & i { //␣
↪遍历得到比较小的子集：数字i二进制为1位置上的非0子集
                        if valid[j] == true {
                                dp[i] = min(dp[i], dp[i^j]+1) // 取补集-
↪异或操作：取dp[i^j]+1操作最小值
                        }
                }
        }
        return dp[total-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}


# 3
func minSessions(tasks []int, sessionTime int) int {
        n := len(tasks)
        total := 1 << n          // 总的状态数
        dp := make([]int, total) // dp[i] => 任务状态为i时的最少工作时间
        for i := 0; i < total; i++ {
                dp[i] = n // 最多n次
        }
        dp[0] = 0
        sum := make([]int, total) // 枚举任务所有状态的和
        for i := 0; i < n; i++ {
                count := 1 << i
                for j := 0; j < count; j++ {
                        sum[count|j] = sum[j] + tasks[i] // 按位或运算：j前面补1=>
↪子集和加上tasks[i]
```

(续下页)

(接上页)

```
                }
        }
        for i := 0; i < total; i++ {
                for j := 1; j <= i; j++ { // 暴力枚举子集
                        if i|j == i && sum[j] <= sessionTime {
                                dp[i] = min(dp[i], dp[i^j]+1) // 取补集-
↪异或操作：取dp[i^j]+1操作最小值
                        }
                }
        }
        return dp[total-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
var res int

func minSessions(tasks []int, sessionTime int) int {
        res = len(tasks)
        dfs(tasks, sessionTime, 0, 0, make([]int, len(tasks)))
        return res
}

func dfs(tasks []int, sessionTime int, index, count int, arr []int) {
        if count >= res {
                return
        }
        if index == len(tasks) {
                res = count
                return
        }
        flag := false
        for i := 0; i < len(tasks); i++ { // 尝试每个工作段
                if arr[i] == 0 && flag == true { // 使用1个新的工作段即可
                        break
                }
                if arr[i]+tasks[index] > sessionTime { // 当前超时，跳过尝试新的工作段
```

(续下页)

(接上页)

```
                        continue
                }
                if arr[i] == 0 {
                        flag = true
                }
                arr[i] = arr[i] + tasks[index]
                if flag == true {
                        dfs(tasks, sessionTime, index+1, count+1, arr) //␣
↪有使用新的工作段
                } else {
                        dfs(tasks, sessionTime, index+1, count, arr) //␣
↪没有使用新的工作段
                }
                arr[i] = arr[i] - tasks[index]
        }
}

# 5
func minSessions(tasks []int, sessionTime int) int {
        sort.Slice(tasks, func(i, j int) bool {
                return tasks[i] > tasks[j]
        })
        left, right := 1, len(tasks)

        for left < right {
                mid := left + (right-left)/2
                arr := make([]int, mid)
                if dfs(tasks, sessionTime, 0, mid, arr) == true {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

func dfs(tasks []int, sessionTime int, index, count int, arr []int) bool {
        if index == len(tasks) { // 到最后退出
                return true
        }
        flag := false
        for i := 0; i < count; i++ { // 遍历每个工作段
                if arr[i] == 0 && flag == true { // 使用1个新的工作段即可
```

(续下页)

(接上页)

```
                        break
                }
                if arr[i]+tasks[index] > sessionTime { // 当前超时，跳过尝试新的工作段
                        continue
                }
                if arr[i] == 0 {
                        flag = true
                }
                arr[i] = arr[i] + tasks[index]
                if dfs(tasks, sessionTime, index+1, count, arr) == true {
                        return true
                }
                arr[i] = arr[i] - tasks[index]
        }
        return false
}
```

59.35 1992. 找到所有的农场组 (2)

- 题目

```
给你一个下标从 0开始，大小为m x n的二进制矩阵land，其中␣
↪0表示一单位的森林土地，1表示一单位的农场土地。
为了让农场保持有序，农场土地之间以矩形的 农场组 的形式存在。每一个农场组都␣
↪仅包含农场土地。
且题目保证不会有两个农场组相邻，也就是说一个农场组中的任何一块土地都␣
↪不会与另一个农场组的任何一块土地在四个方向上相邻。
land可以用坐标系统表示，其中 land左上角坐标为(0, 0)，右下角坐标为(m-1, n-1)。
请你找到所有 农场组最左上角和最右下角的坐标。
一个左上角坐标为(r1, c1)且右下角坐标为(r2, c2)的 农场组 用长度为 4 的数组[r1, c1, r2,␣
↪c2]表示。
请你返回一个二维数组，它包含若干个长度为 4 的子数组，每个子数组表示 land中的一个␣
↪农场组。
如果没有任何农场组，请你返回一个空数组。可以以 任意顺序返回所有农场组。
示例 1: 输入: land = [[1,0,0],[0,1,1],[0,1,1]] 输出: [[0,0,0,0],[1,1,2,2]]
解释: 第一个农场组的左上角为 land[0][0] ，右下角为 land[0][0] 。
第二个农场组的左上角为 land[1][1] ，右下角为 land[2][2] 。
示例 2: 输入: land = [[1,1],[1,1]] 输出: [[0,0,1,1]]
解释: 第一个农场组左上角为 land[0][0] ，右下角为 land[1][1] 。
示例 3: 输入: land = [[0]] 输出: []
解释: 没有任何农场组。
提示: m == land.length
```

(续下页)

(接上页)

```
n == land[i].length
1 <= m, n <= 300
land只包含0和1。
农场组都是 矩形的形状。
```

- 解题思路

```
// 顺时针：上右下左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var res [][]int
var a, b, c, d int

func findFarmland(land [][]int) [][]int {
        res = make([][]int, 0)
        n, m := len(land), len(land[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if land[i][j] == 1 {
                                a, b, c, d = i, j, i, j
                                land[i][j] = 2
                                dfs(land, i, j)
                                res = append(res, []int{a, b, c, d})
                        }
                }
        }
        return res
}

func dfs(land [][]int, i, j int) {
        a, b = min(a, i), min(b, j)
        c, d = max(c, i), max(d, j)
        for k := 0; k < 4; k++ {
                x, y := i+dx[k], j+dy[k]
                if 0 <= x && x < len(land) && 0 <= y && y < len(land[0]) &&␣
↪land[x][y] == 1 {
                        land[x][y] = 2
                        dfs(land, x, y)
                }
        }
}

func min(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func findFarmland(land [][]int) [][]int {
        res := make([][]int, 0)
        n, m := len(land), len(land[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if land[i][j] == 0 {
                                continue
                        }
                        if (0 < i && land[i-1][j] == 1) || // 左边或者上边为1
                                (0 < j && land[i][j-1] == 1) {
                                continue
                        }
                        var a, b, c, d int
                        a, b = i, j
                        for c = i; c+1 < n && land[c+1][j] == 1; c++ { // 往下遍历
                        }
                        for d = j; d+1 < m && land[i][d+1] == 1; d++ { // 往右遍历
                        }
                        res = append(res, []int{a, b, c, d})
                }
        }
        return res
}
```

59.36 1993. 树上的操作 (2)

- 题目

```
给你一棵n个节点的树，编号从0到n -␣
↪1，以父节点数组parent的形式给出，其中parent[i]是第i个节点的父节点。
树的根节点为 0号节点，所以parent[0] = -1，因为它没有父节点。
你想要设计一个数据结构实现树里面对节点的加锁，解锁和升级操作。
数据结构需要支持如下函数：
Lock：指定用户给指定节点␣
↪上锁，上锁后其他用户将无法给同一节点上锁。只有当节点处于未上锁的状态下，才能进行上锁操作。
Unlock：指定用户给指定节点␣
↪解锁，只有当指定节点当前正被指定用户锁住时，才能执行该解锁操作。
Upgrade：指定用户给指定节点上锁，并且将该节点的所有子孙节点解锁。只有如下 3 个条件␣
↪全部 满足时才能执行升级操作：
指定节点当前状态为未上锁。
指定节点至少有一个上锁状态的子孙节点（可以是 任意用户上锁的）。
指定节点没有任何上锁的祖先节点。
请你实现LockingTree类：
LockingTree(int[] parent)用父节点数组初始化数据结构。
lock(int num, int user) 如果id␣
↪为user的用户可以给节点num上锁，那么返回true，否则返回false。
如果可以执行此操作，节点num会被 id 为 user的用户 上锁。
unlock(int num, int user)如果 id 为 user的用户可以给节点␣
↪num解锁，那么返回true，否则返回 false。
如果可以执行此操作，节点 num变为 未上锁状态。
upgrade(int num, int user)如果 id 为 user的用户可以给节点␣
↪num升级，那么返回true，否则返回 false。
如果可以执行此操作，节点 num会被升级 。
示例 1：输入：["LockingTree", "lock", "unlock", "unlock", "lock", "upgrade", "lock"]
[[[-1, 0, 0, 1, 1, 2, 2]], [2, 2], [2, 3], [2, 2], [4, 5], [0, 1], [0, 1]]
输出：[null, true, false, true, true, true, false]
解释：LockingTree lockingTree = new LockingTree([-1, 0, 0, 1, 1, 2, 2]);
lockingTree.lock(2, 2);    // 返回 true ，因为节点 2 未上锁。
                           // 节点 2 被用户 2 上锁。
lockingTree.unlock(2, 3);  // 返回 false ，因为用户 3 无法解锁被用户 2 上锁的节点。
lockingTree.unlock(2, 2);  // 返回 true ，因为节点 2 之前被用户 2 上锁。
                           // 节点 2 现在变为未上锁状态。
lockingTree.lock(4, 5);    // 返回 true ，因为节点 4 未上锁。
                           // 节点 4 被用户 5 上锁。
lockingTree.upgrade(0, 1); // 返回 true ，因为节点 0␣
↪未上锁且至少有一个被上锁的子孙节点（节点 4）。
                           // 节点 0 被用户 1 上锁，节点 4 变为未上锁。
lockingTree.lock(0, 1);    // 返回 false ，因为节点 0 已经被上锁了。
```

(续下页)

(接上页)

```
提示：n == parent.length
2 <= n <= 2000
对于i != 0，满足0 <= parent[i] <= n - 1
parent[0] == -1
0 <= num <= n - 1
1 <= user <= 104
parent表示一棵合法的树。
lock，unlock和upgrade的调用总共不超过2000次。
```

- 解题思路

```
type LockingTree struct {
        m      map[int]int   // 判断该节点是否上锁
        parent map[int]int   // 父节点
        next   map[int][]int // 保存子节点数组
}

func Constructor(parent []int) LockingTree {
        temp := make(map[int]int)
        next := make(map[int][]int)
        for i := 0; i < len(parent); i++ {
                temp[i] = parent[i]
                next[parent[i]] = append(next[parent[i]], i)
        }
        return LockingTree{m: make(map[int]int), parent: temp, next: next}
}

func (this *LockingTree) Lock(num int, user int) bool {
        if _, ok := this.m[num]; ok {
                return false
        }
        this.m[num] = user
        return true
}

func (this *LockingTree) Unlock(num int, user int) bool {
        if v, ok := this.m[num]; ok == false || v != user {
                return false
        }
        delete(this.m, num)
        return true
}

func (this *LockingTree) Upgrade(num int, user int) bool {
```

(续下页)

(接上页)

```
        if _, ok := this.m[num]; ok == true {
                return false // 条件1
        }
        queue := make([]int, 0)
        queue = append(queue, this.next[num]...)
        flag := false
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node < len(this.parent) {
                        queue = append(queue, this.next[node]...)
                        if _, ok := this.m[node]; ok {
                                flag = true
                                break
                        }
                }
        }
        if flag == false {
                return false // 条件2: 至少有1个上锁的子孙节点
        }
        flag = false
        p := this.parent[num]
        for p != -1 {
                if _, ok := this.m[p]; ok {
                        flag = true
                }
                p = this.parent[p]
        }
        if flag == true { // 条件3: 指定节点没有任何上锁的祖先节点
                return false
        }
        this.m[num] = user // 下面是子孙节点解锁
        queue = make([]int, 0)
        queue = append(queue, this.next[num]...)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node < len(this.parent) {
                        queue = append(queue, this.next[node]...)
                        delete(this.m, node)
                }
        }
        return true
```

(续下页)

(接上页)

```
}

# 2
type LockingTree struct {
        m      map[int]int   // 判断该节点是否上锁
        parent map[int]int   // 父节点
        next   map[int][]int // 保存子节点数组
}

func Constructor(parent []int) LockingTree {
        temp := make(map[int]int)
        next := make(map[int][]int)
        for i := 0; i < len(parent); i++ {
                temp[i] = parent[i]
                next[parent[i]] = append(next[parent[i]], i)
        }
        return LockingTree{m: make(map[int]int), parent: temp, next: next}
}

func (this *LockingTree) Lock(num int, user int) bool {
        if _, ok := this.m[num]; ok {
                return false
        }
        this.m[num] = user
        return true
}

func (this *LockingTree) Unlock(num int, user int) bool {
        if v, ok := this.m[num]; ok == false || v != user {
                return false
        }
        delete(this.m, num)
        return true
}

func (this *LockingTree) Upgrade(num int, user int) bool {
        v := num
        for {
                if v >= 0 {
                        if _, ok := this.m[v]; ok {
                                return false
                        }
                        v = this.parent[v]
```

(续下页)

(接上页)

```
                } else {
                        break
                }
        }
        if this.hasLock(num) == false {
                return false
        }
        this.m[num] = user
        this.dfsUnLock(num)
        return true
}

func (this *LockingTree) hasLock(num int) bool {
        for i := 0; i < len(this.next[num]); i++ {
                v := this.next[num][i]
                if _, ok := this.m[v]; ok {
                        return true
                }
                if this.hasLock(v) {
                        return true
                }
        }
        return false
}

func (this *LockingTree) dfsUnLock(num int) {
        for i := 0; i < len(this.next[num]); i++ {
                v := this.next[num][i]
                delete(this.m, v)
                this.dfsUnLock(v)
        }
}
```

59.37 1996. 游戏中弱角色的数量 (2)

- 题目

```
你正在参加一个多角色游戏，每个角色都有两个主要属性：攻击 和 防御 。
给你一个二维整数数组 properties ，其中 properties[i] = [attacki, defensei]␣
↪表示游戏中第 i 个角色的属性。
如果存在一个其他角色的攻击和防御等级 都严格高于␣
↪该角色的攻击和防御等级，则认为该角色为 弱角色 。
```

(续下页)

(接上页)

```
更正式地，如果认为角色 i 弱于 存在的另一个角色 j ，那么 attackj > attacki 且 defensej␣
↪> defensei 。
返回 弱角色 的数量。
示例 1：输入：properties = [[5,5],[6,3],[3,6]] 输出：0
解释：不存在攻击和防御都严格高于其他角色的角色。
示例 2：输入：properties = [[2,2],[3,3]] 输出：1
解释：第一个角色是弱角色，因为第二个角色的攻击和防御严格大于该角色。
示例 3：输入：properties = [[1,5],[10,4],[4,3]] 输出：1
解释：第三个角色是弱角色，因为第二个角色的攻击和防御严格大于该角色。
提示：2 <= properties.length <= 105
properties[i].length == 2
1 <= attacki, defensei <= 105
```

- 解题思路

```
func numberOfWeakCharacters(properties [][]int) int {
        sort.Slice(properties, func(i, j int) bool {
                if properties[i][0] == properties[j][0] {
                        return properties[i][1] < properties[j][1] //␣
↪相同情况下，防御力从小到大
                }
                return properties[i][0] > properties[j][0] // 攻击力从大到小
        })
        res := 0
        maxValue := 0                           // 最大的防御力
        for i := 0; i < len(properties); i++ { // 攻击力从大到小
                // 注意：攻击力相同的情况，防御力是从小到大的，此时会更新maxValue
                if properties[i][1] < maxValue { // 当前面出现防御力大于当前防御力=>+1
                        res++
                } else {
                        maxValue = properties[i][1] // 更新防御力
                }
        }
        return res
}

# 2
func numberOfWeakCharacters(properties [][]int) int {
        sort.Slice(properties, func(i, j int) bool {
                if properties[i][0] == properties[j][0] {
                        return properties[i][1] < properties[j][1] //␣
↪相同情况下，防御力从小到大
                }
                return properties[i][0] > properties[j][0] // 攻击力从大到小
```

(续下页)

(接上页)

```
        })
        res := 0
        stack := make([]int, 0)
        for i := 0; i < len(properties); i++ {
                // 出栈：小于当前防御力的出栈
                for len(stack) > 0 && properties[stack[len(stack)-1]][1] <=␣
↪properties[i][1] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) > 0 { // 栈顶存在大于当前数，弱角色+1
                        res++
                }
                stack = append(stack, i)
        }
        return res
}
```

59.38 1997. 访问完所有房间的第一天 (3)

- 题目

```
你需要访问n 个房间，房间从 0 到 n - 1 编号。
同时，每一天都有一个日期编号，从 0 开始，依天数递增。你每天都会访问一个房间。
最开始的第 0 天，你访问0 号房间。给你一个长度为 n 且 下标从 0 开始 的数组 nextVisit 。
在接下来的几天中，你访问房间的 次序 将根据下面的 规则 决定：
假设某一天，你访问i 号房间。
如果算上本次访问，访问i 号房间的次数为 奇数 ，
那么 第二天 需要访问nextVisit[i] 所指定的房间，其中 0 <= nextVisit[i] <= i 。
如果算上本次访问，访问i 号房间的次数为 偶数 ，那么 第二天 需要访问(i + 1) mod n␣
↪号房间。
请返回你访问完所有房间的第一天的日期编号。题目数据保证总是存在这样的一天。由于答案可能很大，返回对␣
↪109 + 7 取余后的结果。
示例 1：输入：nextVisit = [0,0] 输出：2
解释：
- 第 0 天，你访问房间 0 。访问 0 号房间的总次数为 1 ，次数为奇数。
 下一天你需要访问房间的编号是 nextVisit[0] = 0
- 第 1 天，你访问房间 0 。访问 0 号房间的总次数为 2 ，次数为偶数。
 下一天你需要访问房间的编号是 (0 + 1) mod 2 = 1
- 第 2 天，你访问房间 1 。这是你第一次完成访问所有房间的那天。
示例 2：输入：nextVisit = [0,0,2] 输出：6
解释：你每天访问房间的次序是 [0,0,1,0,0,1,2,...] 。
第 6 天是你访问完所有房间的第一天。
```

(续下页)

(接上页)

```
示例 3：输入：nextVisit = [0,1,2,0] 输出：6
解释：你每天访问房间的次序是 [0,0,1,1,2,2,3,...] 。
第 6 天是你访问完所有房间的第一天。
提示：n == nextVisit.length
2 <= n <= 105
0 <= nextVisit[i] <= i
```

- 解题思路

```
var mod = 1000000007

func firstDayBeenInAllRooms(nextVisit []int) int {
        n := len(nextVisit)
        dp := make([]int, n) // dp[i]=>访问房间i所需要的天数
        dp[0] = 1
        // 访问到i点时，前面所有的点的访问次数必为偶数
        for i := 1; i < n; i++ {
                // dp[i] = (2*dp[i-1] - dp[nextVisit[i-1]] + 2 + mod) % mod
                a := dp[i-1] + 1                        // 第一次：i-1 => i
↪等于到达dp[i-1]的时间+1
                b := dp[i-1] - dp[nextVisit[i-1]] + 1 // 第二次：第一个到达i-
↪1会回去nextVisit[i-1]，2者相减后+1次返回
                dp[i] = (a + b + mod) % mod
        }
        return (dp[n-1] - 1 + mod) % mod // 第几天从0开始，处理下标
}

# 2
var mod = 1000000007

func firstDayBeenInAllRooms(nextVisit []int) int {
        n := len(nextVisit)
        dp := make([]int, n) // dp[i]=>访问房间i所需要的天数
        dp[0] = 1
        // 访问到i点时，前面所有的点的访问次数必为偶数
        for i := 0; i < n-1; i++ {
                a := (dp[i] - dp[nextVisit[i]] + mod) % mod
                dp[i+1] = (dp[i] + a + 2) % mod
        }
        return (dp[n-1] - 1 + mod) % mod // 第几天从0开始，处理下标
}

# 3
var mod = 1000000007
```

(续下页)

(接上页)

```
func firstDayBeenInAllRooms(nextVisit []int) int {
        n := len(nextVisit)
        dp := make([]int, n) // dp[i]=>访问房间i所需要的天数
        dp[0] = 0            // 下标从0开始
        // 访问到i点时，前面所有的点的访问次数必为偶数
        for i := 1; i < n; i++ {
                dp[i] = (2*dp[i-1] - dp[nextVisit[i-1]] + 2 + mod) % mod
        }
        return dp[n-1]
}
```

CHAPTER 60

1901-2000-Hard

60.1 1912. 设计电影租借系统

60.1.1 题目

```
你有一个电影租借公司和 n个电影商店。
你想要实现一个电影租借系统，它支持查询、预订和返还电影的操作。
同时系统还能生成一份当前被借出电影的报告。
所有电影用二维整数数组entries表示，其中entries[i] = [shopi, moviei, pricei]
表示商店 shopi有一份电影moviei的拷贝，租借价格为pricei。
每个商店有至多一份编号为moviei的电影拷贝。
系统需要支持以下操作：
Search：找到拥有指定电影且 未借出的商店中最便宜的 5 个。
商店需要按照价格升序排序，如果价格相同，则shopi较小的商店排在前面。
如果查询结果少于 5 个商店，则将它们全部返回。如果查询结果没有任何商店，则返回空列表。
Rent：从指定商店借出指定电影，题目保证指定电影在指定商店 未借出。
Drop：在指定商店返还 之前已借出的指定电影。
Report：返回 最便宜的 5 部已借出电影（可能有重复的电影 ID），将结果用二维列表res返回，
其中 res[j] = [shopj, moviej]表示第j便宜的已借出电影是从商店shopj借出的电影moviej。
res中的电影需要按 价格升序排序；如果价格相同，则shopj 较小的排在前面；
如果仍然相同，则 moviej 较小 的排在前面。如果当前借出的电影小于 5␣
↪部，则将它们全部返回。
如果当前没有借出电影，则返回一个空的列表。
请你实现MovieRentingSystem类：
```

(续下页)

(接上页)

```
MovieRentingSystem(int n, int[][]␣
↪entries)将MovieRentingSystem对象用n个商店和entries表示的电影列表初始化。
List<Integer> search(int movie) 如上所述，返回 未借出指定 movie的商店列表。
void rent(int shop, int movie)从指定商店 shop借出指定电影movie。
void drop(int shop, int movie)在指定商店 shop返还之前借出的电影movie。
List<List<Integer>> report() 如上所述，返回最便宜的 已借出电影列表。
注意：测试数据保证rent操作中指定商店拥有 未借出 的指定电影，且drop操作指定的商店␣
↪之前已借出指定电影。
示例 1：输入：["MovieRentingSystem", "search", "rent", "rent", "report", "drop",
↪"search"]
[[3, [[0, 1, 5], [0, 2, 6], [0, 3, 7], [1, 1, 4], [1, 2, 7], [2, 1, 5]]], [1],
[0, 1], [1, 2], [], [1, 2], [2]]
输出：[null, [1, 0, 2], null, null, [[0, 1], [1, 2]], null, [0, 1]]
解释：MovieRentingSystem movieRentingSystem =
new MovieRentingSystem(3, [[0, 1, 5], [0, 2, 6], [0, 3, 7], [1, 1, 4], [1, 2, 7], [2,␣
↪1, 5]]);
movieRentingSystem.search(1);  // 返回 [1, 0, 2] ，商店 1，0 和 2 有未借出的 ID 为 1␣
↪的电影。
商店 1 最便宜，商店 0 和 2 价格相同，所以按商店编号排序。
movieRentingSystem.rent(0, 1); // 从商店 0 借出电影 1 。现在商店 0 未借出电影编号为␣
↪[2,3] 。
movieRentingSystem.rent(1, 2); // 从商店 1 借出电影 2 。现在商店 1 未借出的电影编号为␣
↪[1] 。
movieRentingSystem.report();   // 返回 [[0, 1], [1, 2]] 。商店 0 借出的电影 1␣
↪最便宜，然后是商店 1 借出的电影 2 。
movieRentingSystem.drop(1, 2); // 在商店 1 返还电影 2 。现在商店 1 未借出的电影编号为␣
↪[1,2] 。
movieRentingSystem.search(2);  // 返回 [0, 1] 。商店 0 和 1 有未借出的 ID 为 2␣
↪的电影。商店 0 最便宜，然后是商店 1 。
提示：1 <= n <= 3 * 105
1 <= entries.length <= 105
0 <= shopi < n
1 <= moviei, pricei <= 104
每个商店 至多有一份电影moviei的拷贝。
search，rent，drop 和report的调用总共不超过105次。
```

60.1.2 解题思路

60.2 1928. 规定时间内到达终点的最小花费 (3)

- 题目

```
一个国家有 n个城市，城市编号为0到n - 1，题目保证 所有城市都由双向道路 连接在一起。
道路由二维整数数组edges表示，其中edges[i] = [xi, yi, timei]表示城市xi␣
↪和yi之间有一条双向道路，
耗费时间为timei分钟。
两个城市之间可能会有多条耗费时间不同的道路，但是不会有道路两头连接着同一座城市。
每次经过一个城市时，你需要付通行费。通行费用一个长度为 n且下标从␣
↪0开始的整数数组passingFees表示，
其中passingFees[j]是你经过城市 j需要支付的费用。
一开始，你在城市0，你想要在 maxTime分钟以内（包含 maxTime分钟）到达城市n - 1。
旅行的 费用 为你经过的所有城市 通行费之和（包括起点和终点城市的通行费）。
给你maxTime，edges和passingFees，请你返回完成旅行的最小费用，
如果无法在maxTime分钟以内完成旅行，请你返回-1。
示例 1：输入：maxTime = 30, edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,
↪15]],
passingFees = [5,1,2,20,20,3] 输出：11
解释：最优路径为 0 -> 1 -> 2 -> 5 ，总共需要耗费 30 分钟，需要支付 11 的通行费。
示例 2：输入：maxTime = 29, edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,
↪15]],
passingFees = [5,1,2,20,20,3] 输出：48
解释：最优路径为 0 -> 3 -> 4 -> 5 ，总共需要耗费 26 分钟，需要支付 48 的通行费。
你不能选择路径 0 -> 1 -> 2 -> 5 ，因为这条路径耗费的时间太长。
示例 3：输入：maxTime = 25, edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,
↪15]],
passingFees = [5,1,2,20,20,3] 输出：-1
解释：无法在 25 分钟以内从城市 0 到达城市 5 。
提示：1 <= maxTime <= 1000
n == passingFees.length
2 <= n <= 1000
n - 1 <= edges.length <= 1000
0 <= xi, yi <= n - 1
1 <= timei <= 1000
1 <= passingFees[j] <= 1000
图中两个节点之间可能有多条路径。
图中不含有自环。
```

- 解题思路

```
func minCost(maxTime int, edges [][]int, passingFees []int) int {
        maxValue := math.MaxInt32 / 10
        n := len(passingFees)
        dp := make([][]int, maxTime+1) // dp[i][j] => 在i分钟到达j城市的最少花费
        for i := 0; i <= maxTime; i++ {
                dp[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        dp[i][j] = maxValue
                }
        }
        dp[0][0] = passingFees[0] // 出发的城市也要收费
        for i := 1; i <= maxTime; i++ {
                for j := 0; j < len(edges); j++ {
                        a, b, c := edges[j][0], edges[j][1], edges[j][2] // a=>b c
                        if c <= i {                                      // 小于时间i
                                // 注意: 无向图
                                dp[i][a] = min(dp[i][a], dp[i-c][b]+passingFees[a])
                                dp[i][b] = min(dp[i][b], dp[i-c][a]+passingFees[b])
                        }
                }
        }
        res := maxValue
        for i := 1; i <= maxTime; i++ {
                res = min(res, dp[i][n-1])
        }
        if res == maxValue {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minCost(maxTime int, edges [][]int, passingFees []int) int {
        maxValue := math.MaxInt32 / 10
        n := len(passingFees)
        dp := make([][]int, maxTime+1) // dp[i][j] => 在i分钟到达j城市的最少花费
```

(续下页)

(接上页)

```
        for i := 0; i <= maxTime; i++ {
                dp[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        dp[i][j] = maxValue
                }
        }
        arr := make([][][2]int, n)
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1], edges[i][2] // a=>b c
                arr[a] = append(arr[a], [2]int{b, c})
                arr[b] = append(arr[b], [2]int{a, c})
        }
        dp[0][0] = passingFees[0] // 出发的城市也要收费
        for i := 1; i <= maxTime; i++ {
                for j := 0; j < n; j++ {
                        for k := 0; k < len(arr[j]); k++ {
                                a := j
                                b, c := arr[j][k][0], arr[j][k][1]
                                if c <= i { // 小于时间i
                                        // 注意：无向图
                                        dp[i][a] = min(dp[i][a], dp[i-
↪c][b]+passingFees[a])
                                        dp[i][b] = min(dp[i][b], dp[i-
↪c][a]+passingFees[b])
                                }
                        }
                }
        }
        res := maxValue
        for i := 1; i <= maxTime; i++ {
                res = min(res, dp[i][n-1])
        }
        if res == maxValue {
                return -1
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
```

(续下页)

(接上页)

```
}

# 3
func minCost(maxTime int, edges [][]int, passingFees []int) int {
        n := len(passingFees)
        arr := make([][][2]int, n)
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1], edges[i][2] // a=>b c
                arr[a] = append(arr[a], [2]int{b, c})
                arr[b] = append(arr[b], [2]int{a, c})
        }
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [3]int{passingFees[0], maxTime, 0}) //␣
↪费用，剩余时间，下标
        m := make(map[int]int)
        m[0] = maxTime
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([3]int)
                value, leftTime, index := node[0], node[1], node[2]
                if index == n-1 {
                        return value
                }
                for i := 0; i < len(arr[index]); i++ {
                        a, b := arr[index][i][0], arr[index][i][1]
                        if b > leftTime { // 大于剩余时间
                                continue
                        }
                        if v, ok := m[a]; ok == false || leftTime-b > v {
                                m[a] = leftTime - b
                                heap.Push(&intHeap, [3]int{value + passingFees[a],␣
↪leftTime - b, a})
                        }
                }
        }
        return -1
}

type IntHeap [][3]int

func (h IntHeap) Len() int {
        return len(h)
}
```

(续下页)

(接上页)

```
// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][0] < h[j][0]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([3]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

60.3 1931. 用三种不同颜色为网格涂色

60.3.1 题目

```
给你两个整数 m 和 n 。构造一个 m x n 的网格，其中每个单元格最开始是白色。
请你用 红、绿、蓝 三种颜色为每个单元格涂色。所有单元格都需要被涂色。
涂色方案需要满足：不存在相邻两个单元格颜色相同的情况 。返回网格涂色的方法数。
因为答案可能非常大， 返回 对 109 + 7 取余 的结果。
示例 1：输入：m = 1, n = 1 输出：3
解释：如上图所示，存在三种可能的涂色方案。
示例 2：输入：m = 1, n = 2 输出：6
解释：如上图所示，存在六种可能的涂色方案。
示例 3：输入：m = 5, n = 5 输出：580986
提示：1 <= m <= 5
1 <= n <= 1000
```

60.3.2 解题思路

60.4 1944. 队列中可以看到的人数 (2)

- 题目

```
有n个人排成一个队列，从左到右编号为0到n - 1。
给你以一个整数数组heights，每个整数 互不相同，heights[i]表示第i个人的高度。
一个人能 看到 他右边另一个人的条件是这两人之间的所有人都比他们两人 矮。
更正式的，第i个人能看到第j个人的条件是i < j且
min(heights[i], heights[j]) > max(heights[i+1], heights[i+2], ..., heights[j-1])。
请你返回一个长度为 n的数组answer，其中answer[i]是第i个人在他右侧队列中能看到的人数。
示例 1: 输入: heights = [10,6,8,5,11,9] 输出: [3,1,2,1,1,0]
解释: 第 0 个人能看到编号为 1 ，2 和 4 的人。
第 1 个人能看到编号为 2 的人。
第 2 个人能看到编号为 3 和 4 的人。
第 3 个人能看到编号为 4 的人。
第 4 个人能看到编号为 5 的人。
第 5 个人谁也看不到因为他右边没人。
示例 2: 输入: heights = [5,1,2,3,10] 输出: [4,1,1,1,0]
提示: n == heights.length
1 <= n <= 105
1 <= heights[i] <= 105
heights中所有数 互不相同。
```

- 解题思路

```
func canSeePersonsCount(heights []int) []int {
        n := len(heights)
        res := make([]int, n)
        stack := make([]int, 0) // 递减栈
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 {
                        res[i]++ // 答案+1，必然存在1个可以看到的人
                        if heights[i] > heights[stack[len(stack)-1]] {
                                stack = stack[:len(stack)-1]
                        } else {
                                break
                        }
                }
                stack = append(stack, i)
```

(续下页)

(接上页)

```
        }
        return res
}

# 2
func canSeePersonsCount(heights []int) []int {
        n := len(heights)
        res := make([]int, n)
        stack := make([]int, 0) // 递减栈
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 && heights[i] > heights[stack[len(stack)-1]] {
                        res[i]++
                        stack = stack[:len(stack)-1]
                }
                if len(stack) > 0 {
                        res[i]++ // 非空，还可以看到一个人
                }
                stack = append(stack, i)
        }
        return res
}
```

60.5 1955. 统计特殊子序列的数目 (2)

- 题目

```
特殊序列 是由正整数个 0，紧接着正整数个 1，最后 正整数个 2组成的序列。
比方说，[0,1,2] 和[0,0,1,1,1,2]是特殊序列。
相反，[2,1,0]，[1]和[0,1,2,0]就不是特殊序列。
给你一个数组nums（仅包含整数0，1和2），请你返回 不同特殊子序列的数目。
由于答案可能很大，请你将它对109 + 7 取余 后返回。
一个数组的 子序列是从原数组中删除零个或者若干个元素后，剩下元素不改变顺序得到的序列。
如果两个子序列的 下标集合不同，那么这两个子序列是 不同的。
示例 1：输入：nums = [0,1,2,2] 输出：3
解释：特殊子序列为 [0,1,2,2]，[0,1,2,2] 和 [0,1,2,2] 。
示例 2：输入：nums = [2,2,0,0] 输出：0
解释：数组 [2,2,0,0] 中没有特殊子序列。
示例 3：输入：nums = [0,1,2,0,1,2] 输出：7
解释：特殊子序列包括：
- [0,1,2,0,1,2]
- [0,1,2,0,1,2]
- [0,1,2,0,1,2]
```

(续下页)

(接上页)

```
- [0,1,2,0,1,2]
- [0,1,2,0,1,2]
- [0,1,2,0,1,2]
- [0,1,2,0,1,2]
提示：1 <= nums.length <= 105
0 <= nums[i] <= 2
```

- 解题思路

```
var mod = 1000000007

func countSpecialSubsequences(nums []int) int {
        n := len(nums)
        a, b, c := 0, 0, 0
        for i := 0; i < n; i++ {
                if nums[i] == 0 {
                        a = (a*2 + 1) % mod // 之前0组合+之前0组合加上当前0+单独0
                } else if nums[i] == 1 {
                        b = (b*2 + a) % mod //␣
↪之前01组合+之前01组合加上当前1+之前0组合加上当前1
                } else if nums[i] == 2 {
                        c = (c*2 + b) % mod //␣
↪之前012组合+之前012组合加上当前值2+之前01组合加上当前2
                }
        }
        return c
}

# 2
var mod = 1000000007

func countSpecialSubsequences(nums []int) int {
        n := len(nums)
        dp := make([]int, 3)
        for i := 0; i < n; i++ {
                if nums[i] == 0 {
                        dp[0] = (dp[0]*2 + 1) % mod
                } else if nums[i] == 1 {
                        dp[1] = (dp[1]*2 + dp[0]) % mod
                } else if nums[i] == 2 {
                        dp[2] = (dp[2]*2 + dp[1]) % mod
                }
        }
        return dp[2]
```

(续下页)

(接上页)

```
}
```

60.6 1964. 找出到每个位置为止最长的有效障碍赛跑路线 (2)

- 题目

```
你打算构建一些障碍赛跑路线。给你一个 下标从 0 开始 的整数数组 obstacles ，数组长度为n ，
其中 obstacles[i] 表示第 i 个障碍的高度。
对于每个介于 0 和 n - 1 之间（包含 0 和 n - 1）的下标 i ，在满足下述条件的前提下，
请你找出obstacles 能构成的最长障碍路线的长度：
你可以选择下标介于 0 到 i 之间（包含 0 和 i）的任意个障碍。
在这条路线中，必须包含第 i 个障碍。
你必须按障碍在obstacles中的 出现顺序 布置这些障碍。
除第一个障碍外，路线中每个障碍的高度都必须和前一个障碍 相同 或者 更高 。
返回长度为 n 的答案数组 ans ，其中 ans[i] 是上面所述的下标 i对应的最长障碍赛跑路线的长度。
示例 1：输入：obstacles = [1,2,3,2] 输出：[1,2,3,3]
解释：每个位置的最长有效障碍路线是：
- i = 0: [1], [1] 长度为 1
- i = 1: [1,2], [1,2] 长度为 2
- i = 2: [1,2,3], [1,2,3] 长度为 3
- i = 3: [1,2,3,2], [1,2,2] 长度为 3
示例 2：输入：obstacles = [2,2,1] 输出：[1,2,1]
解释：每个位置的最长有效障碍路线是：
- i = 0: [2], [2] 长度为 1
- i = 1: [2,2], [2,2] 长度为 2
- i = 2: [2,2,1], [1] 长度为 1
示例 3：输入：obstacles = [3,1,5,6,4,2] 输出：[1,1,2,3,2,2]
解释：每个位置的最长有效障碍路线是：
- i = 0: [3], [3] 长度为 1
- i = 1: [3,1], [1] 长度为 1
- i = 2: [3,1,5], [3,5] 长度为 2, [1,5] 也是有效的障碍赛跑路线
- i = 3: [3,1,5,6], [3,5,6] 长度为 3, [1,5,6] 也是有效的障碍赛跑路线
- i = 4: [3,1,5,6,4], [3,4] 长度为 2, [1,4] 也是有效的障碍赛跑路线
- i = 5: [3,1,5,6,4,2], [1,2] 长度为 2
提示：n == obstacles.length
1 <= n <= 105
1 <= obstacles[i] <= 107
```

- 解题思路

```
func longestObstacleCourseAtEachPosition(obstacles []int) []int {
        n := len(obstacles)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = 1
        }
        dp := make([]int, 0)
        dp = append(dp, obstacles[0])
        for i := 1; i < n; i++ {
                if dp[len(dp)-1] <= obstacles[i] {
                        dp = append(dp, obstacles[i])
                        res[i] = len(dp)
                } else {
                        left, right := 0, len(dp)-1
                        index := 0
                        for left <= right {
                                mid := left + (right-left)/2
                                if dp[mid] <= obstacles[i] {
                                        left = mid + 1
                                } else {
                                        index = mid
                                        right = mid - 1
                                }
                        }
                        dp[index] = obstacles[i] // 替换为当前元素
                        res[i] = index + 1
                }
        }
        return res
}

# 2
func longestObstacleCourseAtEachPosition(obstacles []int) []int {
        n := len(obstacles)
        res := make([]int, n)
        dp := make([]int, 0)
        for i := 0; i < n; i++ {
                index := sort.SearchInts(dp, obstacles[i]+1)
                if index < len(dp) {
                        dp[index] = obstacles[i]
                } else {
                        dp = append(dp, obstacles[i])
                }
                res[i] = index + 1
```

(续下页)

(接上页)

```
        }
        return res
}
```

60.7 1970. 你能穿过矩阵的最后一天

60.7.1 题目

60.7.2 解题思路

60.8 1987. 不同的好子序列数目

60.8.1 题目

60.8.2 解题思路

CHAPTER 61

2001-2100-Easy

61.1 2006. 差的绝对值为 K 的数对数目 (2)

- 题目

```
给你一个整数数组nums和一个整数k，请你返回数对(i, j)的数目，满足i < j且|nums[i] -␣
↪nums[j]| == k。
|x|的值定义为：
如果x >= 0，那么值为x。
如果x < 0，那么值为-x。
示例 1：输入：nums = [1,2,2,1], k = 1 输出：4
解释：差的绝对值为 1 的数对为：
- [1,2,2,1]
- [1,2,2,1]
- [1,2,2,1]
- [1,2,2,1]
示例 2：输入：nums = [1,3], k = 3 输出：0
解释：没有任何数对差的绝对值为 3 。
示例 3：输入：nums = [3,2,1,5,4], k = 2 输出：3
解释：差的绝对值为 2 的数对为：
- [3,2,1,5,4]
- [3,2,1,5,4]
- [3,2,1,5,4]
提示：1 <= nums.length <= 200
1 <= nums[i] <= 100
```

(续下页)

(接上页)

```
1 <= k <= 99
```

- 解题思路

```
func countKDifference(nums []int, k int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i]-nums[j] == k || nums[j]-nums[i] == k {
                                res++
                        }
                }
        }
        return res
}

# 2
func countKDifference(nums []int, k int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for i := 0; i < len(nums); i++ {
                res = res + m[nums[i]-k]
        }
        return res
}
```

61.2 2011. 执行操作后的变量值 (2)

- 题目

```
存在一种仅支持 4 种操作和 1 个变量 X 的编程语言:
++X 和 X++ 使变量 X 的值 加 1
--X 和 X-- 使变量 X 的值 减 1
最初, X 的值是 0
给你一个字符串数组 operations , 这是由操作组成的一个列表, 返回执行所有操作后, X 的␣
↪最终值 。
示例 1: 输入: operations = ["--X","X++","X++"] 输出: 1
解释: 操作按下述步骤执行:
最初, X = 0
```

(续下页)

(接上页)

```
--X：X 减 1 ，X =  0 - 1 = -1
X++：X 加 1 ，X = -1 + 1 =  0
X++：X 加 1 ，X =  0 + 1 =  1
示例 2：输入：operations = ["++X","++X","X++"] 输出：3
解释：操作按下述步骤执行：
最初，X = 0
++X：X 加 1 ，X = 0 + 1 = 1
++X：X 加 1 ，X = 1 + 1 = 2
X++：X 加 1 ，X = 2 + 1 = 3
示例 3：输入：operations = ["X++","++X","--X","X--"] 输出：0
解释：操作按下述步骤执行：
最初，X = 0
X++：X 加 1 ，X = 0 + 1 = 1
++X：X 加 1 ，X = 1 + 1 = 2
--X：X 减 1 ，X = 2 - 1 = 1
X--：X 减 1 ，X = 1 - 1 = 0
提示：1 <= operations.length <= 100
operations[i] 将会是 "++X"、"X++"、"--X" 或 "X--"
```

- 解题思路

```
func finalValueAfterOperations(operations []string) int {
        res := 0
        for i := 0; i < len(operations); i++ {
                if operations[i][1] == '+' {
                        res++
                } else {
                        res--
                }
        }
        return res
}

# 2
func finalValueAfterOperations(operations []string) int {
        res := 0
        for i := 0; i < len(operations); i++ {
                if strings.Contains(operations[i], "+") {
                        res++
                } else {
                        res--
                }
        }
        return res
```

(续下页)

(接上页)

```
}
```

61.3 2016. 增量元素之间的最大差值 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，该数组的大小为 n ，
请你计算 nums[j] - nums[i] 能求得的 最大差值 ，其中 0 <= i < j < n 且 nums[i] <␣
↪nums[j] 。
返回 最大差值 。如果不存在满足要求的 i 和 j ，返回 -1 。
示例 1：输入：nums = [7,1,5,4] 输出：4
解释：最大差值出现在 i = 1 且 j = 2 时，nums[j] - nums[i] = 5 - 1 = 4 。
注意，尽管 i = 1 且 j = 0 时 ，nums[j] - nums[i] = 7 - 1 = 6 > 4 ，但 i > j␣
↪不满足题面要求，所以 6 不是有效的答案。
示例 2：输入：nums = [9,4,3,2] 输出：-1
解释：不存在同时满足 i < j 和 nums[i] < nums[j] 这两个条件的 i, j 组合。
示例 3：输入：nums = [1,5,2,10] 输出：9
解释：最大差值出现在 i = 0 且 j = 3 时，nums[j] - nums[i] = 10 - 1 = 9 。
提示：n == nums.length
2 <= n <= 1000
1 <= nums[i] <= 109
```

- 解题思路

```
func maximumDifference(nums []int) int {
        res := 0
        minValue := nums[0]
        for i := 1; i < len(nums); i++ {
                res = max(res, nums[i]-minValue)
                minValue = min(minValue, nums[i])
        }
        if res == 0 {
                return -1
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func maximumDifference(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i] < nums[j] {
                                res = max(res, nums[j]-nums[i])
                        }
                }
        }
        if res == 0 {
                return -1
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

61.4 2022. 将一维数组转变成二维数组 (2)

- 题目

```
给你一个下标从 0开始的一维整数数组original和两个整数m和n。
你需要使用original中所有元素创建一个m行n列的二维数组。
original中下标从 0到 n - 1（都 包含 ）的元素构成二维数组的第一行，
下标从 n到 2 * n - 1（都 包含）的元素构成二维数组的第二行，依此类推。
请你根据上述过程返回一个m x␣
↪n的二维数组。如果无法构成这样的二维数组，请你返回一个空的二维数组。
```

(续下页)

(接上页)

```
示例 1：输入：original = [1,2,3,4], m = 2, n = 2 输出：[[1,2],[3,4]]
解释：构造出的二维数组应该包含 2 行 2 列。
original 中第一个 n=2 的部分为 [1,2] ，构成二维数组的第一行。
original 中第二个 n=2 的部分为 [3,4] ，构成二维数组的第二行。
示例 2：输入：original = [1,2,3], m = 1, n = 3 输出：[[1,2,3]]
解释：构造出的二维数组应该包含 1 行 3 列。
将 original 中所有三个元素放入第一行中，构成要求的二维数组。
示例 3：输入：original = [1,2], m = 1, n = 1 输出：[]
解释：original 中有 2 个元素。
无法将 2 个元素放入到一个 1x1 的二维数组中，所以返回一个空的二维数组。
示例 4：输入：original = [3], m = 1, n = 2 输出：[]
解释：original 中只有 1 个元素。
无法将 1 个元素放满一个 1x2 的二维数组，所以返回一个空的二维数组。
提示：1 <= original.length <= 5 * 104
1 <= original[i] <= 105
1 <= m, n <= 4 * 104
```

- 解题思路

```
func construct2DArray(original []int, m int, n int) [][]int {
        total := len(original)
        if n*m != total {
                return nil
        }
        res := make([][]int, 0)
        index := 0
        for i := 0; i < m; i++ {
                temp := make([]int, 0)
                for j := 0; j < n; j++ {
                        temp = append(temp, original[index])
                        index++
                }
                res = append(res, temp)
        }
        return res
}

# 2
func construct2DArray(original []int, m int, n int) [][]int {
        total := len(original)
        if n*m != total {
                return nil
        }
        res := make([][]int, 0)
```

(续下页)

(接上页)

```
        for i := 0; i < total; i = i + n {
                res = append(res, original[i:i+n])
        }
        return res
}
```

61.5 2027. 转换字符串的最少操作次 (2)

- 题目

```
给你一个字符串 s ，由 n 个字符组成，每个字符不是 'X' 就是 'O' 。
一次 操作 定义为从 s 中选出 三个连续字符 并将选中的每个字符都转换为 'O'␣
↪。注意，如果字符已经是 'O' ，只需要保持 不变 。
返回将 s 中所有字符均转换为 'O' 需要执行的最少操作次数。
示例 1：输入：s = "XXX" 输出：1
解释：XXX -> OOO
一次操作，选中全部 3 个字符，并将它们转换为 'O' 。
示例 2：输入：s = "XXOX" 输出：2
解释：XXOX -> OOOX -> OOOO
第一次操作，选择前 3 个字符，并将这些字符转换为 'O' 。
然后，选中后 3 个字符，并执行转换。最终得到的字符串全由字符 'O' 组成。
示例 3：输入：s = "OOOO" 输出：0
解释：s 中不存在需要转换的 'X' 。
提示：3 <= s.length <= 1000
s[i] 为 'X' 或 'O'
```

- 解题思路

```
func minimumMoves(s string) int {
        arr := []byte(s)
        res := 0
        for i := 0; i < len(s); i++ {
                if arr[i] == 'X' {
                        count := 0
                        for j := i; j < len(s); j++ {
                                arr[j] = 'O'
                                count++
                                if count == 3 {
                                        break
                                }
                        }
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 2
func minimumMoves(s string) int {
        res := 0
        for i := 0; i < len(s); i++ {
                if s[i] == 'X' {
                        i = i + 2
                        res++
                }
        }
        return res
}
```

61.6 2032. 至少在两个数组中出现的值 (2)

- 题目

```
给你三个整数数组 nums1、nums2 和 nums3 ，请你构造并返回一个 不同 数组，且由 至少 在
↪两个 数组中出现的所有值组成。
数组中的元素可以按 任意 顺序排列。
示例 1：输入：nums1 = [1,1,3,2], nums2 = [2,3], nums3 = [3] 输出：[3,2]
解释：至少在两个数组中出现的所有值为：
- 3 ，在全部三个数组中都出现过。
- 2 ，在数组 nums1 和 nums2 中出现过。
示例 2：输入：nums1 = [3,1], nums2 = [2,3], nums3 = [1,2] 输出：[2,3,1]
解释：至少在两个数组中出现的所有值为：
- 2 ，在数组 nums2 和 nums3 中出现过。
- 3 ，在数组 nums1 和 nums2 中出现过。
- 1 ，在数组 nums1 和 nums3 中出现过。
示例 3：输入：nums1 = [1,2,2], nums2 = [4,3,3], nums3 = [5] 输出：[]
解释：不存在至少在两个数组中出现的值。
提示：1 <= nums1.length, nums2.length, nums3.length <= 100
1 <= nums1[i], nums2[j], nums3[k] <= 100
```

- 解题思路

```
func twoOutOfThree(nums1 []int, nums2 []int, nums3 []int) []int {
        m1, m2, m3 := make(map[int]int), make(map[int]int), make(map[int]int)
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums1); i++ {
                m1[nums1[i]] = 1
        }
        for i := 0; i < len(nums2); i++ {
                m2[nums2[i]] = 1
        }
        for i := 0; i < len(nums3); i++ {
                m3[nums3[i]] = 1
        }
        res := make([]int, 0)
        for i := 1; i <= 300; i++ {
                a := m1[i] + m2[i] + m3[i]
                if a >= 2 {
                        res = append(res, i)
                }
        }
        return res
}

# 2
func twoOutOfThree(nums1 []int, nums2 []int, nums3 []int) []int {
        m := make(map[int]int)
        arr := [][]int{nums1, nums2, nums3}
        for i := 0; i < len(arr); i++ {
                for j := 0; j < len(arr[i]); j++ {
                        value := arr[i][j]
                        m[value] = m[value] | (1 << i)
                }
        }
        res := make([]int, 0)
        for k, v := range m {
                if bits.OnesCount(uint(v)) >= 2 {
                        res = append(res, k)
                }
        }
        return res
}
```

61.7 2037. 使每位学生都有座位的最少移动次数 (1)

- 题目

```
一个房间里有 n个座位和 n名学生，房间用一个数轴表示。给你一个长度为n的数组seats，其中seats[i] 是第 i个座位的位置。
同时给你一个长度为 n的数组students，其中students[j]是第 j位学生的位置。
你可以执行以下操作任意次：
增加或者减少第i位学生的位置，每次变化量为 1（也就是将第 i位学生从位置 x移动到 x + 1或者 x - 1）
请你返回使所有学生都有座位坐的 最少移动次数，并确保没有两位学生的座位相同。
请注意，初始时有可能有多个座位或者多位学生在 同一位置。
示例 1：输入：seats = [3,1,5], students = [2,7,4] 输出：4
解释：学生移动方式如下：
- 第一位学生从位置 2 移动到位置 1 ，移动 1 次。
- 第二位学生从位置 7 移动到位置 5 ，移动 2 次。
- 第三位学生从位置 4 移动到位置 3 ，移动 1 次。
总共 1 + 2 + 1 = 4 次移动。
示例 2：输入：seats = [4,1,5,9], students = [1,3,2,6] 输出：7
解释：学生移动方式如下：
- 第一位学生不移动。
- 第二位学生从位置 3 移动到位置 4 ，移动 1 次。
- 第三位学生从位置 2 移动到位置 5 ，移动 3 次。
- 第四位学生从位置 6 移动到位置 9 ，移动 3 次。
总共 0 + 1 + 3 + 3 = 7 次移动。
示例 3：输入：seats = [2,2,6,6], students = [1,3,2,6] 输出：4
解释：学生移动方式如下：
- 第一位学生从位置 1 移动到位置 2 ，移动 1 次。
- 第二位学生从位置 3 移动到位置 6 ，移动 3 次。
- 第三位学生不移动。
- 第四位学生不移动。
总共 1 + 3 + 0 + 0 = 4 次移动。
提示：n == seats.length == students.length
1 <= n <= 100
1 <= seats[i], students[j] <= 100
```

- 解题思路

```go
func minMovesToSeat(seats []int, students []int) int {
        res := 0
        sort.Ints(seats)
        sort.Ints(students)
        for i := 0; i < len(seats); i++ {
                res = res + abs(students[i]-seats[i])
```

(续下页)

(接上页)

```
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

61.8 2042. 检查句子中的数字是否递增 (1)

- 题目

```
句子是由若干 token 组成的一个列表，token 间用 单个 空格分隔，句子没有前导或尾随空格。
每个 token 要么是一个由数字 0-9 组成的不含前导零的␣
↪正整数，要么是一个由小写英文字母组成的 单词 。
示例，"a puppy has 2 eyes 4 legs" 是一个由 7 个 token 组成的句子：
"2" 和 "4" 是数字，其他像"puppy" 这样的 tokens 属于单词。
给你一个表示句子的字符串 s ，你需要检查 s 中的 全部 数字是否从左到右严格递增
（即，除了最后一个数字，s 中的 每个 数字都严格小于它 右侧 的数字）。
如果满足题目要求，返回 true，否则，返回 false 。
示例 1：输入：s = "1 box has 3 blue 4 red 6 green and 12 yellow marbles" 输出：true
解释：句子中的数字是：1, 3, 4, 6, 12 。
这些数字是按从左到右严格递增的 1 < 3 < 4 < 6 < 12 。
示例 2：输入：s = "hello world 5 x 5" 输出：false
解释：句子中的数字是：5, 5 。这些数字不是严格递增的。
示例 3：输入：s = "sunset is at 7 51 pm overnight lows will be in the low 50 and 60 s
↪" 输出：false
解释：s 中的数字是：7, 51, 50, 60 。这些数字不是严格递增的。
示例 4：输入：s = "4 5 11 26" 输出：true
解释：s 中的数字是：4, 5, 11, 26 。
这些数字是按从左到右严格递增的：4 < 5 < 11 < 26 。
提示：3 <= s.length <= 200
s 由小写英文字母、空格和数字 0 到 9 组成（包含 0 和 9）
s 中数字 token 的数目在 2 和 100 之间（包含 2 和 100）
s 中的 token 之间由单个空格分隔
s 中至少有 两个 数字
s 中的每个数字都是一个 小于 100 的 正 数，且不含前导零
s 不含前导或尾随空格
```

- 解题思路

```
func areNumbersAscending(s string) bool {
        arr := strings.Split(s, " ")
        prev := -1
        for i := 0; i < len(arr); i++ {
                if '0' <= arr[i][0] && arr[i][0] <= '9' {
                        value, _ := strconv.Atoi(arr[i])
                        if value > prev {
                                prev = value
                        } else {
                                return false
                        }
                }
        }
        return true
}
```

61.9 2047. 句子中的有效单词数 (2)

- 题目

```
句子仅由小写字母 ('a' 到 'z') 、数字 ('0' 到 '9') 、连字符 ('-') 、标点符号 ('!'、'.'
↪和 ',') 以及空格 (' ') 组成。
每个句子可以根据空格分解成 一个或者多个 token ，这些 token 之间由一个或者多个空格 ' '
↪分隔。
如果一个 token 同时满足下述条件，则认为这个 token 是一个有效单词：
仅由小写字母、连字符和/或标点（不含数字）。
至多一个 连字符 '-' 。如果存在，连字符两侧应当都存在小写字母 ("a-b"
↪是一个有效单词，但 "-ab" 和 "ab-" 不是有效单词）。
至多一个 标点符号。如果存在，标点符号应当位于 token 的 末尾 。
这里给出几个有效单词的例子："a-b."、"afad"、"ba-c"、"a!" 和 "!" 。
给你一个字符串 sentence ，请你找出并返回 sentence 中 有效单词的数目 。
示例 1：输入：sentence = "cat and  dog" 输出：3
解释：句子中的有效单词是 "cat"、"and" 和 "dog"
示例 2：输入：sentence = "!this  1-s b8d!" 输出：0
解释：句子中没有有效单词
"!this" 不是有效单词，因为它以一个标点开头
"1-s" 和 "b8d" 也不是有效单词，因为它们都包含数字
示例 3：输入：sentence = "alice and  bob are playing stone-game10" 输出：5
解释：句子中的有效单词是 "alice"、"and"、"bob"、"are" 和 "playing"
"stone-game10" 不是有效单词，因为它含有数字
示例 4：输入：sentence = "he bought 2 pencils, 3 erasers, and 1  pencil-sharpener."
↪输出：6
```

(续下页)

(接上页)

```
解释：句子中的有效单词是 "he"、"bought"、"pencils,"、"erasers,"、"and" 和 "pencil-
↪sharpener."
提示：1 <= sentence.length <= 1000
sentence 由小写英文字母、数字（0-9）、以及字符（' '、'-'、'!'、'.' 和 ','）组成
句子中至少有 1 个 token
```

- 解题思路

```
var m map[byte]bool = map[byte]bool{
        '!': true,
        '.': true,
        ',': true,
}

func countValidWords(sentence string) int {
        res := 0
        arr := strings.Fields(sentence)
        for i := 0; i < len(arr); i++ {
                flag := true
                countA := 0
                countB := 0
                for j := 0; j < len(arr[i]); j++ {
                        if '0' <= arr[i][j] && arr[i][j] <= '9' {
                                flag = false
                                break
                        }
                        if m[arr[i][j]] == true {
                                countA++
                                if j != len(arr[i])-1 {
                                        flag = false
                                        break
                                }
                        }
                        if arr[i][j] == '-' {
                                countB++
                                if j == 0 || j == len(arr[i])-1 {
                                        flag = false
                                        break
                                }
                                if (('a' <= arr[i][j-1] && arr[i][j-1] <= 'z') &&
                                        ('a' <= arr[i][j+1] && arr[i][j+1] <= 'z'))␣
↪== false {
                                        flag = false
                                        break
```

(续下页)

(接上页)

```
                                }
                        }
                }
                if flag == true && countA <= 1 && countB <= 1 {
                        res++
                }
        }
        return res
}

# 2
var m map[byte]bool = map[byte]bool{
        '!': true,
        '.': true,
        ',': true,
}

func countValidWords(sentence string) int {
        res := 0
        arr := strings.Fields(sentence)
        for i := 0; i < len(arr); i++ {
                temp := arr[i]
                if m[temp[len(temp)-1]] == true {
                        temp = temp[:len(temp)-1]
                }
                if strings.ContainsAny(arr[i], "0123456789") ||
                        strings.ContainsAny(temp, "!.,") ||
                        strings.Count(temp, "-") >= 2 {
                        continue
                }
                j := strings.IndexByte(temp, '-')
                // 存在-
                if j >= 0 && (j == 0 || j == len(temp)-1 || unicode.
↪IsLower(rune(temp[j-1])) == false ||
                        unicode.IsLower(rune(temp[j+1])) == false) {
                        continue
                }
                res++
        }
        return res
}
```

61.10 2053. 数组中第 K 个独一无二的字符串 (1)

- 题目

```
独一无二的字符串指的是在一个数组中只出现过 一次的字符串。
给你一个字符串数组arr和一个整数k，请你返回arr中第k个独一无二的字符串。
如果少于k个独一无二的字符串，那么返回空字符串""。
注意，按照字符串在原数组中的 顺序找到第 k个独一无二字符串。
示例 1:输入：arr = ["d","b","c","b","c","a"], k = 2 输出："a"
解释：arr 中独一无二字符串包括 "d" 和 "a"。
"d" 首先出现，所以它是第 1 个独一无二字符串。
"a" 第二个出现，所以它是 2 个独一无二字符串。
由于 k == 2 ，返回 "a" 。
示例 2:输入：arr = ["aaa","aa","a"], k = 1 输出："aaa"
解释：arr 中所有字符串都是独一无二的，所以返回第 1 个字符串 "aaa" 。
示例 3: 输入：arr = ["a","b","a"], k = 3 输出：""
解释：唯一一个独一无二字符串是 "b" 。由于少于 3 个独一无二字符串，我们返回空字符串 ""。
↪。
提示：1 <= k <= arr.length <= 1000
1 <= arr[i].length <= 5
arr[i]只包含小写英文字母。
```

- 解题思路

```go
func kthDistinct(arr []string, k int) string {
        m := make(map[string]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]]++
        }
        for i := 0; i < len(arr); i++ {
                if m[arr[i]] == 1 {
                        k--
                        if k == 0 {
                                return arr[i]
                        }
                }
        }
        return ""
}
```

61.11 2057. 值相等的最小索引 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，返回 nums 中满足 i mod 10 == nums[i]␣
↪的最小下标 i ；
如果不存在这样的下标，返回 -1 。
x mod y 表示 x 除以 y 的余数 。
示例 1：输入：nums = [0,1,2] 输出：0
解释：i=0: 0 mod 10 = 0 == nums[0].
i=1: 1 mod 10 = 1 == nums[1].
i=2: 2 mod 10 = 2 == nums[2].
所有下标都满足 i mod 10 == nums[i] ，所以返回最小下标 0
示例 2：输入：nums = [4,3,2,1] 输出：2
解释：i=0: 0 mod 10 = 0 != nums[0].
i=1: 1 mod 10 = 1 != nums[1].
i=2: 2 mod 10 = 2 == nums[2].
i=3: 3 mod 10 = 3 != nums[3].
2 唯一一个满足 i mod 10 == nums[i] 的下标
示例 3：输入：nums = [1,2,3,4,5,6,7,8,9,0] 输出：-1
解释：不存在满足 i mod 10 == nums[i] 的下标
示例 4：输入：nums = [2,1,3,5,2] 输出：1
解释：1 是唯一一个满足 i mod 10 == nums[i] 的下标
提示：1 <= nums.length <= 100
0 <= nums[i] <= 9
```

- 解题思路

```
func smallestEqual(nums []int) int {
        for i := 0; i < len(nums); i++ {
                if i%10 == nums[i] {
                        return i
                }
        }
        return -1
}
```

61.12 2062. 统计字符串中的元音子字符串 (2)

- 题目

```
子字符串 是字符串中的一个连续（非空）的字符序列。
元音子字符串 是 仅 由元音（'a'、'e'、'i'、'o' 和 'u'）组成的一个子字符串，且必须包含全部五种 元音。
给你一个字符串 word ，统计并返回 word 中 元音子字符串的数目 。
示例 1：输入：word = "aeiouu" 输出：2
解释：下面列出 word 中的元音子字符串（斜体加粗部分）：
- "aeiouu"
- "aeiouu"
示例 2：输入：word = "unicornarihan" 输出：0
解释：word 中不含 5 种元音，所以也不会存在元音子字符串。
示例 3：输入：word = "cuaieuouac" 输出：7
解释：下面列出 word 中的元音子字符串（斜体加粗部分）：
- "cuaieuouac"
- "cuaieuouac"
- "cuaieuouac"
- "cuaieuouac"
- "cuaieuouac"
- "cuaieuouac"
- "cuaieuouac"
示例 4：输入：word = "bbaeixoubb" 输出：0
解释：所有包含全部五种元音的子字符串都含有辅音，所以不存在元音子字符串。
提示：1 <= word.length <= 100
word 仅由小写英文字母组成
```

- 解题思路

```
var m = map[byte]bool{'a': true, 'e': true, 'i': true, 'o': true, 'u': true}

func countVowelSubstrings(word string) int {
        n := len(word)
        res := 0
        for i := 0; i < n; i++ {
                temp := make(map[byte]bool)
                for j := i; j < n; j++ {
                        if m[word[j]] == false {
                                break
                        }
                        temp[word[j]] = true
                        if len(temp) == 5 {
                                res++
```

(续下页)

(接上页)

```
                        }
                }
        }
        return res
}

# 2
var m = map[byte]bool{'a': true, 'e': true, 'i': true, 'o': true, 'u': true}

func countVowelSubstrings(word string) int {
        n := len(word)
        res := 0
        temp := make(map[byte]int)
        count := 1
        start := -1
        for i := 0; i < n; i++ {
                if m[word[i]] == false {
                        temp = make(map[byte]int)
                        count = 1
                        start = i
                        continue
                }
                temp[word[i]]++
                for temp[word[start+count]] > 1 { // 左边多余的字母个数
                        temp[word[start+count]]--
                        count++
                }
                if len(temp) == 5 {
                        res = res + count
                }
        }
        return res
}
```

61.13 2068. 检查两个字符串是否几乎相等 (2)

- 题目

```
如果两个字符串 word1和 word2中从 'a'到 'z'每一个字母出现频率之差都 不超过3，
那么我们称这两个字符串word1 和word2 几乎相等。
给你两个长度都为n的字符串word1␣
↪和word2，如果word1和word2几乎相等，请你返回true，否则返回false。
```

(续下页)

(接上页)

```
一个字母 x的出现 频率指的是它在字符串中出现的次数。
示例 1: 输入: word1 = "aaaa", word2 = "bccb" 输出: false
解释: 字符串 "aaaa" 中有 4 个 'a' ，但是 "bccb" 中有 0 个 'a' 。
两者之差为 4 ，大于上限 3 。
示例 2: 输入: word1 = "abcdeef", word2 = "abaaacc" 输出: true
解释: word1 和 word2 中每个字母出现频率之差至多为 3 :
- 'a' 在 word1 中出现了 1 次，在 word2 中出现了 4 次，差为 3 。
- 'b' 在 word1 中出现了 1 次，在 word2 中出现了 1 次，差为 0 。
- 'c' 在 word1 中出现了 1 次，在 word2 中出现了 2 次，差为 1 。
- 'd' 在 word1 中出现了 1 次，在 word2 中出现了 0 次，差为 1 。
- 'e' 在 word1 中出现了 2 次，在 word2 中出现了 0 次，差为 2 。
- 'f' 在 word1 中出现了 1 次，在 word2 中出现了 0 次，差为 1 。
示例 3: 输入: word1 = "cccddabba", word2 = "babababab" 输出: true
解释: word1 和 word2 中每个字母出现频率之差至多为 3 :
- 'a' 在 word1 中出现了 2 次，在 word2 中出现了 4 次，差为 2 。
- 'b' 在 word1 中出现了 2 次，在 word2 中出现了 5 次，差为 3 。
- 'c' 在 word1 中出现了 3 次，在 word2 中出现了 0 次，差为 3 。
- 'd' 在 word1 中出现了 2 次，在 word2 中出现了 0 次，差为 2 。
提示: n == word1.length == word2.length
1 <= n <= 100
word1 和word2都只包含小写英文字母。
```

- 解题思路

```
func checkAlmostEquivalent(word1 string, word2 string) bool {
        a, b := [26]int{}, [26]int{}
        for i := 0; i < len(word1); i++ {
                a[int(word1[i]-'a')]++
        }
        for i := 0; i < len(word2); i++ {
                b[int(word2[i]-'a')]++
        }
        for i := 0; i < 26; i++ {
                if a[i]-b[i] > 3 || b[i]-a[i] > 3 {
                        return false
                }
        }
        return true
}

# 2
func checkAlmostEquivalent(word1 string, word2 string) bool {
        a := [26]int{}
        for i := 0; i < len(word1); i++ {
```

(续下页)

(接上页)

```
                a[int(word1[i]-'a')]++
        }
        for i := 0; i < len(word2); i++ {
                a[int(word2[i]-'a')]--
        }
        for i := 0; i < 26; i++ {
                if a[i] > 3 || a[i] < -3 {
                        return false
                }
        }
        return true
}
```

61.14 2073. 买票需要的时间 (2)

- 题目

```
有 n 个人前来排队买票，其中第 0 人站在队伍 最前方 ，第 (n - 1) 人站在队伍 最后方 。
给你一个下标从 0 开始的整数数组 tickets ，数组长度为 n ，其中第 i 人想要购买的票数为␣
↪tickets[i] 。
每个人买票都需要用掉 恰好 1 秒 。一个人 一次只能买一张票 ，如果需要购买更多票，
他必须走到 队尾 重新排队（瞬间␣
↪发生，不计时间）。如果一个人没有剩下需要买的票，那他将会 离开 队伍。
返回位于位置 k（下标从 0 开始）的人完成买票需要的时间（以秒为单位）。
示例 1：输入：tickets = [2,3,2], k = 2 输出：6
解释：- 第一轮，队伍中的每个人都买到一张票，队伍变为 [1, 2, 1] 。
- 第二轮，队伍中的每个都又都买到一张票，队伍变为 [0, 1, 0] 。
位置 2 的人成功买到 2 张票，用掉 3 + 3 = 6 秒。
示例 2：输入：tickets = [5,1,1,1], k = 0 输出：8
解释：- 第一轮，队伍中的每个人都买到一张票，队伍变为 [4, 0, 0, 0] 。
- 接下来的 4 轮，只有位置 0 的人在买票。
位置 0 的人成功买到 5 张票，用掉 4 + 1 + 1 + 1 + 1 = 8 秒。
提示：n == tickets.length
1 <= n <= 100
1 <= tickets[i] <= 100
0 <= k < n
```

- 解题思路

```
func timeRequiredToBuy(tickets []int, k int) int {
        res := 1
        for {
```

(续下页)

(接上页)

```
                for i := 0; i < len(tickets); i++ {
                        if tickets[i] == 0 {
                                continue
                        }
                        tickets[i]--
                        if tickets[i] == 0 && i == k {
                                return res
                        }
                        res++
                }
        }
}

# 2
func timeRequiredToBuy(tickets []int, k int) int {
        res := 0
        for i := 0; i < len(tickets); i++ {
                if i <= k {
                        res = res + min(tickets[k], tickets[i]) //␣
↪前面的人，最多跟第k个人一样多
                } else {
                        res = res + min(tickets[k]-1, tickets[i]) //␣
↪后面的人，少一次选择
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

61.15 2078. 两栋颜色不同且距离最远的房子 (2)

- 题目

```
街上有 n 栋房子整齐地排成一列，每栋房子都粉刷上了漂亮的颜色。
给你一个下标从 0 开始且长度为 n 的整数数组 colors ，其中 colors[i] 表示第 i␣
↪栋房子的颜色。
返回 两栋 颜色 不同 房子之间的 最大 距离。
第 i 栋房子和第 j 栋房子之间的距离是 abs(i - j) ，其中 abs(x) 是 x 的绝对值。
示例 1：输入：colors = [1,1,1,6,1,1,1] 输出：3
解释：上图中，颜色 1 标识成蓝色，颜色 6 标识成红色。
两栋颜色不同且距离最远的房子是房子 0 和房子 3 。
房子 0 的颜色是颜色 1 ，房子 3 的颜色是颜色 6 。两栋房子之间的距离是 abs(0 - 3) = 3 。
注意，房子 3 和房子 6 也可以产生最佳答案。
示例 2：输入：colors = [1,8,3,8,3] 输出：4
解释：上图中，颜色 1 标识成蓝色，颜色 8 标识成黄色，颜色 3 标识成绿色。
两栋颜色不同且距离最远的房子是房子 0 和房子 4 。
房子 0 的颜色是颜色 1 ，房子 4 的颜色是颜色 3 。两栋房子之间的距离是 abs(0 - 4) = 4 。
示例 3：输入：colors = [0,1] 输出：1
解释：两栋颜色不同且距离最远的房子是房子 0 和房子 1 。
房子 0 的颜色是颜色 0 ，房子 1 的颜色是颜色 1 。两栋房子之间的距离是 abs(0 - 1) = 1 。
提示：n ==colors.length
2 <= n <= 100
0 <= colors[i] <= 100
生成的测试数据满足 至少 存在 2 栋颜色不同的房子
```

- 解题思路

```
func maxDistance(colors []int) int {
        res := 0
        for i := 0; i < len(colors); i++ {
                for j := i + 1; j < len(colors); j++ {
                        if colors[i] != colors[j] {
                                res = max(res, j-i)
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

# 2
func maxDistance(colors []int) int {
        n := len(colors)
        if colors[0] != colors[n-1] {
                return n - 1
        }
        left, right := 1, n-2
        for colors[n-1] == colors[left] {
                left++
        }
        for colors[0] == colors[right] {
                right--
        }
        return max(right, n-1-left)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

61.16 2085. 统计出现过一次的公共字符串 (1)

- 题目

```
给你两个字符串数组words1和words2，请你返回在两个字符串数组中␣
↪都恰好出现一次的字符串的数目。
示例 1：输入：words1 = ["leetcode","is","amazing","as","is"], words2 = ["amazing",
↪"leetcode","is"] 输出：2
解释：- "leetcode" 在两个数组中都恰好出现一次，计入答案。
- "amazing" 在两个数组中都恰好出现一次，计入答案。
- "is" 在两个数组中都出现过，但在 words1 中出现了 2 次，不计入答案。
- "as" 在 words1 中出现了一次，但是在 words2 中没有出现过，不计入答案。
所以，有 2 个字符串在两个数组中都恰好出现了一次。
示例 2：输入：words1 = ["b","bb","bbb"], words2 = ["a","aa","aaa"] 输出：0
解释：没有字符串在两个数组中都恰好出现一次。
示例 3：输入：words1 = ["a","ab"], words2 = ["a","a","a","ab"] 输出：1
```

(续下页)

(接上页)

```
解释：唯一在两个数组中都出现一次的字符串是 "ab" 。
提示：1 <= words1.length, words2.length <= 1000
1 <= words1[i].length, words2[j].length <= 30
words1[i] 和words2[j]都只包含小写英文字母。
```

- 解题思路

```
func countWords(words1 []string, words2 []string) int {
        a, b := make(map[string]int), make(map[string]int)
        for i := 0; i < len(words1); i++ {
                a[words1[i]]++
        }
        for i := 0; i < len(words2); i++ {
                b[words2[i]]++
        }
        res := 0
        for k, v := range a {
                if v == 1 && b[k] == 1 {
                        res++
                }
        }
        return res
}
```

61.17 2089. 找出数组排序后的目标下标 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums 以及一个目标元素 target 。
目标下标 是一个满足nums[i] == target 的下标 i 。
将 nums 按 非递减 顺序排序后，返回由 nums 中目标下标组成的列表。
如果不存在目标下标，返回一个 空 列表。返回的列表必须按 递增 顺序排列。
示例 1：输入：nums = [1,2,5,2,3], target = 2 输出：[1,2]
解释：排序后，nums 变为 [1,2,2,3,5] 。
满足 nums[i] == 2 的下标是 1 和 2 。
示例 2：输入：nums = [1,2,5,2,3], target = 3 输出：[3]
解释：排序后，nums 变为 [1,2,2,3,5] 。
满足 nums[i] == 3 的下标是 3 。
示例 3：输入：nums = [1,2,5,2,3], target = 5 输出：[4]
解释：排序后，nums 变为 [1,2,2,3,5] 。
满足 nums[i] == 5 的下标是 4 。
示例 4：输入：nums = [1,2,5,2,3], target = 4 输出：[]
```

(续下页)

(接上页)

```
解释：nums 中不含值为 4 的元素。
提示：1 <= nums.length <= 100
1 <= nums[i], target <= 100
```

- 解题思路

```
func targetIndices(nums []int, target int) []int {
        sort.Ints(nums)
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        res = append(res, i)
                }
        }
        return res
}

# 2
func targetIndices(nums []int, target int) []int {
        res := make([]int, 0)
        start := 0
        count := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        count++
                } else if nums[i] < target {
                        start++
                }
        }
        for i := start; i < start+count; i++ {
                res = append(res, i)
        }
        return res
}
```

61.18 2094. 找出 3 位偶数 (2)

- 题目

```
给你一个整数数组 digits ，其中每个元素是一个数字（0 - 9）。数组中可能存在重复元素。
你需要找出 所有 满足下述条件且 互不相同 的整数：
该整数由 digits 中的三个元素按 任意 顺序 依次连接 组成。
```

(续下页)

(接上页)

```
该整数不含 前导零
该整数是一个 偶数
例如，给定的 digits 是 [1, 2, 3] ，整数 132 和 312 满足上面列出的全部条件。
将找出的所有互不相同的整数按 递增顺序 排列，并以数组形式返回。
示例 1：输入：digits = [2,1,3,0] 输出：[102,120,130,132,210,230,302,310,312,320]
解释：所有满足题目条件的整数都在输出数组中列出。
注意，答案数组中不含有 奇数 或带 前导零 的整数。
示例 2：输入：digits = [2,2,8,8,2] 输出：[222,228,282,288,822,828,882]
解释：同样的数字（0 - 9）在构造整数时可以重复多次，重复次数最多与其在 digits␣
↪中出现的次数一样。
在这个例子中，数字 8 在构造 288、828 和 882 时都重复了两次。
示例 3：输入：digits = [3,7,5] 输出：[]
解释：使用给定的 digits 无法构造偶数。
示例 4：输入：digits = [0,2,0,0] 输出：[200]
解释：唯一一个不含 前导零 且满足全部条件的整数是 200 。
示例 5：输入：digits = [0,0,0] 输出：[]
解释：构造的所有整数都会有 前导零 。因此，不存在满足题目条件的整数。
提示：3 <=digits.length <= 100
0 <= digits[i] <= 9
```

- 解题思路

```
func findEvenNumbers(digits []int) []int {
        m := make(map[int]bool)
        n := len(digits)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        for k := 0; k < n; k++ {
                                if i == j || j == k || i == k {
                                        continue
                                }
                                v := digits[i]*100 + digits[j]*10 + digits[k]
                                if 100 <= v && v%2 == 0 {
                                        m[v] = true
                                }
                        }
                }
        }
        arr := make([]int, 0)
        for k := range m {
                arr = append(arr, k)
        }
        sort.Ints(arr)
        return arr
```

(续下页)

(接上页)

```
}

# 2
func findEvenNumbers(digits []int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(digits); i++ {
                m[digits[i]]++
        }
        for i := 100; i < 1000; i = i + 2 {
                temp := make(map[int]int)
                value := i
                flag := true
                for value > 0 {
                        temp[value%10]++
                        if m[value%10] < temp[value%10] {
                                flag = false
                                break
                        }
                        value = value / 10
                }
                if flag == true {
                        res = append(res, i)
                }
        }
        return res
}
```

61.19 2099. 找到和最大的长度为 K 的子序列 (1)

- 题目

```
给你一个整数数组nums和一个整数k。你需要找到nums中长度为 k的␣
↪子序列，且这个子序列的和最大。
请你返回 任意 一个长度为k的整数子序列。
子序列定义为从一个数组里删除一些元素后，不改变剩下元素的顺序得到的数组。
示例 1：输入：nums = [2,1,3,3], k = 2 输出：[3,3]
解释：子序列有最大和：3 + 3 = 6 。
示例 2：输入：nums = [-1,-2,3,4], k = 3 输出：[-1,3,4]
解释：子序列有最大和：-1 + 3 + 4 = 6 。
示例 3：输入：nums = [3,4,3,3], k = 2 输出：[3,4]
解释：子序列有最大和：3 + 4 = 7 。
```

(续下页)

(接上页)

```
另一个可行的子序列为 [4, 3] 。
提示：1 <= nums.length <= 1000
-105<= nums[i] <= 105
1 <= k <= nums.length
```

- 解题思路

```
func maxSubsequence(nums []int, k int) []int {
        n := len(nums)
        arr := make([][2]int, len(nums))
        for i := 0; i < n; i++ {
                arr[i] = [2]int{i, nums[i]}
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] > arr[j][1]
        })
        sort.Slice(arr[:k], func(i, j int) bool {
                return arr[i][0] < arr[j][0]
        })
        res := make([]int, 0)
        for i := 0; i < k; i++ {
                res = append(res, arr[i][1])
        }
        return res
}
```

CHAPTER 62

2001-2100-Medium

62.1 2001. 可互换矩形的组数 (1)

- 题目

```
用一个下标从 0 开始的二维整数数组rectangles 来表示 n 个矩形，
其中 rectangles[i] = [widthi, heighti] 表示第 i 个矩形的宽度和高度。
如果两个矩形 i 和 j（i < j）的宽高比相同，则认为这两个矩形 可互换 。
更规范的说法是，两个矩形满足widthi/heighti == widthj/
↪heightj（使用实数除法而非整数除法），则认为这两个矩形 可互换 。
计算并返回rectangles 中有多少对 可互换 矩形。
示例 1：输入：rectangles = [[4,8],[3,6],[10,20],[15,30]] 输出：6
解释：下面按下标（从 0 开始）列出可互换矩形的配对情况：
- 矩形 0 和矩形 1 ：4/8 == 3/6
- 矩形 0 和矩形 2 ：4/8 == 10/20
- 矩形 0 和矩形 3 ：4/8 == 15/30
- 矩形 1 和矩形 2 ：3/6 == 10/20
- 矩形 1 和矩形 3 ：3/6 == 15/30
- 矩形 2 和矩形 3 ：10/20 == 15/30
示例 2：输入：rectangles = [[4,5],[7,8]] 输出：0
解释：不存在成对的可互换矩形。
提示：n == rectangles.length
1 <= n <= 105
rectangles[i].length == 2
1 <= widthi, heighti <= 105
```

- 解题思路

```
func interchangeableRectangles(rectangles [][]int) int64 {
        m := make(map[string]int64)
        res := int64(0)
        for i := 0; i < len(rectangles); i++ {
                a, b := rectangles[i][0], rectangles[i][1]
                c := gcd(a, b)
                m[fmt.Sprintf("%d,%d", a/c, b/c)]++
        }
        for _, v := range m {
                res = res + (v-1)*v/2
        }
        return res
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}
```

62.2 2002. 两个回文子序列长度的最大乘积 (3)

- 题目

```
给你一个字符串s，请你找到s中两个不相交回文子序列，使得它们长度的乘积最大。
两个子序列在原字符串中如果没有任何相同下标的字符，则它们是不相交的。
请你返回两个回文子序列长度可以达到的最大乘积。
子序列指的是从原字符串中删除若干个字符（可以一个也不删除）后，剩余字符不改变顺序而得到的结果。
如果一个字符串从前往后读和从后往前读一模一样，那么这个字符串是一个 回文字符串。
示例 1：输入：s = "leetcodecom" 输出：9
解释：最优方案是选择 "ete" 作为第一个子序列，"cdc" 作为第二个子序列。
它们的乘积为 3 * 3 = 9 。
示例 2：输入：s = "bb" 输出：1
解释：最优方案为选择 "b" （第一个字符）作为第一个子序列，"b"␣
↪（第二个字符）作为第二个子序列。
它们的乘积为 1 * 1 = 1 。
示例 3：输入：s = "accbcaxxcxx" 输出：25
解释：最优方案为选择 "accca" 作为第一个子序列，"xxcxx" 作为第二个子序列。
它们的乘积为 5 * 5 = 25 。
提示：2 <= s.length <= 12
```

(续下页)

(接上页)

s只含有小写英文字母。

- 解题思路

```
var res int

func maxProduct(s string) int {
        res = 0
        dfs(s, "", "", 0)
        return res
}

func dfs(s string, a, b string, index int) {
        if len(a) > 0 && len(b) > 0 &&
                isPalindrome(a, 0, len(a)-1) && isPalindrome(b, 0, len(b)-1) {
                res = max(res, len(a)*len(b))
        }
        if index == len(s) {
                return
        }
        dfs(s, a, b, index+1)                   // a,b都不选
        dfs(s, a+string(s[index]), b, index+1) // a不选
        dfs(s, a, b+string(s[index]), index+1) // b不选
}

func isPalindrome(s string, i, j int) bool {
        for i < j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
func maxProduct(s string) int {
        res := 0
        n := len(s)
        total := 1 << n
        arr := make([]int, 0)
        for i := 1; i < total; i++ {
                if judge(s, i) {
                        arr = append(arr, i)
                }
        }
        for i := 0; i < len(arr); i++ { // 枚举回文状态
                for j := i + 1; j < len(arr); j++ {
                        if arr[i]&arr[j] == 0 {
                                a, b := bits.OnesCount(uint(arr[i])), bits.
↪OnesCount(uint(arr[j]))
                                res = max(res, a*b)
                        }
                }
        }
        return res
}

func judge(s string, status int) bool {
        left, right := 0, len(s)-1
        for left < right {
                for left < right && (status&(1<<left)) == 0 {
                        left++
                }
                for left < right && (status&(1<<right)) == 0 {
                        right--
                }
                if s[left] != s[right] {
                        return false
                }
                left++
                right--
        }
        return true
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

# 3
func maxProduct(s string) int {
        res := 0
        n := len(s)
        total := 1 << n
        m := make(map[int]int, 0)
        for i := 1; i < total; i++ {
                if judge(s, i) {
                        m[i] = bits.OnesCount(uint(i))
                }
        }
        for i := 1; i < total; i++ { // 遍历状态
                for j := i; j > 0; j = (j - 1) & i { // 枚举子集
                        res = max(res, m[j]*m[j^i]) // 子集 * 子集的补集
                }
        }
        return res
}

func judge(s string, status int) bool {
        left, right := 0, len(s)-1
        for left < right {
                for left < right && (status&(1<<left)) == 0 {
                        left++
                }
                for left < right && (status&(1<<right)) == 0 {
                        right--
                }
                if s[left] != s[right] {
                        return false
                }
                left++
                right--
        }
        return true
}

func max(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return a
        }
        return b
}
```

62.3 2007. 从双倍数组中还原原数组 (2)

- 题目

```
一个整数数组original可以转变成一个 双倍数组changed，转变方式为将 original中每个元素␣
↪值乘以 2 加入数组中，
然后将所有元素 随机打乱。
给你一个数组changed，如果change是双倍数组，那么请你返回original数组，否则请返回空数组。
original的元素可以以任意顺序返回。
示例 1：输入：changed = [1,3,4,2,6,8] 输出：[1,3,4]
解释：一个可能的 original 数组为 [1,3,4] :
- 将 1 乘以 2 ，得到 1 * 2 = 2 。
- 将 3 乘以 2 ，得到 3 * 2 = 6 。
- 将 4 乘以 2 ，得到 4 * 2 = 8 。
其他可能的原数组方案为 [4,3,1] 或者 [3,1,4] 。
示例 2：输入：changed = [6,3,0,1] 输出：[]
解释：changed 不是一个双倍数组。
示例 3：输入：changed = [1] 输出：[]
解释：changed 不是一个双倍数组。
提示：1 <= changed.length <= 105
0 <= changed[i] <= 105
```

- 解题思路

```
func findOriginalArray(changed []int) []int {
        res := make([]int, 0)
        n := len(changed)
        if n%2 == 1 {
                return nil
        }
        sort.Ints(changed)
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                value := changed[i]
                if m[value] == 0 { // 不是双倍的元素
                        m[value*2]++ // 标记双倍
                        res = append(res, value)
```

(续下页)

(接上页)

```
                } else {
                        m[value]--
                        if m[value] == 0 {
                                delete(m, value)
                        }
                }
        }
        if len(m) == 0 {
                return res
        }
        return nil
}

# 2
func findOriginalArray(changed []int) []int {
        res := make([]int, 0)
        n := len(changed)
        if n%2 == 1 {
                return nil
        }
        sort.Ints(changed)
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                m[changed[i]]++
        }
        for i := 0; i < n; i++ {
                if m[changed[i]] != 0 {
                        res = append(res, changed[i])
                        m[changed[i]]--
                        m[changed[i]*2]--
                }
        }
        if len(res)*2 != n {
                return nil
        }
        return res
}
```

62.4 2008. 出租车的最大盈利 (4)

- 题目

```
你驾驶出租车行驶在一条有 n个地点的路上。这 n个地点从近到远编号为1到n，你想要从 1开到n，通过接乘客订单盈利。
你只能沿着编号递增的方向前进，不能改变方向。
乘客信息用一个下标从 0开始的二维数组rides表示，
其中rides[i] = [starti, endi, tipi]表示第i位乘客需要从地点starti前往endi，愿意支付tipi元的小费。
每一位 你选择接单的乘客i，你可以 盈利endi - starti + tipi元。你同时最多只能接一个订单。
给你 n和 rides，请你返回在最优接单方案下，你能盈利最多多少元。
注意：你可以在一个地点放下一位乘客，并在同一个地点接上另一位乘客。
示例 1：输入：n = 5, rides = [[2,5,4],[1,5,1]] 输出：7
解释：我们可以接乘客 0 的订单，获得 5 - 2 + 4 = 7 元。
示例 2：输入：n = 20, rides = [[1,6,1],[3,10,2],[10,12,3],[11,12,2],[12,15,2],[13,18,1]] 输出：20
解释：我们可以接以下乘客的订单：
- 将乘客 1 从地点 3 送往地点 10 ，获得 10 - 3 + 2 = 9 元。
- 将乘客 2 从地点 10 送往地点 12 ，获得 12 - 10 + 3 = 5 元。
- 将乘客 5 从地点 13 送往地点 18 ，获得 18 - 13 + 1 = 6 元。
我们总共获得 9 + 5 + 6 = 20 元。
提示：1 <= n <= 105
1 <= rides.length <= 3 * 104
rides[i].length == 3
1 <= starti < endi <= n
1 <= tipi <= 105
```

- 解题思路

```
func maxTaxiEarnings(n int, rides [][]int) int64 {
        m := len(rides)
        arr := make([][]int64, 0)
        for i := 0; i < m; i++ {
                a, b, c := rides[i][0], rides[i][1], rides[i][2]
                arr = append(arr, []int64{int64(a), int64(b), int64(b - a + c)})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][1] == arr[j][1] {
                        return arr[i][0] < arr[j][0]
                }
                return arr[i][1] < arr[j][1]
        })
```

(续下页)

(接上页)

```
        for i := 1; i < m; i++ {
                target := sort.Search(i, func(j int) bool {
                        return arr[j][1] > arr[i][0]
                })
                if target == 0 {
                        arr[i][2] = max(arr[i][2], arr[i-1][2])
                } else {
                        arr[i][2] = max(arr[i][2]+arr[target-1][2], arr[i-1][2])
                }
        }
        return arr[m-1][2]
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}

# 2
func maxTaxiEarnings(n int, rides [][]int) int64 {
        m := len(rides)
        arr := make([][]int64, 0)
        for i := 0; i < m; i++ {
                a, b, c := rides[i][0], rides[i][1], rides[i][2]
                arr = append(arr, []int64{int64(a), int64(b), int64(b - a + c)})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][1] == arr[j][1] {
                        return arr[i][0] < arr[j][0]
                }
                return arr[i][1] < arr[j][1]
        })
        dp := make([]int64, m)
        dp[0] = arr[0][2]
        for i := 1; i < m; i++ {
                left, right := 0, i-1
                for left < right {
                        mid := left + (right-left)/2
                        if arr[mid+1][1] <= arr[i][0] {
                                left = mid + 1
                        } else {
```

(续下页)

(接上页)

```
                                right = mid
                        }
                }
                if arr[left][1] <= arr[i][0] {
                        dp[i] = max(dp[i-1], dp[left]+arr[i][2])
                } else {
                        dp[i] = max(dp[i-1], arr[i][2])
                }
        }
        return dp[m-1]
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}

# 3
func maxTaxiEarnings(n int, rides [][]int) int64 {
        dp := make([]int, n+1) // dp[i]到达i位置的最大盈利
        arr := make([][][2]int, n+1)
        for i := 0; i < len(rides); i++ {
                a, b, c := rides[i][0], rides[i][1], rides[i][2]
                arr[b] = append(arr[b], [2]int{a, b - a + c})
        }
        for i := 1; i <= n; i++ {
                dp[i] = dp[i-1]
                for j := 0; j < len(arr[i]); j++ {
                        a, c := arr[i][j][0], arr[i][j][1]
                        dp[i] = max(dp[i], dp[a]+c)
                }
        }
        return int64(dp[n])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 4
func maxTaxiEarnings(n int, rides [][]int) int64 {
        dp := make([]int, n+1) // dp[i]到达i位置的最大盈利
        sort.Slice(rides, func(i, j int) bool {
                if rides[i][0] == rides[j][0] {
                        return rides[i][1] < rides[j][1]
                }
                return rides[i][0] < rides[j][0]
        })
        j := 1
        for i := 0; i < len(rides); i++ { // 遍历订单
                a, b, c := rides[i][0], rides[i][1], rides[i][2]
                for j < a { // 更新指针
                        j++
                        dp[j] = max(dp[j], dp[j-1])
                }
                dp[b] = max(dp[b], dp[j]+(b-a+c))
        }
        for ; j <= n; j++ {
                dp[j] = max(dp[j], dp[j-1])
        }
        return int64(dp[n])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.5 2012. 数组美丽值求和 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 nums 。对于每个下标 i (1 <= i <= nums.length -␣
↪2) , nums[i] 的 美丽值 等于:
2, 对于所有 0 <= j < i 且 i < k <= nums.length - 1 , 满足 nums[j] < nums[i] < nums[k]
1, 如果满足 nums[i - 1] < nums[i] < nums[i + 1] , 且不满足前面的条件
0, 如果上述条件全部不满足
返回符合 1 <= i <= nums.length - 2 的所有 nums[i] 的 美丽值的总和 。
```

(续下页)

(接上页)

```
示例 1: 输入: nums = [1,2,3] 输出: 2
解释: 对于每个符合范围 1 <= i <= 1 的下标 i :
- nums[1] 的美丽值等于 2
示例 2: 输入: nums = [2,4,6,4] 输出: 1
解释: 对于每个符合范围 1 <= i <= 2 的下标 i :
- nums[1] 的美丽值等于 1
- nums[2] 的美丽值等于 0
示例 3: 输入: nums = [3,2,1] 输出: 0
解释: 对于每个符合范围 1 <= i <= 1 的下标 i :
- nums[1] 的美丽值等于 0
提示: 3 <= nums.length <= 105
1 <= nums[i] <= 105
```

- 解题思路

```
func sumOfBeauties(nums []int) int {
        res := 0
        n := len(nums)
        arrA := make([]int, n)
        arrA[0] = nums[0]
        for i := 1; i < n; i++ {
                arrA[i] = max(nums[i], arrA[i-1])
        }
        arrB := make([]int, n)
        arrB[n-1] = nums[n-1]
        for i := n - 2; i >= 0; i-- {
                arrB[i] = min(nums[i], arrB[i+1])
        }
        for i := 1; i <= n-2; i++ {
                if arrA[i-1] < nums[i] && nums[i] < arrB[i+1] {
                        res = res + 2
                } else if nums[i-1] < nums[i] && nums[i] < nums[i+1] {
                        res = res + 1
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.6 2013. 检测正方形 (1)

- 题目

```
给你一个在 X-Y 平面上的点构成的数据流。设计一个满足下述要求的算法：
添加 一个在数据流中的新点到某个数据结构中。可以添加 重复的点，并会视作不同的点进行处理。
给你一个查询点，请你从数据结构中选出三个点，使这三个点和查询点一同构成一个 面积为正的 轴对齐正方形 ，统计 满足该要求的方案数目。
轴对齐正方形 是一个正方形，除四条边长度相同外，还满足每条边都与 x-轴 或 y-轴平行或垂直。
实现 DetectSquares 类：
DetectSquares() 使用空数据结构初始化对象
void add(int[] point) 向数据结构添加一个新的点 point = [x, y]
int count(int[] point) 统计按上述方式与点 point = [x, y] 共同构造 轴对齐正方形的方案数。
示例：输入：["DetectSquares", "add", "add", "add", "count", "count", "add", "count"]
[[], [[3, 10]], [[11, 2]], [[3, 2]], [[11, 10]], [[14, 8]], [[11, 2]], [[11, 10]]]
输出：[null, null, null, null, 1, 0, null, 2]
解释：DetectSquares detectSquares = new DetectSquares();
detectSquares.add([3, 10]);
detectSquares.add([11, 2]);
detectSquares.add([3, 2]);
detectSquares.count([11, 10]); // 返回 1 。你可以选择：
                               //   - 第一个，第二个，和第三个点
detectSquares.count([14, 8]);  // 返回 0 。查询点无法与数据结构中的这些点构成正方形。
detectSquares.add([11, 2]);    // 允许添加重复的点。
detectSquares.count([11, 10]); // 返回 2 。你可以选择：
                               //   - 第一个，第二个，和第三个点
                               //   - 第一个，第三个，和第四个点
提示：point.length == 2
0 <= x, y <= 1000
调用add 和 count 的 总次数 最多为 5000
```

- 解题思路

```
type DetectSquares struct {
        m map[int]map[int]int
}

func Constructor() DetectSquares {
        return DetectSquares{
                m: make(map[int]map[int]int),
        }
}

func (this *DetectSquares) Add(point []int) {
        a, b := point[0], point[1]
        if this.m[a] == nil {
                this.m[a] = make(map[int]int)
        }
        this.m[a][b]++
}

func (this *DetectSquares) Count(point []int) int {
        res := 0
        x, y := point[0], point[1]
        for a := range this.m[x] { // 遍历同一列的y坐标
                if a == y {
                        continue
                }
                length := abs(a - y) // 获取正方形边长
                b := x + length      // 右边
                if this.m[b] != nil && this.m[b][a] > 0 && this.m[b][y] > 0 {
                        res = res + this.m[b][a]*this.m[b][y]*this.m[x][a]
                }
                b = x - length // 左边
                if this.m[b] != nil && this.m[b][a] > 0 && this.m[b][y] > 0 {
                        res = res + this.m[b][a]*this.m[b][y]*this.m[x][a]
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

62.7 2017. 网格游戏 (1)

- 题目

```
给你一个下标从 0 开始的二维数组 grid ，数组大小为 2 x n ，其中 grid[r][c] 表示矩阵中␣
↪(r, c) 位置上的点数。
现在有两个机器人正在矩阵上参与一场游戏。
两个机器人初始位置都是 (0, 0) ，目标位置是 (1, n-1) 。
每个机器人只会 向右 ((r, c) 到 (r, c + 1)) 或 向下 ((r, c) 到 (r + 1, c)) 。
游戏开始，第一个 机器人从 (0, 0) 移动到 (1, n-1) ，并收集路径上单元格的全部点数。
对于路径上所有单元格 (r, c) ，途经后 grid[r][c] 会重置为 0 。
然后，第二个 机器人从 (0, 0) 移动到 (1, n-1)␣
↪，同样收集路径上单元的全部点数。注意，它们的路径可能会存在相交的部分。
第一个 机器人想要打击竞争对手，使 第二个 机器人收集到的点数 最小化 。
与此相对，第二个 机器人想要 最大化 自己收集到的点数。
两个机器人都发挥出自己的 最佳水平的前提下，返回 第二个 机器人收集到的 点数 。
示例 1：输入：grid = [[2,5,4],[1,5,1]] 输出：4
解释：第一个机器人的最佳路径如红色所示，第二个机器人的最佳路径如蓝色所示。
第一个机器人访问过的单元格将会重置为 0 。
第二个机器人将会收集到 0 + 0 + 4 + 0 = 4 个点。
示例 2：输入：grid = [[3,3,1],[8,5,2]] 输出：4
解释：第一个机器人的最佳路径如红色所示，第二个机器人的最佳路径如蓝色所示。
第一个机器人访问过的单元格将会重置为 0 。
第二个机器人将会收集到 0 + 3 + 1 + 0 = 4 个点。
示例 3：输入：grid = [[1,3,1,15],[1,3,3,1]] 输出：7
解释：第一个机器人的最佳路径如红色所示，第二个机器人的最佳路径如蓝色所示。
第一个机器人访问过的单元格将会重置为 0 。
第二个机器人将会收集到 0 + 1 + 3 + 3 + 0 = 7 个点。
提示：grid.length == 2
n == grid[r].length
1 <= n <= 5 * 104
1 <= grid[r][c] <= 105
```

- 解题思路

```go
func gridGame(grid [][]int) int64 {
        n := len(grid[0])
        a := make([]int, n) // 前缀和：上边：从右到左
        b := make([]int, n) // 前缀和：下边，从左到右
        for i := n - 1; i > 0; i-- {
                a[i-1] = a[i] + grid[0][i]
        }
        for i := 0; i < n-1; i++ {
                b[i+1] = b[i] + grid[1][i]
```

(续下页)

(接上页)

```
        }
        res := math.MaxInt64
        // 当第一个机器人选择第i点往下走的时候
        // 第一个机器人不需要选择最大值，只需要考虑让第二个机器人选择最小
        for i := 0; i < n; i++ {
                // 第二个机器人只有2个选择，选其中最大的
                // 1、从第0个点往下走，拿到b[i]值
                // 2、一直往右走，拿到a[i]值
                res = min(res, max(a[i], b[i]))
        }
        return int64(res)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.8 2018. 判断单词是否能放入填字游戏内 (1)

- 题目

```
给你一个m x␣
↪n的矩阵board，它代表一个填字游戏当前的状态。填字游戏格子中包含小写英文字母（已填入的单词），
表示空格的' '和表示障碍格子的'#'。
如果满足以下条件，那么我们可以 水平（从左到右 或者从右到左）或 竖直（从上到下␣
↪或者从下到上）填入一个单词：
该单词不占据任何'#'对应的格子。
每个字母对应的格子要么是' '（空格）要么与 board中已有字母 匹配。
如果单词是 水平放置的，那么该单词左边和右边 相邻格子不能为' '或小写英文字母。
如果单词是竖直放置的，那么该单词上边和下边相邻格子不能为' '或小写英文字母。
给你一个字符串word，如果word可以被放入board中，请你返回true，否则请返回false。
示例 1：输入：board = [["#", " ", "#"], [" ", " ", "#"], ["#", "c", " "]], word = "abc
```

(续下页)

(接上页)

```
↪" 输出：true
解释：单词 "abc" 可以如上图放置（从上往下）。
示例 2：输入：board = [[" ", "#", "a"], [" ", "#", "c"], [" ", "#", "a"]], word = "ac
↪" 输出：false
解释：无法放置单词，因为放置该单词后上方或者下方相邻格会有空格。
示例 3：输入：board = [["#", " ", "#"], [" ", " ", "#"], ["#", " ", "c"]], word = "ca
↪" 输出：true
解释：单词 "ca" 可以如上图放置（从右到左）。
提示：m == board.length
n == board[i].length
1 <= m * n <= 2 * 105
board[i][j]可能为' ', '#'或者一个小写英文字母。
1 <= word.length <= max(m, n)
word只包含小写英文字母。
```

- 解题思路

```
func placeWordInCrossword(board [][]byte, word string) bool {
        n, m := len(board), len(board[0])
        arr := make([][]byte, m)
        for i := 0; i < m; i++ {
                arr[i] = make([]byte, n)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr[j][i] = board[i][j] // 转置后统一按行处理
                }
        }
        return checkBoard(board, word) || checkBoard(arr, word)
}

func checkBoard(board [][]byte, word string) bool {
        for i := 0; i < len(board); i++ {
                arr := strings.Split(string(board[i]), "#") // 按#切割
                for j := 0; j < len(arr); j++ {
                        if len(arr[j]) != len(word) { // 长度不等
                                continue
                        }
                        left, right := true, true
                        for k := 0; k < len(word); k++ { // 正向：word == arr[j]
                                if arr[j][k] != ' ' && arr[j][k] != word[k] {
                                        left = false
                                        break
                                }
```

(续下页)

(接上页)

```
                        }
                        for k := 0; k < len(word); k++ { // 反向：word ==␣
↪reverse(arr[j])
                                if arr[j][k] != ' ' && arr[j][k] != word[len(word)-1-
↪k] {
                                        right = false
                                        break
                                }
                        }
                        if left == true || right == true {
                                return true
                        }
                }
        }
        return false
}
```

62.9 2023. 连接后等于目标字符串的字符串对 (2)

- 题目

```
给你一个 数字字符串数组 nums和一个 数字字符串 target，
请你返回 nums[i] + nums[j] （两个字符串连接）结果等于 target的下标 (i, j) （需满足 i !=␣
↪j）的数目。
示例 1：输入：nums = ["777","7","77","77"], target = "7777" 输出：4
解释：符合要求的下标对包括：
- (0, 1)："777" + "7"
- (1, 0)："7" + "777"
- (2, 3)："77" + "77"
- (3, 2)："77" + "77"
示例 2：输入：nums = ["123","4","12","34"], target = "1234" 输出：2
解释：符合要求的下标对包括
- (0, 1)："123" + "4"
- (2, 3)："12" + "34"
示例 3：输入：nums = ["1","1","1"], target = "11" 输出：6
解释：符合要求的下标对包括
- (0, 1)："1" + "1"
- (1, 0)："1" + "1"
- (0, 2)："1" + "1"
- (2, 0)："1" + "1"
- (1, 2)："1" + "1"
- (2, 1)："1" + "1"
```

(续下页)

(接上页)

```
提示：2 <= nums.length <= 100
1 <= nums[i].length <= 100
2 <= target.length <= 100
nums[i]和target只包含数字。
nums[i]和target不含有任何前导 0 。
```

- 解题思路

```
func numOfPairs(nums []string, target string) int {
        res := 0
        n := len(nums)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if nums[i]+nums[j] == target {
                                res++
                        }
                        if nums[j]+nums[i] == target {
                                res++
                        }
                }
        }
        return res
}

# 2
func numOfPairs(nums []string, target string) int {
        res := 0
        n := len(nums)
        m := make(map[string]int)
        for i := 0; i < n; i++ {
                m[nums[i]]++
        }
        for i := 1; i < len(target); i++ {
                a, b := target[:i], target[i:]
                if a == b {
                        res = res + m[a]*(m[a]-1)
                } else {
                        res = res + m[a]*m[b]
                }
        }
        return res
}
```

62.10 2024. 考试的最大困扰度 (3)

- 题目

```
一位老师正在出一场由 n道判断题构成的考试，每道题的答案为 true （用 'T' 表示）或者
↪false （用 'F'表示）。
老师想增加学生对自己做出答案的不确定性，方法是最大化有
↪连续相同结果的题数。（也就是连续出现 true 或者连续出现 false）。
给你一个字符串answerKey，其中answerKey[i]是第 i个问题的正确结果。
除此以外，还给你一个整数 k，表示你能进行以下操作的最多次数：
每次操作中，将问题的正确答案改为'T' 或者'F'（也就是将 answerKey[i] 改为'T'或者'F'）。
请你返回在不超过 k次操作的情况下，最大连续 'T'或者 'F'的数目。
示例 1：输入：answerKey = "TTFF", k = 2 输出：4
解释：我们可以将两个 'F' 都变为 'T' ，得到 answerKey = "TTTT" 。
总共有四个连续的 'T' 。
示例 2：输入：answerKey = "TFFT", k = 1 输出：3
解释：我们可以将最前面的 'T' 换成 'F' ，得到 answerKey = "FFFT" 。
或者，我们可以将第二个 'T' 换成 'F' ，得到 answerKey = "TFFF" 。
两种情况下，都有三个连续的 'F' 。
示例 3：输入：answerKey = "TTFTTFTT", k = 1 输出：5
解释：我们可以将第一个 'F' 换成 'T' ，得到 answerKey = "TTTTTFTT" 。
或者我们可以将第二个 'F' 换成 'T' ，得到 answerKey = "TTFTTTTT" 。
两种情况下，都有五个连续的 'T' 。
提示：n == answerKey.length
1 <= n <= 5 * 104
answerKey[i]要么是'T' ，要么是'F'
1 <= k <= n
```

- 解题思路

```
func maxConsecutiveAnswers(answerKey string, k int) int {
        n := len(answerKey)
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                if answerKey[i] == 'T' {
                        arr[i] = 1
                }
        }
        a := longestOnes(arr, k)
        arr = make([]int, n)
        for i := 0; i < n; i++ {
                if answerKey[i] == 'F' {
                        arr[i] = 1
                }
```

(续下页)

(接上页)

```
        }
        b := longestOnes(arr, k)
        return max(a, b)
}

// leetcode 1004.最大连续1的个数III
func longestOnes(A []int, K int) int {
        res := 0
        left, right := 0, 0
        count := 0
        for right = 0; right < len(A); right++ {
                if A[right] == 0 {
                        count++
                }
                for count > K {
                        if A[left] == 0 {
                                count--
                        }
                        left++
                }
                res = max(res, right-left+1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxConsecutiveAnswers(answerKey string, k int) int {
        n := len(answerKey)
        res := 0
        left, right := 0, 0
        countA, countB := 0, 0
        for ; right < n; right++ {
                if answerKey[right] == 'T' {
                        countA++
                } else {
                        countB++
```

(续下页)

(接上页)

```
                }
                for countA > k && countB > k { // 都大于k时，滑动窗口才移动
                        if answerKey[left] == 'T' {
                                countA--
                        } else {
                                countB--
                        }
                        left++
                }
                res = max(res, right-left+1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxConsecutiveAnswers(answerKey string, k int) int {
        arr := []byte(answerKey)
        return max(longestOnes(arr, k, 'T'), longestOnes(arr, k, 'F'))
}

func longestOnes(A []byte, K int, target byte) int {
        res := 0
        left, right := 0, 0
        count := 0
        for right = 0; right < len(A); right++ {
                if A[right] == target {
                        count++
                }
                for count > K {
                        if A[left] == target {
                                count--
                        }
                        left++
                }
                res = max(res, right-left+1)
        }
```

(续下页)

(接上页)

```
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.11 2028. 找出缺失的观测数据 (2)

- 题目

```
现有一份 n + m次投掷单个 六面 骰子的观测数据，骰子的每个面从 1 到 6 编号。
观测数据中缺失了 n 份，你手上只拿到剩余m 次投掷的数据。幸好你有之前计算过的这 n + m␣
↪次投掷数据的 平均值 。
给你一个长度为 m 的整数数组 rolls ，其中rolls[i] 是第 i 次观测的值。同时给你两个整数␣
↪mean 和 n 。
返回一个长度为 n 的数组，包含所有缺失的观测数据，且满足这 n + m 次投掷的 平均值 是␣
↪mean 。
如果存在多组符合要求的答案，只需要返回其中任意一组即可。如果不存在答案，返回一个空数组。
k个数字的 平均值 为这些数字求和后再除以k 。
注意 mean 是一个整数，所以 n + m 次投掷的总和需要被n + m整除。
示例 1：输入：rolls = [3,2,4,3], mean = 4, n = 2 输出：[6,6]
解释：所有 n + m 次投掷的平均值是 (3 + 2 + 4 + 3 + 6 + 6) / 6 = 4 。
示例 2：输入：rolls = [1,5,6], mean = 3, n = 4 输出：[2,3,2,2]
解释：所有 n + m 次投掷的平均值是 (1 + 5 + 6 + 2 + 3 + 2 + 2) / 7 = 3 。
示例 3：输入：rolls = [1,2,3,4], mean = 6, n = 4 输出：[]
解释：无论丢失的 4 次数据是什么，平均值都不可能是 6 。
示例 4：输入：rolls = [1], mean = 3, n = 1 输出：[5]
解释：所有 n + m 次投掷的平均值是 (1 + 5) / 2 = 3 。
提示：m == rolls.length
1 <= n, m <= 105
1 <= rolls[i], mean <= 6
```

- 解题思路

```
func missingRolls(rolls []int, mean int, n int) []int {
        sum := 0
        m := len(rolls)
        for i := 0; i < m; i++ {
```

(续下页)

(接上页)

```
                sum = sum + rolls[i]
        }
        total := mean * (n + m)
        left := total - sum
        if left > n*6 || left < n {
                return nil
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = left / n // 平均分配
        }
        for i := 0; i < left%n; i++ {
                res[i] = res[i] + 1
        }
        return res
}

# 2
func missingRolls(rolls []int, mean int, n int) []int {
        sum := 0
        m := len(rolls)
        for i := 0; i < m; i++ {
                sum = sum + rolls[i]
        }
        total := mean * (n + m)
        left := total - sum
        if left > n*6 || left < n {
                return nil
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = 1
        }
        left = left - n
        for i := 0; i < n; i++ {
                if left < 6 {
                        res[i] = res[i] + left
                        break
                }
                res[i] = res[i] + 5
                left = left - 5
        }
        return res
}
```

62.12 2029. 石子游戏 IX(1)

- 题目

```
Alice 和 Bob 再次设计了一款新的石子游戏。现有一行 n
↪个石子，每个石子都有一个关联的数字表示它的价值。
给你一个整数数组 stones ，其中 stones[i] 是第 i 个石子的价值。
Alice 和 Bob 轮流进行自己的回合，Alice 先手。每一回合，玩家需要从
↪stones中移除任一石子。
如果玩家移除石子后，导致 所有已移除石子 的价值总和 可以被 3 整除，那么该玩家就
↪输掉游戏 。
如果不满足上一条，且移除后没有任何剩余的石子，那么 Bob 将会直接获胜（即便是在 Alice
↪的回合）。
假设两位玩家均采用最佳 决策。如果 Alice 获胜，返回 true ；如果 Bob 获胜，返回 false 。
示例 1：输入：stones = [2,1] 输出：true
解释：游戏进行如下：
- 回合 1：Alice 可以移除任意一个石子。
- 回合 2：Bob 移除剩下的石子。
已移除的石子的值总和为 1 + 2 = 3 且可以被 3 整除。因此，Bob 输，Alice 获胜。
示例 2：输入：stones = [2] 输出：false
解释：Alice 会移除唯一一个石子，已移除石子的值总和为 2 。
由于所有石子都已移除，且值总和无法被 3 整除，Bob 获胜。
示例 3：输入：stones = [5,1,2,4,3] 输出：false
解释：Bob 总会获胜。其中一种可能的游戏进行方式如下：
- 回合 1：Alice 可以移除值为 1 的第 2 个石子。已移除石子值总和为 1 。
- 回合 2：Bob 可以移除值为 3 的第 5 个石子。已移除石子值总和为 = 1 + 3 = 4 。
- 回合 3：Alices 可以移除值为 4 的第 4 个石子。已移除石子值总和为 = 1 + 3 + 4 = 8 。
- 回合 4：Bob 可以移除值为 2 的第 3 个石子。已移除石子值总和为 = 1 + 3 + 4 + 2 = 10.
- 回合 5：Alice 可以移除值为 5 的第 1 个石子。已移除石子值总和为 = 1 + 3 + 4 + 2 + 5
↪= 15.
Alice 输掉游戏，因为已移除石子值总和（15）可以被 3 整除，Bob 获胜。
提示：1 <= stones.length <= 105
1 <= stones[i] <= 104
```

- 解题思路

```
func stoneGameIX(stones []int) bool {
        m := make(map[int]int)
        for i := 0; i < len(stones); i++ {
                m[stones[i]%3]++
        }
        a, b, c := m[0], m[1], m[2] // 0,1,2的个数
        // 以1开头：1、(1、2)、1...
        // 以2开头：2、(2、1)、2...
```

(续下页)

(接上页)

```
        if a%2 == 0 { // 0为偶数个，可以抵消
                // 获胜策略：选择较少的1或者2
                // 1、1=2 => Alice赢（以1开始或者2开始都赢）
                // 不等的时候
                // 2.1、1>2 => Alice以2开头=> 2、2、1...(2、1)..␣
↪最后Bob没有2可选，只能选择1，Alice就赢了
                // 2.2 2>1 => Alice以1开头=> 1、1、2...(1、2)..␣
↪最后Bob没有1可选，只能选择2，Alice就赢了
                return b > 0 && c > 0
        }
        if a%2 == 1 {
                // 奇数个0
                // 获胜策略：选取较多的1或者2
                // 需要差值大于2个Alice才赢
                // 1>2 => Alice以1开头 => 1、(1、0)、2、1、1 / 1、(0、1)、2、1、1
                // 2>1 => Alice以2开头 => 2、(2、0)、1、2、2 / 2、(0、2)、1、2、2
                return b-c > 2 || c-b > 2
        }
        return false
}
```

62.13 2033. 获取单值网格的最小操作数 (1)

- 题目

```
给你一个大小为m x n 的二维整数网格 grid 和一个整数 x 。每一次操作，你可以对 grid␣
↪中的任一元素 加 x 或 减 x 。
单值网格 是全部元素都相等的网格。
返回使网格化为单值网格所需的 最小 操作数。如果不能，返回 -1 。
示例 1：输入：grid = [[2,4],[6,8]], x = 2 输出：4
解释：可以执行下述操作使所有元素都等于 4 ：
- 2 加 x 一次。
- 6 减 x 一次。
- 8 减 x 两次。
共计 4 次操作。
示例 2：输入：grid = [[1,5],[2,3]], x = 1 输出：5
解释：可以使所有元素都等于 3 。
示例 3：输入：grid = [[1,2],[3,4]], x = 2 输出：-1
解释：无法使所有元素相等。
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 105
```

(续下页)

(接上页)

```
1 <= m * n <= 105
1 <= x, grid[i][j] <= 104
```

- 解题思路

```
func minOperations(grid [][]int, x int) int {
        n, m := len(grid), len(grid[0])
        arr := make([]int, 0)
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr = append(arr, grid[i][j])
                }
        }
        sort.Ints(arr)
        target := arr[len(arr)/2]
        res := 0
        for i := 0; i < len(arr); i++ {
                if abs(target-arr[i])%x != 0 {
                        return -1
                }
                res = res + abs(target-arr[i])/x
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

62.14 2034. 股票价格波动 (2)

- 题目

```
给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳和该时间点股票对应的 价格。
不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是
如果两个有相同时间戳的记录出现在数据流中，前一条记录视为错误记录，后出现的记录␣
↪更正前一条错误的记录。
请你设计一个算法，实现：
更新␣
```

(续下页)

(接上页)

```
↪股票在某一时间戳的股票价格，如果有之前同一时间戳的价格，这一操作将更正之前的错误价格。
找到当前记录里 最新股票价格。最新股票价格定义为时间戳最晚的股票价格。
找到当前记录里股票的 最高价格。
找到当前记录里股票的 最低价格。
请你实现StockPrice类：
StockPrice()初始化对象，当前无股票价格记录。
void update(int timestamp, int price)在时间点 timestamp更新股票价格为 price。
int current()返回股票 最新价格。
int maximum()返回股票 最高价格。
int minimum()返回股票 最低价格。
示例 1：输入：["StockPrice", "update", "update", "current", "maximum", "update",
"maximum", "update", "minimum"]
[[], [1, 10], [2, 5], [], [], [1, 3], [], [4, 2], []]
输出：[null, null, null, 5, 10, null, 5, null, 2]
解释：StockPrice stockPrice = new StockPrice();
stockPrice.update(1, 10); // 时间戳为 [1] ，对应的股票价格为 [10] 。
stockPrice.update(2, 5);  // 时间戳为 [1,2] ，对应的股票价格为 [10,5] 。
stockPrice.current();     // 返回 5 ，最新时间戳为 2 ，对应价格为 5 。
stockPrice.maximum();     // 返回 10 ，最高价格的时间戳为 1 ，价格为 10 。
stockPrice.update(1, 3);  // 之前时间戳为 1 的价格错误，价格更新为 3 。
                          // 时间戳为 [1,2] ，对应股票价格为 [3,5] 。
stockPrice.maximum();     // 返回 5 ，更正后最高价格为 5 。
stockPrice.update(4, 2);  // 时间戳为 [1,2,4] ，对应价格为 [3,5,2] 。
stockPrice.minimum();     // 返回 2 ，最低价格时间戳为 4 ，价格为 2 。
提示：1 <= timestamp, price <= 109
update，current，maximum和minimum总 调用次数不超过105。
current，maximum和minimum被调用时，update操作 至少已经被调用过 一次。
```

- 解题思路

```
type StockPrice struct {
        *redblacktree.Tree
        m                 map[int]int
        curTime, curPrice int
}

func Constructor() StockPrice {
        return StockPrice{
                Tree: redblacktree.NewWithIntComparator(),
                m:    make(map[int]int),
        }
}

func (this *StockPrice) Update(timestamp int, price int) {
```

(续下页)

(接上页)

```
        if v, ok := this.m[timestamp]; ok {
                this.remove(v) // 删除
        }
        this.put(price) // 添加记录
        this.m[timestamp] = price
        if timestamp >= this.curTime { // 更新
                this.curTime, this.curPrice = timestamp, price
        }
}

func (this *StockPrice) put(price int) {
        count := 0
        if v, ok := this.Get(price); ok {
                count = v.(int)
        }
        this.Put(price, count+1)
}

func (this *StockPrice) remove(price int) {
        if v, ok := this.Get(price); ok && v.(int) > 1 {
                this.Put(price, v.(int)-1)
        } else {
                this.Remove(price)
        }
}

func (this *StockPrice) Current() int {
        return this.curPrice
}

func (this *StockPrice) Maximum() int {
        return this.Right().Key.(int)
}

func (this *StockPrice) Minimum() int {
        return this.Left().Key.(int)
}

# 2
type StockPrice struct {
        maxH, minH        *mixHeap
        curTime, curPrice int
        m                 map[int]int
```

(续下页)

(接上页)

```
}

func Constructor() StockPrice {
        return StockPrice{
                maxH: &mixHeap{isBig: true},
                minH: &mixHeap{isBig: false},
                m:    make(map[int]int), // 保存最新的时间戳=>价格数据
        }
}

func (this *StockPrice) Update(timestamp int, price int) {
        if timestamp >= this.curTime { // 更新
                this.curTime, this.curPrice = timestamp, price
        }
        this.maxH.push([]int{timestamp, price})
        this.minH.push([]int{timestamp, price})
        this.m[timestamp] = price // 时间戳=>价格
}

func (this *StockPrice) Current() int {
        return this.curPrice
}

func (this *StockPrice) Maximum() int {
        for this.maxH.Len() > 0 {
                top := this.maxH.Top()
                if top[1] == this.m[top[0]] { // 跟map里面的价格对的上，直接返回
                        return top[1]
                }
                this.maxH.pop() // 剔除旧数据
        }
        return 0
}

func (this *StockPrice) Minimum() int {
        for this.minH.Len() > 0 {
                top := this.minH.Top()
                if top[1] == this.m[top[0]] { // 跟map里面的价格对的上，直接返回
                        return top[1]
                }
                this.minH.pop() // 剔除旧数据
        }
        return 0
```

(续下页)

(接上页)

```
}

type mixHeap struct {
        arr   [][]int
        isBig bool
}

func (m *mixHeap) Len() int {
        return len(m.arr)
}

func (m *mixHeap) Swap(i, j int) {
        m.arr[i], m.arr[j] = m.arr[j], m.arr[i]
}

func (m *mixHeap) Less(i, j int) bool {
        if m.isBig {
                return m.arr[i][1] > m.arr[j][1]
        }
        return m.arr[i][1] < m.arr[j][1]
}

func (m *mixHeap) Push(x interface{}) {
        m.arr = append(m.arr, x.([]int))
}

func (m *mixHeap) Pop() interface{} {
        value := (m.arr)[len(m.arr)-1]
        m.arr = (m.arr)[:len(m.arr)-1]
        return value
}

func (m *mixHeap) push(x []int) {
        heap.Push(m, x)
}

func (m *mixHeap) pop() []int {
        return heap.Pop(m).([]int)
}

func (m *mixHeap) Top() []int {
        if m.Len() > 0 {
                return m.arr[0]
```

(续下页)

(接上页)

```
        }
        return nil
}
```

62.15 2038. 如果相邻两个颜色均相同则删除当前颜色 (2)

- 题目

```
总共有 n个颜色片段排成一列，每个颜色片段要么是'A'要么是'B'。
给你一个长度为n的字符串colors，其中colors[i]表示第i个颜色片段的颜色。
Alice 和 Bob 在玩一个游戏，他们 轮流从这个字符串中删除颜色。Alice 先手。
如果一个颜色片段为 'A'且 相邻两个颜色都是颜色 'A'，那么 Alice 可以删除该颜色片段。
Alice不可以删除任何颜色'B'片段。
如果一个颜色片段为 'B'且 相邻两个颜色都是颜色 'B'，那么 Bob 可以删除该颜色片段。
Bob 不可以删除任何颜色 'A'片段。
Alice 和 Bob 不能从字符串两端删除颜色片段。
如果其中一人无法继续操作，则该玩家 输掉游戏且另一玩家 获胜。
假设 Alice 和 Bob 都采用最优策略，如果 Alice 获胜，请返回true，否则 Bob␣
↪获胜，返回false。
示例 1：输入：colors = "AAABABB" 输出：true
解释：AAABABB -> AABABB
Alice 先操作。
她删除从左数第二个 'A' ，这也是唯一一个相邻颜色片段都是 'A' 的 'A' 。
现在轮到 Bob 操作。
Bob 无法执行任何操作，因为没有相邻位置都是 'B' 的颜色片段 'B' 。
因此，Alice 获胜，返回 true 。
示例 2：输入：colors = "AA" 输出：false
解释：Alice 先操作。
只有 2 个 'A' 且它们都在字符串的两端，所以她无法执行任何操作。
因此，Bob 获胜，返回 false 。
示例 3：输入：colors = "ABBBBBBBAAA" 输出：false
解释：ABBBBBBBAAA -> ABBBBBBBAA
Alice 先操作。
她唯一的选择是删除从右数起第二个 'A' 。
ABBBBBBBAA -> ABBBBBBAA
接下来轮到 Bob 操作。
他有许多选择，他可以选择任何一个 'B' 删除。
然后轮到 Alice 操作，她无法删除任何片段。
所以 Bob 获胜，返回 false 。
提示：1 <=colors.length <= 105
colors只包含字母'A'和'B'
```

- 解题思路

```
func winnerOfGame(colors string) bool {
        arrA := strings.Split(colors, "B")
        arrB := strings.Split(colors, "A")
        countA, countB := 0, 0
        for i := 0; i < len(arrA); i++ {
                if len(arrA[i]) >= 3 {
                        countA = countA + len(arrA[i]) - 2
                }
        }
        for i := 0; i < len(arrB); i++ {
                if len(arrB[i]) >= 3 {
                        countB = countB + len(arrB[i]) - 2
                }
        }
        if countA > countB {
                return true
        }
        return false
}

# 2
func winnerOfGame(colors string) bool {
        countA := 0
        countB := 0
        if len(colors) <= 2 {
                return false
        }
        for i := 2; i < len(colors); i++ {
                if colors[i-1] == colors[i-2] && colors[i-1] == colors[i] {
                        if colors[i] == 'A' {
                                countA++
                        } else {
                                countB++
                        }
                }
        }
        if countA > countB {
                return true
        }
        return false
}
```

62.16 2039. 网络空闲的时刻 (2)

- 题目

```
给你一个有 n个服务器的计算机网络，服务器编号为0到n - 1。
同时给你一个二维整数数组edges，其中edges[i] = [ui, vi]表示服务器ui␣
↪和vi之间有一条信息线路，
在一秒内它们之间可以传输任意数目的信息。再给你一个长度为␣
↪n且下标从0开始的整数数组patience。
题目保证所有服务器都是␣
↪相通的，也就是说一个信息从任意服务器出发，都可以通过这些信息线路直接或间接地到达任何其他服务器。
编号为 0的服务器是 主服务器，其他服务器为␣
↪数据服务器。每个数据服务器都要向主服务器发送信息，并等待回复。
信息在服务器之间按 最优线路传输，也就是说每个信息都会以 最少时间到达主服务器。
主服务器会处理 所有新到达的信息并 立即按照每条信息来时的路线 反方向 发送回复信息。
在 0秒的开始，所有数据服务器都会发送各自需要处理的信息。
从第 1秒开始，每一秒最␣
↪开始时，每个数据服务器都会检查它是否收到了主服务器的回复信息（包括新发出信息的回复信息）：
如果还没收到任何回复信息，那么该服务器会周期性重发信息。数据服务器i每patience[i]秒都会重发一条信息，
也就是说，数据服务器i在上一次发送信息给主服务器后的 patience[i]秒␣
↪后会重发一条信息给主服务器。
否则，该数据服务器不会重发信息。
当没有任何信息在线路上传输或者到达某服务器时，该计算机网络变为 空闲状态。
请返回计算机网络变为 空闲状态的最早秒数。
示例 1：输入：edges = [[0,1],[1,2]], patience = [0,2,1] 输出：8
解释：0 秒最开始时，
- 数据服务器 1 给主服务器发出信息（用 1A 表示）。
- 数据服务器 2 给主服务器发出信息（用 2A 表示）。
1 秒时，
- 信息 1A 到达主服务器，主服务器立刻处理信息 1A 并发出 1A 的回复信息。
- 数据服务器 1 还没收到任何回复。距离上次发出信息过去了 1 秒（1 < patience[1] =␣
↪2），所以不会重发信息。
- 数据服务器 2 还没收到任何回复。距离上次发出信息过去了 1 秒（1 == patience[2] =␣
↪1），所以它重发一条信息（用 2B 表示）。
2 秒时，
- 回复信息 1A 到达服务器 1 ，服务器 1 不会再重发信息。
- 信息 2A 到达主服务器，主服务器立刻处理信息 2A 并发出 2A 的回复信息。
- 服务器 2 重发一条信息（用 2C 表示）。
...
4 秒时，
- 回复信息 2A 到达服务器 2 ，服务器 2 不会再重发信息。
...
7 秒时，回复信息 2D 到达服务器 2 。
从第 8 秒开始，不再有任何信息在服务器之间传输，也不再有信息到达服务器。
```

(续下页)

(接上页)

```
所以第 8 秒是网络变空闲的最早时刻。
示例 2：输入：edges = [[0,1],[0,2],[1,2]], patience = [0,10,10] 输出：3
解释：数据服务器 1 和 2 第 2 秒初收到回复信息。
从第 3 秒开始，网络变空闲。
提示：n == patience.length
2 <= n <= 105
patience[0] == 0
对于1 <= i < n ，满足1 <= patience[i] <= 105
1 <= edges.length <= min(105, n * (n - 1) / 2)
edges[i].length == 2
0 <= ui, vi < n
ui != vi
不会有重边。
每个服务器都直接或间接与别的服务器相连。
```

- 解题思路

```
func networkBecomesIdle(edges [][]int, patience []int) int {
        maxValue := math.MaxInt32 / 10
        n := len(patience)
        arr := make([][][2]int, n) // 邻接表：i=>j的集合
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a=>b
                arr[a] = append(arr[a], [2]int{b, 1})
                arr[b] = append(arr[b], [2]int{a, 1})
        }
        dis := make([]int, n) // k到其他点的距离
        for i := 0; i < n; i++ {
                dis[i] = maxValue
        }
        dis[0] = 0
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, [2]int{0, 0})
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([2]int) // 距离起点最近的点
                a := node[0]
                if dis[a] < node[1] { // 大于最短距离，跳过
                        continue
                }
                for i := 0; i < len(arr[a]); i++ {
                        b, c := arr[a][i][0], arr[a][i][1]
                        if dis[a]+c < dis[b] { // 更新距离
                                dis[b] = dis[a] + c
```

(续下页)

(接上页)

```
                                heap.Push(&intHeap, [2]int{b, dis[b]})
                        }
                }
        }
        res := 0
        for i := 1; i < n; i++ {
                total := (2*dis[i]-1)/patience[i]*patience[i] + 2*dis[i]
                res = max(res, total)
        }
        return res + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type IntHeap [][2]int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i][1] < h[j][1]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.([2]int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

(续下页)

(接上页)

```
# 2
func networkBecomesIdle(edges [][]int, patience []int) int {
        n := len(patience)
        arr := make([][]int, n) // 邻接表：i=>j的集合
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a=>b
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        res := 0
        visited := make([]bool, n)
        visited[0] = true
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{0, 0})
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                a, dis := node[0], node[1]
                if a != 0 {
                        total := (2*dis-1)/patience[a]*patience[a] + 2*dis
                        res = max(res, total)
                }
                for i := 0; i < len(arr[a]); i++ {
                        b := arr[a][i]
                        if visited[b] == false {
                                queue = append(queue, [2]int{b, dis + 1})
                                visited[b] = true
                        }
                }
        }
        return res + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.17 2043. 简易银行系统 (1)

- 题目

```
你的任务是为一个很受欢迎的银行设计一款程序，以自动化执行所有传入的交易（转账，存款和取款）。
银行共有 n 个账户，编号从 1 到 n 。
每个账号的初始余额存储在一个下标从 0 开始的整数数组 balance中，其中第 (i + 1)␣
↪个账户的初始余额是 balance[i] 。
请你执行所有 有效的 交易。如果满足下面全部条件，则交易 有效 ：
指定的账户数量在 1 和 n 之间，且
取款或者转账需要的钱的总数 小于或者等于 账户余额。
实现 Bank 类：
Bank(long[] balance) 使用下标从 0 开始的整数数组 balance 初始化该对象。
boolean transfer(int account1, int account2, long money)
从编号为account1 的账户向编号为 account2 的账户转帐 money 美元。如果交易成功，返回␣
↪true ，否则，返回 false 。
boolean deposit(int account, long money) 向编号为account 的账户存款 money 美元。
如果交易成功，返回 true ；否则，返回 false 。
boolean withdraw(int account, long money) 从编号为 account 的账户取款 money 美元。
如果交易成功，返回 true ；否则，返回 false 。
示例：输入： ["Bank", "withdraw", "transfer", "deposit", "transfer", "withdraw"]
[[[10, 100, 20, 50, 30]], [3, 10], [5, 1, 20], [5, 20], [3, 4, 15], [10, 50]]
输出：[null, true, true, true, false, false]
解释：Bank bank = new Bank([10, 100, 20, 50, 30]);
bank.withdraw(3, 10);    // 返回 true ，账户 3 的余额是 $20 ，所以可以取款 $10 。
                         // 账户 3 余额为 $20 - $10 = $10 。
bank.transfer(5, 1, 20); // 返回 true ，账户 5 的余额是 $30 ，所以可以转账 $20 。
                         // 账户 5 的余额为 $30 - $20 = $10 ，账户 1 的余额为 $10 +
↪$20 = $30 。
bank.deposit(5, 20);     // 返回 true ，可以向账户 5 存款 $20 。
                         // 账户 5 的余额为 $10 + $20 = $30 。
bank.transfer(3, 4, 15); // 返回 false ，账户 3 的当前余额是 $10 。
                         // 所以无法转账 $15 。
bank.withdraw(10, 50);   // 返回 false ，交易无效，因为账户 10 并不存在。
提示：n == balance.length
1 <= n, account, account1, account2 <= 105
0 <= balance[i], money <= 1012
transfer, deposit, withdraw 三个函数，每个 最多调用 104 次
```

- 解题思路

```
type Bank struct {
        arr []int64
        n   int
```

(续下页)

(接上页)

```
}

func Constructor(balance []int64) Bank {
        return Bank{arr: balance, n: len(balance)}
}

func (this *Bank) Transfer(account1 int, account2 int, money int64) bool {
        if account1 > this.n || account2 > this.n || this.arr[account1-1] < money {
                return false
        }
        this.arr[account1-1] = this.arr[account1-1] - money
        this.arr[account2-1] = this.arr[account2-1] + money
        return true
}

func (this *Bank) Deposit(account int, money int64) bool {
        if account > this.n {
                return false
        }
        this.arr[account-1] = this.arr[account-1] + money
        return true
}

func (this *Bank) Withdraw(account int, money int64) bool {
        if account > this.n || this.arr[account-1] < money {
                return false
        }
        this.arr[account-1] = this.arr[account-1] - money
        return true
}
```

62.18 2044. 统计按位或能得到最大值的子集数目 (4)

- 题目

```
给你一个整数数组 nums ，请你找出 nums 子集 按位或 可能得到的 最大值␣
↪，并返回按位或能得到最大值的 不同非空子集的数目 。
如果数组 a 可以由数组 b 删除一些元素（或不删除）得到，则认为数组 a 是数组 b 的一个␣
↪子集 。
如果选中的元素下标位置不一样，则认为两个子集 不同 。
对数组 a 执行 按位或，结果等于 a[0] OR a[1] OR ... OR a[a.length - 1] （下标从 0␣
↪开始）。
```

(续下页)

(接上页)

```
示例 1：输入：nums = [3,1] 输出：2
解释：子集按位或能得到的最大值是 3 。有 2 个子集按位或可以得到 3 ：
- [3]
- [3,1]
示例 2：输入：nums = [2,2,2] 输出：7
解释：[2,2,2] 的所有非空子集的按位或都可以得到 2 。总共有 23 - 1 = 7 个子集。
示例 3：输入：nums = [3,2,1,5] 输出：6
解释：子集按位或可能的最大值是 7 。有 6 个子集按位或可以得到 7 ：
- [3,5]
- [3,1,5]
- [3,2,5]
- [3,2,1,5]
- [2,5]
- [2,1,5]
提示：1 <= nums.length <= 16
1 <= nums[i] <= 105
```

- 解题思路

```
func countMaxOrSubsets(nums []int) int {
        n := len(nums)
        total := 1 << n
        sum := make([]int, total)
        for i := 0; i < n; i++ { // 每次添加1个
                count := 1 << i
                for j := 0; j < count; j++ {
                        sum[count|j] = sum[j] | nums[i] // 按位或运算：j前面补1=>
↪子集和加上tasks[i]
                }
        }
        maxValue := 0
        for i := 0; i < total; i++ {
                maxValue = max(maxValue, sum[i])
        }
        res := 0
        for i := 0; i < total; i++ {
                if sum[i] == maxValue {
                        res++
                }
        }
        return res
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 2
func countMaxOrSubsets(nums []int) int {
        n := len(nums)
        total := 1 << n
        sum := make([]int, total)
        for i := 0; i < total; i++ { // 枚举状态
                for j := 0; j < n; j++ { // 枚举该位
                        if (i & (1 << j)) > 0 {
                                sum[i] = sum[i] | nums[j]
                        }
                }
        }
        maxValue := 0
        for i := 0; i < total; i++ {
                maxValue = max(maxValue, sum[i])
        }
        res := 0
        for i := 0; i < total; i++ {
                if sum[i] == maxValue {
                        res++
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func countMaxOrSubsets(nums []int) int {
        n := len(nums)
        total := 1 << n
        maxValue := 0
```

(续下页)

(接上页)

```
        res := 0
        for i := 0; i < total; i++ { // 枚举状态
                value := 0
                for j := 0; j < n; j++ { // 枚举该位
                        if (i & (1 << j)) > 0 {
                                value = value | nums[j]
                        }
                }
                if value > maxValue {
                        maxValue = value
                        res = 1
                } else if value == maxValue {
                        res++
                }
        }
        return res
}

# 4
var maxValue int
var res int

func countMaxOrSubsets(nums []int) int {
        n := len(nums)
        maxValue = 0
        res = 0
        for i := 0; i < n; i++ {
                maxValue = maxValue | nums[i]
        }
        dfs(nums, 0, 0)
        return res
}

func dfs(nums []int, index, sum int) {
        if index == len(nums) {
                if sum == maxValue {
                        res++
                }
                return
        }
        dfs(nums, index+1, sum)
        dfs(nums, index+1, sum|nums[index])
}
```

62.19 2048. 下一个更大的数值平衡数 (2)

- 题目

```
如果整数 x 满足：对于每个数位d ，这个数位恰好 在 x 中出现 d 次。那么整数 x 就是一个
↪数值平衡数 。
给你一个整数 n ，请你返回 严格大于 n 的 最小数值平衡数 。
示例 1：输入：n = 1 输出：22
解释：22 是一个数值平衡数，因为：
- 数字 2 出现 2 次
这也是严格大于 1 的最小数值平衡数。
示例 2：输入：n = 1000 输出：1333
解释：1333 是一个数值平衡数，因为：
- 数字 1 出现 1 次。
- 数字 3 出现 3 次。
这也是严格大于 1000 的最小数值平衡数。
注意，1022 不能作为本输入的答案，因为数字 0 的出现次数超过了 0 。
示例 3：输入：n = 3000 输出：3133
解释：3133 是一个数值平衡数，因为：
- 数字 1 出现 1 次。
- 数字 3 出现 3 次。
这也是严格大于 3000 的最小数值平衡数。
提示：0 <= n <= 106
```

- 解题思路

```
func nextBeautifulNumber(n int) int {
        for i := n + 1; ; i++ {
                if judge(i) == true {
                        return i
                }
        }
}

func judge(a int) bool {
        s := strconv.Itoa(a)
        m := make(map[int]int)
        for i := 0; i < len(s); i++ {
                m[int(s[i]-'0')]++
        }
        for k, v := range m {
                if k != v {
                        return false
                }
```

(续下页)

(接上页)

```
        }
        return true
}
```

62.20 2049. 统计最高分的节点数目 (3)

- 题目

```
给你一棵根节点为 0 的二叉树，它总共有 n个节点，节点编号为0到n - 1。
同时给你一个下标从0开始的整数数组parents表示这棵树，其中parents[i]是节点 i的父节点。
由于节点 0是根，所以parents[0] == -1。
一个子树的 大小为这个子树内节点的数目。每个节点都有一个与之关联的分数。
求出某个节点分数的方法是，将这个节点和与它相连的边全部 删除，剩余部分是若干个非空子树，
这个节点的 分数为所有这些子树 大小的乘积。
请你返回有 最高得分节点的 数目。
示例1:输入：parents = [-1,2,0,2,0] 输出：3
解释：- 节点 0 的分数为：3 * 1 = 3
- 节点 1 的分数为：4 = 4
- 节点 2 的分数为：1 * 1 * 2 = 2
- 节点 3 的分数为：4 = 4
- 节点 4 的分数为：4 = 4
最高得分为 4 ，有三个节点得分为 4 （分别是节点 1，3 和 4 ）。
示例 2：输入：parents = [-1,2,0]
输出：2
解释：
- 节点 0 的分数为：2 = 2
- 节点 1 的分数为：2 = 2
- 节点 2 的分数为：1 * 1 = 1
最高分数为 2 ，有两个节点分数为 2 （分别为节点 0 和 1 ）。
提示：n == parents.length
2 <= n <= 105
parents[0] == -1
对于i != 0，有0 <= parents[i] <= n - 1
parents表示一棵二叉树。
```

- 解题思路

```
var arr [][]int
var sum []int

func countHighestScoreNodes(parents []int) int {
```

(续下页)

(接上页)

```
        res := 0
        maxValue := 0
        n := len(parents)
        sum = make([]int, n)
        arr = make([][]int, n)
        for i := 1; i < n; i++ { // 邻接表
                a, b := i, parents[i] // b=>a
                arr[b] = append(arr[b], a)
        }
        dfs(0)                    // 先统计
        for i := 0; i < n; i++ { // 后计算
                value := 1
                for j := 0; j < len(arr[i]); j++ { // 计算左右子树乘积
                        if sum[arr[i][j]] > 0 {
                                value = value * sum[arr[i][j]]
                        }
                }
                if parents[i] != -1 { // 计算去掉以根节点位i的剩余部分
                        if sum[parents[i]]-sum[i] > 0 {
                                value = value * (sum[0] - sum[i]) // 根节点总数-
↪节点i总数
                        }
                }
                if value > maxValue {
                        maxValue = value
                        res = 1
                } else if value == maxValue {
                        res++
                }
        }
        return res
}

func dfs(root int) int {
        count := 1
        for i := 0; i < len(arr[root]); i++ {
                count = count + dfs(arr[root][i])
        }
        sum[root] = count
        return count
}

# 2
```

(续下页)

(接上页)

```
var arr [][]int
var n int
var maxValue int
var res int

func countHighestScoreNodes(parents []int) int {
        res = 0
        maxValue = 1
        n = len(parents)
        arr = make([][]int, n)
        for i := 1; i < n; i++ { // 邻接表
                a, b := i, parents[i] // b=>a
                arr[b] = append(arr[b], a)
        }
        dfs(0)
        return res
}

func dfs(root int) int {
        count := 1
        value := 1
        for i := 0; i < len(arr[root]); i++ { // 计算左右子树乘积
                size := dfs(arr[root][i])
                count = count + size
                value = value * size
        }
        if root > 0 { // 计算去掉以根节点位i的剩余部分
                value = value * (n - count)
        }
        if value > maxValue {
                maxValue = value
                res = 1
        } else if value == maxValue {
                res++
        }
        return count
}

# 3
var n int
var maxValue int
var res int
var left, right []int
```

(续下页)

(接上页)

```
func countHighestScoreNodes(parents []int) int {
        res = 0
        maxValue = 1
        n = len(parents)
        left, right = make([]int, n), make([]int, n)
        for i := 0; i < n; i++ {
                left[i], right[i] = -1, -1
        }
        for i := 1; i < n; i++ { // 建立左右子树关系
                a, b := i, parents[i] // b=>a
                if left[b] == -1 {    // 优先放在左子树
                        left[b] = a
                } else {
                        right[b] = a
                }
        }
        dfs(0)
        return res
}

func dfs(root int) int {
        if root == -1 {
                return 0
        }
        a, b := dfs(left[root]), dfs(right[root])
        value := max(a, 1) * max(b, 1) * max(n-a-b-1, 1)
        if value > maxValue {
                maxValue = value
                res = 1
        } else if value == maxValue {
                res++
        }
        return a + b + 1 // 左子树+右子树+1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.21 2054. 两个最好的不重叠活动 (2)

- 题目

```
给你一个下标从 0开始的二维整数数组events，其中events[i] = [startTimei, endTimei,␣
↪valuei]。
第i个活动开始于startTimei，结束于endTimei，如果你参加这个活动，那么你可以得到价值valuei。
你 最多可以参加两个时间不重叠活动，使得它们的价值之和 最大。
请你返回价值之和的 最大值。
注意，活动的开始时间和结束时间是␣
↪包括在活动时间内的，也就是说，你不能参加两个活动且它们之一的开始时间等于另一个活动的结束时间。
更具体的，如果你参加一个活动，且结束时间为 t，那么下一个活动必须在t +␣
↪1或之后的时间开始。
示例 1:输入：events = [[1,3,2],[4,5,2],[2,4,3]] 输出：4
解释：选择绿色的活动 0 和 1 ，价值之和为 2 + 2 = 4 。
示例 2：输入：events = [[1,3,2],[4,5,2],[1,5,5]] 输出：5
解释：选择活动 2 ，价值和为 5 。
示例 3：输入：events = [[1,5,3],[1,5,1],[6,6,5]] 输出：8
解释：选择活动 0 和 2 ，价值之和为 3 + 5 = 8 。
提示：2 <= events.length <= 105
events[i].length == 3
1 <= startTimei <= endTimei <= 109
1 <= valuei <= 106
```

- 解题思路

```
func maxTwoEvents(events [][]int) int {
        res := 0
        sort.Slice(events, func(i, j int) bool {
                return events[i][0] < events[j][0]
        })
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        prevMaxValue := 0
        for i := 0; i < len(events); i++ {
                a, b, c := events[i][0], events[i][1], events[i][2]
                for intHeap.Len() > 0 && intHeap[0][1] < a { // 之前的结束时间 小于␣
↪当前的开始时间
                        value := heap.Pop(&intHeap).([3]int)[2] //␣
↪小根堆取值：按结束时间从小到大
                        prevMaxValue = max(prevMaxValue, value) // 更新之前最大值
                }
                res = max(res, prevMaxValue+c)
                heap.Push(&intHeap, [3]int{a, b, c})
```

(续下页)

(接上页)

```
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

type IntHeap [][3]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][1] < h[j][1] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([3]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 2
func maxTwoEvents(events [][]int) int {
        arr := make([][3]int, 0)
        for i := 0; i < len(events); i++ {
                a, b, c := events[i][0], events[i][1], events[i][2]
                arr = append(arr, [3]int{a, 0, c})
                arr = append(arr, [3]int{b, 1, c}) // 拆分为2块
        }
        sort.Slice(arr, func(i, j int) bool { // 按时间排序
                if arr[i][0] == arr[j][0] {
                        return arr[i][1] < arr[j][1]
                }
                return arr[i][0] < arr[j][0]
        })
        res := 0
        prevMaxValue := 0
        for i := 0; i < len(arr); i++ {
                if arr[i][1] == 0 { // 为开始时间
```

(续下页)

(接上页)

```
                        res = max(res, arr[i][2]+prevMaxValue)
                } else {
                        prevMaxValue = max(prevMaxValue, arr[i][2]) // 更新之前最大值
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.22 2055. 蜡烛之间的盘子 (2)

- 题目

```
给你一个长桌子，桌子上盘子和蜡烛排成一列。给你一个下标从 0开始的字符串s，
它只包含字符'*' 和'|'，其中'*'表示一个 盘子，'|'表示一支蜡烛。
同时给你一个下标从 0开始的二维整数数组queries，
其中queries[i] = [lefti, righti]表示 子字符串s[lefti...righti]（包含左右端点的字符）。
对于每个查询，你需要找到 子字符串中在 两支蜡烛之间的盘子的 数目。
如果一个盘子在 子字符串中左边和右边 都至少有一支蜡烛，那么这个盘子满足在␣
↪两支蜡烛之间。
比方说，s = "||**||**|*"，查询[3, 8]，表示的是子字符串"*||**|"。
子字符串中在两支蜡烛之间的盘子数目为2，子字符串中右边两个盘子在它们左边和右边 都␣
↪至少有一支蜡烛。
请你返回一个整数数组answer，其中answer[i]是第i个查询的答案。
示例 1:输入：s = "**|**|***|", queries = [[2,5],[5,9]] 输出：[2,3]
解释：- queries[0] 有两个盘子在蜡烛之间。
- queries[1] 有三个盘子在蜡烛之间。
示例 2:输入：s = "***|**|*****|**||**|*", queries = [[1,17],[4,5],[14,17],[5,11],[15,
↪16]] 输出：[9,0,0,0,0]
解释：- queries[0] 有 9 个盘子在蜡烛之间。
- 另一个查询没有盘子在蜡烛之间。
提示：3 <= s.length <= 105
s只包含字符'*' 和'|'。
1 <= queries.length <= 105
queries[i].length == 2
0 <= lefti <= righti < s.length
```

- 解题思路

```
func platesBetweenCandles(s string, queries [][]int) []int {
        n := len(s)
        res := make([]int, len(queries))
        sum := make([]int, n+1)
        left, right := make([]int, n), make([]int, n)
        prev := -1
        for i := 0; i < n; i++ {
                sum[i+1] = sum[i]
                if s[i] == '|' { // 蜡烛
                        prev = i
                } else {
                        sum[i+1]++
                }
                left[i] = prev
        }
        prev = n
        for i := n - 1; i >= 0; i-- {
                if s[i] == '|' { // 蜡烛
                        prev = i
                }
                right[i] = prev
        }
        for i := 0; i < len(queries); i++ {
                a, b := right[queries[i][0]], left[queries[i][1]] //␣
↪位子：右边第一个蜡烛，左边第一个蜡烛
                if a < b {                                              // 满足的条件下
                        res[i] = sum[b] - sum[a]
                }
        }
        return res
}

# 2
func platesBetweenCandles(s string, queries [][]int) []int {
        n := len(s)
        res := make([]int, len(queries))
        arr := make([]int, 0)
        for i := 0; i < n; i++ {
                if s[i] == '|' { // 盘子
                        arr = append(arr, i)
                }
        }
        for i := 0; i < len(queries); i++ {
```

(续下页)

(接上页)

```
                a, b := queries[i][0], queries[i][1]
                l := sort.SearchInts(arr, a)
                r := sort.SearchInts(arr, b)
                if r == len(arr) || arr[r] != b { // r超出范围或者找到的下标不是目标数
                        r--
                }
                if l < r {
                        res[i] = arr[r] - arr[l] - (r - l) // 总个数-盘子个数
                }
        }
        return res
}
```

62.23 2058. 找出临界点之间的最小和最大距离 (2)

- 题目

```
链表中的 临界点 定义为一个 局部极大值点 或 局部极小值点 。
如果当前节点的值 严格大于 前一个节点和后一个节点，那么这个节点就是一个 局部极大值点 。
如果当前节点的值 严格小于 前一个节点和后一个节点，那么这个节点就是一个 局部极小值点 。
注意：节点只有在同时存在前一个节点和后一个节点的情况下，才能成为一个 局部极大值点 /␣
↪极小值点 。
给你一个链表 head ，返回一个长度为 2 的数组 [minDistance, maxDistance] ，
其中 minDistance 是任意两个不同临界点之间的最小距离，maxDistance␣
↪是任意两个不同临界点之间的最大距离。
如果临界点少于两个，则返回 [-1, -1] 。
示例 1：输入：head = [3,1] 输出：[-1,-1]
解释：链表 [3,1] 中不存在临界点。
示例 2：输入：head = [5,3,1,2,5,1,2] 输出：[1,3]
解释：存在三个临界点：
- [5,3,1,2,5,1,2]：第三个节点是一个局部极小值点，因为 1 比 3 和 2 小。
- [5,3,1,2,5,1,2]：第五个节点是一个局部极大值点，因为 5 比 2 和 1 大。
- [5,3,1,2,5,1,2]：第六个节点是一个局部极小值点，因为 1 比 5 和 2 小。
第五个节点和第六个节点之间距离最小。minDistance = 6 - 5 = 1 。
第三个节点和第六个节点之间距离最大。maxDistance = 6 - 3 = 3 。
示例 3：输入：head = [1,3,2,2,3,2,2,2,7] 输出：[3,3]
解释：存在两个临界点：
- [1,3,2,2,3,2,2,2,7]：第二个节点是一个局部极大值点，因为 3 比 1 和 2 大。
- [1,3,2,2,3,2,2,2,7]：第五个节点是一个局部极大值点，因为 3 比 2 和 2 大。
最小和最大距离都存在于第二个节点和第五个节点之间。
因此，minDistance 和 maxDistance 是 5 - 2 = 3 。
注意，最后一个节点不算一个局部极大值点，因为它之后就没有节点了。
```

(续下页)

(接上页)

```
示例 4：输入：head = [2,3,3,2] 输出：[-1,-1]
解释：链表 [2,3,3,2] 中不存在临界点。
提示：链表中节点的数量在范围 [2, 105] 内
1 <= Node.val <= 105
```

- 解题思路

```
func nodesBetweenCriticalPoints(head *ListNode) []int {
        a, b, c := head, head.Next, head.Next.Next // 题目保证数量范围>=2
        maxDis, minDis := math.MinInt32, math.MaxInt32
        index := 1
        prev := 0  // 前一个位置
        first := 0 // 第一个位置
        for c != nil {
                if (a.Val < b.Val && b.Val > c.Val) ||
                        (a.Val > b.Val && b.Val < c.Val) {
                        if first == 0 {
                                first = index // 第一次出现
                        }
                        maxDis = max(maxDis, index-first) // 当前位置减去第一次位置
                        if 0 < prev {
                                minDis = min(minDis, index-prev) //␣
↪当前位置减去前一次位置
                        }
                        prev = index
                }
                a, b, c = b, c, c.Next
                index++
        }
        if minDis == math.MaxInt32 {
                return []int{-1, -1}
        }
        return []int{minDis, maxDis}
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return b
        }
        return a
}

# 2
func nodesBetweenCriticalPoints(head *ListNode) []int {
        a, b, c := head, head.Next, head.Next.Next // 题目保证数量范围>=2
        arr := make([]int, 0)
        index := 1
        for c != nil {
                if (a.Val < b.Val && b.Val > c.Val) ||
                        (a.Val > b.Val && b.Val < c.Val) {
                        arr = append(arr, index)
                }
                a, b, c = b, c, c.Next
                index++
        }
        if len(arr) < 2 {
                return []int{-1, -1}
        }
        minDis := math.MaxInt32
        for i := 0; i < len(arr)-1; i++ {
                minDis = min(minDis, arr[i+1]-arr[i])
        }
        return []int{minDis, arr[len(arr)-1] - arr[0]}
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

62.24 2059. 转化数字的最小运算数 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，该数组由 互不相同 的数字组成。另给你两个整数 start 和 goal 。
整数 x 的值最开始设为 start ，你打算执行一些运算使 x 转化为 goal 。你可以对数字 x 重复执行下述运算：
如果 0 <= x <= 1000 ，那么，对于数组中的任一下标 i（0 <= i < nums.length），可以将 x 设为下述任一值：
x + nums[i]
x - nums[i]
x ^ nums[i] （按位异或 XOR）
注意，你可以按任意顺序使用每个 nums[i] 任意次。使 x 越过 0 <= x <= 1000 范围的运算同样可以生效，
但该该运算执行后将不能执行其他运算。
返回将 x = start 转化为 goal 的最小操作数；如果无法完成转化，则返回 -1 。
示例 1：输入：nums = [1,3], start = 6, goal = 4 输出：2
解释：可以按 6 → 7 → 4 的转化路径进行，只需执行下述 2 次运算：
- 6 ^ 1 = 7
- 7 ^ 3 = 4
示例 2：输入：nums = [2,4,12], start = 2, goal = 12 输出：2
解释：可以按 2 → 14 → 12 的转化路径进行，只需执行下述 2 次运算：
- 2 + 12 = 14
- 14 - 2 = 12
示例 3：输入：nums = [3,5,7], start = 0, goal = -4 输出：2
解释：可以按 0 → 3 → -4 的转化路径进行，只需执行下述 2 次运算：
- 0 + 3 = 3
- 3 - 7 = -4
注意，最后一步运算使 x 超过范围 0 <= x <= 1000 ，但该运算仍然可以生效。
示例 4：输入：nums = [2,8,16], start = 0, goal = 1 输出：-1
解释：无法将 0 转化为 1
示例 5：输入：nums = [1], start = 0, goal = 3 输出：3
解释：可以按 0 → 1 → 2 → 3 的转化路径进行，只需执行下述 3 次运算：
- 0 + 1 = 1
- 1 + 1 = 2
- 2 + 1 = 3
提示：1 <= nums.length <= 1000
-109 <= nums[i], goal <= 109
0 <= start <= 1000
start != goal
nums 中的所有整数互不相同
```

- 解题思路

```
func minimumOperations(nums []int, start int, goal int) int {
        m := make(map[int]bool)
        m[start] = true
        queue := make([]int, 0)
        queue = append(queue, start)
        count := 1
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        for j := 0; j < len(nums); j++ {
                                temp := []int{queue[i] + nums[j], queue[i] - nums[j],␣
↪queue[i] ^ nums[j]}
                                for k := 0; k < len(temp); k++ {
                                        if temp[k] == goal {
                                                return count
                                        }
                                        if 0 <= temp[k] && temp[k] <= 1000 &&␣
↪m[temp[k]] == false {
                                                m[temp[k]] = true
                                                queue = append(queue, temp[k])
                                        }
                                }
                        }
                }
                count++
                queue = queue[length:]
        }
        return -1
}
```

62.25 2063. 所有子字符串中的元音 (1)

- 题目

```
给你一个字符串 word ，返回 word 的所有子字符串中 元音的总数 ，元音是指 'a'、'e'、'i'、
↪'o' 和 'u' 。
子字符串 是字符串中一个连续（非空）的字符序列。
注意：由于对 word 长度的限制比较宽松，答案可能超过有符号 32␣
↪位整数的范围。计算时需当心。
示例 1：输入：word = "aba" 输出：6
解释：所有子字符串是："a"、"ab"、"aba"、"b"、"ba" 和 "a" 。
- "b" 中有 0 个元音
```

(续下页)

(接上页)

```
- "a"、"ab"、"ba" 和 "a" 每个都有 1 个元音
- "aba" 中有 2 个元音
因此，元音总数 = 0 + 1 + 1 + 1 + 1 + 2 = 6 。
示例 2：输入：word = "abc" 输出：3
解释：所有子字符串是："a"、"ab"、"abc"、"b"、"bc" 和 "c" 。
- "a"、"ab" 和 "abc" 每个都有 1 个元音
- "b"、"bc" 和 "c" 每个都有 0 个元音
因此，元音总数 = 1 + 1 + 1 + 0 + 0 + 0 = 3 。
示例 3：输入：word = "ltcd" 输出：0
解释："ltcd" 的子字符串均不含元音。
示例 4：输入：word = "noosabasboosa" 输出：237
解释：所有子字符串中共有 237 个元音。
提示：1 <= word.length <= 105
word 由小写英文字母组成
```

- 解题思路

```
var m = map[byte]bool{'a': true, 'e': true, 'i': true, 'o': true, 'u': true}

func countVowels(word string) int64 {
        n := len(word)
        res := int64(0)
        for i := 0; i < n; i++ {
                if m[word[i]] == true {
                        // 计算当前字符出现在多少种组合里面
                        // 总数：left * right
                        res = res + int64(i+1)*int64(n-i) //␣
↪左边个数（包含当前）*右边个数
                }
        }
        return res
}
```

62.26 2064. 分配给商店的最多商品的最小值 (2)

- 题目

```
给你一个整数n，表示有n间零售商店。总共有m种产品，每种产品的数目用一个下标从␣
↪0开始的整数数组quantities表示，
其中quantities[i]表示第i种商品的数目。
你需要将 所有商品分配到零售商店，并遵守这些规则：
一间商店 至多只能有 一种商品 ，但一间商店拥有的商品数目可以为任意件。
```

(续下页)

(接上页)

```
分配后，每间商店都会被分配一定数目的商品（可能为 0件）。
用x表示所有商店中分配商品数目的最大值，你希望 x越小越好。也就是说，你想␣
↪最小化分配给任意商店商品数目的 最大值。
请你返回最小的可能的x。
示例 1: 输入: n = 6, quantities = [11,6] 输出: 3
解释: 一种最优方案为:
- 11 件种类为 0 的商品被分配到前 4 间商店，分配数目分别为: 2, 3, 3, 3 。
- 6 件种类为 1 的商品被分配到另外 2 间商店，分配数目分别为: 3, 3 。
分配给所有商店的最大商品数目为 max(2, 3, 3, 3, 3, 3) = 3 。
示例 2: 输入: n = 7, quantities = [15,10,10] 输出: 5
解释: 一种最优方案为:
- 15 件种类为 0 的商品被分配到前 3 间商店，分配数目为: 5, 5, 5 。
- 10 件种类为 1 的商品被分配到接下来 2 间商店，数目为: 5, 5 。
- 10 件种类为 2 的商品被分配到最后 2 间商店，数目为: 5, 5 。
分配给所有商店的最大商品数目为 max(5, 5, 5, 5, 5, 5, 5) = 5 。
示例 3: 输入: n = 1, quantities = [100000] 输出: 100000
解释: 唯一一种最优方案为:
- 所有 100000 件商品 0 都分配到唯一的商店中。
分配给所有商店的最大商品数目为 max(100000) = 100000 。
提示: m == quantities.length
1 <= m <= n <= 105
1 <= quantities[i] <= 105
```

- 解题思路

```
func minimizedMaximum(n int, quantities []int) int {
        left,right := 1, 100000
        for left < right{
                mid := left+(right-left)/2
                if judge(quantities, mid) <=n {
                        right = mid
                }else {
                        left = mid+1
                }
        }
        return left
}

func judge( quantities []int, per int)int  {
        res := 0
        for i := 0; i < len(quantities); i++{
                res = res + quantities[i]/per
                if quantities[i]%per !=0{
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 2
func minimizedMaximum(n int, quantities []int) int {
        return sort.Search(100000, func(i int) bool {
                return judge(quantities, i) <= n
        })
}

func judge(quantities []int, per int) int {
        if per == 0 { // 注意除0错误
                return math.MaxInt32
        }
        res := 0
        for i := 0; i < len(quantities); i++ {
                // res = res + (quantities[i]+per-1)/per
                res = res + quantities[i]/per
                if quantities[i]%per != 0 {
                        res++
                }
        }
        return res
}
```

62.27 2069. 模拟行走机器人 II(2)

- 题目

```
给你一个在 XY 平面上的width x height的网格图，左下角的格子为(0,␣
↪0)，右上角的格子为(width - 1, height - 1)。
网格图中相邻格子为四个基本方向之一（"North", "East", "South"和"West"）。
一个机器人 初始在格子(0, 0)，方向为"East"。
机器人可以根据指令移动指定的 步数。每一步，它可以执行以下操作。
沿着当前方向尝试 往前一步。
如果机器人下一步将到达的格子 超出了边界，机器人会 逆时针转 90 度，然后再尝试往前一步。
如果机器人完成了指令要求的移动步数，它将停止移动并等待下一个指令。
请你实现Robot类：
Robot(int width, int height)初始化一个width x height的网格图，机器人初始在(0,␣
↪0)，方向朝"East"。
```

(续下页)

(接上页)

```
void move(int num)给机器人下达前进num步的指令。
int[] getPos()返回机器人当前所处的格子位置，用一个长度为 2 的数组[x, y]表示。
String getDir()返回当前机器人的朝向，为"North", "East", "South"或者"West"。
示例 1: 输入: ["Robot", "move", "move", "getPos", "getDir", "move", "move", "move",
↪"getPos", "getDir"]
[[6, 3], [2], [2], [], [], [2], [1], [4], [], []]
输出: [null, null, null, [4, 0], "East", null, null, null, [1, 2], "West"]
解释: Robot robot = new Robot(6, 3); // 初始化网格图，机器人在 (0, 0) ，朝东。
robot.move(2);  // 机器人朝东移动 2 步，到达 (2, 0) ，并朝东。
robot.move(2);  // 机器人朝东移动 2 步，到达 (4, 0) ，并朝东。
robot.getPos(); // 返回 [4, 0]
robot.getDir(); // 返回 "East"
robot.move(2);  // 朝东移动 1 步到达 (5, 0) ，并朝东。
                // 下一步继续往东移动将出界，所以逆时针转变方向朝北。
                // 然后，往北移动 1 步到达 (5, 1) ，并朝北。
robot.move(1);  // 朝北移动 1 步到达 (5, 2) ，并朝 北 (不是朝西)。
robot.move(4);  // 下一步继续往北移动将出界，所以逆时针转变方向朝西。
                // 然后，移动 4 步到 (1, 2) ，并朝西。
robot.getPos(); // 返回 [1, 2]
robot.getDir(); // 返回 "West"
提示: 2 <= width, height <= 100
1 <= num <= 105
move, getPos和getDir总共调用次数不超过104次。
```

- 解题思路

```
var m = map[int]string{0: "East", 1: "North", 2: "West", 3: "South"}
var dx = []int{1, 0, -1, 0}
var dy = []int{0, 1, 0, -1}

type Robot struct {
        w, h, x, y, dir, total int
}

func Constructor(width int, height int) Robot {
        return Robot{w: width, h: height, total: 2*width + 2*height - 4}
}

func (this *Robot) Step(num int) {
        num = num % this.total
        if num == 0 && this.x == 0 && this.y == 0 && this.dir == 0 {
                this.dir = 3 // 注意特判
        }
        for ; num > 0; num-- {
```

(续下页)

(接上页)

```
                newX, newY := this.x+dx[this.dir], this.y+dy[this.dir]
                if 0 <= newX && newX < this.w && 0 <= newY && newY < this.h {
                        this.x = newX
                        this.y = newY
                } else {
                        this.dir = (this.dir + 1) % 4
                        this.x = this.x + dx[this.dir]
                        this.y = this.y + dy[this.dir]
                }
        }
}

func (this *Robot) GetPos() []int {
        return []int{this.x, this.y}
}

func (this *Robot) GetDir() string {
        return m[this.dir%4]
}

# 2
var m = map[int]string{0: "East", 1: "North", 2: "West", 3: "South"}

type Robot struct {
        arr    [][2]int
        dir    []int
        isMove bool
        index  int
}

func Constructor(width int, height int) Robot {
        arr := make([][2]int, 0)
        dir := make([]int, 0)
        for i := 0; i < width; i++ {
                arr = append(arr, [2]int{i, 0})
                dir = append(dir, 0)
        }
        for i := 1; i < height; i++ {
                arr = append(arr, [2]int{width - 1, i})
                dir = append(dir, 1)
        }
        for i := width - 2; i >= 0; i-- {
                arr = append(arr, [2]int{i, height - 1})
```

(续下页)

(接上页)

```
                dir = append(dir, 2)
        }
        for i := height - 2; i > 0; i-- {
                arr = append(arr, [2]int{0, i})
                dir = append(dir, 3)
        }
        dir[0] = 3 // 第0个朝南
        return Robot{arr: arr, dir: dir, isMove: false}
}

func (this *Robot) Step(num int) {
        this.isMove = true
        this.index = (this.index + num) % len(this.arr)
}

func (this *Robot) GetPos() []int {
        return []int{this.arr[this.index][0], this.arr[this.index][1]}
}

func (this *Robot) GetDir() string {
        if this.isMove == false {
                return "East"
        }
        return m[this.dir[this.index]]
}
```

62.28 2070. 每一个查询的最大美丽值 (2)

- 题目

```
给你一个二维整数数组items，其中items[i] = [pricei, beautyi]分别表示每一个物品的␣
↪价格和 美丽值。
同时给你一个下标从 0开始的整数数组queries。对于每个查询queries[j]，
你想求出价格小于等于queries[j]的物品中，最大的美丽值是多少。如果不存在符合条件的物品，那么查询的结果
请你返回一个长度与 queries相同的数组answer，其中answer[j]是第j个查询的答案。
示例 1：输入：items = [[1,2],[3,2],[2,4],[5,6],[3,5]], queries = [1,2,3,4,5,6]␣
↪输出：[2,4,5,5,6,6]
解释：
- queries[0]=1 ，[1,2] 是唯一价格 <= 1 的物品。所以这个查询的答案为 2 。
- queries[1]=2 ，符合条件的物品有 [1,2] 和 [2,4] 。
  它们中的最大美丽值为 4 。
- queries[2]=3 和 queries[3]=4 ，符合条件的物品都为 [1,2] ，[3,2] ，[2,4] 和 [3,5] 。
```

(续下页)

(接上页)

```
  它们中的最大美丽值为 5 。
- queries[4]=5 和 queries[5]=6 ，所有物品都符合条件。
  所以，答案为所有物品中的最大美丽值，为 6 。
示例 2：输入：items = [[1,2],[1,2],[1,3],[1,4]], queries = [1] 输出：[4]
解释：每个物品的价格均为 1 ，所以我们选择最大美丽值 4 。
注意，多个物品可能有相同的价格和美丽值。
示例 3：输入：items = [[10,1000]], queries = [5] 输出：[0]
解释：没有物品的价格小于等于 5 ，所以没有物品可以选择。
因此，查询的结果为 0 。
提示：1 <= items.length, queries.length <= 105
items[i].length == 2
1 <= pricei, beautyi, queries[j] <= 109
```

- 解题思路

```
func maximumBeauty(items [][]int, queries []int) []int {
        n := len(queries)
        m := len(items)
        res := make([]int, n)
        sort.Slice(items, func(i, j int) bool {
                return items[i][0] < items[j][0]
        })
        for i := 1; i < m; i++ {
                items[i][1] = max(items[i][1], items[i-1][1]) // 更新最大美丽值
        }
        for i := 0; i < n; i++ {
                if queries[i] >= items[m-1][0] {
                        res[i] = items[m-1][1]
                        continue
                }
                if queries[i] < items[0][0] {
                        continue
                }
                index := binSearch(items, queries[i]) // 二分查找
                res[i] = items[index][1]
        }
        return res
}

func binSearch(arr [][]int, target int) int {
        left := 0
        right := len(arr)
        for left < right {
                mid := left + (right-left)/2
```

(续下页)

(接上页)

```
                if arr[mid][0] == target {
                        left = mid + 1
                } else if arr[mid][0] < target {
                        left = mid + 1
                } else if arr[mid][0] > target {
                        right = mid
                }
        }
        return left - 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maximumBeauty(items [][]int, queries []int) []int {
        n := len(queries)
        m := len(items)
        res := make([]int, n)
        sort.Slice(items, func(i, j int) bool {
                return items[i][0] < items[j][0]
        })
        arr := make([][2]int, n)
        for i := 0; i < n; i++ {
                arr[i] = [2]int{i, queries[i]}
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] < arr[j][1]
        })
        j := 0
        maxValue := 0
        for i := 0; i < n; i++ {
                index, target := arr[i][0], arr[i][1]
                for ; j < m && items[j][0] <= target; j++ {
                        maxValue = max(maxValue, items[j][1])
                }
                res[index] = maxValue
        }
        return res
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

62.29 2074. 反转偶数长度组的节点 (3)

- 题目

```
给你一个链表的头节点 head 。
链表中的节点 按顺序 划分成若干 非空 组，这些非空组的长度构成一个自然数序列 (1, 2, 3,␣
↪4, ...) 。
一个组的 长度 就是组中分配到的节点数目。换句话说：
节点 1 分配给第一组
节点 2 和 3 分配给第二组
节点 4、5 和 6 分配给第三组，以此类推
注意，最后一组的长度可能小于或者等于 1 + 倒数第二组的长度 。
反转 每个 偶数 长度组中的节点，并返回修改后链表的头节点 head 。
示例 1：输入：head = [5,2,6,3,9,1,7,3,8,4] 输出：[5,6,2,3,9,1,4,8,3,7]
解释：- 第一组长度为 1 ，奇数，没有发生反转。
- 第二组长度为 2 ，偶数，节点反转。
- 第三组长度为 3 ，奇数，没有发生反转。
- 最后一组长度为 4 ，偶数，节点反转。
示例 2：输入：head = [1,1,0,6] 输出：[1,0,1,6]
解释：- 第一组长度为 1 ，没有发生反转。
- 第二组长度为 2 ，节点反转。
- 最后一组长度为 1 ，没有发生反转。
示例 3：输入：head = [2,1] 输出：[2,1]
解释：- 第一组长度为 1 ，没有发生反转。
- 最后一组长度为 1 ，没有发生反转。
示例 4：输入：head = [8] 输出：[8]
解释：只有一个长度为 1 的组，没有发生反转。
提示：链表中节点数目范围是 [1, 105]
0 <= Node.val <= 105
```

- 解题思路

```
func reverseEvenLengthGroups(head *ListNode) *ListNode {
        arr := make([]int, 0)
        for head != nil {
                arr = append(arr, head.Val)
                head = head.Next
        }
        start := 1
        count := 2
        for start < len(arr) {
                end := start + count - 1 // 计算结尾下标
                if end >= len(arr) {
                        end = len(arr) - 1
                }
                if (start-end+1)%2 == 0 { // 偶数个才反转
                        reverse(arr, start, end) // 反转
                }
                start = start + count
                count = count + 1
        }
        temp := &ListNode{}
        node := temp
        for i := 0; i < len(arr); i++ {
                node.Next = &ListNode{Val: arr[i]}
                node = node.Next
        }
        return temp.Next
}

func reverse(arr []int, start, end int) {
        for start < end {
                arr[start], arr[end] = arr[end], arr[start]
                start++
                end--
        }
}

# 2
func reverseEvenLengthGroups(head *ListNode) *ListNode {
        count := 1
        arr := make([]*ListNode, 0)
        for cur := head; cur != nil; cur = cur.Next {
                arr = append(arr, cur)
                if len(arr) == count || cur.Next == nil {
                        if len(arr)%2 == 0 {
```

(续下页)

(接上页)

```
                                for i := 0; i < len(arr)/2; i++ { // 交换值
                                        arr[i].Val, arr[len(arr)-1-i].Val =␣
↪arr[len(arr)-1-i].Val, arr[i].Val
                                }
                        }
                        arr = make([]*ListNode, 0)
                        count++
                }
        }
        return head
}

# 3
func reverseEvenLengthGroups(head *ListNode) *ListNode {
        count := 1
        cur := head
        prev := &ListNode{}
        for cur != nil {
                c := 0
                temp := cur
                for c < count && temp != nil {
                        c++
                        temp = temp.Next
                }
                if c%2 == 1 {
                        for i := 0; i < c; i++ {
                                prev, cur = cur, cur.Next // 指针后移
                        }
                } else { // 反转链表
                        for i := 0; i < c-1; i++ {
                                prev.Next, cur.Next.Next, cur.Next = cur.Next, prev.
↪Next, cur.Next.Next
                        }
                        prev, cur = cur, cur.Next
                }
                count++
        }
        return head
}
```

62.30 2075. 解码斜向换位密码 (1)

- 题目

```
字符串 originalText 使用 斜向换位密码 ，经由 行数固定 为 rows␣
↪的矩阵辅助，加密得到一个字符串 encodedText 。
originalText 先按从左上到右下的方式放置到矩阵中。
先填充蓝色单元格，接着是红色单元格，然后是黄色单元格，以此类推，直到到达 originalText␣
↪末尾。
箭头指示顺序即为单元格填充顺序。所有空单元格用 ' ' 进行填充。矩阵的列数需满足：用␣
↪originalText 填充之后，最右侧列 不为空 。
接着按行将字符附加到矩阵中，构造encodedText 。
先把蓝色单元格中的字符附加到 encodedText␣
↪中，接着是红色单元格，最后是黄色单元格。箭头指示单元格访问顺序。
例如，如果 originalText = "cipher" 且 rows = 3 ，那么我们可以按下述方法将其编码：
蓝色箭头标识 originalText 是如何放入矩阵中的，红色箭头标识形成 encodedText 的顺序。
在上述例子中，encodedText = "ch ie pr" 。
给你编码后的字符串 encodedText 和矩阵的行数 rows ，返回源字符串 originalText 。
注意：originalText 不 含任何尾随空格 ' ' 。生成的测试用例满足 仅存在一个 可能的␣
↪originalText 。
示例 1：输入：encodedText = "ch   ie   pr", rows = 3 输出："cipher"
解释：此示例与问题描述中的例子相同。
示例 2：输入：encodedText = "iveo    eed   l te   olc", rows = 4 输出："i love␣
↪leetcode"
解释：上图标识用于编码 originalText 的矩阵。
蓝色箭头展示如何从 encodedText 找到 originalText 。
示例 3：输入：encodedText = "coding", rows = 1 输出："coding"
解释：由于只有 1 行，所以 originalText 和 encodedText 是相同的。
示例 4：输入：encodedText = " b  ac", rows = 2 输出：" abc"
解释：originalText 不能含尾随空格，但它可能会有一个或者多个前置空格。
提示：0 <= encodedText.length <= 106
encodedText 仅由小写英文字母和 ' ' 组成
encodedText 是对某个 不含 尾随空格的 originalText 的一个有效编码
1 <= rows <= 1000
生成的测试用例满足 仅存在一个 可能的 originalText
```

- 解题思路

```
func decodeCiphertext(encodedText string, rows int) string {
        a, b := rows, len(encodedText)/rows
        res := make([]byte, 0)
        for i := 0; i < b; i++ {
                x, y := 0, i
                for x < a && y < b {
```

(续下页)

(接上页)

```
                        res = append(res, encodedText[x*b+y])
                        x++
                        y++
                }
        }
        return strings.TrimRight(string(res), " ")
}
```

62.31 2079. 给植物浇水 (1)

- 题目

```
你打算用一个水罐给花园里的 n 株植物浇水。植物排成一行，从左到右进行标记，编号从 0 到␣
↪n - 1 。
其中，第 i 株植物的位置是 x = i 。x = -1处有一条河，你可以在那里重新灌满你的水罐。
每一株植物都需要浇特定量的水。你将会按下面描述的方式完成浇水：
按从左到右的顺序给植物浇水。
在给当前植物浇完水之后，如果你没有足够的水 完全␣
↪浇灌下一株植物，那么你就需要返回河边重新装满水罐。
你 不能 提前重新灌满水罐。
最初，你在河边（也就是，x = -1），在 x 轴上每移动 一个单位都需要 一步 。
给你一个下标从 0 开始的整数数组 plants ，数组由 n 个整数组成。其中，plants[i] 为第 i␣
↪株植物需要的水量。
另有一个整数 capacity 表示水罐的容量，返回浇灌所有植物需要的 步数 。
示例 1：输入：plants = [2,2,3,3], capacity = 5 输出：14
解释：从河边开始，此时水罐是装满的：
- 走到植物 0（1 步），浇水。水罐中还有 3 单位的水。
- 走到植物 1（1 步），浇水。水罐中还有 1 单位的水。
- 由于不能完全浇灌植物 2 ，回到河边取水 （2 步）。
- 走到植物 2（3 步），浇水。水罐中还有 2 单位的水。
- 由于不能完全浇灌植物 3 ，回到河边取水 （3 步）。
- 走到植物 3（4 步），浇水。
需要的步数是 = 1 + 1 + 2 + 3 + 3 + 4 = 14 。
示例 2：输入：plants = [1,1,1,4,2,3], capacity = 4 输出：30
解释：从河边开始，此时水罐是装满的：
- 走到植物 0，1，2（3 步），浇水。回到河边取水 （3 步）。
- 走到植物 3（4 步），浇水。回到河边取水 （4 步）。
- 走到植物 4（5 步），浇水。回到河边取水 （5 步）。
- 走到植物 5（6 步），浇水。
需要的步数是 = 3 + 3 + 4 + 4 + 5 + 5 + 6 = 30 。
示例 3：输入：plants = [7,7,7,7,7,7,7], capacity = 8 输出：49
解释：每次浇水都需要重新灌满水罐。
```

(续下页)

(接上页)

```
需要的步数是 = 1 + 1 + 2 + 2 + 3 + 3 + 4 + 4 + 5 + 5 + 6 + 6 + 7 = 49 。
提示：n == plants.length
1 <= n <= 1000
1 <= plants[i] <= 106
max(plants[i]) <= capacity <= 109
```

- 解题思路

```
func wateringPlants(plants []int, capacity int) int {
        res := 0
        n := len(plants)
        value := capacity
        for i := 0; i < n; i++ {
                if plants[i] <= value {
                        res++ // 走1步
                        value = value - plants[i]
                } else {
                        res = res + i + (i + 1) // 回去i，返回i+1
                        value = capacity - plants[i]
                }
        }
        return res
}
```

62.32 2080. 区间内查询数字的频率 (3)

- 题目

```
请你设计一个数据结构，它能求出给定子数组内一个给定值的 频率。
子数组中一个值的 频率指的是这个子数组中这个值的出现次数。
请你实现RangeFreqQuery类：
RangeFreqQuery(int[] arr)用下标从 0开始的整数数组arr构造一个类的实例。
int query(int left, int right, int value)返回子数组arr[left...right]中value的频率。
一个 子数组 指的是数组中一段连续的元素。arr[left...right]指的是 nums中包含下标 left和
↪right在内的中间一段连续元素。
示例 1：输入：["RangeFreqQuery", "query", "query"]
[[[12, 33, 4, 56, 22, 2, 34, 33, 22, 12, 34, 56]], [1, 2, 4], [0, 11, 33]]
输出：[null, 1, 2]
解释：RangeFreqQuery rangeFreqQuery = new RangeFreqQuery([12, 33, 4, 56, 22, 2, 34,
↪33, 22, 12, 34, 56]);
rangeFreqQuery.query(1, 2, 4); // 返回 1 。4 在子数组 [33, 4] 中出现 1 次。
rangeFreqQuery.query(0, 11, 33); // 返回 2 。33 在整个子数组中出现 2 次。
```

(续下页)

(接上页)

```
提示：1 <= arr.length <= 105
1 <= arr[i], value <= 104
0 <= left <= right < arr.length
调用query不超过105次。
```

- 解题思路

```
type RangeFreqQuery struct {
        m map[int][]int
}

func Constructor(arr []int) RangeFreqQuery {
        m := make(map[int][]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]] = append(m[arr[i]], i)
        }
        return RangeFreqQuery{m: m}
}

func (this *RangeFreqQuery) Query(left int, right int, value int) int {
        arr := this.m[value]
        l := lowerBound(arr, left)
        r := upperBound(arr, right)
        return r - l
}

// 返回第一个大于target的位置
func lowerBound(arr []int, target int) int {
        left, right := 0, len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        right = mid // 收缩左边界
                } else if arr[mid] < target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

// 返回第一个大于等于target的位置
func upperBound(arr []int, target int) int {
```

(续下页)

(接上页)

```
        left, right := 0, len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        left = mid + 1 // 收缩左边界
                } else if arr[mid] < target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

# 2
type RangeFreqQuery struct {
        m map[int]sort.IntSlice
}

func Constructor(arr []int) RangeFreqQuery {
        m := make(map[int]sort.IntSlice)
        for i := 0; i < len(arr); i++ {
                m[arr[i]] = append(m[arr[i]], i)
        }
        return RangeFreqQuery{m: m}
}

func (this *RangeFreqQuery) Query(left int, right int, value int) int {
        arr := this.m[value]
        arr = arr[arr.Search(left):]
        return arr.Search(right + 1)
}

# 3
type RangeFreqQuery struct {
        m map[int][]int
}

func Constructor(arr []int) RangeFreqQuery {
        m := make(map[int][]int)
        for i := 0; i < len(arr); i++ {
                m[arr[i]] = append(m[arr[i]], i)
        }
```

(续下页)

(接上页)

```
        return RangeFreqQuery{m: m}
}

func (this *RangeFreqQuery) Query(left int, right int, value int) int {
        arr := this.m[value]
        l := sort.SearchInts(arr, left)
        r := sort.SearchInts(arr, right+1)
        return r - l
}
```

62.33 2086. 从房屋收集雨水需要的最少水桶数 (2)

- 题目

```
给你一个下标从 0开始的字符串street。street中每个字符要么是表示房屋的'H'␣
↪, 要么是表示空位的'.'。
你可以在 空位放置水桶, 从相邻的房屋收集雨水。位置在 i - 1或者 i +␣
↪1的水桶可以收集位置为 i处房屋的雨水。
一个水桶如果相邻两个位置都有房屋, 那么它可以收集 两个 房屋的雨水。
在确保 每个房屋旁边都 至少有一个水桶的前提下, 请你返回需要的␣
↪最少水桶数。如果无解请返回 -1。
示例 1: 输入: street = "H..H" 输出: 2
解释: 我们可以在下标为 1 和 2 处放水桶。
"H..H" -> "HBBH" ('B' 表示放置水桶) 。
下标为 0 处的房屋右边有水桶, 下标为 3 处的房屋左边有水桶。
所以每个房屋旁边都至少有一个水桶收集雨水。
示例 2: 输入: street = ".H.H." 输出: 1
解释: 我们可以在下标为 2 处放置一个水桶。
".H.H." -> ".HBH." ('B' 表示放置水桶) 。
下标为 1 处的房屋右边有水桶, 下标为 3 处的房屋左边有水桶。
所以每个房屋旁边都至少有一个水桶收集雨水。
示例 3: 输入: street = ".HHH." 输出: -1
解释: 没有空位可以放置水桶收集下标为 2 处的雨水。
所以没有办法收集所有房屋的雨水。
示例 4: 输入: street = "H" 输出: -1
解释: 没有空位放置水桶。
所以没有办法收集所有房屋的雨水。
示例 5: 输入: street = "." 输出: 0
解释: 没有房屋需要收集雨水。
所以需要 0 个水桶。
提示: 1 <= street.length <= 105
street[i]要么是'H', 要么是'.' 。
```

- 解题思路

```
func minimumBuckets(street string) int {
        res := 0
        n := len(street)
        for i := 0; i < n; i++ {
                if street[i] == 'H' {
                        if i+1 < n && street[i+1] == '.' { // (H.
↪A)BC的情况，往后数3(+2再+1)位到B，+1
                                res++
                                i = i + 2
                        } else if i >= 1 && street[i-1] == '.' { // (.HH)ABC的情况，+1
                                res++
                        } else {
                                return -1
                        }
                }
        }
        return res
}

# 2
func minimumBuckets(street string) int {
        if street == "H" || strings.Contains(street, "HHH") ||
                strings.HasSuffix(street, "HH") || strings.HasPrefix(street, "HH") {
                return -1
        }
        return strings.Count(street, "H") - strings.Count(street, "H.H")
}
```

62.34 2087. 网格图中机器人回家的最小代价 (2)

- 题目

```
给你一个m x n的网格图，其中(0, 0)是最左上角的格子，(m - 1, n - 1)是最右下角的格子。
给你一个整数数组startPos，startPos = [startrow, startcol]表示 初始有一个␣
↪机器人在格子(startrow, startcol)处。
同时给你一个整数数组homePos，homePos = [homerow, homecol]表示机器人的␣
↪家在格子(homerow, homecol)处。
机器人需要回家。每一步它可以往四个方向移动：上，下，左，右，同时机器人不能移出边界。
每一步移动都有一定代价。再给你两个下标从0开始的额整数数组：长度为m的数组rowCosts␣
↪和长度为 n的数组colCosts。
如果机器人往 上或者往 下移动到第 r行的格子，那么代价为rowCosts[r]。
```

(续下页)

(接上页)

```
如果机器人往 左或者往 右移动到第 c列 的格子，那么代价为colCosts[c]。
请你返回机器人回家需要的 最小总代价。
示例 1：输入：startPos = [1, 0], homePos = [2, 3], rowCosts = [5, 4, 3], colCosts =␣
↪[8, 2, 6, 7] 输出：18
解释：一个最优路径为：
从 (1, 0) 开始
-> 往下走到 (2, 0) 。代价为 rowCosts[2] = 3 。
-> 往右走到 (2, 1) 。代价为 colCosts[1] = 2 。
-> 往右走到 (2, 2) 。代价为 colCosts[2] = 6 。
-> 往右走到 (2, 3) 。代价为 colCosts[3] = 7 。
总代价为 3 + 2 + 6 + 7 = 18
示例 2：输入：startPos = [0, 0], homePos = [0, 0], rowCosts = [5], colCosts = [26]␣
↪输出：0
解释：机器人已经在家了，所以不需要移动。总代价为 0 。
提示：m == rowCosts.length
n == colCosts.length
1 <= m, n <= 105
0 <= rowCosts[r], colCosts[c] <= 104
startPos.length == 2
homePos.length == 2
0 <= startrow, homerow < m
0 <= startcol, homecol < n
```

- 解题思路

```
func minCost(startPos []int, homePos []int, rowCosts []int, colCosts []int) int {
        res := 0
        a, b, c, d := startPos[0], startPos[1], homePos[0], homePos[1]
        res = res - rowCosts[a] - colCosts[b]
        if a > c {
                a, c = c, a
        }
        if b > d {
                b, d = d, b
        }
        for i := a; i <= c; i++ {
                res = res + rowCosts[i]
        }
        for i := b; i <= d; i++ {
                res = res + colCosts[i]
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func minCost(startPos []int, homePos []int, rowCosts []int, colCosts []int) int {
        res := 0
        a, b, c, d := startPos[0], startPos[1], homePos[0], homePos[1]
        if a > c {
                for i := a - 1; i >= c; i-- {
                        res = res + rowCosts[i]
                }
        } else if a < c {
                for i := a + 1; i <= c; i++ {
                        res = res + rowCosts[i]
                }
        }
        if b > d {
                for i := b - 1; i >= d; i-- {
                        res = res + colCosts[i]
                }
        } else if b < d {
                for i := b + 1; i <= d; i++ {
                        res = res + colCosts[i]
                }
        }
        return res
}
```

62.35 2090. 半径为 k 的子数组平均值 (2)

- 题目

```
给你一个下标从 0 开始的数组 nums ，数组中有 n 个整数，另给你一个整数 k 。
半径为 k 的子数组平均值 是指：nums 中一个以下标 i 为 中心 且 半径 为 k␣
↪的子数组中所有元素的平均值，
即下标在i - k 和 i + k 范围（含 i - k 和 i + k）内所有元素的平均值。
如果在下标 i 前或后不足 k 个元素，那么 半径为 k 的子数组平均值 是 -1 。
构建并返回一个长度为 n 的数组 avgs ，其中 avgs[i] 是以下标 i 为中心的子数组的 半径为␣
↪k 的子数组平均值 。
x 个元素的 平均值 是 x 个元素相加之和除以 x ，此时使用截断式 整数除法␣
↪，即需要去掉结果的小数部分。
例如，四个元素 2、3、1 和 5 的平均值是 (2 + 3 + 1 + 5) / 4 = 11 / 4 = 3.
↪75，截断后得到 3 。
示例 1：输入：nums = [7,4,3,9,1,8,5,2,6], k = 3 输出：[-1,-1,-1,5,4,4,-1,-1,-1]
解释：- avg[0]、avg[1] 和 avg[2] 是 -1 ，因为在这几个下标前的元素数量都不足 k 个。
```

(续下页)

(接上页)

```
- 中心为下标 3 且半径为 3 的子数组的元素总和是：7 + 4 + 3 + 9 + 1 + 8 + 5 = 37 。
  使用截断式 整数除法，avg[3] = 37 / 7 = 5 。
- 中心为下标 4 的子数组，avg[4] = (4 + 3 + 9 + 1 + 8 + 5 + 2) / 7 = 4 。
- 中心为下标 5 的子数组，avg[5] = (3 + 9 + 1 + 8 + 5 + 2 + 6) / 7 = 4 。
- avg[6]、avg[7] 和 avg[8] 是 -1 ，因为在这几个下标后的元素数量都不足 k 个。
示例 2：输入：nums = [100000], k = 0 输出：[100000]
解释：- 中心为下标 0 且半径 0 的子数组的元素总和是：100000 。
  avg[0] = 100000 / 1 = 100000 。
示例 3：输入：nums = [8], k = 100000 输出：[-1]
解释：- avg[0] 是 -1 ，因为在下标 0 前后的元素数量均不足 k 。
提示：n == nums.length
1 <= n <= 105
0 <= nums[i], k <= 105
```

- 解题思路

```
func getAverages(nums []int, k int) []int {
        n := len(nums)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = -1
        }
        if 2*k+1 <= n {
                sum := 0
                for i := 0; i < 2*k+1; i++ {
                        sum = sum + nums[i]
                }
                for i := k; i < n-k; i++ {
                        if i != k {
                                sum = sum + nums[i+k] - nums[i-k-1]
                        }
                        res[i] = sum / (2*k + 1)
                }
        }
        return res
}

# 2
func getAverages(nums []int, k int) []int {
        n := len(nums)
        res := make([]int, n)
        left := 0
        sum := 0
        for right := 0; right < n; right++ {
```

(续下页)

(接上页)

```
                res[right] = -1
                sum = sum + nums[right]
                if right >= 2*k {
                        res[right-k] = sum / (2*k + 1)
                        sum = sum - nums[left]
                        left++
                }
        }
        return res
}
```

62.36 2091. 从数组中移除最大值和最小值 (1)

- 题目

```
给你一个下标从 0 开始的数组 nums ，数组由若干 互不相同 的整数组成。
nums 中有一个值最小的元素和一个值最大的元素。分别称为 最小值 和 最大值␣
↪。你的目标是从数组中移除这两个元素。
一次 删除 操作定义为从数组的 前面 移除一个元素或从数组的 后面 移除一个元素。
返回将数组中最小值和最大值 都 移除需要的最小删除次数。
示例 1：输入：nums = [2,10,7,5,4,1,8,6] 输出：5
解释：数组中的最小元素是 nums[5] ，值为 1 。
数组中的最大元素是 nums[1] ，值为 10 。
将最大值和最小值都移除需要从数组前面移除 2 个元素，从数组后面移除 3 个元素。
结果是 2 + 3 = 5 ，这是所有可能情况中的最小删除次数。
示例 2：输入：nums = [0,-4,19,1,8,-2,-3,5] 输出：3
解释：数组中的最小元素是 nums[1] ，值为 -4 。
数组中的最大元素是 nums[2] ，值为 19 。
将最大值和最小值都移除需要从数组前面移除 3 个元素。
结果是 3 ，这是所有可能情况中的最小删除次数。
示例 3：输入：nums = [101] 输出：1
解释：数组中只有这一个元素，那么它既是数组中的最小值又是数组中的最大值。
移除它只需要 1 次删除操作。
提示：1 <= nums.length <= 105
-105 <= nums[i] <= 105
nums 中的整数 互不相同
```

- 解题思路

```
func minimumDeletions(nums []int) int {
        minIndex, maxIndex := 0, 0
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                if nums[i] > nums[maxIndex] {
                        maxIndex = i
                }
                if nums[i] < nums[minIndex] {
                        minIndex = i
                }
        }
        if minIndex > maxIndex { // 保证minIndex <= maxIndex
                minIndex, maxIndex = maxIndex, minIndex
        }
        a := maxIndex + 1                       // 第1种情况：全往左移
        b := len(nums) - minIndex               // 第2种情况：全往右移
        c := minIndex + 1 + len(nums) - maxIndex // 第3种情况：往2边移动
        return min(min(a, b), c)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

62.37 2095. 删除链表的中间节点 (2)

- 题目

```
给你一个链表的头节点 head 。删除 链表的 中间节点 ，并返回修改后的链表的头节点 head 。
长度为 n 链表的中间节点是从头数起第 ⌊n / 2⌋ 个节点（下标从 0 开始），其中 ⌊x⌋␣
↪表示小于或等于 x 的最大整数。
对于 n = 1、2、3、4 和 5 的情况，中间节点的下标分别是 0、1、1、2 和 2 。
示例 1：输入：head = [1,3,4,7,1,2,6] 输出：[1,3,4,1,2,6]
解释：上图表示给出的链表。节点的下标分别标注在每个节点的下方。
由于 n = 7 ，值为 7 的节点 3 是中间节点，用红色标注。
返回结果为移除节点后的新链表。
示例 2：输入：head = [1,2,3,4] 输出：[1,2,4]
解释：上图表示给出的链表。
对于 n = 4 ，值为 3 的节点 2 是中间节点，用红色标注。
示例 3：输入：head = [2,1] 输出：[2]
解释：上图表示给出的链表。
对于 n = 2 ，值为 1 的节点 1 是中间节点，用红色标注。
值为 2 的节点 0 是移除节点 1 后剩下的唯一一个节点。
```

(续下页)

(接上页)

```
提示：链表中节点的数目在范围 [1, 105] 内
1 <= Node.val <= 105
```

- 解题思路

```
func deleteMiddle(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return nil
        }
        slow := head
        fast := head
        prev := &ListNode{}
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                prev = slow
                slow = slow.Next
        }
        prev.Next = prev.Next.Next
        return head
}

# 2
func deleteMiddle(head *ListNode) *ListNode {
        prev := &ListNode{Next: head}
        slow := prev
        fast := prev
        for fast.Next != nil && fast.Next.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
        }
        slow.Next = slow.Next.Next
        return prev.Next
}
```

62.38 2096. 从二叉树一个节点到另一个节点每一步的方向 (2)

- 题目

```
给你一棵␣
↪二叉树的根节点root，这棵二叉树总共有n个节点。每个节点的值为1到n中的一个整数，且互不相同。
给你一个整数startValue，表示起点节点␣
↪s的值，和另一个不同的整数destValue，表示终点节点t的值。
```

(续下页)

(接上页)

```
请找到从节点s到节点 t的 最短路径，并以字符串的形式返回每一步的方向。
每一步用 大写字母'L'，'R'和'U'分别表示一种方向：
'L'表示从一个节点前往它的 左孩子节点。
'R'表示从一个节点前往它的 右孩子节点。
'U'表示从一个节点前往它的 父节点。
请你返回从 s到 t最短路径每一步的方向。
示例 1：输入：root = [5,1,2,3,null,6,4], startValue = 3, destValue = 6 输出："UURL"
解释：最短路径为：3 → 1 → 5 → 2 → 6 。
示例 2：输入：root = [2,1], startValue = 2, destValue = 1 输出："L"
解释：最短路径为：2 → 1 。
提示：树中节点数目为n。
2 <= n <= 105
1 <= Node.val <= n
树中所有节点的值 互不相同。
1 <= startValue, destValue <= n
startValue != destValue
```

- 解题思路

```
var m map[*TreeNode]*TreeNode // 父节点
var start, dest *TreeNode

func getDirections(root *TreeNode, startValue int, destValue int) string {
        m = make(map[*TreeNode]*TreeNode)
        start, dest = &TreeNode{}, &TreeNode{}
        dfs(root, startValue, destValue)            // 构建父节点关系
        a, b := path(start, root), path(dest, root) // 生成根节点到目标节点的路径
        i := 0
        for i = 0; i < len(a) && i < len(b); i++ {
                if a[i] != b[i] {
                        break
                }
        }
        return strings.Repeat("U", len(a)-i) + strings.Join(b[i:], "")
}

func path(cur *TreeNode, root *TreeNode) []string {
        res := make([]string, 0)
        for cur != root {
                prev := m[cur]
                if cur == prev.Left {
                        res = append(res, "L")
                } else {
                        res = append(res, "R")
```

(续下页)

(接上页)

```
                }
                cur = prev
        }
        for i := 0; i < len(res)/2; i++ {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
        }
        return res
}

func dfs(root *TreeNode, startValue, destValue int) {
        if root.Val == startValue {
                start = root
        }
        if root.Val == destValue {
                dest = root
        }
        if root.Left != nil {
                m[root.Left] = root
                dfs(root.Left, startValue, destValue)
        }
        if root.Right != nil {
                m[root.Right] = root
                dfs(root.Right, startValue, destValue)
        }
}

# 2
var a, b []string

func getDirections(root *TreeNode, startValue int, destValue int) string {
        a, b = make([]string, 0), make([]string, 0)
        dfs(root, startValue, destValue, make([]string, 0)) // 构建父节点关系

        i := 0
        for i = 0; i < len(a) && i < len(b); i++ {
                if a[i] != b[i] {
                        break
                }
        }
        return strings.Repeat("U", len(a)-i) + strings.Join(b[i:], "")
}

func dfs(root *TreeNode, startValue, destValue int, path []string) {
```

(续下页)

(接上页)

```
        if root.Val == startValue {
                a = make([]string, len(path))
                copy(a, path)
        }
        if root.Val == destValue {
                b = make([]string, len(path))
                copy(b, path)
        }
        if root.Left != nil {
                path = append(path, "L")
                dfs(root.Left, startValue, destValue, path)
                path = path[:len(path)-1]
        }
        if root.Right != nil {
                path = append(path, "R")
                dfs(root.Right, startValue, destValue, path)
                path = path[:len(path)-1]
        }
}
```

62.39 2100. 适合打劫银行的日子 (1)

- 题目

```
你和一群强盗准备打劫银行。给你一个下标从 0开始的整数数组security，其中security[i]是第␣
↪i天执勤警卫的数量。
日子从 0开始编号。同时给你一个整数time。
如果第 i天满足以下所有条件，我们称它为一个适合打劫银行的日子：
第 i天前和后都分别至少有 time天。
第 i天前连续 time天警卫数目都是非递增的。
第 i天后连续 time天警卫数目都是非递减的。
更正式的，第 i 天是一个合适打劫银行的日子当且仅当：
security[i - time] >= security[i - time + 1] >= ... >= security[i] <= ...
<= security[i + time - 1] <= security[i + time].
请你返回一个数组，包含 所有 适合打劫银行的日子（下标从 0开始）。返回的日子可以␣
↪任意顺序排列。
示例 1：输入：security = [5,3,3,3,5,6,2], time = 2 输出：[2,3]
解释：第 2 天，我们有 security[0] >= security[1] >= security[2] <= security[3] <=␣
↪security[4] 。
第 3 天，我们有 security[1] >= security[2] >= security[3] <= security[4] <=␣
↪security[5] 。
没有其他日子符合这个条件，所以日子 2 和 3 是适合打劫银行的日子。
```

(续下页)

(接上页)

```
示例 2：输入：security = [1,1,1,1,1], time = 0 输出：[0,1,2,3,4]
解释：因为 time 等于 0 ，所以每一天都是适合打劫银行的日子，所以返回每一天。
示例 3：输入：security = [1,2,3,4,5,6], time = 2输出：[]
解释：没有任何一天的前 2 天警卫数目是非递增的。
所以没有适合打劫银行的日子，返回空数组。
示例 4：输入：security = [1], time = 5 输出：[]
解释：没有日子前面和后面有 5 天时间。
所以没有适合打劫银行的日子，返回空数组。
提示：1 <= security.length <= 105
0 <= security[i], time <= 105
```

- 解题思路

```
func goodDaysToRobBank(security []int, time int) []int {
        n := len(security)
        if 2*time >= n {
                return nil
        }
        left, right := make([]int, n), make([]int, n)
        for i := 1; i < n; i++ {
                if security[i-1] >= security[i] {
                        left[i] = left[i-1] + 1
                }
        }
        for i := n - 2; i >= 0; i-- {
                if security[i] <= security[i+1] {
                        right[i] = right[i+1] + 1
                }
        }
        res := make([]int, 0)
        for i := time; i < n-time; i++ {
                if left[i] >= time && right[i] >= time {
                        res = append(res, i)
                }
        }
        return res
}
```

CHAPTER 63

2001-2100-Hard

63.1 2009. 使数组连续的最少操作数 (2)

- 题目

```
给你一个整数数组nums。每一次操作中，你可以将nums中任意一个元素替换成 任意整数。
如果nums满足以下条件，那么它是 连续的：
nums中所有元素都是 互不相同的。
nums中 最大元素与最小元素的差等于nums.length - 1。
比方说，nums = [4, 2, 5, 3]是 连续的，但是nums = [1, 2, 3, 5, 6] 不是连续的。
请你返回使 nums连续的 最少操作次数。
示例 1：输入：nums = [4,2,5,3] 输出：0
解释：nums 已经是连续的了。
示例 2：输入：nums = [1,2,3,5,6] 输出：1
解释：一个可能的解是将最后一个元素变为 4 。
结果数组为 [1,2,3,5,4] ，是连续数组。
示例 3：输入：nums = [1,10,100,1000] 输出：3
解释：一个可能的解是：
- 将第二个元素变为 2 。
- 将第三个元素变为 3 。
- 将第四个元素变为 4 。
结果数组为 [1,2,3,4] ，是连续数组。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 109
```

- 解题思路

```
func minOperations(nums []int) int {
        n := len(nums)
        sort.Ints(nums)
        // 去重
        index := 0
        for i := 1; i < n; i++ {
                if nums[index] != nums[i] {
                        index++
                        nums[index] = nums[i]
                }
        }
        nums = nums[:index+1]
        // 二分查找
        res := 0
        for right := 0; right < len(nums); right++ {
                // 计算区间[nums[i]-n+1, nums[i]]的长度
                left := sort.SearchInts(nums[:right], nums[right]-n+1)
                res = max(res, right-left+1) // 取最长的长度
        }
        return n - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minOperations(nums []int) int {
        n := len(nums)
        sort.Ints(nums)
        // 去重
        index := 0
        for i := 1; i < n; i++ {
                if nums[index] != nums[i] {
                        index++
                        nums[index] = nums[i]
                }
        }
        nums = nums[:index+1]
        // 滑动窗口
```

(续下页)

(接上页)

```
        res := 0
        left := 0
        for right := 0; right < len(nums); right++ {
                for nums[right]-nums[left] > n-1 {
                        left++
                }
                res = max(res, right-left+1)
        }
        return n - res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

63.2 2025. 分割数组的最多方案数 (2)

- 题目

```
给你一个下标从 0开始且长度为 n的整数数组nums。分割数组␣
↪nums的方案数定义为符合以下两个条件的 pivot数目：
1 <= pivot < n
nums[0] + nums[1] + ... + nums[pivot - 1] == nums[pivot] + nums[pivot + 1] + ... +␣
↪nums[n - 1]
同时给你一个整数k。你可以将nums中一个元素变为k或不改变数组。
请你返回在 至多改变一个元素的前提下，最多有多少种方法 分割nums使得上述两个条件都满足。
示例 1：输入：nums = [2,-1,2], k = 3 输出：1
解释：一个最优的方案是将 nums[0] 改为 k。数组变为 [3,-1,2] 。
有一种方法分割数组：
- pivot = 2 ，我们有分割 [3,-1 | 2]：3 + -1 == 2 。
示例 2：输入：nums = [0,0,0], k = 1 输出：2
解释：一个最优的方案是不改动数组。
有两种方法分割数组：
- pivot = 1 ，我们有分割 [0 | 0,0]：0 == 0 + 0 。
- pivot = 2 ，我们有分割 [0,0 | 0]：0 + 0 == 0 。
示例 3：输入：nums = [22,4,-25,-20,-15,15,-16,7,19,-10,0,-13,-14], k = -33 输出：4
解释：一个最优的方案是将 nums[2] 改为 k 。数组变为 [22,4,-33,-20,-15,15,-16,7,19,-10,
↪0,-13,-14] 。
有四种方法分割数组。
```

(续下页)

(接上页)

```
提示：n == nums.length
2 <= n <= 105
-105 <= k, nums[i] <= 105
```

- 解题思路

```
func waysToPartition(nums []int, k int) int {
        res := 0
        n := len(nums)
        sum := 0
        prev := make(map[int]int) // 前缀和
        arr := make([]int, n+1)
        for i := 0; i < n; i++ {
                arr[i+1] = arr[i] + nums[i]
                sum = sum + nums[i]
                if i != n-1 {
                        prev[sum]++
                }
        }
        if sum%2 == 0 {
                res = prev[sum/2] // 不改变的结果
        }
        suf := make(map[int]int) // 后缀和
        sufSum := 0
        temp := sum
        for i := n - 1; i >= 0; i-- { // 枚举每一位修改后的结果
                target := sum - nums[i] + k // 替换后的总和
                temp = temp - nums[i]
                prev[temp]-- // 前缀和 减一
                sufSum = sufSum + k
                suf[sufSum]++
                if target%2 == 0 {
                        res = max(res, prev[target/2]+suf[target/2])
                }
                suf[sufSum]--
                sufSum = sufSum - k + nums[i]
                suf[sufSum]++
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}

# 2
func waysToPartition(nums []int, k int) int {
        res := 0
        n := len(nums)
        prev := make(map[int]int) // 前缀和
        arr := make([]int, n)
        arr[0] = nums[0]
        for i := 1; i < n; i++ {
                arr[i] = arr[i-1] + nums[i]
                prev[arr[i-1]]++
        }
        sum := arr[n-1]
        if sum%2 == 0 {
                res = prev[sum/2] // 不改变的结果
        }
        m := make(map[int]int)
        for i := 0; i < len(arr); i++ {
                diff := k - nums[i]
                if (sum+diff)%2 == 0 {
                        res = max(res, prev[(sum-diff)/2]+m[(sum+diff)/2])
                }
                m[arr[i]]++    // 左侧+1
                prev[arr[i]]-- // 右侧-1
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

63.3 2050. 并行课程 III(2)

- 题目

```
给你一个整数n，表示有n节课，课程编号从1到n。同时给你一个二维整数数组relations，
其中relations[j] = [prevCoursej, nextCoursej]，
表示课程prevCoursej必须在课程nextCoursej之前完成（先修课的关系）。
同时给你一个下标从 0开始的整数数组time，其中time[i]表示完成第(i+1)门课程需要花费的␣
↪月份数。
请你根据以下规则算出完成所有课程所需要的 最少月份数：
如果一门课的所有先修课都已经完成，你可以在 任意时间开始这门课程。
你可以同时上任意门课程。
请你返回完成所有课程所需要的 最少月份数。
注意：测试数据保证一定可以完成所有课程（也就是先修课的关系构成一个有向无环图）。
示例1:输入：n = 3, relations = [[1,3],[2,3]], time = [3,2,5] 输出：8
解释：上图展示了输入数据所表示的先修关系图，以及完成每门课程需要花费的时间。
你可以在月份 0 同时开始课程 1 和 2 。
课程 1 花费 3 个月，课程 2 花费 2 个月。
所以，最早开始课程 3 的时间是月份 3 ，完成所有课程所需时间为 3 + 5 = 8 个月。
示例 2：输入：n = 5, relations = [[1,5],[2,5],[3,5],[3,4],[4,5]], time = [1,2,3,4,5]␣
↪输出：12
解释：上图展示了输入数据所表示的先修关系图，以及完成每门课程需要花费的时间。
你可以在月份 0 同时开始课程 1 ，2 和 3 。
在月份 1，2 和 3 分别完成这三门课程。
课程 4 需在课程 3 之后开始，也就是 3 个月后。课程 4 在 3 + 4 = 7 月完成。
课程 5 需在课程 1，2，3 和 4 之后开始，也就是在 max(1,2,3,7) = 7 月开始。
所以完成所有课程所需的最少时间为 7 + 5 = 12 个月。
提示：1 <= n <= 5 * 104
0 <= relations.length <= min(n * (n - 1) / 2, 5 * 104)
relations[j].length == 2
1 <= prevCoursej, nextCoursej <= n
prevCoursej != nextCoursej
所有的先修课程对[prevCoursej, nextCoursej]都是 互不相同的。
time.length == n
1 <= time[i] <= 104
先修课程图是一个有向无环图。
```

- 解题思路

```
func minimumTime(n int, relations [][]int, time []int) int {
        res := 0
        degree := make([]int, n+1)
        dis := make([]int, n+1)    // 计算入度
        arr := make([][]int, n+1) // 邻接表
```

(续下页)

(接上页)

```
        for i := 0; i < len(relations); i++ {
                a, b := relations[i][0], relations[i][1] // a => b
                arr[a] = append(arr[a], b)
                degree[b]++ // 入度+1
        }
        queue := make([]int, 0)
        for i := 1; i <= n; i++ {
                if degree[i] == 0 { // 入度为0: 起点
                        queue = append(queue, i)
                        dis[i] = time[i-1]
                        res = max(res, dis[i])
                }
        }
        for len(queue) > 0 {
                cur := queue[0]
                queue = queue[1:]
                for i := 0; i < len(arr[cur]); i++ {
                        next := arr[cur][i]
                        dis[next] = max(dis[next], dis[cur]+time[next-1]) //␣
↪更新为较大的结果
                        degree[next]--                                    //␣
↪next节点入度-1
                        if degree[next] == 0 {                            // 入度=0
                                queue = append(queue, next)
                        }
                        res = max(res, dis[next])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var m map[int]int
var arr [][]int
var temp []int
```

(续下页)

(接上页)

```
func minimumTime(n int, relations [][]int, time []int) int {
        arr = make([][]int, n+1)
        degree := make([]int, n+1) // 出度
        temp = make([]int, n+1)
        m = make(map[int]int)
        copy(temp, time)
        for i := 0; i < len(relations); i++ {
                a, b := relations[i][0], relations[i][1] // a => b
                arr[b] = append(arr[b], a)
                degree[a]++
        }
        res := 0
        for i := 1; i <= n; i++ {
                if degree[i] == 0 { // 从出度为0的节点往前递归
                        res = max(res, dfs(i))
                }
        }
        return res
}

func dfs(root int) int {
        if _, ok := m[root]; ok {
                return m[root]
        }
        sum := 0
        for i := 0; i < len(arr[root]); i++ {
                sum = max(sum, dfs(arr[root][i]))
        }
        sum = sum + temp[root-1]
        m[root] = sum
        return sum
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

63.4 2065. 最大化一张图中的路径价值 (2)

- 题目

```
给你一张 无向图，图中有 n个节点，节点编号从 0到 n - 1（都包括）。同时给你一个下标从0开始的整数数组values，
其中values[i]是第 i个节点的 价值。同时给你一个下标从 0开始的二维整数数组edges，
其中edges[j] = [uj, vj, timej]表示节点uj 和vj之间有一条需要timej秒才能通过的无向边。
最后，给你一个整数maxTime。
合法路径指的是图中任意一条从节点0开始，最终回到节点 0，且花费的总时间 不超过maxTime秒的一条路径。
你可以访问一个节点任意次。
一条合法路径的 价值定义为路径中 不同节点的价值 之和（每个节点的价值至多算入价值总和中一次）。
请你返回一条合法路径的 最大价值。
注意：每个节点 至多有 四条边与之相连。
示例 1：输入：values = [0,32,10,43], edges = [[0,1,10],[1,2,15],[0,3,10]], maxTime = 49 输出：75
解释：一条可能的路径为：0 -> 1 -> 0 -> 3 -> 0 。总花费时间为 10 + 10 + 10 + 10 = 40 <= 49 。
访问过的节点为 0 ，1 和 3 ，最大路径价值为 0 + 32 + 43 = 75 。
示例 2：输入：values = [5,10,15,20], edges = [[0,1,10],[1,2,10],[0,3,10]], maxTime = 30 输出：25
解释：一条可能的路径为：0 -> 3 -> 0 。总花费时间为 10 + 10 = 20 <= 30 。
访问过的节点为 0 和 3 ，最大路径价值为 5 + 20 = 25 。
示例 3：输入：values = [1,2,3,4], edges = [[0,1,10],[1,2,11],[2,3,12],[1,3,13]], maxTime = 50 输出：7
解释：一条可能的路径为：0 -> 1 -> 3 -> 1 -> 0 。总花费时间为 10 + 13 + 13 + 10 = 46 <= 50 。
访问过的节点为 0 ，1 和 3 ，最大路径价值为 1 + 2 + 4 = 7 。
示例 4：输入：values = [0,1,2], edges = [[1,2,10]], maxTime = 10 输出：0
解释：唯一一条路径为 0 。总花费时间为 0 。
唯一访问过的节点为 0 ，最大路径价值为 0 。
提示：n == values.length
1 <= n <= 1000
0 <= values[i] <= 108
0 <= edges.length <= 2000
edges[j].length == 3
0 <= uj < vj <= n - 1
10 <= timej, maxTime <= 100
[uj, vj]所有节点对 互不相同。
每个节点 至多有四条边。
图可能不连通。
```

- 解题思路

```
var res int
var arr [][][2]int
var visited []bool

func maximalPathQuality(values []int, edges [][]int, maxTime int) int {
        n := len(values)
        arr = make([][][2]int, n) // 邻接表
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1], edges[i][2]
                arr[a] = append(arr[a], [2]int{b, c})
                arr[b] = append(arr[b], [2]int{a, c})
        }
        visited = make([]bool, n)
        visited[0] = true
        res = 0
        dfs(values, maxTime, 0, 0, values[0]) // 初始带着0点的价值
        return res
}

func dfs(values []int, maxTime int, start int, t int, sum int) {
        if start == 0 {
                res = max(res, sum)
        }
        for i := 0; i < len(arr[start]); i++ {
                next, c := arr[start][i][0], arr[start][i][1]
                if t+c <= maxTime { // 时间在范围内
                        if visited[next] == false { // 该点没有出现过，加上该点的价值
                                visited[next] = true
                                dfs(values, maxTime, next, t+c, sum+values[next])
                                visited[next] = false
                        } else { // 该点出现过，不加价值，只加时间
                                dfs(values, maxTime, next, t+c, sum)
                        }
                }
        }
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
var res int
var arr [][][2]int
var visited []int

func maximalPathQuality(values []int, edges [][]int, maxTime int) int {
        n := len(values)
        arr = make([][][2]int, n) // 邻接表
        for i := 0; i < len(edges); i++ {
                a, b, c := edges[i][0], edges[i][1], edges[i][2]
                arr[a] = append(arr[a], [2]int{b, c})
                arr[b] = append(arr[b], [2]int{a, c})
        }
        res = 0
        visited = make([]int, n)
        dfs(values, maxTime, 0, 0, 0)
        return res
}

func dfs(values []int, maxTime int, start int, t int, sum int) {
        if t > maxTime {
                return
        }

        if visited[start] == 0 {
                sum = sum + values[start]
        }
        visited[start]++
        if start == 0 {
                res = max(res, sum)
        }
        for i := 0; i < len(arr[start]); i++ {
                next, c := arr[start][i][0], arr[start][i][1]
                dfs(values, maxTime, next, t+c, sum)
        }
        visited[start]--
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

63.5 2076. 处理含限制条件的好友请求 (2)

- 题目

```
给你一个整数 n ，表示网络上的用户数目。每个用户按从 0 到 n - 1 进行编号。
给你一个下标从 0 开始的二维整数数组 restrictions ，
其中 restrictions[i] = [xi, yi] 意味着用户 xi 和用户 yi 不能 成为 朋友 ，不管是 直接␣
↪还是通过其他用户 间接 。
最初，用户里没有人是其他用户的朋友。给你一个下标从 0 开始的二维整数数组 requests␣
↪表示好友请求的列表，
其中 requests[j] = [uj, vj] 是用户 uj 和用户 vj 之间的一条好友请求。
如果 uj 和 vj 可以成为 朋友 ，那么好友请求将会 成功 。
每个好友请求都会按列表中给出的顺序进行处理（即，requests[j] 会在 requests[j + 1]␣
↪前）。
一旦请求成功，那么对所有未来的好友请求而言， uj 和 vj 将会 成为直接朋友 。
返回一个 布尔数组 result ，其中元素遵循此规则：如果第 j 个好友请求 成功 ，那么␣
↪result[j] 就是 true ；否则，为 false 。
注意：如果 uj 和 vj 已经是直接朋友，那么他们之间的请求将仍然成功 。
示例 1：输入：n = 3, restrictions = [[0,1]], requests = [[0,2],[2,1]] 输出：[true,
↪false]
解释：请求 0 ：用户 0 和 用户 2 可以成为朋友，所以他们成为直接朋友。
请求 1 ：用户 2 和 用户 1 不能成为朋友，因为这会使 用户 0 和 用户 1 成为间接朋友 (1--
↪2--0) 。
示例 2：输入：n = 3, restrictions = [[0,1]], requests = [[1,2],[0,2]] 输出：[true,
↪false]
解释：请求 0 ：用户 1 和 用户 2 可以成为朋友，所以他们成为直接朋友。
请求 1 ：用户 0 和 用户 2 不能成为朋友，因为这会使 用户 0 和 用户 1 成为间接朋友 (0--
↪2--1) 。
示例 3：输入：n = 5, restrictions = [[0,1],[1,2],[2,3]], requests = [[0,4],[1,2],[3,
↪1],[3,4]]
输出：[true,false,true,false]
解释：请求 0 ：用户 0 和 用户 4 可以成为朋友，所以他们成为直接朋友。
请求 1 ：用户 1 和 用户 2 不能成为朋友，因为他们之间存在限制。
请求 2 ：用户 3 和 用户 1 可以成为朋友，所以他们成为直接朋友。
请求 3 ：用户 3 和 用户 4 不能成为朋友，因为这会使 用户 0 和 用户 1 成为间接朋友 (0--
↪4--3--1) 。
提示：2 <= n <= 1000
0 <= restrictions.length <= 1000
restrictions[i].length == 2
0 <= xi, yi <= n - 1
xi != yi
1 <= requests.length <= 1000
requests[j].length == 2
0 <= uj, vj <= n - 1
```

(续下页)

(接上页)

```
uj != vj
```

- 解题思路

```
func friendRequests(n int, restrictions [][]int, requests [][]int) []bool {
        fa = Init(n)
        res := make([]bool, len(requests))
        for i := 0; i < len(requests); i++ {
                a, b := requests[i][0], requests[i][1]
                x, y := find(a), find(b)
                if x == y {
                        res[i] = true
                } else {
                        flag := true
                        for j := 0; j < len(restrictions); j++ { // 尝试每个限制条件
                                c, d := restrictions[j][0], restrictions[j][1]
                                u, v := find(c), find(d)
                                if (x == u && y == v) || (x == v && y == u) { //␣
↪有限制
                                        flag = false
                                        break
                                }
                        }
                        if flag == true {
                                res[i] = true
                                union(x, y)
                        }
                }
        }
        return res
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
```

(续下页)

(接上页)

```
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}

# 2
func friendRequests(n int, restrictions [][]int, requests [][]int) []bool {
        fa = Init(n)
        res := make([]bool, len(requests))
        for i := 0; i < len(requests); i++ {
                a, b := requests[i][0], requests[i][1]
                temp := make([]int, n)
                copy(temp, fa) // 备份当前的结果
                union(a, b)    // 尝试连接
                flag := true
                for j := 0; j < len(restrictions); j++ {
                        c, d := restrictions[j][0], restrictions[j][1]
                        if query(c, d) == true { // 查看是否有限制
                                flag = false
                                break
                        }
                }
                if flag == true {
                        res[i] = true
                } else {
                        fa = make([]int, n) // 不满足条件，恢复回去
                        copy(fa, temp)
                }
        }
        return res
}
```

(续下页)

(接上页)

```
var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

63.6 2092. 找出知晓秘密的所有专家

63.6.1 题目

```
给你一个整数 n ，表示有 n 个专家从 0 到 n - 1 编号。另外给你一个下标从 0␣
↪开始的二维整数数组 meetings ，
其中 meetings[i] = [xi, yi, timei] 表示专家 xi 和专家 yi 在时间 timei 要开一场会。
一个专家可以同时参加 多场会议 。最后，给你一个整数 firstPerson 。
专家 0 有一个 秘密 ，最初，他在时间0 将这个秘密分享给了专家 firstPerson 。
接着，这个秘密会在每次有知晓这个秘密的专家参加会议时进行传播。
更正式的表达是，每次会议，如果专家 xi 在时间 timei 时知晓这个秘密，那么他将会与专家␣
↪yi 分享这个秘密，反之亦然。
秘密共享是 瞬时发生␣
```

(续下页)

(接上页)

```
↪的。也就是说，在同一时间，一个专家不光可以接收到秘密，还能在其他会议上与其他专家分享。
在所有会议都结束之后，返回所有知晓这个秘密的专家列表。你可以按 任何顺序 返回答案。
示例 1：输入：n = 6, meetings = [[1,2,5],[2,3,8],[1,5,10]], firstPerson = 1 输出：[0,
↪1,2,3,5]
解释：时间 0 ，专家 0 将秘密与专家 1 共享。
时间 5 ，专家 1 将秘密与专家 2 共享。
时间 8 ，专家 2 将秘密与专家 3 共享。
时间 10 ，专家 1 将秘密与专家 5 共享。
因此，在所有会议结束后，专家 0、1、2、3 和 5 都将知晓这个秘密。
示例 2：输入：n = 4, meetings = [[3,1,3],[1,2,2],[0,3,3]], firstPerson = 3 输出：[0,1,
↪3]
解释：时间 0 ，专家 0 将秘密与专家 3 共享。
时间 2 ，专家 1 与专家 2 都不知晓这个秘密。
时间 3 ，专家 3 将秘密与专家 0 和专家 1 共享。
因此，在所有会议结束后，专家 0、1 和 3 都将知晓这个秘密。
示例 3：输入：n = 5, meetings = [[3,4,2],[1,2,1],[2,3,1]], firstPerson = 1 输出：[0,1,
↪2,3,4]
解释：时间 0 ，专家 0 将秘密与专家 1 共享。
时间 1 ，专家 1 将秘密与专家 2 共享，专家 2 将秘密与专家 3 共享。
注意，专家 2 可以在收到秘密的同一时间分享此秘密。
时间 2 ，专家 3 将秘密与专家 4 共享。
因此，在所有会议结束后，专家 0、1、2、3 和 4 都将知晓这个秘密。
示例 4：输入：n = 6, meetings = [[0,2,1],[1,3,1],[4,5,1]], firstPerson = 1 输出：[0,1,
↪2,3]
解释：时间 0 ，专家 0 将秘密与专家 1 共享。
时间 1 ，专家 0 将秘密与专家 2 共享，专家 1 将秘密与专家 3 共享。
因此，在所有会议结束后，专家 0、1、2 和 3 都将知晓这个秘密。
提示：2 <= n <= 105
1 <= meetings.length <= 105
meetings[i].length == 3
0 <= xi, yi <= n - 1
xi != yi
1 <= timei <= 105
1 <= firstPerson <= n - 1
```

63.6.2 解题思路

63.7 2097. 合法重新排列数对 (2)

- 题目

```
给你一个下标从 0开始的二维整数数组pairs，其中pairs[i] = [starti, endi]。
如果 pairs的一个重新排列，满足对每一个下标 i （1 <= i < pairs.length）都有endi-1 ==␣
↪starti ，
那么我们就认为这个重新排列是pairs 的一个 合法重新排列 。
请你返回 任意一个pairs 的合法重新排列。
注意：数据保证至少存在一个 pairs的合法重新排列。
示例 1：输入：pairs = [[5,1],[4,5],[11,9],[9,4]] 输出：[[11,9],[9,4],[4,5],[5,1]]
解释：输出的是一个合法重新排列，因为每一个 endi-1 都等于 starti。
end0 = 9 == 9 = start1
end1 = 4 == 4 = start2
end2 = 5 == 5 = start3
示例 2：输入：pairs = [[1,3],[3,2],[2,1]] 输出：[[1,3],[3,2],[2,1]]
解释：输出的是一个合法重新排列，因为每一个 endi-1 都等于 starti。
end0 = 3 == 3 = start1
end1 = 2 == 2 = start2
重新排列后的数组 [[2,1],[1,3],[3,2]] 和 [[3,2],[2,1],[1,3]] 都是合法的。
示例 3：输入：pairs = [[1,2],[1,3],[2,1]] 输出：[[1,2],[2,1],[1,3]]
解释：输出的是一个合法重新排列，因为每一个 endi-1 都等于 starti。
end0 = 2 == 2 = start1
end1 = 1 == 1 = start2
提示：1 <= pairs.length <= 105
pairs[i].length == 2
0 <= starti, endi <= 109
starti != endi
pairs中不存在一模一样的数对。
至少 存在 一个合法的pairs重新排列。
```

- 解题思路

```
var arr map[int][]int
var res [][]int

func validArrangement(pairs [][]int) [][]int {
        res = make([][]int, 0)
        arr = make(map[int][]int) // 有向图邻接表
```

(续下页)

(接上页)

```
        m := make(map[int]int)
        for i := 0; i < len(pairs); i++ {
                a, b := pairs[i][0], pairs[i][1]
                arr[a] = append(arr[a], b)
                m[b]++ // 入度+1
                m[a]-- // 出度-1
        }
        start := pairs[0][0] // 寻找起始节点
        for k, v := range m {
                if v == -1 {
                        start = k
                        break
                }
        }
        dfs(start)
        for i := 0; i < len(res)/2; i++ {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
        }
        return res
}

func dfs(start int) {
        for len(arr[start]) > 0 {
                next := arr[start][0]
                arr[start] = arr[start][1:]
                dfs(next)
                res = append(res, []int{start, next})
        }
}

# 2
var arr map[int][]int
var path []int

func validArrangement(pairs [][]int) [][]int {
        path = make([]int, 0)
        arr = make(map[int][]int) // 有向图邻接表
        m := make(map[int]int)
        for i := 0; i < len(pairs); i++ {
                a, b := pairs[i][0], pairs[i][1]
                arr[a] = append(arr[a], b)
                m[b]++ // 入度+1
                m[a]-- // 出度-1
```

(续下页)

(接上页)

```
        }
        start := pairs[0][0] // 寻找起始节点
        for k, v := range m {
                if v == -1 {
                        start = k
                        break
                }
        }
        dfs(start)
        res := make([][]int, 0)
        for i := len(path) - 1; i > 0; i-- {
                res = append(res, []int{path[i], path[i-1]})
        }
        return res
}

func dfs(start int) {
        for len(arr[start]) > 0 {
                next := arr[start][0]
                arr[start] = arr[start][1:]
                dfs(next)
        }
        path = append(path, start)
}
```

CHAPTER 64

2101-2200-Easy

64.1 2108. 找出数组中的第一个回文字符串 (1)

- 题目

```
给你一个字符串数组 words ，找出并返回数组中的 第一个回文字符串␣
↪。如果不存在满足要求的字符串，返回一个 空字符串 "" 。
回文字符串 的定义为：如果一个字符串正着读和反着读一样，那么该字符串就是一个␣
↪回文字符串 。
示例 1：输入：words = ["abc","car","ada","racecar","cool"] 输出："ada"
解释：第一个回文字符串是 "ada" 。
注意，"racecar" 也是回文字符串，但它不是第一个。
示例 2：输入：words = ["notapalindrome","racecar"] 输出："racecar"
解释：第一个也是唯一一个回文字符串是 "racecar" 。
示例 3：输入：words = ["def","ghi"] 输出：""
解释：不存在回文字符串，所以返回一个空字符串。
提示：1 <= words.length <= 100
1 <= words[i].length <= 100
words[i] 仅由小写英文字母组成
```

- 解题思路

```
func firstPalindrome(words []string) string {
        for i := 0; i < len(words); i++ {
                if isPalindrome(words[i]) == true {
```

(续下页)

(接上页)

```
                        return words[i]
                }
        }
        return ""
}

func isPalindrome(a string) bool {
        i, j := 0, len(a)-1
        for i < j {
                if a[i] != a[j] {
                        return false
                }
                i++
                j--
        }
        return true
}
```

64.2 2103. 环和杆 (2)

- 题目

```
总计有 n 个环，环的颜色可以是红、绿、蓝中的一种。这些环分布穿在 10 根编号为 0 到 9␣
↪的杆上。
给你一个长度为 2n 的字符串 rings ，表示这 n 个环在杆上的分布。rings␣
↪中每两个字符形成一个 颜色位置对 ，用于描述每个环：
第 i 对中的 第一个 字符表示第 i 个环的 颜色 ('R'、'G'、'B')。
第 i 对中的 第二个 字符表示第 i 个环的 位置，也就是位于哪根杆上 ('0' 到 '9')。
例如，"R3G2B1" 表示：共有 n == 3 个环，红色的环在编号为 3 的杆上，绿色的环在编号为 2␣
↪的杆上，蓝色的环在编号为 1 的杆上。
找出所有集齐 全部三种颜色 环的杆，并返回这种杆的数量。
示例 1：输入：rings = "B0B6G0R6R0R6G9" 输出：1
解释：- 编号 0 的杆上有 3 个环，集齐全部颜色：红、绿、蓝。
- 编号 6 的杆上有 3 个环，但只有红、蓝两种颜色。
- 编号 9 的杆上只有 1 个绿色环。
因此，集齐全部三种颜色环的杆的数目为 1 。
示例 2：输入：rings = "B0R0G0R9R0B0G0" 输出：1
解释：- 编号 0 的杆上有 6 个环，集齐全部颜色：红、绿、蓝。
- 编号 9 的杆上只有 1 个红色环。
因此，集齐全部三种颜色环的杆的数目为 1 。
示例 3：输入：rings = "G4"输出：0
解释：只给了一个环，因此，不存在集齐全部三种颜色环的杆。
```

(续下页)

(接上页)

```
提示：rings.length == 2 * n
1 <= n <= 100
如 i 是 偶数 ，则rings[i] 的值可以取 'R'、'G' 或 'B'（下标从 0 开始计数）
如 i 是 奇数 ，则rings[i] 的值可以取 '0' 到 '9' 中的一个数字（下标从 0 开始计数）
```

- 解题思路

```
func countPoints(rings string) int {
        m := map[byte]int{
                'R': 1,
                'G': 2,
                'B': 4,
        }
        arr := make([]int, 10)
        for i := 0; i < len(rings); i = i + 2 {
                v := int(rings[i+1] - '0')
                arr[v] = arr[v] | m[rings[i]]
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                if arr[i] == 7 {
                        res++
                }
        }
        return res
}

# 2
func countPoints(rings string) int {
        m := make(map[byte]map[byte]bool)
        for i := 0; i < 10; i++ {
                m[byte(i+'0')] = make(map[byte]bool)
        }
        for i := 0; i < len(rings); i = i + 2 {
                m[rings[i+1]][rings[i]] = true
        }
        res := 0
        for _, v := range m {
                if len(v) == 3 {
                        res++
                }
        }
        return res
}
```

64.3 2114. 句子中的最多单词数 (2)

- 题目

```
一个 句子由一些 单词以及它们之间的单个空格组成，句子的开头和结尾不会有多余空格。
给你一个字符串数组sentences，其中sentences[i]表示单个 句子。
请你返回单个句子里 单词的最多数目。
示例 1：输入：sentences = ["alice and bob love leetcode", "i think so too", "this is␣
↪great thanks very much"]
输出：6
解释：- 第一个句子 "alice and bob love leetcode" 总共有 5 个单词。
- 第二个句子 "i think so too" 总共有 4 个单词。
- 第三个句子 "this is great thanks very much" 总共有 6 个单词。
所以，单个句子中有最多单词数的是第三个句子，总共有 6 个单词。
示例 2：输入：sentences = ["please wait", "continue to fight", "continue to win
↪"]输出：3
解释：可能有多个句子有相同单词数。
这个例子中，第二个句子和第三个句子（加粗斜体）有相同数目的单词数。
提示：1 <= sentences.length <= 100
1 <= sentences[i].length <= 100
sentences[i]只包含小写英文字母和' '。
sentences[i]的开头和结尾都没有空格。
sentences[i]中所有单词由单个空格隔开。
```

- 解题思路

```
func mostWordsFound(sentences []string) int {
        res := 0
        for i := 0; i < len(sentences); i++ {
                arr := strings.Fields(sentences[i])
                if len(arr) > res {
                        res = len(arr)
                }
        }
        return res
}

# 2

func mostWordsFound(sentences []string) int {
        res := 0
        for i := 0; i < len(sentences); i++ {
                count := strings.Count(sentences[i], " ") + 1
                if count > res {
```

(续下页)

(接上页)

```
                        res = count
                }
        }
        return res
}
```

64.4 2119. 反转两次的数字 (1)

- 题目

```
反转 一个整数意味着倒置它的所有位。
例如，反转 2021 得到 1202 。反转 12300 得到 321 ，不保留前导零 。
给你一个整数 num ，反转 num 得到 reversed1 ，接着反转 reversed1 得到 reversed2 。
如果 reversed2 等于 num ，返回 true ；否则，返回 false 。
示例 1：输入：num = 526 输出：true
解释：反转 num 得到 625 ，接着反转 625 得到 526 ，等于 num 。
示例 2：输入：num = 1800 输出：false
解释：反转 num 得到 81 ，接着反转 81 得到 18 ，不等于 num 。
示例 3：输入：num = 0 输出：true
解释：反转 num 得到 0 ，接着反转 0 得到 0 ，等于 num 。
提示：0 <= num <= 106
```

- 解题思路

```
func isSameAfterReversals(num int) bool {
        return num == 0 || num%10 != 0
}
```

64.5 2124. 检查是否所有 A 都在 B 之前 (2)

- 题目

```
给你一个 仅 由字符 'a' 和 'b' 组成的字符串 s 。
如果字符串中 每个 'a' 都出现在 每个 'b' 之前，返回 true ；否则，返回 false 。
示例 1：输入：s = "aaabbb" 输出：true
解释：'a' 位于下标 0、1 和 2 ；而 'b' 位于下标 3、4 和 5 。
因此，每个 'a' 都出现在每个 'b' 之前，所以返回 true 。
示例 2：输入：s = "abab" 输出：false
解释：存在一个 'a' 位于下标 2 ，而一个 'b' 位于下标 1 。
因此，不能满足每个 'a' 都出现在每个 'b' 之前，所以返回 false 。
```

(续下页)

(接上页)

```
示例 3: 输入: s = "bbb" 输出: true
解释: 不存在 'a' ，因此可以视作每个 'a' 都出现在每个 'b' 之前，所以返回 true 。
提示: 1 <= s.length <= 100
s[i] 为 'a' 或 'b'
```

- 解题思路

```
func checkString(s string) bool {
        arr := []byte(s)
        sort.Slice(arr, func(i, j int) bool {
                return arr[i] < arr[j]
        })
        return string(arr) == s
}

# 2
func checkString(s string) bool {
        return strings.Contains(s, "ba") == false
}
```

64.6 2129. 将标题首字母大写 (1)

- 题目

```
给你一个字符串title，它由单个空格连接一个或多个单词组成，每个单词都只包含英文字母。请你按以下规则将每
↪大写:
如果单词的长度为1或者2，所有字母变成小写。
否则，将单词首字母大写，剩余字母变成小写。
请你返回 大写后的title。
示例 1: 输入: title = "capiTalIze tHe titLe" 输出: "Capitalize The Title"
解释: 由于所有单词的长度都至少为 3 ，将每个单词首字母大写，剩余字母变为小写。
示例 2: 输入: title = "First leTTeR of EACH Word" 出: "First Letter of Each Word"
解释: 单词 "of" 长度为 2 ，所以它保持完全小写。
其他单词长度都至少为 3 ，所以其他单词首字母大写，剩余字母小写。
示例 3: 输入: title = "i lOve leetcode" 出: "i Love Leetcode"
解释: 单词 "i" 长度为 1 ，所以它保留小写。
其他单词长度都至少为 3 ，所以其他单词首字母大写，剩余字母小写。
提示: 1 <= title.length <= 100
title由单个空格隔开的单词组成，且不含有任何前导或后缀空格。
每个单词由大写和小写英文字母组成，且都是 非空的。
```

- 解题思路

```
func capitalizeTitle(title string) string {
        arr := strings.Fields(title)
        for i := 0; i < len(arr); i++ {
                arr[i] = strings.ToLower(arr[i])
                if len(arr[i]) > 2 {
                        arr[i] = strings.Title(arr[i])
                }
        }
        return strings.Join(arr, " ")
}
```

64.7 2133. 检查是否每一行每一列都包含全部整数 (1)

- 题目

```
对一个大小为 n x n 的矩阵而言，如果其每一行和每一列都包含从 1 到 n 的 全部 整数（含 1
↪和 n），则认为该矩阵是一个 有效 矩阵。
给你一个大小为 n x n 的整数矩阵 matrix ，请你判断矩阵是否为一个有效矩阵：如果是，返回
↪true ；否则，返回 false 。
示例 1：输入：matrix = [[1,2,3],[3,1,2],[2,3,1]] 输出：true
解释：在此例中，n = 3 ，每一行和每一列都包含数字 1、2、3 。
因此，返回 true 。
示例 2：输入：matrix = [[1,1,1],[1,2,3],[1,2,3]] 输出：false
解释：在此例中，n = 3 ，但第一行和第一列不包含数字 2 和 3 。
因此，返回 false 。
提示：n == matrix.length == matrix[i].length
1 <= n <= 100
1 <= matrix[i][j] <= n
```

- 解题思路

```
func checkValid(matrix [][]int) bool {
        n, m := len(matrix), len(matrix[0])
        for i := 0; i < n; i++ {
                visited := make(map[int]bool)
                for j := 0; j < m; j++ {
                        if visited[matrix[i][j]] == true {
                                return false
                        }
                        visited[matrix[i][j]] = true
                }
        }
        for j := 0; j < m; j++ {
```

(续下页)

(接上页)

```
                visited := make(map[int]bool)
                for i := 0; i < n; i++ {
                        if visited[matrix[i][j]] == true {
                                return false
                        }
                        visited[matrix[i][j]] = true
                }
        }
        return true
}
```

64.8 2138. 将字符串拆分为若干长度为 k 的组 (1)

- 题目

```
字符串 s 可以按下述步骤划分为若干长度为 k 的组：
第一组由字符串中的前 k 个字符组成，第二组由接下来的 k␣
↪个字符串组成，依此类推。每个字符都能够成为 某一个 组的一部分。
对于最后一组，如果字符串剩下的字符 不足 k 个，需使用字符 fill 来补全这一组字符。
注意，在去除最后一个组的填充字符␣
↪fill（如果存在的话）并按顺序连接所有的组后，所得到的字符串应该是 s 。
给你一个字符串 s ，以及每组的长度 k 和一个用于填充的字符 fill ，
按上述步骤处理之后，返回一个字符串数组，该数组表示 s 分组后 每个组的组成情况 。
示例 1：输入：s = "abcdefghi", k = 3, fill = "x" 输出：["abc","def","ghi"]
解释：前 3 个字符是 "abc" ，形成第一组。
接下来 3 个字符是 "def" ，形成第二组。
最后 3 个字符是 "ghi" ，形成第三组。
由于所有组都可以由字符串中的字符完全填充，所以不需要使用填充字符。
因此，形成 3 组，分别是 "abc"、"def" 和 "ghi" 。
示例 2：输入：s = "abcdefghij", k = 3, fill = "x" 出：["abc","def","ghi","jxx"]
解释：与前一个例子类似，形成前三组 "abc"、"def" 和 "ghi" 。
对于最后一组，字符串中仅剩下字符 'j' 可以用。为了补全这一组，使用填充字符 'x' 两次。
因此，形成 4 组，分别是 "abc"、"def"、"ghi" 和 "jxx" 。
提示：1 <= s.length <= 100
s 仅由小写英文字母组成
1 <= k <= 100
fill 是一个小写英文字母
```

- 解题思路

```
func divideString(s string, k int, fill byte) []string {
        n := len(s)
```

(续下页)

(接上页)

```
        res := make([]string, 0)
        for i := 0; i < n; i = i + k {
                if i+k <= n {
                        res = append(res, s[i:i+k])
                } else {
                        res = append(res, s[i:]+strings.Repeat(string(fill), k-(n-i)))
                }
        }
        return res
}
```

64.9 2144. 打折购买糖果的最小开销 (1)

- 题目

```
一家商店正在打折销售糖果。每购买 两个糖果，商店会 免费送一个糖果。
免费送的糖果唯一的限制是：它的价格需要小于等于购买的两个糖果价格的 较小值。
比方说，总共有 4个糖果，价格分别为1，2，3和4，一位顾客买了价格为2
↪和3的糖果，那么他可以免费获得价格为 1的糖果，
但不能获得价格为4的糖果。
给你一个下标从 0开始的整数数组cost，其中cost[i]表示第i个糖果的价格，请你返回获得
↪所有糖果的 最小总开销。
示例 1：输入：cost = [1,2,3] 输出：5
解释：我们购买价格为 2 和 3 的糖果，然后免费获得价格为 1 的糖果。
总开销为 2 + 3 = 5 。这是开销最小的 唯一方案。
注意，我们不能购买价格为 1 和 3 的糖果，并免费获得价格为 2 的糖果。
这是因为免费糖果的价格必须小于等于购买的 2 个糖果价格的较小值。
示例 2：输入：cost = [6,5,7,9,2,2] 输出：23
解释：最小总开销购买糖果方案为：
- 购买价格为 9 和 7 的糖果
- 免费获得价格为 6 的糖果
- 购买价格为 5 和 2 的糖果
- 免费获得价格为 2 的最后一个糖果
因此，最小总开销为 9 + 7 + 5 + 2 = 23 。
示例 3：输入：cost = [5,5] 输出：10
解释：由于只有 2 个糖果，我们需要将它们都购买，而且没有免费糖果。
所以总最小开销为 5 + 5 = 10 。
提示：1 <= cost.length <= 100
1 <= cost[i] <= 100
```

- 解题思路

```
func minimumCost(cost []int) int {
        sort.Slice(cost, func(i, j int) bool {
                return cost[i] > cost[j]
        })
        res := 0
        for i := 0; i < len(cost); i++ {
                if i%3 != 2 {
                        res = res + cost[i]
                }
        }
        return res
}
```

64.10 2148. 元素计数 (2)

- 题目

```
给你一个整数数组 nums ，统计并返回在 nums␣
↪中同时具有一个严格较小元素和一个严格较大元素的元素数目。
示例 1：输入：nums = [11,7,2,15] 输出：2
解释：元素 7 ：严格较小元素是元素 2 ，严格较大元素是元素 11 。
元素 11 ：严格较小元素是元素 7 ，严格较大元素是元素 15 。
总计有 2 个元素都满足在 nums 中同时存在一个严格较小元素和一个严格较大元素。
示例 2：输入：nums = [-3,3,3,90] 输出：2
解释：元素 3 ：严格较小元素是元素 -3 ，严格较大元素是元素 90 。
由于有两个元素的值为 3 ，总计有 2 个元素都满足在 nums␣
↪中同时存在一个严格较小元素和一个严格较大元素。
提示：1 <= nums.length <= 100
-105 <= nums[i] <= 105
```

- 解题思路

```
func countElements(nums []int) int {
        sort.Ints(nums)
        minValue, maxValue := nums[0], nums[len(nums)-1]
        res := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] != minValue && nums[i] != maxValue {
                        res++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func countElements(nums []int) int {
        minValue, maxValue := nums[0], nums[0]
        minCount, maxCount := 1, 1
        for i := 1; i < len(nums); i++ {
                if nums[i] > maxValue {
                        maxValue = nums[i]
                        maxCount = 1
                } else if nums[i] == maxValue {
                        maxCount++
                }
                if nums[i] < minValue {
                        minValue = nums[i]
                        minCount = 1
                } else if nums[i] == minValue {
                        minCount++
                }
        }
        if maxValue == minValue {
                return 0
        }
        return len(nums) - maxCount - minCount
}
```

64.11 2154. 将找到的值乘以 2(2)

- 题目

```
给你一个整数数组 nums ，另给你一个整数 original ，这是需要在 nums 中搜索的第一个数字。
接下来，你需要按下述步骤操作：
如果在 nums 中找到 original ，将 original乘以 2 ，得到新 original（即，令 original =␣
↪2 * original）。
否则，停止这一过程。
只要能在数组中找到新 original ，就对新 original 继续 重复 这一过程。
返回 original 的 最终 值。
示例 1：输入：nums = [5,3,6,1,12], original = 3 输出：24
解释：  - 3 能在 nums 中找到。3 * 2 = 6 。
- 6 能在 nums 中找到。6 * 2 = 12 。
- 12 能在 nums 中找到。12 * 2 = 24 。
- 24 不能在 nums 中找到。因此，返回 24 。
示例 2：输入：nums = [2,7,9], original = 4 输出：4
```

(续下页)

(接上页)

```
解释：- 4 不能在 nums 中找到。因此，返回 4 。
提示：1 <= nums.length <= 1000
1 <= nums[i], original <= 1000
```

- 解题思路

```
func findFinalValue(nums []int, original int) int {
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = true
        }
        for m[original] == true {
                original = original * 2
        }
        return original
}

# 2
func findFinalValue(nums []int, original int) int {
        sort.Ints(nums)
        for i := 0; i < len(nums); i++ {
                if nums[i] == original {
                        original = original * 2
                }
        }
        return original
}
```

64.12 2160. 拆分数位后四位数字的最小和 (1)

- 题目

```
给你一个四位正整数num。请你使用 num中的 数位 ，将num拆成两个新的整数new1和new2。
new1 和new2中可以有前导 0，且num中 所有数位都必须使用。
比方说，给你num = 2932，你拥有的数位包括：两个2，一个9和一个3。
一些可能的[new1, new2]数对为[22, 93]，[23, 92]，[223, 9] 和[2, 329]。
请你返回可以得到的new1和 new2的 最小和。
示例 1：输入：num = 2932 输出：52
解释：可行的 [new1, new2] 数对为 [29, 23] ，[223, 9] 等等。
最小和为数对 [29, 23] 的和：29 + 23 = 52 。
示例 2：输入：num = 4009 输出：13
解释：可行的 [new1, new2] 数对为 [0, 49] ，[490, 0] 等等。
```

(续下页)

(接上页)

```
最小和为数对 [4, 9] 的和：4 + 9 = 13 。
提示：1000 <= num <= 9999
```

- 解题思路

```
func minimumSum(num int) int {
        arr := make([]int, 0)
        for num > 0 {
                arr = append(arr, num%10)
                num = num / 10
        }
        sort.Ints(arr)
        return 10*(arr[0]+arr[1]) + arr[2] + arr[3]
}
```

64.13 2164. 对奇偶下标分别排序 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 nums 。根据下述规则重排 nums 中的值：
按 非递增 顺序排列 nums 奇数下标 上的所有值。
举个例子，如果排序前 nums = [4,1,2,3] ，对奇数下标的值排序后变为 [4,3,2,1] 。奇数下标␣
↪1 和 3 的值按照非递增顺序重排。
按 非递减 顺序排列 nums 偶数下标 上的所有值。
举个例子，如果排序前 nums = [4,1,2,3] ，对偶数下标的值排序后变为 [2,1,4,3] 。偶数下标␣
↪0 和 2 的值按照非递减顺序重排。
返回重排 nums 的值之后形成的数组。
示例 1：输入：nums = [4,1,2,3] 输出：[2,3,4,1]
解释：首先，按非递增顺序重排奇数下标 (1 和 3) 的值。
所以，nums 从 [4,1,2,3] 变为 [4,3,2,1] 。
然后，按非递减顺序重排偶数下标 (0 和 2) 的值。
所以，nums 从 [4,1,2,3] 变为 [2,3,4,1] 。
因此，重排之后形成的数组是 [2,3,4,1] 。
示例 2：输入：nums = [2,1] 输出：[2,1]
解释：由于只有一个奇数下标和一个偶数下标，所以不会发生重排。
形成的结果数组是 [2,1] ，和初始数组一样。
提示：1 <= nums.length <= 100
1 <= nums[i] <= 100
```

- 解题思路

```
func sortEvenOdd(nums []int) []int {
        even, odd := make([]int, 0), make([]int, 0)
        for i := 0; i < len(nums); i++ {
                if i%2 == 0 {
                        even = append(even, nums[i])
                } else {
                        odd = append(odd, nums[i])
                }
        }
        sort.Ints(even)
        sort.Slice(odd, func(i, j int) bool {
                return odd[i] > odd[j]
        })
        for i := 0; i < len(even); i++ {
                nums[2*i] = even[i]
        }
        for i := 0; i < len(odd); i++ {
                nums[2*i+1] = odd[i]
        }
        return nums
}
```

64.14 2169. 得到 0 的操作数 (2)

- 题目

```
给你两个 非负 整数 num1 和 num2 。
每一步 操作中，如果 num1 >= num2 ，你必须用 num1 减 num2 ；否则，你必须用 num2 减␣
↪num1 。
例如，num1 = 5 且 num2 = 4 ，应该用num1 减 num2 ，因此，得到 num1 = 1 和 num2 = 4 。
然而，如果 num1 = 4且 num2 = 5 ，一步操作后，得到 num1 = 4 和 num2 = 1 。
返回使 num1 = 0 或 num2 = 0 的 操作数 。
示例 1：输入：num1 = 2, num2 = 3 输出：3
解释：- 操作 1 ：num1 = 2 ，num2 = 3 。由于 num1 < num2 ，num2 减 num1 得到 num1 = 2␣
↪，num2 = 3 - 2 = 1 。
- 操作 2 ：num1 = 2 ，num2 = 1 。由于 num1 > num2 ，num1 减 num2 。
- 操作 3 ：num1 = 1 ，num2 = 1 。由于 num1 == num2 ，num1 减 num2 。
此时 num1 = 0 ，num2 = 1 。由于 num1 == 0 ，不需要再执行任何操作。
所以总操作数是 3 。
示例 2：输入：num1 = 10, num2 = 10 输出：1
解释：- 操作 1 ：num1 = 10 ，num2 = 10 。由于 num1 == num2 ，num1 减 num2 得到 num1 =␣
↪10 - 10 = 0 。
```

(续下页)

(接上页)

```
此时 num1 = 0 ，num2 = 10 。由于 num1 == 0 ，不需要再执行任何操作。
所以总操作数是 1 。
提示：0 <= num1, num2 <= 105
```

- 解题思路

```
func countOperations(num1 int, num2 int) int {
        res := 0
        for num1 > 0 {
                res = res + num2/num1
                num1, num2 = num2%num1, num1
        }
        return res
}

# 2
func countOperations(num1 int, num2 int) int {
        res := 0
        for num1 > 0 && num2 > 0 {
                if num1 >= num2 {
                        num1 = num1 - num2
                } else {
                        num2 = num2 - num1
                }
                res++
        }
        return res
}
```

64.15 2176. 统计数组中相等且可以被整除的数对 (1)

- 题目

```
给你一个下标从 0开始长度为 n的整数数组nums和一个整数k，
请你返回满足0 <= i < j < n，nums[i] == nums[j] 且(i * j)能被k整除的数对(i, j)的数目。
示例 1：输入：nums = [3,1,2,2,2,1,3]，k = 2 输出：4
解释：总共有 4 对数符合所有要求：
- nums[0] == nums[6] 且 0 * 6 == 0 ，能被 2 整除。
- nums[2] == nums[3] 且 2 * 3 == 6 ，能被 2 整除。
- nums[2] == nums[4] 且 2 * 4 == 8 ，能被 2 整除。
- nums[3] == nums[4] 且 3 * 4 == 12 ，能被 2 整除。
示例 2：输入：nums = [1,2,3,4]，k = 1 输出：0
```

(续下页)

(接上页)

```
解释：由于数组中没有重复数值，所以没有数对 (i,j) 符合所有要求。
提示：1 <= nums.length <= 100
1 <= nums[i], k <= 100
```

- 解题思路

```
func countPairs(nums []int, k int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i] == nums[j] && i*j%k == 0 {
                                res++
                        }
                }
        }
        return res
}
```

64.16 2180. 统计各位数字之和为偶数的整数个数 (2)

- 题目

```
给你一个正整数 num ，请你统计并返回 小于或等于 num 且各位数字之和为 偶数␣
↪的正整数的数目。
正整数的 各位数字之和 是其所有位上的对应数字相加的结果。
示例 1：输入：num = 4 输出：2
解释：只有 2 和 4 满足小于等于 4 且各位数字之和为偶数。
示例 2：输入：num = 30 输出：14
解释：只有 14 个整数满足小于等于 30 且各位数字之和为偶数，分别是：
2、4、6、8、11、13、15、17、19、20、22、24、26 和 28 。
提示：1 <= num <= 1000
```

- 解题思路

```
func countEven(num int) int {
        res := 0
        for i := 2; i <= num; i++ {
                sum := 0
                temp := i
                for temp > 0 {
                        sum = sum + temp%10
                        temp = temp / 10
```

(续下页)

(接上页)

```
                }
                if sum%2 == 0 {
                        res++
                }
        }
        return res
}

# 2
func countEven(num int) int {
        a := num / 10 * 5 // 每10包含5个数
        b := 0            // 十位以上的数字之和
        for temp := num / 10; temp > 0; temp = temp / 10 {
                b = b + temp%10
        }
        if b%2 == 0 { // 偶数
                return a + (num%10+2)/2 - 1
        }
        return a + (num%10+1)/2 - 1
}
```

64.17 2185. 统计包含给定前缀的字符串 (2)

- 题目

```
给你一个字符串数组 words 和一个字符串 pref 。
返回 words 中以 pref 作为 前缀 的字符串的数目。
字符串 s 的 前缀 就是 s 的任一前导连续字符串。
示例 1：输入：words = ["pay","attention","practice","attend"], pref = "at" 输出：2
解释：以 "at" 作为前缀的字符串有两个，分别是："attention" 和 "attend" 。
示例 2：输入：words = ["leetcode","win","loops","success"], pref = "code" 输出：0
解释：不存在以 "code" 作为前缀的字符串。
提示：1 <= words.length <= 100
1 <= words[i].length, pref.length <= 100
words[i] 和 pref 由小写英文字母组成
```

- 解题思路

```
func prefixCount(words []string, pref string) int {
        res := 0
        for i := 0; i < len(words); i++ {
                if strings.HasPrefix(words[i], pref) {
```

(续下页)

(接上页)

```
                        res++
                }
        }
        return res
}

# 2
func prefixCount(words []string, pref string) int {
        res := 0
        for i := 0; i < len(words); i++ {
                if len(words[i]) >= len(pref) && words[i][:len(pref)] == pref {
                        res++
                }
        }
        return res
}
```

64.18 2190. 数组中紧跟 key 之后出现最频繁的数字 (2)

- 题目

```
给你一个下标从 0开始的整数数组nums，同时给你一个整数key，它在nums出现过。
统计在 nums数组中紧跟着␣
↪key后面出现的不同整数target的出现次数。换言之，target的出现次数为满足以下条件的␣
↪i的数目：
0 <= i <= n - 2
nums[i] == key且
nums[i + 1] == target。
请你返回出现 最多次数的target。测试数据保证出现次数最多的 target是唯一的。
示例 1：输入：nums = [1,100,200,1,100], key = 1 输出：100
解释：对于 target = 100 ，在下标 1 和 4 处出现过 2 次，且都紧跟着 key 。
没有其他整数在 key 后面紧跟着出现，所以我们返回 100 。
示例 2：输入：nums = [2,2,2,2,3], key = 2 输出：2
解释：对于 target = 2 ，在下标 1 ，2 和 3 处出现过 3 次，且都紧跟着 key 。
对于 target = 3 ，在下标 4 出出现过 1 次，且紧跟着 key 。
target = 2 是紧跟着 key 之后出现次数最多的数字，所以我们返回 2 。
提示：2 <= nums.length <= 1000
1 <= nums[i] <= 1000
测试数据保证答案是唯一的。
```

- 解题思路

```
func mostFrequent(nums []int, key int) int {
        m := make(map[int]int)
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] == key {
                        m[nums[i+1]]++
                }
        }
        res, count := 0, 0
        for k, v := range m {
                if v > count {
                        count = v
                        res = k
                }
        }
        return res
}

# 2
func mostFrequent(nums []int, key int) int {
        m := make(map[int]int)
        res, count := 0, 0
        for i := 0; i < len(nums)-1; i++ {
                if nums[i] == key {
                        m[nums[i+1]]++
                        if m[nums[i+1]] > count {
                                count = m[nums[i+1]]
                                res = nums[i+1]
                        }
                }
        }
        return res
}
```

64.19 2191. 将杂乱无章的数字排序 (1)

- 题目

```
给你一个下标从 0开始的整数数组mapping，它表示一个十进制数的映射规则，
mapping[i] = j表示这个规则下将数位i映射为数位 j。
一个整数 映射后的值为将原数字每一个数位 i（0 <= i <= 9）映射为mapping[i]。
另外给你一个整数数组nums，请你将数组nums中每个数按照它们映射后对应数字非递减顺序排序后返回。
注意：如果两个数字映射后对应的数字大小相同，则将它们按照输入中的 相对顺序排序。
```

(续下页)

(接上页)

```
nums中的元素只有在排序的时候需要按照映射后的值进行比较，返回的值应该是输入的元素本身。
示例 1：输入：mapping = [8,9,4,0,2,1,3,5,7,6], nums = [991,338,38] 输出：[338,38,991]
解释：将数字 991 按照如下规则映射：
1. mapping[9] = 6 ，所有数位 9 都会变成 6 。
2. mapping[1] = 9 ，所有数位 1 都会变成 8 。
所以，991 映射的值为 669 。
338 映射为 007 ，去掉前导 0 后得到 7 。
38 映射为 07 ，去掉前导 0 后得到 7 。
由于 338 和 38 映射后的值相同，所以它们的前后顺序保留原数组中的相对位置关系，338 在
↪38 的前面。
所以，排序后的数组为 [338,38,991] 。
示例 2：输入：mapping = [0,1,2,3,4,5,6,7,8,9], nums = [789,456,123] 输出：[123,456,
↪789]
解释：789 映射为 789 ，456 映射为 456 ，123 映射为 123 。所以排序后数组为 [123,456,
↪789] 。
提示：mapping.length == 10
0 <= mapping[i] <= 9
mapping[i]的值 互不相同。
1 <= nums.length <= 3 * 104
0 <= nums[i] < 109
```

- 解题思路

```
func sortJumbled(mapping []int, nums []int) []int {
        sort.SliceStable(nums, func(i, j int) bool {
                return transfer(mapping, nums[i]) < transfer(mapping, nums[j])
        })
        return nums
}

func transfer(mapping []int, num int) int {
        res := 0
        if num == 0 {
                return mapping[0]
        }
        v := 1
        for ; num > 0; num = num / 10 {
                res = res + mapping[num%10]*v
                v = v * 10
        }
        return res
}
```

64.20 2194.Excel 表中某个范围内的单元格 (1)

- 题目

```
Excel 表中的一个单元格 (r, c) 会以字符串 "<col><row>" 的形式进行表示，其中：
<col> 即单元格的列号 c 。用英文字母表中的 字母 标识。
例如，第 1 列用 'A' 表示，第 2 列用 'B' 表示，第 3 列用 'C' 表示，以此类推。
<row> 即单元格的行号 r 。第 r 行就用 整数 r 标识。
给你一个格式为 "<col1><row1>:<col2><row2>" 的字符串 s ，
其中 <col1> 表示 c1 列，<row1> 表示 r1 行，<col2> 表示 c2 列，<row2> 表示 r2␣
↪行，并满足 r1 <= r2 且 c1 <= c2 。
找出所有满足r1 <= x <= r2 且 c1 <= y <= c2 的单元格，并以列表形式返回。
单元格应该按前面描述的格式用 字符串 表示，并以 非递减 顺序排列（先按列排，再按行排）。
示例 1：输入：s = "K1:L2" 输出：["K1","K2","L1","L2"]
解释：上图显示了列表中应该出现的单元格。
红色箭头指示单元格的出现顺序。
示例 2：输入：s = "A1:F1" 输出：["A1","B1","C1","D1","E1","F1"]
解释：上图显示了列表中应该出现的单元格。
红色箭头指示单元格的出现顺序。
提示：s.length == 5
'A' <= s[0] <= s[3] <= 'Z'
'1' <= s[1] <= s[4] <= '9'
s 由大写英文字母、数字、和 ':' 组成
```

- 解题思路

```go
func cellsInRange(s string) []string {
        res := make([]string, 0)
        for i := s[0]; i <= s[3]; i++ {
                for j := s[1]; j <= s[4]; j++ {
                        res = append(res, string(i)+string(j))
                }
        }
        return res
}
```

64.21 2200. 找出数组中的所有 K 近邻下标 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums 和两个整数 key 和 k 。
K 近邻下标 是 nums 中的一个下标 i ，并满足至少存在一个下标 j 使得 |i - j| <= k 且␣
↪nums[j] == key 。
以列表形式返回按 递增顺序 排序的所有 K 近邻下标。
示例 1：输入：nums = [3,4,9,1,3,9,5], key = 9, k = 1 输出：[1,2,3,4,5,6]
解释：因此，nums[2] == key 且 nums[5] == key 。
- 对下标 0 ，|0 - 2| > k 且 |0 - 5| > k ，所以不存在 j 使得 |0 - j| <= k 且 nums[j]␣
↪== key 。
所以 0 不是一个 K 近邻下标。
- 对下标 1 ，|1 - 2| <= k 且 nums[2] == key ，所以 1 是一个 K 近邻下标。
- 对下标 2 ，|2 - 2| <= k 且 nums[2] == key ，所以 2 是一个 K 近邻下标。
- 对下标 3 ，|3 - 2| <= k 且 nums[2] == key ，所以 3 是一个 K 近邻下标。
- 对下标 4 ，|4 - 5| <= k 且 nums[5] == key ，所以 4 是一个 K 近邻下标。
- 对下标 5 ，|5 - 5| <= k 且 nums[5] == key ，所以 5 是一个 K 近邻下标。
- 对下标 6 ，|6 - 5| <= k 且 nums[5] == key ，所以 6 是一个 K 近邻下标。
因此，按递增顺序返回 [1,2,3,4,5,6] 。
示例 2：输入：nums = [2,2,2,2,2], key = 2, k = 2 输出：[0,1,2,3,4]
解释：对 nums 的所有下标 i ，总存在某个下标 j 使得 |i - j| <= k 且 nums[j] == key␣
↪，所以每个下标都是一个 K 近邻下标。
因此，返回 [0,1,2,3,4] 。
提示：1 <= nums.length <= 1000
1 <= nums[i] <= 1000
key 是数组 nums 中的一个整数
1 <= k <= nums.length
```

- 解题思路

```
func findKDistantIndices(nums []int, key int, k int) []int {
        n := len(nums)
        res := make([]int, 0)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if nums[j] == key && abs(i-j) <= k {
                                res = append(res, i)
                                break
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

# 2
func findKDistantIndices(nums []int, key int, k int) []int {
        n := len(nums)
        res := make([]int, 0)
        right := 0
        for j := 0; j < n; j++ {
                if nums[j] == key {
                        left := max(right, j-k)   // 往前k or 已经访问过的
                        right = min(n-1, j+k) + 1 // 更新
                        for i := left; i < right; i++ {
                                res = append(res, i)
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

CHAPTER 65

2101-2200-Medium

65.1 2101. 引爆最多的炸弹 (2)

- 题目

```
给你一个炸弹列表。一个炸弹的 爆炸范围定义为以炸弹为圆心的一个圆。
炸弹用一个下标从 0开始的二维整数数组bombs表示，其中bombs[i] = [xi, yi, ri]。
xi 和yi表示第 i个炸弹的 X 和 Y 坐标，ri表示爆炸范围的 半径。
你需要选择引爆 一个炸弹。当这个炸弹被引爆时，所有 在它爆炸范围内的炸弹都会被引爆，
这些炸弹会进一步将它们爆炸范围内的其他炸弹引爆。
给你数组bombs，请你返回在引爆一个炸弹的前提下，最多能引爆的炸弹数目。
示例 1: 输入: bombs = [[2,1,3],[6,1,4]] 输出: 2
解释: 上图展示了 2 个炸弹的位置和爆炸范围。
如果我们引爆左边的炸弹，右边的炸弹不会被影响。
但如果我们引爆右边的炸弹，两个炸弹都会爆炸。
所以最多能引爆的炸弹数目是 max(1, 2) = 2 。
示例 2: 输入: bombs = [[1,1,5],[10,10,5]] 输出: 1
解释: 引爆任意一个炸弹都不会引爆另一个炸弹。所以最多能引爆的炸弹数目为 1 。
示例 3: 输入: bombs = [[1,2,3],[2,3,1],[3,4,2],[4,5,3],[5,6,4]] 输出: 5
解释: 最佳引爆炸弹为炸弹 0 ，因为:
- 炸弹 0 引爆炸弹 1 和 2 。红色圆表示炸弹 0 的爆炸范围。
- 炸弹 2 引爆炸弹 3 。蓝色圆表示炸弹 2 的爆炸范围。
- 炸弹 3 引爆炸弹 4 。绿色圆表示炸弹 3 的爆炸范围。
所以总共有 5 个炸弹被引爆。
提示: 1 <= bombs.length<= 100
```

(续下页)

(接上页)

```
bombs[i].length == 3
1 <= xi, yi, ri <= 105
```

- 解题思路

```
func maximumDetonation(bombs [][]int) int {
        n := len(bombs)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if i != j && judge(bombs[i][0], bombs[i][1], bombs[j][0],␣
↪bombs[j][1], bombs[i][2]) == true {
                                arr[i] = append(arr[i], j)
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                count := 1
                visited := make([]bool, n)
                queue := make([]int, 0)
                queue = append(queue, i)
                visited[i] = true
                for len(queue) > 0 {
                        node := queue[0]
                        queue = queue[1:]
                        for j := 0; j < len(arr[node]); j++ {
                                next := arr[node][j]
                                if visited[next] == false {
                                        count++
                                        queue = append(queue, next)
                                        visited[next] = true
                                }
                        }
                }
                if count > res {
                        res = count
                }
        }
        return res
}

func judge(a, b, c, d, r int) bool {
        return r*r >= (a-c)*(a-c)+(b-d)*(b-d)
```

(续下页)

(接上页)

```
}

# 2
var arr [][]int
var count int

func maximumDetonation(bombs [][]int) int {
        n := len(bombs)
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if i != j && judge(bombs[i][0], bombs[i][1], bombs[j][0],␣
↪bombs[j][1], bombs[i][2]) == true {
                                arr[i] = append(arr[i], j)
                        }
                }
        }
        res := 0
        for i := 0; i < n; i++ {
                count = 0
                dfs(i, make([]bool, n))
                if count > res {
                        res = count
                }
        }
        return res
}

func dfs(start int, visited []bool) {
        visited[start] = true
        count++
        for i := 0; i < len(arr[start]); i++ {
                next := arr[start][i]
                if visited[next] == false {
                        dfs(next, visited)
                }
        }
}

func judge(a, b, c, d, r int) bool {
        return r*r >= (a-c)*(a-c)+(b-d)*(b-d)
}
```

65.2 2104. 子数组范围和 (2)

- 题目

```
给你一个整数数组 nums 。nums 中，子数组的 范围 是子数组中最大元素和最小元素的差值。
返回 nums 中 所有 子数组范围的 和 。
子数组是数组中一个连续 非空 的元素序列。
示例 1：输入：nums = [1,2,3]输出：4
解释：nums 的 6 个子数组如下所示：
[1]，范围 = 最大 - 最小 = 1 - 1 = 0
[2]，范围 = 2 - 2 = 0
[3]，范围 = 3 - 3 = 0
[1,2]，范围 = 2 - 1 = 1
[2,3]，范围 = 3 - 2 = 1
[1,2,3]，范围 = 3 - 1 = 2
所有范围的和是 0 + 0 + 0 + 1 + 1 + 2 = 4
示例 2：输入：nums = [1,3,3]输出：4
解释：nums 的 6 个子数组如下所示：
[1]，范围 = 最大 - 最小 = 1 - 1 = 0
[3]，范围 = 3 - 3 = 0
[3]，范围 = 3 - 3 = 0
[1,3]，范围 = 3 - 1 = 2
[3,3]，范围 = 3 - 3 = 0
[1,3,3]，范围 = 3 - 1 = 2
所有范围的和是 0 + 0 + 0 + 2 + 0 + 2 = 4
示例 3：输入：nums = [4,-2,-3,4,1]输出：59
解释：nums 中所有子数组范围的和是 59
提示：1 <= nums.length <= 1000
-109 <= nums[i] <= 109
进阶：你可以设计一种时间复杂度为 O(n) 的解决方案吗？
```

- 解题思路

```
func subArrayRanges(nums []int) int64 {
        res := int64(0)
        for i := 0; i < len(nums); i++ {
                minValue, maxValue := nums[i], nums[i]
                for j := i + 1; j < len(nums); j++ {
                        if nums[j] > maxValue {
                                maxValue = nums[j]
                        }
                        if nums[j] < minValue {
                                minValue = nums[j]
                        }
```

(续下页)

(接上页)

```
                        res = res + int64(maxValue-minValue)
                }
        }
        return res
}

# 2
func subArrayRanges(nums []int) int64 {
        return int64(sumSubarrayMaxs(nums)) - int64(sumSubarrayMins(nums))
}

// leetcode907_子数组的最小值之和
func sumSubarrayMins(arr []int) int {
        res := 0
        stack := make([]int, 0) // 递增栈
        stack = append(stack, -1)
        total := 0
        for i := 0; i < len(arr); i++ {
                for len(stack) > 1 && arr[i] < arr[stack[len(stack)-1]] { // 小于栈顶
                        prev := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        total = total - arr[prev]*(prev-stack[len(stack)-1])
                }
                stack = append(stack, i)
                total = total + arr[i]*(i-stack[len(stack)-2])
                res = res + total
        }
        return res
}

func sumSubarrayMaxs(arr []int) int {
        res := 0
        stack := make([]int, 0) // 递减栈
        stack = append(stack, -1)
        total := 0
        for i := 0; i < len(arr); i++ {
                for len(stack) > 1 && arr[i] > arr[stack[len(stack)-1]] { // 大于栈顶
                        prev := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        total = total - arr[prev]*(prev-stack[len(stack)-1])
                }
                stack = append(stack, i)
                total = total + arr[i]*(i-stack[len(stack)-2])
```

(续下页)

(接上页)

```
                res = res + total
        }
        return res
}
```

65.3 2105. 给植物浇水 II(2)

- 题目

```
Alice 和 Bob 打算给花园里的 n 株植物浇水。植物排成一行，从左到右进行标记，编号从 0 到␣
↪n - 1 。
其中，第 i 株植物的位置是 x = i 。
每一株植物都需要浇特定量的水。Alice 和 Bob 每人有一个水罐，最初是满的␣
↪。他们按下面描述的方式完成浇水：
Alice 按 从左到右 的顺序给植物浇水，从植物 0 开始。Bob 按 从右到左␣
↪的顺序给植物浇水，从植物 n - 1 开始。
他们 同时 给植物浇水。
如果没有足够的水 完全 浇灌下一株植物，他 / 她会立即重新灌满浇水罐。
不管植物需要多少水，浇水所耗费的时间都是一样的。
不能 提前重新灌满水罐。
每株植物都可以由 Alice 或者 Bob 来浇水。
如果 Alice 和 Bob 到达同一株植物，那么当前水罐中水更多的人会给这株植物浇水。
如果他俩水量相同，那么 Alice 会给这株植物浇水。
给你一个下标从 0 开始的整数数组 plants ，数组由 n 个整数组成。
其中，plants[i] 为第 i 株植物需要的水量。另有两个整数 capacityA 和capacityB 分别表示␣
↪Alice 和 Bob 水罐的容量。
返回两人浇灌所有植物过程中重新灌满水罐的 次数 。
示例 1：输入：plants = [2,2,3,3], capacityA = 5, capacityB = 5 输出：1
解释：- 最初，Alice 和 Bob 的水罐中各有 5 单元水。
- Alice 给植物 0 浇水，Bob 给植物 3 浇水。
- Alice 和 Bob 现在分别剩下 3 单元和 2 单元水。
- Alice 有足够的水给植物 1 ，所以她直接浇水。Bob 的水不够给植物 2␣
↪，所以他先重新装满水，再浇水。
所以，两人浇灌所有植物过程中重新灌满水罐的次数 = 0 + 0 + 1 + 0 = 1 。
示例 2：输入：plants = [2,2,3,3], capacityA = 3, capacityB = 4 输出：2
解释：- 最初，Alice 的水罐中有 3 单元水，Bob 的水罐中有 4 单元水。
- Alice 给植物 0 浇水，Bob 给植物 3 浇水。
- Alice 和 Bob 现在都只有 1 单元水，并分别需要给植物 1 和植物 2 浇水。
- 由于他们的水量均不足以浇水，所以他们重新灌满水罐再进行浇水。
所以，两人浇灌所有植物过程中重新灌满水罐的次数 = 0 + 1 + 1 + 0 = 2 。
示例 3：输入：plants = [5], capacityA = 10, capacityB = 8 输出：0
解释：- 只有一株植物
```

(续下页)

(接上页)

```
- Alice 的水罐有 10 单元水，Bob 的水罐有 8 单元水。因此 Alice␣
↪的水罐中水更多，她会给这株植物浇水。
所以，两人浇灌所有植物过程中重新灌满水罐的次数 = 0 。
示例 4：输入：plants = [1,2,4,4,5], capacityA = 6, capacityB = 5 输出：2
解释：- 最初，Alice 的水罐中有 6 单元水，Bob 的水罐中有 5 单元水。
- Alice 给植物 0 浇水，Bob 给植物 4 浇水。
- Alice 和 Bob 现在分别剩下 5 单元和 0 单元水。
- Alice 有足够的水给植物 1 ，所以她直接浇水。Bob 的水不够给植物 3␣
↪，所以他先重新装满水，再浇水。
- Alice 和 Bob 现在分别剩下 3 单元和 1 单元水。
- 由于 Alice 的水更多，所以由她给植物 2 浇水。然而，她水罐里的水不够给植物 2␣
↪，所以她先重新装满水，再浇水。
所以，两人浇灌所有植物过程中重新灌满水罐的次数 = 0 + 0 + 1 + 1 + 0 = 2 。
示例 5：输入：plants = [2,2,5,2,2], capacityA = 5, capacityB = 5 输出：1
解释：Alice 和 Bob 都会到达中间的植物，并且此时他俩剩下的水量相同，所以 Alice␣
↪会给这株植物浇水。
由于她到达时只剩下 1 单元水，所以需要重新灌满水罐。
这是唯一一次需要重新灌满水罐的情况。所以，两人浇灌所有植物过程中重新灌满水罐的次数 =␣
↪1 。
提示：n == plants.length
1 <= n <= 105
1 <= plants[i] <= 106
max(plants[i]) <= capacityA, capacityB <= 109
```

- 解题思路

```go
func minimumRefill(plants []int, capacityA int, capacityB int) int {
        res := 0
        i, j := 0, len(plants)-1
        a, b := capacityA, capacityB
        for i <= j {
                if i == j { // 相等
                        if a >= b && a < plants[i] {
                                res++
                        }
                        if a < b && b < plants[i] {
                                res++
                        }
                        return res
                }
                if a < plants[i] {
                        res++
                        a = capacityA - plants[i]
                } else {
```

(续下页)

(接上页)

```
                        a = a - plants[i]
                }
                i++
                if b < plants[j] {
                        res++
                        b = capacityB - plants[j]
                } else {
                        b = b - plants[j]
                }
                j--
        }
        return res
}
```

65.4 2109. 向字符串添加空格 (1)

- 题目

```
给你一个下标从 0 开始的字符串 s ，以及一个下标从 0 开始的整数数组 spaces 。
数组 spaces 描述原字符串中需要添加空格的下标。每个空格都应该插入到给定索引处的字符值␣
↪之前 。
例如，s = "EnjoyYourCoffee" 且 spaces = [5, 9] ，那么我们需要在 'Y' 和 'C'␣
↪之前添加空格，
这两个字符分别位于下标 5 和下标 9 。因此，最终得到 "Enjoy Your Coffee" 。
请你添加空格，并返回修改后的字符串。
示例 1：输入：s = "LeetcodeHelpsMeLearn", spaces = [8,13,15] 输出："Leetcode Helps Me␣
↪Learn"
解释：下标 8、13 和 15 对应 "LeetcodeHelpsMeLearn" 中加粗斜体字符。
接着在这些字符前添加空格。
示例 2：输入：s = "icodeinpython", spaces = [1,5,7,9] 输出："i code in py thon"
解释：下标 1、5、7 和 9 对应 "icodeinpython" 中加粗斜体字符。
接着在这些字符前添加空格。
示例 3：输入：s = "spacing", spaces = [0,1,2,3,4,5,6] 输出：" s p a c i n g"
解释：字符串的第一个字符前可以添加空格。
提示：1 <= s.length <= 3 * 105
s 仅由大小写英文字母组成
1 <= spaces.length <= 3 * 105
0 <= spaces[i] <= s.length - 1
spaces 中的所有值 严格递增
```

- 解题思路

```
func addSpaces(s string, spaces []int) string {
        res := make([]byte, 0)
        if len(spaces) == 0 {
                return s
        }
        j := 0
        for i := 0; i < len(s); i++ {
                if j < len(spaces) && i == spaces[j] {
                        res = append(res, ' ')
                        j++
                }
                res = append(res, s[i])
        }
        return string(res)
}
```

65.5 2110. 股票平滑下跌阶段的数目 (1)

- 题目

```
给你一个整数数组prices，表示一支股票的历史每日股价，其中prices[i]是这支股票第i天的价格。
一个 平滑下降的阶段定义为：对于连续一天或者多天，每日股价都比 前一日股价恰好少␣
↪1，这个阶段第一天的股价没有限制。
请你返回 平滑下降阶段的数目。
示例 1：输入：prices = [3,2,1,4] 输出：7
解释：总共有 7 个平滑下降阶段：
[3], [2], [1], [4], [3,2], [2,1] 和 [3,2,1]
注意，仅一天按照定义也是平滑下降阶段。
示例 2：输入：prices = [8,6,7,7] 输出：4
解释：总共有 4 个连续平滑下降阶段：[8], [6], [7] 和 [7]
由于 8 - 6 ≠ 1 ，所以 [8,6] 不是平滑下降阶段。
示例 3：输入：prices = [1] 输出：1
解释：总共有 1 个平滑下降阶段：[1]
提示：1 <= prices.length <= 105
1 <= prices[i] <= 105
```

- 解题思路

```
func getDescentPeriods(prices []int) int64 {
        res := int64(1)
        if len(prices) == 1 {
                return 1
        }
```

(续下页)

(接上页)

```
        count := int64(1)
        for i := 1; i < len(prices); i++ {
                if prices[i-1]-prices[i] == 1 {
                        count++
                } else {
                        count = int64(1)
                }
                res = res + count
        }
        return res
}
```

65.6 2115. 从给定原材料中找到所有可以做出的菜 (1)

- 题目

```
你有 n道不同菜的信息。给你一个字符串数组recipes和一个二维字符串数组ingredients。
第i道菜的名字为recipes[i]，如果你有它所有的原材料ingredients[i]，
那么你可以做出这道菜。一道菜的原材料可能是另一道菜，也就是说ingredients[i]可能包含recipes中另一个字符
同时给你一个字符串数组supplies，它包含你初始时拥有的所有原材料，每一种原材料你都有无限多。
请你返回你可以做出的所有菜。你可以以 任意顺序返回它们。
注意两道菜在它们的原材料中可能互相包含。
示例 1：输入：recipes = ["bread"], ingredients = [["yeast","flour"]],
supplies = ["yeast","flour","corn"]输出：["bread"]
解释：我们可以做出 "bread" ，因为我们有原材料 "yeast" 和 "flour" 。
示例 2：输入：recipes = ["bread","sandwich"], ingredients = [["yeast","flour"],["bread
↪","meat"]],
supplies = ["yeast","flour","meat"] 输出：["bread","sandwich"]
解释：我们可以做出 "bread" ，因为我们有原材料 "yeast" 和 "flour" 。
我们可以做出 "sandwich" ，因为我们有原材料 "meat" 且可以做出原材料 "bread" 。
示例 3：输入：recipes = ["bread","sandwich","burger"],
ingredients = [["yeast","flour"],["bread","meat"],["sandwich","meat","bread"]],
supplies = ["yeast","flour","meat"] 输出：["bread","sandwich","burger"]
解释：我们可以做出 "bread" ，因为我们有原材料 "yeast" 和 "flour" 。
我们可以做出 "sandwich" ，因为我们有原材料 "meat" 且可以做出原材料 "bread" 。
我们可以做出 "burger" ，因为我们有原材料 "meat" 且可以做出原材料 "bread" 和 "sandwich
↪" 。
示例 4：输入：recipes = ["bread"], ingredients = [["yeast","flour"]], supplies = [
↪"yeast"] 输出：[]
解释：我们没法做出任何菜，因为我们只有原材料 "yeast" 。
提示：n == recipes.length == ingredients.length
1 <= n <= 100
```

(续下页)

(接上页)

```
1 <= ingredients[i].length, supplies.length <= 100
1 <= recipes[i].length, ingredients[i][j].length, supplies[k].length <= 10
recipes[i], ingredients[i][j]和supplies[k]只包含小写英文字母。
所有recipes 和supplies中的值互不相同。
ingredients[i]中的字符串互不相同。
```

- 解题思路

```
func findAllRecipes(recipes []string, ingredients [][]string, supplies []string)
↪[]string {
        res := make([]string, 0)
        arr := make(map[string][]string)
        inDegree := make(map[string]int)
        for i := 0; i < len(recipes); i++ {
                a := recipes[i]
                for j := 0; j < len(ingredients[i]); j++ {
                        b := ingredients[i][j] // b=>a 原材料=>菜
                        arr[b] = append(arr[b], a)
                        inDegree[a]++ // 菜的入度+1
                }
        }
        for len(supplies) > 0 {
                b := supplies[0]
                supplies = supplies[1:]
                for i := 0; i < len(arr[b]); i++ {
                        a := arr[b][i]
                        inDegree[a]--
                        if inDegree[a] == 0 { // 入度=0, 代表该菜的原材料都有
                                res = append(res, a)
                                supplies = append(supplies, a)
                        }
                }
        }
        return res
}
```

65.7 2116. 判断一个括号字符串是否有效 (1)

- 题目

```
一个括号字符串是只由'(' 和')'组成的非空字符串。如果一个字符串满足下面␣
↪任意一个条件，那么它就是有效的：
字符串为().
它可以表示为 AB（A与B连接），其中A 和B都是有效括号字符串。
它可以表示为(A)，其中A是一个有效括号字符串。
给你一个括号字符串s和一个字符串locked，两者长度都为n。locked是一个二进制字符串，只包含
↪'0'和'1'。对于locked中每一个下标i ：
如果locked[i]是'1'，你 不能改变s[i]。
如果locked[i]是'0'，你可以将s[i]变为'('或者')'。
如果你可以将 s变为有效括号字符串，请你返回true，否则返回false。
示例 1：输入：s = "))()))", locked = "010100" 输出：true
解释：locked[1] == '1' 和 locked[3] == '1' ，所以我们无法改变 s[1] 或者 s[3] 。
我们可以将 s[0] 和 s[4] 变为 '(' ，不改变 s[2] 和 s[5] ，使 s 变为有效字符串。
示例 2：输入：s = "()()", locked = "0000" 输出：true
解释：我们不需要做任何改变，因为 s 已经是有效字符串了。
示例 3：输入：s = ")", locked = "0" 输出：false
解释：locked 允许改变 s[0] 。
但无论将 s[0] 变为 '(' 或者 ')' 都无法使 s 变为有效字符串。
提示：n == s.length == locked.length
1 <= n <= 105
s[i]要么是'('要么是')'。
locked[i] 要么是'0'要么是'1' 。
```

- 解题思路

```
func canBeValid(s string, locked string) bool {
        if len(s)%2 == 1 {
                return false
        }
        var arr []byte
        for i := 0; i < len(locked); i++ {
                if locked[i] == '1' {
                        arr = append(arr, s[i])
                } else {
                        arr = append(arr, '*')
                }
        }
        return checkValidString(string(arr))
}
```

(续下页)

(接上页)

```
// leetcode678.有效的括号字符串
func checkValidString(s string) bool {
        // 第1次把星号当左括号看
        left, right := 0, 0
        for i := 0; i < len(s); i++ {
                if s[i] == ')' {
                        right++
                } else {
                        left++
                }
                if right > left {
                        return false
                }
        }
        // 第2次把星号当右括号看
        left, right = 0, 0
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '(' {
                        left++
                } else {
                        right++
                }
                if left > right {
                        return false
                }
        }
        return true
}
```

65.8 2120. 执行所有后缀指令 (1)

- 题目

```
现有一个 n x n 大小的网格，左上角单元格坐标 (0, 0) ，右下角单元格坐标 (n - 1, n - 1)␣
↪。
给你整数 n 和一个整数数组 startPos ，
其中 startPos = [startrow, startcol] 表示机器人最开始在坐标为 (startrow, startcol)␣
↪的单元格上。
另给你一个长度为 m 、下标从 0 开始的字符串 s ，其中 s[i] 是对机器人的第 i 条指令：
'L'（向左移动），'R'（向右移动），'U'（向上移动）和 'D'（向下移动）。
机器人可以从 s 中的任一第 i 条指令开始执行。它将会逐条执行指令直到 s␣
↪的末尾，但在满足下述条件之一时，机器人将会停止：
```

(续下页)

(接上页)

```
下一条指令将会导致机器人移动到网格外。
没有指令可以执行。
返回一个长度为 m 的数组 answer ，其中 answer[i] 是机器人从第 i条指令 开始，可以执行的
↪指令数目 。
示例 1：输入：n = 3, startPos = [0,1], s = "RRDDLU" 输出：[1,5,4,3,1,0]
解释：机器人从 startPos 出发，并从第 i 条指令开始执行：
- 0: "RRDDLU" 在移动到网格外之前，只能执行一条 "R" 指令。
- 1:  "RDDLU" 可以执行全部五条指令，机器人仍在网格内，最终到达 (0, 0) 。
- 2:   "DDLU" 可以执行全部四条指令，机器人仍在网格内，最终到达 (0, 0) 。
- 3:    "DLU" 可以执行全部三条指令，机器人仍在网格内，最终到达 (0, 0) 。
- 4:     "LU" 在移动到网格外之前，只能执行一条 "L" 指令。
- 5:      "U" 如果向上移动，将会移动到网格外。
示例 2：输入：n = 2, startPos = [1,1], s = "LURD" 输出：[4,1,0,0]
解释：- 0: "LURD"
- 1:  "URD"
- 2:   "RD"
- 3:    "D"
示例 3：输入：n = 1, startPos = [0,0], s = "LRUD" 输出：[0,0,0,0]
解释：无论机器人从哪条指令开始执行，都会移动到网格外。
提示：m == s.length
1 <= n, m <= 500
startPos.length == 2
0 <= startrow, startcol < n
s 由 'L'、'R'、'U' 和 'D' 组成
```

- 解题思路

```
func executeInstructions(n int, startPos []int, s string) []int {
        res := make([]int, 0)
        for i := 0; i < len(s); i++ {
                count := 0
                a, b := startPos[0], startPos[1]
                for j := i; j < len(s); j++ {
                        switch s[j] {
                        case 'L':
                                b--
                        case 'R':
                                b++
                        case 'U':
                                a--
                        case 'D':
                                a++
                        }
                        if 0 <= a && a < n && 0 <= b && b < n {
```

(续下页)

(接上页)

```
                                count++
                        } else {
                                break
                        }
                }
                res = append(res, count)
        }
        return res
}
```

65.9 2121. 相同元素的间隔之和 (2)

- 题目

```
给你一个下标从 0 开始、由 n 个整数组成的数组 arr 。
arr 中两个元素的 间隔 定义为它们下标之间的 绝对差 。更正式地，arr[i] 和 arr[j]␣
↪之间的间隔是 |i - j| 。
返回一个长度为 n 的数组intervals ，
其中 intervals[i] 是 arr[i] 和 arr 中每个相同元素（与 arr[i] 的值相同）的 间隔之和 。
注意：|x| 是 x 的绝对值。
示例 1：输入：arr = [2,1,3,1,2,3,3] 输出：[4,2,7,2,4,4,5]
解释：- 下标 0 ：另一个 2 在下标 4 ，|0 - 4| = 4
- 下标 1 ：另一个 1 在下标 3 ，|1 - 3| = 2
- 下标 2 ：另两个 3 在下标 5 和 6 ，|2 - 5| + |2 - 6| = 7
- 下标 3 ：另一个 1 在下标 1 ，|3 - 1| = 2
- 下标 4 ：另一个 2 在下标 0 ，|4 - 0| = 4
- 下标 5 ：另两个 3 在下标 2 和 6 ，|5 - 2| + |5 - 6| = 4
- 下标 6 ：另两个 3 在下标 2 和 5 ，|6 - 2| + |6 - 5| = 5
示例 2：输入：arr = [10,5,10,10] 输出：[5,0,3,4]
解释：
- 下标 0 ：另两个 10 在下标 2 和 3 ，|0 - 2| + |0 - 3| = 5
- 下标 1 ：只有这一个 5 在数组中，所以到相同元素的间隔之和是 0
- 下标 2 ：另两个 10 在下标 0 和 3 ，|2 - 0| + |2 - 3| = 3
- 下标 3 ：另两个 10 在下标 0 和 2 ，|3 - 0| + |3 - 2| = 4
提示：n == arr.length
1 <= n <= 105
1 <= arr[i] <= 105
```

- 解题思路

```
func getDistances(arr []int) []int64 {
        n := len(arr)
```

(续下页)

(接上页)

```
        res := make([]int64, n)
        m := make(map[int][]int) // 对相同数字进行分组
        for i := 0; i < n; i++ {
                m[arr[i]] = append(m[arr[i]], i)
        }
        for _, v := range m { // 分组
                arr := getSumAbsoluteDifferences(v)
                for i := 0; i < len(v); i++ {
                        res[v[i]] = int64(arr[i])
                }
        }
        return res
}

// leetcode1685.有序数组中差绝对值之和
func getSumAbsoluteDifferences(nums []int) []int {
        n := len(nums)
        res := make([]int, 0)
        arr := make([]int, 0)
        sum := 0
        for i := 0; i < n; i++ {
                sum = sum + nums[i]
                arr = append(arr, sum)
        }
        res = append(res, sum-n*nums[0])
        for i := 1; i < n; i++ {
                left := nums[i]*i - arr[i-1]              // 左边和
                right := (sum - arr[i]) - nums[i]*(n-1-i) // 右边和
                res = append(res, left+right)
        }
        return res
}

# 2
func getDistances(arr []int) []int64 {
        n := len(arr)
        res := make([]int64, n)
        m := make(map[int][]int) // 对相同数字进行分组
        for i := 0; i < n; i++ {
                m[arr[i]] = append(m[arr[i]], i)
        }
        for _, v := range m { // 分组
                arr := getSumAbsoluteDifferences(v)
```

(续下页)

(接上页)

```
                for i := 0; i < len(v); i++ {
                        res[v[i]] = int64(arr[i])
                }
        }
        return res
}

// leetcode1685.有序数组中差绝对值之和
func getSumAbsoluteDifferences(nums []int) []int {
        n := len(nums)
        res := make([]int, 0)
        right := 0 // 右边和
        left := 0  // 左边和
        for i := 1; i < n; i++ {
                right = right + (nums[i] - nums[0])
        }
        res = append(res, right)
        for i := 1; i < n; i++ {
                diff := nums[i] - nums[i-1]
                left = left + diff*i
                right = right - diff*(n-i)
                res = append(res, left+right)
        }
        return res
}
```

65.10 2125. 银行中的激光束数量 (2)

- 题目

```
银行内部的防盗安全装置已经激活。给你一个下标从 0 开始的二进制字符串数组 bank ，
表示银行的平面图，这是一个大小为 m x n 的二维矩阵。
bank[i] 表示第 i 行的设备分布，由若干 '0' 和若干 '1' 组成。'0' 表示单元格是空的，而 '1
↪' 表示单元格有一个安全设备。
对任意两个安全设备而言，如果同时 满足下面两个条件，则二者之间存在 一个 激光束：
两个设备位于两个 不同行 ：r1 和 r2 ，其中 r1 < r2 。
满足r1 < i < r2的 所有行i，都没有安全设备 。
激光束是独立的，也就是说，一个激光束既不会干扰另一个激光束，也不会与另一个激光束合并成一束。
返回银行中激光束的总数量。
示例 1：输入：bank = ["011001","000000","010100","001000"] 输出：8
解释：在下面每组设备对之间，存在一条激光束。总共是 8 条激光束：
 * bank[0][1] -- bank[2][1]
```

(续下页)

(接上页)

```
 * bank[0][1] -- bank[2][3]
 * bank[0][2] -- bank[2][1]
 * bank[0][2] -- bank[2][3]
 * bank[0][5] -- bank[2][1]
 * bank[0][5] -- bank[2][3]
 * bank[2][1] -- bank[3][2]
 * bank[2][3] -- bank[3][2]
注意，第 0 行和第 3 行上的设备之间不存在激光束。
这是因为第 2 行存在安全设备，这不满足第 2 个条件。
示例 2：输入：bank = ["000","111","000"] 输出：0
解释：不存在两个位于不同行的设备
提示：m == bank.length
n == bank[i].length
1 <= m, n <= 500
bank[i][j] 为 '0' 或 '1'
```

- 解题思路

```
func numberOfBeams(bank []string) int {
        res := 0
        prev := 0
        n, m := len(bank), len(bank[0])
        for i := 0; i < n; i++ {
                cur := 0
                for j := 0; j < m; j++ {
                        if bank[i][j] == '1' {
                                cur++
                        }
                }
                if cur == 0 {
                        continue
                }
                res = res + cur*prev
                prev = cur
        }
        return res
}

# 2
func numberOfBeams(bank []string) int {
        res := 0
        prev := 0
        for i := 0; i < len(bank); i++ {
                cur := strings.Count(bank[i], "1")
```

(续下页)

(接上页)

```
                if cur > 0 {
                        res = res + cur*prev
                        prev = cur
                }
        }
        return res
}
```

65.11 2126. 摧毁小行星 (1)

- 题目

```
给你一个整数mass，它表示一颗行星的初始质量。再给你一个整数数组asteroids，其中asteroids[i]是第i颗小行
你可以按 任意顺序重新安排小行星的顺序，然后让行星跟它们发生碰撞。
如果行星碰撞时的质量 大于等于小行星的质量，那么小行星被 摧毁，并且行星会␣
↪获得这颗小行星的质量。否则，行星将被摧毁。
如果所有小行星 都能被摧毁，请返回 true，否则返回 false。
示例 1：输入：mass = 10, asteroids = [3,9,19,5,21] 输出：true
解释：一种安排小行星的方式为 [9,19,5,3,21] ：
- 行星与质量为 9 的小行星碰撞。新的行星质量为：10 + 9 = 19
- 行星与质量为 19 的小行星碰撞。新的行星质量为：19 + 19 = 38
- 行星与质量为 5 的小行星碰撞。新的行星质量为：38 + 5 = 43
- 行星与质量为 3 的小行星碰撞。新的行星质量为：43 + 3 = 46
- 行星与质量为 21 的小行星碰撞。新的行星质量为：46 + 21 = 67
所有小行星都被摧毁。
示例 2：输入：mass = 5, asteroids = [4,9,23,4] 输出：false
解释：行星无论如何没法获得足够质量去摧毁质量为 23 的小行星。
行星把别的小行星摧毁后，质量为 5 + 4 + 9 + 4 = 22 。
它比 23 小，所以无法摧毁最后一颗小行星。
提示：1 <= mass <= 105
1 <= asteroids.length <= 105
1 <= asteroids[i] <= 105
```

- 解题思路

```
func asteroidsDestroyed(mass int, asteroids []int) bool {
        sort.Ints(asteroids)
        temp := int64(mass)
        for i := 0; i < len(asteroids); i++ {
                if int64(asteroids[i]) < temp {
                        return false
                }
```

(续下页)

(接上页)

```
                temp = temp + int64(asteroids[i])
        }
        return true
}
```

65.12 2130. 链表最大孪生和 (2)

- 题目

```
在一个大小为n且 n为偶数 的链表中，对于0 <= i <= (n / 2) - 1的 i，
第i个节点（下标从 0开始）的孪生节点为第(n-1-i)个节点 。
比方说，n = 4那么节点0是节点 3的孪生节点，节点 1是节点 2的孪生节点。这是长度为 n =␣
↪4的链表中所有的孪生节点。
孪生和定义为一个节点和它孪生节点两者值之和。
给你一个长度为偶数的链表的头节点head，请你返回链表的 最大孪生和。
示例1：输入：head = [5,4,2,1] 输出：6
解释：节点 0 和节点 1 分别是节点 3 和 2 的孪生节点。孪生和都为 6 。
链表中没有其他孪生节点。
所以，链表的最大孪生和是 6 。
示例 2：输入：head = [4,2,2,3] 出：7
解释：链表中的孪生节点为：
- 节点 0 是节点 3 的孪生节点，孪生和为 4 + 3 = 7 。
- 节点 1 是节点 2 的孪生节点，孪生和为 2 + 2 = 4 。
所以，最大孪生和为 max(7, 4) = 7 。
示例 3：输入：head = [1,100000] 出：100001
解释：链表中只有一对孪生节点，孪生和为 1 + 100000 = 100001 。
提示：链表的节点数目是[2, 105]中的偶数。
1 <= Node.val <= 105
```

- 解题思路

```
func pairSum(head *ListNode) int {
        arr := make([]int, 0)
        for head != nil {
                arr = append(arr, head.Val)
                head = head.Next
        }
        res := 0
        for i := 0; i < len(arr)/2; i++ {
                res = max(res, arr[i]+arr[len(arr)-1-i])
        }
        return res
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func pairSum(head *ListNode) int {
        res := 0
        slow, fast := head, head.Next
        for fast.Next != nil {
                slow = slow.Next
                fast = fast.Next.Next
        }
        slow = reverseList(slow.Next)
        fast = head
        for slow != nil {
                res = max(res, fast.Val+slow.Val)
                fast = fast.Next
                slow = slow.Next
        }
        return res
}

func reverseList(head *ListNode) *ListNode {
        var result *ListNode
        var temp *ListNode
        for head != nil {
                temp = head.Next
                head.Next = result
                result = head
                head = temp
        }
        return result
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

65.13 2131. 连接两字母单词得到的最长回文串 (1)

- 题目

```
给你一个字符串数组words。words中每个元素都是一个包含 两个小写英文字母的单词。
请你从 words中选择一些元素并按 任意顺序连接它们，并得到一个␣
↪尽可能长的回文串。每个元素 至多只能使用一次。
请你返回你能得到的最长回文串的 长度。如果没办法得到任何一个回文串，请你返回 0。
回文串指的是从前往后和从后往前读一样的字符串。
示例 1：输入：words = ["lc","cl","gg"] 输出：6
解释：一个最长的回文串为 "lc" + "gg" + "cl" = "lcggcl" ，长度为 6 。
"clgglc" 是另一个可以得到的最长回文串。
示例 2：输入：words = ["ab","ty","yt","lc","cl","ab"] 输出：8
解释：最长回文串是 "ty" + "lc" + "cl" + "yt" = "tylcclyt" ，长度为 8 。
"lcyttycl" 是另一个可以得到的最长回文串。
示例 3：输入：words = ["cc","ll","xx"]
输出：2 解释：最长回文串是 "cc" ，长度为 2 。
"ll" 是另一个可以得到的最长回文串。"xx" 也是。
提示：1 <= words.length <= 105
words[i].length == 2
words[i]仅包含小写英文字母。
```

- 解题思路

```
func longestPalindrome(words []string) int {
        res := 0
        m := make(map[string]int)
        for i := 0; i < len(words); i++ {
                m[words[i]]++
        }
        // 相同：偶数次数都可以用上；多个奇数出现次数的只能用1个奇数次数
        // 不同：取反后取最小次数
        mid := false
        for k, v := range m {
                str := string(k[1]) + string(k[0])
                if k == str { // 相同
                        if v%2 == 1 { // 奇数个aa可以拿1个放在中间
                                mid = true
                        }
```

(续下页)

(接上页)

```
                        res = res + 2*(v/2*2)
                } else {
                        res = res + 2*min(v, m[str]) // 最少的1方
                }
        }
        if mid == true { // 有中间的+2
                res = res + 2
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

65.14 2134. 最少交换次数来组合所有的 1II(2)

- 题目

```
交换 定义为选中一个数组中的两个 互不相同 的位置并交换二者的值。
环形 数组是一个数组，可以认为 第一个 元素和 最后一个 元素 相邻 。
给你一个 二进制环形 数组 nums ，返回在 任意位置 将数组中的所有 1␣
↪聚集在一起需要的最少交换次数。
示例 1：输入：nums = [0,1,0,1,1,0,0] 输出：1
解释：这里列出一些能够将所有 1 聚集在一起的方案：
[0,0,1,1,1,0,0] 交换 1 次。
[0,1,1,1,0,0,0] 交换 1 次。
[1,1,0,0,0,0,1] 交换 2 次（利用数组的环形特性）。
无法在交换 0 次的情况下将数组中的所有 1 聚集在一起。
因此，需要的最少交换次数为 1 。
示例 2：输入：nums = [0,1,1,1,0,0,1,1,0] 输出：2
解释：这里列出一些能够将所有 1 聚集在一起的方案：
[1,1,1,0,0,0,0,1,1] 交换 2 次（利用数组的环形特性）。
[1,1,1,1,1,0,0,0,0] 交换 2 次。
无法在交换 0 次或 1 次的情况下将数组中的所有 1 聚集在一起。
因此，需要的最少交换次数为 2 。
示例 3：输入：nums = [1,1,0,0,1] 输出：0
解释：得益于数组的环形特性，所有的 1 已经聚集在一起。
因此，需要的最少交换次数为 0 。
```

(续下页)

(接上页)

```
提示：1 <= nums.length <= 105
nums[i] 为 0 或者 1
```

- 解题思路

```
func minSwaps(nums []int) int {
        n := len(nums)
        count := 0 // 1的个数=>滑动窗口的大小
        for i := 0; i < n; i++ {
                if nums[i] == 1 {
                        count++
                }
        }
        res := 0 // 枚举所有起点，统计滑动窗口内0的个数
        for i := 0; i < count; i++ {
                if nums[i] == 0 {
                        res++
                }
        }
        temp := res
        for i := 0; i < n-1; i++ {
                if nums[i] == 0 { // 左边
                        temp--
                }
                if nums[(i+count)%n] == 0 { // 右边
                        temp++
                }
                res = min(res, temp)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minSwaps(nums []int) int {
        n := len(nums)
        count := 0 // 1的个数=>滑动窗口的大小
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                if nums[i] == 1 {
                        count++
                }
        }
        res := n // 枚举所有起点，统计滑动窗口内0的个数
        nums = append(nums, nums...)
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + (1 - nums[i])
                if i >= count {
                        sum = sum - (1 - nums[i-count])
                        res = min(res, sum)
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

65.15 2135. 统计追加字母可以获得的单词数 (2)

- 题目

```
给你两个下标从 0 开始的字符串数组 startWords 和 targetWords 。每个字符串都仅由␣
↪小写英文字母 组成。
对于 targetWords 中的每个字符串，检查是否能够从 startWords 中选出一个字符串，执行一次␣
↪转换操作，
得到的结果与当前targetWords 字符串相等。
转换操作 如下面两步所述：
追加 任何 不存在 于当前字符串的任一小写字母到当前字符串的末尾。
例如，如果字符串为 "abc" ，那么字母 'd'、'e' 或 'y' 都可以加到该字符串末尾，但 'a'␣
↪就不行。
如果追加的是 'd' ，那么结果字符串为 "abcd" 。
重排 新字符串中的字母，可以按 任意 顺序重新排布字母。
例如，"abcd" 可以重排为 "acbd"、"bacd"、"cbda"，以此类推。注意，它也可以重排为 "abcd"␣
↪自身。
找出 targetWords 中有多少字符串能够由startWords 中的 任一 字符串执行上述转换操作获得。
```

(续下页)

(接上页)

```
返回 targetWords 中这类 字符串的数目 。
注意：你仅能验证 targetWords 中的字符串是否可以由 startWords␣
↪中的某个字符串经执行操作获得。
startWords 中的字符串在这一过程中 不 发生实际变更。
示例 1：输入：startWords = ["ant","act","tack"], targetWords = ["tack","act","acti"]␣
↪输出：2
解释：
- 为了形成 targetWords[0] = "tack" ，可以选用 startWords[1] = "act" ，追加字母 'k'␣
↪，并重排 "actk" 为 "tack" 。
- startWords 中不存在可以用于获得 targetWords[1] = "act" 的字符串。
  注意 "act" 确实存在于 startWords ，但是 必须 在重排前给这个字符串追加一个字母。
- 为了形成 targetWords[2] = "acti" ，可以选用 startWords[1] = "act" ，追加字母 'i'␣
↪，并重排 "acti" 为 "acti" 自身。
示例 2：输入：startWords = ["ab","a"], targetWords = ["abc","abcd"] 输出：1
解释：- 为了形成 targetWords[0] = "abc" ，可以选用 startWords[0] = "ab" ，追加字母 'c
↪' ，并重排为 "abc" 。
- startWords 中不存在可以用于获得 targetWords[1] = "abcd" 的字符串。
提示：1 <= startWords.length, targetWords.length <= 5 * 104
1 <= startWords[i].length, targetWords[j].length <= 26
startWords 和 targetWords 中的每个字符串都仅由小写英文字母组成
在 startWords 或 targetWords 的任一字符串中，每个字母至多出现一次
```

- 解题思路

```
func wordCount(startWords []string, targetWords []string) int {
        m := make(map[int]bool)
        for i := 0; i < len(startWords); i++ {
                status := getStatus(startWords[i])
                for j := 0; j < 26; j++ { // 枚举所有操作后的状态
                        if (status>>j)&1 == 0 {
                                m[status|(1<<j)] = true
                        }
                }
        }
        res := 0
        for i := 0; i < len(targetWords); i++ {
                if m[getStatus(targetWords[i])] == true {
                        res++
                }
        }
        return res
}

func getStatus(str string) int {
```

(续下页)

(接上页)

```
        status := 0
        for i := 0; i < len(str); i++ {
                v := int(str[i] - 'a')
                status = status | (1 << v)
        }
        return status
}

# 2
func wordCount(startWords []string, targetWords []string) int {
        m := make(map[int]bool)
        for i := 0; i < len(startWords); i++ {
                status := getStatus(startWords[i])
                m[status] = true
        }
        res := 0
        for i := 0; i < len(targetWords); i++ {
                status := getStatus(targetWords[i])
                for j := 0; j < len(targetWords[i]); j++ {
                        v := int(targetWords[i][j] - 'a')
                        if m[status^(1<<v)] == true { // 去除字符
                                res++
                                break
                        }
                }
        }
        return res
}

func getStatus(str string) int {
        status := 0
        for i := 0; i < len(str); i++ {
                v := int(str[i] - 'a')
                status = status | (1 << v)
        }
        return status
}
```

65.16 2139. 得到目标值的最少行动次数 (1)

- 题目

```
你正在玩一个整数游戏。从整数 1 开始，期望得到整数 target 。
在一次行动中，你可以做下述两种操作之一：
递增，将当前整数的值加 1（即， x = x + 1）。
加倍，使当前整数的值翻倍（即，x = 2 * x）。
在整个游戏过程中，你可以使用 递增 操作 任意 次数。但是只能使用 加倍 操作 至多␣
↪maxDoubles 次。
给你两个整数 target 和 maxDoubles ，返回从 1 开始得到 target 需要的最少行动次数。
示例 1：输入：target = 5, maxDoubles = 0 输出：4
解释：一直递增 1 直到得到 target 。
示例 2：输入：target = 19, maxDoubles = 2 输出：7
解释：最初，x = 1 。
递增 3 次，x = 4 。
加倍 1 次，x = 8 。
递增 1 次，x = 9 。
加倍 1 次，x = 18 。
递增 1 次，x = 19 。
示例 3：输入：target = 10, maxDoubles = 4 输出：4
解释：最初，x = 1 。
递增 1 次，x = 2 。
加倍 1 次，x = 4 。
递增 1 次，x = 5 。
加倍 1 次，x = 10 。
提示：1 <= target <= 109
0 <= maxDoubles <= 100
```

- 解题思路

```
func minMoves(target int, maxDoubles int) int {
        res := 0
        // 方向操作：从大到小，优先减半
        for maxDoubles > 0 && target != 1 {
                res++
                if target%2 == 1 { // 奇数减去1
                        target--
                } else { // 偶数/2
                        maxDoubles--
                        target = target / 2
                }
        }
        res = res + (target - 1)
```

(续下页)

(接上页)

```
        return res
}
```

65.17 2140. 解决智力问题 (2)

- 题目

```
给你一个下标从 0开始的二维整数数组questions，其中questions[i] = [pointsi,␣
↪brainpoweri]。
这个数组表示一场考试里的一系列题目，你需要 按顺序（也就是从问题␣
↪0开始依次解决），针对每个问题选择 解决或者 跳过操作。
解决问题 i将让你 获得pointsi的分数，但是你将 无法解决接下来的brainpoweri个问题
（即只能跳过接下来的␣
↪brainpoweri个问题）。如果你跳过问题i，你可以对下一个问题决定使用哪种操作。
比方说，给你questions = [[3, 2], [4, 3], [4, 4], [2, 5]]：
如果问题0被解决了， 那么你可以获得3分，但你不能解决问题1 和2。
如果你跳过问题0，且解决问题1，你将获得 4 分但是不能解决问题2 和3。
请你返回这场考试里你能获得的 最高分数。
示例 1：输入：questions = [[3,2],[4,3],[4,4],[2,5]] 输出：5
解释：解决问题 0 和 3 得到最高分。
- 解决问题 0 ：获得 3 分，但接下来 2 个问题都不能解决。
- 不能解决问题 1 和 2
- 解决问题 3 ：获得 2 分
总得分为：3 + 2 = 5 。没有别的办法获得 5 分或者多于 5 分。
示例 2：输入：questions = [[1,1],[2,2],[3,3],[4,4],[5,5]] 输出：7
解释：解决问题 1 和 4 得到最高分。
- 跳过问题 0
- 解决问题 1 ：获得 2 分，但接下来 2 个问题都不能解决。
- 不能解决问题 2 和 3
- 解决问题 4 ：获得 5 分
总得分为：2 + 5 = 7 。没有别的办法获得 7 分或者多于 7 分。
提示：1 <= questions.length <= 105
questions[i].length == 2
1 <= pointsi, brainpoweri <= 105
```

- 解题思路

```
func mostPoints(questions [][]int) int64 {
        n := len(questions)
        dp := make([]int, n+1) // dp[i] => 解决第i道题后的最高分数
        // 反向
        for i := n - 1; i >= 0; i-- {
```

(续下页)

(接上页)

```
                next := min(n, i+questions[i][1]+1)
                dp[i] = max(dp[i+1], dp[next]+questions[i][0])
        }
        return int64(dp[0])
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func mostPoints(questions [][]int) int64 {
        n := len(questions)
        dp := make([]int, n+1) // dp[i] => 解决第i道题后的最高分数
        for i := 0; i < n; i++ {
                next := min(n, i+questions[i][1]+1)
                dp[i+1] = max(dp[i+1], dp[i])                    // 跳过
                dp[next] = max(dp[next], dp[i]+questions[i][0]) // 不跳过
        }
        return int64(dp[n])
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

65.18 2145. 统计隐藏数组数目 (1)

- 题目

```
给你一个下标从 0开始且长度为 n的整数数组differences，它表示一个长度为n +␣
↪1的隐藏数组相邻元素之间的差值。
更正式的表述为：我们将隐藏数组记作hidden，那么differences[i] = hidden[i + 1] -␣
↪hidden[i]。
同时给你两个整数lower 和upper，它们表示隐藏数组中所有数字的值都在 闭区间[lower,␣
↪upper]之间。
比方说，differences = [1, -3, 4]，lower = 1，upper = 6，
那么隐藏数组是一个长度为 4且所有值都在1和6（包含两者）之间的数组。
[3, 4, 1, 5] 和[4, 5, 2, 6]都是符合要求的隐藏数组。
[5, 6, 3, 7]不符合要求，因为它包含大于 6的元素。
[1, 2, 3, 4]不符合要求，因为相邻元素的差值不符合给定数据。
请你返回 符合要求的隐藏数组的数目。如果没有符合要求的隐藏数组，请返回 0。
示例 1：输入：differences = [1,-3,4]，lower = 1，upper = 6 输出：2
解释：符合要求的隐藏数组为：
- [3, 4, 1, 5]
- [4, 5, 2, 6]
所以返回 2 。
示例 2：输入：differences = [3,-4,5,1,-2]，lower = -4，upper = 5 输出：4
解释：符合要求的隐藏数组为：
- [-3, 0, -4, 1, 2, 0]
- [-2, 1, -3, 2, 3, 1]
- [-1, 2, -2, 3, 4, 2]
- [0, 3, -1, 4, 5, 3]
所以返回 4 。
示例 3：输入：differences = [4,-7,2]，lower = 3，upper = 6 输出：0
解释：没有符合要求的隐藏数组，所以返回 0 。
提示：n == differences.length
1 <= n <= 105
-105 <= differences[i] <= 105
-105 <= lower <= upper <= 105
```

- 解题思路

```
func numberOfArrays(differences []int, lower int, upper int) int {
        sum := 0
```

(续下页)

(接上页)

```
        maxValue, minValue := 0, 0
        for i := 0; i < len(differences); i++ {
                sum = sum + differences[i]
                maxValue = max(maxValue, sum)
                minValue = min(minValue, sum)
        }
        return max(0, upper-lower+1-(maxValue-minValue))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

65.19 2146. 价格范围内最高排名的 K 样物品 (2)

- 题目

```
给你一个下标从 0开始的二维整数数组grid, 它的大小为m x␣
↪n, 表示一个商店中物品的分布图。数组中的整数含义为:
0表示无法穿越的一堵墙。
1表示可以自由通过的一个空格子。
所有其他正整数表示该格子内的一样物品的价格。你可以自由经过这些格子。
从一个格子走到上下左右相邻格子花费1步。
同时给你一个整数数组pricing 和start, 其中pricing = [low, high] 且start = [row, col],
表示你开始位置为(row, col), 同时你只对物品价格在闭区间[low,␣
↪high]之内的物品感兴趣。同时给你一个整数k。
你想知道给定范围 内且 排名最高的 k件物品的␣
↪位置。排名按照优先级从高到低的以下规则制定:
距离: 定义为从start到一件物品的最短路径需要的步数 (较近距离的排名更高) 。
价格: 较低价格的物品有更高优先级, 但只考虑在给定范围之内的价格。
行坐标: 较小行坐标的有更高优先级。
列坐标: 较小列坐标的有更高优先级。
```

(续下页)

(接上页)

请你返回给定价格内排名最高的 k件物品的坐标，将它们按照排名排序后返回。
如果给定价格内少于 k件物品，那么请将它们的坐标全部返回。
示例 1：输入：grid = [[1,2,0,1],[1,3,0,1],[0,2,5,1]], pricing = [2,5], start = [0,0], k = 3
输出：[[0,1],[1,1],[2,1]]
解释：起点为 (0,0) 。
价格范围为 [2,5] ，我们可以选择的物品坐标为 (0,1)，(1,1)，(2,1) 和 (2,2) 。
这些物品的排名为：
- (0,1) 距离为 1
- (1,1) 距离为 2
- (2,1) 距离为 3
- (2,2) 距离为 4
所以，给定价格范围内排名最高的 3 件物品的坐标为 (0,1)，(1,1) 和 (2,1) 。
示例 2：输入：grid = [[1,2,0,1],[1,3,3,1],[0,2,5,1]], pricing = [2,3], start = [2,3], k = 2
输出：[[2,1],[1,2]]
解释：起点为 (2,3) 。
价格范围为 [2,3] ，我们可以选择的物品坐标为 (0,1)，(1,1)，(1,2) 和 (2,1) 。
这些物品的排名为：
- (2,1) 距离为 2 ，价格为 2
- (1,2) 距离为 2 ，价格为 3
- (1,1) 距离为 3
- (0,1) 距离为 4
所以，给定价格范围内排名最高的 2 件物品的坐标为 (2,1) 和 (1,2) 。
示例 3：输入：grid = [[1,1,1],[0,0,1],[2,3,4]], pricing = [2,3], start = [0,0], k = 3
输出：[[2,1],[2,0]]
解释：起点为 (0,0) 。
价格范围为 [2,3] ，我们可以选择的物品坐标为 (2,0) 和 (2,1) 。
这些物品的排名为：
- (2,1) 距离为 5
- (2,0) 距离为 6
所以，给定价格范围内排名最高的 2 件物品的坐标为 (2,1) 和 (2,0) 。
注意，k = 3 但给定价格范围内只有 2 件物品。
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 105
1 <= m * n <= 105
0 <= grid[i][j] <= 105
pricing.length == 2
2 <= low <= high <= 105
start.length == 2
0 <= row <= m - 1
0 <= col <= n - 1

(续下页)

(接上页)

```
grid[row][col] > 0
1 <= k <= m * n
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func highestRankedKItems(grid [][]int, pricing []int, start []int, k int) [][]int {
        n, m := len(grid), len(grid[0])
        arr := make([][]int, 0) // [dis, price, x, y]
        queue := make([][]int, 0)
        queue = append(queue, []int{start[0], start[1], 0})
↪         // [x,y,dis]
        if pricing[0] <= grid[start[0]][start[1]] && grid[start[0]][start[1]] <=
↪pricing[1] { // 起点
                arr = append(arr, []int{0, grid[start[0]][start[1]], start[0],
↪start[1]})
        }
        grid[start[0]][start[1]] = 0
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                x, y, dis := node[0], node[1], node[2]
                for i := 0; i < 4; i++ {
                        newX := x + dx[i]
                        newY := y + dy[i]
                        if 0 <= newX && newX < n && 0 <= newY && newY < m &&
↪grid[newX][newY] > 0 {
                                if pricing[0] <= grid[newX][newY] && grid[newX][newY]
↪<= pricing[1] {
                                        arr = append(arr, []int{dis + 1,
↪grid[newX][newY], newX, newY})
                                }
                                queue = append(queue, []int{newX, newY, dis + 1})
                                grid[newX][newY] = 0
                        }
                }
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][0] == arr[j][0] {
                        if arr[i][1] == arr[j][1] {
                                if arr[i][2] == arr[j][2] {
                                        return arr[i][3] < arr[j][3]
```

(续下页)

(接上页)

```
                                }
                                return arr[i][2] < arr[j][2]
                        }
                        return arr[i][1] < arr[j][1]
                }
                return arr[i][0] < arr[j][0]
        })
        res := make([][]int, 0)
        for i := 0; i < k && i < len(arr); i++ {
                res = append(res, []int{arr[i][2], arr[i][3]})
        }
        return res
}

# 2
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func highestRankedKItems(grid [][]int, pricing []int, start []int, k int) [][]int {
        n, m := len(grid), len(grid[0])
        arr := make([][]int, 0) // [dis, price, x, y]
        queue := make([][]int, 0)
        queue = append(queue, []int{start[0], start[1], 0})  // [x,y,dis]
        grid[start[0]][start[1]] = -grid[start[0]][start[1]] // 置反
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                x, y, dis := node[0], node[1], node[2]
                if pricing[0] <= -grid[x][y] && -grid[x][y] <= pricing[1] { //␣
↪起点判断
                        arr = append(arr, []int{dis, -grid[x][y], x, y})
                }
                for i := 0; i < 4; i++ {
                        newX := x + dx[i]
                        newY := y + dy[i]
                        if 0 <= newX && newX < n && 0 <= newY && newY < m &&␣
↪grid[newX][newY] > 0 {
                                queue = append(queue, []int{newX, newY, dis + 1})
                                grid[newX][newY] = -grid[newX][newY]
                        }
                }
        }
        sort.Slice(arr, func(i, j int) bool {
```

(续下页)

(接上页)

```
                if arr[i][0] == arr[j][0] {
                        if arr[i][1] == arr[j][1] {
                                if arr[i][2] == arr[j][2] {
                                        return arr[i][3] < arr[j][3]
                                }
                                return arr[i][2] < arr[j][2]
                        }
                        return arr[i][1] < arr[j][1]
                }
                return arr[i][0] < arr[j][0]
        })
        res := make([][]int, 0)
        for i := 0; i < k && i < len(arr); i++ {
                res = append(res, []int{arr[i][2], arr[i][3]})
        }
        return res
}
```

65.20 2149. 按符号重排数组 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，数组长度为 偶数␣
↪，由数目相等的正整数和负整数组成。
你需要 重排 nums 中的元素，使修改后的数组满足下述条件：
任意连续 的两个整数 符号相反
对于符号相同的所有整数，保留 它们在 nums 中的 顺序 。
重排后数组以正整数开头。
重排元素满足上述条件后，返回修改后的数组。
示例 1：输入：nums = [3,1,-2,-5,2,-4] 输出：[3,-2,1,-5,2,-4]
解释：nums 中的正整数是 [3,1,2] ，负整数是 [-2,-5,-4] 。
重排的唯一可行方案是 [3,-2,1,-5,2,-4]，能满足所有条件。
像 [1,-2,2,-5,3,-4]、[3,1,2,-2,-5,-4]、[-2,3,-5,1,-4,2]␣
↪这样的其他方案是不正确的，因为不满足一个或者多个条件。
示例 2：输入：nums = [-1,1] 输出：[1,-1]
解释：1 是 nums 中唯一一个正整数，-1 是 nums 中唯一一个负整数。
所以 nums 重排为 [1,-1] 。
提示：2 <= nums.length <= 2 * 105
nums.length 是 偶数
1 <= |nums[i]| <= 105
nums 由 相等 数量的正整数和负整数组成
```

- 解题思路

```
func rearrangeArray(nums []int) []int {
        res := make([]int, len(nums))
        i, j := 0, 1
        for k := 0; k < len(nums); k++ {
                if nums[k] > 0 {
                        res[i] = nums[k]
                        i = i + 2
                } else {
                        res[j] = nums[k]
                        j = j + 2
                }
        }
        return res
}
```

65.21 2150. 找出数组中的所有孤独数字 (2)

- 题目

```
给你一个整数数组 nums 。如果数字 x 在数组中仅出现 一次 ，
且没有 相邻 数字（即，x + 1 和 x - 1）出现在数组中，则认为数字 x 是 孤独数字 。
返回 nums 中的 所有 孤独数字。你可以按 任何顺序 返回答案。
示例 1：输入：nums = [10,6,5,8] 输出：[10,8]
解释：- 10 是一个孤独数字，因为它只出现一次，并且 9 和 11 没有在 nums 中出现。
- 8 是一个孤独数字，因为它只出现一次，并且 7 和 9 没有在 nums 中出现。
- 5 不是一个孤独数字，因为 6 出现在 nums 中，反之亦然。
因此，nums 中的孤独数字是 [10, 8] 。
注意，也可以返回 [8, 10] 。
示例 2：输入：nums = [1,3,5,3] 输出：[1,5]
解释：- 1 是一个孤独数字，因为它只出现一次，并且 0 和 2 没有在 nums 中出现。
- 5 是一个孤独数字，因为它只出现一次，并且 4 和 6 没有在 nums 中出现。
- 3 不是一个孤独数字，因为它出现两次。
因此，nums 中的孤独数字是 [1, 5] 。
注意，也可以返回 [5, 1] 。
提示：1 <= nums.length <= 105
0 <= nums[i] <= 106
```

- 解题思路

```
func findLonely(nums []int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                m[nums[i]]++
        }
        for k, v := range m {
                if v == 1 && m[k-1] == 0 && m[k+1] == 0 {
                        res = append(res, k)
                }
        }
        return res
}

# 2
func findLonely(nums []int) []int {
        res := make([]int, 0)
        nums = append(nums, math.MinInt32)
        nums = append(nums, math.MaxInt32)
        sort.Ints(nums)
        for i := 1; i < len(nums)-1; i++ {
                if nums[i]-nums[i-1] > 1 && nums[i+1]-nums[i] > 1 {
                        res = append(res, nums[i])
                }
        }
        return res
}
```

65.22 2155. 分组得分最高的所有下标 (2)

- 题目

```
给你一个下标从 0 开始的二进制数组 nums ，数组长度为 n 。
nums 可以按下标 i（ 0 <= i <= n ）拆分成两个数组（可能为空）：numsleft 和 numsright 。
numsleft 包含 nums 中从下标 0 到 i - 1 的所有元素（包括 0 和 i - 1 ），
而 numsright 包含 nums 中从下标 i 到 n - 1 的所有元素（包括 i 和 n - 1 ）。
如果 i == 0 ，numsleft 为 空 ，而 numsright 将包含 nums 中的所有元素。
如果 i == n ，numsleft 将包含 nums 中的所有元素，而 numsright 为 空 。
下标 i 的 分组得分 为 numsleft 中 0 的个数和 numsright 中 1 的个数之 和 。
返回 分组得分 最高 的 所有不同下标 。你可以按 任意顺序 返回答案。
示例 1：输入：nums = [0,0,1,0] 输出：[2,4]
解释：按下标分组
- 0 ：numsleft 为 [] 。numsright 为 [0,0,1,0] 。得分为 0 + 1 = 1 。
- 1 ：numsleft 为 [0] 。numsright 为 [0,1,0] 。得分为 1 + 1 = 2 。
- 2 ：numsleft 为 [0,0] 。numsright 为 [1,0] 。得分为 2 + 1 = 3 。
- 3 ：numsleft 为 [0,0,1] 。numsright 为 [0] 。得分为 2 + 0 = 2 。
```

(续下页)

(接上页)

```
- 4 ：numsleft 为 [0,0,1,0] 。numsright 为 [] 。得分为 3 + 0 = 3 。
下标 2 和 4 都可以得到最高的分组得分 3 。
注意，答案 [4,2] 也被视为正确答案。
示例 2：输入：nums = [0,0,0] 输出：[3]
解释：按下标分组
- 0 ：numsleft 为 [] 。numsright 为 [0,0,0] 。得分为 0 + 0 = 0 。
- 1 ：numsleft 为 [0] 。numsright 为 [0,0] 。得分为 1 + 0 = 1 。
- 2 ：numsleft 为 [0,0] 。numsright 为 [0] 。得分为 2 + 0 = 2 。
- 3 ：numsleft 为 [0,0,0] 。numsright 为 [] 。得分为 3 + 0 = 3 。
只有下标 3 可以得到最高的分组得分 3 。
示例 3：输入：nums = [1,1] 输出：[0]
解释：按下标分组
- 0 ：numsleft 为 [] 。numsright 为 [1,1] 。得分为 0 + 2 = 2 。
- 1 ：numsleft 为 [1] 。numsright 为 [1] 。得分为 0 + 1 = 1 。
- 2 ：numsleft 为 [1,1] 。numsright 为 [] 。得分为 0 + 0 = 0 。
只有下标 0 可以得到最高的分组得分 2 。
提示：n == nums.length
1 <= n <= 105
nums[i] 为 0 或 1
```

- 解题思路

```
func maxScoreIndices(nums []int) []int {
        res := []int{0}
        sum := 0
        maxValue := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        sum++
                } else {
                        sum--
                }
                if sum > maxValue {
                        maxValue = sum
                        res = []int{i + 1}
                } else if sum == maxValue {
                        res = append(res, i+1)
                }
        }
        return res
}

# 2
func maxScoreIndices(nums []int) []int {
```

(续下页)

(接上页)

```
        res := []int{0}
        left, right := 0, 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == 1 {
                        right++
                }
        }
        maxValue := left + right
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        left++
                } else {
                        right--
                }
                if left+right > maxValue {
                        maxValue = left + right
                        res = []int{i + 1}
                } else if left+right == maxValue {
                        res = append(res, i+1)
                }
        }
        return res
}
```

65.23 2156. 查找给定哈希值的子串 (1)

- 题目

```
给定整数 p和 m，一个长度为 k且下标从 0开始的字符串s的哈希值按照如下函数计算：
hash(s, p, m) = (val(s[0]) * p0 + val(s[1]) * p1 + ... + val(s[k-1]) * pk-1) mod m.
其中val(s[i])表示s[i]在字母表中的下标，从val('a') = 1 到val('z') = 26。
给你一个字符串s和整数power，modulo，k和hashValue。
请你返回 s中 第一个 长度为 k的 子串sub，满足hash(sub, power, modulo) == hashValue。
测试数据保证一定 存在至少一个这样的子串。
子串 定义为一个字符串中连续非空字符组成的序列。
示例 1：输入：s = "leetcode", power = 7, modulo = 20, k = 2, hashValue = 0 输出："ee"
解释："ee" 的哈希值为 hash("ee", 7, 20) = (5 * 1 + 5 * 7) mod 20 = 40 mod 20 = 0 。
"ee" 是长度为 2 的第一个哈希值为 0 的子串，所以我们返回 "ee" 。
示例 2：输入：s = "fbxzaad", power = 31, modulo = 100, k = 3, hashValue = 32 输出：
↪"fbx"
解释："fbx" 的哈希值为 hash("fbx", 31, 100) = (6 * 1 + 2 * 31 + 24 * 312) mod 100 =␣
↪23132 mod 100 = 32 。
```

(续下页)

(接上页)

```
"bxz" 的哈希值为 hash("bxz", 31, 100) = (2 * 1 + 24 * 31 + 26 * 312) mod 100 = 25732␣
↪mod 100 = 32 。
"fbx" 是长度为 3 的第一个哈希值为 32 的子串，所以我们返回 "fbx" 。
注意，"bxz" 的哈希值也为 32 ，但是它在字符串中比 "fbx" 更晚出现。
提示：1 <= k <= s.length <= 2 * 104
1 <= power, modulo <= 109
0 <= hashValue < modulo
s只包含小写英文字母。
测试数据保证一定 存在满足条件的子串。
```

- 解题思路

```
func subStrHash(s string, power int, modulo int, k int, hashValue int) string {
        n := len(s)
        p := 1   // power的k次方
        sum := 0 // 和
        // 第一个长度为k的结果
        for i := n - 1; i >= n-k; i-- {
                p = p * power % modulo
                a := int(s[i]-'a') + 1
                sum = (a + sum*power) % modulo
        }
        res := s[n-k:] // 题目保证至少有1个
        // 逆序计算：滑动窗口
        for i := n - k - 1; i >= 0; i-- {
                a := int(s[i]-'a') + 1 + sum*power       // k+1位
                b := (int(s[i+k]-'a') + 1) * p % modulo // s[i+k]*power^k
                // 模p运算：分配律
                // (a+b) % mod  = (a%mod + b%mod) % mod
                // (a-b) % mod  = (a%mod - b%mod ) % mod
                // (a-b) % mod  = (a%mod - b%mod + mod ) % mod：go有负数，+mod再mod
                sum = (a - b + modulo) % modulo
                if sum == hashValue {
                        res = s[i : i+k]
                }
        }
        return res
}
```

65.24 2161. 根据给定数字划分数组 (3)

- 题目

```
给你一个下标从␣
↪0开始的整数数组nums和一个整数pivot。请你将nums重新排列，使得以下条件均成立：
所有小于pivot的元素都出现在所有大于pivot的元素之前。
所有等于pivot的元素都出现在小于和大于 pivot的元素 中间。
小于 pivot的元素之间和大于 pivot的元素之间的 相对顺序不发生改变。
更正式的，考虑每一对pi，pj，pi是初始时位置␣
↪i元素的新位置，pj是初始时位置j元素的新位置。
对于小于pivot的元素，如果i < j且nums[i] < pivot 和nums[j] < pivot都成立，那么pi <␣
↪pj也成立。
类似的，对于大于pivot的元素，如果i < j 且nums[i] > pivot 和nums[j] >␣
↪pivot都成立，那么pi < pj。
请你返回重新排列 nums数组后的结果数组。
示例 1：输入：nums = [9,12,5,10,14,3,10], pivot = 10 输出：[9,5,3,10,10,12,14]
解释：元素 9 ，5 和 3 小于 pivot ，所以它们在数组的最左边。
元素 12 和 14 大于 pivot ，所以它们在数组的最右边。
小于 pivot 的元素的相对位置和大于 pivot 的元素的相对位置分别为 [9, 5, 3] 和 [12, 14]␣
↪,
它们在结果数组中的相对顺序需要保留。
示例 2：输入：nums = [-3,4,3,2], pivot = 2 输出：[-3,2,4,3]
解释：元素 -3 小于 pivot ，所以在数组的最左边。
元素 4 和 3 大于 pivot ，所以它们在数组的最右边。
小于 pivot 的元素的相对位置和大于 pivot 的元素的相对位置分别为 [-3] 和 [4, 3]␣
↪，它们在结果数组中的相对顺序需要保留。
提示：1 <= nums.length <= 105
-106 <= nums[i] <= 106
pivot等于nums中的一个元素。
```

- 解题思路

```
func pivotArray(nums []int, pivot int) []int {
        n := len(nums)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = pivot
        }
        left, right := 0, n-1
        for i := 0; i < n; i++ {
                if nums[i] < pivot {
                        res[left] = nums[i]
                        left++
```

(续下页)

(接上页)

```
                } else if nums[i] > pivot {
                        res[right] = nums[i]
                        right--
                }
        }
        reverse(res[right+1:]) // 反转后面顺序
        return res
}

func reverse(arr []int) {
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
}

# 2
func pivotArray(nums []int, pivot int) []int {
        n := len(nums)
        res := make([]int, 0)
        for i := 0; i < n; i++ {
                if nums[i] < pivot {
                        res = append(res, nums[i])
                }
        }
        for i := 0; i < n; i++ {
                if nums[i] == pivot {
                        res = append(res, nums[i])
                }
        }
        for i := 0; i < n; i++ {
                if nums[i] > pivot {
                        res = append(res, nums[i])
                }
        }
        return res
}

# 3
func pivotArray(nums []int, pivot int) []int {
        n := len(nums)
        a, b, c := make([]int, 0), make([]int, 0), make([]int, 0)
        for i := 0; i < n; i++ {
                if nums[i] < pivot {
```

(续下页)

(接上页)

```
                        a = append(a, nums[i])
                } else if nums[i] == pivot {
                        b = append(b, nums[i])
                } else {
                        c = append(c, nums[i])
                }
        }
        return append(a, append(b, c...)...)
}
```

65.25 2162. 设置时间的最少代价 (2)

- 题目

```
常见的微波炉可以设置加热时间，且加热时间满足以下条件：
至少为 1秒钟。
至多为99分99秒。
你可以 最多输入4 个数字来设置加热时间。如果你输入的位数不足 4 位，微波炉会自动加␣
↪前缀0来补足 4 位。
微波炉会将设置好的四位数中，前两位当作分钟数，后两位当作秒数。它们所表示的总时间就是加热时间。比方说
你输入9 5 4（三个数字），被自动补足为0954，并表示9分54秒。
你输入0 0 0 8（四个数字），表示0分8秒。
你输入8 0 9 0，表示80分90秒。
你输入8 1 3 0，表示81分30秒。
给你整数startAt，moveCost，pushCost和targetSeconds。
一开始，你的手指在数字startAt处。将手指移到任何其他数字，需要花费moveCost的单位代价。
每输入你手指所在位置的数字一次，需要花费pushCost的单位代价。
要设置targetSeconds秒的加热时间，可能会有多种设置方法。你想要知道这些方法中，总代价最小为多少。
请你能返回设置targetSeconds秒钟加热时间需要花费的最少代价。
请记住，虽然微波炉的秒数最多可以设置到 99秒，但一分钟等于60秒。
示例 1：输入：startAt = 1, moveCost = 2, pushCost = 1, targetSeconds = 600 输出：6
解释：以下为设置加热时间的所有方法。
- 1 0 0 0，表示 10 分 0 秒。
 手指一开始就在数字 1 处，输入 1 （代价为 1），移到 0 处（代价为 2），
 输入 0（代价为 1），输入 0（代价为 1），输入 0（代价为 1）。
 总代价为：1 + 2 + 1 + 1 + 1 = 6 。这是所有方案中的最小代价。
- 0 9 6 0，表示 9 分 60 秒。它也表示 600 秒。
 手指移到 0 处（代价为 2），输入 0 （代价为 1），移到 9 处（代价为 2），
 输入 9（代价为 1），移到 6 处（代价为 2），输入 6（代价为 1），移到 0 处（代价为␣
↪2），输入 0（代价为 1）。
 总代价为：2 + 1 + 2 + 1 + 2 + 1 + 2 + 1 = 12 。
- 9 6 0，微波炉自动补全为 0960 ，表示 9 分 60 秒。
```

(续下页)

(接上页)

```
 手指移到 9 处（代价为 2），输入 9 （代价为 1），移到 6 处（代价为 2），
 输入 6（代价为 1），移到 0 处（代价为 2），输入 0（代价为 1）。
 总代价为：2 + 1 + 2 + 1 + 2 + 1 = 9 。
示例 2：输入：startAt = 0, moveCost = 1, pushCost = 2, targetSeconds = 76 输出：6
解释：最优方案为输入两个数字 7 6，表示 76 秒。
手指移到 7 处（代价为 1），输入 7 （代价为 2），移到 6 处（代价为 1），输入 6（代价为
↪2）。
总代价为：1 + 2 + 1 + 2 = 6
其他可行方案为 0076 ，076 ，0116 和 116 ，但是它们的代价都比 6 大。
提示：0 <= startAt <= 9
1 <= moveCost, pushCost <= 105
1 <= targetSeconds <= 6039
```

- 解题思路

```
func minCostSetTime(startAt int, moveCost int, pushCost int, targetSeconds int) int {
        m, s := targetSeconds/60, targetSeconds%60
        a := cost(startAt, moveCost, pushCost, m, s)      // 目标1
        b := cost(startAt, moveCost, pushCost, m-1, s+60) // 目标2
        return min(a, b)
}

func cost(startAt int, moveCost int, pushCost int, m, s int) int {
        res := 0
        if m < 0 || m > 99 || s < 0 || s > 99 {
                return math.MaxInt32
        }
        arr := []int{m / 10, m % 10, s / 10, s % 10}
        i := 0 // 需要忽略0后的起始位置（前面0不需要输入）
        for i < 4 && arr[i] == 0 {
                i++
        }
        for j := i; j < 4; j++ { // 从i开始
                if startAt != arr[j] { // 不相同需要移动
                        startAt = arr[j]
                        res = res + moveCost // 移动
                }
                res = res + pushCost
        }

        return res
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}
```

65.26 2165. 重排数字的最小值 (1)

- 题目

```
给你一个整数 num 。重排 num 中的各位数字，使其值 最小化 且不含 任何 前导零。
返回不含前导零且值最小的重排数字。
注意，重排各位数字后，num 的符号不会改变。
示例 1：输入：num = 310 输出：103
解释：310 中各位数字的可行排列有：013、031、103、130、301、310 。
不含任何前导零且值最小的重排数字是 103 。
示例 2：输入：num = -7605 输出：-7650
解释：-7605 中各位数字的部分可行排列为：-7650、-6705、-5076、-0567。
不含任何前导零且值最小的重排数字是 -7650 。
提示：-1015 <= num <= 1015
```

- 解题思路

```
func smallestNumber(num int64) int64 {
        arr := []byte(strconv.FormatInt(num, 10))
        flag := int64(1)
        if num <= 0 {
                flag = -1
                arr = arr[1:]
        }
        if flag == 1 { // 正数
                sort.Slice(arr, func(i, j int) bool {
                        return arr[i] < arr[j]
                })
                i := 0
                for arr[i] == '0' {
                        i++
                }
                arr[0], arr[i] = arr[i], arr[0] // 交换第一个非0
        } else {
                sort.Slice(arr, func(i, j int) bool {
                        return arr[i] > arr[j]
```

(续下页)

(接上页)

```
                })
        }
        res, _ := strconv.ParseInt(string(arr), 10, 64)
        return flag * res
}
```

65.27 2166. 设计位集 (2)

- 题目

```
位集 Bitset 是一种能以紧凑形式存储位的数据结构。
请你实现 Bitset 类。
Bitset(int size) 用 size 个位初始化 Bitset ，所有位都是 0 。
void fix(int idx) 将下标为 idx 的位上的值更新为 1 。如果值已经是 1␣
↪，则不会发生任何改变。
void unfix(int idx) 将下标为 idx 的位上的值更新为 0 。如果值已经是 0␣
↪，则不会发生任何改变。
void flip() 翻转 Bitset 中每一位上的值。换句话说，所有值为 0 的位将会变成 1␣
↪，反之亦然。
boolean all() 检查Bitset 中 每一位 的值是否都是 1 。如果满足此条件，返回 true␣
↪；否则，返回 false 。
boolean one() 检查Bitset 中 是否至少一位 的值是 1 。如果满足此条件，返回 true␣
↪；否则，返回 false 。
int count() 返回 Bitset 中值为 1 的位的 总数 。
String toString() 返回 Bitset 的当前组成情况。注意，在结果字符串中，第 i␣
↪个下标处的字符应该与 Bitset 中的第 i 位一致。
示例：输入["Bitset", "fix", "fix", "flip", "all", "unfix", "flip", "one", "unfix",
↪"count", "toString"]
[[5], [3], [1], [], [], [0], [], [], [0], [], []]
输出[null, null, null, null, false, null, null, true, null, 2, "01010"]
解释 Bitset bs = new Bitset(5); // bitset = "00000".
bs.fix(3);     // 将 idx = 3 处的值更新为 1 ，此时 bitset = "00010" 。
bs.fix(1);     // 将 idx = 1 处的值更新为 1 ，此时 bitset = "01010" 。
bs.flip();     // 翻转每一位上的值，此时 bitset = "10101" 。
bs.all();      // 返回 False ，bitset 中的值不全为 1 。
bs.unfix(0);   // 将 idx = 0 处的值更新为 0 ，此时 bitset = "00101" 。
bs.flip();     // 翻转每一位上的值，此时 bitset = "11010" 。
bs.one();      // 返回 True ，至少存在一位的值为 1 。
bs.unfix(0);   // 将 idx = 0 处的值更新为 0 ，此时 bitset = "01010" 。
bs.count();    // 返回 2 ，当前有 2 位的值为 1 。
bs.toString(); // 返回 "01010" ，即 bitset 的当前组成情况。
提示：1 <= size <= 105
```

(续下页)

(接上页)

```
0 <= idx <= size - 1
至多调用fix、unfix、flip、all、one、count 和 toString 方法 总共 105 次
至少调用 all、one、count 或 toString 方法一次
至多调用toString 方法 5 次
```

- 解题思路

```
type Bitset struct {
        arr       []int
        count     int // 1的数量
        flipCount int // 反转的次量
}

func Constructor(size int) Bitset {
        return Bitset{
                arr:       make([]int, size),
                count:     0,
                flipCount: 0,
        }
}

func (this *Bitset) Fix(idx int) {
        if (this.flipCount%2 == 0 && this.arr[idx] == 0) ||
                (this.flipCount%2 == 1 && this.arr[idx] == 1) {
                this.count++
                this.arr[idx] = 1 - this.arr[idx]
        }
}

func (this *Bitset) Unfix(idx int) {
        if (this.flipCount%2 == 0 && this.arr[idx] == 1) ||
                (this.flipCount%2 == 1 && this.arr[idx] == 0) {
                this.count--
                this.arr[idx] = 1 - this.arr[idx]
        }
}

func (this *Bitset) Flip() {
        this.flipCount++
        this.count = len(this.arr) - this.count
}

func (this *Bitset) All() bool {
        return len(this.arr) == this.count
```

(续下页)

(接上页)

```
}

func (this *Bitset) One() bool {
        return this.count > 0
}

func (this *Bitset) Count() int {
        return this.count
}

func (this *Bitset) ToString() string {
        temp := make([]byte, len(this.arr))
        for i := 0; i < len(this.arr); i++ {
                v := this.arr[i]
                if this.flipCount%2 == 1 {
                        v = 1 - v
                }
                temp[i] = byte('0' + v)
        }
        return string(temp)
}

# 2
type Bitset struct {
        arr       []byte
        count     int // 1的数量
        flipCount int // 反转的次量
}

func Constructor(size int) Bitset {
        arr := bytes.Repeat([]byte{'0'}, size)
        return Bitset{
                arr:       arr,
                count:     0,
                flipCount: 0,
        }
}

func (this *Bitset) Fix(idx int) {
        if (this.flipCount%2 == 0 && this.arr[idx] == '0') ||
                (this.flipCount%2 == 1 && this.arr[idx] == '1') {
                this.count++
                this.arr[idx] = '0' + ('1' - this.arr[idx])
```

(续下页)

(接上页)

```
        }
}

func (this *Bitset) Unfix(idx int) {
        if (this.flipCount%2 == 0 && this.arr[idx] == '1') ||
                (this.flipCount%2 == 1 && this.arr[idx] == '0') {
                this.count--
                this.arr[idx] = '0' + ('1' - this.arr[idx])
        }
}

func (this *Bitset) Flip() {
        this.flipCount++
        this.count = len(this.arr) - this.count
}

func (this *Bitset) All() bool {
        return len(this.arr) == this.count
}

func (this *Bitset) One() bool {
        return this.count > 0
}

func (this *Bitset) Count() int {
        return this.count
}

func (this *Bitset) ToString() string {
        if this.flipCount%2 == 1 {
                temp := make([]byte, len(this.arr))
                for i := 0; i < len(this.arr); i++ {
                        temp[i] = '0' + ('1' - this.arr[i])
                }
                return string(temp)
        }
        return string(this.arr)
}
```

65.28 2170. 使数组变成交替数组的最少操作数 (2)

- 题目

```
给你一个下标从 0 开始的数组 nums ，该数组由 n 个正整数组成。
如果满足下述条件，则数组 nums 是一个 交替数组 ：
nums[i - 2] == nums[i] ，其中 2 <= i <= n - 1 。
nums[i - 1] != nums[i] ，其中 1 <= i <= n - 1 。
在一步 操作 中，你可以选择下标 i 并将 nums[i] 更改 为 任一 正整数。
返回使数组变成交替数组的 最少操作数 。
示例 1：输入：nums = [3,1,3,2,4,3] 输出：3
解释：使数组变成交替数组的方法之一是将该数组转换为 [3,1,3,1,3,1] 。
在这种情况下，操作数为 3 。
可以证明，操作数少于 3 的情况下，无法使数组变成交替数组。
示例 2：输入：nums = [1,2,2,2,2] 输出：2
解释：使数组变成交替数组的方法之一是将该数组转换为 [1,2,1,2,1].
在这种情况下，操作数为 2 。
注意，数组不能转换成 [2,2,2,2,2] 。因为在这种情况下，nums[0] ==␣
↪nums[1]，不满足交替数组的条件。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 105
```

- 解题思路

```
func minimumOperations(nums []int) int {
        m := [2]map[int]int{}
        for i := 0; i < 2; i++ {
                m[i] = make(map[int]int)
        }
        for i := 0; i < len(nums); i++ {
                if i%2 == 0 {
                        m[0][nums[i]]++
                } else {
                        m[1][nums[i]]++
                }
        }
        a := getMaxValue(m[0])
        b := getMaxValue(m[1])
        if a[0] == b[0] {
                return len(nums) - max(a[1]+b[3], a[3]+b[1])
        }
        return len(nums) - a[1] - b[1]
}
```

(续下页)

(接上页)

```
func getMaxValue(m map[int]int) []int {
        firstValue, firstIndex := 0, 0
        secondValue, secondIndex := 0, 0
        for k, v := range m {
                if v > firstValue {
                        secondIndex, secondValue = firstIndex, firstValue
                        firstIndex, firstValue = k, v
                } else if v > secondValue {
                        secondIndex, secondValue = k, v
                }
        }
        return []int{firstIndex, firstValue, secondIndex, secondValue}
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minimumOperations(nums []int) int {
        m := [2]map[int]int{}
        for i := 0; i < 2; i++ {
                m[i] = make(map[int]int)
        }
        for i := 0; i < len(nums); i++ {
                if i%2 == 0 {
                        m[0][nums[i]]++
                } else {
                        m[1][nums[i]]++
                }
        }
        a := make([][2]int, 0)
        b := make([][2]int, 0)
        for i := 0; i < 2; i++ {
                temp := make([][2]int, 2) // 0补足
                for k, v := range m[i] {
                        temp = append(temp, [2]int{k, v})
                }
                sort.Slice(temp, func(i, j int) bool {
                        return temp[i][1] > temp[j][1]
```

(续下页)

(接上页)

```
                })
                if i%2 == 0 {
                        a = temp[:2]
                } else {
                        b = temp[:2]
                }
        }
        if a[0][0] == b[0][0] {
                return len(nums) - max(a[0][1]+b[1][1], a[1][1]+b[0][1])
        }
        return len(nums) - a[0][1] - b[0][1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

65.29 2171. 拿出最少数目的魔法豆 (1)

- 题目

```
给你一个 正整数数组beans，其中每个整数表示一个袋子里装的魔法豆的数目。
请你从每个袋子中拿出一些豆子（也可以不拿出），使得剩下的 非空 袋子中（即 至少还有一颗魔法豆的袋子）魔法豆的数目相等。
一旦魔法豆从袋子中取出，你不能将它放到任何其他的袋子中。
请你返回你需要拿出魔法豆的 最少数目。
示例 1：输入：beans = [4,1,6,5] 输出：4
解释：- 我们从有 1 个魔法豆的袋子中拿出 1 颗魔法豆。
  剩下袋子中魔法豆的数目为：[4,0,6,5]
- 然后我们从有 6 个魔法豆的袋子中拿出 2 个魔法豆。
  剩下袋子中魔法豆的数目为：[4,0,4,5]
- 然后我们从有 5 个魔法豆的袋子中拿出 1 个魔法豆。
  剩下袋子中魔法豆的数目为：[4,0,4,4]
总共拿出了 1 + 2 + 1 = 4 个魔法豆，剩下非空袋子中魔法豆的数目相等。
没有比取出 4 个魔法豆更少的方案。
示例 2：输入：beans = [2,10,3,2] 输出：7
解释：- 我们从有 2 个魔法豆的其中一个袋子中拿出 2 个魔法豆。
  剩下袋子中魔法豆的数目为：[0,10,3,2]
- 然后我们从另一个有 2 个魔法豆的袋子中拿出 2 个魔法豆。
```

(续下页)

(接上页)

```
  剩下袋子中魔法豆的数目为：[0,10,3,0]
- 然后我们从有 3 个魔法豆的袋子中拿出 3 个魔法豆。
  剩下袋子中魔法豆的数目为：[0,10,0,0]
总共拿出了 2 + 2 + 3 = 7 个魔法豆，剩下非空袋子中魔法豆的数目相等。
没有比取出 7 个魔法豆更少的方案。
提示：1 <= beans.length <= 105
1 <= beans[i] <= 105
```

- 解题思路

```
func minimumRemoval(beans []int) int64 {
        n := len(beans)
        sum := int64(0)
        for i := 0; i < n; i++ {
                sum = sum + int64(beans[i])
        }
        res := sum
        sort.Ints(beans)
        for i := 0; i < n; i++ {
                res = min(res, sum-int64(beans[i])*int64(n-i)) // 把较大的n-
↪i个数都变为beans[i]
        }
        return res
}

func min(a, b int64) int64 {
        if a > b {
                return b
        }
        return a
}
```

65.30 2177. 找到和为给定整数的三个连续整数 (1)

- 题目

```
给你一个整数num，请你返回三个连续的整数，它们的和为num。如果num无法被表示成三个连续整数的和，请你返回
↪空数组。
示例 1：输入：num = 33 输出：[10,11,12]
解释：33 可以表示为 10 + 11 + 12 = 33 。
10, 11, 12 是 3 个连续整数，所以返回 [10, 11, 12] 。
示例 2：输入：num = 4 输出：[]
```

(续下页)

(接上页)

```
解释：没有办法将 4 表示成 3 个连续整数的和。
提示：0 <= num <= 1015
```

- 解题思路

```
func sumOfThree(num int64) []int64 {
        if num%3 == 0 {
                target := num / 3
                return []int64{target - 1, target, target + 1}
        }
        return nil
}
```

65.31 2178. 拆分成最多数目的正偶数之和 (1)

- 题目

```
给你一个整数finalSum。请你将它拆分成若干个互不相同␣
↪的正偶数之和，且拆分出来的正偶数数目最多。
比方说，给你finalSum = 12，那么这些拆分是符合要求␣
↪的（互不相同的正偶数且和为finalSum）：
(2 + 10)，(2 + 4 + 6)和(4 + 8)。
它们中，(2 + 4 + 6)包含最多数目的整数。注意finalSum不能拆分成(2 + 2 + 4 +␣
↪4)，因为拆分出来的整数必须互不相同。
请你返回一个整数数组，表示将整数拆分成 最多 数目的正偶数数组。
如果没有办法将finalSum进行拆分，请你返回一个空数组。你可以按 任意顺序返回这些整数。
示例 1：输入：finalSum = 12 输出：[2,4,6]
解释：以下是一些符合要求的拆分：(2 + 10)，(2 + 4 + 6) 和 (4 + 8) 。
(2 + 4 + 6) 为最多数目的整数，数目为 3 ，所以我们返回 [2,4,6] 。
[2,6,4] ，[6,2,4] 等等也都是可行的解。
示例 2：输入：finalSum = 7 输出：[]
解释：没有办法将 finalSum 进行拆分。
所以返回空数组。
示例 3：输入：finalSum = 28 输出：[6,8,2,12]
解释：以下是一些符合要求的拆分：(2 + 26)，(6 + 8 + 2 + 12) 和 (4 + 24) 。
(6 + 8 + 2 + 12) 有最多数目的整数，数目为 4 ，所以我们返回 [6,8,2,12] 。
[10,2,4,12] ，[6,2,4,16] 等等也都是可行的解。
提示：1 <= finalSum <= 1010
```

- 解题思路

```
func maximumEvenSplit(finalSum int64) []int64 {
        if finalSum%2 == 1 {
                return nil
        }
        res := make([]int64, 0)
        for i := int64(2); i <= finalSum; i = i + 2 {
                res = append(res, i)
                finalSum = finalSum - i // 减去当前值
        }
        res[len(res)-1] = res[len(res)-1] + finalSum // 剩下数加到最后1位
        return res
}
```

65.32 2181. 合并零之间的节点 (2)

- 题目

```
给你一个链表的头节点 head ，该链表包含由 0 分隔开的一连串整数。链表的 开端 和 末尾␣
↪的节点都满足 Node.val == 0 。
对于每两个相邻的 0␣
↪，请你将它们之间的所有节点合并成一个节点，其值是所有已合并节点的值之和。
然后将所有 0 移除，修改后的链表不应该含有任何 0 。
返回修改后链表的头节点 head 。
示例 1：输入：head = [0,3,1,0,4,5,2,0] 输出：[4,11]
解释：上图表示输入的链表。修改后的链表包含：
- 标记为绿色的节点之和：3 + 1 = 4
- 标记为红色的节点之和：4 + 5 + 2 = 11
示例 2：输入：head = [0,1,0,3,0,2,2,0] 输出：[1,3,4]
解释：上图表示输入的链表。修改后的链表包含：
- 标记为绿色的节点之和：1 = 1
- 标记为红色的节点之和：3 = 3
- 标记为黄色的节点之和：2 + 2 = 4
提示：列表中的节点数目在范围 [3, 2 * 105] 内
0 <= Node.val <= 1000
不 存在连续两个Node.val == 0 的节点
链表的 开端 和 末尾 节点都满足 Node.val == 0
```

- 解题思路

```
func mergeNodes(head *ListNode) *ListNode {
        temp := make([]int, 0)
        for head != nil {
                temp = append(temp, head.Val)
```

(续下页)

(接上页)

```
                head = head.Next
        }
        sum := 0
        arr := make([]int, 0)
        for i := 1; i < len(temp); i++ {
                if temp[i] == 0 && sum != 0 {
                        arr = append(arr, sum)
                        sum = 0
                } else {
                        sum = sum + temp[i]
                }
        }
        res := &ListNode{}
        t := res
        for i := 0; i < len(arr); i++ {
                t.Next = &ListNode{
                        Val:  arr[i],
                        Next: nil,
                }
                t = t.Next
        }
        return res.Next
}

# 2
func mergeNodes(head *ListNode) *ListNode {
        res := &ListNode{}
        t := res
        sum := 0
        for cur := head.Next; cur != nil; cur = cur.Next {
                if cur.Val == 0 {
                        node := &ListNode{
                                Val: sum,
                        }
                        t.Next = node
                        t = t.Next
                        sum = 0
                } else {
                        sum = sum + cur.Val
                }
        }
        return res.Next
}
```

65.33 2182. 构造限制重复的字符串 (1)

- 题目

```
给你一个字符串 s 和一个整数 repeatLimit ，用 s 中的字符构造一个新字符串␣
↪repeatLimitedString ，
使任何字母 连续 出现的次数都不超过 repeatLimit 次。你不必使用 s 中的全部字符。
返回 字典序最大的 repeatLimitedString 。
如果在字符串 a 和 b 不同的第一个位置，字符串 a 中的字母在字母表中出现时间比字符串 b␣
↪对应的字母晚，
则认为字符串 a 比字符串 b 字典序更大 。如果字符串中前 min(a.length, b.length)␣
↪个字符都相同，那么较长的字符串字典序更大。
示例 1：输入：s = "cczazcc", repeatLimit = 3 输出："zzcccac"
解释：使用 s 中的所有字符来构造 repeatLimitedString "zzcccac"。
字母 'a' 连续出现至多 1 次。
字母 'c' 连续出现至多 3 次。
字母 'z' 连续出现至多 2 次。
因此，没有字母连续出现超过 repeatLimit 次，字符串是一个有效的 repeatLimitedString 。
该字符串是字典序最大的 repeatLimitedString ，所以返回 "zzcccac" 。
注意，尽管 "zzcccca" 字典序更大，但字母 'c' 连续出现超过 3 次，所以它不是一个有效的␣
↪repeatLimitedString 。
示例 2：输入：s = "aababab", repeatLimit = 2 输出："bbabaa"
解释：使用 s 中的一些字符来构造 repeatLimitedString "bbabaa"。
字母 'a' 连续出现至多 2 次。
字母 'b' 连续出现至多 2 次。
因此，没有字母连续出现超过 repeatLimit 次，字符串是一个有效的 repeatLimitedString 。
该字符串是字典序最大的 repeatLimitedString ，所以返回 "bbabaa" 。
注意，尽管 "bbabaaa" 字典序更大，但字母 'a' 连续出现超过 2 次，所以它不是一个有效的␣
↪repeatLimitedString 。
提示：1 <= repeatLimit <= s.length <= 105
s 由小写英文字母组成
```

- 解题思路

```go
func repeatLimitedString(s string, repeatLimit int) string {
        arr := make([]int, 26)
        for i := 0; i < len(s); i++ {
                arr[int(s[i]-'a')]++
        }
        res := make([]byte, 0)
        for i := 25; i >= 0; i-- {
                prev := i - 1
                for {
                        count := min(repeatLimit, arr[i])
```

(续下页)

(接上页)

```
                        arr[i] = arr[i] - count
                        res = append(res, bytes.Repeat([]byte{byte('a' + i)}, count)..
↪.)
                        if arr[i] == 0 { // 用完退出
                                break
                        }
                        for prev >= 0 && arr[prev] == 0 {
                                prev--
                        }
                        if prev < 0 { // 没有次大值退出
                                break
                        }
                        arr[prev]--
                        res = append(res, byte('a'+prev)) // 添加次大值
                }
        }
        return string(res)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

65.34 2186. 使两字符串互为字母异位词的最少步骤数 (1)

- 题目

```
给你两个字符串 s 和 t 。在一步操作中，你可以给 s 或者 t 追加 任一字符 。
返回使 s 和 t 互为 字母异位词 所需的最少步骤数。
字母异位词 指字母相同但是顺序不同（或者相同）的字符串。
示例 1：输入：s = "leetcode", t = "coats" 输出：7
解释：- 执行 2 步操作，将 "as" 追加到 s = "leetcode" 中，得到 s = "leetcodeas" 。
- 执行 5 步操作，将 "leede" 追加到 t = "coats" 中，得到 t = "coatsleede" 。
"leetcodeas" 和 "coatsleede" 互为字母异位词。
总共用去 2 + 5 = 7 步。
可以证明，无法用少于 7 步操作使这两个字符串互为字母异位词。
示例 2：输入：s = "night", t = "thing" 输出：0
解释：给出的字符串已经互为字母异位词。因此，不需要任何进一步操作。
提示：1 <= s.length, t.length <= 2 * 105
```

(续下页)

(接上页)

```
s 和 t 由小写英文字符组成
```

- 解题思路

```
func minSteps(s string, t string) int {
        arr := make([]int, 26)
        for i := 0; i < len(s); i++ {
                arr[int(s[i]-'a')]++
        }
        for i := 0; i < len(t); i++ {
                arr[int(t[i]-'a')]--
        }
        res := 0
        for i := 0; i < 26; i++ {
                res = res + abs(arr[i])
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

65.35 2187. 完成旅途的最少时间 (2)

- 题目

```
给你一个数组time，其中time[i]表示第 i辆公交车完成 一趟旅途所需要花费的时间。
每辆公交车可以 连续 完成多趟旅途，也就是说，一辆公交车当前旅途完成后，可以␣
↪立马开始下一趟旅途。
每辆公交车 独立运行，也就是说可以同时有多辆公交车在运行且互不影响。
给你一个整数totalTrips，表示所有公交车总共需要完成的旅途数目。请你返回完成␣
↪至少totalTrips趟旅途需要花费的 最少时间。
示例 1：输入：time = [1,2,3], totalTrips = 5 输出：3
解释：- 时刻 t = 1 ，每辆公交车完成的旅途数分别为 [1,0,0] 。
  已完成的总旅途数为 1 + 0 + 0 = 1 。
- 时刻 t = 2 ，每辆公交车完成的旅途数分别为 [2,1,0] 。
  已完成的总旅途数为 2 + 1 + 0 = 3 。
- 时刻 t = 3 ，每辆公交车完成的旅途数分别为 [3,1,1] 。
```

(续下页)

(接上页)

```
  已完成的总旅途数为 3 + 1 + 1 = 5 。
所以总共完成至少 5 趟旅途的最少时间为 3 。
示例 2：输入：time = [2], totalTrips = 1 输出：2
解释：只有一辆公交车，它将在时刻 t = 2 完成第一趟旅途。
所以完成 1 趟旅途的最少时间为 2 。
提示：1 <= time.length <= 105
1 <= time[i], totalTrips <= 107
```

- 解题思路

```
func minimumTime(time []int, totalTrips int) int64 {
        n := len(time)
        maxValue := 0
        for i := 0; i < n; i++ {
                maxValue = max(maxValue, time[i])
        }
        left := int64(1)
        right := int64(totalTrips) * int64(maxValue)
        for left < right {
                mid := left + (right-left)/2
                if check(time, mid) >= totalTrips {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}

func check(arr []int, per int64) int {
        count := 0
        for i := 0; i < len(arr); i++ {
                count = count + int(per/int64(arr[i]))
        }
        return count
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
func minimumTime(time []int, totalTrips int) int64 {
        n := len(time)
        maxValue := 0
        for i := 0; i < n; i++ {
                maxValue = max(maxValue, time[i])
        }
        right := totalTrips * maxValue
        return int64(sort.Search(right, func(per int) bool {
                count := 0
                for i := 0; i < len(time); i++ {
                        count = count + int(per/time[i])
                }
                return count >= totalTrips
        }))
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

65.36 2192. 有向无环图中一个节点的所有祖先 (1)

- 题目

```
给你一个正整数n，它表示一个 有向无环图中节点的数目，节点编号为0到n - 1（包括两者）。
给你一个二维整数数组edges，其中edges[i] = [fromi,␣
↪toi]表示图中一条从fromi到toi的单向边。
请你返回一个数组answer，其中answer[i]是第i个节点的所有祖先，这些祖先节点升序排序。
如果 u通过一系列边，能够到达 v，那么我们称节点 u是节点 v的 祖先节点。
示例 1：输入：n = 8, edgeList = [[0,3],[0,4],[1,3],[2,4],[2,7],[3,5],[3,6],[3,7],[4,
↪6]]
输出：[[],[],[],[0,1],[0,2],[0,1,3],[0,1,2,3,4],[0,1,2,3]]
解释：上图为输入所对应的图。
- 节点 0，1 和 2 没有任何祖先。
- 节点 3 有 2 个祖先 0 和 1 。
- 节点 4 有 2 个祖先 0 和 2 。
- 节点 5 有 3 个祖先 0，1 和 3 。
- 节点 6 有 5 个祖先 0，1，2，3 和 4 。
```

(续下页)

(接上页)

```
- 节点 7 有 4 个祖先 0 ，1 ，2 和 3 。
示例 2：输入：n = 5, edgeList = [[0,1],[0,2],[0,3],[0,4],[1,2],[1,3],[1,4],[2,3],[2,
↪4],[3,4]]
输出：[[],[0],[0,1],[0,1,2],[0,1,2,3]]
解释：上图为输入所对应的图。
- 节点 0 没有任何祖先。
- 节点 1 有 1 个祖先 0 。
- 节点 2 有 2 个祖先 0 和 1 。
- 节点 3 有 3 个祖先 0 ，1 和 2 。
- 节点 4 有 4 个祖先 0 ，1 ，2 和 3 。
提示：1 <= n <= 1000
0 <= edges.length <= min(2000, n * (n - 1) / 2)
edges[i].length == 2
0 <= fromi, toi <= n - 1
fromi != toi
图中不会有重边。
图是 有向 且 无环 的。
```

- 解题思路

```go
func getAncestors(n int, edges [][]int) [][]int {
        m := make([]map[int]bool, n) // 祖先节点要去重
        for i := 0; i < n; i++ {
                m[i] = make(map[int]bool)
        }
        arr := make([][]int, n)
        inDegree := make([]int, n) // 入度
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1] // a=>b
                arr[a] = append(arr[a], b)
                inDegree[b]++
        }
        queue := make([]int, 0)
        for i := 0; i < n; i++ {
                if inDegree[i] == 0 { // 入度为0的
                        queue = append(queue, i)
                }
        }
        for len(queue) > 0 {
                cur := queue[0]
                queue = queue[1:]
                for i := 0; i < len(arr[cur]); i++ {
                        next := arr[cur][i]
                        m[next][cur] = true     // 保存父节点
```

(续下页)

(接上页)

```
                        for k := range m[cur] { // 把父节点的祖先节点也保存下来
                                m[next][k] = true
                        }
                        inDegree[next]--
                        if inDegree[next] == 0 {
                                queue = append(queue, next)
                        }
                }
        }
        // 排序
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                for k, _ := range m[i] {
                        res[i] = append(res[i], k)
                }
                sort.Ints(res[i])
        }
        return res
}
```

65.37 2195. 向数组中追加 K 个整数

65.37.1 题目

```
给你一个整数数组 nums 和一个整数 k 。
请你向 nums 中追加 k 个 未 出现在 nums 中的、互不相同 的 正␣
↪整数，并使结果数组的元素和 最小 。
返回追加到 nums 中的 k 个整数之和。
示例 1：输入：nums = [1,4,25,10,25], k = 2 输出：5
解释：在该解法中，向数组中追加的两个互不相同且未出现的正整数是 2 和 3 。
nums 最终元素和为 1 + 4 + 25 + 10 + 25 + 2 + 3 = 70 ，这是所有情况中的最小值。
所以追加到数组中的两个整数之和是 2 + 3 = 5 ，所以返回 5 。
示例 2：输入：nums = [5,6], k = 6 输出：25
解释：在该解法中，向数组中追加的两个互不相同且未出现的正整数是 1 、2 、3 、4 、7 和 8␣
↪。
nums 最终元素和为 5 + 6 + 1 + 2 + 3 + 4 + 7 + 8 = 36 ，这是所有情况中的最小值。
所以追加到数组中的两个整数之和是 1 + 2 + 3 + 4 + 7 + 8 = 25 ，所以返回 25 。
提示：1 <= nums.length <= 105
1 <= nums[i], k <= 109
```

65.37.2 解题思路

```
func minimalKSum(nums []int, k int) int64 {
        sum := int64(0)
        sort.Ints(nums)
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] == false && nums[i] <= k {
                        sum = sum + int64(nums[i]) // 存在小于K的，加起来
                        k++                        // k+1
                }
                m[nums[i]] = true
        }
        return int64((k+1)*k/2) - sum
}
```

65.38 2196. 根据描述创建二叉树 (1)

- 题目

```
给你一个二维整数数组 descriptions ，其中 descriptions[i] = [parenti, childi, isLefti]
表示 parenti 是 childi 在 二叉树 中的 父节点，二叉树中各节点的值 互不相同 。此外：
如果 isLefti == 1 ，那么 childi 就是 parenti 的左子节点。
如果 isLefti == 0 ，那么 childi 就是 parenti 的右子节点。
请你根据 descriptions 的描述来构造二叉树并返回其 根节点 。
测试用例会保证可以构造出 有效 的二叉树。
示例 1：输入：descriptions = [[20,15,1],[20,17,0],[50,20,1],[50,80,0],[80,19,1]]
↪输出：[50,20,80,15,17,19]
解释：根节点是值为 50 的节点，因为它没有父节点。
结果二叉树如上图所示。
示例 2：输入：descriptions = [[1,2,1],[2,3,0],[3,4,1]] 输出：[1,2,null,null,3,4]
解释：根节点是值为 1 的节点，因为它没有父节点。
结果二叉树如上图所示。
提示：1 <= descriptions.length <= 104
descriptions[i].length == 3
1 <= parenti, childi <= 105
0 <= isLefti <= 1
descriptions 所描述的二叉树是一棵有效二叉树
```

- 解题思路

```
func createBinaryTree(descriptions [][]int) *TreeNode {
        m := make(map[int]*TreeNode)
```

(续下页)

(接上页)

```
        visited := make(map[int]bool) // 保存子节点
        for i := 0; i < len(descriptions); i++ {
                a, b, c := descriptions[i][0], descriptions[i][1], descriptions[i][2]
                if m[a] == nil {
                        m[a] = &TreeNode{Val: a}
                }
                if m[b] == nil {
                        m[b] = &TreeNode{Val: b}
                }
                if c == 1 {
                        m[a].Left = m[b]
                } else {
                        m[a].Right = m[b]
                }
                visited[b] = true
        }
        for k := range m {
                if visited[k] == false { // 根节点不属于子节点
                        return m[k]
                }
        }
        return nil
}
```

CHAPTER 66

2101-2200-Hard

66.1 2106. 摘水果 (2)

- 题目

```
在一个无限的 x 坐标轴上，有许多水果分布在其中某些位置。
给你一个二维整数数组 fruits ，其中 fruits[i] = [positioni, amounti] 表示共有 amounti
↪个水果放置在 positioni 上。
fruits 已经按 positioni 升序排列 ，每个 positioni 互不相同 。
另给你两个整数 startPos 和 k 。最初，你位于 startPos 。从任何位置，你可以选择
↪向左或者向右 走。
在 x 轴上每移动 一个单位 ，就记作 一步 。你总共可以走 最多 k 步。
你每达到一个位置，都会摘掉全部的水果，水果也将从该位置消失（不会再生）。
返回你可以摘到水果的 最大总数 。
示例 1：输入：fruits = [[2,8],[6,3],[8,6]], startPos = 5, k = 4 输出：9
解释：最佳路线为：
- 向右移动到位置 6 ，摘到 3 个水果
- 向右移动到位置 8 ，摘到 6 个水果
移动 3 步，共摘到 3 + 6 = 9 个水果
示例 2：输入：fruits = [[0,9],[4,1],[5,7],[6,2],[7,4],[10,9]], startPos = 5, k = 4
↪输出：14
解释：可以移动最多 k = 4 步，所以无法到达位置 0 和位置 10 。
最佳路线为：
- 在初始位置 5 ，摘到 7 个水果
- 向左移动到位置 4 ，摘到 1 个水果
```

(续下页)

(接上页)

```
- 向右移动到位置 6 ，摘到 2 个水果
- 向右移动到位置 7 ，摘到 4 个水果
移动 1 + 3 = 4 步，共摘到 7 + 1 + 2 + 4 = 14 个水果
示例 3：输入：fruits = [[0,3],[6,4],[8,5]], startPos = 3, k = 2 输出：0
解释：最多可以移动 k = 2 步，无法到达任一有水果的地方
提示：1 <= fruits.length <= 105
fruits[i].length == 2
0 <= startPos, positioni <= 2 * 105
对于任意 i > 0 ，positioni-1 < positioni 均成立（下标从 0 开始计数）
1 <= amounti <= 104
0 <= k <= 2 * 105
```

- 解题思路

```
func maxTotalFruits(fruits [][]int, startPos int, k int) int {
        res := 0
        n := len(fruits)
        sum := make([]int, n+1)
        position := make([]int, n)
        for i := 0; i < n; i++ {
                position[i] = fruits[i][0]
                sum[i+1] = sum[i] + fruits[i][1]
        }
        for i := 0; i <= k/2; i++ { // 尝试所有组合，先i后j
                j := k - 2*i
                res = max(res, getValue(position, sum, startPos-i, startPos+j)) //
↪i往左边、j往右边
                res = max(res, getValue(position, sum, startPos-j, startPos+i)) //
↪i往右边、j往左边
        }
        return res
}

func getValue(arr []int, sum []int, left, right int) int {
        l := sort.Search(len(arr), func(i int) bool {
                return arr[i] >= left // 第一个大于等于left的下标
        })
        r := sort.Search(len(arr), func(i int) bool {
                return arr[i] > right // 第一个大于right的下标
        })
        return sum[r] - sum[l]
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 2
func maxTotalFruits(fruits [][]int, startPos int, k int) int {
        res := 0
        sum := 0
        i := 0
        for j := 0; j < len(fruits); j++ {
                // 滑动窗口：左右距离+2者最小距离<=k
                for i <= j && (min(abs(startPos-fruits[i][0]), abs(startPos-
↪fruits[j][0]))+
                        fruits[j][0]-fruits[i][0]) > k {
                        sum = sum - fruits[i][1]
                        i++
                }
                sum = sum + fruits[j][1]
                res = max(res, sum)
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
```

(续下页)

(接上页)

```
        return a
}
```

66.2 2111. 使数组 K 递增的最少操作次数 (2)

- 题目

```
给你一个下标从 0开始包含 n个正整数的数组arr，和一个正整数k。
如果对于每个满足k <= i <= n-1的下标i，都有arr[i-k] <= arr[i]，那么我们称arr是 K递增的。
比方说，arr = [4, 1, 5, 2, 6, 2]对于k = 2是 K 递增的，因为：
arr[0] <= arr[2] (4 <= 5)
arr[1] <= arr[3] (1 <= 2)
arr[2] <= arr[4] (5 <= 6)
arr[3] <= arr[5] (2 <= 2)
但是，相同的数组arr对于k = 1不是 K 递增的（因为arr[0] > arr[1]），
对于k = 3也不是 K 递增的（因为arr[0] > arr[3]）。
每一次 操作中，你可以选择一个下标i 并将arr[i] 改成任意正整数。
请你返回对于给定的 k，使数组变成 K 递增的 最少操作次数。
示例 1：输入：arr = [5,4,3,2,1], k = 1 输出：4
解释：对于 k = 1 ，数组最终必须变成非递减的。
可行的 K 递增结果数组为 [5,6,7,8,9]，[1,1,1,1,1]，[2,2,3,4,4] 。它们都需要 4 次操作。
次优解是将数组变成比方说 [6,7,8,9,10] ，因为需要 5 次操作。
显然我们无法使用少于 4 次操作将数组变成 K 递增的。
示例 2：输入：arr = [4,1,5,2,6,2], k = 2 输出：0
解释：这是题目描述中的例子。
对于每个满足 2 <= i <= 5 的下标 i ，有 arr[i-2] <= arr[i] 。
由于给定数组已经是 K 递增的，我们不需要进行任何操作。
示例 3：输入：arr = [4,1,5,2,6,2], k = 3 输出：2
解释：下标 3 和 5 是仅有的 3 <= i <= 5 且不满足 arr[i-3] <= arr[i] 的下标。
将数组变成 K 递增的方法之一是将 arr[3] 变为 4 ，且将 arr[5] 变成 5 。
数组变为 [4,1,5,4,6,5] 。
可能有其他方法将数组变为 K 递增的，但没有任何一种方法需要的操作次数小于 2 次。
提示：1 <= arr.length <= 105
1 <= arr[i], k <= arr.length
```

- 解题思路

```
func kIncreasing(arr []int, k int) int {
        res := 0
        for i := 1; i <= k; i++ {
                num := make([]int, 0)
```

(续下页)

(接上页)

```
                num = append(num, arr[i-1])
                for j := i - 1; j < len(arr)-k; j = j + k {
                        num = append(num, arr[j+k])
                }
                count := lengthOfLIS(num)
                res = res + len(num) - count
        }
        return res
}

func lengthOfLIS(nums []int) int {
        if len(nums) < 2 {
                return len(nums)
        }
        arr := make([]int, 0)
        arr = append(arr, nums[0])
        for i := 1; i < len(nums); i++ {
                index := upperBound(arr, nums[i])
                if index == len(arr) {
                        arr = append(arr, nums[i])
                } else {
                        arr[index] = nums[i]
                }
        }
        return len(arr)
}

// 返回第一个大于等于target的位置
func upperBound(arr []int, target int) int {
        left, right := 0, len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        left = mid + 1 // 收缩左边界
                } else if arr[mid] < target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}
```

(续下页)

(接上页)

```
# 2
func kIncreasing(arr []int, k int) int {
        res := 0
        for i := 0; i < k; i++ {
                dp := make([]int, 0)
                count := 0
                for j := i; j < len(arr); j = j + k {
                        count++
                        index := upperBound(dp, arr[j])
                        if index == len(dp) {
                                dp = append(dp, arr[j])
                        } else {
                                dp[index] = arr[j]
                        }
                }
                res = res + count - len(dp)
        }
        return res
}

// 返回第一个大于等于target的位置
func upperBound(arr []int, target int) int {
        left, right := 0, len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        left = mid + 1 // 收缩左边界
                } else if arr[mid] < target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}
```

66.3 2122. 还原原数组 (2)

- 题目

```
Alice 有一个下标从 0 开始的数组 arr ，由 n 个正整数组成。
她会选择一个任意的 正整数 k 并按下述方式创建两个下标从 0 开始的新整数数组 lower 和
↪higher ：
对每个满足 0 <= i < n 的下标 i ，lower[i] = arr[i] - k
对每个满足 0 <= i < n 的下标 i ，higher[i] = arr[i] + k
不幸地是，Alice 丢失了全部三个数组。
但是，她记住了在数组 lower 和 higher
↪中出现的整数，但不知道每个整数属于哪个数组。请你帮助 Alice 还原原数组。
给你一个由 2n 个整数组成的整数数组 nums ，其中 恰好 n 个整数出现在 lower
↪，剩下的出现在 higher ，还原并返回 原数组 arr 。
如果出现答案不唯一的情况，返回 任一 有效数组。
注意：生成的测试用例保证存在 至少一个 有效数组 arr 。
示例 1：输入：nums = [2,10,6,4,8,12] 输出：[3,7,11]
解释：如果 arr = [3,7,11] 且 k = 1 ，那么 lower = [2,6,10] 且 higher = [4,8,12] 。
组合 lower 和 higher 得到 [2,6,10,4,8,12] ，这是 nums 的一个排列。
另一个有效的数组是 arr = [5,7,9] 且 k = 3 。在这种情况下，lower = [2,4,6] 且 higher =
↪[8,10,12] 。
示例 2：输入：nums = [1,1,3,3] 输出：[2,2]
解释：如果 arr = [2,2] 且 k = 1 ，那么 lower = [1,1] 且 higher = [3,3] 。
组合 lower 和 higher 得到 [1,1,3,3] ，这是 nums 的一个排列。
注意，数组不能是 [1,3] ，因为在这种情况下，获得 [1,1,3,3] 唯一可行的方案是 k = 0 。
这种方案是无效的，k 必须是一个正整数。
示例 3：输入：nums = [5,435] 输出：[220]
解释：唯一可行的组合是 arr = [220] 且 k = 215 。在这种情况下，lower = [5] 且 higher =
↪[435] 。
提示：2 * n == nums.length
1 <= n <= 1000
1 <= nums[i] <= 109
生成的测试用例保证存在 至少一个 有效数组 arr
```

- 解题思路

```
func recoverArray(nums []int) []int {
        n := len(nums)
        sort.Ints(nums)
        for i := 1; i < n; i++ {
                if nums[i] == nums[0] || (nums[i]-nums[0])%2 == 1 || nums[i] ==
↪nums[i-1] {
                        continue
                }
```

(续下页)

(接上页)

```
                visited := make([]bool, n)
                visited[0], visited[i] = true, true
                res := make([]int, 0)
                k := (nums[i] - nums[0]) / 2
                res = append(res, nums[0]+k)
                left := 1
                right := i
                for j := 2; j <= n/2; j++ { // 还需要遍历n/2-1次
                        for visited[left] == true {
                                left++
                        }
                        for right < n && (visited[right] == true || nums[right]-
↪nums[left] != 2*k) {
                                right++
                        }
                        if right == n {
                                break
                        }
                        res = append(res, nums[left]+k)
                        visited[left], visited[right] = true, true
                }
                if len(res) == n/2 {
                        return res
                }
        }
        return nil
}

# 2
func recoverArray(nums []int) []int {
        n := len(nums)
        sort.Ints(nums)
        for i := 1; i < n; i++ {
                if nums[i] == nums[0] || (nums[i]-nums[0])%2 == 1 || nums[i] ==␣
↪nums[i-1] {
                        continue
                }
                // leetcode954.二倍数对数组
                m := make(map[int]int)
                for i := 0; i < n; i++ {
                        m[nums[i]]++
                }
                res := make([]int, 0)
```

(续下页)

(接上页)

```
                k := (nums[i] - nums[0]) / 2
                for j := 0; j < n; j++ {
                        if m[nums[j]] == 0 {
                                continue
                        }
                        if m[nums[j]+2*k] == 0 {
                                break
                        }
                        m[nums[j]]--
                        m[nums[j]+2*k]--
                        res = append(res, nums[j]+k)
                }
                if len(res) == n/2 {
                        return res
                }
        }
        return nil
}
```

66.4 2132. 用邮票贴满网格图

66.4.1 题目

```
给你一个m x n的二进制矩阵grid，每个格子要么为0（空）要么为1（被占据）。
给你邮票的尺寸为stampHeight x␣
↪stampWidth。我们想将邮票贴进二进制矩阵中，且满足以下限制和要求：
覆盖所有 空格子。
不覆盖任何 被占据的格子。
我们可以放入任意数目的邮票。
邮票可以相互有 重叠部分。
邮票不允许 旋转。
邮票必须完全在矩阵 内。
如果在满足上述要求的前提下，可以放入邮票，请返回true，否则返回false。
示例 1：输入：grid = [[1,0,0,0],[1,0,0,0],[1,0,0,0],[1,0,0,0],[1,0,0,0]], stampHeight␣
↪= 4, stampWidth = 3
输出：true
解释：我们放入两个有重叠部分的邮票（图中标号为 1 和 2），它们能覆盖所有与空格子。
示例 2：输入：grid = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]], stampHeight = 2,␣
↪stampWidth = 2  输出：false
解释：没办法放入邮票覆盖所有的空格子，且邮票不超出网格图以外。
提示：m == grid.length
```

(续下页)

(接上页)

```
n == grid[r].length
1 <= m, n <= 105
1 <= m * n <= 2 * 105
grid[r][c] 要么是0，要么是1 。
1 <= stampHeight, stampWidth <= 105
```

66.4.2 解题思路

```

```

66.5 2136. 全部开花的最早一天 (1)

- 题目

```
你有 n
↪枚花的种子。每枚种子必须先种下，才能开始生长、开花。播种需要时间，种子的生长也是如此。
给你两个下标从 0 开始的整数数组 plantTime 和 growTime ，每个数组的长度都是 n ：
plantTime[i] 是 播种 第 i 枚种子所需的 完整天数 。每天，你只能为播种某一枚种子而劳作。
无须 连续几天都在种同一枚种子，但是种子播种必须在你工作的天数达到 plantTime[i]
↪之后才算完成。
growTime[i] 是第 i 枚种子完全种下后生长所需的 完整天数 。在它生长的最后一天 之后
↪，将会开花并且永远 绽放 。
从第 0 开始，你可以按 任意 顺序播种种子。
返回所有种子都开花的 最早 一天是第几天。
示例 1：输入：plantTime = [1,4,3], growTime = [2,3,1] 输出：9
解释：灰色的花盆表示播种的日子，彩色的花盆表示生长的日子，花朵表示开花的日子。
一种最优方案是：
第 0 天，播种第 0 枚种子，种子生长 2 整天。并在第 3 天开花。
第 1、2、3、4 天，播种第 1 枚种子。种子生长 3 整天，并在第 8 天开花。
第 5、6、7 天，播种第 2 枚种子。种子生长 1 整天，并在第 9 天开花。
因此，在第 9 天，所有种子都开花。
示例 2：输入：plantTime = [1,2,3,2], growTime = [2,1,2,1] 输出：9
解释：灰色的花盆表示播种的日子，彩色的花盆表示生长的日子，花朵表示开花的日子。
一种最优方案是：
第 1 天，播种第 0 枚种子，种子生长 2 整天。并在第 4 天开花。
第 0、3 天，播种第 1 枚种子。种子生长 1 整天，并在第 5 天开花。
第 2、4、5 天，播种第 2 枚种子。种子生长 2 整天，并在第 8 天开花。
第 6、7 天，播种第 3 枚种子。种子生长 1 整天，并在第 9 天开花。
因此，在第 9 天，所有种子都开花。
示例 3：输入：plantTime = [1], growTime = [1] 输出：2
```

(续下页)

(接上页)

```
解释：第 0 天，播种第 0 枚种子。种子需要生长 1 整天，然后在第 2 天开花。
因此，在第 2 天，所有种子都开花。
提示：n == plantTime.length == growTime.length
1 <= n <= 105
1 <= plantTime[i], growTime[i] <= 104
```

- 解题思路

```
func earliestFullBloom(plantTime []int, growTime []int) int {
        n := len(plantTime)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = []int{plantTime[i], growTime[i]}
        }
        sort.Slice(arr, func(i, j int) bool { // 按生长日期降序排序
                return arr[i][1] > arr[j][1]
        })
        res := 0
        sum := 0
        // 播种的总时间不变（串行），生长时间尽可能小（并行）
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i][0]
                if sum+arr[i][1] > res {
                        res = sum + arr[i][1]
                }
        }
        return res
}
```

66.6 2141. 同时运行 N 台电脑的最长时间 (4)

- 题目

```
你有n台电脑。给你整数n和一个下标从␣
↪0开始的整数数组batteries，其中第i个电池可以让一台电脑 运行batteries[i]分钟。
你想使用这些电池让全部n台电脑 同时运行。
一开始，你可以给每台电脑连接␣
↪至多一个电池。然后在任意整数时刻，你都可以将一台电脑与它的电池断开连接，并连接另一个电池，
你可以进行这个操作 任意次。
新连接的电池可以是一个全新的电池，也可以是别的电脑用过的电池。断开连接和连接新的电池不会花费任何时间
注意，你不能给电池充电。
请你返回你可以让 n台电脑同时运行的 最长分钟数。
```

(续下页)

(接上页)

```
示例 1：输入：n = 2, batteries = [3,3,3] 输出：4
解释：一开始，将第一台电脑与电池 0 连接，第二台电脑与电池 1 连接。
2 分钟后，将第二台电脑与电池 1 断开连接，并连接电池 2 。注意，电池 0 还可以供电 1␣
↪分钟。
在第 3 分钟结尾，你需要将第一台电脑与电池 0 断开连接，然后连接电池 1 。
在第 4 分钟结尾，电池 1 也被耗尽，第一台电脑无法继续运行。
我们最多能同时让两台电脑同时运行 4 分钟，所以我们返回 4 。
示例 2：输入：n = 2, batteries = [1,1,1,1] 输出：2
解释：一开始，将第一台电脑与电池 0 连接，第二台电脑与电池 2 连接。
一分钟后，电池 0 和电池 2 同时耗尽，所以你需要将它们断开连接，并将电池 1␣
↪和第一台电脑连接，电池 3 和第二台电脑连接。
1 分钟后，电池 1 和电池 3 也耗尽了，所以两台电脑都无法继续运行。
我们最多能让两台电脑同时运行 2 分钟，所以我们返回 2 。
提示：1 <= n <= batteries.length <= 105
1 <= batteries[i] <= 109
```

- 解题思路

```
func maxRunTime(n int, batteries []int) int64 {
        sum := 0
        for i := 0; i < len(batteries); i++ {
                sum = sum + batteries[i]
        }
        left, right := 1, sum/n
        res := 0
        for left <= right {
                mid := left + (right-left)/2
                total := 0
                for j := 0; j < len(batteries); j++ {
                        // 假设让n台电脑运行mid分钟
                        // 电量>=mid的电池，只能被使用mid分钟，仅能给1台电脑充电
                        total = total + min(batteries[j], mid)
                }
                if total >= n*mid {
                        res = mid
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return int64(res)
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}

# 2
func maxRunTime(n int, batteries []int) int64 {
        sum := 0
        for i := 0; i < len(batteries); i++ {
                sum = sum + batteries[i]
        }
        return int64(sort.Search(sum/n, func(target int) bool {
                target++
                total := 0
                for i := 0; i < len(batteries); i++ {
                        total = total + min(batteries[i], target)
                }
                return target*n > total
        }))
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func maxRunTime(n int, batteries []int) int64 {
        sort.Slice(batteries, func(i, j int) bool {
                return batteries[i] > batteries[j]
        })
        sum := 0
        for i := 0; i < len(batteries); i++ {
                sum = sum + batteries[i]
        }
        for i := 0; i < len(batteries); i++ {
                if batteries[i] <= sum/n { // 不够用，返回
                        return int64(sum / n)
                }
                sum = sum - batteries[i] // 去除该电池
```

(续下页)

(接上页)

```
                n--
        }
        return 0
}

# 4
func maxRunTime(n int, batteries []int) int64 {
        sum := 0
        for i := 0; i < len(batteries); i++ {
                sum = sum + batteries[i]
        }
        left, right := 1, sum/n
        for left <= right {
                mid := left + (right-left)/2
                total := 0
                for j := 0; j < len(batteries); j++ {
                        // 假设让n台电脑运行mid分钟
                        // 电量>=mid的电池，只能被使用mid分钟，仅能给1台电脑充电
                        total = total + min(batteries[j], mid)
                }
                if total >= n*mid {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return int64(right)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

66.7 2147. 分隔长廊的方案数 (1)

- 题目

```
在一个图书馆的长廊里，有一些座位和装饰植物排成一列。给你一个下标从 0开始，长度为n
↪的字符串corridor，
它包含字母'S' 和'P'，其中每个'S'表示一个座位，每个'P'表示一株植物。
在下标 0的左边和下标 n - 1的右边 已经分别各放了一个屏风。你还需要额外放置一些屏风。
每一个位置i - 1 和i之间 (1 <= i <= n - 1)，至多能放一个屏风。
请你将走廊用屏风划分为若干段，且每一段内都
↪恰好有两个座位，而每一段内植物的数目没有要求。
可能有多种划分方案，如果两个方案中有任何一个屏风的位置不同，那么它们被视为 不同 方案。
请你返回划分走廊的方案数。由于答案可能很大，请你返回它对109 +
↪7取余的结果。如果没有任何方案，请返回0。
示例 1：输入：corridor = "SSPPSPS" 输出：3
解释：总共有 3 种不同分隔走廊的方案。
上图中黑色的竖线表示已经放置好的屏风。
上图每种方案中，每一段都恰好有 两个座位。
示例 2：输入：corridor = "PPSPSP" 输出：1
解释：只有 1 种分隔走廊的方案，就是不放置任何屏风。
放置任何的屏风都会导致有一段无法恰好有 2 个座位。
示例 3：输入：corridor = "S" 输出：0
解释：没有任何方案，因为总是有一段无法恰好有 2 个座位。
提示：n == corridor.length
1 <= n <= 105
corridor[i]要么是'S'，要么是'P' 。
```

- 解题思路

```
var mod = 1000000007

func numberOfWays(corridor string) int {
        res := 1
        prev := -1
        count := 0
        for i := 0; i < len(corridor); i++ {
                if corridor[i] == 'S' {
                        count++
                        // 统计每2个1组之间有多少可能；相乘即可
                        if count >= 3 && count%2 == 1 { // S>=3+奇数
                                res = res * (i - prev) % mod
                        }
                        prev = i // 上一个座位的位置
                }
```

(续下页)

(接上页)

```
        }
        if count < 2 || count%2 == 1 {
                return 0
        }
        return res
}
```

66.8 2157. 字符串分组

66.8.1 题目

```
给你一个下标从0开始的字符串数组words。每个字符串都只包含小写英文字母。words中任意一个子串中，每个字母都至多只出现一次。
如果通过以下操作之一，我们可以从 s1的字母集合得到s2的字母集合，那么我们称这两个字符串为 关联的：
往s1的字母集合中添加一个字母。
从s1的字母集合中删去一个字母。
将 s1中的一个字母替换成另外任意一个字母（也可以替换为这个字母本身）。
数组words可以分为一个或者多个无交集的 组。一个字符串与一个组如果满足以下任一条件，它就属于这个组：
它与组内 至少一个其他字符串关联。
它是这个组中 唯一的字符串。
注意，你需要确保分好组后，一个组内的任一字符串与其他组的字符串都不关联。可以证明在这个条件下，分组方
请你返回一个长度为 2的数组ans：
ans[0]是words分组后的总组数。
ans[1]是字符串数目最多的组所包含的字符串数目。
示例 1：输入：words = ["a","b","ab","cde"] 输出：[2,3]
解释：- words[0] 可以得到 words[1] （将 'a' 替换为 'b'）和 words[2] （添加 'b'）。
所以 words[0] 与 words[1] 和 words[2] 关联。
- words[1] 可以得到 words[0] （将 'b' 替换为 'a'）和 words[2] （添加 'a'）。
所以 words[1] 与 words[0] 和 words[2] 关联。
- words[2] 可以得到 words[0] （删去 'b'）和 words[1] （删去 'a'）。所以 words[2] 与words[0] 和 words[1] 关联。
- words[3] 与 words 中其他字符串都不关联。
所以，words 可以分成 2 个组 ["a","b","ab"] 和 ["cde"] 。最大的组大小为 3 。
示例 2：输入：words = ["a","ab","abc"] 输出：[1,3]
解释：- words[0] 与 words[1] 关联。
- words[1] 与 words[0] 和 words[2] 关联。
- words[2] 与 words[1] 关联。
由于所有字符串与其他字符串都关联，所以它们全部在同一个组内。
所以最大的组大小为 3 。
```

(续下页)

(接上页)

```
提示：1 <= words.length <= 2 * 104
1 <= words[i].length <= 26
words[i]只包含小写英文字母。
words[i] 中每个字母最多只出现一次。
```

66.8.2 解题思路

66.9 2167. 移除所有载有违禁货物车厢所需的最少时间 (3)

- 题目

```
给你一个下标从 0 开始的二进制字符串 s ，表示一个列车车厢序列。s[i] = '0' 表示第 i␣
↪节车厢 不 含违禁货物，
而 s[i] = '1' 表示第 i 节车厢含违禁货物。
作为列车长，你需要清理掉所有载有违禁货物的车厢。你可以不限次数执行下述三种操作中的任意一个：
从列车 左 端移除一节车厢（即移除 s[0]），用去 1 单位时间。
从列车 右 端移除一节车厢（即移除 s[s.length - 1]），用去 1 单位时间。
从列车车厢序列的 任意位置 移除一节车厢，用去 2 单位时间。
返回移除所有载有违禁货物车厢所需要的 最少 单位时间数。
注意，空的列车车厢序列视为没有车厢含违禁货物。
示例 1：输入：s = "1100101" 输出：5
解释：一种从序列中移除所有载有违禁货物的车厢的方法是：
- 从左端移除一节车厢 2 次。所用时间是 2 * 1 = 2 。
- 从右端移除一节车厢 1 次。所用时间是 1 。
- 移除序列中间位置载有违禁货物的车厢。所用时间是 2 。
总时间是 2 + 1 + 2 = 5 。
一种替代方法是：
- 从左端移除一节车厢 2 次。所用时间是 2 * 1 = 2 。
- 从右端移除一节车厢 3 次。所用时间是 3 * 1 = 3 。
总时间也是 2 + 3 = 5 。
5 是移除所有载有违禁货物的车厢所需要的最少单位时间数。
没有其他方法能够用更少的时间移除这些车厢。
示例 2：输入：s = "0010" 输出：2
解释：一种从序列中移除所有载有违禁货物的车厢的方法是：
- 从左端移除一节车厢 3 次。所用时间是 3 * 1 = 3 。
总时间是 3.
另一种从序列中移除所有载有违禁货物的车厢的方法是：
- 移除序列中间位置载有违禁货物的车厢。所用时间是 2 。
```

(续下页)

(接上页)

```
总时间是 2.
另一种从序列中移除所有载有违禁货物的车厢的方法是:
- 从右端移除一节车厢 2 次。所用时间是 2 * 1 = 2 。
总时间是 2.
2 是移除所有载有违禁货物的车厢所需要的最少单位时间数。
没有其他方法能够用更少的时间移除这些车厢。
提示: 1 <= s.length <= 2 * 105
s[i] 为 '0' 或 '1'
```

- 解题思路

```
func minimumTime(s string) int {
        n := len(s)
        res := n
        pre := make([]int, n+1) // 前缀和
        for i := 0; i < n; i++ {
                pre[i+1] = pre[i] + int(s[i]-'0')
        }
        // left + middle(left,right)*2 + right
        // => i + (n-1-j) + 2*countOne(left,right)
        // => i + (n-1-j) + 2 *(pre[j+1] - pre[i])
        // => (i-2*pre[i]) + (2*pre[j+1]-j) + (n-1)
        // 求最小值
        a := 0
        for j := 0; j < n; j++ {
                a = min(a, j-2*pre[j])
                b := 2*pre[j+1] - j
                res = min(res, a+b+n-1)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minimumTime(s string) int {
        n := len(s)
        res := n
        // left + middle(left,right)*2 + right
```

(续下页)

(接上页)

```
        // => i + (n-1-j) + 2*countOne(left,right)
        // => i + (n-1-j) + 2 *(pre[j+1] - pre[i])
        // => (i-2*pre[i]) + (2*pre[j+1]-j) + (n-1)
        // 求最小值
        a := 0
        sum := 0
        for j := 0; j < n; j++ {
                a = min(a, j-2*sum)
                sum = sum + int(s[j]-'0')
                b := 2*sum - j
                res = min(res, a+b+n-1)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func minimumTime(s string) int {
        n := len(s)
        res := n
        count := 0
        for i := 0; i < n; i++ {
                if s[i] == '1' {
                        count = min(count+2, i+1) // left+middle: 左边+中间最小值
                }
                res = min(res, count+n-1-i) // (left+middle)+right
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

66.10 2183. 统计可以被 K 整除的下标对数目 (2)

- 题目

```
给你一个下标从 0 开始、长度为 n 的整数数组 nums 和一个整数 k␣
↪，返回满足下述条件的下标对 (i, j) 的数目：
0 <= i < j <= n - 1 且 nums[i] * nums[j] 能被 k 整除。
示例 1：输入：nums = [1,2,3,4,5], k = 2 输出：7
解释：共有 7 对下标的对应积可以被 2 整除：
(0, 1)、(0, 3)、(1, 2)、(1, 3)、(1, 4)、(2, 3) 和 (3, 4)
它们的积分别是 2、4、6、8、10、12 和 20 。
其他下标对，例如 (0, 2) 和 (2, 4) 的乘积分别是 3 和 15 ，都无法被 2 整除。
示例 2：输入：nums = [1,2,3,4], k = 5 输出：0
解释：不存在对应积可以被 5 整除的下标对。
提示：1 <= nums.length <= 105
1 <= nums[i], k <= 105
```

- 解题思路

```
func countPairs(nums []int, k int) int64 {
        res := int64(0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[gcd(k, nums[i])]++ // k和nums[i]的最大公约数
        }
        for k1, v1 := range m {
                for k2, v2 := range m {
                        // 核心：a*b是k的倍数 => gcd(a,k)*gcd(b,k)是k的倍数
                        if k1*k2%k == 0 {
                                res = res + int64(v1)*int64(v2) // 组合数
                        }
                }
        }
        for i := 0; i < len(nums); i++ {
                if nums[i]*nums[i]%k == 0 { // 本身x本身：不满足要求
                        res--
                }
        }
        return res / 2 // 要满足有序要求
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
```

(续下页)

(接上页)

```
        }
        return b
}

# 2
func countPairs(nums []int, k int) int64 {
        res := int64(0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[gcd(k, nums[i])]++ // k和nums[i]的最大公约数
        }
        for k1, v1 := range m {
                for k2, v2 := range m {
                        // 核心: a*b是k的倍数 => gcd(a,k)*gcd(b,k)是k的倍数
                        if k1*k2%k == 0 {
                                if k1 < k2 {
                                        res = res + int64(v1)*int64(v2)
                                } else if k1 == k2 {
                                        res = res + int64(v1)*int64(v1-1)/2
                                }
                        }
                }
        }
        return res
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}
```

66.11 2188. 完成比赛的最少时间 (1)

- 题目

```
给你一个下标从 0开始的二维整数数组tires, 其中tires[i] = [fi,␣
↪ri]表示第i种轮胎如果连续使用,
第x圈需要耗时fi * ri(x-1)秒。
比方说, 如果fi = 3且ri = 2, 且一直使用这种类型的同一条轮胎,
那么该轮胎完成第1圈赛道耗时 3秒, 完成第 2圈耗时3 * 2 = 6秒, 完成第 3圈耗时3 * 22 =␣
```

(续下页)

(接上页)

```
↪12秒，依次类推。
同时给你一个整数changeTime和一个整数numLaps。
比赛总共包含numLaps圈，你可以选择 任意一种轮胎开始比赛。每一种轮胎都有 无数条。
每一圈后，你可以选择耗费 changeTime秒 换成任意一种轮胎（也可以换成当前种类的新轮胎）。
请你返回完成比赛需要耗费的 最少时间。
示例 1：输入：tires = [[2,3],[3,4]], changeTime = 5, numLaps = 4 输出：21
解释：第 1 圈：使用轮胎 0 ，耗时 2 秒。
第 2 圈：继续使用轮胎 0 ，耗时 2 * 3 = 6 秒。
第 3 圈：耗费 5 秒换一条新的轮胎 0 ，然后耗时 2 秒完成这一圈。
第 4 圈：继续使用轮胎 0 ，耗时 2 * 3 = 6 秒。
总耗时 = 2 + 6 + 5 + 2 + 6 = 21 秒。
完成比赛的最少时间为 21 秒。
示例 2：输入：tires = [[1,10],[2,2],[3,4]], changeTime = 6, numLaps = 5 输出：25
解释：第 1 圈：使用轮胎 1 ，耗时 2 秒。
第 2 圈：继续使用轮胎 1 ，耗时 2 * 2 = 4 秒。
第 3 圈：耗时 6 秒换一条新的轮胎 1 ，然后耗时 2 秒完成这一圈。
第 4 圈：继续使用轮胎 1 ，耗时 2 * 2 = 4 秒。
第 5 圈：耗时 6 秒换成轮胎 0 ，然后耗时 1 秒完成这一圈。
总耗时 = 2 + 4 + 6 + 2 + 4 + 6 + 1 = 25 秒。
完成比赛的最少时间为 25 秒。
提示：1 <= tires.length <= 105
tires[i].length == 2
1 <= fi, changeTime <= 105
2 <= ri <= 105
1 <= numLaps <= 1000
```

- 解题思路

```
func minimumFinishTime(tires [][]int, changeTime int, numLaps int) int {
        minArr := make([]int, 20) // 用1个轮胎跑i圈的最小花费
        for i := 0; i < 20; i++ {
                minArr[i] = math.MaxInt32 / 10
        }
        for i := 0; i < len(tires); i++ {
                f, r := tires[i][0], tires[i][1]
                first := f + changeTime // 第一次用该胎的耗费
                sum := first
                for j := 1; f <= first; j++ {
                        minArr[j] = min(minArr[j], sum)
                        f = f * r
                        sum = sum + f
                }
        }
        dp := make([]int, numLaps+1) // dp[i] => 表示跑i圈的最小花费
```

(续下页)

(接上页)

```
        for i := 0; i <= numLaps; i++ {
                dp[i] = math.MaxInt32 / 10
        }
        dp[0] = 0
        for i := 1; i <= numLaps; i++ {
                for j := 1; j <= 19 && j <= i; j++ { // 遍历跑i圈换胎
                        dp[i] = min(dp[i], dp[i-j]+minArr[j])
                }
        }
        return dp[numLaps] - changeTime // 减去最后一次换胎时间
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

66.12 2197. 替换数组中的非互质数 (1)

- 题目

```
给你一个整数数组 nums 。请你对数组执行下述操作：
从 nums 中找出 任意 两个 相邻 的 非互质 数。
如果不存在这样的数，终止 这一过程。
否则，删除这两个数，并 替换 为它们的 最小公倍数 (Least Common Multiple，LCM) 。
只要还能找出两个相邻的非互质数就继续 重复 这一过程。
返回修改后得到的 最终 数组。可以证明的是，以 任意␣
↪顺序替换相邻的非互质数都可以得到相同的结果。
生成的测试用例可以保证最终数组中的值 小于或者等于 108 。
两个数字 x 和 y 满足 非互质数 的条件是：GCD(x, y) > 1 ，其中 GCD(x, y) 是 x 和 y 的␣
↪最大公约数 。
示例 1 ：输入：nums = [6,4,3,2,7,6,2] 输出：[12,7,6]
解释：- (6, 4) 是一组非互质数，且 LCM(6, 4) = 12 。得到 nums = [12,3,2,7,6,2] 。
- (12, 3) 是一组非互质数，且 LCM(12, 3) = 12 。得到 nums = [12,2,7,6,2] 。
- (12, 2) 是一组非互质数，且 LCM(12, 2) = 12 。得到 nums = [12,7,6,2] 。
- (6, 2) 是一组非互质数，且 LCM(6, 2) = 6 。得到 nums = [12,7,6] 。
现在，nums 中不存在相邻的非互质数。
因此，修改后得到的最终数组是 [12,7,6] 。
注意，存在其他方法可以获得相同的最终数组。
示例 2 ：输入：nums = [2,2,1,1,3,3,3] 输出：[2,1,1,3]
```

(续下页)

(接上页)

```
解释：- (3, 3) 是一组非互质数，且 LCM(3, 3) = 3 。得到 nums = [2,2,1,1,3,3] 。
- (3, 3) 是一组非互质数，且 LCM(3, 3) = 3 。得到 nums = [2,2,1,1,3] 。
- (2, 2) 是一组非互质数，且 LCM(2, 2) = 2 。得到 nums = [2,1,1,3] 。
现在，nums 中不存在相邻的非互质数。
因此，修改后得到的最终数组是 [2,1,1,3] 。
注意，存在其他方法可以获得相同的最终数组。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 105
生成的测试用例可以保证最终数组中的值 小于或者等于 108 。
```

- 解题思路

```
func replaceNonCoprimes(nums []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                v := nums[i]
                for len(res) > 0 {
                        if gcd(v, res[len(res)-1]) > 1 {
                                v = lcm(v, res[len(res)-1])
                                res = res[:len(res)-1]
                        } else {
                                break
                        }
                }
                res = append(res, v)
        }
        return res
}

func lcm(x, y int) int {
        return x * y / gcd(x, y)
}

func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}
```

CHAPTER 67

2201-2300-Easy

67.1 2206. 将数组划分成相等数对 (2)

- 题目

```
给你一个整数数组nums，它包含2 * n个整数。
你需要将nums 划分成n个数对，满足：
每个元素 只属于一个 数对。
同一数对中的元素 相等。
如果可以将 nums划分成 n个数对，请你返回 true，否则返回 false。
示例 1：输入：nums = [3,2,3,2,2,2] 输出：true
解释：nums中总共有 6 个元素，所以它们应该被划分成 6 / 2 = 3 个数对。
nums 可以划分成 (2, 2) ，(3, 3) 和 (2, 2) ，满足所有要求。
示例 2：输入：nums = [1,2,3,4] 输出：false
解释：无法将 nums 划分成 4 / 2 = 2 个数对且满足所有要求。
提示：nums.length == 2 * n
1 <= n <= 500
1 <= nums[i] <= 500
```

- 解题思路

```
func divideArray(nums []int) bool {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
```

(续下页)

(接上页)

```
        }
        for _, v := range m {
                if v%2 == 1 {
                        return false
                }
        }
        return true
}

# 2
func divideArray(nums []int) bool {
        sort.Ints(nums)
        for i := 1; i < len(nums); i = i + 2 {
                if nums[i] != nums[i-1] {
                        return false
                }
        }
        return true
}
```

67.2 2210. 统计数组中峰和谷的数量 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums 。如果两侧距 i 最近的不相等邻居的值均小于␣
↪nums[i] ，
则下标 i 是 nums 中，某个峰的一部分。类似地，如果两侧距 i 最近的不相等邻居的值均大于␣
↪nums[i] ，
则下标 i 是 nums 中某个谷的一部分。
对于相邻下标i 和 j ，如果nums[i] == nums[j] ， 则认为这两下标属于 同一个 峰或谷。
注意，要使某个下标所做峰或谷的一部分，那么它左右两侧必须 都 存在不相等邻居。
返回 nums 中峰和谷的数量。
示例 1: 输入: nums = [2,4,1,1,6,5] 输出: 3
解释: 在下标 0 : 由于 2 的左侧不存在不相等邻居，所以下标 0 既不是峰也不是谷。
在下标 1 : 4 的最近不相等邻居是 2 和 1 。由于 4 > 2 且 4 > 1 ，下标 1 是一个峰。
在下标 2 : 1 的最近不相等邻居是 4 和 6 。由于 1 < 4 且 1 < 6 ，下标 2 是一个谷。
在下标 3 : 1 的最近不相等邻居是 4 和 6 。由于 1 < 4 且 1 < 6 ，
下标 3 符合谷的定义，但需要注意它和下标 2 是同一个谷的一部分。
在下标 4 : 6 的最近不相等邻居是 1 和 5 。由于 6 > 1 且 6 > 5 ，下标 4 是一个峰。
在下标 5 : 由于 5 的右侧不存在不相等邻居，所以下标 5 既不是峰也不是谷。
共有 3 个峰和谷，所以返回 3 。
示例 2: 输入: nums = [6,6,5,5,4,1] 输出: 0
```

(续下页)

(接上页)

```
解释：在下标 0 ：由于 6 的左侧不存在不相等邻居，所以下标 0 既不是峰也不是谷。
在下标 1 ：由于 6 的左侧不存在不相等邻居，所以下标 1 既不是峰也不是谷。
在下标 2 ：5 的最近不相等邻居是 6 和 4 。由于 5 < 6 且 5 > 4 ，下标 2␣
↪既不是峰也不是谷。
在下标 3 ：5 的最近不相等邻居是 6 和 4 。由于 5 < 6 且 5 > 4 ，下标 3␣
↪既不是峰也不是谷。
在下标 4 ：4 的最近不相等邻居是 5 和 1 。由于 4 < 5 且 4 > 1 ，下标 4␣
↪既不是峰也不是谷。
在下标 5 ：由于 1 的右侧不存在不相等邻居，所以下标 5 既不是峰也不是谷。
共有 0 个峰和谷，所以返回 0 。
提示：3 <= nums.length <= 100
1 <= nums[i] <= 100
```

- 解题思路

```
func countHillValley(nums []int) int {
        res := 0
        n := len(nums)
        for i := 1; i < n-1; i++ {
                if nums[i] == nums[i-1] {
                        continue
                }
                a, b := -1, -1
                for j := i - 1; j >= 0; j-- {
                        if nums[i] != nums[j] {
                                a = nums[j]
                                break
                        }
                }
                for j := i + 1; j < n; j++ {
                        if nums[i] != nums[j] {
                                b = nums[j]
                                break
                        }
                }
                if a == -1 || b == -1 {
                        continue
                }
                if (a < nums[i] && b < nums[i]) || (a > nums[i] && b > nums[i]) {
                        res++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func countHillValley(nums []int) int {
        res := 0
        n := len(nums)
        flag := 0 // 1: 递增, 2: 递减
        for i := 1; i < n; i++ {
                if nums[i-1] < nums[i] {
                        if flag == 2 {
                                res++
                        }
                        flag = 1
                } else if nums[i-1] > nums[i] {
                        if flag == 1 {
                                res++
                        }
                        flag = 2
                }
        }
        return res
}
```

67.3 2215. 找出两数组的不同 (1)

- 题目

```
给你两个下标从 0 开始的整数数组 nums1 和 nums2 ，请你返回一个长度为 2 的列表 answer
↪, 其中：
answer[0] 是 nums1 中所有 不 存在于 nums2 中的 不同 整数组成的列表。
answer[1] 是 nums2 中所有 不 存在于 nums1 中的 不同 整数组成的列表。
注意：列表中的整数可以按 任意 顺序返回。
示例 1：输入：nums1 = [1,2,3], nums2 = [2,4,6] 输出：[[1,3],[4,6]]
解释：对于 nums1 ，nums1[1] = 2 出现在 nums2 中下标 0 处，然而 nums1[0] = 1 和
↪nums1[2] = 3 没有出现在 nums2 中。
因此，answer[0] = [1,3]。
对于 nums2 ，nums2[0] = 2 出现在 nums1 中下标 1 处，然而 nums2[1] = 4 和 nums2[2] = 6
↪没有出现在 nums2 中。
因此，answer[1] = [4,6]。
示例 2：输入：nums1 = [1,2,3,3], nums2 = [1,1,2,2] 输出：[[3],[]]
解释：对于 nums1 ，nums1[2] 和 nums1[3] 没有出现在 nums2 中。
由于 nums1[2] == nums1[3] ，二者的值只需要在 answer[0] 中出现一次，故 answer[0] =
↪[3]。
```

(续下页)

(接上页)

```
nums2 中的每个整数都在 nums1 中出现，因此，answer[1] = [] 。
提示：1 <= nums1.length, nums2.length <= 1000
-1000 <= nums1[i], nums2[i] <= 1000
```

- 解题思路

```
func findDifference(nums1 []int, nums2 []int) [][]int {
        m1, m2 := make(map[int]bool), make(map[int]bool)
        a, b := make([]int, 0), make([]int, 0)
        for i := 0; i < len(nums1); i++ {
                m1[nums1[i]] = true
        }
        for i := 0; i < len(nums2); i++ {
                m2[nums2[i]] = true
        }
        for k := range m1 {
                if m2[k] == false {
                        a = append(a, k)
                }
        }
        for k := range m2 {
                if m1[k] == false {
                        b = append(b, k)
                }
        }
        return [][]int{a, b}
}
```

67.4 2220. 转换数字的最少位翻转次数 (3)

- 题目

```
一次 位翻转定义为将数字x二进制中的一个位进行 翻转操作，即将0变成1，或者将1变成0。
比方说，x = 7，二进制表示为111，我们可以选择任意一个位（包含没有显示的前导 0␣
↪）并进行翻转。
比方说我们可以翻转最右边一位得到110，或者翻转右边起第二位得到101，
或者翻转右边起第五位（这一位是前导 0 ）得到10111等等。
给你两个整数start 和goal，请你返回将start转变成goal的最少位翻转次数。
示例 1：输入：start = 10, goal = 7 输出：3
解释：10 和 7 的二进制表示分别为 1010 和 0111 。我们可以通过 3 步将 10 转变成 7 ：
- 翻转右边起第一位得到：1010 -> 1011 。
- 翻转右边起第三位：1011 -> 1111 。
```

(续下页)

(接上页)

```
- 翻转右边起第四位：1111 -> 0111 。
我们无法在 3 步内将 10 转变成 7 。所以我们返回 3 。
示例 2：输入：start = 3, goal = 4 输出：3
解释：3 和 4 的二进制表示分别为 011 和 100 。我们可以通过 3 步将 3 转变成 4 ：
- 翻转右边起第一位：011 -> 010 。
- 翻转右边起第二位：010 -> 000 。
- 翻转右边起第三位：000 -> 100 。
我们无法在 3 步内将 3 变成 4 。所以我们返回 3 。
提示：0 <= start, goal <= 109
```

- 解题思路

```
func minBitFlips(start int, goal int) int {
        res := 0
        a := fmt.Sprintf("%032b", start)
        b := fmt.Sprintf("%032b", goal)
        for i := 0; i < 32; i++ {
                if a[i] != b[i] {
                        res++
                }
        }
        return res
}

# 2
func minBitFlips(start int, goal int) int {
        res := 0
        temp := start ^ goal
        for ; temp > 0; temp = temp / 2 {
                res = res + temp%2
        }
        return res
}

# 3
func minBitFlips(start int, goal int) int {
        return bits.OnesCount(uint(start ^ goal))
}
```

67.5 2224. 转化时间需要的最少操作数 (1)

- 题目

```
给你两个字符串 current 和 correct ，表示两个 24 小时制时间 。
24 小时制时间 按 "HH:MM" 进行格式化，其中 HH 在 00 和 23 之间，而 MM 在 00 和 59
↪之间。
最早的 24 小时制时间为 00:00 ，最晚的是 23:59 。
在一步操作中，你可以将 current 这个时间增加 1、5、15 或 60 分钟。你可以执行这一操作
↪任意 次数。
返回将current 转化为 correct 需要的 最少操作数 。
示例 1：输入：current = "02:30", correct = "04:35" 输出：3
解释：可以按下述 3 步操作将 current 转换为 correct ：
- 为 current 加 60 分钟，current 变为 "03:30" 。
- 为 current 加 60 分钟，current 变为 "04:30" 。
- 为 current 加 5 分钟，current 变为 "04:35" 。
可以证明，无法用少于 3 步操作将 current 转化为 correct 。
示例 2：输入：current = "11:00", correct = "11:01" 输出：1
解释：只需要为 current 加一分钟，所以最小操作数是 1 。
提示：current 和 correct 都符合 "HH:MM" 格式
current <= correct
```

- 解题思路

```
func convertTime(current string, correct string) int {
        diff := parseTime(correct) - parseTime(current)
        res := 0
        arr := []int{60, 15, 5, 1}
        for i := 0; i < len(arr); i++ {
                res = res + diff/arr[i]
                diff = diff % arr[i]
        }
        return res
}

func parseTime(s string) int {
        a, b := int(s[0]-'0'), int(s[1]-'0')
        c, d := int(s[3]-'0'), int(s[4]-'0')
        return (a*10+b)*60 + (c*10 + d)
}
```

67.6 2231. 按奇偶性交换后的最大数字 (1)

- 题目

```
给你一个正整数 num 。你可以交换 num 中 奇偶性␣
↪相同的任意两位数字（即，都是奇数或者偶数）。
返回交换 任意 次之后 num 的 最大 可能值。
示例 1：输入：num = 1234 输出：3412
解释：交换数字 3 和数字 1 ，结果得到 3214 。
交换数字 2 和数字 4 ，结果得到 3412 。
注意，可能存在其他交换序列，但是可以证明 3412 是最大可能值。
注意，不能交换数字 4 和数字 1 ，因为它们奇偶性不同。
示例 2：输入：num = 65875 输出：87655
解释：交换数字 8 和数字 6 ，结果得到 85675 。
交换数字 5 和数字 7 ，结果得到 87655 。
注意，可能存在其他交换序列，但是可以证明 87655 是最大可能值。
提示：1 <= num <= 109
```

- 解题思路

```
func largestInteger(num int) int {
        value := []byte(strconv.Itoa(num))
        n := len(value)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if int(value[i]-'0')%2 == int(value[j]-'0')%2 &&
                                value[i] < value[j] {
                                value[i], value[j] = value[j], value[i]
                        }
                }
        }
        res, _ := strconv.Atoi(string(value))
        return res
}
```

67.7 2235. 两整数相加 (1)

- 题目

```
给你两个整数num1 和 num2，返回这两个整数的和。
示例 1：输入：num1 = 12, num2 = 5 输出：17
解释：num1 是 12，num2 是 5 ，它们的和是 12 + 5 = 17 ，因此返回 17 。
示例 2：输入：num1 = -10, num2 = 4 输出：-6
```

(续下页)

(接上页)

```
解释：num1 + num2 = -6 ，因此返回 -6 。
提示：-100 <= num1, num2 <= 100
```

- 解题思路

```
func sum(num1 int, num2 int) int {
        return num1 + num2
}
```

67.8 2236. 判断根结点是否等于子结点之和 (1)

- 题目

```
给你一个 二叉树 的根结点root，该二叉树由恰好3个结点组成：根结点、左子结点和右子结点。
如果根结点值等于两个子结点值之和，返回true，否则返回false 。
示例 1：输入：root = [10,4,6] 输出：true
解释：根结点、左子结点和右子结点的值分别是 10 、4 和 6 。
由于 10 等于 4 + 6 ，因此返回 true 。
示例 2：输入：root = [5,3,1] 输出：false
解释：根结点、左子结点和右子结点的值分别是 5 、3 和 1 。
由于 5 不等于 3 + 1 ，因此返回 false 。
提示：树只包含根结点、左子结点和右子结点
-100 <= Node.val <= 100
```

- 解题思路

```
func checkTree(root *TreeNode) bool {
        return root.Val == root.Left.Val+root.Right.Val
}
```

67.9 2239. 找到最接近 0 的数字 (1)

- 题目

```
给你一个长度为 n的整数数组nums，请你返回 nums中最
↪接近0的数字。如果有多个答案，请你返回它们中的 最大值。
示例 1：输入：nums = [-4,-2,1,4,8] 输出：1
解释：-4 到 0 的距离为 |-4| = 4 。
-2 到 0 的距离为 |-2| = 2 。
1 到 0 的距离为 |1| = 1 。
```

(续下页)

(接上页)

```
4 到 0 的距离为 |4| = 4 。
8 到 0 的距离为 |8| = 8 。
所以，数组中距离 0 最近的数字为 1 。
示例 2：输入：nums = [2,-1,1] 输出：1
解释：1 和 -1 都是距离 0 最近的数字，所以返回较大值 1 。
提示：1 <= n <= 1000
-105 <= nums[i] <= 105
```

- 解题思路

```
func findClosestNumber(nums []int) int {
        res := math.MaxInt32 / 10
        for i := 0; i < len(nums); i++ {
                if abs(nums[i]) < abs(res) {
                        res = nums[i]
                } else if abs(nums[i]) == abs(res) {
                        res = max(res, nums[i])
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

67.10 2243. 计算字符串的数字和 (1)

- 题目

```
给你一个由若干数字（0 - 9）组成的字符串 s ，和一个整数。
如果 s 的长度大于 k ，则可以执行一轮操作。在一轮操作中，需要完成以下工作：
将 s 拆分 成长度为 k 的若干 连续数字组 ，使得前 k 个字符都分在第一组，接下来的 k
↪个字符都分在第二组，依此类推。
注意，最后一个数字组的长度可以小于 k 。
用表示每个数字组中所有数字之和的字符串来 替换 对应的数字组。例如，"346" 会替换为 "13"
↪，因为 3 + 4 + 6 = 13 。
合并 所有组以形成一个新字符串。如果新字符串的长度大于 k 则重复第一步。
返回在完成所有轮操作后的 s 。
示例 1：输入：s = "11111222223", k = 3 输出："135"
解释：- 第一轮，将 s 分成："111"、"112"、"222" 和 "23" 。
  接着，计算每一组的数字和：1 + 1 + 1 = 3、1 + 1 + 2 = 4、2 + 2 + 2 = 6 和 2 + 3 = 5
↪。
 这样，s 在第一轮之后变成 "3" + "4" + "6" + "5" = "3465" 。
- 第二轮，将 s 分成："346" 和 "5" 。
 接着，计算每一组的数字和：3 + 4 + 6 = 13 、5 = 5 。
 这样，s 在第二轮之后变成 "13" + "5" = "135" 。
现在，s.length <= k ，所以返回 "135" 作为答案。
示例 2：输入：s = "00000000", k = 3输出："000"
解释：将 "000", "000", and "00".
接着，计算每一组的数字和：0 + 0 + 0 = 0 、0 + 0 + 0 = 0 和 0 + 0 = 0 。
s 变为 "0" + "0" + "0" = "000" ，其长度等于 k ，所以返回 "000" 。
提示：1 <= s.length <= 100
2 <= k <= 100
s 仅由数字（0 - 9）组成。
```

- 解题思路

```
func digitSum(s string, k int) string {
        arr := []byte(s)
        for len(arr) > k {
                temp := make([]byte, 0)
                for i := 0; i < len(arr); i = i + k {
                        sum := 0
                        for j := i; j < i+k && j < len(arr); j++ {
                                sum = sum + int(arr[j]-'0')
                        }
                        temp = append(temp, []byte(strconv.Itoa(sum))...)
                }
                arr = temp
```

(续下页)

(接上页)

```
        }
        return string(arr)
}
```

67.11 2248. 多个数组求交集 (2)

- 题目

```
给你一个二维整数数组 nums ，其中 nums[i] 是由 不同 正整数组成的一个非空数组，
按 升序排列 返回一个数组，数组中的每个元素在 nums所有数组 中都出现过。
示例 1：输入：nums = [[3,1,2,4,5],[1,2,3,4],[3,4,5,6]] 输出：[3,4]
解释：nums[0] = [3,1,2,4,5]，nums[1] = [1,2,3,4]，nums[2] = [3,4,5,6]，
在 nums 中每个数组中都出现的数字是 3 和 4 ，所以返回 [3,4] 。
示例 2：输入：nums = [[1,2,3],[4,5,6]] 输出：[]
解释：不存在同时出现在 nums[0] 和 nums[1] 的整数，所以返回一个空列表 [] 。
提示：1 <= nums.length <= 1000
1 <= sum(nums[i].length) <= 1000
1 <= nums[i][j] <= 1000
nums[i] 中的所有值 互不相同
```

- 解题思路

```
func intersection(nums [][]int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                for j := 0; j < len(nums[i]); j++ {
                        m[nums[i][j]]++
                }
        }
        res := make([]int, 0)
        for k, v := range m {
                if v == len(nums) {
                        res = append(res, k)
                }
        }
        sort.Ints(res)
        return res
}

# 2
func intersection(nums [][]int) []int {
        arr := make([]int, 1001)
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                for j := 0; j < len(nums[i]); j++ {
                        arr[nums[i][j]]++
                }
        }
        res := make([]int, 0)
        for i := 1; i <= 1000; i++ {
                if arr[i] == len(nums) {
                        res = append(res, i)
                }
        }
        return res
}
```

67.12 2255. 统计是给定字符串前缀的字符串数目 (1)

- 题目

```
给你一个字符串数组words和一个字符串s，其中words[i]和s只包含小写英文字母。
请你返回 words中是字符串 s前缀的 字符串数目。
一个字符串的␣
↪前缀是出现在字符串开头的子字符串。子字符串是一个字符串中的连续一段字符序列。
示例 1：输入：words = ["a","b","c","ab","bc","abc"], s = "abc" 输出：3
解释：words 中是 s = "abc" 前缀的字符串为：
"a" ，"ab" 和 "abc" 。
所以 words 中是字符串 s 前缀的字符串数目为 3 。
示例 2：输入：words = ["a","a"], s = "aa" 输出：2
解释：两个字符串都是 s 的前缀。
注意，相同的字符串可能在 words 中出现多次，它们应该被计数多次。
提示：1 <= words.length <= 1000
1 <= words[i].length, s.length <= 10
words[i]和s只包含小写英文字母。
```

- 解题思路

```
func countPrefixes(words []string, s string) int {
        res := 0
        for _, v := range words {
                if strings.HasPrefix(s, v) == true {
                        res++
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

67.13 2259. 移除指定数字得到的最大结果 (2)

- 题目

```
给你一个表示某个正整数的字符串 number 和一个字符 digit 。
从 number 中 恰好 移除 一个 等于digit 的字符后，找出并返回按 十进制 表示 最大的结果字符串。
生成的测试用例满足 digit 在 number 中出现至少一次。
示例 1：输入：number = "123", digit = "3" 输出："12"
解释："123" 中只有一个 '3' ，在移除 '3' 之后，结果为 "12" 。
示例 2：输入：number = "1231", digit = "1" 输出："231"
解释：可以移除第一个 '1' 得到 "231" 或者移除第二个 '1' 得到 "123" 。
由于 231 > 123 ，返回 "231" 。
示例 3：输入：number = "551", digit = "5" 输出："51"
解释：可以从 "551" 中移除第一个或者第二个 '5' 。
两种方案的结果都是 "51" 。
提示：2 <= number.length <= 100
number 由数字 '1' 到 '9' 组成
digit 是 '1' 到 '9' 中的一个数字
digit 在 number 中出现至少一次
```

- 解题思路

```
func removeDigit(number string, digit byte) string {
        res := -1
        for i := 0; i < len(number); i++ {
                if number[i] == digit {
                        res = i
                        if i < len(number)-1 && number[i] < number[i+1] { //贪心：找到第一个大于后面相邻的数
                                break
                        }
                }
        }
        return number[:res] + number[res+1:]
}

# 2
func removeDigit(number string, digit byte) string {
```

(续下页)

(接上页)

```
        res := ""
        for i := 0; i < len(number); i++ {
                if number[i] == digit {
                        v := number[:i] + number[i+1:]
                        if v > res {
                                res = v
                        }
                }
        }
        return res
}
```

67.14 2264. 字符串中最大的 3 位相同数字 (2)

- 题目

```
给你一个字符串 num ，表示一个大整数。如果一个整数满足下述所有条件，则认为该整数是一个优质整数 ：
该整数是 num 的一个长度为 3 的 子字符串 。
该整数由唯一一个数字重复 3 次组成。
以字符串形式返回 最大的优质整数 。如果不存在满足要求的整数，则返回一个空字符串 "" 。
注意：子字符串 是字符串中的一个连续字符序列。
num 或优质整数中可能存在 前导零 。
示例 1：输入：num = "6777133339" 输出："777"
解释：num 中存在两个优质整数："777" 和 "333" 。
"777" 是最大的那个，所以返回 "777" 。
示例 2：输入：num = "2300019" 输出："000"
解释："000" 是唯一一个优质整数。
示例 3：输入：num = "42352338" 输出：""
解释：不存在长度为 3 且仅由一个唯一数字组成的整数。因此，不存在优质整数。
提示：3 <= num.length <= 1000
num 仅由数字（0 - 9）组成
```

- 解题思路

```
func largestGoodInteger(num string) string {
        for i := '9'; i >= '0'; i-- {
                target := strings.Repeat(string(i), 3)
                if strings.Contains(num, target) == true {
                        return target
                }
        }
```

(续下页)

(接上页)

```
        return ""
}

# 2
func largestGoodInteger(num string) string {
        res := ""
        for i := 2; i < len(num); i++ {
                if num[i] == num[i-1] && num[i-1] == num[i-2] && num[i-2:i+1] > res {
                        res = num[i-2 : i+1]
                }
        }
        return res
}
```

67.15 2269. 找到一个数字的 K 美丽值 (2)

- 题目

```
一个整数 num的k美丽值定义为num中符合以下条件的子字符串数目:
子字符串长度为k。
子字符串能整除 num 。
给你整数num 和k，请你返回num的 k 美丽值。
注意: 允许有前缀0。
0不能整除任何值。
一个 子字符串是一个字符串里的连续一段字符序列。
示例 1: 输入: num = 240, k = 2 输出: 2
解释: 以下是 num 里长度为 k 的子字符串:
- "240" 中的 "24" : 24 能整除 240 。
- "240" 中的 "40" : 40 能整除 240 。
所以，k 美丽值为 2 。
示例 2: 输入: num = 430043, k = 2 输出: 2
解释: 以下是 num 里长度为 k 的子字符串:
- "430043" 中的 "43" : 43 能整除 430043 。
- "430043" 中的 "30" : 30 不能整除 430043 。
- "430043" 中的 "00" : 0 不能整除 430043 。
- "430043" 中的 "04" : 4 不能整除 430043 。
- "430043" 中的 "43" : 43 能整除 430043 。
所以，k 美丽值为 2 。
提示: 1 <= num <= 109
1 <= k <= num.length (将num视为字符串)
```

- 解题思路

```
func divisorSubstrings(num int, k int) int {
        res := 0
        str := strconv.Itoa(num)
        for i := k; i <= len(str); i++ {
                v, _ := strconv.Atoi(str[i-k : i])
                if v > 0 && num%v == 0 {
                        res++
                }
        }
        return res
}

# 2
func divisorSubstrings(num int, k int) int {
        res := 0
        target := int(math.Pow10(k))
        for v := num; v >= target/10; v = v / 10 {
                value := v % target
                if value > 0 && num%value == 0 {
                        res++
                }
        }
        return res
}
```

67.16 2273. 移除字母异位词后的结果数组 (2)

- 题目

```
给你一个下标从 0 开始的字符串 words ，其中 words[i] 由小写英文字符组成。
在一步操作中，需要选出任一下标 i ，从 words 中 删除 words[i] 。其中下标 i␣
↪需要同时满足下述两个条件：
0 < i < words.length
words[i - 1] 和 words[i] 是 字母异位词 。
只要可以选出满足条件的下标，就一直执行这个操作。
在执行所有操作后，返回 words␣
↪。可以证明，按任意顺序为每步操作选择下标都会得到相同的结果。
字母异位词␣
↪是由重新排列源单词的字母得到的一个新单词，所有源单词中的字母通常恰好只用一次。
例如，"dacb" 是 "abdc" 的一个字母异位词。
示例 1：输入：words = ["abba","baba","bbaa","cd","cd"] 输出：["abba","cd"]
解释：获取结果数组的方法之一是执行下述步骤：
```

(续下页)

(接上页)

```
- 由于 words[2] = "bbaa" 和 words[1] = "baba" 是字母异位词，选择下标 2 并删除␣
↪words[2] 。
  现在 words = ["abba","baba","cd","cd"] 。
- 由于 words[1] = "baba" 和 words[0] = "abba" 是字母异位词，选择下标 1 并删除␣
↪words[1] 。
  现在 words = ["abba","cd","cd"] 。
- 由于 words[2] = "cd" 和 words[1] = "cd" 是字母异位词，选择下标 2 并删除 words[2] 。
  现在 words = ["abba","cd"] 。
无法再执行任何操作，所以 ["abba","cd"] 是最终答案。
示例 2：输入：words = ["a","b","c","d","e"] 输出：["a","b","c","d","e"]
解释：words 中不存在互为字母异位词的两个相邻字符串，所以无需执行任何操作。
提示：1 <= words.length <= 100
1 <= words[i].length <= 10
words[i] 由小写英文字母组成
```

- 解题思路

```
func removeAnagrams(words []string) []string {
        flag := true
        for flag == true {
                for i := len(words) - 1; i >= 0; i-- {
                        if i == 0 {
                                flag = false
                                break
                        }
                        if judge(words[i], words[i-1]) == true {
                                words = append(words[:i], words[i+1:]...)
                                flag = true
                                break
                        }
                }
        }
        return words
}

func judge(a, b string) bool {
        m := make(map[rune]int)
        for _, v := range a {
                m[v]++
        }
        for _, v := range b {
                m[v]--
        }
        for _, v := range m {
```

(续下页)

(接上页)

```
                if v != 0 {
                        return false
                }
        }
        return true
}

# 2
func removeAnagrams(words []string) []string {
        res := []string{words[0]}
        for i := 1; i < len(words); i++ {
                if judge(res[len(res)-1], words[i]) == false {
                        res = append(res, words[i])
                }
        }
        return res
}

func judge(a, b string) bool {
        arr := [256]int{}
        for _, v := range a {
                arr[v]++
        }
        for _, v := range b {
                arr[v]--
        }
        return arr == [256]int{}
}
```

67.17 2278. 字母在字符串中的百分比 (2)

- 题目

```
给你一个字符串 s 和一个字符 letter ，返回在 s 中等于letter字符所占的 百分比␣
↪，向下取整到最接近的百分比。
示例 1：输入：s = "foobar", letter = "o" 输出：33
解释：等于字母 'o' 的字符在 s 中占到的百分比是 2 / 6 * 100% = 33%␣
↪，向下取整，所以返回 33 。
示例 2：输入：s = "jjjj", letter = "k" 输出：0
解释：等于字母 'k' 的字符在 s 中占到的百分比是 0% ，所以返回 0 。
提示：1 <= s.length <= 100
s 由小写英文字母组成
```

(续下页)

(接上页)

```
letter 是一个小写英文字母
```

- 解题思路

```
func percentageLetter(s string, letter byte) int {
        return 100 * strings.Count(s, string(letter)) / len(s)
}

# 2
func percentageLetter(s string, letter byte) int {
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == letter {
                        count++
                }
        }
        return 100 * count / len(s)
}
```

67.18 2283. 判断一个数的数字计数是否等于数位的值 (1)

- 题目

```
给你一个下标从 0开始长度为 n的字符串num，它只包含数字。
如果对于 每个0 <= i < n的下标i，都满足数位i在␣
↪num中出现了num[i]次，那么请你返回true，否则返回false。
示例 1：输入：num = "1210" 输出：true
解释：num[0] = '1' 。数字 0 在 num 中出现了一次。
num[1] = '2' 。数字 1 在 num 中出现了两次。
num[2] = '1' 。数字 2 在 num 中出现了一次。
num[3] = '0' 。数字 3 在 num 中出现了零次。
"1210" 满足题目要求条件，所以返回 true 。
示例 2：输入：num = "030" 输出：false
解释：num[0] = '0' 。数字 0 应该出现 0 次，但是在 num 中出现了一次。
num[1] = '3' 。数字 1 应该出现 3 次，但是在 num 中出现了零次。
num[2] = '0' 。数字 2 在 num 中出现了 0 次。
下标 0 和 1 都违反了题目要求，所以返回 false 。
提示：n == num.length
1 <= n <= 10
num只包含数字。
```

- 解题思路

```
func digitCount(num string) bool {
        m := make(map[int]int)
        for i := 0; i < len(num); i++ {
                m[int(num[i]-'0')]++
        }
        for i := 0; i < len(num); i++ {
                if m[i] != int(num[i]-'0') {
                        return false
                }
        }
        return true
}
```

67.19 2287. 重排字符形成目标字符串 (1)

- 题目

```
给你两个下标从 0 开始的字符串 s 和 target 。你可以从 s␣
↪取出一些字符并将其重排，得到若干新的字符串。
从 s 中取出字符并重新排列，返回可以形成 target 的 最大 副本数。
示例 1：输入：s = "ilovecodingonleetcode", target = "code" 输出：2
解释：对于 "code" 的第 1 个副本，选取下标为 4 、5 、6 和 7 的字符。
对于 "code" 的第 2 个副本，选取下标为 17 、18 、19 和 20 的字符。
形成的字符串分别是 "ecod" 和 "code" ，都可以重排为 "code" 。
可以形成最多 2 个 "code" 的副本，所以返回 2 。
示例 2：输入：s = "abcba", target = "abc" 输出：1
解释：选取下标为 0 、1 和 2 的字符，可以形成 "abc" 的 1 个副本。
可以形成最多 1 个 "abc" 的副本，所以返回 1 。
注意，尽管下标 3 和 4 分别有额外的 'a' 和 'b' ，但不能重用下标 2 处的 'c'␣
↪，所以无法形成 "abc" 的第 2 个副本。
示例 3：输入：s = "abbaccaddaeea", target = "aaaaa" 输出：1
解释：选取下标为 0 、3 、6 、9 和 12 的字符，可以形成 "aaaaa" 的 1 个副本。
可以形成最多 1 个 "aaaaa" 的副本，所以返回 1 。
提示：1 <= s.length <= 100
1 <= target.length <= 10
```

- 解题思路

```
func rearrangeCharacters(s string, target string) int {
        m1, m2 := make(map[byte]int), make(map[byte]int)
        for i := 0; i < len(s); i++ {
                m1[s[i]]++
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(target); i++ {
                m2[target[i]]++
        }
        res := len(s)
        for k, v := range m2 {
                value := m1[k] / v
                if value < res {
                        res = value
                }
        }
        return res
}
```

67.20 2293. 极大极小游戏 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，其长度是 2 的幂。
对 nums 执行下述算法：
设 n 等于 nums 的长度，如果 n == 1 ，终止 算法过程。否则，创建␣
↪一个新的整数数组newNums ，
新数组长度为 n / 2 ，下标从 0 开始。
对于满足0 <= i < n / 2 的每个 偶数 下标 i ，将 newNums[i] 赋值 为 min(nums[2 * i],␣
↪nums[2 * i + 1]) 。
对于满足0 <= i < n / 2 的每个 奇数 下标 i ，将 newNums[i] 赋值 为 max(nums[2 * i],␣
↪nums[2 * i + 1]) 。
用 newNums 替换 nums 。
从步骤 1 开始 重复 整个过程。
执行算法后，返回 nums 中剩下的那个数字。
示例 1：输入：nums = [1,3,5,2,4,8,2,2] 输出：1
解释：重复执行算法会得到下述数组。
第一轮：nums = [1,5,4,2]
第二轮：nums = [1,4]
第三轮：nums = [1]
1 是最后剩下的那个数字，返回 1 。
示例 2：输入：nums = [3] 输出：3
解释：3 就是最后剩下的数字，返回 3 。
提示：1 <= nums.length <= 1024
1 <= nums[i] <= 109
nums.length 是 2 的幂
```

- 解题思路

```
func minMaxGame(nums []int) int {
        for len(nums) > 1 {
                temp := make([]int, len(nums)/2)
                for i := 0; i < len(nums)/2; i++ {
                        if i%2 == 0 {
                                temp[i] = min(nums[i*2], nums[i*2+1])
                        } else {
                                temp[i] = max(nums[i*2], nums[i*2+1])
                        }
                }
                nums = temp
        }
        return nums[0]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minMaxGame(nums []int) int {
        n := len(nums)
        for n > 1 {
                for i := 0; i < n/2; i++ {
                        if i%2 == 0 {
                                nums[i] = min(nums[i*2], nums[i*2+1])
                        } else {
                                nums[i] = max(nums[i*2], nums[i*2+1])
                        }
                }
                n = n / 2
        }
        return nums[0]
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

67.21 2299. 强密码检验器 II(2)

- 题目

```
如果一个密码满足以下所有条件，我们称它是一个 强密码：
它有至少 8个字符。
至少包含 一个小写英文字母。
至少包含 一个大写英文字母。
至少包含 一个数字。
至少包含 一个特殊字符。特殊字符为："!@#$%^&*()-+"中的一个。
它 不包含2个连续相同的字符（比方说"aab"不符合该条件，但是"aba"符合该条件）。
给你一个字符串password，如果它是一个强密码，返回true，否则返回false。
示例 1：输入：password = "IloveLe3tcode!" 输出：true
解释：密码满足所有的要求，所以我们返回 true 。
示例 2：输入：password = "Me+You--IsMyDream"  输出：false
解释：密码不包含数字，且包含 2 个连续相同的字符。所以我们返回 false 。
示例 3：输入：password = "1aB!"  输出：false
解释：密码不符合长度要求。所以我们返回 false 。
提示：1 <= password.length <= 100
password包含字母，数字和"!@#$%^&*()-+"这些特殊字符。
```

- 解题思路

```
func strongPasswordCheckerII(password string) bool {
        if len(password) < 8 {
                return false
        }
```

(续下页)

(接上页)

```
        s := "!@#$%^&*()-+"
        mm := make(map[byte]bool)
        for i := 0; i < len(s); i++ {
                mm[s[i]] = true
        }
        m := make(map[int]int)
        arr := []byte(password)
        for k, v := range arr {
                if k >= 1 {
                        if v == arr[k-1] {
                                return false
                        }
                }
                if '0' <= v && v <= '9' {
                        m[1] = 1
                }
                if 'a' <= v && v <= 'z' {
                        m[2] = 1
                }
                if 'A' <= v && v <= 'Z' {
                        m[3] = 1
                }
                if mm[v] == true {
                        m[4] = 1
                }
        }
        return len(m) == 4
}

# 2
func strongPasswordCheckerII(password string) bool {
        if len(password) < 8 {
                return false
        }
        m := make(map[int]int)
        for i := 0; i < len(password); i++ {
                if i >= 1 {
                        if password[i] == password[i-1] {
                                return false
                        }
                }
                v := rune(password[i])
                switch {
```

(续下页)

(接上页)

```
                case unicode.IsLower(v):
                        m[1] = 1
                case unicode.IsUpper(v):
                        m[2] = 1
                case unicode.IsDigit(v):
                        m[3] = 1
                default:
                        m[4] = 1
                }
        }
        return len(m) == 4
}
```

CHAPTER 68

2201-2300-Medium

68.1 2201. 统计可以提取的工件 (2)

- 题目

```
存在一个 n x n 大小、下标从 0 开始的网格，网格中埋着一些工件。
给你一个整数 n 和一个下标从 0 开始的二维整数数组 artifacts ，artifacts␣
↪描述了矩形工件的位置，
其中 artifacts[i] = [r1i, c1i, r2i, c2i] 表示第 i 个工件在子网格中的填埋情况：
(r1i, c1i) 是第 i 个工件 左上 单元格的坐标，且
(r2i, c2i) 是第 i 个工件 右下 单元格的坐标。
你将会挖掘网格中的一些单元格，并清除其中的填埋物。如果单元格中埋着工件的一部分，那么该工件这一部分将
如果一个工件的所有部分都都裸露出来，你就可以提取该工件。
给你一个下标从 0 开始的二维整数数组 dig ，其中 dig[i] = [ri, ci] 表示你将会挖掘单元格␣
↪(ri, ci) ，返回你可以提取的工件数目。
生成的测试用例满足：
不存在重叠的两个工件。
每个工件最多只覆盖 4 个单元格。
dig 中的元素互不相同。
示例 1：输入：n = 2, artifacts = [[0,0,0,0],[0,1,1,1]], dig = [[0,0],[0,1]] 输出：1
解释：  不同颜色表示不同的工件。挖掘的单元格用 'D' 在网格中进行标记。
有 1 个工件可以提取，即红色工件。
蓝色工件在单元格 (1,1) 的部分尚未裸露出来，所以无法提取该工件。
因此，返回 1 。
示例 2：输入：n = 2, artifacts = [[0,0,0,0],[0,1,1,1]], dig = [[0,0],[0,1],[1,1]]␣
```

(续下页)

(接上页)

```
↪输出：2
解释：红色工件和蓝色工件的所有部分都裸露出来（用 'D' 标记），都可以提取。因此，返回 2␣
↪。
提示：1 <= n <= 1000
1 <= artifacts.length, dig.length <= min(n2, 105)
artifacts[i].length == 4
dig[i].length == 2
0 <= r1i, c1i, r2i, c2i, ri, ci <= n - 1
r1i <= r2i
c1i <= c2i
不存在重叠的两个工件
每个工件 最多 只覆盖 4 个单元格
dig 中的元素互不相同
```

- 解题思路

```
func digArtifacts(n int, artifacts [][]int, dig [][]int) int {
        res := 0
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < len(dig); i++ {
                a, b := dig[i][0], dig[i][1]
                arr[a][b] = 1
        }
        for i := 0; i < len(artifacts); i++ {
                a, b, c, d := artifacts[i][0], artifacts[i][1], artifacts[i][2],␣
↪artifacts[i][3]
                flag := true
                for x := a; x <= c; x++ {
                        for y := b; y <= d; y++ {
                                if arr[x][y] == 0 {
                                        flag = false
                                }
                        }
                }
                if flag == true {
                        res++
                }
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func digArtifacts(n int, artifacts [][]int, dig [][]int) int {
        res := 0
        m := make(map[[2]int]int)
        for i := 0; i < len(dig); i++ {
                a, b := dig[i][0], dig[i][1]
                m[[2]int{a, b}] = 1
        }
        for i := 0; i < len(artifacts); i++ {
                a, b, c, d := artifacts[i][0], artifacts[i][1], artifacts[i][2],␣
↪artifacts[i][3]
                flag := true
                for x := a; x <= c; x++ {
                        for y := b; y <= d; y++ {
                                if m[[2]int{x, y}] == 0 {
                                        flag = false
                                }
                        }
                }
                if flag == true {
                        res++
                }
        }
        return res
}
```

68.2 2202.K 次操作后最大化顶端元素 (1)

- 题目

```
给你一个下标从 0开始的整数数组nums，它表示一个 栈 ，其中 nums[0]是栈顶的元素。
每一次操作中，你可以执行以下操作 之一：
如果栈非空，那么 删除栈顶端的元素。
如果存在 1␣
↪个或者多个被删除的元素，你可以从它们中选择任何一个，添加回栈顶，这个元素成为新的栈顶元素。
同时给你一个整数k，它表示你总共需要执行操作的次数。
请你返回 恰好执行 k次操作以后，栈顶元素的 最大值。如果执行完␣
↪k次操作以后，栈一定为空，请你返回 -1。
示例 1：输入：nums = [5,2,2,4,0,6], k = 4 输出：5
解释：4 次操作后，栈顶元素为 5 的方法之一为：
- 第 1 次操作：删除栈顶元素 5 ，栈变为 [2,2,4,0,6] 。
- 第 2 次操作：删除栈顶元素 2 ，栈变为 [2,4,0,6] 。
```

(续下页)

(接上页)

```
- 第 3 次操作：删除栈顶元素 2 ，栈变为 [4,0,6] 。
- 第 4 次操作：将 5 添加回栈顶，栈变为 [5,4,0,6] 。
注意，这不是最后栈顶元素为 5 的唯一方式。但可以证明，4 次操作以后 5␣
↪是能得到的最大栈顶元素。
示例 2：输入：nums = [2], k = 1 输出：-1
解释：第 1 次操作中，我们唯一的选择是将栈顶元素弹出栈。
由于 1 次操作后无法得到一个非空的栈，所以我们返回 -1 。
提示：1 <= nums.length <= 105
0 <= nums[i], k <= 109
```

- 解题思路

```
func maximumTop(nums []int, k int) int {
        n := len(nums)
        if n == 1 && k%2 == 1 { // 特殊情况：操作1次删除后栈为空
                return -1
        }
        res := 0
        // 2种情况：
        // 1、k-1中的最大值，最后1次选放回最大值（不会选中第k个数）
        // 2、k+1个位置，删除k个数，剩下的就是栈顶
        for i := 1; i <= n; i++ {
                if i <= k+1 && i != k {
                        res = max(res, nums[i-1])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

68.3 2207. 字符串中最多数目的子字符串 (2)

- 题目

```
给你一个下标从 0开始的字符串text和另一个下标从 0开始且长度为2的字符串pattern，两者都只包含小写英文字母。
你可以在 text中任意位置插入 一个 字符，这个插入的字符必须是pattern[0]或者pattern[1]。
注意，这个字符可以插入在 text开头或者结尾的位置。
请你返回插入一个字符后，text中最多包含多少个等于 pattern的 子序列。
子序列指的是将一个字符串删除若干个字符后（也可以不删除），剩余字符保持原本顺序得到的字符串。
示例 1：输入：text = "abdcdbc", pattern = "ac" 输出：4
解释：
如果我们在 text[1] 和 text[2] 之间添加 pattern[0] = 'a' ，那么我们得到 "abadcdbc"。那么 "ac" 作为子序列出现 4 次。
其他得到 4 个 "ac" 子序列的方案还有 "aabdcdbc" 和 "abdacdbc" 。
但是，"abdcadbc" ，"abdccdbc" 和 "abdcdbcc" 这些字符串虽然是可行的插入方案，
但是只出现了 3 次 "ac" 子序列，所以不是最优解。
可以证明插入一个字符后，无法得到超过 4 个 "ac" 子序列。
示例 2：输入：text = "aabb", pattern = "ab" 输出：6
解释：可以得到 6 个 "ab" 子序列的部分方案为 "aaabb" ，"aaabb" 和 "aabbb" 。
提示：1 <= text.length <= 105
pattern.length == 2
text 和pattern都只包含小写英文字母。
```

- 解题思路

```
func maximumSubsequenceCount(text string, pattern string) int64 {
        a, b := pattern[0], pattern[1]
        n := len(text)
        if a == b { // 相同情况：count+1的组合
                count := int64(strings.Count(text, string(a)))
                return (count + 1) * count / 2
        }
        var countA, countB int64
        res := int64(0)
        for i := 0; i < n; i++ {
                if text[i] == a {
                        countA++
                } else if text[i] == b {
                        countB++
                        res = res + countA // 子序列数量：加上前面的a
                }
        }
```

(续下页)

(接上页)

```
        return res + max(countA, countB) //␣
↪最后决定插入a还是b: 插入a在最前面, 插入b在最后面
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}

# 2
func maximumSubsequenceCount(text string, pattern string) int64 {
        a, b := pattern[0], pattern[1]
        n := len(text)
        var countA, countB int64
        res := int64(0)
        for i := 0; i < n; i++ {
                if text[i] == b { // 存在相同的情况, 先判断b
                        countB++
                        res = res + countA // 子序列数量: 加上前面的a
                }
                if text[i] == a {
                        countA++
                }
        }
        return res + max(countA, countB) //␣
↪最后决定插入a还是b: 插入a在最前面, 插入b在最后面
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}
```

68.4 2208. 将数组和减半的最少操作次数 (1)

- 题目

```
给你一个正整数数组nums。每一次操作中，你可以从nums中选择 任意一个数并将它减小到恰好一半。
（注意，在后续操作中你可以对减半过的数继续执行操作）
请你返回将 nums数组和 至少减少一半的 最少操作数。
示例 1：输入：nums = [5,19,8,1] 输出：3
解释：初始 nums 的和为 5 + 19 + 8 + 1 = 33 。
以下是将数组和减少至少一半的一种方法：
选择数字 19 并减小为 9.5 。
选择数字 9.5 并减小为 4.75 。
选择数字 8 并减小为 4 。
最终数组为 [5, 4.75, 4, 1] ，和为 5 + 4.75 + 4 + 1 = 14.75 。
nums 的和减小了 33 - 14.75 = 18.25 ，减小的部分超过了初始数组和的一半，18.25 >= 33/2 = 16.5 。
我们需要 3 个操作实现题目要求，所以返回 3 。
可以证明，无法通过少于 3 个操作使数组和减少至少一半。
示例 2：输入：nums = [3,8,20] 输出：3
解释：初始 nums 的和为 3 + 8 + 20 = 31 。
以下是将数组和减少至少一半的一种方法：
选择数字 20 并减小为 10 。
选择数字 10 并减小为 5 。
选择数字 3 并减小为 1.5 。
最终数组为 [1.5, 8, 5] ，和为 1.5 + 8 + 5 = 14.5 。
nums 的和减小了 31 - 14.5 = 16.5 ，减小的部分超过了初始数组和的一半， 16.5 >= 31/2 = 16.5 。
我们需要 3 个操作实现题目要求，所以返回 3 。
可以证明，无法通过少于 3 个操作使数组和减少至少一半。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 107
```

- 解题思路

```go
func halveArray(nums []int) int {
        n := len(nums)
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        sum := float64(0)
        for i := 0; i < n; i++ {
                sum = sum + float64(nums[i])
                heap.Push(&intHeap, []float64{float64(nums[i])})
        }
```

(续下页)

(接上页)

```
        target := sum / 2
        res := 0
        for sum > target {
                node := heap.Pop(&intHeap).([]float64)
                v := node[0]
                v = v / 2
                sum = sum - v
                heap.Push(&intHeap, []float64{v})
                res++
        }
        return res
}

type IntHeap [][]float64

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][0] > h[j][0] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([]float64)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}
```

68.5 2211. 统计道路上的碰撞次数 (2)

- 题目

```
在一条无限长的公路上有 n 辆汽车正在行驶。汽车按从左到右的顺序按从 0 到 n - 1␣
↪编号，每辆车都在一个 独特的 位置。
给你一个下标从 0 开始的字符串 directions ，长度为 n 。
directions[i] 可以是 'L'、'R' 或 'S' 分别表示第 i 辆车是向 左 、向 右 或者 停留␣
↪在当前位置。每辆车移动时 速度相同 。
碰撞次数可以按下述方式计算：
当两辆移动方向相反的车相撞时，碰撞次数加 2 。
当一辆移动的车和一辆静止的车相撞时，碰撞次数加 1 。
碰撞发生后，涉及的车辆将无法继续移动并停留在碰撞位置。除此之外，汽车不能改变它们的状态或移动方向。
返回在这条道路上发生的 碰撞总次数 。
示例 1：输入：directions = "RLRSLL" 输出：5
```

(续下页)

(接上页)

```
解释：将会在道路上发生的碰撞列出如下：
- 车 0 和车 1 会互相碰撞。由于它们按相反方向移动，碰撞数量变为 0 + 2 = 2 。
- 车 2 和车 3 会互相碰撞。由于 3 是静止的，碰撞数量变为 2 + 1 = 3 。
- 车 3 和车 4 会互相碰撞。由于 3 是静止的，碰撞数量变为 3 + 1 = 4 。
- 车 4 和车 5 会互相碰撞。在车 4 和车 3 碰撞之后，车 4 会待在碰撞位置，接着和车 5␣
↪碰撞。碰撞数量变为 4 + 1 = 5 。
因此，将会在道路上发生的碰撞总次数是 5 。
示例 2：输入：directions = "LLRR" 输出：0
解释：不存在会发生碰撞的车辆。因此，将会在道路上发生的碰撞总次数是 0 。
提示：1 <= directions.length <= 105
directions[i] 的值为 'L'、'R' 或 'S'
```

- 解题思路

```
func countCollisions(directions string) int {
        s := strings.TrimLeft(directions, "L") // 左边向左不会发生碰撞
        s = strings.TrimRight(s, "R")          // 右边向右不会发生碰撞
        return len(s) - strings.Count(s, "S")  // 剩下的车都会发生碰撞
}

# 2
func countCollisions(directions string) int {
        n := len(directions)
        i, j := 0, n-1
        for i < n && directions[i] == 'L' {
                i++
        }
        for j >= 0 && directions[j] == 'R' {
                j--
        }
        count := 0
        for k := i; k <= j; k++ {
                if directions[k] == 'S' {
                        count++
                }
        }
        return j - i + 1 - count
}
```

68.6 2212. 射箭比赛中的最大得分 (1)

- 题目

```
Alice 和 Bob 是一场射箭比赛中的对手。比赛规则如下：
Alice 先射 numArrows 支箭，然后 Bob 也射 numArrows 支箭。
分数按下述规则计算：
箭靶有若干整数计分区域，范围从 0 到 11 （含 0 和 11）。
箭靶上每个区域都对应一个得分 k（范围是 0 到 11），Alice 和 Bob 分别在得分 k区域射中ak
↪和 bk 支箭。
如果 ak >= bk ，那么 Alice 得 k 分。如果 ak < bk ，则 Bob 得 k 分
如果 ak == bk == 0 ，那么无人得到 k 分。
例如，Alice 和 Bob 都向计分为 11 的区域射 2 支箭，那么 Alice 得 11 分。
如果 Alice 向计分为 11 的区域射 0 支箭，但 Bob 向同一个区域射 2 支箭，那么 Bob 得11
↪分。
给你整数 numArrows 和一个长度为 12 的整数数组 aliceArrows ，该数组表示 Alice 射中0 到
↪11 每个计分区域的箭数量。
现在，Bob 想要尽可能 最大化 他所能获得的总分。
返回数组 bobArrows ，该数组表示 Bob 射中0 到 11 每个 计分区域的箭数量。且 bobArrows
↪的总和应当等于 numArrows 。
如果存在多种方法都可以使 Bob 获得最大总分，返回其中 任意一种 即可。
示例 1：输入：numArrows = 9, aliceArrows = [1,1,0,1,0,0,2,1,0,1,2,0] 输出：[0,0,0,0,1,
↪1,0,0,1,2,3,1]
解释：上表显示了比赛得分情况。
Bob 获得总分 4 + 5 + 8 + 9 + 10 + 11 = 47 。
可以证明 Bob 无法获得比 47 更高的分数。
示例 2：输入：numArrows = 3, aliceArrows = [0,0,1,0,0,0,0,0,0,0,0,2] 输出：[0,0,0,0,0,
↪0,0,0,1,1,1,0]
解释：上表显示了比赛得分情况。
Bob 获得总分 8 + 9 + 10 = 27 。
可以证明 Bob 无法获得比 27 更高的分数。
提示：1 <= numArrows <= 105
aliceArrows.length == bobArrows.length == 12
0 <= aliceArrows[i], bobArrows[i] <= numArrows
sum(aliceArrows[i]) == numArrows
```

- 解题思路

```
func maximumBobPoints(numArrows int, aliceArrows []int) []int {
        n := len(aliceArrows)
        total := 1 << n // 所有状态
        maxValue := 0
        maxState := 0
        for state := 0; state < total; state++ { // 枚举所有状态
```

(续下页)

(接上页)

```
                sum := 0
                score := 0
                for i := 0; i < n; i++ {
                        if (state>>i)&1 > 0 {
                                sum = sum + aliceArrows[i] + 1 // 箭的数量
                                score = score + i              // 得分
                        }
                }
                if sum <= numArrows && score > maxValue {
                        maxValue = score
                        maxState = state
                }
        }
        res := make([]int, n)
        for i := 0; i < n; i++ {
                if (maxState>>i)&1 > 0 {
                        res[i] = aliceArrows[i] + 1
                        numArrows = numArrows - res[i]
                }
        }
        res[0] = res[0] + numArrows // 剩下放在任何1个里面即可
        return res
}
```

68.7 2216. 美化数组的最少删除数 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，如果满足下述条件，则认为数组 nums 是一个␣
↪美丽数组 ：
nums.length 为偶数
对所有满足 i % 2 == 0 的下标 i ，nums[i] != nums[i + 1] 均成立
注意，空数组同样认为是美丽数组。
你可以从 nums 中删除任意数量的元素。
当你删除一个元素时，被删除元素右侧的所有元素将会向左移动一个单位以填补空缺，而左侧的元素将会保持␣
↪不变 。
返回使 nums 变为美丽数组所需删除的 最少 元素数目。
示例 1：输入：nums = [1,1,2,3,5] 输出：1
解释：可以删除 nums[0] 或 nums[1] ，这样得到的 nums = [1,2,3,5] 是一个美丽数组。
可以证明，要想使 nums 变为美丽数组，至少需要删除 1 个元素。
示例 2：输入：nums = [1,1,2,2,3,3] 输出：2
解释：可以删除 nums[0] 和 nums[5] ，这样得到的 nums = [1,2,2,3] 是一个美丽数组。
```

(续下页)

(接上页)

```
可以证明，要想使 nums 变为美丽数组，至少需要删除 2 个元素。
提示：1 <= nums.length <= 105
0 <= nums[i] <= 105
```

- 解题思路

```
func minDeletion(nums []int) int {
        res := 0
        n := len(nums)
        for i := 0; i < n; {
                j := i + 1
                for j < n && nums[i] == nums[j] { //␣
↪找到后面1个不相同的组成1对，然后开始找下一对
                        j++
                }
                if j < n {
                        res = res + 2
                }
                i = j + 1
        }
        return n - res
}

# 2
func minDeletion(nums []int) int {
        res := 0
        n := len(nums)
        for i := 0; i < n-1; i++ {
                if nums[i] == nums[i+1] {
                        res++
                } else {
                        i++
                }
        }
        return res + (n-res)%2 // 剩下奇数个要+1
}
```

68.8 2217. 找到指定长度的回文数 (2)

- 题目

```
给你一个整数数组queries和一个 正整数intLength，请你返回一个数组answer，
其中answer[i] 是长度为intLength的正回文数␣
↪中第queries[i]小的数字，如果不存在这样的回文数，则为 -1。
回文数 指的是从前往后和从后往前读一模一样的数字。回文数不能有前导 0 。
示例 1：输入：queries = [1,2,3,4,5,90], intLength = 3 输出：[101,111,121,131,141,999]
解释：长度为 3 的最小回文数依次是：
101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, ...
第 90 个长度为 3 的回文数是 999 。
示例 2：输入：queries = [2,4,6], intLength = 4 输出：[1111,1331,1551]
解释：长度为 4 的前 6 个回文数是：
1001, 1111, 1221, 1331, 1441 和 1551 。
提示：1 <= queries.length <= 5 * 104
1 <= queries[i] <= 109
1 <= intLength<= 15
```

- 解题思路

```
func kthPalindrome(queries []int, intLength int) []int64 {
        n := (intLength + 1) / 2          // 回文前半部分的长度：12321=>长度5=>
↪前半部分长度3
        start := int(math.Pow10(n-1) - 1) // n长度对应的回文数量下限 ：2=>10^2-1=99
        limit := int(math.Pow10(n) - 1)   // n长度对应的回文数量上限
        res := make([]int64, 0)
        for i := 0; i < len(queries); i++ {
                if start+queries[i] > limit {
                        res = append(res, -1)
                        continue
                }
                res = append(res, getKthPalindrome(intLength, start+queries[i]))
        }
        return res
}

func getKthPalindrome(intLength, num int) int64 {
        arr := []byte(strconv.Itoa(num))
        if intLength%2 == 0 { // 偶数
                for i := len(arr) - 1; i >= 0; i-- {
                        arr = append(arr, arr[i])
                }
        } else {
```

(续下页)

(接上页)

```
                for i := len(arr) - 2; i >= 0; i-- {
                        arr = append(arr, arr[i])
                }
        }
        res, _ := strconv.ParseInt(string(arr), 10, 64)
        return res
}

# 2
func kthPalindrome(queries []int, intLength int) []int64 {
        n := (intLength + 1) / 2          // 回文前半部分的长度: 12321=>长度5=>
↪前半部分长度3
        start := int(math.Pow10(n-1) - 1) // n长度对应的回文数量下限 : 2=>10^2-1=99
        limit := int(math.Pow10(n) - 1)   // n长度对应的回文数量上限
        res := make([]int64, 0)
        for i := 0; i < len(queries); i++ {
                if start+queries[i] > limit {
                        res = append(res, -1)
                        continue
                }
                res = append(res, getKthPalindrome(intLength, start+queries[i]))
        }
        return res
}

func getKthPalindrome(intLength, num int) int64 {
        temp := num
        if intLength%2 == 1 {
                temp = temp / 10
        }
        res := int64(num)
        for ; temp > 0; temp = temp / 10 {
                res = res*10 + int64(temp%10)
        }
        return res
}
```

68.9 2221. 数组的三角和 (2)

- 题目

```
给你一个下标从 0开始的整数数组nums，其中nums[i]是 0到 9之间（两者都包含）的一个数字。
nums的 三角和是执行以下操作以后最后剩下元素的值：
nums初始包含n个元素。如果n == 1，终止操作。否则，创建一个新的下标从0开始的长度为 n -␣
↪1的整数数组newNums。
对于满足0 <= i <n - 1的下标i，newNums[i] 赋值为(nums[i] + nums[i+1]) % 10，
↪%表示取余运算。
将newNums替换 数组nums。
从步骤 1 开始重复整个过程。
请你返回nums的三角和。
示例 1：输入：nums = [1,2,3,4,5] 输出：8
解释：上图展示了得到数组三角和的过程。
示例 2：输入：nums = [5] 输出：5
解释：由于 nums 中只有一个元素，数组的三角和为这个元素自己。
提示：1 <= nums.length <= 1000
0 <= nums[i] <= 9
```

- 解题思路

```
func triangularSum(nums []int) int {
        for len(nums) > 1 {
                arr := make([]int, len(nums)-1)
                for i := 0; i < len(nums)-1; i++ {
                        arr[i] = (nums[i] + nums[i+1]) % 10
                }
                nums = arr
        }
        return nums[0]
}

# 2
func triangularSum(nums []int) int {
        for n := len(nums) - 1; n > 0; n-- {
                for i := 0; i < n; i++ {
                        nums[i] = (nums[i] + nums[i+1]) % 10
                }
        }
        return nums[0]
}
```

68.10 2222. 选择建筑的方案数 (2)

- 题目

```
给你一个下标从 0开始的二进制字符串s，它表示一条街沿途的建筑类型，其中：
s[i] = '0'表示第i栋建筑是一栋办公楼，
s[i] = '1'表示第i栋建筑是一间餐厅。
作为市政厅的官员，你需要随机选择3 栋建筑。然而，为了确保多样性，选出来的 3 栋建筑
↪相邻的两栋不能是同一类型。
比方说，给你s = "001101"，我们不能选择第1，3和5栋建筑，
因为得到的子序列是"011"，有相邻两栋建筑是同一类型，所以 不合题意。
请你返回可以选择 3 栋建筑的 有效方案数。
示例 1：输入：s = "001101" 输出：6
解释：以下下标集合是合法的：
- [0,2,4] ，从 "001101" 得到 "010"
- [0,3,4] ，从 "001101" 得到 "010"
- [1,2,4] ，从 "001101" 得到 "010"
- [1,3,4] ，从 "001101" 得到 "010"
- [2,4,5] ，从 "001101" 得到 "101"
- [3,4,5] ，从 "001101" 得到 "101"
没有别的合法选择，所以总共有 6 种方法。
示例 2：输入：s = "11100" 输出：0
解释：没有任何符合题意的选择。
提示：3 <= s.length <= 105
s[i]要么是'0'，要么是'1'。
```

- 解题思路

```
func numberOfWays(s string) int64 {
        res := int64(0)
        n := len(s)
        count1 := strings.Count(s, "1") // 1的总个数
        count0 := n - count1            // 0的总个数
        count := 0                      // 遍历时候1的个数
        for i := 0; i < n; i++ {        // 枚举以当前字母作为中间元素
                if s[i] == '1' { // 010
                        left0 := i - count       // 左边0的个数
                        right0 := count0 - left0 // 右边0的个数
                        res = res + int64(left0)*int64(right0)
                        count++
                } else { // 101
                        left1 := count           // 左边1的个数
                        right1 := count1 - count // 右边1的个数
                        res = res + int64(left1)*int64(right1)
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 2
func numberOfWays(s string) int64 {
        return getCount(s, "101") + getCount(s, "010")
}

func getCount(s string, str string) int64 {
        var a, b, c int64
        for i := 0; i < len(s); i++ {
                if s[i] == str[0] { // 以101为例：计算1的个数
                        a++
                }
                if s[i] == str[1] { // 以101为例：计算01的个数
                        b = b + a
                }
                if s[i] == str[2] { // 以101为例：计算101的个数
                        c = c + b
                }
        }
        return c
}
```

68.11 2225. 找出输掉零场或一场比赛的玩家 (2)

- 题目

```
给你一个整数数组 matches 其中 matches[i] = [winneri, loseri] 表示在一场比赛中 winneri␣
↪击败了 loseri 。
返回一个长度为 2 的列表 answer ：
answer[0] 是所有 没有 输掉任何比赛的玩家列表。
answer[1] 是所有恰好输掉 一场 比赛的玩家列表。
两个列表中的值都应该按 递增 顺序返回。
注意：只考虑那些参与 至少一场 比赛的玩家。
生成的测试用例保证 不存在 两场比赛结果 相同 。
示例 1：输入：matches = [[1,3],[2,3],[3,6],[5,6],[5,7],[4,5],[4,8],[4,9],[10,4],[10,
↪9]]
输出：[[1,2,10],[4,5,7,8]]
解释：玩家 1、2 和 10 都没有输掉任何比赛。
```

(续下页)

(接上页)

```
玩家 4、5、7 和 8 每个都输掉一场比赛。
玩家 3、6 和 9 每个都输掉两场比赛。
因此，answer[0] = [1,2,10] 和 answer[1] = [4,5,7,8] 。
示例 2：输入：matches = [[2,3],[1,3],[5,4],[6,4]] 输出：[[1,2,5,6],[]]
解释：玩家 1、2、5 和 6 都没有输掉任何比赛。
玩家 3 和 4 每个都输掉两场比赛。
因此，answer[0] = [1,2,5,6] 和 answer[1] = [] 。
提示：1 <= matches.length <= 105
matches[i].length == 2
1 <= winneri, loseri <= 105
winneri != loseri
所有 matches[i] 互不相同
```

- 解题思路

```
func findWinners(matches [][]int) [][]int {
        m1, m2 := make(map[int]int), make(map[int]int)
        for i := 0; i < len(matches); i++ {
                a, b := matches[i][0], matches[i][1]
                m1[a]++ // 赢的+1
                m2[b]-- // 输的-1
        }
        arr1, arr2 := make([]int, 0), make([]int, 0)
        for k := range m1 {
                if _, ok := m2[k]; ok == false {
                        arr1 = append(arr1, k)
                }
        }
        for k, v := range m2 {
                if v == -1 {
                        arr2 = append(arr2, k)
                }
        }
        sort.Ints(arr1)
        sort.Ints(arr2)
        return [][]int{arr1, arr2}
}

# 2
func findWinners(matches [][]int) [][]int {
        m := make(map[int]int)
        for i := 0; i < len(matches); i++ {
                a, b := matches[i][0], matches[i][1]
                if _, ok := m[a]; ok == false {
```

(续下页)

(接上页)

```
                        m[a] = 0 // 赢的标记为0
                }
                m[b]++ // 输的+1
        }
        arr := [2][]int{}
        for k, v := range m {
                if v < 2 { // 0: 全赢， 1: 输1次
                        arr[v] = append(arr[v], k)
                }
        }
        sort.Ints(arr[0])
        sort.Ints(arr[1])
        return [][]int{arr[0], arr[1]}
}
```

68.12 2226. 每个小孩最多能分到多少糖果 (3)

- 题目

```
给你一个 下标从 0 开始 的整数数组 candies 。数组中的每个元素表示大小为 candies[i]␣
↪的一堆糖果。
你可以将每堆糖果分成任意数量的 子堆 ，但 无法 再将两堆合并到一起。
另给你一个整数 k 。你需要将这些糖果分配给 k 个小孩，使每个小孩分到 相同 数量的糖果。
每个小孩可以拿走 至多一堆 糖果，有些糖果可能会不被分配。
返回每个小孩可以拿走的 最大糖果数目 。
示例 1: 输入: candies = [5,8,6], k = 3 输出: 5
解释: 可以将 candies[1] 分成大小分别为 5 和 3 的两堆，然后把 candies[2]␣
↪分成大小分别为 5 和 1 的两堆。
现在就有五堆大小分别为 5、5、3、5 和 1 的糖果。可以把 3 堆大小为 5 的糖果分给 3␣
↪个小孩。
可以证明无法让每个小孩得到超过 5 颗糖果。
示例 2: 输入: candies = [2,5], k = 11 输出: 0
解释: 总共有 11 个小孩，但只有 7␣
↪颗糖果，但如果要分配糖果的话，必须保证每个小孩至少能得到 1 颗糖果。
因此，最后每个小孩都没有得到糖果，答案是 0 。
提示: 1 <= candies.length <= 105
1 <= candies[i] <= 107
1 <= k <= 1012
```

- 解题思路

```
func maximumCandies(candies []int, k int64) int {
        sum := int64(0)
        maxValue := 0
        for i := 0; i < len(candies); i++ {
                sum = sum + int64(candies[i])
                maxValue = max(maxValue, candies[i])
        }
        if sum < k {
                return 0
        }
        left := 1
        right := maxValue
        for left <= right {
                mid := left + (right-left)/2
                total := check(candies, mid)
                if total >= k {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        return left - 1
}

func check(arr []int, target int) int64 {
        res := int64(0)
        for i := 0; i < len(arr); i++ {
                res = res + int64(arr[i]/target)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maximumCandies(candies []int, k int64) int {
        sum := int64(0)
        maxValue := 0
        for i := 0; i < len(candies); i++ {
```

(续下页)

(接上页)

```
                sum = sum + int64(candies[i])
                maxValue = max(maxValue, candies[i])
        }
        if sum < k {
                return 0
        }
        left := 1
        right := maxValue + 1
        for left < right {
                mid := left + (right-left)/2
                total := check(candies, mid)
                if total >= k {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left - 1
}

func check(arr []int, target int) int64 {
        res := int64(0)
        for i := 0; i < len(arr); i++ {
                res = res + int64(arr[i]/target)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maximumCandies(candies []int, k int64) int {
        sum := int64(0)
        maxValue := 0
        for i := 0; i < len(candies); i++ {
                sum = sum + int64(candies[i])
                maxValue = max(maxValue, candies[i])
        }
```

(续下页)

(接上页)

```
        if sum < k {
                return 0
        }
        return sort.Search(maxValue+1, func(target int) bool {
                if target == 0 {
                        return false
                }
                res := int64(0)
                for i := 0; i < len(candies); i++ {
                        res = res + int64(candies[i]/target)
                }
                return res < k
        }) - 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

68.13 2232. 向表达式添加括号后的最小结果 (1)

- 题目

```
给你一个下标从 0 开始的字符串 expression ，格式为 "<num1>+<num2>" ，其中 <num1> 和
↪<num2> 表示正整数。
请你向 expression 中添加一对括号，使得在添加之后， expression␣
↪仍然是一个有效的数学表达式，并且计算后可以得到 最小 可能值。
左括号 必须 添加在 '+' 的左侧，而右括号必须添加在 '+' 的右侧。
返回添加一对括号后形成的表达式expression ，且满足 expression 计算得到 最小 可能值。
如果存在多个答案都能产生相同结果，返回任意一个答案。
生成的输入满足：expression 的原始值和添加满足要求的任一对括号之后 expression␣
↪的值，都符合 32-bit 带符号整数范围。
示例 1：输入：expression = "247+38" 输出："2(47+38)"
解释：表达式计算得到 2 * (47 + 38) = 2 * 85 = 170 。
注意 "2(4)7+38" 不是有效的结果，因为右括号必须添加在 '+' 的右侧。
可以证明 170 是最小可能值。
示例 2：输入：expression = "12+34" 输出："1(2+3)4"
解释：表达式计算得到 1 * (2 + 3) * 4 = 1 * 5 * 4 = 20 。
示例 3：输入：expression = "999+999" 输出："(999+999)"
```

(续下页)

(接上页)

```
解释：表达式计算得到 999 + 999 = 1998 。
提示：3 <= expression.length <= 10
expression 仅由数字 '1' 到 '9' 和 '+' 组成
expression 由数字开始和结束
expression 恰好仅含有一个 '+'.
expression 的原始值和添加满足要求的任一对括号之后 expression 的值，都符合 32-bit␣
↪带符号整数范围
```

- 解题思路

```
func minimizeResult(expression string) string {
        res := ""
        arr := strings.Split(expression, "+")
        left, right := arr[0], arr[1]
        minValue := math.MaxInt32
        for i := 0; i < len(left); i++ {
                for j := 1; j <= len(right); j++ {
                        var a, b, c, d int
                        if i == 0 { // 左边
                                a = 1
                        } else {
                                a, _ = strconv.Atoi(left[:i])
                        }
                        b, _ = strconv.Atoi(left[i:])
                        c, _ = strconv.Atoi(right[:j])
                        d = 1
                        if j < len(right) { // 右边
                                d, _ = strconv.Atoi(right[j:])
                        }
                        if a*(b+c)*d < minValue {
                                minValue = a * (b + c) * d
                                res = fmt.Sprintf("%s(%s+%s)%s", left[:i], left[i:],␣
↪right[:j], right[j:])
                        }
                }
        }
        return res
}
```

68.14 2233.K 次增加后的最大乘积 (2)

- 题目

```
给你一个非负整数数组nums和一个整数k。每次操作，你可以选择nums中 任一元素并将它 增加1。
请你返回 至多k次操作后，能得到的nums的最大乘积。由于答案可能很大，请你将答案对109 +␣
↪7取余后返回。
示例 1：输入：nums = [0,4], k = 5 输出：20
解释：将第一个数增加 5 次。
得到 nums = [5, 4] ，乘积为 5 * 4 = 20 。
可以证明 20 是能得到的最大乘积，所以我们返回 20 。
存在其他增加 nums 的方法，也能得到最大乘积。
示例 2：输入：nums = [6,3,3,2], k = 2 输出：216
解释：将第二个数增加 1 次，将第四个数增加 1 次。
得到 nums = [6, 4, 3, 3] ，乘积为 6 * 4 * 3 * 3 = 216 。
可以证明 216 是能得到的最大乘积，所以我们返回 216 。
存在其他增加 nums 的方法，也能得到最大乘积。
提示：1 <= nums.length, k <= 105
0 <= nums[i] <= 106
```

- 解题思路

```
var mod = 1000000007

func maximumProduct(nums []int, k int) int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < len(nums); i++ {
                heap.Push(&intHeap, nums[i])
        }
        // x y：如果x>y,y+1后的结果较大
        // (x+1)*y = xy+y
        // x*(y+1) = xy+x
        for i := 1; i <= k; i++ {
                node := heap.Pop(&intHeap).(int)
                node++
                heap.Push(&intHeap, node)
        }
        res := 1
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).(int)
                res = res * node % mod
        }
        return res
```

(续下页)

(接上页)

```
}

type IntHeap []int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i] < h[j] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.(int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 2
var mod = 1000000007

func maximumProduct(nums []int, k int) int {
        res := 1
        sort.Ints(nums)
        n := len(nums)
        nums = append(nums, math.MaxInt32/100)
        for i := 0; i < n; i++ {
                sum := (i + 1) * (nums[i+1] - nums[i])
                if k >= sum {
                        k = k - sum
                        continue
                }
                b := k / (i + 1) // 加多少
                d := k % (i + 1) // 剩下多少+1
                for j := 0; j <= i; j++ {
                        nums[j] = nums[i] + b // 在nums[i]基础上加
                        if j < d {
                                nums[j]++
                        }
                }
                break
        }
        for i := 0; i < n; i++ {
                res = res * nums[i] % mod
        }
```

(续下页)

(接上页)

```
        return res
}
```

68.15 2240. 买钢笔和铅笔的方案数 (1)

- 题目

```
给你一个整数total，表示你拥有的总钱数。同时给你两个整数cost1
↪和cost2，分别表示一支钢笔和一支铅笔的价格。
你可以花费你部分或者全部的钱，去买任意数目的两种笔。
请你返回购买钢笔和铅笔的不同方案数目。
示例 1：输入：total = 20, cost1 = 10, cost2 = 5 输出：9
解释：一支钢笔的价格为 10 ，一支铅笔的价格为 5 。
- 如果你买 0 支钢笔，那么你可以买 0 ，1 ，2 ，3 或者 4 支铅笔。
- 如果你买 1 支钢笔，那么你可以买 0 ，1 或者 2 支铅笔。
- 如果你买 2 支钢笔，那么你没法买任何铅笔。
所以买钢笔和铅笔的总方案数为 5 + 3 + 1 = 9 种。
示例 2：输入：total = 5, cost1 = 10, cost2 = 10 输出：1
解释：钢笔和铅笔的价格都为 10 ，都比拥有的钱数多，所以你没法购买任何文具。所以只有 1
↪种方案：买 0 支钢笔和 0 支铅笔。
提示：1 <= total, cost1, cost2 <= 106
```

- 解题思路

```
func waysToBuyPensPencils(total int, cost1 int, cost2 int) int64 {
        res := int64(0)
        for i := 0; i <= total/cost1; i++ {
                res = res + int64((total-cost1*i)/cost2) + 1 // 可以买0支铅笔，次数+1
        }
        return res
}
```

68.16 2241. 设计一个 ATM 机器 (1)

- 题目

```
一个 ATM 机器，存有5种面值的钞票：20，50，100，200和500美元。初始时，ATM 机是空的。
用户可以用它存或者取任意数目的钱。
取款时，机器会优先取 较大数额的钱。
比方说，你想取$300，并且机器里有2张 $50的钞票，1张$100的钞票和1张$200的钞票，
```

(续下页)

(接上页)

```
那么机器会取出$100 和$200的钞票。
但是，如果你想取$600，机器里有3张$200的钞票和1张$500的钞票，那么取款请求会被拒绝，
因为机器会先取出$500的钞票，然后无法取出剩余的$100。注意，因为有
↪$500钞票的存在，机器不能取$200的钞票。
请你实现 ATM 类：
ATM()初始化 ATM 对象。
void deposit(int[] banknotesCount)分别存入$20，$50，$100，$200和$500钞票的数目。
int[] withdraw(int amount)返回一个长度为5的数组，分别表示$20，$50，$100，$200和
↪$500钞票的数目，
并且更新 ATM 机里取款后钞票的剩余数量。如果无法取出指定数额的钱，请返回[-
↪1]（这种情况下 不取出任何钞票）。
示例 1：输入： ["ATM", "deposit", "withdraw", "deposit", "withdraw", "withdraw"]
[[], [[0,0,1,2,1]], [600], [[0,1,0,1,1]], [600], [550]]
输出：[null, null, [0,0,1,0,1], null, [-1], [0,1,0,0,1]]
解释：ATM atm = new ATM();
atm.deposit([0,0,1,2,1]); // 存入 1 张 $100 ，2 张 $200 和 1 张 $500 的钞票。
atm.withdraw(600);        // 返回 [0,0,1,0,1] 。机器返回 1 张 $100 和 1 张 $500␣
↪的钞票。
机器里剩余钞票的数量为 [0,0,0,2,0] 。
atm.deposit([0,1,0,1,1]); // 存入 1 张 $50 ，1 张 $200 和 1 张 $500 的钞票。
                          // 机器中剩余钞票数量为 [0,1,0,3,1] 。
atm.withdraw(600);        // 返回 [-1] 。机器会尝试取出 $500␣
↪的钞票，然后无法得到剩余的 $100 ，所以取款请求会被拒绝。
                          // 由于请求被拒绝，机器中钞票的数量不会发生改变。
atm.withdraw(550);        // 返回 [0,1,0,0,1] ，机器会返回 1 张 $50 的钞票和 1 张
↪$500 的钞票。
提示：banknotesCount.length == 5
0 <= banknotesCount[i] <= 109
1 <= amount <= 109
总共最多有5000次withdraw 和deposit的调用。
函数 withdraw 和deposit至少各有 一次调用。
```

- 解题思路

```
var Array = [5]int{20, 50, 100, 200, 500}

type ATM struct {
        arr [5]int
}

func Constructor() ATM {
        return ATM{arr: [5]int{}}
}
```

(续下页)

(接上页)

```
func (this *ATM) Deposit(banknotesCount []int) {
        for i := 0; i < len(banknotesCount); i++ {
                this.arr[i] = this.arr[i] + banknotesCount[i]
        }
}

func (this *ATM) Withdraw(amount int) []int {
        res := make([]int, 5)
        for i := 4; i >= 0; i-- {
                res[i] = min(this.arr[i], amount/Array[i])
                amount = amount - Array[i]*res[i]
        }
        if amount > 0 {
                return []int{-1}
        }
        for i := 0; i < 5; i++ {
                this.arr[i] = this.arr[i] - res[i]
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

68.17 2244. 完成所有任务需要的最少轮数 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 tasks ，其中 tasks[i] 表示任务的难度级别。
在每一轮中，你可以完成 2 个或者 3 个 相同难度级别 的任务。
返回完成所有任务需要的 最少 轮数，如果无法完成所有任务，返回 -1 。
示例 1：输入：tasks = [2,2,3,3,2,4,4,4,4,4] 输出：4
解释：要想完成所有任务，一个可能的计划是：
- 第一轮，完成难度级别为 2 的 3 个任务。
- 第二轮，完成难度级别为 3 的 2 个任务。
- 第三轮，完成难度级别为 4 的 3 个任务。
- 第四轮，完成难度级别为 4 的 2 个任务。
可以证明，无法在少于 4 轮的情况下完成所有任务，所以答案为 4 。
```

(续下页)

(接上页)

```
示例 2：输入：tasks = [2,3,3] 输出：-1
解释：难度级别为 2 的任务只有 1 个，但每一轮执行中，只能选择完成 2 个或者 3
↪个相同难度级别的任务。
因此，无法完成所有任务，答案为 -1 。
提示：1 <= tasks.length <= 105
1 <= tasks[i] <= 109
```

- 解题思路

```
func minimumRounds(tasks []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(tasks); i++ {
                m[tasks[i]]++
        }
        for _, v := range m {
                if v == 1 { // 1个直接返回
                        return -1
                }
                if v%3 == 0 {
                        res = res + v/3
                } else {
                        res = res + v/3 + 1 // %v=1 拆成2+2; %v=2, 拆成3+2
                }
        }
        return res
}
```

68.18 2245. 转角路径的乘积中最多能有几个尾随零

68.18.1 题目

```
给你一个二维整数数组 grid ，大小为 m x n，其中每个单元格都含一个正整数。
转角路径 定义为：包含至多一个弯的一组相邻单元。具体而言，路径应该完全 向水平方向 或者
↪向竖直方向 移动过弯（如果存在弯），
而不能访问之前访问过的单元格。在过弯之后，路径应当完全朝 另一个 方向行进：
如果之前是向水平方向，那么就应该变为向竖直方向；反之亦然。当然，同样不能访问之前已经访问过的单元格。
一条路径的 乘积 定义为：路径上所有值的乘积。
请你从 grid 中找出一条乘积中尾随零数目最多的转角路径，并返回该路径中尾随零的数目。
注意：水平 移动是指向左或右移动。 竖直 移动是指向上或下移动。
示例 1：输入：grid = [[23,17,15,3,20],[8,1,20,27,11],[9,4,6,2,21],[40,9,1,10,6],[22,7,
```

(续下页)

(接上页)

```
↪4,5,3]] 输出：3
解释：左侧的图展示了一条有效的转角路径。
其乘积为 15 * 20 * 6 * 1 * 10 = 18000 ，共计 3 个尾随零。
可以证明在这条转角路径的乘积中尾随零数目最多。
中间的图不是一条有效的转角路径，因为它有不止一个弯。
右侧的图也不是一条有效的转角路径，因为它需要重复访问已经访问过的单元格。
示例 2：输入：grid = [[4,3,2],[7,6,1],[8,8,8]] 输出：0
解释：网格如上图所示。
不存在乘积含尾随零的转角路径。
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 105
1 <= m * n <= 105
1 <= grid[i][j] <= 1000
```

68.18.2 解题思路

68.19 2249. 统计圆内格点数目 (1)

- 题目

```
给你一个二维整数数组 circles ，其中 circles[i] = [xi, yi, ri]
表示网格上圆心为 (xi, yi) 且半径为 ri 的第 i 个圆，返回出现在 至少一个 圆内的␣
↪格点数目 。
注意：格点 是指整数坐标对应的点。
圆周上的点 也被视为出现在圆内的点。
示例 1：输入：circles = [[2,2,1]] 输出：5
解释：给定的圆如上图所示。
出现在圆内的格点为 (1, 2)、(2, 1)、(2, 2)、(2, 3) 和 (3, 2)，在图中用绿色标识。
像 (1, 1) 和 (1, 3) 这样用红色标识的点，并未出现在圆内。
因此，出现在至少一个圆内的格点数目是 5 。
示例 2：输入：circles = [[2,2,2],[3,4,1]] 输出：16
解释：给定的圆如上图所示。
共有 16 个格点出现在至少一个圆内。
其中部分点的坐标是 (0, 2)、(2, 0)、(2, 4)、(3, 2) 和 (4, 4) 。
提示：1 <= circles.length <= 200
circles[i].length == 3
1 <= xi, yi <= 100
1 <= ri <= min(xi, yi)
```

- 解题思路

```
func countLatticePoints(circles [][]int) int {
        res := 0
        for i := 0; i <= 200; i++ {
                for j := 0; j <= 200; j++ {
                        for k := 0; k < len(circles); k++ {
                                x, y, r := circles[k][0], circles[k][1], circles[k][2]
                                if (i-x)*(i-x)+(j-y)*(j-y) <= r*r {
                                        res++
                                        break
                                }
                        }
                }
        }
        return res
}
```

68.20 2250. 统计包含每个点的矩形数目 (2)

- 题目

```
给你一个二维整数数组rectangles，其中rectangles[i] = [li,
↪hi]表示第i个矩形长为li高为hi。
给你一个二维整数数组points，其中points[j] = [xj, yj]是坐标为(xj, yj)的一个点。
第i个矩形的 左下角在(0, 0)处，右上角在(li, hi)。
请你返回一个整数数组count，长度为points.length，其中count[j]是 包含
↪第j个点的矩形数目。
如果0 <= xj <= li 且0 <= yj <= hi，那么我们说第i个矩形包含第j个点。
如果一个点刚好在矩形的 边上，这个点也被视为被矩形包含。
示例 1：输入：rectangles = [[1,2],[2,3],[2,5]], points = [[2,1],[1,4]]输出：[2,1]
解释：第一个矩形不包含任何点。
第二个矩形只包含一个点 (2, 1) 。
第三个矩形包含点 (2, 1) 和 (1, 4) 。
包含点 (2, 1) 的矩形数目为 2 。
包含点 (1, 4) 的矩形数目为 1 。
所以，我们返回 [2, 1] 。
示例 2：输入：rectangles = [[1,1],[2,2],[3,3]], points = [[1,3],[1,1]] 输出：[1,3]
解释：第一个矩形只包含点 (1, 1) 。
第二个矩形只包含点 (1, 1) 。
第三个矩形包含点 (1, 3) 和 (1, 1) 。
包含点 (1, 3) 的矩形数目为 1 。
包含点 (1, 1) 的矩形数目为 3 。
```

(续下页)

(接上页)

```
所以，我们返回 [1, 3] 。
提示：1 <= rectangles.length, points.length <= 5 * 104
rectangles[i].length == points[j].length == 2
1 <= li, xj <= 109
1 <= hi, yj <= 100
所有rectangles互不相同。
所有points 互不相同。
```

- 解题思路

```
func countRectangles(rectangles [][]int, points [][]int) []int {
        n := len(points)
        res := make([]int, n)
        arr := make([][]int, 101)
        for i := 0; i < len(rectangles); i++ {
                x, y := rectangles[i][0], rectangles[i][1]
                arr[y] = append(arr[y], x)
        }
        for i := 0; i < 101; i++ {
                sort.Ints(arr[i])
        }
        for i := 0; i < n; i++ {
                x, y := points[i][0], points[i][1]
                for j := y; j < 101; j++ {
                        total := len(arr[j]) - sort.SearchInts(arr[j], x) // 总和-
↪不满足要求的点
                        res[i] = res[i] + total                           // 累加
                }
        }
        return res
}

# 2
func countRectangles(rectangles [][]int, points [][]int) []int {
        n := len(points)
        res := make([]int, n)
        for i := 0; i < n; i++ {
                points[i] = append(points[i], i) // 添加下标
        }
        sort.Slice(points, func(i, j int) bool {
                return points[i][0] > points[j][0] // 横坐标排序
        })
        sort.Slice(rectangles, func(i, j int) bool {
                return rectangles[i][0] > rectangles[j][0] // 横坐标排序
```

(续下页)

(接上页)

```
        })
        start := 0
        arr := make([]int, 101)
        for i := 0; i < n; i++ {
                x, y, index := points[i][0], points[i][1], points[i][2]
                for ; start < len(rectangles) && x <= rectangles[start][0]; start++ {
                        arr[rectangles[start][1]]++ // 把纵坐标次数+1
                }
                for j := y; j < 101; j++ { // 遍历大于当前纵坐标
                        res[index] = res[index] + arr[j] // 累加次数
                }
        }
        return res
}
```

68.21 2256. 最小平均差 (1)

- 题目

```
给你一个下标从 0开始长度为 n的整数数组nums。
下标 i处的 平均差指的是 nums中 前i + 1个元素平均值和 后n - i - 1个元素平均值的␣
↪绝对差。
两个平均值都需要 向下取整到最近的整数。
请你返回产生 最小平均差的下标。如果有多个下标最小平均差相等，请你返回 最小的一个下标。
注意：两个数的绝对差是两者差的绝对值。
n个元素的平均值是 n个元素之 和除以（整数除法）n。
0个元素的平均值视为0。
示例 1：输入：nums = [2,5,3,9,5,3] 输出：3
解释：- 下标 0 处的平均差为：|2 / 1 - (5 + 3 + 9 + 5 + 3) / 5| = |2 / 1 - 25 / 5| =␣
↪|2 - 5| = 3 。
- 下标 1 处的平均差为：|(2 + 5) / 2 - (3 + 9 + 5 + 3) / 4| = |7 / 2 - 20 / 4| = |3 -␣
↪5| = 2 。
- 下标 2 处的平均差为：|(2 + 5 + 3) / 3 - (9 + 5 + 3) / 3| = |10 / 3 - 17 / 3| = |3 -␣
↪5| = 2 。
- 下标 3 处的平均差为：|(2 + 5 + 3 + 9) / 4 - (5 + 3) / 2| = |19 / 4 - 8 / 2| = |4 -␣
↪4| = 0 。
- 下标 4 处的平均差为：|(2 + 5 + 3 + 9 + 5) / 5 - 3 / 1| = |24 / 5 - 3 / 1| = |4 - 3|␣
↪= 1 。
- 下标 5 处的平均差为：|(2 + 5 + 3 + 9 + 5 + 3) / 6 - 0| = |27 / 6 - 0| = |4 - 0| = 4␣
↪。
下标 3 处的平均差为最小平均差，所以返回 3 。
示例 2：输入：nums = [0] 输出：0
```

(续下页)

(接上页)

```
解释：唯一的下标是 0 ，所以我们返回 0 。
下标 0 处的平均差为：|0 / 1 - 0| = |0 - 0| = 0 。
提示：1 <= nums.length <= 105
0 <= nums[i] <= 105
```

- 解题思路

```
func minimumAverageDifference(nums []int) int {
        res := 0
        right := 0
        for _, v := range nums {
                right = right + v
        }
        left := 0
        minValue := math.MaxInt32
        for i := 0; i < len(nums); i++ {
                left = left + nums[i]
                right = right - nums[i]
                var a, b int
                a = left / (i + 1)
                if i != len(nums)-1 {
                        b = right / (len(nums) - i - 1)
                }
                if abs(a-b) < minValue { // 更新结果
                        minValue = abs(a - b)
                        res = i
                }
        }
        return res
}

func abs(a int) int {
        if a < 0 {
                return -a
        }
        return a
}
```

68.22 2257. 统计网格图中没有被保卫的格子数 (2)

- 题目

```
给你两个整数m和n表示一个下标从0开始的m x n网格图。
同时给你两个二维整数数组guards 和walls，其中guards[i] = [rowi, coli]且walls[j] =␣
↪[rowj, colj]，
分别表示第 i个警卫和第 j座墙所在的位置。
一个警卫能看到 4 个坐标轴方向（即东、南、西、北）的␣
↪所有格子，除非他们被一座墙或者另外一个警卫 挡住了视线。
如果一个格子能被 至少一个警卫看到，那么我们说这个格子被 保卫了。
请你返回空格子中，有多少个格子是 没被保卫的。
示例 1：输入：m = 4, n = 6, guards = [[0,0],[1,1],[2,3]], walls = [[0,1],[2,2],[1,4]]␣
↪输出：7
解释：上图中，被保卫和没有被保卫的格子分别用红色和绿色表示。
总共有 7 个没有被保卫的格子，所以我们返回 7 。
示例 2：输入：m = 3, n = 3, guards = [[1,1]], walls = [[0,1],[1,0],[2,1],[1,2]]␣
↪输出：4
解释：上图中，没有被保卫的格子用绿色表示。
总共有 4 个没有被保卫的格子，所以我们返回 4 。
提示：1 <= m, n <= 105
2 <= m * n <= 105
1 <= guards.length, walls.length <= 5 * 104
2 <= guards.length + walls.length <= m * n
guards[i].length == walls[j].length == 2
0 <= rowi, rowj < m
0 <= coli, colj < n
guards和walls中所有位置 互不相同。
```

- 解题思路

```
func countUnguarded(m int, n int, guards [][]int, walls [][]int) int {
        arr := make([][]int, m)
        for i := 0; i < m; i++ {
                arr[i] = make([]int, n)
        }
        for i := 0; i < len(walls); i++ {
                a, b := walls[i][0], walls[i][1]
                arr[a][b] = 'W'
        }
        for i := 0; i < len(guards); i++ {
                a, b := guards[i][0], guards[i][1]
                arr[a][b] = 'G'
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len(guards); i++ {
                a, b := guards[i][0], guards[i][1]
                x, y := a, b-1
                for y >= 0 && arr[x][y] != 'G' && arr[x][y] != 'W' {
                        arr[x][y] = 'B'
                        y--
                }
                x, y = a-1, b
                for x >= 0 && arr[x][y] != 'G' && arr[x][y] != 'W' {
                        arr[x][y] = 'B'
                        x--
                }
                x, y = a, b+1
                for y < n && arr[x][y] != 'G' && arr[x][y] != 'W' {
                        arr[x][y] = 'B'
                        y++
                }
                x, y = a+1, b
                for x < m && arr[x][y] != 'G' && arr[x][y] != 'W' {
                        arr[x][y] = 'B'
                        x++
                }
        }
        res := 0
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if arr[i][j] == 0 {
                                res++
                        }
                }
        }
        return res
}

# 2
// 顺时针：上右下左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func countUnguarded(m int, n int, guards [][]int, walls [][]int) int {
        arr := make([][]int, m)
        for i := 0; i < m; i++ {
                arr[i] = make([]int, n)
```

(续下页)

(接上页)

```
        }
        for i := 0; i < len(walls); i++ {
                a, b := walls[i][0], walls[i][1]
                arr[a][b] = 1
        }
        for i := 0; i < len(guards); i++ {
                a, b := guards[i][0], guards[i][1]
                arr[a][b] = 2
        }
        for i := 0; i < len(guards); i++ {
                a, b := guards[i][0], guards[i][1]
                for k := 0; k < 4; k++ {
                        x, y := a+dx[k], b+dy[k]
                        for 0 <= x && x < m && 0 <= y && y < n && arr[x][y] != 1 &&␣
↪arr[x][y] != 2 {
                                arr[x][y] = 3
                                x, y = x+dx[k], y+dy[k]
                        }
                }
        }
        res := 0
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if arr[i][j] == 0 {
                                res++
                        }
                }
        }
        return res
}
```

68.23 2260. 必须拿起的最小连续卡牌数 (1)

- 题目

```
给你一个整数数组 cards ，其中 cards[i] 表示第 i 张卡牌的 值␣
↪。如果两张卡牌的值相同，则认为这一对卡牌 匹配 。
返回你必须拿起的最小连续卡牌数，以使在拿起的卡牌中有一对匹配的卡牌。如果无法得到一对匹配的卡牌，返回␣
↪-1 。
示例 1：输入：cards = [3,4,2,3,4,7] 输出：4
解释：拿起卡牌 [3,4,2,3] 将会包含一对值为 3 的匹配卡牌。注意，拿起 [4,2,3,4]␣
↪也是最优方案。
```

(续下页)

(接上页)

```
示例 2: 输入: cards = [1,0,5,3] 输出: -1
解释: 无法找出含一对匹配卡牌的一组连续卡牌。
提示: 1 <= cards.length <= 105
0 <= cards[i] <= 106
```

- 解题思路

```
func minimumCardPickup(cards []int) int {
        n := len(cards)
        res := n + 1
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                if v, ok := m[cards[i]]; ok && i-v+1 < res {
                        res = i - v + 1
                }
                m[cards[i]] = i
        }
        if res == n+1 {
                return -1
        }
        return res
}
```

68.24 2261. 含最多 K 个可整除元素的子数组 (2)

- 题目

```
给你一个整数数组 nums 和两个整数 k 和 p␣
↪, 找出并返回满足要求的不同的子数组数, 要求子数组中最多 k 个可被 p 整除的元素。
如果满足下述条件之一, 则认为数组 nums1 和 nums2 是 不同 数组:
两数组长度 不同 , 或者
存在 至少 一个下标 i 满足 nums1[i] != nums2[i] 。
子数组 定义为: 数组中的连续元素组成的一个 非空 序列。
示例 1: 输入: nums = [2,3,3,2,2], k = 2, p = 2 输出: 11
解释: 位于下标 0、3 和 4 的元素都可以被 p = 2 整除。
共计 11 个不同子数组都满足最多含 k = 2 个可以被 2 整除的元素:
[2]、[2,3]、[2,3,3]、[2,3,3,2]、[3]、[3,3]、[3,3,2]、[3,3,2,2]、[3,2]、[3,2,2] 和 [2,
↪2] 。
注意, 尽管子数组 [2] 和 [3] 在 nums 中出现不止一次, 但统计时只计数一次。
子数组 [2,3,3,2,2] 不满足条件, 因为其中有 3 个元素可以被 2 整除。
示例 2: 输入: nums = [1,2,3,4], k = 4, p = 1 输出: 10
解释: nums 中的所有元素都可以被 p = 1 整除。
```

(续下页)

(接上页)

```
此外，nums 中的每个子数组都满足最多 4 个元素可以被 1 整除。
因为所有子数组互不相同，因此满足所有限制条件的子数组总数为 10 。
提示：1 <= nums.length <= 200
1 <= nums[i], p <= 200
1 <= k <= nums.length
```

- 解题思路

```
func countDistinct(nums []int, k int, p int) int {
        m := make(map[[200]int]bool)
        n := len(nums)
        for i := 0; i < n; i++ { // 左边界
                count := 0
                for j := i + 1; j <= n; j++ { // 右边界
                        if nums[j-1]%p == 0 {
                                count++
                        }
                        if count <= k {
                                temp := [200]int{}
                                for k := i; k < j; k++ {
                                        temp[k-i] = nums[k]
                                }
                                m[temp] = true
                        }
                }
        }
        return len(m)
}

# 2
func countDistinct(nums []int, k int, p int) int {
        m := make(map[[200]int]bool)
        n := len(nums)
        for i := 0; i < n; i++ { // 左边界
                count := 0
                temp := [200]int{}
                for j := i; j < n; j++ { // 右边界
                        if nums[j]%p == 0 {
                                count++
                        }
                        temp[j-i] = nums[j]
                        if count <= k {
                                m[temp] = true
                        }
```

(续下页)

(接上页)

```
                }
        }
        return len(m)
}
```

68.25 2265. 统计值等于子树平均值的节点数 (2)

- 题目

```
给你一棵二叉树的根节点 root ，找出并返回满足要求的节点数，要求节点的值等于其 子树
↪中值的 平均值 。
注意：n 个元素的平均值可以由 n 个元素 求和 然后再除以 n ，并 向下舍入 到最近的整数。
root 的 子树 由 root 和它的所有后代组成。
示例 1：输入：root = [4,8,5,0,1,null,6] 输出：5
解释：对值为 4 的节点：子树的平均值 (4 + 8 + 5 + 0 + 1 + 6) / 6 = 24 / 6 = 4 。
对值为 5 的节点：子树的平均值 (5 + 6) / 2 = 11 / 2 = 5 。
对值为 0 的节点：子树的平均值 0 / 1 = 0 。
对值为 1 的节点：子树的平均值 1 / 1 = 1 。
对值为 6 的节点：子树的平均值 6 / 1 = 6 。
示例 2：输入：root = [1] 输出：1
解释：对值为 1 的节点：子树的平均值 1 / 1 = 1。
提示：树中节点数目在范围 [1, 1000] 内
0 <= Node.val <= 1000
```

- 解题思路

```
var res int

func averageOfSubtree(root *TreeNode) int {
        res = 0
        dfs(root)
        return res
}

func dfs(root *TreeNode) (sum, count int) {
        sum, count = root.Val, 1
        if root.Left != nil {
                sL, cL := dfs(root.Left)
                sum = sum + sL
                count = count + cL
        }
        if root.Right != nil {
```

(续下页)

(接上页)

```
                sR, cR := dfs(root.Right)
                sum = sum + sR
                count = count + cR
        }
        if sum/count == root.Val {
                res++
        }
        return sum, count
}

# 2
var res int

func averageOfSubtree(root *TreeNode) int {
        res = 0
        dfs(root)
        return res
}

func dfs(root *TreeNode) (sum, count int) {
        if root == nil {
                return 0, 0
        }
        sL, cL := dfs(root.Left)
        sR, cR := dfs(root.Right)
        sum = root.Val + sL + sR
        count = 1 + cL + cR
        if sum/count == root.Val {
                res++
        }
        return sum, count
}
```

68.26 2266. 统计打字方案数

68.26.1 题目

```
Alice 在给 Bob 用手机打字。数字到字母的 对应如下图所示。
为了 打出一个字母，Alice 需要 按对应字母 i次，i是该字母在这个按键上所处的位置。
比方说，为了按出字母's'，Alice 需要按'7'四次。类似的， Alice 需要按'5'两次得到字母'k
↪'。
```

(续下页)

(接上页)

```
注意，数字'0' 和'1'不映射到任何字母，所以Alice 不使用它们。
但是，由于传输的错误，Bob 没有收到 Alice 打字的字母信息，反而收到了 按键的字符串信息。
比方说，Alice 发出的信息为"bob"，Bob 将收到字符串"2266622"。
给你一个字符串pressedKeys，表示 Bob 收到的字符串，请你返回 Alice总共可能发出多少种文字信息。
由于答案可能很大，将它对109 + 7取余 后返回。
示例 1：输入：pressedKeys = "22233" 输出：8
解释：Alice 可能发出的文字信息包括：
"aaadd", "abdd", "badd", "cdd", "aaae", "abe", "bae" 和 "ce" 。
由于总共有 8 种可能的信息，所以我们返回 8 。
示例 2：输入：pressedKeys = "222222222222222222222222222222222222" 输出：82876089
解释：总共有 2082876103 种 Alice 可能发出的文字信息。
由于我们需要将答案对 109 + 7 取余，所以我们返回 2082876103 % (109 + 7) = 82876089 。
提示：1 <= pressedKeys.length <= 105
pressedKeys 只包含数字'2'到'9'。
```

68.26.2 解题思路

68.27 2270. 分割数组的方案数 (1)

- 题目

```
给你一个下标从 0开始长度为 n的整数数组nums。
如果以下描述为真，那么 nums在下标 i处有一个 合法的分割：
前i + 1个元素的和 大于等于剩下的n - i - 1个元素的和。
下标 i的右边 至少有一个元素，也就是说下标i满足0 <= i < n - 1。
请你返回nums中的合法分割方案数。
示例 1：输入：nums = [10,4,-8,7] 输出：2
解释：总共有 3 种不同的方案可以将 nums 分割成两个非空的部分：
- 在下标 0 处分割 nums 。那么第一部分为 [10] ，和为 10 。第二部分为 [4,-8,7] ，和为 3。
因为 10 >= 3 ，所以 i = 0 是一个合法的分割。
- 在下标 1 处分割 nums 。那么第一部分为 [10,4] ，和为 14 。第二部分为 [-8,7] ，和为 -1 。
因为 14 >= -1 ，所以 i = 1 是一个合法的分割。
- 在下标 2 处分割 nums 。那么第一部分为 [10,4,-8] ，和为 6 。第二部分为 [7] ，和为 7。
因为 6 < 7 ，所以 i = 2 不是一个合法的分割。
```

(续下页)

(接上页)

```
所以 nums 中总共合法分割方案受为 2 。
示例 2: 输入: nums = [2,3,1,0] 输出: 2
解释: 总共有 2 种 nums 的合法分割:
- 在下标 1 处分割 nums 。那么第一部分为 [2,3] ，和为 5 。第二部分为 [1,0] ，和为 1 。
因为 5 >= 1 ，所以 i = 1 是一个合法的分割。
- 在下标 2 处分割 nums 。那么第一部分为 [2,3,1] ，和为 6 。第二部分为 [0] ，和为 0 。
因为 6 >= 0 ，所以 i = 2 是一个合法的分割。
提示: 2 <= nums.length <= 105
-105 <= nums[i] <= 105
```

- 解题思路

```
func waysToSplitArray(nums []int) int {
        res := 0
        sum, temp := 0, 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        for i := 0; i < len(nums)-1; i++ {
                temp = temp + nums[i]
                sum = sum - nums[i]
                if temp >= sum {
                        res++
                }
        }
        return res
}
```

68.28 2271. 毯子覆盖的最多白色砖块数 (3)

- 题目

```
给你一个二维整数数组tiles，其中tiles[i] = [li, ri]，表示所有在li <= j <=␣
↪ri之间的每个瓷砖位置 j都被涂成了白色。
同时给你一个整数carpetLen，表示可以放在任何位置的一块毯子。
请你返回使用这块毯子，最多可以盖住多少块瓷砖。
示例 1: 输入: tiles = [[1,5],[10,11],[12,18],[20,25],[30,32]], carpetLen = 10 输出: 9
解释: 将毯子从瓷砖 10 开始放置。
总共覆盖 9 块瓷砖，所以返回 9 。
注意可能有其他方案也可以覆盖 9 块瓷砖。
可以看出，瓷砖无法覆盖超过 9 块瓷砖。
示例 2: 输入: tiles = [[10,11],[1,1]], carpetLen = 2 输出: 2
```

(续下页)

(接上页)

```
解释：将毯子从瓷砖 10 开始放置。
总共覆盖 2 块瓷砖，所以我们返回 2 。
提示：1 <= tiles.length <= 5 * 104
tiles[i].length == 2
1 <= li <= ri <= 109
1 <= carpetLen <= 109
tiles互相 不会重叠。
```

- 解题思路

```
func maximumWhiteTiles(tiles [][]int, carpetLen int) int {
        res := 0
        sort.Slice(tiles, func(i, j int) bool {
                return tiles[i][0] < tiles[j][0]
        })
        count := 0
        left := 0
        for right := 0; right < len(tiles); right++ {
                l, r := tiles[right][0], tiles[right][1]
                count = count + r - l + 1
                for tiles[left][1]+carpetLen-1 < r { // 左边瓷砖右边界+毯子␣
↪无法覆盖右边瓷砖右边界：left指针右移
                        count = count - (tiles[left][1] - tiles[left][0] + 1)
                        left++
                }
                leftNum := r - tiles[left][0] - carpetLen + 1 //␣
↪未覆盖的数量：左边不在瓷砖内部的数量
                if leftNum < 0 {                                //␣
↪毯子够长：毯子左侧有多余
                        leftNum = 0
                }
                res = max(res, count-leftNum)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
func maximumWhiteTiles(tiles [][]int, carpetLen int) int {
        res := 0
        sort.Slice(tiles, func(i, j int) bool {
                return tiles[i][0] < tiles[j][0]
        })
        count := 0
        right := 0
        n := len(tiles)
        for left := 0; left < n; left++ {
                for right < n && tiles[right][1]-tiles[left][0]+1 < carpetLen { //␣
↪左边瓷砖左边界+毯子 能覆盖右边瓷砖右边界：right指针右移
                        count = count + (tiles[right][1] - tiles[right][0] + 1)
                        right++
                }
                if right < n {
                        rightNum := tiles[left][0] + carpetLen - tiles[right][0] //␣
↪右边覆盖的数量：右边在瓷砖内部的数量
                        if rightNum < 0 {                                        //␣
↪毯子不够长，覆盖不到右边
                                rightNum = 0
                        }
                        res = max(res, count+rightNum)
                } else {
                        res = max(res, count)
                }
                count = count - (tiles[left][1] - tiles[left][0] + 1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maximumWhiteTiles(tiles [][]int, carpetLen int) int {
        res := 0
        sort.Slice(tiles, func(i, j int) bool {
                return tiles[i][0] < tiles[j][0]
        })
```

(续下页)

(接上页)

```
        n := len(tiles)
        arr := make([]int, n+1)

        for i := 0; i < n; i++ {
                v := tiles[i][1] - tiles[i][0] + 1
                arr[i+1] = arr[i] + v
        }
        for i := 0; i < n; i++ {
                right := tiles[i][0] + carpetLen - 1        // 右边界
                index := sort.Search(n, func(j int) bool { // 右边界>=right的下标
                        return tiles[j][1] >= right
                })
                if index >= n {
                        res = max(res, arr[n]-arr[i])
                } else {
                        rightNum := right - tiles[index][0] + 1 //␣
↪右边覆盖的数量：右边在瓷砖内部的数量
                        if rightNum <= 0 {                      //␣
↪毯子不够长，覆盖不到右边
                                rightNum = 0
                        }
                        res = max(res, arr[index]-arr[i]+rightNum)
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

68.29 2274. 不含特殊楼层的最大连续楼层数 (2)

- 题目

```
Alice 管理着一家公司，并租用大楼的部分楼层作为办公空间。Alice 决定将一些楼层作为␣
↪特殊楼层 ，仅用于放松。
给你两个整数 bottom 和 top ，表示 Alice 租用了从 bottom 到 top（含 bottom 和 top␣
↪在内）的所有楼层。
```

(续下页)

(接上页)

```
另给你一个整数数组 special ，其中 special[i] 表示 Alice 指定用于放松的特殊楼层。
返回不含特殊楼层的 最大 连续楼层数。
示例 1：输入：bottom = 2, top = 9, special = [4,6] 输出：3
解释：下面列出的是不含特殊楼层的连续楼层范围：
- (2, 3) ，楼层数为 2 。
- (5, 5) ，楼层数为 1 。
- (7, 9) ，楼层数为 3 。
因此，返回最大连续楼层数 3 。
示例 2：输入：bottom = 6, top = 8, special = [7,6,8] 输出：0
解释：每层楼都被规划为特殊楼层，所以返回 0 。
提示 1 <= special.length <= 105
1 <= bottom <= special[i] <= top <= 109
special 中的所有值 互不相同
```

- 解题思路

```
func maxConsecutive(bottom int, top int, special []int) int {
        res := 0
        sort.Ints(special)
        for i := 0; i < len(special); i++ {
                if i == 0 {
                        res = max(res, special[i]-bottom) // 跟bottom比
                } else {
                        res = max(res, special[i]-special[i-1]-1)
                }
                if i == len(special)-1 { // top跟最大比
                        res = max(res, top-special[i])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxConsecutive(bottom int, top int, special []int) int {
        res := 0
        special = append(special, bottom-1, top+1) // 补齐
        sort.Ints(special)
```

(续下页)

(接上页)

```
        for i := 0; i < len(special)-1; i++ {
                res = max(res, special[i+1]-special[i]-1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

68.30 2275. 按位与结果大于零的最长组合 (2)

- 题目

```
对数组nums 执行 按位与 相当于对数组nums 中的所有整数执行 按位与 。
例如，对 nums = [1, 5, 3] 来说，按位与等于 1 & 5 & 3 = 1 。
同样，对 nums = [7] 而言，按位与等于 7 。
给你一个正整数数组 candidates 。计算 candidates 中的数字每种组合下 按位与 的结果。
candidates 中的每个数字在每种组合中只能使用 一次 。
返回按位与结果大于 0 的 最长 组合的长度。
示例 1：输入：candidates = [16,17,71,62,12,24,14] 输出：4
解释：组合 [16,17,62,24] 的按位与结果是 16 & 17 & 62 & 24 = 16 > 0 。
组合长度是 4 。
可以证明不存在按位与结果大于 0 且长度大于 4 的组合。
注意，符合长度最大的组合可能不止一种。
例如，组合 [62,12,24,14] 的按位与结果是 62 & 12 & 24 & 14 = 8 > 0 。
示例 2：输入：candidates = [8,8] 输出：2
解释：最长组合是 [8,8] ，按位与结果 8 & 8 = 8 > 0 。
组合长度是 2 ，所以返回 2 。
提示：1 <= candidates.length <= 105
1 <= candidates[i] <= 107
```

- 解题思路

```
func largestCombination(candidates []int) int {
        arr := [32]int{}
        for i := 0; i < len(candidates); i++ {
                v := candidates[i]
                j := 0
```

(续下页)

(接上页)

```
                for v > 0 {
                        if v%2 == 1 {
                                arr[j]++
                        }
                        v = v / 2
                        j++
                }
        }
        res := 0
        for i := 0; i < 32; i++ {
                res = max(res, arr[i]) // 计算二进制1出现最多的次数
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func largestCombination(candidates []int) int {
        res := 0
        for i := 0; i < 32; i++ {
                count := 0
                for j := 0; j < len(candidates); j++ {
                        if (candidates[j]>>i)%2 == 1 {
                                count++
                        }
                }
                res = max(res, count) // 计算二进制1出现最多的次数
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

68.31 2279. 装满石头的背包的最大数量 (1)

- 题目

```
现有编号从0 到 n - 1 的 n 个背包。给你两个下标从 0 开始的整数数组 capacity 和 rocks 。
第 i 个背包最大可以装 capacity[i] 块石头，当前已经装了 rocks[i] 块石头。
另给你一个整数 additionalRocks ，表示你可以放置的额外石头数量，石头可以往 任意␣
↪背包中放置。
请你将额外的石头放入一些背包中，并返回放置后装满石头的背包的 最大 数量。
示例 1: 输入: capacity = [2,3,4,5], rocks = [1,2,4,4], additionalRocks = 2 输出: 3
解释: 1 块石头放入背包 0 ，1 块石头放入背包 1 。
每个背包中的石头总数是 [2,3,4,4] 。
背包 0 、背包 1 和 背包 2 都装满石头。
总计 3 个背包装满石头，所以返回 3 。
可以证明不存在超过 3 个背包装满石头的情况。
注意，可能存在其他放置石头的方案同样能够得到 3 这个结果。
示例 2: 输入: capacity = [10,2,2], rocks = [2,2,0], additionalRocks = 100 输出: 3
解释: 8 块石头放入背包 0 ，2 块石头放入背包 2 。
每个背包中的石头总数是 [10,2,2] 。
背包 0 、背包 1 和背包 2 都装满石头。
总计 3 个背包装满石头，所以返回 3 。
可以证明不存在超过 3 个背包装满石头的情况。
注意，不必用完所有的额外石头。
提示: n == capacity.length == rocks.length
1 <= n <= 5 * 104
1 <= capacity[i] <= 109
0 <= rocks[i] <= capacity[i]
1 <= additionalRocks <= 109
```

- 解题思路

```
func maximumBags(capacity []int, rocks []int, additionalRocks int) int {
        res := 0
        for i := 0; i < len(capacity); i++ {
                capacity[i] = capacity[i] - rocks[i]
        }
        sort.Ints(capacity)
        for i := 0; i < len(capacity); i++ {
                if capacity[i] <= additionalRocks {
                        res++
                        additionalRocks = additionalRocks - capacity[i]
                } else {
                        break
                }
```

(续下页)

(接上页)

```
        }
        return res
}
```

68.32 2280. 表示一个折线图的最少线段数 (1)

- 题目

```
给你一个二维整数数组stockPrices ，其中stockPrices[i] = [dayi,␣
↪pricei]表示股票在dayi的价格为pricei。
折线图是一个二维平面上的若干个点组成的图，横坐标表示日期，纵坐标表示价格，折线图由相邻的点连接而成。
请你返回要表示一个折线图所需要的 最少线段数。
示例 1：输入：stockPrices = [[1,7],[2,6],[3,5],[4,4],[5,4],[6,3],[7,2],[8,1]] 输出：3
解释：上图为输入对应的图，横坐标表示日期，纵坐标表示价格。
以下 3 个线段可以表示折线图：
- 线段 1 （红色）从 (1,7) 到 (4,4) ，经过 (1,7) ，(2,6) ，(3,5) 和 (4,4) 。
- 线段 2 （蓝色）从 (4,4) 到 (5,4) 。
- 线段 3 （绿色）从 (5,4) 到 (8,1) ，经过 (5,4) ，(6,3) ，(7,2) 和 (8,1) 。
可以证明，无法用少于 3 条线段表示这个折线图。
示例 2：输入：stockPrices = [[3,4],[1,2],[7,8],[2,3]] 输出：1
解释：如上图所示，折线图可以用一条线段表示。
提示：1 <= stockPrices.length <= 105
stockPrices[i].length == 2
1 <= dayi, pricei <= 109
所有dayi互不相同。
```

- 解题思路

```
func minimumLines(stockPrices [][]int) int {
        sort.Slice(stockPrices, func(i, j int) bool {
                return stockPrices[i][0] < stockPrices[j][0]
        })
        res := 0
        dx, dy := 0, 0
        for i := 1; i < len(stockPrices); i++ {
                x := stockPrices[i][0] - stockPrices[i-1][0]
                y := stockPrices[i][1] - stockPrices[i-1][1]
                if i == 1 {
                        dx, dy = x, y
                        res++
                } else if dx*y != x*dy { // y/x = dy/dx =>  dx*y == x*dy
                        dx, dy = x, y
```

(续下页)

(接上页)

```
                        res++
                }
        }
        return res
}
```

68.33 2284. 最多单词数的发件人 (2)

- 题目

```
给你一个聊天记录，共包含 n条信息。给你两个字符串数组messages 和senders，
其中messages[i]是senders[i]发出的一条信息。
一条 信息是若干用单个空格连接的 单词，信息开头和结尾不会有多余空格。
发件人的 单词计数是这个发件人总共发出的单词数。注意，一个发件人可能会发出多于一条信息。
请你返回发出单词数 最多的发件人名字。如果有多个发件人发出最多单词数，请你返回字典序最大的名字。
注意：字典序里，大写字母小于小写字母。"Alice" 和"alice"是不同的名字。
示例 1：输入：messages = ["Hello userTwooo","Hi userThree","Wonderful day Alice",
"Nice day userThree"],
senders = ["Alice","userTwo","userThree","Alice"]输出："Alice"
解释：Alice 总共发出了 2 + 3 = 5 个单词。
userTwo 发出了 2 个单词。
userThree 发出了 3 个单词。
由于 Alice 发出单词数最多，所以我们返回 "Alice" 。
示例 2：输入：messages = ["How is leetcode for everyone","Leetcode is useful for
practice"],
senders = ["Bob","Charlie"]输出："Charlie"
解释：Bob 总共发出了 5 个单词。
Charlie 总共发出了 5 个单词。
由于最多单词数打平，返回字典序最大的名字，也就是 Charlie 。
提示：n == messages.length == senders.length
1 <= n <= 104
1 <= messages[i].length <= 100
1 <= senders[i].length <= 10
messages[i]包含大写字母、小写字母和' '。
messages[i]中所有单词都由 单个空格隔开。
messages[i]不包含前导和后缀空格。
senders[i]只包含大写英文字母和小写英文字母。
```

- 解题思路

```
func largestWordCount(messages []string, senders []string) string {
        res := ""
        maxValue := 0
        m := make(map[string]int)
        for i := 0; i < len(senders); i++ {
                count := len(strings.Fields(messages[i]))
                m[senders[i]] = m[senders[i]] + count
                if m[senders[i]] > maxValue {
                        maxValue = m[senders[i]]
                        res = senders[i]
                } else if m[senders[i]] == maxValue && senders[i] > res {
                        res = senders[i]
                }
        }
        return res
}

# 2
func largestWordCount(messages []string, senders []string) string {
        res := ""
        maxValue := 0
        m := make(map[string]int)
        for i := 0; i < len(senders); i++ {
                count := strings.Count(messages[i], " ") + 1
                m[senders[i]] = m[senders[i]] + count
                if m[senders[i]] > maxValue {
                        maxValue = m[senders[i]]
                        res = senders[i]
                } else if m[senders[i]] == maxValue && senders[i] > res {
                        res = senders[i]
                }
        }
        return res
}
```

68.34 2285. 道路的最大总重要性 (1)

- 题目

```
给你一个整数n，表示一个国家里的城市数目。城市编号为0到n - 1。
给你一个二维整数数组roads，其中roads[i] = [ai, bi]表示城市ai和bi之间有一条双向道路。
你需要给每个城市安排一个从 1到 n之间的整数值，且每个值只能被使用 一次。
```

(续下页)

(接上页)

```
道路的 重要性定义为这条道路连接的两座城市数值 之和。
请你返回在最优安排下，所有道路重要性 之和 最大为多少。
示例 1：输入：n = 5, roads = [[0,1],[1,2],[2,3],[0,2],[1,3],[2,4]] 输出：43
解释：上图展示了国家图和每个城市被安排的值 [2,4,5,3,1] 。
- 道路 (0,1) 重要性为 2 + 4 = 6 。
- 道路 (1,2) 重要性为 4 + 5 = 9 。
- 道路 (2,3) 重要性为 5 + 3 = 8 。
- 道路 (0,2) 重要性为 2 + 5 = 7 。
- 道路 (1,3) 重要性为 4 + 3 = 7 。
- 道路 (2,4) 重要性为 5 + 1 = 6 。
所有道路重要性之和为 6 + 9 + 8 + 7 + 7 + 6 = 43 。
可以证明，重要性之和不可能超过 43 。
示例 2：输入：n = 5, roads = [[0,3],[2,4],[1,3]] 输出：20
解释：上图展示了国家图和每个城市被安排的值 [4,3,2,5,1] 。
- 道路 (0,3) 重要性为 4 + 5 = 9 。
- 道路 (2,4) 重要性为 2 + 1 = 3 。
- 道路 (1,3) 重要性为 3 + 5 = 8 。
所有道路重要性之和为 9 + 3 + 8 = 20 。
可以证明，重要性之和不可能超过 20 。
提示：2 <= n <= 5 * 104
1 <= roads.length <= 5 * 104
roads[i].length == 2
0 <= ai, bi <= n - 1
ai != bi
没有重复道路。
```

- 解题思路

```
func maximumImportance(n int, roads [][]int) int64 {
        arr := make([]int, n)
        for i := 0; i < len(roads); i++ {
                a, b := roads[i][0], roads[i][1]
                arr[a]++
                arr[b]++
        }
        sort.Ints(arr)
        res := int64(0)
        for i := 1; i <= n; i++ {
                res = res + int64(arr[i-1])*int64(i)
        }
        return res
}
```

68.35 2288. 价格减免 (2)

- 题目

```
句子␣
↪是由若干个单词组成的字符串，单词之间用单个空格分隔，其中每个单词可以包含数字、小写字母、和美元符号␣
↪'$' 。
如果单词的形式为美元符号后跟着一个非负实数，那么这个单词就表示一个价格。
例如 "$100"、"$23" 和 "$6.75" 表示价格，而 "100"、"$" 和 "2$3" 不是。
注意：本题输入中的价格均为整数。
给你一个字符串 sentence 和一个整数 discount 。对于每个表示价格的单词，
都在价格的基础上减免 discount% ，并 更新␣
↪该单词到句子中。所有更新后的价格应该表示为一个 恰好保留小数点后两位 的数字。
返回表示修改后句子的字符串。
示例 1：输入：sentence = "there are $1 $2 and 5$ candies in the shop", discount = 50
输出："there are $0.50 $1.00 and 5$ candies in the shop"
解释：表示价格的单词是 "$1" 和 "$2" 。
- "$1" 减免 50% 为 "$0.50" ，所以 "$1" 替换为 "$0.50" 。
- "$2" 减免 50% 为 "$1" ，所以 "$1" 替换为 "$1.00" 。
示例 2：输入：sentence = "1 2 $3 4 $5 $6 7 8$ $9 $10$", discount = 100
输出："1 2 $0.00 4 $0.00 $0.00 7 8$ $0.00 $10$"
解释：任何价格减免 100% 都会得到 0 。
表示价格的单词分别是 "$3"、"$5"、"$6" 和 "$9"。
每个单词都替换为 "$0.00"。
提示：1 <= sentence.length <= 105
sentence 由小写英文字母、数字、' ' 和'$' 组成
sentence 不含前导和尾随空格
sentence 的所有单词都用单个空格分隔
所有价格都是 正 整数且不含前导零
所有价格 最多 为 10 位数字
0 <= discount <= 100
```

- 解题思路

```
func discountPrices(sentence string, discount int) string {
        arr := strings.Fields(sentence)
        for i := 0; i < len(arr); i++ {
                if arr[i][0] == '$' && len(arr[i]) > 1 &&
                        strings.Count(arr[i], "$") == 1 && check(arr[i][1:]) == true {
                        v, _ := strconv.Atoi(arr[i][1:])
                        vf := float64(v) * float64(100-discount) / 100
                        arr[i] = fmt.Sprintf("$%0.2f", vf)
                }
        }
```

(续下页)

(接上页)

```
        return strings.Join(arr, " ")
}

func check(s string) bool {
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        continue
                }
                return false
        }
        return true
}

# 2
func discountPrices(sentence string, discount int) string {
        arr := strings.Fields(sentence)
        for i := 0; i < len(arr); i++ {
                if arr[i][0] == '$' {
                        v, err := strconv.Atoi(arr[i][1:])
                        if err == nil {
                                vf := float64(v) * float64(100-discount) / 100
                                arr[i] = fmt.Sprintf("$%0.2f", vf)
                        }
                }
        }
        return strings.Join(arr, " ")
}
```

68.36 2294. 划分数组使最大差为 K(1)

- 题目

```
给你一个整数数组 nums 和一个整数 k 。你可以将 nums 划分成一个或多个 子序列 ，使 nums␣
↪中的每个元素都 恰好 出现在一个子序列中。
在满足每个子序列中最大值和最小值之间的差值最多为 k 的前提下，返回需要划分的 最少␣
↪子序列数目。
子序列␣
↪本质是一个序列，可以通过删除另一个序列中的某些元素（或者不删除）但不改变剩下元素的顺序得到。
示例 1：输入：nums = [3,6,1,2,5], k = 2 输出：2
解释：可以将 nums 划分为两个子序列 [3,1,2] 和 [6,5] 。
第一个子序列中最大值和最小值的差值是 3 - 1 = 2 。
第二个子序列中最大值和最小值的差值是 6 - 5 = 1 。
```

(续下页)

(接上页)

```
由于创建了两个子序列，返回 2 。可以证明需要划分的最少子序列数目就是 2 。
示例 2：输入：nums = [1,2,3], k = 1输出：2
解释：可以将 nums 划分为两个子序列 [1,2] 和 [3] 。
第一个子序列中最大值和最小值的差值是 2 - 1 = 1 。
第二个子序列中最大值和最小值的差值是 3 - 3 = 0 。
由于创建了两个子序列，返回 2 。注意，另一种最优解法是将 nums 划分成子序列 [1] 和 [2,
↪3] 。
示例 3：输入：nums = [2,2,4,5], k = 0 输出：3
解释：可以将 nums 划分为三个子序列 [2,2]、[4] 和 [5] 。
第一个子序列中最大值和最小值的差值是 2 - 2 = 0 。
第二个子序列中最大值和最小值的差值是 4 - 4 = 0 。
第三个子序列中最大值和最小值的差值是 5 - 5 = 0 。
由于创建了三个子序列，返回 3 。可以证明需要划分的最少子序列数目就是 3 。
提示：1 <= nums.length <= 105
0 <= nums[i] <= 105
0 <= k <= 105
```

- 解题思路

```
func partitionArray(nums []int, k int) int {
        res := 1
        sort.Ints(nums)
        minValue := nums[0]
        for i := 0; i < len(nums); i++ {
                // 贪心：依次从小往大开始，把连续的、满足最大最小差值<=k划分为1组
                // 只需要考虑一组内的最大最小值，无需考虑顺序（划分为1组即可）
                if nums[i]-minValue > k {
                        res++
                        minValue = nums[i]
                }
        }
        return res
}
```

68.37 2295. 替换数组中的元素 (1)

- 题目

```
给你一个下标从 0开始的数组nums，它包含 n个 互不相同的正整数。
请你对这个数组执行 m个操作，在第 i个操作中，你需要将数字operations[i][0]␣
↪替换成operations[i][1]。
题目保证在第 i个操作中：
```

(续下页)

(接上页)

```
operations[i][0]在nums中存在。
operations[i][1]在nums中不存在。
请你返回执行完所有操作后的数组。
示例 1：输入：nums = [1,2,4,6], operations = [[1,3],[4,7],[6,1]] 输出：[3,2,7,1]
解释：我们对 nums 执行以下操作：
- 将数字 1 替换为 3 。nums 变为 [3,2,4,6] 。
- 将数字 4 替换为 7 。nums 变为 [3,2,7,6] 。
- 将数字 6 替换为 1 。nums 变为 [3,2,7,1] 。
返回最终数组 [3,2,7,1] 。
示例 2：输入：nums = [1,2], operations = [[1,3],[2,1],[3,2]] 输出：[2,1]
解释：我们对 nums 执行以下操作：
- 将数字 1 替换为 3 。nums 变为 [3,2] 。
- 将数字 2 替换为 1 。nums 变为 [3,1] 。
- 将数字 3 替换为 2 。nums 变为 [2,1] 。
返回最终数组 [2,1] 。
提示：n == nums.length
m == operations.length
1 <= n, m <= 105
nums中所有数字 互不相同。
operations[i].length == 2
1 <= nums[i], operations[i][0], operations[i][1] <= 106
在执行第i 个操作时，operations[i][0]在nums中存在。
在执行第i个操作时，operations[i][1]在nums中不存在。
```

- 解题思路

```
func arrayChange(nums []int, operations [][]int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = i
        }
        for i := 0; i < len(operations); i++ {
                a, b := operations[i][0], operations[i][1]
                index := m[a]
                delete(m, a)
                m[b] = index
                nums[index] = b
        }
        return nums
}

# 2
func arrayChange(nums []int, operations [][]int) []int {
        m := make(map[int]int)
```

(续下页)

(接上页)

```
        for i := len(operations) - 1; i >= 0; i-- {
                a, b := operations[i][0], operations[i][1]
                if v, ok := m[b]; ok {
                        b = v // 在后面出现过，使用最后的结果
                }
                m[a] = b
        }
        for i := 0; i < len(nums); i++ {
                if v, ok := m[nums[i]]; ok {
                        nums[i] = v
                }
        }
        return nums
}
```

68.38 2300. 咒语和药水的成功对数 (2)

- 题目

```
给你两个正整数数组spells 和potions，长度分别为n 和m，
其中spells[i]表示第i个咒语的能量强度，potions[j]表示第j瓶药水的能量强度。
同时给你一个整数success。一个咒语和药水的能量强度 相乘␣
↪如果大于等于success，那么它们视为一对成功的组合。
请你返回一个长度为 n的整数数组pairs，其中pairs[i]是能跟第 i个咒语成功组合的 药水数目。
示例 1：输入：spells = [5,1,3], potions = [1,2,3,4,5], success = 7 输出：[4,0,3]
解释：- 第 0 个咒语：5 * [1,2,3,4,5] = [5,10,15,20,25] 。总共 4 个成功组合。
- 第 1 个咒语：1 * [1,2,3,4,5] = [1,2,3,4,5] 。总共 0 个成功组合。
- 第 2 个咒语：3 * [1,2,3,4,5] = [3,6,9,12,15] 。总共 3 个成功组合。
所以返回 [4,0,3] 。
示例 2：输入：spells = [3,1,2], potions = [8,5,8], success = 16 输出：[2,0,2]
解释：- 第 0 个咒语：3 * [8,5,8] = [24,15,24] 。总共 2 个成功组合。
- 第 1 个咒语：1 * [8,5,8] = [8,5,8] 。总共 0 个成功组合。
- 第 2 个咒语：2 * [8,5,8] = [16,10,16] 。总共 2 个成功组合。
所以返回 [2,0,2] 。
提示：n == spells.length
m == potions.length
1 <= n, m <= 105
1 <= spells[i], potions[i] <= 105
1 <= success <= 1010
```

- 解题思路

```
func successfulPairs(spells []int, potions []int, success int64) []int {
        n := len(spells)
        res := make([]int, n)
        sort.Ints(potions)
        for i := 0; i < n; i++ {
                index := sort.Search(len(potions), func(j int) bool {
                        return int64(potions[j])*int64(spells[i]) >= success
                })
                res[i] = len(potions) - index
        }
        return res
}

# 2
func successfulPairs(spells []int, potions []int, success int64) []int {
        sort.Ints(potions)
        for i := 0; i < len(spells); i++ {
                // xy>= success => y >= success/x => y > floor((success-1)/x)
                spells[i] = len(potions) - sort.SearchInts(potions, int(success-1)/
↪spells[i]+1)
        }
        return spells
}
```

CHAPTER 69

2201-2300-Hard

69.1 2209. 用地毯覆盖后的最少白色砖块 (2)

- 题目

```
给你一个下标从0开始的 二进制字符串floor，它表示地板上砖块的颜色。
floor[i] = '0'表示地板上第i块砖块的颜色是 黑色。
floor[i] = '1'表示地板上第i块砖块的颜色是 白色。
同时给你numCarpets␣
↪和carpetLen。你有numCarpets条黑色的地毯，每一条黑色的地毯长度都为carpetLen块砖块。
请你使用这些地毯去覆盖砖块，使得未被覆盖的剩余 白色砖块的数目␣
↪最小。地毯相互之间可以覆盖。
请你返回没被覆盖的白色砖块的 最少数目。
示例 1：输入：floor = "10110101", numCarpets = 2, carpetLen = 2 输出：2
解释：上图展示了剩余 2 块白色砖块的方案。
没有其他方案可以使未被覆盖的白色砖块少于 2 块。
示例 2：输入：floor = "11111", numCarpets = 2, carpetLen = 3 输出：0
解释：上图展示了所有白色砖块都被覆盖的一种方案。
注意，地毯相互之间可以覆盖。
提示：1 <= carpetLen <= floor.length <= 1000
floor[i] 要么是'0'，要么是'1'。
1 <= numCarpets <= 1000
```

- 解题思路

```
func minimumWhiteTiles(floor string, numCarpets int, carpetLen int) int {
        n := numCarpets
        m := len(floor)
        dp := make([][]int, n+1) // dp[i][j]=>
↪表示i条毛毯覆盖前j块砖块时白色砖块的最少数目
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m)
        }
        dp[0][0] = int(floor[0] - '0')
        for j := 1; j < m; j++ {
                dp[0][j] = dp[0][j-1] + int(floor[j]-'0')
        }
        for i := 1; i <= n; i++ {
                for j := carpetLen; j < m; j++ { // 遍历覆盖终点
                        // 不覆盖：dp[i][j-1]+int(floor[j]-'0')
                        // 覆盖：dp[i-1][j-carpetLen]（其中j-carpetLen是起点）
                        dp[i][j] = min(dp[i][j-1]+int(floor[j]-'0'), dp[i-1][j-
↪carpetLen])
                }
        }
        return dp[n][m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minimumWhiteTiles(floor string, numCarpets int, carpetLen int) int {
        n := numCarpets
        m := len(floor)
        dp := make([][]int, n+1) // dp[i][j]=>
↪表示i条毛毯覆盖前j块砖块时白色砖块的最少数目
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, m)
        }
        dp[0][0] = int(floor[0] - '0')
        for j := 1; j < m; j++ {
                dp[0][j] = dp[0][j-1] + int(floor[j]-'0')
        }
        for i := 1; i <= n; i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < m; j++ { // 遍历覆盖终点
                        if j < carpetLen {
                                dp[i][j] = 0
                                continue
                        }
                        // 不覆盖：dp[i][j-1]+int(floor[j]-'0')
                        // 覆盖：dp[i-1][j-carpetLen]（其中j-carpetLen是起点）
                        dp[i][j] = min(dp[i][j-1]+int(floor[j]-'0'), dp[i-1][j-
↪carpetLen])
                }
        }
        return dp[n][m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

69.2 2213. 由单个字符重复的最长子字符串

69.2.1 题目

```
给你一个下标从 0 开始的字符串 s 。另给你一个下标从 0 开始、长度为 k 的字符串␣
↪queryCharacters ，
一个下标从 0 开始、长度也是 k 的整数 下标 数组queryIndices ，这两个都用来描述 k␣
↪个查询。
第 i 个查询会将 s 中位于下标 queryIndices[i] 的字符更新为 queryCharacters[i] 。
返回一个长度为 k 的数组 lengths ，
其中 lengths[i] 是在执行第 i 个查询 之后 s 中仅由 单个字符重复 组成的 最长子字符串 的␣
↪长度 。
示例 1：输入：s = "babacc", queryCharacters = "bcb", queryIndices = [1,3,3] 输出：[3,
↪3,4]
解释：- 第 1 次查询更新后 s = "bbbacc" 。由单个字符重复组成的最长子字符串是 "bbb"␣
↪，长度为 3 。
- 第 2 次查询更新后 s = "bbbccc" 。由单个字符重复组成的最长子字符串是 "bbb" 或 "ccc
↪"，长度为 3 。
- 第 3 次查询更新后 s = "bbbbcc" 。由单个字符重复组成的最长子字符串是 "bbbb" ，长度为␣
↪4 。
```

(续下页)

(接上页)

```
因此，返回 [3,3,4] 。
示例 2：输入：s = "abyzz", queryCharacters = "aa", queryIndices = [2,1] 输出：[2,3]
解释：- 第 1 次查询更新后 s = "abazz" 。由单个字符重复组成的最长子字符串是 "zz"␣
↪，长度为 2 。
- 第 2 次查询更新后 s = "aaazz" 。由单个字符重复组成的最长子字符串是 "aaa" ，长度为 3␣
↪。
因此，返回 [2,3] 。
提示：1 <= s.length <= 105
s 由小写英文字母组成
k == queryCharacters.length == queryIndices.length
1 <= k <= 105
queryCharacters 由小写英文字母组成
0 <= queryIndices[i] < s.length
```

69.2.2 解题思路

```
func minimumRounds(tasks []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(tasks); i++ {
                m[tasks[i]]++
        }
        for _, v := range m {
                if v == 1 { // 1个直接返回
                        return -1
                }
                if v%3 == 0 {
                        res = res + v/3
                } else {
                        res = res + v/3 + 1 // %v=1 拆成2+2; %v=2, 拆成3+2
                }
        }
        return res
}
```

69.3 2218. 从栈中取出 K 个硬币的最大面值和 (2)

- 题目

```
一张桌子上总共有 n个硬币 栈。每个栈有 正整数个带面值的硬币。
每一次操作中，你可以从任意一个栈的 顶部取出 1 个硬币，从栈中移除它，并放入你的钱包里。
给你一个列表piles，其中piles[i]是一个整数数组，分别表示第 i个栈里 从顶到底的硬币面值。
同时给你一个正整数k，请你返回在恰好进行k次操作的前提下，你钱包里硬币面值之和最大为多少。
示例 1：输入：piles = [[1,100,3],[7,8,9]], k = 2 输出：101
解释：上图展示了几种选择 k 个硬币的不同方法。
我们可以得到的最大面值为 101 。
示例 2：输入：piles = [[100],[100],[100],[100],[100],[100],[1,1,1,1,1,1,700]], k = 7
↪输出：706
解释：如果我们所有硬币都从最后一个栈中取，可以得到最大面值和。
提示：n == piles.length
1 <= n <= 1000
1 <= piles[i][j] <= 105
1 <= k <= sum(piles[i].length) <= 2000
```

- 解题思路

```
func maxValueOfCoins(piles [][]int, k int) int {
        n := len(piles)
        dp := make([][]int, n+1) // 背包问题：dp[i][j]=>表示i个栈取j个硬币的最大值
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
        }
        for i := 1; i <= n; i++ {
                length := len(piles[i-1])
                for j := 1; j <= k; j++ {
                        dp[i][j] = dp[i-1][j]
                        sum := 0
                        for x := 1; x <= min(j, length); x++ { //
↪枚举第i个栈能取到的长度
                                sum = sum + piles[i-1][x-1]                //
↪第i个栈的前缀和
                                dp[i][j] = max(dp[i][j], dp[i-1][j-x]+sum) //
↪x个在第i个栈取+j-x在前i-1个栈取:j减去x + x个前缀和
                        }
                }
        }
        return dp[n][k]
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func maxValueOfCoins(piles [][]int, k int) int {
        n := len(piles)
        dp := make([]int, k+1) // 背包问题：dp[i]=>表示取i个硬币的最大值
        for i := 1; i <= n; i++ {
                length := len(piles[i-1])
                for j := k; j >= 1; j-- {
                        sum := 0
                        for x := 1; x <= min(j, length); x++ { //␣
↪枚举第i个栈能取到的长度
                                sum = sum + piles[i-1][x-1]     // 第i个栈的前缀和
                                dp[j] = max(dp[j], dp[j-x]+sum) // x个在第i个栈取+j-
↪x在前i-1个栈取:j减去x + x个前缀和
                        }
                }
        }
        return dp[k]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
```

(续下页)

(接上页)

```
        }
        return a
}
```

69.4 2223. 构造字符串的总得分和

69.4.1 题目

69.4.2 解题思路

69.5 2227. 加密解密字符串 (1)

- 题目

```
给你一个字符数组 keys ，由若干 互不相同 的字符组成。还有一个字符串数组 values␣
↪，内含若干长度为 2 的字符串。
另给你一个字符串数组 dictionary ，包含解密后所有允许的原字符串。
请你设计并实现一个支持加密及解密下标从 0 开始字符串的数据结构。
字符串 加密 按下述步骤进行：
对字符串中的每个字符 c ，先从 keys 中找出满足 keys[i] == c 的下标 i 。
在字符串中，用values[i] 替换字符 c 。
字符串 解密 按下述步骤进行：
将字符串每相邻 2 个字符划分为一个子字符串，对于每个子字符串 s ，找出满足 values[i] ==␣
↪s 的一个下标 i 。
如果存在多个有效的 i ，从中选择 任意␣
↪一个。这意味着一个字符串解密可能得到多个解密字符串。
在字符串中，用 keys[i] 替换 s 。
实现 Encrypter 类：
Encrypter(char[] keys, String[] values, String[] dictionary) 用 keys、
values 和 dictionary 初始化 Encrypter 类。
String encrypt(String word1) 按上述加密过程完成对 word1 的加密，并返回加密后的字符串。
int decrypt(String word2) 统计并返回可以由 word2 解密得到且出现在 dictionary␣
↪中的字符串数目。
示例：输入：["Encrypter", "encrypt", "decrypt"]
```

(续下页)

(接上页)

```
[[['a', 'b', 'c', 'd'], ["ei", "zf", "ei", "am"],
["abcd", "acbd", "adbc", "badc", "dacb", "cadb", "cbda", "abad"]], ["abcd"], [
↪"eizfeiam"]]
输出：[null, "eizfeiam", 2]
解释：
Encrypter encrypter = new Encrypter([['a', 'b', 'c', 'd'],
["ei", "zf", "ei", "am"], ["abcd", "acbd", "adbc", "badc", "dacb", "cadb", "cbda",
↪"abad"]);
encrypter.encrypt("abcd"); // 返回 "eizfeiam"。
                           // 'a' 映射为 "ei"，'b' 映射为 "zf"，'c' 映射为 "ei"，'d'␣
↪映射为 "am"。
encrypter.decrypt("eizfeiam"); // return 2.
                               // "ei" 可以映射为 'a' 或 'c'，"zf" 映射为 'b'，"am"␣
↪映射为 'd'。
                               // 因此，解密后可以得到的字符串是 "abad"，"cbad"，"abcd
↪" 和 "cbcd"。
                               // 其中 2 个字符串，"abad" 和 "abcd"，在 dictionary␣
↪中出现，所以答案是 2 。
提示：1 <= keys.length == values.length <= 26
values[i].length == 2
1 <= dictionary.length <= 100
1 <= dictionary[i].length <= 100
所有 keys[i] 和 dictionary[i] 互不相同
1 <= word1.length <= 2000
1 <= word2.length <= 200
所有 word1[i] 都出现在 keys 中
word2.length 是偶数
keys、values[i]、dictionary[i]、word1 和 word2 只含小写英文字母
至多调用 encrypt 和 decrypt 总计 200 次
```

- 解题思路

```
type Encrypter struct {
        arr [26]string
        m   map[string]int
}

func Constructor(keys []byte, values []string, dictionary []string) Encrypter {
        arr := [26]string{}
        for i := 0; i < len(keys); i++ {
                arr[int(keys[i]-'a')] = values[i]
        }
        e := Encrypter{
                arr: arr,
```

(续下页)

(接上页)

```
                m:   map[string]int{},
        }
        // 加密所有值
        for i := 0; i < len(dictionary); i++ {
                e.m[e.Encrypt(dictionary[i])]++
        }
        return e
}

func (this *Encrypter) Encrypt(word1 string) string {
        res := make([]byte, 0)
        for i := 0; i < len(word1); i++ {
                v := this.arr[int(word1[i]-'a')]
                if v == "" {
                        return ""
                }
                res = append(res, v...)
        }
        return string(res)
}

func (this *Encrypter) Decrypt(word2 string) int {
        return this.m[word2]
}
```

69.6 2246. 相邻字符不同的最长路径 (2)

- 题目

```
给你一棵 树（即一个连通、无向、无环图），根节点是节点 0 ，这棵树由编号从 0 到 n - 1
↪的 n 个节点组成。
用下标从 0 开始、长度为 n 的数组 parent 来表示这棵树，
其中 parent[i] 是节点 i 的父节点，由于节点 0 是根节点，所以 parent[0] == -1 。
另给你一个字符串 s ，长度也是 n ，其中 s[i] 表示分配给节点 i 的字符。
请你找出路径上任意一对相邻节点都没有分配到相同字符的 最长路径 ，并返回该路径的长度。
示例 1：输入：parent = [-1,0,0,1,1,2], s = "abacbe" 输出：3
解释：任意一对相邻节点字符都不同的最长路径是：0 -> 1 -> 3 。该路径的长度是 3
↪，所以返回 3 。
可以证明不存在满足上述条件且比 3 更长的路径。
示例 2：输入：parent = [-1,0,0,0], s = "aabc" 输出：3
解释：任意一对相邻节点字符都不同的最长路径是：2 -> 0 -> 3 。该路径的长度为 3
↪，所以返回 3 。
```

(续下页)

(接上页)

```
提示：n == parent.length == s.length
1 <= n <= 105
对所有 i >= 1 ，0 <= parent[i] <= n - 1 均成立
parent[0] == -1
parent 表示一棵有效的树
s 仅由小写英文字母组成
```

- 解题思路

```
var arr [][]int
var res int

func longestPath(parent []int, s string) int {
        res = 0
        n := len(parent)
        arr = make([][]int, n)
        for i := 1; i < n; i++ {
                p := parent[i]
                arr[p] = append(arr[p], i)
        }
        dfs(0, s)
        return res + 1 // 加上自身
}

func dfs(x int, s string) int {
        result := 0 // 以该节点单边路径最大长度
        for i := 0; i < len(arr[x]); i++ {
                value := dfs(arr[x][i], s) + 1 // 子节点的最长路径
                if s[x] != s[arr[x][i]] {
                        res = max(res, result+value) //␣
↪更新：子树的最长路径+子树的最长路径
                        result = max(result, value)
                }
        }
        return result
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
# 2
var arr [][]int
var res int

func longestPath(parent []int, s string) int {
        res = 0
        n := len(parent)
        arr = make([][]int, n)
        for i := 1; i < n; i++ {
                p := parent[i]
                arr[p] = append(arr[p], i)
        }
        dfs(0, s)
        return res
}

func dfs(x int, s string) int {
        var a, b int // 2个节点值 a>b
        for i := 0; i < len(arr[x]); i++ {
                v := dfs(arr[x][i], s)
                if s[x] != s[arr[x][i]] {
                        if v > a {
                                a, b = v, a
                        } else if v > b {
                                b = v
                        }
                }
        }
        res = max(res, 1+a+b)
        return a + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

69.7 2251. 花期内花的数目

69.7.1 题目

```
给你一个下标从 0开始的二维整数数组flowers, 其中flowers[i] = [starti, endi]
表示第i朵花的 花期从starti到endi (都 包含) 。
同时给你一个下标从 0开始大小为 n的整数数组persons, persons[i]是第i个人来看花的时间。
请你返回一个大小为 n的整数数组answer, 其中answer[i]是第i个人到达时在花期内花的数目。
示例 1: 输入: flowers = [[1,6],[3,7],[9,12],[4,13]], persons = [2,3,7,11] 输出: [1,2,
↪2,2]
解释: 上图展示了每朵花的花期时间, 和每个人的到达时间。
对每个人, 我们返回他们到达时在花期内花的数目。
示例 2: 输入: flowers = [[1,10],[3,3]], persons = [3,3,2] 输出: [2,2,1]
解释: 上图展示了每朵花的花期时间, 和每个人的到达时间。
对每个人, 我们返回他们到达时在花期内花的数目。
提示: 1 <= flowers.length <= 5 * 104
flowers[i].length == 2
1 <= starti <= endi <= 109
1 <= persons.length <= 5 * 104
1 <= persons[i] <= 109
```

69.7.2 解题思路

```

```

69.8 2262. 字符串的总引力 (2)

- 题目

```
字符串的 引力 定义为: 字符串中 不同 字符的数量。
例如, "abbca" 的引力为 3 , 因为其中有 3 个不同字符 'a'、'b' 和 'c' 。
给你一个字符串 s , 返回 其所有子字符串的总引力 。
子字符串 定义为: 字符串中的一个连续字符序列。
示例 1: 输入: s = "abbca" 输出: 28
解释: "abbca" 的子字符串有:
- 长度为 1 的子字符串: "a"、"b"、"b"、"c"、"a" 的引力分别为 1、1、1、1、1, 总和为 5 。
- 长度为 2 的子字符串: "ab"、"bb"、"bc"、"ca" 的引力分别为 2、1、2、2 , 总和为 7 。
- 长度为 3 的子字符串: "abb"、"bbc"、"bca" 的引力分别为 2、2、3 , 总和为 7 。
- 长度为 4 的子字符串: "abbc"、"bbca" 的引力分别为 3、3 , 总和为 6 。
- 长度为 5 的子字符串: "abbca" 的引力为 3 , 总和为 3 。
```

(续下页)

(接上页)

```
引力总和为 5 + 7 + 7 + 6 + 3 = 28 。
示例 2: 输入: s = "code" 输出: 20
解释: "code" 的子字符串有:
- 长度为 1 的子字符串: "c"、"o"、"d"、"e" 的引力分别为 1、1、1、1 ，总和为 4 。
- 长度为 2 的子字符串: "co"、"od"、"de" 的引力分别为 2、2、2 ，总和为 6 。
- 长度为 3 的子字符串: "cod"、"ode" 的引力分别为 3、3 ，总和为 6 。
- 长度为 4 的子字符串: "code" 的引力为 4 ，总和为 4 。
引力总和为 4 + 6 + 6 + 4 = 20 。
提示: 1 <= s.length <= 105
s 由小写英文字母组成
```

- 解题思路

```
func appealSum(s string) int64 {
        res := int64(0)
        n := len(s)
        m := make(map[int]int)
        for i := 0; i < 26; i++ {
                m[i] = -1 // 上一次出现的位置
        }
        sum := 0
        for i := 0; i < n; i++ {
                v := int(s[i] - 'a')
                // 1、新字符没出现: 在sum基础上加上(i+1)
                // 2、新字符出现过: 在sum基础上加上i-
↪m[v](到上一次出现的距离, 只加后一段)
                sum = sum + (i - m[v])
                res = res + int64(sum)
                m[v] = i
        }
        return res
}

# 2
func appealSum(s string) int64 {
        res := int64(0)
        n := len(s)
        dp := make([]int, n+1) // 以长度为i结尾的子字符串总引力
        m := make(map[int]int)
        for i := 0; i < 26; i++ {
                m[i] = -1 // 上一次出现的位置
        }
        for i := 0; i < n; i++ {
                v := int(s[i] - 'a')
```

(续下页)

(接上页)

```
                // 1、新字符没出现：在dp[i]基础上加上(i+1)
                // 2、新字符出现过：在dp[i]基础上加上i-
↪m[v](到上一次出现的距离，只加后一段)
                dp[i+1] = dp[i] + (i - m[v])
                m[v] = i
        }
        for i := 1; i <= n; i++ {
                res = res + int64(dp[i])
        }
        return res
}
```

69.9 2276. 统计区间中的整数数目

69.9.1 题目

69.9.2 解题思路

```

```

69.10 2290. 到达角落需要移除障碍物的最小数目

69.10.1 题目

```
给你一个下标从 0 开始的二维整数数组 grid ，数组大小为 m x n␣
↪。每个单元格都是两个值之一：
0 表示一个 空 单元格，
1 表示一个可以移除的 障碍物 。
你可以向上、下、左、右移动，从一个空单元格移动到另一个空单元格。
现在你需要从左上角(0, 0) 移动到右下角 (m - 1, n - 1) ，返回需要移除的障碍物的 最小␣
↪数目。
示例 1：输入：grid = [[0,1,1],[1,1,0],[1,1,0]] 输出：2
解释：可以移除位于 (0, 1) 和 (0, 2) 的障碍物来创建从 (0, 0) 到 (2, 2) 的路径。
可以证明我们至少需要移除两个障碍物，所以返回 2 。
注意，可能存在其他方式来移除 2 个障碍物，创建出可行的路径。
示例 2：输入：grid = [[0,1,0,0,0],[0,1,0,1,0],[0,0,0,1,0]] 输出：0
解释：不移除任何障碍物就能从 (0, 0) 到 (2, 4) ，所以返回 0 。
提示：m == grid.length
```

(续下页)

(接上页)

```
n == grid[i].length
1 <= m, n <= 105
2 <= m * n <= 105
grid[i][j] 为 0 或 1
grid[0][0] == grid[m - 1][n - 1] == 0
```

69.10.2 解题思路

CHAPTER 70

2301-2400-Easy

70.1 2303. 计算应缴税款总额 (2)

- 题目

```
给你一个下标从 0 开始的二维整数数组 brackets ，其中 brackets[i] = [upperi, percenti]␣
↪，
表示第 i 个税级的上限是 upperi ，征收的税率为 percenti 。
税级按上限 从低到高排序（在满足 0 < i < brackets.length 的前提下，upperi-1 <␣
↪upperi）。
税款计算方式如下：
不超过 upper0 的收入按税率 percent0 缴纳
接着 upper1 - upper0 的部分按税率 percent1 缴纳
然后 upper2 - upper1 的部分按税率 percent2 缴纳
以此类推
给你一个整数 income 表示你的总收入。返回你需要缴纳的税款总额。与标准答案误差不超 10-5␣
↪的结果将被视作正确答案。
示例 1：输入：brackets = [[3,50],[7,10],[12,25]], income = 10 输出：2.65000
解释：前 $3 的税率为 50% 。需要支付税款 $3 * 50% = $1.50 。
接下来 $7 - $3 = $4 的税率为 10% 。需要支付税款 $4 * 10% = $0.40 。
最后 $10 - $7 = $3 的税率为 25% 。需要支付税款 $3 * 25% = $0.75 。
需要支付的税款总计 $1.50 + $0.40 + $0.75 = $2.65 。
示例 2：输入：brackets = [[1,0],[4,25],[5,50]], income = 2 输出：0.25000
解释：前 $1 的税率为 0% 。需要支付税款 $1 * 0% = $0 。
剩下 $1 的税率为 25% 。需要支付税款 $1 * 25% = $0.25 。
```

(续下页)

(接上页)

```
需要支付的税款总计 $0 + $0.25 = $0.25 。
示例 3：输入：brackets = [[2,50]], income = 0 输出：0.00000
解释：没有收入，无需纳税，需要支付的税款总计 $0 。
提示：1 <= brackets.length <= 100
1 <= upperi <= 1000
0 <= percenti <= 100
0 <= income <= 1000
upperi 按递增顺序排列
upperi 中的所有值 互不相同
最后一个税级的上限大于等于 income
```

- 解题思路

```
func calculateTax(brackets [][]int, income int) float64 {
        res := 0
        prev := 0
        for i := 0; i < len(brackets); i++ {
                a, b := brackets[i][0], brackets[i][1]
                if income <= a {
                        res = res + (income-prev)*b
                        break
                }
                res = res + (a-prev)*b
                prev = a
        }
        return float64(res) / 100
}

# 2
func calculateTax(brackets [][]int, income int) float64 {
        res := 0
        prev := 0
        for i := 0; i < len(brackets); i++ {
                a, b := brackets[i][0], brackets[i][1]
                if income < prev {
                        break
                }
                res = res + (min(income, a)-prev)*b
                prev = a
        }
        return float64(res) / 100
}

func min(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return b
        }
        return a
}
```

70.2 2309. 兼具大小写的最好英文字母 (1)

- 题目

```
给你一个由英文字母组成的字符串 s ，请你找出并返回 s 中的 最好␣
↪英文字母。返回的字母必须为大写形式。
如果不存在满足条件的字母，则返回一个空字符串。
最好 英文字母的大写和小写形式必须 都 在 s 中出现。
英文字母 b 比另一个英文字母a更好 的前提是：英文字母表中，b 在 a 之 后 出现。
示例 1：输入：s = "lEeTcOdE" 输出："E"
解释：字母 'E' 是唯一一个大写和小写形式都出现的字母。
示例 2：输入：s = "arRAzFif" 输出："R"
解释：字母 'R' 是大写和小写形式都出现的最好英文字母。
注意 'A' 和 'F' 的大写和小写形式也都出现了，但是 'R' 比 'F' 和 'A' 更好。
示例 3：输入：s = "AbCdEfGhIjK" 输出：""
解释：不存在大写和小写形式都出现的字母。
提示：1 <= s.length <= 1000
s 由小写和大写英文字母组成
```

- 解题思路

```
func greatestLetter(s string) string {
        m := make(map[byte]bool)
        for i := 0; i < len(s); i++ {
                m[s[i]] = true
        }
        for i := 25; i >= 0; i-- {
                x := byte('a' + i)
                y := byte('A' + i)
                if m[x] == true && m[y] == true {
                        return string(y)
                }
        }
        return ""
}
```

70.3 2315. 统计星号 (2)

- 题目

```
给你一个字符串s，每两个连续竖线'|'为 一对。换言之，第一个和第二个'|
↪'为一对，第三个和第四个'|'为一对，以此类推。
请你返回 不在 竖线对之间，s中'*'的数目。
注意，每个竖线'|'都会 恰好属于一个对。
示例 1：输入：s = "l|*e*et|c**o|*de|" 输出：2
解释：不在竖线对之间的字符加粗加斜体后，得到字符串："l|*e*et|c**o|*de|" 。
第一和第二条竖线 '|' 之间的字符不计入答案。
同时，第三条和第四条竖线 '|' 之间的字符也不计入答案。
不在竖线对之间总共有 2 个星号，所以我们返回 2 。
示例 2：输入：s = "iamprogrammer" 输出：0
解释：在这个例子中，s 中没有星号。所以返回 0 。
示例 3：输入：s = "yo|uar|e**|b|e***au|tifu|l" 输出：5
解释：需要考虑的字符加粗加斜体后："yo|uar|e**|b|e***au|tifu|l" 。不在竖线对之间总共有␣
↪5 个星号。所以我们返回 5 。
提示：1 <= s.length <= 1000
s只包含小写英文字母，竖线'|'和星号'*'。
s包含 偶数个竖线'|' 。
```

- 解题思路

```
func countAsterisks(s string) int {
        res := 0
        arr := strings.Split(s, "|")
        for i := 0; i < len(arr); i = i + 2 {
                res = res + strings.Count(arr[i], "*")
        }
        return res
}

# 2

func countAsterisks(s string) int {
        res := 0
        count := 0
        for i := 0; i < len(s); i++ {
                if s[i] == '|' {
                        count++
                }
                if s[i] == '*' && count%2 == 0 {
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}
```

70.4 2319. 判断矩阵是否是一个 X 矩阵 (1)

- 题目

```
如果一个正方形矩阵满足下述 全部 条件，则称之为一个 X 矩阵 ：
矩阵对角线上的所有元素都 不是 0
矩阵中所有其他元素都是 0
给你一个大小为 n x n 的二维整数数组 grid ，表示一个正方形矩阵。如果 grid 是一个 X␣
↪矩阵 ，返回 true ；否则，返回 false 。
示例 1：输入：grid = [[2,0,0,1],[0,3,1,0],[0,5,2,0],[4,0,0,2]] 输出：true
解释：矩阵如上图所示。
X 矩阵应该满足：绿色元素（对角线上）都不是 0 ，红色元素都是 0 。
因此，grid 是一个 X 矩阵。
示例 2：输入：grid = [[5,7,0],[0,3,1],[0,5,0]] 输出：false
解释：矩阵如上图所示。
X 矩阵应该满足：绿色元素（对角线上）都不是 0 ，红色元素都是 0 。
因此，grid 不是一个 X 矩阵。
提示：n == grid.length == grid[i].length
3 <= n <= 100
0 <= grid[i][j] <= 105
```

- 解题思路

```
func checkXMatrix(grid [][]int) bool {
        n := len(grid)
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if i == j || (i+j == n-1) {
                                if grid[i][j] == 0 {
                                        return false
                                }
                        } else {
                                if grid[i][j] != 0 {
                                        return false
                                }
                        }
                }
```

(续下页)

(接上页)

```
        }
        return true
}
```

70.5 2325. 解密消息 (1)

- 题目

```
给你字符串 key 和 message ，分别表示一个加密密钥和一段加密消息。解密 message␣
↪的步骤如下：
使用 key 中 26 个英文小写字母第一次出现的顺序作为替换表中的字母 顺序 。
将替换表与普通英文字母表对齐，形成对照表。
按照对照表 替换 message 中的每个字母。
空格 ' ' 保持不变。
例如，key = "happy boy"（实际的加密密钥会包含字母表中每个字母 至少一次），
据此，可以得到部分对照表（'h' -> 'a'、'a' -> 'b'、'p' -> 'c'、'y' -> 'd'、'b' -> 'e'、
↪'o' -> 'f'）。
返回解密后的消息。
示例 1：输入：key = "the quick brown fox jumps over the lazy dog", message = "vkbs bs␣
↪t suepuv"
输出："this is a secret"
解释：对照表如上图所示。
提取 "the quick brown fox jumps over the lazy dog"␣
↪中每个字母的首次出现可以得到替换表。
示例 2：输入：key = "eljuxhpwnyrdgtqkviszcfmabo", message = "zwx hnfx lqantp mnoeius␣
↪ycgk vcnjrdb"
输出："the five boxing wizards jump quickly"
解释：对照表如上图所示。
提取 "eljuxhpwnyrdgtqkviszcfmabo" 中每个字母的首次出现可以得到替换表。
提示：26 <= key.length <= 2000
key 由小写英文字母及 ' ' 组成
key 包含英文字母表中每个字符（'a' 到 'z'）至少一次
1 <= message.length <= 2000
message 由小写英文字母和 ' ' 组成
```

- 解题思路

```
func decodeMessage(key string, message string) string {
    m := make(map[byte]byte)
    cur := byte('a')
    for i := 0; i < len(key); i++ {
        if key[i] != ' ' && m[key[i]] == 0 {
```

(续下页)

(接上页)

```
            m[key[i]] = cur
            cur++
        }
    }
    arr := []byte(message)
    for i := 0; i < len(message); i++ {
        if message[i] != ' ' {
            arr[i] = m[message[i]]
        }
    }
    return string(arr)
}
```

70.6 2331. 计算布尔二叉树的值 (1)

- 题目

```
给你一棵 完整二叉树的根，这棵树有以下特征：
叶子节点要么值为0要么值为1，其中0 表示False，1 表示True。
非叶子节点 要么值为 2要么值为 3，其中2表示逻辑或OR ，3表示逻辑与AND。
计算一个节点的值方式如下：
如果节点是个叶子节点，那么节点的 值为它本身，即True或者False。
否则，计算两个孩子的节点值，然后将该节点的运算符对两个孩子值进行 运算。
返回根节点root的布尔运算值。
完整二叉树是每个节点有 0个或者 2个孩子的二叉树。
叶子节点是没有孩子的节点。
示例 1：输入：root = [2,1,3,null,null,0,1] 输出：true
解释：上图展示了计算过程。
AND 与运算节点的值为 False AND True = False 。
OR 运算节点的值为 True OR False = True 。
根节点的值为 True ，所以我们返回 true 。
示例 2：输入：root = [0] 输出：false
解释：根节点是叶子节点，且值为 false，所以我们返回 false 。
提示：树中节点数目在[1, 1000]之间。
0 <= Node.val <= 3
每个节点的孩子数为0 或2。
叶子节点的值为0或1。
非叶子节点的值为2或3 。
```

- 解题思路

```
func evaluateTree(root *TreeNode) bool {
        if root.Left == nil || root.Right == nil { // 叶子节点
                return root.Val == 1 //
        }
        if root.Val == 2 {
                return evaluateTree(root.Left) || evaluateTree(root.Right)
        }
        if root.Val == 3 {
                return evaluateTree(root.Left) && evaluateTree(root.Right)
        }
        return true
}
```

70.7 2335. 装满杯子需要的最短总时长 (2)

- 题目

```
现有一台饮水机，可以制备冷水、温水和热水。每秒钟，可以装满 2 杯 不同 类型的水或者 1␣
↪杯任意类型的水。
给你一个下标从 0 开始、长度为 3 的整数数组 amount ，
其中 amount[0]、amount[1] 和 amount[2] 分别表示需要装满冷水、温水和热水的杯子数量。
返回装满所有杯子所需的 最少 秒数。
示例 1：输入：amount = [1,4,2]输出：4
解释：下面给出一种方案：
第 1 秒：装满一杯冷水和一杯温水。
第 2 秒：装满一杯温水和一杯热水。
第 3 秒：装满一杯温水和一杯热水。
第 4 秒：装满一杯温水。
可以证明最少需要 4 秒才能装满所有杯子。
示例 2：输入：amount = [5,4,4] 输出：7
解释：下面给出一种方案：
第 1 秒：装满一杯冷水和一杯热水。
第 2 秒：装满一杯冷水和一杯温水。
第 3 秒：装满一杯冷水和一杯温水。
第 4 秒：装满一杯温水和一杯热水。
第 5 秒：装满一杯冷水和一杯热水。
第 6 秒：装满一杯冷水和一杯温水。
第 7 秒：装满一杯热水。
示例 3：输入：amount = [5,0,0] 输出：5
解释：每秒装满一杯冷水。
提示：amount.length == 3
0 <= amount[i] <= 100
```

- 解题思路

```
func fillCups(amount []int) int {
        res := 0
        for {
                sort.Ints(amount)
                if amount[1] == 0 {
                        break
                }
                res++
                amount[1]--
                amount[2]--
        }
        return res + amount[2]
}

# 2
func fillCups(amount []int) int {
        sort.Ints(amount)
        if amount[0]+amount[1] <= amount[2] {
                return amount[2]
        }
        // a+b>c
        // 超出部分：a+b-c
        return (amount[0]+amount[1]-amount[2]+1)/2 + amount[2]
}
```

70.8 2341. 数组能形成多少数对 (2)

- 题目

```
给你一个下标从 0 开始的整数数组 nums 。在一步操作中，你可以执行以下步骤：
从 nums 选出 两个 相等的 整数
从 nums 中移除这两个整数，形成一个 数对
请你在 nums 上多次执行此操作直到无法继续执行。
返回一个下标从 0 开始、长度为 2 的整数数组 answer 作为答案，
其中 answer[0] 是形成的数对数目，answer[1] 是对 nums␣
↪尽可能执行上述操作后剩下的整数数目。
示例 1：输入：nums = [1,3,2,1,3,2,2] 输出：[3,1]
解释：nums[0] 和 nums[3] 形成一个数对，并从 nums 中移除，nums = [3,2,3,2,2] 。
nums[0] 和 nums[2] 形成一个数对，并从 nums 中移除，nums = [2,2,2] 。
nums[0] 和 nums[1] 形成一个数对，并从 nums 中移除，nums = [2] 。
无法形成更多数对。总共形成 3 个数对，nums 中剩下 1 个数字。
```

(续下页)

(接上页)

```
示例 2：输入：nums = [1,1] 输出：[1,0]
解释：nums[0] 和 nums[1] 形成一个数对，并从 nums 中移除，nums = [] 。
无法形成更多数对。总共形成 1 个数对，nums 中剩下 0 个数字。
示例 3：输入：nums = [0] 输出：[0,1]
解释：无法形成数对，nums 中剩下 1 个数字。
提示：1 <= nums.length <= 100
0 <= nums[i] <= 100
```

- 解题思路

```
func numberOfPairs(nums []int) []int {
        a, b := 0, 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for _, v := range m {
                a = a + v/2
                b = b + v%2
        }
        return []int{a, b}
}

# 2
func numberOfPairs(nums []int) []int {
        a := 0
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
                if m[nums[i]]%2 == 0 {
                        a++
                }
        }
        return []int{a, len(nums) - 2*a}
}
```

70.9 2347. 最好的扑克手牌 (1)

- 题目

```
给你一个整数数组 ranks 和一个字符数组 suit 。你有 5 张扑克牌，第 i 张牌大小为 ranks[i] ，花色为 suits
下述是从好到坏你可能持有的 手牌类型 ：
"Flush"：同花，五张相同花色的扑克牌。
"Three of a Kind"：三条，有 3 张大小相同的扑克牌。
"Pair"：对子，两张大小一样的扑克牌。
"High Card"：高牌，五张大小互不相同的扑克牌。
请你返回一个字符串，表示给定的 5 张牌中，你能组成的 最好手牌类型 。
注意：返回的字符串 大小写 需与题目描述相同。
示例 1：输入：ranks = [13,2,3,1,9], suits = ["a","a","a","a","a"] 输出："Flush"
解释：5 张扑克牌的花色相同，所以返回 "Flush" 。
示例 2：输入：ranks = [4,4,2,4,4], suits = ["d","a","a","b","c"] 输出："Three of a␣
↪Kind"
解释：第一、二和四张牌组成三张相同大小的扑克牌，所以得到 "Three of a Kind" 。
注意我们也可以得到 "Pair" ，但是 "Three of a Kind" 是更好的手牌类型。
有其他的 3 张牌也可以组成 "Three of a Kind" 手牌类型。
示例 3：输入：ranks = [10,10,2,12,9], suits = ["a","b","c","a","d"] 输出："Pair"
解释：第一和第二张牌大小相同，所以得到 "Pair" 。
我们无法得到 "Flush" 或者 "Three of a Kind" 。
提示：ranks.length == suits.length == 5
1 <= ranks[i] <= 13
'a' <= suits[i] <= 'd'
任意两张扑克牌不会同时有相同的大小和花色。
```

- 解题思路

```
func bestHand(ranks []int, suits []byte) string {
        if strings.Count(string(suits), string(suits[0])) == 5 {
                return "Flush"
        }
        m := make(map[int]int)
        for _, v := range ranks {
                m[v]++
                if m[v] == 3 {
                        return "Three of a Kind"
                }
        }
        for _, v := range m {
                if v >= 2 {
                        return "Pair"
                }
```

(续下页)

(接上页)

```
        }
        return "High Card"
}
```

70.10 2351. 第一个出现两次的字母 (2)

- 题目

```
给你一个由小写英文字母组成的字符串 s ，请你找出并返回第一个出现 两次 的字母。
注意：如果 a 的 第二次 出现比 b 的 第二次 出现在字符串中的位置更靠前，则认为字母 a␣
↪在字母 b 之前出现两次。
s 包含至少一个出现两次的字母。
示例 1：输入：s = "abccbaacz" 输出："c"
解释：字母 'a' 在下标 0 、5 和 6 处出现。
字母 'b' 在下标 1 和 4 处出现。
字母 'c' 在下标 2 、3 和 7 处出现。
字母 'z' 在下标 8 处出现。
字母 'c' 是第一个出现两次的字母，因为在所有字母中，'c' 第二次出现的下标是最小的。
示例 2：输入：s = "abcdd" 输出："d"
解释：只有字母 'd' 出现两次，所以返回 'd' 。
提示：2 <= s.length <= 100
s 由小写英文字母组成
s 包含至少一个重复字母
```

- 解题思路

```
func repeatedCharacter(s string) byte {
        m := make(map[byte]bool)
        for _, v := range s {
                if m[byte(v)] == true {
                        return byte(v)
                }
                m[byte(v)] = true
        }
        return 0
}

# 2
func repeatedCharacter(s string) byte {
        mask := 0
        for _, v := range s {
                target := 1 << (v - 'a')
```

(续下页)

(接上页)

```
                if mask&target != 0 {
                        return byte(v)
                }
                mask = mask | target
        }
        return 0
}
```

70.11 2357. 使数组中所有元素都等于零 (1)

- 题目

```
给你一个非负整数数组 nums 。在一步操作中，你必须：
选出一个正整数 x ，x 需要小于或等于 nums 中 最小 的 非零 元素。
nums 中的每个正整数都减去 x。
返回使 nums 中所有元素都等于 0 需要的 最少 操作数。
示例 1：输入：nums = [1,5,0,3,5] 输出：3
解释：第一步操作：选出 x = 1 ，之后 nums = [0,4,0,2,4] 。
第二步操作：选出 x = 2 ，之后 nums = [0,2,0,0,2] 。
第三步操作：选出 x = 2 ，之后 nums = [0,0,0,0,0] 。
示例 2：输入：nums = [0] 输出：0
解释：nums 中的每个元素都已经是 0 ，所以不需要执行任何操作。
提示：1 <= nums.length <= 100
0 <= nums[i] <= 100
```

- 解题思路

```
func minimumOperations(nums []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                if nums[i] > 0 {
                        m[nums[i]] = true // 统计不为0的数字去重个数
                }
        }
        return len(m)
}
```

70.12 2363. 合并相似的物品 (1)

- 题目

```
给你两个二维整数数组items1 和items2，表示两个物品集合。每个数组items有以下特质：
items[i] = [valuei, weighti] 其中valuei表示第i件物品的价值，weighti表示第 i件物品的重量。
items中每件物品的价值都是 唯一的。
请你返回一个二维数组ret，其中ret[i] = [valuei, weighti]，weighti是所有价值为valuei物品的重量之和。
注意：ret应该按价值 升序排序后返回。
示例 1：输入：items1 = [[1,1],[4,5],[3,8]], items2 = [[3,1],[1,5]] 输出：[[1,6],[3,9],[4,5]]
解释：value = 1 的物品在 items1 中 weight = 1 ，在 items2 中 weight = 5 ，总重量为 1 + 5 = 6 。
value = 3 的物品再 items1 中 weight = 8 ，在 items2 中 weight = 1 ，总重量为 8 + 1 = 9 。
value = 4 的物品在 items1 中 weight = 5 ，总重量为 5 。
所以，我们返回 [[1,6],[3,9],[4,5]] 。
示例 2：输入：items1 = [[1,1],[3,2],[2,3]], items2 = [[2,1],[3,2],[1,3]] 输出：[[1,4],[2,4],[3,4]]
解释：value = 1 的物品在 items1 中 weight = 1 ，在 items2 中 weight = 3 ，总重量为 1 + 3 = 4 。
value = 2 的物品在 items1 中 weight = 3 ，在 items2 中 weight = 1 ，总重量为 3 + 1 = 4 。
value = 3 的物品在 items1 中 weight = 2 ，在 items2 中 weight = 2 ，总重量为 2 + 2 = 4 。
所以，我们返回 [[1,4],[2,4],[3,4]] 。
示例 3：输入：items1 = [[1,3],[2,2]], items2 = [[7,1],[2,2],[1,4]] 输出：[[1,7],[2,4],[7,1]]
解释：value = 1 的物品在 items1 中 weight = 3 ，在 items2 中 weight = 4 ，总重量为 3 + 4 = 7 。
value = 2 的物品在 items1 中 weight = 2 ，在 items2 中 weight = 2 ，总重量为 2 + 2 = 4 。
value = 7 的物品在 items2 中 weight = 1 ，总重量为 1 。
所以，我们返回 [[1,7],[2,4],[7,1]] 。
提示：1 <= items1.length, items2.length <= 1000
items1[i].length == items2[i].length == 2
1 <= valuei, weighti <= 1000
items1中每个 valuei都是 唯一的。
items2中每个 valuei都是 唯一的。
```

- 解题思路

```
func mergeSimilarItems(items1 [][]int, items2 [][]int) [][]int {
        m := make(map[int]int)
        for i := 0; i < len(items1); i++ {
                a, b := items1[i][0], items1[i][1]
                m[a] += b
        }
        for i := 0; i < len(items2); i++ {
                a, b := items2[i][0], items2[i][1]
                m[a] += b
        }
        res := make([][]int, 0)
        for k, v := range m {
                res = append(res, []int{k, v})
        }
        sort.Slice(res, func(i, j int) bool {
                return res[i][0] < res[j][0]
        })
        return res
}
```

70.13 2367. 算术三元组的数目 (1)

- 题目

```
给你一个下标从 0 开始、严格递增 的整数数组 nums 和一个正整数 diff␣
↪。如果满足下述全部条件，则三元组 (i, j, k) 就是一个 算术三元组 ：
i < j < k ，
nums[j] - nums[i] == diff 且
nums[k] - nums[j] == diff
返回不同 算术三元组 的数目。
示例 1：输入：nums = [0,1,4,6,7,10], diff = 3 输出：2
解释：(1, 2, 4) 是算术三元组：7 - 4 == 3 且 4 - 1 == 3 。
(2, 4, 5) 是算术三元组：10 - 7 == 3 且 7 - 4 == 3 。
示例 2：输入：nums = [4,5,6,7,8,9], diff = 2 输出：2
解释：(0, 2, 4) 是算术三元组：8 - 6 == 2 且 6 - 4 == 2 。
(1, 3, 5) 是算术三元组：9 - 7 == 2 且 7 - 5 == 2 。
提示：3 <= nums.length <= 200
0 <= nums[i] <= 200
1 <= diff <= 50
nums 严格 递增
```

- 解题思路

```
func arithmeticTriplets(nums []int, diff int) int {
        res := 0
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = true
        }
        for i := 0; i < len(nums); i++ {
                if m[nums[i]-diff] && m[nums[i]+diff] {
                        res++
                }
        }
        return res
}
```

70.14 2373. 矩阵中的局部最大值 (1)

- 题目

```
给你一个大小为 n x n 的整数矩阵 grid 。
生成一个大小为(n - 2) x (n - 2) 的整数矩阵 maxLocal ，并满足：
maxLocal[i][j] 等于 grid 中以 i + 1 行和 j + 1 列为中心的 3 x 3 矩阵中的 最大值 。
换句话说，我们希望找出 grid 中每个3 x 3 矩阵中的最大值。
返回生成的矩阵。
示例 1: 输入: grid = [[9,9,8,1],[5,6,2,6],[8,2,6,4],[6,2,2,2]] 输出: [[9,9],[8,6]]
解释: 原矩阵和生成的矩阵如上图所示。
注意，生成的矩阵中，每个值都对应 grid 中一个相接的 3 x 3 矩阵的最大值。
示例 2: 输入: grid = [[1,1,1,1,1],[1,1,1,1,1],[1,1,2,1,1],[1,1,1,1,1],[1,1,1,1,1]]
输出: [[2,2,2],[2,2,2],[2,2,2]]
解释: 注意，2 包含在 grid 中每个 3 x 3 的矩阵中。
提示: n == grid.length == grid[i].length
3 <= n <= 100
1 <= grid[i][j] <= 100
```

- 解题思路

```
func largestLocal(grid [][]int) [][]int {
        n := len(grid)
        res := make([][]int, n-2)
        for i := 0; i < n-2; i++ {
                res[i] = make([]int, n-2)
        }
        for i := 1; i < n-1; i++ {
                for j := 1; j < n-1; j++ {
```

(续下页)

(接上页)

```
                        maxValue := grid[i][j]
                        for a := i - 1; a <= i+1; a++ {
                                for b := j - 1; b <= j+1; b++ {
                                        maxValue = max(maxValue, grid[a][b])
                                }
                        }
                        res[i-1][j-1] = maxValue
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

70.15 2379. 得到 K 个黑块的最少涂色次数

70.15.1 题目

```
给你一个长度为 n 下标从 0 开始的字符串 blocks ，blocks[i] 要么是 'W' 要么是 'B' ，
表示第 i 块的颜色。字符 'W' 和 'B' 分别表示白色和黑色。
给你一个整数 k ，表示想要 连续 黑色块的数目。
每一次操作中，你可以选择一个白色块将它 涂成 黑色块。
请你返回至少出现 一次 连续 k 个黑色块的 最少 操作次数。
示例 1：输入：blocks = "WBBWWBBWBW", k = 7 输出：3
解释：一种得到 7 个连续黑色块的方法是把第 0 ，3 和 4 个块涂成黑色。
得到 blocks = "BBBBBBBWBW" 。
可以证明无法用少于 3 次操作得到 7 个连续的黑块。
所以我们返回 3 。
示例 2：输入：blocks = "WBWBBBW", k = 2 输出：0
解释：不需要任何操作，因为已经有 2 个连续的黑块。
所以我们返回 0 。
提示：n == blocks.length
1 <= n <= 100
blocks[i] 要么是 'W' ，要么是 'B' 。
1 <= k <= n
```

70.15.2 解题思路

CHAPTER 71

2301-2400-Medium

71.1 2304. 网格中的最小路径代价 (2)

- 题目

```
给你一个下标从 0 开始的整数矩阵grid ，矩阵大小为 m x n ，由从 0 到 m * n - 1␣
↪的不同整数组成。
你可以在此矩阵中，从一个单元格移动到 下一行 的任何其他单元格。如果你位于单元格 (x, y)␣
↪，
且满足 x < m - 1 ，你可以移动到 (x + 1, 0), (x + 1, 1), ..., (x + 1, n - 1)␣
↪中的任何一个单元格。
注意：在最后一行中的单元格不能触发移动。
每次可能的移动都需要付出对应的代价，代价用一个下标从 0 开始的二维数组 moveCost␣
↪表示，该数组大小为 (m * n) x n ，
其中 moveCost[i][j] 是从值为 i 的单元格移动到下一行第 j 列单元格的代价。从grid␣
↪最后一行的单元格移动的代价可以忽略。
grid 一条路径的代价是：所有路径经过的单元格的 值之和 加上 所有移动的 代价之和 。
从 第一行 任意单元格出发，返回到达 最后一行 任意单元格的最小路径代价。
示例 1：输入：grid = [[5,3],[4,0],[2,1]], moveCost = [[9,8],[1,5],[10,12],[18,6],[2,
↪4],[14,3]] 输出：17
解释：最小代价的路径是 5 -> 0 -> 1 。
- 路径途经单元格值之和 5 + 0 + 1 = 6 。
- 从 5 移动到 0 的代价为 3 。
- 从 0 移动到 1 的代价为 8 。
路径总代价为 6 + 3 + 8 = 17 。
```

(续下页)

(接上页)

```
示例 2：输入：grid = [[5,1,2],[4,0,3]],
moveCost = [[12,10,15],[20,23,8],[21,7,1],[8,1,13],[9,10,25],[5,3,2]] 输出：6
解释：最小代价的路径是 2 -> 3 。
- 路径途经单元格值之和 2 + 3 = 5 。
- 从 2 移动到 3 的代价为 1 。
路径总代价为 5 + 1 = 6 。
提示：m == grid.length
n == grid[i].length
2 <= m, n <= 50
grid 由从 0 到 m * n - 1 的不同整数组成
moveCost.length == m * n
moveCost[i].length == n
1 <= moveCost[i][j] <= 100
```

- 解题思路

```
func minPathCost(grid [][]int, moveCost [][]int) int {
        n, m := len(grid), len(grid[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        for j := 0; j < m; j++ {
                dp[0][j] = grid[0][j] // 第一行的代价
        }
        for i := 1; i < n; i++ {
                for j := 0; j < m; j++ {
                        value := math.MaxInt32
                        for k := 0; k < m; k++ {
                                // 上一行的代价+当前的单元值+移动的代价
                                prev := grid[i-1][k] // 上一层的值
                                value = min(value, dp[i-
↪1][k]+grid[i][j]+moveCost[prev][j])
                        }
                        dp[i][j] = value
                }
        }
        res := math.MaxInt32
        for j := 0; j < m; j++ {
                res = min(res, dp[n-1][j])
        }
        return res
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minPathCost(grid [][]int, moveCost [][]int) int {
        n, m := len(grid), len(grid[0])
        dp := make([]int, m)
        copy(dp, grid[0])
        for i := 1; i < n; i++ {
                temp := make([]int, m)
                for j := 0; j < m; j++ {
                        value := math.MaxInt32
                        for k := 0; k < m; k++ {
                                // 上一行的代价+当前的单元值+移动的代价
                                prev := grid[i-1][k] // 上一层的值
                                value = min(value, dp[k]+grid[i][j]+moveCost[prev][j])
                        }
                        temp[j] = value
                }
                copy(dp, temp)
        }
        res := math.MaxInt32
        for j := 0; j < m; j++ {
                res = min(res, dp[j])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

71.2 2305. 公平分发饼干 (3)

- 题目

```
给你一个整数数组 cookies ，其中 cookies[i] 表示在第 i 个零食包中的饼干数量。
另给你一个整数 k 表示等待分发零食包的孩子数量，所有 零食包都需要分发。
在同一个零食包中的所有饼干都必须分发给同一个孩子，不能分开。
分发的 不公平程度 定义为单个孩子在分发过程中能够获得饼干的最大总数。
返回所有分发的最小不公平程度。
示例 1：输入：cookies = [8,15,10,20,8], k = 2 输出：31
解释：一种最优方案是 [8,15,8] 和 [10,20] 。
- 第 1 个孩子分到 [8,15,8] ，总计 8 + 15 + 8 = 31 块饼干。
- 第 2 个孩子分到 [10,20] ，总计 10 + 20 = 30 块饼干。
分发的不公平程度为 max(31,30) = 31 。
可以证明不存在不公平程度小于 31 的分发方案。
示例 2：输入：cookies = [6,1,3,2,2,4,1,2], k = 3 输出：7
解释：一种最优方案是 [6,1]、[3,2,2] 和 [4,1,2] 。
- 第 1 个孩子分到 [6,1] ，总计 6 + 1 = 7 块饼干。
- 第 2 个孩子分到 [3,2,2] ，总计 3 + 2 + 2 = 7 块饼干。
- 第 3 个孩子分到 [4,1,2] ，总计 4 + 1 + 2 = 7 块饼干。
分发的不公平程度为 max(7,7,7) = 7 。
可以证明不存在不公平程度小于 7 的分发方案。
提示：2 <= cookies.length <= 8
1 <= cookies[i] <= 105
2 <= k <= cookies.length
```

解题思路

```
# 题目同leetcode 1723.完成所有工作的最短时间
func distributeCookies(cookies []int, k int) int {
        n := len(cookies)
        total := 1 << n
        sum := make([]int, total)
        for i := 0; i < n; i++ { // 预处理：饼干分配的状态和，分配给某一个人的和
                count := 1 << i
                for j := 0; j < count; j++ {
                        sum[count|j] = sum[j] + cookies[i] // 按位或运算：j前面补1=>
↪子集和加上tasks[i]
                }
        }
        dp := make([][]int, k) // f[i][j]=>
↪给前i个人分配工作，饼干的分配情况为j时，最小不公平程度
        for i := 0; i < k; i++ {
                dp[i] = make([]int, total)
```

(续下页)

(接上页)

```
        }
        for i := 0; i < total; i++ { // 第0个人的时候
                dp[0][i] = sum[i]
        }
        for i := 1; i < k; i++ {
                for j := 0; j < total; j++ {
                        minValue := math.MaxInt32            // dp[i][j]未赋值，为0
                        for a := j; a > 0; a = (a - 1) & j { //␣
↪遍历得到比较小的子集：数字j二进制为1位置上的非0子集
                                // 取子集的补集：j-a 或者 使用异或j^a
                                // minValue = min(minValue, max(dp[i-1][j-a], sum[a]))
                                minValue = min(minValue, max(dp[i-1][j^a], sum[a]))
                        }
                        dp[i][j] = minValue
                }
        }
        return dp[k-1][total-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
var res int

func distributeCookies(cookies []int, k int) int {
        res = math.MaxInt32
        dfs(cookies, make([]int, k), 0, 0, 0)
        return res
}

// index => cookies的下标；count => 已经分配的个数
```

(续下页)

(接上页)

```
func dfs(cookies []int, arr []int, index int, maxValue int, count int) {
        if maxValue > res { // 剪枝
                return
        }
        if index == len(cookies) {
                res = maxValue
                return
        }
        if count < len(arr) {
                arr[count] = cookies[index]
                dfs(cookies, arr, index+1, max(maxValue, arr[count]), count+1)
                arr[count] = 0
        }
        for i := 0; i < count; i++ {
                arr[i] = arr[i] + cookies[index]
                dfs(cookies, arr, index+1, max(maxValue, arr[i]), count)
                arr[i] = arr[i] - cookies[index]
        }
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

71.3 2310. 个位数字为 K 的整数之和 (2)

- 题目

```
给你两个整数 num 和 k ，考虑具有以下属性的正整数多重集：
每个整数个位数字都是 k 。
所有整数之和是 num 。
返回该多重集的最小大小，如果不存在这样的多重集，返回 -1 。
注意：多重集与集合类似，但多重集可以包含多个同一整数，空多重集的和为 0 。
个位数字 是数字最右边的数位。
示例 1：输入：num = 58, k = 9 输出：2
解释：多重集 [9,49] 满足题目条件，和为 58 且每个整数的个位数字是 9 。
另一个满足条件的多重集是 [19,39] 。
可以证明 2 是满足题目条件的多重集的最小长度。
示例 2：输入：num = 37, k = 2 输出：-1
```

(续下页)

(接上页)

```
解释：个位数字为 2 的整数无法相加得到 37 。
示例 3：输入：num = 0, k = 7 输出：0
解释：空多重集的和为 0 。
提示：0 <= num <= 3000
0 <= k <= 9
```

- 解题思路

```
func minimumNumbers(num int, k int) int {
        if num == 0 {
                return 0
        }
        if k == 0 {
                if num%10 == 0 {
                        return 1
                }
                return -1
        }
        if num%2 == 1 && k%2 == 0 {
                return -1
        }
        for i := 1; i <= num; i++ {
                sum := i * k
                left := num - sum
                if left%10 == 0 {
                        return i
                }
                if left < 0 {
                        return -1
                }
        }
        return -1
}

# 2
func minimumNumbers(num int, k int) int {
        if num == 0 {
                return 0
        }
        for i := 1; i <= 10; i++ {
                // 只考虑个位数
                if i*k <= num && (i*k)%10 == num%10 {
                        return i
                }
```

(续下页)

(接上页)

```
        }
        return -1
}
```

71.4 2311. 小于等于 K 的最长二进制子序列 (2)

- 题目

```
给你一个二进制字符串s和一个正整数k。
请你返回 s的 最长子序列，且该子序列对应的 二进制数字小于等于 k。
注意：子序列可以有 前导 0。 空字符串视为0。
子序列是指从一个字符串中删除零个或者多个字符后，不改变顺序得到的剩余字符序列。
示例 1：输入：s = "1001010", k = 5 输出：5
解释：s 中小于等于 5 的最长子序列是 "00010" ，对应的十进制数字是 2 。
注意 "00100" 和 "00101" 也是可行的最长子序列，十进制分别对应 4 和 5 。
最长子序列的长度为 5 ，所以返回 5 。
示例 2：输入：s = "00101001", k = 1 输出：6
解释："000001" 是 s 中小于等于 1 的最长子序列，对应的十进制数字是 1 。
最长子序列的长度为 6 ，所以返回 6 。
提示：1 <= s.length <= 1000
s[i] 要么是'0'，要么是'1' 。
1 <= k <= 109
```

- 解题思路

```
func longestSubsequence(s string, k int) int {
        res := 0
        sum, bitValue := int64(0), int64(1)
        target := int64(k)
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '0' { // 0全部加上
                        res++
                } else if sum <= target {
                        sum = sum + bitValue
                        if sum <= target { // 小于<=k加上
                                res++
                        }
                }
                if sum <= target && bitValue <= target {
                        bitValue = bitValue * 2
                }
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func longestSubsequence(s string, k int) int {
        res := 0
        sum, bitValue := 0, 1
        for i := len(s) - 1; i >= 0; i-- {
                if s[i] == '0' { // 0全部加上
                        res++
                } else {
                        if sum+bitValue <= k {
                                res++
                                sum = sum + bitValue
                        }
                }
                if bitValue <= k {
                        bitValue = bitValue * 2
                }
        }
        return res
}
```

71.5 2316. 统计无向图中无法互相到达点对数 (2)

- 题目

```
给你一个整数n，表示一张无向图中有 n个节点，编号为0到n - 1。
同时给你一个二维整数数组edges，其中edges[i] = [ai, bi]表示节点ai␣
↪和bi之间有一条无向边。
请你返回 无法互相到达的不同 点对数目。
示例 1：输入：n = 3, edges = [[0,1],[0,2],[1,2]] 输出：0
解释：所有点都能互相到达，意味着没有点对无法互相到达，所以我们返回 0 。
示例 2：输入：n = 7, edges = [[0,2],[0,5],[2,4],[1,6],[5,4]] 输出：14
解释：总共有 14 个点对互相无法到达：
[[0,1],[0,3],[0,6],[1,2],[1,3],[1,4],[1,5],[2,3],[2,6],[3,4],[3,5],[3,6],[4,6],[5,6]]
所以我们返回 14 。
提示：1 <= n <= 105
0 <= edges.length <= 2 * 105
edges[i].length == 2
0 <= ai, bi < n
ai != bi
```

(续下页)

(接上页)

不会有重复边。

- 解题思路

```
func countPairs(n int, edges [][]int) int64 {
        res := int64(0)
        fa = Init(n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                union(a, b)
        }
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                m[find(i)]++ // 统计每组个数
        }
        for _, v := range m {
                res = res + int64(v)*int64(n-v)
        }
        return res / 2
}

var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        if fa[x] != x {
                fa[x] = find(fa[x])
        }
        return fa[x]
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}
```

(续下页)

(接上页)

```
# 2
var arr [][]int
var visited []bool
var count int

func countPairs(n int, edges [][]int) int64 {
        res := int64(0)
        arr = make([][]int, n)
        visited = make([]bool, n)
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }

        for i := 0; i < n; i++ {
                if visited[i] == false {
                        count = 0
                        dfs(i)
                        res = res + int64(count)*int64(n-count)
                }
        }
        return res / 2
}

func dfs(start int) {
        visited[start] = true
        count++
        for i := 0; i < len(arr[start]); i++ {
                next := arr[start][i]
                if visited[next] == false {
                        dfs(next)
                }
        }
}
```

71.6 2317. 操作后的最大异或和 (1)

- 题目

```
给你一个下标从 0开始的整数数组nums。一次操作中，选择 任意非负整数x和一个下标i，
更新nums[i]为nums[i] AND (nums[i] XOR x)。
注意，AND是逐位与运算，XOR是逐位异或运算。
请你执行 任意次更新操作，并返回nums中所有元素最大逐位异或和。
示例 1：输入：nums = [3,2,4,6] 输出：7
解释：选择 x = 4 和 i = 3 进行操作，num[3] = 6 AND (6 XOR 4) = 6 AND 2 = 2 。
现在，nums = [3, 2, 4, 2] 且所有元素逐位异或得到 3 XOR 2 XOR 4 XOR 2 = 7 。
可知 7 是能得到的最大逐位异或和。
注意，其他操作可能也能得到逐位异或和 7 。
示例 2：输入：nums = [1,2,3,9,2] 输出：11
解释：执行 0 次操作。
所有元素的逐位异或和为 1 XOR 2 XOR 3 XOR 9 XOR 2 = 11 。
可知 11 是能得到的最大逐位异或和。
提示：1 <= nums.length <= 105
0 <= nums[i] <= 108
```

- 解题思路

```
func maximumXOR(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                // 最大逐位异或和：在某位上有奇数个1，能达到最大
                // 更新操作：and操作能把部分位从1置为0，但是不能把0修改为1
                // 求解方案：使用更新操作把相同位置保留1个1，其它置为0 => 转为求位或
                // => 有多少位存在1
                res = res | nums[i] // XOR位或，只要该位为1，结果就是1
        }
        return res
}
```

71.7 2320. 统计放置房子的方式数 (3)

- 题目

```
一条街道上共有 n * 2 个 地块 ，街道的两侧各有 n 个地块。每一边的地块都按从 1 到 n␣
↪编号。每个地块上都可以放置一所房子。
现要求街道同一侧不能存在两所房子相邻的情况，请你计算并返回放置房屋的方式数目。由于答案可能很大，需要
↪109 + 7 取余后再返回。
注意，如果一所房子放置在这条街某一侧上的第 i 个地块，不影响在另一侧的第 i␣
```

(续下页)

(接上页)

```
↪个地块放置房子。
示例 1：输入：n = 1 输出：4
解释：可能的放置方式：
1. 所有地块都不放置房子。
2. 一所房子放在街道的某一侧。
3. 一所房子放在街道的另一侧。
4. 放置两所房子，街道两侧各放置一所。
示例 2：输入：n = 2 输出：9
解释：如上图所示，共有 9 种可能的放置方式。
提示：1 <= n <= 104
```

- 解题思路

```
var mod = 1000000007

func countHousePlacements(n int) int {
        dp := make([][4]int, n+1)
        dp[1][0] = 1 // 2侧没有房子
        dp[1][1] = 1 // 1侧有房子：上
        dp[1][2] = 1 // 1侧有房子：下
        dp[1][3] = 1 // 2侧都有房子
        for i := 2; i <= n; i++ {
                dp[i][0] = (dp[i-1][0] + dp[i-1][1] + dp[i-1][2] + dp[i-1][3]) % mod
                dp[i][1] = (dp[i-1][0] + dp[i-1][2]) % mod
                dp[i][2] = (dp[i-1][0] + dp[i-1][1]) % mod
                dp[i][3] = (dp[i-1][0]) % mod
        }
        sum := 0
        for i := 0; i < 4; i++ {
                sum = (sum + dp[n][i]) % mod
        }
        return sum
}

# 2
var mod = 1000000007

func countHousePlacements(n int) int {
        dp := make([]int, n+3) // 只考虑单侧的方案
        dp[0] = 1
        dp[1] = 2
        for i := 2; i <= n; i++ {
                // 不放；dp[i] = dp[i-1]
                // 放：i-1不能放，dp[i] = dp[i-2]
```

(续下页)

(接上页)

```
                dp[i] = (dp[i-1] + dp[i-2]) % mod
        }
        return dp[n] * dp[n] % mod // 考虑2侧情况
}

# 3
var mod = 1000000007

func countHousePlacements(n int) int {
        var a, b, c, d int
        a = 1 // 2侧没有房子
        b = 1 // 1侧有房子：上
        c = 1 // 1侧有房子：下
        d = 1 // 2侧都有房子
        for i := 2; i <= n; i++ {
                a, b, c, d = (a+b+c+d)%mod, (a+c)%mod, (a+b)%mod, a%mod
        }
        return (a + b + c + d) % mod
}
```

71.8 2326. 螺旋矩阵 IV(1)

- 题目

```
给你两个整数：m 和 n ，表示矩阵的维数。
另给你一个整数链表的头节点 head 。
请你生成一个大小为 m x n 的螺旋矩阵，矩阵包含链表中的所有整数。
链表中的整数从矩阵 左上角 开始、顺时针 按 螺旋 顺序填充。如果还存在剩余的空格，则用 -
↪1 填充。
返回生成的矩阵。
示例 1：输入：m = 3, n = 5, head = [3,0,2,6,8,1,7,9,4,2,5,5,0]
输出：[[3,0,2,6,8],[5,0,-1,-1,1],[5,2,4,9,7]]
解释：上图展示了链表中的整数在矩阵中是如何排布的。
注意，矩阵中剩下的空格用 -1 填充。
示例 2：输入：m = 1, n = 4, head = [0,1,2] 输出：[[0,1,2,-1]]
解释：上图展示了链表中的整数在矩阵中是如何从左到右排布的。
注意，矩阵中剩下的空格用 -1 填充。
提示：1 <= m, n <= 105
1 <= m * n <= 105
链表中节点数目在范围 [1, m * n] 内
0 <= Node.val <= 1000
```

- 解题思路

```
// 顺时针：上右下左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func spiralMatrix(m int, n int, head *ListNode) [][]int {
        res := make([][]int, m)
        for i := 0; i < m; i++ {
                res[i] = make([]int, n)
                for j := 0; j < n; j++ {
                        res[i][j] = -1
                }
        }

        x, y, dir := 0, 0, 0
        for ; head != nil; head = head.Next {
                res[x][y] = head.Val
                newX, newY := x+dx[dir], y+dy[dir]
                if 0 > newX || newX >= m || 0 > newY || newY >= n || res[newX][newY] !
↪= -1 {
                        dir = (dir + 1) % 4 // 换方向
                }
                x = x + dx[dir]
                y = y + dy[dir]
        }
        return res
}
```

71.9 2327. 知道秘密的人数

71.9.1 题目

71.9.2 解题思路

71.10 2332. 坐上公交的最晚时间

71.10.1 题目

给你一个下标从 0开始长度为 n的整数数组buses，其中buses[i]表示第 i辆公交车的出发时间。同时给你一个下标从 0开始长度为␣
↪m的整数数组passengers，其中passengers[j]表示第j位乘客的到达时间。

(续下页)

(接上页)

```
所有公交车出发的时间互不相同，所有乘客到达的时间也互不相同。
给你一个整数capacity，表示每辆公交车最多能容纳的乘客数目。
每位乘客都会搭乘下一辆有座位的公交车。如果你在 y时刻到达，公交在x时刻出发，
满足y <= x且公交没有满，那么你可以搭乘这一辆公交。
最早到达的乘客优先上车。
返回你可以搭乘公交车的最晚到达公交站时间。你 不能跟别的乘客同时刻到达。
注意：数组buses 和passengers不一定是有序的。
示例 1：输入：buses = [10,20], passengers = [2,17,18,19], capacity = 2 输出：16
解释：第 1 辆公交车载着第 1 位乘客。
第 2 辆公交车载着你和第 2 位乘客。
注意你不能跟其他乘客同一时间到达，所以你必须在第二位乘客之前到达。
示例 2：输入：buses = [20,30,10], passengers = [19,13,26,4,25,11,21], capacity = 2␣
↪输出：20
解释：第 1 辆公交车载着第 4 位乘客。
第 2 辆公交车载着第 6 位和第 2 位乘客。
第 3 辆公交车载着第 1 位乘客和你。
提示：n == buses.length
m == passengers.length
1 <= n, m, capacity <= 105
2 <= buses[i], passengers[i] <= 109
buses中的元素 互不相同。
passengers中的元素 互不相同。
```

71.10.2 解题思路

71.11 2336. 无限集中的最小数字 (2)

- 题目

```
现有一个包含所有正整数的集合 [1, 2, 3, 4, 5, ...] 。
实现 SmallestInfiniteSet 类：
SmallestInfiniteSet() 初始化 SmallestInfiniteSet 对象以包含 所有 正整数。
int popSmallest() 移除 并返回该无限集中的最小整数。
void addBack(int num) 如果正整数 num 不 存在于无限集中，则将一个 num 添加␣
↪到该无限集中。
示例：输入["SmallestInfiniteSet", "addBack", "popSmallest", "popSmallest",
"popSmallest", "addBack", "popSmallest", "popSmallest", "popSmallest"]
[[], [2], [], [], [], [1], [], [], []]
```

(续下页)

(接上页)

```
输出 [null, null, 1, 2, 3, null, 1, 4, 5]
解释 SmallestInfiniteSet smallestInfiniteSet = new SmallestInfiniteSet();
smallestInfiniteSet.addBack(2);    // 2 已经在集合中，所以不做任何变更。
smallestInfiniteSet.popSmallest(); // 返回 1 ，因为 1␣
↪是最小的整数，并将其从集合中移除。
smallestInfiniteSet.popSmallest(); // 返回 2 ，并将其从集合中移除。
smallestInfiniteSet.popSmallest(); // 返回 3 ，并将其从集合中移除。
smallestInfiniteSet.addBack(1);    // 将 1 添加到该集合中。
smallestInfiniteSet.popSmallest(); // 返回 1 ，因为 1 在上一步中被添加到集合中，
                                   // 且 1 是最小的整数，并将其从集合中移除。
smallestInfiniteSet.popSmallest(); // 返回 4 ，并将其从集合中移除。
smallestInfiniteSet.popSmallest(); // 返回 5 ，并将其从集合中移除。
提示：1 <= num <= 1000
最多调用 popSmallest 和 addBack 方法 共计 1000 次
```

- 解题思路

```
type SmallestInfiniteSet struct {
        m map[int]bool
}

func Constructor() SmallestInfiniteSet {
        return SmallestInfiniteSet{m: make(map[int]bool)}
}

func (this *SmallestInfiniteSet) PopSmallest() int {
        for i := 1; i <= 1000; i++ {
                if this.m[i] == false {
                        this.m[i] = true
                        return i
                }
        }
        return 0
}

func (this *SmallestInfiniteSet) AddBack(num int) {
        delete(this.m, num)
}

# 2
type SmallestInfiniteSet struct {
        arr []bool
}
```

(续下页)

(接上页)

```
func Constructor() SmallestInfiniteSet {
        return SmallestInfiniteSet{arr: make([]bool, 1001)}
}

func (this *SmallestInfiniteSet) PopSmallest() int {
        for i := 1; i <= 1000; i++ {
                if this.arr[i] == false {
                        this.arr[i] = true
                        return i
                }
        }
        return 0
}

func (this *SmallestInfiniteSet) AddBack(num int) {
        this.arr[num] = false
}
```

71.12 2337. 移动片段得到字符串

71.12.1 题目

```
给你两个字符串 start 和 target ，长度均为 n 。每个字符串 仅 由字符 'L'、'R' 和 '_' 组成，其中：
字符 'L' 和 'R' 表示片段，其中片段 'L' 只有在其左侧直接存在一个 空位 时才能向 左 移动，
而片段 'R' 只有在其右侧直接存在一个 空位 时才能向 右 移动。
字符 '_' 表示可以被 任意 'L' 或 'R' 片段占据的空位。
如果在移动字符串 start 中的片段任意次之后可以得到字符串 target ，返回 true；否则，返回 false 。
示例 1：输入：start = "_L__R__R_", target = "L______RR" 输出：true
解释：可以从字符串 start 获得 target ，需要进行下面的移动：
- 将第一个片段向左移动一步，字符串现在变为 "L___R__R_" 。
- 将最后一个片段向右移动一步，字符串现在变为 "L___R___R" 。
- 将第二个片段向右移动散步，字符串现在变为 "L______RR" 。
可以从字符串 start 得到 target ，所以返回 true 。
示例 2：输入：start = "R_L_", target = "__LR" 输出：false
解释：字符串 start 中的 'R' 片段可以向右移动一步得到 "_RL_" 。
但是，在这一步之后，不存在可以移动的片段，所以无法从字符串 start 得到 target 。
示例 3：输入：start = "_R", target = "R_" 输出：false
解释：字符串 start 中的片段只能向右移动，所以无法从字符串 start 得到 target 。
```

(续下页)

(接上页)

```
提示：n == start.length == target.length
1 <= n <= 105
start 和 target 由字符 'L'、'R' 和 '_' 组成
```

71.12.2 解题思路

71.13 2342. 数位和相等数对的最大和 (1)

- 题目

```
给你一个下标从 0 开始的数组 nums ，数组中的元素都是 正 整数。请你选出两个下标 i 和␣
↪j (i != j) ，
且 nums[i] 的数位和 与 nums[j] 的数位和相等。
请你找出所有满足条件的下标 i 和 j ，找出并返回 nums[i] + nums[j] 可以得到的 最大值 。
示例 1：输入：nums = [18,43,36,13,7] 输出：54
解释：满足条件的数对 (i, j) 为：
- (0, 2) ，两个数字的数位和都是 9 ，相加得到 18 + 36 = 54 。
- (1, 4) ，两个数字的数位和都是 7 ，相加得到 43 + 7 = 50 。
所以可以获得的最大和是 54 。
示例 2：输入：nums = [10,12,19,14] 输出：-1
解释：不存在满足条件的数对，返回 -1 。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 109
```

- 解题思路

```go
func maximumSum(nums []int) int {
        res := -1
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                v := calculate(nums[i])
                if m[v] > 0 { // 存在值
                        res = max(res, m[v]+nums[i]) // 更新最大值
                }
                m[v] = max(m[v], nums[i])
        }
        return res
}
```

(续下页)

(接上页)

```
func calculate(a int) int {
        res := 0
        for a > 0 {
                res = res + a%10
                a = a / 10
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

71.14 2348. 全 0 子数组的数目 (2)

- 题目

```
给你一个整数数组nums，返回全部为0的子数组数目。
子数组是一个数组中一段连续非空元素组成的序列。
示例 1：输入：nums = [1,3,0,0,2,0,0,4] 输出：6
解释：子数组 [0] 出现了 4 次。
子数组 [0,0] 出现了 2 次。
不存在长度大于 2 的全 0 子数组，所以我们返回 6 。
示例 2：输入：nums = [0,0,0,2,0,0] 输出：9
解释：子数组 [0] 出现了 5 次。
子数组 [0,0] 出现了 3 次。
子数组 [0,0,0] 出现了 1 次。
不存在长度大于 3 的全 0 子数组，所以我们返回 9 。
示例 3：输入：nums = [2,10,2019] 输出：0
解释：没有全 0 子数组，所以我们返回 0 。
提示：1 <= nums.length <= 105
-109 <= nums[i] <= 109
```

- 解题思路

```
func zeroFilledSubarray(nums []int) int64 {
        res := int64(0)
        count := int64(0)
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        count++
                        res = res + count
                } else {
                        count = 0
                }
        }
        return res
}

# 2
func zeroFilledSubarray(nums []int) int64 {
        res := int64(0)
        count := int64(0)
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        count++
                } else {
                        res = res + count*(count+1)/2
                        count = 0
                }
        }
        res = res + count*(count+1)/2
        return res
}
```

71.15 2352. 相等行列对 (1)

- 题目

```
给你一个下标从 0 开始、大小为 n x n 的整数矩阵 grid ，返回满足 Ri 行和 Cj 列相等的行列对 (Ri, Cj) 的数目。
如果行和列以相同的顺序包含相同的元素（即相等的数组），则认为二者是相等的。
示例 1：输入：grid = [[3,2,1],[1,7,6],[2,7,7]] 输出：1
解释：存在一对相等行列对：- (第 2 行，第 1 列)：[2,7,7]
示例 2：输入：grid = [[3,1,2,2],[1,4,4,5],[2,4,2,2],[2,4,2,2]] 输出：3
解释：存在三对相等行列对：
- (第 0 行，第 0 列)：[3,1,2,2]
- (第 2 行，第 2 列)：[2,4,2,2]
- (第 3 行，第 2 列)：[2,4,2,2]
提示：n == grid.length == grid[i].length
```

(续下页)

(接上页)

```
1 <= n <= 200
1 <= grid[i][j] <= 105
```

- 解题思路

```
func equalPairs(grid [][]int) int {
        res := 0
        n := len(grid)
        countMap := make(map[[200]int]int)
        for i := 0; i < n; i++ {
                arr := [200]int{}
                for j := 0; j < n; j++ {
                        arr[j] = grid[i][j]
                }
                countMap[arr]++
        }
        for j := 0; j < n; j++ {
                arr := [200]int{}
                for i := 0; i < n; i++ {
                        arr[i] = grid[i][j]
                }
                res = res + countMap[arr]
        }
        return res
}
```

71.16 2358. 分组的最大数量 (3)

- 题目

```
给你一个正整数数组 grades ，表示大学中一些学生的成绩。你打算将 所有 学生分为一些 有序␣
↪的非空分组，其中分组间的顺序满足以下全部条件：
第 i 个分组中的学生总成绩 小于 第 (i + 1)␣
↪个分组中的学生总成绩，对所有组均成立（除了最后一组）。
第 i 个分组中的学生总数 小于 第 (i + 1)␣
↪个分组中的学生总数，对所有组均成立（除了最后一组）。
返回可以形成的 最大 组数。
示例 1：输入：grades = [10,6,12,7,3,5] 输出：3
解释：下面是形成 3 个分组的一种可行方法：
- 第 1 个分组的学生成绩为 grades = [12] ，总成绩：12 ，学生数：1
- 第 2 个分组的学生成绩为 grades = [6,7] ，总成绩：6 + 7 = 13 ，学生数：2
- 第 3 个分组的学生成绩为 grades = [10,3,5] ，总成绩：10 + 3 + 5 = 18 ，学生数：3
```

(续下页)

(接上页)

```
可以证明无法形成超过 3 个分组。
示例 2: 输入: grades = [8,8] 输出: 1
解释: 只能形成 1 个分组，因为如果要形成 2 个分组的话，会导致每个分组中的学生数目相等。
提示: 1 <= grades.length <= 105
1 <= grades[i] <= 105
```

- 解题思路

```
func maximumGroups(grades []int) int {
        n := len(grades)
        res := 1 // 至少可以分为1组
        left, right := 1, n
        for left <= right {
                mid := left + (right-left)/2 // 分为mid组:1+2+3+...
↪+mid (多余的分在最后一组)
                total := (1 + mid) * mid / 2 // 分为mid组至少需要的人数
                if total > n {
                        right = mid - 1
                } else {
                        res = mid
                        left = mid + 1
                }
        }
        return res
}

# 2
func maximumGroups(grades []int) int {
        n := len(grades)
        // k*(k+1)/2 <= n
        // k^2 + k - 2*n <= 0
        ans := math.Sqrt(0.25+2*float64(n)) - 0.5
        // 求根公式
        // ans = (math.Sqrt(1+8*float64(n)) - 1) / 2
        return int(math.Floor(ans)) // 向下取整
}

# 3
func maximumGroups(grades []int) int {
        n := len(grades)
        res := 0
        start := 1
        for start <= n {
                res++
```

(续下页)

(接上页)

```
                n = n - start
                start++
        }
        return res
}
```

71.17 2364. 统计坏数对的数目 (1)

- 题目

```
给你一个下标从0开始的整数数组nums。如果 i < j且j - i != nums[j] -␣
↪nums[i]，那么我们称(i, j)是一个 坏数对。
请你返回 nums中 坏数对的总数目。
示例 1：输入：nums = [4,1,3,3] 输出：5
解释：数对 (0, 1) 是坏数对，因为 1 - 0 != 1 - 4 。
数对 (0, 2) 是坏数对，因为 2 - 0 != 3 - 4, 2 != -1 。
数对 (0, 3) 是坏数对，因为 3 - 0 != 3 - 4, 3 != -1 。
数对 (1, 2) 是坏数对，因为 2 - 1 != 3 - 1, 1 != 2 。
数对 (2, 3) 是坏数对，因为 3 - 2 != 3 - 3, 1 != 0 。
总共有 5 个坏数对，所以我们返回 5 。
示例 2：输入：nums = [1,2,3,4,5] 输出：0
解释：没有坏数对。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 109
```

- 解题思路

```
func countBadPairs(nums []int) int64 {
        res := int64(0)
        n := int64(len(nums))
        m := make(map[int]int64)
        for i := 0; i < len(nums); i++ {
                diff := nums[i] - i
                res = res + m[diff] // 累计相等
                m[diff]++
        }
        return n*(n-1)/2 - res // 总数-不满足条件的数量
}
```

71.18 2365. 任务调度器 II(1)

- 题目

```
给你一个下标从 0开始的正整数数组tasks，表示需要␣
↪按顺序完成的任务，其中tasks[i]表示第i件任务的 类型。
同时给你一个正整数space，表示一个任务完成后，另一个相同类型任务完成前需要间隔的最少天数。
在所有任务完成前的每一天，你都必须进行以下两种操作中的一种：
完成tasks中的下一个任务
休息一天
请你返回完成所有任务所需的 最少天数。
示例 1：输入：tasks = [1,2,1,2,3,1], space = 3 输出：9
解释：9 天完成所有任务的一种方法是：
第 1 天：完成任务 0 。
第 2 天：完成任务 1 。
第 3 天：休息。
第 4 天：休息。
第 5 天：完成任务 2 。
第 6 天：完成任务 3 。
第 7 天：休息。
第 8 天：完成任务 4 。
第 9 天：完成任务 5 。
可以证明无法少于 9 天完成所有任务。
示例 2：输入：tasks = [5,8,8,5], space = 2 输出：6
解释：6 天完成所有任务的一种方法是：
第 1 天：完成任务 0 。
第 2 天：完成任务 1 。
第 3 天：休息。
第 4 天：休息。
第 5 天：完成任务 2 。
第 6 天：完成任务 3 。
可以证明无法少于 6 天完成所有任务。
提示：1 <= tasks.length <= 105
1 <= tasks[i] <= 109
1 <= space <= tasks.length
```

- 解题思路

```
func taskSchedulerII(tasks []int, space int) int64 {
        res := int64(0)
        m := make(map[int]int64)
        for i := 0; i < len(tasks); i++ {
                if m[tasks[i]] == 0 {
                        res = res + 1 // 没出现+1
```

(续下页)

(接上页)

```
                } else {
                        prev := m[tasks[i]]
                        res = max(res+1, prev+int64(space)+1) // 出现：取较大
                }
                m[tasks[i]] = res
        }
        return res
}

func max(a, b int64) int64 {
        if a < b {
                return b
        }
        return a
}
```

71.19 2369. 检查数组是否存在有效划分 (1)

- 题目

```
给你一个下标从 0 开始的整数数组 nums ，你必须将数组划分为一个或多个 连续 子数组。
如果获得的这些子数组中每个都能满足下述条件 之一 ，则可以称其为数组的一种 有效 划分：
子数组 恰 由 2 个相等元素组成，例如，子数组 [2,2] 。
子数组 恰 由 3 个相等元素组成，例如，子数组 [4,4,4] 。
子数组 恰 由 3 个连续递增元素组成，并且相邻元素之间的差值为 1 。
例如，子数组 [3,4,5] ，但是子数组 [1,3,5] 不符合要求。
如果数组 至少 存在一种有效划分，返回 true ，否则，返回 false 。
示例 1：输入：nums = [4,4,4,5,6] 输出：true
解释：数组可以划分成子数组 [4,4] 和 [4,5,6] 。
这是一种有效划分，所以返回 true 。
示例 2：输入：nums = [1,1,1,2] 输出：false
解释：该数组不存在有效划分。
提示：2 <= nums.length <= 105
1 <= nums[i] <= 106
```

- 解题思路

```
func validPartition(nums []int) bool {
        n := len(nums)
        dp := make([]bool, n+1) // dp[i]表示长度为i的数组nums[:i]能否划分成功
        dp[0] = true
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                // 3种有效划分方式
                if i > 0 && dp[i-1] == true && nums[i] == nums[i-1] {
                        dp[i+1] = true
                }
                if i > 1 && dp[i-2] == true && nums[i] == nums[i-1] && nums[i] ==␣
↪nums[i-2] {
                        dp[i+1] = true
                }
                if i > 1 && dp[i-2] == true && nums[i] == nums[i-1]+1 && nums[i] ==␣
↪nums[i-2]+2 {
                        dp[i+1] = true
                }
        }
        return dp[n]
}
```

71.20 2374. 边积分最高的节点 (1)

- 题目

```
给你一个有向图，图中有 n 个节点，节点编号从 0 到 n - 1 ，其中每个节点都 恰有一条␣
↪出边。
图由一个下标从 0 开始、长度为 n 的整数数组 edges 表示，其中 edges[i]␣
↪表示存在一条从节点 i 到节点 edges[i] 的 有向 边。
节点 i 的 边积分 定义为：所有存在一条指向节点 i 的边的节点的 编号 总和。
返回 边积分 最高的节点。如果多个节点的 边积分 相同，返回编号 最小 的那个。
示例 1：输入：edges = [1,0,0,0,0,7,7,5] 输出：7
解释：- 节点 1、2、3 和 4 都有指向节点 0 的边，节点 0 的边积分等于 1 + 2 + 3 + 4 = 10␣
↪。
- 节点 0 有一条指向节点 1 的边，节点 1 的边积分等于 0 。
- 节点 7 有一条指向节点 5 的边，节点 5 的边积分等于 7 。
- 节点 5 和 6 都有指向节点 7 的边，节点 7 的边积分等于 5 + 6 = 11 。
节点 7 的边积分最高，所以返回 7 。
示例 2：输入：edges = [2,0,0,2] 输出：0
解释：- 节点 1 和 2 都有指向节点 0 的边，节点 0 的边积分等于 1 + 2 = 3 。
- 节点 0 和 3 都有指向节点 2 的边，节点 2 的边积分等于 0 + 3 = 3 。
节点 0 和 2 的边积分都是 3 。由于节点 0 的编号更小，返回 0 。
提示：n == edges.length
2 <= n <= 105
0 <= edges[i] < n
edges[i] != i
```

- 解题思路

```
func edgeScore(edges []int) int {
        res := 0
        m := make(map[int]int)
        for i := 0; i < len(edges); i++ {
                v := edges[i]
                m[v] = m[v] + i
                if m[v] > m[res] || m[v] == m[res] && v < res {
                        res = v
                }
        }
        return res
}
```

71.21 2375. 根据模式串构造最小数字

71.21.1 题目

```
给你下标从 0 开始、长度为 n的字符串pattern，它包含两种字符，'I'表示 上升，'D'表示
↪下降。
你需要构造一个下标从 0开始长度为n + 1的字符串，且它要满足以下条件：
num包含数字'1'到'9'，其中每个数字至多使用一次。
如果pattern[i] == 'I'，那么num[i] < num[i + 1]。
如果pattern[i] == 'D'，那么num[i] > num[i + 1]。
请你返回满足上述条件字典序 最小的字符串num。
示例 1：输入：pattern = "IIIDIDDD" 输出："123549876"
解释：下标 0 ，1 ，2 和 4 处，我们需要使 num[i] < num[i+1] 。
下标 3 ，5 ，6 和 7 处，我们需要使 num[i] > num[i+1] 。
一些可能的 num 的值为 "245639871" ，"135749862" 和 "123849765" 。
"123549876" 是满足条件最小的数字。
注意，"123414321" 不是可行解因为数字 '1' 使用次数超过 1 次。
示例 2：输入：pattern = "DDD" 输出："4321"
解释：一些可能的 num 的值为 "9876" ，"7321" 和 "8742" 。
"4321" 是满足条件最小的数字。
提示：1 <= pattern.length <= 8
pattern只包含字符'I' 和'D' 。
```

71.21.2 解题思路

CHAPTER 72

2301-2400-Hard

72.1 2302. 统计得分小于 K 的子数组数目 (2)

- 题目

```
一个数字的 分数定义为数组之和 乘以数组的长度。
比方说，[1, 2, 3, 4, 5]的分数为(1 + 2 + 3 + 4 + 5) * 5 = 75。
给你一个正整数数组nums和一个整数k，请你返回nums中分数严格小于k的非空整数子数组数目。
子数组 是数组中的一个连续元素序列。
示例 1：输入：nums = [2,1,4,3,5], k = 10 输出：6
解释：有 6 个子数组的分数小于 10 ：
- [2] 分数为 2 * 1 = 2 。
- [1] 分数为 1 * 1 = 1 。
- [4] 分数为 4 * 1 = 4 。
- [3] 分数为 3 * 1 = 3 。
- [5] 分数为 5 * 1 = 5 。
- [2,1] 分数为 (2 + 1) * 2 = 6 。
注意，子数组 [1,4] 和 [4,3,5] 不符合要求，因为它们的分数分别为 10 和␣
↪36，但我们要求子数组的分数严格小于 10 。
示例 2：输入：nums = [1,1,1], k = 5 输出：5
解释：除了 [1,1,1] 以外每个子数组分数都小于 5 。
[1,1,1] 分数为 (1 + 1 + 1) * 3 = 9 ，大于 5 。
所以总共有 5 个子数组得分小于 5 。
提示：1 <= nums.length <= 105
1 <= nums[i] <= 105
```

(续下页)

(接上页)

```
1 <= k <= 1015
```

- 解题思路

```
func countSubarrays(nums []int, k int64) int64 {
        var res, sum, count int64
        n := len(nums)
        left := -1
        for right := 0; right < n; right++ {
                sum = sum + int64(nums[right])
                count++
                for left < n && sum*count >= k {
                        left++
                        sum = sum - int64(nums[left])
                        count--
                }
                res = res + count
        }
        return res
}

# 2
func countSubarrays(nums []int, k int64) int64 {
        var res, sum int64
        var left int
        for right := 0; right < len(nums); right++ {
                sum = sum + int64(nums[right])
                for sum*int64(right-left+1) >= k {
                        sum = sum - int64(nums[left])
                        left++
                }
                res = res + int64(right-left+1)
        }
        return res
}
```

72.2 2312. 卖木头块 (3)

- 题目

```
给你两个整数m 和n，分别表示一块矩形木块的高和宽。同时给你一个二维整数数组prices，
其中prices[i] = [hi, wi,␣
↪pricei]表示你可以以pricei元的价格卖一块高为hi宽为wi的矩形木块。
每一次操作中，你必须按下述方式之一执行切割操作，以得到两块更小的矩形木块：
沿垂直方向按高度 完全 切割木块，或
沿水平方向按宽度 完全 切割木块
在将一块木块切成若干小木块后，你可以根据 prices卖木块。你可以卖多块同样尺寸的木块。
你不需要将所有小木块都卖出去。你 不能旋转切好后木块的高和宽。
请你返回切割一块大小为m x n 的木块后，能得到的最多钱数。
注意你可以切割木块任意次。
示例 1: 输入: m = 3, n = 5, prices = [[1,4,2],[2,2,7],[2,1,3]] 输出: 19
解释: 上图展示了一个可行的方案。包括:
- 2 块 2 x 2 的小木块，售出 2 * 7 = 14 元。
- 1 块 2 x 1 的小木块，售出 1 * 3 = 3 元。
- 1 块 1 x 4 的小木块，售出 1 * 2 = 2 元。
总共售出 14 + 3 + 2 = 19 元。
19 元是最多能得到的钱数。
示例 2: 输入: m = 4, n = 6, prices = [[3,2,10],[1,4,2],[4,1,3]] 输出: 32
解释: 上图展示了一个可行的方案。包括:
- 3 块 3 x 2 的小木块，售出 3 * 10 = 30 元。
- 1 块 1 x 4 的小木块，售出 1 * 2 = 2 元。
总共售出 30 + 2 = 32 元。
32 元是最多能得到的钱数。
注意我们不能旋转 1 x 4 的木块来得到 4 x 1 的木块。
提示: 1 <= m, n <= 200
1 <= prices.length <= 2 * 104
prices[i].length == 3
1 <= hi <= m
1 <= wi <= n
1 <= pricei <= 106
所有(hi, wi) 互不相同。
```

- 解题思路

```
func sellingWood(m int, n int, prices [][]int) int64 {
        dp := make([][]int64, m+1)
        for i := 0; i <= m; i++ {
                dp[i] = make([]int64, n+1)
        }
        temp := make(map[[2]int]int) // 转为map方便查找
```

(续下页)

(接上页)

```
        for i := 0; i < len(prices); i++ {
                a, b, c := prices[i][0], prices[i][1], prices[i][2]
                temp[[2]int{a, b}] = c // [a,b] => c
        }
        for i := 1; i <= m; i++ {
                for j := 1; j <= n; j++ {
                        dp[i][j] = int64(temp[[2]int{i, j}])
                        // 2个方向进行切割
                        for k := 1; k < i; k++ {
                                dp[i][j] = max(dp[i][j], dp[k][j]+dp[i-k][j])
                        }
                        for k := 1; k < j; k++ {
                                dp[i][j] = max(dp[i][j], dp[i][k]+dp[i][j-k])
                        }
                }
        }
        return dp[m][n]
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}

# 2
func sellingWood(m int, n int, prices [][]int) int64 {
        dp := make([][]int64, m+1)
        for i := 0; i <= m; i++ {
                dp[i] = make([]int64, n+1)
        }
        for i := 0; i < len(prices); i++ {
                a, b, c := prices[i][0], prices[i][1], prices[i][2]
                dp[a][b] = int64(c) // [a,b] => c
        }
        for i := 1; i <= m; i++ {
                for j := 1; j <= n; j++ {
                        // 2个方向进行切割
                        for k := 1; k <= i/2; k++ {
                                dp[i][j] = max(dp[i][j], dp[k][j]+dp[i-k][j])
                        }
                        for k := 1; k <= j/2; k++ {
```

(续下页)

(接上页)

```
                                dp[i][j] = max(dp[i][j], dp[i][k]+dp[i][j-k])
                        }
                }
        }
        return dp[m][n]
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}

# 3
var temp map[[2]int]int
var dp [][]int64

func sellingWood(m int, n int, prices [][]int) int64 {
        dp = make([][]int64, m+1)
        for i := 0; i <= m; i++ {
                dp[i] = make([]int64, n+1)
                for j := 0; j <= n; j++ {
                        dp[i][j] = -1
                }
        }
        temp = make(map[[2]int]int) // 转为map方便查找
        for i := 0; i < len(prices); i++ {
                a, b, c := prices[i][0], prices[i][1], prices[i][2]
                temp[[2]int{a, b}] = c // [a,b] => c
        }
        for i := 1; i <= m; i++ {
                for j := 1; j <= n; j++ {
                        dp[i][j] = int64(temp[[2]int{i, j}])
                        // 2个方向进行切割
                        for k := 1; k < i; k++ {
                                dp[i][j] = max(dp[i][j], dp[k][j]+dp[i-k][j])
                        }
                        for k := 1; k < j; k++ {
                                dp[i][j] = max(dp[i][j], dp[i][k]+dp[i][j-k])
                        }
                }
        }
```

(续下页)

(接上页)

```
        return dfs(m, n)
}

func dfs(m int, n int) int64 {
        if dp[m][n] != -1 {
                return dp[m][n]
        }
        res := int64(temp[[2]int{m, n}])
        // 2个方向进行切割
        for k := 1; k < m; k++ {
                res = max(res, dfs(k, n)+dfs(m-k, n))
        }
        for k := 1; k < n; k++ {
                res = max(res, dfs(m, k)+dfs(m, n-k))
        }
        dp[m][n] = res
        return res
}

func max(a, b int64) int64 {
        if a > b {
                return a
        }
        return b
}
```

72.3 2321. 拼接数组的最大分数 (1)

- 题目

```
给你两个下标从 0 开始的整数数组 nums1 和 nums2 ，长度都是 n 。
你可以选择两个整数 left 和 right ，其中 0 <= left <= right < n ，
接着 交换 两个子数组 nums1[left...right] 和 nums2[left...right] 。
例如，设 nums1 = [1,2,3,4,5] 和 nums2 = [11,12,13,14,15] ，
整数选择 left = 1 和 right = 2，那么 nums1 会变为 [1,12,13,4,5] 而 nums2 会变为 [11,2,
↪3,14,15] 。
你可以选择执行上述操作 一次 或不执行任何操作。
数组的 分数 取 sum(nums1) 和 sum(nums2) 中的最大值，其中 sum(arr) 是数组 arr␣
↪中所有元素之和。
返回 可能的最大分数 。
子数组 是数组中连续的一个元素序列。arr[left...right]
表示子数组包含 nums 中下标 left 和 right 之间的元素（含 下标 left 和 right␣
```

(续下页)

(接上页)

```
↪对应元素）。
示例 1：输入：nums1 = [60,60,60], nums2 = [10,90,10] 输出：210
解释：选择 left = 1 和 right = 1 ，得到 nums1 = [60,90,60] 和 nums2 = [10,60,10] 。
分数为 max(sum(nums1), sum(nums2)) = max(210, 80) = 210 。
示例 2：输入：nums1 = [20,40,20,70,30], nums2 = [50,20,50,40,20] 输出：220
解释：选择 left = 3 和 right = 4 ，得到 nums1 = [20,40,20,40,20] 和 nums2 = [50,20,50,
↪70,30] 。
分数为 max(sum(nums1), sum(nums2)) = max(140, 220) = 220 。
示例 3：输入：nums1 = [7,11,13], nums2 = [1,1,1] 输出：31
解释：选择不交换任何子数组。
分数为 max(sum(nums1), sum(nums2)) = max(31, 3) = 31 。
提示：n == nums1.length == nums2.length
1 <= n <= 105
1 <= nums1[i], nums2[i] <= 104
```

- 解题思路

```go
func maximumsSplicedArray(nums1 []int, nums2 []int) int {
        return max(solve(nums1, nums2), solve(nums2, nums1)) // 取2种情况最大值
}

func solve(nums1 []int, nums2 []int) int {
        maxDiff := 0
        sum1 := 0
        sum := 0
        // nums1 + maxDiff(num2[left:right]-sum1[left:right])
        // 求maxDiff最大
        // 使用最大子序和来计算maxDiff
        for i := 0; i < len(nums1); i++ {
                sum1 = sum1 + nums1[i]
                // leetcode 53.最大子序和
                sum = max(0, sum+nums2[i]-nums1[i])
                maxDiff = max(maxDiff, sum)
        }
        return sum1 + maxDiff
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

CHAPTER 73

2401-2500-Easy

CHAPTER 74

2401-2500-Medium

CHAPTER 75

2401-2500-Hard

CHAPTER 76

剑指 offer

76.1 参考资料

- leetcode 专栏链接 https://leetcode.cn/problemset/lcof/
- http://zhedahht.blog.163.com/
- 《剑指 Offer》第二版源代码 C++ https://github.com/zhedahht/CodingInterviewChinese2

76.2 面试题 03. 数组中重复的数字 (6)

- 题目

```
找出数组中重复的数字。
在一个长度为 n 的数组 nums 里的所有数字都在 0~n-1 的范围内。
数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。
请找出数组中任意一个重复的数字。
示例 1: 输入: [2, 3, 1, 0, 2, 5, 3] 输出: 2 或 3
限制:
2 <= n <= 100000
```

- 解题思路

```
func findRepeatNumber(nums []int) int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                if _, ok := m[nums[i]]; ok {
                        return nums[i]
                }
                m[nums[i]]++
        }
        return -1
}

#
func findRepeatNumber(nums []int) int {
        sort.Ints(nums)
        prev := nums[0]
        for i := 1; i < len(nums); i++ {
                if nums[i] == prev {
                        return nums[i]
                }
                prev = nums[i]
        }
        return -1
}

#
func findRepeatNumber(nums []int) int {
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i] == nums[j] {
                                return nums[i]
                        }
                }
        }
        return -1
}

#
func findRepeatNumber(nums []int) int {
        for key, value := range nums {
                if key == value {
                        continue
                }
                if value == nums[value] {
                        return nums[value]
```

(续下页)

(接上页)

```
                }
                nums[key], nums[value] = nums[value], nums[key]
        }
        return -1
}

#
func findRepeatNumber(nums []int) int {
        countZero := 0
        for _, value := range nums {
                if value == 0 {
                        if countZero > 0 {
                                return 0
                        }
                        countZero++
                        continue
                }
                if value < 0 {
                        value = -value
                }
                if nums[value] < 0 {
                        return value
                }
                nums[value] = -1 * nums[value]
        }
        return -1
}

#
func findRepeatNumber(nums []int) int {
        for i := 0; i < len(nums); i++ {
                for nums[i] != i {
                        if nums[i] == nums[nums[i]] {
                                return nums[i]
                        }
                        nums[i], nums[nums[i]] = nums[nums[i]], nums[i]
                }
        }
        return -1
}
```

76.3 面试题 04. 二维数组中的查找 (6)

- 题目

```
在一个 n * m␣
↪的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。
请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。
示例：
现有矩阵 matrix 如下：
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
给定 target = 5，返回 true。
给定 target = 20，返回 false。
限制：
0 <= n <= 1000
0 <= m <= 1000
注意：本题与主站 240 题相同：https://leetcode.cn/problems/search-a-2d-matrix-ii/
```

- 解题思路

```
func findNumberIn2DArray(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == target {
                                return true
                        }
                }
        }
        return false
}

# 2
func findNumberIn2DArray(matrix [][]int, target int) bool {
```

(续下页)

(接上页)

```
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        for j := 0; j < len(matrix[i]); j++ {
                                if matrix[i][j] == target {
                                        return true
                                }
                        }
                }
        }
        return false
}

# 3
func findNumberIn2DArray(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        res := binarySearch(matrix[i], target)
                        if res == true {
                                return true
                        }
                }
        }
        return false
}

func binarySearch(arr []int, target int) bool {
        left := 0
        right := len(arr) - 1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] == target {
```

(续下页)

(接上页)

```
                        return true
                } else if arr[mid] > target {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return false
}

# 4
func findNumberIn2DArray(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := len(matrix) - 1
        j := 0
        for i >= 0 && j < len(matrix[0]) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        i--
                } else {
                        j++
                }
        }
        return false
}

# 5
func findNumberIn2DArray(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := 0
        j := len(matrix[0]) - 1
        for j >= 0 && i < len(matrix) {
```

(续下页)

(接上页)

```
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        j--
                } else {
                        i++
                }
        }
        return false
}

# 6
func findNumberIn2DArray(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                index := sort.SearchInts(matrix[i], target)
                if index < len(matrix[i]) && target == matrix[i][index] {
                        return true
                }
        }
        return false
}
```

76.4 面试题 05. 替换空格 (2)

- 题目

```
请实现一个函数，把字符串 s 中的每个空格替换成"%20"。
示例 1：输入：s = "We are happy." 输出："We%20are%20happy."
限制：0 <= s 的长度 <= 10000
```

- 解题思路

```
func replaceSpace(s string) string {
        return strings.Replace(s," ","%20",-1)
        // return strings.ReplaceAll(s, " ", "%20")
}
```

(续下页)

(接上页)

```
#
func replaceSpace(s string) string {
        res := ""
        for i := 0; i < len(s); i++ {
                if s[i] == ' ' {
                        res = res + "%20"
                } else {
                        res = res + string(s[i])
                }
        }
        return res
}
```

76.5 面试题 06. 从尾到头打印链表 (5)

- 题目

```
输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。
示例 1: 输入: head = [1,3,2] 输出: [2,3,1]
限制: 0 <= 链表长度 <= 10000
```

- 解题思路

```
func reversePrint(head *ListNode) []int {
        res := make([]int, 0)
        for head != nil {
                res = append(res, head.Val)
                head = head.Next
        }
        i := 0
        for i < len(res)/2 {
                res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
                i++
        }
        return res
}

# 2
func reversePrint(head *ListNode) []int {
        if head == nil {
```

(续下页)

(接上页)

```
                return []int{}
        }
        res := reversePrint(head.Next)
        res = append(res, head.Val)
        return res
}

# 3
func reversePrint(head *ListNode) []int {
        res := make([]int, 0)
        if head == nil {
                return res
        }
        var newHead *ListNode
        for head != nil {
                next := head.Next
                head.Next = newHead
                newHead = head
                head = next
        }

        for newHead != nil {
                res = append(res, newHead.Val)
                newHead = newHead.Next
        }
        return res
}

# 4
func reversePrint(head *ListNode) []int {
        res := make([]int,0)
        if head == nil{
                return res
        }
        stack := make([]*ListNode, 0)
        for head != nil{
                stack = append(stack, head)
                head = head.Next
        }
        for len(stack) > 0{
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                res = append(res, node.Val)
```

(续下页)

(接上页)

```
        }
        return res
}

# 5
func reversePrint(head *ListNode) []int {
        cur := head
        count := 0
        for head != nil {
                count++
                head = head.Next
        }
        res := make([]int, count)
        for cur != nil {
                res[count-1] = cur.Val
                count--
                cur = cur.Next
        }
        return res
}
```

76.6 面试题 07. 重建二叉树 (3)

- 题目

```
输入某二叉树的前序遍历和中序遍历的结果，请重建该二叉树。
假设输入的前序遍历和中序遍历的结果中都不含重复的数字。
例如，给出
前序遍历 preorder = [3,9,20,15,7]
中序遍历 inorder = [9,3,15,20,7]
返回如下的二叉树：
    3
   / \
  9  20
    /  \
   15   7
限制：0 <= 节点个数 <= 5000
注意：本题与主站 105 题重复：
https://leetcode.cn/problems/
construct-binary-tree-from-preorder-and-inorder-traversal/
```

- 解题思路

```
func buildTree(preorder []int, inorder []int) *TreeNode {
        for k := range inorder {
                if inorder[k] == preorder[0] {
                        return &TreeNode{
                                Val:   preorder[0],
                                Left:  buildTree(preorder[1:k+1], inorder[0:k]),
                                Right: buildTree(preorder[k+1:], inorder[k+1:]),
                        }
                }
        }
        return nil
}

#

func buildTree(preorder []int, inorder []int) *TreeNode {
        if preorder == nil || len(preorder) == 0 {
                return nil
        }
        root := &TreeNode{
                Val: preorder[0],
        }
        length := len(preorder)
        stack := make([]*TreeNode, 0)
        stack = append(stack, root)
        index := 0
        for i := 1; i < length; i++ {
                value := preorder[i]
                node := stack[len(stack)-1]
                if node.Val != inorder[index] {
                        node.Left = &TreeNode{Val: value}
                        stack = append(stack, node.Left)
                } else {
                        for len(stack) > 0 && stack[len(stack)-1].Val ==␣
↪inorder[index] {
                                node = stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                index++
                        }
                        node.Right = &TreeNode{Val: value}
                        stack = append(stack, node.Right)
                }
        }
        return root
```

(续下页)

(接上页)

```
}

#
func buildTree(preorder []int, inorder []int) *TreeNode {
        if len(preorder) == 0 {
                return nil
        }
        return helper(preorder, inorder)
}

func helper(preorder []int, inorder []int) *TreeNode {
        var root *TreeNode
        for k := range inorder {
                if inorder[k] == preorder[0] {
                        root = &TreeNode{Val: preorder[0]}
                        root.Left = helper(preorder[1:k+1], inorder[0:k])
                        root.Right = helper(preorder[k+1:], inorder[k+1:])
                }
        }
        return root
}
```

76.7 面试题 09. 用两个栈实现队列 (1)

- 题目

```
用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，
分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead␣
↪操作返回 -1 )
示例 1: 输入: ["CQueue","appendTail","deleteHead","deleteHead"]
[[],[3],[],[]]
输出: [null,null,3,-1]
示例 2: 输入:
["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]
[[],[],[5],[2],[],[]]
输出: [null,-1,null,null,5,2]
提示:
    1 <= values <= 10000
    最多会对 appendTail、deleteHead 进行 10000 次调用
```

- 解题思路

```
type stack []int

func (s *stack) Push(value int) {
        *s = append(*s, value)
}
func (s *stack) Pop() int {
        value := (*s)[len(*s)-1]
        *s = (*s)[:len(*s)-1]
        return value
}

type CQueue struct {
        tail stack
        head stack
}

func Constructor() CQueue {
        return CQueue{}
}

// 1.入队，tail栈保存
// 2.出队，head不为空，出head；head为空，tail出到head里，最后出head
func (this *CQueue) AppendTail(value int) {
        this.tail.Push(value)
}

func (this *CQueue) DeleteHead() int {
        if len(this.head) != 0 {
                return this.head.Pop()
        } else if len(this.tail) != 0 {
                for len(this.tail) > 0 {
                        this.head.Push(this.tail.Pop())
                }
                return this.head.Pop()
        }
        return -1
}
```

76.8 面试题 10- I. 斐波那契数列 (5)

- 题目

```
写一个函数，输入 n ，求斐波那契（Fibonacci）数列的第 n 项。斐波那契数列的定义如下：
F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), 其中 N > 1.
斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。
答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。
示例 1：输入：n = 2输出：1
示例 2：输入：n = 5输出：5
提示：
    0 <= n <= 100
注意：本题与主站 509 题相同：https://leetcode.cn/problems/fibonacci-number/
```

- 解题思路

```
func fib(n int) int {
        if n == 0 {
                return 0
        }
        if n == 1 {
                return 1
        }
        n1, n2 := 0, 1
        for i := 2; i <= n; i++ {
                n1, n2 = n2, (n1+n2)%1000000007
        }
        return n2
}

#
func fib(n int) int {
        if n == 0 {
                return 0
        }
        if n == 1 {
                return 1
        }
        res := make([]int, n+1)
        res[0] = 0
        res[1] = 1
        for i := 2; i <= n; i++ {
                res[i] = (res[i-1] + res[i-2]) % 1000000007
```

(续下页)

(接上页)

```
        }
        return res[n]
}

#
func fib(n int) int {
        if n == 0{
                return 0
        }
        /*
        ans = [Fn+1 Fn
                   Fn Fn-1]
                = [ 1 0
                        0 1]
        */
        ans := matrix{
                a: 1,
                b: 0,
                c: 0,
                d: 1,
        }
        m := matrix{
                a: 1,
                b: 1,
                c: 1,
                d: 0,
        }
        for n > 0{
                if n % 2 == 1{
                        ans = multi(ans, m)
                }
                m = multi(m, m)
                n = n >> 1
        }
        return ans.b
}

/*
a b
c d
*/
type matrix struct {
        a, b, c, d int
```

(续下页)

(接上页)

```
}

// 矩阵乘法
func multi(x, y matrix) matrix {
        newA := x.a*y.a + x.b*y.c
        newB := x.a*y.b + x.b*y.d
        newC := x.c*y.a + x.d*y.c
        newD := x.c*y.b + x.d*y.d
        return matrix{
                a: newA% 1000000007,
                b: newB% 1000000007,
                c: newC% 1000000007,
                d: newD% 1000000007,
        }
}

#
func fib(n int) int {
        if n == 0 {
                return 0
        }
        /*
                ans = [Fn+1 Fn
                           Fn Fn-1]
                        = [ 1 0
                                  0 1]
        */
        ans := matrix{
                a: 1,
                b: 0,
                c: 0,
                d: 1,
        }
        m := matrix{
                a: 1,
                b: 1,
                c: 1,
                d: 0,
        }
        for n > 0 {
                if n%2 == 1 {
                        ans = multi(ans, m)
                }
```

(续下页)

(接上页)

```
                m = multi(m, m)
                n = n >> 1
        }
        return ans.b
}

/*
a b
c d
*/
type matrix struct {
        a, b, c, d int
}

// 矩阵乘法
func multi(x, y matrix) matrix {
        newA := x.a*y.a + x.b*y.c
        newB := x.a*y.b + x.b*y.d
        newC := x.c*y.a + x.d*y.c
        newD := x.c*y.b + x.d*y.d
        return matrix{
                a: newA % 1000000007,
                b: newB % 1000000007,
                c: newC % 1000000007,
                d: newD % 1000000007,
        }
}


# 4
func fib(n int) int {
        if n == 0 {
                return 0
        }
        /*
                ans = [Fn+1 Fn
                           Fn Fn-1]
                       = [ 1 0
                                0 1]
        */
        ans := matrix{
                a: 1,
                b: 0,
```

(续下页)

(接上页)

```
                c: 0,
                d: 1,
        }
        m := matrix{
                a: 1,
                b: 1,
                c: 1,
                d: 0,
        }
        for n > 0 {
                ans = multi(ans, m)
                n--
        }
        return ans.b
}

/*
a b
c d
*/
type matrix struct {
        a, b, c, d int
}

// 矩阵乘法
func multi(x, y matrix) matrix {
        newA := x.a*y.a + x.b*y.c
        newB := x.a*y.b + x.b*y.d
        newC := x.c*y.a + x.d*y.c
        newD := x.c*y.b + x.d*y.d
        return matrix{
                a: newA % 1000000007,
                b: newB % 1000000007,
                c: newC % 1000000007,
                d: newD % 1000000007,
        }
}

# 5
var m = make(map[int]int)

func fib(n int) int {
        if n == 0 {
```

(续下页)

(接上页)

```
                return 0
        }
        if n == 1 {
                return 1
        }
        if m[n] > 0 {
                return m[n]
        }
        m[n] = (fib(n-1) + fib(n-2)) % 1000000007
        return m[n]
}
```

76.9 面试题 10-II. 青蛙跳台阶问题 (3)

- 题目

```
一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n␣
↪级的台阶总共有多少种跳法。
答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。
示例 1：输入：n = 2输出：2
示例 2：输入：n = 7输出：21
提示：
    0 <= n <= 100
注意：本题与主站 70 题相同：https://leetcode.cn/problems/climbing-stairs/
```

- 解题思路

```
func numWays(n int) int {
        if n <= 1 {
                return 1
        }
        if n == 2 {
                return 2
        }
        dp := make([]int, n+1)
        dp[1] = 1
        dp[2] = 2
        for i := 3; i <= n; i++ {
                dp[i] = (dp[i-1] + dp[i-2]) % 1000000007
        }
        return dp[n]
}
```

(续下页)

(接上页)

```
#
func numWays(n int) int {
        if n <= 1 {
                return 1
        }
        first := 1
        second := 2
        for i := 3; i <= n; i++ {
                third := (first + second) % 1000000007
                first = second
                second = third
        }
        return second
}

# 3
var m = make(map[int]int)

func numWays(n int) int {
        if n <= 1 {
                return 1
        }
        if m[n] > 0 {
                return m[n]
        } else {
                m[n] = (numWays(n-1) + numWays(n-2)) % 1000000007
        }
        return m[n]
}
```

76.10 面试题 11. 旋转数组的最小数字 (4)

- 题目

```
把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。
输入一个递增排序的数组的一个旋转，输出旋转数组的最小元素。
例如，数组 [3,4,5,1,2] 为 [1,2,3,4,5] 的一个旋转，该数组的最小值为1。

示例 1：输入：[3,4,5,1,2]输出：1

示例 2：输入：[2,2,2,0,1]输出：0
```

(续下页)

(接上页)

注意：本题与主站 154 题相同：
https://leetcode.cn/problems/find-minimum-in-rotated-sorted-array-ii/

- 解题思路

```
func minArray(numbers []int) int {
        left := 0
        right := len(numbers) - 1
        for left < right {
                if numbers[left] < numbers[right] {
                        return numbers[left]
                }
                mid := left + (right-left)/2
                if numbers[mid] > numbers[left] {
                        left = mid + 1
                } else if numbers[mid] < numbers[left] {
                        right = mid
                } else {
                        left++
                }
        }
        return numbers[left]
}

#
func minArray(numbers []int) int {
        sort.Ints(numbers)
        return numbers[0]
}

#
func minArray(numbers []int) int {
        for i := 1; i < len(numbers); i++ {
                if numbers[i] < numbers[i-1] {
                        return numbers[i]
                }
        }
        return numbers[0]
}

#
func minArray(numbers []int) int {
        left := 0
        right := len(numbers) - 1
```

(续下页)

(接上页)

```
        mid := left
        for numbers[left] >= numbers[right] {
                if right-left == 1 {
                        mid = right
                        break
                }
                mid = (left + right) / 2
                if numbers[left] == numbers[right] && numbers[mid] == numbers[left] {
                        return minInorder(numbers, left, right)
                }
                if numbers[mid] >= numbers[left] {
                        left = mid
                } else if numbers[mid] <= numbers[right] {
                        right = mid
                }
        }
        return numbers[mid]
}

func minInorder(numbers []int, left, right int) int {
        result := numbers[left]
        for i := left + 1; i <= right; i++ {
                if result > numbers[i] {
                        result = numbers[i]
                }
        }
        return result
}
```

76.11 面试题 12. 矩阵中的路径 (2)

- 题目

```
请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。
路径可以从矩阵中的任意一格开始，每一步可以在矩阵中向左、右、上、下移动一格。
如果一条路径经过了矩阵的某一格，那么该路径不能再次进入该格子。
例如，在下面的3×4的矩阵中包含一条字符串“bfce”的路径（路径中的字母用加粗标出）。

[["a","b","c","e"],
["s","f","c","s"],
["a","d","e","e"]]
但矩阵中不包含字符串“abfb”的路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，
```

(续下页)

(接上页)

```
路径不能再次进入这个格子。
示例 1：输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],
word = "ABCCED"
输出：true
示例 2：输入：board = [["a","b"],["c","d"]], word = "abcd" 输出：false
提示：
    1 <= board.length <= 200
    1 <= board[i].length <= 200
注意：本题与主站 79 题相同：https://leetcode.cn/problems/word-search/
```

- 解题思路

```
func exist(board [][]byte, word string) bool {
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[0]); j++ {
                        if dfs(board, i, j, word, 0) {
                                return true
                        }
                }
        }
        return false
}

func dfs(board [][]byte, i, j int, word string, level int) bool {
        if i < 0 || i >= len(board) || j < 0 || j >= len(board[0]) ||
                board[i][j] != word[level] {
                return false
        }
        if level == len(word)-1 {
                return true
        }
        temp := board[i][j]
        board[i][j] = ' '
        res := dfs(board, i+1, j, word, level+1) ||
                dfs(board, i-1, j, word, level+1) ||
                dfs(board, i, j+1, word, level+1) ||
                dfs(board, i, j-1, word, level+1)
        board[i][j] = temp
        return res
}

#
func exist(board [][]byte, word string) bool {
        visited := make([][]bool, len(board))
```

(续下页)

(接上页)

```
        for i := 0; i < len(board); i++ {
                visited[i] = make([]bool, len(board[0]))
        }
        for i := 0; i < len(board); i++ {
                for j := 0; j < len(board[0]); j++ {
                        if dfs(board, i, j, word, 0, visited) {
                                return true
                        }
                }
        }
        return false
}

func dfs(board [][]byte, i, j int, word string, level int, visited [][]bool) bool {
        res := false
        if i >= 0 && i < len(board) && j >= 0 && j < len(board[0]) &&
                visited[i][j] == false && board[i][j] == word[level] {
                if level == len(word)-1 {
                        return true
                }
                visited[i][j] = true
                level = level + 1
                res = dfs(board, i+1, j, word, level, visited) ||
                        dfs(board, i-1, j, word, level, visited) ||
                        dfs(board, i, j+1, word, level, visited) ||
                        dfs(board, i, j-1, word, level, visited)
                if !res {
                        visited[i][j] = false
                        level = level - 1
                }
        }
        return res
}
```

76.12 面试题 13. 机器人的运动范围 (3)

- 题目

```
地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。
一个机器人从坐标 [0, 0]␣
↪的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），
也不能进入行坐标和列坐标的数位之和大于k的格子。
```

(续下页)

(接上页)

```
例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。
但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

示例 1: 输入：m = 2, n = 3, k = 1 输出：3
示例 2: 输入：m = 3, n = 1, k = 0 输出：1
提示：
    1 <= n,m <= 100
    0 <= k <= 20
```

- 解题思路

```
func movingCount(m int, n int, k int) int {
        if k < 0 || m <= 0 || n <= 0 {
                return 0
        }
        visited := make([]bool, m*n)
        res := dfs(m, n, k, visited, 0, 0)
        return res
}

func dfs(m, n, k int, visited []bool, x, y int) int {
        res := 0
        if check(m, n, k, visited, x, y) {
                visited[x*n+y] = true
                res = 1 + dfs(m, n, k, visited, x+1, y) +
                        dfs(m, n, k, visited, x-1, y) +
                        dfs(m, n, k, visited, x, y+1) +
                        dfs(m, n, k, visited, x, y-1)
        }
        return res
}

func check(m, n, k int, visited []bool, x, y int) bool {
        if x >= 0 && x < m && y >= 0 && y < n &&
                getDigiSum(x)+getDigiSum(y) <= k && visited[x*n+y] == false {
                return true
        }
        return false
}

func getDigiSum(num int) int {
        sum := 0
        for num > 0 {
                sum = sum + num%10
```

(续下页)

(接上页)

```
                num = num / 10
        }
        return sum
}

#
func movingCount(m int, n int, k int) int {
        if k < 0 || m <= 0 || n <= 0 {
                return 0
        }
        res := 1
        visited := make([][]bool, m)
        for i := 0; i < m; i++ {
                visited[i] = make([]bool, n)
        }
        visited[0][0] = true
        for i := 0; i < m; i++ {
                for j := 0; j < n; j++ {
                        if (i-1 >= 0 && visited[i-1][j] == true) || (j-1 >= 0 &&␣
↪visited[i][j-1] == true) {
                                value := getDigiSum(i) + getDigiSum(j)
                                if value <= k {
                                        res++
                                        visited[i][j] = true
                                }
                        }
                }
        }
        return res
}

func getDigiSum(num int) int {
        sum := 0
        for num > 0 {
                sum = sum + num%10
                num = num / 10
        }
        return sum
}

#
func movingCount(m int, n int, k int) int {
        if k < 0 || m <= 0 || n <= 0 {
```

(续下页)

(接上页)

```
                return 0
        }
        res := 0
        visited := make([][]bool, m)
        for i := 0; i < m; i++ {
                visited[i] = make([]bool, n)
        }
        visited[0][0] = true
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{0, 0})
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                x := node[0]
                y := node[1]
                if getDigiSum(x)+getDigiSum(y) <= k {
                        res++
                }
                if x+1 < m && getDigiSum(x+1)+getDigiSum(y) <= k && visited[x+1][y]␣
↪== false {
                        queue = append(queue, [2]int{x + 1, y})
                        visited[x+1][y] = true
                }
                if x-1 >= 0 && getDigiSum(x-1)+getDigiSum(y) <= k && visited[x-1][y]␣
↪== false {
                        queue = append(queue, [2]int{x - 1, y})
                        visited[x-1][y] = true
                }
                if y+1 < n && getDigiSum(x)+getDigiSum(y+1) <= k && visited[x][y+1]␣
↪== false {
                        queue = append(queue, [2]int{x, y + 1})
                        visited[x][y+1] = true
                }
                if y-1 >= 0 && getDigiSum(x)+getDigiSum(y-1) <= k && visited[x][y-1]␣
↪== false {
                        queue = append(queue, [2]int{x, y - 1})
                        visited[x][y-1] = true
                }
        }
        return res
}

func getDigiSum(num int) int {
```

(续下页)

(接上页)

```
        sum := 0
        for num > 0 {
                sum = sum + num%10
                num = num / 10
        }
        return sum
}
```

76.13 面试题 14- I. 剪绳子 (2)

- 题目

```
给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），
每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]*k[1]*...*k[m-1]␣
↪可能的最大乘积是多少？
例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

示例 1：  输入：2 输出：1
解释：2 = 1 + 1, 1 × 1 = 1
示例 2：输入：10 输出：36
解释：10 = 3 + 3 + 4, 3 × 3 × 4 = 36
提示：2 <= n <= 58
注意：本题与主站 343 题相同：https://leetcode.cn/problems/integer-break/
```

- 解题思路

```
func cuttingRope(n int) int {
        if n < 2 {
                return 0
        }
        if n == 2 {
                return 1
        }
        if n == 3 {
                return 2
        }
        dp := make([]int, n+1)
        dp[0] = 0
        dp[1] = 1
        dp[2] = 2
        dp[3] = 3
        for i := 4; i <= n; i++ {
```

(续下页)

(接上页)

```
                max := 0
                for j := 1; j <= i/2; j++ {
                        length := dp[j] * dp[i-j]
                        if length > max {
                                max = length
                        }
                        dp[i] = max
                }
        }
        return dp[n]
}

#
func cuttingRope(n int) int {
        if n < 2 {
                return 0
        }
        if n == 2 {
                return 1
        }
        if n == 3 {
                return 2
        }
        timesOf3 := n / 3
        if n-timesOf3*3 == 1 {
                timesOf3 = timesOf3 - 1
        }
        timesOf2 := (n - timesOf3*3) / 2
        return int(math.Pow(float64(2), float64(timesOf2))) *
                int(math.Pow(float64(3), float64(timesOf3)))
}
```

76.14 面试题 14-II. 剪绳子 II(2)

- 题目

```
给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），
每段绳子的长度记为 k[0],k[1]...k[m] 。请问 k[0]*k[1]*...*k[m] 可能的最大乘积是多少？
例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。
答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。
示例 1：输入：2 输出：1 解释：2 = 1 + 1, 1 × 1 = 1
示例 2:输入：10 输出：36 解释：10 = 3 + 3 + 4, 3 × 3 × 4 = 36
```

(续下页)

(接上页)

```
提示：2 <= n <= 1000
注意：本题与主站 343 题相同：https://leetcode.cn/problems/integer-break/
```

- 解题思路

```
func cuttingRope(n int) int {
        if n < 2 {
                return 0
        }
        if n == 2 {
                return 1
        }
        if n == 3 {
                return 2
        }
        timesOf3 := n / 3
        if n-timesOf3*3 == 1 {
                timesOf3 = timesOf3 - 1
        }
        timesOf2 := (n - timesOf3*3) / 2
        return int(math.Pow(float64(2), float64(timesOf2))) *
                myPow3(timesOf3) % 1000000007
}

func myPow3(n int) int {
        res := 1
        for i := 0; i < n; i++{
                res = (res * 3) % 1000000007
        }
        return res
}

#
func cuttingRope(n int) int {
        if n < 2 {
                return 0
        }
        if n == 2 {
                return 1
        }
        if n == 3 {
                return 2
        }
        timesOf3 := n / 3
```

(续下页)

(接上页)

```
        if n-timesOf3*3 == 1 {
                timesOf3 = timesOf3 - 1
        }
        timesOf2 := (n - timesOf3*3) / 2
        return int(math.Pow(float64(2), float64(timesOf2))) *
                myPow3(timesOf3) % 1000000007
}

func myPow3(n int) int {
        res := 1
        for i := 0; i < n; i++ {
                res = (res * 3) % 1000000007
        }
        return res
}
```

76.15 面试题 15. 二进制中 1 的个数 (4)

- 题目

```
请实现一个函数，输入一个整数，输出该数二进制表示中 1 的个数。
例如，把 9 表示成二进制是 1001，有 2 位是 1。因此，如果输入 9，则该函数输出 2。
示例 1：  输入：00000000000000000000000000001011 输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
示例 2：输入：00000000000000000000000010000000 输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
示例 3：输入：11111111111111111111111111111101 输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
注意：本题与主站 191 题相同：https://leetcode.cn/problems/number-of-1-bits/
```

- 解题思路

```
func hammingWeight(num uint32) int {
        count := 0
        for num != 0 {
                if num&1 == 1 {
                        count++
                }
                num = num >> 1
        }
        return count
}
```

(续下页)

(接上页)

```
#
func hammingWeight(num uint32) int {
        count := 0
        for num != 0 {
                num = num & (num - 1)
                count++
        }
        return count
}

#
func hammingWeight(num uint32) int {
        return strings.Count(strconv.FormatInt(int64(num), 2), "1")
        // return strings.Count(fmt.Sprintf("%b",num),"1")
}

#
func hammingWeight(num uint32) int {
        count := 0
        flag := uint32(1)
        for flag != 0{
                if num & flag == flag{
                        count++
                }
                flag = flag << 1
        }
        return count
}
```

76.16 面试题 16. 数值的整数次方 (4)

- 题目

```
实现函数double Power(double base, int exponent)，求base的exponent次方。
不得使用库函数，同时不需要考虑大数问题。
示例 1:输入: 2.00000, 10 输出: 1024.00000
示例 2:输入: 2.10000, 3 输出: 9.26100
示例 3:输入: 2.00000, -2 输出: 0.25000
解释: 2-2 = 1/22 = 1/4 = 0.25
说明:
    -100.0 < x < 100.0
```

(续下页)

(接上页)

```
    n 是 32 位有符号整数，其数值范围是 [-231, 231 - 1] 。
注意：本题与主站 50 题相同：
https://leetcode.cn/problems/powx-n/
```

- 解题思路

```
func myPow(x float64, n int) float64 {
        if n == 0 {
                return 1
        }
        if n == 1 {
                return x
        }
        res := 1.0
        if n > 0 {
                res = myPow(x, n/2)
                return res * res * myPow(x, n%2)
        } else {
                res = myPow(x, -n/2)
                res = res * res * myPow(x, -n%2)
                return 1 / res
        }
}

#
func myPow(x float64, n int) float64 {
        if n == 0 {
                return 1
        }
        if n == 1 {
                return x
        }
        res := 1.0
        if n < 0 {
                n = -n
                x = 1 / x
        }
        for n >= 1 {
                if n%2 == 1 {
                        res = res * x
                        n--
                } else {
                        x = x * x
                        n = n / 2
```

(续下页)

(接上页)

```
                }
        }
        return res
}

#
func myPow(x float64, n int) float64 {
        return math.Pow(x, float64(n))
}

#
func myPow(x float64, n int) float64 {
        if n == 0 {
                return 1
        }
        if n < 0 {
                return 1 / myPow(x, -n)
        }
        if n%2 == 1 {
                return x * myPow(x, n-1)
        }
        return myPow(x*x, n/2)
}
```

76.17 面试题 17. 打印从 1 到最大的 n 位数 (4)

- 题目

```
输入数字 n，按顺序打印出从 1 到最大的 n 位十进制数。
比如输入 3，则打印出 1、2、3 一直到最大的 3 位数 999。
示例 1: 输入: n = 1 输出: [1,2,3,4,5,6,7,8,9]
说明:
    用返回一个整数列表来代替打印
    n 为正整数
```

- 解题思路

```
func printNumbers(n int) []int {
        res := make([]int, 0)
        maxValue := 0
        for n > 0 {
                maxValue = maxValue*10 + 9
```

(续下页)

(接上页)

```
                n--
        }
        for i := 1; i <= maxValue; i++ {
                res = append(res, i)
        }
        return res
}

#
func printNumbers(n int) []int {
        res := make([]int, 0)
        maxValue := 1
        for i := 1; i <= n; i++ {
                maxValue = maxValue * 10
        }
        maxValue = maxValue - 1
        for i := 1; i <= maxValue; i++ {
                res = append(res, i)
        }
        return res
}

# 3
var temp []string

func printNumbers(n int) []int {
        if n <= 0 {
                return nil
        }
        temp = make([]string, 0)
        arr := make([]rune, n)
        for i := 0; i < 10; i++ {
                arr[0] = rune(i + '0')
                dfs(arr, n, 0)
        }

        res := make([]int, 0)
        for i := 0; i < len(temp); i++ {
                value, _ := strconv.Atoi(temp[i])
                res = append(res, value)
        }
        return res
}
```

(续下页)

(接上页)

```
func dfs(arr []rune, n int, index int) {
        if index == n-1 {
                if printNum(arr) != "" {
                        temp = append(temp, printNum(arr))
                }
                return
        }
        for i := 0; i < 10; i++ {
                arr[index+1] = rune(i + '0')
                dfs(arr, n, index+1)
        }
}

func printNum(arr []rune) string {
        res := ""
        isBeginning := true
        for i := 0; i < len(arr); i++ {
                if isBeginning && arr[i] != '0' {
                        isBeginning = false
                }
                if !isBeginning {
                        res = res + string(arr[i])
                }
        }
        return res
}

# 4
func printNumbers(n int) []int {
        res := make([]int, 0)
        if n <= 0 {
                return nil
        }
        temp := make([]string, 0)
        arr := make([]rune, n)
        for i := 0; i < len(arr); i++ {
                arr[i] = '0'
        }
        for !increment(arr) {
                if printNum(arr) != "" {
                        temp = append(temp, printNum(arr))
                }
```

(续下页)

(接上页)

```
        }
        for i := 0; i < len(temp); i++ {
                value, _ := strconv.Atoi(temp[i])
                res = append(res, value)
        }
        return res
}

func increment(arr []rune) bool {
        isOverflow := false
        nTakeOver := 0
        for i := len(arr) - 1; i >= 0; i-- {
                sum := int(arr[i]-'0') + nTakeOver
                if i == len(arr)-1 {
                        sum++
                }
                if sum >= 10 {
                        if i == 0 {
                                isOverflow = true
                        } else {
                                sum = sum - 10
                                nTakeOver = 1
                                arr[i] = rune('0' + sum)
                        }
                } else {
                        arr[i] = rune('0' + sum)
                        break
                }
        }
        return isOverflow
}

func printNum(arr []rune) string {
        res := ""
        isBeginning := true
        for i := 0; i < len(arr); i++ {
                if isBeginning && arr[i] != '0' {
                        isBeginning = false
                }
                if !isBeginning {
                        res = res + string(arr[i])
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

76.18 面试题 18. 删除链表的节点 (2)

- 题目

```
给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。
返回删除后的链表的头节点。
注意：此题对比原题有改动
示例 1:输入: head = [4,5,1,9], val = 5 输出: [4,1,9]
解释: 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -
↪> 9.
示例 2:输入: head = [4,5,1,9], val = 1 输出: [4,5,9]
解释: 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -
↪> 9.
说明:
    题目保证链表中节点的值互不相同
    若使用 C 或 C++ 语言，你不需要 free 或 delete 被删除的节点
```

- 解题思路

```
func deleteNode(head *ListNode, val int) *ListNode {
        headPre := &ListNode{Next: head}
        temp := headPre
        for temp.Next != nil {
                if temp.Next.Val == val {
                        temp.Next = temp.Next.Next
                        break
                } else {
                        temp = temp.Next
                }
        }
        return headPre.Next
}

#
func deleteNode(head *ListNode, val int) *ListNode {
        if head == nil {
                return nil
        }
        head.Next = deleteNode(head.Next, val)
```

(续下页)

(接上页)

```
        if head.Val == val {
                return head.Next
        }
        return head
}
```

76.19 面试题 19. 正则表达式匹配 (3)

- 题目

```
请实现一个函数用来匹配包含'. '和'*'的正则表达式。模式中的字符'.'表示任意一个字符，
而'*'表示它前面的字符可以出现任意次（含0次）。
在本题中，匹配是指字符串的所有字符匹配整个模式。
例如，字符串"aaa"与模式"a.a"和"ab*ac*a"匹配，但与"aa.a"和"ab*a"均不匹配。

示例 1:输入: s = "aa" p = "a" 输出: false
解释: "a" 无法匹配 "aa" 整个字符串。
示例 2:输入: s = "aa" p = "a*" 输出: true
解释: 因为 '*' 代表可以匹配零个或多个前面的那一个元素, 在这里前面的元素就是 'a'。
因此，字符串 "aa" 可被视为 'a' 重复了一次。
示例 3:输入:s = "ab" p = ".*" 输出: true
解释: ".*" 表示可匹配零个或多个 ('*') 任意字符 ('.') 。
示例 4: 输入:s = "aab" p = "c*a*b" 输出: true
解释: 因为 '*' 表示零个或多个，这里 'c' 为 0 个，'a' 被重复一次。因此可以匹配字符串
↪"aab"。
示例 5: 输入: s = "mississippi" p = "mis*is*p*." 输出: false
    s 可能为空，且只包含从 a-z 的小写字母。
    p 可能为空，且只包含从 a-z 的小写字母以及字符 . 和 *，无连续的 '*'。
注意: 本题与主站 10 题相同:
https://leetcode.cn/problems/regular-expression-matching/
```

- 解题思路

```
func isMatch(s string, p string) bool {
        return dfs(s, p, 0, 0)
}

func dfs(s string, p string, i, j int) bool {
        if i >= len(s) && j >= len(p) {
                return true
        }
        if i <= len(s) && j >= len(p) {
```

(续下页)

(接上页)

```
                return false
        }
        if j+1 < len(p) && p[j+1] == '*' {
                if (i < len(s) && p[j] == s[i]) || (p[j] == '.' && i < len(s)) {
                        return dfs(s, p, i+1, j+2) ||
                                dfs(s, p, i+1, j) ||
                                dfs(s, p, i, j+2)
                } else {
                        return dfs(s, p, i, j+2)
                }
        }
        if (i < len(s) && s[i] == p[j]) || (p[j] == '.' && i < len(s)) {
                return dfs(s, p, i+1, j+1)
        }
        return false
}

#
func isMatch(s string, p string) bool {
        // dp[i][j]表示p[:i]能否正则匹配s[:j]
        dp := make([][]bool, len(p)+1)
        for i := 0; i < len(p)+1; i++ {
                dp[i] = make([]bool, len(s)+1)
        }
        // 1.初始化
        dp[0][0] = true
        for i := 2; i < len(p)+1; i++ {
                if i%2 == 0 && p[i-1] == '*' {
                        dp[i][0] = dp[i-2][0]
                }
        }
        // 2.dp状态转移
        for i := 1; i < len(p)+1; i++ {
                for j := 1; j < len(s)+1; j++ {
                        // 2.1 相同或者 .
                        if p[i-1] == s[j-1] || p[i-1] == '.' {
                                dp[i][j] = dp[i-1][j-1]
                        } else if p[i-1] == '*' {
                                if i > 1 {
                                        if p[i-2] == s[j-1] || p[i-2] == '.' {
                                                dp[i][j] = dp[i][j-1] || dp[i-2][j-1]␣
↪|| dp[i-2][j]
                                        } else {
```

(续下页)

(接上页)

```
                                                dp[i][j] = dp[i-2][j]
                                        }
                                }
                        }
                }
        }
        return dp[len(p)][len(s)]
}

# 3
func isMatch(s string, p string) bool {
        if len(s) == 0 && len(p) == 0 {
                return true
        } else if len(p) == 0 {
                return false
        }
        match := false
        // 正常匹配条件=>相等，或者 p[0]等于.就不用管s[0]
        if len(s) > 0 && (s[0] == p[0] || p[0] == '.') {
                match = true
        }
        // 匹配多个 就把 s 往后移1位，注意p不移动
        // 匹配0个 就把 p 往后移2位，相当于p的*当前作废
        if len(p) > 1 && p[1] == '*' {
                return (match && isMatch(s[1:], p)) || isMatch(s, p[2:])
        }
        // 匹配当前成功，同时往后移
        return match && isMatch(s[1:], p[1:])
}
```

76.20 面试题 20. 表示数值的字符串 (1)

- 题目

```
请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100"、"5e2
↪"、"-123"、"3.1416"、"0123"都表示数值，
但"12e"、"1a3.14"、"1.2.3"、"+-5"、"-1E-16"及"12e+5.4"都不是。
注意：本题与主站 65 题相同：https://leetcode.cn/problems/valid-number/
```

- 解题思路

```
func isNumber(s string) bool {
        s = strings.Trim(s, " ")
        if s == "" || len(s) == 0 || len(s) == 0 {
                return false
        }
        arr := []byte(s)
        i := 0
        numeric := scanInteger(&arr, &i)
        if i < len(arr) && arr[i] == '.' {
                i++
                numeric = scanUnsignedInteger(&arr, &i) || numeric
        }
        if i < len(arr) && (arr[i] == 'e' || arr[i] == 'E') {
                i++
                numeric = numeric && scanInteger(&arr, &i)
        }
        return numeric && len(arr) == i
}

func scanInteger(arr *[]byte, index *int) bool {
        if len(*arr) <= *index {
                return false
        }
        if (*arr)[*index] == '+' || (*arr)[*index] == '-' {
                *index++
        }
        return scanUnsignedInteger(arr, index)
}

func scanUnsignedInteger(arr *[]byte, index *int) bool {
        j := *index
        for *index < len(*arr) {
                if (*arr)[*index] < '0' || (*arr)[*index] > '9' {
                        break
                }
                *index++
        }
        return j < *index
}
```

76.21 面试题 21. 调整数组顺序使奇数位于偶数前面 (4)

- 题目

```
输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，
所有偶数位于数组的后半部分。
示例：输入：nums = [1,2,3,4] 输出：[1,3,2,4]
注：[3,1,2,4] 也是正确的答案之一。
提示：
    1 <= nums.length <= 50000
    1 <= nums[i] <= 10000
```

- 解题思路

```
func exchange(nums []int) []int {
        i := 0
        j := len(nums) - 1
        for i < j {
                if nums[i]%2 == 1 {
                        i++
                } else if nums[j]%2 == 0 {
                        j--
                } else {
                        nums[i], nums[j] = nums[j], nums[i]
                }
        }
        return nums
}

#
func exchange(nums []int) []int {
        i := 0
        j := len(nums) - 1
        for i < j {
                for i < j && nums[i]%2 == 1 {
                        i++
                }
                for i < j && nums[j]%2 == 0 {
                        j--
                }
                nums[i], nums[j] = nums[j], nums[i]
        }
        return nums
}
```

(续下页)

(接上页)

```
#
func exchange(nums []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(nums); i++ {
                if nums[i]%2 == 1 {
                        res = append(res, nums[i])
                }
        }
        for i := 0; i < len(nums); i++ {
                if nums[i]%2 == 0 {
                        res = append(res, nums[i])
                }
        }
        return res
}

#
func exchange(nums []int) []int {
        count := 0
        for i := 0; i < len(nums); i++ {
                if nums[i]%2 == 1 {
                        nums[count], nums[i] = nums[i], nums[count]
                        count++
                }
        }
        return nums
}
```

76.22 面试题 22. 链表中倒数第 k 个节点 (5)

- 题目

```
输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，
即链表的尾节点是倒数第1个节点。
例如，一个链表有6个节点，从头节点开始，它们的值依次是1、2、3、4、5、6。
这个链表的倒数第3个节点是值为4的节点。
示例：给定一个链表：1->2->3->4->5，和 k = 2. 返回链表 4->5.
```

- 解题思路

```
func getKthFromEnd(head *ListNode, k int) *ListNode {
        arr := make([]*ListNode, 0)
        for head != nil {
                arr = append(arr, head)
                head = head.Next
        }
        if len(arr) >= k {
                return arr[len(arr)-k]
        }
        return nil
}

#
func getKthFromEnd(head *ListNode, k int) *ListNode {
        fast := head
        for k > 0 && head != nil {
                fast = fast.Next
                k--
        }
        if k > 0 {
                return nil
        }
        slow := head
        for fast != nil {
                fast = fast.Next
                slow = slow.Next
        }
        return slow
}

#
func getKthFromEnd(head *ListNode, k int) *ListNode {
        temp := head
        count := 0
        for temp != nil {
                count++
                temp = temp.Next
        }
        if count < k {
                return nil
        }
        for i := 0; i < count-k; i++ {
                head = head.Next
        }
```

(续下页)

(接上页)

```
        return head
}

#
func getKthFromEnd(head *ListNode, k int) *ListNode {
        res, count := dfs(head, k)
        if count > 0 {
                return nil
        }
        return res
}

func dfs(node *ListNode, k int) (*ListNode, int) {
        if node == nil {
                return node, k
        }
        next, nextValue := dfs(node.Next, k)
        if nextValue <= 0 {
                return next, nextValue
        }
        nextValue = nextValue - 1
        return node, nextValue
}

#
var res *ListNode
var count int

func getKthFromEnd(head *ListNode, k int) *ListNode {
        if head == nil {
                count = 0
                return nil
        }
        getKthFromEnd(head.Next, k)
        count++
        if count == k {
                res = head
        }
        return res
}
```

76.23 面试题 24. 反转链表 (4)

- 题目

```
定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。
示例:
输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL
限制: 0 <= 节点个数 <= 5000
注意: 本题与主站 206 题相同: https://leetcode.cn/problems/reverse-linked-list/
```

- 解题思路

```
func reverseList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        result := reverseList(head.Next)
        head.Next.Next = head
        head.Next = nil
        return result
}

#
func reverseList(head *ListNode) *ListNode {
        var result *ListNode
        var temp *ListNode
        for head != nil {
                temp = head.Next
                head.Next = result
                result = head
                head = temp
        }
        return result
}

#
func reverseList(head *ListNode) *ListNode {
        result := &ListNode{}
        arr := make([]*ListNode, 0)
        for head != nil {
                arr = append(arr, head)
                head = head.Next
        }
```

(续下页)

(接上页)

```
        temp := result
        for i := len(arr) - 1; i >= 0; i-- {
                arr[i].Next = nil
                temp.Next = arr[i]
                temp = temp.Next
        }
        return result.Next
}

#
func reverseList(head *ListNode) *ListNode {
        var res *ListNode
        for {
                if head == nil {
                        break
                }
                res = &ListNode{head.Val, res}
                head = head.Next
        }
        return res
}
```

76.24 面试题 25. 合并两个排序的链表 (3)

- 题目

```
输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。
示例1：输入：1->2->4, 1->3->4 输出：1->1->2->3->4->4
限制：0 <= 链表长度 <= 1000
注意：本题与主站 21 题相同：https://leetcode.cn/problems/merge-two-sorted-lists/
```

- 解题思路

```
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        if l1 == nil {
                return l2
        }
        if l2 == nil {
                return l1
        }

        var head, node *ListNode
```

(续下页)

(接上页)

```
        if l1.Val < l2.Val {
                head = l1
                node = l1
                l1 = l1.Next
        } else {
                head = l2
                node = l2
                l2 = l2.Next
        }

        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        node.Next = l1
                        l1 = l1.Next
                } else {
                        node.Next = l2
                        l2 = l2.Next
                }
                node = node.Next
        }
        if l1 != nil {
                node.Next = l1
        }
        if l2 != nil {
                node.Next = l2
        }
        return head
}

#
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        if l1 == nil {
                return l2
        }
        if l2 == nil {
                return l1
        }

        if l1.Val < l2.Val {
                l1.Next = mergeTwoLists(l1.Next, l2)
                return l1
        } else {
                l2.Next = mergeTwoLists(l1, l2.Next)
```

(续下页)

(接上页)

```
                return l2
        }
}

#
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}
```

76.25 面试题 26. 树的子结构 (2)

- 题目

```
输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)
B是A的子结构， 即 A中有出现和B相同的结构和节点值。
例如:给定的树 A:

     3
    / \
   4   5
  / \
 1   2
给定的树 B:
   4
  /
```

(续下页)

(接上页)

```
 1
返回 true, 因为 B 与 A 的一个子树拥有相同的结构和节点值。
示例 1: 输入: A = [1,2,3], B = [3,1]输出: false
示例 2: 输入: A = [3,4,5,1,2], B = [4,1]输出: true
限制: 0 <= 节点个数 <= 10000
```

- 解题思路

```
func isSubStructure(A *TreeNode, B *TreeNode) bool {
        if A == nil || B == nil {
                return false
        }
        return dfs(A, B) || isSubStructure(A.Left, B) || isSubStructure(A.Right, B)
}

func dfs(A *TreeNode, B *TreeNode) bool {
        if B == nil {
                return true
        }
        if A == nil {
                return false
        }
        return dfs(A.Left, B.Left) && dfs(A.Right, B.Right) && A.Val == B.Val
}

#
func isSubStructure(A *TreeNode, B *TreeNode) bool {
        if A == nil || B == nil {
                return false
        }

        res := false
        list := make([]*TreeNode, 0)
        list = append(list, A)
        for len(list) > 0 {
                node := list[0]
                list = list[1:]
                if node.Val == B.Val {
                        res = dfs(node, B)
                        if res == true {
                                return true
                        }
                }
                if node.Left != nil {
```

(续下页)

(接上页)

```
                        list = append(list, node.Left)
                }
                if node.Right != nil {
                        list = append(list, node.Right)
                }
        }
        return res
}

func dfs(A *TreeNode, B *TreeNode) bool {
        fmt.Println(A, B)
        if B == nil {
                return true
        }
        if A == nil {
                return false
        }
        return dfs(A.Left, B.Left) && dfs(A.Right, B.Right) && A.Val == B.Val
}
```

76.26 面试题 27. 二叉树的镜像 (2)

- 题目

```
请完成一个函数，输入一个二叉树，该函数输出它的镜像。
例如输入：
     4
   /   \
  2     7
 / \   / \
1   3 6   9
镜像输出：
     4
   /   \
  7     2
 / \   / \
9   6 3   1
示例 1：输入：root = [4,2,7,1,3,6,9] 输出：[4,7,2,9,6,3,1]
限制：0 <= 节点个数 <= 1000
注意：本题与主站 226 题相同：https://leetcode.cn/problems/invert-binary-tree/
```

- 解题思路

```
func mirrorTree(root *TreeNode) *TreeNode {
        if root == nil || (root.Left == nil && root.Right == nil) {
                return root
        }
        root.Left, root.Right = mirrorTree(root.Right), mirrorTree(root.Left)
        return root
}

#
func mirrorTree(root *TreeNode) *TreeNode {
        if root == nil {
                return root
        }
        var queue []*TreeNode
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                node.Left, node.Right = node.Right, node.Left
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return root
}
```

76.27 面试题 28. 对称的二叉树 (2)

- 题目

```
请实现一个函数，用来判断一棵二叉树是不是对称的。如果一棵二叉树和它的镜像一样，那么它是对称的。
例如，二叉树 [1,2,2,3,4,4,3] 是对称的。
    1
   / \
  2   2
 / \ / \
3  4 4  3
但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的:
    1
```

(续下页)

(接上页)

```
   / \
  2   2
   \   \
   3    3
示例 1: 输入: root = [1,2,2,3,4,4,3] 输出: true
示例 2: 输入: root = [1,2,2,null,3,null,3] 输出: false
限制: 0 <= 节点个数 <= 1000
注意: 本题与主站 101 题相同: https://leetcode.cn/problems/symmetric-tree/
```

- 解题思路

```
func isSymmetric(root *TreeNode) bool {
        if root == nil {
                return true
        }
        return recur(root.Left, root.Right)
}

func recur(left, right *TreeNode) bool {
        if left == nil && right == nil {
                return true
        }
        if left == nil || right == nil {
                return false
        }

        return left.Val == right.Val &&
                recur(left.Left, right.Right) &&
                recur(left.Right, right.Left)
}

#
func isSymmetric(root *TreeNode) bool {
        leftQ := make([]*TreeNode, 0)
        rightQ := make([]*TreeNode, 0)
        leftQ = append(leftQ, root)
        rightQ = append(rightQ, root)

        for len(leftQ) != 0 && len(rightQ) != 0 {
                leftCur, rightCur := leftQ[0], rightQ[0]
                leftQ, rightQ = leftQ[1:], rightQ[1:]

                if leftCur == nil && rightCur == nil {
                        continue
```

(续下页)

(接上页)

```
                } else if leftCur != nil && rightCur != nil && leftCur.Val ==␣
↪rightCur.Val {
                        leftQ = append(leftQ, leftCur.Left, leftCur.Right)
                        rightQ = append(rightQ, rightCur.Right, rightCur.Left)
                } else {
                        return false
                }
        }

        if len(leftQ) == 0 && len(rightQ) == 0 {
                return true
        } else {
                return false
        }
}
```

76.28 面试题 29. 顺时针打印矩阵 (2)

- 题目

```
输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。
示例 1：输入：matrix = [[1,2,3],[4,5,6],[7,8,9]] 输出：[1,2,3,6,9,8,7,4,5]
示例 2：输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]] 输出：[1,2,3,4,8,12,11,10,9,
↪5,6,7]
限制：0 <= matrix.length <= 100  0 <= matrix[i].length <= 100
注意：本题与主站 54 题相同：https://leetcode.cn/problems/spiral-matrix/
```

- 解题思路

```
var res []int

func spiralOrder(matrix [][]int) []int {
        res = make([]int, 0)
        rows := len(matrix)
        if rows == 0 {
                return res
        }
        cols := len(matrix[0])
        if cols == 0 {
                return res
        }
        start := 0
```

(续下页)

(接上页)

```
        for cols > start*2 && rows > start*2 {
                printCircle(matrix, cols, rows, start)
                start++
        }
        return res
}

func printCircle(matrix [][]int, cols, rows, start int) {
        x := cols - 1 - start
        y := rows - 1 - start
        // 左到右
        for i := start; i <= x; i++ {
                res = append(res, matrix[start][i])
        }
        // 上到下
        if start < y {
                for i := start + 1; i <= y; i++ {
                        res = append(res, matrix[i][x])
                }
        }
        // 右到左
        if start < x && start < y {
                for i := x - 1; i >= start; i-- {
                        res = append(res, matrix[y][i])
                }
        }
        // 下到上
        if start < x && start < y-1 {
                for i := y - 1; i >= start+1; i-- {
                        res = append(res, matrix[i][start])
                }
        }
}

#
func spiralOrder(matrix [][]int) []int {
        res := make([]int, 0)
        rows := len(matrix)
        if rows == 0 {
                return res
        }
        cols := len(matrix[0])
        if cols == 0 {
```

(续下页)

(接上页)

```
                return res
        }
        x1, x2, y1, y2 := 0, rows-1, 0, cols-1
        direct := 0
        for x1 <= x2 && y1 <= y2 {
                direct = (direct + 4) % 4
                if direct == 0 {
                        for i := y1; i <= y2; i++ {
                                res = append(res, matrix[x1][i])
                        }
                        x1++
                } else if direct == 1 {
                        for i := x1; i <= x2; i++ {
                                res = append(res, matrix[i][y2])
                        }
                        y2--
                } else if direct == 2 {
                        for i := y2; i >= y1; i-- {
                                res = append(res, matrix[x2][i])
                        }
                        x2--
                } else if direct == 3 {
                        for i := x2; i >= x1; i-- {
                                res = append(res, matrix[i][y1])
                        }
                        y1++
                }
                direct++
        }
        return res
}
```

76.29 面试题 30. 包含 min 函数的栈 (2)

- 题目

```
定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，
调用 min、push 及 pop 的时间复杂度都是 O(1)。
示例:
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
```

(续下页)

(接上页)

```
minStack.push(-3);
minStack.min();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.min();   --> 返回 -2.
提示：
    各函数的调用总次数不超过 20000 次
注意：本题与主站 155 题相同：https://leetcode.cn/problems/min-stack/
```

- 解题思路

```
type item struct {
        min, x int
}

type MinStack struct {
        stack []item
}

func Constructor() MinStack {
        return MinStack{}
}

func (m *MinStack) Push(x int) {
        min := x
        if len(m.stack) > 0 && m.Min() < x {
                min = m.Min()
        }
        m.stack = append(m.stack, item{
                min: min,
                x:   x,
        })
}

func (m *MinStack) Pop() {
        m.stack = m.stack[:len(m.stack)-1]
}

func (m *MinStack) Top() int {
        if len(m.stack) == 0 {
                return 0
        }
        return m.stack[len(m.stack)-1].x
}
```

(续下页)

(接上页)

```
func (m *MinStack) Min() int {
        if len(m.stack) == 0 {
                return 0
        }
        return m.stack[len(m.stack)-1].min
}

#
type MinStack struct {
        data []int
        min  []int
}

func Constructor() MinStack {
        return MinStack{[]int{}, []int{}}
}

func (m *MinStack) Push(x int) {
        if len(m.data) == 0 || x <= m.Min() {
                m.min = append(m.min, x)
        }
        m.data = append(m.data, x)
}

func (m *MinStack) Pop() {
        x := m.data[len(m.data)-1]
        m.data = m.data[:len(m.data)-1]
        if x == m.Min() {
                m.min = m.min[:len(m.min)-1]
        }
}

func (m *MinStack) Top() int {
        if len(m.data) == 0 {
                return 0
        }
        return m.data[len(m.data)-1]
}

func (m *MinStack) Min() int {
        return m.min[len(m.min)-1]
}
```

76.30 面试题 31. 栈的压入弹出序列 (2)

- 题目

```
输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。
假设压入栈的所有数字均不相等。
例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1}␣
↪是该压栈序列对应的一个弹出序列，
但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。
示例 1：输入：pushed = [1,2,3,4,5], popped = [4,5,3,2,1] 输出：true
解释：我们可以按以下顺序执行：
push(1), push(2), push(3), push(4), pop() -> 4,
push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
示例 2：输入：pushed = [1,2,3,4,5], popped = [4,3,5,1,2] 输出：false
解释：1 不能在 2 之前弹出。
提示：
    0 <= pushed.length == popped.length <= 1000
    0 <= pushed[i], popped[i] < 1000
    pushed 是 popped 的排列。
注意：本题与主站 946 题相同：https://leetcode.cn/problems/validate-stack-sequences/
```

- 解题思路

```
func validateStackSequences(pushed []int, popped []int) bool {
        stack := make([]int, 0)
        j := 0
        for i := 0; i < len(pushed); i++ {
                stack = append(stack, pushed[i])
                for len(stack) > 0 && stack[len(stack)-1] == popped[j] {
                        stack = stack[:len(stack)-1]
                        j++
                }
        }
        if len(stack) == 0 {
                return true
        }
        return false
}

#
func validateStackSequences(pushed []int, popped []int) bool {
        stack := make([]int, 0)
        res := false
        i := 0
```

(续下页)

(接上页)

```
        j := 0
        for j < len(popped) {
                for len(stack) == 0 || stack[len(stack)-1] != popped[j] {
                        if i == len(pushed) {
                                break
                        }
                        stack = append(stack, pushed[i])
                        i++
                }
                if stack[len(stack)-1] != popped[j] {
                        break
                }
                stack = stack[:len(stack)-1]
                j++
        }
        if len(stack) == 0 && j == len(popped) {
                res = true
        }
        return res
}
```

76.31 面试题 32-I. 从上到下打印二叉树 (2)

- 题目

```
从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。
例如:给定二叉树: [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回：[3,9,20,15,7]
提示：节点总数 <= 1000
```

- 解题思路

```
func levelOrder(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil {
                return res
        }
```

(续下页)

(接上页)

```
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                for i := 0; i < length; i++ {
                        node := list[i]
                        res = append(res, node.Val)
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
                                list = append(list, node.Right)
                        }
                }
                list = list[length:]
        }
        return res
}

#
func levelOrder(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil {
                return res
        }
        arr := levelArr(root)
        for i := 0; i < len(arr); i++ {
                res = append(res, arr[i]...)
        }
        return res
}

func levelArr(root *TreeNode) [][]int {
        temp := make([][]int, 0)
        dfs(root, &temp, 0)
        return temp
}

func dfs(root *TreeNode, temp *[][]int, level int) {
        if root == nil {
                return
        }
        if len(*temp)-1 < level {
```

(续下页)

(接上页)

```
                *temp = append(*temp, make([]int, 0))
        }
        (*temp)[level] = append((*temp)[level], root.Val)
        level = level + 1
        dfs(root.Left, temp, level)
        dfs(root.Right, temp, level)
}
```

76.32 面试题 32-II. 从上到下打印二叉树 II(2)

- 题目

```
从上到下按层打印二叉树，同一层的节点按从左到右的顺序打印，每一层打印到一行。
例如:
给定二叉树: [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回其层次遍历结果:
[
  [3],
  [9,20],
  [15,7]
]
提示: 节点总数 <= 1000
注意: 本题与主站 102 题相同:
https://leetcode.cn/problems/binary-tree-level-order-traversal/
```

- 解题思路

```
func levelOrder(root *TreeNode) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                temp := make([]int,0)
```

(续下页)

(接上页)

```
                for i := 0; i < length; i++ {
                        node := list[i]
                        temp = append(temp, node.Val)
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
                                list = append(list, node.Right)
                        }
                }
                list = list[length:]
                res = append(res, temp)
        }
        return res
}

#
func levelOrder(root *TreeNode) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        return levelArr(root)
}

func levelArr(root *TreeNode) [][]int {
        temp := make([][]int, 0)
        dfs(root, &temp, 0)
        return temp
}

func dfs(root *TreeNode, temp *[][]int, level int) {
        if root == nil {
                return
        }
        if len(*temp)-1 < level {
                *temp = append(*temp, make([]int, 0))
        }
        (*temp)[level] = append((*temp)[level], root.Val)
        level = level + 1
        dfs(root.Left, temp, level)
        dfs(root.Right, temp, level)
}
```

76.33 面试题 32-III. 从上到下打印二叉树 III(2)

- 题目

```
请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，
第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。
例如:给定二叉树: [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回其层次遍历结果:
[
  [3],
  [20,9],
  [15,7]
]
提示: 节点总数 <= 1000
```

- 解题思路

```
func levelOrder(root *TreeNode) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        level := 0
        for len(list) > 0 {
                length := len(list)
                temp := make([]int, 0)
                for i := 0; i < length; i++ {
                        node := list[i]
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
                                list = append(list, node.Right)
                        }
                }
                if level%2 == 0 {
                        for i := 0; i < length; i++ {
```

(续下页)

(接上页)

```
                                temp = append(temp, list[i].Val)
                        }
                } else {
                        for i := length - 1; i >= 0; i-- {
                                temp = append(temp, list[i].Val)
                        }
                }
                list = list[length:]
                res = append(res, temp)
                level++
        }
        return res
}

#
func levelOrder(root *TreeNode) [][]int {
        if root == nil {
                return nil
        }
        temp := make([][]int, 0)
        dfs(root, &temp, 0)
        return temp
}

func dfs(root *TreeNode, temp *[][]int, level int) {
        if root == nil {
                return
        }
        if len(*temp)-1 < level {
                *temp = append(*temp, make([]int, 0))
        }
        if level%2 == 0 {
                (*temp)[level] = append((*temp)[level], root.Val)
        } else {
                (*temp)[level] = append([]int{root.Val}, (*temp)[level]...)
        }
        level = level + 1
        dfs(root.Left, temp, level)
        dfs(root.Right, temp, level)
}
```

76.34 面试题 33. 二叉搜索树的后序遍历序列 (3)

- 题目

```
输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历结果。如果是则返回␣
↪true，否则返回 false。
假设输入的数组的任意两个数字都互不相同。
参考以下这颗二叉搜索树：
     5
    / \
   2   6
  / \
 1   3
示例 1：输入：[1,6,3,2,5]输出：false
示例 2：输入：[1,3,2,6,5]输出：true
提示：
    数组长度 <= 1000
```

- 解题思路

```
func verifyPostorder(postorder []int) bool {
        return dfs(postorder, 0, len(postorder)-1)
}

func dfs(postorder []int, start, end int) bool {
        if start >= end {
                return true
        }
        i := 0
        for i = start; i < end; i++ {
                if postorder[i] > postorder[end] {
                        break
                }
        }
        for j := i + 1; j < end; j++ {
                if postorder[j] < postorder[end] {
                        return false
                }
        }
        return dfs(postorder, start, i-1) && dfs(postorder, i, end-1)
}

#
func verifyPostorder(postorder []int) bool {
```

(续下页)

(接上页)

```
        if len(postorder) <= 2 {
                return true
        }
        last := len(postorder) - 1
        for last > 0 {
                i := 0
                for postorder[i] < postorder[last] {
                        i++
                }
                for postorder[i] > postorder[last] {
                        i++
                }
                if i != last {
                        return false
                }
                last--
        }
        return true
}

#
func verifyPostorder(postorder []int) bool {
        if len(postorder) <= 2 {
                return true
        }
        stack := make([]int, 0)
        rootValue := math.MaxInt32
        for i := len(postorder) - 1; i >= 0; i-- {
                if postorder[i] > rootValue {
                        // 左子树元素必须要小于递增栈根节点
                        return false
                }
                // 数组元素小于单调栈的元素了，表示往左子树走了，记录上个根节点
                for len(stack) > 0 && postorder[i] < stack[len(stack)-1] {
                        rootValue = stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                }
                stack = append(stack, postorder[i])
        }
        return true
}
```

76.35 面试题 34. 二叉树中和为某一值的路径 (2)

- 题目

```
输入一棵二叉树和一个整数，打印出二叉树中节点值的和为输入整数的所有路径。
从树的根节点开始往下一直到叶节点所经过的节点形成一条路径。
示例:给定如下二叉树，以及目标和 sum = 22，
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
返回:
[
   [5,4,11,2],
   [5,8,4,5]
]
提示：节点总数 <= 10000
注意：本题与主站 113 题相同：https://leetcode.cn/problems/path-sum-ii/
```

- 解题思路

```
var res [][]int

func pathSum(root *TreeNode, sum int) [][]int {
        if root == nil {
                return nil
        }
        res = make([][]int, 0)
        var arr []int
        dfs(root, sum, arr)
        return res
}

func dfs(root *TreeNode, sum int, arr []int) {
        if root == nil {
                return
        }
        arr = append(arr, root.Val)
        if root.Val == sum && root.Left == nil && root.Right == nil {
                temp := make([]int, len(arr))
                copy(temp, arr)
```

(续下页)

(接上页)

```
                res = append(res, temp)
        }
        dfs(root.Left, sum-root.Val, arr)
        dfs(root.Right, sum-root.Val, arr)
        arr = arr[:len(arr)-1]
}

#
func pathSum(root *TreeNode, sum int) [][]int {
        res := make([][]int, 0)
        if root == nil {
                return res
        }
        temp := make([]int, 0)
        stack := make([]*TreeNode, 0)
        visited := make(map[*TreeNode]bool)
        curSum := 0
        for root != nil || len(stack) > 0 {
                for root != nil {
                        temp = append(temp, root.Val)
                        curSum = curSum + root.Val
                        visited[root] = true
                        stack = append(stack, root)
                        root = root.Left
                }
                node := stack[len(stack)-1]
                if node.Right == nil || visited[node.Right] {
                        if node.Left == nil && node.Right == nil && curSum == sum {
                                tmp := make([]int, len(temp))
                                copy(tmp, temp)
                                res = append(res, tmp)
                        }
                        stack = stack[:len(stack)-1]
                        temp = temp[:len(temp)-1]
                        curSum = curSum - node.Val
                        root = nil
                } else {
                        root = node.Right
                }
        }
        return res
}
```

76.36 面试题 35. 复杂链表的复制 (3)

- 题目

```
请实现 copyRandomList 函数，复制一个复杂链表。
在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，
还有一个 random 指针指向链表中的任意节点或者 null。
示例 1： 输入：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
输出：[[7,null],[13,0],[11,4],[10,2],[1,0]]
示例 2：输入：head = [[1,1],[2,1]] 输出：[[1,1],[2,1]]
示例 3：输入：head = [[3,null],[3,0],[3,null]]
输出：[[3,null],[3,0],[3,null]]
示例 4：输入：head = [] 输出：[]
解释：给定的链表为空（空指针），因此返回 null。
提示：
    -10000 <= Node.val <= 10000
    Node.random 为空（null）或指向链表中的节点。
    节点数目不超过 1000 。
注意：本题与主站 138 题相同：
https://leetcode.cn/problems/copy-list-with-random-pointer/
```

- 解题思路

```
var m map[*Node]*Node

func copyRandomList(head *Node) *Node {
        m = make(map[*Node]*Node)
        return copyList(head)
}

func copyList(head *Node) *Node {
        if head == nil {
                return head
        }
        if node, ok := m[head]; ok {
                return node
        }
        temp := &Node{
                Val:    head.Val,
                Next:   nil,
                Random: nil,
        }
        m[head] = temp
        temp.Next = copyList(head.Next)
```

(续下页)

(接上页)

```
        temp.Random = copyList(head.Random)
        return temp
}

#
func copyRandomList(head *Node) *Node {
        if head == nil {
                return nil
        }
        res := new(Node)
        m := make(map[*Node]*Node)
        temp := head
        p := res
        for temp != nil {
                node := &Node{
                        Val:    temp.Val,
                        Next:   nil,
                        Random: nil,
                }
                m[temp] = node
                p.Next = node
                p = p.Next
                temp = temp.Next

        }
        temp = head
        p = res.Next
        for temp != nil {
                p.Random = m[temp.Random]
                p = p.Next
                temp = temp.Next
        }
        return res.Next
}

# 3
func copyRandomList(head *Node) *Node {
        if head == nil {
                return nil
        }
        res := copyNext(head)
        res = copyRandom(res)
        res = cutEven(res)
```

(续下页)

(接上页)

```
        return res
}

// 原1-复制1-原2-复制2
func copyNext(head *Node) *Node {
        p := head
        for p != nil {
                node := new(Node)
                node.Val = p.Val
                node.Next = p.Next
                p.Next = node
                p = node.Next
        }
        return head
}

func copyRandom(head *Node) *Node {
        p := head
        for p != nil {
                if p.Random != nil {
                        p.Next.Random = p.Random.Next
                }
                p = p.Next.Next
        }
        return head
}

func cutEven(head *Node) *Node {
        oldNode := head
        newNode := head.Next
        cur := newNode
        for oldNode != nil {
                oldNode.Next = oldNode.Next.Next
                if newNode.Next != nil{
                        newNode.Next = newNode.Next.Next
                }
                oldNode = oldNode.Next
                newNode = newNode.Next
        }
        return cur
}
```

76.37 面试题 38. 字符串的排列 (2)

- 题目

```
输入一个字符串，打印出该字符串中字符的所有排列。
你可以以任意顺序返回这个字符串数组，但里面不能有重复元素。
示例:输入：s = "abc" 输出：["abc","acb","bac","bca","cab","cba"]
限制：1 <= s 的长度 <= 8
```

- 解题思路

```
var m map[string]bool

func permutation(s string) []string {
        m = make(map[string]bool)
        dfs(s, "")
        res := make([]string, 0)
        for key := range m {
                res = append(res, key)
        }
        return res
}

func dfs(s string, str string) {
        if len(s) == 0 {
                m[str] = true
        }
        for i := 0; i < len(s); i++ {
                arr := []byte(s)
                temp := arr[i]
                if len(arr)-1 == i {
                        arr = arr[:len(arr)-1]
                } else {
                        arr = append(arr[:i], arr[i+1:]...)
                }
                dfs(string(arr), str+string(temp))
        }
}

#
var res []string

func permutation(s string) []string {
        res = make([]string, 0)
```

(续下页)

(接上页)

```
        dfs([]byte(s), 0)
        return res
}

func dfs(arr []byte, index int) {
        if len(arr)-1 == index {
                res = append(res, string(arr))
                return
        }
        m := make(map[byte]bool)
        for i := index; i < len(arr); i++ {
                if _, ok := m[arr[i]]; ok {
                        continue
                }
                m[arr[i]] = true
                arr[i], arr[index] = arr[index], arr[i]
                dfs(arr, index+1)
                arr[i], arr[index] = arr[index], arr[i]
        }
}
```

76.38 面试题 39. 数组中出现次数超过一半的数字 (5)

- 题目

```
数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。
你可以假设数组是非空的，并且给定的数组总是存在多数元素。
示例 1:输入: [1, 2, 3, 2, 2, 2, 5, 4, 2] 输出: 2
限制: 1 <= 数组长度 <= 50000
注意: 本题与主站 169 题相同: https://leetcode.cn/problems/majority-element/
```

- 解题思路

```
func majorityElement(nums []int) int {
        sort.Ints(nums)
        return nums[len(nums)/2]
}

# 2
func majorityElement(nums []int) int {
        m := make(map[int]int)
        result := 0
```

(续下页)

(接上页)

```
        for _, v := range nums{
                if _,ok := m[v];ok{
                        m[v]++
                }else {
                        m[v]=1
                }
                if m[v] > (len(nums)/2){
                        result = v
                }
        }
        return result
}

# 3
func majorityElement(nums []int) int {
        result, count := 0, 0
        for i := 0; i < len(nums); i++ {
                if count == 0 {
                        result = nums[i]
                        count++
                } else if result == nums[i] {
                        count++
                } else {
                        count--
                }
        }
        return result
}

# 4
func majorityElement(nums []int) int {
        if len(nums) == 1 {
                return nums[0]
        }
        result := int32(0)
        // 64位有坑
        mask := int32(1)
        for i := 0; i < 32; i++ {
                count := 0
                for j := 0; j < len(nums); j++ {
                        if mask&int32(nums[j]) == mask {
                                count++
                        }
```

(续下页)

(接上页)

```
                }
                if count > len(nums)/2 {
                        result = result | mask
                }
                mask = mask << 1
        }
        return int(result)
}

# 5
func majority(nums []int, start, end int) int {
        if start == end {
                return nums[start]
        }

        mid := (start + end) / 2

        left := majority(nums, start, mid)
        right := majority(nums, mid+1, end)
        if left == right {
                return left
        }

        leftCount := count(nums, left, start, end)
        rightCount := count(nums, right, start, end)
        if leftCount > rightCount {
                return left
        }
        return right
}
```

76.39 面试题 40. 最小的 k 个数 (4)

- 题目

```
输入整数数组 arr ，找出其中最小的 k 个数。
例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。
示例 1: 输入: arr = [3,2,1], k = 2 输出: [1,2] 或者 [2,1]
示例 2: 输入: arr = [0,1,2,1], k = 1 输出: [0]
限制:
    0 <= k <= arr.length <= 10000
    0 <= arr[i] <= 10000
```

- 解题思路

```
func getLeastNumbers(arr []int, k int) []int {
        if len(arr) == 0 || k == 0 {
                return nil
        }
        sort.Ints(arr)
        return arr[:k]
}

#
type IntHeap []int

func (i IntHeap) Len() int {
        return len(i)
}

func (i IntHeap) Less(x, y int) bool {
        return i[x] > i[y]
}

func (i IntHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *IntHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *IntHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

func getLeastNumbers(arr []int, k int) []int {
        if len(arr) == 0 || k == 0 {
                return nil
        }
        intHeap := make(IntHeap, 0, k)
        heap.Init(&intHeap)

        for i := 0; i < len(arr); i++ {
                if len(intHeap) < k {
                        heap.Push(&intHeap, arr[i])
                } else if len(intHeap) == k {
```

(续下页)

(接上页)

```
                        if arr[i] < intHeap[0] {
                                heap.Pop(&intHeap)
                                heap.Push(&intHeap, arr[i])
                        }
                }
        }
        return intHeap
}

#
func getLeastNumbers(arr []int, k int) []int {
        if len(arr) == 0 || k == 0 {
                return nil
        }
        a := make([]int, 10001)
        for _, v := range arr {
                a[v]++
        }
        res := make([]int, 0)
        for key, value := range a {
                if value > 0 {
                        for i := 0; i < value; i++ {
                                res = append(res, key)
                                k--
                                if k <= 0 {
                                        return res
                                }
                        }
                }
        }
        return res
}

#
func getLeastNumbers(arr []int, k int) []int {
        if len(arr) == 0 || k == 0 {
                return nil
        }
        left := 0
        right := len(arr) - 1
        for {
                index := partition(arr, left, right)
                if index+1 == k {
```

(续下页)

(接上页)

```
                        break
                } else if index+1 < k {
                        left = index + 1
                } else {
                        right = index - 1
                }
        }
        return arr[:k]
}

func partition(arr []int, left, right int) int {
        if left == right {
                return left
        }
        value := arr[left]
        i := left
        j := right
        for {
                for arr[j] >= value && i < j {
                        j--
                }
                for arr[i] <= value && i < j {
                        i++
                }
                if i >= j {
                        break
                }
                arr[i], arr[j] = arr[j], arr[i]
        }
        arr[left] = arr[i]
        arr[i] = value
        return i
}
```

76.40 面试题 41. 数据流中的中位数 (1)

- 题目

```
如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数
如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。
例如，
[2,3,4] 的中位数是 3
```

(续下页)

(接上页)

```
[2,3] 的中位数是 (2 + 3) / 2 = 2.5
设计一个支持以下两种操作的数据结构：
    void addNum(int num) - 从数据流中添加一个整数到数据结构中。
    double findMedian() - 返回目前所有元素的中位数。

示例 1：输入：
["MedianFinder","addNum","addNum","findMedian","addNum","findMedian"]
[[],[1],[2],[],[3],[]]
输出：[null,null,null,1.50000,null,2.00000]

示例 2：  输入：
["MedianFinder","addNum","findMedian","addNum","findMedian"]
[[],[2],[],[3],[]]
输出：[null,null,2.00000,null,2.50000]

限制：最多会对 addNum、findMedia进行 50000 次调用。
注意：本题与主站 295 题相同：
https://leetcode.cn/problems/find-median-from-data-stream/
```

- 解题思路

```
type MinHeap []int

func (i MinHeap) Len() int {
        return len(i)
}

func (i MinHeap) Less(x, y int) bool {
        return i[x] < i[y]
}

func (i MinHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *MinHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *MinHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}
```

(续下页)

(接上页)

```
type MaxHeap []int

func (i MaxHeap) Len() int {
        return len(i)
}

func (i MaxHeap) Less(x, y int) bool {
        return i[x] > i[y]
}

func (i MaxHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *MaxHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *MaxHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

type MedianFinder struct {
        minArr *MinHeap
        maxArr *MaxHeap
}

func Constructor() MedianFinder {
        res := new(MedianFinder)
        res.minArr = new(MinHeap)
        res.maxArr = new(MaxHeap)
        heap.Init(res.minArr)
        heap.Init(res.maxArr)
        return *res
}

func (this *MedianFinder) AddNum(num int) {
        if this.maxArr.Len() == this.minArr.Len() {
                heap.Push(this.minArr, num)
                heap.Push(this.maxArr, heap.Pop(this.minArr))
        } else {
                heap.Push(this.maxArr, num)
```

(续下页)

(接上页)

```
                heap.Push(this.minArr, heap.Pop(this.maxArr))
        }
}

func (this *MedianFinder) FindMedian() float64 {
        if this.minArr.Len() == this.maxArr.Len() {
                return (float64((*this.maxArr)[0]) + float64((*this.minArr)[0])) / 2
        } else {
                return float64((*this.maxArr)[0])
        }
}
```

76.41 面试题 42. 连续子数组的最大和 (4)

- 题目

```
输入一个整型数组，数组里有正数也有负数。数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的
要求时间复杂度为O(n)。
示例1:输入: nums = [-2,1,-3,4,-1,2,1,-5,4]输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。
提示：
    1 <= arr.length <= 10^5
    -100 <= arr[i] <= 100
注意：本题与主站 53 题相同：https://leetcode.cn/problems/maximum-subarray/
```

- 解题思路

```
func maxSubArray(nums []int) int {
        result := nums[0]
        sum := 0
        for i := 0; i < len(nums); i++ {
                if sum > 0 {
                        sum += nums[i]
                } else {
                        sum = nums[i]
                }
                if sum > result {
                        result = sum
                }
        }
        return result
}
```

(续下页)

(接上页)

```
#
func maxSubArray(nums []int) int {
        dp := make([]int, len(nums))
        dp[0] = nums[0]
        result := nums[0]

        for i := 1; i < len(nums); i++ {
                if dp[i-1]+nums[i] > nums[i] {
                        dp[i] = dp[i-1] + nums[i]
                } else {
                        dp[i] = nums[i]
                }

                if dp[i] > result {
                        result = dp[i]
                }
        }
        return result
}

#
func maxSubArray(nums []int) int {
        dp := nums[0]
        result := dp
        for i := 1; i < len(nums); i++ {
                if dp+nums[i] > nums[i] {
                        dp = dp + nums[i]
                } else {
                        dp = nums[i]
                }

                if dp > result {
                        result = dp
                }
        }
        return result
}

#
func maxSubArray(nums []int) int {
        result := maxSubArr(nums, 0, len(nums)-1)
        return result
```

(续下页)

(接上页)

```
}

func maxSubArr(nums []int, left, right int) int {
        if left == right {
                return nums[left]
        }

        mid := (left + right) / 2
        leftSum := maxSubArr(nums, left, mid)        // 最大子序在左边
        rightSum := maxSubArr(nums, mid+1, right)    // 最大子序在右边
        midSum := findMaxArr(nums, left, mid, right) // 跨中心
        result := max(leftSum, rightSum)
        result = max(result, midSum)
        return result
}

func findMaxArr(nums []int, left, mid, right int) int {
        leftSum := math.MinInt32
        sum := 0
        // 从右到左
        for i := mid; i >= left; i-- {
                sum += nums[i]
                leftSum = max(leftSum, sum)
        }

        rightSum := math.MinInt32
        sum = 0
        // 从左到右
        for i := mid + 1; i <= right; i++ {
                sum += nums[i]
                rightSum = max(rightSum, sum)
        }
        return leftSum + rightSum
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

76.42 面试题 43.1 ~ n 整数中 1 出现的次数 (3)

- 题目

```
输入一个整数 n ，求1～n这n个整数的十进制表示中1出现的次数。
例如，输入12，1～12这些整数中包含1 的数字有1、10、11和12，1一共出现了5次。
示例 1: 输入: n = 12 出: 5
示例 2: 输入: n = 13 出: 6
限制： 1 <= n < 2^31
注意: 本题与主站 233 题相同:
https://leetcode.cn/problems/number-of-digit-one/
```

- 解题思路

```
func countDigitOne(n int) int {
        res := 0
        digit := 1
        high := n / 10
        cur := n % 10
        low := 0
        for high != 0 || cur != 0 {
                if cur == 0 {
                        res = res + high*digit
                } else if cur == 1 {
                        res = res + high*digit + low + 1
                } else {
                        res = res + (high+1)*digit
                }
                low = low + cur*digit
                cur = high % 10
                high = high / 10
                digit = digit * 10
        }
        return res
}

#
func countDigitOne(n int) int {
        if n <= 0 {
                return 0
        }
        str := strconv.Itoa(n)
        return dfs(str)
}
```

(续下页)

(接上页)

```
func dfs(str string) int {
        if str == "" {
                return 0
        }
        first := int(str[0] - '0')
        if len(str) == 1 && first == 0 {
                return 0
        }
        if len(str) == 1 && first >= 1 {
                return 1
        }
        count := 0
        if first > 1 {
                count = int(math.Pow(float64(10), float64(len(str)-1)))
        } else if first == 1 {
                count, _ = strconv.Atoi(str[1:])
                count = count + 1
        }
        other := first * (len(str) - 1) * int(math.Pow(float64(10), float64(len(str)-
↪2)))
        numLeft := dfs(str[1:])
        return count + numLeft + other
}

#
func countDigitOne(n int) int {
        if n <= 0 {
                return 0
        }
        res := 0
        for i := 1; i <= n; i = i * 10 {
                left := n / i
                right := n % i
                res = res + (left+8)/10*i
                if left%10 == 1 {
                        res = res + right + 1
                }
        }
        return res
}
```

76.43 面试题 44. 数字序列中某一位的数字 (2)

- 题目

```
数字以0123456789101112131415…的格式序列化到一个字符序列中。
在这个序列中，第5位（从下标0开始计数）是5，第13位是1，第19位是4，等等。
请写一个函数，求任意第n位对应的数字。
示例 1：输入：n = 3 输出：3
示例 2：输入：n = 11 输出：0
限制： 0 <= n < 2^31
注意：本题与主站 400 题相同：https://leetcode.cn/problems/nth-digit/
```

- 解题思路

```
func findNthDigit(n int) int {
        if n < 0 {
                return -1
        }
        digits := 1
        for {
                numbers := countOfIntegers(digits)
                if n < numbers*digits {
                        return digitAtIndex(n, digits)
                }
                n = n - numbers*digits
                digits++
        }
}

func countOfIntegers(n int) int {
        if n == 1 {
                return 10
        }
        count := math.Pow(float64(10), float64(n-1))
        return 9 * int(count)
}

func digitAtIndex(n, digits int) int {
        number := beginNumber(digits) + n/digits
        indexFromRight := digits - n%digits
        for i := 1; i < indexFromRight; i++ {
                number = number / 10
        }
        return number % 10
```

(续下页)

(接上页)

```
}

func beginNumber(digits int) int {
        if digits == 1 {
                return 0
        }
        return int(math.Pow(float64(10), float64(digits-1)))
}

#
/*
1-9         9*1 1位
10-99         90*2 2位
100-999 900*3 3位
*/
func findNthDigit(n int) int {
        if n < 10 {
                return n
        }
        digits := 1
        count := 9
        number := 1
        for n-digits*count > 0 {
                n = n - digits*count
                digits++
                count = count * 10
                number = number * 10
        }
        number = number + (n-1)/digits
        index := (n - 1) % digits
        str := strconv.Itoa(number)
        return int(str[index] - '0')
}
```

76.44 面试题 45. 把数组排成最小的数 (3)

- 题目

```
输入一个非负整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。
示例 1:输入: [10,2] 输出: "102"
示例 2:输入: [3,30,34,5,9] 输出: "3033459"
提示:0 < nums.length <= 100
```

(续下页)

(接上页)

```
说明：
    输出结果可能非常大，所以你需要返回一个字符串而不是整数
    拼接起来的数字可能会有前导 0，最后结果不需要去掉前导 0
```

- 解题思路

```
func minNumber(nums []int) string {
        sort.Slice(nums, func(i, j int) bool {
                if fmt.Sprintf("%d%d", nums[i], nums[j]) <
                        fmt.Sprintf("%d%d", nums[j], nums[i]) {
                        return true
                }
                return false
        })
        str := ""
        for i := 0; i < len(nums); i++ {
                str = str + fmt.Sprintf("%d", nums[i])
        }
        return str
}

#
var arr []string

func minNumber(nums []int) string {
        arr = make([]string, 0)
        for i := 0; i < len(nums); i++ {
                arr = append(arr, strconv.Itoa(nums[i]))
        }
        quickSort(0, len(arr)-1)
        str := ""
        for i := 0; i < len(arr); i++ {
                str = str + arr[i]
        }
        return str
}

func quickSort(start, end int) {
        if start >= end {
                return
        }
        temp := arr[start]
        i := start
        j := end
```

(续下页)

(接上页)

```
        for i < j {
                for i < j && arr[j]+temp >= temp+arr[j] {
                        j--
                }
                for i < j && arr[i]+temp <= temp+arr[i] {
                        i++
                }
                arr[i], arr[j] = arr[j], arr[i]
        }
        arr[start], arr[i] = arr[i], temp
        quickSort(start, i-1)
        quickSort(i+1, end)
}

#
type Arr []string

func (a Arr) Len() int {
        return len(a)
}

func (a Arr) Less(i, j int) bool {
        if a[i]+a[j] < a[j]+a[i] {
                return true
        }
        return false
}

func (a Arr) Swap(i, j int) {
        a[i], a[j] = a[j], a[i]
}

func minNumber(nums []int) string {
        var arr Arr
        for i := 0; i < len(nums); i++ {
                arr = append(arr, strconv.Itoa(nums[i]))
        }
        sort.Sort(arr)
        str := ""
        for i := 0; i < len(arr); i++ {
                str = str + arr[i]
        }
        return str
}
```

76.45 面试题 46. 把数字翻译成字符串 (4)

- 题目

```
给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a” ，1 翻译成␣
↪“b”，……，
11 翻译成 “l”，……，25 翻译成 “z”。
一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。
示例 1:输入: 12258 输出: 5
解释: 12258有5种不同的翻译，分别是"bccfi", "bwfi", "bczi", "mcfi"和"mzi"
提示： 0 <= num < 231
```

- 解题思路

```
func translateNum(num int) int {
        if num < 0 {
                return 0
        }
        return getTranslateNum(strconv.Itoa(num))
}

func getTranslateNum(str string) int {
        length := len(str)
        arr := make([]int, length)
        count := 0
        for i := length - 1; i >= 0; i-- {
                count = 0
                if i < length-1 {
                        count = arr[i+1]
                } else {
                        count = 1
                }
                if i < length-1 {
                        digit1 := str[i] - '0'
                        digit2 := str[i+1] - '0'
                        value := digit1*10 + digit2
                        if value >= 10 && value <= 25 {
                                if i < length-2 {
                                        count += arr[i+2]
                                } else {
                                        count += 1
                                }
                        }
                }
```

(续下页)

(接上页)

```
                arr[i] = count
        }
        return arr[0]
}

#
func translateNum(num int) int {
        if num < 10 {
                return 1
        }
        var res int
        if num%100 <= 25 && num%100 > 9 {
                res = res + translateNum(num/100)
                res = res + translateNum(num/10)
        } else {
                res = res + translateNum(num/10)
        }
        return res
}

#
// f(i)=f(i-2)+f(i-1)
// f(i)=f(i-1)
func translateNum(num int) int {
        if num < 0 {
                return 0
        }
        arr := make([]int, 1)
        arr[0] = 1
        i := 0
        prev := -1
        for num > 0 {
                i++
                arr = append(arr, 0)
                arr[i] = arr[i-1]
                digit1 := num % 10
                num = num / 10
                if digit1 != 0 && prev >= 0 && digit1*10+prev <= 25 {
                        arr[i] = arr[i] + arr[i-2]
                }
                prev = digit1
        }
        return arr[i]
```

(续下页)

(接上页)

```
}

#
// f(i)=f(i-2)+f(i-1)
// f(i)=f(i-1)
func translateNum(num int) int {
        if num < 0 {
                return 0
        }
        str := strconv.Itoa(num)
        arr := make([]int, len(str))
        for i := 0; i < len(str); i++ {
                arr[i] = int(str[i] - '0')
        }
        dp := make([]int, len(str)+1)
        dp[0] = 1
        dp[1] = 1
        for i := 2; i < len(str)+1; i++ {
                if arr[i-2] != 0 && (arr[i-2]*10+arr[i-1] <= 25) {
                        dp[i] = dp[i-1] + dp[i-2]
                } else {
                        dp[i] = dp[i-1]
                }
        }
        return dp[len(str)]
}
```

76.46 面试题 47. 礼物的最大价值 (2)

- 题目

```
在一个 m*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。
你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。
给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？
示例 1:输入:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
输出: 12 解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物
提示:
```

(续下页)

(接上页)

```
    0 < grid.length <= 200
    0 < grid[0].length <= 200
```

- 解题思路

```
func maxValue(grid [][]int) int {
        if len(grid) == 0 || len(grid[0]) == 0 {
                return 0
        }
        dp := make([][]int, len(grid))
        for i := 0; i < len(grid); i++ {
                dp[i] = make([]int, len(grid[0]))
        }
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[0]); j++ {
                        left := 0
                        up := 0
                        if i > 0 {
                                left = dp[i-1][j]
                        }
                        if j > 0 {
                                up = dp[i][j-1]
                        }
                        // dp[i][j]=grid[i-1][j-1]+max(dp[i][j-1],dp[i-1][j])
                        dp[i][j] = max(left, up) + grid[i][j]
                }
        }
        return dp[len(grid)-1][len(grid[0])-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func maxValue(grid [][]int) int {
        if len(grid) == 0 || len(grid[0]) == 0 {
                return 0
        }
        dp := make([]int, len(grid))
        for i := 0; i < len(grid); i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < len(grid[0]); j++ {
                        left := 0
                        up := 0
                        if i > 0 {
                                left = dp[j]
                        }
                        if j > 0 {
                                up = dp[j-1]
                        }
                        // dp[j]=grid[i-1][j-1]+max(dp[j-1],dp[j])
                        dp[j] = max(left, up) + grid[i][j]
                }
        }
        return dp[len(grid[0])-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

76.47 面试题 48. 最长不含重复字符的子字符串 (4)

- 题目

```
请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。
示例 1:输入: "abcabcbb" 输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
示例 2:输入: "bbbbb" 输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
示例 3:输入: "pwwkew" 输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
提示：s.length <= 40000
注意：本题与主站 3 题相同：
https://leetcode.cn/problems/longest-substring-without-repeating-characters/
```

- 解题思路

```
func lengthOfLongestSubstring(s string) int {
        if len(s) < 1 {
                return 0
        }
        m := make(map[int32]int)
        arr := make([]int32, 0)
        res := 0
        for _, value := range s {
                if v, ok := m[value]; ok && v > 0 {
                        for len(arr) > 0 && arr[0] != value {
                                m[arr[0]]--
                                arr = arr[1:]
                        }
                        m[arr[0]]--
                        arr = arr[1:]
                }
                m[value]++
                arr = append(arr, value)
                if len(arr) > res {
                        res = len(arr)
                }
        }
        return res
}

#
func lengthOfLongestSubstring(s string) int {
        if len(s) < 1 {
                return 0
        }
        left := 0
        right := 0
        res := 0
        for left <= right {
                m := make(map[byte]int)
                for i := left; i < right; i++ {
                        m[s[i]]++
                }
                for right < len(s) {
                        if value, ok := m[s[right]]; ok && value > 0 {
                                if right-left > res {
                                        res = right - left
                                }
                                left++
```

(续下页)

(接上页)

```
                                break
                        } else {
                                m[s[right]]++
                                right++
                        }
                }
                if right-left > res {
                        res = right - left
                }
                if right >= len(s)-1 {
                        break
                }
        }
        return res
}

#
// dp[n]=dp[n-1]+1
// dp[n]=n-lastIndex
func lengthOfLongestSubstring(s string) int {
        if len(s) < 1 {
                return 0
        }
        dp := make([]int, len(s))
        dp[0] = 1
        res := 1
        m := make(map[byte]int)
        m[s[0]] = 0
        for i := 1; i < len(s); i++ {
                index := -1
                if value, ok := m[s[i]]; ok {
                        index = value
                }
                if i-index > dp[i-1] {
                        dp[i] = dp[i-1] + 1
                } else {
                        dp[i] = i - index
                }
                m[s[i]] = i
                if dp[i] > res {
                        res = dp[i]
                }
        }
```

(续下页)

(接上页)

```
        return res
}

#
func lengthOfLongestSubstring(s string) int {
        if len(s) < 1 {
                return 0
        }
        res := 1
        arr := make(map[byte]int)
        curLength := 0
        for i := 0; i < len(s); i++ {
                if preIndex, ok := arr[s[i]]; !ok || i-preIndex > curLength {
                        curLength++
                } else {
                        if curLength > res {
                                res = curLength
                        }
                        curLength = i - preIndex
                }
                arr[s[i]] = i
        }
        if curLength > res {
                res = curLength
        }
        return res
}
```

76.48 面试题 49. 丑数 (1)

- 题目

```
我们把只包含因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n␣
↪个丑数。
示例:输入: n = 10 输出: 12
解释: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 是前 10 个丑数。
说明: 1 是丑数。n 不超过1690。
注意：本题与主站 264 题相同：https://leetcode.cn/problems/ugly-number-ii/
```

- 解题思路

```
func nthUglyNumber(n int) int {
        dp := make([]int, n)
        dp[0] = 1
    // 丑数*2或3或5之后还是丑数
        idx2, idx3, idx5 := 0, 0, 0
        for i := 1; i < n; i++ {
                dp[i] = min(dp[idx2]*2, min(dp[idx3]*3, dp[idx5]*5))
                if dp[i] == dp[idx2]*2 {
                        idx2++
                }
                if dp[i] == dp[idx3]*3 {
                        idx3++
                }
                if dp[i] == dp[idx5]*5 {
                        idx5++
                }
        }
        return dp[n-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

76.49 面试题 50. 第一个只出现一次的字符 (3)

- 题目

```
在字符串 s 中找出第一个只出现一次的字符。如果没有，返回一个单空格。 s 只包含小写字母。
示例:
s = "abaccdeff" 返回 "b"
s = "" 返回 " "
限制: 0 <= s 的长度 <= 50000
```

- 解题思路

```
func firstUniqChar(s string) byte {
        res := byte(' ')
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
```

(续下页)

(接上页)

```
                m[s[i]]++
        }
        for i := 0; i < len(s); i++ {
                if m[s[i]] == 1 {
                        return s[i]
                }
        }
        return res
}

#
func firstUniqChar(s string) byte {
        res := byte(' ')
        m := [26]int{}
        for i := 0; i < len(s); i++ {
                m[s[i]-'a']++
        }
        for i := 0; i < len(s); i++ {
                if m[s[i]-'a'] == 1 {
                        return s[i]
                }
        }
        return res
}

#
func firstUniqChar(s string) byte {
        res := byte(' ')
        for i := 0; i < len(s); i++ {
                flag := true
                for j := 0; j < len(s); j++ {
                        if s[i] == s[j] && i != j {
                                flag = false
                                break
                        }
                }
                if flag {
                        return s[i]
                }
        }
        return res
}
```

76.50 面试题 51. 数组中的逆序对 (1)

- 题目

```
在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。
输入一个数组，求出这个数组中的逆序对的总数。
示例 1:输入: [7,5,6,4] 输出: 5
限制: 0 <= 数组长度 <= 50000
```

- 解题思路

```
var res int

func reversePairs(nums []int) int {
        res = 0
        if len(nums) <= 1 {
                return res
        }
        merge(nums, 0, len(nums)-1)
        return res
}

func merge(nums []int, left, right int) {
        if left >= right {
                return
        }
        mid := (left + right) / 2
        i, j := left, mid+1
        merge(nums, left, mid)
        merge(nums, mid+1, right)

        temp := make([]int, 0)
        for i <= mid && j <= right {
                if nums[i] <= nums[j] {
                        temp = append(temp, nums[i])
                        i++
                } else {
                        res = res + mid - i + 1
                        temp = append(temp, nums[j])
                        j++
                }
        }
        temp = append(temp, nums[i:mid+1]...)
        temp = append(temp, nums[j:right+1]...)
```

(续下页)

(接上页)

```
        for key, value := range temp {
                nums[left+key] = value
        }
}
```

76.51 面试题 52. 两个链表的第一个公共节点 (4)

- 题目

```
输入两个链表，找出它们的第一个公共节点。
如下面的两个链表：
在节点 c1 开始相交。
示例 1：
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB␣
↪= 3
输出：Reference of the node with value = 8
输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
示例 2：
输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
输出：Reference of the node with value = 2
输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为␣
↪0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A␣
↪中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。
示例 3：
输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
输出：null
输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,
↪5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB␣
↪可以是任意值。
解释：这两个链表不相交，因此返回 null。
注意：
    如果两个链表没有交点，返回 null.
    在返回结果后，两个链表仍须保持原有的结构。
    可假定整个链表结构中没有循环。
    程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。
    本题与主站 160 题相同：https://leetcode.cn/problems/intersection-of-two-linked-
↪lists/
```

- 解题思路

```
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        ALength := 0
        A := headA
        for A != nil {
                ALength++
                A = A.Next
        }
        BLength := 0
        B := headB
        for B != nil {
                BLength++
                B = B.Next
        }

        pA := headA
        pB := headB
        if ALength > BLength {
                n := ALength - BLength
                for n > 0 {
                        pA = pA.Next
                        n--
                }
        } else {
                n := BLength - ALength
                for n > 0 {
                        pB = pB.Next
                        n--
                }
        }

        for pA != pB {
                pA = pA.Next
                pB = pB.Next
        }
        return pA
}

#
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        A, B := headA, headB
        for A != B {
                if A != nil {
                        A = A.Next
                } else {
```

(续下页)

(接上页)

```
                        A = headB
                }
                if B != nil {
                        B = B.Next
                } else {
                        B = headA
                }
        }
        return A
}

#
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        A, B := headA, headB
        for A != nil {
                for B != nil {
                        if A == B {
                                return A
                        }
                        B = B.Next
                }
                A = A.Next
                B = headB
        }
        return nil
}

# 4
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for headA != nil {
                m[headA] = true
                headA = headA.Next
        }

        for headB != nil {
                if _, ok := m[headB]; ok {
                        return headB
                }
                headB = headB.Next
        }
        return nil
}
```

76.52 面试题 53-I. 在排序数组中查找数字 I(5)

- 题目

```
统计一个数字在排序数组中出现的次数。
示例 1:输入: nums = [5,7,7,8,8,10], target = 8 输出: 2
示例 2:输入: nums = [5,7,7,8,8,10], target = 6 输出: 0
限制: 0 <= 数组长度 <= 50000
注意: 本题与主站 34 题相同 (仅返回值不同):
https://leetcode.cn/problems/find-first-and-last-position-of-element-in-sorted-array/
```

- 解题思路

```
func search(nums []int, target int) int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        m[target]++
                }
        }
        return m[target]
}

#
func search(nums []int, target int) int {
        count := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == target {
                        count++
                }
        }
        return count
}

#
func search(nums []int, target int) int {
        i := 0
        j := len(nums) - 1
        for i < len(nums) && nums[i] != target {
                i++
        }
        for j >= 0 && nums[j] != target {
                j--
        }
```

(续下页)

(接上页)

```
        if i > j {
                return 0
        }
        return j - i + 1
}

#
func search(nums []int, target int) int {
        left := 0
        right := len(nums) - 1
        count := 0
        for left <= right{
                mid := left+(right-left)/2
                if nums[mid] == target{
                        count++
                        for i := mid+1; i < len(nums); i++{
                                if nums[i] == target{
                                        count++
                                }else {
                                        break
                                }
                        }
                        for i := mid-1; i >= 0; i--{
                                if nums[i] == target{
                                        count++
                                }else {
                                        break
                                }
                        }
                        return count
                }
                if nums[mid] > target{
                        right = mid-1
                }else {
                        left = mid+1
                }
        }
        return count
}

#
func search(nums []int, target int) int {
        count := 0
```

(续下页)

(接上页)

```
        if len(nums) > 0 {
                first := getFirstK(nums, target, 0, len(nums)-1)
                last := getLastK(nums, target, 0, len(nums)-1)
                if first > -1 && last > -1 {
                        count = last - first + 1
                }
        }
        return count
}

func getFirstK(nums []int, target int, start, end int) int {
        if start > end {
                return -1
        }
        mid := start + (end-start)/2
        if nums[mid] == target {
                if (mid > 0 && nums[mid-1] != target) || mid == 0 {
                        return mid
                } else {
                        end = mid - 1
                }
        } else if nums[mid] > target {
                end = mid - 1
        } else {
                start = mid + 1
        }
        return getFirstK(nums, target, start, end)
}

func getLastK(nums []int, target int, start, end int) int {
        if start > end {
                return -1
        }
        mid := start + (end-start)/2
        if nums[mid] == target {
                if (mid < len(nums)-1 && nums[mid+1] != target) || mid == len(nums)-1
↪{
                        return mid
                } else {
                        start = mid + 1
                }
        } else if nums[mid] < target {
                start = mid + 1
```

(续下页)

(接上页)

```
        } else {
                end = mid - 1
        }
        return getLastK(nums, target, start, end)
}
```

76.53 面试题 53-II.0 ~ n-1 中缺失的数字 (6)

- 题目

```
一个长度为n-1的递增排序数组中的所有数字都是唯一的，并且每个数字都在范围0~n-1之内。
在范围0~n-1内的n个数字中有且只有一个数字不在该数组中，请找出这个数字。
示例 1:输入: [0,1,3] 输出: 2
示例 2:输入: [0,1,2,3,4,5,6,7,9] 输出: 8
限制: 1 <= 数组长度 <= 10000
```

- 解题思路

```
func missingNumber(nums []int) int {
        n := len(nums)
        sum := n * (n + 1) / 2
        for i := 0; i < n; i++ {
                sum = sum - nums[i]
        }
        return sum
}


# 2
func missingNumber(nums []int) int {
        for i := 0; i < len(nums); i++ {
                if nums[i] != i {
                        return i
                }
        }
        return len(nums)
}

# 3
func missingNumber(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                res = res ^ (i+1) ^ nums[i]
        }
        return res
}

# 4
func missingNumber(nums []int) int {
        m := make(map[int]bool)
        for i := range nums{
                m[nums[i]] = true
        }
        for i := 0; i <= len(nums); i++{
                if m[i] == false{
                        return i
                }
        }
        return 0
}

# 5
func missingNumber(nums []int) int {
        left := 0
        right := len(nums) - 1
        for left <= right {
                mid := left + (right-left)/2
                if nums[mid] != mid {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return left
}

#
func missingNumber(nums []int) int {
        return sort.Search(len(nums), func(i int) bool {
                return nums[i] != i
        })
}
```

76.54 面试题 54. 二叉搜索树的第 k 大节点 (3)

- 题目

```
给定一棵二叉搜索树，请找出其中第k大的节点。
示例 1:
输入: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
输出: 4
示例 2:输入: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
输出: 4
限制：1 ≤ k ≤ 二叉搜索树元素个数
```

- 解题思路

```
var count int
var res int

func kthLargest(root *TreeNode, k int) int {
        count = k
        res = 0
        dfs(root)
        return res
}
func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Right)
        count--
        if count == 0 {
                res = root.Val
                return
```

(续下页)

(接上页)

```
        }
        dfs(root.Left)
}

#
var arr []int

func kthLargest(root *TreeNode, k int) int {
        arr = make([]int, 0)
        dfs(root)
        return arr[k-1]
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Right)
        arr = append(arr, root.Val)
        dfs(root.Left)
}

#
func kthLargest(root *TreeNode, k int) int {
        if root == nil {
                return 0
        }
        arr := make([]int, 0)
        stack := make([]*TreeNode, 0)
        cur := root
        for len(stack) > 0 || cur != nil {
                for cur != nil {
                        stack = append(stack, cur)
                        cur = cur.Left
                }
                node := stack[len(stack)-1]
                arr = append(arr, node.Val)
                stack = stack[:len(stack)-1]
                cur = node.Right
        }
        return arr[len(arr)-k]
}
```

76.55 面试题 55-I. 二叉树的深度 (2)

- 题目

```
输入一棵二叉树的根节点，求该树的深度。
从根节点到叶节点依次经过的节点（含根、叶节点）形成树的一条路径，最长路径的长度为树的深度。
例如：
给定二叉树 [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
返回它的最大深度 3 。
提示：节点总数 <= 10000
注意：本题与主站 104 题相同：
https://leetcode.cn/problems/maximum-depth-of-binary-tree/
```

- 解题思路

```
func maxDepth(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := maxDepth(root.Left)
        right := maxDepth(root.Right)

        return max(left, right) + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func maxDepth(root *TreeNode) int {
        if root == nil {
                return 0
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
```

(续下页)

(接上页)

```
        depth := 0
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        node := queue[0]
                        queue = queue[1:]
                        if node.Left != nil {
                                queue = append(queue, node.Left)
                        }
                        if node.Right != nil {
                                queue = append(queue, node.Right)
                        }
                }
                depth++
        }
        return depth
}
```

76.56 面试题 55-II. 平衡二叉树 (2)

- 题目

```
输入一棵二叉树的根节点，判断该树是不是平衡二叉树。
如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。
示例 1:给定二叉树 [3,9,20,null,null,15,7]
    3
   / \
  9  20
    /  \
   15   7
返回 true 。
示例 2:给定二叉树 [1,2,2,3,3,null,null,4,4]
       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
返回 false 。
限制：1 <= 树的结点个数 <= 10000
注意：本题与主站 110 题相同：
```

(续下页)

(接上页)

```
https://leetcode.cn/problems/balanced-binary-tree/
```

- 解题思路

```
func isBalanced(root *TreeNode) bool {
        _, isBalanced := recur(root)
        return isBalanced
}

func recur(root *TreeNode) (int, bool) {
        if root == nil {
                return 0, true
        }
        leftDepth, leftIsBalanced := recur(root.Left)
        if leftIsBalanced == false {
                return 0, false
        }
        rightDepth, rightIsBalanced := recur(root.Right)
        if rightIsBalanced == false {
                return 0, false
        }
        if -1 <= leftDepth-rightDepth &&
                leftDepth-rightDepth <= 1 {
                return max(leftDepth, rightDepth) + 1, true
        }
        return 0, false
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

#
func isBalanced(root *TreeNode) bool {
        return dfs(root) != -1
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
```

(续下页)

(接上页)

```
        left := dfs(root.Left)
        right := dfs(root.Right)
        if left != -1 && right != -1 &&
                abs(left, right) <= 1 {
                return max(left, right) + 1
        }
        return -1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

76.57 面试题 56-I. 数组中数字出现的次数 (5)

- 题目

```
一个整型数组 nums 里除两个数字之外，其他数字都出现了两次。
请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。
示例 1： 输入：nums = [4,1,4,6]输出：[1,6] 或 [6,1]
示例 2：输入：nums = [1,2,10,4,1,4,3,3] 输出：[2,10] 或 [10,2]
限制：
    2 <= nums.length <= 10000
```

- 解题思路

```
func singleNumbers(nums []int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        res := make([]int, 0)
```

(续下页)

(接上页)

```
        for i := range m {
                if m[i] == 1 {
                        res = append(res, i)
                }
        }
        return res
}

#
func singleNumbers(nums []int) []int {
        res := make([]int, 0)
        sort.Ints(nums)
        count := 0
        for i := 0; i < len(nums)-2; {
                if nums[i] == nums[i+1] {
                        i = i + 2
                }
                if i == len(nums)-1 {
                        res = append(res, nums[i])
                        return res
                }
                if nums[i] != nums[i+1] {
                        res = append(res, nums[i])
                        i++
                        count++
                        if count == 2 {
                                return res
                        }
                }
        }
        return res
}

#
/*
a = 10
1.a   00001010
2.取反 11110101
3.取反+1 11110110
4. a & (-a)
00001010
11110110
00000010
```

(续下页)

(接上页)

```
*/
func singleNumbers(nums []int) []int {
        res := make([]int, 2)
        temp := 0
        for i := 0; i < len(nums); i++ {
                temp = temp ^ nums[i]
        }
        last := temp & (-temp)
        for i := 0; i < len(nums); i++ {
                if nums[i]&last == 0 {
                        res[0] = res[0] ^ nums[i]
                } else {
                        res[1] = res[1] ^ nums[i]
                }
        }
        return res
}

#
func singleNumbers(nums []int) []int {
        res := make([]int, 2)
        temp := 0
        for i := 0; i < len(nums); i++ {
                temp = temp ^ nums[i]
        }
        last := 1
        for temp&last == 0 {
                last = last << 1
        }
        for i := 0; i < len(nums); i++ {
                if nums[i]&last == 0 {
                        res[0] = res[0] ^ nums[i]
                } else {
                        res[1] = res[1] ^ nums[i]
                }
        }
        return res
}

#
func singleNumbers(nums []int) []int {
        res := make([]int, 2)
        temp := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(nums); i++ {
                temp = temp ^ nums[i]
        }
        index := firstBit(temp)
        for i := 0; i < len(nums); i++ {
                if isBit(nums[i], index) {
                        res[0] = res[0] ^ nums[i]
                } else {
                        res[1] = res[1] ^ nums[i]
                }
        }
        return res
}

func firstBit(num int) int {
        res := 0
        for num&1 == 0 {
                res++
                num = num >> 1
        }
        return res
}

func isBit(num int, index int) bool {
        num = num >> index
        return num&1 == 1
}
```

76.58 面试题 56-II. 数组中数字出现的次数 II(5)

- 题目

```
在一个数组 nums␣
↪中除一个数字只出现一次之外，其他数字都出现了三次。请找出那个只出现一次的数字。
示例 1: 输入: nums = [3,4,3,3] 输出: 4
示例 2: 输入: nums = [9,1,7,9,7,9,7] 输出: 1
限制:
    1 <= nums.length <= 10000
    1 <= nums[i] < 2^31
```

- 解题思路

```
func singleNumber(nums []int) int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        for i := 0; i < len(nums); i++ {
                if m[nums[i]] == 1 {
                        return nums[i]
                }
        }
        return 0
}

# 2
func singleNumber(nums []int) int {
        arr := make([]int, 32)
        for i := 0; i < len(nums); i++ {
                value := nums[i]
                for j := 31; j >= 0; j-- {
                        arr[j] = arr[j] + value&1
                        value = value / 2
                }
        }
        res := 0
        for i := 0; i < 32; i++ {
                res = res << 1
                res = res + arr[i]%3
        }
        return res
}

# 3
func singleNumber(nums []int) int {
        sort.Ints(nums)
        if len(nums) == 1 {
                return nums[0]
        }
        i := 0
        for i < len(nums) {
                if i == len(nums)-1 {
                        return nums[len(nums)-1]
                }
                if nums[i] == nums[i+1] {
                        i = i + 3
```

(续下页)

(接上页)

```
                } else {
                        return nums[i]
                }
        }
        return nums[i]
}

# 4
func singleNumber(nums []int) int {
        m := make(map[int]int)
        sum1 := 0
        sum2 := 0
        for i := 0; i < len(nums); i++ {
                sum1 = sum1 + nums[i]
                if _, ok := m[nums[i]]; !ok {
                        sum2 = sum2 + nums[i]
                }
                m[nums[i]]++
        }
        return (sum2*3 - sum1) / 2
}

# 5
func singleNumber(nums []int) int {
        res, temp := 0, 0
        for i := 0; i < len(nums); i++ {
                res = (res ^ nums[i]) & ^temp
                temp = (temp ^ nums[i]) & ^res
        }
        return res
}
```

76.59 面试题 57. 和为 s 的两个数字 (2)

- 题目

```
输入一个递增排序的数组和一个数字s，在数组中查找两个数，使得它们的和正好是s。
如果有多对数字的和等于s，则输出任意一对即可。
示例 1：输入：nums = [2,7,11,15], target = 9 输出：[2,7] 或者 [7,2]
示例 2：输入：nums = [10,26,30,31,47,60], target = 40 输出：[10,30] 或者 [30,10]
限制：
1 <= nums.length <= 10^5
```

(续下页)

(接上页)

```
1 <= nums[i] <= 10^6
```

- 解题思路

```
func twoSum(nums []int, target int) []int {
        i := 0
        j := len(nums) - 1
        for i < j {
                sum := nums[i] + nums[j]
                if sum == target {
                        return []int{nums[i], nums[j]}
                } else if sum > target {
                        j--
                } else {
                        i++
                }
        }
        return nil
}

#
func twoSum(nums []int, target int) []int {
        m := make(map[int]int, len(nums))
        for i, b := range nums {
                if j, ok := m[target-b]; ok {
                        return []int{nums[j], nums[i]}
                }
                m[b] = i
        }
        return nil
}
```

76.60 面试题 57-II. 和为 s 的连续正数序列 (4)

- 题目

```
输入一个正整数 target ，输出所有和为 target 的连续正整数序列（至少含有两个数）。
序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。
示例 1: 输入: target = 9 输出: [[2,3,4],[4,5]]
示例 2: 输入: target = 15 输出: [[1,2,3,4,5],[4,5,6],[7,8]]
限制: 1 <= target <= 10^5
```

- 解题思路

```
func findContinuousSequence(target int) [][]int {
        res := make([][]int, 0)
        i := 1
        j := 2
        for i < j {
                sum := (i + j) * (j - i + 1) / 2
                if sum == target {
                        arr := make([]int, 0)
                        for k := i; k <= j; k++ {
                                arr = append(arr, k)
                        }
                        res = append(res, arr)
                        i++
                } else if sum < target {
                        j++
                } else {
                        i++
                }
        }
        return res
}

#
func findContinuousSequence(target int) [][]int {
        res := make([][]int, 0)
        i := 1
        j := 2
        mid := (1 + target) / 2
        curSum := i + j
        for i < mid {
                if curSum == target {
                        arr := make([]int, 0)
                        for k := i; k <= j; k++ {
                                arr = append(arr, k)
                        }
                        res = append(res, arr)
                }
                for curSum > target && i < mid {
                        curSum = curSum - i
                        i++
                        if curSum == target {
                                arr := make([]int, 0)
                                for k := i; k <= j; k++ {
                                        arr = append(arr, k)
```

(续下页)

(接上页)

```
                                }
                                res = append(res, arr)
                        }
                }
                j++
                curSum = curSum + j
        }
        return res
}

#
func findContinuousSequence(target int) [][]int {
        res := make([][]int, 0)
        arr := make([]int, target+1)
        for i := 1; i <= target; i++ {
                arr[i] = arr[i-1] + i
        }
        for i := 1; i <= (target+1)/2; i++ {
                for j := i + 1; j <= target && arr[j]-arr[i-1] <= target; j++ {
                        if arr[j]-arr[i-1] == target {
                                temp := make([]int, 0)
                                for k := i; k <= j; k++ {
                                        temp = append(temp, k)
                                }
                                res = append(res, temp)
                                break
                        }
                }
        }
        return res
}

#
// target = nA1 + n(n-1)/2
func findContinuousSequence(target int) [][]int {
        res := make([][]int, 0)
        for i := (target + 1) / 2; i >= 2; i-- {
                nA1 := target - i*(i-1)/2
                if nA1 <= 0 {
                        continue
                }
                if nA1%i == 0 {
                        start := nA1 / i
```

(续下页)

(接上页)

```
                        arr := make([]int, 0)
                        for j := 0; j < i; j++ {
                                arr = append(arr, start+j)
                        }
                        res = append(res, arr)
                }
        }
        return res
}
```

76.61 面试题 58-I. 翻转单词顺序 (3)

- 题目

```
输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。
为简单起见，标点符号和普通字母一样处理。
例如输入字符串"I am a student. "，则输出"student. a am I"。
示例 1：输入："the sky is blue"  输出："blue is sky the"
示例 2：输入："  hello world!  " 输出："world! hello"
解释：输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
示例 3：输入："a good   example" 输出："example good a"
解释：如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
说明：
    无空格字符构成一个单词。
    输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
    如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
注意：本题与主站 151 题相同：https://leetcode.cn/problems/reverse-words-in-a-string/
注意：此题对比原题有改动
```

- 解题思路

```
func reverseWords(s string) string {
        s = strings.Trim(s, " ")
        arr := strings.Fields(s)
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
        return strings.Join(arr, " ")
}

#
func reverseWords(s string) string {
```

(续下页)

(接上页)

```
        arr := []byte(s)
        arr = reverse(arr)
        i := 0
        j := 0
        res := ""
        flag := false
        for i < len(arr) {
                if arr[i] == ' ' {
                        if flag == true {
                                res = res + " " + string(reverse(arr[j:i]))
                                flag = false
                        }
                        i++
                        j = i
                } else {
                        i++
                        if i == len(arr) {
                                res = res + " " + string(reverse(arr[j:i]))
                        }
                        flag = true
                }
        }
        if len(res) > 0 {
                return res[1:]
        }
        return res
}

func reverse(arr []byte) []byte {
        start := 0
        end := len(arr) - 1
        for start < end {
                arr[start], arr[end] = arr[end], arr[start]
                start++
                end--
        }
        return arr
}

#
func reverseWords(s string) string {
        arr := []byte(s)
        i := len(arr) - 1
```

(续下页)

(接上页)

```
        j := len(arr)
        res := ""
        flag := false
        for i >= 0 {
                if arr[i] == ' ' {
                        if flag == true {
                                res = res + " " + string(arr[i+1:j])
                                flag = false
                        }
                        j = i
                        i--
                } else {
                        if i == 0 {
                                res = res + " " + string(arr[i:j])
                        }
                        i--
                        flag = true
                }
        }
        if len(res) > 0 {
                return res[1:]
        }
        return res
}
```

76.62 面试题 58-II. 左旋转字符串 (2)

- 题目

```
字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的
比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。
示例 1：输入：s = "abcdefg", k = 2 输出："cdefgab"
示例 2：输入：s = "lrloseumgh", k = 6 输出："umghlrlose"
限制：  1 <= k < s.length <= 10000
```

- 解题思路

```
func reverseLeftWords(s string, n int) string {
        n = n % len(s)
        return s[n:] + s[:n]
}
```

(续下页)

(接上页)

```
#
func reverseLeftWords(s string, n int) string {
        n = n % len(s)
        arr := []byte(s)
        reverse(arr, 0, n-1)
        reverse(arr, n, len(arr)-1)
        reverse(arr, 0, len(arr)-1)
        return string(arr)
}

func reverse(arr []byte, start, end int) []byte {
        for start < end {
                arr[start], arr[end] = arr[end], arr[start]
                start++
                end--
        }
        return arr
}
```

76.63 面试题 59-I. 滑动窗口的最大值 (4)

- 题目

```
给定一个数组 nums 和滑动窗口的大小 k，请找出所有滑动窗口里的最大值。
示例:输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3 输出: [3,3,5,5,6,7]
解释:
  滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
提示：你可以假设 k 总是有效的，在输入数组不为空的情况下，1 ≤ k ≤ 输入数组的大小。
注意：本题与主站 239 题相同：https://leetcode.cn/problems/sliding-window-maximum/
```

- 解题思路

```
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
```

(续下页)

(接上页)

```
                return res
        }
        for i := 0; i < len(nums)-k+1; i++ {
                max := nums[i]
                for j := i; j < i+k; j++ {
                        if nums[j] > max {
                                max = nums[j]
                        }
                }
                res = append(res, max)
        }
        return res
}

#
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        max := math.MaxInt32
        for i := 0; i < len(nums)-k+1; i++ {
                if i == 0 || nums[i-1] == max {
                        max = nums[i]
                        for j := i; j < i+k; j++ {
                                if nums[j] > max {
                                        max = nums[j]
                                }
                        }
                } else {
                        if nums[i+k-1] > max {
                                max = nums[i+k-1]
                        }
                }
                res = append(res, max)
        }
        return res
}

#
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
```

(续下页)

(接上页)

```
                return res
        }
        deque := make([]int, 0)
        for i := 0; i < k; i++ {
                for len(deque) > 0 && nums[i] >= nums[deque[len(deque)-1]] {
                        deque = deque[:len(deque)-1]
                }
                deque = append(deque, i)
        }
        for i := k; i < len(nums); i++ {
                res = append(res, nums[deque[0]])
                for len(deque) > 0 && nums[i] >= nums[deque[len(deque)-1]] {
                        deque = deque[:len(deque)-1]
                }
                if len(deque) > 0 && deque[0] <= i-k {
                        deque = deque[1:]
                }
                deque = append(deque, i)
        }
        res = append(res, nums[deque[0]])
        return res
}

#
func maxSlidingWindow(nums []int, k int) []int {
        res := make([]int, 0)
        if len(nums) == 0 {
                return res
        }
        intHeap := make(IntHeap, 0, k)
        heap.Init(&intHeap)
        for i := 0; i < k; i++ {
                heap.Push(&intHeap, nums[i])
        }
        for i := k; i < len(nums); i++ {
                temp := heap.Pop(&intHeap).(int)
                res = append(res, temp)
                if temp != nums[i-k] {
                        intHeap.Remove(nums[i-k])
                        heap.Push(&intHeap, temp)
                        heap.Push(&intHeap, nums[i])
                } else {
                        heap.Push(&intHeap, nums[i])
```

(续下页)

(接上页)

```
                }
        }
        res = append(res, heap.Pop(&intHeap).(int))
        return res
}

type IntHeap []int

func (i IntHeap) Len() int {
        return len(i)
}

func (i IntHeap) Less(x, y int) bool {
        return i[x] > i[y]
}

func (i IntHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *IntHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *IntHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

func (i *IntHeap) Remove(x interface{}) {
        for j := 0; j < len(*i); j++ {
                if (*i)[j] == x {
                        *i = append((*i)[:j], (*i)[j+1:]...)
                        break
                }
        }
        heap.Init(i)
}
```

76.64 面试题 59-II. 队列的最大值 (2)

- 题目

```
请定义一个队列并实现函数 max_value 得到队列里的最大值，
要求函数max_value、push_back 和 pop_front 的均摊时间复杂度都是O(1)。
若队列为空，pop_front 和 max_value 需要返回 -1
示例 1：输入：
["MaxQueue","push_back","push_back","max_value","pop_front","max_value"]
[[],[1],[2],[],[],[]]
输出：[null,null,null,2,1,2]

示例 2：输入：
["MaxQueue","pop_front","max_value"]
[[],[],[]]
输出：[null,-1,-1]
限制：
    1 <= push_back,pop_front,max_value的总操作数 <= 10000
    1 <= value <= 10^5
```

- 解题思路

```
type MaxQueue struct {
        data []int
        max  []int
}

func Constructor() MaxQueue {
        return MaxQueue{
                data: make([]int, 0),
                max:  make([]int, 0),
        }
}

func (this *MaxQueue) Max_value() int {
        if len(this.max) == 0 {
                return -1
        }
        return this.max[0]
}

func (this *MaxQueue) Push_back(value int) {
        this.data = append(this.data, value)
        for len(this.max) > 0 && value > this.max[len(this.max)-1] {
```

(续下页)

(接上页)

```
                this.max = this.max[:len(this.max)-1]
        }
        this.max = append(this.max, value)
}

func (this *MaxQueue) Pop_front() int {
        res := -1
        if len(this.data) > 0 {
                res = this.data[0]
                this.data = this.data[1:]
                if res == this.max[0] {
                        this.max = this.max[1:]
                }
        }
        return res
}

#
type MaxQueue struct {
        data *list.List
        max  []int
}

func Constructor() MaxQueue {
        return MaxQueue{
                data:list.New(),
                max:  make([]int, 0),
        }
}

func (this *MaxQueue) Max_value() int {
        if len(this.max) == 0 {
                return -1
        }
        return this.max[0]
}

func (this *MaxQueue) Push_back(value int) {
        this.data.PushBack(value)
        for len(this.max) > 0 && value > this.max[len(this.max)-1] {
                this.max = this.max[:len(this.max)-1]
        }
        this.max = append(this.max, value)
```

(续下页)

(接上页)

```
}

func (this *MaxQueue) Pop_front() int {
        res := -1
        if this.data.Len() > 0 {
                res = this.data.Front().Value.(int)
                this.data.Remove(this.data.Front())
                if res == this.max[0] {
                        this.max = this.max[1:]
                }
        }
        return res
}
```

76.65 面试题 60.n 个骰子的点数 (2)

- 题目

```
把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率。
你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能掷出的点数集合中第␣
↪i 小的那个的概率。
示例 1:输入: 1
输出: [0.16667,0.16667,0.16667,0.16667,0.16667,0.16667]
示例 2:输入: 2 输出:
[0.02778,0.05556,0.08333,0.11111,0.13889,0.16667,0.13889,0.11111,
0.08333,0.05556,0.02778]
限制：
1 <= n <= 11
```

- 解题思路

```
func twoSum(n int) []float64 {
        res := make([]float64, 0)
        if n < 1 {
                return res
        }
        arr := [2][]int{}
        arr[0] = make([]int, 6*n+1)
        arr[1] = make([]int, 6*n+1)
        flag := 0
        for i := 1; i <= 6; i++ {
                arr[flag][i] = 1
```

(续下页)

(接上页)

```
        }
        for k := 2; k <= n; k++ {
                for i := 0; i < k; i++ {
                        arr[1-flag][i] = 0
                }
                for i := k; i <= 6*n; i++ {
                        arr[1-flag][i] = 0
                        // 当前轮的第N位等于前一个数组第N-1,N-2,N-3,N-4,N-5,N-6位之和
                        for j := 1; j <= i && j <= 6; j++ {
                                arr[1-flag][i] += arr[flag][i-j]
                        }
                }
                flag = 1 - flag
        }
        total := math.Pow(float64(6), float64(n))
        for i := n; i <= 6*n; i++ {
                ratio := float64(arr[flag][i]) / total
                res = append(res, ratio)
        }
        return res
}

#
// f(n,k)=f(n-1,k-1)+f(n-1,k-2)+f(n-1,k-3)+f(n-1,k-4)+f(n-1,k-5)+f(n-1,k-6)
func twoSum(n int) []float64 {
        res := make([]float64, 0)
        if n < 1 {
                return res
        }
        arr := make([]int, 6*n+1)
        for i := 1; i <= 6; i++ {
                arr[i] = 1
        }
        for i := 2; i <= n; i++ {
                for j := 6 * i; j >= i; j-- {
                        arr[j] = 0
                        for k := 1; k <= 6; k++ {
                                if j-k >= i-1 {
                                        arr[j] = arr[j] + arr[j-k]
                                }
                        }
                }
        }
```

(续下页)

(接上页)

```
        total := math.Pow(float64(6), float64(n))
        for i := n; i <= 6*n; i++ {
                ratio := float64(arr[i]) / total
                res = append(res, ratio)
        }
        return res
}

#
var arr []int
var start int

func twoSum(n int) []float64 {
        res := make([]float64, 0)
        if n < 1 {
                return res
        }
        start = n
        arr = make([]int, 5*n+1)
        for i := 1; i <= 6; i++ {
                dfs(n, i)
        }
        total := math.Pow(float64(6), float64(n))
        for i := n; i <= 6*n; i++ {
                ratio := float64(arr[i-n]) / total
                res = append(res, ratio)
        }
        return res
}

func dfs(current, sum int) {
        if current == 1 {
                arr[sum-start]++
        } else {
                for i := 1; i <= 6; i++ {
                        dfs(current-1, sum+i)
                }
        }
}
```

76.66 面试题 61. 扑克牌中的顺子 (3)

- 题目

```
从扑克牌中随机抽5张牌，判断是不是一个顺子，即这5张牌是不是连续的。
2～10为数字本身，A为1，J为11，Q为12，K为13，而大、小王为 0 ，可以看成任意数字。A
↪不能视为 14。
示例 1:输入: [1,2,3,4,5] 输出: True
示例 2:输入: [0,0,1,2,5] 输出: True
限制: 数组长度为 5 数组的数取值为 [0, 13] .
```

- 解题思路

```
func isStraight(nums []int) bool {
        sort.Ints(nums)
        sum := 0
        for i := 0; i < 4; i++ {
                if nums[i] == 0 {
                        continue
                }
                // 非王重复
                if nums[i] == nums[i+1] {
                        return false
                }
                sum = sum + nums[i+1] - nums[i]
        }
        return sum <= 4
}

#
func isStraight(nums []int) bool {
        m := make(map[int]bool)
        max, min := -1, 14
        countZero := 0
        for i := 0; i < 5; i++ {
                if nums[i] == 0 {
                        countZero++
                        continue
                }
                if m[nums[i]] {
                        return false
                }
                m[nums[i]] = true
                if nums[i] > max {
```

(续下页)

(接上页)

```
                        max = nums[i]
                }
                if nums[i] < min {
                        min = nums[i]
                }
        }
        if countZero == 0 {
                return max-min == 4
        }
        return max-min <= 4
}

#
func isStraight(nums []int) bool {
        sort.Ints(nums)
        zero := 0
        gap := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        zero++
                }
        }
        small := zero
        big := small + 1
        for big < len(nums) {
                if nums[small] == nums[big] {
                        return false
                }
                gap = gap + nums[big] - nums[small] - 1
                small++
                big++
        }
        if gap > zero {
                return false
        }
        return true
}
```

76.67 面试题 62. 圆圈中最后剩下的数字 (2)

- 约瑟夫环理解参考: https://leetcode.cn/problems/yuan-quan-zhong-zui-hou-sheng-xia-de-shu-zi-lcof/solution/huan-ge-jiao-du-ju-li-jie-jue-yue-se-fu-huan-by-as/
- 题目

```
0,1,,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字。
求出这个圆圈里剩下的最后一个数字。
例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，
则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3。
示例 1：输入：n = 5, m = 3 输出：3
示例 2：输入：n = 10, m = 17 输出：2
限制：
    1 <= n <= 10^5
    1 <= m <= 10^6
```

- 解题思路

```
func lastRemaining(n int, m int) int {
        if n == 1 {
                return 0
        }
        return (lastRemaining(n-1, m) + m) % n
}

# 2
func lastRemaining(n int, m int) int {
        res := 0
        for i := 2; i <= n; i++ {
                res = (res + m) % i
        }
        return res
}

# 超时
func lastRemaining(n int, m int) int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        last := 0
        for len(arr) > 1 {
                index := (last + m - 1) % len(arr)
                arr = remove(arr, index)
```

(续下页)

(接上页)

```
                last = index
        }
        return arr[0]
}

func remove(arr []int, index int) []int {
        if index == 0 {
                return arr[1:]
        }
        if index == len(arr)-1 {
                return arr[:len(arr)-1]
        }
        return append(arr[:index], arr[index+1:]...)
}
```

76.68 面试题 63. 股票的最大利润 (3)

- 题目

```
假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少？
示例 1:输入: [7,1,5,3,6,4] 输出: 5
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，
最大利润 = 6-1 = 5 。注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格。
示例 2:输入: [7,6,4,3,1] 输出: 0
解释: 在这种情况下, 没有交易完成, 所以最大利润为 0。
限制： 0 <= 数组长度 <= 10^5
注意：本题与主站 121 题相同：
https://leetcode.cn/problems/best-time-to-buy-and-sell-stock/
```

- 解题思路

```
func maxProfit(prices []int) int {
        max := 0
        length := len(prices)
        for i := 0; i < length-1; i++ {
                for j := i + 1; j <= length-1; j++ {
                        if prices[j]-prices[i] > max {
                                max = prices[j] - prices[i]
                        }
                }
        }
        return max
```

(续下页)

(接上页)

```
}

#
func maxProfit(prices []int) int {
        if len(prices) < 2 {
                return 0
        }
        min := prices[0]
        profit := 0
        for i := 1; i < len(prices); i++ {
                if prices[i] < min {
                        min = prices[i]
                }
                if profit < prices[i]-min {
                        profit = prices[i] - min
                }
        }
        return profit
}

#
func maxProfit(prices []int) int {
        if len(prices) < 2 {
                return 0
        }
        max := 0
        profit := 0
        for i := len(prices) - 1; i >= 0; i-- {
                if max < prices[i] {
                        max = prices[i]
                }
                if profit < max-prices[i] {
                        profit = max - prices[i]
                }
        }
        return profit
}
```

76.69 面试题 64. 求 1+2+…+n(2)

- 题目

```
求 1+2+...+n ，要求不能使用乘除法、
for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。
示例 1：输入：n = 3 输出：6
示例 2：输入：n = 9 输出：45
限制：1 <= n <= 10000
```

- 解题思路

```
var res int

func sumNums(n int) int {
        res = 0
        dfs(n)
        return res
}

func dfs(n int) bool {
        res = res + n
        return n > 0 && dfs(n-1)
}

#
func sumNums(n int) int {
        return (int(math.Pow(float64(n), float64(2))) + n) >> 1
}
```

76.70 面试题 65. 不用加减乘除做加法 (2)

- 题目

```
写一个函数，求两个整数之和，要求在函数体内不得使用 “+”、“-”、“*”、“/” ␣
↪四则运算符号。
示例:输入：a = 1, b = 1 输出：2
提示：
    a, b 均可能是负数或 0
    结果不会溢出 32 位整数
```

- 解题思路

```
// 非进位和：异或运算
// 进位：与运算+左移一位
func add(a int, b int) int {
        for b != 0 {
                a, b = a^b, (a&b)<<1
        }
        return a
}

#
func add(a int, b int) int {
        if b == 0 {
                return a
        }
        return add(a^b, (a&b)<<1)
}
```

76.71 面试题 66. 构建乘积数组 (2)

- 题目

```
给定一个数组 A[0,1,…,n-1]，请构建一个数组 B[0,1,…,n-1]，
其中 B 中的元素 B[i]=A[0]×A[1]×…×A[i-1]×A[i+1]×…×A[n-1]。不能使用除法。
示例:输入: [1,2,3,4,5] 输出: [120,60,40,30,24]
提示：所有元素乘积之和不会溢出 32 位整数
    a.length <= 100000
```

- 解题思路

```
func constructArr(a []int) []int {
        res := make([]int, len(a))
        if len(a) == 0 {
                return res
        }
        res[0] = 1
        for i := 1; i < len(res); i++ {
                res[i] = res[i-1] * a[i-1]
        }
        temp := 1
        for i := len(res) - 2; i >= 0; i-- {
                res[i] = res[i] * a[i+1] * temp
                temp = temp * a[i+1]
        }
```

(续下页)

(接上页)

```
        return res
}

#
func constructArr(a []int) []int {
        res := make([]int, len(a))
        if len(a) == 0 {
                return res
        }
        left := make([]int, len(a))
        left[0] = 1
        right := make([]int, len(a))
        right[len(a)-1] = 1
        for i := 1; i < len(a); i++ {
                left[i] = left[i-1] * a[i-1]
        }
        for i := len(a) - 2; i >= 0; i-- {
                right[i] = right[i+1] * a[i+1]
        }
        for i := 0; i < len(a); i++ {
                res[i] = left[i] * right[i]
        }
        return res
}
```

76.72 面试题 67. 把字符串转换成整数 (2)

- 题目

```
写一个函数 StrToInt，实现把字符串转换成整数这个功能。不能使用 atoi␣
↪或者其他类似的库函数。
首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。
当我们寻找到的第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字组合起来，
作为该整数的正负号；假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成整数。
该字符串除了有效的整数部分之后也可能会存在多余的字符，这些字符可以被忽略，它们对于函数不应该造成影响
注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、
字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换。
在任何情况下，若函数不能进行有效的转换时，请返回 0。
说明：
假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 [-231,  231 -␣
↪1]。如果数值超过这个范围，请返回  INT_MAX (231 - 1) 或 INT_MIN (-231) 。
```

(续下页)

(接上页)

```
示例 1:输入: "42"输出: 42
示例 2:输入: "   -42"输出: -42
解释: 第一个非空白字符为 '-', 它是一个负号。
我们尽可能将负号与后面所有连续出现的数字组合起来, 最后得到 -42 。
示例 3:输入: "4193 with words"输出: 4193
解释: 转换截止于数字 '3' , 因为它的下一个字符不为数字。
示例 4:输入: "words and 987" 输出: 0
解释: 第一个非空字符是 'w', 但它不是数字或正、负号。 因此无法执行有效的转换。
示例 5:输入: "-91283472332" 输出: -2147483648 因此返回 INT_MIN (-231) 。
注意: 本题与主站 8 题相同: https://leetcode.cn/problems/string-to-integer-atoi/
```

- 解题思路

```
func strToInt(str string) int {
        i := 0
        for i < len(str) && str[i] == ' ' {
                i++
        }
        str = str[i:]
        arr := make([]byte, 0)
        isFlag := byte(' ')
        for j := 0; j < len(str); j++ {
                if str[j] >= '0' && str[j] <= '9' {
                        arr = append(arr, str[j])
                } else {
                        if len(arr) > 0 {
                                break
                        }
                        if str[j] != ' ' && str[j] != '+' && str[j] != '-' {
                                return 0
                        }
                        if isFlag != ' ' {
                                return 0
                        }
                        isFlag = str[j]
                }
        }
        res := 0
        for i := 0; i < len(arr); i++ {
                value := int(arr[i] - '0')
                res = res*10 + value
                if isFlag == '-' {
                        if -1*res < math.MinInt32 {
                                return math.MinInt32
```

(续下页)

(接上页)

```
                        }
                } else if isFlag == ' ' || isFlag == '+' {
                        if res > math.MaxInt32 {
                                return math.MaxInt32
                        }
                }
        }
        if isFlag == '-' {
                return -1 * res
        }
        return res
}

#
func strToInt(str string) int {
        re := regexp.MustCompile(`^[+-]?\d+`)
        arrS := re.FindAllString(strings.Trim(str, " "), -1)
        if len(arrS) == 0{
                return 0
        }
        arr := arrS[0]
        res := 0
        isFlag := byte(' ')
        if !(arr[0] >= '0' && arr[0] <= '9') {
                isFlag = arr[0]
                arr = arr[1:]
        }
        for i := 0; i < len(arr); i++ {
                value := int(arr[i] - '0')
                if isFlag == '-' {
                        if res > 214748364 || (res==214748364 && value >= 8) {
                                return math.MinInt32
                        }
                } else if isFlag == ' ' || isFlag == '+' {
                        if res > 214748364 || (res==214748364 && value >= 7) {
                                return math.MaxInt32
                        }
                }
                res = res*10 + value
        }
        if isFlag == '-' {
                return -1 * res
        }
```

(续下页)

(接上页)

```
        return res
}
```

CHAPTER 77

剑指 OfferII-Easy

77.1 剑指 OfferII001. 整数除法 (2)

- 题目

```
给定两个整数 a 和 b ，求它们的除法的商 a/b ，要求不得使用乘号 '*'、除号 '/'␣
↪以及求余符号 '%'。
注意：整数除法的结果应当截去（truncate）其小数部分，例如：truncate(8.345) =␣
↪8以及truncate(-2.7335) = -2
假设我们的环境只能存储 32 位有符号整数，其数值范围是 [-231,
↪231-1]。本题中，如果除法结果溢出，则返回 231- 1
示例 1：输入：a = 15, b = 2 输出：7
解释：15/2 = truncate(7.5) = 7
示例 2：输入：a = 7, b = -3 输出：-2
解释：7/-3 = truncate(-2.33333..) = -2
示例 3：输入：a = 0, b = 1 输出：0
示例 4：输入：a = 1, b = 1 输出：1
提示:-231<= a, b <= 231- 1
b != 0
注意：本题与主站 29题相同
```

- 解题思路

```
func divide(a int, b int) int {
        if b == 0 || a == 0 {
```

(续下页)

(接上页)

```
                return 0
        }
        if b == 1 {
                return a
        }
        flag, count := 1, 1
        if a < 0 {
                flag = -flag
                a = -a
        }
        if b < 0 {
                flag = -flag
                b = -b
        }
        x, y, z := a, b, 0
        temp := y
        for x-y >= 0 {
                for x-y >= 0 {
                        x = x - y
                        z = z + count
                        y = y + y
                        count = count + count
                }
                y = temp
                count = 1
        }
        if z > math.MaxInt32 {
                return math.MaxInt32
        }
        if flag < 0 {
                return -z
        }
        return z
}

# 2
func divide(a int, b int) int {
        res := a / b
        if res > math.MaxInt32 {
                return math.MaxInt32
        }
        return res
}
```

77.2 剑指 OfferII002. 二进制加法 (2)

- 题目

```
给定两个 01 字符串a和b，请计算它们的和，并以二进制字符串的形式输出。
输入为 非空 字符串且只包含数字1和0。
示例1:输入: a = "11", b = "10" 输出: "101"
示例2:输入: a = "1010", b = "1011" 输出: "10101"
提示: 每个字符串仅由字符 '0' 或 '1' 组成。
1 <= a.length, b.length <= 10^4
字符串如果不是 "0" ，就都不含前导零。
注意: 本题与主站 67题相同
```

- 解题思路

```go
func addBinary(a string, b string) string {
        if len(a) < len(b) {
                a, b = b, a
        }
        length := len(a)

        A := transToInt(a, length)
        B := transToInt(b, length)

        return makeString(add(A, B))
}

func transToInt(s string, length int) []int {
        result := make([]int, length)
        ls := len(s)
        for i, b := range s {
                result[length-ls+i] = int(b - '0')
        }
        return result
}

func add(a, b []int) []int {
        length := len(a) + 1
        result := make([]int, length)
        for i := length - 1; i >= 1; i-- {
                temp := result[i] + a[i-1] + b[i-1]
                result[i] = temp % 2
                result[i-1] = temp / 2
        }
```

(续下页)

(接上页)

```
        i := 0
        for i < length-1 && result[i] == 0 {
                i++
        }
        return result[i:]
}

func makeString(nums []int) string {
        bytes := make([]byte, len(nums))
        for i := range bytes {
                bytes[i] = byte(nums[i]) + '0'
        }
        return string(bytes)
}

// 2
func addBinary(a string, b string) string {
        i := len(a) - 1
        j := len(b) - 1
        result := ""
        flag := 0
        current := 0

        for i >= 0 || j >= 0 {
                intA, intB := 0, 0
                if i >= 0 {
                        intA = int(a[i] - '0')
                }
                if j >= 0 {
                        intB = int(b[j] - '0')
                }
                current = intA + intB + flag
                flag = 0
                if current >= 2 {
                        flag = 1
                        current = current - 2
                }
                cur := strconv.Itoa(current)
                result = cur + result
                i--
                j--
        }
        if flag == 1 {
```

(续下页)

(接上页)

```
                result = "1" + result
        }
        return result
}
```

77.3 剑指 OfferII003. 前 n 个数字二进制中 1 的个数 (4)

- 题目

```
给定一个非负整数 n，请计算 0 到 n 之间的每个数字的二进制表示中 1␣
↪的个数，并输出一个数组。
示例 1:输入: n = 2 输出: [0,1,1]
解释: 0 --> 0
1 --> 1
2 --> 10
示例2:输入: n = 5 输出: [0,1,1,2,1,2]
解释:0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
说明 :0 <= n <= 105
进阶:给出时间复杂度为O(n*sizeof(integer))的解答非常容易。但你可以在线性时间O(n)内用一趟扫描做到吗？
要求算法的空间复杂度为O(n)。
你能进一步完善解法吗？要求在C++或任何其他语言中不使用任何内置函数（如 C++ 中的__
↪builtin_popcount）来执行此操作。
注意: 本题与主站 338题相同
```

- 解题思路

```
func countBits(n int) []int {
        res := make([]int, n+1)
        for i := 1; i <= n; i++ {
                res[i] = res[i&(i-1)] + 1
        }
        return res
}

# 2
func countBits(n int) []int {
        dp := make([]int, n+1)
```

(续下页)

(接上页)

```
        for i := 1; i <= n; i++ {
                if i%2 == 0 {
                        dp[i] = dp[i/2]
                } else {
                        dp[i] = dp[i-1] + 1
                }
        }
        return dp
}

# 3
func countBits(n int) []int {
        res := make([]int, 0)
        for i := 0; i <= n; i++ {
                count := 0
                value := i
                for value != 0 {
                        if value%2 == 1 {
                                count++
                        }
                        value = value / 2
                }
                res = append(res, count)
        }
        return res
}

# 4
func countBits(n int) []int {
        res := make([]int, 0)
        for i := 0; i <= n; i++ {
                count := bits.OnesCount(uint(i))
                res = append(res, count)
        }
        return res
}
```

77.4 剑指 OfferII006. 排序数组中两个数字之和 (4)

- 题目

```
给定一个已按照 升序排列 的整数数组numbers␣
↪，请你从数组中找出两个数满足相加之和等于目标数target 。
函数应该以长度为 2 的整数数组的形式返回这两个数的下标值。numbers 的下标 从 0开始计数␣
↪，
所以答案数组应当满足 0<= answer[0] < answer[1] <numbers.length。
假设数组中存在且只存在一对符合条件的数字，同时一个数字不能使用两次。
示例 1: 输入: numbers = [1,2,4,6,10], target = 8 输出: [1,3]
解释: 2 与 6 之和等于目标数 8 。因此 index1 = 1, index2 = 3 。
示例 2: 输入: numbers = [2,3,4], target = 6 输出: [0,2]
示例 3: 输入: numbers = [-1,0], target = -1 输出: [0,1]
提示: 2 <= numbers.length <= 3 * 104
-1000 <= numbers[i] <= 1000
numbers 按 递增顺序 排列
-1000 <= target <= 1000
仅存在一个有效答案
注意: 本题与主站 167 题相似 (下标起点不同)
```

- 解题思路

```
func twoSum(numbers []int, target int) []int {
        m := make(map[int]int, len(numbers))
        for i, n := range numbers {
                if m[target-n] != 0 {
                        return []int{m[target-n] - 1, i}
                }
                m[n] = i + 1
        }
        return nil
}

#2
func twoSum(nums []int, target int) []int {
        m := make(map[int]int, len(nums))
        for k, v := range nums {
                m[v] = k
        }

        for i := 0; i < len(nums); i++ {
                b := target - nums[i]
                if num, ok := m[b]; ok && num != i {
```

(续下页)

(接上页)

```
                        return []int{i, m[b]}
                }
        }
        return []int{}
}

# 3
func twoSum(nums []int, target int) []int {
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if nums[i]+nums[j] == target {
                                return []int{i, j}
                        }
                }
        }
        return []int{}
}

# 4
func twoSum(numbers []int, target int) []int {
        first := 0
        last := len(numbers) - 1
        result := make([]int, 2)

        for {
                if numbers[first]+numbers[last] == target {
                        result[0] = first
                        result[1] = last
                        return result
                } else if numbers[first]+numbers[last] > target {
                        last--
                } else {
                        first++
                }
        }
}
```

77.5 剑指 OfferII012. 左右两边子数组的和相等 (2)

- 题目

```
给你一个整数数组nums ，请计算数组的 中心下标 。
数组 中心下标 是数组的一个下标，其左侧所有元素相加的和等于右侧所有元素相加的和。
如果中心下标位于数组最左端，那么左侧数之和视为 0␣
↪，因为在下标的左侧不存在元素。这一点对于中心下标位于数组最右端同样适用。
如果数组有多个中心下标，应该返回 最靠近左边 的那一个。如果数组不存在中心下标，返回 -1␣
↪。
示例 1：输入：nums = [1,7,3,6,5,6] 输出：3
解释：中心下标是 3 。
左侧数之和 sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11 ，
右侧数之和 sum = nums[4] + nums[5] = 5 + 6 = 11 ，二者相等。
示例 2：输入：nums = [1, 2, 3] 输出：-1
解释：数组中不存在满足此条件的中心下标。
示例 3：输入：nums = [2, 1, -1]输出：0
解释：中心下标是 0 。
左侧数之和 sum = 0 ，（下标 0 左侧不存在元素），
右侧数之和 sum = nums[1] + nums[2] = 1 + -1 = 0 。
提示：1 <= nums.length <= 104
-1000 <= nums[i] <= 1000
注意：本题与主站 724题相同：
```

- 解题思路

```
func pivotIndex(nums []int) int {
        sum := 0
        for i := range nums {
                sum = sum + nums[i]
        }
        left := 0
        for i := range nums {
                if left*2+nums[i] == sum {
                        return i
                }
                left = left + nums[i]
        }
        return -1
}

# 2
func pivotIndex(nums []int) int {
        if len(nums) == 0 {
```

(续下页)

(接上页)

```
                return -1
        }
        arr := make([]int, len(nums))
        arr[0] = nums[0]
        for i := 1; i < len(nums); i++ {
                arr[i] = arr[i-1] + nums[i]
        }
        for i := 0; i < len(nums); i++ {
                var left, right int
                if i == 0 {
                        left = 0
                } else {
                        left = arr[i-1]
                }
                r := i + 1
                if r > len(nums)-1 {
                        right = 0
                } else {
                        right = arr[len(nums)-1] - arr[i]
                }
                if left == right {
                        return i
                }
        }
        return -1
}
```

77.6 剑指 OfferII018. 有效的回文 (2)

- 题目

```
给定一个字符串 s ，验证 s是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。
本题中，将空字符串定义为有效的回文串。
示例 1:输入: s = "A man, a plan, a canal: Panama" 输出: true
解释："amanaplanacanalpanama" 是回文串
示例 2:输入: s = "race a car" 输出: false
解释："raceacar" 不是回文串
提示：1 <= s.length <= 2 * 105
字符串 s 由 ASCII 字符组成
注意: 本题与主站 125题相同
```

- 解题思路

```
func isPalindrome(s string) bool {
        s = strings.ToLower(s)
        i, j := 0, len(s)-1

        for i < j {
                for i < j && !isChar(s[i]) {
                        i++
                }
                for i < j && !isChar(s[j]) {
                        j--
                }
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

func isChar(c byte) bool {
        if ('a' <= c && c <= 'z') || ('0' <= c && c <= '9') {
                return true
        }
        return false
}

# 2
func isPalindrome(s string) bool {
        str := ""
        s = strings.ToLower(s)
        for _, value := range s {
                if (value >= '0' && value <= '9') || (value >= 'a' && value <= 'z') {
                        str += string(value)
                }
        }
        if len(str) == 0 {
                return true
        }
        i := 0
        j := len(str) - 1
        for i <= j {
                if str[i] != str[j] {
                        return false
```

(续下页)

(接上页)

```
                }
                i++
                j--
        }
        return true
}
```

77.7 剑指 OfferII019. 最多删除一个字符得到回文 (2)

- 题目

```
给定一个非空字符串s，请判断如果最多 从字符串中删除一个字符能否得到一个回文字符串。
示例 1:输入: s = "aba" 输出: true
示例 2:输入: s = "abca" 输出: true
解释: 可以删除 "c" 字符 或者 "b" 字符
示例 3:输入: s = "abc" 输出: false
提示:1 <= s.length <= 105
s 由小写英文字母组成
注意: 本题与主站 680题相同
```

- 解题思路

```
func validPalindrome(s string) bool {
        i := 0
        j := len(s) - 1
        for i < j {
                if s[i] != s[j] {
                        return isPalindrome(s, i, j-1) || isPalindrome(s, i+1, j)
                }
                i++
                j--
        }
        return true
}

func isPalindrome(s string, i, j int) bool {
        for i < j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
```

(续下页)

(接上页)

```
        }
        return true
}

# 2
func validPalindrome(s string) bool {
        length := len(s)
        if length < 2 {
                return true
        }
        if s[0] == s[length-1] {
                return validPalindrome(s[1 : length-1])
        }
        return isPalindrome(s[0:length-1]) || isPalindrome(s[1:length])
}

func isPalindrome(s string) bool {
        i := 0
        j := len(s) - 1
        for i < j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}
```

77.8 剑指 OfferII023. 两个链表的第一个重合节点 (4)

- 题目

```
给定两个单链表的头节点headA 和 headB␣
↪，请找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 null 。
图示两个链表在节点 c1 开始相交：
题目数据 保证 整个链式结构中不存在环。
注意，函数返回结果后，链表必须 保持其原始结构 。
示例 1：输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2,
↪ skipB = 3
输出：Intersected at '8'
解释：相交节点的值为 8 （注意，如果两个链表相交则不能为 0）。
```

(续下页)

(接上页)

```
从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
示例2：输入：intersectVal= 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB␣
↪= 1
输出：Intersected at '2'
解释：相交节点的值为 2 （注意，如果两个链表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。
在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。
示例3：输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2␣
↪输出：null
解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。
由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
这两个链表不相交，因此返回 null 。
提示：listA 中节点数目为 m
listB 中节点数目为 n
0 <= m, n <= 3 * 104
1 <= Node.val <= 105
0 <= skipA <= m
0 <= skipB <= n
如果 listA 和 listB 没有交点，intersectVal 为 0
如果 listA 和 listB 有交点，intersectVal == listA[skipA + 1] == listB[skipB + 1]
进阶：能否设计一个时间复杂度 O(n) 、仅用 O(1) 内存的解决方案？
注意：本题与主站 160题相同
```

- 解题思路

```
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        ALength := 0
        A := headA
        for A != nil {
                ALength++
                A = A.Next
        }
        BLength := 0
        B := headB
        for B != nil {
                BLength++
                B = B.Next
        }

        pA := headA
        pB := headB
        if ALength > BLength {
                n := ALength - BLength
```

(续下页)

(接上页)

```
                for n > 0 {
                        pA = pA.Next
                        n--
                }
        } else {
                n := BLength - ALength
                for n > 0 {
                        pB = pB.Next
                        n--
                }
        }

        for pA != pB {
                pA = pA.Next
                pB = pB.Next
        }
        return pA
}

# 2
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        A, B := headA, headB
        for A != B {
                if A != nil {
                        A = A.Next
                } else {
                        A = headB
                }
                if B != nil {
                        B = B.Next
                } else {
                        B = headA
                }
        }
        return A
}

# 3
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        A, B := headA, headB
        for A != nil {
                for B != nil {
                        if A == B {
```

(续下页)

(接上页)

```
                                return A
                        }
                        B = B.Next
                }
                A = A.Next
                B = headB
        }
        return nil
}

# 4
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for headA != nil {
                m[headA] = true
                headA = headA.Next
        }

        for headB != nil {
                if _, ok := m[headB]; ok {
                        return headB
                }
                headB = headB.Next
        }
        return nil
}
```

77.9 剑指 OfferII024. 反转链表 (4)

- 题目

```
给定单链表的头节点 head ，请反转链表，并返回反转后的链表的头节点。
示例 1: 输入: head = [1,2,3,4,5] 输出: [5,4,3,2,1]
示例 2: 输入: head = [1,2] 输出: [2,1]
示例 3: 输入: head = [] 输出: []
提示: 链表中节点的数目范围是 [0, 5000]
-5000 <= Node.val <= 5000
进阶: 链表可以选用迭代或递归方式完成反转。你能否用两种方法解决这道题？
注意: 本题与主站 206题相同:
```

- 解题思路

```
func reverseList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }

        result := reverseList(head.Next)
        head.Next.Next = head
        head.Next = nil
        return result
}

# 2
func reverseList(head *ListNode) *ListNode {
        var result *ListNode
        var temp *ListNode
        for head != nil {
                temp = head.Next
                head.Next = result
                result = head
                head = temp
        }
        return result
}

# 3
func reverseList(head *ListNode) *ListNode {
        result := &ListNode{}
        arr := make([]*ListNode, 0)
        for head != nil {
                arr = append(arr, head)
                head = head.Next
        }
        temp := result
        for i := len(arr) - 1; i >= 0; i-- {
                arr[i].Next = nil
                temp.Next = arr[i]
                temp = temp.Next
        }
        return result.Next
}

# 4
func reverseList(head *ListNode) *ListNode {
        var res *ListNode
```

(续下页)

(接上页)

```
        for {
                if head == nil {
                        break
                }
                res = &ListNode{head.Val, res}
                head = head.Next
        }
        return res
}
```

77.10 剑指 OfferII027. 回文链表 (4)

- 题目

```
给定一个链表的 头节点head，请判断其是否为回文链表。
如果一个链表是回文，那么链表节点序列从前往后看和从后往前看是相同的。
示例 1：输入：head = [1,2,3,3,2,1] 输出：true
示例 2：输入：head = [1,2] 输出：false
提示：链表 L 的长度范围为 [1, 105]
0<= node.val <= 9
进阶：能否用O(n) 时间复杂度和 O(1) 空间复杂度解决此题？
注意：本题与主站 234题相同
```

- 解题思路

```
func isPalindrome(head *ListNode) bool {
        m := make([]int, 0)
        for head != nil {
                m = append(m, head.Val)
                head = head.Next
        }
        i, j := 0, len(m)-1
        for i < j {
                if m[i] != m[j] {
                        return false
                }
                i++
                j--
        }
        return true
}
```

(续下页)

(接上页)

```
# 2
func isPalindrome(head *ListNode) bool {
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
        }
        var pre *ListNode
        cur := slow
        for cur != nil{
                next := cur.Next
                cur.Next = pre
                pre = cur
                cur = next
        }
        for pre != nil{
                if head.Val != pre.Val{
                        return false
                }
                pre = pre.Next
                head = head.Next
        }
        return true
}

# 3
func isPalindrome(head *ListNode) bool {
        m := make([]int, 0)
        temp := head
        for temp != nil {
                m = append(m, temp.Val)
                temp = temp.Next
        }
        for head != nil {
                val := m[len(m)-1]
                m = m[:len(m)-1]
                if head.Val != val {
                        return false
                }
                head = head.Next
        }
        return true
}
```

(续下页)

(接上页)

```
# 4
var p *ListNode
func isPalindrome(head *ListNode) bool {
        if head == nil{
                return true
        }
        if p == nil{
                p = head
        }
        if isPalindrome(head.Next) && (p.Val == head.Val){
                p = p.Next
                return true
        }
        p = nil
        return false
}
```

77.11 剑指 OfferII032. 有效的变位词 (2)

- 题目

```
给定两个字符串 s 和 t ，编写一个函数来判断它们是不是一组变位词（字母异位词）。
注意：若s 和 t中每个字符出现的次数都相同且字符顺序不完全相同，则称s 和␣
↪t互为变位词（字母异位词）。
示例1:输入：s = "anagram", t = "nagaram" 输出：true
示例 2:输入：s = "rat", t = "car" 输出：false
示例 3:输入：s = "a", t = "a" 输出：false
提示:1 <= s.length, t.length <= 5 * 104
sandt仅包含小写字母
进阶:如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？
注意：本题与主站 242题相似（字母异位词定义不同）
```

- 解题思路

```
func isAnagram(s string, t string) bool {
        if len(s) != len(t) || s == t {
                return false
        }

        sr := []rune(s)
        tr := []rune(t)
```

(续下页)

(接上页)

```
        rec := make(map[rune]int, len(sr))
        for i := range sr {
                rec[sr[i]]++
                rec[tr[i]]--
        }

        for _, n := range rec {
                if n != 0 {
                        return false
                }
        }
        return true
}

# 2
func isAnagram(s string, t string) bool {
        if len(s) != len(t) || s == t {
                return false
        }
        sArr := make([]int, len(s))
        tArr := make([]int, len(t))
        for i := 0; i < len(s); i++ {
                sArr[i] = int(s[i] - 'a')
                tArr[i] = int(t[i] - 'a')
        }
        sort.Ints(sArr)
        sort.Ints(tArr)
        for i := 0; i < len(s); i++ {
                if sArr[i] != tArr[i] {
                        return false
                }
        }
        return true
}
```

77.12 剑指 OfferII034. 外星语言是否排序 (2)

- 题目

```
某种外星语也使用英文小写字母，但可能顺序 order␣
↪不同。字母表的顺序（order）是一些小写字母的排列。
给定一组用外星语书写的单词 words，以及其字母表的顺序␣
↪order，只有当给定的单词在这种外星语中按字典序排列时，返回 true；
否则，返回 false。
示例 1：输入：words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"␣
↪输出：true
解释：在该语言的字母表中，'h' 位于 'l' 之前，所以单词序列是按字典序排列的。
示例 2：输入：words = ["word","world","row"], order = "worldabcefghijkmnpqstuvxyz"␣
↪输出：false
解释：在该语言的字母表中，'d' 位于 'l' 之后，那么 words[0] >␣
↪words[1]，因此单词序列不是按字典序排列的。
示例 3：输入：words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"␣
↪输出：false
解释：当前三个字符 "app" 匹配时，第二个字符串相对短一些，然后根据词典编纂规则 "apple"␣
↪> "app",
因为 'l' > 'ø'，其中 'ø' 是空白字符，定义为比任何其他字符都小（更多信息）。
提示：1 <= words.length <= 100
1 <= words[i].length <= 20
order.length == 26
在words[i]和order中的所有字符都是英文小写字母。
注意：本题与主站 953题相同：
```

- 解题思路

```
func isAlienSorted(words []string, order string) bool {
        newWords := make([]string, len(words))
        m := make(map[byte]int)
        for i := 0; i < len(order); i++ {
                m[order[i]] = i
        }
        for i := 0; i < len(words); i++ {
                str := ""
                for j := 0; j < len(words[i]); j++ {
                        str = str + string(m[words[i][j]]+'a')
                }
                newWords[i] = str
        }
        for i := 0; i < len(newWords)-1; i++ {
                if newWords[i] > newWords[i+1] {
```

(续下页)

(接上页)

```
                        return false
                }
        }
        return true
}

#
func isAlienSorted(words []string, order string) bool {
        m := make(map[byte]int)
        for i := 0; i < len(order); i++ {
                m[order[i]] = i
        }

        for i := 0; i < len(words)-1; i++ {
                length := len(words[i])
                if len(words[i+1]) < length {
                        length = len(words[i+1])
                }
                for j := 0; j < length; j++ {
                        if m[words[i][j]] < m[words[i+1][j]] {
                                break
                        }
                        if m[words[i][j]] > m[words[i+1][j]] {
                                return false
                        }
                        if j == length-1 {
                                if len(words[i]) > len(words[i+1]) {
                                        return false
                                }
                        }
                }
        }
        return true
}
```

77.13 剑指 OfferII041. 滑动窗口的平均值 (1)

- 题目

```
给定一个整数数据流和一个窗口大小，根据该滑动窗口的大小，计算滑动窗口里所有数字的平均值。
实现 MovingAverage 类：
MovingAverage(int size) 用窗口大小 size 初始化对象。
double next(int val)成员函数 next每次调用的时候都会往滑动窗口增加一个整数，
请计算并返回数据流中最后 size 个值的移动平均值，即滑动窗口里所有数字的平均值。
示例：输入：inputs = ["MovingAverage", "next", "next", "next", "next"] inputs = [[3],␣
↪[1], [10], [3], [5]]
输出：[null, 1.0, 5.5, 4.66667, 6.0]
解释： MovingAverage movingAverage = new MovingAverage(3);
movingAverage.next(1); // 返回 1.0 = 1 / 1
movingAverage.next(10); // 返回 5.5 = (1 + 10) / 2
movingAverage.next(3); // 返回 4.66667 = (1 + 10 + 3) / 3
movingAverage.next(5); // 返回 6.0 = (10 + 3 + 5) / 3
提示：1 <= size <= 1000
-105 <= val <= 105
最多调用 next 方法 104 次
注意：本题与主站 346题相同：
```

- 解题思路

```
type MovingAverage struct {
        sum  int
        size int
        arr  []int
}

func Constructor(size int) MovingAverage {
        return MovingAverage{
                sum:  0,
                size: size,
                arr:  make([]int, 0),
        }
}

func (this *MovingAverage) Next(val int) float64 {
        this.sum = this.sum + val
        this.arr = append(this.arr, val)
        if len(this.arr) > this.size {
                this.sum = this.sum - this.arr[0]
                this.arr = this.arr[1:]
```

(续下页)

(接上页)

```
        }
        return float64(this.sum) / float64(len(this.arr))
}
```

77.14 剑指 OfferII042. 最近请求次数 (2)

- 题目

```
写一个RecentCounter类来计算特定时间范围内最近的请求。
请实现 RecentCounter 类：
RecentCounter() 初始化计数器，请求数为 0 。
int ping(int t) 在时间 t 添加一个新请求，其中 t 表示以毫秒为单位的某个时间，
并返回过去 3000 毫秒内发生的所有请求数（包括新请求）。确切地说，返回在 [t-3000, t]␣
↪内发生的请求数。
保证 每次对 ping 的调用都使用比之前更大的 t 值。
示例：输入： inputs = ["RecentCounter", "ping", "ping", "ping", "ping"]
inputs = [[], [1], [100], [3001], [3002]]
输出：[null, 1, 2, 3, 3]
解释：RecentCounter recentCounter = new RecentCounter();
recentCounter.ping(1);     // requests = [1]，范围是 [-2999,1]，返回 1
recentCounter.ping(100);   // requests = [1, 100]，范围是 [-2900,100]，返回 2
recentCounter.ping(3001);  // requests = [1, 100, 3001]，范围是 [1,3001]，返回 3
recentCounter.ping(3002);  // requests = [1, 100, 3001, 3002]，范围是 [2,3002]，返回 3
提示：1 <= t <= 109
保证每次对 ping 调用所使用的 t 值都 严格递增
至多调用 ping 方法 104 次
注意：本题与主站 933题相同：
```

- 解题思路

```
type RecentCounter struct {
        arr []int
}

func Constructor() RecentCounter {
        return RecentCounter{
                arr: make([]int, 0),
        }
}

func (r *RecentCounter) Ping(t int) int {
        r.arr = append(r.arr, t)
```

(续下页)

(接上页)

```
        res := 1
        for i := len(r.arr) - 2; i >= 0; i-- {
                if t-r.arr[i] <= 3000 {
                        res++
                } else {
                        r.arr = r.arr[i+1:]
                        break
                }
        }
        return res
}

#
type RecentCounter struct {
        arr []int
}

func Constructor() RecentCounter {
        return RecentCounter{
                arr: make([]int, 0),
        }
}

func (r *RecentCounter) Ping(t int) int {
        r.arr = append(r.arr, t)
        start := t - 3000
        for len(r.arr) > 0 && r.arr[0] < start {
                r.arr = r.arr[1:]
        }
        return len(r.arr)
}
```

77.15 剑指 OfferII052. 展平二叉搜索树 (3)

- 题目

```
给你一棵二叉搜索树，请按中序遍历 将其重新排列为一棵递增顺序搜索树，
使树中最左边的节点成为树的根节点，并且每个节点没有左子节点，只有一个右子节点。
示例 1：输入：root = [5,3,6,2,4,null,8,1,null,null,null,7,9]
输出：[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]
示例 2：输入：root = [5,1,7] 输出：[1,null,5,null,7]
提示：树中节点数的取值范围是 [1, 100]
```

(续下页)

(接上页)

```
0 <= Node.val <= 1000
注意：本题与主站 897题相同：
```

- 解题思路

```
func increasingBST(root *TreeNode) *TreeNode {
        arr := make([]int, 0)
        dfs(root, &arr)
        if len(arr) == 0 {
                return root
        }
        newRoot := &TreeNode{Val: arr[0]}
        cur := newRoot
        for i := 1; i < len(arr); i++ {
                cur.Right = &TreeNode{Val: arr[i]}
                cur = cur.Right
        }
        return newRoot
}

func dfs(node *TreeNode, arr *[]int) {
        if node == nil {
                return
        }
        dfs(node.Left, arr)
        *arr = append(*arr, node.Val)
        dfs(node.Right, arr)
}

# 2
var prev *TreeNode

func increasingBST(root *TreeNode) *TreeNode {
        prev = &TreeNode{}
        head := prev
        dfs(root)
        return head.Right
}

func dfs(node *TreeNode) {
        if node == nil {
                return
        }
        dfs(node.Left)
```

(续下页)

(接上页)

```
        node.Left = nil
        prev.Right = node
        prev = node
        dfs(node.Right)
}

# 3
func increasingBST(root *TreeNode) *TreeNode {
        stack := make([]*TreeNode, 0)
        newRoot := &TreeNode{}
        stack = append(stack, root)
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                if node.Right != nil {
                        stack = append(stack, node.Right)
                        node.Right = nil
                        continue
                }
                stack = stack[:len(stack)-1]
                node.Right = newRoot.Right
                newRoot.Right = node
                if node.Left != nil {
                        stack = append(stack, node.Left)
                        node.Left = nil
                }
        }
        return newRoot.Right
}
```

77.16 剑指 OfferII056. 二叉搜索树中两个节点之和 (4)

- 题目

```
给定一个二叉搜索树的 根节点 root和一个整数 k ,␣
↪请判断该二叉搜索树中是否存在两个节点它们的值之和等于 k 。
假设二叉搜索树中节点的值均唯一。
示例 1: 输入: root = [8,6,10,5,7,9,11], k = 12 输出: true
解释: 节点 5 和节点 7 之和等于 12
示例 2: 输入: root = [8,6,10,5,7,9,11], k = 22 输出: false
解释: 不存在两个节点值之和为 22 的节点
提示: 二叉树的节点个数的范围是[1, 104].
-104<= Node.val <= 104
```

(续下页)

(接上页)

```
root为二叉搜索树
-105<= k <= 105
注意：本题与主站 653 题相同：
```

- 解题思路

```
func findTarget(root *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        m := map[int]int{}
        return dfs(root, k, m)
}

func dfs(node *TreeNode, k int, m map[int]int) bool {
        if node == nil {
                return false
        }
        if _, ok := m[k-node.Val]; ok {
                return true
        }
        m[node.Val] = node.Val
        return dfs(node.Left, k, m) || dfs(node.Right, k, m)
}

# 2
func dfs(root, searchRoot *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        found := findNode(searchRoot, k-root.Val)
        if found != nil && found != root {
                return true
        }
        return dfs(root.Left, searchRoot, k) ||
                dfs(root.Right, searchRoot, k)
}

func findNode(root *TreeNode, target int) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Val == target {
                return root
```

(续下页)

(接上页)

```
        }
        if root.Val < target {
                return findNode(root.Right, target)
        }
        return findNode(root.Left, target)
}

# 3
func findTarget(root *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        m := make(map[int]int)
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[len(queue)-1]
                queue = queue[:len(queue)-1]
                if _, ok := m[k-node.Val]; ok {
                        return true
                }
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
                m[node.Val] = 1
        }
        return false
}

# 4
var arr []int

func findTarget(root *TreeNode, k int) bool {
        if root == nil {
                return false
        }
        arr = make([]int, 0)
        dfs(root)
        i := 0
        j := len(arr) - 1
```

(续下页)

(接上页)

```
        for i < j {
                if arr[i]+arr[j] == k {
                        return true
                } else if arr[i]+arr[j] > k {
                        j--
                } else {
                        i++
                }
        }
        return false
}

func dfs(node *TreeNode) {
        if node == nil {
                return
        }
        dfs(node.Left)
        arr = append(arr, node.Val)
        dfs(node.Right)
}
```

77.17 剑指 OfferII059. 数据流的第 K 大数值 (2)

- 题目

```
设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k␣
↪个不同的元素。
请实现 KthLargest类：
KthLargest(int k, int[] nums) 使用整数 k 和整数流 nums 初始化对象。
int add(int val) 将 val 插入数据流 nums 后，返回当前数据流中第 k 大的元素。
示例：输入： ["KthLargest", "add", "add", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
输出：[null, 4, 5, 5, 8, 8]
解释：KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);
kthLargest.add(3);   // return 4
kthLargest.add(5);   // return 5
kthLargest.add(10);  // return 5
kthLargest.add(9);   // return 8
kthLargest.add(4);   // return 8
提示：1 <= k <= 104
0 <= nums.length <= 104
-104 <= nums[i] <= 104
```

(续下页)

(接上页)

```
-104 <= val <= 104
最多调用 add 方法 104 次
题目数据保证，在查找第 k 大元素时，数组中至少有 k 个元素
注意：本题与主站 703题相同：
```

- 解题思路

```
type KthLargest struct {
        k    int
        heap intHeap
}

func Constructor(k int, nums []int) KthLargest {
        h := intHeap(nums)
        heap.Init(&h)

        for len(h) > k {
                heap.Pop(&h)
        }
        return KthLargest{
                k:    k,
                heap: h,
        }
}

func (k *KthLargest) Add(val int) int {
        heap.Push(&k.heap, val)
        if len(k.heap) > k.k {
                heap.Pop(&k.heap)
        }
        return k.heap[0]
}

// 内置heap，实现接口
/*
type Interface interface {
        sort.Interface
        Push(x interface{}) // add x as element Len()
        Pop() interface{}   // remove and return element Len() - 1.
}
*/
type intHeap []int

func (h intHeap) Len() int {
```

(续下页)

(接上页)

```
        return len(h)
}

func (h intHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h intHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *intHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *intHeap) Pop() interface{} {
        res := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return res
}

# 2
type KthLargest struct {
        nums []int
        k    int
}

func Constructor(k int, nums []int) KthLargest {
        if k < len(nums) {
                sort.Ints(nums)
                nums = nums[len(nums)-k:]
        }
        // 向上调整
        Up(nums)
        return KthLargest{
                nums: nums,
                k:    k,
        }
}

func (k *KthLargest) Add(val int) int {
        if k.k > len(k.nums) {
                k.nums = append(k.nums, val)
```

(续下页)

(接上页)

```
                Up(k.nums)
        } else {
                if val > k.nums[0] {
                        // 在堆顶，向下调整
                        k.nums[0] = val
                        Down(k.nums, 0)
                }
        }
        return k.nums[0]
}

func Down(nums []int, index int) {
        length := len(nums)
        minIndex := index
        for {
                left := 2*index + 1
                right := 2*index + 2
                if left < length && nums[left] < nums[minIndex] {
                        minIndex = left
                }
                if right < length && nums[right] < nums[minIndex] {
                        minIndex = right
                }
                if minIndex == index {
                        break
                }
                swap(nums, index, minIndex)
                index = minIndex
        }
}

func Up(nums []int) {
        length := len(nums)
        for i := length/2 - 1; i >= 0; i-- {
                minIndex := i
                left := 2*i + 1
                right := 2*i + 2
                if left < length && nums[left] < nums[minIndex] {
                        minIndex = left
                }
                if right < length && nums[right] < nums[minIndex] {
                        minIndex = right
                }
```

(续下页)

(接上页)

```
                if i != minIndex {
                        swap(nums, i, minIndex)
                }
        }
}

func swap(nums []int, i, j int) {
        nums[i], nums[j] = nums[j], nums[i]
}
```

77.18 剑指 OfferII068. 查找插入位置 (3)

- 题目

```
给定一个排序的整数数组 nums和一个整数目标值 target␣
↪，请在数组中找到target，并返回其下标。
如果目标值不存在于数组中，返回它将会被按顺序插入的位置。
请必须使用时间复杂度为 O(log n) 的算法。
示例 1:输入: nums = [1,3,5,6], target = 5 输出: 2
示例2:输入: nums = [1,3,5,6], target = 2 输出: 1
示例 3:输入: nums = [1,3,5,6], target = 7 输出: 4
示例 4:输入: nums = [1,3,5,6], target = 0 输出: 0
示例 5:输入: nums = [1], target = 0 输出: 0
提示:1 <= nums.length <= 104
-104 <= nums[i] <= 104
nums 为无重复元素的升序排列数组
-104 <= target <= 104
注意：本题与主站 35题相同：
```

- 解题思路

```
// 二分查找
func searchInsert(nums []int, target int) int {
        low, high := 0, len(nums)-1
        for low <= high {
                mid := (low + high) / 2
                switch {
                case nums[mid] < target:
                        low = mid + 1
                case nums[mid] > target:
                        high = mid - 1
                default:
```

(续下页)

(接上页)

```
                        return mid
                }
        }
        return low
}

// 顺序查找
func searchInsert(nums []int, target int) int {
        i := 0
        for i < len(nums) && nums[i] < target {
                if nums[i] == target {
                        return i
                }
                i++
        }
        return i
}

// 顺序查找
func searchInsert(nums []int, target int) int {
        for i := 0; i < len(nums); i++ {
                if nums[i] >= target {
                        return i
                }
        }
        return len(nums)
}
```

77.19 剑指 OfferII069. 山峰数组的顶部 (3)

- 题目

```
符合下列属性的数组 arr 称为 山峰数组（山脉数组）：
arr.length >= 3
存在 i（0 < i< arr.length - 1）使得：
arr[0] < arr[1] < ... arr[i-1] < arr[i]
arr[i] > arr[i+1] > ... > arr[arr.length - 1]
给定由整数组成的山峰数组 arr ，返回任何满足
arr[0] < arr[1] < ... arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1]␣
↪的下标 i，即山峰顶部。
示例 1：输入：arr = [0,1,0] 输出：1
示例 2：输入：arr = [1,3,5,4,2] 输出：2
```

(续下页)

(接上页)

```
示例 3: 输入: arr = [0,10,5,2] 输出: 1
示例 4: 输入: arr = [3,4,5,1] 输出: 2
示例 5: 输入: arr = [24,69,100,99,79,78,67,36,26,19] 输出: 2
提示: 3 <= arr.length <= 104
0 <= arr[i] <= 106
题目数据保证 arr 是一个山脉数组
进阶: 很容易想到时间复杂度 O(n) 的解决方案, 你可以设计一个 O(log(n)) 的解决方案吗?
注意: 本题与主站 852题相同
```

- 解题思路

```
func peakIndexInMountainArray(A []int) int {
        n := len(A)
        for i := 0; i < n-1; i++ {
                if A[i] > A[i+1] {
                        return i
                }
        }
        return 0
}

#
func peakIndexInMountainArray(A []int) int {
        left, right := 0, len(A)-1
        for {
                mid := left + (right-left)/2
                if A[mid] > A[mid+1] && A[mid] > A[mid-1] {
                        return mid
                }
                if A[mid] > A[mid-1] {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
}

# 3
func peakIndexInMountainArray(arr []int) int {
        n := len(arr)
        return sort.Search(n-1, func(i int) bool {
                return arr[i] > arr[i+1]
        })
}
```

77.20 剑指 OfferII072. 求平方根 (5)

- 题目

```
给定一个非负整数 x ，计算并返回 x 的平方根，即实现int sqrt(int x)函数。
正数的平方根有两个，只输出其中的正数平方根。
如果平方根不是整数，输出只保留整数的部分，小数部分将被舍去。
示例 1:输入: x = 4 输出: 2
示例 2:输入: x = 8 输出: 2
解释: 8 的平方根是 2.82842...，由于小数部分将被舍去，所以返回 2
提示:0 <= x <= 231- 1
注意: 本题与主站 69题相同:
```

- 解题思路

```
// 系统函数
func mySqrt(x int) int {
        result := int(math.Sqrt(float64(x)))
        return result
}

// 系统函数
func mySqrt(x int) int {
        result := math.Floor(math.Sqrt(float64(x)))
        return int(result)
}

// 牛顿迭代法
func mySqrt(x int) int {
        result := x
        for result*result > x {
                result = (result + x/result) / 2
        }
        return result
}

// 二分查找法
func mySqrt(x int) int {
        left := 1
        right := x
        for left <= right {
                mid := (left + right) / 2
                if mid == x/mid {
                        return mid
```

(续下页)

(接上页)

```
                } else if mid < x/mid {
                        left = mid + 1
                } else {
                        right = mid - 1
                }
        }
        if left * left <= x{
                return left
        }else {
                return left-1
        }
}

// 暴力法:遍历
func mySqrt(x int) int {
        result := 0
        for i := 1; i <= x/i; i++ {
                if i*i == x {
                        return i
                }
                result = i
        }
        return result
}
```

77.21 剑指 OfferII075. 数组相对排序 (3)

- 题目

```
给定两个数组，arr1 和arr2，
arr2中的元素各不相同
arr2 中的每个元素都出现在arr1中
对 arr1中的元素进行排序，使 arr1 中项的相对顺序和arr2中的相对顺序相同。
未在arr2中出现过的元素需要按照升序放在arr1的末尾。
示例：输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6] 输出：[2,2,2,1,4,3,
↪3,9,6,7,19]
提示：1 <= arr1.length, arr2.length <= 1000
0 <= arr1[i], arr2[i] <= 1000
arr2中的元素arr2[i]各不相同
arr2 中的每个元素arr2[i]都出现在arr1中
注意：本题与主站 1122题相同：
```

- 解题思路

```
func relativeSortArray(arr1 []int, arr2 []int) []int {
        if len(arr2) == 0 {
                sort.Ints(arr1)
                return arr1
        }
        res := make([]int, 0)
        m := make(map[int]int)
        for i := range arr1 {
                m[arr1[i]]++
        }
        for i := 0; i < len(arr2); i++ {
                for j := 0; j < m[arr2[i]]; j++ {
                        res = append(res, arr2[i])
                }
                m[arr2[i]] = 0
        }
        tempArr := make([]int, 0)
        for key, value := range m {
                for value > 0 {
                        tempArr = append(tempArr, key)
                        value--
                }
        }
        sort.Ints(tempArr)
        res = append(res, tempArr...)
        return res
}

# 2
func relativeSortArray(arr1 []int, arr2 []int) []int {
        count := 0
        for i := 0; i < len(arr2); i++ {
                for j := count; j < len(arr1); j++ {
                        if arr2[i] == arr1[j] {
                                arr1[count], arr1[j] = arr1[j], arr1[count]
                                count++
                        }
                }
        }
        sort.Ints(arr1[count:])
        return arr1
}

# 3
```

(续下页)

(接上页)

```
func relativeSortArray(arr1 []int, arr2 []int) []int {
        temp := make([]int, 1001)
        for i := range arr1 {
                temp[arr1[i]]++
        }
        count := 0
        for i := range arr2 {
                for temp[arr2[i]] > 0 {
                        arr1[count] = arr2[i]
                        temp[arr2[i]]--
                        count++
                }
        }
        for i := 0; i < len(temp); i++ {
                for temp[i] > 0 {
                        arr1[count] = i
                        temp[i]--
                        count++
                }
        }
        return arr1
}
```

77.22 剑指 OfferII088. 爬楼梯的最少成本 (3)

- 题目

```
数组的每个下标作为一个阶梯，第 i 个阶梯对应着一个非负数的体力花费值cost[i]（下标从 0␣
↪开始）。
每当爬上一个阶梯都要花费对应的体力值，一旦支付了相应的体力值，就可以选择向上爬一个阶梯或者爬两个阶梯
请找出达到楼层顶部的最低花费。在开始时，你可以选择从下标为 0 或 1 的元素作为初始阶梯。
示例1：输入：cost = [10, 15, 20] 输出：15
解释：最低花费是从 cost[1] 开始，然后走两步即可到阶梯顶，一共花费 15 。
示例 2：输入：cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1] 输出：6
解释：最低花费方式是从 cost[0] 开始，逐个经过那些 1 ，跳过 cost[3] ，一共花费 6 。
提示：2 <= cost.length <= 1000
0 <= cost[i] <= 999
注意：本题与主站 746题相同：
```

- 解题思路

```
/*
用dp[i]表示爬i个台阶所需要的成本，所以dp[0]=0，dp[1]=0
每次爬i个楼梯，计算的都是从倒数第一个结束，还是从倒数第二个结束
动态转移方程为:
dp[i] = min{dp[i-2]+cost[i-2] , dp[i-1]+cost[i-1]};
*/
func minCostClimbingStairs(cost []int) int {
        n := len(cost)
        dp := make([]int, n+1)
        dp[0] = 0
        dp[1] = 0
        for i := 2; i <= n; i++ {
                dp[i] = min(dp[i-1]+cost[i-1], dp[i-2]+cost[i-2])
        }
        return dp[n]
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

# 2
func minCostClimbingStairs(cost []int) int {
        a := 0
        b := 0
        for i := 2; i <= len(cost); i++ {
                a, b = b, min(b+cost[i-1], a+cost[i-2])
        }
        return b
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}

# 3
var arr []int
```

(续下页)

(接上页)

```
func minCostClimbingStairs(cost []int) int {
        arr = make([]int, len(cost)+1)
        return ClimbingStais(cost, len(cost))
}

func ClimbingStais(cost []int, i int) int {
        if i == 0 || i == 1 {
                return 0
        }
        if arr[i] == 0 {
                arr[i] = min(ClimbingStais(cost, i-1)+cost[i-1],
                        ClimbingStais(cost, i-2)+cost[i-2])
        }
        return arr[i]
}

func min(a, b int) int {
        if a < b {
                return a
        }
        return b
}
```

77.23 剑指 OfferII101. 分割等和子集 (2)

- 题目

```
给定一个非空的正整数数组 nums ，请判断能否将这些数字分成元素和相等的两部分。
示例1: 输入: nums = [1,5,11,5] 输出: true
解释: nums 可以分割成 [1, 5, 5] 和 [11] 。
示例2: 输入: nums = [1,2,3,5] 输出: false
解释: nums 不可以分为和相等的两部分
提示: 1 <= nums.length <= 200
1 <= nums[i] <= 100
注意: 本题与主站 416题相同:
```

- 解题思路

```
func canPartition(nums []int) bool {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
```

(续下页)

(接上页)

```
        }
        if sum%2 == 1 {
                return false
        }
        target := sum / 2
        // 题目转换为0-1背包问题，容量为sum/2
        dp := make([][]bool, len(nums)+1)
        for i := 0; i <= len(nums); i++ {
                dp[i] = make([]bool, target+1)
                dp[i][0] = true
        }
        for i := 1; i <= len(nums); i++ {
                for j := 1; j <= target; j++ {
                        if j-nums[i-1] < 0 {
                                dp[i][j] = dp[i-1][j]
                        } else {
                                dp[i][j] = dp[i-1][j] || dp[i-1][j-nums[i-1]]
                        }
                }
        }
        return dp[len(nums)][target]
}

# 2
func canPartition(nums []int) bool {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum%2 == 1 {
                return false
        }
        target := sum / 2
        // 题目转换为0-1背包问题，容量为sum/2
        dp := make([]bool, target+1)
        dp[0] = true
        for i := 0; i < len(nums); i++ {
                for j := target; j >= 0; j-- {
                        if j-nums[i] >= 0 && dp[j-nums[i]] == true {
                                dp[j] = true
                        }
                }
        }
```

(续下页)

(接上页)

```
        return dp[target]
}
```

CHAPTER 78

剑指 OfferII-Medium

78.1 剑指 OfferII004. 只出现一次的数字 (5)

- 题目

```
给你一个整数数组nums ，除某个元素仅出现 一次 外，其余每个元素都恰出现 三次。
↪请你找出并返回那个只出现了一次的元素。
示例 1：输入：nums = [2,2,3,2] 输出：3
示例 2：输入：nums = [0,1,0,1,0,1,100] 输出：100
提示：1 <= nums.length <= 3 * 104
-231 <= nums[i] <= 231 - 1
nums 中，除某个元素仅出现 一次 外，其余每个元素都恰出现 三次
进阶：你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？
注意：本题与主站 137题相同
```

- 解题思路

```
func singleNumber(nums []int) int {
        m := make(map[int]int)
        for _, v := range nums {
                m[v]++
        }
        for k, v := range m {
                if v == 1 {
                        return k
```

(续下页)

(接上页)

```
                }
        }
        return 0
}

# 2
func singleNumber(nums []int) int {
        sort.Ints(nums)
        for i := 0; i < len(nums)-1; i=i+3{
                if nums[i] != nums[i+1]{
                        return nums[i]
                }
        }
        return nums[len(nums)-1]
}

# 3
func singleNumber(nums []int) int {
        var res int
        for i := 0; i < 64; i++ {
                count := 0
                for j := 0; j < len(nums); j++ {
                        if (nums[j]>>i)&1 == 1 {
                                count++
                        }
                }
                res = res | ((count % 3) << i) // 哪一位出现求余后1次，该位置为1
        }
        return res
}

# 4
func singleNumber(nums []int) int {
        a, b := 0, 0
        for i := 0; i < len(nums); i++ {
                a = (a ^ nums[i]) & (^b) // a: 保留出现1次的数
                b = (b ^ nums[i]) & (^a) // b: 保留出现2次的数
        }
        return a // 最后返回只出现1次的数
}

# 5
func singleNumber(nums []int) int {
```

(续下页)

(接上页)

```
        m := make(map[int]int)
        sum := 0
        singleSum := 0
        for _, v := range nums {
                if m[v] == 0{
                        singleSum = singleSum+v
                }
                m[v] = 1
                sum = sum + v
        }
        return (singleSum*3-sum)/2
}
```

78.2 剑指 OfferII005. 单词长度的最大乘积 (2)

- 题目

```
给定一个字符串数组words，请计算当两个字符串 words[i] 和 words[j]␣
↪不包含相同字符时，它们长度的乘积的最大值。
假设字符串中只包含英语的小写字母。如果没有不包含相同字符的一对字符串，返回 0。
示例1:输入: words = ["abcw","baz","foo","bar","fxyz","abcdef"] 输出: 16
解释: 这两个单词为 "abcw", "fxyz"。它们不包含相同字符，且长度的乘积最大。
示例 2:输入: words = ["a","ab","abc","d","cd","bcd","abcd"] 输出: 4
解释: 这两个单词为 "ab", "cd"。
示例 3:输入: words = ["a","aa","aaa","aaaa"] 输出: 0
解释: 不存在这样的两个单词。
提示：2 <= words.length <= 1000
1 <= words[i].length <= 1000
words[i]仅包含小写字母
注意：本题与主站 318题相同
```

- 解题思路

```
func maxProduct(words []string) int {
        res := 0
        for i := 0; i < len(words); i++ {
                for j := i + 1; j < len(words); j++ {
                        if strings.ContainsAny(words[i], words[j]) == false &&
                                res < len(words[i])*len(words[j]) {
                                res = len(words[i]) * len(words[j])
                        }
                }
```

(续下页)

(接上页)

```
        }
        return res
}

# 2
func maxProduct(words []string) int {
        res := 0
        arr := make([]int, len(words))
        for i := 0; i < len(words); i++ {
                for _, char := range words[i] {
                        // 位或 只要有1，那么就是1
                        arr[i] = arr[i] | 1<<uint(char-'a')
                }
        }
        for i := 0; i < len(arr); i++ {
                for j := i + 1; j < len(arr); j++ {
                        if arr[i]&arr[j] == 0 && res < len(words[i])*len(words[j]) {
                                res = len(words[i]) * len(words[j])
                        }
                }
        }
        return res
}
```

78.3 剑指 OfferII007. 数组中和为 0 的三个数 (2)

- 题目

```
给定一个包含 n 个整数的数组nums，判断nums中是否存在三个元素a ，b ，c ，使得a + b + c␣
↪= 0 ？
请找出所有和为 0 且不重复的三元组。
示例 1：输入：nums = [-1,0,1,2,-1,-4] 输出：[[-1,-1,2],[-1,0,1]]
示例 2：输入：nums = [] 输出：[]
示例 3：输入：nums = [0] 出：[]
提示：0 <= nums.length <= 3000
-105 <= nums[i] <= 105
注意：本题与主站 15题相同：
```

- 解题思路

```
func threeSum(nums []int) [][]int {
        res := make([][]int, 0)
```

(续下页)

(接上页)

```
        sort.Ints(nums)
        for i := 0; i < len(nums)-1; i++ {
                target := 0 - nums[i]
                left := i + 1
                right := len(nums) - 1
                if nums[i] > 0 || nums[i]+nums[left] > 0 {
                        break
                }
                if i > 0 && nums[i] == nums[i-1] {
                        continue
                }
                for left < right {
                        if left > i+1 && nums[left] == nums[left-1] {
                                left++
                                continue
                        }
                        if right < len(nums)-2 && nums[right] == nums[right+1] {
                                right--
                                continue
                        }
                        if nums[left]+nums[right] > target {
                                right--
                        } else if nums[left]+nums[right] < target {
                                left++
                        } else {
                                res = append(res, []int{nums[i], nums[left],␣
↪nums[right]})
                                left++
                                right--
                        }
                }
        }
        return res
}

# 2
func threeSum(nums []int) [][]int {
        res := make([][]int, 0)
        m := make(map[[2]int]int)
        p := make(map[int]int)
        sort.Ints(nums)
        for k, v := range nums {
                p[v] = k
```

(续下页)

(接上页)

```
        }
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums); j++ {
                        if j != i+1 && nums[j] == nums[j-1] {
                                continue
                        }
                        sum := nums[i] + nums[j]
                        if sum > 0 {
                                break
                        }
                        if value, ok := p[-sum]; ok && value > j {
                                if _, ok2 := m[[2]int{nums[i], nums[j]}]; !ok2 {
                                        res = append(res, []int{nums[i], nums[j], 0 -␣
↪nums[i] - nums[j]})
                                        m[[2]int{nums[i], nums[j]}] = 1
                                }
                        }
                }
        }
        return res
}
```

78.4 剑指 OfferII008. 和大于等于 target 的最短子数组 (3)

- 题目

```
给定一个含有n个正整数的数组和一个正整数 target 。
找出该数组中满足其和 ≥ target 的长度最小的 连续子数组[numsl, numsl+1, ..., numsr-1,␣
↪numsr] ，并返回其长度。
如果不存在符合条件的子数组，返回 0 。
示例 1：输入：target = 7, nums = [2,3,1,2,4,3] 输出：2
解释：子数组[4,3]是该条件下的长度最小的子数组。
示例 2：输入：target = 4, nums = [1,4,4] 输出：1
示例 3：输入：target = 11, nums = [1,1,1,1,1,1,1,1] 输出：0
提示：1 <= target <= 109
1 <= nums.length <= 105
1 <= nums[i] <= 105
进阶：如果你已经实现 O(n) 时间复杂度的解法，请尝试设计一个 O(n log(n))␣
↪时间复杂度的解法。
注意：本题与主站 209题相同
```

- 解题思路

```
func minSubArrayLen(target int, nums []int) int {
        res := math.MaxInt32
        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum = sum + nums[j]
                        if sum >= target {
                                if res > j-i+1 {
                                        res = j - i + 1
                                }
                                break
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}

# 2
func minSubArrayLen(target int, nums []int) int {
        res := math.MaxInt32
        arr := make([]int, len(nums)+1)
        for i := 1; i <= len(nums); i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        for i := 1; i <= len(nums); i++ {
                target := target + arr[i-1]
                index := sort.SearchInts(arr, target)
                if index <= len(nums) {
                        if res > index-i+1 {
                                res = index - i + 1
                        }
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}

# 3
func minSubArrayLen(s int, nums []int) int {
```

(续下页)

(接上页)

```
        res := math.MaxInt32
        i, j := 0, 0
        sum := 0
        for ; j < len(nums); j++ {
                sum = sum + nums[j]
                for sum >= s {
                        if res > j-i+1 {
                                res = j - i + 1
                        }
                        sum = sum - nums[i]
                        i++
                }
        }
        if res == math.MaxInt32 {
                return 0
        }
        return res
}
```

78.5 剑指 OfferII009. 乘积小于 K 的子数组 (1)

- 题目

```
给定一个正整数数组nums和整数 k，请找出该数组内乘积小于k的连续的子数组的个数。
示例 1:输入: nums = [10,5,2,6], k = 100 输出: 8
解释: 8 个乘积小于 100 的子数组分别为: [10], [5], [2], [6], [10,5], [5,2], [2,6], [5,
↪2,6]。
需要注意的是 [10,5,2] 并不是乘积小于100的子数组。
示例 2:输入: nums = [1,2,3], k = 0 输出: 0
提示:1 <= nums.length <= 3 * 104
1 <= nums[i] <= 1000
0 <= k <= 106
注意: 本题与主站 713题相同
```

- 解题思路

```
func numSubarrayProductLessThanK(nums []int, k int) int {
        if k <= 1 {
                return 0
        }
        res := 0
        left := 0
```

(续下页)

(接上页)

```
        total := 1
        for right := 0; right < len(nums); right++ {
                total = total * nums[right]
                for k <= total {
                        total = total / nums[left]
                        left++
                }
                res = res + right - left + 1
        }
        return res
}
```

78.6 剑指 OfferII010. 和为 k 的子数组 (4)

- 题目

```
给定一个整数数组和一个整数k ，请找到该数组中和为k的连续子数组的个数。
示例 1 :输入:nums = [1,1,1], k = 2 输出: 2
解释: 此题 [1,1] 与 [1,1] 为两种不同的情况
示例 2:输入:nums = [1,2,3], k = 3 输出: 2
提示:1 <= nums.length <= 2 * 104
-1000 <= nums[i] <= 1000
-107<= k <= 107
注意: 本题与主站 560题相同
```

- 解题思路

```
func subarraySum(nums []int, k int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum = sum + nums[j]
                        if sum == k {
                                res++
                        }
                }
        }
        return res
}

# 2
```

(续下页)

(接上页)

```
func subarraySum(nums []int, k int) int {
        if len(nums) == 0 {
                return 0
        }
        res := 0
        arr := make([]int, len(nums)+1)
        arr[0] = 0
        for i := 1; i <= len(nums); i++ {
                arr[i] = arr[i-1] + nums[i-1]
        }
        for i := 0; i <= len(nums); i++ {
                for j := 0; j < i; j++ {
                        if arr[i]-arr[j] == k {
                                res++
                        }
                }
        }
        return res
}

# 3
func subarraySum(nums []int, k int) int {
        res := 0
        m := make(map[int]int)
        m[0] = 1 // 保证第一个k的存在
        sum := 0
        // sum[i:j]= sum[0:j]-sum[0:i], 把sum[i:j]设为k,
        // 于是可以转化为sum[0:j]-k=sum[0:i]
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if _, ok := m[sum-k]; ok {
                        res = res + m[sum-k]
                }
                m[sum]++
        }
        return res
}

# 4
func subarraySum(nums []int, k int) int {
        res := 0
        m := make(map[int][]int)
        m[0] = []int{-1} // 保证第一个k的存在
```

(续下页)

(接上页)

```
        sum := 0
        // sum[i:j]= sum[0:j]-sum[0:i], 把sum[i:j]设为k,
        // 于是可以转化为sum[0:j]-k=sum[0:i]
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if _, ok := m[sum-k]; ok {
                        res = res + len(m[sum-k])
            // 输出满足条件的子数组下标
                        // for _, v := range m[sum-k] {
                        //         fmt.Println(v+1, i)
                        // }
                }
                m[sum] = append(m[sum], i)
        }
        return res
}
```

78.7 剑指 OfferII011.0 和 1 个数相同的子数组 (1)

- 题目

```
给定一个二进制数组 nums , 找到含有相同数量的 0 和 1␣
↪的最长连续子数组，并返回该子数组的长度。
示例 1:输入: nums = [0,1] 输出: 2
说明: [0, 1] 是具有相同数量 0 和 1 的最长连续子数组。
示例 2:输入: nums = [0,1,0] 输出: 2
说明: [0, 1] (或 [1, 0]) 是具有相同数量 0 和 1 的最长连续子数组。
提示: 1 <= nums.length <= 105
nums[i] 不是 0 就是 1
注意: 本题与主站 525题相同
```

- 解题思路

```
func findMaxLength(nums []int) int {
        res := 0
        m := make(map[int]int)
        m[0] = -1
        total := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == 0 {
                        total--
                } else {
```

(续下页)

(接上页)

```
                        total++
                }
                if first, ok := m[total]; !ok {
                        m[total] = i
                } else {
                        if i-first > res {
                                res = i - first
                        }
                }
        }
        return res
}
```

78.8 剑指 OfferII013. 二维子矩阵的和 (1)

- 题目

```
给定一个二维矩阵 matrix，以下类型的多个请求：
计算其子矩形范围内元素的总和，该子矩阵的左上角为 (row1,col1) ，右下角为 (row2,col2) 。
实现 NumMatrix 类：NumMatrix(int[][] matrix)给定整数矩阵 matrix 进行初始化
int sumRegion(int row1, int col1, int row2, int col2)返回左上角 (row1,col1)、
右下角(row2,col2)的子矩阵的元素总和。
示例 1：输入：  ["NumMatrix","sumRegion","sumRegion","sumRegion"]
[[[[3,0,1,4,2],[5,6,3,2,1],[1,2,0,1,5],[4,1,0,1,7],[1,0,3,0,5]]],[2,1,4,3],[1,1,2,2],
↪[1,2,2,4]]
输出：[null, 8, 11, 12]
解释:NumMatrix numMatrix = new NumMatrix([[3,0,1,4,2],[5,6,3,2,1],[1,2,0,1,5],[4,1,0,
↪1,7],[1,0,3,0,5]]]);
numMatrix.sumRegion(2, 1, 4, 3); // return 8 (红色矩形框的元素总和)
numMatrix.sumRegion(1, 1, 2, 2); // return 11 (绿色矩形框的元素总和)
numMatrix.sumRegion(1, 2, 2, 4); // return 12 (蓝色矩形框的元素总和)
提示：m == matrix.length
n == matrix[i].length
1 <= m,n <=200
-105<= matrix[i][j] <= 105
0 <= row1 <= row2 < m
0 <= col1 <= col2 < n
最多调用 104 次sumRegion 方法
注意：本题与主站 304题相同：
```

- 解题思路

```
type NumMatrix struct {
        arr [][]int
}

func Constructor(matrix [][]int) NumMatrix {
        if matrix == nil || len(matrix) == 0 || matrix[0] == nil || len(matrix[0]) ==␣
↪0 {
                arr := make([][]int, 1)
                for i := 0; i < 1; i++ {
                        arr[i] = make([]int, 1)
                }
                return NumMatrix{arr: arr}
        }
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n+1)
        for i := 0; i < n+1; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        arr[i][j] = arr[i][j-1] + arr[i-1][j] - arr[i-1][j-1] +␣
↪matrix[i-1][j-1]
                }
        }
        return NumMatrix{arr: arr}
}

func (this *NumMatrix) SumRegion(row1 int, col1 int, row2 int, col2 int) int {
        return this.arr[row2+1][col2+1] - this.arr[row2+1][col1] - this.
↪arr[row1][col2+1] + this.arr[row1][col1]
}
```

78.9 剑指 OfferII014. 字符串中的变位词 (2)

- 题目

```
给定两个字符串s1和s2，写一个函数来判断 s2 是否包含 s1的某个变位词。
换句话说，第一个字符串的排列之一是第二个字符串的 子串 。
示例 1：输入：s1 = "ab" s2 = "eidbaooo" 输出：True
解释：s2 包含 s1 的排列之一 ("ba").
示例 2：输入：s1= "ab" s2 = "eidboaoo" 输出：False
提示：1 <= s1.length, s2.length <= 104
```

(续下页)

(接上页)

```
s1 和 s2 仅包含小写字母
注意：本题与主站 567题相同
```

- 解题思路

```
func checkInclusion(s1 string, s2 string) bool {
        if len(s1) > len(s2) {
                return false
        }
        arr1, arr2 := [26]int{}, [26]int{}
        for i := 0; i < len(s1); i++ {
                arr1[s1[i]-'a']++
                arr2[s2[i]-'a']++
        }
        for i := 0; i < len(s2)-len(s1); i++ {
                if arr1 == arr2 {
                        return true
                }
                arr2[s2[i]-'a']--
                arr2[s2[i+len(s1)]-'a']++
        }
        return arr1 == arr2
}

# 2
func checkInclusion(s1 string, s2 string) bool {
        if len(s1) > len(s2) {
                return false
        }
        m1, m2 := make(map[byte]int), make(map[byte]int)
        for i := 0; i < len(s1); i++ {
                m1[s1[i]-'a']++
                m2[s2[i]-'a']++
        }
        for i := 0; i < len(s2)-len(s1); i++ {
                if compare(m1, m2) {
                        return true
                }
                m2[s2[i]-'a']--
                if m2[s2[i]-'a'] == 0 {
                        delete(m2, s2[i]-'a')
                }
                m2[s2[i+len(s1)]-'a']++
        }
```

(续下页)

(接上页)

```
        return compare(m1, m2)
}

func compare(m1, m2 map[byte]int) bool {
        if len(m1) != len(m2) {
                return false
        }
        for k := range m1 {
                if m2[k] != m1[k] {
                        return false
                }
        }
        return true
}
```

78.10 剑指 OfferII015. 字符串中的所有变位词 (2)

- 题目

```
给定两个字符串s和p，找到s中所有 p␣
↪的变位词的子串，返回这些子串的起始索引。不考虑答案输出的顺序。
变位词 指字母相同，但排列不同的字符串。
示例1:输入: s = "cbaebabacd", p = "abc" 输出: [0,6]
解释:起始索引等于 0 的子串是 "cba", 它是 "abc" 的变位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的变位词。
示例 2:输入: s = "abab", p = "ab" 输出: [0,1,2]
解释:起始索引等于 0 的子串是 "ab", 它是 "ab" 的变位词。
起始索引等于 1 的子串是 "ba", 它是 "ab" 的变位词。
起始索引等于 2 的子串是 "ab", 它是 "ab" 的变位词。
提示:1 <= s.length, p.length <= 3 * 104
s和 p 仅包含小写字母
注意: 本题与主站 438题相同:
```

- 解题思路

```
func findAnagrams(s string, p string) []int {
        res := make([]int, 0)
        if len(p) > len(s) {
                return res
        }
        arr1, arr2 := [26]int{}, [26]int{}
        for i := 0; i < len(p); i++ {
```

(续下页)

(接上页)

```
                arr1[p[i]-'a']++
                arr2[s[i]-'a']++
        }
        for i := 0; i < len(s)-len(p); i++ {
                if arr1 == arr2 {
                        res = append(res, i)
                }
                arr2[s[i]-'a']--
                arr2[s[i+len(p)]-'a']++
        }
        if arr1 == arr2 {
                res = append(res, len(s)-len(p))
        }
        return res
}

# 2
func findAnagrams(s string, p string) []int {
        res := make([]int, 0)
        if len(p) > len(s) {
                return res
        }
        m1, m2 := make(map[byte]int), make(map[byte]int)
        for i := 0; i < len(p); i++ {
                m1[p[i]-'a']++
                m2[s[i]-'a']++
        }
        for i := 0; i < len(s)-len(p); i++ {
                if compare(m1, m2) {
                        res = append(res, i)
                }
                m2[s[i]-'a']--
                if m2[s[i]-'a'] == 0 {
                        delete(m2, s[i]-'a')
                }
                m2[s[i+len(p)]-'a']++
        }
        if compare(m1, m2) {
                res = append(res, len(s)-len(p))
        }
        return res
}
```

(续下页)

(接上页)

```
func compare(m1, m2 map[byte]int) bool {
        if len(m1) != len(m2) {
                return false
        }
        for k := range m1 {
                if m2[k] != m1[k] {
                        return false
                }
        }
        return true
}
```

78.11 剑指 OfferII016. 不含重复字符的最长子字符串 (5)

- 题目

```
给定一个字符串 s ，请你找出其中不含有重复字符的最长连续子字符串的长度。
示例1:输入: s = "abcabcbb" 输出: 3
解释: 因为无重复字符的最长子字符串是 "abc", 所以其长度为 3。
示例 2:输入: s = "bbbbb" 输出: 1
解释: 因为无重复字符的最长子字符串是 "b", 所以其长度为 1。
示例 3:输入: s = "pwwkew" 输出: 3
解释: 因为无重复字符的最长子串是"wke", 所以其长度为 3。
    请注意，你的答案必须是 子串 的长度，"pwke"是一个子序列，不是子串。
示例 4:输入: s = "" 输出: 0
提示: 0 <= s.length <= 5 * 104
s由英文字母、数字、符号和空格组成
注意: 本题与主站 3题相同:
```

- 解题思路

```
func lengthOfLongestSubstring(s string) int {
        arr := [256]int{}
        for i := range arr {
                arr[i] = -1
        }
        max, j := 0, 0
        for i := 0; i < len(s); i++ {
                if arr[s[i]] >= j {
                        j = arr[s[i]] + 1
                } else if i+1-j > max {
                        max = i + 1 - j
```

(续下页)

(接上页)

```
                }
                arr[s[i]] = i
        }
        return max
}

# 2
func lengthOfLongestSubstring(s string) int {
        max, j := 0, 0
        for i := 0; i < len(s); i++ {
                index := strings.Index(s[j:i], string(s[i]))
                if index == -1 {
                        continue
                }
                if i-j > max {
                        max = i - j
                }
                j = j + index + 1
        }
        if len(s)-j > max {
                max = len(s) - j
        }
        return max
}

# 3
func lengthOfLongestSubstring(s string) int {
        m := make(map[uint8]int)
        max, j := 0, 0
        for i := 0; i < len(s); i++ {
                if v, ok := m[s[i]]; ok && v >= j {
                        j = v + 1
                } else if i+1-j > max {
                        max = i + 1 - j
                }
                m[s[i]] = i
        }
        return max
}

# 4
func lengthOfLongestSubstring(s string) int {
        if len(s) < 1 {
```

(续下页)

(接上页)

```
                return 0
        }
        dp := make([]int, len(s))
        dp[0] = 1
        res := 1
        m := make(map[byte]int)
        m[s[0]] = 0
        for i := 1; i < len(s); i++ {
                index := -1
                if value, ok := m[s[i]]; ok {
                        index = value
                }
                if i-index > dp[i-1] {
                        dp[i] = dp[i-1] + 1
                } else {
                        dp[i] = i - index
                }
                m[s[i]] = i
                if dp[i] > res {
                        res = dp[i]
                }
        }
        return res
}

# 5
func lengthOfLongestSubstring(s string) int {
        arr := [256]int{}
        for i := range arr {
                arr[i] = -1
        }
        res, j := 0, -1
        for i := 0; i < len(s); i++ {
                if arr[s[i]] > j { // 出现重复了，更新下标
                        j = arr[s[i]]
                } else {
                        res = max(res, i-j) // 没有重复，更新长度
                }
                arr[s[i]] = i
        }
        return res
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

78.12 剑指 OfferII020. 回文子字符串的个数 (5)

- 题目

```
给定一个字符串 s ，请计算这个字符串中有多少个回文子字符串。
具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。
示例 1: 输入: s = "abc" 输出: 3
解释: 三个回文子串: "a", "b", "c"
示例 2: 输入: s = "aaa" 输出: 6
解释: 6个回文子串: "a", "a", "a", "aa", "aa", "aaa"
提示: 1 <= s.length <= 1000
s 由小写英文字母组成
注意: 本题与主站 647 题相同:
```

- 解题思路

```
func countSubstrings(s string) int {
        n := len(s)
        res := 0
        for i := 0; i < 2*n-1; i++ {
                left, right := i/2, i/2+i%2
                for ; 0 <= left && right < n && s[left] == s[right]; left, right =␣
↪left-1, right+1 {
                        res++
                }
        }
        return res
}

# 2
func countSubstrings(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        str := add(s)
```

(续下页)

(接上页)

```
        length := len(str)
        res := 0
        for i := 0; i < length; i++ {
                curLength := search(str, i)
                res = res + curLength/2 + curLength%2
        }
        return res
}

func add(s string) string {
        var res []rune
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        return string(res)
}

func search(s string, center int) int {
        i := center - 1
        j := center + 1
        step := 0
        for ; i >= 0 && j < len(s) && s[i] == s[j]; i, j = i-1, j+1 {
                step++
        }
        return step
}

# 3
func countSubstrings(s string) int {
        var res []rune
        res = append(res, '$')
        for _, v := range s {
                res = append(res, '#')
                res = append(res, v)
        }
        res = append(res, '#')
        res = append(res, '!')
        str := string(res)
        n := len(str) - 1
        arr := make([]int, n)
        leftMax, rightMax, result := 0, 0, 0
```

(续下页)

(接上页)

```
        for i := 1; i < n; i++ {
                if i <= rightMax {
                        arr[i] = min(rightMax-i+1, arr[2*leftMax-i])
                } else {
                        arr[i] = 1
                }
                for str[i+arr[i]] == str[i-arr[i]] {
                        arr[i]++
                }
                if i+arr[i]-1 > rightMax {
                        leftMax = i
                        rightMax = i + arr[i] - 1
                }
                result = result + arr[i]/2
        }
        return result
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func countSubstrings(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        dp := make([][]bool, len(s))
        res := 0
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
                dp[r][r] = true
                res++
                for l := 0; l < r; l++ {
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
                        }
                        if dp[l][r] == true {
```

(续下页)

(接上页)

```
                                res++
                        }
                }
        }
        return res
}

# 5
func countSubstrings(s string) int {
        if len(s) <= 1 {
                return len(s)
        }
        res := len(s)
        for i := 0; i < len(s)-1; i++ {
                for j := i + 1; j < len(s); j++ {
                        if s[i] == s[j] && judge(s, i, j) == true {
                                res++
                        }
                }
        }
        return res
}

func judge(s string, i, j int) bool {
        for i <= j {
                if s[i] != s[j] {
                        return false
                }
                i++
                j--
        }
        return true
}
```

78.13 剑指 OfferII021. 删除链表的倒数第 n 个结点 (3)

- 题目

```
给定一个链表，删除链表的倒数第n个结点，并且返回链表的头结点。
示例 1: 输入: head = [1,2,3,4,5], n = 2 输出: [1,2,3,5]
示例 2: 输入: head = [1], n = 1 输出: []
示例 3: 输入: head = [1,2], n = 1 输出: [1]
```

(续下页)

(接上页)

```
提示：链表中结点的数目为 sz
1 <= sz <= 30
0 <= Node.val <= 100
1 <= n <= sz
进阶：能尝试使用一趟扫描实现吗？
注意：本题与主站 19题相同
```

- 解题思路

```
func removeNthFromEnd(head *ListNode, n int) *ListNode {
        temp := &ListNode{Next: head}
        cur := temp
        total := 0
        for cur.Next != nil {
                cur = cur.Next
                total++
        }
        cur = temp
        count := 0
        for cur.Next != nil {
                if total-n == count {
                        cur.Next = cur.Next.Next
                        break
                }
                cur = cur.Next
                count++
        }
        return temp.Next
}

# 2
func removeNthFromEnd(head *ListNode, n int) *ListNode {
        temp := &ListNode{Next: head}
        fast, slow := temp, temp
        for i := 0; i < n; i++ {
                fast = fast.Next
        }
        for fast.Next != nil {
                fast = fast.Next
                slow = slow.Next
        }
        slow.Next = slow.Next.Next
        return temp.Next
}
```

(续下页)

(接上页)

```
# 3
var count int

func removeNthFromEnd(head *ListNode, n int) *ListNode {
        if head == nil {
                count = 0
                return nil
        }
        head.Next = removeNthFromEnd(head.Next, n)
        count = count + 1
        if count == n {
                return head.Next
        }
        return head
}
```

78.14 剑指 OfferII022. 链表中环的入口节点 (3)

- 题目

```
给定一个链表，返回链表开始入环的第一个节点。 从链表的头节点开始沿着 next␣
↪指针进入环的第一个节点为环的入口节点。
如果链表无环，则返回null。
为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0␣
↪开始）。
如果 pos 是 -1，则在该链表中没有环。注意，pos␣
↪仅仅是用于标识环的情况，并不会作为参数传递到函数中。
说明：不允许修改给定的链表。
示例 1：输入：head = [3,2,0,-4], pos = 1 输出：返回索引为 1 的链表节点
解释：链表中有一个环，其尾部连接到第二个节点。
示例2：输入：head = [1,2], pos = 0 输出：返回索引为 0 的链表节点
解释：链表中有一个环，其尾部连接到第一个节点。
示例 3：输入：head = [1], pos = -1 输出：返回 null
解释：链表中没有环。
提示：链表中节点的数目范围在范围 [0, 104] 内
-105 <= Node.val <= 105
pos 的值为 -1 或者链表中的一个有效索引
进阶：是否可以使用 O(1) 空间解决此题？
注意：本题与主站 142题相同
```

- 解题思路

```
func detectCycle(head *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for head != nil {
                if m[head] {
                        return head
                }
                m[head] = true
                head = head.Next
        }
        return nil
}

# 2
func detectCycle(head *ListNode) *ListNode {
        if head == nil {
                return nil
        }
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
                if fast == slow {
                        break
                }
        }
        if fast == nil || fast.Next == nil {
                return nil
        }
        slow = head
        for fast != slow {
                fast = fast.Next
                slow = slow.Next
        }
        return slow
}

# 3
func detectCycle(head *ListNode) *ListNode {
        for head != nil {
                if head.Val == math.MaxInt32 {
                        return head
                }
                head.Val = math.MaxInt32
                head = head.Next
```

(续下页)

(接上页)

```
        }
        return head
}
```

78.15 剑指 OfferII025. 链表中的两数相加 (3)

- 题目

```
给定两个 非空链表 l1和
↪l2来代表两个非负整数。数字最高位位于链表开始位置。它们的每个节点只存储一位数字。
将这两数相加会返回一个新的链表。
可以假设除了数字 0 之外，这两个数字都不会以零开头。
示例1：输入：l1 = [7,2,4,3], l2 = [5,6,4] 输出：[7,8,0,7]
示例2：输入：l1 = [2,4,3], l2 = [5,6,4] 输出：[8,0,7]
示例3：输入：l1 = [0], l2 = [0] 输出：[0]
提示：链表的长度范围为 [1, 100]
0 <= node.val <= 9
输入数据保证链表代表的数字无前导 0
进阶：如果输入链表不能修改该如何处理？换句话说，不能对列表中的节点进行翻转。
注意：本题与主站 445题相同：
```

- 解题思路

```
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        l1 = reverse(l1)
        l2 = reverse(l2)
        res := &ListNode{}
        cur := res
        carry := 0
        for l1 != nil || l2 != nil || carry > 0 {
                sum := carry
                if l1 != nil {
                        sum += l1.Val
                        l1 = l1.Next
                }
                if l2 != nil {
                        sum += l2.Val
                        l2 = l2.Next
                }
                carry = sum / 10 // 进位
                cur.Next = &ListNode{Val: sum % 10}
                cur = cur.Next
```

(续下页)

(接上页)

```
        }
        return reverse(res.Next)
}

func reverse(head *ListNode) *ListNode {
        var result *ListNode
        var temp *ListNode
        for head != nil {
                temp = head.Next
                head.Next = result
                result = head
                head = temp
        }
        return result
}

# 2
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        stack1 := make([]int, 0)
        stack2 := make([]int, 0)
        for l1 != nil {
                stack1 = append(stack1, l1.Val)
                l1 = l1.Next
        }
        for l2 != nil {
                stack2 = append(stack2, l2.Val)
                l2 = l2.Next
        }
        var res *ListNode
        carry := 0
        for len(stack1) > 0 || len(stack2) > 0 || carry > 0 {
                if len(stack1) > 0 {
                        carry = carry + stack1[len(stack1)-1]
                        stack1 = stack1[:len(stack1)-1]
                }
                if len(stack2) > 0 {
                        carry = carry + stack2[len(stack2)-1]
                        stack2 = stack2[:len(stack2)-1]
                }
                temp := &ListNode{
                        Val:  carry % 10,
                        Next: res,
                }
```

(续下页)

(接上页)

```
                carry = carry / 10
                res = temp
        }
        return res
}

# 3
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        a, b := l1, l2
        length1, length2 := 0, 0
        for a != nil {
                length1++
                a = a.Next
        }
        for b != nil {
                length2++
                b = b.Next
        }
        res, carry := add(l1, l2, length1, length2)
        if carry > 0 {
                return &ListNode{Val: carry, Next: res}
        }
        return res
}

func add(l1, l2 *ListNode, length1, length2 int) (res *ListNode, carry int) {
        if l1 != nil && l2 != nil {
                if l1.Next == nil && l2.Next == nil {
                        val := l1.Val + l2.Val
                        carry = val / 10
                        res = &ListNode{Val: val % 10, Next: nil}
                        return
                }
        }
        a := &ListNode{}
        var b, n int
        if length1 > length2 {
                a, b = add(l1.Next, l2, length1-1, length2)
                n = l1.Val + b
        } else if length1 < length2 {
                a, b = add(l1, l2.Next, length1, length2-1)
                n = l2.Val + b
        } else {
```

(续下页)

(接上页)

```
                a, b = add(l1.Next, l2.Next, length1-1, length2-1)
                n = l1.Val + l2.Val + b
        }
        res = &ListNode{Val: n % 10, Next: a}
        carry = n / 10
        return
}
```

78.16 剑指 OfferII026. 重排链表 (3)

- 题目

```
给定一个单链表 L 的头节点 head ，单链表 L 表示为：
L0→ L1→ … → Ln-1→ Ln
请将其重新排列后变为：
L0→Ln→L1→Ln-1→L2→Ln-2→ …
不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。
示例 1:输入: head = [1,2,3,4] 输出: [1,4,2,3]
示例 2:输入: head = [1,2,3,4,5] 输出: [1,5,2,4,3]
提示：链表的长度范围为 [1, 5 * 104]
1 <= node.val <= 1000
注意：本题与主站 143题相同
```

- 解题思路

```
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
                return
        }
        cur := head
        arr := make([]*ListNode, 0)
        for cur != nil {
                arr = append(arr, cur)
                cur = cur.Next
        }
        res := make([]*ListNode, 0)
        for i := 0; i < len(arr)/2; i++ {
                res = append(res, arr[i], arr[len(arr)-1-i])
        }
        if len(arr)%2 == 1 {
                res = append(res, arr[len(arr)/2])
        }
```

(续下页)

(接上页)

```
        cur = head
        for i := 1; i < len(res); i++ {
                cur.Next = res[i]
                cur = cur.Next
        }
        cur.Next = nil
}

# 2
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
                return
        }
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
                slow = slow.Next
        }
        second := reverse(slow.Next)
        slow.Next = nil
        cur := head
        count := 0
        for cur != nil && second != nil {
                a := cur.Next
                b := second.Next
                if count%2 == 0 {
                        cur.Next = second
                        cur = a
                } else {
                        second.Next = cur
                        second = b
                }
                count++
        }
}

func reverse(head *ListNode) *ListNode {
        var res *ListNode
        for head != nil {
                next := head.Next
                head.Next = res
                res = head
                head = next
```

(续下页)

(接上页)

```
        }
        return res
}

# 3
func reorderList(head *ListNode) {
        if head == nil || head.Next == nil {
                return
        }
        length := 0
        cur := head
        for cur != nil {
                length++
                cur = cur.Next
        }
        helper(head, length)
}

func helper(head *ListNode, length int) *ListNode {
        if length == 1 {
                next := head.Next
                head.Next = nil
                return next
        }
        if length == 2 {
                next := head.Next.Next
                head.Next.Next = nil
                return next
        }
        tail := helper(head.Next, length-2)
        next := tail.Next
        temp := head.Next
        head.Next = tail
        tail.Next = temp
        return next
}
```

78.17 剑指 OfferII028. 展平多级双向链表 (3)

- 题目

```
多级双向链表中，除了指向下一个节点和前一个节点指针之外，它还有一个子链表指针，可能指向单独的双向链表
这些子列表也可能会有一个或多个自己的子项，依此类推，生成多级数据结构，如下面的示例所示。
给定位于列表第一级的头节点，请扁平化列表，即将这样的多级双向链表展平成普通的双向链表，使所有结点出现
示例 1：输入：head = [1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12] 输出：[1,2,
↪3,7,8,11,12,9,10,4,5,6]
解释：输入的多级列表如下图所示：
扁平化后的链表如下图：
示例 2：输入：head = [1,2,null,3] 输出：[1,3,2]
解释：输入的多级列表如下图所示：
  1---2---NULL
  |
  3---NULL
示例 3：输入：head = [] 输出：[]
如何表示测试用例中的多级链表？
以 示例 1 为例：
 1---2---3---4---5---6--NULL
         |
         7---8---9---10--NULL
             |
             11--12--NULL
序列化其中的每一级之后：
[1,2,3,4,5,6,null]
[7,8,9,10,null]
[11,12,null]
为了将每一级都序列化到一起，我们需要每一级中添加值为 null␣
↪的元素，以表示没有节点连接到上一级的上级节点。
[1,2,3,4,5,6,null]
[null,null,7,8,9,10,null]
[null,11,12,null]
合并所有序列化结果，并去除末尾的 null 。
[1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12]
提示：节点数目不超过 1000
1 <= Node.val <= 10^5
注意：本题与主站 430题相同
```

- 解题思路

```
func flatten(root *Node) *Node {
        if root == nil {
                return nil
```

(续下页)

(接上页)

```
        }
        res := &Node{}
        cur := res
        for root != nil {
                cur.Next = root
                root.Prev = cur
                cur = cur.Next
                root = root.Next
                // 处理子节点
                if cur.Child != nil {
                        ch := flatten(cur.Child)
                        cur.Child = nil
                        cur.Next = ch
                        ch.Prev = cur
                        // 指针移动
                        for cur.Next != nil {
                                cur = cur.Next
                        }
                }
        }
        res.Next.Prev = nil
        return res.Next
}

# 2
var arr []*Node

func flatten(root *Node) *Node {
        arr = make([]*Node, 0)
        dfs(root)
        for i := 0; i < len(arr); i++ {
                if i+1 < len(arr) {
                        arr[i].Next = arr[i+1]
                }
                if i > 0 {
                        arr[i].Prev = arr[i-1]
                }
                arr[i].Child = nil
        }
        return root
}

func dfs(root *Node) {
```

(续下页)

(接上页)

```
        if root == nil {
                return
        }
        arr = append(arr, root)
        dfs(root.Child)
        dfs(root.Next)
}

# 3
func flatten(root *Node) *Node {
        cur := root
        stack := make([]*Node, 0)
        for cur != nil {
                // 处理child
                if cur.Child != nil {
                        if cur.Next != nil {
                                stack = append(stack, cur.Next)
                        }
                        cur.Child.Prev = cur
                        cur.Next = cur.Child
                        cur.Child = nil
                        continue
                }
                if cur.Next != nil {
            cur.Child = nil
                        cur = cur.Next
                        continue
                }
                if len(stack) == 0 {
                        break
                }
                last := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                cur.Next = last
                last.Prev = cur
                cur = last
        }
        return root
}
```

78.18 剑指 OfferII029. 排序的循环链表 (1)

- 题目

```
给定循环升序列表中的一个点，写一个函数向这个列表中插入一个新元素insertVal␣
↪，使这个列表仍然是循环升序的。
给定的可以是这个列表中任意一个顶点的指针，并不一定是这个列表中最小元素的指针。
如果有多个满足条件的插入位置，可以选择任意一个位置插入新的值，插入后整个列表仍然保持有序。
如果列表为空（给定的节点是␣
↪null），需要创建一个循环有序列表并返回这个节点。否则。请返回原先给定的节点。
示例 1：输入：head = [3,4,1], insertVal = 2 输出：[3,4,1,2]
解释：在上图中，有一个包含三个元素的循环有序列表，你获得值为 3␣
↪的节点的指针，我们需要向表中插入元素 2 。
新插入的节点应该在 1 和 3 之间，插入之后，整个列表如上图所示，最后返回节点 3 。
示例 2：输入：head = [], insertVal = 1 输出：[1]
解释：列表为空（给定的节点是 null），创建一个循环有序列表并返回这个节点。
示例 3：输入：head = [1], insertVal = 0 输出：[1,0]
提示：0 <= Number of Nodes <= 5 * 10^4
-10^6 <= Node.val <= 10^6
-10^6 <=insertVal <= 10^6
注意：本题与主站 708题相同：
```

- 解题思路

```
func insert(aNode *Node, x int) *Node {
        if aNode == nil {
                res := &Node{Val: x}
                res.Next = res
                return res
        }
        cur := aNode
        for cur.Next != aNode {
                if (cur.Val <= x && x <= cur.Next.Val) || // a<x<b
                        (cur.Val > cur.Next.Val && (cur.Val <= x || x <= cur.Next.
↪Val)) { // 插入最大值或者最小值
                        break
                }
                cur = cur.Next
        }
        cur.Next = &Node{
                Val:  x,
                Next: cur.Next,
        }
        return aNode
```

(续下页)

(接上页)

```
}
```

78.19 剑指 OfferII030. 插入、删除和随机访问都是 O(1) 的容器 (2)

- 题目

```
设计一个支持在平均时间复杂度 O(1)下，执行以下操作的数据结构：
insert(val)：当元素 val 不存在时返回 true，并向集合中插入该项，否则返回 false 。
remove(val)：当元素 val 存在时返回 true，并从集合中移除该项，否则返回 false。
getRandom：随机返回现有集合中的一项。每个元素应该有相同的概率被返回。
示例 :输入：inputs = ["RandomizedSet", "insert",
"remove", "insert", "getRandom", "remove", "insert", "getRandom"]
[[], [1], [2], [2], [], [1], [2], []]
输出：[null, true, false, true, 2, true, false, 2]
解释:RandomizedSet randomSet = new RandomizedSet();  // 初始化一个空的集合
randomSet.insert(1); // 向集合中插入 1 ， 返回 true 表示 1 被成功地插入
randomSet.remove(2); // 返回 false，表示集合中不存在 2
randomSet.insert(2); // 向集合中插入 2 返回 true ，集合现在包含 [1,2]
randomSet.getRandom(); // getRandom 应随机返回 1 或 2
randomSet.remove(1); // 从集合中移除 1 返回 true 。集合现在包含 [2]
randomSet.insert(2); // 2 已在集合中，所以返回 false
randomSet.getRandom(); // 由于 2 是集合中唯一的数字，getRandom 总是返回 2
提示：-231<= val <= 231- 1
最多进行 2 * 105 次insert ， remove 和 getRandom 方法调用
当调用getRandom 方法时，集合中至少有一个元素
注意：本题与主站 380题相同
```

- 解题思路

```
type RandomizedSet struct {
        m   map[int]int
        arr []int
}

func Constructor() RandomizedSet {
        return RandomizedSet{
                m:   make(map[int]int),
                arr: make([]int, 0),
        }
}

func (this *RandomizedSet) Insert(val int) bool {
```

(续下页)

(接上页)

```
        if _, ok := this.m[val]; ok {
                return false
        }
        this.arr = append(this.arr, val)
        this.m[val] = len(this.arr) - 1
        return true
}

func (this *RandomizedSet) Remove(val int) bool {
        if _, ok := this.m[val]; !ok {
                return false
        }
        index := this.m[val]
        this.arr[index], this.arr[len(this.arr)-1] = this.arr[len(this.arr)-1], this.
↪arr[index]
        this.m[this.arr[index]] = index
        this.arr = this.arr[:len(this.arr)-1]
        delete(this.m, val)
        return true
}

func (this *RandomizedSet) GetRandom() int {
        if len(this.arr) == 0 {
                return -1
        }
        index := rand.Intn(len(this.arr))
        return this.arr[index]
}

# 2
type RandomizedSet struct {
        m map[int]bool
}

func Constructor() RandomizedSet {
        return RandomizedSet{
                m: make(map[int]bool),
        }
}

func (this *RandomizedSet) Insert(val int) bool {
        if _, ok := this.m[val]; ok {
                return false
```

(续下页)

(接上页)

```
        }
        this.m[val] = true
        return true
}

func (this *RandomizedSet) Remove(val int) bool {
        if _, ok := this.m[val]; !ok {
                return false
        }
        delete(this.m, val)
        return true
}

func (this *RandomizedSet) GetRandom() int {
        if len(this.m) == 0 {
                return -1
        }
        index := rand.Intn(len(this.m))
        res := -1
        for res = range this.m {
                index--
                if index == -1 {
                        break
                }
        }
        return res
}
```

78.20 剑指 OfferII031. 最近最少使用缓存 (1)

- 题目

```
运用所掌握的数据结构，设计和实现一个 LRU (Least Recently Used，最近最少使用) 缓存机制␣
↪。
实现 LRUCache 类：
LRUCache(int capacity) 以正整数作为容量capacity 初始化 LRU 缓存
int get(int key) 如果关键字 key 存在于缓存中，则返回关键字的值，否则返回 -1 。
void put(int key, int␣
↪value)如果关键字已经存在，则变更其数据值；如果关键字不存在，则插入该组「关键字-
↪值」。
当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。
示例：输入 ["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
```

(续下页)

(接上页)

```
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
输出[null, null, null, 1, null, -1, null, -1, 3, 4]
解释 LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // 缓存是 {1=1}
lRUCache.put(2, 2); // 缓存是 {1=1, 2=2}
lRUCache.get(1);    // 返回 1
lRUCache.put(3, 3); // 该操作会使得关键字 2 作废，缓存是 {1=1, 3=3}
lRUCache.get(2);    // 返回 -1 (未找到)
lRUCache.put(4, 4); // 该操作会使得关键字 1 作废，缓存是 {4=4, 3=3}
lRUCache.get(1);    // 返回 -1 (未找到)
lRUCache.get(3);    // 返回 3
lRUCache.get(4);    // 返回 4
提示：1 <= capacity <= 3000
0 <= key <= 10000
0 <= value <= 105
最多调用 2 * 105 次 get 和 put
进阶：是否可以在O(1) 时间复杂度内完成这两种操作？
注意：本题与主站 146题相同
```

- 解题思路

```
type Node struct {
        key   int
        value int
        prev  *Node
        next  *Node
}

type LRUCache struct {
        cap    int
        header *Node
        tail   *Node
        m      map[int]*Node
}

func Constructor(capacity int) LRUCache {
        cache := LRUCache{
                cap:    capacity,
                header: &Node{},
                tail:   &Node{},
                m:      make(map[int]*Node, capacity),
        }
        cache.header.next = cache.tail
        cache.tail.prev = cache.header
```

(续下页)

(接上页)

```
        return cache
}

func (this *LRUCache) Get(key int) int {
        if node, ok := this.m[key]; ok {
                this.remove(node)
                this.putHead(node)
                return node.value
        }
        return -1
}

func (this *LRUCache) Put(key int, value int) {
        if node, ok := this.m[key]; ok {
                node.value = value
                this.remove(node)
                this.putHead(node)
                return
        }
        if this.cap <= len(this.m) {
                // 删除尾部
                deleteKey := this.tail.prev.key
                this.remove(this.tail.prev)
                delete(this.m, deleteKey)
        }
        // 插入到头部
        newNode := &Node{key: key, value: value}
        this.putHead(newNode)
        this.m[key] = newNode
}

// 删除尾部节点
func (this *LRUCache) remove(node *Node) {
        node.prev.next = node.next
        node.next.prev = node.prev
}

// 插入头部
func (this *LRUCache) putHead(node *Node) {
        next := this.header.next
        this.header.next = node
        node.next = next
        next.prev = node
```

(续下页)

(接上页)

```
        node.prev = this.header
}
```

78.21 剑指 OfferII033. 变位词组 (2)

- 题目

```
给定一个字符串数组 strs ，将变位词组合在一起。 可以按任意顺序返回结果列表。
注意：若两个字符串中每个字符出现的次数都相同，则称它们互为变位词。
示例 1:输入: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
输出: [["bat"],["nat","tan"],["ate","eat","tea"]]
示例 2:输入: strs = [""] 输出: [[""]]
示例 3:输入: strs = ["a"] 输出: [["a"]]
提示：1 <= strs.length <= 104
0 <= strs[i].length <= 100
strs[i]仅包含小写字母
注意：本题与主站 49题相同：
```

- 解题思路

```
func groupAnagrams(strs []string) [][]string {
        m := make(map[string]int)
        res := make([][]string, 0)
        for i := 0; i < len(strs); i++ {
                arr := []byte(strs[i])
                sort.Slice(arr, func(i, j int) bool {
                        return arr[i] < arr[j]
                })
                newStr := string(arr)
                if _, ok := m[newStr]; ok {
                        res[m[newStr]] = append(res[m[newStr]], strs[i])
                } else {
                        m[newStr] = len(res)
                        res = append(res, []string{strs[i]})
                }
        }
        return res
}

#
func groupAnagrams(strs []string) [][]string {
        m := make(map[[26]int]int)
```

(续下页)

(接上页)

```
        res := make([][]string, 0)
        for i := 0; i < len(strs); i++ {
                arr := [26]int{}
                for j := 0; j < len(strs[i]); j++{
                        arr[strs[i][j]-'a']++
                }
                if _, ok := m[arr]; ok {
                        res[m[arr]] = append(res[m[arr]], strs[i])
                } else {
                        m[arr] = len(res)
                        res = append(res, []string{strs[i]})
                }
        }
        return res
}
```

78.22 剑指 OfferII035. 最小时间差 (2)

- 题目

```
给定一个 24 小时制（小时:分钟 "HH:MM
↪"）的时间列表，找出列表中任意两个时间的最小时间差并以分钟数表示。
示例 1：输入：timePoints = ["23:59","00:00"] 输出：1
示例 2：输入：timePoints = ["00:00","23:59","00:00"] 输出：0
提示：2 <= timePoints <= 2 * 104
timePoints[i] 格式为 "HH:MM"
注意：本题与主站 539题相同：
```

- 解题思路

```
func findMinDifference(timePoints []string) int {
        m := make(map[int]bool)
        for i := 0; i < len(timePoints); i++ {
                value := getValue(timePoints[i])
                if _, ok := m[value]; ok {
                        return 0
                }
                m[value] = true
        }
        arr := make([]int, 0)
        for k := range m {
                arr = append(arr, k)
```

(续下页)

(接上页)

```
        }
        sort.Ints(arr)
        res := math.MaxInt32
        arr = append(arr, arr[0]+1440)
        for i := 1; i < len(arr); i++ {
                if res > arr[i]-arr[i-1] {
                        res = arr[i] - arr[i-1]
                }
        }
        return res
}

func getValue(str string) int {
        hour, _ := strconv.Atoi(str[:2])
        minute, _ := strconv.Atoi(str[3:])
        return hour*60 + minute
}

# 2
func findMinDifference(timePoints []string) int {
        arr := make([]int, 0)
        for i := 0; i < len(timePoints); i++ {
                value := getValue(timePoints[i])
                arr = append(arr, value)
        }
        sort.Ints(arr)
        res := math.MaxInt32
        arr = append(arr, arr[0]+1440)
        for i := 1; i < len(arr); i++ {
                if res > arr[i]-arr[i-1] {
                        res = arr[i] - arr[i-1]
                }
        }
        return res
}

func getValue(str string) int {
        hour, _ := strconv.Atoi(str[:2])
        minute, _ := strconv.Atoi(str[3:])
        return hour*60 + minute
}
```

78.23 剑指 OfferII036. 后缀表达式 (1)

- 题目

```
根据 逆波兰表示法，求该后缀表达式的计算结果。
有效的算符包括+、-、*、/。每个运算对象可以是整数，也可以是另一个逆波兰表达式。
说明：整数除法只保留整数部分。
给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。
示例1：输入：tokens = ["2","1","+","3","*"] 输出：9
解释：该算式转化为常见的中缀算术表达式为：((2 + 1) * 3) = 9
示例2：输入：tokens = ["4","13","5","/","+"] 输出：6
解释：该算式转化为常见的中缀算术表达式为：(4 + (13 / 5)) = 6
示例3：输入：tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
↪输出：22
解释：该算式转化为常见的中缀算术表达式为：
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
提示：1 <= tokens.length <= 104
tokens[i] 要么是一个算符（"+"、"-"、"*" 或 "/"），要么是一个在范围 [-200, 200]
↪内的整数
逆波兰表达式：逆波兰表达式是一种后缀表达式，所谓后缀就是指算符写在后面。
平常使用的算式则是一种中缀表达式，如 ( 1 + 2 ) * ( 3 + 4 ) 。
该算式的逆波兰表达式写法为 ( ( 1 2 + ) ( 3 4 + ) * ) 。
逆波兰表达式主要有以下两个优点：
去掉括号后表达式无歧义，上式即便写成 1 2 + 3 4 + * 也可以依据次序计算出正确结果。
适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中。
注意：本题与主站 150题相同：
```

- 解题思路

```go
func evalRPN(tokens []string) int {
        stack := make([]int, 0)
        for _, v := range tokens {
                length := len(stack)
                if v == "+" || v == "-" || v == "*" || v == "/" {
                        a := stack[length-2]
                        b := stack[length-1]
                        stack = stack[:length-2]
                        var value int
```

(续下页)

(接上页)

```
                        if v == "+" {
                                value = a + b
                        } else if v == "-" {
                                value = a - b
                        } else if v == "*" {
                                value = a * b
                        } else {
                                value = a / b
                        }
                        stack = append(stack, value)
                } else {
                        value, _ := strconv.Atoi(v)
                        stack = append(stack, value)
                }
        }
        return stack[0]
}
```

78.24 剑指 OfferII037. 小行星碰撞 (2)

- 题目

```
给定一个整数数组 asteroids，表示在同一行的小行星。
对于数组中的每一个元素，其绝对值表示小行星的大小，正负表示小行星的移动方向（正表示向右移动，负表示向
每一颗小行星以相同的速度移动。
找出碰撞后剩下的所有小行星。碰撞规则：两个行星相互碰撞，较小的行星会爆炸。如果两颗行星大小相同，则两
两颗移动方向相同的行星，永远不会发生碰撞。
示例 1：输入：asteroids = [5,10,-5] 输出：[5,10]
解释：10 和 -5 碰撞后只剩下 10 。 5 和 10 永远不会发生碰撞。
示例 2：输入：asteroids = [8,-8] 输出：[]
解释：8 和 -8 碰撞后，两者都发生爆炸。
示例 3：输入：asteroids = [10,2,-5] 输出：[10]
解释：2 和 -5 发生碰撞后剩下 -5 。10 和 -5 发生碰撞后剩下 10 。
示例 4：输入：asteroids = [-2,-1,1,2] 输出：[-2,-1,1,2]
解释：-2 和 -1 向左移动，而 1 和 2 向右移动。␣
↪由于移动方向相同的行星不会发生碰撞，所以最终没有行星发生碰撞。
提示：2 <= asteroids.length<= 104
-1000 <= asteroids[i] <= 1000
asteroids[i] != 0
注意：本题与主站 735题相同：
```

- 解题思路

```
func asteroidCollision(asteroids []int) []int {
        left := make([]int, 0)
        right := make([]int, 0)
        for i := 0; i < len(asteroids); i++ {
                if asteroids[i] > 0 {
                        right = append(right, asteroids[i])
                } else {
                        if len(right) > 0 {
                                for {
                                        if len(right) == 0 {
                                                left = append(left, asteroids[i])
                                                break
                                        }
                                        sum := asteroids[i] + right[len(right)-1]
                                        if sum == 0 {
                                                right = right[:len(right)-1]
                                                break
                                        } else if sum > 0 {
                                                break
                                        } else {
                                                right = right[:len(right)-1]
                                        }
                                }
                        } else {
                                left = append(left, asteroids[i])
                        }
                }

        }
        return append(left, right...)
}

# 2
func asteroidCollision(asteroids []int) []int {
        res := make([]int, 0)
        for i := 0; i < len(asteroids); i++ {
                value := asteroids[i]
                for value < 0 && len(res) > 0 && res[len(res)-1] > 0 {
                        sum := value + res[len(res)-1]
                        if sum >= 0 {
                                value = 0
                        }
                        if sum <= 0 {
                                res = res[:len(res)-1]
```

(续下页)

(接上页)

```
                        }
                }
                if value != 0 {
                        res = append(res, value)
                }
        }
        return res
}
```

78.25 剑指 OfferII038. 每日温度 (3)

- 题目

```
请根据每日 气温 列表␣
↪temperatures，重新生成一个列表，要求其对应位置的输出为：要想观测到更高的气温，至少需要等待的天数。
如果气温在这之后都不会升高，请在该位置用0 来代替。
示例 1:输入: temperatures = [73,74,75,71,69,72,76,73] 输出:[1,1,4,2,1,1,0,0]
示例 2:输入: temperatures = [30,40,50,60] 输出:[1,1,1,0]
示例 3:输入: temperatures = [30,60,90] 输出: [1,1,0]
提示：1 <=temperatures.length <= 105
30 <=temperatures[i]<= 100
注意：本题与主站 739题相同
```

- 解题思路

```
func dailyTemperatures(temperatures []int) []int {
        res := make([]int, len(temperatures))
        stack := make([]int, 0) // 栈保存递减数据的下标
        for i := 0; i < len(temperatures); i++ {
                for len(stack) > 0 && temperatures[i] > temperatures[stack[len(stack)-
↪1]] {
                        last := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        res[last] = i - last
                }
                stack = append(stack, i)
        }
        return res
}

# 2
func dailyTemperatures(temperatures []int) []int {
```

(续下页)

(接上页)

```
        res := make([]int, len(temperatures))
        arr := make([]int, 101)
        for i := 0; i < len(arr); i++ {
                arr[i] = math.MaxInt64
        }
        for i := len(temperatures) - 1; i >= 0; i-- {
                temp := math.MaxInt64
                for t := temperatures[i] + 1; t < 101; t++ {
                        if arr[t] < temp {
                                temp = arr[t]
                        }
                }
                if temp < math.MaxInt64 {
                        res[i] = temp - i
                }
                arr[temperatures[i]] = i
        }
        return res
}

# 3
func dailyTemperatures(temperatures []int) []int {
        j := 0
        for i := 0; i < len(temperatures); i++ {
                for j = i + 1; j < len(temperatures); j++ {
                        if temperatures[j] > temperatures[i] {
                                temperatures[i] = j - i
                                break
                        }
                }
                if j == len(temperatures) {
                        temperatures[i] = 0
                }
        }
        return temperatures
}
```

78.26 剑指 OfferII043. 往完全二叉树添加节点 (1)

- 题目

```
完全二叉树是每一层（除最后一层外）都是完全填充（即，节点数达到最大，第 n 层有 2n-
↪1个节点）的，
并且所有的节点都尽可能地集中在左侧。
设计一个用完全二叉树初始化的数据结构CBTInserter，它支持以下几种操作：
CBTInserter(TreeNode root)使用根节点为root的给定树初始化该数据结构；
CBTInserter.insert(int v) 向树中插入一个新节点，节点类型为 TreeNode，值为 v 。
使树保持完全二叉树的状态，并返回插入的新节点的父节点的值；
CBTInserter.get_root() 将返回树的根节点。
示例 1：输入：inputs = ["CBTInserter","insert","get_root"], inputs = [[[1]],[2],[]]␣
↪输出：[null,1,[1,2]]
示例 2：输入：inputs = ["CBTInserter","insert","insert","get_root"], inputs = [[[1,2,
↪3,4,5,6]],[7],[8],[]]
输出：[null,3,4,[1,2,3,4,5,6,7,8]]
提示：最初给定的树是完全二叉树，且包含1到1000个节点。
每个测试用例最多调用CBTInserter.insert 操作10000次。
给定节点或插入节点的每个值都在0到5000之间。
注意：本题与主站 919题相同：
```

- 解题思路

```
type CBTInserter struct {
        root *TreeNode
        arr  []*TreeNode
}

func Constructor(root *TreeNode) CBTInserter {
        arr := make([]*TreeNode, 0)
        queue := make([]*TreeNode, 0)
        arr = append(arr, root)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                                arr = append(arr, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                                arr = append(arr, queue[i].Right)
```

(续下页)

(接上页)

```
                        }
                }
                queue = queue[length:]
        }
        return CBTInserter{root: root, arr: arr}
}

func (this *CBTInserter) Insert(v int) int {
        newNode := &TreeNode{Val: v}
        this.arr = append(this.arr, newNode)
        n := len(this.arr)
        target := this.arr[n/2-1]
        if n%2 == 0 {
                target.Left = newNode
        } else {
                target.Right = newNode
        }
        return target.Val
}

func (this *CBTInserter) Get_root() *TreeNode {
        return this.root
}
```

78.27 剑指 OfferII044. 二叉树每层的最大值 (2)

- 题目

```
给定一棵二叉树的根节点root ，请找出该二叉树中每一层的最大值。
示例1: 输入: root = [1,3,2,5,3,null,9] 输出: [1,3,9]
解释:
          1
         / \
        3   2
       / \   \
      5   3   9
示例2: 输入: root = [1,2,3] 输出: [1,3]
解释:
          1
         / \
        2   3
示例3: 输入: root = [1] 输出: [1]
```

(续下页)

(接上页)

```
示例4：输入：root = [1,null,2] 输出：[1,2]
解释：
          1
           \
            2
示例5：输入：root = [] 输出：[]
提示：二叉树的节点个数的范围是 [0,104]
-231<= Node.val <= 231- 1
注意：本题与主站 515题相同：
```

- 解题思路

```
func largestValues(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil {
                return res
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                maxValue := math.MinInt32
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                        if maxValue < queue[i].Val {
                                maxValue = queue[i].Val
                        }
                }
                res = append(res, maxValue)
                queue = queue[length:]
        }
        return res
}

# 2
var res []int

func largestValues(root *TreeNode) []int {
        res = make([]int, 0)
```

(续下页)

(接上页)

```
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        if level >= len(res) {
                res = append(res, math.MinInt32)
        }
        res[level] = max(res[level], root.Val)
        dfs(root.Left, level+1)
        dfs(root.Right, level+1)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

78.28 剑指 OfferII045. 二叉树最底层最左边的值 (2)

- 题目

```
给定一个二叉树的 根节点 root，请找出该二叉树的最底层最左边节点的值。
假设二叉树中至少有一个节点。
示例 1:输入: root = [2,1,3] 输出: 1
示例 2:输入: [1,2,3,4,null,5,6,null,null,7] 输出: 7
提示:二叉树的节点个数的范围是 [1,104]
-231<= Node.val <= 231- 1
注意: 本题与主站 513题相同:
```

- 解题思路

```
func findBottomLeftValue(root *TreeNode) int {
        res := 0
```

(续下页)

(接上页)

```
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                length := len(queue)
                res = queue[0].Val
                for i := 0; i < length; i++ {
                        if queue[i].Left != nil {
                                queue = append(queue, queue[i].Left)
                        }
                        if queue[i].Right != nil {
                                queue = append(queue, queue[i].Right)
                        }
                }
                queue = queue[length:]
        }
        return res
}

# 2
var res int
var maxLevel int

func findBottomLeftValue(root *TreeNode) int {
        res = 0
        maxLevel = -1
        if root == nil {
                return res
        }
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        dfs(root.Left, level+1)
        if level > maxLevel {
                maxLevel = level
                res = root.Val
        }
        dfs(root.Right, level+1)
}
```

78.29 剑指 OfferII046. 二叉树的右侧视图 (2)

- 题目

```
给定一个二叉树的 根节点␣
↪root，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。
示例 1:输入:[1,2,3,null,5,null,4] 输出:[1,3,4]
示例 2:输入:[1,null,3] 输出:[1,3]
示例 3:输入:[] 输出:[]
提示:二叉树的节点个数的范围是 [0,100]
-100<= Node.val <= 100
注意：本题与主站 199题相同：
```

- 解题思路

```
func rightSideView(root *TreeNode) []int {
        res := make([]int, 0)
        if root == nil{
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                res = append(res, list[0].Val)
                for i := 0; i < length; i++ {
                        node := list[i]
                        if node.Right != nil {
                                list = append(list, node.Right)
                        }
                        if node.Left != nil {
                                list = append(list, node.Left)
                        }
                }
                list = list[length:]
        }
        return res
}

#
var res []int

func rightSideView(root *TreeNode) []int {
        res = make([]int, 0)
```

(续下页)

(接上页)

```
        if root == nil {
                return res
        }
        dfs(root, 1)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
        if level > len(res) {
                res = append(res, root.Val)
        }
        dfs(root.Right, level+1)
        dfs(root.Left, level+1)
}
```

78.30 剑指 OfferII047. 二叉树剪枝 (1)

- 题目

```
给定一个二叉树 根节点root，树的每个节点的值要么是 0，要么是␣
↪1。请剪除该二叉树中所有节点的值为 0 的子树。
节点 node 的子树为node 本身，以及所有 node的后代。
示例 1:输入：[1,null,0,0,1] 输出：[1,null,0,null,1]
解释：只有红色节点满足条件“所有不包含 1 的子树”。
右图为返回的答案。
示例 2:输入：[1,0,1,0,0,0,1] 输出：[1,null,1,null,1]
解释：
示例 3:输入：[1,1,0,1,1,0,1,0] 输出：[1,1,0,1,1,null,1]
解释：
提示:二叉树的节点个数的范围是 [1,200]
二叉树节点的值只会是 0 或 1
注意：本题与主站 814题相同：
```

- 解题思路

```
func pruneTree(root *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
```

(续下页)

(接上页)

```
        root.Left = pruneTree(root.Left)
        root.Right = pruneTree(root.Right)
        if root.Left == nil && root.Right == nil && root.Val == 0 {
                return nil
        }
        return root
}
```

78.31 剑指 OfferII049. 从根节点到叶节点的路径数字之和 (2)

- 题目

```
给定一个二叉树的根节点 root ，树中每个节点都存放有一个 0 到 9 之间的数字。
每条从根节点到叶节点的路径都代表一个数字：
例如，从根节点到叶节点的路径 1 -> 2 -> 3 表示数字 123 。
计算从根节点到叶节点生成的 所有数字之和 。
叶节点 是指没有子节点的节点。
示例 1：输入：root = [1,2,3] 输出：25
解释：从根到叶子节点路径 1->2 代表数字 12
从根到叶子节点路径 1->3 代表数字 13
因此，数字总和 = 12 + 13 = 25
示例 2：输入：root = [4,9,0,5,1] 输出：1026
解释：从根到叶子节点路径 4->9->5 代表数字 495
从根到叶子节点路径 4->9->1 代表数字 491
从根到叶子节点路径 4->0 代表数字 40
因此，数字总和 = 495 + 491 + 40 = 1026
提示：树中节点的数目在范围 [1, 1000] 内
0 <= Node.val <= 9
树的深度不超过 10
注意：本题与主站 129题相同：
```

- 解题思路

```
var res int

func sumNumbers(root *TreeNode) int {
        res = 0
        dfs(root, 0)
        return res
}

func dfs(root *TreeNode, sum int) {
```

(续下页)

(接上页)

```
        if root == nil {
                return
        }
        sum = sum*10 + root.Val
        if root.Left == nil && root.Right == nil {
                res = res + sum
        }
        dfs(root.Left, sum)
        dfs(root.Right, sum)
}

#
func sumNumbers(root *TreeNode) int {
        res := 0
        if root == nil {
                return res
        }
        list := make([]*TreeNode, 0)
        list = append(list, root)
        for len(list) > 0 {
                length := len(list)
                for i := 0; i < length; i++ {
                        node := list[i]
                        value := node.Val
                        if node.Left == nil && node.Right == nil {
                                res = res + value
                        }
                        if node.Left != nil {
                                node.Left.Val = node.Left.Val + value*10
                                list = append(list, node.Left)
                        }
                        if node.Right != nil {
                                node.Right.Val = node.Right.Val + value*10
                                list = append(list, node.Right)
                        }
                }
                list = list[length:]
        }
        return res
}
```

78.32 剑指 OfferII050. 向下的路径节点之和 (4)

- 题目

```
给定一个二叉树的根节点 root，和一个整数 targetSum ，求该二叉树里节点值之和等于␣
↪targetSum 的 路径 的数目。
路径␣
↪不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。
示例 1：输入：root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8 输出：3
解释：和等于 8 的路径有 3 条，如图所示。
示例 2：输入：root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22 输出：3
提示:二叉树的节点个数的范围是 [0,1000]
-109<= Node.val <= 109
-1000<= targetSum<= 1000
注意：本题与主站 437题相同
```

- 解题思路

```
func pathSum(root *TreeNode, targetSum int) int {
        if root == nil {
                return 0
        }
        res := 0
        var helper func(*TreeNode, int)
        helper = func(node *TreeNode, sum int) {
                if node == nil {
                        return
                }
                sum = sum - node.Val
                // 路径不需要从根节点开始，也不需要在叶子节点结束
                if sum == 0 {
                        res++
                }
                helper(node.Left, sum)
                helper(node.Right, sum)
        }
        helper(root, targetSum)
        return res + pathSum(root.Left, targetSum) + pathSum(root.Right, targetSum)
}

# 2
func helper(node *TreeNode, targetSum int) int {
        if node == nil {
                return 0
```

(续下页)

(接上页)

```
        }
        targetSum = targetSum - node.Val
        res := 0
        if targetSum == 0 {
                res = 1
        }
        return res + helper(node.Left, targetSum) + helper(node.Right, targetSum)
}

func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
        }
        return helper(root, sum) + pathSum(root.Left, sum) + pathSum(root.Right, sum)
}

# 3
func helper(node *TreeNode, targetSum int, curSum int) int {
        res := 0
        curSum = curSum + node.Val
        if curSum == targetSum {
                res++
        }
        if node.Left != nil {
                res += helper(node.Left, targetSum, curSum)
        }
        if node.Right != nil {
                res += helper(node.Right, targetSum, curSum)
        }
        return res
}

func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        res := 0
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                tempSum := 0
```

(续下页)

(接上页)

```
                res += helper(node, sum, tempSum)
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return res
}

# 4
func helper(node *TreeNode, sum int, path []int, level int) int {
        if node == nil {
                return 0
        }
        res := 0
        if sum == node.Val {
                res = 1
        }
        temp := node.Val
        for i := level - 1; i >= 0; i-- {
                temp = temp + path[i]
                if temp == sum {
                        res++
                }
        }
        path[level] = node.Val
        return res + helper(node.Left, sum, path, level+1) +
                helper(node.Right, sum, path, level+1)
}

func pathSum(root *TreeNode, targetSum int) int {
        return helper(root, targetSum, make([]int, 1001), 0)
}
```

78.33 剑指 OfferII053. 二叉搜索树中的中序后继 (2)

- 题目

```
给定一棵二叉搜索树和其中的一个节点 p␣
↪，找到该节点在树中的中序后继。如果节点没有中序后继，请返回 null 。
节点p的后继是值比p.val大的节点中键值最小的节点，即按中序遍历的顺序节点 p␣
↪的下一个节点。
示例 1：输入：root = [2,1,3], p = 1 输出：2
解释：这里 1 的中序后继是 2。请注意 p 和返回值都应是 TreeNode 类型。
示例2：输入：root = [5,3,6,2,4,null,null,1], p = 6 输出：null
解释：因为给出的节点没有中序后继，所以答案就返回 null 了。
提示：树中节点的数目在范围 [1, 104] 内。
-105 <= Node.val <= 105
树中各节点的值均保证唯一。
注意：本题与主站 285题相同：
```

- 解题思路

```
func inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {
        var res *TreeNode
        for root != nil {
                if root.Val > p.Val {
                        res = root
                        root = root.Left
                } else {
                        root = root.Right
                }
        }
        return res
}

# 2
func inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Val > p.Val {
                res := inorderSuccessor(root.Left, p)
                if res == nil {
                        return root
                }
                return res
        } else {
```

(续下页)

(接上页)

```
                return inorderSuccessor(root.Right, p)
        }
}
```

78.34 剑指 OfferII054. 所有大于等于节点的值之和 (2)

- 题目

```
给定一个二叉搜索树，请将它的每个节点的值替换成树中大于或者等于该节点值的所有节点值之和。
提醒一下，二叉搜索树满足下列约束条件：
节点的左子树仅包含键 小于 节点键的节点。
节点的右子树仅包含键 大于 节点键的节点。
左右子树也必须是二叉搜索树。
示例 1：输入：root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
输出：[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
示例 2：输入：root = [0,null,1] 输出：[1,null,1]
示例 3：输入：root = [1,0,2] 输出：[3,3,2]
示例 4：输入：root = [3,2,4,1] 输出：[7,9,4,10]
提示：树中的节点数介于 0和 104之间。
每个节点的值介于 -104和104之间。
树中的所有值 互不相同 。
给定的树为二叉搜索树。
注意：本题与主站 538题相同：
本题与主站 1038题相同：
```

- 解题思路

```
func convertBST(root *TreeNode) *TreeNode {
        sum := 0
        dfs(root, &sum)
        return root
}

func dfs(root *TreeNode, sum *int) {
        if root == nil {
                return
        }
        dfs(root.Right, sum)
        *sum = *sum + root.Val
        root.Val = *sum
        dfs(root.Left, sum)
}
```

(续下页)

(接上页)

```
# 2
func convertBST(root *TreeNode) *TreeNode {
        if root == nil {
                return root
        }
        stack := make([]*TreeNode, 0)
        temp := root
        sum := 0
        for {
                if temp != nil {
                        stack = append(stack, temp)
                        temp = temp.Right
                } else if len(stack) != 0 {
                        temp = stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        temp.Val = temp.Val + sum
                        sum = temp.Val
                        temp = temp.Left
                } else {
                        break
                }
        }
        return root
}
```

78.35 剑指 OfferII055. 二叉搜索树迭代器 (2)

- 题目

```
实现一个二叉搜索树迭代器类BSTIterator ，表示一个按中序遍历二叉搜索树（BST）的迭代器：
BSTIterator(TreeNode root) 初始化 BSTIterator 类的一个对象。BST 的根节点 root␣
↪会作为构造函数的一部分给出。
指针应初始化为一个不存在于 BST 中的数字，且该数字小于 BST 中的任何元素。
boolean hasNext() 如果向指针右侧遍历存在数字，则返回 true ；否则返回 false 。
int next()将指针向右移动，然后返回指针处的数字。
注意，指针初始化为一个不存在于 BST 中的数字，所以对 next() 的首次调用将返回 BST␣
↪中的最小元素。
可以假设next()调用总是有效的，也就是说，当调用 next()时，BST␣
↪的中序遍历中至少存在一个下一个数字。
示例：输入inputs = ["BSTIterator", "next", "next", "hasNext", "next", "hasNext",
"next", "hasNext", "next", "hasNext"]
```

(续下页)

(接上页)

```
inputs = [[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []]
输出[null, 3, 7, true, 9, true, 15, true, 20, false]
解释 BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]);
bSTIterator.next();    // 返回 3
bSTIterator.next();    // 返回 7
bSTIterator.hasNext(); // 返回 True
bSTIterator.next();    // 返回 9
bSTIterator.hasNext(); // 返回 True
bSTIterator.next();    // 返回 15
bSTIterator.hasNext(); // 返回 True
bSTIterator.next();    // 返回 20
bSTIterator.hasNext(); // 返回 False
提示：树中节点的数目在范围 [1, 105] 内
0 <= Node.val <= 106
最多调用 105 次 hasNext 和 next 操作
进阶：你可以设计一个满足下述条件的解决方案吗？next() 和 hasNext()␣
↪操作均摊时间复杂度为 O(1) ，并使用 O(h) 内存。
其中 h 是树的高度。
注意：本题与主站 173题相同：
```

- 解题思路

```
type BSTIterator struct {
        arr  []int
        root *TreeNode
}

func Constructor(root *TreeNode) BSTIterator {
        arr := make([]int, 0)
        inorder(root, &arr)
        return BSTIterator{
                arr:  arr,
                root: root,
        }
}

func inorder(root *TreeNode, nums *[]int) {
        if root == nil {
                return
        }
        inorder(root.Left, nums)
        *nums = append(*nums, root.Val)
        inorder(root.Right, nums)
}
```

(续下页)

(接上页)

```
func (this *BSTIterator) Next() int {
        if len(this.arr) == 0 {
                return -1
        }
        res := this.arr[0]
        this.arr = this.arr[1:]
        return res
}

func (this *BSTIterator) HasNext() bool {
        if len(this.arr) > 0 {
                return true
        }
        return false
}

# 2
type BSTIterator struct {
        stack []*TreeNode
}

func Constructor(root *TreeNode) BSTIterator {
        res := BSTIterator{}
        res.left(root)
        return res
}

func (this *BSTIterator) left(root *TreeNode) {
        for root != nil {
                this.stack = append(this.stack, root)
                root = root.Left
        }
}

func (this *BSTIterator) Next() int {
        node := this.stack[len(this.stack)-1]
        this.stack = this.stack[:len(this.stack)-1]
        if node.Right != nil {
                this.left(node.Right)
        }
        return node.Val
}
```

(续下页)

(接上页)

```
func (this *BSTIterator) HasNext() bool {
        return len(this.stack) > 0
}
```

78.36 剑指 OfferII057. 值和下标之差都在给定的范围内 (2)

- 题目

```
给你一个整数数组 nums 和两个整数k 和 t 。请你判断是否存在 两个不同下标 i 和 j,
使得abs(nums[i] - nums[j]) <= t ，同时又满足 abs(i - j) <= k 。
如果存在则返回 true，不存在返回 false。
示例1：输入：nums = [1,2,3,1], k = 3, t = 0 输出：true
示例 2：输入：nums = [1,0,1,1], k = 1, t = 2 输出：true
示例 3：输入：nums = [1,5,9,1,5,9], k = 2, t = 3 输出：false
提示：0 <= nums.length <= 2 * 104
-231 <= nums[i] <= 231 - 1
0 <= k <= 104
0 <= t <= 231 - 1
注意：本题与主站 220题相同：
```

- 解题思路

```
func containsNearbyAlmostDuplicate(nums []int, k int, t int) bool {
        if len(nums) <= 1 {
                return false
        }
        for i := 0; i < len(nums); i++ {
                for j := i + 1; j < len(nums) && j <= i+k; j++ {
                        if abs(nums[i], nums[j]) <= t {
                                return true
                        }
                }
        }
        return false
}

# 2
func containsNearbyAlmostDuplicate(nums []int, k int, t int) bool {
        if len(nums) <= 1 || t < 0 {
                return false
        }
```

(续下页)

(接上页)

```
        m := make(map[int]int)
        width := t + 1
        for i := 0; i < len(nums); i++ {
                key := getKey(nums[i], width)
                if _, ok := m[key]; ok {
                        return true
                }
                if value, ok := m[key-1]; ok && abs(nums[i], value) < width {
                        return true
                }
                if value, ok := m[key+1]; ok && abs(nums[i], value) < width {
                        return true
                }
                m[key] = nums[i]
                if i >= k {
                        // 满足i和j的差的绝对值也小于等于k
                        delete(m, getKey(nums[i-k], width))
                }
        }
        return false
}

func getKey(value, width int) int {
        if value < 0 {
                return (value+1)/width - 1
        }
        return value / width
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

78.37 剑指 OfferII058. 日程表 (2)

- 题目

```
请实现一个 MyCalendar␣
↪类来存放你的日程安排。如果要添加的时间内没有其他安排，则可以存储这个新的日程安排。
MyCalendar 有一个 book(int start, int end)方法。它意味着在 start 到 end␣
↪时间内增加一个日程安排，
注意，这里的时间是半开区间，即 [start, end), 实数x 的范围为， start <= x < end。
当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生重复预订。
每次调用 MyCalendar.
↪book方法时，如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true。
否则，返回 false并且不要将该日程安排添加到日历中。
请按照以下步骤调用 MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.
↪book(start, end)
示例:输入: ["MyCalendar","book","book","book"] [[],[10,20],[15,25],[20,30]]
输出: [null,true,false,true]
解释: MyCalendar myCalendar = new MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false ，第二个日程安排不能添加到日历中，因为时间␣
↪15 已经被第一个日程安排预定了
MyCalendar.book(20, 30); // returns true␣
↪，第三个日程安排可以添加到日历中，因为第一个日程安排并不包含时间 20
提示: 每个测试用例，调用MyCalendar.book函数最多不超过1000次。
0 <= start < end <= 109
注意: 本题与主站 729题相同:
```

- 解题思路

```
type MyCalendar struct {
        arr [][2]int
}

func Constructor() MyCalendar {
        return MyCalendar{arr: make([][2]int, 0)}
}

func (this *MyCalendar) Book(start int, end int) bool {
        for i := 0; i < len(this.arr); i++ {
                if this.arr[i][0] < end && start < this.arr[i][1] {
                        return false
                }
        }
        this.arr = append(this.arr, [2]int{start, end})
```

(续下页)

(接上页)

```
        return true
}

# 2
type MyCalendar struct {
        root *Node
}

func Constructor() MyCalendar {
        return MyCalendar{root: nil}
}

func (this *MyCalendar) Book(start int, end int) bool {
        node := &Node{
                start: start,
                end:   end,
                left:  nil,
                right: nil,
        }
        if this.root == nil {
                this.root = node
                return true
        }
        return this.root.Insert(node)
}

type Node struct {
        start int
        end   int
        left  *Node
        right *Node
}

func (this *Node) Insert(node *Node) bool {
        if node.start >= this.end {
                if this.right == nil {
                        this.right = node
                        return true
                }
                return this.right.Insert(node)
        } else if node.end <= this.start {
                if this.left == nil {
                        this.left = node
```

(续下页)

(接上页)

```
                        return true
                }
                return this.left.Insert(node)
        }
        return false
}
```

78.38 剑指 OfferII060. 出现频率最高的 k 个数字 (3)

- 题目

```
给定一个整数数组 nums 和一个整数 k，请返回其中出现频率前 k 高的元素。可以按 任意顺序␣
↪返回答案。
示例 1:输入: nums = [1,1,1,2,2,3], k = 2 输出: [1,2]
示例 2:输入: nums = [1], k = 1 输出: [1]
提示：1 <= nums.length <= 105
k 的取值范围是 [1, 数组中不相同的元素的个数]
题目数据保证答案唯一，换句话说，数组中前 k 个高频元素的集合是唯一的
进阶：所设计算法的时间复杂度 必须 优于 O(n log n) ，其中 n是数组大小。
注意：本题与主站 347题相同
```

- 解题思路

```
func topKFrequent(nums []int, k int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        arr := make([][2]int, 0)
        for k, v := range m {
                arr = append(arr, [2]int{k, v})
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][1] > arr[j][1]
        })
        res := make([]int, 0)
        for i := 0; i < k; i++ {
                res = append(res, arr[i][0])
        }
        return res
}
```

(续下页)

(接上页)

```
# 2
func topKFrequent(nums []int, k int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        var h IntHeap
        heap.Init(&h)
        for k, v := range m {
                heap.Push(&h, [2]int{k, v})
        }
        res := make([]int, 0)
        for h.Len() > 0 && k > 0 {
                k--
                node := heap.Pop(&h).([2]int)
                res = append(res, node[0])
        }
        return res
}

type IntHeap [][2]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][1] > h[j][1] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([2]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 3
func topKFrequent(nums []int, k int) []int {
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
                m[nums[i]]++
        }
        arr := make([][]int, len(nums)+1)
        temp := make(map[int][]int)
        for key, value := range m {
```

(续下页)

(接上页)

```
                temp[value] = append(temp[value], key)
                arr[value] = append(arr[value], key)
        }
        res := make([]int, 0)
        for i := len(arr) - 1; i >= 0; i-- {
                // 避免出现0=>x次的情况
                if _, ok := temp[i]; ok {
                        for j := 0; j < len(arr[i]); j++ {
                                k--
                                if k < 0 {
                                        break
                                }
                                res = append(res, arr[i][j])
                        }
                }
        }
        return res
}
```

78.39 剑指 OfferII061. 和最小的 k 个数对 (2)

- 题目

```
给定两个以升序排列的整数数组 nums1 和 nums2,以及一个整数 k。
定义一对值(u,v)，其中第一个元素来自nums1，第二个元素来自 nums2。
请找到和最小的 k个数对(u1,v1), (u2,v2) ... (uk,vk)。
示例 1:输入: nums1 = [1,7,11], nums2 = [2,4,6], k = 3 输出: [1,2],[1,4],[1,6]
解释: 返回序列中的前 3 对数:
    [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
示例 2:输入: nums1 = [1,1,2], nums2 = [1,2,3], k = 2 输出: [1,1],[1,1]
解释: 返回序列中的前 2 对数:
    [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
示例 3:输入: nums1 = [1,2], nums2 = [3], k = 3  输出: [1,3],[2,3]
解释: 也可能序列中所有的数对都被返回:[1,3],[2,3]
提示:1 <= nums1.length, nums2.length <= 104
-109 <= nums1[i], nums2[i] <= 109
nums1, nums2 均为升序排列
1 <= k <= 1000
注意：本题与主站 373题相同：
```

- 解题思路

```
func kSmallestPairs(nums1 []int, nums2 []int, k int) [][]int {
        Heap := &NodeHeap{}
        heap.Init(Heap)
        for i := 0; i < len(nums1); i++ {
                for j := 0; j < len(nums2); j++ {
                        heap.Push(Heap, Node{
                                i: nums1[i],
                                j: nums2[j],
                        })
                        if Heap.Len() > k {
                                heap.Pop(Heap)
                        }
                }
        }
        res := make([][]int, 0)
        for Heap.Len() > 0 {
                node := heap.Pop(Heap).(Node)
                res = append(res, []int{node.i, node.j})
        }
        return res
}

type Node struct {
        i int
        j int
}

type NodeHeap []Node

func (h NodeHeap) Len() int {
        return len(h)
}

func (h NodeHeap) Less(i, j int) bool {
        return h[i].i+h[i].j > h[j].i+h[j].j
}

func (h NodeHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *NodeHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}
```

(续下页)

(接上页)

```
func (h *NodeHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func kSmallestPairs(nums1 []int, nums2 []int, k int) [][]int {
        arr := make([][]int, 0)
        for i := 0; i < len(nums1); i++ {
                for j := 0; j < len(nums2); j++ {
                        arr = append(arr, []int{nums1[i], nums2[j]})
                }
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i][0]+arr[i][1] < arr[j][0]+arr[j][1]
        })
        if len(arr) < k {
                return arr
        }
        return arr[:k]
}
```

78.40 剑指 OfferII062. 实现前缀树 (2)

- 题目

```
Trie（发音类似 "try"）或者说 前缀树
↪是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。
这一数据结构有相当多的应用情景，例如自动补完和拼写检查。
请你实现 Trie 类：
Trie() 初始化前缀树对象。
void insert(String word) 向前缀树中插入字符串 word 。
boolean search(String word) 如果字符串 word 在前缀树中，返回
↪true（即，在检索之前已经插入）；否则，返回 false 。
boolean startsWith(String prefix) 如果之前已经插入的字符串word 的前缀之一为 prefix
↪，返回 true ；否则，返回 false 。
示例：输入inputs = ["Trie", "insert", "search", "search", "startsWith", "insert",
↪"search"]
inputs = [[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出[null, null, true, false, true, null, true]
```

(续下页)

(接上页)

```
解释 Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True
提示：1 <= word.length, prefix.length <= 2000
word 和 prefix 仅由小写英文字母组成
insert、search 和 startsWith 调用次数 总计 不超过 3 * 104 次
注意：本题与主站 208 题相同：
```

- 解题思路

```
type Trie struct {
        next   [26]*Trie
        ending int
}

func Constructor() Trie {
        return Trie{
                next:   [26]*Trie{},
                ending: 0,
        }
}

func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

func (this *Trie) Search(word string) bool {
        temp := this
        for _, v := range word {
```

(续下页)

(接上页)

```
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}

func (this *Trie) StartsWith(prefix string) bool {
        temp := this
        for _, v := range prefix {
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        return true
}

# 2
type Trie struct {
        next   map[byte]*Trie
        ending int
}

/** Initialize your data structure here. */
func Constructor() Trie {
        return Trie{
                next:   make(map[byte]*Trie),
                ending: 0,
        }
}

/** Inserts a word into the trie. */
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := byte(v - 'a')
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
```

(续下页)

(接上页)

```
                                next:   make(map[byte]*Trie),
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

/** Returns if the word is in the trie. */
func (this *Trie) Search(word string) bool {
        temp := this
        for _, v := range word {
                value := byte(v - 'a')
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}

/** Returns if there is any word in the trie that starts with the given prefix. */
func (this *Trie) StartsWith(prefix string) bool {
        temp := this
        for _, v := range prefix {
                value := byte(v - 'a')
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        return true
}
```

78.41 剑指 OfferII063. 替换单词 (2)

- 题目

```
在英语中，有一个叫做词根(root)␣
↪的概念，它可以跟着其他一些词组成另一个较长的单词——我们称这个词为继承词(successor)。
例如，词根an，跟随着单词other(其他)，可以形成新的单词another(另一个)。
现在，给定一个由许多词根组成的词典和一个句子，需要将句子中的所有继承词用词根替换掉。
如果继承词有许多可以形成它的词根，则用最短的词根替换它。
需要输出替换之后的句子。
示例 1：输入：dictionary = ["cat","bat","rat"], sentence = "the cattle was rattled by␣
↪the battery"
输出："the cat was rat by the bat"
示例 2：输入：dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"␣
↪输出："a a b c"
示例 3：
输入：dictionary = ["a", "aa", "aaa", "aaaa"], sentence = "a aa a aaaa aaa aaa aaa␣
↪aaaaaa bbb baba ababa"
输出："a a a a a a a bbb baba a"
示例 4：输入：dictionary = ["catt","cat","bat","rat"], sentence = "the cattle was␣
↪rattled by the battery"
输出："the cat was rat by the bat"
示例 5：输入：dictionary = ["ac","ab"], sentence = "it is abnormal that this solution␣
↪is accepted"
输出："it is ab that this solution is ac"
提示：1 <= dictionary.length<= 1000
1 <= dictionary[i].length <= 100
dictionary[i]仅由小写字母组成。
1 <= sentence.length <= 10^6
sentence仅由小写字母和空格组成。
sentence 中单词的总量在范围 [1, 1000] 内。
sentence 中每个单词的长度在范围 [1, 1000] 内。
sentence 中单词之间由一个空格隔开。
sentence没有前导或尾随空格。
注意：本题与主站 648题相同：
```

- 解题思路

```
func replaceWords(dictionary []string, sentence string) string {
        sort.Strings(dictionary)
        arr := strings.Split(sentence, " ")
        for i := 0; i < len(arr); i++ {
                for _, v := range dictionary {
                        if strings.HasPrefix(arr[i], v) {
```

(续下页)

(接上页)

```
                                arr[i] = v
                                break
                        }
                }
        }
        return strings.Join(arr, " ")
}

# 2
func replaceWords(dictionary []string, sentence string) string {
        trie := Constructor()
        for i := 0; i < len(dictionary); i++ {
                trie.Insert(dictionary[i])
        }
        arr := strings.Split(sentence, " ")
        for i := 0; i < len(arr); i++ {
                result := trie.Search(arr[i])
                if result != "" {
                        arr[i] = result
                }
        }
        return strings.Join(arr, " ")
}

type Trie struct {
        next   [26]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int       // 次数（可以改为bool）
}

func Constructor() Trie {
        return Trie{
                next:   [26]*Trie{},
                ending: 0,
        }
}

// 插入word
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
```

(续下页)

(接上页)

```
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

// 查找
func (this *Trie) Search(word string) string {
        temp := this
        res := ""
        for _, v := range word {
                res = res + string(v)
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return ""
                }
                if temp.ending > 0 {
                        return res
                }
        }
        return ""
}
```

78.42 剑指 OfferII064. 神奇的字典 (3)

- 题目

```
设计一个使用单词列表进行初始化的数据结构，单词列表中的单词 互不相同 。
如果给出一个单词，请判定能否只将这个单词中一个字母换成另一个字母，使得所形成的新单词存在于已构建的神
实现 MagicDictionary 类：
MagicDictionary() 初始化对象
void buildDict(String[]dictionary) 使用字符串数组dictionary␣
↪设定该数据结构，dictionary 中的字符串互不相同
bool search(String searchWord) 给定一个字符串 searchWord ，
判定能否只将字符串中 一个␣
↪字母换成另一个字母，使得所形成的新字符串能够与字典中的任一字符串匹配。
如果可以，返回 true ；否则，返回 false 。
示例：输入inputs = ["MagicDictionary", "buildDict", "search", "search", "search",
↪"search"]
```

(续下页)

(接上页)

```
inputs = [[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]
输出[null, null, false, true, false, false]
解释MagicDictionary magicDictionary = new MagicDictionary();
magicDictionary.buildDict(["hello", "leetcode"]);
magicDictionary.search("hello"); // 返回 False
magicDictionary.search("hhllo"); // 将第二个 'h' 替换为 'e' 可以匹配 "hello"
↪, 所以返回 True
magicDictionary.search("hell"); // 返回 False
magicDictionary.search("leetcoded"); // 返回 False
提示：1 <=dictionary.length <= 100
1 <=dictionary[i].length <= 100
dictionary[i] 仅由小写英文字母组成
dictionary 中的所有字符串 互不相同
1 <=searchWord.length <= 100
searchWord 仅由小写英文字母组成
buildDict 仅在 search 之前调用一次
最多调用 100 次 search
注意：本题与主站 676题相同：
```

- 解题思路

```
type MagicDictionary struct {
        m map[int][]string
}

func Constructor() MagicDictionary {
        return MagicDictionary{m: map[int][]string{}}
}

func (this *MagicDictionary) BuildDict(dictionary []string) {
        for i := 0; i < len(dictionary); i++ {
                this.m[len(dictionary[i])] = append(this.m[len(dictionary[i])],
↪dictionary[i])
        }
}

func (this *MagicDictionary) Search(searchWord string) bool {
        if len(this.m[len(searchWord)]) == 0 {
                return false
        }
        for i := 0; i < len(this.m[len(searchWord)]); i++ {
                word := this.m[len(searchWord)][i]
                count := 0
                for j := 0; j < len(searchWord); j++ {
```

(续下页)

(接上页)

```
                        if word[j] != searchWord[j] {
                                count++
                                if count > 1 {
                                        break
                                }
                        }
                }
                if count == 1 {
                        return true
                }
        }
        return false
}

# 2
type MagicDictionary struct {
        arr []string
}

func Constructor() MagicDictionary {
        return MagicDictionary{arr: make([]string, 0)}
}

func (this *MagicDictionary) BuildDict(dictionary []string) {
        this.arr = dictionary
}

func (this *MagicDictionary) Search(searchWord string) bool {
        for i := 0; i < len(this.arr); i++ {
                word := this.arr[i]
                if len(word) != len(searchWord) {
                        continue
                }
                count := 0
                for j := 0; j < len(searchWord); j++ {
                        if word[j] != searchWord[j] {
                                count++
                                if count > 1 {
                                        break
                                }
                        }
                }
                if count == 1 {
```

(续下页)

(接上页)

```
                        return true
                }
        }
        return false
}

# 3
type MagicDictionary struct {
        next   [26]*MagicDictionary // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int                  // 次数（可以改为bool）
}

func Constructor() MagicDictionary {
        return MagicDictionary{
                next:   [26]*MagicDictionary{},
                ending: 0,
        }
}

func (this *MagicDictionary) BuildDict(dictionary []string) {
        for i := 0; i < len(dictionary); i++ {
                word := dictionary[i]
                temp := this
                for _, v := range word {
                        value := v - 'a'
                        if temp.next[value] == nil {
                                temp.next[value] = &MagicDictionary{
                                        next:   [26]*MagicDictionary{},
                                        ending: 0,
                                }
                        }
                        temp = temp.next[value]
                }
                temp.ending++
        }
}

func (this *MagicDictionary) Search(searchWord string) bool {
        cur := this
        arr := []byte(searchWord)
        for i := 0; i < len(searchWord); i++ {
                b := searchWord[i]
                for j := 0; j < 26; j++ {
```

(续下页)

(接上页)

```
                        if j+'a' == int(b) {
                                continue
                        }
                        arr[i] = byte('a' + j)
                        if cur.SearchWord(string(arr[i:])) == true {
                                return true
                        }
                }
                arr[i] = b
                if cur.next[int(b-'a')] == nil {
                        return false
                }
                cur = cur.next[int(b-'a')]
        }
        return false
}

func (this *MagicDictionary) SearchWord(word string) bool {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return false
                }
        }
        if temp.ending > 0 {
                return true
        }
        return false
}
```

78.43 剑指 OfferII065. 最短的单词编码 (3)

- 题目

```
单词数组words 的 有效编码 由任意助记字符串 s 和下标数组 indices 组成，且满足：
words.length == indices.length
助记字符串 s 以 '#' 字符结尾
对于每个下标 indices[i] ，s 的一个从 indices[i] 开始、
到下一个 '#' 字符结束（但不包括 '#'）的 子字符串 恰好与 words[i] 相等
给定一个单词数组words ，返回成功对 words 进行编码的最小助记字符串 s 的长度 。
示例 1：输入：words = ["time", "me", "bell"] 输出：10
```

(续下页)

(接上页)

```
解释：一组有效编码为 s = "time#bell#" 和 indices = [0, 2, 5] 。
words[0] = "time" ，s 开始于 indices[0] = 0 到下一个 '#'␣
↪结束的子字符串，如加粗部分所示 "time#bell#"
words[1] = "me" ，s 开始于 indices[1] = 2 到下一个 '#' 结束的子字符串，如加粗部分所示
↪"time#bell#"
words[2] = "bell" ，s 开始于 indices[2] = 5 到下一个 '#'␣
↪结束的子字符串，如加粗部分所示 "time#bell#"
示例 2：输入：words = ["t"] 输出：2
解释：一组有效编码为 s = "t#" 和 indices = [0] 。
提示：1 <= words.length <= 2000
1 <= words[i].length <= 7
words[i] 仅由小写字母组成
注意：本题与主站 820题相同：
```

- 解题思路

```
func minimumLengthEncoding(words []string) int {
        res := 0
        m := make(map[string]bool)
        for i := 0; i < len(words); i++ {
                m[words[i]] = true
        }
        for k := range m {
                for i := 1; i < len(k); i++ {
                        delete(m, k[i:])
                }
        }
        for k := range m {
                res = res + len(k) + 1
        }
        return res
}

# 2
func minimumLengthEncoding(words []string) int {
        res := 0
        m := make(map[string]bool)
        for i := 0; i < len(words); i++ {
                m[words[i]] = true
        }
        words = make([]string, 0)
        for k := range m {
                words = append(words, k)
        }
```

(续下页)

(接上页)

```
        sort.Slice(words, func(i, j int) bool {
                return len(words[i]) < len(words[j])
        })
        for i := len(words) - 1; i >= 0; i-- {
                if m[words[i]] == false {
                        continue
                }
                for j := i - 1; j >= 0; j-- {
                        if strings.HasSuffix(words[i], words[j]) == true {
                                m[words[j]] = false
                        }
                }
        }
        for k := range m {
                if m[k] == true {
                        res = res + len(k) + 1
                }
        }
        return res
}

# 3
func minimumLengthEncoding(words []string) int {
        res := 0
        arr := make([]string, 0)
        for k := range words {
                arr = append(arr, reverse(words[k]))
        }
        sort.Strings(arr)
        for i := 0; i < len(arr)-1; i++ {
                length := len(arr[i])
                if length <= len(arr[i+1]) && arr[i] == arr[i+1][:length] {
                        continue
                }
                res = res + length + 1
        }
        return res + len(arr[len(arr)-1]) + 1
}

func reverse(str string) string {
        res := make([]byte, 0)
        for i := len(str) - 1; i >= 0; i-- {
                res = append(res, str[i])
```

(续下页)

(接上页)

```
        }
        return string(res)
}
```

78.44 剑指 OfferII066. 单词之和 (3)

- 题目

```
实现一个 MapSum 类，支持两个方法，insert和sum:
MapSum() 初始化 MapSum 对象
void insert(String key, int val) 插入 key-val 键值对，字符串表示键 key ，整数表示值␣
↪val 。
如果键 key 已经存在，那么原来的键值对将被替代成新的键值对。
int sum(string prefix) 返回所有以该前缀 prefix 开头的键 key 的值的总和。
示例：输入：inputs = ["MapSum", "insert", "sum", "insert", "sum"]
inputs = [[], ["apple", 3], ["ap"], ["app", 2], ["ap"]]
输出：[null, null, 3, null, 5]
解释：MapSum mapSum = new MapSum();
mapSum.insert("apple", 3);
mapSum.sum("ap");           // return 3 (apple = 3)
mapSum.insert("app", 2);
mapSum.sum("ap");           // return 5 (apple + app = 3 + 2 = 5)
提示：1 <= key.length, prefix.length <= 50
key 和 prefix 仅由小写英文字母组成
1 <= val <= 1000
最多调用 50 次 insert 和 sum
注意：本题与主站 677题相同
```

- 解题思路

```
type MapSum struct {
        val  int
        next map[int32]*MapSum
}

func Constructor() MapSum {
        return MapSum{
                val:  0,
                next: make(map[int32]*MapSum),
        }
}
```

(续下页)

(接上页)

```
func (this *MapSum) Insert(key string, val int) {
        node := this
        for _, v := range key {
                if _, ok := node.next[v]; ok == false {
                        temp := Constructor()
                        node.next[v] = &temp
                }
                node = node.next[v]
        }
        node.val = val
}

func (this *MapSum) Sum(prefix string) int {
        node := this
        for _, v := range prefix {
                if _, ok := node.next[v]; ok == false {
                        return 0
                }
                node = node.next[v]
        }
        res := 0
        queue := make([]*MapSum, 0)
        queue = append(queue, node)
        for len(queue) > 0 {
                temp := queue[0]
                queue = queue[1:]
                res = res + temp.val
                for _, v := range temp.next {
                        queue = append(queue, v)
                }
        }
        return res
}

# 2
type MapSum struct {
        m    map[string]int
        data map[string]map[string]bool
}

func Constructor() MapSum {
        return MapSum{
                m:    make(map[string]int),
```

(续下页)

(接上页)

```
                data: make(map[string]map[string]bool),
        }
}

func (this *MapSum) Insert(key string, val int) {
        this.m[key] = val
        for i := 1; i <= len(key); i++ {
                str := key[:i]
                if _, ok := this.data[str]; ok == false {
                        this.data[str] = make(map[string]bool)
                }
                this.data[str][key] = true
        }
}

func (this *MapSum) Sum(prefix string) int {
        res := 0
        for key := range this.data[prefix] {
                res = res + this.m[key]
        }
        return res
}

# 3
type MapSum struct {
        m map[string]int
}

func Constructor() MapSum {
        return MapSum{
                m: make(map[string]int),
        }
}

func (this *MapSum) Insert(key string, val int) {
        this.m[key] = val
}

func (this *MapSum) Sum(prefix string) int {
        res := 0
        for key, value := range this.m {
                if strings.HasPrefix(key, prefix) {
                        res = res + value
```

(续下页)

(接上页)

```
                }
        }
        return res
}
```

78.45 剑指 OfferII067. 最大的异或 (2)

- 题目

```
给定一个整数数组 nums ，返回 nums[i] XOR nums[j] 的最大运算结果，其中 0 ≤ i ≤ j < n 。
示例 1：输入：nums = [3,10,5,25,2,8] 输出：28
解释：最大运算结果是 5 XOR 25 = 28.
示例 2：输入：nums = [0] 输出：0
示例 3：输入：nums = [2,4] 输出：6
示例 4：输入：nums = [8,10,2] 输出：10
示例 5：输入：nums = [14,70,53,83,49,91,36,80,92,51,66,70] 输出：127
提示：1 <= nums.length <= 2 * 104
0 <= nums[i] <= 231 - 1
进阶：你可以在 O(n) 的时间解决这个问题吗？
注意：本题与主站 421题相同：
```

- 解题思路

```
func findMaximumXOR(nums []int) int {
        res := 0
        target := 0
        for i := 31; i >= 0; i-- { // 枚举每一位（第i位，从右到左），判断该为能否为1
                m := make(map[int]bool)
                target = target | (1 << i) // target第i位置1
                for j := 0; j < len(nums); j++ {
                        m[nums[j]&target] = true // 高位&：取前缀
                }
                temp := res | (1 << i) // 假设结果第i位为1
                // a ^ b = temp
                // temp ^ a = b
                for k := range m {
                        if _, ok := m[temp^k]; ok {
                                res = temp // 能取到1
                                break
                        }
                }
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func findMaximumXOR(nums []int) int {
        n := len(nums)
        if n <= 1 {
                return 0
        }
        res := 0
        root := Trie{
                next: make([]*Trie, 2), // 0和1
        }
        for i := 0; i < n; i++ {
                temp := &root
                for j := 31; j >= 0; j-- {
                        value := (nums[i] >> j) & 1
                        if temp.next[value] == nil {
                                temp.next[value] = &Trie{
                                        next: make([]*Trie, 2),
                                }
                        }
                        temp = temp.next[value]
                }
        }
        for i := 0; i < n; i++ {
                temp := &root
                cur := 0
                for j := 31; j >= 0; j-- {
                        value := (nums[i] >> j) & 1
                        if temp.next[value^1] != nil { // 能取到1
                                cur = cur | (1 << j) // 结果在该位可以为1
                                temp = temp.next[value^1]
                        } else {
                                temp = temp.next[value]
                        }
                }
                res = max(res, cur)
        }
        return res
}

type Trie struct {
```

(续下页)

(接上页)

```
        next []*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

78.46 剑指 OfferII070. 排序数组中只出现一次的数字 (3)

- 题目

```
给定一个只包含整数的有序数组␣
↪nums，每个元素都会出现两次，唯有一个数只会出现一次，请找出这个唯一的数字。
示例 1:输入: nums = [1,1,2,3,3,4,4,8,8] 输出: 2
示例 2:输入: nums =  [3,3,7,7,10,11,11] 输出: 10
提示:1 <= nums.length <= 105
0 <= nums[i]<= 105
进阶:采用的方案可以在 O(log n) 时间复杂度和 O(1) 空间复杂度中运行吗？
注意: 本题与主站 540题相同:
```

- 解题思路

```
func singleNonDuplicate(nums []int) int {
        for i := 0; i < len(nums)-1; i = i + 2 {
                if nums[i] != nums[i+1] {
                        return nums[i]
                }
        }
        return nums[len(nums)-1]
}

# 2
func singleNonDuplicate(nums []int) int {
        n := len(nums)
        left, right := 0, n-1
        for left < right {
                mid := left + (right-left)/2
                if mid%2 == 1 {
                        mid--
```

(续下页)

(接上页)

```
                }
                if nums[mid] == nums[mid+1] {
                        left = mid + 2
                } else {
                        right = mid
                }
        }
        return nums[left]
}

# 3
func singleNonDuplicate(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                res = res ^ nums[i]
        }
        return res
}
```

78.47 剑指 OfferII071. 按权重生成随机数 (1)

- 题目

```
给定一个正整数数组w ，其中w[i]代表下标 i的权重（下标从 0 开始），
请写一个函数pickIndex，它可以随机地获取下标 i，选取下标 i的概率与w[i]成正比。
例如，对于 w = [1, 3]，挑选下标 0 的概率为 1 / (1 + 3)= 0.25 （即，25%），
而选取下标 1 的概率为 3 / (1 + 3)= 0.75（即，75%）。
也就是说，选取下标 i 的概率为 w[i] / sum(w) 。
示例 1：输入：inputs = ["Solution","pickIndex"] inputs = [[[1]],[]] 输出：[null,0]
解释：Solution solution = new Solution([1]);
solution.pickIndex(); // 返回 0，因为数组中只有一个元素，所以唯一的选择是返回下标 0。
示例 2：输入：inputs = ["Solution","pickIndex","pickIndex","pickIndex","pickIndex",
↪"pickIndex"]
inputs = [[[1,3]],[],[],[],[],[]]
输出：[null,1,1,1,1,0]
解释：Solution solution = new Solution([1, 3]);
solution.pickIndex(); // 返回 1，返回下标 1，返回该下标概率为 3/4 。
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 0，返回下标 0，返回该下标概率为 1/4 。
由于这是一个随机问题，允许多个答案，因此下列输出都可以被认为是正确的：
```

(续下页)

(接上页)

```
[null,1,1,1,1,0]
[null,1,1,1,1,1]
[null,1,1,1,0,0]
[null,1,1,1,0,1]
[null,1,0,1,0,0]
......
诸若此类。
提示：1 <= w.length <= 10000
1 <= w[i] <= 10^5
pickIndex将被调用不超过10000次
注意：本题与主站 528题相同：
```

- 解题思路

```
type Solution struct {
        nums  []int
        total int
}

func Constructor(w []int) Solution {
        total := 0
        arr := make([]int, len(w)) // 前缀和
        for i := 0; i < len(w); i++ {
                total = total + w[i]
                arr[i] = total
        }
        return Solution{
                nums:  arr,
                total: total,
        }
}

func (this *Solution) PickIndex() int {
        target := rand.Intn(this.total)
        left, right := 0, len(this.nums)
        for left < right {
                mid := left + (right-left)/2
                if this.nums[mid] <= target {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
```

(续下页)

(接上页)

```
}
```

78.48 剑指 OfferII073. 狒狒吃香蕉 (2)

- 题目

```
狒狒喜欢吃香蕉。这里有N堆香蕉，第 i␣
↪堆中有piles[i]根香蕉。警卫已经离开了，将在H小时后回来。
狒狒可以决定她吃香蕉的速度K（单位：根/小时）。每个小时，她将会选择一堆香蕉，从中吃掉␣
↪K 根。
如果这堆香蕉少于 K␣
↪根，她将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉，下一个小时才会开始吃另一堆的香蕉。
狒狒喜欢慢慢吃，但仍然想在警卫回来前吃掉所有的香蕉。
返回她可以在 H 小时内吃掉所有香蕉的最小速度 K（K 为整数）。
示例 1：输入：piles = [3,6,7,11], H = 8 输出：4
示例2：输入：piles = [30,11,23,4,20], H = 5 输出：30
示例3：输入：piles = [30,11,23,4,20], H = 6 输出：23
提示：1 <= piles.length <= 10^4
piles.length <= H <= 10^9
1 <= piles[i] <= 10^9
注意：本题与主站 875题相同：
```

- 解题思路

```
func minEatingSpeed(piles []int, h int) int {
        maxValue := piles[0]
        for i := 1; i < len(piles); i++ {
                maxValue = max(maxValue, piles[i])
        }
        left, right := 1, maxValue
        for left < right {
                mid := left + (right-left)/2
                if judge(piles, mid, h) == true {
                        left = mid + 1
                } else {
                        right = mid
                }
        }
        return left
}

func judge(piles []int, speed int, H int) bool {
```

(续下页)

(接上页)

```
        total := 0
        for i := 0; i < len(piles); i++ {
                total = total + piles[i]/speed
                if piles[i]%speed > 0 {
                        total = total + 1
                }
        }
        return total > H
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func minEatingSpeed(piles []int, h int) int {
        maxValue := piles[0]
        for i := 1; i < len(piles); i++ {
                maxValue = max(maxValue, piles[i])
        }
        return sort.Search(maxValue, func(speed int) bool {
                if speed == 0 { // 避免除0
                        return false
                }
                total := 0
                for i := 0; i < len(piles); i++ {
                        total = total + piles[i]/speed
                        if piles[i]%speed > 0 {
                                total = total + 1
                        }
                }
                return total <= h
        })

}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

78.49 剑指 OfferII074. 合并区间 (2)

- 题目

```
以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。
请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。
示例 1：输入：intervals = [[1,3],[2,6],[8,10],[15,18]] 输出：[[1,6],[8,10],[15,18]]
解释：区间 [1,3] 和 [2,6] 重叠，将它们合并为 [1,6].
示例2：输入：intervals = [[1,4],[4,5]] 输出：[[1,5]]
解释：区间 [1,4] 和 [4,5] 可被视为重叠区间。
提示：1 <= intervals.length <= 104
intervals[i].length == 2
0 <= starti <= endi <= 104
注意：本题与主站 56题相同：
```

- 解题思路

```
func merge(intervals [][]int) [][]int {
        res := make([][]int, 0)
        if len(intervals) == 0 {
                return nil
        }
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][0] < intervals[j][0]
        })
        res = append(res, intervals[0])
        for i := 1; i < len(intervals); i++ {
                arr := res[len(res)-1]
                if intervals[i][0] > arr[1] {
                        res = append(res, intervals[i])
                } else if intervals[i][1] > arr[1] {
                        res[len(res)-1][1] = intervals[i][1]
                }
        }
        return res
}

# 2
func merge(intervals [][]int) [][]int {
```

(续下页)

(接上页)

```
        res := make([][]int, 0)
        if len(intervals) == 0 {
                return nil
        }
        sort.Slice(intervals, func(i, j int) bool {
                return intervals[i][0] < intervals[j][0]
        })
        for i := 0; i < len(intervals); {
                end := intervals[i][1]
                j := i + 1
                for j < len(intervals) && intervals[j][0] <= end {
                        if intervals[j][1] > end {
                                end = intervals[j][1]
                        }
                        j++
                }
                res = append(res, []int{intervals[i][0], end})
                i = j
        }
        return res
}
```

78.50 剑指 OfferII076. 数组中的第 k 大的数字 (3)

- 题目

```
给定整数数组 nums 和整数 k，请返回数组中第 k 个最大的元素。
请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。
示例 1:输入: [3,2,1,5,6,4] 和 k = 2 输出: 5
示例2:输入: [3,2,3,1,2,4,5,5,6] 和 k = 4 输出: 4
提示：1 <= k <= nums.length <= 104
-104<= nums[i] <= 104
注意：本题与主站 215题相同：
```

- 解题思路

```
func findKthLargest(nums []int, k int) int {
        sort.Ints(nums)
        return nums[len(nums)-k]
}

# 2
```

(续下页)

(接上页)

```
func findKthLargest(nums []int, k int) int {
        heapSize := len(nums)
        buildMaxHeap(nums, heapSize)
        for i := len(nums) - 1; i >= len(nums)-k+1; i-- {
                nums[0], nums[i] = nums[i], nums[0]
                heapSize--
                maxHeapify(nums, 0, heapSize)
        }
        return nums[0]
}

func buildMaxHeap(a []int, heapSize int) {
        for i := heapSize / 2; i >= 0; i-- {
                maxHeapify(a, i, heapSize)
        }
}

func maxHeapify(a []int, i, heapSize int) {
        l, r, largest := i*2+1, i*2+2, i
        if l < heapSize && a[l] > a[largest] {
                largest = l
        }
        if r < heapSize && a[r] > a[largest] {
                largest = r
        }
        if largest != i {
                a[i], a[largest] = a[largest], a[i]
                maxHeapify(a, largest, heapSize)
        }
}

# 3
func findKthLargest(nums []int, k int) int {
        return findK(nums, 0, len(nums)-1, k)
}

func findK(nums []int, start, end int, k int) int {
        if start >= end {
                return nums[end]
        }
        index := partition(nums, start, end)
        if index+1 == k {
                return nums[index]
```

(续下页)

(接上页)

```
        } else if index+1 < k {
                return findK(nums, index+1, end, k)
        }
        return findK(nums, start, index-1, k)
}

func partition(nums []int, start, end int) int {
        temp := nums[end]
        i := start
        for j := start; j < end; j++ {
                if nums[j] > temp {
                        if i != j {
                                nums[i], nums[j] = nums[j], nums[i]
                        }
                        i++
                }
        }
        nums[i], nums[end] = nums[end], nums[i]
        return i
}
```

78.51 剑指 OfferII077. 链表排序 (3)

- 题目

```
给定链表的头结点head，请将其按 升序 排列并返回 排序后的链表 。
示例 1：输入：head = [4,2,1,3] 输出：[1,2,3,4]
示例 2：输入：head = [-1,5,3,4,0] 输出：[-1,0,3,4,5]
示例 3：输入：head = [] 输出：[]
提示：链表中节点的数目在范围[0, 5 * 104]内
-105<= Node.val <= 105
进阶：你可以在O(nlogn) 时间复杂度和常数级空间复杂度下，对链表进行排序吗？
注意：本题与主站 148题相同
```

- 解题思路

```
func sortList(head *ListNode) *ListNode {
        quickSort(head, nil)
        return head
}

func quickSort(head, end *ListNode) {
```

(续下页)

(接上页)

```
        if head == end || head.Next == end {
                return
        }
        temp := head.Val
        fast, slow := head.Next, head
        for fast != end {
                if fast.Val < temp {
                        slow = slow.Next
                        slow.Val, fast.Val = fast.Val, slow.Val
                }
                fast = fast.Next
        }
        slow.Val, head.Val = head.Val, slow.Val
        quickSort(head, slow)
        quickSort(slow.Next, end)
}

# 2
func sortList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        slow, fast := head, head.Next
        for fast != nil && fast.Next != nil {
                slow = slow.Next
                fast = fast.Next.Next
        }
        right := sortList(slow.Next)
        slow.Next = nil
        left := sortList(head)
        return mergeTwoLists(left, right)
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
```

(续下页)

(接上页)

```
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}

# 3
func sortList(head *ListNode) *ListNode {
        if head == nil || head.Next == nil {
                return head
        }
        res := &ListNode{Next: head}
        cur := head
        var left, right *ListNode
        length := 0
        for cur != nil {
                length++
                cur = cur.Next
        }
        for i := 1; i < length; i = i * 2 {
                cur = res.Next
                tail := res
                for cur != nil {
                        left = cur
                        right = split(left, i)
                        cur = split(right, i)
                        tail.Next = mergeTwoLists(left, right)
                        for tail.Next != nil {
                                tail = tail.Next
                        }
                }
        }
        return res.Next
}

func split(head *ListNode, length int) *ListNode {
        cur := head
        var right *ListNode
```

(续下页)

(接上页)

```
        length--
        for length > 0 && cur != nil {
                length--
                cur = cur.Next
        }
        if cur == nil {
                return nil
        }
        right = cur.Next
        cur.Next = nil
        return right
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}
```

78.52 剑指 OfferII079. 所有子集 (4)

- 题目

```
给定一个整数数组nums ，数组中的元素 互不相同 。返回该数组所有可能的子集（幂集）。
解集 不能 包含重复的子集。你可以按 任意顺序 返回解集。
示例 1: 输入: nums = [1,2,3] 输出: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
示例 2: 输入: nums = [0] 输出: [[],[0]]
```

(续下页)

(接上页)

```
提示：1 <= nums.length <= 10
-10 <= nums[i] <= 10
nums 中的所有元素 互不相同
注意：本题与主站 78题相同：
```

- 解题思路

```
var res [][]int

func subsets(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, make([]int, 0), 0)
        return res
}

func dfs(nums []int, arr []int, level int) {
        temp := make([]int, len(arr))
        copy(temp, arr)
        res = append(res, temp)
        for i := level; i < len(nums); i++ {
                dfs(nums, append(arr, nums[i]), i+1)
        }
}

# 2
func subsets(nums []int) [][]int {
        res := make([][]int, 0)
        res = append(res, []int{})
        for i := 0; i < len(nums); i++ {
                temp := make([][]int, len(res))
                for key, value := range res {
                        value = append(value, nums[i])
                        temp[key] = append(temp[key], value...)
                }
                for _, v := range temp {
                        res = append(res, v)
                }
        }
        return res
}

# 3
func subsets(nums []int) [][]int {
        res := make([][]int, 0)
```

(续下页)

(接上页)

```
        n := len(nums)
        left := 1 << n
        right := 1 << (n + 1)
        for i := left; i < right; i++ {
                temp := make([]int, 0)
                for j := 0; j < n; j++ {
                        if i&(1<<j) != 0 {
                                temp = append(temp, nums[j])
                        }
                }
                res = append(res, temp)
        }
        return res
}

# 4
var res [][]int

func subsets(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, make([]int, 0), 0)
        return res
}

func dfs(nums []int, arr []int, level int) {
        if level >= len(nums) {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        dfs(nums, arr, level+1)
        dfs(nums, append(arr, nums[level]), level+1)
}
```

78.53 剑指 OfferII080. 含有 k 个元素的组合 (5)

- 题目

```
给定两个整数 n 和 k，返回 1 ... n 中所有可能的 k 个数的组合。
示例 1:输入:n = 4, k = 2 输出:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
示例 2:输入:n = 1, k = 1 输出: [[1]]
提示:1 <= n <= 20
1 <= k <= n
注意：本题与主站 77题相同：
```

- 解题思路

```
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        nums := make([]int, 0)
        for i := 1; i <= n; i++ {
                nums = append(nums, i)
        }
        dfs(nums, 0, k)
        return res
}

func dfs(nums []int, index, k int) {
        if index == k {
                temp := make([]int, k)
                copy(temp, nums[:k])
                res = append(res, temp)
                return
        }
        for i := index; i < len(nums); i++ {
                if index == 0 || nums[i] > nums[index-1] {
                        nums[i], nums[index] = nums[index], nums[i]
                        dfs(nums, index+1, k)
```

(续下页)

(接上页)

```
                        nums[i], nums[index] = nums[index], nums[i]
                }
        }
}

# 2
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        dfs(n, k, 1, make([]int, 0))
        return res
}

func dfs(n, k, index int, arr []int) {
        if len(arr) == k {
                temp := make([]int, k)
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i <= n; i++ {
                arr = append(arr, i)
                dfs(n, k, i+1, arr)
                arr = arr[:len(arr)-1]
        }
}

# 3
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        nums := make([]int, 0)
        for i := 1; i <= n; i++ {
                nums = append(nums, i)
        }
        dfs(nums, 0, k, make([]int, 0))
        return res
}

func dfs(nums []int, index, k int, arr []int) {
        if len(arr) == k {
```

(续下页)

(接上页)

```
                temp := make([]int, k)
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(nums); i++ {
                arr = append(arr, nums[i])
                dfs(nums, i+1, k, arr)
                arr = arr[:len(arr)-1]
        }
}

# 4
func combine(n int, k int) [][]int {
        res := make([][]int, 0)
        arr := make([]int, 0)
        for i := 1; i <= k; i++ {
                arr = append(arr, 0)
        }
        i := 0
        for i >= 0 {
                arr[i]++
                if arr[i] > n {
                        i--
                } else if i == k-1 {
                        temp := make([]int, k)
                        copy(temp, arr)
                        res = append(res, temp)
                } else {
                        i++
                        arr[i] = arr[i-1]
                }
        }
        return res
}

# 5
var res [][]int

func combine(n int, k int) [][]int {
        res = make([][]int, 0)
        dfs(n, k, 1, make([]int, 0))
        return res
```

(续下页)

(接上页)

```
}

func dfs(n, k, index int, arr []int) {
        if index > n+1 {
                return
        }
        if len(arr) == k {
                temp := make([]int, k)
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        dfs(n, k, index+1, arr)
        arr = append(arr, index)
        dfs(n, k, index+1, arr)
}
```

78.54 剑指 OfferII081. 允许重复选择元素的组合 (2)

- 题目

```
给定一个无重复元素的正整数数组candidates和一个正整数target，
找出candidates中所有可以使数字和为目标数target的唯一组合。
candidates中的数字可以无限制重复被选取。如果至少一个所选数字数量不同，则两种组合是唯一的。
对于给定的输入，保证和为target 的唯一组合数少于 150 个。
示例1: 输入: candidates = [2,3,6,7], target = 7 输出: [[7],[2,2,3]]
示例2: 输入: candidates = [2,3,5], target = 8 输出: [[2,2,2,2],[2,3,3],[3,5]]
示例 3: 输入: candidates = [2], target = 1 输出: []
示例 4: 输入: candidates = [1], target = 1 输出: [[1]]
示例 5: 输入: candidates = [1], target = 2 输出: [[1,1]]
提示: 1 <= candidates.length <= 30
1 <= candidates[i] <= 200
candidate 中的每个元素都是独一无二的。
1 <= target <= 500
注意: 本题与主站 39题相同:
```

- 解题思路

```
var res [][]int

func combinationSum(candidates []int, target int) [][]int {
        res = make([][]int, 0)
```

(续下页)

(接上页)

```
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        if target < 0{
                return
        }
        for i := index; i < len(candidates); i++ {
                arr = append(arr, candidates[i])
                dfs(candidates, target-candidates[i], arr, i)
                arr = arr[:len(arr)-1]
        }
}

# 2
var res [][]int

func combinationSum(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(candidates); i++ {
                if target < candidates[i] {
                        return
                }
```

(续下页)

(接上页)

```
                dfs(candidates, target-candidates[i], append(arr, candidates[i]), i)
        }
}
```

78.55 剑指 OfferII082. 含有重复元素集合的组合 (2)

- 题目

```
给定一个可能有重复数字的整数数组candidates和一个目标数target，找出candidates中所有可以使数字和为targe
candidates中的每个数字在每个组合中只能使用一次，解集不能包含重复的组合。
示例1:输入: candidates =[10,1,2,7,6,1,5], target =8, 输出:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
示例2:输入: candidates =[2,5,2,1,2], target =5, 输出:
[
[1,2,2],
[5]
]
提示:1 <=candidates.length <= 100
1 <=candidates[i] <= 50
1 <= target <= 30
注意：本题与主站 40题相同：
```

- 解题思路

```
var res [][]int

func combinationSum2(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
```

(续下页)

(接上页)

```
                res = append(res, temp)
                return
        }
        if target < 0 {
                return
        }
        for i := index; i < len(candidates); i++ {
                origin := i
                for i < len(candidates)-1 && candidates[i] == candidates[i+1] {
                        i++
                }
                arr = append(arr, candidates[i])
                dfs(candidates, target-candidates[i], arr, origin+1)
                arr = arr[:len(arr)-1]
        }
}

# 2
var res [][]int

func combinationSum2(candidates []int, target int) [][]int {
        res = make([][]int, 0)
        sort.Ints(candidates)
        dfs(candidates, target, []int{}, 0)
        return res
}

func dfs(candidates []int, target int, arr []int, index int) {
        if target == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(candidates); i++ {
                if i != index && candidates[i] == candidates[i-1] {
                        continue
                }
                if target < 0 {
                        return
                }
                arr = append(arr, candidates[i])
                dfs(candidates, target-candidates[i], arr, i+1)
```

(续下页)

(接上页)

```
                arr = arr[:len(arr)-1]
        }
}
```

78.56 剑指 OfferII083. 没有重复元素集合的全排列 (3)

- 题目

```
给定一个不含重复数字的整数数组 nums ，返回其 所有可能的全排列 。可以 按任意顺序␣
↪返回答案。
示例 1：输入：nums = [1,2,3] 输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
示例 2：输入：nums = [0,1] 输出：[[0,1],[1,0]]
示例 3：输入：nums = [1] 输出：[[1]]
提示：1 <= nums.length <= 6
-10 <= nums[i] <= 10
nums 中的所有整数 互不相同
注意：本题与主站 46题相同
```

- 解题思路

```
var res [][]int

func permute(nums []int) [][]int {
        res = make([][]int, 0)
        arr := make([]int, 0)
        visited := make(map[int]bool)
        dfs(nums, 0, arr, visited)
        return res
}

func dfs(nums []int, index int, arr []int, visited map[int]bool) {
        if index == len(nums) {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == false {
                        arr = append(arr, nums[i])
                        visited[i] = true
                        dfs(nums, index+1, arr, visited)
```

(续下页)

(接上页)

```
                        arr = arr[:len(arr)-1]
                        visited[i] = false
                }
        }
}

# 2
func permute(nums []int) [][]int {
        if len(nums) == 1 {
                return [][]int{nums}
        }
        res := make([][]int, 0)
        for i := 0; i < len(nums); i++ {
                tempArr := make([]int, len(nums)-1)
                copy(tempArr[0:], nums[:i])
                copy(tempArr[i:], nums[i+1:])
                arr := permute(tempArr)
                for _, v := range arr {
                        res = append(res, append(v, nums[i]))
                }
        }
        return res
}

# 3
var res [][]int

func permute(nums []int) [][]int {
        res = make([][]int, 0)
        arr := make([]int, len(nums))
        dfs(nums, 0, arr)
        return res
}

func dfs(nums []int, index int, arr []int) {
        if index == len(nums) {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := index; i < len(nums); i++ {
                arr[index] = nums[i]
```

(续下页)

(接上页)

```
                nums[i], nums[index] = nums[index], nums[i]
                dfs(nums, index+1, arr)
                nums[i], nums[index] = nums[index], nums[i]
        }
}
```

78.57 剑指 OfferII084. 含有重复元素集合的全排列 (3)

- 题目

```
给定一个可包含重复数字的整数集合nums ，按任意顺序 返回它所有不重复的全排列。
示例 1: 输入: nums = [1,1,2]
输出: [[1,1,2],
 [1,2,1],
 [2,1,1]]
示例 2: 输入: nums = [1,2,3] 输出: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
提示: 1 <= nums.length <= 8
-10 <= nums[i] <= 10
注意: 本题与主站 47题相同:
```

- 解题思路

```
var res [][]int

func permuteUnique(nums []int) [][]int {
        res = make([][]int, 0)
        sort.Ints(nums)
        dfs(nums, 0, make([]int, len(nums)), make([]int, 0))
        return res
}

func dfs(nums []int, index int, visited []int, arr []int) {
        if len(nums) == index {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == 1 {
                        continue
                }
```

(续下页)

(接上页)

```
                // visited[i-1] == 0 或者 visited[i-1] == 1都可以
                if i > 0 && nums[i] == nums[i-1] && visited[i-1] == 0 {
                        // if i > 0 && nums[i] == nums[i-1] && visited[i-1] == 1 {
                        continue
                }
                arr = append(arr, nums[i])
                visited[i] = 1
                dfs(nums, index+1, visited, arr)
                visited[i] = 0
                arr = arr[:len(arr)-1]
        }
}

# 2
var res [][]int

func permuteUnique(nums []int) [][]int {
        res = make([][]int, 0)
        sort.Ints(nums)
        dfs(nums, 0)
        return res
}

func dfs(nums []int, index int) {
        if index == len(nums) {
                temp := make([]int, len(nums))
                copy(temp, nums)
                res = append(res, temp)
                return
        }
        m := make(map[int]int)
        for i := index; i < len(nums); i++ {
                if _, ok := m[nums[i]]; ok {
                        continue
                }
                m[nums[i]] = 1
                nums[i], nums[index] = nums[index], nums[i]
                dfs(nums, index+1)
                nums[i], nums[index] = nums[index], nums[i]
        }
}

# 3
```

(续下页)

(接上页)

```
var res [][]int

func permuteUnique(nums []int) [][]int {
        res = make([][]int, 0)
        sort.Ints(nums)
        dfs(nums, make([]int, 0))
        return res
}

func dfs(nums []int, arr []int) {
        if len(nums) == 0 {
                temp := make([]int, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(nums); i++ {
                if i != 0 && nums[i] == nums[i-1] {
                        continue
                }
                tempArr := make([]int, len(nums))
                copy(tempArr, nums)
                arr = append(arr, nums[i])
                dfs(append(tempArr[:i], tempArr[i+1:]...), arr)
                arr = arr[:len(arr)-1]
        }
}
```

78.58 剑指 OfferII085. 生成匹配的括号 (3)

- 题目

```
正整数n代表生成括号的对数，请设计一个函数，用于能够生成所有可能的并且 有效的括号组合。
示例 1：输入：n = 3 输出：["((()))","(()())","(())()","()(())","()()()"]
示例 2：输入：n = 1 输出：["()"]
提示：1 <= n <= 8
注意：本题与主站 22题相同：
```

- 解题思路

```
var res []string

func generateParenthesis(n int) []string {
        res = make([]string, 0)
        dfs(0, 0, n, "")
        return res
}

func dfs(left, right, max int, str string) {
        if left == right && left == max {
                res = append(res, str)
                return
        }
        if left < max {
                dfs(left+1, right, max, str+"(")
        }
        if right < left {
                dfs(left, right+1, max, str+")")
        }
}

# 2
/*
dp[i]表示n=i时括号的组合
dp[i]="(" + dp[j] + ")"+dp[i-j-1] (j<i)
dp[0] = ""
*/
func generateParenthesis(n int) []string {
        dp := make([][]string, n+1)
        dp[0] = make([]string, 0)
        if n == 0 {
                return dp[0]
        }
        dp[0] = append(dp[0], "")
        for i := 1; i <= n; i++ {
                dp[i] = make([]string, 0)
                for j := 0; j < i; j++ {
                        for _, a := range dp[j] {
                                for _, b := range dp[i-j-1] {
                                        str := "(" + a + ")" + b
                                        dp[i] = append(dp[i], str)
                                }
                        }
                }
```

(续下页)

(接上页)

```
        }
        return dp[n]
}

# 3
type Node struct {
        str   string
        left  int
        right int
}

func generateParenthesis(n int) []string {
        res := make([]string, 0)
        if n == 0 {
                return res
        }
        queue := make([]*Node, 0)
        queue = append(queue, &Node{
                str:   "",
                left:  n,
                right: n,
        })
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.left == 0 && node.right == 0 {
                        res = append(res, node.str)
                }
                if node.left > 0 {
                        queue = append(queue, &Node{
                                str:   node.str + "(",
                                left:  node.left - 1,
                                right: node.right,
                        })
                }
                if node.right > 0 && node.left < node.right {
                        queue = append(queue, &Node{
                                str:   node.str + ")",
                                left:  node.left,
                                right: node.right - 1,
                        })
                }
        }
```

(续下页)

(接上页)

```
        return res
}
```

78.59 剑指 OfferII086. 分割回文子字符串 (2)

- 题目

```
给定一个字符串 s ，请将 s 分割成一些子串，使每个子串都是 回文串 ，返回 s␣
↪所有可能的分割方案。
回文串是正着读和反着读都一样的字符串。
示例 1：输入：s = "google" 输出：[["g","o","o","g","l","e"],["g","oo","g","l","e"],[
↪"goog","l","e"]]
示例 2：输入：s = "aab" 输出：[["a","a","b"],["aa","b"]]
示例 3：输入：s = "a" 输出：[["a"]
提示：1 <= s.length <= 16
s 仅由小写英文字母组成
注意：本题与主站 131题相同：
```

- 解题思路

```
var res [][]string

func partition(s string) [][]string {
        res = make([][]string, 0)
        arr := make([]string, 0)
        dfs(s, 0, arr)
        return res
}

func dfs(s string, level int, arr []string) {
        if level == len(s) {
                temp := make([]string, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := level; i < len(s); i++ {
                str := s[level : i+1]
                if judge(str) == true {
                        dfs(s, i+1, append(arr, str))
                }
        }
```

(续下页)

(接上页)

```
}

func judge(s string) bool {
        for i := 0; i < len(s)/2; i++ {
                if s[i] != s[len(s)-1-i] {
                        return false
                }
        }
        return true
}

# 2
var res [][]string
var dp [][]bool

func partition(s string) [][]string {
        res = make([][]string, 0)
        arr := make([]string, 0)
        dp = make([][]bool, len(s))
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
                dp[r][r] = true
                for l := 0; l < r; l++ {
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
                        }
                }
        }
        dfs(s, 0, arr)
        return res
}

func dfs(s string, level int, arr []string) {
        if level == len(s) {
                temp := make([]string, len(arr))
                copy(temp, arr)
                res = append(res, temp)
                return
        }
        for i := level; i < len(s); i++ {
                str := s[level : i+1]
```

(续下页)

(接上页)

```
                if dp[level][i] == true {
                        dfs(s, i+1, append(arr, str))
                }
        }
}
```

78.60 剑指 OfferII087. 复原 IP(2)

- 题目

```
给定一个只包含数字的字符串 s ，用以表示一个 IP 地址，返回所有可能从s 获得的 有效 IP␣
↪地址 。你可以按任何顺序返回答案。
有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导␣
↪0），整数之间用 '.' 分隔。
例如："0.1.2.201" 和 "192.168.1.1" 是 有效 IP 地址，
但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 无效 IP 地址。
示例 1：输入：s = "25525511135" 输出：["255.255.11.135","255.255.111.35"]
示例 2：输入：s = "0000" 输出：["0.0.0.0"]
示例 3：输入：s = "1111" 输出：["1.1.1.1"]
示例 4：输入：s = "010010" 输出：["0.10.0.10","0.100.1.0"]
示例 5：输入：s = "10203040" 输出：["10.20.30.40","102.0.30.40","10.203.0.40"]
提示：0 <= s.length <= 3000
s 仅由数字组成
注意：本题与主站 93题相同
```

- 解题思路

```
var res []string

func restoreIpAddresses(s string) []string {
        res = make([]string, 0)
        if len(s) < 4 || len(s) > 12 {
                return nil
        }
        dfs(s, make([]string, 0), 0)
        return res
}

func dfs(s string, arr []string, level int) {
        if level == 4 {
                if len(s) == 0 {
                        str := strings.Join(arr, ".")
```

(续下页)

(接上页)

```
                        res = append(res, str)
                }
                return
        }
        for i := 1; i <= 3; i++ {
                if i <= len(s) {
                        value, _ := strconv.Atoi(s[:i])
                        if value <= 255 {
                                str := s[i:]
                                dfs(str, append(arr, s[:i]), level+1)
                        }
                        if value == 0 {
                                // 避免出现001,01这种情况
                                break
                        }
                }
        }
}

# 2
func restoreIpAddresses(s string) []string {
        res := make([]string, 0)
        if len(s) < 4 || len(s) > 12 {
                return nil
        }
        for i := 1; i <= 3 && i < len(s)-2; i++ {
                for j := i + 1; j <= i+3 && j < len(s)-1; j++ {
                        for k := j + 1; k <= j+3 && k < len(s); k++ {
                                if judge(s[:i]) && judge(s[i:j]) &&
                                        judge(s[j:k]) && judge(s[k:]) {
                                        res = append(res, s[:i]+"."+s[i:j]+"."+s[j:k]+
↪"."+s[k:])
                                }
                        }
                }
        }
        return res
}

func judge(s string) bool {
        if len(s) > 1 && s[0] == '0' {
                return false
        }
```

(续下页)

(接上页)

```
        value, _ := strconv.Atoi(s)
        if value > 255 {
                return false
        }
        return true
}
```

78.61 剑指 OfferII089. 房屋偷盗 (4)

- 题目

```
一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，
影响小偷偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯
给定一个代表每个房屋存放金额的非负整数数组 nums，请计算不触动警报装置的情况下␣
↪，一夜之内能够偷窃到的最高金额。
示例 1：输入：nums = [1,2,3,1] 输出：4
解释：偷窃 1 号房屋 （金额 = 1） ，然后偷窃 3 号房屋 （金额 = 3）。
    偷窃到的最高金额 = 1 + 3 = 4 。
示例 2：输入：nums = [2,7,9,3,1] 输出：12
解释：偷窃 1 号房屋 （金额 = 2），偷窃 3 号房屋 （金额 = 9），接着偷窃 5 号房屋 （金额 =␣
↪1）。
    偷窃到的最高金额 = 2 + 9 + 1 = 12 。
提示：1 <= nums.length <= 100
0 <= nums[i] <= 400
注意：本题与主站 198题相同：
```

- 解题思路

```
func rob(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
                return nums[0]
        }
        a := nums[0]
        b := max(a, nums[1])

        for i := 2; i < len(nums); i++ {
                a, b = b, max(a+nums[i], b)
        }
        return b
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        }
        if n == 1 {
                return nums[0]
        }
        dp := make([]int, n)
        dp[0] = nums[0]
        if nums[0] > nums[1] {
                dp[1] = nums[0]
        } else {
                dp[1] = nums[1]
        }
        for i := 2; i < n; i++ {
                dp[i] = max(dp[i-1], dp[i-2]+nums[i])
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func rob(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        if len(nums) == 1 {
                return nums[0]
        }
        n := len(nums)
        dp := make([][]int, n)
        for n := range dp {
                dp[n] = make([]int, 2)
        }
        dp[0][0], dp[0][1] = 0, nums[0]
        for i := 1; i < n; i++ {
                dp[i][0] = max(dp[i-1][0], dp[i-1][1])
                dp[i][1] = dp[i-1][0] + nums[i]
        }
        return max(dp[n-1][0], dp[n-1][1])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func rob(nums []int) int {
        var a, b int
        for i, v := range nums {
                if i%2 == 0 {
                        a = max(a+v, b)
                } else {
                        b = max(a, b+v)
                }
        }
        return max(a, b)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

78.62 剑指 OfferII090. 环形房屋偷盗 (3)

- 题目

```
一个专业的小偷，计划偷窃一个环形街道上沿街的房屋，每间房内都藏有一定的现金。
这个地方所有的房屋都 围成一圈 ，这意味着第一个房屋和最后一个房屋是紧挨着的。
同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警␣
↪。
给定一个代表每个房屋存放金额的非负整数数组 nums ，请计算在不触动警报装置的情况下␣
↪，今晚能够偷窃到的最高金额。
示例1: 输入: nums = [2,3,2] 输出: 3
解释: 你不能先偷窃 1 号房屋（金额 = 2），然后偷窃 3 号房屋（金额 = 2），␣
↪因为他们是相邻的。
示例 2: 输入: nums = [1,2,3,1] 输出: 4
解释: 你可以先偷窃 1 号房屋（金额 = 1），然后偷窃 3 号房屋（金额 = 3）。
    偷窃到的最高金额 = 1 + 3 = 4 。
示例 3: 输入: nums = [0] 输出: 0
提示: 1 <= nums.length <= 100
0 <= nums[i] <= 1000
注意: 本题与主站 213题相同:
```

- 解题思路

```
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        } else if n == 1 {
                return nums[0]
        }
        dp1 := make([]int, n) // 从第一家开始打劫，最后一家不可选
        dp2 := make([]int, n) // 从第二家开始打劫，最后一家可以选
        dp1[0] = nums[0]
        dp1[1] = max(nums[0], nums[1])
        dp2[0] = 0
        dp2[1] = nums[1]
        for i := 2; i < n; i++ {
                dp1[i] = max(dp1[i-1], dp1[i-2]+nums[i])
                dp2[i] = max(dp2[i-1], dp2[i-2]+nums[i])
        }
        return max(dp1[n-2], dp2[n-1])
}

func max(a, b int) int {
```

(续下页)

(接上页)

```
        if a > b {
                return a
        }
        return b
}

# 2
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        } else if n == 1 {
                return nums[0]
        } else if n == 2 {
                return max(nums[0], nums[1])
        }
        return max(getMax(nums[:n-1]), getMax(nums[1:]))
}

func getMax(nums []int) int {
        n := len(nums)
        dp := make([]int, n+1)
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])
        for i := 2; i < n; i++ {
                dp[i] = max(dp[i-1], dp[i-2]+nums[i])
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func rob(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        } else if n == 1 {
```

(续下页)

(接上页)

```
                return nums[0]
        } else if n == 2 {
                return max(nums[0], nums[1])
        }
        return max(getMax(nums[:n-1]), getMax(nums[1:]))
}

func getMax(nums []int) int {
        var a, b int
        for i, v := range nums {
                if i%2 == 0 {
                        a = max(a+v, b)
                } else {
                        b = max(a, b+v)
                }
        }
        return max(a, b)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

78.63 剑指 OfferII091. 粉刷房子 (2)

- 题目

```
假如有一排房子，共 n 个，每个房子可以被粉刷成红色、蓝色或者绿色这三种颜色中的一种，
你需要粉刷所有的房子并且使其相邻的两个房子颜色不能相同。
当然，因为市场上不同颜色油漆的价格不同，所以房子粉刷成不同颜色的花费成本也是不同的。
每个房子粉刷成不同颜色的花费是以一个n x 3的正整数矩阵 costs 来表示的。
例如，costs[0][0] 表示第 0 号房子粉刷成红色的成本花费；costs[1][2]表示第 1␣
↪号房子粉刷成绿色的花费，以此类推。
请计算出粉刷完所有房子最少的花费成本。
示例 1：输入：costs = [[17,2,17],[16,16,5],[14,3,19]] 输出：10
解释：将 0 号房子粉刷成蓝色，1 号房子粉刷成绿色，2 号房子粉刷成蓝色。
    最少花费：2 + 5 + 3 = 10。
示例 2：输入：costs = [[7,6,2]] 输出：2
提示:costs.length == n
```

(续下页)

(接上页)

```
costs[i].length == 3
1 <= n <= 100
1 <= costs[i][j] <= 20
注意：本题与主站 256题相同：
```

- 解题思路

```
func minCost(costs [][]int) int {
        n := len(costs)
        dp := make([][3]int, n) // dp[i][j] 表示涂前i间房子的最小成本
        for j := 0; j < 3; j++ {
                dp[0][j] = costs[0][j]
        }
        for i := 1; i < n; i++ {
                dp[i][0] = min(dp[i-1][1], dp[i-1][2]) + costs[i][0]
                dp[i][1] = min(dp[i-1][0], dp[i-1][2]) + costs[i][1]
                dp[i][2] = min(dp[i-1][0], dp[i-1][1]) + costs[i][2]
        }
        return min(dp[n-1][0], min(dp[n-1][1], dp[n-1][2]))
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minCost(costs [][]int) int {
        n := len(costs)
        a, b, c := costs[0][0], costs[0][1], costs[0][2]
        for i := 1; i < n; i++ {
                a, b, c = min(b, c)+costs[i][0], min(a, c)+costs[i][1], min(a,␣
↪b)+costs[i][2]
        }
        return min(a, min(b, c))
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
```

(续下页)

(接上页)

```
}
```

78.64 剑指 OfferII092. 翻转字符 (3)

- 题目

```
如果一个由'0' 和 '1'组成的字符串，是以一些 '0'（可能没有 '0'）后面跟着一些 '1
↪'（也可能没有 '1'）的形式组成的，
那么该字符串是单调递增的。
我们给出一个由字符 '0' 和 '1'组成的字符串 s，我们可以将任何'0' 翻转为'1'或者将'1
↪'翻转为'0'。
返回使 s单调递增的最小翻转次数。
示例 1：输入：s = "00110" 输出：1
解释：我们翻转最后一位得到 00111.
示例 2：输入：s = "010110" 输出：2
解释：我们翻转得到 011111，或者是 000111。
示例 3：输入：s = "00011000" 输出：2
解释：我们翻转得到 00000000。
提示：1 <= s.length <= 20000
s 中只包含字符'0'和'1'
注意：本题与主站 926题相同：
```

- 解题思路

```
func minFlipsMonoIncr(S string) int {
    n := len(S)
    dpA := make([]int, n) // 0 结尾
    dpB := make([]int, n) // 1 结尾
    if S[0] == '1' {
            dpA[0] = 1
    } else {
            dpB[0] = 1
    }
    for i := 1; i < n; i++ {
            if S[i] == '1' {
                    dpA[i] = dpA[i-1] + 1             // 需要改为0
                    dpB[i] = min(dpB[i-1], dpA[i-1]) // 1结尾和0结尾的最小值
            } else {
                    dpA[i] = dpA[i-1]                    // 不需要改为0
                    dpB[i] = min(dpB[i-1], dpA[i-1]) + 1 // 1结尾和0结尾的最小值+1
            }
    }
```

(续下页)

(接上页)

```
    return min(dpA[n-1], dpB[n-1])
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
}

# 2
func minFlipsMonoIncr(S string) int {
    n := len(S)
    a := 0 // 0 结尾
    b := 0 // 1 结尾
    if S[0] == '1' {
            a = 1
    } else {
            b = 1
    }
    for i := 1; i < n; i++ {
            if S[i] == '1' {
                    a, b = a+1, min(a, b)
            } else {
                    a, b = a, min(a, b)+1
            }
    }
    return min(a, b)
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
}

# 3
func minFlipsMonoIncr(S string) int {
    n := len(S)
    arr := make([]int, n+1)
    for i := 1; i <= n; i++ {
            arr[i] = arr[i-1]
```

(续下页)

(接上页)

```
            if S[i-1] == '1' {
                    arr[i]++
            }
    }
    res := n
    for i := 0; i <= n; i++ {
            left := arr[i]
            right := n - i - (arr[n] - arr[i])
            res = min(res, left+right)
    }
    return res
}

func min(a, b int) int {
    if a > b {
            return b
    }
    return a
}
```

78.65 剑指 OfferII093. 最长斐波那契数列 (2)

- 题目

```
如果序列X_1, X_2, ..., X_n满足下列条件，就说它是斐波那契式的：
n >= 3
对于所有i + 2 <= n，都有X_i + X_{i+1} = X_{i+2}
给定一个严格递增的正整数数组形成序列 arr，找到 arr␣
↪中最长的斐波那契式的子序列的长度。如果一个不存在，返回0 。
（回想一下，子序列是从原序列 arr 中派生出来的，它从 arr␣
↪中删掉任意数量的元素（也可以不删），而不改变其余元素的顺序。
例如，[3, 5, 8]是[3, 4, 5, 6, 7, 8]的一个子序列）
示例 1：输入：arr = [1,2,3,4,5,6,7,8] 输出：5
解释：最长的斐波那契式子序列为 [1,2,3,5,8] 。
示例2：输入：arr = [1,3,7,11,12,14,18] 输出：3
解释：最长的斐波那契式子序列有 [1,11,12]、[3,11,14] 以及 [7,11,18] 。
提示：3 <= arr.length <= 1000
1 <= arr[i] < arr[i + 1] <= 10^9
注意：本题与主站 873题相同：
```

- 解题思路

```
func lenLongestFibSubseq(arr []int) int {
        n := len(arr)
        m := make(map[int]bool)
        for i := 0; i < n; i++ {
                m[arr[i]] = true
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        count := 2
                        a, b := arr[i], arr[j]
                        for m[a+b] == true {
                                count++
                                a, b = b, a+b
                        }
                        if count > res && count > 2 {
                                res = count
                        }
                }
        }
        return res
}

# 2
func lenLongestFibSubseq(arr []int) int {
        n := len(arr)
        m := make(map[int]int)
        for i := 0; i < n; i++ {
                m[arr[i]] = i
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        dp[i][j] = 2
                }
        }
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
                        index, ok := m[arr[i]-arr[j]]
                        if ok && arr[index] < arr[j] {
```

(续下页)

(接上页)

```
                                dp[j][i] = dp[index][j] + 1
                                if dp[j][i] > 2 && dp[j][i] > res {
                                        res = dp[j][i]
                                }
                        }
                }
        }
        return res
}
```

78.66 剑指 OfferII095. 最长公共子序列 (3)

- 题目

```
给定两个字符串text1 和text2，返回这两个字符串的最长 公共子序列 的长度。如果不存在␣
↪公共子序列 ，返回 0 。
一个字符串的子序列是指这样一个新的字符串：
它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。
例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。
两个字符串的 公共子序列 是这两个字符串所共同拥有的子序列。
示例 1：输入：text1 = "abcde", text2 = "ace"  输出：3
解释：最长公共子序列是 "ace" ，它的长度为 3 。
示例 2：输入：text1 = "abc", text2 = "abc" 输出：3
解释：最长公共子序列是 "abc" ，它的长度为 3 。
示例 3：输入：text1 = "abc", text2 = "def" 输出：0
解释：两个字符串没有公共子序列，返回 0 。
提示：1 <= text1.length, text2.length <= 1000
text1 和text2 仅由小写英文字符组成。
注意：本题与主站 1143题相同：
```

- 解题思路

```
func longestCommonSubsequence(text1 string, text2 string) int {
        n, m := len(text1), len(text2)
        dp := make([][]int, n+1)
        for i := 0; i < n+1; i++ {
                dp[i] = make([]int, m+1)
        }
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if text1[i-1] == text2[j-1] {
                                dp[i][j] = dp[i-1][j-1] + 1
```

(续下页)

(接上页)

```
                        } else {
                                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
                        }
                }
        }
        return dp[n][m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestCommonSubsequence(text1 string, text2 string) int {
        n, m := len(text1), len(text2)
        prev := make([]int, m+1)
        cur := make([]int, m+1)
        for i := 1; i <= n; i++ {
                for j := 1; j <= m; j++ {
                        if text1[i-1] == text2[j-1] {
                                cur[j] = prev[j-1] + 1
                        } else {
                                cur[j] = max(prev[j], cur[j-1])
                        }
                }
                copy(prev, cur)
        }
        return cur[m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func longestCommonSubsequence(text1 string, text2 string) int {
        n, m := len(text1), len(text2)
```

(续下页)

(接上页)

```
        cur := make([]int, m+1)
        for i := 1; i <= n; i++ {
                pre := cur[0]
                for j := 1; j <= m; j++ {
                        temp := cur[j]
                        if text1[i-1] == text2[j-1] {
                                cur[j] = pre + 1
                        } else {
                                cur[j] = max(cur[j], cur[j-1])
                        }
                        pre = temp
                }
        }
        return cur[m]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

78.67 剑指 OfferII096. 字符串交织 (2)

- 题目

```
给定三个字符串s1、s2、s3，请判断s3能不能由s1和s2交织（交错）组成。
两个字符串 s 和 t 交织的定义与过程如下，其中每个字符串都会被分割成若干 非空 子字符串：
s = s1 + s2 + ... + sn
t = t1 + t2 + ... + tm
|n - m| <= 1
交织 是 s1 + t1 + s2 + t2 + s3 + t3 + ... 或者 t1 + s1 + t2 + s2 + t3 + s3 + ...
提示：a + b 意味着字符串 a 和 b 连接。
示例 1：输入：s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac" 输出：true
示例 2：输入：s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"  输出：false
示例 3：输入：s1 = "", s2 = "", s3 = "" 输出：true
提示：0 <= s1.length, s2.length <= 100
0 <= s3.length <= 200
s1、s2、和 s3 都由小写英文字母组成
注意：本题与主站 97题相同：
```

- 解题思路

```
func isInterleave(s1 string, s2 string, s3 string) bool {
        n, m, t := len(s1), len(s2), len(s3)
        if n+m != t {
                return false
        }
        // dp[i][j]表示s1的前i个元素和s2的前j个元素是否能交错组成s3的前i+j个元素
        dp := make([][]bool, n+1)
        for i := 0; i <= n; i++ {
                dp[i] = make([]bool, m+1)
        }
        dp[0][0] = true
        for i := 0; i <= n; i++ {
                for j := 0; j <= m; j++ {
                        total := i + j - 1
                        if i > 0 && dp[i-1][j] == true && s1[i-1] == s3[total] {
                                dp[i][j] = true
                        }
                        if j > 0 && dp[i][j-1] == true && s2[j-1] == s3[total] {
                                dp[i][j] = true
                        }
                }
        }
        return dp[n][m]
}

# 2
func isInterleave(s1 string, s2 string, s3 string) bool {
        n, m, t := len(s1), len(s2), len(s3)
        if n+m != t {
                return false
        }
        // dp[j]表示s1的前i个元素和s2的前j个元素是否能交错组成s3的前i+j个元素
        dp := make([]bool, m+1)
        dp[0] = true
        for i := 0; i <= n; i++ {
                for j := 0; j <= m; j++ {
                        total := i + j - 1
                        if i > 0 {
                                if dp[j] == true && s1[i-1] == s3[total] {
                                        dp[j] = true
                                } else {
                                        dp[j] = false
                                }
                        }
```

(续下页)

(接上页)

```
                        if j > 0 {
                                if dp[j] == true || (dp[j-1] == true && s2[j-1] ==␣
↪s3[total]) {
                                        dp[j] = true
                                } else {
                                        dp[j] = false
                                }
                        }
                }
        }
        return dp[m]
}
```

78.68 剑指 OfferII098. 路径的数目 (4)

- 题目

```
一个机器人位于一个 m x n网格的左上角 （起始点在下图中标记为 “Start” ）。
机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为␣
↪“Finish” ）。
问总共有多少条不同的路径？
示例 1: 输入: m = 3, n = 7 输出: 28
示例 2: 输入: m = 3, n = 2 输出: 3
解释: 从左上角开始，总共有 3 条路径可以到达右下角。
1. 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右
3. 向下 -> 向右 -> 向下
示例 3: 输入: m = 7, n = 3 输出: 28
示例 4: 输入: m = 3, n = 3 输出: 6
提示: 1 <= m, n <= 100
题目数据保证答案小于等于 2 * 109
注意: 本题与主站 62题相同:
```

- 解题思路

```
// dp[i][j] = dp[i-1][j] + dp[i][j-1]
func uniquePaths(m int, n int) int {
        if m <= 0 || n <= 0 {
                return 0
        }
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                dp[i] = make([]int, m)
                dp[i][0] = 1
        }
        for i := 0; i < m; i++ {
                dp[0][i] = 1
        }
        for i := 1; i < n; i++ {
                for j := 1; j < m; j++ {
                        dp[i][j] = dp[i-1][j] + dp[i][j-1]
                }
        }
        return dp[n-1][m-1]
}

#
// dp[i]= dp[i-1] + dp[i]
func uniquePaths(m int, n int) int {
        if m <= 0 || n <= 0 {
                return 0
        }
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = 1
        }
        for i := 1; i < m; i++ {
                for j := 1; j < n; j++ {
                        dp[j] = dp[j] + dp[j-1]
                }
        }
        return dp[n-1]
}

# 3
func uniquePaths(m int, n int) int {
        if m == 1 || n == 1 {
                return 1
        }
        if m > n {
                m, n = n, m
        }
        a := 1
        for i := 1; i <= m-1; i++ {
                a = a * i
```

(续下页)

(接上页)

```
        }
        b := 1
        for i := n; i <= m+n-2; i++ {
                b = b * i
        }
        return b / a
}

# 4
var arr [][]int

func uniquePaths(m int, n int) int {
        arr = make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        return dfs(m, n)
}

func dfs(m, n int) int {
        if m <= 0 || n <= 0 {
                return 0
        }
        if m == 1 || n == 1 {
                return 1
        }
        if arr[n][m] > 0 {
                return arr[n][m]
        }
        arr[n][m] = dfs(m, n-1) + dfs(m-1, n)
        return arr[n][m]
}
```

78.69 剑指 OfferII099. 最小路径之和 (4)

- 题目

```
给定一个包含非负整数的 mxn网格grid␣
↪，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。
说明：一个机器人每次只能向下或者向右移动一步。
示例 1：输入：grid = [[1,3,1],[1,5,1],[4,2,1]] 输出：7
解释：因为路径 1→3→1→1→1 的总和最小。
```

(续下页)

(接上页)

```
示例 2: 输入: grid = [[1,2,3],[4,5,6]] 输出: 12
提示: m == grid.length
n == grid[i].length
1 <= m, n <= 200
0 <= grid[i][j] <= 100
注意: 本题与主站 64题相同:
```

- 解题思路

```
func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
        }
        dp[0][0] = grid[0][0]
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i == 0 && j != 0 {
                                dp[i][j] = dp[i][j-1] + grid[i][j]
                        } else if i != 0 && j == 0 {
                                dp[i][j] = dp[i-1][j] + grid[i][j]
                        } else if i != 0 && j != 0 {
                                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
                        }
                }
        }
        return dp[n-1][m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

#
func minPathSum(grid [][]int) int {
        n := len(grid)
```

(续下页)

(接上页)

```
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if i == 0 && j != 0 {
                                grid[i][j] = grid[i][j-1] + grid[i][j]
                        } else if i != 0 && j == 0 {
                                grid[i][j] = grid[i-1][j] + grid[i][j]
                        } else if i != 0 && j != 0 {
                                grid[i][j] = min(grid[i-1][j], grid[i][j-1]) +␣
↪grid[i][j]
                        }
                }
        }
        return grid[n-1][m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        dp := make([]int, m)
        dp[0] = grid[0][0]

        for i := 1; i < m; i++ {
                dp[i] = dp[i-1] + grid[0][i]
        }
        for i := 1; i < n; i++ {
                dp[0] = dp[0] + grid[i][0]
                for j := 1; j < m; j++ {
                        dp[j] = min(dp[j-1], dp[j]) + grid[i][j]
```

(续下页)

(接上页)

```
                }
        }
        return dp[m-1]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
var arr [][]int

func minPathSum(grid [][]int) int {
        n := len(grid)
        if n == 0 {
                return 0
        }
        m := len(grid[0])
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        return dfs(grid, n-1, m-1)
}

func dfs(grid [][]int, n, m int) int {
        if m == 0 && n == 0 {
                arr[0][0] = grid[0][0]
                return grid[0][0]
        }
        if n == 0 {
                return grid[0][m] + dfs(grid, 0, m-1)
        }
        if m == 0 {
                return grid[n][0] + dfs(grid, n-1, 0)
        }
        if arr[n][m] > 0 {
                return arr[n][m]
        }
        arr[n][m] = min(dfs(grid, n-1, m), dfs(grid, n, m-1)) + grid[n][m]
```

(续下页)

(接上页)

```
        return arr[n][m]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

78.70 剑指 OfferII100. 三角形中最小路径之和 (5)

- 题目

```
给定一个三角形 triangle ，找出自顶向下的最小路径和。
每一步只能移动到下一行中相邻的结点上。
相邻的结点 在这里指的是 下标 与 上一层结点下标 相同或者等于 上一层结点下标 + 1␣
↪的两个结点。
也就是说，如果正位于当前行的下标 i ，那么下一步可以移动到下一行的下标 i 或 i + 1 。
示例 1：输入：triangle = [[2],[3,4],[6,5,7],[4,1,8,3]] 输出：11
解释：如下面简图所示：
   2
  3 4
 6 5 7
4 1 8 3
自顶向下的最小路径和为11（即，2+3+5+1= 11）。
示例 2：输入：triangle = [[-10]] 输出：-10
提示：1 <= triangle.length <= 200
triangle[0].length == 1
triangle[i].length == triangle[i - 1].length + 1
-104 <= triangle[i][j] <= 104
进阶：你可以只使用 O(n)的额外空间（n 为三角形的总行数）来解决这个问题吗？
注意：本题与主站 120题相同：
```

- 解题思路

```
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, n)
        }
```

(续下页)

(接上页)

```
        dp[0][0] = triangle[0][0]
        for i := 1; i < n; i++ {
                dp[i][0] = dp[i-1][0] + triangle[i][0]
                for j := 1; j < i; j++ {
                        dp[i][j] = min(dp[i-1][j-1], dp[i-1][j]) + triangle[i][j]
                }
                dp[i][i] = dp[i-1][i-1] + triangle[i][i]
        }
        res := dp[n-1][0]
        for i := 1; i < n; i++ {
                res = min(res, dp[n-1][i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        dp := [2][]int{}
        for i := 0; i < 2; i++ {
                dp[i] = make([]int, n)
        }
        dp[0][0] = triangle[0][0]
        for i := 1; i < n; i++ {
                cur := i % 2
                prev := 1 - cur
                dp[cur][0] = dp[prev][0] + triangle[i][0]
                for j := 1; j < i; j++ {
                        dp[cur][j] = min(dp[prev][j-1], dp[prev][j]) + triangle[i][j]
                }
                dp[cur][i] = dp[prev][i-1] + triangle[i][i]
        }
        res := dp[(n-1)%2][0]
        for i := 1; i < n; i++ {
                res = min(res, dp[(n-1)%2][i])
        }
```

(续下页)

(接上页)

```
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        dp := make([]int, n)
        dp[0] = triangle[0][0]
        for i := 1; i < n; i++ {
                dp[i] = dp[i-1] + triangle[i][i]
                for j := i - 1; j > 0; j-- {
                        dp[j] = min(dp[j-1], dp[j]) + triangle[i][j]
                }
                dp[0] = dp[0] + triangle[i][0]
        }
        res := dp[0]
        for i := 1; i < n; i++ {
                res = min(res, dp[i])
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func minimumTotal(triangle [][]int) int {
        n := len(triangle)
        for i := n - 2; i >= 0; i-- {
                for j := 0; j < len(triangle[i]); j++ {
                        triangle[i][j] = min(triangle[i+1][j], triangle[i+1][j+1]) +␣
↪triangle[i][j]
```

(续下页)

(接上页)

```
                }
        }
        return triangle[0][0]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 5
var dp [][]int

func minimumTotal(triangle [][]int) int {
        dp = make([][]int, len(triangle))
        for i := 0; i < len(triangle); i++ {
                dp[i] = make([]int, len(triangle))
        }
        return dfs(triangle, 0, 0)
}

func dfs(triangle [][]int, i, j int) int {
        if i == len(triangle) {
                return 0
        }
        if dp[i][j] != 0 {
                return dp[i][j]
        }
        dp[i][j] = min(dfs(triangle, i+1, j), dfs(triangle, i+1, j+1)) +␣
↪triangle[i][j]
        return dp[i][j]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

78.71 剑指 OfferII102. 加减的目标值 (5)

- 题目

```
给定一个正整数数组 nums 和一个整数 target 。
向数组中的每个整数前添加'+' 或 '-' ，然后串联起所有整数，可以构造一个 表达式 ：
例如，nums = [2, 1] ，可以在 2 之前添加 '+' ，在 1 之前添加 '-'␣
↪，然后串联起来得到表达式 "+2-1" 。
返回可以通过上述方法构造的、运算结果等于 target 的不同 表达式 的数目。
示例 1：输入：nums = [1,1,1,1,1], target = 3 输出：5
解释：一共有 5 种方法让最终目标和为 3 。
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
示例 2：输入：nums = [1], target = 1 输出：1
提示：1 <= nums.length <= 20
0 <= nums[i] <= 1000
0 <= sum(nums[i]) <= 1000
-1000 <= target <= 1000
注意：本题与主站 494题相同：
```

- 解题思路

```
func findTargetSumWays(nums []int, S int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
                if nums[0] == 0 && S == 0 {
                        return 2
                }
                if nums[0] == S || nums[0] == -S {
                        return 1
                }
        }
        value := nums[0]
        nums = nums[1:]
        return findTargetSumWays(nums, S-value) + findTargetSumWays(nums, S+value)
}

# 2
func findTargetSumWays(nums []int, S int) int {
```

(续下页)

(接上页)

```
        dp := make(map[int]int)
        dp[nums[0]]++
        dp[-nums[0]]++
        for i := 1; i < len(nums); i++ {
                temp := make(map[int]int)
                for k, v := range dp {
                        temp[k-nums[i]] = temp[k-nums[i]] + v
                        temp[k+nums[i]] = temp[k+nums[i]] + v
                }
                dp = temp
        }
        return dp[S]
}

# 3
var res int

func findTargetSumWays(nums []int, S int) int {
        res = 0
        dfs(nums, 0, S)
        return res
}

func dfs(nums []int, index int, target int) {
        if index == len(nums) {
                if target == 0 {
                        res++
                }
                return
        }
        dfs(nums, index+1, target+nums[index])
        dfs(nums, index+1, target-nums[index])
}

# 4
func findTargetSumWays(nums []int, S int) int {
        sum := 0
        // 非负整数数组
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum < int(math.Abs(float64(S))) || (sum+S)%2 == 1 {
                return 0
```

(续下页)

(接上页)

```
        }
        // 一个正数x,一个负数背包y => x+y=sum, x-y=S => (sum+S)/2=x
        target := (sum + S) / 2
        dp := make([]int, target+1)
        dp[0] = 1
        for i := 1; i <= len(nums); i++ {
        // 从后往前，避免覆盖
                for j := target; j >= 0; j-- {
                        if j >= nums[i-1] {
                                // 背包足够大，都选
                                dp[j] = dp[j] + dp[j-nums[i-1]]
                        } else {
                                // 容量不够，不选
                                dp[j] = dp[j]
                        }
                }
        }
        return dp[target]
}

# 5
func findTargetSumWays(nums []int, S int) int {
        sum := 0
        // 非负整数数组
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum < int(math.Abs(float64(S))) || (sum+S)%2 == 1 {
                return 0
        }
        // 一个正数x,一个负数背包y => x+y=sum, x-y=S => (sum+S)/2=x
        target := (sum + S) / 2
        // 在前i个物品中选择，若当前背包的容量为j，则最多有x种方法可以恰好装满背包。
        dp := make([][]int, len(nums)+1)
        for i := 0; i <= len(nums); i++ {
                dp[i] = make([]int, target+1)
                dp[i][0] = 1 // 容量为0，  只有都不选
        }
        for i := 1; i <= len(nums); i++ {
                for j := 0; j <= target; j++ {
                        if j >= nums[i-1] {
                                // 背包足够大，都选
                                dp[i][j] = dp[i-1][j] + dp[i-1][j-nums[i-1]]
```

(续下页)

(接上页)

```
                        } else {
                                // 容量不够，不选
                                dp[i][j] = dp[i-1][j]
                        }
                }
        }
        return dp[len(nums)][target]
}
```

78.72 剑指 OfferII103. 最少的硬币数目 (3)

- 题目

```
给定不同面额的硬币 coins 和一个总金额␣
↪amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。
如果没有任何一种硬币组合能组成总金额，返回-1。
你可以认为每种硬币的数量是无限的。
示例1：输入：coins = [1, 2, 5], amount = 11 输出：3
解释：11 = 5 + 5 + 1
示例 2：输入：coins = [2], amount = 3 输出：-1
示例 3：输入：coins = [1], amount = 0 输出：0
示例 4：输入：coins = [1], amount = 1 输出：1
示例 5：输入：coins = [1], amount = 2 输出：2
提示：1 <= coins.length <= 12
1 <= coins[i] <= 231 - 1
0 <= amount <= 104
注意：本题与主站 322题相同：
```

- 解题思路

```
func coinChange(coins []int, amount int) int {
        dp := make([]int, amount+1)
        for i := 1; i <= amount; i++ {
                dp[i] = -1
                for j := 0; j < len(coins); j++ {
                        prev := i - coins[j]
                        if i < coins[j] || dp[prev] == -1 {
                                continue
                        }
                        if dp[i] == -1 || dp[i] > dp[prev]+1 {
                                dp[i] = dp[prev] + 1
                        }
```

(续下页)

(接上页)

```
                }
        }
        return dp[amount]
}

# 2
func coinChange(coins []int, amount int) int {
        dp := make([]int, amount+1)
        for i := 0; i <= amount; i++ {
                dp[i] = amount + 1
        }
        dp[0] = 0
        for i := 0; i < len(coins); i++ {
                for j := coins[i]; j < amount+1; j++ {
                        dp[j] = min(dp[j], dp[j-coins[i]]+1)
                }
        }
        if dp[amount] == amount+1 {
                return -1
        }
        return dp[amount]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func coinChange(coins []int, amount int) int {
        if amount == 0 {
                return 0
        }
        res := 1
        sort.Ints(coins)
        list := make([]int, 0)
        list = append(list, amount)
        arr := make([]bool, amount+1)
        arr[amount] = true
        for len(list) > 0 {
                length := len(list)
```

(续下页)

(接上页)

```
                for i := 0; i < length; i++ {
                        value := list[i]
                        for j := 0; j < len(coins); j++ {
                                next := value - coins[j]
                                if next == 0 {
                                        return res
                                }
                                if next < 0 {
                                        break
                                }
                                if arr[next] == false {
                                        list = append(list, next)
                                        arr[next] = true
                                }
                        }
                }
                list = list[length:]
                res++
        }
        return -1
}
```

78.73 剑指 OfferII104. 排列的数目 (2)

- 题目

```
给定一个由 不同正整数组成的数组 nums ，和一个目标整数 target 。
请从 nums 中找出并返回总和为 target 的元素组合的个数。
数组中的数字可以在一次排列中出现任意次，但是顺序不同的序列被视作不同的组合。
题目数据保证答案符合 32 位整数范围。
示例 1：输入：nums = [1,2,3], target = 4 输出：7
解释：所有可能的组合为：
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
请注意，顺序不同的序列被视作不同的组合。
示例 2：输入：nums = [9], target = 3 输出：0
提示：1 <= nums.length <= 200
```

(续下页)

(接上页)

```
1 <= nums[i] <= 1000
nums 中的所有元素 互不相同
1 <= target <= 1000
进阶：如果给定的数组中含有负数会发生什么？问题会产生何种变化？如果允许负数出现，需要向题目中添加哪些
注意：本题与主站 377题相同：
```

- 解题思路

```
func combinationSum4(nums []int, target int) int {
        // 等价于：
        // 假设你正在爬楼梯。需要n阶你才能到达楼顶。
        // 每次你可以爬num(num in nums)级台阶。
        // 你有多少种不同的方法可以爬到楼顶呢？
        dp := make([]int, target+1)
        dp[0] = 1 // 爬0楼1种解法
        for i := 1; i <= target; i++ {
                for j := 0; j < len(nums); j++ {
                        if i-nums[j] >= 0 {
                                dp[i] = dp[i] + dp[i-nums[j]]
                        }
                }
        }
        return dp[target]
}

# 2
var m map[int]int

func combinationSum4(nums []int, target int) int {
        m = make(map[int]int)
        res := dfs(nums, target)
        if res == -1 {
                return 0
        }
        return res
}

func dfs(nums []int, target int) int {
        if target == 0 {
                return 1
        }
        if target < 0 {
                return -1
        }
```

(续下页)

(接上页)

```
        if v, ok := m[target]; ok {
                return v
        }
        temp := 0
        for i := 0; i < len(nums); i++ {
                if dfs(nums, target-nums[i]) != -1 {
                        temp = temp + dfs(nums, target-nums[i])
                }
        }
        m[target] = temp
        return temp
}
```

78.74 剑指 OfferII105. 岛屿的最大面积 (2)

- 题目

```
给定一个由0 和 1 组成的非空二维数组grid，用来表示海洋岛屿地图。
一个岛屿是由一些相邻的1(代表土地) 构成的组合，这里的「相邻」要求两个 1
↪必须在水平或者竖直方向上相邻。
你可以假设grid 的四个边缘都被 0（代表水）包围着。
找到给定的二维数组中最大的岛屿面积。如果没有岛屿，则返回面积为 0 。
示例 1:输入: grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,1,1,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,1,1,0,0,1,0,1,0,0],[0,1,0,0,1,1,0,0,1,1,1,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,0,1,1,0,0,0,0]]
输出: 6
解释: 对于上面这个给定矩阵应返回6。注意答案不应该是 11
↪，因为岛屿只能包含水平或垂直的四个方向的 1 。
示例 2:输入: grid = [[0,0,0,0,0,0,0,0]] 输出: 0
提示：m == grid.length
n == grid[i].length
1 <= m, n <= 50
grid[i][j] is either 0 or 1
注意：本题与主站 695题相同：
```

- 解题思路

```
func maxAreaOfIsland(grid [][]int) int {
        maxArea := 0
        for i := range grid {
                for j := range grid[i] {
                        maxArea = max(maxArea, getArea(grid, i, j))
```

(续下页)

(接上页)

```
                }
        }
        return maxArea
}

func getArea(grid [][]int, i, j int) int {
        if grid[i][j] == 0 {
                return 0
        }
        grid[i][j] = 0
        area := 1
        if i != 0 {
                area = area + getArea(grid, i-1, j)
        }
        if j != 0 {
                area = area + getArea(grid, i, j-1)
        }
        if i != len(grid)-1 {
                area = area + getArea(grid, i+1, j)
        }
        if j != len(grid[0])-1 {
                area = area + getArea(grid, i, j+1)
        }
        return area
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxAreaOfIsland(grid [][]int) int {
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if grid[i][j] == 1 {
                                value := dfs(grid, i, j)
                                if value > res {
                                        res = value
                                }
```

(续下页)

(接上页)

```
                        }
                }
        }
        return res
}

func dfs(grid [][]int, i, j int) int {
        if i < 0 || j < 0 || i >= len(grid) || j >= len(grid[0]) ||
                grid[i][j] == 0 {
                return 0
        }
        grid[i][j] = 0
        res := 1
        res = res + dfs(grid, i+1, j)
        res = res + dfs(grid, i-1, j)
        res = res + dfs(grid, i, j+1)
        res = res + dfs(grid, i, j-1)
        return res
}
```

78.75 剑指 OfferII106. 二分图 (3)

- 题目

```
存在一个 无向图 ，图中有 n 个节点。其中每个节点都有一个介于 0 到 n - 1␣
↪之间的唯一编号。
给定一个二维数组 graph，表示图，其中 graph[u] 是一个节点数组，由节点 u␣
↪的邻接节点组成。
形式上，对于graph[u] 中的每个 v ，都存在一条位于节点 u 和节点 v␣
↪之间的无向边。该无向图同时具有以下属性：
不存在自环（graph[u] 不包含 u）。
不存在平行边（graph[u] 不包含重复值）。
如果 v 在 graph[u] 内，那么 u 也应该在 graph[v] 内（该图是无向图）
这个图可能不是连通图，也就是说两个节点 u 和 v 之间可能不存在一条连通彼此的路径。
二分图 定义：如果能将一个图的节点集合分割成两个独立的子集 A 和 B␣
↪，并使图中的每一条边的两个节点一个来自 A 集合，
一个来自 B 集合，就将这个图称为 二分图 。
如果图是二分图，返回 true ；否则，返回 false 。
示例 1：输入：graph = [[1,2,3],[0,2],[0,1,3],[0,2]] 输出：false
解释：不能将节点分割成两个独立的子集，以使每条边都连通一个子集中的一个节点与另一个子集中的一个节点。
示例 2：输入：graph = [[1,3],[0,2],[1,3],[0,2]] 输出：true
解释：可以将节点分成两组：{0, 2} 和 {1, 3} 。
```

(续下页)

(接上页)

```
提示：graph.length == n
1 <= n <= 100
0 <= graph[u].length < n
0 <= graph[u][i] <= n - 1
graph[u] 不会包含 u
graph[u] 的所有值 互不相同
如果 graph[u] 包含 v，那么 graph[v] 也会包含 u
注意：本题与主站 785题相同：
```

- 解题思路

```
// 思路同leetcode886.可能的二分法
var m map[int]int

func isBipartite(graph [][]int) bool {
        n := len(graph)
        m = make(map[int]int) // 分组： 0一组，1一组
        for i := 0; i < n; i++ {
                // 没有被分配过，分配到0一组
                if _, ok := m[i]; ok == false && dfs(graph, i, 0) == false {
                        return false
                }
        }
        return true
}

func dfs(arr [][]int, index int, value int) bool {
        if v, ok := m[index]; ok {
                return v == value // 已经分配，查看是否同一组
        }
        m[index] = value
        for i := 0; i < len(arr[index]); i++ {
                target := arr[index][i]
                if dfs(arr, target, 1-value) == false { //␣
↪不喜欢的人，分配到对立组：1-value
                        return false
                }
        }
        return true
}

# 2
func isBipartite(graph [][]int) bool {
        n := len(graph)
```

(续下页)

(接上页)

```
        m := make(map[int]int) // 分组： 0一组，1一组
        for i := 0; i < n; i++ {
                // 没有被分配过，分配到0一组
                if _, ok := m[i]; ok == true {
                        continue
                }
                m[i] = 0
                queue := make([]int, 0)
                queue = append(queue, i)
                for len(queue) > 0 {
                        node := queue[0]
                        queue = queue[1:]
                        for i := 0; i < len(graph[node]); i++ {
                                target := graph[node][i]
                                if _, ok := m[target]; ok == false {
                                        m[target] = 1 - m[node] // 相反一组
                                        queue = append(queue, target)
                                } else if m[node] == m[target] { //␣
↪已经分配，查看是否同一组
                                        return false
                                }
                        }
                }
        }
        return true
}

# 3
func isBipartite(graph [][]int) bool {
        n := len(graph)
        fa = Init(n)
        for i := 0; i < n; i++ {
                for j := 0; j < len(graph[i]); j++ {
                        target := graph[i][j]
                        if find(i) == find(target) { // 和不喜欢的人在相同组，失败
                                return false
                        }
                        union(graph[i][0], target) // 不喜欢的人在同一组
                }
        }
        return true
}
```

(续下页)

(接上页)

```
var fa []int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        return arr
}

// 查询
func find(x int) int {
        for x != fa[x] {
                fa[x] = fa[fa[x]]
                x = fa[x]
        }
        return x
}

// 合并
func union(i, j int) {
        fa[find(i)] = find(j)
}
```

78.76 剑指 OfferII107. 矩阵中的距离 (3)

- 题目

```
给定一个由 0 和 1 组成的矩阵 mat，请输出一个大小相同的矩阵，其中每一个格子是 mat␣
↪中对应位置元素到最近的 0 的距离。
两个相邻元素间的距离为 1 。
示例 1：输入：mat = [[0,0,0],[0,1,0],[0,0,0]] 输出：[[0,0,0],[0,1,0],[0,0,0]]
示例 2：输入：mat = [[0,0,0],[0,1,0],[1,1,1]] 输出：[[0,0,0],[0,1,0],[1,2,1]]
提示：m == mat.length
n == mat[i].length
1 <= m, n <= 104
1 <= m * n <= 104
mat[i][j] is either 0 or 1.
mat 中至少有一个 0
注意：本题与主站 542题相同：
```

- 解题思路

```
func updateMatrix(matrix [][]int) [][]int {
        n := len(matrix)
        m := len(matrix[0])
        dp := make([][]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 1 {
                                dp[i][j] = math.MaxInt32 / 10
                                if i > 0 {
                                        dp[i][j] = min(dp[i][j], dp[i-1][j]+1)
                                }
                                if j > 0 {
                                        dp[i][j] = min(dp[i][j], dp[i][j-1]+1)
                                }
                        } else {
                                dp[i][j] = 0
                        }
                }
        }
        for i := n - 1; i >= 0; i-- {
                for j := m - 1; j >= 0; j-- {
                        if dp[i][j] > 1 {
                                if i < n-1 {
                                        dp[i][j] = min(dp[i][j], dp[i+1][j]+1)
                                }
                                if j < m-1 {
                                        dp[i][j] = min(dp[i][j], dp[i][j+1]+1)
                                }
                        }
                }
        }
        return dp
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
var dx = []int{-1, 1, 0, 0}
```

(续下页)

(接上页)

```
var dy = []int{0, 0, -1, 1}

func updateMatrix(matrix [][]int) [][]int {
        n := len(matrix)
        m := len(matrix[0])
        queue := make([][2]int, 0)
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 0 {
                                queue = append(queue, [2]int{i, j})
                        } else {
                                matrix[i][j] = -1
                        }
                }
        }
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                for i := 0; i < 4; i++ {
                        x := node[0] + dx[i]
                        y := node[1] + dy[i]
                        if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y] == -1 {
                                matrix[x][y] = matrix[node[0]][node[1]] + 1
                                queue = append(queue, [2]int{x, y})
                        }
                }
        }
        return matrix
}

# 3
func updateMatrix(matrix [][]int) [][]int {
        n := len(matrix)
        m := len(matrix[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == 1 {
                                matrix[i][j] = math.MaxInt32 / 10
                                if i > 0 {
                                        matrix[i][j] = min(matrix[i][j], matrix[i-
↪1][j]+1)
                                }
                                if j > 0 {
```

(续下页)

(接上页)

```
                                        matrix[i][j] = min(matrix[i][j], matrix[i][j-
↪1]+1)
                                }
                        } else {
                                matrix[i][j] = 0
                        }
                }
        }
        for i := n - 1; i >= 0; i-- {
                for j := m - 1; j >= 0; j-- {
                        if matrix[i][j] > 1 {
                                if i < n-1 {
                                        matrix[i][j] = min(matrix[i][j],␣
↪matrix[i+1][j]+1)
                                }
                                if j < m-1 {
                                        matrix[i][j] = min(matrix[i][j],␣
↪matrix[i][j+1]+1)
                                }
                        }
                }
        }
        return matrix
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

78.77 剑指 OfferII109. 开密码锁 (1)

- 题目

```
一个密码锁由 4个环形拨轮组成，每个拨轮都有 10 个数字： '0', '1', '2', '3', '4', '5',
↪'6', '7', '8', '9' 。
每个拨轮可以自由旋转：例如把 '9' 变为'0', '0' 变为 '9'␣
↪。每次旋转都只能旋转一个拨轮的一位数字。
锁的初始数字为 '0000' ，一个代表四个拨轮的数字的字符串。
列表 deadends␣
```

(续下页)

(接上页)

```
↪包含了一组死亡数字，一旦拨轮的数字和列表里的任何一个元素相同，这个锁将会被永久锁定，无法再被旋转。
字符串 target␣
↪代表可以解锁的数字，请给出解锁需要的最小旋转次数，如果无论如何不能解锁，返回 -1 。
示例 1:输入：deadends = ["0201","0101","0102","1212","2002"], target = "0202" 输出：6
解释：可能的移动序列为 "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" ->
↪"0202"。
注意 "0000" -> "0001" -> "0002" -> "0102" -> "0202"␣
↪这样的序列是不能解锁的，因为当拨动到 "0102" 时这个锁就会被锁定。
示例 2:输入：deadends = ["8888"], target = "0009" 输出：1
解释：把最后一位反向旋转一次即可 "0000" -> "0009"。
示例 3:输入：deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"],␣
↪target = "8888" 输出：-1
解释：无法旋转到目标数字且不被锁定。
示例 4:输入：deadends = ["0000"], target = "8888" 输出：-1
提示：1 <=deadends.length <= 500
deadends[i].length == 4
target.length == 4
target 不在 deadends 之中
target 和 deadends[i] 仅由若干位数字组成
注意：本题与主站 752题相同：
```

- 解题思路

```
func openLock(deadends []string, target string) int {
        m := make(map[string]int)
        m["0000"] = 0
        for i := 0; i < len(deadends); i++ {
                if deadends[i] == "0000" {
                        return -1
                }
                m[deadends[i]] = 0
        }
        if target == "0000" {
                return 0
        }
        if _, ok := m[target]; ok {
                return -1
        }
        queue := make([]string, 0)
        queue = append(queue, "0000")
        res := 0
        dir := []int{1, -1}
        for len(queue) > 0 {
                res++
```

(续下页)

(接上页)

```
                length := len(queue)
                for i := 0; i < length; i++ {
                        str := queue[i]
                        for j := 0; j < 4; j++ {
                                for k := 0; k < len(dir); k++ {
                                        char := string((int(str[j]-'0')+10+dir[k])%10+ '0')
                                        newStr := str[:j] + char + str[j+1:]
                                        if _, ok := m[newStr]; ok {
                                                continue
                                        }
                                        queue = append(queue, newStr)
                                        m[newStr] = 1
                                        if newStr == target {
                                                return res
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return -1
}
```

78.78 剑指 OfferII110. 所有路径 (1)

- 题目

```
给定一个有n个节点的有向无环图，用二维数组graph表示，请找到所有从0到n-1的路径并输出（不要求按顺序）。
graph的第 i 个数组中的单元都表示有向图中 i号节点所能到达的下一些结点（译者注：有向图是有方向的，即规定了 a→b 你就不能从 b→a）
，若为空，就是没有下一个节点了。
示例 1：输入：graph = [[1,2],[3],[3],[]] 输出：[[0,1,3],[0,2,3]]
解释：有两条路径 0 -> 1 -> 3 和 0 -> 2 -> 3
示例 2：输入：graph = [[4,3,1],[3,2,4],[3],[4],[]] 输出：[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]
示例 3：输入：graph = [[1],[]] 输出：[[0,1]]
示例 4：输入：graph = [[1,2,3],[2],[3],[]] 输出：[[0,1,2,3],[0,2,3],[0,3]]
示例 5：输入：graph = [[1,3],[2],[3],[]] 输出：[[0,1,2,3],[0,3]]
提示：n == graph.length
2 <= n <= 15
```

(续下页)

(接上页)

```
0 <= graph[i][j] < n
graph[i][j] != i
保证输入为有向无环图 (GAD)
注意：本题与主站 797题相同：
```

- 解题思路

```
var res [][]int

func allPathsSourceTarget(graph [][]int) [][]int {
        res = make([][]int, 0)
        dfs(graph, 0, len(graph)-1, make([]int, 0))
        return res
}

func dfs(graph [][]int, cur, target int, path []int) {
        if cur == target {
                path = append(path, cur)
                temp := make([]int, len(path))
                copy(temp, path)
                res = append(res, temp)
                return
        }
        for i := 0; i < len(graph[cur]); i++ {
                dfs(graph, graph[cur][i], target, append(path, cur))
        }
}
```

78.79 剑指 OfferII111. 计算除法 (3)

- 题目

```
给定一个变量对数组 equations 和一个实数值数组 values 作为已知条件，
其中 equations[i] = [Ai, Bi] 和 values[i] 共同表示等式 Ai / Bi = values[i] 。
每个 Ai 或 Bi 是一个表示单个变量的字符串。
另有一些以数组 queries 表示的问题，其中 queries[j] = [Cj, Dj] 表示第 j 个问题，
请你根据已知条件找出 Cj / Dj = ? 的结果作为答案。
返回 所有问题的答案 。如果存在某个无法确定的答案，则用 -1.0 替代这个答案。
如果问题中出现了给定的已知条件中没有出现的字符串，也需要用 -1.0 替代这个答案。
注意：输入总是有效的。可以假设除法运算中不会出现除数为 0␣
↪的情况，且不存在任何矛盾的结果。
示例 1：输入：equations = [["a","b"],["b","c"]], values = [2.0,3.0],
```

(续下页)

(接上页)

```
queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]
输出：[6.00000,0.50000,-1.00000,1.00000,-1.00000]
解释：条件：a / b = 2.0, b / c = 3.0
问题：a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
结果：[6.0, 0.5, -1.0, 1.0, -1.0 ]
示例 2：输入：equations = [["a","b"],["b","c"],["bc","cd"]],
values = [1.5,2.5,5.0], queries = [["a","c"],["c","b"],["bc","cd"],["cd","bc"]]
输出：[3.75000,0.40000,5.00000,0.20000]
示例 3：输入：equations = [["a","b"]], values = [0.5], queries = [["a","b"],["b","a"],
↪["a","c"],["x","y"]]
输出：[0.50000,2.00000,-1.00000,-1.00000]
提示：1 <= equations.length <= 20
equations[i].length == 2
1 <= Ai.length, Bi.length <= 5
values.length == equations.length
0.0 < values[i] <= 20.0
1 <= queries.length <= 20
queries[i].length == 2
1 <= Cj.length, Dj.length <= 5
Ai, Bi, Cj, Dj 由小写英文字母与数字组成
注意：本题与主站 399题相同：
```

- 解题思路

```
type Node struct {
        to    int
        value float64
}

func calcEquation(equations [][]string, values []float64, queries [][]string)␣
↪[]float64 {
        m := make(map[string]int) // 计算对应的id
        for i := 0; i < len(equations); i++ {
                a, b := equations[i][0], equations[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = len(m)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = len(m)
                }
        }
        arr := make([][]Node, len(m)) // 邻接表
        for i := 0; i < len(equations); i++ {
                a, b := m[equations[i][0]], m[equations[i][1]]
```

(续下页)

(接上页)

```
                arr[a] = append(arr[a], Node{to: b, value: values[i]})
                arr[b] = append(arr[b], Node{to: a, value: 1 / values[i]}) // 除以
        }
        res := make([]float64, len(queries))
        for i := 0; i < len(queries); i++ {
                a, okA := m[queries[i][0]]
                b, okB := m[queries[i][1]]
                if okA == false || okB == false {
                        res[i] = -1
                } else {
                        res[i] = bfs(arr, a, b) // 广度优先查找
                }
        }
        return res
}

func bfs(arr [][]Node, start, end int) float64 {
        temp := make([]float64, len(arr)) // 结果的比例
        temp[start] = 1
        queue := make([]int, 0)
        queue = append(queue, start)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node == end {
                        return temp[node]
                }
                for i := 0; i < len(arr[node]); i++ {
                        next := arr[node][i].to
                        if temp[next] == 0 {
                                temp[next] = temp[node] * arr[node][i].value
                                queue = append(queue, next)
                        }
                }
        }
        return -1
}

# 2
func calcEquation(equations [][]string, values []float64, queries [][]string)␣
↪[]float64 {
        m := make(map[string]int) // 计算对应的id
        for i := 0; i < len(equations); i++ {
```

(续下页)

(接上页)

```
                a, b := equations[i][0], equations[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = len(m)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = len(m)
                }
        }
        arr := make([][]float64, len(m)) // 邻接矩阵
        for i := 0; i < len(m); i++ {
                arr[i] = make([]float64, len(m))
        }
        for i := 0; i < len(equations); i++ {
                a, b := m[equations[i][0]], m[equations[i][1]]
                arr[a][b] = values[i]
                arr[b][a] = 1 / values[i]
        }
        for k := 0; k < len(arr); k++ { // Floyd
                for i := 0; i < len(arr); i++ {
                        for j := 0; j < len(arr); j++ {
                                if arr[i][k] > 0 && arr[k][j] > 0 {
                                        arr[i][j] = arr[i][k] * arr[k][j]
                                }
                        }
                }
        }
        res := make([]float64, len(queries))
        for i := 0; i < len(queries); i++ {
                a, okA := m[queries[i][0]]
                b, okB := m[queries[i][1]]
                if okA == false || okB == false || arr[a][b] == 0 {
                        res[i] = -1
                } else {
                        res[i] = arr[a][b]
                }
        }
        return res
}

# 3
func calcEquation(equations [][]string, values []float64, queries [][]string)␣
↪[]float64 {
        m := make(map[string]int) // 计算对应的id
```

(续下页)

(接上页)

```
        for i := 0; i < len(equations); i++ {
                a, b := equations[i][0], equations[i][1]
                if _, ok := m[a]; ok == false {
                        m[a] = len(m)
                }
                if _, ok := m[b]; ok == false {
                        m[b] = len(m)
                }
        }
        fa, rank = Init(len(m))
        for i := 0; i < len(equations); i++ {
                a, b := m[equations[i][0]], m[equations[i][1]]
                union(a, b, values[i])
        }
        res := make([]float64, len(queries))
        for i := 0; i < len(queries); i++ {
                a, okA := m[queries[i][0]]
                b, okB := m[queries[i][1]]
                if okA == true && okB == true && find(a) == find(b) {
                        res[i] = rank[a] / rank[b]
                } else {
                        res[i] = -1
                }
        }
        return res
}

var fa []int
var rank []float64

// 初始化
func Init(n int) ([]int, []float64) {
        arr := make([]int, n)
        r := make([]float64, n)
        for i := 0; i < n; i++ {
                arr[i] = i
                r[i] = 1
        }
        return arr, r
}

// 查询
func find(x int) int {
```

(续下页)

(接上页)

```
        // 彻底路径压缩
        if fa[x] != x {
                origin := fa[x]
                fa[x] = find(fa[x])
                rank[x] = rank[x] * rank[origin] // 秩处理是难点
        }
        return fa[x]
}

// 合并
func union(i, j int, value float64) {
        x, y := find(i), find(j)
        rank[x] = value * rank[j] / rank[i] // 秩处理是难点
        fa[x] = y
}
```

78.80 剑指 OfferII113. 课程顺序 (2)

- 题目

```
现在总共有 numCourses门课需要选，记为0到numCourses-1。
给定一个数组prerequisites ，它的每一个元素prerequisites[i]表示两门课程之间的先修顺序。
例如prerequisites[i] = [ai, bi]表示想要学习课程 ai，需要先完成课程 bi。
请根据给出的总课程数 numCourses 和表示先修顺序的prerequisites得出一个可行的修课序列。
可能会有多个正确的顺序，只要任意返回一种就可以了。如果不可能完成所有课程，返回一个空数组。
示例1:输入: numCourses = 2, prerequisites = [[1,0]]  输出: [0,1]
解释:总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1]␣
↪。
示例2:输入: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]] 输出: [0,1,2,3]␣
↪or [0,2,1,3]
解释:总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。并且课程 1 和课程 2␣
↪都应该排在课程 0 之后。
因此，一个正确的课程顺序是[0,1,2,3] 。另一个正确的排序是[0,2,1,3] 。
示例 3:输入: numCourses = 1, prerequisites = []  输出: [0]
解释:总共 1 门课，直接修第一门课就可。
提示:1 <= numCourses <= 2000
0 <= prerequisites.length <= numCourses * (numCourses - 1)
prerequisites[i].length == 2
0 <= ai, bi < numCourses
ai != bi
prerequisites中不存在重复元素
注意：本题与主站 210题相同：
```

- 解题思路

```
var res bool
var visited []int
var path []int
var edges [][]int

func findOrder(numCourses int, prerequisites [][]int) []int {
        res = true
        edges = make([][]int, numCourses) // 邻接表
        visited = make([]int, numCourses)
        path = make([]int, 0)
        for i := 0; i < len(prerequisites); i++ {
                // prev->cur
                prev := prerequisites[i][1]
                cur := prerequisites[i][0]
                edges[prev] = append(edges[prev], cur)
        }
        for i := 0; i < numCourses; i++ {
                if visited[i] == 0 {
                        dfs(i)
                }
                if res == false {
                        return nil
                }
        }
        for i := 0; i < len(path)/2; i++ {
                path[i], path[len(path)-1-i] = path[len(path)-1-i], path[i]
        }
        return path
}

func dfs(start int) {
        // 0 未搜索
        // 1 搜索中
        // 2 已完成
        visited[start] = 1
        for i := 0; i < len(edges[start]); i++ {
                out := edges[start][i]
                if visited[out] == 0 {
                        dfs(out)
                        if res == false {
                                return
                        }
                } else if visited[out] == 1 {
```

(续下页)

(接上页)

```
                        res = false
                        return
                }
        }
        visited[start] = 2
        path = append(path, start)
}

# 2
func findOrder(numCourses int, prerequisites [][]int) []int {
        edges := make([][]int, numCourses)
        path := make([]int, 0)
        inEdges := make([]int, numCourses)
        for i := 0; i < len(prerequisites); i++ {
                // prev->cur
                prev := prerequisites[i][1]
                cur := prerequisites[i][0]
                edges[prev] = append(edges[prev], cur)
                inEdges[cur]++ // 入度
        }
        // 入度为0
        queue := make([]int, 0)
        for i := 0; i < numCourses; i++ {
                if inEdges[i] == 0 {
                        queue = append(queue, i)
                }
        }
        for len(queue) > 0 {
                start := queue[0]
                queue = queue[1:]
                path = append(path, start)
                for i := 0; i < len(edges[start]); i++ {
                        out := edges[start][i]
                        inEdges[out]--
                        if inEdges[out] == 0 {
                                queue = append(queue, out)
                        }
                }
        }
        if len(path) != numCourses {
                return nil
        }
        return path
}
```

78.81 剑指 OfferII115. 重建序列 (1)

- 题目

```
请判断原始的序列org是否可以从序列集seqs中唯一地 重建。
序列org是 1 到 n 整数的排列，其中 1 ≤ n ≤ 104。
重建是指在序列集 seqs␣
↪中构建最短的公共超序列，即seqs中的任意序列都是该最短序列的子序列。
示例 1：输入：org = [1,2,3], seqs = [[1,2],[1,3]] 输出：false
解释：[1,2,3] 不是可以被重建的唯一的序列，因为 [1,3,2] 也是一个合法的序列。
示例 2：输入：org = [1,2,3], seqs = [[1,2]] 输出：false
解释：可以重建的序列只有 [1,2]。
示例 3：输入：org = [1,2,3], seqs = [[1,2],[1,3],[2,3]] 输出：true
解释：序列 [1,2], [1,3] 和 [2,3] 可以被唯一地重建为原始的序列 [1,2,3]。
示例 4：输入：org = [4,1,5,2,6,3], seqs = [[5,2,6,3],[4,1,5,2]] 输出：true
提示：1 <= n <= 104
org 是数字 1 到 n 的一个排列
1 <= seqs[i].length <= 105
seqs[i][j] 是 32 位有符号整数
注意：本题与主站 444题相同：
```

- 解题思路

```
func sequenceReconstruction(org []int, seqs [][]int) bool {
        n := len(org)
        degree := make(map[int]int) // 入度
        arr := make([][]bool, n+1)  // 邻接矩阵
        for i := 0; i < n+1; i++ {
                arr[i] = make([]bool, n+1)
        }
        for i := 0; i < len(seqs); i++ {
                for j := 0; j < len(seqs[i]); j++ {
                        if seqs[i][j] < 1 || seqs[i][j] > n { // 范围不对
                                return false
                        }
                        if _, ok := degree[seqs[i][j]]; ok == false {
                                degree[seqs[i][j]] = 0 // 入度设置为0
                        }
                        if 0 < j {
                                if arr[seqs[i][j-1]][seqs[i][j]] == false {
                                        arr[seqs[i][j-1]][seqs[i][j]] = true // a=>
↪b: seqs[i][j-1] => seqs[i][j]
                                        degree[seqs[i][j]]++
                                }
```

(续下页)

(接上页)

```
                        }
                }
        }
        if len(degree) != n { // 数量不对
                return false
        }
        queue := make([]int, 0) // 拓扑排序: 入度=0进队列
        for i := 1; i <= n; i++ {
                if v, ok := degree[i]; ok == true && v == 0 {
                        queue = append(queue, i)
                }
        }
        index := 0 // 依次对比数据
        for len(queue) > 0 {
                length := len(queue)
                if length > 1 { // 多个入度=0不能唯一重建
                        return false
                }
                if org[index] != queue[0] { // 序列不对
                        return false
                }
                for i := 0; i < len(arr[queue[0]]); i++ {
                        if arr[queue[0]][i] == true {
                                degree[i]--
                                if degree[i] == 0 {
                                        queue = append(queue, i)
                                }
                        }
                }
                queue = queue[1:]
                index++
        }
        return index == n
}
```

78.82 剑指 OfferII116. 省份数量 (3)

- 题目

```
有 n 个城市，其中一些彼此相连，另一些没有相连。
如果城市 a 与城市 b 直接相连，且城市 b 与城市 c 直接相连，那么城市 a 与城市 c␣
↪间接相连。
省份 是一组直接或间接相连的城市，组内不含其他没有相连的城市。
给你一个 n x n 的矩阵 isConnected ，其中 isConnected[i][j] = 1 表示第 i 个城市和第 j␣
↪个城市直接相连，
而 isConnected[i][j] = 0 表示二者不直接相连。
返回矩阵中 省份 的数量。
示例 1：输入：isConnected = [[1,1,0],[1,1,0],[0,0,1]] 输出：2
示例 2：输入：isConnected = [[1,0,0],[0,1,0],[0,0,1]] 输出：3
提示：1 <= n <= 200
n == isConnected.length
n == isConnected[i].length
isConnected[i][j] 为 1 或 0
isConnected[i][i] == 1
isConnected[i][j] == isConnected[j][i]
注意：本题与主站 547题相同：
```

- 解题思路

```
func findCircleNum(M [][]int) int {
        n := len(M)
        fa = Init(n)
        count = n
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if M[i][j] == 1 {
                                union(i, j)
                        }
                }
        }
        return getCount()
}

var fa []int
var count int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        count = n
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
        }
}

func query(i, j int) bool {
        return find(i) == find(j)
}

func getCount() int {
        return count
}

# 2
var arr []bool

func findCircleNum(M [][]int) int {
        n := len(M)
        arr = make([]bool, n)
        res := 0
        for i := 0; i < n; i++ {
                if arr[i] == false {
```

(续下页)

(接上页)

```
                        dfs(M, i)
                        res++
                }
        }
        return res
}

func dfs(M [][]int, index int) {
        for i := 0; i < len(M); i++ {
                if arr[i] == false && M[index][i] == 1 {
                        arr[i] = true
                        dfs(M, i)
                }
        }
}

# 3
func findCircleNum(M [][]int) int {
        n := len(M)
        arr := make([]bool, n)
        res := 0
        queue := make([]int, 0)
        for i := 0; i < n; i++ {
                if arr[i] == false {
                        queue = append(queue, i)
                        for len(queue) > 0 {
                                node := queue[0]
                                queue = queue[1:]
                                arr[node] = true
                                for j := 0; j < n; j++ {
                                        if M[node][j] == 1 && arr[j] == false {
                                                queue = append(queue, j)
                                        }
                                }
                        }
                        res++
                }
        }
        return res
}
```

78.83 剑指 OfferII118. 多余的边 (1)

- 题目

```
树可以看成是一个连通且 无环的无向图。
给定往一棵n 个节点 (节点值1～n) 的树中添加一条边后的图。
添加的边的两个顶点包含在 1 到 n中间，且这条附加的边不属于树中已存在的边。
图的信息记录于长度为 n 的二维数组 edges，edges[i] = [ai, bi]表示图中在 ai 和 bi
↪之间存在一条边。
请找出一条可以删去的边，删除后可使得剩余部分是一个有着 n
↪个节点的树。如果有多个答案，则返回数组edges中最后出现的边。
示例 1：输入：edges = [[1,2],[1,3],[2,3]] 输出：[2,3]
示例 2：输入：edges = [[1,2],[2,3],[3,4],[1,4],[1,5]] 输出：[1,4]
提示:n == edges.length
3 <= n <= 1000
edges[i].length == 2
1 <= ai< bi<= edges.length
ai != bi
edges 中无重复元素
给定的图是连通的
注意：本题与主站 684题相同
```

- 解题思路

```
func findRedundantConnection(edges [][]int) []int {
        n := len(edges) + 1
        fa := make([]int, n)
        for i := 0; i < n; i++ {
                fa[i] = i
        }
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                if find(fa, a) == find(fa, b) {
                        return edges[i]
                }
                union(fa, a, b)
        }
        return nil
}

func union(fa []int, a, b int) {
        fa[find(fa, a)] = find(fa, b)
}
```

(续下页)

(接上页)

```
func find(fa []int, a int) int {
        for fa[a] != a {
                fa[a] = fa[fa[a]]
                a = fa[a]
        }
        return a
}
```

78.84 剑指 OfferII119. 最长连续序列 (4)

- 题目

```
给定一个未排序的整数数组 nums␣
↪，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。
示例 1：输入：nums = [100,4,200,1,3,2] 输出：4
解释：最长数字连续序列是 [1, 2, 3, 4]。它的长度为 4。
示例 2：输入：nums = [0,3,7,2,5,8,4,6,0,1] 输出：9
提示：0 <= nums.length <= 104
-109 <= nums[i] <= 109
进阶：可以设计并实现时间复杂度为O(n) 的解决方案吗？
注意：本题与主站 128题相同：
```

- 解题思路

```
func longestConsecutive(nums []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = true
        }
        res := 0
        for i := 0; i < len(nums); i++ {
                if _, ok := m[nums[i]-1]; !ok {
                        cur := nums[i]
                        count := 1
                        for m[cur+1] == true {
                                count = count + 1
                                cur = cur + 1
                        }
                        res = max(res, count)
                }
        }
        return res
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func longestConsecutive(nums []int) int {
        if len(nums) <= 1 {
                return len(nums)
        }
        sort.Ints(nums)
        res := 1
        count := 1
        for i := 1; i < len(nums); i++ {
                if nums[i] == nums[i-1] {
                        continue
                } else if nums[i] == nums[i-1]+1 {
                        count++
                } else {
                        res = max(res, count)
                        count = 1
                }
        }
        res = max(res, count)
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func longestConsecutive(nums []int) int {
        m := make(map[int]int)
        res := 0
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                if m[nums[i]] > 0 {
                        continue
                }
                left := m[nums[i]-1]
                right := m[nums[i]+1]
                sum := left + 1 + right
                res = max(res, sum)
                m[nums[i]] = sum
                m[nums[i]-left] = sum
                m[nums[i]+right] = sum
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func longestConsecutive(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        m := make(map[int]int)
        res := 1
        fa = Init(nums)
        for i := 0; i < len(nums); i++ {
                union(nums[i], nums[i]+1)
                m[nums[i]]++
        }
        for i := 0; i < len(nums); i++ {
                res = max(res, find(nums[i])-nums[i]+1)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

var fa map[int]int

// 初始化
func Init(data []int) map[int]int {
        n := len(data)
        arr := make(map[int]int)
        for i := 0; i < n; i++ {
                arr[data[i]] = data[i]
        }
        return arr
}

// 查询
func find(x int) int {
        if _, ok := fa[x]; !ok {
                return math.MinInt32 // 特殊处理
        }
        res := x
        for res != fa[res] {
                res = fa[res]
        }
        return res
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x == y {
                return
        } else if x == math.MinInt32 || y == math.MinInt32 {
                return
        }
        fa[x] = y
}

func query(i, j int) bool {
        return find(i) == find(j)
}
```

CHAPTER 79

剑指 OfferII-Hard

79.1 剑指 OfferII017. 含有所有字符的最短字符串 (2)

- 题目

```
给定两个字符串 s 和t 。返回 s 中包含t的所有字符的最短子字符串。如果 s␣
↪中不存在符合条件的子字符串，则返回空字符串 "" 。
如果 s 中存在多个符合条件的子字符串，返回任意一个。
注意： 对于 t 中重复字符，我们寻找的子字符串中该字符数量必须不少于 t 中该字符数量。
示例 1：输入：s = "ADOBECODEBANC", t = "ABC" 输出："BANC"
解释：最短子字符串 "BANC" 包含了字符串 t 的所有字符 'A'、'B'、'C'
示例 2：输入：s = "a", t = "a" 输出："a"
示例 3：输入：s = "a", t = "aa" 输出：""
解释：t 中两个字符 'a' 均应包含在 s␣
↪的子串中，因此没有符合条件的子字符串，返回空字符串。
提示：1 <= s.length, t.length <= 105
s 和 t 由英文字母组成
进阶：你能设计一个在 o(n) 时间内解决此问题的算法吗？
注意：本题与主站 76题相似（本题答案不唯一）：
```

- 解题思路

```
func minWindow(s string, t string) string {
        if len(s) < len(t) {
                return ""
```

(续下页)

(接上页)

```
        }
        window := make(map[byte]int)
        need := make(map[byte]int)
        for i := 0; i < len(t); i++ {
                need[t[i]]++
        }
        left, right := -1, -1
        minLength := math.MaxInt32
        for l, r := 0, 0; r < len(s); r++ {
                if r < len(s) && need[s[r]] > 0 {
                        window[s[r]]++
                }
                // 找到，然后left往右移
                for check(need, window) == true && l <= r {
                        if r-l+1 < minLength {
                                minLength = r - l + 1
                                left, right = l, r+1
                        }
                        if _, ok := need[s[l]]; ok {
                                window[s[l]]--
                        }
                        l++
                }
        }
        if left == -1 {
                return ""
        }
        return s[left:right]
}

func check(need, window map[byte]int) bool {
        for k, v := range need {
                if window[k] < v {
                        return false
                }
        }
        return true
}

# 2
func minWindow(s string, t string) string {
        if len(s) < len(t) {
                return ""
```

(续下页)

(接上页)

```
        }
        arr := make(map[byte]int)
        for i := 0; i < len(t); i++ {
                arr[t[i]]++
        }
        l, count := 0, 0
        res := ""
        minLength := math.MaxInt32
        for r := 0; r < len(s); r++ {
                arr[s[r]]--
                if arr[s[r]] >= 0 {
                        count++
                }
                // left往右边移动
                for count == len(t) {
                        if minLength > r-l+1 {
                                minLength = r - l + 1
                                res = s[l : r+1]
                        }
                        arr[s[l]]++
                        if arr[s[l]] > 0 {
                                count--
                        }
                        l++
                }
        }
        return res
}
```

79.2 剑指 OfferII039. 直方图最大矩形面积 (3)

- 题目

```
给定非负整数数组␣
↪heights，数组中的数字用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。
求在该柱状图中，能够勾勒出来的矩形的最大面积。
示例 1:输入：heights = [2,1,5,6,2,3] 输出：10
解释：最大的矩形为图中红色区域，面积为 10
示例 2：输入： heights = [2,4] 输出： 4
提示：1 <= heights.length <=105
0 <= heights[i] <= 104
注意：本题与主站 84题相同：
```

- 解题思路

```
func largestRectangleArea(heights []int) int {
        n := len(heights)
        res := 0
        left := make([]int, n)
        right := make([]int, n)
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                for len(stack) > 0 && heights[stack[len(stack)-1]] >= heights[i] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        left[i] = -1
                } else {
                        left[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }
        stack = make([]int, 0)
        for i := n - 1; i >= 0; i-- {
                for len(stack) > 0 && heights[stack[len(stack)-1]] >= heights[i] {
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        right[i] = n
                } else {
                        right[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }
        for i := 0; i < n; i++ {
                res = max(res, heights[i]*(right[i]-left[i]-1))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
func largestRectangleArea(heights []int) int {
        n := len(heights)
        res := 0
        left := make([]int, n)
        right := make([]int, n)
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
                right[i] = n
        }
        for i := 0; i < n; i++ {
                for len(stack) > 0 && heights[stack[len(stack)-1]] >= heights[i] {
                        right[stack[len(stack)-1]] = i
                        stack = stack[:len(stack)-1]
                }
                if len(stack) == 0 {
                        left[i] = -1
                } else {
                        left[i] = stack[len(stack)-1]
                }
                stack = append(stack, i)
        }

        for i := 0; i < n; i++ {
                res = max(res, heights[i]*(right[i]-left[i]-1))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func largestRectangleArea(heights []int) int {
        heights = append([]int{0}, heights...)
        heights = append(heights, 0)
        n := len(heights)
        res := 0
        stack := make([]int, 0)
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                // 递增栈
                for len(stack) > 0 && heights[stack[len(stack)-1]] > heights[i] {
                        height := heights[stack[len(stack)-1]]
                        stack = stack[:len(stack)-1]
                        width := i - stack[len(stack)-1] - 1
                        res = max(res, height*width)
                }
                stack = append(stack, i)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

79.3 剑指 OfferII040. 矩阵中最大的矩形 (2)

- 题目

```
给定一个由0 和 1组成的矩阵 matrix，找出只包含 1 的最大矩形，并返回其面积。
注意：此题 matrix输入格式为一维 01 字符串数组。
示例 1：输入：matrix = ["10100","10111","11111","10010"] 输出：6
解释：最大矩形如上图所示。
示例 2：输入：matrix = [] 输出：0
示例 3：输入：matrix = ["0"] 输出：0
示例 4：输入：matrix = ["1"] 输出：1
示例 5：输入：matrix = ["00"] 输出：0
提示：rows == matrix.length
cols == matrix[0].length
0 <= row, cols <= 200
matrix[i][j] 为 '0' 或 '1'
注意：本题与主站 85 题相同（输入参数格式不同）：
```

- 解题思路

```
func maximalRectangle(matrix []string) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
```

(续下页)

(接上页)

```
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        height := make([]int, m) // 高度
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '0' {
                                height[j] = 0
                        } else {
                                height[j] = height[j] + 1
                        }
                }
                res = max(res, getMaxArea(height))
        }
        return res
}

func getMaxArea(heights []int) int {
        heights = append([]int{0}, heights...)
        heights = append(heights, 0)
        n := len(heights)
        res := 0
        stack := make([]int, 0) // 递增栈
        for i := 0; i < n; i++ {
                for len(stack) > 0 && heights[stack[len(stack)-1]] > heights[i] {
                        height := heights[stack[len(stack)-1]]
                        stack = stack[:len(stack)-1]
                        width := i - stack[len(stack)-1] - 1
                        res = max(res, height*width)
                }
                stack = append(stack, i)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
```

(续下页)

(接上页)

```
func maximalRectangle(matrix []string) int {
        if len(matrix) == 0 || len(matrix[0]) == 0 {
                return 0
        }
        res := 0
        n, m := len(matrix), len(matrix[0])
        left, right, height := make([]int, m), make([]int, m), make([]int, m)
        for i := 0; i < m; i++ {
                right[i] = m
        }
        for i := 0; i < n; i++ {
                curLeft, curRight := 0, m
                // 高度
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '1' {
                                height[j]++
                        } else {
                                height[j] = 0
                        }
                }
                // 左边
                for j := 0; j < m; j++ {
                        if matrix[i][j] == '1' {
                                left[j] = max(left[j], curLeft)
                        } else {
                                left[j] = 0
                                curLeft = j + 1
                        }
                }
                // 右边
                for j := m - 1; j >= 0; j-- {
                        if matrix[i][j] == '1' {
                                right[j] = min(right[j], curRight)
                        } else {
                                right[j] = m
                                curRight = j
                        }
                }
                for j := 0; j < m; j++ {
                        res = max(res, height[j]*(right[j]-left[j]))
                }
        }
        return res
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

79.4 剑指 OfferII048. 序列化与反序列化二叉树 (2)

- 题目

```
序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存
同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。
请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 /␣
↪反序列化算法执行逻辑，
只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。
示例 1：输入：root = [1,2,3,null,null,4,5] 输出：[1,2,3,null,null,4,5]
示例 2：输入：root = [] 输出：[]
示例 3：输入：root = [1] 输出：[1]
示例 4：输入：root = [1,2] 输出：[1,2]
提示：输入输出格式与 LeetCode 目前使用的方式一致，详情请参阅LeetCode␣
↪序列化二叉树的格式。
你并非必须采取这种方式，也可以采用其他的方法解决这个问题。
树中结点数在范围 [0, 104] 内
-1000 <= Node.val <= 1000
注意：本题与主站 297题相同：
```

- 解题思路

```
type Codec struct {
        res []string
}
```

(续下页)

(接上页)

```
func Constructor() Codec {
        return Codec{}
}

// Serializes a tree to a single string.
func (this *Codec) serialize(root *TreeNode) string {
        if root == nil {
                return "#"
        }
        return strconv.Itoa(root.Val) + "," + this.serialize(root.Left) + "," + this.
↪serialize(root.Right)
}

// Deserializes your encoded data to tree.
func (this *Codec) deserialize(data string) *TreeNode {
        this.res = strings.Split(data, ",")
        return this.dfsDeserialize()
}

func (this *Codec) dfsDeserialize() *TreeNode {
        node := this.res[0]
        this.res = this.res[1:]
        if node == "#" {
                return nil
        }
        value, _ := strconv.Atoi(node)
        return &TreeNode{
                Val:   value,
                Left:  this.dfsDeserialize(),
                Right: this.dfsDeserialize(),
        }
}

# 2
type Codec struct {
        res []string
}

func Constructor() Codec {
        return Codec{}
}

// Serializes a tree to a single string.
```

(续下页)

(接上页)

```
func (this *Codec) serialize(root *TreeNode) string {
        if root == nil {
                return ""
        }
        res := make([]string, 0)
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node != nil {
                        res = append(res, strconv.Itoa(node.Val))
                        queue = append(queue, node.Left, node.Right)
                } else {
                        res = append(res, "#")
                }
        }
        return strings.Join(res, ",")
}

// Deserializes your encoded data to tree.
func (this *Codec) deserialize(data string) *TreeNode {
        if len(data) == 0 || data == "" {
                return nil
        }
        res := strings.Split(data, ",")
        root := &TreeNode{}
        root.Val, _ = strconv.Atoi(res[0])
        res = res[1:]
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        for len(queue) > 0 {
                if res[0] != "#" {
                        left, _ := strconv.Atoi(res[0])
                        queue[0].Left = &TreeNode{Val: left}
                        queue = append(queue, queue[0].Left)
                }
                if res[1] != "#" {
                        right, _ := strconv.Atoi(res[1])
                        queue[0].Right = &TreeNode{Val: right}
                        queue = append(queue, queue[0].Right)
                }
                queue = queue[1:]
```

(续下页)

(接上页)

```
                res = res[2:]
        }
        return root
}
```

79.5 剑指 OfferII051. 节点之和最大的路径 (2)

- 题目

```
路径 被定义为一条从树中任意节点出发，沿父节点-
↪子节点连接，达到任意节点的序列。同一个节点在一条路径序列中 至多出现一次 。
该路径 至少包含一个 节点，且不一定经过根节点。
路径和 是路径中各节点值的总和。
给定一个二叉树的根节点 root ，返回其 最大路径和，即所有路径上节点值之和的最大值。
示例 1：输入：root = [1,2,3] 输出：6
解释：最优路径是 2 -> 1 -> 3 ，路径和为 2 + 1 + 3 = 6
示例 2：输入：root = [-10,9,20,null,null,15,7] 输出：42
解释：最优路径是 15 -> 20 -> 7 ，路径和为 15 + 20 + 7 = 42
提示：树中节点数目范围是 [1, 3 * 104]
-1000 <= Node.val <= 1000
注意：本题与主站 124题相同：
```

- 解题思路

```
var res int

func maxPathSum(root *TreeNode) int {
        res = math.MinInt32
        dfs(root)
        return res
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := max(dfs(root.Left), 0)
        right := max(dfs(root.Right), 0)
        // 该顶点路径和=root.Val+2边和
        value := left + right + root.Val
        res = max(res, value)
        // 单分支
```

(续下页)

(接上页)

```
        return root.Val + max(left, right)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var res int

func maxPathSum(root *TreeNode) int {
        res = math.MinInt32
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        stack := make([]*TreeNode, 0)
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
                stack = append(stack, node)
        }
        for len(stack) > 0 {
                node := stack[len(stack)-1]
                stack = stack[:len(stack)-1]
                res = max(res, node.Val)
                var left, right int
                if node.Left == nil {
                        left = 0
                } else {
                        left = max(node.Left.Val, 0)
                }
                if node.Right == nil {
                        right = 0
                } else {
                        right = max(node.Right.Val, 0)
```

(续下页)

(接上页)

```
                }
                sum := node.Val + left + right
                res = max(res, sum)
                node.Val = node.Val + max(left, right)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

79.6 剑指 OfferII078. 合并排序链表 (4)

- 题目

```
给定一个链表数组，每个链表都已经按升序排列。
请将所有链表合并到一个升序链表中，返回合并后的链表。
示例 1: 输入: lists = [[1,4,5],[1,3,4],[2,6]] 输出: [1,1,2,3,4,4,5,6]
解释: 链表数组如下:
[
  1->4->5,
  1->3->4,
  2->6
]
将它们合并到一个有序链表中得到。
1->1->2->3->4->4->5->6
示例 2: 输入: lists = [] 输出: []
示例 3: 输入: lists = [[]] 输出: []
提示: k == lists.length
0 <= k <= 10^4
0 <= lists[i].length <= 500
-10^4 <= lists[i][j] <= 10^4
lists[i] 按 升序 排列
lists[i].length 的总和不超过 10^4
注意: 本题与主站 23题相同:
```

- 解题思路

```
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        temp := &ListNode{}
        for i := 0; i < len(lists); i++ {
                temp.Next = mergeTwoLists(temp.Next, lists[i])
        }
        return temp.Next
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}

# 2
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        if len(lists) == 1 {
                return lists[0]
        }
        first := mergeKLists(lists[:len(lists)/2])
        second := mergeKLists(lists[len(lists)/2:])
        return mergeTwoLists(first, second)
}
```

(续下页)

(接上页)

```
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        temp := res
        for l1 != nil && l2 != nil {
                if l1.Val < l2.Val {
                        temp.Next = l1
                        l1 = l1.Next
                } else {
                        temp.Next = l2
                        l2 = l2.Next
                }
                temp = temp.Next
        }
        if l1 != nil {
                temp.Next = l1
        } else {
                temp.Next = l2
        }
        return res.Next
}

# 3
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        var h IntHeap
        heap.Init(&h)
        for i := 0; i < len(lists); i++ {
                if lists[i] != nil {
                        heap.Push(&h, lists[i])
                }
        }
        res := &ListNode{}
        temp := res
        for h.Len() > 0 {
                minItem := heap.Pop(&h).(*ListNode)
                temp.Next = minItem
                temp = temp.Next
                if minItem.Next != nil {
                        heap.Push(&h, minItem.Next)
                }
```

(续下页)

(接上页)

```
        }
        return res.Next
}

type IntHeap []*ListNode

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i].Val < h[j].Val }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.(*ListNode)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 4
func mergeKLists(lists []*ListNode) *ListNode {
        if len(lists) == 0 {
                return nil
        }
        arr := make([]*ListNode, 0)
        for i := 0; i < len(lists); i++ {
                temp := lists[i]
                for temp != nil {
                        arr = append(arr, temp)
                        temp = temp.Next
                }
        }
        if len(arr) == 0 {
                return nil
        }
        sort.Slice(arr, func(i, j int) bool {
                return arr[i].Val < arr[j].Val
        })
        for i := 0; i < len(arr)-1; i++ {
                arr[i].Next = arr[i+1]
        }
        arr[len(arr)-1].Next = nil
        return arr[0]
}
```

79.7 剑指 OfferII094. 最少回文分割 (2)

- 题目

```
给定一个字符串 s，请将 s 分割成一些子串，使每个子串都是回文串。
返回符合要求的 最少分割次数 。
示例 1：输入：s = "aab" 输出：1
解释：只需一次分割就可将s 分割成 ["aa","b"] 这样两个回文子串。
示例 2：输入：s = "a" 输出：0
示例 3：输入：s = "ab" 输出：1
提示：1 <= s.length <= 2000
s 仅由小写英文字母组成
注意：本题与主站 132题相同：
```

- 解题思路

```
func minCut(s string) int {
        if len(s) == 0 || len(s) == 1 {
                return 0
        }
        dp := make([]int, len(s)+1)
        dp[0] = -1
        dp[1] = 1
        for i := 1; i <= len(s); i++ {
                dp[i] = i - 1 // 长度N切分n-1次
                for j := 0; j < i; j++ {
                        if judge(s[j:i]) {
                                dp[i] = min(dp[i], dp[j]+1)
                        }
                }
        }
        return dp[len(s)]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func judge(s string) bool {
        for i := 0; i < len(s)/2; i++ {
                if s[i] != s[len(s)-1-i] {
```

(续下页)

(接上页)

```
                        return false
                }
        }
        return true
}

# 2
func minCut(s string) int {
        if len(s) == 0 || len(s) == 1 {
                return 0
        }
        dp := make([]int, len(s)+1)
        dp[0] = -1
        dp[1] = 1
        arr := getDP(s)
        for i := 1; i <= len(s); i++ {
                dp[i] = i - 1 // 长度N切分n-1次
                for j := 0; j < i; j++ {
                        if arr[j][i-1] == true {
                                dp[i] = min(dp[i], dp[j]+1)
                        }
                }
        }
        return dp[len(s)]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func getDP(s string) [][]bool {
        dp := make([][]bool, len(s))
        for r := 0; r < len(s); r++ {
                dp[r] = make([]bool, len(s))
                dp[r][r] = true
                for l := 0; l < r; l++ {
                        if s[l] == s[r] && (r-l <= 2 || dp[l+1][r-1] == true) {
                                dp[l][r] = true
                        } else {
                                dp[l][r] = false
```

(续下页)

(接上页)

```
                        }
                }
        }
        return dp
}
```

79.8 剑指 OfferII097. 子序列的数目 (2)

- 题目

```
给定一个字符串 s 和一个字符串 t ，计算在 s 的子序列中 t 出现的个数。
字符串的一个 子序列␣
↪是指，通过删除一些（也可以不删除）字符且不干扰剩余字符相对位置所组成的新字符串。
（例如，"ACE"是"ABCDE"的一个子序列，而"AEC"不是）
题目数据保证答案符合 32 位带符号整数范围。
示例1：输入：s = "rabbbit", t = "rabbit" 输出：3
解释：如下图所示，有 3 种可以从 s 中得到 "rabbit" 的方案。
rabbbit
rabbbit
rabbbit
示例2：输入：s = "babgbag", t = "bag" 输出：5
解释：如下图所示，有 5 种可以从 s 中得到 "bag" 的方案。
babgbag
babgbag
babgbag
babgbag
babgbag
提示：0 <= s.length, t.length <= 1000
s 和 t 由英文字母组成
注意：本题与主站 115题相同
```

- 解题思路

```
func numDistinct(s string, t string) int {
        dp := make([]int, len(t)+1)
        dp[0] = 1
        for i := 1; i <= len(s); i++ {
                for j := len(t); j >= 1; j-- {
                        if s[i-1] == t[j-1] {
                                dp[j] = dp[j] + dp[j-1]
                        }
                }
```

(续下页)

(接上页)

```
        }
        return dp[len(t)]
}

# 2
func numDistinct(s string, t string) int {
        // dp[i][j]为使用s的前i个字符能够最多组成多少个t的前j个字符
        dp := make([][]int, len(s)+1)
        for i := 0; i <= len(s); i++ {
                dp[i] = make([]int, len(t)+1)
        }
        for i := 0; i <= len(s); i++ {
                dp[i][0] = 1
        }
        for i := 1; i <= len(s); i++ {
                for j := 1; j <= len(t); j++ {
                        if s[i-1] == t[j-1] {
                                // s用最后一位的 +不用最后一位
                                dp[i][j] = dp[i-1][j-1] + dp[i-1][j]
                        } else {
                                dp[i][j] = dp[i-1][j]
                        }
                }
        }
        return dp[len(s)][len(t)]
}
```

79.9 剑指 OfferII108. 单词演变 (2)

- 题目

```
在字典（单词列表）wordList 中，从单词 beginWord和 endWord 的 转换序列␣
↪是一个按下述规格形成的序列：
序列中第一个单词是 beginWord 。
序列中最后一个单词是 endWord 。
每次转换只能改变一个字母。
转换过程中的中间单词必须是字典wordList 中的单词。
给定两个长度相同但内容不同的单词 beginWord和 endWord 和一个字典 wordList ，
找到从beginWord 到endWord 的 最短转换序列 中的 单词数目␣
↪。如果不存在这样的转换序列，返回 0。
示例 1：输入：beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot",
↪"log","cog"] 输出：5
```

(续下页)

(接上页)

```
解释：一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog", 返回它的长度 5。
示例 2：输入：beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot",
↪"log"] 输出：0
解释：endWord "cog" 不在字典中，所以无法进行转换。
提示：1 <= beginWord.length <= 10
endWord.length == beginWord.length
1 <= wordList.length <= 5000
wordList[i].length == beginWord.length
beginWord、endWord 和 wordList[i] 由小写英文字母组成
beginWord != endWord
wordList 中的所有字符串 互不相同
注意：本题与主站 127题相同：
```

- 解题思路

```
func ladderLength(beginWord string, endWord string, wordList []string) int {
        m := make(map[string]int)
        for i := 0; i < len(wordList); i++ {
                m[wordList[i]] = 1
        }
        if m[endWord] == 0 {
                return 0
        }
        preMap := make(map[string][]string)
        for i := 0; i < len(wordList); i++ {
                for j := 0; j < len(wordList[i]); j++ {
                        newStr := wordList[i][:j] + "*" + wordList[i][j+1:]
                        if _, ok := preMap[newStr]; !ok {
                                preMap[newStr] = make([]string, 0)
                        }
                        preMap[newStr] = append(preMap[newStr], wordList[i])
                }
        }
        visited := make(map[string]bool)
        count := 0
        queue := make([]string, 0)
        queue = append(queue, beginWord)
        for len(queue) > 0 {
                count++
                length := len(queue)
                for i := 0; i < length; i++ {
                        for j := 0; j < len(beginWord); j++ {
                                newStr := queue[i][:j] + "*" + queue[i][j+1:]
                                for _, word := range preMap[newStr] {
```

(续下页)

(接上页)

```
                                        if word == endWord {
                                                return count + 1
                                        }
                                        if visited[word] == false {
                                                visited[word] = true
                                                queue = append(queue, word)
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return 0
}

# 2
func ladderLength(beginWord string, endWord string, wordList []string) int {
        m := make(map[string]int)
        for i := 0; i < len(wordList); i++ {
                m[wordList[i]] = 1
        }
        if m[endWord] == 0 {
                return 0
        }
        queue := make([]string, 0)
        queue = append(queue, beginWord)
        count := 0
        for len(queue) > 0 {
                count++
                length := len(queue)
                for i := 0; i < length; i++ {
                        for _, word := range wordList {
                                diff := 0
                                for j := 0; j < len(queue[i]); j++ {
                                        if queue[i][j] != word[j] {
                                                diff++
                                        }
                                        if diff > 1 {
                                                break
                                        }
                                }
                                if diff == 1 && m[word] != 2 {
                                        if word == endWord {
```

(续下页)

(接上页)

```
                                                return count + 1
                                        }
                                        m[word] = 2
                                        queue = append(queue, word)
                                }
                        }
                }
                queue = queue[length:]
        }
        return 0
}
```

79.10 剑指 OfferII112. 最长递增路径 (3)

- 题目

```
给定一个m x n 整数矩阵matrix ，找出其中 最长递增路径 的长度。
对于每个单元格，你可以往上，下，左，右四个方向移动。 不能 在 对角线␣
↪方向上移动或移动到 边界外（即不允许环绕）。
示例 1：输入：matrix = [[9,9,4],[6,6,8],[2,1,1]] 输出：4
解释：最长递增路径为[1, 2, 6, 9]。
示例 2：输入：matrix = [[3,4,5],[3,2,6],[2,2,1]] 输出：4
解释：最长递增路径是[3, 4, 5, 6]。注意不允许在对角线方向上移动。
示例 3：输入：matrix = [[1]] 输出：1
提示：m == matrix.length
n == matrix[i].length
1 <= m, n <= 200
0 <= matrix[i][j] <= 231 - 1
注意：本题与主站 329题相同：
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var n, m int
var arr [][]int

func longestIncreasingPath(matrix [][]int) int {
        n, m = len(matrix), len(matrix[0])
        arr = make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
```

(续下页)

(接上页)

```
        }
        res := 0
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        res = max(res, dfs(matrix, i, j))
                }
        }
        return res
}

func dfs(matrix [][]int, i, j int) int {
        if arr[i][j] != 0 {
                return arr[i][j]
        }
        arr[i][j]++ // 长度为1
        for k := 0; k < 4; k++ {
                x, y := i+dx[k], j+dy[k]
                if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y] > matrix[i][j] {
                        arr[i][j] = max(arr[i][j], dfs(matrix, x, y)+1)
                }
        }
        return arr[i][j]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func longestIncreasingPath(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
        }
        queue := make([][2]int, 0) // 从最大数开始广度优先搜索
        for i := 0; i < n; i++ {
```

(续下页)

(接上页)

```
                for j := 0; j < m; j++ {
                        for k := 0; k < 4; k++ {
                                x, y := i+dx[k], j+dy[k]
                                if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y]␣
↪> matrix[i][j] {
                                        arr[i][j]++ // 四周大于当前的个数
                                }
                        }
                        if arr[i][j] == 0 { // 四周没有大于当前的数
                                queue = append(queue, [2]int{i, j})
                        }
                }
        }
        res := 0
        for len(queue) > 0 {
                res++
                length := len(queue)
                for i := 0; i < length; i++ {
                        a, b := queue[i][0], queue[i][1]
                        for k := 0; k < 4; k++ {
                                x, y := a+dx[k], b+dy[k]
                                if 0 <= x && x < n && 0 <= y && y < m && matrix[a][b]␣
↪> matrix[x][y] {
                                        arr[x][y]--
                                        if arr[x][y] == 0 { // 个数为0, 加入队列
                                                queue = append(queue, [2]int{x, y})
                                        }
                                }
                        }
                }
                queue = queue[length:]
        }
        return res
}

# 3
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func longestIncreasingPath(matrix [][]int) int {
        n, m := len(matrix), len(matrix[0])
        dp := make([][]int, n)
        temp := make([][3]int, 0)
```

(续下页)

(接上页)

```
        for i := 0; i < n; i++ {
                dp[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        dp[i][j] = 1
                        temp = append(temp, [3]int{i, j, matrix[i][j]})
                }
        }
        sort.Slice(temp, func(i, j int) bool {
                return temp[i][2] < temp[j][2]
        })
        res := 1 // 一个数的时候，没有周围4个数，此时为1
        for i := 0; i < len(temp); i++ {
                a, b := temp[i][0], temp[i][1]
                for k := 0; k < 4; k++ {
                        x, y := a+dx[k], b+dy[k]
                        if 0 <= x && x < n && 0 <= y && y < m && matrix[x][y] >␣
↪matrix[a][b] {
                                dp[x][y] = max(dp[x][y], dp[a][b]+1) // 更新长度
                                res = max(res, dp[x][y])
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

79.11 剑指 OfferII114. 外星文字典 (1)

- 题目

```
现有一种使用英语字母的外星文语言，这门语言的字母顺序与英语顺序不同。
给定一个字符串列表 words ，作为这门语言的词典，words 中的字符串已经␣
↪按这门新语言的字母顺序进行了排序 。
请你根据该词典还原出此语言中已知的字母顺序，并 按字母递增顺序␣
↪排列。若不存在合法字母顺序，返回 "" 。
若存在多种可能的合法字母顺序，返回其中 任意一种 顺序即可。
```

(续下页)

(接上页)

```
字符串 s 字典顺序小于 字符串 t 有两种情况：
在第一个不同字母处，如果 s 中的字母在这门外星语言的字母顺序中位于 t 中字母之前，那么s␣
↪的字典顺序小于 t 。
如果前面 min(s.length, t.length) 字母都相同，那么 s.length < t.length 时，s␣
↪的字典顺序也小于 t 。
示例 1：输入：words = ["wrt","wrf","er","ett","rftt"] 输出："wertf"
示例 2：输入：words = ["z","x"] 输出："zx"
示例 3：输入：words = ["z","x","z"] 输出：""
解释：不存在合法字母顺序，因此返回 "" 。
提示：1 <= words.length <= 100
1 <= words[i].length <= 100
words[i] 仅由小写英文字母组成
注意：本题与主站 269题相同：
```

- 解题思路

```
func alienOrder(words []string) string {
        n := len(words)
        degree := make(map[int]int) // 入度
        arr := [26][]int{}
        for i := 0; i < n; i++ {
                for j := 0; j < len(words[i]); j++ {
                        degree[int(words[i][j]-'a')] = 0
                }
        }
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        length := min(len(words[i]), len(words[j]))
                        for k := 0; k < length; k++ {
                                a, b := int(words[i][k]-'a'), int(words[j][k]-'a')
                                if a == b {
                                        if k == length-1 && len(words[i]) >␣
↪len(words[j]) { // 不合法
                                                return ""
                                        }
                                        continue
                                }
                                arr[a] = append(arr[a], b) // 有序关系：a < b => a=>b
                                degree[b]++
                                break // 已经有有序关系，退出
                        }
                }
        }
        // 拓扑排序
```

(续下页)

(接上页)

```
        res := make([]byte, 0)
        queue := make([]int, 0) // 入度=0 入队
        for i := 0; i < 26; i++ {
                if v, ok := degree[i]; ok && v == 0 {
                        queue = append(queue, i)
                }
        }
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                res = append(res, byte('a'+node))
                for i := 0; i < len(arr[node]); i++ {
                        next := arr[node][i]
                        degree[next]--
                        if degree[next] == 0 {
                                queue = append(queue, next)
                        }
                }
        }
        /*
                // 通过判断长度来判断拓扑排序完成
                if len(res) != len(degree) {
                        return ""
                }
        */
        // 通过判断度数来判断拓扑排序完成
        for i := 0; i < 26; i++ {
                if degree[i] > 0 {
                        return ""
                }
        }
        return string(res)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

79.12 剑指 OfferII117. 相似的字符串 (1)

- 题目

```
如果交换字符串X 中的两个不同位置的字母，使得它和字符串Y 相等，那么称 X 和 Y␣
↪两个字符串相似。
如果这两个字符串本身是相等的，那它们也是相似的。
例如，"tars" 和 "rats" 是相似的 (交换 0 与 2 的位置)；
"rats" 和 "arts" 也是相似的，但是 "star" 不与 "tars"，"rats"，或 "arts" 相似。
总之，它们通过相似性形成了两个关联组：{"tars", "rats", "arts"} 和 {"star"}。
注意，"tars" 和 "arts" 是在同一组中，即使它们并不相似。
形式上，对每个组而言，要确定一个单词在组中，只需要这个词和该组中至少一个单词相似。
给定一个字符串列表 strs。列表中的每个字符串都是 strs␣
↪中其它所有字符串的一个字母异位词。
请问 strs 中有多少个相似字符串组？
字母异位词（anagram），一种把某个字符串的字母的位置（顺序）加以改换所形成的新词。
示例 1：输入：strs = ["tars","rats","arts","star"] 输出：2
示例 2：输入：strs = ["omv","ovm"] 输出：1
提示：1 <= strs.length <= 300
1 <= strs[i].length <= 300
strs[i] 只包含小写字母。
strs 中的所有单词都具有相同的长度，且是彼此的字母异位词。
注意：本题与主站 839题相同：
```

- 解题思路

```
func numSimilarGroups(strs []string) int {
        n := len(strs)
        fa = Init(n)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        if judge(strs[i], strs[j]) == true { // 满足条件，连通
                                union(i, j)
                        }
                }
        }
        return count
}

func judge(a, b string) bool {
        if a == b {
                return true
        }
        count := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(a); i++ {
                if a[i] != b[i] {
                        count++
                        if count > 2 {
                                return false
                        }
                }
        }
        return true
}

var fa []int
var count int

// 初始化
func Init(n int) []int {
        arr := make([]int, n)
        for i := 0; i < n; i++ {
                arr[i] = i
        }
        count = n
        return arr
}

// 查询
func find(x int) int {
        if fa[x] == x {
                return x
        }
        // 路径压缩
        fa[x] = find(fa[x])
        return fa[x]
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x != y {
                fa[x] = y
                count--
        }
}
```

CHAPTER 80

程序员面试金典

80.1 面试题 01.01. 判定字符是否唯一 (5)

- 题目

```
实现一个算法，确定一个字符串 s 的所有字符是否全都不同。
示例 1: 输入: s = "leetcode" 输出: false
示例 2: 输入: s = "abc" 输出: true
限制:
    0 <= len(s) <= 100
    如果你不使用额外的数据结构，会很加分。
```

- 解题思路

```
func isUnique(astr string) bool {
        m := make(map[byte]bool)
        for i := 0; i < len(astr); i++ {
                if m[astr[i]] == true {
                        return false
                }
                m[astr[i]] = true
        }
        return true
}
```

(续下页)

(接上页)

```
# 2
func isUnique(astr string) bool {
        value := uint32(0)
        for i := 0; i < len(astr); i++ {
                index := astr[i] - 'a'
                if value&(1<<index) == (1 << index) {
                        return false
                }
                value = value ^ (1 << index)
        }
        return true
}

# 3
func isUnique(astr string) bool {
        for i := 0; i < len(astr); i++ {
                for j := i + 1; j < len(astr); j++ {
                        if astr[i] == astr[j] {
                                return false
                        }
                }
        }
        return true
}

# 4
func isUnique(astr string) bool {
        arr := []byte(astr)
        sort.Slice(arr, func(i, j int) bool {
                return arr[i] < arr[j]
        })
        for i := 1; i < len(arr); i++ {
                if arr[i] == arr[i-1] {
                        return false
                }
        }
        return true
}

# 5
func isUnique(astr string) bool {
        arr := make([]int, 256)
        for i := 0; i < len(astr); i++ {
```

(续下页)

(接上页)

```
                if arr[astr[i]] > 0 {
                        return false
                }
                arr[astr[i]] = 1
        }
        return true
}
```

80.2 面试题 01.02. 判定是否互为字符重排 (2)

- 题目

```
给定两个字符串 s1 和␣
↪s2，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。
示例 1: 输入: s1 = "abc", s2 = "bca" 输出: true
示例 2: 输入: s1 = "abc", s2 = "bad" 输出: false
说明:
    0 <= len(s1) <= 100
    0 <= len(s2) <= 100
```

- 解题思路

```
func CheckPermutation(s1 string, s2 string) bool {
        arr1 := strings.Split(s1, "")
        arr2 := strings.Split(s2, "")
        sort.Strings(arr1)
        sort.Strings(arr2)
        return strings.Join(arr1,"") == strings.Join(arr2,"")
        // return reflect.DeepEqual(arr1, arr2)
}

#
func CheckPermutation(s1 string, s2 string) bool {
        if len(s1) != len(s2) {
                return false
        }
        m := make(map[byte]int)
        for i := 0; i < len(s1); i++ {
                m[s1[i]]++
                m[s2[i]]--
        }
        for _, v := range m {
```

(续下页)

(接上页)

```
                if v != 0 {
                        return false
                }
        }
        return true
}

#
func CheckPermutation(s1 string, s2 string) bool {
        if len(s1) != len(s2) {
                return false
        }
        arr := [256]int{}
        for i := 0; i < len(s1); i++ {
                arr[s1[i]]++
                arr[s2[i]]--
        }
        for _, v := range arr {
                if v != 0 {
                        return false
                }
        }
        return true
}
```

80.3 面试题 01.03.URL 化 (2)

- 题目

```
URL化。编写一种方法，将字符串中的空格全部替换为
↪%20。假定该字符串尾部有足够的空间存放新增字符，
并且知道字符串的“真实”长度。（注：用Java实现的话，请使用字符数组实现，以便直接在数组上操作。）
示例1:输入："Mr John Smith    ", 13 输出："Mr%20John%20Smith"
示例2:输入："               ", 5 输出："%20%20%20%20%20"
提示：
    字符串长度在[0, 500000]范围内。
```

- 解题思路

```
func replaceSpaces(S string, length int) string {
        return strings.ReplaceAll(S[:length], " ","%20")
}
```

(续下页)

(接上页)

```
#
func replaceSpaces(S string, length int) string {
        res := make([]byte,0)
        for i := 0; i < length; i++ {
                if S[i] == ' ' {
                        res = append(res,'%')
                        res = append(res,'2')
                        res = append(res,'0')
                } else {
                        res = append(res,S[i])
                }
        }
        return string(res)
}
```

80.4 面试题 01.04. 回文排列 (2)

- 题目

```
给定一个字符串，编写一个函数判定其是否为某个回文串的排列之一。
回文串是指正反两个方向都一样的单词或短语。排列是指字母的重新排列。
回文串不一定是字典当中的单词。
示例1：输入："tactcoa" 输出：true（排列有"tacocat"、"atcocta"，等等）
```

- 解题思路

```
func canPermutePalindrome(s string) bool {
        m := make(map[byte]int)
        for i := 0; i < len(s); i++ {
                m[s[i]]++
                if m[s[i]] == 2 {
                        delete(m, s[i])
                }
        }
        return len(m) <= 1
}

#
func canPermutePalindrome(s string) bool {
        arr := [256]int{}
        for i := 0; i < len(s); i++ {
```

(续下页)

(接上页)

```
                arr[s[i]]++
        }
        count := 0
        for i := 0; i < len(arr); i++{
                if arr[i] % 2== 1{
                        count++
                }
        }
        return count <= 1
}
```

80.5 面试题 01.05. 一次编辑 (2)

- 题目

```
字符串有三种编辑操作:插入一个字符、删除一个字符或者替换一个字符。
给定两个字符串，编写一个函数判定它们是否只需要一次(或者零次)编辑。
示例 1:输入: first = "pale"second = "ple" 输出: True
示例 2:输入: first = "pales"second = "pal" 输出: False
```

- 解题思路

```
func oneEditAway(first string, second string) bool {
        if len(first)-len(second) > 1 || len(second)-len(first) > 1 {
                return false
        }
        if first == second {
                return true
        }
        i := 0
        for ; i < len(first) && i < len(second); i++ {
                if first[i] != second[i] {
                        if len(first) == len(second) {
                                if first[i+1:] == second[i+1:] {
                                        return true
                                }
                        } else if len(first) < len(second) {
                                if first[i:] == second[i+1:] {
                                        return true
                                }
                        } else {
                                if first[i+1:] == second[i:] {
```

(续下页)

(接上页)

```
                                        return true
                                }
                        }
                        break
                }
        }
        if i == len(first) || i == len(second) {
                return true
        }
        return false
}

#
func oneEditAway(first string, second string) bool {
        if len(first)-len(second) > 1 || len(second)-len(first) > 1 {
                return false
        }
        if first == second {
                return true
        }
        if len(first) > len(second) {
                first, second = second, first
        }
        for i := 0; i < len(first); i++ {
                if first[i] == second[i] {
                        continue
                }
                return first[i:] == second[i+1:] || first[i+1:] == second[i+1:]
        }
        return true
}
```

80.6 面试题 01.06. 字符串压缩 (2)

- 题目

```
字符串压缩。利用字符重复出现的次数，编写一种方法，实现基本的字符串压缩功能。
比如，字符串aabcccccaaa会变为a2b1c5a3。若“压缩”后的字符串没有变短，则返回原先的字符串。
你可以假设字符串中只包含大小写英文字母（a至z）。
示例1:输入："aabcccccaaa" 输出："a2b1c5a3"
示例2:输入："abbccd" 输出："abbccd"
解释："abbccd"压缩后为"a1b2c2d1"，比原字符串长度更长。
```

(续下页)

(接上页)

提示：字符串长度在[0, 50000]范围内。

- 解题思路

```
func compressString(S string) string {
        if len(S) <= 1 {
                return S
        }
        prev := S[0]
        count := 1
        res := ""
        for i := 1; i < len(S); i++ {
                if prev == S[i] {
                        count++
                } else {
                        res = res + string(prev) + strconv.Itoa(count)
                        prev = S[i]
                        count = 1
                }
        }
        res = res + string(prev) + strconv.Itoa(count)
        if len(res) >= len(S) {
                return S
        }
        return res
}

#
func compressString(S string) string {
        if len(S) <= 1 {
                return S
        }
        i := 0
        j := 0
        res := ""
        for j = 1; j < len(S); j++ {
                if S[i] != S[j] {
                        res = res + string(S[i]) + strconv.Itoa(j-i)
                        i = j
                }
        }
        res = res + string(S[i]) + strconv.Itoa(j-i)
        if len(res) >= len(S) {
                return S
```

(续下页)

(接上页)

```
        }
        return res
}
```

80.7 面试题 01.07. 旋转矩阵 (3)

- 题目

```
给你一幅由 N × N 矩阵表示的图像，其中每个像素的大小为 4␣
↪字节。请你设计一种算法，将图像旋转 90 度。
不占用额外内存空间能否做到？
示例 1:给定 matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],
原地旋转输入矩阵，使其变为:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
示例 2:给定 matrix =
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],
原地旋转输入矩阵，使其变为:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

- 解题思路

```
func rotate(matrix [][]int) {
        n := len(matrix)
```

(续下页)

(接上页)

```
        // 同行逆置
        // [[1 2 3] [4 5 6] [7 8 9]]
        // [[3 2 1] [6 5 4] [9 8 7]]
        for i := 0; i < n; i++ {
                for j := 0; j < n/2; j++ {
                        matrix[i][j], matrix[i][n-1-j] = matrix[i][n-1-j],␣
↪matrix[i][j]
                }
        }
        // 左下右上对角线对互换
        // [[3 2 1] [6 5 4] [9 8 7]]
        // [[7 4 1] [8 5 2] [9 6 3]]
        for i := 0; i < n-1; i++ {
                for j := 0; j < n-1-i; j++ {
                        matrix[i][j], matrix[n-1-j][n-1-i] = matrix[n-1-j][n-1-i],␣
↪matrix[i][j]
                }
        }
}

# 2
func rotate(matrix [][]int) {
        n := len(matrix)
        for start, end := 0, n-1; start < end; {
                for s, e := start, end; s < end; {
                        matrix[start][s], matrix[e][start], matrix[end][e],␣
↪matrix[s][end] =
                                matrix[e][start], matrix[end][e], matrix[s][end],␣
↪matrix[start][s]
                        s++
                        e--
                }
                start++
                end--
        }
}

# 3
func rotate(matrix [][]int) {
        n := len(matrix)
        arr := make([][]int, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]int, n)
```

(续下页)

(接上页)

```
        }
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        arr[j][n-1-i] = matrix[i][j]
                }
        }
        copy(matrix, arr)
}
```

80.8 面试题 01.08. 零矩阵 (4)

- 题目

```
编写一种算法，若M × N矩阵中某个元素为0，则将其所在的行与列清零。
示例 1: 输入:
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
输出:
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
示例 2: 输入:
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
输出:
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
```

- 解题思路

```
func setZeroes(matrix [][]int) {
        x := make(map[int]int)
        y := make(map[int]int)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                x[i] = 1
                                y[j] = 1
                        }
                }
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if x[i] == 1 || y[j] == 1 {
                                matrix[i][j] = 0
                        }
                }
        }
}

# 2
func setZeroes(matrix [][]int) {
        m := make(map[[2]int]bool)
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == math.MinInt32 {
                                m[[2]int{i, j}] = true
                        }
                }
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                for k := 0; k < len(matrix); k++ {
                                        for l := 0; l < len(matrix[k]); l++ {
                                                if (k == i || l == j) && matrix[k][l]␣
↪!= 0 {
                                                        delete(m, [2]int{k, l})
                                                        matrix[k][l] = math.MinInt32
                                                }
                                        }
                                }
                        }
                }
```

(续下页)

(接上页)

```
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == math.MinInt32 && m[[2]int{i, j}] == false {
                                matrix[i][j] = 0
                        }
                }
        }
}

# 3
func setZeroes(matrix [][]int) {
        flag := false
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] == 0 {
                        flag = true
                }
                for j := 1; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                matrix[i][0] = 0
                                matrix[0][j] = 0
                        }
                }
        }
        for i := 1; i < len(matrix); i++ {
                for j := 1; j < len(matrix[i]); j++ {
                        if matrix[i][0] == 0 || matrix[0][j] == 0 {
                                matrix[i][j] = 0
                        }
                }
        }
        // 第一行处理
        if matrix[0][0] == 0 {
                for j := 0; j < len(matrix[0]); j++ {
                        matrix[0][j] = 0
                }
        }
        // 第一列处理
        if flag == true {
                for i := 0; i < len(matrix); i++ {
                        matrix[i][0] = 0
                }
        }
```

(续下页)

(接上页)

```
}

# 4
func setZeroes(matrix [][]int) {
        flag := false
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] == 0 {
                        flag = true
                }
                for j := 1; j < len(matrix[i]); j++ {
                        if matrix[i][j] == 0 {
                                matrix[i][0] = 0
                                matrix[0][j] = 0
                        }
                }
        }
        for i := len(matrix) - 1; i >= 0; i-- {
                for j := len(matrix[i]) - 1; j >= 1; j-- {
                        if matrix[i][0] == 0 || matrix[0][j] == 0 {
                                matrix[i][j] = 0
                        }
                }
        }
        // 第一列处理
        if flag == true {
                for i := 0; i < len(matrix); i++ {
                        matrix[i][0] = 0
                }
        }
}
```

80.9 面试题 01.09. 字符串轮转 (2)

- 题目

```
字符串轮转。给定两个字符串s1和s2，请编写代码检查s2是否为s1旋转而成（
比如，waterbottle是erbottlewat旋转后的字符串）。
示例1: 输入：s1 = "waterbottle", s2 = "erbottlewat" 输出：True
示例2:输入：s1 = "aa", s2 = "aba" 输出：False
提示：字符串长度在[0, 100000]范围内。
说明：你能只调用一次检查子串的方法吗？
```

- 解题思路

```
func isFlipedString(s1 string, s2 string) bool {
        if len(s1) != len(s2){
                return false
        }
        return strings.Contains(s1+s1, s2)
}

#
func isFlipedString(s1 string, s2 string) bool {
        if s1 == s2 {
                return true
        }
        if len(s1) != len(s2) {
                return false
        }
        for i := 0; i < len(s1); i++ {
                s1 = s1[1:] + string(s1[0])
                if s1 == s2 {
                        return true
                }
        }
        return false
}
```

80.10 面试题 02.01. 移除重复节点 (3)

- 题目

```
编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。
示例1:输入：[1, 2, 3, 3, 2, 1]输出：[1, 2, 3]
示例2:输入：[1, 1, 1, 1, 2]输出：[1, 2]
提示：
    链表长度在[0, 20000]范围内。
    链表元素在[0, 20000]范围内。
进阶：如果不得使用临时缓冲区，该怎么解决？
```

- 解题思路

```
func removeDuplicateNodes(head *ListNode) *ListNode {
        if head == nil {
                return head
        }
        m := make(map[int]bool)
```

(续下页)

(接上页)

```
        m[head.Val] = true
        temp := head
        for temp.Next != nil {
                if m[temp.Next.Val] == true {
                        temp.Next = temp.Next.Next
                } else {
                        m[temp.Next.Val] = true
                        temp = temp.Next
                }
        }
        return head
}

# 2
func removeDuplicateNodes(head *ListNode) *ListNode {
        if head == nil {
                return head
        }
        temp := head
        for temp != nil {
                second := temp
                for second.Next != nil {
                        if second.Next.Val == temp.Val {
                                second.Next = second.Next.Next
                        } else {
                                second = second.Next
                        }
                }
                temp = temp.Next
        }
        return head
}

# 3
var m map[int]bool

func removeDuplicateNodes(head *ListNode) *ListNode {
        m = make(map[int]bool)
        return remove(head)

}

func remove(head *ListNode) *ListNode {
```

(续下页)

(接上页)

```
        if head == nil {
                return head
        }
        if m[head.Val] == true {
                return remove(head.Next)
        }
        m[head.Val] = true
        head.Next = remove(head.Next)
        return head
}
```

80.11 面试题 02.02. 返回倒数第 k 个节点 (4)

- 题目

```
实现一种算法，找出单向链表中倒数第 k 个节点。返回该节点的值。
注意：本题相对原题稍作改动
示例：输入： 1->2->3->4->5 和 k = 2 输出： 4
说明：给定的 k 保证是有效的。
```

- 解题思路

```
func kthToLast(head *ListNode, k int) int {
        arr := make([]*ListNode, 0)
        for head != nil {
                arr = append(arr, head)
                head = head.Next
        }
        if len(arr) >= k {
                return arr[len(arr)-k].Val
        }
        return -1
}

# 2
func kthToLast(head *ListNode, k int) int {
        fast := head
        for k > 0 && head != nil {
                fast = fast.Next
                k--
        }
        if k > 0 {
```

(续下页)

(接上页)

```
                return -1
        }
        slow := head
        for fast != nil {
                fast = fast.Next
                slow = slow.Next
        }
        return slow.Val
}

# 3
func kthToLast(head *ListNode, k int) int {
        temp := head
        count := 0
        for temp != nil {
                count++
                temp = temp.Next
        }
        if count < k {
                return -1
        }
        for i := 0; i < count-k; i++ {
                head = head.Next
        }
        return head.Val
}

# 4
func kthToLast(head *ListNode, k int) int {
        res, count := dfs(head, k)
        if count > 0 {
                return -1
        }
        return res.Val
}

func dfs(node *ListNode, k int) (*ListNode, int) {
        if node == nil {
                return node, k
        }
        next, nextValue := dfs(node.Next, k)
        if nextValue <= 0 {
                return next, nextValue
```

(续下页)

(接上页)

```
        }
        nextValue = nextValue - 1
        return node, nextValue
}
```

80.12 面试题 02.03. 删除中间节点 (1)

- 题目

```
实现一种算法，删除单向链表中间的某个节点（即不是第一个或最后一个节点），假定你只能访问该节点。
示例：输入：单向链表a->b->c->d->e->f中的节点c
结果：不返回任何数据，但该链表变为a->b->d->e->f
```

- 解题思路

```
func deleteNode(node *ListNode) {
        // *node = *node.Next
        node.Val = node.Next.Val
        node.Next = node.Next.Next
}
```

80.13 面试题 02.04. 分割链表 (2)

- 题目

```
编写程序以 x 为基准分割链表，使得所有小于 x 的节点排在大于或等于 x 的节点之前。
如果链表中包含 x，x 只需出现在小于 x 的元素之后(如下所示)。
分割元素 x 只需处于“右半部分”即可，其不需要被置于左右两部分之间。
示例:输入：head = 3->5->8->5->10->2->1, x = 5
输出：3->1->2->10->5->5->8
```

- 解题思路

```
func partition(head *ListNode, x int) *ListNode {
        first := &ListNode{}
        second := &ListNode{}
        a := first
        b := second
        for head != nil {
                if head.Val < x {
```

(续下页)

(接上页)

```
                        a.Next = head
                        a = head
                } else {
                        b.Next = head
                        b = head
                }
                head = head.Next
        }
        b.Next = nil
        a.Next = second.Next
        return first.Next
}

# 2
func partition(head *ListNode, x int) *ListNode {
        a := make([]*ListNode, 0)
        b := make([]*ListNode, 0)

        for head != nil {
                if head.Val < x {
                        a = append(a, head)
                } else {
                        b = append(b, head)
                }
                head = head.Next
        }
        temp := &ListNode{}
        node := temp
        for i := 0; i < len(a); i++ {
                node.Next = a[i]
                node = node.Next
        }
        for i := 0; i < len(b); i++ {
                node.Next = b[i]
                node = node.Next
        }
        node.Next = nil
        return temp.Next
}
```

80.14 面试题 02.05. 链表求和 (2)

- 题目

```
给定两个用链表表示的整数，每个节点包含一个数位。
这些数位是反向存放的，也就是个位排在链表首部。
编写函数对这两个整数求和，并用链表形式返回结果。
示例：输入：(7 -> 1 -> 6) + (5 -> 9 -> 2)，即617 + 295 输出：2 -> 1 -> 9，即912
进阶：假设这些数位是正向存放的，请再做一遍。
示例：输入：(6 -> 1 -> 7) + (2 -> 9 -> 5)，即617 + 295 输出：9 -> 1 -> 2，即912
```

- 解题思路

```
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        res := &ListNode{}
        cur := res
        carry := 0
        for l1 != nil || l2 != nil || carry > 0 {
                sum := carry
                if l1 != nil {
                        sum += l1.Val
                        l1 = l1.Next
                }
                if l2 != nil {
                        sum += l2.Val
                        l2 = l2.Next
                }
                carry = sum / 10 // 进位
                cur.Next = &ListNode{Val: sum % 10}
                cur = cur.Next
        }
        return res.Next
}

# 2
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
        if l1 == nil && l2 == nil {
                return nil
        }
        if l1 == nil {
                return l2
        }
        if l2 == nil {
                return l1
```

(续下页)

(接上页)

```
        }
        sum := l1.Val + l2.Val
        res := &ListNode{Val: sum % 10}
        if sum >= 10 {
                l1.Next = addTwoNumbers(l1.Next, &ListNode{Val: 1})
        }
        res.Next = addTwoNumbers(l1.Next, l2.Next)
        return res
}
```

80.15 面试题 02.06. 回文链表 (4)

- 题目

```
编写一个函数，检查输入的链表是否是回文的。
示例 1: 输入: 1->2 输出: false
示例 2: 输入: 1->2->2->1 输出: true
进阶：你能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题？
```

- 解题思路

```
func isPalindrome(head *ListNode) bool {
        m := make([]int, 0)
        for head != nil {
                m = append(m, head.Val)
                head = head.Next
        }
        i, j := 0, len(m)-1
        for i < j {
                if m[i] != m[j] {
                        return false
                }
                i++
                j--
        }
        return true
}

# 2
func isPalindrome(head *ListNode) bool {
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
```

(续下页)

(接上页)

```
                fast = fast.Next.Next
                slow = slow.Next
        }
        var pre *ListNode
        cur := slow
        for cur != nil{
                next := cur.Next
                cur.Next = pre
                pre = cur
                cur = next
        }
        for pre != nil{
                if head.Val != pre.Val{
                        return false
                }
                pre = pre.Next
                head = head.Next
        }
        return true
}

# 3
func isPalindrome(head *ListNode) bool {
        m := make([]int, 0)
        temp := head
        for temp != nil {
                m = append(m, temp.Val)
                temp = temp.Next
        }
        for head != nil {
                val := m[len(m)-1]
                m = m[:len(m)-1]
                if head.Val != val {
                        return false
                }
                head = head.Next
        }
        return true
}

# 4
var p *ListNode
func isPalindrome(head *ListNode) bool {
```

(续下页)

(接上页)

```
        if head == nil{
                return true
        }
        if p == nil{
                p = head
        }
        if isPalindrome(head.Next) && (p.Val == head.Val){
                p = p.Next
                return true
        }
        p = nil
        return false
}
```

80.16 面试题 02.07. 链表相交 (4)

- 题目

```
给定两个（单向）链表，判定它们是否相交并返回交点。请注意相交的定义基于节点的引用，而不是基于节点的值
换句话说，如果一个链表的第k个节点与另一个链表的第j个节点是同一节点（引用完全相同），则这两个链表相交
示例 1：
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB␣
↪= 3
输出：Reference of the node with value = 8
输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为␣
↪0）。从各自的表头开始算起，
链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
示例 2：输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3,␣
↪skipB = 1
输出：Reference of the node with value = 2
输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。
在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。
示例 3：输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
输出：null
输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。
由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
解释：这两个链表不相交，因此返回 null。
注意：
    如果两个链表没有交点，返回 null 。
    在返回结果后，两个链表仍须保持原有的结构。
```

(续下页)

(接上页)

可假定整个链表结构中没有循环。
程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。

- 解题思路

```
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        ALength := 0
        A := headA
        for A != nil {
                ALength++
                A = A.Next
        }
        BLength := 0
        B := headB
        for B != nil {
                BLength++
                B = B.Next
        }

        pA := headA
        pB := headB
        if ALength > BLength {
                n := ALength - BLength
                for n > 0 {
                        pA = pA.Next
                        n--
                }
        } else {
                n := BLength - ALength
                for n > 0 {
                        pB = pB.Next
                        n--
                }
        }

        for pA != pB {
                pA = pA.Next
                pB = pB.Next
        }
        return pA
}

# 2
func getIntersectionNode(headA, headB *ListNode) *ListNode {
```

(续下页)

(接上页)

```
        A, B := headA, headB
        for A != B {
                if A != nil {
                        A = A.Next
                } else {
                        A = headB
                }
                if B != nil {
                        B = B.Next
                } else {
                        B = headA
                }
        }
        return A
}

# 3
func getIntersectionNode(headA, headB *ListNode) *ListNode {
    A, B := headA, headB
    for A != nil {
        for B != nil {
            if A == B {
                return A
            }
            B = B.Next
        }
        A = A.Next
        B = headB
    }
    return nil
}

# 4
func getIntersectionNode(headA, headB *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for headA != nil {
                m[headA] = true
                headA = headA.Next
        }

        for headB != nil {
                if _, ok := m[headB]; ok {
                        return headB
```

(续下页)

(接上页)

```
                }
                headB = headB.Next
        }
        return nil
}
```

80.17 面试题 02.08. 环路检测 (3)

- 题目

```
给定一个链表，如果它是有环链表，实现一个算法返回环路的开头节点。
有环链表的定义：在链表中某个节点的next元素指向在它前面出现过的节点，则表明该链表存在环路。
示例 1：输入：head = [3,2,0,-4], pos = 1 输出：tail connects to node index 1
解释：链表中有一个环，其尾部连接到第二个节点。
示例 2：输入：head = [1,2], pos = 0 输出：tail connects to node index 0
解释：链表中有一个环，其尾部连接到第一个节点。
示例 3：输入：head = [1], pos = -1 输出：no cycle
解释：链表中没有环。
进阶：你是否可以不用额外空间解决此题？
```

- 解题思路

```
func detectCycle(head *ListNode) *ListNode {
        m := make(map[*ListNode]bool)
        for head != nil {
                if m[head] {
                        return head
                }
                m[head] = true
                head = head.Next
        }
        return nil
}

# 2
func detectCycle(head *ListNode) *ListNode {
        if head == nil {
                return nil
        }
        fast, slow := head, head
        for fast != nil && fast.Next != nil {
                fast = fast.Next.Next
```

(续下页)

(接上页)

```
                slow = slow.Next
                if fast == slow {
                        break
                }
        }
        if fast == nil || fast.Next == nil {
                return nil
        }
        slow = head
        for fast != slow {
                fast = fast.Next
                slow = slow.Next
        }
        return slow
}

# 3
func detectCycle(head *ListNode) *ListNode {
        for head != nil {
                if head.Val == math.MaxInt32 {
                        return head
                }
                head.Val = math.MaxInt32
                head = head.Next
        }
        return head
}
```

80.18 面试题 03.01. 三合一 (1)

- 题目

```
三合一。描述如何只用一个数组来实现三个栈。
你应该实现push(stackNum,␣
↪value)、pop(stackNum)、isEmpty(stackNum)、peek(stackNum)方法。stackNum表示栈下标，value表示压入的值
构造函数会传入一个stackSize参数，代表每个栈的大小。
示例1:输入：["TripleInOne", "push", "push", "pop", "pop", "pop", "isEmpty"]
[[1], [0, 1], [0, 2], [0], [0], [0], [0]]
 输出：[null, null, null, 1, -1, -1, true]
说明：当栈为空时`pop, peek`返回-1，当栈满时`push`不压入元素。
示例2:输入： ["TripleInOne", "push", "push", "push", "pop", "pop", "pop", "peek"]
[[2], [0, 1], [0, 2], [0, 3], [0], [0], [0], [0]]
```

(续下页)

(接上页)

```
输出：[null, null, null, null, 2, 1, -1, -1]
```

- 解题思路

```
type TripleInOne struct {
        arr    []int
        length int
        index  [3]int
}

func Constructor(stackSize int) TripleInOne {
        return TripleInOne{
                arr:    make([]int, stackSize*3),
                length: stackSize,
                index:  [3]int{0, 0, 0},
        }
}

func (this *TripleInOne) Push(stackNum int, value int) {
        if this.index[stackNum] < this.length {
                this.arr[3*this.index[stackNum]+stackNum] = value
                this.index[stackNum]++
        }
}

func (this *TripleInOne) Pop(stackNum int) int {
        res := -1
        if this.index[stackNum] != 0 {
                this.index[stackNum]--
                res = this.arr[3*this.index[stackNum]+stackNum]
        }
        return res
}

func (this *TripleInOne) Peek(stackNum int) int {
        res := -1
        if this.index[stackNum] != 0 {
                res = this.arr[3*(this.index[stackNum]-1)+stackNum]
        }
        return res
}

func (this *TripleInOne) IsEmpty(stackNum int) bool {
        if this.index[stackNum] == 0 {
```

(续下页)

(接上页)

```
                return true
        }
        return false
}
```

80.19 面试题 03.02. 栈的最小值 (2)

- 题目

```
请设计一个栈，除了常规栈支持的pop与push函数以外，还支持min函数，该函数返回栈元素中的最小值。
执行push、pop和min操作的时间复杂度必须为O(1)。
示例：MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

- 解题思路

```
type item struct {
        min, x int
}

type MinStack struct {
        stack []item
}

func Constructor() MinStack {
        return MinStack{}
}

func (this *MinStack) Push(x int) {
        min := x
        if len(this.stack) > 0 && this.GetMin() < x {
                min = this.GetMin()
        }
        this.stack = append(this.stack, item{
                min: min,
                x:   x,
```

(续下页)

(接上页)

```
        })
}

func (this *MinStack) Pop() {
        this.stack = this.stack[:len(this.stack)-1]
}

func (this *MinStack) Top() int {
        if len(this.stack) == 0 {
                return 0
        }
        return this.stack[len(this.stack)-1].x
}

func (this *MinStack) GetMin() int {
        if len(this.stack) == 0 {
                return 0
        }
        return this.stack[len(this.stack)-1].min
}

# 2
type MinStack struct {
        data []int
        min  []int
}

func Constructor() MinStack {
        return MinStack{[]int{}, []int{}}
}

func (this *MinStack) Push(x int) {
        if len(this.data) == 0 || x <= this.GetMin() {
                this.min = append(this.min, x)
        }
        this.data = append(this.data, x)
}

func (this *MinStack) Pop() {
        x := this.data[len(this.data)-1]
        this.data = this.data[:len(this.data)-1]
        if x == this.GetMin() {
                this.min = this.min[:len(this.min)-1]
```

(续下页)

(接上页)

```
        }
}

func (this *MinStack) Top() int {
        if len(this.data) == 0 {
                return 0
        }
        return this.data[len(this.data)-1]
}

func (this *MinStack) GetMin() int {
        return this.min[len(this.min)-1]
}
```

80.20 面试题 03.03. 堆盘子 (1)

- 题目

```
堆盘子。设想有一堆盘子，堆太高可能会倒下来。因此，在现实生活中，盘子堆到一定高度时，我们就会另外堆一
请实现数据结构SetOfStacks，模拟这种行为。SetOfStacks应该由多个栈组成，并且在前一个栈填满时新建一个栈
此外，SetOfStacks.push()和SetOfStacks.pop()应该与普通栈的操作方法相同
（也就是说，pop()返回的值，应该跟只有一个栈时的情况一样）。
进阶：实现一个popAt(int index)方法，根据指定的子栈，执行pop操作。
当某个栈为空时，应当删除该栈。当栈中没有元素或不存在该栈时，pop，popAt 应返回 -1.
示例1: 输入：["StackOfPlates", "push", "push", "popAt", "pop", "pop"]
[[1], [1], [2], [1], [], []]
 输出：[null, null, null, 2, 1, -1]
示例2:输入： ["StackOfPlates", "push", "push", "push", "popAt", "popAt", "popAt"]
[[2], [1], [2], [3], [0], [0], [0]]
 输出：[null, null, null, null, 2, 1, 3]
```

- 解题思路

```
type StackOfPlates struct {
        cap   int
        stack [][]int
}

func Constructor(cap int) StackOfPlates {
        return StackOfPlates{
                cap:   cap,
                stack: make([][]int, 0),
```

(续下页)

(接上页)

```
        }
}

func (this *StackOfPlates) Push(val int) {
        if this.cap == 0 {
                return
        }
        if len(this.stack) == 0 {
                newStack := make([]int, 0)
                newStack = append(newStack, val)
                this.stack = append(this.stack, newStack)
                return
        }
        last := this.stack[len(this.stack)-1]
        if len(last) == this.cap {
                newStack := make([]int, 0)
                newStack = append(newStack, val)
                this.stack = append(this.stack, newStack)
                return
        }
        last = append(last, val)
        this.stack[len(this.stack)-1] = last
}

func (this *StackOfPlates) Pop() int {
        if len(this.stack) == 0 {
                return -1
        }
        last := this.stack[len(this.stack)-1]
        res := last[len(last)-1]
        last = last[:len(last)-1]
        this.stack[len(this.stack)-1] = last
        if len(last) == 0 {
                this.stack = this.stack[:len(this.stack)-1]
        }
        return res
}

func (this *StackOfPlates) PopAt(index int) int {
        if index >= len(this.stack) {
                return -1
        }
        arr := this.stack[index]
```

(续下页)

(接上页)

```
        res := arr[len(arr)-1]
        arr = arr[:len(arr)-1]
        this.stack[index] = arr
        if len(arr) == 0 {
                this.stack = append(this.stack[:index], this.stack[index+1:]...)
        }
        return res
}
```

80.21 面试题 03.04. 化栈为队 (3)

- 题目

```
实现一个MyQueue类，该类用两个栈来实现一个队列。
示例: MyQueue queue = new MyQueue();
queue.push(1);
queue.push(2);
queue.peek();  // 返回 1
queue.pop();   // 返回 1
queue.empty(); // 返回 false
说明:  你只能使用标准的栈操作 -- 也就是只有 push to top, peek/pop from top, size
和 is empty 操作是合法的。
    你所使用的语言也许不支持栈。
    你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。
    假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。
```

- 解题思路

```
type MyQueue struct {
        a []int
}

func Constructor() MyQueue {
        return MyQueue{}
}

func (m *MyQueue) Push(x int) {
        m.a = append(m.a, x)
}

func (m *MyQueue) Pop() int {
        if len(m.a) == 0 {
```

(续下页)

(接上页)

```
                return 0
        }
        first := m.a[0]
        m.a = m.a[1:]
        return first
}

func (m *MyQueue) Peek() int {
        if len(m.a) == 0 {
                return 0
        }
        return m.a[0]
}

func (m *MyQueue) Empty() bool {
        if len(m.a) == 0 {
                return true
        }
        return false
}

# 2
/*
入队：直接入栈a
出队：栈b为空，则把栈a中全部数据出栈进入栈b，然后出栈b,不为空直接出栈b
*/
type MyQueue struct {
        a, b *Stack
}

func Constructor() MyQueue {
        return MyQueue{
                a: NewStack(),
                b: NewStack(),
        }
}

func (m *MyQueue) Push(x int) {
        m.a.Push(x)
}

func (m *MyQueue) Pop() int {
        if m.b.Len() == 0 {
```

(续下页)

(接上页)

```
                for m.a.Len() > 0 {
                        m.b.Push(m.a.Pop())
                }
        }
        return m.b.Pop()
}

func (m *MyQueue) Peek() int {
        res := m.Pop()
        m.b.Push(res)
        return res
}

func (m *MyQueue) Empty() bool {
        return m.a.Len() == 0 && m.b.Len() == 0
}

type Stack struct {
        nums []int
}

func NewStack() *Stack {
        return &Stack{
                nums: []int{},
        }
}

func (s *Stack) Push(n int) {
        s.nums = append(s.nums, n)
}

func (s *Stack) Pop() int {
        res := s.nums[len(s.nums)-1]
        s.nums = s.nums[:len(s.nums)-1]
        return res
}

func (s *Stack) Len() int {
        return len(s.nums)
}

func (s *Stack) IsEmpty() bool {
        return s.Len() == 0
```

(续下页)

(接上页)

```
}

# 3
type MyQueue struct {
        a []int
        b []int
}

func Constructor() MyQueue {
        return MyQueue{}
}

func (m *MyQueue) Push(x int) {
        m.a = append(m.a, x)
}

func (m *MyQueue) Pop() int {
        m.Peek()
        temp := m.b[len(m.b)-1]
        m.b = m.b[:len(m.b)-1]
        return temp
}

func (m *MyQueue) Peek() int {
        if len(m.b) == 0 {
                for len(m.a) > 0 {
                        m.b = append(m.b, m.a[len(m.a)-1])
                        m.a = m.a[:len(m.a)-1]
                }
        }
        if len(m.b) == 0 {
                return -1
        }
        return m.b[len(m.b)-1]
}

func (m *MyQueue) Empty() bool {
        return len(m.a) == 0 && len(m.b) == 0
}
```

80.22 面试题 03.05. 栈排序 (1)

- 题目

```
栈排序。 编写程序，对栈进行排序使最小元素位于栈顶。
最多只能使用一个其他的临时栈存放数据，但不得将元素复制到别的数据结构（如数组）中。
该栈支持如下操作：push、pop、peek 和 isEmpty。当栈为空时，peek 返回 -1。
示例1:输入：["SortedStack", "push", "push", "peek", "pop", "peek"]
[[], [1], [2], [], [], []]
 输出：[null,null,null,1,null,2]
示例2:输入： ["SortedStack", "pop", "pop", "push", "pop", "isEmpty"]
[[], [], [], [1], [], []]
 输出：[null,null,null,null,null,true]
说明:栈中的元素数目在[0, 5000]范围内。
```

- 解题思路

```
type SortedStack struct {
        stack []int
        temp  []int
}

func Constructor() SortedStack {
        return SortedStack{}
}

func (this *SortedStack) Push(val int) {
        for len(this.stack) > 0 && val >= this.stack[len(this.stack)-1] {
                this.temp = append(this.temp, this.stack[len(this.stack)-1])
                this.stack = this.stack[:len(this.stack)-1]
        }
        this.stack = append(this.stack, val)
        for len(this.temp) > 0 {
                this.stack = append(this.stack, this.temp[len(this.temp)-1])
                this.temp = this.temp[:len(this.temp)-1]
        }
}

func (this *SortedStack) Pop() {
        if len(this.stack) == 0 {
                return
        }
        this.stack = this.stack[:len(this.stack)-1]
}
```

(续下页)

(接上页)

```
func (this *SortedStack) Peek() int {
        if len(this.stack) == 0 {
                return -1
        }
        return this.stack[len(this.stack)-1]
}

func (this *SortedStack) IsEmpty() bool {
        return len(this.stack) == 0
}
```

80.23 面试题 03.06. 动物收容所 (2)

- 题目

```
动物收容所。有家动物收容所只收容狗与猫，且严格遵守“先进先出”的原则。
在收养该收容所的动物时，收养人只能收养所有动物中“最老”（由其进入收容所的时间长短而定）的动物，
或者可以挑选猫或狗（同时必须收养此类动物中“最老”的）。
换言之，收养人不能自由挑选想收养的对象。
请创建适用于这个系统的数据结构，实现各种操作方法，
比如enqueue、dequeueAny、dequeueDog和dequeueCat。允许使用Java内置的LinkedList数据结构。
enqueue方法有一个animal参数，animal[0]代表动物编号，animal[1]代表动物种类，其中 0␣
↪代表猫，1 代表狗。
dequeue*方法返回一个列表[动物编号，动物种类]，若没有可以收养的动物，则返回[-1,-1]。
示例1:输入：
["AnimalShelf", "enqueue", "enqueue", "dequeueCat", "dequeueDog", "dequeueAny"]
[[], [[0, 0]], [[1, 0]], [], [], []]
 输出：[null,null,null,[0,0],[-1,-1],[1,0]]
示例2:输入：
["AnimalShelf", "enqueue", "enqueue", "enqueue", "dequeueDog", "dequeueCat",
↪"dequeueAny"]
[[], [[0, 0]], [[1, 0]], [[2, 1]], [], [], []]
输出：[null,null,null,null,[2,1],[0,0],[1,0]]
说明:收纳所的最大容量为20000
```

- 解题思路

```
type AnimalShelf struct {
        cat [][]int
        dog [][]int
}
```

(续下页)

(接上页)

```
func Constructor() AnimalShelf {
        return AnimalShelf{
                cat: make([][]int, 0),
                dog: make([][]int, 0),
        }
}

func (this *AnimalShelf) Enqueue(animal []int) {
        if animal[1] == 0 {
                this.cat = append(this.cat, animal)
        } else {
                this.dog = append(this.dog, animal)
        }
}

func (this *AnimalShelf) DequeueAny() []int {
        if len(this.dog) == 0 && len(this.cat) == 0 {
                return []int{-1, -1}
        }
        if len(this.dog) == 0 || len(this.cat) == 0 {
                if len(this.dog) == 0 {
                        res := this.cat[0]
                        this.cat = this.cat[1:]
                        return res
                }
                res := this.dog[0]
                this.dog = this.dog[1:]
                return res
        }
        if this.dog[0][0] > this.cat[0][0] {
                res := this.cat[0]
                this.cat = this.cat[1:]
                return res

        }
        res := this.dog[0]
        this.dog = this.dog[1:]
        return res
}

func (this *AnimalShelf) DequeueDog() []int {
        if len(this.dog) == 0 {
```

(续下页)

(接上页)

```
                return []int{-1, -1}
        }
        res := this.dog[0]
        this.dog = this.dog[1:]
        return res
}

func (this *AnimalShelf) DequeueCat() []int {
        if len(this.cat) == 0 {
                return []int{-1, -1}
        }
        res := this.cat[0]
        this.cat = this.cat[1:]
        return res
}

# 2
type AnimalShelf struct {
        arr [2]*list.List
}

func Constructor() AnimalShelf {
        return AnimalShelf{
                arr: [2]*list.List{list.New(), list.New()},
        }
}

func (this *AnimalShelf) Enqueue(animal []int) {
        this.arr[animal[1]].PushBack(animal[0])
}

func (this *AnimalShelf) DequeueAny() []int {
        if this.arr[0].Len() == 0 && this.arr[1].Len() == 0 {
                return []int{-1, -1}
        }
        if this.arr[1].Len() > 0 &&
                (this.arr[0].Len() == 0 || this.arr[1].Front().Value.(int) < this.
↪arr[0].Front().Value.(int)) {
                return []int{this.arr[1].Remove(this.arr[1].Front()).(int), 1}
        }
        return []int{this.arr[0].Remove(this.arr[0].Front()).(int), 0}
}
```

(续下页)

(接上页)

```
func (this *AnimalShelf) DequeueDog() []int {
        if this.arr[1].Len() > 0 {
                return []int{this.arr[1].Remove(this.arr[1].Front()).(int), 1}
        }
        return []int{-1, -1}
}

func (this *AnimalShelf) DequeueCat() []int {
        if this.arr[0].Len() > 0 {
                return []int{this.arr[0].Remove(this.arr[0].Front()).(int), 0}
        }
        return []int{-1, -1}
}
```

80.24 面试题 04.01. 节点间通路 (2)

- 题目

```
节点间通路。给定有向图，设计一个算法，找出两个节点之间是否存在一条路径。
示例1:输入：n = 3, graph = [[0, 1], [0, 2], [1, 2], [1, 2]], start = 0, target = 2
输出：true
示例2:输入：n = 5, graph = [[0, 1], [0, 2], [0, 4], [0, 4], [0, 1], [1, 3],
[1, 4], [1, 3], [2, 3], [3, 4]], start = 0, target = 4
输出 true
提示：节点数量n在[0, 1e5]范围内。
    节点编号大于等于 0 小于 n。
    图中可能存在自环和平行边。
```

- 解题思路

```
func findWhetherExistsPath(n int, graph [][]int, start int, target int) bool {
        edges := make([][]int, n)
        // 邻接表
        for i := 0; i < len(graph); i++ {
                a := graph[i][0]
                b := graph[i][1]
                edges[a] = append(edges[a], b)
        }
        queue := make([]int, 0)
        queue = append(queue, start)
        visited := make([]bool, n)
        for len(queue) > 0 {
```

(续下页)

(接上页)

```
                node := queue[0]
                queue = queue[1:]
                visited[node] = true
                if node == target {
                        return true
                }
                for i := 0; i < len(edges[node]); i++ {
                        if visited[edges[node][i]] == false {
                                if edges[node][i] == target {
                                        return true
                                }
                                queue = append(queue, edges[node][i])
                        }
                }
        }
        return false
}

# 2
func findWhetherExistsPath(n int, graph [][]int, start int, target int) bool {
        edges := make([][]int, n)
        // 邻接表
        for i := 0; i < len(graph); i++ {
                a := graph[i][0]
                b := graph[i][1]
                edges[a] = append(edges[a], b)
        }

        visited := make([]bool, n)
        return dfs(edges, visited, start, target)
}

func dfs(edges [][]int, visited []bool, start, target int) bool {
        if start == target {
                return true
        }
        visited[start] = true
        for i := 0; i < len(edges[start]); i++ {
                if visited[edges[start][i]] == false {
                        if edges[start][i] == target {
                                return true
                        }
                        if dfs(edges, visited, edges[start][i], target) {
```

(续下页)

(接上页)

```
                                return true
                        }
                }
        }
        return false
}
```

80.25 面试题 04.02. 最小高度树 (2)

- 题目

```
给定一个有序整数数组，元素各不相同且按升序排列，编写一个算法，创建一棵高度最小的二叉搜索树。
示例:给定有序数组: [-10,-3,0,5,9],
一个可能的答案是: [0,-3,9,-10,null,5], 它可以表示下面这个高度平衡二叉搜索树:
          0
         / \
       -3   9
       /   /
     -10  5
```

- 解题思路

```
func sortedArrayToBST(nums []int) *TreeNode {
        if len(nums) == 0 {
                return nil
        }
        mid := len(nums) / 2
        return &TreeNode{
                Val:   nums[mid],
                Left:  sortedArrayToBST(nums[:mid]),
                Right: sortedArrayToBST(nums[mid+1:]),
        }
}

# 2
type MyTreeNode struct {
        root  *TreeNode
        start int
        end   int
}

func sortedArrayToBST(nums []int) *TreeNode {
```

(续下页)

(接上页)

```
        if len(nums) == 0 {
                return nil
        }
        queue := make([]MyTreeNode, 0)
        root := &TreeNode{Val: 0}
        queue = append(queue, MyTreeNode{root, 0, len(nums)})
        for len(queue) > 0 {
                myRoot := queue[0]
                queue = queue[1:]
                start := myRoot.start
                end := myRoot.end
                mid := (start + end) / 2
                curRoot := myRoot.root
                curRoot.Val = nums[mid]
                if start < mid {
                        curRoot.Left = &TreeNode{Val: 0}
                        queue = append(queue, MyTreeNode{curRoot.Left, start, mid})
                }
                if mid+1 < end {
                        curRoot.Right = &TreeNode{Val: 0}
                        queue = append(queue, MyTreeNode{curRoot.Right, mid + 1, end})
                }
        }
        return root
}
```

80.26 面试题 04.03. 特定深度节点链表 (2)

- 题目

```
给定一棵二叉树，设计一个算法，创建含有某一深度上所有节点的链表
（比如，若一棵树的深度为 D，则会创建出 D 个链表）。返回一个包含所有深度的链表的数组。
示例：输入：[1,2,3,4,5,null,7,8]
        1
       /  \
      2    3
     / \    \
    4   5    7
   /
  8
输出：[[1],[2,3],[4,5,7],[8]]
```

- 解题思路

```
func listOfDepth(tree *TreeNode) []*ListNode {
        res := make([]*ListNode, 0)
        if tree == nil {
                return res
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, tree)
        for len(queue) > 0 {
                length := len(queue)
                node := &ListNode{}
                tempNode := node
                for i := 0; i < length; i++ {
                        node := queue[i]
                        tempNode.Next = &ListNode{
                                Val: node.Val,
                        }
                        tempNode = tempNode.Next
                        if node.Left != nil {
                                queue = append(queue, node.Left)
                        }
                        if node.Right != nil {
                                queue = append(queue, node.Right)
                        }
                }
                res = append(res, node.Next)
                queue = queue[length:]
        }
        return res
}

# 2
var res []*ListNode

func listOfDepth(tree *TreeNode) []*ListNode {
        level := 0
        res = make([]*ListNode, 0)
        dfs(tree, level)
        return res
}

func dfs(root *TreeNode, level int) {
        if root == nil {
                return
        }
```

(续下页)

(接上页)

```
        if level >= len(res) {
                res = append(res, &ListNode{root.Val, nil})
        } else {
                head := res[level]
                for head.Next != nil {
                        head = head.Next
                }
                head.Next = &ListNode{root.Val, nil}
        }
        dfs(root.Left, level+1)
        dfs(root.Right, level+1)
}
```

80.27 面试题 04.04. 检查平衡性 (3)

- 题目

```
实现一个函数，检查二叉树是否平衡。在这个问题中，平衡树的定义如下：
任意一个节点，其两棵子树的高度差不超过 1。
示例 1:给定二叉树 [3,9,20,null,null,15,7]
    3
   / \
  9  20
    /  \
   15   7
返回 true 。
示例 2:给定二叉树 [1,2,2,3,3,null,null,4,4]
      1
     / \
    2   2
   / \
  3   3
 / \
4   4
返回 false 。
```

- 解题思路

```
func isBalanced(root *TreeNode) bool {
        _, isBalanced := dfs(root)
        return isBalanced
}
```

(续下页)

(接上页)

```
func dfs(root *TreeNode) (int, bool) {
        if root == nil {
                return 0, true
        }

        leftDepth, leftIsBalanced := dfs(root.Left)
        if leftIsBalanced == false {
                return 0, false
        }
        rightDepth, rightIsBalanced := dfs(root.Right)
        if rightIsBalanced == false {
                return 0, false
        }

        if -1 <= leftDepth-rightDepth &&
                leftDepth-rightDepth <= 1 {
                return max(leftDepth, rightDepth) + 1, true
        }
        return 0, false
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func isBalanced(root *TreeNode) bool {
        return dfs(root) != -1
}

func dfs(root *TreeNode) int {
        if root == nil {
                return 0
        }
        left := dfs(root.Left)
        right := dfs(root.Right)
        if left != -1 && right != -1 &&
                abs(left, right) <= 1 {
                return max(left, right) + 1
```

(续下页)

(接上页)

```
        }
        return -1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

# 3
func isBalanced(root *TreeNode) bool {
        if root == nil {
                return true
        }
        if math.Abs(dfs(root.Left)-dfs(root.Right)) <= 1 {
                return isBalanced(root.Left) && isBalanced(root.Right)
        }
        return false
}

func dfs(root *TreeNode) float64 {
        if root == nil {
                return 0
        }
        return math.Max(dfs(root.Left), dfs(root.Right)) + 1
}
```

80.28 面试题 04.05. 合法二叉搜索树 (5)

- 题目

```
实现一个函数，检查一棵二叉树是否为二叉搜索树。
示例 1:输入:
    2
   / \
  1   3
输出: true
示例 2:输入:
    5
   / \
  1   4
     / \
    3   6
输出: false
解释: 输入为: [5,1,4,null,null,3,6]。根节点的值为 5 ，但是其右子节点值为 4 。
```

- 解题思路

```
func isValidBST(root *TreeNode) bool {
        return dfs(root, math.MinInt64, math.MaxInt64)
}

func dfs(root *TreeNode, left, right int) bool {
        if root == nil {
                return true
        }
        if left >= root.Val || right <= root.Val {
                return false
        }
        return dfs(root.Left, left, root.Val) && dfs(root.Right, root.Val, right)
}

# 2
var res []int

func isValidBST(root *TreeNode) bool {
        res = make([]int, 0)
        dfs(root)
        for i := 0; i < len(res)-1; i++ {
                if res[i] >= res[i+1] {
                        return false
```

(续下页)

(接上页)

```
                }
        }
        return true
}

func dfs(root *TreeNode) {
        if root != nil {
                dfs(root.Left)
                res = append(res, root.Val)
                dfs(root.Right)
        }
}

# 3
func isValidBST(root *TreeNode) bool {
    if root == nil {
        return true
    }
    stack := make([]*TreeNode, 0)
    res := make([]int, 0)
    for len(stack) > 0 || root != nil {
        for root != nil {
            stack = append(stack, root)
            root = root.Left
        }
        last := len(stack) - 1
        res = append(res, stack[last].Val)
        root = stack[last].Right
        stack = stack[:last]
    }
    for i := 0; i < len(res)-1; i++ {
        if res[i] >= res[i+1] {
            return false
        }
    }
    return true
}

# 4
func isValidBST(root *TreeNode) bool {
        if root == nil {
                return true
        }
```

(续下页)

(接上页)

```
        stack := make([]*TreeNode, 0)
        pre := math.MinInt64
        for len(stack) > 0 || root != nil {
                for root != nil {
                        stack = append(stack, root)
                        root = root.Left
                }
                last := len(stack) - 1
                if stack[last].Val <= pre {
                        return false
                }
                pre = stack[last].Val
                root = stack[last].Right
                stack = stack[:last]
        }
        return true
}

# 5
var pre int

func isValidBST(root *TreeNode) bool {
        pre = math.MinInt64
        return dfs(root)
}

func dfs(root *TreeNode) bool {
        if root == nil {
                return true
        }
        if dfs(root.Left) == false {
                return false
        }
        if root.Val <= pre {
                return false
        }
        pre = root.Val
        return dfs(root.Right)
}
```

80.29 面试题 04.06. 后继者 (3)

- 题目

```
设计一个算法，找出二叉搜索树中指定节点的“下一个”节点（也即中序后继）。
如果指定节点没有对应的“下一个”节点，则返回null。
示例 1:输入: root = [2,1,3], p = 1
  2
 / \
1   3
输出: 2
示例 2:输入: root = [5,3,6,2,4,null,null,1], p = 6
      5
     / \
    3   6
   / \
  2   4
 /
1
输出: null
```

- 解题思路

```
var res []*TreeNode

func inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {
        res = make([]*TreeNode, 0)
        dfs(root)
        for i := 0; i < len(res)-1; i++ {
                if res[i] == p {
                        return res[i+1]
                }
        }
        return nil
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
        res = append(res, root)
        dfs(root.Right)
}
```

(续下页)

(接上页)

```
# 2
func inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        if p.Val >= root.Val {
                return inorderSuccessor(root.Right, p)
        }
        res := inorderSuccessor(root.Left, p)
        if res == nil {
                return root
        }
        return res
}

# 3
func inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {
        var res *TreeNode
        cur := root
        for cur != nil {
                if p.Val >= cur.Val {
                        cur = cur.Right
                } else {
                        res = cur
                        cur = cur.Left
                }
        }
        return res
}
```

80.30 面试题 04.08. 首个共同祖先 (2)

- 题目

```
设计并实现一个算法，找出二叉树中某两个节点的第一个共同祖先。不得将其他的节点存储在另外的数据结构中。
注意：这不一定是二叉搜索树。
例如，给定如下二叉树: root = [3,5,1,6,2,0,8,null,null,7,4]
    3
   / \
  5   1
 / \ / \
```

(续下页)

(接上页)

```
6  2 0  8
  / \
 7   4
示例 1:输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1输出: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。
示例 2:输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4 输出: 5
解释: 节点 5 和节点 4 的最近公共祖先是节点␣
↪5。因为根据定义最近公共祖先节点可以为节点本身。
说明:所有节点的值都是唯一的。p、q 为不同节点且均存在于给定的二叉树中。
```

- 解题思路

```
func lowestCommonAncestor(root *TreeNode, p *TreeNode, q *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        if root.Val == p.Val || root.Val == q.Val {
                return root
        }
        left := lowestCommonAncestor(root.Left, p, q)
        right := lowestCommonAncestor(root.Right, p, q)
        if left != nil && right != nil {
                return root
        }
        if left == nil {
                return right
        }
        return left
}

# 2
func lowestCommonAncestor(root *TreeNode, p *TreeNode, q *TreeNode) *TreeNode {
        if root == nil {
                return nil
        }
        m = make(map[int]*TreeNode)
        dfs(root)
        visited := make(map[int]bool)
        for p != nil {
                visited[p.Val] = true
                p = m[p.Val]
        }
        for q != nil {
                if visited[q.Val] == true {
```

(续下页)

(接上页)

```
                        return q
                }
                q = m[q.Val]
        }
        return nil
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        if root.Left != nil {
                m[root.Left.Val] = root
                dfs(root.Left)
        }
        if root.Right != nil {
                m[root.Right.Val] = root
                dfs(root.Right)
        }
}
```

80.31 面试题 04.09. 二叉搜索树序列 (1)

- 题目

```
从左向右遍历一个数组，通过不断将其中的元素插入树中可以逐步地生成一棵二叉搜索树。
给定一个由不同节点组成的二叉搜索树，输出所有可能生成此树的数组。
示例：给定如下二叉树
        2
       / \
      1   3
返回：[
   [2,1,3],
   [2,3,1]
]
```

- 解题思路

```
var res [][]int

func BSTSequences(root *TreeNode) [][]int {
        res = make([][]int, 0)
```

(续下页)

(接上页)

```
        if root == nil {
                res = append(res, []int{})
                return res
        }
        dfs(append([]*TreeNode{}, root), make([]int, 0))
        return res
}

func dfs(arr []*TreeNode, path []int) {
        if len(arr) == 0 {
                res = append(res, path)
        }
        for i, node := range arr {
                temp := make([]int, len(path))
                copy(temp, path)
                temp = append(temp, node.Val)
                tempNode := make([]*TreeNode, len(arr))
                copy(tempNode, arr)
                tempNode = append(tempNode[:i], tempNode[i+1:]...) // 去除当前用过的
                if node.Left != nil {
                        tempNode = append(tempNode, node.Left)
                }
                if node.Right != nil {
                        tempNode = append(tempNode, node.Right)
                }
                dfs(tempNode, temp)
        }
}
```

80.32 面试题 04.10. 检查子树 (2)

- 题目

```
检查子树。你有两棵非常大的二叉树：T1，有几万个节点；T2，有几万个节点。
设计一个算法，判断 T2 是否为 T1 的子树。
如果 T1 有这么一个节点 n，其子树与 T2 一模一样，则 T2 为 T1 的子树，
也就是说，从节点 n 处把树砍断，得到的树与 T2 完全相同。
示例1:输入：t1 = [1, 2, 3], t2 = [2] 输出：true
示例2:输入：t1 = [1, null, 2, 4], t2 = [3, 2] 输出：false
提示：树的节点数目范围为[0, 20000]。
```

- 解题思路

```
func checkSubTree(t1 *TreeNode, t2 *TreeNode) bool {
        if t1 == nil {
                return false
        }
        return isSame(t1, t2) || checkSubTree(t1.Left, t2) || checkSubTree(t1.Right,␣
↪t2)
}

func isSame(s *TreeNode, t *TreeNode) bool {
        if s == nil || t == nil {
                return t == s
        }
        return isSame(s.Left, t.Left) && isSame(s.Right, t.Right) && s.Val == t.Val
}

# 2
func checkSubTree(t1 *TreeNode, t2 *TreeNode) bool {
        sStr := dfs(t1, "")
        tStr := dfs(t2, "")
        return strings.Contains(sStr, tStr)
}

func dfs(s *TreeNode, pre string) string {
        if s == nil {
                return pre
        }
        return fmt.Sprintf("#%d%s%s", s.Val, dfs(s.Left, "l"), dfs(s.Right, "r"))
}

# 3
func checkSubTree(t1 *TreeNode, t2 *TreeNode) bool {
        sStr := preOrder(t1)
        tStr := preOrder(t2)
        return strings.Contains(sStr, tStr)
}

func preOrder(root *TreeNode) string {
        if root == nil {
                return ""
        }
        res := "!"
        stack := make([]*TreeNode, 0)
        temp := root
        for {
```

(续下页)

(接上页)

```
                for temp != nil {
                        res += strconv.Itoa(temp.Val)
                        res += "!"
                        stack = append(stack, temp)
                        temp = temp.Left
                }
                res += "#!"
                if len(stack) > 0 {
                        node := stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        temp = node.Right
                } else {
                        break
                }
        }
        return res
}
```

80.33 面试题 04.12. 求和路径 (4)

- 题目

```
给定一棵二叉树，其中每个节点都含有一个整数数值(该值或正或负)。设计一个算法，
打印节点数值总和等于某个给定值的所有路径的数量。
注意，路径不一定非得从二叉树的根节点或叶节点开始或结束，但是其方向必须向下(只能从父节点指向子节点方向
示例:给定如下二叉树，以及目标和 sum = 22，
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
返回:3
解释：和为 22 的路径有：[5,4,11,2], [5,8,4,5], [4,11,7]
提示：节点总数 <= 10000
```

- 解题思路

```
func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
```

(续下页)

(接上页)

```
        }
        res := 0
        var dfs func(*TreeNode, int)
        dfs = func(node *TreeNode, sum int) {
                if node == nil {
                        return
                }
                sum = sum - node.Val
                // 路径不需要从根节点开始，也不需要在叶子节点结束
                if sum == 0 {
                        res++
                }
                dfs(node.Left, sum)
                dfs(node.Right, sum)
        }
        dfs(root, sum)
        return res + pathSum(root.Left, sum) + pathSum(root.Right, sum)
}

# 2
func dfs(node *TreeNode, sum int) int {
        if node == nil {
                return 0
        }
        sum = sum - node.Val
        res := 0
        if sum == 0 {
                res = 1
        }
        return res + dfs(node.Left, sum) + dfs(node.Right, sum)
}

func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
        }
        return dfs(root, sum) + pathSum(root.Left, sum) + pathSum(root.Right, sum)
}

# 3
func pathSum(root *TreeNode, sum int) int {
        if root == nil {
                return 0
```

(续下页)

(接上页)

```
        }
        queue := make([]*TreeNode, 0)
        queue = append(queue, root)
        res := 0
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                tempSum := 0
                res += dfs(node, sum, tempSum)
                if node.Left != nil {
                        queue = append(queue, node.Left)
                }
                if node.Right != nil {
                        queue = append(queue, node.Right)
                }
        }
        return res
}

func dfs(node *TreeNode, sum int, curSum int) int {
        res := 0
        curSum = curSum + node.Val
        if curSum == sum {
                res++
        }
        if node.Left != nil {
                res += dfs(node.Left, sum, curSum)
        }
        if node.Right != nil {
                res += dfs(node.Right, sum, curSum)
        }
        return res
}

# 4
func pathSum(root *TreeNode, sum int) int {
        return dfs(root, sum, make([]int, 1001), 0)
}

func dfs(node *TreeNode, sum int, path []int, level int) int {
        if node == nil {
                return 0
        }
```

(续下页)

(接上页)

```
        res := 0
        if sum == node.Val {
                res = 1
        }
        temp := node.Val
        for i := level - 1; i >= 0; i-- {
                temp = temp + path[i]
                if temp == sum {
                        res++
                }
        }
        path[level] = node.Val
        return res + dfs(node.Left, sum, path, level+1) +
                dfs(node.Right, sum, path, level+1)
}
```

80.34 面试题 05.01. 插入 (4)

- 题目

```
插入。给定两个32位的整数N与M，以及表示比特位置的i与j。
编写一种方法，将M插入N，使得M从N的第j位开始，到第i位结束。假定从j位到i位足以容纳M，也即若M␣
↪= 10 011，
那么j和i之间至少可容纳5个位。例如，不可能出现j = 3和i =␣
↪2的情况，因为第3位和第2位之间放不下M。
示例1:输入：N = 1024(10000000000), M = 19(10011), i = 2, j = 6 输出：N =␣
↪1100(10001001100)
示例2:输入： N = 0, M = 31(11111), i = 0, j = 4 输出：N = 31(11111)
```

- 解题思路

```
func insertBits(N int, M int, i int, j int) int {
        a := (N >> (j + 1)) << (j + 1)
        b := (N>>i)<<i ^ N
        c := M << i
        return a | b | c
}

# 2
func insertBits(N int, M int, i int, j int) int {
        for k := i; k <= j; k++ {
                if N&(1<<k) != 0 {
```

(续下页)

(接上页)

```
                        N = N - 1<<k
                }
        }
        N = N + (M << i)
        return N
}

# 3
func insertBits(N int, M int, i int, j int) int {
        arr := make([]byte, 32)
        for i := 0; i < 32; i++ {
                arr[i] = '0'
        }
        a := fmt.Sprintf("%b", N)
        b := fmt.Sprintf("%b", M)
        for k := len(a) - 1; k >= 0; k-- {
                arr[31-(len(a)-1-k)] = a[k]
        }
        count := 0
        for k := 31 - i; k >= 31-j; k-- {
                if count < len(b) {
                        arr[k] = b[len(b)-1-count]
                        count++
                } else {
                        arr[k] = '0'
                }
        }
        value, _ := strconv.ParseInt(string(arr), 2, 64)
        return int(value)
}

# 4
func insertBits(N int, M int, i int, j int) int {
        res := N
        setZero := 0
        for k := i; k <= j; k++ {
                setZero = setZero | (1 << k)
        }
        res = res&setZero ^ N
        res = res | (M << i)
        return res
}
```

80.35 面试题 05.02. 二进制数转字符串 (2)

- 题目

```
二进制数转字符串。给定一个介于0和1之间的实数（如0.
↪72），类型为double，打印它的二进制表达式。
如果该数字不在0和1之间，或者无法精确地用32位以内的二进制表示，则打印“ERROR”。
示例1:输入：0.625 输出："0.101"
示例2:输入：0.1 输出："ERROR"
提示：0.1无法被二进制准确表示
提示：32位包括输出中的"0."这两位。
```

- 解题思路

```
func printBin(num float64) string {
        res := "0."
        for num != float64(0) {
                num = num * 2
                if num >= 1 {
                        res = res + "1"
                        num = num - 1.0
                } else {
                        res = res + "0"
                }
                if len(res) > 32 {
                        return "ERROR"
                }
        }
        return res
}

# 2
func printBin(num float64) string {
        res := "0."
        value := float64(1)
        for i := 1; i <= 32; i++ {
                value = value / 2
                if num < value {
                        res = res + "0"
                        continue
                }
                res = res + "1"
                num = num - value
                if num == 0 {
```

(续下页)

(接上页)

```
                        return res
                }
        }
        return "ERROR"
}
```

80.36 面试题 05.03. 翻转数位 (2)

- 题目

```
给定一个32位整数␣
↪num，你可以将一个数位从0变为1。请编写一个程序，找出你能够获得的最长的一串1的长度。
示例 1：输入：num = 1775(110111011112)输出：8
示例 2：输入：num = 7(01112)输出：4
```

- 解题思路

```
func reverseBits(num int) int {
        res := 0
        a, b := 0, 0
        for num != 0 {
                if num%2 == 1 {
                        a++
                } else {
                        b = a
                        a = 0
                }
                res = max(res, a+b)
                num = num / 2
        }
        return res + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func reverseBits(num int) int {
```

(续下页)

(接上页)

```
        res := 0
        arr := make([]int, 0)
        count := 0
        for num != 0 {
                if num%2 == 1 {
                        count++
                } else {
                        arr = append(arr, count)
                        count = 0
                }
                num = num / 2
        }
        arr = append(arr, count)
        if len(arr) == 1 {
                return arr[0] + 1
        }
        for i := 1; i < len(arr); i++ {
                res = max(res, arr[i]+arr[i-1])
        }
        return res + 1
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

80.37 面试题 05.04. 下一个数

80.37.1 题目

```
下一个数。给定一个正整数，找出与其二进制表达式中1的个数相同且大小最接近的那两个数（一个略大，一个略小
示例1:输入：num = 2（或者0b10） 输出：[4, 1] 或者（[0b100, 0b1]）
示例2:输入：num = 1输出：[2, -1]
提示:num的范围在[1, 2147483647]之间；
    如果找不到前一个或者后一个满足条件的正数，那么输出 -1。
```

80.37.2 解题思路

80.38 面试题 05.06. 整数转换 (4)

- 题目

```
整数转换。编写一个函数，确定需要改变几个位才能将整数A转成整数B。
示例1:输入：A = 29 （或者0b11101），B = 15（或者0b01111）输出：2
示例2:输入：A = 1，B = 2 输出：2
提示：A，B范围在[-2147483648, 2147483647]之间
```

- 解题思路

```
func convertInteger(A int, B int) int {
        C := uint32(A) ^ uint32(B)
        return bits.OnesCount(uint(C))
}

# 2
func convertInteger(A int, B int) int {
        C := uint32(A) ^ uint32(B)
        res := 0
        for C != 0 {
                if C&1 == 1 {
                        res++
                }
                C = C >> 1
        }
        return res
}

# 3
func convertInteger(A int, B int) int {
        C := uint32(A) ^ uint32(B)
        res := 0
        for C != 0 {
                res++
                C = C & (C - 1)
        }
        return res
}
```

(续下页)

(接上页)

```
# 4
func convertInteger(A int, B int) int {
        C := A ^ B
        res := 0
        for i := 0; i < 32; i++{
                if C & 1 ==1{
                        res++
                }
                C = C >> 1
        }
        return res
}
```

80.39 面试题 05.07. 配对交换 (2)

- 题目

```
配对交换。编写程序，交换某个整数的奇数位和偶数位，尽量使用较少的指令
（也就是说，位0与位1交换，位2与位3交换，以此类推）。
示例1:输入：num = 2（或者0b10）输出 1 （或者 0b01）
示例2:输入：num = 3 输出：3
提示:num的范围在[0, 2^30 - 1]之间，不会发生整数溢出。
```

- 解题思路

```
func exchangeBits(num int) int {
        // 0x55555555 = 01010101010101010101010101010101 提取偶数位=>左移
        // 0xaaaaaaaa = 10101010101010101010101010101010 提取奇数位=>右移
        a := (num & 0x55555555) << 1
        b := (num & 0xaaaaaaaa) >> 1
        return a | b
}

# 2
func exchangeBits(num int) int {
        a := fmt.Sprintf("%b", num)
        arr := make([]byte, 32)
        for i := 0; i < 32; i++ {
                arr[i] = '0'
        }
        count := 31
```

(续下页)

(接上页)

```
        for i := len(a) - 1; i >= 0; i-- {
                arr[count] = a[i]
                count--
        }
        for i := len(arr) - 2; i >= 0; i = i - 2 {
                arr[i], arr[i+1] = arr[i+1], arr[i]
        }
        value, _ := strconv.ParseInt(string(arr), 2, 64)
        return int(value)
}
```

80.40 面试题 05.08. 绘制直线 (2)

- 题目

```
绘制直线。
有个单色屏幕存储在一个一维数组中，使得32个连续像素可以存放在一个 int 里。
屏幕宽度为w，且w可被32整除（即一个 int␣
↪不会分布在两行上），屏幕高度可由数组长度及屏幕宽度推算得出。
请实现一个函数，绘制从点(x1, y)到点(x2, y)的水平线。
给出数组的长度 length，宽度 w（以比特为单位）、直线开始位置␣
↪x1（比特为单位）、直线结束位置 x2（比特为单位）、直线所在行数y。
返回绘制过后的数组。
示例1:输入：length = 1, w = 32, x1 = 30, x2 = 31, y = 0 输出：[3]
说明：在第0行的第30位到第31为画一条直线，屏幕表示为[0b00000000000000000000000000000011]
示例2:输入：length = 3, w = 96, x1 = 0, x2 = 95, y = 0 输出：[-1, -1, -1]
```

- 解题思路

```
func drawLine(length int, w int, x1 int, x2 int, y int) []int {
        res := make([]int, length)
        for i := 0; i < length; i++ {
                start := i * 32
                var value int32
                for j := 0; j < 32; j++ {
                        if x1+y*w <= start+j && start+j <= x2+y*w {
                                value = value ^ (1 << (31 - j)) // 画线：把第31-
↪j位变为1
                        }
                }
                res[i] = int(value)
        }
```

(续下页)

(接上页)

```
        return res
}

# 2
func drawLine(length int, w int, x1 int, x2 int, y int) []int {
        arr := make([]int32, length)
        width := w / 32
        for i := x1; i <= x2; i++ {
                index := width*y + (i / 32)
                arr[index] = arr[index] ^ (1 << (31 - (i % 32)))
        }
        res := make([]int, length)
        for i := 0; i < length; i++ {
                res[i] = int(arr[i])
        }
        return res
}
```

80.41 面试题 08.01. 三步问题 (2)

- 题目

```
三步问题。有个小孩正在上楼梯，楼梯有n阶台阶，小孩一次可以上1阶、2阶或3阶。
实现一种方法，计算小孩有多少种上楼梯的方式。结果可能很大，你需要对结果模1000000007。
示例1:输入： n = 3 输出： 4
说明: 有四种走法
示例2:输入： n = 5输出： 13
提示:n范围在[1, 1000000]之间
```

- 解题思路

```
func waysToStep(n int) int {
        if n == 1 {
                return 1
        }
        if n == 2 {
                return 2
        }
        if n == 3 {
                return 4
        }
        a, b, c := 1, 2, 4
```

(续下页)

(接上页)

```
        for i := 4; i <= n; i++ {
                a, b, c = b, c, (a+b+c)%1000000007
        }
        return c
}

# 2
func waysToStep(n int) int {
        dp := make([]int, n+3)
        dp[0] = 1
        dp[1] = 2
        dp[2] = 4
        for i := 3; i < n; i++ {
                dp[i] = (dp[i-1] + dp[i-2] + dp[i-3]) % 1000000007
        }
        return dp[n-1]
}
```

80.42 面试题 08.02. 迷路的机器人 (2)

- 题目

```
设想有个机器人坐在一个网格的左上角，网格 r 行 c 列。
机器人只能向下或向右移动，但不能走到一些被禁止的网格（有障碍物）。
设计一种算法，寻找机器人从左上角移动到右下角的路径。
网格中的障碍物和空位置分别用 1 和 0 来表示。
返回一条可行的路径，路径由经过的网格的行号和列号组成。左上角为 0 行 0 列。
如果没有可行的路径，返回空数组。
示例 1:输入:
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
输出: [[0,0],[0,1],[0,2],[1,2],[2,2]]
解释: 输入中标粗的位置即为输出表示的路径，即
0行0列（左上角） -> 0行1列 -> 0行2列 -> 1行2列 -> 2行2列（右下角）
说明：r 和 c 的值均不超过 100。
```

- 解题思路

```
var res [][]int

func pathWithObstacles(obstacleGrid [][]int) [][]int {
        res = make([][]int, 0)
        path := make([][]int, 0)
        path = append(path, []int{0, 0})
        dfs(obstacleGrid, path)
        return res
}

func dfs(arr [][]int, path [][]int) {
        if len(res) == 0 {
                x, y := path[len(path)-1][0], path[len(path)-1][1]
                if arr[x][y] == 0 {
                        arr[x][y] = 1
                        if x < len(arr)-1 {
                                dfs(arr, append(path, []int{x + 1, y}))
                        }
                        if y < len(arr[0])-1 {
                                dfs(arr, append(path, []int{x, y + 1}))
                        }
                        if x == len(arr)-1 && y == len(arr[0])-1 {
                                res = make([][]int, len(path))
                                copy(res, path)
                        }
                }
        }
}

# 2
func pathWithObstacles(obstacleGrid [][]int) [][]int {
        res := make([][]int, 0)
        n := len(obstacleGrid)
        m := len(obstacleGrid[0])
        if obstacleGrid[0][0] == 1 || obstacleGrid[n-1][m-1] == 1 {
                return res
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if obstacleGrid[i][j] == 1 {
                                obstacleGrid[i][j] = 0
                                continue
                        }
                        if i == 0 && j == 0 {
```

(续下页)

(接上页)

```
                                obstacleGrid[i][j] = 1
                        } else if i == 0 {
                                obstacleGrid[i][j] = obstacleGrid[i][j-1] + 1
                        } else if j == 0 {
                                obstacleGrid[i][j] = obstacleGrid[i-1][j] + 1
                        } else {
                                obstacleGrid[i][j] = max(obstacleGrid[i][j-1],␣
↪obstacleGrid[i-1][j]) + 1
                        }
                }
        }
        total := n + m - 1
        if obstacleGrid[n-1][m-1] != total {
                return res
        }
        i, j := n-1, m-1
        for i >= 0 && j >= 0 {
                if obstacleGrid[i][j] == total {
                        newArr := make([][]int, 0)
                        newArr = append(newArr, []int{i, j})
                        res = append(newArr, res...)
                        total = total - 1
                }
                if i == 0 && j == 0 {
                        break
                }
                if i == 0 && obstacleGrid[i][j-1] == total {
                        j--
                } else if j == 0 && obstacleGrid[i-1][j] == total {
                        i--
                } else if obstacleGrid[i-1][j] == total {
                        i--
                } else if obstacleGrid[i][j-1] == total {
                        j--
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}
```

80.43 面试题 08.03. 魔术索引 (2)

- 题目

```
魔术索引。 在数组A[0...n-1]中，有所谓的魔术索引，满足条件A[i] = i。
给定一个有序整数数组，编写一种方法找出魔术索引，若有的话，在数组A中找出一个魔术索引，如果没有，则返回
↪1。
若有多个魔术索引，返回索引值最小的一个。
示例1:输入：nums = [0, 2, 3, 4, 5] 输出：0 说明：0下标的元素为0
示例2:输入：nums = [1, 1, 1] 输出：1
说明:
    nums长度在[1, 1000000]之间
    此题为原书中的 Follow-up，即数组中可能包含重复元素的版本
```

- 解题思路

```
func findMagicIndex(nums []int) int {
        for i := 0; i < len(nums); i++{
                if nums[i] == i{
                        return i
                }
        }
        return -1
}

# 2
func findMagicIndex(nums []int) int {
        return search(nums, 0, len(nums)-1)
}

func search(nums []int, left, right int) int {
        if left > right {
                return -1
        }
        mid := left + (right-left)/2
        res := search(nums, left, mid-1)
        if res != -1 {
                return res
        } else if nums[mid] == mid {
```

(续下页)

(接上页)

```
                return mid
        }
        return search(nums, mid+1, right)
}
```

80.44 面试题 08.04. 幂集 (3)

- 题目

```
幂集。编写一种方法，返回某集合的所有子集。集合中不包含重复的元素。
说明：解集不能包含重复的子集。
示例:输入： nums = [1,2,3]输出：
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

- 解题思路

```
var res [][]int

func subsets(nums []int) [][]int {
        res = make([][]int, 0)
        dfs(nums, make([]int, 0), 0)
        return res
}

func dfs(nums []int, arr []int, level int) {
        temp := make([]int, len(arr))
        copy(temp, arr)
        res = append(res, temp)
        for i := level; i < len(nums); i++ {
                // dfs(nums, append(arr, nums[i]), i+1)
                arr = append(arr, nums[i])
                dfs(nums, arr, i+1)
                arr = arr[:len(arr)-1]
```

(续下页)

(接上页)

```
        }
}

# 2
func subsets(nums []int) [][]int {
        res := make([][]int, 0)
        res = append(res, []int{})
        for i := 0; i < len(nums); i++ {
                temp := make([][]int, len(res))
                for key, value := range res {
                        value = append(value, nums[i])
                        temp[key] = append(temp[key], value...)
                }
                for _, v := range temp {
                        res = append(res, v)
                }
        }
        return res
}

# 3
func subsets(nums []int) [][]int {
        res := make([][]int, 0)
        n := len(nums)
        left := 1 << n
        right := 1 << (n + 1)
        for i := left; i < right; i++ {
                temp := make([]int, 0)
                for j := 0; j < n; j++ {
                        if i&(1<<j) != 0 {
                                temp = append(temp, nums[j])
                        }
                }
                res = append(res, temp)
        }
        return res
}
```

80.45 面试题 08.05. 递归乘法 (3)

- 题目

```
递归乘法。 写一个递归函数，不使用 * 运算符，␣
↪实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。
示例1:输入：A = 1, B = 10 输出：10
示例2:输入：A = 3, B = 4 输出：12
提示:保证乘法范围不会溢出
```

- 解题思路

```
func multiply(A int, B int) int {
        if B == 0 {
                return 0
        }
        return multiply(A, B-1) + A
}

# 2
func multiply(A int, B int) int {
        if B == 0 {
                return 0
        }
        if B == 1 {
                return A
        }
        if B%2 == 1 {
                return multiply(A<<1, B>>1) + A
        }
        return multiply(A<<1, B>>1)
}

# 3
func multiply(A int, B int) int {
        res := 0
        for B != 0{
                if B % 2==1{
                        res = res + A
                }
                A = A+A
                B = B >> 1
        }
        return res
```

(续下页)

(接上页)

```
}
```

80.46 面试题 08.06. 汉诺塔问题 (1)

- 题目

```
在经典汉诺塔问题中，有 3 根柱子及 N 个不同大小的穿孔圆盘，盘子可以滑入任意一根柱子。
一开始，所有盘子自上而下按升序依次套在第一根柱子上(即每一个盘子只能放在更大的盘子上面)。
移动圆盘时受到以下限制:
(1) 每次只能移动一个盘子;
(2) 盘子只能从柱子顶端滑出移到下一根柱子;
(3) 盘子只能叠在比它大的盘子上。
请编写程序，用栈将所有盘子从第一根柱子移到最后一根柱子。
你需要原地修改栈。
示例1:输入：A = [2, 1, 0], B = [], C = [] 输出：C = [2, 1, 0]
示例2:输入：A = [1, 0], B = [], C = [] 输出：C = [1, 0]
提示:A中盘子的数目不大于14个。
```

- 解题思路

```
func hanota(A []int, B []int, C []int) []int {
        if A == nil {
                return nil
        }
        move(len(A), &A, &B, &C)
        return C
}

func move(num int, A, B, C *[]int) {
        if num < 0 {
                return
        }
        if num == 1 {
                *C = append(*C, (*A)[len(*A)-1])
                *A = (*A)[:len(*A)-1]
                return
        }
        move(num-1, A, C, B)
        move(1, A, B, C)
        move(num-1, B, A, C)
}
```

80.47 面试题 08.07. 无重复字符串的排列组合 (3)

- 题目

```
无重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合，字符串每个字符均不相同。
示例1:输入：S = "qwe"输出：["qwe", "qew", "wqe", "weq", "ewq", "eqw"]
示例2:输入：S = "ab"输出：["ab", "ba"]
提示:字符都是英文字母。
    字符串长度在[1, 9]之间。
```

- 解题思路

```
var res []string

func permutation(S string) []string {
        res = make([]string, 0)
        nums := []byte(S)
        visited := make(map[int]bool)
        dfs(nums, 0, "", visited)
        return res
}

func dfs(nums []byte, index int, str string, visited map[int]bool) {
        if index == len(nums) {
                res = append(res, str)
                return
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == false {
                        str = str + string(nums[i])
                        visited[i] = true
                        dfs(nums, index+1, str, visited)
                        str = str[:len(str)-1]
                        visited[i] = false
                }
        }
}

# 2
func permutation(S string) []string {
        if len(S) == 1 {
                return []string{S}
        }
        res := make([]string, 0)
```

(续下页)

(接上页)

```
        for i := 0; i < len(S); i++ {
                str := S[:i] + S[i+1:]
                arr := permutation(str)
                for _, v := range arr {
                        res = append(res, v+string(S[i]))
                }
        }
        return res
}

# 3
var res []string

func permutation(S string) []string {
        res = make([]string, 0)
        nums := []byte(S)
        dfs(nums, 0, "")
        return res
}

func dfs(nums []byte, index int, str string) {
        if index == len(nums) {
                res = append(res, str)
                return
        }
        for i := index; i < len(nums); i++ {
                str = str + string(nums[i])
                nums[i], nums[index] = nums[index], nums[i]
                dfs(nums, index+1, str)
                nums[i], nums[index] = nums[index], nums[i]
                str = str[:len(str)-1]
        }
}
```

80.48 面试题 08.08. 有重复字符串的排列组合 (3)

- 题目

```
有重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合。
示例1:输入：S = "qqe" 输出：["eqq","qeq","qqe"]
示例2:输入：S = "ab"输出：["ab", "ba"]
提示:
```

(续下页)

(接上页)

```
    字符都是英文字母。
    字符串长度在[1, 9]之间。
```

- 解题思路

```
var res []string

func permutation(S string) []string {
        res = make([]string, 0)
        nums := []byte(S)
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] < nums[j]
        })
        dfs(nums, 0, make([]int, len(nums)), "")
        return res
}

func dfs(nums []byte, index int, visited []int, str string) {
        if len(nums) == index {
                res = append(res, str)
                return
        }
        for i := 0; i < len(nums); i++ {
                if visited[i] == 1 {
                        continue
                }
                if i > 0 && nums[i] == nums[i-1] && visited[i-1] == 0 {
                        continue
                }
                str = str + string(nums[i])
                visited[i] = 1
                dfs(nums, index+1, visited, str)
                visited[i] = 0
                str = str[:len(str)-1]
        }
}

# 2
var res []string

func permutation(S string) []string {
        res = make([]string, 0)
        nums := []byte(S)
        sort.Slice(nums, func(i, j int) bool {
```

(续下页)

(接上页)

```
                return nums[i] < nums[j]
        })
        dfs(nums, 0)
        return res
}

func dfs(nums []byte, index int) {
        if index == len(nums) {
                res = append(res, string(nums))
                return
        }
        m := make(map[byte]int)
        for i := index; i < len(nums); i++ {
                if _, ok := m[nums[i]]; ok {
                        continue
                }
                m[nums[i]] = 1
                nums[i], nums[index] = nums[index], nums[i]
                dfs(nums, index+1)
                nums[i], nums[index] = nums[index], nums[i]
        }
}

# 3
var res []string

func permutation(S string) []string {
        res = make([]string, 0)
        nums := []byte(S)
        sort.Slice(nums, func(i, j int) bool {
                return nums[i] < nums[j]
        })
        dfs(nums, "")
        return res
}

func dfs(nums []byte, str string) {
        if len(nums) == 0 {
                res = append(res, str)
                return
        }
        for i := 0; i < len(nums); i++ {
                if i > 0 && nums[i] == nums[i-1] {
```

(续下页)

(接上页)

```
                        continue
                }
                str = str + string(nums[i])
                arr := append([]byte{}, nums[:i]...)
                arr = append(arr, nums[i+1:]...)
                dfs(arr, str)
                str = str[:len(str)-1]
        }
}
```

80.49 面试题 08.09. 括号 (3)

- 题目

```
括号。设计一种算法，打印n对括号的所有合法的（例如，开闭一一对应）组合。
说明：解集不能包含重复的子集。
例如，给出 n = 3，生成结果为：
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

- 解题思路

```
var res []string

func generateParenthesis(n int) []string {
        res = make([]string, 0)
        dfs(0, 0, n, "")
        return res
}

func dfs(left, right, max int, str string) {
        if left == right && left == max {
                res = append(res, str)
                return
        }
        if left < max {
                dfs(left+1, right, max, str+"(")
```

(续下页)

(接上页)

```
        }
        if right < left {
                dfs(left, right+1, max, str+")")
        }
}

# 2
/*
dp[i]表示n=i时括号的组合
dp[i]="(" + dp[j] + ")"+dp[i-j-1] (j<i)
dp[0] = ""
*/
func generateParenthesis(n int) []string {
        dp := make([][]string, n+1)
        dp[0] = make([]string, 0)
        if n == 0 {
                return dp[0]
        }
        dp[0] = append(dp[0], "")
        for i := 1; i <= n; i++ {
                dp[i] = make([]string, 0)
                for j := 0; j < i; j++ {
                        for _, a := range dp[j] {
                                for _, b := range dp[i-j-1] {
                                        str := "(" + a + ")" + b
                                        dp[i] = append(dp[i], str)
                                }
                        }
                }
        }
        return dp[n]
}

# 3
type Node struct {
        str   string
        left  int
        right int
}

func generateParenthesis(n int) []string {
        res := make([]string, 0)
        if n == 0 {
```

(续下页)

(接上页)

```
                return res
        }
        queue := make([]*Node, 0)
        queue = append(queue, &Node{
                str:   "",
                left:  n,
                right: n,
        })
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                if node.left == 0 && node.right == 0 {
                        res = append(res, node.str)
                }
                if node.left > 0 {
                        queue = append(queue, &Node{
                                str:   node.str + "(",
                                left:  node.left - 1,
                                right: node.right,
                        })
                }
                if node.right > 0 && node.left < node.right {
                        queue = append(queue, &Node{
                                str:   node.str + ")",
                                left:  node.left,
                                right: node.right - 1,
                        })
                }
        }
        return res
}
```

80.50 面试题 08.10. 颜色填充 (2)

- 题目

```
编写函数，实现许多图片编辑软件都支持的「颜色填充」功能。
待填充的图像用二维数组 image 表示，元素为初始颜色值。初始坐标点的横坐标为 sr 纵坐标为␣
↪sc。
需要填充的新颜色为 newColor 。
「周围区域」是指颜色相同且在上、下、左、右四个方向上存在相连情况的若干元素。
请用新颜色填充初始坐标点的周围区域，并返回填充后的图像。
```

(续下页)

(接上页)

```
示例：输入： image = [[1,1,1],[1,1,0],[1,0,1]]  sr = 1, sc = 1, newColor = 2
输出：[[2,2,2],[2,2,0],[2,0,1]]
解释: 初始坐标点位于图像的正中间，坐标 (sr,sc)=(1,1) 。
初始坐标点周围区域上所有符合条件的像素点的颜色都被更改成 2 。
注意，右下角的像素没有更改为 2 ，因为它不属于初始坐标点的周围区域。
提示：
    image 和 image[0] 的长度均在范围 [1, 50] 内。
    初始坐标点 (sr,sc) 满足 0 <= sr < image.length 和 0 <= sc < image[0].length 。
    image[i][j] 和 newColor 表示的颜色值在范围 [0, 65535] 内。
```

- 解题思路

```
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func floodFill(image [][]int, sr int, sc int, newColor int) [][]int {
        oldColor := image[sr][sc]
        if oldColor == newColor {
                return image
        }
        m, n := len(image), len(image[0])
        list := make([][]int, 1)
        list[0] = []int{sr, sc}

        for len(list) > 0 {
                node := list[0]
                list = list[1:]
                image[node[0]][node[1]] = newColor
                for i := 0; i < 4; i++ {
                        x := node[0] + dx[i]
                        y := node[1] + dy[i]
                        if 0 <= x && x < m && 0 <= y && y < n &&
                                image[x][y] == oldColor {
                                list = append(list, []int{x, y})
                        }
                }
        }
        return image
}

# 2
var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}
```

(续下页)

(接上页)

```
func floodFill(image [][]int, sr int, sc int, newColor int) [][]int {
        if sr < 0 || sc < 0 || sr >= len(image) ||
                sc >= len(image[sr]) || image[sr][sc] == newColor {
                return image
        }
        oldColor := image[sr][sc]
        image[sr][sc] = newColor
        for i := 0; i < 4; i++ {
                x := sr + dx[i]
                y := sc + dy[i]
                if 0 <= x && x < len(image) && 0 <= y && y < len(image[x]) &&
                        image[x][y] == oldColor {
                        floodFill(image, x, y, newColor)
                }
        }
        return image
}
```

80.51 面试题 08.11. 硬币 (2)

- 题目

```
硬币。给定数量不限的硬币，币值为25分、10分、5分和1分，编写代码计算n分有几种表示法。
(结果可能会很大，你需要将结果模上1000000007)
示例1:输入: n = 5 输出：2 解释: 有两种方式可以凑成总金额:
5=5
5=1+1+1+1+1
示例2:输入: n = 10 输出：4 解释: 有四种方式可以凑成总金额:
10=10
10=5+5
10=5+1+1+1+1+1
10=1+1+1+1+1+1+1+1+1+1
说明：注意: 你可以假设： 0 <= n (总金额) <= 1000000
```

- 解题思路

```
func waysToChange(n int) int {
        coins := []int{1, 5, 10, 25}
        dp := make([][]int, 5)
        for i := 0; i <= 4; i++ {
                dp[i] = make([]int, n+1)
                dp[i][0] = 1 // 金额为0的情况，只有都不选，组合情况为1
```

(续下页)

(接上页)

```
        }
        for i := 1; i <= 4; i++ {
                for j := 1; j <= n; j++ {
                        if j-coins[i-1] >= 0 {
                                dp[i][j] = dp[i-1][j] + dp[i][j-coins[i-1]]
                        } else {
                                dp[i][j] = dp[i-1][j]
                        }
                }
        }
        return dp[4][n] % 1000000007
}

# 2
func waysToChange(n int) int {
        coins := []int{1, 5, 10, 25}
        dp := make([]int, n+1)
        dp[0] = 1
        for i := 1; i <= 4; i++ {
                for j := 1; j <= n; j++ {
                        if j-coins[i-1] >= 0 {
                                dp[j] = dp[j] + dp[j-coins[i-1]]
                        }
                }
        }
        return dp[n] % 1000000007
}
```

80.52 面试题 08.12. 八皇后 (3)

- 题目

```
设计一种算法，打印 N 皇后在 N × N␣
↪棋盘上的各种摆法，其中每个皇后都不同行、不同列，也不在对角线上。
这里的“对角线”指的是所有的对角线，不只是平分整个棋盘的那两条对角线。
注意：本题相对原题做了扩展
示例:输入：4 输出：[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
解释：4皇后问题存在如下两个不同的解法。
[
 [".Q..",  // 解法 1
  "...Q",
  "Q...",
```

(续下页)

(接上页)

```
  "..Q."],
 ["..Q.",  // 解法 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

- 解题思路

```
var res [][]string

func solveNQueens(n int) [][]string {
        res = make([][]string, 0)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0)
        return res
}

func dfs(arr [][]string, row int) {
        if len(arr) == row {
                temp := make([]string, 0)
                for i := 0; i < len(arr); i++ {
                        str := ""
                        for j := 0; j < len(arr[i]); j++ {
                                str = str + arr[i][j]
                        }
                        temp = append(temp, str)
                }
                res = append(res, temp)
                return
        }
        // 每列尝试
        for col := 0; col < len(arr[0]); col++ {
                if valid(arr, row, col) == false {
                        continue
                }
```

(续下页)

(接上页)

```
                arr[row][col] = "Q"
                dfs(arr, row+1)
                arr[row][col] = "."
        }
}

func valid(arr [][]string, row, col int) bool {
        n := len(arr)
        // 当前列判断(竖着)
        for row := 0; row < n; row++ {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        // 左上角
        for row, col := row-1, col-1; row >= 0 && col >= 0; row, col = row-1, col-1 {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        // 右上角
        for row, col := row-1, col+1; row >= 0 && col < n; row, col = row-1, col+1 {
                if arr[row][col] == "Q" {
                        return false
                }
        }
        return true
}

# 2
var res [][]string
var rows, left, right []bool

func solveNQueens(n int) [][]string {
        res = make([][]string, 0)
        rows, left, right = make([]bool, n), make([]bool, 2*n-1), make([]bool, 2*n-1)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
```

(续下页)

(接上页)

```
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0)
        return res
}

func dfs(arr [][]string, row int) {
        n := len(arr)
        if len(arr) == row {
                temp := make([]string, 0)
                for i := 0; i < n; i++ {
                        str := ""
                        for j := 0; j < n; j++ {
                                str = str + arr[i][j]
                        }
                        temp = append(temp, str)
                }
                res = append(res, temp)
                return
        }
        // 每列尝试
        for col := 0; col < n; col++ {
                if rows[col] == true || left[row-col+n-1] == true || right[row+col]␣
↪== true {
                        continue
                }
                rows[col], left[row-col+n-1], right[row+col] = true, true, true
                arr[row][col] = "Q"
                dfs(arr, row+1)
                arr[row][col] = "."
                rows[col], left[row-col+n-1], right[row+col] = false, false, false
        }
}

# 3
var res [][]string

func solveNQueens(n int) [][]string {
        res = make([][]string, 0)
        // 初始化棋盘
        arr := make([][]string, n)
        for i := 0; i < n; i++ {
                arr[i] = make([]string, n)
```

(续下页)

(接上页)

```
                for j := 0; j < n; j++ {
                        arr[i][j] = "."
                }
        }
        // 从第1行开始,上层是满足条件
        dfs(arr, 0, 0, 0, 0)
        return res
}

func dfs(arr [][]string, row int, rows, left, right int) {
        n := len(arr)
        if len(arr) == row {
                temp := make([]string, 0)
                for i := 0; i < n; i++ {
                        str := ""
                        for j := 0; j < n; j++ {
                                str = str + arr[i][j]
                        }
                        temp = append(temp, str)
                }
                res = append(res, temp)
                return
        }
        // 每列尝试
        for col := 0; col < n; col++ {
                a := uint(col)
                b := uint(row - col + n - 1)
                c := uint(row + col)
                if ((rows>>a)&1) != 0 || ((left>>b)&1) != 0 || ((right>>c)&1) != 0 {
                        continue
                }
                arr[row][col] = "Q"
                dfs(arr, row+1, rows^(1<<a), left^(1<<b), right^(1<<c))
                arr[row][col] = "."
        }
}
```

80.53 面试题 08.13. 堆箱子 (1)

- 题目

```
堆箱子。给你一堆n个箱子，箱子宽 wi、深 di、高 hi。
箱子不能翻转，将箱子堆起来时，下面箱子的宽度、高度和深度必须大于上面的箱子。
实现一种方法，搭出最高的一堆箱子。箱堆的高度为每个箱子高度的总和。
输入使用数组[wi, di, hi]表示每个箱子。
示例1:输入：box = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]输出：6
示例2:输入：box = [[1, 1, 1], [2, 3, 4], [2, 6, 7], [3, 4, 5]] 输出：10
提示:箱子的数目不大于3000个。
```

- 解题思路

```
func pileBox(box [][]int) int {
        sort.Slice(box, func(i, j int) bool {
                if box[i][0] == box[j][0] {
                        if box[i][1] == box[j][1] {
                                return box[i][2] < box[j][2]
                        }
                        return box[i][1] < box[j][1]
                }
                return box[i][0] < box[j][0]
        })
        n, res := len(box), 0
        dp := make([]int, n)
        for i := 0; i < n; i++ {
                dp[i] = box[i][2]
        }
        for i := 0; i < n; i++ {
                for j := 0; j < i; j++ {
                        if box[j][0] < box[i][0] && box[j][1] < box[i][1] &&␣
↪box[j][2] < box[i][2] {
                                dp[i] = max(dp[i], dp[j]+box[i][2])
                        }
                }
                res = max(res, dp[i])
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
```

(续下页)

(接上页)

```
        }
        return b
}
```

80.54 面试题 08.14. 布尔运算 (3)

- 题目

```
给定一个布尔表达式和一个期望的布尔结果 result，布尔表达式由 0 (false)、1 (true)、&␣
↪(AND)、
| (OR) 和 ^ (XOR) 符号组成。实现一个函数，算出有几种可使该表达式得出 result␣
↪值的括号方法。
示例 1:输入: s = "1^0|0|1", result = 0  输出: 2
解释:两种可能的括号方法是
1^(0|(0|1))
1^((0|0)|1)
示例 2:输入: s = "0&0&0&1^1|0", result = 1 输出: 10
提示: 运算符的数量不超过 19 个
```

- 解题思路

```
func countEval(s string, result int) int {
        n := len(s)
        // dp[i][j][0/1] => s[i:j+1]结果为0/1的方法数
        dp := make([][][2]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([][2]int, n)
        }
        for i := n - 1; i >= 0; i = i - 2 {
                for j := i; j < n; j = j + 2 {
                        if i == j {
                                if s[i] == '0' {
                                        dp[i][j][0]++
                                } else {
                                        dp[i][j][1]++
                                }
                                continue
                        }
                        for k := i + 1; k < j; k = k + 2 { // 枚举操作符
                                for a := 0; a <= 1; a++ {
                                        for b := 0; b <= 1; b++ {
                                                if getValue(a, b, s[k]) == 0 {
```

(续下页)

(接上页)

```
                                                        dp[i][j][0] = dp[i][j][0] +
                                                                dp[i][k-
↪1][a]*dp[k+1][j][b]
                                                } else {
                                                        dp[i][j][1] = dp[i][j][01] +
                                                                dp[i][k-
↪1][a]*dp[k+1][j][b]
                                                }
                                        }
                                }
                        }
                }
        }
        return dp[0][n-1][result]
}

func getValue(a, b int, op byte) int {
        if op == '&' {
                return a & b
        } else if op == '|' {
                return a | b
        }
        return a ^ b
}

# 2
func countEval(s string, result int) int {
        n := len(s)
        // dp[i][j][0/1] => s[i:j+1]结果为0/1的方法数
        dp := make([][][2]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([][2]int, n)
        }
        for i := n - 1; i >= 0; i = i - 2 {
                for j := i; j < n; j = j + 2 {
                        if i == j {
                                dp[i][j][s[i]-'0']++
                                continue
                        }
                        for k := i + 1; k < j; k = k + 2 { // 枚举操作符
                                for a := 0; a <= 1; a++ {
                                        for b := 0; b <= 1; b++ {
                                                temp := getValue(a, b, s[k])
```

(续下页)

(接上页)

```
                                                dp[i][j][temp] = dp[i][j][temp] +
                                                        dp[i][k-1][a]*dp[k+1][j][b]
                                        }
                                }
                        }
                }
        }
        return dp[0][n-1][result]
}

func getValue(a, b int, op byte) int {
        if op == '&' {
                return a & b
        } else if op == '|' {
                return a | b
        }
        return a ^ b
}

# 3
func countEval(s string, result int) int {
        n := len(s)
        // dp[i][j][0/1] => s[i:j+1]结果为0/1的方法数
        dp := make([][][2]int, n)
        for i := 0; i < n; i++ {
                dp[i] = make([][2]int, n)
                if i%2 == 0 {
                        dp[i][i][int(s[i]-'0')]++
                }
        }
        for length := 2; length <= n; length = length + 2 { // 枚举长度
                for i := 0; i <= n-length; i = i + 2 { // 枚举起点
                        j := i + length                    // 确定终点
                        for k := i + 1; k < j; k = k + 2 { // 枚举操作符
                                for a := 0; a <= 1; a++ {
                                        for b := 0; b <= 1; b++ {
                                                temp := getValue(a, b, s[k])
                                                dp[i][j][temp] = dp[i][j][temp] +
                                                        dp[i][k-1][a]*dp[k+1][j][b]
                                        }
                                }
                        }
                }
```

(续下页)

(接上页)

```
        }
        return dp[0][n-1][result]
}

func getValue(a, b int, op byte) int {
        if op == '&' {
                return a & b
        } else if op == '|' {
                return a | b
        }
        return a ^ b
}
```

80.55 面试题 10.01. 合并排序的数组 (3)

- 题目

```
给定两个排序后的数组 A 和 B，其中 A 的末端有足够的缓冲空间容纳 B。 编写一个方法，将 B␣
↪合并入 A 并排序。
初始化 A 和 B 的元素数量分别为 m 和 n。
示例:输入:
A = [1,2,3,0,0,0], m = 3
B = [2,5,6],       n = 3
输出: [1,2,2,3,5,6]
说明:A.length == n + m
```

- 解题思路

```
func merge(A []int, m int, B []int, n int) {
        A = A[:m]
        A = append(A, B[:n]...)
        sort.Ints(A)
}

# 2
func merge(A []int, m int, B []int, n int) {
        for m > 0 && n > 0 {
                if A[m-1] < B[n-1] {
                        A[m+n-1] = B[n-1]
                        n--
                } else {
                        A[m+n-1] = A[m-1]
```

(续下页)

(接上页)

```
                        m--
                }
        }
        if m == 0 && n > 0 {
                for n > 0 {
                        A[n-1] = B[n-1]
                        n--
                }
        }
}

# 3
func merge(A []int, m int, B []int, n int) {
        temp := make([]int, m)
        copy(temp, A)

        if n == 0 {
                return
        }
        first, second := 0, 0
        for i := 0; i < len(A); i++ {
                if second >= n {
                        A[i] = temp[first]
                        first++
                        continue
                }
                if first >= m {
                        A[i] = B[second]
                        second++
                        continue
                }
                if temp[first] < B[second] {
                        A[i] = temp[first]
                        first++
                } else {
                        A[i] = B[second]
                        second++
                }
        }
}
```

80.56 面试题 10.02. 变位词组 (2)

- 题目

```
编写一种方法，对字符串数组进行排序，将所有变位词组合在一起。变位词是指字母相同，但排列不同的字符串。
注意：本题相对原题稍作修改
示例:输入: ["eat", "tea", "tan", "ate", "nat", "bat"], 输出:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
说明：所有输入均为小写字母。
    不考虑答案输出的顺序。
```

- 解题思路

```
func groupAnagrams(strs []string) [][]string {
        m := make(map[string]int)
        res := make([][]string, 0)
        for i := 0; i < len(strs); i++ {
                arr := []byte(strs[i])
                sort.Slice(arr, func(i, j int) bool {
                        return arr[i] < arr[j]
                })
                newStr := string(arr)
                if _, ok := m[newStr]; ok {
                        res[m[newStr]] = append(res[m[newStr]], strs[i])
                } else {
                        m[newStr] = len(res)
                        res = append(res, []string{strs[i]})
                }
        }
        return res
}

# 2
func groupAnagrams(strs []string) [][]string {
        m := make(map[[26]int]int)
        res := make([][]string, 0)
        for i := 0; i < len(strs); i++ {
                arr := [26]int{}
                for j := 0; j < len(strs[i]); j++{
                        arr[strs[i][j]-'a']++
```

(续下页)

(接上页)

```
                }
                if _, ok := m[arr]; ok {
                        res[m[arr]] = append(res[m[arr]], strs[i])
                } else {
                        m[arr] = len(res)
                        res = append(res, []string{strs[i]})
                }
        }
        return res
}
```

80.57 面试题 10.03. 搜索旋转数组 (2)

- 题目

```
搜索旋转数组。给定一个排序后的数组，包含n个整数，但这个数组已被旋转过很多次了，次数不详。
请编写代码找出数组中的某个元素，假设数组元素原先是按升序排列的。若有多个相同元素，返回索引值最小的一
示例1: 输入: arr = [15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14], target = 5
输出: 8 (元素5在该数组中的索引)
示例2:输入: arr = [15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14], target = 11
输出: -1 (没有找到)
提示:arr 长度范围在[1, 1000000]之间
```

- 解题思路

```
func search(nums []int, target int) int {
        left, right := 0, len(nums)-1
        for left < right {
                mid := left + (right-left)/2
                if nums[left] < nums[mid] { // 左边升序的情况
                        if nums[left] <= target && target <= nums[mid] {
                                right = mid
                        } else {
                                left = mid + 1
                        }
                } else if nums[left] > nums[mid] { // 右边升序
                        if nums[mid] < target && target <= nums[right] && nums[left] >
↪ nums[right] {
                                left = mid + 1
                        } else {
                                right = mid
                        }
```

(续下页)

(接上页)

```
                } else if nums[left] == nums[mid] {
                        if nums[left] != target {
                                left++
                        } else {
                                return left
                        }
                }
        }
        if nums[left] == target {
                return left
        }
        return -1
}

#
func search(nums []int, target int) int {
        for i := 0; i < len(nums); i++ {
                if target == nums[i] {
                        return i
                }
        }
        return -1
}
```

80.58 面试题 10.05. 稀疏数组搜索 (2)

- 题目

```
稀疏数组搜索。有个排好序的字符串数组，其中散布着一些空字符串，编写一种方法，找出给定字符串的位置。
示例1:
输入: words = ["at", "", "", "", "ball", "", "", "car", "", "","dad", "", ""], s = "ta
↪"
输出: -1
说明: 不存在返回-1。
示例2:
输入: words = ["at", "", "", "", "ball", "", "", "car", "", "","dad", "", ""], s =
↪"ball"
输出: 4
提示: words的长度在[1, 1000000]之间
```

- 解题思路

```
func findString(words []string, s string) int {
        left := 0
        right := len(words) - 1
        for left <= right {
                mid := left + (right-left)/2
                index := mid
                word := words[mid]
                if word == "" {
                        for index = mid; index <= right; index++ {
                                if words[index] != "" {
                                        word = words[index]
                                        break
                                }
                        }
                }
                if word == s {
                        return index
                } else if word < s {
                        left = index + 1
                } else {
                        right = mid - 1
                }
        }
        return -1
}

# 2
func findString(words []string, s string) int {
        for i := 0; i < len(words); i++ {
                if s == words[i] {
                        return i
                }
        }
        return -1
}
```

80.59 面试题 10.09. 排序矩阵查找 (6)

- 题目

```
给定M×N矩阵，每一行、每一列都按升序排列，请编写代码找出某元素。
示例:现有矩阵 matrix 如下：
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
给定 target = 5，返回 true。
给定 target = 20，返回 false。
```

- 解题思路

```
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                for j := 0; j < len(matrix[i]); j++ {
                        if matrix[i][j] == target {
                                return true
                        }
                }
        }
        return false
}

# 2
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
```

(续下页)

(接上页)

```
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        for j := 0; j < len(matrix[i]); j++ {
                                if matrix[i][j] == target {
                                        return true
                                }
                        }
                }
        }
        return false
}

# 3
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                if matrix[i][0] <= target && matrix[i][len(matrix[i])-1] >= target {
                        res := binarySearch(matrix[i], target)
                        if res == true {
                                return true
                        }
                }
        }
        return false
}

func binarySearch(arr []int, target int) bool {
        left := 0
        right := len(arr) - 1
        for left <= right {
                mid := left + (right-left)/2
                if arr[mid] == target {
                        return true
                } else if arr[mid] > target {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
```

(续下页)

(接上页)

```
        return false
}

# 4
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := len(matrix) - 1
        j := 0
        for i >= 0 && j < len(matrix[0]) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        i--
                } else {
                        j++
                }
        }
        return false
}

# 5
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        i := 0
        j := len(matrix[0]) - 1
        for j >= 0 && i < len(matrix) {
                if matrix[i][j] == target {
                        return true
                } else if matrix[i][j] > target {
                        j--
                } else {
                        i++
                }
```

(续下页)

(接上页)

```
        }
        return false
}

# 6
func searchMatrix(matrix [][]int, target int) bool {
        if len(matrix) == 0 {
                return false
        }
        if len(matrix[0]) == 0 {
                return false
        }
        for i := 0; i < len(matrix); i++ {
                index := sort.SearchInts(matrix[i], target)
                if index < len(matrix[i]) && target == matrix[i][index] {
                        return true
                }
        }
        return false
}
```

80.60 面试题 10.10. 数字流的秩 (3)

- 题目

```
假设你正在读取一串整数。每隔一段时间，你希望能找出数字 x 的秩(小于或等于 x␣
↪的值的个数)。
请实现数据结构和算法来支持这些操作，也就是说：
实现 track(int x)方法，每读入一个数字都会调用该方法；
实现 getRankOfNumber(int x) 方法，返回小于或等于 x 的值的个数。
注意：本题相对原题稍作改动
示例:输入: ["StreamRank", "getRankOfNumber", "track", "getRankOfNumber"] [[], [1],␣
↪[0], [0]]
输出: [null,0,null,1]
提示：x <= 50000
track和getRankOfNumber 方法的调用次数均不超过 2000 次
```

- 解题思路

```
type StreamRank struct {
        length int
        c      []int
```

(续下页)

(接上页)

```
}

func Constructor() StreamRank {
        return StreamRank{
                length: 50002,
                c:      make([]int, 50003),
        }
}

func (this *StreamRank) Track(x int) {
        this.upData(x+1, 1)
}

func (this *StreamRank) GetRankOfNumber(x int) int {
        return this.getSum(x + 1)
}

func (this *StreamRank) lowBit(x int) int {
        return x & (-x)
}

// 单点修改
func (this *StreamRank) upData(i, k int) { // 在i位置加上k
        for i <= this.length {
                this.c[i] = this.c[i] + k
                i = i + this.lowBit(i) // i = i + 2^k
        }
}

// 区间查询
func (this *StreamRank) getSum(i int) int {
        res := 0
        for i > 0 {
                res = res + this.c[i]
                i = i - this.lowBit(i)
        }
        return res
}

# 2
type StreamRank struct {
        m map[int]int
}
```

(续下页)

(接上页)

```
func Constructor() StreamRank {
        return StreamRank{m: make(map[int]int)}
}

func (this *StreamRank) Track(x int) {
        this.m[x]++
}

func (this *StreamRank) GetRankOfNumber(x int) int {
        res := 0
        for k, v := range this.m {
                if k <= x {
                        res = res + v
                }
        }
        return res
}

# 3
type StreamRank struct {
        arr []int
}

func Constructor() StreamRank {
        return StreamRank{arr: make([]int, 0)}
}

func (this *StreamRank) Track(x int) {
        index := sort.Search(len(this.arr), func(i int) bool {
                return x <= this.arr[i]
        })
        temp := append([]int{}, this.arr[:index]...)
        temp = append(temp, x)
        temp = append(temp, this.arr[index:]...)
        this.arr = temp
}

func (this *StreamRank) GetRankOfNumber(x int) int {
        return sort.Search(len(this.arr), func(i int) bool {
                return x < this.arr[i]
        })
}
```

80.61 面试题 10.11. 峰与谷 (2)

- 题目

```
在一个整数数组中，“峰”是大于或等于相邻整数的元素，相应地，“谷”是小于或等于相邻整数的元素。
例如，在数组{5, 8, 4, 2, 3, 4, 6}中，{8, 6}是峰， {5, 2}是谷。
现在给定一个整数数组，将该数组按峰与谷的交替顺序排序。
示例:输入: [5, 3, 1, 2, 3] 输出:[5, 1, 3, 2, 3]
提示：nums.length <= 10000
```

- 解题思路

```
func wiggleSort(nums []int) {
        for i := 0; i < len(nums)-1; i++ {
                if (i%2 == 1 && nums[i] > nums[i+1]) ||
                        (i%2 == 0 && nums[i] < nums[i+1]) {
                        nums[i], nums[i+1] = nums[i+1], nums[i]
                }
        }
}

# 2
func wiggleSort(nums []int) {
        sort.Ints(nums)
        for i := 0; i < len(nums)-1; i = i + 2 {
                nums[i], nums[i+1] = nums[i+1], nums[i]
        }
}
```

80.62 面试题 16.01. 交换数字 (3)

- 题目

```
编写一个函数，不用临时变量，直接交换numbers = [a, b]中a与b的值。
示例：输入：numbers = [1,2] 输出：[2,1]
提示：  numbers.length == 2
```

- 解题思路

```
func swapNumbers(numbers []int) []int {
        return []int{numbers[1], numbers[0]}
}
```

(续下页)

(接上页)

```
# 2
func swapNumbers(numbers []int) []int {
        numbers[0] = numbers[0] ^ numbers[1]
        numbers[1] = numbers[1] ^ numbers[0]
        numbers[0] = numbers[0] ^ numbers[1]
        return numbers
}

# 3
func swapNumbers(numbers []int) []int {
        numbers[0] = numbers[0] + numbers[1]
        numbers[1] = numbers[0] - numbers[1]
        numbers[0] = numbers[0] - numbers[1]
        return numbers
}
```

80.63 面试题 16.02. 单词频率 (2)

- 题目

```
设计一个方法，找出任意指定单词在一本书中的出现频率。
你的实现应该支持如下操作：
    WordsFrequency(book)构造函数，参数为字符串数组构成的一本书
    get(word)查询指定单词在书中出现的频率
示例：WordsFrequency wordsFrequency =
new WordsFrequency({"i", "have", "an", "apple", "he", "have", "a", "pen"});
wordsFrequency.get("you"); //返回0，"you"没有出现过
wordsFrequency.get("have"); //返回2，"have"出现2次
wordsFrequency.get("an"); //返回1
wordsFrequency.get("apple"); //返回1
wordsFrequency.get("pen"); //返回1
提示：book[i]中只包含小写字母
    1 <= book.length <= 100000
    1 <= book[i].length <= 10
    get函数的调用次数不会超过100000
```

- 解题思路

```
type WordsFrequency struct {
        m map[string]int
}
```

(续下页)

(接上页)

```
func Constructor(book []string) WordsFrequency {
        res := WordsFrequency{m: make(map[string]int)}
        for k := range book {
                res.m[book[k]]++
        }
        return res
}

func (this *WordsFrequency) Get(word string) int {
        return this.m[word]
}

# 2
type WordsFrequency struct {
        ending int
        next   [26]*WordsFrequency
}

func Constructor(book []string) WordsFrequency {
        res := WordsFrequency{}
        for _, v := range book {
                res.Insert(v)
        }
        return res
}

func (this *WordsFrequency) Get(word string) int {
        temp := this
        for _, v := range word {
                nextWord := v - 'a'
                if temp.next[nextWord] == nil {
                        return 0
                }
                temp = temp.next[nextWord]
        }
        return temp.ending
}

func (this *WordsFrequency) Insert(word string) {
        temp := this
        for _, v := range word {
                nextWord := v - 'a'
                if temp.next[nextWord] == nil {
```

(续下页)

(接上页)

```
                        temp.next[nextWord] = &WordsFrequency{}
                }
                temp = temp.next[nextWord]
        }
        temp.ending = temp.ending + 1
}
```

80.64 面试题 16.04. 井字游戏 (1)

- 题目

```
设计一个算法，判断玩家是否赢了井字游戏。输入是一个 N x N 的数组棋盘，由字符" "，"X"和
↪"O"组成，其中字符" "代表一个空位。
以下是井字游戏的规则：
玩家轮流将字符放入空位（" "）中。
第一个玩家总是放字符"O"，且第二个玩家总是放字符"X"。
"X"和"O"只允许放置在空位中，不允许对已放有字符的位置进行填充。
当有N个相同（且非空）的字符填充任何行、列或对角线时，游戏结束，对应该字符的玩家获胜。
当所有位置非空时，也算为游戏结束。
如果游戏结束，玩家不允许再放置字符。
如果游戏存在获胜者，就返回该游戏的获胜者使用的字符（"X"或"O"）；
如果游戏以平局结束，则返回 "Draw"；
如果仍会有行动（游戏未结束），则返回 "Pending"。
示例 1：输入： board = ["O X"," XO","X O"] 输出： "X"
示例 2：输入： board = ["OOX","XXO","OXO"] 输出： "Draw"
解释： 没有玩家获胜且不存在空位
示例 3：输入： board = ["OOX","XXO","OX "] 输出： "Pending"
解释： 没有玩家获胜且仍存在空位
提示：1 <= board.length == board[i].length <= 100
输入一定遵循井字棋规则
```

- 解题思路

```go
func tictactoe(board []string) string {
        n := len(board)
        flag := false                       // 有没有空格
        rows := make([][2]int, n)           // 行
        cols := make([][2]int, n)           // 列
        left, right := [2]int{}, [2]int{} // 对角线
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if board[i][j] == ' ' {
```

(续下页)

(接上页)

```
                                flag = true
                        } else if board[i][j] == 'X' {
                                rows[i][0]++
                                cols[j][0]++
                                if i == j {
                                        left[0]++
                                }
                                if i == n-1-j {
                                        right[0]++
                                }
                        } else if board[i][j] == 'O' {
                                rows[i][1]++
                                cols[j][1]++
                                if i == j {
                                        left[1]++
                                }
                                if i == n-1-j {
                                        right[1]++
                                }
                        }
                }
        }
        for i := 0; i < n; i++ { // 行列判断
                if rows[i][0] == n || cols[i][0] == n {
                        return "X"
                }
                if rows[i][1] == n || cols[i][1] == n {
                        return "O"
                }
        }
        if left[0] == n || right[0] == n { // 对角线判断
                return "X"
        }
        if left[1] == n || right[1] == n {
                return "O"
        }
        if flag == true {
                return "Pending"
        }
        return "Draw"
}
```

80.65 面试题 16.05. 阶乘尾数 (1)

- 题目

```
设计一个算法，算出 n 阶乘有多少个尾随零。
示例 1:输入: 3 输出: 0 解释: 3! = 6, 尾数中没有零。
示例 2:输入: 5输出: 1 解释: 5! = 120, 尾数中有 1 个零.
说明: 你算法的时间复杂度应为 O(log n) 。
```

- 解题思路

```
// N!有多少个后缀0，即N!有多少个质因数5。
// N!有多少个质因数5，即N可以划分成多少组5个数字一组，
// 加上划分成多少组25个数字一组，加上划分多少组成125个数字一组，等等
// Ans = N/5 + N/(5^2) + N/(5^3) + ...
func trailingZeroes(n int) int {
        result := 0
        for n >= 5 {
                n = n / 5
                result = result + n
        }
        return result
}
```

80.66 面试题 16.06. 最小差 (2)

- 题目

```
给定两个整数数组a和b，计算具有最小差绝对值的一对数值（每个数组中取一个值），并返回该对数值的差
示例：输入：{1, 3, 15, 11, 2}, {23, 127, 235, 19, 8} 输出： 3, 即数值对(11, 8)
提示：
    1 <= a.length, b.length <= 100000
    -2147483648 <= a[i], b[i] <= 2147483647
    正确结果在区间[-2147483648, 2147483647]内
```

- 解题思路

```
func smallestDifference(a []int, b []int) int {
        sort.Ints(a)
        sort.Ints(b)
        i, j := 0, 0
        res := math.MaxInt32
        for i < len(a) && j < len(b) {
```

(续下页)

(接上页)

```
                res = min(res, abs(a[i], b[j]))
                if a[i] > b[j] {
                        j++
                } else {
                        i++
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

# 2
func smallestDifference(a []int, b []int) int {
        sort.Ints(b)
        res := math.MaxInt32
        for i := 0; i < len(a); i++ {
                left, right := 0, len(b)-1
                for left <= right {
                        mid := left + (right-left)/2
                        if b[mid] == a[i] {
                                return 0
                        } else if b[mid] > a[i] {
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                }
                if left < len(b) {
                        res = min(res, abs(a[i], b[left]))
                }
```

(续下页)

(接上页)

```
                if left > 0 {
                        res = min(res, abs(a[i], b[left-1]))
                }
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

80.67 面试题 16.07. 最大数值 (3)

- 题目

```
编写一个方法，找出两个数字a和b中最大的那一个。不得使用if-else或其他比较运算符。
示例：输入： a = 1, b = 2 输出： 2
```

- 解题思路

```
func maximum(a int, b int) int {
        // max(a,b) = (abs(a-b)+a+b)/2
        return (int(math.Abs(float64(a-b))) + a + b) / 2
}

# 2
func maximum(a int, b int) int {
        return int(math.Max(float64(a), float64(b)))
}

# 3
func maximum(a int, b int) int {
```

(续下页)

(接上页)

```
        value := int(uint64(a-b) >> 63) // 取符号位，a-b>0 => 符号位为0 a-b<0 =>
↪符号位为1
        return value*b + int(1^value)*a // value=0=> 0^1=1 1^1=0
}
```

80.68 面试题 16.08. 整数的英语表示 (2)

- 题目

```
给定一个整数，打印该整数的英文描述。
示例 1:输入: 123 输出: "One Hundred Twenty Three"
示例 2:输入: 12345 输出: "Twelve Thousand Three Hundred Forty Five"
示例 3:输入: 1234567
输出: "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
示例 4:输入: 1234567891
输出: "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven
Thousand Eight Hundred Ninety One"
```

- 解题思路

```
func numberToWords(num int) string {
        if num == 0 {
                return "Zero"
        }
        res := ""
        billion := num / 1000000000
        million := (num - billion*1000000000) / 1000000
        thousand := (num - billion*1000000000 - million*1000000) / 1000
        left := num - billion*1000000000 - million*1000000 - thousand*1000
        if billion != 0 {
                res += three(billion) + " Billion"
        }
        if million != 0 {
                if res != "" {
                        res += " "
                }
                res += three(million) + " Million"
        }
        if thousand != 0 {
                if res != "" {
                        res += " "
                }
```

(续下页)

(接上页)

```
                res += three(thousand) + " Thousand"
        }
        if left != 0 {
                if res != "" {
                        res += " "
                }
                res += three(left)
        }
        return res
}

func three(num int) string {
        hundred := num / 100
        left := num - hundred*100
        if hundred == 0 {
                return two(num)
        }
        res := transfer[hundred] + " Hundred"
        if left != 0 {
                res += " " + two(left)
        }
        return res
}

func two(num int) string {
        if num == 0 {
                return ""
        } else if num < 10 {
                return transfer[num]
        } else if num < 20 {
                return transfer[num]
        }
        ten := num / 10
        left := num - ten*10
        ten = ten * 10
        res := transfer[ten]
        if left != 0 {
                res += " " + transfer[left]
        }
        return res
}

var transfer = map[int]string{
```

(续下页)

(接上页)

```
        0:  "Zero",
        1:  "One",
        2:  "Two",
        3:  "Three",
        4:  "Four",
        5:  "Five",
        6:  "Six",
        7:  "Seven",
        8:  "Eight",
        9:  "Nine",
        10: "Ten",
        11: "Eleven",
        12: "Twelve",
        13: "Thirteen",
        14: "Fourteen",
        15: "Fifteen",
        16: "Sixteen",
        17: "Seventeen",
        18: "Eighteen",
        19: "Nineteen",
        20: "Twenty",
        30: "Thirty",
        40: "Forty",
        50: "Fifty",
        60: "Sixty",
        70: "Seventy",
        80: "Eighty",
        90: "Ninety",
}

# 2
func numberToWords(num int) string {
        if num == 0 {
                return "Zero"
        }
        return strings.Trim(dfs(num), " ")
}

func dfs(n int) string {
        if n < 20 {
                return transfer[n]
        }
        if n < 100 {
```

(续下页)

(接上页)

```
                return transfer[n/10*10] + dfs(n%10)
        }
        if n < 1000 {
                return transfer[n/100] + "Hundred " + dfs(n%100)
        }
        if n < 1000000 {
                return dfs(n/1000) + "Thousand " + dfs(n%1000)
        }
        if n < 1000000000 {
                return dfs(n/1000000) + "Million " + dfs(n%1000000)
        }
        return dfs(n/1000000000) + "Billion " + dfs(n%1000000000)
}

var transfer = map[int]string{
        1:  "One ",
        2:  "Two ",
        3:  "Three ",
        4:  "Four ",
        5:  "Five ",
        6:  "Six ",
        7:  "Seven ",
        8:  "Eight ",
        9:  "Nine ",
        10: "Ten ",
        11: "Eleven ",
        12: "Twelve ",
        13: "Thirteen ",
        14: "Fourteen ",
        15: "Fifteen ",
        16: "Sixteen ",
        17: "Seventeen ",
        18: "Eighteen ",
        19: "Nineteen ",
        20: "Twenty ",
        30: "Thirty ",
        40: "Forty ",
        50: "Fifty ",
        60: "Sixty ",
        70: "Seventy ",
        80: "Eighty ",
        90: "Ninety ",
}
```

80.69 面试题 16.10. 生存人数 (2)

- 题目

```
给定N个人的出生年份和死亡年份，第i个人的出生年份为birth[i]，死亡年份为death[i]，
实现一个方法以计算生存人数最多的年份。
你可以假设所有人都出生于1900年至2000年（含1900和2000）之间。
如果一个人在某一年的任意时期都处于生存状态，那么他们应该被纳入那一年的统计中。
例如，生于1908年、死于1909年的人应当被列入1908年和1909年的计数。
如果有多个年份生存人数相同且均为最大值，输出其中最小的年份。
示例：输入：birth = {1900, 1901, 1950} death = {1948, 1951, 2000} 输出： 1901
提示：0 < birth.length == death.length <= 10000
    birth[i] <= death[i]
```

- 解题思路

```
func maxAliveYear(birth []int, death []int) int {
        sort.Ints(birth)
        sort.Ints(death)
        res := birth[0]
        max := 0
        j := 0
        count := 0
        for i := 0; i < len(birth); i++ {
                count++
                for birth[i] > death[j] {
                        count--
                        j++
                }
                if count > max {
                        max = count
                        res = birth[i]
                }
        }
        return res
}

# 2
func maxAliveYear(birth []int, death []int) int {
        arr := make([]int, 102)
        for i := 0; i < len(birth); i++ {
                arr[birth[i]-1900]++
                arr[death[i]-1900+1]--
        }
```

(续下页)

(接上页)

```
        max := 0
        sum := 0
        res := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i]
                if sum > max {
                        max = sum
                        res = i + 1900
                }
        }
        return res
}
```

80.70 面试题 16.11. 跳水板 (2)

- 题目

```
你正在使用一堆木板建造跳水板。有两种类型的木板，其中长度较短的木板长度为shorter，
长度较长的木板长度为longer。你必须正好使用k块木板。编写一个方法，生成跳水板所有可能的长度。
返回的长度需要从小到大排列。
示例 1 输入：shorter = 1 longer = 2 k = 3 输出： [3,4,5,6]
解释：可以使用 3 次 shorter，得到结果 3；使用 2 次 shorter 和 1 次 longer，得到结果 4␣
↪。
以此类推，得到最终结果。
提示：
    0 < shorter <= longer
    0 <= k <= 100000
```

- 解题思路

```
func divingBoard(shorter int, longer int, k int) []int {
        res := make([]int, 0)
        if k == 0 {
                return res
        }
        if shorter == longer {
                return []int{shorter * k}
        }
        for i := 0; i <= k; i++ {
                res = append(res, shorter*(k-i)+longer*i)
        }
        return res
```

(续下页)

(接上页)

```
}

#
func divingBoard(shorter int, longer int, k int) []int {
        res := make([]int, 0)
        if k == 0 {
                return res
        }
        if shorter == longer {
                return []int{shorter * k}
        }
        start := shorter * k
        diff := longer - shorter
        for i := 0; i <= k; i++ {
                res = append(res, start+i*diff)
        }
        return res
}
```

80.71 面试题 16.13. 平分正方形 (1)

- 题目

```
给定两个正方形及一个二维平面。请找出将这两个正方形分割成两半的一条直线。假设正方形顶边和底边与␣
↪x 轴平行。
每个正方形的数据square包含3个数值，正方形的左下顶点坐标[X,Y] = [square[0],
↪square[1]]，以及正方形的边长square[2]。
所求直线穿过两个正方形会形成4个交点，请返回4个交点形成线段的两端点坐标（两个端点即为4个交点中距离最远
这2个点所连成的线段一定会穿过另外2个交点）。2个端点坐标[X1,Y1]和[X2,Y2]的返回格式为
↪{X1,Y1,X2,Y2}，
要求若X1 != X2，需保证X1 < X2，否则需保证Y1 <= Y2。
若同时有多条直线满足要求，则选择斜率最大的一条计算并返回（与Y轴平行的直线视为斜率无穷大）。
示例：输入：square1 = {-1, -1, 2} square2 = {0, -1, 2} 输出： {-1,0,2,0}
解释： 直线 y = 0 能将两个正方形同时分为等面积的两部分，返回的两线段端点为[-1,0]和[2,
↪0]
提示：square.length == 3
square[2] > 0
```

- 解题思路

```
func cutSquares(square1 []int, square2 []int) []float64 {
        // 2个正方形的中点坐标
```

(续下页)

(接上页)

```
        x1, y1, z1 := float64(square1[0])+float64(square1[2])/2,␣
↪float64(square1[1])+float64(square1[2])/2, float64(square1[2])
        x2, y2, z2 := float64(square2[0])+float64(square2[2])/2,␣
↪float64(square2[1])+float64(square2[2])/2, float64(square2[2])
        var a, b, c, d float64
        if x1 == x2 { // 1、垂直
                a, b, c, d = x1, min(float64(square1[1]), float64(square2[1])), x1,␣
↪max(float64(square1[1])+z1, float64(square2[1])+z2)
                return []float64{a, b, c, d}
        }
        // 2、有斜率：y = kx + b1
        k := (y1 - y2) / (x1 - x2)
        b1 := y1 - k*x1
        if abs(k) > 1 { // 斜率大于1，交点通过正方形的上边+下边（根据纵坐标求横坐标）
                b = min(float64(square1[1]), float64(square2[1]))
                d = max(float64(square1[1])+z1, float64(square2[1])+z2)
                a = (b - b1) / k
                c = (d - b1) / k
        } else { // 斜率小于等于1，交点通过正方形的左边+右边（根据横坐标求纵坐标）
                a = min(float64(square1[0]), float64(square2[0]))
                c = max(float64(square1[0])+z1, float64(square2[0])+z2)
                b = a*k + b1
                d = c*k + b1
        }
        if a > c {
                a, c = c, a
                b, d = d, b
        }
        return []float64{a, b, c, d}
}

func abs(a float64) float64 {
        if a < 0 {
                return -a
        }
        return a
}

func max(a, b float64) float64 {
        if a > b {
                return a
        }
        return b
```

(续下页)

(接上页)

```
}

func min(a, b float64) float64 {
        if a > b {
                return b
        }
        return a
}
```

80.72 面试题 16.14. 最佳直线 (2)

- 题目

```
给定一个二维平面及平面上的 N 个点列表Points，其中第i个点的坐标为Points[i]=[Xi,
↪Yi]。请找出一条直线，其通过的点的数目最多。
设穿过最多点的直线所穿过的全部点编号从小到大排序的列表为S，你仅需返回[S[0],
↪S[1]]作为答案，
若有多条直线穿过了相同数量的点，则选择S[0]值较小的直线返回，S[0]相同则选择S[1]值较小的直线返回。
示例：输入： [[0,0],[1,1],[1,0],[2,0]] 输出： [0,2]
解释： 所求直线穿过的3个点的编号为[0,2,3]
提示：2 <= len(Points) <= 300
len(Points[i]) = 2
```

- 解题思路

```
func bestLine(points [][]int) []int {
        res := []int{0, 1}
        maxCount := 0
        n := len(points)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        count := 2
                        x1 := points[i][0] - points[j][0]
                        y1 := points[i][1] - points[j][1]
                        for k := j + 1; k < n; k++ {
                                x2 := points[i][0] - points[k][0]
                                y2 := points[i][1] - points[k][1]
                                if x1*y2 == x2*y1 { // 斜率相同+1
                                        count++
                                }
                        }
                        if count > maxCount {
```

(续下页)

(接上页)

```
                                maxCount = count
                                res[0] = i
                                res[1] = j
                        }
                }
        }
        return res
}

# 2
func bestLine(points [][]int) []int {
        res := []int{0, 1}
        maxCount := 0
        n := len(points)
        m := make(map[[3]int]int)
        mToKey := make(map[[3]int][]int)
        for i := 0; i < n; i++ {
                for j := i + 1; j < n; j++ {
                        // AX+BY+C=0
                        A := points[j][1] - points[i][1]
                        B := points[i][0] - points[j][0]
                        C := points[i][1]*points[j][0] - points[i][0]*points[j][1]
                        com := gcd(gcd(A, B), C)
                        A, B, C = A/com, B/com, C/com
                        node := [3]int{A, B, C}
                        if m[node] == 0 {
                                mToKey[node] = []int{i, j}
                        }
                        m[node]++
                        if m[node] > maxCount {
                                maxCount = m[node]
                                res = mToKey[node]
                        } else if m[node] == maxCount {
                                if mToKey[node][0] < res[0] ||
                                        (mToKey[node][0] == res[0] && mToKey[node][1]
↪< res[1]) {
                                        res = mToKey[node]
                                }
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func gcd(a, b int) int {
        for a != 0 {
                a, b = b%a, a
        }
        return b
}
```

80.73 面试题 16.15. 珠玑妙算 (2)

- 题目

```
珠玑妙算游戏（the game of master mind）的玩法如下。
计算机有4个槽，每个槽放一个球，颜色可能是红色（R）、黄色（Y）、绿色（G）或蓝色（B）。
例如，计算机可能有RGGB 4种（槽1为红色，槽2、3为绿色，槽4为蓝色）。
作为用户，你试图猜出颜色组合。打个比方，你可能会猜YRGB。
要是猜对某个槽的颜色，则算一次“猜中”；要是只猜对颜色但槽位猜错了，则算一次“伪猜中”。
注意，“猜中”不能算入“伪猜中”。
给定一种颜色组合solution和一个猜测guess，
编写一个方法，返回猜中和伪猜中的次数answer，其中answer[0]为猜中的次数，answer[1]为伪猜中的次数。
示例：输入： solution="RGBY",guess="GGRR" 输出： [1,1]
解释： 猜中1次，伪猜中1次。
提示：len(solution) = len(guess) = 4
    solution和guess仅包含"R","G","B","Y"这4种字符
```

- 解题思路

```
func masterMind(solution string, guess string) []int {
        m := make(map[byte]int)
        a, b := 0, 0
        for i := 0; i < len(solution); i++ {
                if solution[i] == guess[i] {
                        a++
                } else {
                        m[solution[i]]++
                }
        }
        for i := 0; i < len(guess); i++ {
                if solution[i] != guess[i] {
                        if m[guess[i]] > 0 {
                                b++
                                m[guess[i]]--
```

(续下页)

(接上页)

```
                        }
                }
        }
        return []int{a, b}
}

# 2
func masterMind(solution string, guess string) []int {
        arr := [256]int{}
        a, b := 0, 0
        for i := 0; i < len(solution); i++ {
                if solution[i] == guess[i] {
                        a++
                } else {
                        arr[solution[i]]++
                }
        }
        for i := 0; i < len(guess); i++ {
                if solution[i] != guess[i] {
                        if arr[guess[i]] > 0 {
                                b++
                                arr[guess[i]]--
                        }
                }
        }
        return []int{a, b}
}
```

80.74 面试题 16.16. 部分排序 (2)

- 题目

```
给定一个整数数组，编写一个函数，找出索引m和n，只要将索引区间[m,
↪n]的元素排好序，整个数组就是有序的。
注意：n-m尽量最小，也就是说，找出符合条件的最短序列。
函数返回值为[m,n]，若不存在这样的m和n（例如整个数组是有序的），请返回[-1,-1]。
示例：输入：  [1,2,4,7,10,11,7,12,6,7,16,18,19] 输出：  [3,9]
提示：0 <= len(array) <= 1000000
```

- 解题思路

```
func subSort(array []int) []int {
        temp := make([]int, len(array))
        copy(temp, array)
        sort.Ints(temp)
        left, right := -1, -1
        for i := 0; i < len(array); i++ {
                if temp[i] != array[i] {
                        left = i
                        break
                }
        }
        for i := len(array) - 1; i >= 0; i-- {
                if temp[i] != array[i] {
                        right = i
                        break
                }
        }
        return []int{left, right}
}

# 2
func subSort(array []int) []int {
        left, right := -1, -1
        maxValue := math.MinInt32
        minValue := math.MaxInt32
        for i := 0; i < len(array); i++ {
                if array[i] >= maxValue {
                        maxValue = array[i]
                } else {
                        right = i
                }
        }
        for i := len(array) - 1; i >= 0; i-- {
                if minValue >= array[i] {
                        minValue = array[i]
                } else {
                        left = i
                }
        }
        return []int{left, right}
}
```

80.75 面试题 16.17. 连续数列 (5)

- 题目

```
给定一个整数数组，找出总和最大的连续数列，并返回总和。
示例：输入： [-2,1,-3,4,-1,2,1,-5,4]输出： 6
解释： 连续子数组 [4,-1,2,1] 的和最大，为 6。
进阶：如果你已经实现复杂度为 O(n) 的解法，尝试使用更为精妙的分治法求解。
```

- 解题思路

```
func maxSubArray(nums []int) int {
        result := nums[0]
        sum := 0
        for i := 0; i < len(nums); i++ {
                if sum > 0 {
                        sum += nums[i]
                } else {
                        sum = nums[i]
                }
                if sum > result {
                        result = sum
                }
        }
        return result
}

# 2
func maxSubArray(nums []int) int {
        result := math.MinInt32
        for i := 0; i < len(nums); i++ {
                sum := 0
                for j := i; j < len(nums); j++ {
                        sum += nums[j]
                        if sum > result {
                                result = sum
                        }
                }
        }
        return result
}

# 3
// dp[i] = max(dp[i-1]+nums[i], nums[i])
```

(续下页)

(接上页)

```
// res = max(dp[i], res)
func maxSubArray(nums []int) int {
        dp := make([]int, len(nums))
        dp[0] = nums[0]
        result := nums[0]
        for i := 1; i < len(nums); i++ {
                if dp[i-1]+nums[i] > nums[i] {
                        dp[i] = dp[i-1] + nums[i]
                } else {
                        dp[i] = nums[i]
                }
                if dp[i] > result {
                        result = dp[i]
                }
        }
        return result
}

# 4
func maxSubArray(nums []int) int {
        dp := nums[0]
        result := dp
        for i := 1; i < len(nums); i++ {
                if dp+nums[i] > nums[i] {
                        dp = dp + nums[i]
                } else {
                        dp = nums[i]
                }

                if dp > result {
                        result = dp
                }
        }
        return result
}

# 5
func maxSubArray(nums []int) int {
        result := maxSubArr(nums, 0, len(nums)-1)
        return result
}

func maxSubArr(nums []int, left, right int) int {
```

(续下页)

(接上页)

```
        if left == right {
                return nums[left]
        }

        mid := (left + right) / 2
        leftSum := maxSubArr(nums, left, mid)        // 最大子序在左边
        rightSum := maxSubArr(nums, mid+1, right)    // 最大子序在右边
        midSum := findMaxArr(nums, left, mid, right) // 跨中心
        result := max(leftSum, rightSum)
        result = max(result, midSum)
        return result
}

func findMaxArr(nums []int, left, mid, right int) int {
        leftSum := math.MinInt32
        sum := 0
        // 从右到左
        for i := mid; i >= left; i-- {
                sum += nums[i]
                leftSum = max(leftSum, sum)
        }
        rightSum := math.MinInt32
        sum = 0
        // 从左到右
        for i := mid + 1; i <= right; i++ {
                sum += nums[i]
                rightSum = max(rightSum, sum)
        }
        return leftSum + rightSum
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

80.76 面试题 16.18. 模式匹配 (1)

- 题目

```
你有两个字符串，即pattern和value。 pattern字符串由字母"a"和"b
↪"组成，用于描述字符串中的模式。
例如，字符串"catcatgocatgo"匹配模式"aabab"（其中"cat"是"a"，"go"是"b"），
该字符串也匹配像"a"、"ab"和"b"这样的模式。
但需注意"a"和"b
↪"不能同时表示相同的字符串。编写一个方法判断value字符串是否匹配pattern字符串。
示例 1: 输入： pattern = "abba", value = "dogcatcatdog" 输出： true
示例 2: 输入： pattern = "abba", value = "dogcatcatfish" 输出： false
示例 3: 输入： pattern = "aaaa", value = "dogcatcatdog" 输出： false
示例 4: 输入： pattern = "abba", value = "dogdogdogdog" 输出： true
解释： "a"="dogdog",b="", 反之也符合规则
提示：1 <= len(pattern) <= 1000
0 <= len(value) <= 1000
你可以假设pattern只包含字母"a"和"b", value仅包含小写字母。
```

- 解题思路

```go
func patternMatching(pattern string, value string) bool {
        countA := 0
        for i := 0; i < len(pattern); i++ {
                if pattern[i] == 'a' {
                        countA++
                }
        }
        countB := len(pattern) - countA
        if value == "" {
                return countA == 0 || countB == 0
        }
        if countA < countB { // 令a>b
                countA, countB = countB, countA
                str := ""
                for i := 0; i < len(pattern); i++ {
                        if pattern[i] == 'a' {
                                str = str + "b"
                        } else {
                                str = str + "a"
                        }
                }
                pattern = str
        }
```

(续下页)

(接上页)

```
        for a := 0; a <= len(value)/countA; a++ { // 枚举
                if judge(pattern, value, a, countA, countB) {
                        return true
                }
        }
        return false
}

func judge(pattern string, value string, a, countA, countB int) bool {
        left := len(value) - a*countA
        if (countB == 0 && left == 0) || (countB > 0 && left%countB == 0) {
                var b int
                if countB > 0 {
                        b = left / countB
                }
                var strA, strB string
                index := 0
                flag := true
                for i := 0; i < len(pattern); i++ {
                        if pattern[i] == 'a' {
                                str := value[index : index+a]
                                if strA == "" {
                                        strA = str
                                } else if str != strA {
                                        flag = false
                                        break
                                }
                                index = index + a
                        } else {
                                str := value[index : index+b]
                                if strB == "" {
                                        strB = str
                                } else if str != strB {
                                        flag = false
                                        break
                                }
                                index = index + b
                        }
                }
                if flag == true && strA != strB {
                        return true
                }
        }
```

(续下页)

(接上页)

```
        return false
}
```

80.77 面试题 16.19. 水域大小 (2)

- 题目

```
你有一个用于表示一片土地的整数矩阵land，该矩阵中每个点的值代表对应地点的海拔高度。
若值为0则表示水域。由垂直、水平或对角连接的水域为池塘。池塘的大小是指相连接的水域的个数。
编写一个方法来计算矩阵中所有池塘的大小，返回值需要从小到大排序。
示例：输入：
[
  [0,2,1,0],
  [0,1,0,1],
  [1,1,0,1],
  [0,1,0,1]
]
输出：  [1,2,4]
提示：
    0 < len(land) <= 1000
    0 < len(land[i]) <= 1000
```

- 解题思路

```
func pondSizes(land [][]int) []int {
        res := make([]int, 0)
        for i := range land {
                for j := range land[i] {
                        if land[i][j] == 0 {
                                res = append(res, getArea(land, i, j))
                        }
                }
        }
        sort.Ints(res)
        return res
}

func getArea(grid [][]int, i, j int) int {
        if grid[i][j] != 0 {
                return 0
        }
        grid[i][j] = 1
```

(续下页)

(接上页)

```
        area := 1
        for a := i - 1; a <= i+1; a++ {
                for b := j - 1; b <= j+1; b++ {
                        if (i == a && j == b) || a < 0 || a >= len(grid) ||
                                b < 0 || b >= len(grid[0]) {
                                continue
                        }
                        area = area + getArea(grid, a, b)
                }
        }
        return area
}

# 2
func pondSizes(land [][]int) []int {
        res := make([]int, 0)
        for i := range land {
                for j := range land[i] {
                        if land[i][j] == 0 {
                                res = append(res, getArea(land, i, j))
                        }
                }
        }
        sort.Ints(res)
        return res
}

func getArea(grid [][]int, i, j int) int {
        if i < 0 || i >= len(grid) ||
                j < 0 || j >= len(grid[0]) || grid[i][j] != 0 {
                return 0
        }

        grid[i][j] = 1
        res := 1
        res = res + getArea(grid, i+1, j)
        res = res + getArea(grid, i+1, j+1)
        res = res + getArea(grid, i+1, j-1)
        res = res + getArea(grid, i-1, j)
        res = res + getArea(grid, i-1, j+1)
        res = res + getArea(grid, i-1, j-1)
        res = res + getArea(grid, i, j+1)
        res = res + getArea(grid, i, j-1)
```

(续下页)

(接上页)

```
        return res
}
```

80.78 面试题 16.20.T9 键盘 (1)

- 题目

```
在老式手机上，用户通过数字键盘输入，手机将提供与这些数字相匹配的单词列表。每个数字映射到0至4个字母。
给定一个数字序列，实现一个算法来返回匹配单词的列表。你会得到一张含有有效单词的列表。映射如下图所示：
示例 1:输入: num = "8733", words = ["tree", "used"] 输出: ["tree", "used"]
示例 2:输入: num = "2", words = ["a", "b", "c", "d"] 输出: ["a", "b", "c"]
提示：num.length <= 1000
    words.length <= 500
    words[i].length == num.length
    num中不会出现 0, 1 这两个数字
```

- 解题思路

```
var m [26]byte = [26]byte{
        '2', '2', '2',
        '3', '3', '3',
        '4', '4', '4',
        '5', '5', '5',
        '6', '6', '6',
        '7', '7', '7', '7',
        '8', '8', '8',
        '9', '9', '9', '9',
}

func getValidT9Words(num string, words []string) []string {
        res := make([]string, 0)
        for _, str := range words {
                if len(str) != len(num) {
                        continue
                }
                flag := true
                for i := 0; i < len(str); i++ {
                        if num[i] != m[str[i]-'a'] {
                                flag = false
                                break
                        }
                }
```

(续下页)

(接上页)

```
                if flag {
                        res = append(res, str)
                }
        }
        return res
}
```

80.79 面试题 16.21. 交换和 (1)

- 题目

```
给定两个整数数组，请交换一对数值（每个数组中取一个数值），使得两个数组所有元素的和相等。
返回一个数组，第一个元素是第一个数组中要交换的元素，第二个元素是第二个数组中要交换的元素。
若有多个答案，返回任意一个均可。若无满足条件的数值，返回空数组。
示例:输入: array1 = [4, 1, 2, 1, 1, 2], array2 = [3, 6, 3, 3] 输出: [1, 3]
示例:输入: array1 = [1, 2, 3], array2 = [4, 5, 6] 输出: []
提示：1 <= array1.length, array2.length <= 100000
```

- 解题思路

```
func findSwapValues(array1 []int, array2 []int) []int {
        m := make(map[int]bool)
        sumA, sumB := 0, 0
        for i := 0; i < len(array1); i++ {
                sumA = sumA + array1[i]
                m[array1[i]] = true
        }
        for i := 0; i < len(array2); i++ {
                sumB = sumB + array2[i]
        }
        if (sumA+sumB)%2 == 1 {
                return nil
        }
        half := (sumA - sumB) / 2
        a, b := 0, 0
        // sumA-A[i]+B[j] == sumB-B[j]+A[i]
        // sumA-sumB=2(A[i]-B[j])
        // (sumA-sumB)/2 = A[i]-B[j]
        for _, b = range array2 {
                a = b + half
                if m[a] == true {
                        return []int{a, b}
```

(续下页)

(接上页)

```
                }
        }
        return nil
}
```

80.80 面试题 16.22. 兰顿蚂蚁 (1)

- 题目

```
一只蚂蚁坐在由白色和黑色方格构成的无限网格上。开始时，网格全白，蚂蚁面向右侧。
每行走一步，蚂蚁执行以下操作。
(1) 如果在白色方格上，则翻转方格的颜色，向右(顺时针)转 90 度，并向前移动一个单位。
(2) 如果在黑色方格上，则翻转方格的颜色，向左(逆时针方向)转 90 度，并向前移动一个单位。
编写程序来模拟蚂蚁执行的前 K 个动作，并返回最终的网格。
网格由数组表示，每个元素是一个字符串，代表网格中的一行，黑色方格由'X'表示，白色方格由
↪'_'表示，
蚂蚁所在的位置由'L', 'U', 'R', 'D'表示，分别表示蚂蚁左、上、右、下 的朝向。
只需要返回能够包含蚂蚁走过的所有方格的最小矩形。
示例 1:输入: 0 输出: ["R"]
示例 2:输入: 2 输出:
[
 "_X",
 "LX"
]
示例 3:输入: 5 输出:
[
 "_U",
 "X_",
 "XX"
]
说明: K <= 100000
```

- 解题思路

```
func printKMoves(K int) []string {
        var dirArr = []byte{'R', 'D', 'L', 'U'}
        var dx = []int{1, 0, -1, 0}
        var dy = []int{0, -1, 0, 1}
        dir := 0 // 向右
        x, y := 0, 0
        left, right := 0, 0
        up, down := 0, 0
```

(续下页)

(接上页)

```
        m := make(map[[2]int]int) // 1黑色，0白色
        for i := 0; i < K; i++ {
                if m[[2]int{x, y}] == 1 { // 变方向
                        dir = (dir + 3) % 4 // 逆时针
                } else {
                        dir = (dir + 1) % 4 // 顺时针
                }
                m[[2]int{x, y}] = 1 - m[[2]int{x, y}]
                x = x + dx[dir]
                y = y + dy[dir]
                left = min(left, x)
                right = max(right, x)
                down = min(down, y)
                up = max(up, y)
        }
        w := right - left + 1
        h := up - down + 1
        res := make([]string, 0)
        for i := 0; i < h; i++ {
                arr := make([]byte, w)
                for j := 0; j < w; j++ {
                        newX := j + left
                        newY := up - i
                        arr[j] = '_'
                        if v, ok := m[[2]int{newX, newY}]; ok && v == 1 {
                                arr[j] = 'X'
                        }
                        if newX == x && newY == y {
                                arr[j] = dirArr[dir]
                        }
                }
                res = append(res, string(arr))
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

(续下页)

(接上页)

```
func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

80.81 面试题 16.24. 数对和 (2)

- 题目

```
设计一个算法，找出数组中两数之和为指定值的所有整数对。一个数只能属于一个数对。
示例 1:输入: nums = [5,6,5], target = 11 输出: [[5,6]]
示例 2:输入: nums = [5,6,5,6], target = 11 输出: [[5,6],[5,6]]
提示：nums.length <= 100000
```

- 解题思路

```
func pairSums(nums []int, target int) [][]int {
        res := make([][]int, 0)
        sort.Ints(nums)
        left, right := 0, len(nums)-1
        for left < right {
                sum := nums[left] + nums[right]
                if target == sum {
                        res = append(res, []int{nums[left], nums[right]})
                        left++
                        right--
                } else if target > sum {
                        left++
                } else {
                        right--
                }
        }
        return res
}

# 2
func pairSums(nums []int, target int) [][]int {
        res := make([][]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(nums); i++ {
```

(续下页)

(接上页)

```
                x := target - nums[i]
                if m[x] > 0 {
                        res = append(res, []int{nums[i], x})
                        m[x]--
                        continue
                }
                m[nums[i]]++
        }
        return res
}
```

80.82 面试题 16.25.LRU 缓存 (1)

- 题目

```
设计和构建一个“最近最少使用”缓存，该缓存会删除最近最少使用的项目。
缓存应该从键映射到值(允许你插入和检索特定键对应的值)，并在初始化时指定最大容量。
当缓存被填满时，它应该删除最近最少使用的项目。
它应该支持以下操作： 获取数据 get 和 写入数据 put 。
获取数据 get(key) - 如果密钥 (key) 存在于缓存中，则获取密钥的值（总是正数），否则返回␣
↪-1。
写入数据 put(key, value) - 如果密钥不存在，则写入其数据值。
当缓存容量达到上限时，它应该在写入新数据之前删除最近最少使用的数据值，从而为新的数据值留出空间。
示例:LRUCache cache = new LRUCache( 2 /* 缓存容量 */ );
cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // 返回  1
cache.put(3, 3);    // 该操作会使得密钥 2 作废
cache.get(2);       // 返回 -1 (未找到)
cache.put(4, 4);    // 该操作会使得密钥 1 作废
cache.get(1);       // 返回 -1 (未找到)
cache.get(3);       // 返回  3
cache.get(4);       // 返回  4
```

- 解题思路

```
type Node struct {
        key   int
        value int
        prev  *Node
        next  *Node
}
```

(续下页)

(接上页)

```
type LRUCache struct {
        cap    int
        header *Node
        tail   *Node
        m      map[int]*Node
}

func Constructor(capacity int) LRUCache {
        cache := LRUCache{
                cap:    capacity,
                header: &Node{},
                tail:   &Node{},
                m:      make(map[int]*Node, capacity),
        }
        cache.header.next = cache.tail
        cache.tail.prev = cache.header
        return cache
}

func (this *LRUCache) Get(key int) int {
        if node, ok := this.m[key]; ok {
                this.remove(node)
                this.putHead(node)
                return node.value
        }
        return -1
}

func (this *LRUCache) Put(key int, value int) {
        if node, ok := this.m[key]; ok {
                node.value = value
                this.remove(node)
                this.putHead(node)
                return
        }
        if this.cap <= len(this.m) {
                // 删除尾部
                deleteKey := this.tail.prev.key
                this.remove(this.tail.prev)
                delete(this.m, deleteKey)
        }
        // 插入到头部
```

(续下页)

(接上页)

```
        newNode := &Node{key: key, value: value}
        this.putHead(newNode)
        this.m[key] = newNode
}

// 删除尾部节点
func (this *LRUCache) remove(node *Node) {
        node.prev.next = node.next
        node.next.prev = node.prev
}

// 插入头部
func (this *LRUCache) putHead(node *Node) {
        next := this.header.next
        this.header.next = node
        node.next = next
        next.prev = node
        node.prev = this.header
}
```

80.83 面试题 16.26. 计算器 (2)

- 题目

```
给定一个包含正整数、加(+)、减(-)、乘(*)、除(/)的算数表达式(括号除外)，计算其结果。
表达式仅包含非负整数，+， - ，*，/ 四种运算符和空格  。 整数除法仅保留整数部分。
示例 1:输入: "3+2*2" 输出: 7
示例 2:输入: " 3/2 " 输出: 1
示例 3:输入: " 3+5 / 2 " 输出: 5
说明: 你可以假设所给定的表达式都是有效的。
    请不要使用内置的库函数 eval。
```

- 解题思路

```
func calculate(s string) int {
        stack := make([]int, 0)
        op := make([]int, 0)
        num := 0
        for i := 0; i < len(s); i++ {
                if '0' <= s[i] && s[i] <= '9' {
                        num = 0
                        for i < len(s) && '0' <= s[i] && s[i] <= '9' {
```

(续下页)

(接上页)

```
                                num = num*10 + int(s[i]-'0')
                                i++
                        }
                        // 处理乘除计算
                        if len(op) > 0 && op[len(op)-1] > 1 {
                                if op[len(op)-1] == 2 {
                                        stack[len(stack)-1] = stack[len(stack)-1] *␣
↪num
                                } else {
                                        stack[len(stack)-1] = stack[len(stack)-1] /␣
↪num
                                }
                                op = op[:len(op)-1]
                        } else {
                                stack = append(stack, num)
                        }
                        i--
                } else if s[i] == '+' {
                        op = append(op, 1)
                } else if s[i] == '-' {
                        op = append(op, -1)
                } else if s[i] == '*' {
                        op = append(op, 2)
                } else if s[i] == '/' {
                        op = append(op, 3)
                }
        }
        // 处理加减
        for len(op) > 0 {
                stack[1] = stack[0] + stack[1]*op[0]
                stack = stack[1:]
                op = op[1:]
        }
        return stack[0]
}

# 2
func calculate(s string) int {
        s = strings.Trim(s, " ") // 避免"3/2 "的情况
        stack := make([]int, 0)
        num := 0
        sign := byte('+')
        for i := 0; i < len(s); i++ {
```

(续下页)

(接上页)

```
                if s[i] == ' ' {
                        continue
                }
                if '0' <= s[i] && s[i] <= '9' {
                        num = num*10 + int(s[i]-'0')
                }
                if s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' || i ==␣
↪len(s)-1 {
                        // 处理前一个符号
                        switch sign {
                        case '+':
                                stack = append(stack, num)
                        case '-':
                                stack = append(stack, -num)
                        case '*':
                                prev := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                stack = append(stack, num*prev)
                        case '/':
                                prev := stack[len(stack)-1]
                                stack = stack[:len(stack)-1]
                                stack = append(stack, prev/num)
                        }
                        num = 0
                        sign = s[i]
                }
        }
        res := 0
        for i := 0; i < len(stack); i++ {
                res = res + stack[i]
        }
        return res
}
```

80.84 面试题 17.01. 不用加号的加法 (2)

- 题目

```
设计一个函数把两个数字相加。不得使用 + 或者其他算术运算符。
示例:输入: a = 1, b = 1 输出: 2
提示:  a, b 均可能是负数或 0
    结果不会溢出 32 位整数
```

- 解题思路

```
/*
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 (进位 1)
异或的一个重要特性是无进位加法
(a 和 b 的无进位结果) + (a 和 b 的进位结果)
*/
func add(a int, b int) int {
        for b != 0 {
                a, b = a^b, (a&b)<<1
        }
        return a
}

#
func add(a int, b int) int {
        if b == 0 {
                return a
        }
        return add(a^b, (a&b)<<1)
}
```

80.85 面试题 17.04. 消失的数字 (5)

- 题目

```
数组nums包含从0到n的所有整数，但其中缺了一个。请编写代码找出那个缺失的整数。你有办法在O(n)时间内完成
注意：本题相对书上原题稍作改动
示例 1：输入：[3,0,1]输出：2
示例 2：输入：[9,6,4,2,3,5,7,0,1] 输出：8
```

- 解题思路

```
func missingNumber(nums []int) int {
        n := len(nums)
        sum := n * (n + 1) / 2
        for i := 0; i < n; i++ {
                sum = sum - nums[i]
        }
        return sum
```

(续下页)

(接上页)

```
}

# 2
func missingNumber(nums []int) int {
        sort.Ints(nums)
        for i := 0; i < len(nums); i++ {
                if nums[i] != i {
                        return i
                }
        }
        return len(nums)
}

# 3
func missingNumber(nums []int) int {
        res := 0
        for i := 0; i < len(nums); i++ {
                res = res ^ (i+1) ^ nums[i]
        }
        return res
}

# 4
func missingNumber(nums []int) int {
        n := len(nums)
        index := n
        for i := 0; i < n; {
                if nums[i] == n{
                        index = i
                        i++
                        continue
                }
                if i == nums[i]{
                        i++
                        continue
                }
                nums[i], nums[nums[i]] = nums[nums[i]], nums[i]
        }
        return index
}

# 5
func missingNumber(nums []int) int {
```

(续下页)

(接上页)

```
        m := make(map[int]bool)
        for i := range nums {
                m[nums[i]] = true
        }
        for i := 0; i <= len(nums); i++ {
                if m[i] == false {
                        return i
                }
        }
        return 0
}
```

80.86 面试题 17.05. 字母与数字 (1)

- 题目

```
给定一个放有字符和数字的数组，找到最长的子数组，且包含的字符和数字的个数相同。
返回该子数组，若存在多个最长子数组，返回左端点最小的。若不存在这样的数组，返回一个空数组。
示例 1:
输入: ["A","1","B","C","D","2","3","4","E","5","F","G","6","7","H","I","J","K","L","M
↪"]
输出: ["A","1","B","C","D","2","3","4","E","5","F","G","6","7"]
示例 2:输入: ["A","A"]输出: []
提示：array.length <= 100000
```

- 解题思路

```
func findLongestSubarray(array []string) []string {
        m := make(map[int]int)
        m[0] = 0
        res := 0
        begin := 0
        total := 0
        for i := 0; i < len(array); i++ {
                if '0' <= array[i][0] && array[i][0] <= '9' {
                        total++
                } else {
                        total--
                }
                if total == 0 {
                        begin = 0
                        res = i + 1
```

(续下页)

(接上页)

```
                } else if index, ok := m[total]; ok {
                        if i-index > res {
                                res = i - index
                                begin = index + 1
                        }
                } else {
                        m[total] = i
                }
        }
        return array[begin : begin+res]
}
```

80.87 面试题 17.06.2 出现的次数 (3)

- 题目

```
编写一个方法，计算从 0 到 n (含 n) 中数字 2 出现的次数。
示例:输入: 25 输出: 9
解释: (2, 12, 20, 21, 22, 23, 24, 25)(注意 22 应该算作两次)
提示：n <= 10^9
```

- 解题思路

```
func numberOf2sInRange(n int) int {
        if n <= 0 {
                return 0
        }
        res := 0
        for i := 1; i <= n; i = i * 10 {
                left := n / i
                right := n % i
                res = res + (left+7)/10*i
                if left%10 == 2 {
                        res = res + right + 1
                }
        }
        return res
}

# 2
func numberOf2sInRange(n int) int {
        res := 0
```

(续下页)

(接上页)

```
        digit := 1
        high := n / 10
        cur := n % 10
        low := 0
        for high != 0 || cur != 0 {
                if cur > 2 {
                        res = res + (high+1)*digit
                } else if cur == 2 {
                        res = res + high*digit + low + 1
                } else {
                        res = res + high*digit
                }
                low = low + cur*digit
                cur = high % 10
                high = high / 10
                digit = digit * 10
        }
        return res
}

# 3
func numberOf2sInRange(n int) int {
        if n <= 0 {
                return 0
        }
        str := strconv.Itoa(n)
        return dfs(str)
}

func dfs(str string) int {
        if str == "" {
                return 0
        }
        first := int(str[0] - '0')
        if len(str) == 1 && first == 0 {
                return 0
        }
        if len(str) == 1 && first >= 2 {
                return 1
        }
        count := 0
        if first > 2 {
                count = int(math.Pow(float64(10), float64(len(str)-1)))
```

(续下页)

(接上页)

```
        } else if first == 2 {
                count, _ = strconv.Atoi(str[1:])
                count = count + 1
        }
        other := first * (len(str) - 1) * int(math.Pow(float64(10), float64(len(str)-
↪2)))
        numLeft := dfs(str[1:])
        return count + numLeft + other
}
```

80.88 面试题 17.07. 婴儿名字 (1)

- 题目

```
每年，政府都会公布一万个最常见的婴儿名字和它们出现的频率，也就是同名婴儿的数量。
有些名字有多种拼法，例如，John 和 Jon 本质上是相同的名字，但被当成了两个名字公布出来。
给定两个列表，一个是名字及对应的频率，另一个是本质相同的名字对。设计一个算法打印出每个真实名字的实际
注意，如果 John 和 Jon 是相同的，并且 Jon 和 Johnny 相同，
则 John 与 Johnny 也相同，即它们有传递和对称性。
在结果列表中，选择 字典序最小 的名字作为真实名字。
示例：输入：names = ["John(15)","Jon(12)","Chris(13)","Kris(4)","Christopher(19)"],
synonyms = ["(Jon,John)","(John,Johnny)","(Chris,Kris)","(Chris,Christopher)"]
输出：["John(27)","Chris(36)"]
提示：names.length <= 100000
```

- 解题思路

```
func trulyMostPopular(names []string, synonyms []string) []string {
        res := make([]string, 0)
        node = Node{}
        nameArr := make([]string, 0)
        countArr := make([]int, 0)
        m := make(map[string]int)
        for i := 0; i < len(names); i++ {
                arr := strings.Split(names[i], "(")
                nameArr = append(nameArr, arr[0])
                tempArr := strings.Split(arr[1], ")")
                count, _ := strconv.Atoi(tempArr[0])
                countArr = append(countArr, count)
                m[arr[0]] = i
        }
        Init(nameArr, countArr)
```

(续下页)

(接上页)

```
        for i := 0; i < len(synonyms); i++ {
                str := strings.TrimLeft(synonyms[i], "(")
                str = strings.TrimRight(str, ")")
                arr := strings.Split(str, ",")
                a := m[arr[0]]
                b := m[arr[1]]
                union(a, b)
        }
        for i := 0; i < len(node.fa); i++ {
                if node.fa[i] < 0 {
                        temp := node.names[i] + "(" + strconv.Itoa(node.count[i]) + ")
↪"
                        res = append(res, temp)
                }
        }
        return res
}

var node Node

type Node struct {
        fa    []int
        names []string
        count []int
}

// 初始化
func Init(names []string, count []int) {
        node.fa = make([]int, len(names))
        for i := 0; i < len(names); i++ {
                node.fa[i] = -1
        }
        node.names = names
        node.count = count
}

// 查询
func find(x int) int {
        if node.fa[x] < 0 {
                return x
        }
        res := find(node.fa[x])
```

(续下页)

(接上页)

```
        node.fa[x] = res
        return res
}

// 合并
func union(i, j int) {
        x, y := find(i), find(j)
        if x == y {
                return
        }
        if node.names[x] <= node.names[y] {
                node.fa[y] = x
                node.count[x] = node.count[x] + node.count[y]
        } else {
                node.fa[x] = y
                node.count[y] = node.count[y] + node.count[x]
        }
}
```

80.89 面试题 17.08. 马戏团人塔 (2)

- 题目

```
有个马戏团正在设计叠罗汉的表演节目，一个人要站在另一人的肩膀上。
出于实际和美观的考虑，在上面的人要比下面的人矮一点且轻一点。
已知马戏团每个人的身高和体重，请编写代码计算叠罗汉最多能叠几个人。
示例：输入：height = [65,70,56,75,60,68] weight = [100,150,90,190,95,110] 输出：6
解释：从上往下数，叠罗汉最多能叠 6 层：
(56,90), (60,95),(65,100), (68,110), (70,150), (75,190)
提示：  height.length == weight.length <= 10000
```

- 解题思路

```
func bestSeqAtIndex(height []int, weight []int) int {
        arr := make([][2]int, 0)
        for i := 0; i < len(height); i++ {
                arr = append(arr, [2]int{height[i], weight[i]})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][0] == arr[j][0] {
                        return arr[i][1] < arr[j][1]
                }
```

(续下页)

(接上页)

```
                return arr[i][0] > arr[j][0]
        })
        res := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                if len(res) == 0 || arr[res[len(res)-1]][0] > arr[i][0] &&
                        arr[res[len(res)-1]][1] > arr[i][1] {
                        res = append(res, i)
                } else {
                        left := 0
                        right := len(res) - 1
                        for left <= right {
                                mid := left + (right-left)/2
                                if arr[res[mid]][0] > arr[i][0] && arr[res[mid]][1] >␣
↪arr[i][1] {
                                        left = mid + 1
                                } else {
                                        right = mid - 1
                                }
                        }
                        res[left] = i
                }
        }
        return len(res)
}

# 2
func bestSeqAtIndex(height []int, weight []int) int {
        arr := make([][2]int, 0)
        for i := 0; i < len(height); i++ {
                arr = append(arr, [2]int{height[i], weight[i]})
        }
        sort.Slice(arr, func(i, j int) bool {
                if arr[i][0] == arr[j][0] {
                        return arr[i][1] < arr[j][1]
                }
                return arr[i][0] > arr[j][0]
        })
        res := make([]int, 0)
        for i := 0; i < len(arr); i++ {
                index := sort.Search(len(res), func(j int) bool {
                        return arr[res[j]][0] <= arr[i][0] || arr[res[j]][1] <=␣
↪arr[i][1]
                })
```

(续下页)

(接上页)

```
                if index == len(res) {
                        res = append(res, i)
                } else {
                        res[index] = i
                }
        }
        return len(res)
}
```

80.90 面试题 17.09. 第 k 个数 (1)

- 题目

```
有些数的素因子只有 3，5，7，请设计一个算法找出第 k 个数。
注意，不是必须有这些素因子，而是必须不包含其他的素因子。
例如，前几个数按顺序应该是 1，3，5，7，9，15，21。
示例 1:输入: k = 5输出: 9
```

- 解题思路

```
func getKthMagicNumber(k int) int {
        dp := make([]int, k)
        dp[0] = 1
        // *3或5或7之后得到
        idx3, idx5, idx7 := 0, 0, 0
        for i := 1; i < k; i++ {
                dp[i] = min(dp[idx3]*3, min(dp[idx5]*5, dp[idx7]*7))
                if dp[i] == dp[idx3]*3 {
                        idx3++
                }
                if dp[i] == dp[idx5]*5 {
                        idx5++
                }
                if dp[i] == dp[idx7]*7 {
                        idx7++
                }
        }
        return dp[k-1]
}

func min(a, b int) int {
        if a > b {
```

(续下页)

(接上页)

```
                return b
        }
        return a
}
```

80.91 面试题 17.10. 主要元素 (5)

- 题目

```
数组中占比超过一半的元素称之为主要元素。给定一个整数数组，找到它的主要元素。若没有，返回-
↪1。
示例 1: 输入: [1,2,5,9,5,9,5,5,5]输出: 5
示例 2: 输入: [3,2]输出: -1
示例 3: 输入: [2,2,1,1,1,2,2]输出: 2
说明: 你有办法在时间复杂度为 O(N)，空间复杂度为 O(1) 内完成吗？
```

- 解题思路

```
func majorityElement(nums []int) int {
        m := make(map[int]int)
        result := -1
        for _, v := range nums{
                if _,ok := m[v];ok{
                        m[v]++
                }else {
                        m[v]=1
                }
                if m[v] > (len(nums)/2){
                        result = v
                }
        }
        return result
}

# 2
func majorityElement(nums []int) int {
        result, count := 0, 0
        for i := 0; i < len(nums); i++ {
                if count == 0 {
                        result = nums[i]
                        count++
                } else if result == nums[i] {
```

(续下页)

(接上页)

```
                        count++
                } else {
                        count--
                }
        }
        total := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == result {
                        total++
                }
        }
        if total <= len(nums)/2 {
                return -1
        }
        return result
}

# 3
func majorityElement(nums []int) int {
        sort.Ints(nums)
        for i := 0; i <= len(nums)/2; i++ {
                if nums[i] == nums[i+len(nums)/2] {
                        return nums[i]
                }
        }
        return -1
}

# 4
func majorityElement(nums []int) int {
        if len(nums) == 1 {
                return nums[0]
        }
        result := int32(0)
        mask := int32(1)
        for i := 0; i < 32; i++ {
                count := 0
                for j := 0; j < len(nums); j++ {
                        if mask&int32(nums[j]) == mask {
                                count++
                        }
                }
                if count > len(nums)/2 {
```

(续下页)

(接上页)

```
                        result = result | mask
                }
                mask = mask << 1
        }
        total := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == int(result) {
                        total++
                }
        }
        if total <= len(nums)/2 {
                return -1
        }
        return int(result)
}

# 5
func majorityElement(nums []int) int {
        res := majority(nums, 0, len(nums)-1)
        total := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] == res {
                        total++
                }
        }
        if total <= len(nums)/2 {
                return -1
        }
        return res
}

func count(nums []int, target int, start int, end int) int {
        countNum := 0
        for i := start; i <= end; i++ {
                if nums[i] == target {
                        countNum++
                }
        }
        return countNum
}

func majority(nums []int, start, end int) int {
        if start == end {
```

(续下页)

(接上页)

```
                return nums[start]
        }
        mid := (start + end) / 2
        left := majority(nums, start, mid)
        right := majority(nums, mid+1, end)
        if left == right {
                return left
        }
        leftCount := count(nums, left, start, end)
        rightCount := count(nums, right, start, end)
        if leftCount > rightCount {
                return left
        }
        return right
}
```

80.92 面试题 17.11. 单词距离 (2)

- 题目

```
有个内含单词的超大文本文件，给定任意两个单词，找出在这个文件中这两个单词的最短距离(相隔单词数)。
如果寻找过程在这个文件中会重复多次，而每次寻找的单词不同，你能对此优化吗?
示例：输入：words = ["I","am","a","student","from","a","university","in","a","city"],
word1 = "a", word2 = "student"
输出：1
提示：words.length <= 100000
```

- 解题思路

```
func findClosest(words []string, word1 string, word2 string) int {
        res := len(words) - 1
        a, b := -1, -1
        for i := 0; i < len(words); i++ {
                if words[i] == word1 {
                        a = i
                }
                if words[i] == word2 {
                        b = i
                }
                if a != -1 && b != -1 && abs(a, b) < res {
                        res = abs(a, b)
                }
```

(续下页)

(接上页)

```
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}

# 2
func findClosest(words []string, word1 string, word2 string) int {
        res := len(words) - 1
        arrA, arrB := make([]int, 0), make([]int, 0)
        for i := 0; i < len(words); i++ {
                if words[i] == word1 {
                        arrA = append(arrA, i)
                }
                if words[i] == word2 {
                        arrB = append(arrB, i)
                }
        }
        i, j := 0, 0
        for i < len(arrA) && j < len(arrB) {
                if abs(arrA[i], arrB[j]) < res {
                        res = abs(arrA[i], arrB[j])
                }
                if arrA[i] < arrB[j] {
                        i++
                } else {
                        j++
                }
        }
        return res
}

func abs(a, b int) int {
        if a > b {
                return a - b
        }
        return b - a
}
```

80.93 面试题 17.12.BiNode(2)

- 题目

```
二叉树数据结构TreeNode可用来表示单向链表（其中left置空，right为下一个链表节点）。
实现一个方法，把二叉搜索树转换为单向链表，要求依然符合二叉搜索树的性质，转换操作应是原址的，
也就是在原始的二叉搜索树上直接修改。
返回转换后的单向链表的头节点。
注意：本题相对原题稍作改动
示例：输入：  [4,2,5,1,3,null,6,0]
输出：  [0,null,1,null,2,null,3,null,4,null,5,null,6]
提示：节点数量不会超过 100000。
```

- 解题思路

```
func convertBiNode(root *TreeNode) *TreeNode {
        head := &TreeNode{}
        cur := head
        dfs(root, cur)
        return head.Right
}

func dfs(root, cur *TreeNode) *TreeNode {
        if root != nil {
                cur = dfs(root.Left, cur)
                root.Left = nil
                cur.Right = root
                cur = root
                cur = dfs(root.Right, cur)
        }
        return cur
}

# 2
func convertBiNode(root *TreeNode) *TreeNode {
        head := &TreeNode{}
        cur := head
        stack := make([]*TreeNode, 0)
        node := root
        for node != nil || len(stack) > 0 {
                if node != nil {
                        stack = append(stack, node)
                        node = node.Left
                } else {
```

(续下页)

(接上页)

```
                        node = stack[len(stack)-1]
                        stack = stack[:len(stack)-1]
                        node.Left = nil
                        cur.Right = node
                        cur = node
                        node = node.Right
                }
        }
        return head.Right
}
```

80.94 面试题 17.13. 恢复空格 (2)

- 题目

```
哦，不！你不小心把一个长篇文章中的空格、标点都删掉了，并且大写也弄成了小写。
像句子"I reset the computer. It still didn't boot!"已经变成了
↪"iresetthecomputeritstilldidntboot"。
在处理标点符号和大小写之前，你得先把它断成词语。
当然了，你有一本厚厚的词典dictionary，不过，有些词没在词典里。
假设文章用sentence表示，设计一个算法，把文章断开，要求未识别的字符最少，返回未识别的字符数。
注意：本题相对原题稍作改动，只需返回未识别的字符数
示例：输入： dictionary = ["looked","just","like","her","brother"]
sentence = "jesslookedjustliketimherbrother"
输出： 7
解释： 断句后为"jess looked just like tim her brother"，共7个未识别字符。
提示：0 <= len(sentence) <= 1000
dictionary中总字符数不超过 150000。
你可以认为dictionary和sentence中只包含小写字母。
```

- 解题思路

```
func respace(dictionary []string, sentence string) int {
        n := len(sentence)
        root := &Trie{
                next: [26]*Trie{},
        }
        for i := 0; i < len(dictionary); i++ {
                root.Insert(reverse(dictionary[i])) // 反序插入
        }
        dp := make([]int, n+1)
        for i := 1; i <= n; i++ {
```

(续下页)

(接上页)

```
                dp[i] = dp[i-1] + 1 // 上一个长度+1
                cur := root
                for j := i; j >= 1; j-- {
                        value := int(sentence[j-1] - 'a')
                        if cur.next[value] == nil {
                                break
                        } else if cur.next[value].ending > 0 { // 找到，更新
                                dp[i] = min(dp[i], dp[j-1])
                        }
                        if dp[i] == 0 {
                                break
                        }
                        cur = cur.next[value]
                }
        }
        return dp[n]
}

func reverse(s string) string {
        arr := []byte(s)
        for i := 0; i < len(s)/2; i++ {
                arr[i], arr[len(s)-1-i] = arr[len(s)-1-i], arr[i]
        }
        return string(arr)
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

type Trie struct {
        next   [26]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int       // 次数（可以改为bool）
}

// 插入word
func (this *Trie) Insert(word string) {
        temp := this
        for _, v := range word {
                value := v - 'a'
```

(续下页)

(接上页)

```
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
        temp.ending++
}

# 2
func respace(dictionary []string, sentence string) int {
        n := len(sentence)
        m := make(map[string]bool)
        for i := 0; i < len(dictionary); i++ {
                m[dictionary[i]] = true
        }
        dp := make([]int, n+1)
        for i := 1; i <= n; i++ {
                dp[i] = dp[i-1] + 1 // 上一个长度+1
                for j := i; j >= 1; j-- {
                        str := sentence[j-1 : i]
                        if m[str] == true {
                                dp[i] = min(dp[i], dp[j-1])
                        }
                        if dp[i] == 0 {
                                break
                        }
                }
        }
        return dp[n]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

80.95 面试题 17.14. 最小 K 个数 (3)

- 题目

```
设计一个算法，找出数组中最小的k个数。以任意顺序返回这k个数均可。
示例：输入： arr = [1,3,5,7,2,4,6,8], k = 4 输出： [1,2,3,4]
提示：0 <= len(arr) <= 100000
    0 <= k <= min(100000, len(arr))
```

- 解题思路

```
func smallestK(arr []int, k int) []int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        for i := 0; i < len(arr); i++ {
                heap.Push(&intHeap, arr[i])
        }
        res := make([]int, 0)
        for i := 0; i < k; i++ {
                value := heap.Pop(&intHeap).(int)
                res = append(res, value)
        }
        return res
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}
```

(续下页)

(接上页)

```
func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func smallestK(arr []int, k int) []int {
        return quickSort(arr, 0, len(arr)-1, k)
}

func quickSort(arr []int, left, right, k int) []int {
        if left > right {
                return nil
        }
        index := partition(arr, left, right)
        if index == k {
                return arr[:k]
        } else if index < k {
                return quickSort(arr, index+1, right, k)
        }
        return quickSort(arr, left, index-1, k)
}

func partition(arr []int, left, right int) int {
        baseValue := arr[left] // 基准值
        for left < right {
                for baseValue <= arr[right] && left < right {
                        right-- // 依次查找大于基准值的位置
                }
                arr[left] = arr[right]
                for arr[left] <= baseValue && left < right {
                        left++ // 依次查找小于基准值的位置
                }
                arr[right] = arr[left]
        }
        arr[right] = baseValue
        return right
}

# 3
func smallestK(arr []int, k int) []int {
        sort.Ints(arr)
```

(续下页)

(接上页)

```
        return arr[:k]
}
```

80.96 面试题 17.15. 最长单词 (2)

- 题目

```
给定一组单词words，编写一个程序，找出其中的最长单词，且该单词由这组单词中的其他单词组合而成。
若有多个长度相同的结果，返回其中字典序最小的一项，若没有符合要求的单词则返回空字符串。
示例：输入： ["cat","banana","dog","nana","walk","walker","dogwalker"] 输出：
↪"dogwalker"
解释： "dogwalker"可由"dog"和"walker"组成。
提示：0 <= len(words) <= 200
1 <= len(words[i]) <= 100
```

- 解题思路

```
var m map[string]bool

func longestWord(words []string) string {
        m = make(map[string]bool)
        n := len(words)
        for i := 0; i < n; i++ {
                m[words[i]] = true
        }
        sort.Slice(words, func(i, j int) bool {
                if len(words[i]) == len(words[j]) {
                        return words[i] < words[j]
                }
                return len(words[i]) > len(words[j])
        })
        for i := 0; i < n; i++ { // 从最长最小字典序的开始找
                m[words[i]] = false
                if dfs(words[i]) == true {
                        return words[i]
                }
        }
        return ""
}

func dfs(str string) bool {
        if len(str) == 0 || m[str] == true {
```

(续下页)

(接上页)

```
                return true
        }
        for i := 1; i <= len(str); i++ {
                subStr := str[:i]
                if m[subStr] == true {
                        if dfs(str[i:]) == true {
                                return true
                        }
                }
        }
        return false
}

# 2
var m map[string]bool

func longestWord(words []string) string {
        m = make(map[string]bool)
        n := len(words)
        for i := 0; i < n; i++ {
                m[words[i]] = true
        }
        sort.Slice(words, func(i, j int) bool {
                if len(words[i]) == len(words[j]) {
                        return words[i] < words[j]
                }
                return len(words[i]) > len(words[j])
        })
        // 从最长最小字典序的开始找
        for i := 0; i < n; i++ {
                m[words[i]] = false
                if judge(words[i]) == true {
                        return words[i]
                }
        }
        return ""
}

// leetcode 139.单词拆分
func judge(s string) bool {
        dp := make([]bool, len(s)+1)
        dp[0] = true
        n := len(s)
```

(续下页)

(接上页)

```
        for i := 1; i <= n; i++ {
                for j := 0; j < i; j++ {
                        if dp[j] == true && m[s[j:i]] == true {
                                dp[i] = true
                                break
                        }
                }
        }
        return dp[n]
}
```

80.97 面试题 17.16. 按摩师 (4)

- 题目

```
一个有名的按摩师会收到源源不断的预约请求，每个预约都可以选择接或不接。
在每次预约服务之间要有休息时间，因此她不能接受相邻的预约。
给定一个预约请求序列，替按摩师找到最优的预约集合（总预约时间最长），返回总的分钟数。
注意：本题相对原题稍作改动
示例 1：输入： [1,2,3,1] 输出： 4
解释： 选择 1 号预约和 3 号预约，总时长 = 1 + 3 = 4。
示例 2：输入： [2,7,9,3,1] 输出： 12
解释： 选择 1 号预约、 3 号预约和 5 号预约，总时长 = 2 + 9 + 1 = 12。
示例 3：输入： [2,1,4,5,3,1,1,3] 输出： 12
解释： 选择 1 号预约、 3 号预约、 5 号预约和 8 号预约，总时长 = 2 + 4 + 3 + 3 = 12。
```

- 解题思路

```
func massage(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
        if len(nums) == 1 {
                return nums[0]
        }
        a := nums[0]
        b := max(a, nums[1])

        for i := 2; i < len(nums); i++ {
                a, b = b, max(a+nums[i], b)
        }
        return b
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func massage(nums []int) int {
        n := len(nums)
        if n == 0 {
                return 0
        }
        if n == 1 {
                return nums[0]
        }
        dp := make([]int, n)
        dp[0] = nums[0]
        if nums[0] > nums[1] {
                dp[1] = nums[0]
        } else {
                dp[1] = nums[1]
        }
        for i := 2; i < n; i++ {
                dp[i] = max(dp[i-1], dp[i-2]+nums[i])
        }
        return dp[n-1]
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func massage(nums []int) int {
        if len(nums) == 0 {
                return 0
        }
```

(续下页)

(接上页)

```
        if len(nums) == 1 {
                return nums[0]
        }
        n := len(nums)
        dp := make([][]int, n)
        for n := range dp {
                dp[n] = make([]int, 2)
        }
        dp[0][0], dp[0][1] = 0, nums[0]
        for i := 1; i < n; i++ {
                dp[i][0] = max(dp[i-1][0], dp[i-1][1])
                dp[i][1] = dp[i-1][0] + nums[i]
        }
        return max(dp[n-1][0], dp[n-1][1])
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 4
func massage(nums []int) int {
        var a, b int
        for i, v := range nums {
                if i%2 == 0 {
                        a = max(a+v, b)
                } else {
                        b = max(a, b+v)
                }
        }
        return max(a, b)
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

80.98 面试题 17.17. 多次搜索 (3)

- 题目

```
给定一个较长字符串big和一个包含较短字符串的数组smalls，设计一个方法，根据smalls中的每一个较短字符串，
输出smalls中的字符串在big里出现的所有位置positions，其中positions[i]为smalls[i]出现的所有位置。
示例：输入：big = "mississippi" smalls = ["is","ppi","hi","sis","i","ssippi"]
输出： [[1,4],[8],[],[3],[1,4,7,10],[5]]
提示：0 <= len(big) <= 1000
0 <= len(smalls[i]) <= 1000
smalls的总字符数不会超过 100000。
你可以认为smalls中没有重复字符串。
所有出现的字符均为英文小写字母。
```

- 解题思路

```
func multiSearch(big string, smalls []string) [][]int {
        n := len(smalls)
        res := make([][]int, n)
        arr := suffixarray.New([]byte(big)) // 创建后缀树
        for i := 0; i < n; i++ {
                target := []byte(smalls[i])
                temp := arr.Lookup(target, -1) //␣
↪返回arr中所有target出现的位置，从后往前
                sort.Ints(temp)
                res[i] = temp
        }
        return res
}

# 2
func multiSearch(big string, smalls []string) [][]int {
        n := len(smalls)
        res := make([][]int, n)
        for i := 0; i < n; i++ {
                arr := make([]int, 0)
                if smalls[i] == "" {
                        res[i] = arr
                        continue
                }
                for j := 0; j+len(smalls[i]) <= len(big); j++ {
                        if big[j:j+len(smalls[i])] == smalls[i] {
                                arr = append(arr, j)
                        }
```

(续下页)

(接上页)

```
                }
                res[i] = arr
        }
        return res
}

# 3
func multiSearch(big string, smalls []string) [][]int {
        n := len(smalls)
        res := make([][]int, n)
        root := &Trie{
                next: [26]*Trie{},
        }
        for i := 0; i < n; i++ {
                root.Insert(smalls[i], i+1)
        }
        for i := 0; i < len(big); i++ {
                temp := root.Search(big[i:])
                for j := 0; j < len(temp); j++ {
                        res[temp[j]] = append(res[temp[j]], i)
                }
        }
        return res
}

type Trie struct {
        next   [26]*Trie // 下一级指针，如不限于小写字母，[26]=>[256]
        ending int       // 下标，从1开始
}

// 插入word
func (this *Trie) Insert(word string, index int) {
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp.next[value] == nil {
                        temp.next[value] = &Trie{
                                next:   [26]*Trie{},
                                ending: 0,
                        }
                }
                temp = temp.next[value]
        }
```

(续下页)

(接上页)

```
        temp.ending = index
}

// 查找
func (this *Trie) Search(word string) []int {
        arr := make([]int, 0) // 存放匹配到的下标列表
        temp := this
        for _, v := range word {
                value := v - 'a'
                if temp = temp.next[value]; temp == nil {
                        return arr
                }
                if temp.ending > 0 {
                        arr = append(arr, temp.ending-1)
                }
        }
        return arr
}
```

80.99 面试题 17.18. 最短超串 (1)

- 题目

```
假设你有两个数组，一个长一个短，短的元素均不相同。
找到长数组中包含短数组所有的元素的最短子数组，其出现顺序无关紧要。
返回最短子数组的左端点和右端点，如有多个满足条件的子数组，返回左端点最小的一个。若不存在，返回空数组。
示例 1:输入:big = [7,5,9,0,2,1,3,5,7,9,1,1,5,8,8,9,7] small = [1,5,9] 输出: [7,10]
示例 2:输入: big = [1,2,3] small = [4] 输出: []
提示： big.length <= 100000
    1 <= small.length <= 100000
```

- 解题思路

```
func shortestSeq(big []int, small []int) []int {
        res := make([]int, 0)
        m := make(map[int]int)
        for i := 0; i < len(small); i++ {
                m[small[i]]++
        }
        total := len(m)
        j := 0
        for i := 0; i < len(big); i++ {
```

(续下页)

(接上页)

```
                m[big[i]]--
                if m[big[i]] == 0 {
                        total--
                }
                for total == 0 {
                        m[big[j]]++
                        if m[big[j]] > 0 {
                                total++
                                if len(res) == 0 || res[1]-res[0] > i-j {
                                        res = []int{j, i}
                                }
                        }
                        j++
                }
        }
        return res
```

80.100 面试题 17.19. 消失的两个数字 (4)

- 题目

```
给定一个数组，包含从 1 到 N 所有的整数，但其中缺了两个数字。
你能在 O(N) 时间内只用 O(1) 的空间找到它们吗？
以任意顺序返回这两个数字均可。
示例 1:输入: [1] 输出: [2,3]
示例 2:输入: [2,3] 输出: [1,4]
提示：nums.length <= 30000
```

- 解题思路

```
func missingTwo(nums []int) []int {
        res := make([]int, 0)
        m := make(map[int]bool)
        for i := 0; i < len(nums); i++ {
                m[nums[i]] = true
        }
        for i := 1; i <= len(nums)+2; i++ {
                if m[i] == false {
                        res = append(res, i)
                }
        }
        return res
```

(续下页)

(接上页)

```
}

# 2
func missingTwo(nums []int) []int {
        n := len(nums) + 2
        sum := (1 + n) * n / 2
        total := 0
        for i := 0; i < len(nums); i++ {
                total = total + nums[i]
        }
        diff := sum - total // a+b
        mid := diff / 2     // (a+b)/2
        tempSum := (1 + mid) * mid / 2
        temp := 0
        for i := 0; i < len(nums); i++ {
                if nums[i] <= mid {
                        temp = temp + nums[i]
                }
        }
        a := tempSum - temp
        b := diff - a
        return []int{a, b}
}

# 3
func missingTwo(nums []int) []int {
        res := make([]int, 0)
        nums = append(nums, -1, -1, 0)
        for i := 0; i < len(nums); i++ {
                for nums[i] != -1 && nums[i] != i {
                        nums[nums[i]], nums[i] = nums[i], nums[nums[i]]
                }
        }
        for i := 1; i < len(nums); i++ {
                if nums[i] == -1 {
                        res = append(res, i)
                }
        }
        return res
}

# 4
func missingTwo(nums []int) []int {
```

(续下页)

(接上页)

```
        temp := 0
        for i := 0; i < len(nums); i++ {
                temp = temp ^ nums[i]
        }
        for i := 1; i <= len(nums)+2; i++ {
                temp = temp ^ i
        }
        a := 0
        diff := temp & (-temp)
        for i := 1; i <= len(nums)+2; i++ {
                if diff&i != 0 {
                        a = a ^ i
                }
        }
        for i := 0; i < len(nums); i++ {
                if diff&nums[i] != 0 {
                        a = a ^ nums[i]
                }
        }
        return []int{a, a ^ temp}
}
```

80.101 面试题 17.20. 连续中值 (1)

- 题目

```
随机产生数字并传递给一个方法。你能否完成这个方法，在每次产生新值时，寻找当前所有值的中间值（中位数）
中位数是有序列表中间的数。如果列表长度是偶数，中位数则是中间两个数的平均值。
例如，[2,3,4]的中位数是 3
[2,3] 的中位数是 (2 + 3) / 2 = 2.5
设计一个支持以下两种操作的数据结构：
void addNum(int num) - 从数据流中添加一个整数到数据结构中。
double findMedian() - 返回目前所有元素的中位数。
示例：addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

- 解题思路

```
type MinHeap []int

func (i MinHeap) Len() int {
        return len(i)
}

func (i MinHeap) Less(x, y int) bool {
        return i[x] < i[y]
}

func (i MinHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *MinHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *MinHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
}

type MaxHeap []int

func (i MaxHeap) Len() int {
        return len(i)
}

func (i MaxHeap) Less(x, y int) bool {
        return i[x] > i[y]
}

func (i MaxHeap) Swap(x, y int) {
        i[x], i[y] = i[y], i[x]
}
func (i *MaxHeap) Push(v interface{}) {
        *i = append(*i, v.(int))
}

func (i *MaxHeap) Pop() interface{} {
        value := (*i)[len(*i)-1]
        *i = (*i)[:len(*i)-1]
        return value
```

(续下页)

(接上页)

```
}

type MedianFinder struct {
        minArr *MinHeap
        maxArr *MaxHeap
}

func Constructor() MedianFinder {
        res := new(MedianFinder)
        res.minArr = new(MinHeap)
        res.maxArr = new(MaxHeap)
        heap.Init(res.minArr)
        heap.Init(res.maxArr)
        return *res
}

func (this *MedianFinder) AddNum(num int) {
        if this.maxArr.Len() == this.minArr.Len() {
                heap.Push(this.minArr, num)
                heap.Push(this.maxArr, heap.Pop(this.minArr))
        } else {
                heap.Push(this.maxArr, num)
                heap.Push(this.minArr, heap.Pop(this.maxArr))
        }
}

func (this *MedianFinder) FindMedian() float64 {
        if this.minArr.Len() == this.maxArr.Len() {
                return (float64((*this.maxArr)[0]) + float64((*this.minArr)[0])) / 2
        } else {
                return float64((*this.maxArr)[0])
        }
}
```

80.102 面试题 17.21. 直方图的水量 (4)

- 题目

```
给定一个直方图(也称柱状图)，假设有人从上面源源不断地倒水，最后直方图能存多少水量?
↪直方图的宽度为 1。
上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的直方图，
在这种情况下，可以接 6 个单位的水（蓝色部分表示水）。 感谢 Marcos 贡献此图。
```

(续下页)

(接上页)

```
示例:输入: [0,1,0,2,1,0,1,3,2,1,2,1] 输出: 6
```

- 解题思路

```
func trap(height []int) int {
        res := 0
        for i := 0; i < len(height); i++ {
                left, right := 0, 0
                for j := i; j >= 0; j-- {
                        left = max(left, height[j])
                }
                for j := i; j < len(height); j++ {
                        right = max(right, height[j])
                }
                // 当前坐标形成的面积=(min(左边最高, 右边最高)-当前高度) * 宽度(1,
↪可省略)
                area := min(left, right) - height[i]
                res = res + area
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func trap(height []int) int {
        res := 0
        if len(height) == 0 {
                return 0
        }
        left := make([]int, len(height))
        right := make([]int, len(height))
```

(续下页)

(接上页)

```
        left[0] = height[0]
        right[len(right)-1] = height[len(height)-1]
        for i := 1; i < len(height); i++ {
                left[i] = max(height[i], left[i-1])
        }
        for i := len(height) - 2; i >= 0; i-- {
                right[i] = max(height[i], right[i+1])
        }
        for i := 0; i < len(height); i++ {
                // 当前坐标形成的面积=(min(左边最高, 右边最高)-当前高度) * 宽度(1,
↪可省略)
                area := min(left[i], right[i]) - height[i]
                res = res + area
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 3
func trap(height []int) int {
        res := 0
        stack := make([]int, 0)
        for i := 0; i < len(height); i++ {
                for len(stack) > 0 && height[i] > height[stack[len(stack)-1]] {
                        bottom := height[stack[len(stack)-1]]
                        stack = stack[:len(stack)-1]
                        if len(stack) > 0 {
                                prev := stack[len(stack)-1]
                                // 横着的面积=长(min(height[i], height[prev])-
↪bottom)*宽(i-prev-1)
```

(续下页)

(接上页)

```
                                h := min(height[i], height[prev]) - bottom
                                w := i - prev - 1
                                area := h * w
                                res = res + area
                        }
                }
                stack = append(stack, i)
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 4
func trap(height []int) int {
        res := 0
        if len(height) == 0 {
                return 0
        }
        left := 0
        right := len(height) - 1
        leftMax := 0  // 左边的最大值
        rightMax := 0 // 右边的最大值
        for left < right {
                // 当前坐标形成的面积=(min(左边最高, 右边最高)-当前高度) * 宽度(1,
↪可省略)
                // 选择高度低的一边处理并求最大值, 说明当前侧最大值小于另一侧
                if height[left] < height[right] {
                        // 也可以写成这样
                        // leftMax = max(leftMax, height[left])
                        // res = res + leftMax - height[left]
                        if height[left] >= leftMax { // 递增无法蓄水
                                leftMax = height[left]
                        } else {
                                res = res + leftMax - height[left]
                        }
                        left++
                } else {
```

(续下页)

(接上页)

```
                        // 也可以写成这样
                        // rightMax = max(rightMax, height[right])
                        // res = res + rightMax - height[right]
                        if height[right] >= rightMax { // 递减无法蓄水
                                rightMax = height[right]
                        } else {
                                res = res + rightMax - height[right]
                        }
                        right--
                }
        }
        return res
}
```

80.103 面试题 17.22. 单词转换 (2)

- 题目

```
给定字典中的两个词，长度相等。写一个方法，把一个词转换成另一个词，␣
↪但是一次只能改变一个字符。
每一步得到的新词都必须能在字典中找到。
编写一个程序，返回一个可能的转换序列。如有多个可能的转换序列，你可以返回任何一个。
示例 1:输入:beginWord = "hit", endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
输出:["hit","hot","dot","lot","log","cog"]
示例 2:输入:beginWord = "hit" endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
输出: []
解释: endWord "cog" 不在字典中，所以不存在符合要求的转换序列。
```

- 解题思路

```
func findLadders(beginWord string, endWord string, wordList []string) []string {
        m, preMap := make(map[string]int), make(map[string][]string)
        for i := 0; i < len(wordList); i++ {
                m[wordList[i]] = 1
        }
        if m[endWord] == 0 {
                return nil
        }
        for i := 0; i < len(wordList); i++ {
                for j := 0; j < len(wordList[i]); j++ {
```

(续下页)

(接上页)

```
                        newStr := wordList[i][:j] + "*" + wordList[i][j+1:]
                        preMap[newStr] = append(preMap[newStr], wordList[i])
                }
        }
        visited := make(map[string]bool)
        queue, path := make([]string, 0), make([][]string, 0)
        queue, path = append(queue, beginWord), append(path, []string{beginWord})
        for len(queue) > 0 {
                length := len(queue)
                for i := 0; i < length; i++ {
                        for j := 0; j < len(beginWord); j++ {
                                newStr := queue[i][:j] + "*" + queue[i][j+1:]
                                temp := make([]string, len(path[i]))
                                copy(temp, path[i])
                                for _, word := range preMap[newStr] {
                                        if word == endWord {
                                                return append(temp, endWord)
                                        } else if visited[word] == false {
                                                visited[word] = true
                                                queue, path = append(queue, word),␣
↪append(path, append(temp, word))
                                        }
                                }
                        }
                }
                queue, path = queue[length:], path[length:]
        }
        return nil
}

# 2
func findLadders(beginWord string, endWord string, wordList []string) []string {
        m := make(map[string]int)
        for i := 0; i < len(wordList); i++ {
                m[wordList[i]] = 1
        }
        if m[endWord] == 0 {
                return nil
        }
        preMap := make(map[string][]string)
        for i := 0; i < len(wordList); i++ {
                for j := 0; j < len(wordList[i]); j++ {
                        newStr := wordList[i][:j] + "*" + wordList[i][j+1:]
```

(续下页)

(接上页)

```
                        if _, ok := preMap[newStr]; !ok {
                                preMap[newStr] = make([]string, 0)
                        }
                        preMap[newStr] = append(preMap[newStr], wordList[i])
                }
        }
        visited := make(map[string]bool)
        count := 0
        queue := make([]string, 0)
        queue = append(queue, beginWord)
        path := make([][]string, 0)
        path = append(path, []string{beginWord})
        for len(queue) > 0 {
                count++
                node := queue[0]
                queue = queue[1:]
                arr := path[0]
                path = path[1:]
                for j := 0; j < len(beginWord); j++ {
                        newStr := node[:j] + "*" + node[j+1:]
                        temp := make([]string, len(arr))
                        copy(temp, arr)
                        for _, word := range preMap[newStr] {
                                if word == endWord {
                                        return append(temp, endWord)
                                }
                                if visited[word] == false {
                                        visited[word] = true
                                        queue = append(queue, word)
                                        path = append(path, append(temp, word))
                                }
                        }
                }
        }
        return nil
}
```

80.104 面试题 17.23. 最大黑方阵

80.104.1 题目

```
给定一个方阵，其中每个单元(像素)非黑即白。设计一个算法，找出 4␣
↪条边皆为黑色像素的最大子方阵。
返回一个数组 [r, c, size] ，其中r,c分别代表子方阵左上角的行号和列号，size␣
↪是子方阵的边长。
若有多个满足条件的子方阵，返回 r 最小的，若 r 相同，返回 c 最小的子方阵。
若无满足条件的子方阵，返回空数组。
示例 1:输入:
[
  [1,0,1],
  [0,0,1],
  [0,0,1]
]
输出: [1,0,2]
解释: 输入中 0 代表黑色，1 代表白色，标粗的元素即为满足条件的最大子方阵
示例 2:输入:
[
  [0,1,1],
  [1,0,1],
  [1,1,0]
]
输出: [0,0,1]
提示：matrix.length == matrix[0].length <= 200
```

80.104.2 解题思路

80.105 面试题 17.24. 最大子矩阵 (3)

- 题目

```
给定一个正整数、负整数和 0 组成的 N × M矩阵，编写代码找出元素总和最大的子矩阵。
返回一个数组 [r1, c1, r2, c2]，其中 r1, c1 分别代表子矩阵左上角的行号和列号，r2, c2␣
↪分别代表右下角的行号和列号。
若有多个满足条件的子矩阵，返回任意一个均可。
注意：本题相对书上原题稍作改动
```

(续下页)

(接上页)

```
示例： 输入：
[
  [-1,0],
  [0,-1]
]
输出：[0,1,0,1]
解释：输入中标粗的元素即为输出所表示的矩阵
说明：1 <= matrix.length, matrix[0].length <= 200
```

- 解题思路

```
func getMaxMatrix(matrix [][]int) []int {
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr[i+1][j+1] = matrix[i][j] + arr[i+1][j] + arr[i][j+1] -␣
↪arr[i][j]
                }
        }
        maxValue := math.MinInt32
        res := make([]int, 0)
        for a := 1; a <= n; a++ { // 上边界
                for b := a; b <= n; b++ { // 下边界
                        left := 1
                        value := 0
                        for right := 1; right <= m; right++ {
                                value = arr[b][right] - arr[b][left-1] - arr[a-
↪1][right] + arr[a-1][left-1]
                                if value > maxValue {
                                        maxValue = value
                                        res = []int{a - 1, left - 1, b - 1, right - 1}
                                }
                                if value < 0 {
                                        value = 0
                                        left = right + 1
                                }
                        }
                }
        }
        return res
```

(续下页)

(接上页)

```
}

# 2
func getMaxMatrix(matrix [][]int) []int {
        n, m := len(matrix), len(matrix[0])
        arr := make([][]int, n+1)
        for i := 0; i <= n; i++ {
                arr[i] = make([]int, m+1)
        }
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        arr[i+1][j+1] = matrix[i][j] + arr[i+1][j] + arr[i][j+1] -␣
↪arr[i][j]
                }
        }
        maxValue := math.MinInt32
        res := make([]int, 0)
        for a := 0; a < n; a++ { // 上边界
                for b := a; b < n; b++ { // 下边界
                        left := 0
                        value := 0
                        for right := 0; right < m; right++ {
                                value = arr[b+1][right+1] - arr[b+1][left] -␣
↪arr[a][right+1] + arr[a][left]
                                if value > maxValue {
                                        maxValue = value
                                        res = []int{a, left, b, right}
                                }
                                if value < 0 {
                                        value = 0
                                        left = right + 1
                                }
                        }
                }
        }
        return res
}

# 3
func getMaxMatrix(matrix [][]int) []int {
        n, m := len(matrix), len(matrix[0])
        maxValue := math.MinInt32
        res := make([]int, 0)
```

(续下页)

(接上页)

```
        for a := 0; a < n; a++ { // 上边界
                arr := make([]int, m)
                for b := a; b < n; b++ { // 下边界
                        left := 0
                        value := 0
                        for right := 0; right < m; right++ {
                                arr[right] = arr[right] + matrix[b][right]
                                value = value + arr[right]
                                if value > maxValue {
                                        maxValue = value
                                        res = []int{a, left, b, right}
                                }
                                if value < 0 {
                                        value = 0
                                        left = right + 1
                                }
                        }
                }
        }
        return res
}
```

80.106 面试题 17.26. 稀疏相似度 (1)

- 题目

```
两个(具有不同单词的)文档的交集(intersection)中元素的个数除以并集(union)中元素的个数，
就是这两个文档的相似度。
例如，{1, 5, 3} 和 {1, 7, 2, 3} 的相似度是 0.4，其中，交集的元素有 2 个，并集的元素有␣
↪5 个。
给定一系列的长篇文档，每个文档元素各不相同，并与一个 ID 相关联。
它们的相似度非常“稀疏”，也就是说任选 2 个文档，相似度都很接近 0。
请设计一个算法返回每对文档的 ID 及其相似度。
只需输出相似度大于 0 的组合。请忽略空文档。
为简单起见，可以假定每个文档由一个含有不同整数的数组表示。
输入为一个二维数组 docs，docs[i]表示id 为 i 的文档。
返回一个数组，其中每个元素是一个字符串，代表每对相似度大于 0 的文档，
其格式为 {id1},{id2}: {similarity}，其中 id1 为两个文档中较小的 id，similarity␣
↪为相似度，
精确到小数点后 4 位。以任意顺序返回数组均可。
示例:输入:
[
```

(续下页)

(接上页)

```
 [14, 15, 100, 9, 3],
 [32, 1, 9, 3, 5],
 [15, 29, 2, 6, 8, 7],
 [7, 10]
]
输出:
[
 "0,1: 0.2500",
 "0,2: 0.1000",
 "2,3: 0.1429"
]
提示: docs.length <= 500
docs[i].length <= 500
```

- 解题思路

```
func computeSimilarities(docs [][]int) []string {
        res := make([]string, 0)
        n := len(docs)
        m := make(map[[2]int]int)
        m1 := make(map[int][]int) // 字符出现的位置
        for i := 0; i < n; i++ {
                for j := 0; j < len(docs[i]); j++ {
                        char := docs[i][j]
                        for _, v := range m1[char] {
                                m[[2]int{v, i}]++
                        }
                        m1[char] = append(m1[char], i)
                }
        }
        for k, v := range m {
                x := v
                y := len(docs[k[0]]) + len(docs[k[1]]) - v
                res = append(res, fmt.Sprintf("%d,%d: %.4f",
                        k[0], k[1], float64(x)/float64(y)+1e-9))
        }
        return res
}
```

CHAPTER 81

LCP

81.1 LCP01. 猜数字 (2)

- 题目

```
小A 和 小B 在玩猜数字。小B 每次从 1, 2, 3 中随机选择一个，小A 每次也从 1, 2, 3␣
↪中选择一个猜。
他们一共进行三次这个游戏，请返回 小A 猜对了几次？
输入的guess数组为 小A 每次的猜测，answer数组为 小B 每次的选择。
guess和answer的长度都等于3。
示例 1：输入：guess = [1,2,3], answer = [1,2,3] 输出：3
解释：小A 每次都猜对了。
示例 2：输入：guess = [2,2,3], answer = [3,2,1] 输出：1
解释：小A 只猜对了第二次。
限制：
    guess的长度 = 3
    answer的长度 = 3
    guess的元素取值为 {1, 2, 3} 之一。
    answer的元素取值为 {1, 2, 3} 之一。
```

- 解题思路

```
func game(guess []int, answer []int) int {
        res := 0
        for i := 0; i < len(guess); i++ {
```

(续下页)

(接上页)

```
                if guess[i] == answer[i] {
                        res++
                }
        }
        return res
}

#
func game(guess []int, answer []int) int {
        res := 0
        for i := 0; i < len(guess); i++ {
                if guess[i]^answer[i] == 0 {
                        res++
                }
        }
        return res
}
```

81.2 LCP02. 分式化简 (2)

- 题目

```
有一个同学在学习分式。他需要将一个连分数化成最简分数，你能帮助他吗？
连分数是形如上图的分式。在本题中，所有系数都是大于等于0的整数。
输入的cont代表连分数的系数（cont[0]代表上图的a0，以此类推）。
返回一个长度为2的数组[n, m]，使得连分数的值等于n / m，且n, m最大公约数为1。
示例 1：输入：cont = [3, 2, 0, 2] 输出：[13, 4]
解释：原连分数等价于3 + (1 / (2 + (1 / (0 + 1 / 2))))。注意[26, 8], [-13, -
↪4]都不是正确答案。
示例 2：输入：cont = [0, 0, 3] 输出：[3, 1]
解释：如果答案是整数，令分母为1即可。
限制：
    cont[i] >= 0
    1 <= cont的长度 <= 10
    cont最后一个元素不等于0
    答案的n, m的取值都能被32位int整型存下（即不超过2 ^ 31 - 1）。
```

- 解题思路

```
func fraction(cont []int) []int {
        n, m := 1, cont[len(cont)-1]
        for i := len(cont) - 2; i >= 0; i-- {
```

(续下页)

(接上页)

```
                n, m = m, cont[i]*m+n
        }
        return []int{m, n}
}

#
func fraction(cont []int) []int {
        if len(cont) == 1 {
                return []int{cont[0], 1}
        }
        n := fraction(cont[1:])
        m := cont[0]
        return []int{m*n[0] + n[1], n[0]}
}
```

81.3 LCP03. 机器人大冒险 (1)

- 题目

```
力扣团队买了一个可编程机器人，机器人初始位置在原点(0, 0)。
小伙伴事先给机器人输入一串指令command，机器人就会无限循环这条指令的步骤进行移动。指令有两种：
    U: 向y轴正方向移动一格
    R: 向x轴正方向移动一格。
不幸的是，在 xy␣
↪平面上还有一些障碍物，他们的坐标用obstacles表示。机器人一旦碰到障碍物就会被损毁。
给定终点坐标(x, y)，返回机器人能否完好地到达终点。如果能，返回true；否则返回false。
示例 1：输入：command = "URR", obstacles = [], x = 3, y = 2 输出：true
解释：U(0, 1) -> R(1, 1) -> R(2, 1) -> U(2, 2) -> R(3, 2)。
示例 2：输入：command = "URR", obstacles = [[2, 2]], x = 3, y = 2 输出：false
解释：机器人在到达终点前会碰到(2, 2)的障碍物。
示例 3：输入：command = "URR", obstacles = [[4, 2]], x = 3, y = 2 输出：true
解释：到达终点后，再碰到障碍物也不影响返回结果。
限制：
    2 <= command的长度 <= 1000
    command由U，R构成，且至少有一个U，至少有一个R
    0 <= x <= 1e9, 0 <= y <= 1e9
    0 <= obstacles的长度 <= 1000
    obstacles[i]不为原点或者终点
```

- 解题思路

```
func robot(command string, obstacles [][]int, x int, y int) bool {
        if judge(command, x, y) == false {
                return false
        }
        for _, node := range obstacles {
                if x >= node[0] && y >= node[1] && judge(command, node[0], node[1]) {
                        return false
                }
        }
        return true
}

func judge(command string, x, y int) bool {
        u := strings.Count(command, "U")
        r := strings.Count(command, "R")
        times := (x + y) / len(command)
        last := command[:(x+y)%len(command)]
        uNum := u*times + strings.Count(last, "U")
        rNum := r*times + strings.Count(last, "R")
        if uNum == y && rNum == x {
                return true
        }
        return false
}
```

81.4 LCP06. 拿硬币 (2)

- 题目

```
桌上有 n 堆力扣币，每堆的数量保存在数组 coins 中。
我们每次可以选择任意一堆，拿走其中的一枚或者两枚，求拿完所有力扣币的最少次数。
示例 1：输入：[4,2,1]输出：4
解释：第一堆力扣币最少需要拿 2 次，第二堆最少需要拿 1 次，第三堆最少需要拿 1 次，总共␣
↪4 次即可拿完。
示例 2：  输入：[2,3,10]输出：8
限制：
    1 <= n <= 4
    1 <= coins[i] <= 10
```

- 解题思路

```
func minCount(coins []int) int {
        res := 0
```

(续下页)

(接上页)

```
        for i := 0; i < len(coins); i++ {
                res = res + coins[i]/2
                if coins[i]%2 == 1 {
                        res = res + 1
                }
        }
        return res
}

#
func minCount(coins []int) int {
        res := 0
        for i := 0; i < len(coins); i++ {
                res = res + int(math.Ceil(float64(coins[i])/2))
        }
        return res
}
```

81.5 LCP07. 传递信息 (5)

- 题目

```
小朋友 A 在和 ta 的小伙伴们玩传信息游戏，游戏规则如下：
    有 n 名玩家，所有玩家编号分别为 0 ~ n-1，其中小朋友 A 的编号为 0
    每个玩家都有固定的若干个可传信息的其他玩家（也可能没有）。
    传信息的关系是单向的（比如 A 可以向 B 传信息，但 B 不能向 A 传信息）。
    每轮信息必须需要传递给另一个人，且信息可重复经过同一个人
给定总玩家数 n，以及按 [玩家编号,对应可传递玩家编号] 关系组成的二维数组 relation。
返回信息从小 A (编号 0 ) 经过 k 轮传递到编号为 n-1␣
↪的小伙伴处的方案数；若不能到达，返回 0。

示例 1：
    输入：n = 5, relation = [[0,2],[2,1],[3,4],[2,3],[1,4],[2,0],[0,4]], k = 3
    输出：3
    解释：信息从小 A 编号 0 处开始，经 3 轮传递，到达编号 4。
    共有 3 种方案，分别是 0->2->0->4， 0->2->1->4， 0->2->3->4。
示例 2：
    输入：n = 3, relation = [[0,2],[2,1]], k = 2
    输出：0
    解释：信息不能从小 A 处经过 2 轮传递到编号 2
限制：
    2 <= n <= 10
```

(续下页)

(接上页)

```
    1 <= k <= 5
    1 <= relation.length <= 90, 且 relation[i].length == 2
    0 <= relation[i][0],relation[i][1] < n 且 relation[i][0] != relation[i][1]
```

- 解题思路

```
var ways [][]bool

func numWays(n int, relation [][]int, k int) int {
        ways = make([][]bool, n)
        for i := range ways {
                ways[i] = make([]bool, n)
        }
        sum := 0
        for i := 0; i < len(relation); i++ {
                ways[relation[i][0]][relation[i][1]] = true
        }
        for i := 0; i < n; i++ {
                if ways[0][i] == true {
                        sum = sum + dfs(i, n, k-1)
                }
        }
        return sum
}

func dfs(i, n, k int) int {
        sum := 0
        if k < 0 || i < 0 || i >= n {
                return 0
        }
        if k == 0 && i == n-1 {
                return 1
        } else {
                for j := 0; j < n; j++ {
                        if ways[i][j] == true {
                                sum += dfs(j, n, k-1)
                        }
                }
        }
        return sum
}

#
var res, K, N int
```

(续下页)

(接上页)

```
func numWays(n int, relation [][]int, k int) int {
        res = 0
        K = k
        N = n
        dfs(0, relation, 0)
        return res
}

func dfs(n int, relation [][]int, k int) {
        if n == N-1 && k == K {
                res++
        }
        if k > K {
                return
        }
        for i := 0; i < len(relation); i++ {
                if relation[i][0] == n {
                        dfs(relation[i][1], relation, k+1)
                }
        }
}

# 3
func numWays(n int, relation [][]int, k int) int {
        m := make(map[int][]int)
        for i := 0; i < len(relation); i++ {
                m[relation[i][0]] = append(m[relation[i][0]], relation[i][1])
        }
        start := []int{0}
        for i := 0; i < k; i++ {
                arr := make([]int, 0)
                for j := 0; j < len(start); j++ {
                        for k := 0; k < len(m[start[j]]); k++ {
                                arr = append(arr, m[start[j]][k])
                        }
                }
                start = arr
        }
        res := 0
        for i := 0; i < len(start); i++ {
                if start[i] == n-1 {
                        res++
```

(续下页)

(接上页)

```
                }
        }
        return res
}

# 4
func numWays(n int, relation [][]int, k int) int {
        dp := make([]int, n)
        dp[0] = 1
        for i := 0; i < k; i++ {
                temp := make([]int, n)
                for _, v := range relation {
                        temp[v[1]] = temp[v[1]] + dp[v[0]]
                }
                dp = temp
        }
        return dp[n-1]
}

# 5
func numWays(n int, relation [][]int, k int) int {
        dp := make([][]int, k+1)
        for i := 0; i < k+1; i++ {
                dp[i] = make([]int, n)
        }
        dp[0][0] = 1
        for i := 0; i < k; i++ {
                for _, v := range relation {
                        dp[i+1][v[1]] = dp[i+1][v[1]] + dp[i][v[0]]
                }
        }
        return dp[k][n-1]
}
```

81.6 LCP08. 剧情触发时间 (2)

- 题目

```
在战略游戏中，玩家往往需要发展自己的势力来触发各种新的剧情。一个势力的主要属性有三种，
分别是文明等级（C），资源储备（R）以及人口数量（H）。在游戏开始时（第 0␣
↪天），三种属性的值均为 0。
随着游戏进程的进行，每一天玩家的三种属性都会对应增加，我们用一个二维数组 increase␣
```

(续下页)

(接上页)

```
↪来表示每天的增加情况。
这个二维数组的每个元素是一个长度为 3 的一维数组，
例如 [[1,2,1],[3,4,2]] 表示第一天三种属性分别增加 1,2,1 而第二天分别增加 3,4,2。
所有剧情的触发条件也用一个二维数组 requirements 表示。
这个二维数组的每个元素是一个长度为 3 的一维数组，对于某个剧情的触发条件 c[i], r[i],␣
↪h[i]，
如果当前 C >= c[i] 且 R >= r[i] 且 H >= h[i] ，则剧情会被触发。
根据所给信息，请计算每个剧情的触发时间，并以一个数组返回。
如果某个剧情不会被触发，则该剧情对应的触发时间为 -1 。
示例 1：
    输入： increase = [[2,8,4],[2,5,0],[10,9,8]]
    requirements = [[2,11,3],[15,10,7],[9,17,12],[8,1,14]]
    输出：[2,-1,3,-1]
    解释：
    初始时，C = 0，R = 0，H = 0
    第 1 天，C = 2，R = 8，H = 4
    第 2 天，C = 4，R = 13，H = 4，此时触发剧情 0
    第 3 天，C = 14，R = 22，H = 12，此时触发剧情 2
    剧情 1 和 3 无法触发。
示例 2：
    输入： increase = [[0,4,5],[4,8,8],[8,6,1],[10,10,0]]
    requirements = [[12,11,16],[20,2,6],[9,2,6],[10,18,3],[8,14,9]]
    输出：[-1,4,3,3,3]
示例 3：输入： increase = [[1,1,1]] requirements = [[0,0,0]] 输出：[0]
限制：
    1 <= increase.length <= 10000
    1 <= requirements.length <= 100000
    0 <= increase[i] <= 10
    0 <= requirements[i] <= 100000
```

- 解题思路

```
func getTriggerTime(increase [][]int, requirements [][]int) []int {
        for i := 1; i < len(increase); i++ {
                increase[i][0] = increase[i][0] + increase[i-1][0]
                increase[i][1] = increase[i][1] + increase[i-1][1]
                increase[i][2] = increase[i][2] + increase[i-1][2]
        }
        res := make([]int, len(requirements))
        for i := 0; i < len(requirements); i++ {
                C, R, H := requirements[i][0], requirements[i][1], requirements[i][2]
                if C == 0 && R == 0 && H == 0 {
                        res[i] = 0
                        continue
```

(续下页)

(接上页)

```
                }
                if C > increase[len(increase)-1][0] ||
                        R > increase[len(increase)-1][1] ||
                        H > increase[len(increase)-1][2] {
                        res[i] = -1
                        continue
                }
                left, right := 0, len(increase)-1
                index := -1
                for left <= right {
                        mid := left + (right-left)/2
                        if increase[mid][0] >= C && increase[mid][1] >= R &&␣
↪increase[mid][2] >= H {
                                index = mid + 1
                                right = mid - 1
                        } else {
                                left = mid + 1
                        }
                }
                res[i] = index
        }
        return res
}

#
func getTriggerTime(increase [][]int, requirements [][]int) []int {
        for i := 1; i < len(increase); i++ {
                increase[i][0] = increase[i][0] + increase[i-1][0]
                increase[i][1] = increase[i][1] + increase[i-1][1]
                increase[i][2] = increase[i][2] + increase[i-1][2]
        }

        res := make([]int, len(requirements))
        for i := 0; i < len(requirements); i++ {
                C, R, H := requirements[i][0], requirements[i][1], requirements[i][2]
                if C == 0 && R == 0 && H == 0 {
                        res[i] = 0
                        continue
                }
                if C > increase[len(increase)-1][0] ||
                        R > increase[len(increase)-1][1] ||
                        H > increase[len(increase)-1][2] {
                        res[i] = -1
```

(续下页)

(接上页)

```
                        continue
                }
                index := sort.Search(len(increase), func(j int) bool {
                        return increase[j][0] >= requirements[i][0] &&
                                increase[j][1] >= requirements[i][1] &&
                                increase[j][2] >= requirements[i][2]
                })
                if index == len(increase) {
                        index = -2
                }
                res[i] = index + 1
        }
        return res
}
```

81.7 LCP09. 最小跳跃次数

81.7.1 题目

```
为了给刷题的同学一些奖励，力扣团队引入了一个弹簧游戏机。
游戏机由 N 个特殊弹簧排成一排，编号为 0 到 N-1。初始有一个小球在编号 0 的弹簧处。
若小球在编号为 i 的弹簧处，通过按动弹簧，可以选择把小球向右弹射jump[i]␣
↪的距离，或者向左弹射到任意左侧弹簧的位置。
也就是说，在编号为 i 弹簧处按动弹簧，小球可以弹向 0 到 i-1 中任意弹簧或者 i+jump[i]␣
↪的弹簧（若 i+jump[i]>=N ，
则表示小球弹出了机器）。小球位于编号 0 处的弹簧时不能再向左弹。
为了获得奖励，你需要将小球弹出机器。
请求出最少需要按动多少次弹簧，可以将小球从编号 0 弹簧弹出整个机器，即向右越过编号 N-1␣
↪的弹簧。
示例 1：输入：jump = [2, 5, 1, 1, 1, 1]输出：3
解释：小 Z 最少需要按动 3 次弹簧，小球依次到达的顺序为 0 -> 2 -> 1 ->␣
↪6，最终小球弹出了机器。
限制：1 <= jump.length <= 10^6
1 <= jump[i] <= 10000
```

81.7.2 解题思路

81.8 LCP11. 期望个数统计 (2)

- 题目

```
某互联网公司一年一度的春招开始了，一共有 n 名面试者入选。
每名面试者都会提交一份简历，公司会根据提供的简历资料产生一个预估的能力值，数值越大代表越有可能通过面
小 A 和小 B␣
↪负责审核面试者，他们均有所有面试者的简历，并且将各自根据面试者能力值从大到小的顺序浏览。
由于简历事先被打乱过，能力值相同的简历的出现顺序是从它们的全排列中等可能地取一个。
现在给定 n 名面试者的能力值 scores，
设 X 代表小 A 和小 B 的浏览顺序中出现在同一位置的简历数，求 X 的期望。
提示：离散的非负随机变量的期望计算公式为 1。在本题中，由于 X 的取值为 0 到 n␣
↪之间，期望计算公式可以是 2。
示例 1： 输入：scores = [1,2,3] 输出：3
    解释：由于面试者能力值互不相同，小 A 和小 B 的浏览顺序一定是相同的。X的期望是 3 。
示例 2：输入：scores = [1,1] 输出：1 解释：设两位面试者的编号为 0，1。
    由于他们的能力值都是 1，小 A 和小 B 的浏览顺序都为从全排列 [[0,1],[1,0]]␣
↪中等可能地取一个。
    如果小 A 和小 B 的浏览顺序都是 [0,1] 或者 [1,0] ，那么出现在同一位置的简历数为 2␣
↪，否则是 0 。
    所以 X 的期望是 (2+0+2+0) * 1/4 = 1
示例 3：输入：scores = [1,1,2] 输出：2
限制：
    1 <= scores.length <= 10^5
    0 <= scores[i] <= 10^6
```

- 解题思路

```
func expectNumber(scores []int) int {
        m := make(map[int]bool)
        for i := 0; i < len(scores); i++{
                m[scores[i]] = true
        }
        return len(m)
}

#
func expectNumber(scores []int) int {
        sort.Ints(scores)
```

(续下页)

(接上页)

```
        count := 0
        for i := 1; i < len(scores); i++ {
                if scores[i] == scores[i-1] {
                        count++
                }
        }
        return len(scores) - count
}
```

81.9 LCP12. 小张刷题计划 (2)

- 题目

```
为了提高自己的代码能力，小张制定了 LeetCode 刷题计划，他选中了 LeetCode 题库中的 n␣
↪道题，
编号从 0 到 n-1，并计划在 m␣
↪天内按照题目编号顺序刷完所有的题目（注意，小张不能用多天完成同一题）。
在小张刷题计划中，小张需要用 time[i] 的时间完成编号 i␣
↪的题目。此外，小张还可以使用场外求助功能，
通过询问他的好朋友小杨题目的解法，可以省去该题的做题时间。为了防止“小张刷题计划”变成“小杨刷题计划”
小张每天最多使用一次求助。
我们定义 m 天中做题时间最多的一天耗时为 T（小杨完成的题目不计入做题总时间）。
请你帮小张求出最小的 T是多少。
示例 1：
    输入：time = [1,2,3,3], m = 2
    输出：3
  ␣
↪解释：第一天小张完成前三题，其中第三题找小杨帮忙；第二天完成第四题，并且找小杨帮忙。
    这样做题时间最多的一天花费了 3 的时间，并且这个值是最小的。
示例 2：输入：time = [999,999,999], m = 4 输出：0
  ␣
↪解释：在前三天中，小张每天求助小杨一次，这样他可以在三天内完成所有的题目并不花任何时间。
限制：
    1 <= time.length <= 10^5
    1 <= time[i] <= 10000
    1 <= m <= 1000
```

- 解题思路

```
func minTime(time []int, m int) int {
        left, right, mid := 0, 0, 0
        for i := 0; i < len(time); i++ {
```

(续下页)

(接上页)

```
                right = right + time[i]
        }
        // 二分查找一个数mid, 使time数组能分割成m个和不小于mid的子数组
        for left <= right {
                mid = left + (right-left)/2
                if check(time, mid, m) {
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return left
}

func check(arr []int, mid, m int) bool {
        maxValue := 0
        sum := 0
        count := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i]
                if arr[i] > maxValue {
                        maxValue = arr[i]
                }
                if sum-maxValue > mid {
                        count++
                        if count >= m {
                                return false
                        }
                        sum = arr[i]
                        maxValue = arr[i]
                }
        }
        return true
}

#
func minTime(time []int, m int) int {
        left, right, mid := 0, 0, 0
        for i := 0; i < len(time); i++ {
                right = right + time[i]
        }
        // 二分查找一个数mid, 使time数组能分割成m个和不小于mid的子数组
        res := math.MaxInt32
```

(续下页)

(接上页)

```
        for left <= right {
                mid = left + (right-left)/2
                if check(time, mid) <= m {
                        if mid < res {
                                res = mid
                        }
                        right = mid - 1
                } else {
                        left = mid + 1
                }
        }
        return res
}

func check(arr []int, mid int) int {
        res := 1
        maxValue := 0
        sum := 0
        for i := 0; i < len(arr); i++ {
                sum = sum + arr[i]
                if arr[i] > maxValue {
                        maxValue = arr[i]
                }
                if sum-maxValue > mid {
                        sum = arr[i]
                        maxValue = arr[i]
                        res++
                }
        }
        return res
}
```

81.10 LCP17. 速算机器人 (2)

- 题目

```
小扣在秋日市集发现了一款速算机器人。店家对机器人说出两个数字（记作 x 和␣
↪y），请小扣说出计算指令：
    "A" 运算：使 x = 2 * x + y；
    "B" 运算：使 y = 2 * y + x。
在本次游戏中，店家说出的数字为 x = 1 和 y = 0，小扣说出的计算指令记作仅由大写字母␣
↪A、B 组成的字符串 s，
```

(续下页)

(接上页)

```
字符串中字符的顺序表示计算顺序，请返回最终 x 与 y 的和为多少。
示例 1： 输入：s = "AB" 输出：4
解释：经过一次 A 运算后，x = 2, y = 0。
    再经过一次 B 运算，x = 2, y = 2。
    最终 x 与 y 之和为 4。
提示：0 <= s.length <= 10
    s 由 'A' 和 'B' 组成
```

- 解题思路

```
func calculate(s string) int {
        x, y := 1, 0
        for i := 0; i < len(s); i++ {
                if s[i] == 'A' {
                        x = 2*x + y
                } else if s[i] == 'B' {
                        y = 2*y + x
                }
        }
        return x + y
}

# 2
func calculate(s string) int {
        return 1 << len(s)
}
```

81.11 LCP18. 早餐组合 (3)

- 题目

```
小扣在秋日市集选择了一家早餐摊位，一维整型数组 staple 中记录了每种主食的价格，
一维整型数组 drinks␣
↪中记录了每种饮料的价格。小扣的计划选择一份主食和一款饮料，且花费不超过 x 元。
请返回小扣共有多少种购买方案。
注意：答案需要以 1e9 + 7 (1000000007)␣
↪为底取模，如：计算初始结果为：1000000008，请返回 1
示例 1：输入：staple = [10,20,5], drinks = [5,5,2], x = 15 输出：6
    解释：小扣有 6 种购买方案，所选主食与所选饮料在数组中对应的下标分别是：
    第 1 种方案：staple[0] + drinks[0] = 10 + 5 = 15;
    第 2 种方案：staple[0] + drinks[1] = 10 + 5 = 15;
    第 3 种方案：staple[0] + drinks[2] = 10 + 2 = 12;
```

(续下页)

(接上页)

```
    第 4 种方案：staple[2] + drinks[0] = 5 + 5 = 10；
    第 5 种方案：staple[2] + drinks[1] = 5 + 5 = 10；
    第 6 种方案：staple[2] + drinks[2] = 5 + 2 = 7。
示例 2：  输入：staple = [2,1,1], drinks = [8,9,5,1], x = 9 输出：8
    解释：小扣有 8 种购买方案，所选主食与所选饮料在数组中对应的下标分别是：
    第 1 种方案：staple[0] + drinks[2] = 2 + 5 = 7；
    第 2 种方案：staple[0] + drinks[3] = 2 + 1 = 3；
    第 3 种方案：staple[1] + drinks[0] = 1 + 8 = 9；
    第 4 种方案：staple[1] + drinks[2] = 1 + 5 = 6；
    第 5 种方案：staple[1] + drinks[3] = 1 + 1 = 2；
    第 6 种方案：staple[2] + drinks[0] = 1 + 8 = 9；
    第 7 种方案：staple[2] + drinks[2] = 1 + 5 = 6；
    第 8 种方案：staple[2] + drinks[3] = 1 + 1 = 2；
提示：1 <= staple.length <= 10^5
    1 <= drinks.length <= 10^5
    1 <= staple[i],drinks[i] <= 10^5
    1 <= x <= 2*10^5
```

- 解题思路

```
func breakfastNumber(staple []int, drinks []int, x int) int {
        sort.Ints(staple)
        sort.Ints(drinks)
        res := 0
        j := len(drinks) - 1
        for i := 0; i < len(staple); i++ {
                for j >= 0 && staple[i]+drinks[j] > x {
                        j--
                }
                res = (res + j + 1) % 1000000007
        }
        return res
}

# 2
func breakfastNumber(staple []int, drinks []int, x int) int {
        res := 0
        arr := make([]int, x+1)
        for i := 0; i < len(staple); i++ {
                if staple[i] < x {
                        arr[staple[i]]++
                }
        }
        for i := 1; i < len(arr); i++ {
```

(续下页)

(接上页)

```
                arr[i] = arr[i-1] + arr[i]
        }
        for i := 0; i < len(drinks); i++ {
                target := x - drinks[i]
                if target <= 0 {
                        continue
                }
                res = (res + arr[target]) % 1000000007
        }
        return res
}

# 3
func breakfastNumber(staple []int, drinks []int, x int) int {
        sort.Ints(staple)
        sort.Ints(drinks)
        res := 0
        for i := 0; i < len(staple); i++ {
                target := x - staple[i]
                if target <= 0 {
                        break
                }
                j := binarySearch(drinks, target)
                res = (res + j) % 1000000007
        }
        return res
}

func binarySearch(arr []int, target int) int {
        left, right := 0, len(arr)
        for left < right {
                mid := left + (right-left)/2
                if arr[mid] > target {
                        right = mid
                } else {
                        left = mid + 1
                }
        }
        return left
}
```

81.12 LCP19. 秋叶收藏集 (2)

- 题目

```
小扣出去秋游，途中收集了一些红叶和黄叶，他利用这些叶子初步整理了一份秋叶收藏集␣
↪leaves，
字符串 leaves 仅包含小写字符 r 和 y， 其中字符 r 表示一片红叶，字符 y 表示一片黄叶。
出于美观整齐的考虑，小扣想要将收藏集中树叶的排列调整成「红、黄、红」三部分。
每部分树叶数量可以不相等，但均需大于等于 1。每次调整操作，
小扣可以将一片红叶替换成黄叶或者将一片黄叶替换成红叶。
请问小扣最少需要多少次调整操作才能将秋叶收藏集调整完毕。
示例 1：输入：leaves = "rrryyyrryyyrr" 输出：2
解释：调整两次，将中间的两片红叶替换成黄叶，得到 "rrryyyyyyyyrr"
示例 2：输入：leaves = "ryr" 输出：0
解释：已符合要求，不需要额外操作
提示：3 <= leaves.length <= 10^5
    leaves 中只包含字符 'r' 和字符 'y'
```

- 解题思路

```
func minimumOperations(leaves string) int {
        n := len(leaves)
        // 长度i+1
        // dp[i][0] 全部变成r的步数
        // dp[i][1] 变成r...ry...y的步数
        // dp[i][2] 变成r...ry...yr...r的步数
        dp := make([][3]int, n)
        if leaves[0] == 'y' {
                dp[0][0] = 1 // 1个y变为r需要1步
        }
        for i := 1; i < n; i++ {
                if leaves[i] == 'r' {
                        dp[i][0] = dp[i-1][0]     // 不需要改变，同前一个
                        dp[i][1] = dp[i-1][0] + 1 // 全r +␣
↪当前r，需要改变一个y，步数+1
                        if i > 1 {
                                dp[i][1] = min(dp[i][1], dp[i-1][1]+1)
                                dp[i][2] = dp[i-1][1]
                        }
                        if i > 2 {
                                dp[i][2] = min(dp[i][2], dp[i-1][2])
                        }
                } else {
                        dp[i][0] = dp[i-1][0] + 1 // 需要改变，步数+1
```

(续下页)

(接上页)

```
                        dp[i][1] = dp[i-1][0]      // 前一个全r + 当前y,不需要改变
                        if i > 1 {
                                dp[i][1] = min(dp[i][1], dp[i-1][1]) // 同前一个不变
                                dp[i][2] = dp[i-1][1] + 1            // 调整+1
                        }
                        if i > 2 {
                                dp[i][2] = min(dp[i][2], dp[i-1][2]+1) //␣
↪在前一个基础上调整
                        }
                }
        }
        return dp[n-1][2]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func minimumOperations(leaves string) int {
        n := len(leaves)
        // 长度i+1
        // dp[i][0] 全部变成r的步数
        // dp[i][1] 变成r...ry...y的步数
        // dp[i][2] 变成r...ry...yr...r的步数
        dp := make([][3]int, n)
        maxValue := math.MaxInt32 / 10
        if leaves[0] == 'y' {
                dp[0][0] = 1 // 1个y变为r需要1步
                dp[0][1] = maxValue
                dp[0][2] = maxValue
        }
        for i := 1; i < n; i++ {
                dp[i][2] = maxValue
                dp[i][1] = maxValue
                if leaves[i] == 'r' {
                        dp[i][0] = dp[i-1][0]                       //␣
↪不需要改变，同前一个
                        dp[i][1] = min(dp[i-1][0]+1, dp[i-1][1]+1) //␣
↪前一个r+1，前一个y+1
```

(续下页)

(接上页)

```
                        if i >= 2 {
                                dp[i][2] = min(dp[i-1][1], dp[i-1][2]) //␣
↪前一个y不变，前一个r不变
                        }
                } else {
                        dp[i][0] = dp[i-1][0] + 1               // 需要改变，步数+1
                        dp[i][1] = min(dp[i-1][0], dp[i-1][1]) //␣
↪前一个r不变，前一个y不变
                        if i >= 2 {
                                dp[i][2] = min(dp[i-1][1]+1, dp[i-1][2]+1) //␣
↪前一个y+1，前一个r+1
                        }
                }
        }
        return dp[n-1][2]
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

81.13 LCP20. 快速公交 (1)

- 题目

```
小扣打算去秋日市集，由于游客较多，小扣的移动速度受到了人流影响：
小扣从 x 号站点移动至 x + 1 号站点需要花费的时间为 inc；
小扣从 x 号站点移动至 x - 1 号站点需要花费的时间为 dec。
现有 m 辆公交车，编号为 0 到 m-1。小扣也可以通过搭乘编号为 i 的公交车，从 x␣
↪号站点移动至 jump[i]*x 号站点，
耗时仅为 cost[i]。小扣可以搭乘任意编号的公交车且搭乘公交次数不限。
假定小扣起始站点记作 0，秋日市集站点记作␣
↪target，请返回小扣抵达秋日市集最少需要花费多少时间。
由于数字较大，最终答案需要对 1000000007 (1e9 + 7) 取模。
注意：小扣可在移动过程中到达编号大于 target 的站点。
示例 1：输入：target = 31, inc = 5, dec = 3, jump = [6], cost = [10] 输出：33
解释：小扣步行至 1 号站点，花费时间为 5；
小扣从 1 号站台搭乘 0 号公交至 6 * 1 = 6 站台，花费时间为 10；
小扣从 6 号站台步行至 5 号站台，花费时间为 3；
```

(续下页)

(接上页)

```
小扣从 5 号站台搭乘 0 号公交至 6 * 5 = 30 站台，花费时间为 10；
小扣从 30 号站台步行至 31 号站台，花费时间为 5；
最终小扣花费总时间为 33。
示例 2：输入：target = 612, inc = 4, dec = 5, jump = [3,6,8,11,5,10,4], cost = [4,7,6,
↪3,7,6,4] 输出：26
解释：小扣步行至 1 号站点，花费时间为 4；
小扣从 1 号站台搭乘 0 号公交至 3 * 1 = 3 站台，花费时间为 4；
小扣从 3 号站台搭乘 3 号公交至 11 * 3 = 33 站台，花费时间为 3；
小扣从 33 号站台步行至 34 站台，花费时间为 4；
小扣从 34 号站台搭乘 0 号公交至 3 * 34 = 102 站台，花费时间为 4；
小扣从 102 号站台搭乘 1 号公交至 6 * 102 = 612 站台，花费时间为 7；
最终小扣花费总时间为 26。
提示：1 <= target <= 10^9
1 <= jump.length, cost.length <= 10
2 <= jump[i] <= 10^6
1 <= inc, dec, cost[i] <= 10^6
```

- 解题思路

```
var mod = 1000000007

var res int

func busRapidTransit(target int, inc int, dec int, jump []int, cost []int) int {
        res = target * inc // 最坏的情况：全+1
        n := len(jump)
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        visited := make(map[int]bool)
        heap.Push(&intHeap, []int{0, target}) // 时间+当前位置：从后往前走
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([]int)
                t, cur := node[0], node[1]
                if t >= res { // 跳过
                        continue
                }
                res = min(res, t+cur*inc) // 用+1补
                for i := 0; i < n; i++ {
                        diff, next := cur%jump[i], cur/jump[i]
                        if diff == 0 { // 直接坐公交
                                if visited[next] == false {
                                        heap.Push(&intHeap, []int{t + cost[i], next})
                                }
                        } else {
```

(续下页)

(接上页)

```
                                if visited[next] == false { // 向左走坐公交
                                        heap.Push(&intHeap, []int{t + cost[i] +␣
↪diff*inc, next})
                                }
                                if visited[next+1] == false { // 向右走坐公交
                                        heap.Push(&intHeap, []int{t + cost[i] +␣
↪(jump[i]-diff)*dec, next + 1})
                                }
                        }
                }
        }
        return res % mod
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

type IntHeap [][]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][0] < h[j][0] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}

# 2
var mod = 1000000007

var visited map[int]int

func busRapidTransit(target int, inc int, dec int, jump []int, cost []int) int {
        visited = make(map[int]int)
        return dfs(inc, dec, jump, cost, target) % mod
```

(续下页)

(接上页)

```
}

func dfs(inc int, dec int, jump []int, cost []int, cur int) int {
        if cur == 0 {
                return 0
        }
        if cur == 1 {
                return inc
        }
        if visited[cur] > 0 {
                return visited[cur]
        }
        res := cur * inc // 最坏的情况：全+1
        for i := 0; i < len(jump); i++ {
                diff, next := cur%jump[i], cur/jump[i]
                if diff == 0 { // 直接坐公交
                        res = min(res, dfs(inc, dec, jump, cost, next)+cost[i])
                } else {
                        // 向左走坐公交
                        res = min(res, dfs(inc, dec, jump, cost,␣
↪next)+cost[i]+diff*inc)
                        // 向右走坐公交
                        res = min(res, dfs(inc, dec, jump, cost,␣
↪next+1)+cost[i]+(jump[i]-diff)*dec)
                }
        }
        visited[cur] = res
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

81.14 LCP22. 黑白方格画 (1)

- 题目

```
小扣注意到秋日市集上有一个创作黑白方格画的摊位。摊主给每个顾客提供一个固定在墙上的白色画板，画板不能
画板上有 n * n 的网格。绘画规则为，小扣可以选择任意多行以及任意多列的格子涂成黑色，
所选行数、列数均可为 0。
小扣希望最终的成品上需要有 k 个黑色格子，请返回小扣共有多少种涂色方案。
注意：两个方案中任意一个相同位置的格子颜色不同，就视为不同的方案。
示例 1：输入：n = 2, k = 2 输出：4
解释：一共有四种不同的方案：
第一种方案：涂第一列；
第二种方案：涂第二列；
第三种方案：涂第一行；
第四种方案：涂第二行。
示例 2：输入：n = 2, k = 1 输出：0
解释：不可行，因为第一次涂色至少会涂两个黑格。
示例 3：输入：n = 2, k = 4 输出：1
解释：共有 2*2=4 个格子，仅有一种涂色方案。
限制：1 <= n <= 6
0 <= k <= n * n
```

- 解题思路

```
func paintingPlan(n int, k int) int {
        if k == n*n || k == 0 { // 全部涂满或者不涂只有1种方案
                return 1
        }
        if k < n { // 最少大于等于n
                return 0
        }
        res := 0
        // 暴力枚举行和列
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        a := i * n
                        b := j * n
                        if a+b-i*j == k {
                                res = res + C(n, i)*C(n, j) // 求组合数
                        }
                }
        }
        return res
}
```

(续下页)

(接上页)

```
func C(n, m int) int {
        a := 1
        for i := 1; i <= m; i++ {
                a = a * (n - i + 1)
        }

        b := 1
        for i := 1; i <= m; i++ {
                b = b * i
        }
        return a / b
}
```

81.15 LCP23. 魔术排列 (1)

- 题目

```
秋日市集上，魔术师邀请小扣与他互动。魔术师的道具为分别写有数字 1~N 的 N␣
↪张卡牌，然后请小扣思考一个 N 张卡牌的排列 target。
魔术师的目标是找到一个数字 k（k >= 1），使得初始排列顺序为 1~N␣
↪的卡牌经过特殊的洗牌方式最终变成小扣所想的排列 target，
特殊的洗牌方式为：
第一步，魔术师将当前位于 偶数位置 的卡牌（下标自 1 开始），保持 当前排列顺序 放在位于␣
↪奇数位置 的卡牌之前。
例如：将当前排列 [1,2,3,4,5] 位于偶数位置的 [2,4] 置于奇数位置的 [1,3,5] 前，排列变为␣
↪[2,4,1,3,5]；
第二步，若当前卡牌数量小于等于 k，则魔术师按排列顺序取走全部卡牌；
若当前卡牌数量大于 k，则取走前 k␣
↪张卡牌，剩余卡牌继续重复这两个步骤，直至所有卡牌全部被取走；
卡牌按照魔术师取走顺序构成的新排列为「魔术取数排列」，请返回是否存在这个数字 k␣
↪使得「魔术取数排列」恰好就是 target，
从而让小扣感到大吃一惊。
示例 1：输入：target = [2,4,3,1,5] 输出：true
解释：排列 target 长度为 5，初始排列为：1,2,3,4,5。我们选择 k = 2：
第一次：将当前排列 [1,2,3,4,5] 位于偶数位置的 [2,4] 置于奇数位置的 [1,3,5]␣
↪前，排列变为 [2,4,1,3,5]。
取走前 2 张卡牌 2,4，剩余 [1,3,5]；
第二次：将当前排列 [1,3,5] 位于偶数位置的 [3] 置于奇数位置的 [1,5] 前，排列变为 [3,1,
↪5]。
取走前 2 张 3,1，剩余 [5]；
第三次：当前排列为 [5]，全部取出。
```

(续下页)

(接上页)

```
最后，数字按照取出顺序构成的「魔术取数排列」2,4,3,1,5 恰好为 target。
示例 2：输入：target = [5,4,3,2,1] 出：false
解释：无法找到一个数字 k 可以使「魔术取数排列」恰好为 target。
提示：1 <= target.length = N <= 5000
题目保证 target 是 1~N 的一个排列。
```

- 解题思路

```
func isMagic(target []int) bool {
        n := len(target)
        arr := make([]int, n) // 构建初始数组
        for i := 0; i < n; i++ {
                arr[i] = i + 1
        }
        arr = Shuffle(arr)
        k := 0 // 找到k：可以证明只能正好匹配到k个；其中1~N不重复
        for ; k < n && arr[k] == target[k]; k++ {
        }
        if k == 0 { // 不满足
                return false
        }
        for { // 按规则模拟
                if k >= len(arr) {
                        return judge(arr, target, len(arr))
                }
                if judge(arr, target, k) == false {
                        return false
                }
                arr = arr[k:]
                arr = Shuffle(arr)
                target = target[k:]
        }
        return false
}

func Shuffle(arr []int) []int {
        temp := make([]int, 0)
        for i := 1; i < len(arr); i = i + 2 {
                temp = append(temp, arr[i])
        }
        for i := 0; i < len(arr); i = i + 2 {
                temp = append(temp, arr[i])
        }
        return temp
```

(续下页)

(接上页)

```
}

func judge(a, b []int, k int) bool {
        for i := 0; i < k; i++ {
                if a[i] != b[i] {
                        return false
                }
        }
        return true
}
```

81.16 LCP28. 采购方案 (3)

- 题目

```
小力将 N 个零件的报价存于数组 nums。小力预算为 target，假定小力仅购买两个零件，
要求购买零件的花费不超过预算，请问他有多少种采购方案。
注意：答案需要以 1e9 + 7 (1000000007)␣
↪为底取模，如：计算初始结果为：1000000008，请返回 1
示例 1：输入：nums = [2,5,3,5], target = 6 输出：1
解释：预算内仅能购买 nums[0] 与 nums[2]。
示例 2：输入：nums = [2,2,1,9], target = 10 输出：4
解释：符合预算的采购方案如下：
nums[0] + nums[1] = 4
nums[0] + nums[2] = 3
nums[1] + nums[2] = 3
nums[2] + nums[3] = 10
提示：2 <= nums.length <= 10^5
1 <= nums[i], target <= 10^5
```

- 解题思路

```
func purchasePlans(nums []int, target int) int {
        sort.Ints(nums)
        j := len(nums) - 1
        res := 0
        for i := 0; i < len(nums); i++ {
                for i < j {
                        if nums[i]+nums[j] <= target {
                                break
                        }
                        j--
```

(续下页)

(接上页)

```
                }
                if i < j {
                        res = res + (j - i)
                }
        }
        return res % 1000000007
}

# 2
func purchasePlans(nums []int, target int) int {
        sort.Ints(nums)
        res := 0
        left, right := 0, len(nums)-1
        for left < right {
                for left < right && nums[left]+nums[right] > target {
                        right--
                }
                res = res + right - left
                left++
        }
        return res % 1000000007
}

# 3
func purchasePlans(nums []int, target int) int {
        sort.Ints(nums)
        ln := len(nums)
        res := 0
        for i := 0; i < ln; i++ {
                target := target - nums[i]
                index := search(nums[i+1:], target)
                res = res + index
        }
        return res % 1000000007
}

func search(nums []int, target int) int {
        left := 0
        right := len(nums) - 1
        for left <= right {
                mid := left + (right-left)/2
                if nums[mid] > target {
                        right = mid - 1
```

(续下页)

(接上页)

```
                } else {
                        left = mid + 1
                }
        }
        return left
}
```

81.17 LCP29. 乐团站位 (2)

- 题目

```
某乐团的演出场地可视作 num * num 的二维矩阵 grid (左上角坐标为 [0,
↪0]), 每个位置站有一位成员。
乐团共有 9 种乐器, 乐器编号为 1~9, 每位成员持有 1 个乐器。
为保证声乐混合效果, 成员站位规则为: 自 grid 左上角开始顺时针螺旋形向内循环以 1, 2, ...
↪, 9 循环重复排列。
例如当 num = 5 时, 站位如图所示
请返回位于场地坐标 [Xpos,Ypos] 的成员所持乐器编号。
示例 1: 输入: num = 3, Xpos = 0, Ypos = 2 输出: 3
解释:
示例 2: 输入: num = 4, Xpos = 1, Ypos = 2 输出: 5
解释:
提示: 1 <= num <= 10^9
0 <= Xpos, Ypos < num
```

- 解题思路

```
func orchestraLayout(num int, xPos int, yPos int) int {
        x := min(xPos, num-1-xPos)
        y := min(yPos, num-1-yPos)
        k := min(x, y) // 在第几圈 (从0开始)
        // n*n - (n-2k)*(n-2k) = n*n-(n*n+4k*k-4nk)= 4nk-4k*k
        total := 4 * k * (num - k) % 9 // 第几圈外总共的个数
        if xPos == k {                  // 上边
                return (total+yPos-k)%9 + 1
        } else if yPos == num-1-k { // 右边
                before := num - 2*k - 1
                return (total+before+xPos-k)%9 + 1
        } else if xPos == num-1-k { // 下边
                before := (num - 2*k - 1) * 2
                return (total+before+num-k-1-yPos)%9 + 1
        } else if yPos == k { // 左边
```

(续下页)

(接上页)

```
                before := (num - 2*k - 1) * 3
                return (total+before+num-k-1-xPos)%9 + 1
        }
        return 0
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func orchestraLayout(num int, xPos int, yPos int) int {
        x := min(xPos, num-1-xPos)
        y := min(yPos, num-1-yPos)
        k := min(x, y) // 在第几圈（从0开始）
        // n*n - (n-2k)*(n-2k) = n*n-(n*n+4k*k-4nk)= 4nk-4k*k
        if xPos <= yPos {
                total := num*num - (num-2*k)*(num-2*k)
                return (total+xPos-k+yPos-k)%9 + 1
        } else {
                total := num*num - (num-(2*k+2))*(num-(2*k+2))
                return (total-(xPos-k)-(yPos-k))%9 + 1
        }
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}
```

81.18 LCP30. 魔塔游戏 (2)

- 题目

```
小扣当前位于魔塔游戏第一层，共有 N 个房间，编号为 0 ~ N-1。
每个房间的补血道具/怪物对于血量影响记于数组nums，其中正数表示道具补血数值，即血量增加对应数值；
负数表示怪物造成伤害值，即血量减少对应数值；0 表示房间对血量无影响。
小扣初始血量为 1，且无上限。假定小扣原计划按房间编号升序访问所有房间补血/打怪，为保证血量始终为正值，
小扣需对房间访问顺序进行调整，每次仅能将一个怪物房间（负数的房间）调整至访问顺序末尾。
请返回小扣最少需要调整几次，才能顺利访问所有房间。若调整顺序也无法访问完全部房间，请返回-1。
示例 1：输入：nums = [100,100,100,-250,-60,-140,-50,-50,100,150] 输出：1
解释：初始血量为 1。至少需要将 nums[3] 调整至访问顺序末尾以满足要求。
示例 2：输入：nums = [-200,-300,400,0]输出：-1
解释：调整访问顺序也无法完成全部房间的访问。
提示：1 <= nums.length <= 10^5
-10^5 <= nums[i] <= 10^5
```

- 解题思路

```
func magicTower(nums []int) int {
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        blood := 0
        sum := 0
        res := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
                if nums[i] < 0 {
                        heap.Push(&intHeap, nums[i])
                        if blood+nums[i] < 0 {
                                res++
                                minValue := heap.Pop(&intHeap).(int)
                                blood = blood - minValue
                        }
                }
                blood = blood + nums[i]
        }
        if sum < 0 {
                return -1
        }
        return res
```

(续下页)

(接上页)

```
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}

# 2
func magicTower(nums []int) int {
        sum := 0
        for i := 0; i < len(nums); i++ {
                sum = sum + nums[i]
        }
        if sum < 0 {
                return -1
        }
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        blood := 0
        res := 0
        for i := 0; i < len(nums); i++ {
                heap.Push(&intHeap, nums[i])
                blood = blood + nums[i]
```

(续下页)

(接上页)

```
                if blood < 0 {
                        minValue := heap.Pop(&intHeap).(int)
                        blood = blood - minValue
                        res++
                }
        }
        return res
}

type IntHeap []int

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i] < h[j]
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

81.19 LCP33. 蓄水 (2)

- 题目

```
给定 N 个无限容量且初始均空的水缸，每个水缸配有一个水桶用来打水，
第 i 个水缸配备的水桶容量记作 bucket[i]。小扣有以下两种操作：
升级水桶：选择任意一个水桶，使其容量增加为 bucket[i]+1
蓄水：将全部水桶接满水，倒入各自对应的水缸
```

(续下页)

(接上页)

```
每个水缸对应最低蓄水量记作␣
↪vat[i]，返回小扣至少需要多少次操作可以完成所有水缸蓄水要求。
注意：实际蓄水量 达到或超过 最低蓄水量，即完成蓄水要求。
示例 1：输入：bucket = [1,3], vat = [6,8] 输出：4
解释：第 1 次操作升级 bucket[0];
第 2 ~ 4 次操作均选择蓄水，即可完成蓄水要求。
示例 2：输入：bucket = [9,0,1], vat = [0,2,2] 输出：3
解释：第 1 次操作均选择升级 bucket[1]
第 2~3 次操作选择蓄水，即可完成蓄水要求。
提示：1 <= bucket.length == vat.length <= 100
0 <= bucket[i], vat[i] <= 10^4
```

- 解题思路

```
func storeWater(bucket []int, vat []int) int {
        n := len(vat)
        maxValue := 0
        for i := 0; i < n; i++ {
                maxValue = max(maxValue, vat[i])
        }
        if maxValue == 0 {
                return 0
        }
        res := math.MaxInt32
        for k := 1; k <= maxValue; k++ { // 枚举蓄水的次数
                temp := k
                for i := 0; i < n; i++ {
                        begin := vat[i] / k // 需要升级到的目的容量
                        if vat[i]%k > 0 {
                                begin++
                        }
                        if begin > bucket[i] {
                                temp = temp + begin - bucket[i] // 升级次数
                        }
                }
                res = min(res, temp)
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

# 2
func storeWater(bucket []int, vat []int) int {
        n := len(vat)
        nodeHeap := make(IntHeap, 0)
        heap.Init(&nodeHeap)
        count := 0 // 需要升级的次数
        for i := 0; i < n; i++ {
                if bucket[i] == 0 && vat[i] > 0 {
                        bucket[i] = 1
                        count++
                }
                if vat[i] > 0 {
                        heap.Push(&nodeHeap, Node{
                                bucket: bucket[i],
                                vat:    vat[i],
                                count:  (vat[i]-1)/bucket[i] + 1,
                        })
                }
        }
        res := math.MaxInt32 // 总次数
        for nodeHeap.Len() > 0 {
                node := heap.Pop(&nodeHeap).(Node)
                if count >= res {
                        break
                }
        res = min(res, node.count+count) // 堆里面最大的蓄水次数+升级的次数
                heap.Push(&nodeHeap, Node{
                        bucket: node.bucket + 1,
                        vat:    node.vat,
                        count:  (node.vat-1)/(node.bucket+1) + 1,
                })
                count++
        }
```

(续下页)

(接上页)

```
        if res == math.MaxInt32 {
                return 0
        }
        return res
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

type Node struct {
        bucket int
        vat    int
        count  int
}

type IntHeap []Node

func (h IntHeap) Len() int {
        return len(h)
}

// 小根堆<,大根堆变换方向>
func (h IntHeap) Less(i, j int) bool {
        return h[i].count > h[j].count
}

func (h IntHeap) Swap(i, j int) {
        h[i], h[j] = h[j], h[i]
}

func (h *IntHeap) Push(x interface{}) {
        *h = append(*h, x.(Node))
}

func (h *IntHeap) Pop() interface{} {
        value := (*h)[len(*h)-1]
        *h = (*h)[:len(*h)-1]
        return value
}
```

81.20 LCP34. 二叉树染色 (1)

- 题目

```
小扣有一个根结点为 root␣
↪的二叉树模型，初始所有结点均为白色，可以用蓝色染料给模型结点染色，
模型的每个结点有一个 val 价值。
小扣出于美观考虑，希望最后二叉树上每个蓝色相连部分的结点个数不能超过 k 个，
求所有染成蓝色的结点价值总和最大是多少？
示例 1：输入：root = [5,2,3,4], k = 2输出：12
解释：结点 5、3、4 染成蓝色，获得最大的价值 5+3+4=12
示例 2：输入：root = [4,1,3,9,null,null,2], k = 2 输出：16
解释：结点 4、3、9 染成蓝色，获得最大的价值 4+3+9=16
提示：1 <= k <= 10
1 <= val <= 10000
1 <= 结点数量 <= 10000
```

- 解题思路

```
func maxValue(root *TreeNode, k int) int {
        dp := dfs(root, k)
        return maxArr(dp)
}

func dfs(root *TreeNode, k int) []int {
        dp := make([]int, k+1) // dp[i]表示，染色数为i的最大值
        if root == nil {
                return dp
        }
        left := dfs(root.Left, k)
        right := dfs(root.Right, k)
        dp[0] = maxArr(left) + maxArr(right) // 当前节点不染色
        for i := 1; i <= k; i++ {            // 当前节点染色
                for j := 0; j < i; j++ {
                        dp[i] = max(dp[i], left[j]+right[i-1-j]+root.Val)
                }
        }
        return dp
}

func max(a, b int) int {
        if a > b {
                return a
        }
```

(续下页)

(接上页)

```
        return b
}

func maxArr(arr []int) int {
        res := 0
        for i := 0; i < len(arr); i++ {
                res = max(res, arr[i])
        }
        return res
}
```

81.21 LCP35. 电动车游城市 (1)

- 题目

```
小明的电动车电量充满时可行驶距离为 cnt，每行驶 1 单位距离消耗 1 单位电量，且花费 1␣
↪单位时间。
小明想选择电动车作为代步工具。地图上共有 N 个景点，景点编号为 0 ~ N-1。
他将地图信息以 [城市 A 编号,城市 B 编号,两城市间距离] 格式整理在在二维数组␣
↪paths，表示城市 A、B 间存在双向通路。
初始状态，电动车电量为 0。每个城市都设有充电桩，charge[i] 表示第 i 个城市每充 1␣
↪单位电量需要花费的单位时间。
请返回小明最少需要花费多少单位时间从起点城市 start 抵达终点城市 end。
示例 1：输入：paths = [[1,3,3],[3,2,1],[2,1,3],[0,1,4],[3,0,5]], cnt = 6,
start = 1, end = 0, charge = [2,10,4,1] 输出：43
解释：最佳路线为：1->3->0。
在城市 1 仅充 3 单位电至城市 3，然后在城市 3 充 5 单位电，行驶至城市 5。
充电用时共 3*10 + 5*1= 35
行驶用时 3 + 5 = 8，此时总用时最短 43。
示例 2：输入：paths = [[0,4,2],[4,3,5],[3,0,5],[0,1,5],[3,2,4],[1,2,8]], cnt = 8,
start = 0, end = 2, charge = [4,1,1,3,2] 输出：38
解释：最佳路线为：0->4->3->2。
城市 0 充电 2 单位，行驶至城市 4 充电 8 单位，行驶至城市 3 充电 1␣
↪单位，最终行驶至城市 2。
充电用时 4*2+2*8+3*1 = 27
行驶用时 2+5+4 = 11，总用时最短 38。
提示：1 <= paths.length <= 200
paths[i].length == 3
2 <= charge.length == n <= 100
0 <= path[i][0],path[i][1],start,end < n
1 <= cnt <= 100
1 <= path[i][2] <= cnt
```

(续下页)

(接上页)

```
1 <= charge[i] <= 100
题目保证所有城市相互可以到达
```

- 解题思路

```
func electricCarPlan(paths [][]int, cnt int, start int, end int, charge []int) int {
        n := len(charge)
        arr := make([][][]int, n) // 邻接表
        for i := 0; i < len(paths); i++ {
                a, b, c := paths[i][0], paths[i][1], paths[i][2] // a<=>b
                arr[a] = append(arr[a], []int{b, c})
                arr[b] = append(arr[b], []int{a, c})
        }
        dis := make([][]int, n) // start到i点在j电量下的花费
        for i := 0; i < n; i++ {
                dis[i] = make([]int, cnt+1)
                for j := 0; j < cnt+1; j++ {
                        dis[i][j] = math.MaxInt32
                }
        }
        dis[start][0] = 0 // 开始花费为0
        intHeap := make(IntHeap, 0)
        heap.Init(&intHeap)
        heap.Push(&intHeap, []int{0, start, 0}) // 时间+位置+电量：按时间堆排序
        for intHeap.Len() > 0 {
                node := heap.Pop(&intHeap).([]int)
                t, cur, value := node[0], node[1], node[2]
                if t > dis[cur][value] { // 大于跳过
                        continue
                }
                if cur == end { // 终点，直接返回
                        return t
                }
                // 核心点：可以在一个城市充满电，可以不需要在其它城市充电；
                //␣
↪这样可以尝试在一个城市充满一定电量后往下走；不一定非要完全充满电或者只充到达下一个城市所需的电量
                // 因为每个城市的充电单价不一样
                if value < cnt { // 去第cur城市充电1单位；入堆后可以继续充电1单位
                        nextTime := t + charge[cur]
                        if nextTime < dis[cur][value+1] {
                                dis[cur][value+1] = nextTime
                                heap.Push(&intHeap, []int{nextTime, cur, value + 1})
                        }
                }
```

(续下页)

(接上页)

```
                // 电量满足到达下一个城市后可以开往下一个城市
                for i := 0; i < len(arr[cur]); i++ {
                        next, nextDis := arr[cur][i][0], arr[cur][i][1]
                        if value >= nextDis && t+nextDis < dis[next][value-nextDis] {
                                dis[next][value-nextDis] = t + nextDis
                                heap.Push(&intHeap, []int{t + nextDis, next, value -␣
↪nextDis})
                        }
                }
        }
        return -1
}

type IntHeap [][]int

func (h IntHeap) Len() int            { return len(h) }
func (h IntHeap) Less(i, j int) bool  { return h[i][0] < h[j][0] }
func (h IntHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h *IntHeap) Push(x interface{}) { *h = append(*h, x.([]int)) }
func (h *IntHeap) Pop() interface{} {
        old := *h
        n := len(old)
        x := old[n-1]
        *h = old[0 : n-1]
        return x
}
```

81.22 LCP36. 最多牌组数

81.22.1 题目

```
麻将的游戏规则中，共有两种方式凑成「一组牌」：
顺子：三张牌面数字连续的麻将，例如 [4,5,6]
刻子：三张牌面数字相同的麻将，例如 [10,10,10]
给定若干数字作为麻将牌的数值（记作一维数组 tiles），请返回所给 tiles␣
↪最多可组成的牌组数。
注意：凑成牌组时，每张牌仅能使用一次。
示例 1：输入：tiles = [2,2,2,3,4] 输出：1
解释：最多可以组合出 [2,2,2] 或者 [2,3,4] 其中一组牌。
示例 2：输入：tiles = [2,2,2,3,4,1,3] 输出：2
解释：最多可以组合出 [1,2,3] 与 [2,3,4] 两组牌。
```

(续下页)

(接上页)

```
提示：1 <= tiles.length <= 10^5
1 <= tiles[i] <= 10^9
```

81.22.2 解题思路

```

```

81.23 LCP39. 无人机方阵 (1)

- 题目

```
在 「力扣挑战赛」 开幕式的压轴节目 「无人机方阵」中，每一架无人机展示一种灯光颜色。␣
↪无人机方阵通过两种操作进行颜色图案变换：
调整无人机的位置布局
切换无人机展示的灯光颜色
给定两个大小均为 N*M 的二维数组 source 和 target 表示无人机方阵表演的两种颜色图案，
由于无人机切换灯光颜色的耗能很大，请返回从 source 到 target␣
↪最少需要多少架无人机切换灯光颜色。
注意： 调整无人机的位置布局时无人机的位置可以随意变动。
示例 1：输入：source = [[1,3],[5,4]], target = [[3,1],[6,5]] 输出：1
解释：最佳方案为
将 [0,1] 处的无人机移动至 [0,0] 处；
将 [0,0] 处的无人机移动至 [0,1] 处；
将 [1,0] 处的无人机移动至 [1,1] 处；
将 [1,1] 处的无人机移动至 [1,0] 处，其灯光颜色切换为颜色编号为 6 的灯光；
因此从source 到 target 所需要的最少灯光切换次数为 1。
示例 2：输入：source = [[1,2,3],[3,4,5]], target = [[1,3,5],[2,3,4]]输出：0
解释：仅需调整无人机的位置布局，便可完成图案切换。因此不需要无人机切换颜色
提示：n == source.length == target.length
m == source[i].length == target[i].length
1 <= n, m <=100
1 <= source[i][j], target[i][j] <=10^4
```

- 解题思路

```
func minimumSwitchingTimes(source [][]int, target [][]int) int {
        temp := make(map[int]int)
        n, m := len(source), len(source[0])
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
```

(续下页)

(接上页)

```
                        temp[source[i][j]]++
                        temp[target[i][j]]--
                }
        }
        res := 0
        for _, v := range temp {
                if v > 0 {
                        res = res + v
                }
        }
        return res
}
```

81.24 LCP40. 心算挑战 (3)

- 题目

```
「力扣挑战赛」心算项目的挑战比赛中，要求选手从 N 张卡牌中选出 cnt 张卡牌，若这 cnt␣
↪张卡牌数字总和为偶数，
则选手成绩「有效」且得分为 cnt 张卡牌数字总和。
给定数组 cards 和 cnt，其中 cards[i] 表示第 i 张卡牌上的数字。␣
↪请帮参赛选手计算最大的有效得分。
若不存在获取有效得分的卡牌方案，则返回 0。
示例 1：输入：cards = [1,2,8,9], cnt = 3 输出：18
解释：选择数字为 1、8、9 的这三张卡牌，此时可获得最大的有效得分 1+8+9=18。
示例 2：输入：cards = [3,3,1], cnt = 1 输出：0
解释：不存在获取有效得分的卡牌方案。
提示：1 <= cnt <= cards.length <= 10^5
1 <= cards[i] <= 1000
```

- 解题思路

```
func maxmiumScore(cards []int, cnt int) int {
        a, b := make([]int, 0), make([]int, 0)
        for i := 0; i < len(cards); i++ {
                if cards[i]%2 == 0 {
                        a = append(a, cards[i])
                } else {
                        b = append(b, cards[i])
                }
        }
        sort.Slice(a, func(i, j int) bool {
```

(续下页)

(接上页)

```
                return a[i] > a[j]
        })
        sort.Slice(b, func(i, j int) bool {
                return b[i] > b[j]
        })
        x, y := len(a), len(b)
        arrA, arrB := make([]int, x+1), make([]int, y+1)
        for i := 0; i < x; i++ {
                arrA[i+1] = arrA[i] + a[i]
        }
        for i := 0; i < y; i++ {
                arrB[i+1] = arrB[i] + b[i]
        }
        res := 0
        for i := 0; i <= cnt; i++ { // 枚举奇数的个数
                n, m := cnt-i, i
                if n <= x && m <= y && (arrA[n]+arrB[m])%2 == 0 {
                        res = max(res, arrA[n]+arrB[m])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
func maxmiumScore(cards []int, cnt int) int {
        sort.Slice(cards, func(i, j int) bool {
                return cards[i] > cards[j]
        })
        a, b := make([]int, 1), make([]int, 1)
        for i := 0; i < len(cards); i++ {
                if cards[i]%2 == 0 {
                        a = append(a, cards[i]+a[len(a)-1])
                } else {
                        b = append(b, cards[i]+b[len(b)-1])
                }
        }
```

(续下页)

(接上页)

```
        res := 0
        for i := 0; i <= cnt; i++ { // 枚举奇数的个数
                n, m := cnt-i, i
                if n < len(a) && m < len(b) && (a[n]+b[m])%2 == 0 {
                        res = max(res, a[n]+b[m])
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 3
func maxmiumScore(cards []int, cnt int) int {
        sort.Slice(cards, func(i, j int) bool {
                return cards[i] > cards[j]
        })
        res := 0
        sum := 0
        for i := 0; i < cnt; i++ {
                sum = sum + cards[i]
        }
        if sum%2 == 0 { // 偶数直接返回
                return sum
        }
        // 使用不同奇偶性的数替换:
        // 1、找个一个较小的数替换: cards[cnt-1]
        // 2、找到一个较小的数替换: 跟cards[cnt-1]不同的较小的数
        for i := cnt; i < len(cards); i++ { //␣
↪情况1: 在后面找1个数替换前cnt个最大数的最后1个数
                if cards[i]%2 != cards[cnt-1]%2 {
                        res = max(res, sum-cards[cnt-1]+cards[i])
                        break
                }
        }
        for i := cnt - 2; i >= 0; i-- { // 情况2: 尝试找到1个奇偶性不同于cards[cnt-
↪1]的数, 然后替换掉
                if cards[i]%2 != cards[cnt-1]%2 {
```

(续下页)

(接上页)

```
                        for j := cnt; j < len(cards); j++ {
                                if cards[j]%2 != cards[i]%2 {
                                        res = max(res, sum-cards[i]+cards[j])
                                }
                        }
                        break
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

81.25 LCP41. 黑白翻转棋 (1)

- 题目

```
在 n*m 大小的棋盘中，有黑白两种棋子，黑棋记作字母 "X", 白棋记作字母 "O", 空余位置记作 "."。
当落下的棋子与其他相同颜色的棋子在行、列或对角线完全包围（中间不存在空白位置）另一种颜色的棋子，则可翻转这些棋子。
「力扣挑战赛」黑白翻转棋项目中，将提供给选手一个未形成可翻转棋子的棋盘残局，其状态记作 chessboard。
若下一步可放置一枚黑棋，请问选手最多能翻转多少枚白棋。
注意：若翻转白棋成黑棋后，棋盘上仍存在可以翻转的白棋，将可以 继续 翻转白棋
输入数据保证初始棋盘状态无可以翻转的棋子且存在空余位置
示例 1：输入：chessboard = ["....X.","....X.","XOOO..","......","......"] 输出：3
解释：可以选择下在 [2,4] 处，能够翻转白方三枚棋子。
示例 2：输入：chessboard = [".X.",".O.","XO."] 输出：2
解释：可以选择下在 [2,2] 处，能够翻转白方两枚棋子。
示例 3：输入：chessboard = [".......",".......",".......","X......",".O.....","..O....","....OOX"] 输出：4
解释：  可以选择下在 [6,3] 处，能够翻转白方四枚棋子。
提示：1 <= chessboard.length, chessboard[i].length <= 8
chessboard[i] 仅包含 "."、"O" 和 "X"
```

- 解题思路

```
var res int
var n, m int
var dx = []int{-1, 1, 0, 0, 1, 1, -1, -1}
var dy = []int{0, 0, -1, 1, 1, -1, -1, 1}

func flipChess(chessboard []string) int {
        res = 0
        n, m = len(chessboard), len(chessboard[0])
        temp := make([][]byte, n)
        for i := 0; i < n; i++ {
                for j := 0; j < m; j++ {
                        if chessboard[i][j] == '.' {
                                for a := 0; a < n; a++ {
                                        temp[a] = []byte(chessboard[a])
                                }
                                temp[i][j] = 'X'
                                dfs(temp, i, j) // 尝试在i,j位置下黑棋
                                count := 0
                                for a := 0; a < n; a++ { // 统计结果
                                        for b := 0; b < m; b++ {
                                                if temp[a][b] == 'X' &&␣
↪chessboard[a][b] == 'O' {
                                                        count++
                                                }
                                        }
                                }
                                res = max(res, count) // 更新结果
                        }
                }
        }
        return res
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

func dfs(arr [][]byte, i, j int) {
        arr[i][j] = 'X'
        for k := 0; k < 8; k++ {
                if ok, path := judge(arr, i, j, 'X', dx[k], dy[k]); ok == true {
```

(续下页)

(接上页)

```
                        for c := 0; c < len(path); c++ {
                                a, b := path[c][0], path[c][1]
                                arr[a][b] = 'X'
                                dfs(arr, a, b)
                        }
                }
        }
}

// leetcode1958.检查操作是否合法
func judge(board [][]byte, rMove int, cMove int, color byte, dirX, dirY int) (res␣
↪bool, path [][]int) {
        x, y := rMove+dirX, cMove+dirY
        count := 1
        for 0 <= x && x < n && 0 <= y && y < m {
                if board[x][y] == '.' {
                        return false, nil
                }
                path = append(path, []int{x, y})
                if count == 1 {
                        if board[x][y] == color {
                                return false, nil
                        }
                } else {
                        if board[x][y] == color {
                                return true, path
                        }
                }
                count++
                x = x + dirX
                y = y + dirY
        }
        return false, nil
}
```

81.26 LCP42. 玩具套圈

81.26.1 题目

81.26.2 解题思路

81.27 LCP44. 开幕式焰火 (1)

- 题目

```
「力扣挑战赛」开幕式开始了，空中绽放了一颗二叉树形的巨型焰火。
给定一棵二叉树 root␣
↪代表焰火，节点值表示巨型焰火这一位置的颜色种类。请帮小扣计算巨型焰火有多少种不同的颜色。
示例 1：输入：root = [1,3,2,1,null,2] 输出：3
解释：焰火中有 3 个不同的颜色，值分别为 1、2、3
示例 2：输入：root = [3,3,3] 输出：1
解释：焰火中仅出现 1 个颜色，值为 3
提示：1 <= 节点个数 <= 1000
1 <= Node.val <= 1000
```

- 解题思路

```
var m map[int]int

func numColor(root *TreeNode) int {
        m = make(map[int]int)
        dfs(root)
        return len(m)
}

func dfs(root *TreeNode) {
        if root != nil {
                m[root.Val] = 1
                dfs(root.Left)
                dfs(root.Right)
        }
}
```

81.28 LCP45. 自行车炫技赛场 (2)

- 题目

```
「力扣挑战赛」中 N*M 大小的自行车炫技赛场的场地由一片连绵起伏的上下坡组成，
场地的高度值记录于二维数组 terrain 中，场地的减速值记录于二维数组 obstacle 中。
若选手骑着自行车从高度为 h1 且减速值为 o1 的位置到高度为 h2 且减速值为 o2␣
↪的相邻位置（上下左右四个方向），
速度变化值为 h1-h2-o2（负值减速，正值增速）。
选手初始位于坐标 position 处且初始速度为␣
↪1，请问选手可以刚好到其他哪些位置时速度依旧为 1。
请以二维数组形式返回这些位置。若有多个位置则按行坐标升序排列，若有多个位置行坐标相同则按列坐标升序排
注意： 骑行过程中速度不能为零或负值
示例 1：输入：position = [0,0], terrain = [[0,0],[0,0]], obstacle = [[0,0],[0,0]]␣
↪输出：[[0,1],[1,0],[1,1]]
解释：由于当前场地属于平地，根据上面的规则，选手从[0,
↪0]的位置出发都能刚好在其他处的位置速度为 1。
示例 2：输入：position = [1,1], terrain = [[5,0],[0,6]], obstacle = [[0,6],[7,0]]␣
↪输出：[[0,1]]
解释：选手从 [1,1] 处的位置出发，到 [0,1] 处的位置时恰好速度为 1。
提示：n == terrain.length == obstacle.length
m == terrain[i].length == obstacle[i].length
1 <= n <= 100
1 <= m <= 100
0 <= terrain[i][j], obstacle[i][j] <= 100
position.length == 2
0 <= position[0] < n
0 <= position[1] < m
```

- 解题思路

```
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var res [][]int
var visited map[[3]int]bool
var n, m int
var originX, originY int

func bicycleYard(position []int, terrain [][]int, obstacle [][]int) [][]int {
        res = make([][]int, 0)
        n, m = len(terrain), len(terrain[0])
        originX, originY = position[0], position[1]
        visited = make(map[[3]int]bool)
        visited[[3]int{originX, originY, 1}] = true
```

(续下页)

(接上页)

```
        dfs(terrain, obstacle, originX, originY, 1)
        sort.Slice(res, func(i, j int) bool {
                if res[i][0] == res[j][0] {
                        return res[i][1] < res[j][1]
                }
                return res[i][0] < res[j][0]
        })
        return res
}

func dfs(terrain [][]int, obstacle [][]int, i, j int, speed int) {
        if speed == 1 && (i == originX && j == originY) == false {
                res = append(res, []int{i, j})
        }
        for k := 0; k < 4; k++ {
                x, y := i+dx[k], j+dy[k]
                if 0 <= x && x < n && 0 <= y && y < m {
                        // next = speed + h1-h2-o2
                        next := speed + (terrain[i][j] - terrain[x][y] -␣
↪obstacle[x][y])
                        if next > 0 && visited[[3]int{x, y, next}] == false {
                                visited[[3]int{x, y, next}] = true
                                dfs(terrain, obstacle, x, y, next)
                        }
                }
        }
}

# 2
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}

func bicycleYard(position []int, terrain [][]int, obstacle [][]int) [][]int {
        res := make([][]int, 0)
        n, m := len(terrain), len(terrain[0])
        originX, originY := position[0], position[1]
        visited := make(map[[3]int]bool)
        visited[[3]int{originX, originY, 1}] = true
        queue := make([][3]int, 0)
        queue = append(queue, [3]int{originX, originY, 1})
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
```

(续下页)

(接上页)

```
                i, j, speed := node[0], node[1], node[2]
                if speed == 1 && (i == originX && j == originY) == false {
                        res = append(res, []int{i, j})
                }
                for k := 0; k < 4; k++ {
                        x, y := i+dx[k], j+dy[k]
                        if 0 <= x && x < n && 0 <= y && y < m {
                                // next = speed + h1-h2-o2
                                next := speed + (terrain[i][j] - terrain[x][y] -␣
↪obstacle[x][y])
                                if next > 0 && visited[[3]int{x, y, next}] == false {
                                        visited[[3]int{x, y, next}] = true
                                        queue = append(queue, [3]int{x, y, next})
                                }
                        }
                }
        }
        sort.Slice(res, func(i, j int) bool {
                if res[i][0] == res[j][0] {
                        return res[i][1] < res[j][1]
                }
                return res[i][0] < res[j][0]
        })
        return res
}
```

81.29 LCP46. 志愿者调配 (1)

- 题目

```
「力扣挑战赛」有 n 个比赛场馆（场馆编号从 0␣
↪开始），场馆之间的通道分布情况记录于二维数组 edges 中，
edges[i]= [x, y] 表示第 i 条通道连接场馆 x 和场馆 y(即两个场馆相邻)。
初始每个场馆中都有一定人数的志愿者（不同场馆人数可能不同），后续 m␣
↪天每天均会根据赛事热度进行志愿者人数调配。
调配方案分为如下三种：
将编号为 idx 的场馆内的志愿者人数减半；
将编号为 idx 的场馆相邻的场馆的志愿者人数都加上编号为 idx 的场馆的志愿者人数；
将编号为 idx 的场馆相邻的场馆的志愿者人数都减去编号为 idx 的场馆的志愿者人数。
所有的调配信息记录于数组 plans 中，plans[i] = [num,idx] 表示第 i 天对编号 idx␣
↪的场馆执行了第 num 种调配方案。
在比赛结束后对调配方案进行复盘时，不慎将第 0␣
```

(续下页)

(接上页)

```
↪个场馆的最终志愿者人数丢失，只保留了初始所有场馆的志愿者总人数 totalNum ，
以及记录了第 1 ~ n-1 个场馆的最终志愿者人数的一维数组 finalCnt。
请你根据现有的信息求出初始每个场馆的志愿者人数，并按场馆编号顺序返回志愿者人数列表。
注意：测试数据保证当某场馆进行第一种调配时，该场馆的志愿者人数一定为偶数；
测试数据保证当某场馆进行第三种调配时，该场馆的相邻场馆志愿者人数不为负数；
测试数据保证比赛开始时每个场馆的志愿者人数都不超过 10^9；
测试数据保证给定的场馆间的道路分布情况中不会出现自环、重边的情况。
示例 1：输入：finalCnt = [1,16], totalNum = 21, edges = [[0,1],[1,2]],
plans = [[2,1],[1,0],[3,0]] 输出：[5,7,9]
解释：
示例 2 ：输入：finalCnt = [4,13,4,3,8], totalNum = 54, edges = [[0,3],[1,3],[4,3],[2,
↪3],[2,5]],
plans = [[1,1],[3,3],[2,5],[1,0]] 输出：[10,16,9,4,7,8]
提示：2 <= n <= 5*10^4
1 <= edges.length <= min((n * (n - 1)) / 2, 5*10^4)
0 <= edges[i][0], edges[i][1] < n
1 <= plans.length <= 10
1 <= plans[i][0] <=3
0 <= plans[i][1] < n
finalCnt.length = n-1
0 <= finalCnt[i] < 10^9
0 <= totalNum < 5*10^13
```

- 解题思路

```
type Node struct {
        a float64 // 未知final[0] x的系数
        b int     // 已知的系数
}

func volunteerDeployment(finalCnt []int, totalNum int64, edges [][]int, plans␣
↪[][]int) []int {
        n := len(finalCnt) + 1
        arr := make([][]int, n) // 邻接表
        for i := 0; i < len(edges); i++ {
                a, b := edges[i][0], edges[i][1]
                arr[a] = append(arr[a], b)
                arr[b] = append(arr[b], a)
        }
        final := make([]Node, 0)
        final = append(final, Node{a: 1, b: 0}) //␣
↪final为最终的结果，a为final[0]未知x的系数
        for i := 0; i < len(finalCnt); i++ {
                final = append(final, Node{a: 0, b: finalCnt[i]})
```

(续下页)

(接上页)

```
        }
        for i := len(plans) - 1; i >= 0; i-- { // 倒推
                a, b := plans[i][0], plans[i][1]
                if a == 1 {
                        final[b] = Node{
                                a: final[b].a * 2,
                                b: final[b].b * 2,
                        }
                } else if a == 2 {
                        for j := 0; j < len(arr[b]); j++ {
                                next := arr[b][j]
                                final[next] = Node{
                                        a: final[next].a - final[b].a,
                                        b: final[next].b - final[b].b,
                                }
                        }
                } else if a == 3 {
                        for j := 0; j < len(arr[b]); j++ {
                                next := arr[b][j]
                                final[next] = Node{
                                        a: final[next].a + final[b].a,
                                        b: final[next].b + final[b].b,
                                }
                        }
                }
        }
        a, b := float64(0), int64(0)
        for i := 0; i < len(final); i++ {
                a = a + final[i].a
                b = b + int64(final[i].b)
        }
        per := float64(totalNum-b) / a // 求x的值
        res := make([]int, n)
        for i := 0; i < n; i++ {
                res[i] = int(final[i].a*per) + final[i].b
        }
        return res
}
```

81.30 LCP47. 入场安检 (2)

- 题目

```
「力扣挑战赛」 的入场仪式马上就要开始了，由于安保工作的需要，
设置了可容纳人数总和为 M 的 N 个安检室，capacities[i] 记录第 i␣
↪个安检室可容纳人数。安检室拥有两种类型：
先进先出：在安检室中的所有观众中，最早进入安检室的观众最先离开
后进先出：在安检室中的所有观众中，最晚进入安检室的观众最先离开
恰好 M+1 位入场的观众（编号从 0 开始）需要排队依次入场安检， 入场安检的规则如下：
观众需要先进入编号 0 的安检室
当观众将进入编号 i 的安检室时（0 <= i < N），
若安检室未到达可容纳人数上限，该观众可直接进入；
若安检室已到达可容纳人数上限，在该观众进入安检室之前需根据当前安检室类型选择一位观众离开后才能进入；
当观众离开编号 i 的安检室时 （0 <= i < N-1），将进入编号 i+1 的安检室接受安检。
若可以任意设定每个安检室的类型，请问有多少种设定安检室类型的方案可以使得编号 k␣
↪的观众第一个通过最后一个安检室入场。
注意：观众不可主动离开安检室，只有当安检室容纳人数达到上限，且又有新观众需要进入时，才可根据安检室的
由于方案数可能过大，请将答案对 1000000007 取模后返回。
示例 1：输入：capacities = [2,2,3], k = 2 输出：2
解释：存在两种设定的 2 种方案：
方案 1：将编号为 0 、1 的实验室设置为 后进先出 的类型，编号为 2 的实验室设置为␣
↪先进先出 的类型；
方案 2：将编号为 0 、1 的实验室设置为 先进先出 的类型，编号为 2 的实验室设置为␣
↪后进先出 的类型。
以下是方案 1 的示意图：
示例 2：输入：capacities = [3,3], k = 3 输出：0
示例 3：输入：capacities = [4,3,2,2], k = 6 输出：2
提示:1 <= capacities.length <= 200
1 <= capacities[i] <= 200
0 <= k <= sum(capacities)
```

- 解题思路

```
var mod = 1000000007

func securityCheck(capacities []int, k int) int {
        n := len(capacities)
        dp := make([][]int, n+1) // dp[i][j] => 使得i个实验室的情况下第j个人␣
↪第1个离开的方案数量
        for i := 0; i <= n; i++ {
                dp[i] = make([]int, k+1)
        }
        dp[0][0] = 1
```

(续下页)

(接上页)

```
        for i := 1; i <= n; i++ {
                // 先进先出不改变 入场顺序
                // 改变为 后进先出 的入场顺序：会留住 capacities[i]-1 个人
                // 把c（capacities[i]-1）看成物品大小，k看成背包容量，等价求 01背包的方案数。
                // 从后向前遍历
                c := capacities[i-1] - 1
                for j := 0; j <= k; j++ {
                        dp[i][j] = dp[i-1][j]
                        if j >= c {
                                dp[i][j] = (dp[i][j] + dp[i-1][j-c]) % mod
                        }
                }
        }
        return dp[n][k]
}

# 2
var mod = 1000000007

func securityCheck(capacities []int, k int) int {
        dp := make([]int, k+1) // dp[i] => 使得第i个人 第1个离开的方案数量
        dp[0] = 1
        for i := 0; i < len(capacities); i++ {
                // 先进先出不改变 入场顺序
                // 改变为 后进先出 的入场顺序：会留住 capacities[i]-1 个人
                // 把c（capacities[i]-1）看成物品大小，k看成背包容量，等价求 01背包的方案数。
                // 从后向前遍历
                c := capacities[i] - 1
                for j := k; j >= c; j-- {
                        dp[j] = (dp[j] + dp[j-c]) % mod
                }
        }
        return dp[k]
}
```

81.31 LCP50. 宝石补给 (1)

- 题目

```
欢迎各位勇者来到力扣新手村，在开始试炼之前，请各位勇者先进行「宝石补给」。
每位勇者初始都拥有一些能量宝石， gem[i] 表示第 i␣
↪位勇者的宝石数量。现在这些勇者们进行了一系列的赠送，
operations[j] = [x, y] 表示在第 j 次的赠送中 第 x␣
↪位勇者将自己一半的宝石（需向下取整）赠送给第 y 位勇者。
在完成所有的赠送后，请找到拥有最多宝石的勇者和拥有最少宝石的勇者，并返回他们二者的宝石数量之差。
注意：赠送将按顺序逐步进行。
示例 1：输入：gem = [3,1,2], operations = [[0,2],[2,1],[2,0]]输出：2
解释：第 1 次操作，勇者 0 将一半的宝石赠送给勇者 2， gem = [2,1,3]
第 2 次操作，勇者 2 将一半的宝石赠送给勇者 1， gem = [2,2,2]
第 3 次操作，勇者 2 将一半的宝石赠送给勇者 0， gem = [3,2,1]
返回 3 - 1 = 2
示例 2：输入：gem = [100,0,50,100], operations = [[0,2],[0,1],[3,0],[3,0]] 输出：75
解释：第 1 次操作，勇者 0 将一半的宝石赠送给勇者 2， gem = [50,0,100,100]
第 2 次操作，勇者 0 将一半的宝石赠送给勇者 1， gem = [25,25,100,100]
第 3 次操作，勇者 3 将一半的宝石赠送给勇者 0， gem = [75,25,100,50]
第 4 次操作，勇者 3 将一半的宝石赠送给勇者 0， gem = [100,25,100,25]
返回 100 - 25 = 75
示例 3：输入：gem = [0,0,0,0], operations = [[1,2],[3,1],[1,2]] 输出：0
提示：2 <= gem.length <= 10^3
0 <= gem[i] <= 10^3
0 <= operations.length <= 10^4
operations[i].length == 2
0 <= operations[i][0], operations[i][1] < gem.length
```

- 解题思路

```
func giveGem(gem []int, operations [][]int) int {
        for i := 0; i < len(operations); i++ {
                a, b := operations[i][0], operations[i][1]
                v := gem[a] / 2
                gem[a] = gem[a] - v
                gem[b] = gem[b] + v
        }
        maxValue, minValue := math.MinInt32, math.MaxInt32
        for i := 0; i < len(gem); i++ {
                maxValue = max(maxValue, gem[i])
                minValue = min(minValue, gem[i])
        }
        return maxValue - minValue
```

(续下页)

(接上页)

```
}

func min(a, b int) int {
        if a > b {
                return b
        }
        return a
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

81.32 LCP51. 烹饪料理 (2)

- 题目

```
欢迎各位勇者来到力扣城，城内设有烹饪锅供勇者制作料理，为自己恢复状态。
勇者背包内共有编号为 0 ~ 4 的五种食材，其中 materials[j] 表示第 j 种食材的数量。
通过这些食材可以制作若干料理，cookbooks[i][j] 表示制作第 i 种料理需要第 j␣
↪种食材的数量，
而 attribute[i] = [x,y] 表示第 i 道料理的美味度 x 和饱腹感 y。
在饱腹感不小于 limit␣
↪的情况下，请返回勇者可获得的最大美味度。如果无法满足饱腹感要求，则返回 -1。
注意：每种料理只能制作一次。
示例 1：输入：materials = [3,2,4,1,2] cookbooks = [[1,1,0,1,2],[2,1,4,0,0],[3,2,4,1,
↪0]]
attribute = [[3,2],[2,4],[7,6]]  limit = 5  输出：7
解释：食材数量可以满足以下两种方案：
方案一：制作料理 0 和料理 1，可获得饱腹感 2+4、美味度 3+2
方案二：仅制作料理 2， 可饱腹感为 6、美味度为 7
因此在满足饱腹感的要求下，可获得最高美味度 7
示例 2：输入：materials = [10,10,10,10,10] cookbooks = [[1,1,1,1,1],[3,3,3,3,3],[10,
↪10,10,10,10]]
attribute = [[5,5],[6,6],[10,10]] limit = 1 输出：11
解释：通过制作料理 0 和 1，可满足饱腹感，并获得最高美味度 11
提示：materials.length == 5
1 <= cookbooks.length == attribute.length <= 8
cookbooks[i].length == 5
```

(续下页)

(接上页)

```
attribute[i].length == 2
0 <= materials[i], cookbooks[i][j], attribute[i][j] <= 20
1 <= limit <= 100
```

- 解题思路

```
var res int

func perfectMenu(materials []int, cookbooks [][]int, attribute [][]int, limit int)␣
↪int {
        res = -1
        arr := make([]int, 5)
        dfs(materials, cookbooks, attribute, limit, 0, 0, arr, 0)
        return res
}

func dfs(materials []int, cookbooks [][]int, attribute [][]int, limit int, sumX, sumY␣
↪int, arr []int, index int) {
        flag := true
        for i := 0; i < 5; i++ {
                if arr[i] > materials[i] {
                        flag = false
                        break
                }
        }
        if flag == false {
                return
        }
        if sumY >= limit {
                res = max(res, sumX)
        }
        if index >= len(cookbooks) {
                return
        }
        tempA, tempB := make([]int, 5), make([]int, 5)
        copy(tempA, arr)
        copy(tempB, arr)
        for i := 0; i < 5; i++ {
                tempA[i] = tempA[i] + cookbooks[index][i]
        }
        dfs(materials, cookbooks, attribute, limit, sumX+attribute[index][0],␣
↪sumY+attribute[index][1], tempA, index+1) // 加上
        dfs(materials, cookbooks, attribute, limit, sumX, sumY, tempB, index+1)       ␣
↪                                  // 不加上
```

(续下页)

(接上页)

```
}

func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}

# 2
var res int

func perfectMenu(materials []int, cookbooks [][]int, attribute [][]int, limit int)␣
↪int {
        res = -1
        arr := make([]int, 5)
        dfs(materials, cookbooks, attribute, limit, 0, 0, arr, 0)
        return res
}

func dfs(materials []int, cookbooks [][]int, attribute [][]int, limit int, sumX, sumY␣
↪int, arr []int, index int) {
        if sumY >= limit {
                res = max(res, sumX)
        }
        if index >= len(cookbooks) {
                return
        }
        tempA, tempB := make([]int, 5), make([]int, 5)
        copy(tempA, arr)
        copy(tempB, arr)
        dfs(materials, cookbooks, attribute, limit, sumX, sumY, tempB, index+1) //␣
↪不加上
        for i := 0; i < 5; i++ {
                tempA[i] = tempA[i] + cookbooks[index][i]
                if tempA[i] > materials[i] {
                        return
                }
        }
        dfs(materials, cookbooks, attribute, limit, sumX+attribute[index][0],␣
↪sumY+attribute[index][1], tempA, index+1) // 加上
}
```

(续下页)

(接上页)

```
func max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

81.33 LCP52. 二叉搜索树染色 (2)

- 题目

```
欢迎各位勇者来到力扣城，本次试炼主题为「二叉搜索树染色」。
每位勇士面前设有一个二叉搜索树的模型，模型的根节点为root，树上的各个节点值均不重复。初始时，所有节点均为蓝色。
现在按顺序对这棵二叉树进行若干次操作， ops[i] = [type, x, y] 表示第 i 次操作为：
type 等于 0 时，将节点值范围在 [x, y] 的节点均染蓝
type 等于 1 时，将节点值范围在 [x, y] 的节点均染红
请返回完成所有染色后，该二叉树中红色节点的数量。
注意：题目保证对于每个操作的 x、y 值定出现在二叉搜索树节点中
示例 1：输入：root = [1,null,2,null,3,null,4,null,5], ops = [[1,2,4],[1,1,3],[0,3,5]]输出：2
解释：第 0 次操作，将值为 2、3、4 的节点染红；
第 1 次操作，将值为 1、2、3 的节点染红；
第 2 次操作，将值为 3、4、5 的节点染蓝；
因此，最终值为 1、2 的节点为红色节点，返回数量 2
示例 2：输入：root = [4,2,7,1,null,5,null,null,null,null,6] ops = [[0,2,2],[1,1,5],[0,4,5],[1,5,7]]
输出：5
解释：第 0 次操作，将值为 2 的节点染蓝；
第 1 次操作，将值为 1、2、4、5 的节点染红；
第 2 次操作，将值为 4、5 的节点染蓝；
第 3 次操作，将值为 5、6、7 的节点染红；
因此，最终值为 1、2、5、6、7 的节点为红色节点，返回数量 5
提示：1 <= 二叉树节点数量 <= 10^5
1 <= ops.length <= 10^5
ops[i].length == 3
ops[i][0] 仅为 0 or 1
0 <= ops[i][1] <= ops[i][2] <= 10^9
0 <= 节点值 <= 10^9
```

- 解题思路

```
var arr []int

func getNumber(root *TreeNode, ops [][]int) int {
        res := 0 // 蓝色的数量
        n := len(arr)
        arr = make([]int, 0)
        dfs(root)
        for i := len(ops) - 1; i >= 0; i-- {
                if len(arr) == 0 {
                        break
                }
                t, left, right := ops[i][0], ops[i][1], ops[i][2]
                index := sort.Search(len(arr), func(j int) bool {
                        return arr[j] >= left
                })
                temp := make([]int, index)
                copy(temp, arr[:index])
                var k int
                for k = index; k < len(arr) && arr[k] <= right; k++ {
                        if t == 0 { // 染蓝
                                res++
                        }
                }
                temp = append(temp, arr[k:]...)
                arr = temp
        }
        res = res + len(arr) // 染蓝的数量+开始蓝色的数量
        return n - res       // 红色数量
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
        arr = append(arr, root.Val)
        dfs(root.Right)
}

# 2
var arr []int

func getNumber(root *TreeNode, ops [][]int) int {
        res := 0 // 红色数量
```

(续下页)

(接上页)

```
        arr = make([]int, 0)
        dfs(root)
        for i := 0; i < len(arr); i++ {
                for j := len(ops) - 1; j >= 0; j-- {
                        t, left, right := ops[j][0], ops[j][1], ops[j][2]
                        if left <= arr[i] && arr[i] <= right {
                                res = res + t // 红色+1, 蓝色+0
                                break
                        }
                }
        }
        return res // 红色数量
}

func dfs(root *TreeNode) {
        if root == nil {
                return
        }
        dfs(root.Left)
        arr = append(arr, root.Val)
        dfs(root.Right)
}
```

81.34 LCP55. 采集果实 (1)

- 题目

```
欢迎各位勇者来到力扣新手村，本次训练内容为「采集果实」。
在新手村中，各位勇者需要采集一些果实来制作药剂。time[i]
表示勇者每次采集 1~limit 颗第 i 种类型的果实需要的时间（即每次最多可以采集 limit␣
↪颗果实）。
当前勇者需要完成「采集若干批果实」的任务， fruits[j] = [type, num] 表示第 j␣
↪批需要采集 num 颗 type 类型的果实。
采集规则如下：
按 fruits 给定的顺序依次采集每一批次
采集完当前批次的果实才能开始采集下一批次
勇者完成当前批次的采集后将清空背包（即多余的果实将清空）
请计算并返回勇者完成采集任务最少需要的时间。
示例 1：输入：time = [2,3,2], fruits = [[0,2],[1,4],[2,1]], limit = 3 输出：10
解释： 由于单次最多采集 3 颗
第 0 批需要采集 2 颗第 0 类型果实，需要采集 1 次，耗时为 2*1=2
第 1 批需要采集 4 颗第 1 类型果实，需要采集 2 次，耗时为 3*2=6
```

(续下页)

(接上页)

```
第 2 批需要采集 1 颗第 2 类型果实，需要采集 1 次，耗时为 2*1=2
返回总耗时 2+6+2=10
示例 2：输入：time = [1], fruits = [[0,3],[0,5]], limit = 2 输出：5
解释： 由于单次最多采集 2 颗
第 0 批需要采集 3 颗第 0 类型果实，需要采集 2 次，耗时为 1*2=2
第 1 批需要采集 5 颗第 0 类型果实，需要采集 3 次，耗时为 1*3=3
需按照顺序依次采集，返回 2+3=5
提示：1 <= time.length <= 100
1 <= time[i] <= 100
1 <= fruits.length <= 10^3
0 <= fruits[i][0] < time.length
1 <= fruits[i][1] < 10^3
1 <= limit <= 100
```

- 解题思路

```
func getMinimumTime(time []int, fruits [][]int, limit int) int {
        res := 0
        for i := 0; i < len(fruits); i++ {
                a, b := fruits[i][0], fruits[i][1]
                res = res + ((b-1)/limit+1)*time[a]
        }
        return res
}
```

81.35 LCP56. 信物传送 (2)

- 题目

```
欢迎各位勇者来到力扣城，本次试炼主题为「信物传送」。
本次试炼场地设有若干传送带，matrix[i][j] 表示第 i 行 j 列的传送带运作方向，"^","v","<
↪","
这四种符号分别表示 上、下、左、右 四个方向。
信物会随传送带的方向移动。勇者每一次施法操作，可临时变更一处传送带的方向，在物品经过后传送带恢复原方
通关信物初始位于坐标 start处，勇者需要将其移动到坐标 end␣
↪处，请返回勇者施法操作的最少次数。
注意：start 和 end 的格式均为 [i,j]
示例 1:输入：matrix = [">>v","v^<","<><"], start = [0,1], end = [2,0] 输出：1
解释：如上图所示
当信物移动到 [1,1] 时，勇者施法一次将 [1,1] 的传送方向 ^ 从变更为 <
从而信物移动到 [1,0]，后续到达 end 位置
因此勇者最少需要施法操作 1 次
```

(续下页)

(接上页)

```
示例 2:输入：matrix = [">>v",">>v","^<<"], start = [0,0], end = [1,1] 输出：0
解释：勇者无需施法，信物将自动传送至 end 位置
示例 3:输入：matrix = [">^^>","<^v>","^v^<"], start = [0,0], end = [1,3] 输出：3
提示：matrix 中仅包含 '^'、'v'、'<'、'>'
0 < matrix.length <= 100
0 < matrix[i].length <= 100
0 <= start[0],end[0] < matrix.length
0 <= start[1],end[1] < matrix[i].length
```

- 解题思路

```
// 顺时针：上右下左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var dir = []byte{'>', 'v', '<', '^'} // 注意对准坐标方向

func conveyorBelt(matrix []string, start []int, end []int) int {
        n, m := len(matrix), len(matrix[0])
        total := n * m
        arr := make([][]int, n) // 到达下标i,j的最少次数
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        arr[i][j] = total
                }
        }
        arr[start[0]][start[1]] = 0
        queue := make([][2]int, 0)
        queue = append(queue, [2]int{start[0], start[1]})
        for len(queue) > 0 {
                node := queue[0]
                queue = queue[1:]
                x, y := node[0], node[1]
                for i := 0; i < 4; i++ {
                        newX, newY := x+dx[i], y+dy[i]
                        if 0 <= newX && newX < n && 0 <= newY && newY < m {
                                if matrix[x][y] == dir[i] { // 不变换方向
                                        if arr[newX][newY] > arr[x][y] {
                                                arr[newX][newY] = arr[x][y] //␣
↪更新次数
                                                queue = append(queue, [2]int{newX,␣
↪newY})
                                        }
                                } else { // 变换方向，次数+1
```

(续下页)

(接上页)

```
                                        if arr[newX][newY] > arr[x][y]+1 {
                                                arr[newX][newY] = arr[x][y] + 1 //␣
↪更新次数
                                                queue = append(queue, [2]int{newX,␣
↪newY})
                                        }
                                }
                        }
                }
        }
        return arr[end[0]][end[1]]
}

# 2
// 顺时针：上右下左
var dx = []int{0, 1, 0, -1}
var dy = []int{1, 0, -1, 0}
var dir = []byte{'>', 'v', '<', '^'} // 注意对准坐标方向

var arr [][]int
var n, m int

func conveyorBelt(matrix []string, start []int, end []int) int {
        n, m = len(matrix), len(matrix[0])
        total := n * m
        arr = make([][]int, n) // 到达下标i,j的最少次数
        for i := 0; i < n; i++ {
                arr[i] = make([]int, m)
                for j := 0; j < m; j++ {
                        arr[i][j] = total
                }
        }
        dfs(matrix, start[0], start[1], 0)
        return arr[end[0]][end[1]]
}

func dfs(matrix []string, x, y int, count int) {
        if x < 0 || x >= n || y < 0 || y >= m {
                return
        }
        if count >= arr[x][y] {
                return
        }
```

(续下页)

(接上页)

```
        arr[x][y] = count
        index := 0
        for i := 0; i < 4; i++ {
                if matrix[x][y] == dir[i] {
                        index = i
                        break
                }
        }
        dfs(matrix, x+dx[index], y+dy[index], count) // 先走不变换方向：次数不变
        for i := 0; i < 4; i++ {
                newX, newY := x+dx[i], y+dy[i]
                if matrix[x][y] == dir[i] {
                        continue
                }
                dfs(matrix, newX, newY, count+1) // 不同次数+1
        }
}
```

81.36 LCP57. 打地鼠

81.36.1 题目

```
欢迎各位勇者来到力扣城，本次试炼主题为「打地鼠」。
勇者面前有一个大小为 3*3 的打地鼠游戏机，地鼠将随机出现在各个位置，moles[i] = [t,x,y]
表示在第 t 秒会有地鼠出现在 (x,y) 位置上，并于第 t+1 秒该地鼠消失。
勇者有一把可敲打地鼠的锤子，初始时刻（即第 0 秒）锤子位于正中间的格子 (1,
↪1)，锤子的使用规则如下：
锤子每经过 1 秒可以往上、下、左、右中的一个方向移动一格，也可以不移动
锤子只可敲击所在格子的地鼠，敲击不耗时
请返回勇者最多能够敲击多少只地鼠。
注意：输入用例保证在相同时间相同位置最多仅有一只地鼠
示例 1：输入： moles = [[1,1,0],[2,0,1],[4,2,2]] 输出： 2
解释：第 0 秒，锤子位于 (1,1)
第 1 秒，锤子移动至 (1,0) 并敲击地鼠
第 2 秒，锤子移动至 (2,0)
第 3 秒，锤子移动至 (2,1)
第 4 秒，锤子移动至 (2,2) 并敲击地鼠
因此勇者最多可敲击 2 只地鼠
示例 2：输入：moles = [[2,0,2],[5,2,0],[4,1,0],[1,2,1],[3,0,2]] 输出：3
解释：第 0 秒，锤子位于 (1,1)
第 1 秒，锤子移动至 (2,1) 并敲击地鼠
```

(续下页)

(接上页)

```
第 2 秒，锤子移动至 (1,1)
第 3 秒，锤子移动至 (1,0)
第 4 秒，锤子在 (1,0) 不移动并敲击地鼠
第 5 秒，锤子移动至 (2,0) 并敲击地鼠
因此勇者最多可敲击 3 只地鼠
示例 3：输入：moles = [[0,1,0],[0,0,1]] 输出：0
解释：第 0 秒，锤子初始位于 (1,1)，此时并不能敲击 (1,0)、(0,1) 位置处的地鼠
提示：1 <= moles.length <= 10^5
moles[i].length == 3
0 <= moles[i][0] <= 10^9
0 <= moles[i][1], moles[i][2] < 3
```

81.36.2 解题思路

CHAPTER 82

LCS

82.1 LCS01. 下载插件 (3)

- 题目

```
小扣打算给自己的 VS code 安装使用插件，初始状态下带宽每分钟可以完成 1个插件的下载。假定每分钟选择以下两种策略之一：
使用当前带宽下载插件
将带宽加倍（下载插件数量随之加倍）
请返回小扣完成下载 n 个插件最少需要多少分钟。
注意：实际的下载的插件数量可以超过 n 个
示例 1：输入：n = 2 输出：2
解释：以下两个方案，都能实现 2 分钟内下载 2 个插件
方案一：第一分钟带宽加倍，带宽可每分钟下载 2 个插件；第二分钟下载 2 个插件
方案二：第一分钟下载 1 个插件，第二分钟下载 1 个插件
示例 2：输入：n = 4 输出：3
解释：最少需要 3 分钟可完成 4 个插件的下载，以下是其中一种方案：
第一分钟带宽加倍，带宽可每分钟下载 2 个插件；
第二分钟下载 2 个插件；
第三分钟下载 2 个插件。
提示：1 <= n <= 10^5
```

- 解题思路

```
func leastMinutes(n int) int {
        res := 0
        for i := 1; i < n; i = i * 2 {
                res++
        }
        return res + 1
}

# 2
func leastMinutes(n int) int {
        return bits.Len(uint(n)-1)+1
}

# 3
func leastMinutes(n int) int {
        res := int(math.Ceil(math.Log(float64(n)) / math.Log(2)))
        return res + 1
}
```

82.2 LCS02. 完成一半题目 (1)

- 题目

```
有 N 位扣友参加了微软与力扣举办了「以扣会友」线下活动。
主办方提供了 2*N 道题目，整型数组 questions␣
↪中每个数字对应了每道题目所涉及的知识点类型。
若每位扣友选择不同的一题，请返回被选的 N 道题目至少包含多少种知识点类型。
示例 1：输入：questions = [2,1,6,2] 输出：1
解释：有 2 位扣友在 4 道题目中选择 2 题。
可选择完成知识点类型为 2 的题目时，此时仅一种知识点类型
因此至少包含 1 种知识点类型。
示例 2：输入：questions = [1,5,1,3,4,5,2,5,3,3,8,6] 输出：2
解释：有 6 位扣友在 12 道题目中选择题目，需要选择 6 题。
选择完成知识点类型为 3、5 的题目，因此至少包含 2 种知识点类型。
提示：questions.length == 2*n
2 <= questions.length <= 10^5
1 <= questions[i] <= 1000
```

- 解题思路

```
func halfQuestions(questions []int) int {
        res := 0
        n := len(questions)
```

(续下页)

(接上页)

```
        arr := make([]int, 1001)
        for i := 0; i < n; i++ {
                arr[questions[i]]++
        }
        sort.Ints(arr)
        count := n / 2
        for i := 1000; i >= 0; i-- {
                res++
                count = count - arr[i]
                if count <= 0 {
                        break
                }
        }
        return res
}
```

82.3 LCS03. 主题空间 (1)

- 题目

```
「以扣会友」线下活动所在场地由若干主题空间与走廊组成，场地的地图记作由一维字符串型数组grid，
字符串中仅包含 "0"～"5" 这 6 个字符。
地图上每一个字符代表面积为 1 的区域，其中 "0" 表示走廊，其他字符表示主题空间。
相同且连续（连续指上、下、左、右四个方向连接）的字符组成同一个主题空间。
假如整个 grid区域的外侧均为走廊。请问，不与走廊直接相邻的主题空间的最大面积是多少？如果不存在这样的空间请返回0。
示例 1:输入：grid = ["110","231","221"] 输出：1
解释：4 个主题空间中，只有 1 个不与走廊相邻，面积为 1。
示例 2:输入：grid = ["11111100000","21243101111","21224101221","11111101111"] 输出：3
解释：8 个主题空间中，有 5 个不与走廊相邻，面积分别为 3、1、1、1、2，最大面积为 3。
提示：1 <= grid.length <= 500
1 <= grid[i].length <= 500
grid[i][j] 仅可能是 "0"～"5"
```

- 解题思路

```
var count int
var flag bool

func largestArea(grid []string) int {
```

(续下页)

(接上页)

```
        res := 0
        for i := 0; i < len(grid); i++ {
                for j := 0; j < len(grid[i]); j++ {
                        if '1' <= grid[i][j] && grid[i][j] <= '5' {
                                count, flag = 0, true
                                dfs(grid, i, j, grid[i][j])
                                if flag == true && count > res {
                                        res = count
                                }
                        }
                }
        }
        return res
}

func dfs(grid []string, i, j int, char byte) {
        if i < 0 || i >= len(grid) || j < 0 || j >= len(grid[0]) || grid[i][j] == '0'
↪{ // 不满足条件
                flag = false
                return
        }
        if grid[i][j] != char { // 不一致
                return
        }
        count++
        arr := []byte(grid[i])
        arr[j] = arr[j] + 5 // 变换为其它的数
        grid[i] = string(arr)
        dfs(grid, i+1, j, char)
        dfs(grid, i-1, j, char)
        dfs(grid, i, j+1, char)
        dfs(grid, i, j-1, char)
        return
}
```

CHAPTER 83

PAT (Basic Level) Practice 乙级

83.1 1001. 害死人不偿命的 (3n+1) 猜想 (1)

- 题目

```
卡拉兹(Callatz)猜想：
对任何一个正整数 n，如果它是偶数，那么把它砍掉一半；如果它是奇数，那么把 (3n+1)␣
↪砍掉一半。
这样一直反复砍下去，最后一定在某一步得到 n=1。卡拉兹在 1950␣
↪年的世界数学家大会上公布了这个猜想，
传说当时耶鲁大学师生齐动员，拼命想证明这个貌似很傻很天真的命题，结果闹得学生们无心学业，
一心只证␣
↪(3n+1)，以至于有人说这是一个阴谋，卡拉兹是在蓄意延缓美国数学界教学与科研的进展……
我们今天的题目不是证明卡拉兹猜想，而是对给定的任一不超过 1000 的正整数 n，
简单地数一下，需要多少步（砍几下）才能得到 n=1？
输入格式：每个测试输入包含 1 个测试用例，即给出正整数 n 的值。
输出格式：输出从 n 计算到 1 需要的步数。
输入样例：3
输出样例：5
```

- 解题思路

```
package main

import "fmt"
```

(续下页)

(接上页)

```
func main() {
        var n int
        fmt.Scanf("%d ", &n)
        var res = 0
        for n != 1 {
                if n%2 == 0 {
                        n = n / 2
                        res++
                } else {
                        n = (3*n + 1) / 2
                        res++
                }
        }
        fmt.Println(res)
}
```

83.2 1002. 写出这个数 (1)

- 题目

```
读入一个正整数 n，计算其各位数字之和，用汉语拼音写出和的每一位数字。
输入格式：
每个测试输入包含 1 个测试用例，即给出自然数 n 的值。这里保证 n 小于 10^100。
输出格式：
在一行内输出 n 的各位数字之和的每一位，拼音数字间有 1␣
↪空格，但一行中最后一个拼音数字后没有空格。
输入样例：12345678909876543211234567 89
输出样例：yi san wu
```

- 解题思路

```
package main

import (
        "fmt"
        "strconv"
)

func main() {
        var str string
        m := map[uint8]string{
```

(续下页)

(接上页)

```
                '0': "ling",
                '1': "yi",
                '2': "er",
                '3': "san",
                '4': "si",
                '5': "wu",
                '6': "liu",
                '7': "qi",
                '8': "ba",
                '9': "jiu",
        }
        fmt.Scanf("%s", &str)
        sum := 0
        for k := range str {
                sum = sum + int(str[k]-'0')
        }

        toString := strconv.Itoa(sum)
        for k := range toString {
                if k != 0 {
                        fmt.Print(" ")
                }
                fmt.Print(m[toString[k]])
        }
}
```

83.3 1003. 我要通过!(1)

- 题目

```
“答案正确”是自动判题系统给出的最令人欢喜的回复。
本题属于 PAT 的“答案正确”大派送 —— 只要读入的字符串满足下列条件，
系统就输出“答案正确”，否则输出“答案错误”。
得到“答案正确”的条件是：
    字符串中必须仅有 P、 A、 T这三种字符，不可以包含其它字符；
    任意形如 xPATx 的字符串都可以获得“答案正确”，其中 x␣
↪或者是空字符串，或者是仅由字母 A 组成的字符串；
    如果 aPbTc 是正确的，那么 aPbATca 也是正确的，其中 a、 b、 c 均或者是空字符串，
    或者是仅由字母 A 组成的字符串。
现在就请你为 PAT 写一个自动裁判程序，判定哪些字符串是可以获得“答案正确”的。
输入格式：
每个测试输入包含 1 个测试用例。第 1 行给出一个正整数 n (<10)，是需要检测的字符串个数。
```

(续下页)

(接上页)

```
接下来每个字符串占一行，字符串长度不超过 100，且不包含空格。
输出格式：每个字符串的检测结果占一行，如果该字符串可以获得“答案正确”，则输出␣
↪YES，否则输出 NO。
输入样例：
8
PAT
PAAT
AAPATAA
AAPAATAAAA
xPATx
PT
Whatever
APAAATAA
输出样例：
YES
YES
YES
YES
NO
NO
NO
NO
```

- 解题思路

```
package main

import "fmt"

func main() {
        var n int
        fmt.Scanf("%d", &n)
        for i := 0; i < n; i++ {
                var str string
                fmt.Scanf("%s", &str)
                fmt.Println(judge(str))
        }
}

func judge(str string) string {
        length := len(str)
        left, right := 0, 0
        charMap := make(map[uint8]int)
        for i := 0; i < length; i++ {
```

(续下页)

(接上页)

```
                charMap[str[i]]++
                if str[i] == 'P' {
                        left = i
                }
                if str[i] == 'T' {
                        right = i
                }
        }
        l := left
        m := right - left - 1
        r := length - right - 1
        // 2.任意形如 xPATx 的字符串都可以获得“答案正确”，其中 x␣
↪或者是空字符串，或者是仅由字母 A 组成的字符串；
        // 3.如果 aPbTc 是正确的，那么 aPbATca 也是正确的，其中 a、 b、 c␣
↪均或者是空字符串，或者是仅由字母 A 组成的字符串。
        // P的个数1， T的个数1， A的个数不为0 只有3种数据 左边 * 中间 = 右边
        // =>
↪就是P和T中间每增加一个A，需要将P之前的内容复制到字符串末尾，得到的新字符串就也是正确的。
        if charMap['P'] == 1 && charMap['T'] == 1 && charMap['A'] != 0 &&␣
↪len(charMap) == 3 &&
                right-length != 1 && l*m == r {
                return "YES"
        }
        return "NO"
}
```

83.4 1004. 成绩排名 (1)

- 题目

```
读入 n (>0) 名学生的姓名、学号、成绩，分别输出成绩最高和成绩最低学生的姓名和学号。
输入格式：每个测试输入包含 1 个测试用例，格式为
第 1 行：正整数 n
第 2 行：第 1 个学生的姓名 学号 成绩
第 3 行：第 2 个学生的姓名 学号 成绩
  ... ... ...
第 n+1 行：第 n 个学生的姓名 学号 成绩
其中姓名和学号均为不超过 10 个字符的字符串，成绩为 0 到 100 之间的一个整数，
这里保证在一组测试用例中没有两个学生的成绩是相同的。
输出格式：
对每个测试用例输出 2 行，
第 1 行是成绩最高学生的姓名和学号，第 2 行是成绩最低学生的姓名和学号，字符串间有 1␣
```

(续下页)

(接上页)

```
↪空格。
输入样例：
3
Joe Math990112 89
Mike CS991301 100
Mary EE990830 95

输出样例：
Mike CS991301
Joe Math990112
```

- 解题思路

```
package main

import "fmt"

type Student struct {
        Name  string
        Num   string
        Grade int
}

func main() {
        var n int
        fmt.Scanf("%d", &n)
        m := make(map[string]Student)
        var max, low int
        var maxName, lowName string
        for i := 0; i < n; i++ {
                var s Student
                fmt.Scanf("%s %s %d", &s.Name, &s.Num, &s.Grade)
                m[s.Name] = s
                if i == 0 {
                        max = s.Grade
                        maxName = s.Name
                        low = s.Grade
                        lowName = s.Name
                } else {
                        if s.Grade > max {
                                max = s.Grade
                                maxName = s.Name
                        }
                        if s.Grade < low {
```

(续下页)

(接上页)

```
                                low = s.Grade
                                lowName = s.Name
                        }
                }
        }
        fmt.Println(m[maxName].Name, m[maxName].Num)
        fmt.Println(m[lowName].Name, m[lowName].Num)
}
```

83.5 1005. 继续 (3n+1) 猜想 (1)

- 题目

```
卡拉兹(Callatz)猜想已经在1001中给出了描述。在这个题目里，情况稍微有些复杂。
当我们验证卡拉兹猜想的时候，为了避免重复计算，可以记录下递推过程中遇到的每一个数。
例如对 n=3 进行验证的时候，我们需要计算 3、5、8、4、2、1，则当我们对 n=5、8、4、2␣
↪进行验证的时候，
就可以直接判定卡拉兹猜想的真伪，而不需要重复计算，因为这 4␣
↪个数已经在验证3的时候遇到过了，
我们称 5、8、4、2 是被 3“覆盖”的数。我们称一个数列中的某个数 n 为“关键数”，
如果 n 不能被数列中的其他数字所覆盖。
现在给定一系列待验证的数字，我们只需要验证其中的几个关键数，就可以不必再重复验证余下的数字。
你的任务就是找出这些关键数字，并按从大到小的顺序输出它们。
输入格式：
每个测试输入包含 1 个测试用例，第 1 行给出一个正整数 K (<100)，
第 2 行给出 K 个互不相同的待验证的正整数 n (1<n≤100)的值，数字间用空格隔开。
输出格式：
每个测试用例的输出占一行，按从大到小的顺序输出关键数字。数字间用 1 个空格隔开，
但一行中最后一个数字后没有空格。
输入样例：
6
3 5 6 7 8 11
输出样例：
7 6
```

- 解题思路

```
package main

import (
        "fmt"
        "sort"
```

(续下页)

(接上页)

```
)

func main() {
        var n int
        fmt.Scanf("%d", &n)
        arr := make([]int, 10001)
        maps := make(map[int]int, n)
        for i := 0; i < n; i++ {
                var num int
                fmt.Scanf("%d", &num)
                maps[num] = num // map保存输入的值
                if num == 1 {
                        continue
                }
                for {
                        if num%2 != 0 {
                                num = 3*num + 1
                        }
                        num = num / 2
                        if num == 1 {
                                break
                        }
                        //fmt.Println("num: 出现:\t",num,arr[num] == 1)
                        arr[num] = 1
                }
        }
        tempArr := make([]int, 0)
        for k := range maps {
                tempArr = append(tempArr, maps[k])
        }
        sort.Ints(tempArr)
        var flag = false
        for i := len(tempArr) - 1; i >= 0; i-- {
                if arr[tempArr[i]] == 0 {
                        if flag == true {
                                fmt.Print(" ")
                        }
                        fmt.Print(tempArr[i])
                        flag = true
                }
        }
}
```

83.6 1006. 换个格式输出整数 (1)

- 题目

```
让我们用字母 B 来表示“百”、字母 S 表示“十”，用 12...n 来表示不为零的个位数字 n（
↪<10）；
换个格式来输出任一个不超过 3 位的正整数。例如 234 应该被输出为 BBSSS1234，
因为它有 2 个“百”、3 个“十”、以及个位的 4。
输入格式：每个测试输入包含 1 个测试用例，给出正整数 n（<1000）。
输出格式：每个测试用例的输出占一行，用规定的格式输出 n。
输入样例 1：234
输出样例 1：BBSSS1234
输入样例 2：23
输出样例 2：SS123
```

- 解题思路

```
package main

import "fmt"

func main() {
        var n int
        fmt.Scanf("%d", &n)
        arr := make([]int, 3)
        i := 0
        for {
                if n == 0 {
                        break
                }
                arr[i] = n % 10
                n = n / 10
                i++
        }
        for k := 0; k < arr[2]; k++ {
                fmt.Print("B")
        }
        for k := 0; k < arr[1]; k++ {
                fmt.Print("S")
        }
        for k := 0; k < arr[0]; k++ {
                fmt.Print(k + 1)
        }
}
```

83.7 1007. 素数对猜想 (1)

- 题目

```
让我们定义dn为：dn=pn+1-pn，其中pi是第i个素数。
显然有d1=1，且对于n>1有dn是偶数。“素数对猜想”认为“存在无穷多对相邻且差为2的素数”。

现给定任意正整数N(<10^5)，请计算不超过N的满足猜想的素数对的个数。
输入格式：输入在一行给出正整数N。
输出格式:在一行中输出不超过N的满足猜想的素数对的个数。
输入样例:20
输出样例:4
```

- 解题思路

```
package main

import "fmt"

func main() {
        var n int
        fmt.Scanf("%d", &n)
        count := 0
        // 2 3 5 7
        for i := 5; i <= n; i = i + 2 {
                if isPrime(i) && isPrime(i-2) {
                        count++
                }
        }
        fmt.Println(count)
}

func isPrime(n int) bool {
        for i := 2; i*i <= n; i++ {
                if n%i == 0 {
                        return false
                }
        }
        return true
}
```

83.8 1008. 数组元素循环右移问题 (1)

- 题目

```
一个数组A中存有N（>
↪0）个整数，在不允许使用另外数组的前提下，将每个整数循环向右移M（≥0）个位置，
即将A中的数据由（A0A1⋯AN-1）变换为（AN-M⋯AN-1A0A1⋯AN-M-1）
（最后M个数循环移至最前面的M个位置）。
如果需要考虑程序移动数据的次数尽量少，要如何设计移动的方法？
输入格式:
每个输入包含一个测试用例，第1行输入N（1≤N≤100）和M（≥0）；第2行输入N个整数，之间用空格分隔。
输出格式:在一行中输出循环右移M位以后的整数序列，之间用空格分隔，序列结尾不能有多余空格。
输入样例:
6 2
1 2 3 4 5 6
输出样例:5 6 1 2 3 4
```

- 解题思路

```
package main

import (
        "fmt"
)

func main() {
        var N, M int
        fmt.Scanf("%d %d", &N, &M)

        M = M % N
        arr := make([]int, 0)
        for i := 0; i < N; i++ {
                var num int
                fmt.Scanf("%d", &num)
                arr = append(arr, num)
        }
        if M != 0 {
                reverse(arr)
                reverse(arr[:M])
                reverse(arr[M:])
        }
        for i := 0; i < len(arr)-1; i++ {
                fmt.Print(arr[i])
                fmt.Print(" ")
```

(续下页)

(接上页)

```
        }
        fmt.Print(arr[len(arr)-1])
}

func reverse(arr []int) {
        for i := 0; i < len(arr)/2; i++ {
                arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]
        }
}
```

83.9 1009. 说反话 (1)

- 题目

```
给定一句英语，要求你编写程序，将句中所有单词的顺序颠倒输出。
输入格式：
测试输入包含一个测试用例，在一行内给出总长度不超过 80␣
↪的字符串。字符串由若干单词和若干空格组成，
其中单词是由英文字母（大小写有区分）组成的字符串，单词之间用 1␣
↪个空格分开，输入保证句子末尾没有多余的空格。
输出格式：每个测试用例的输出占一行，输出倒序后的句子。
输入样例：Hello World Here I Come
输出样例：Come I Here World Hello
```

- 解题思路

```
package main

import (
        "bufio"
        "fmt"
        "os"
        "strings"
)

func main() {
        var str string
        reader := bufio.NewReader(os.Stdin)
        data, _, _ := reader.ReadLine()
        str = string(data)
        arr := strings.Split(str, " ")
        for k := len(arr) - 1; k >= 0; k-- {
```

(续下页)

(接上页)

```
                fmt.Print(arr[k])
                if k != 0 {
                        fmt.Print(" ")
                }
        }
}
```

83.10 1010. 一元多项式求导 (1)

- 题目

```
设计函数求一元多项式的导数。（注：xn（n为整数）的一阶导数为nxn-1。）
输入格式:
以指数递降方式输入多项式非零项系数和指数（绝对值均为不超过 1000␣
↪的整数）。数字间以空格分隔。
输出格式:
以与输入相同的格式输出导数多项式非零项的系数和指数。数字间以空格分隔，但结尾不能有多余空格。
注意“零多项式”的指数和系数都是 0，但是表示为 0 0。
输入样例:3 4 -5 2 6 1 -2 0
输出样例:12 3 -10 1 6 0
```

- 解题思路

```
package main

import (
        "bufio"
        "fmt"
        "os"
        "strconv"
        "strings"
)

func main() {
        var str string
        reader := bufio.NewReader(os.Stdin)
        data, _, _ := reader.ReadLine()
        str = string(data)
        var flag = false
        arr := strings.Fields(str)
        for k := 0; k < len(arr)-1; k = k + 2 {
                a, _ := strconv.Atoi(arr[k])
```

(续下页)

(接上页)

```
                b, _ := strconv.Atoi(arr[k+1])
                if a == 0 && b == 0 {
                        fmt.Print("0 0")
                        flag = true
                }
                if b != 0 {
                        if flag == true {
                                fmt.Print(" ")
                        }
                        fmt.Print(a*b, " ", b-1)
                        flag = true
                }
        }
        if flag == false {
                fmt.Print("0 0")
        }
}
```

83.11 1011.A+B 和 C(1)

- 题目

```
给定区间 [-2^31,2^31] 内的 3 个整数 A、B 和 C，请判断 A+B 是否大于 C。
输入格式：
输入第 1 行给出正整数 T (≤10)，是测试用例的个数。随后给出 T 组测试用例，每组占一行，
顺序给出 A、B 和 C。整数间以空格分隔。
输出格式：
对每组测试用例，在一行中输出 Case #X: true 如果 A+B>C，
否则输出 Case #X: false，其中 X 是测试用例的编号（从 1 开始）。
输入样例：
4
1 2 3
2 3 4
2147483647 0 2147483646
0 -2147483648 -2147483647
输出样例：
Case #1: false
Case #2: true
Case #3: true
Case #4: false
```

- 解题思路

```
package main

import "fmt"

func main() {
        var n int
        _, _ = fmt.Scanf("%d", &n)
        for i := 0; i < n; i++ {
                var a, b, c int
                _, _ = fmt.Scanf("%d %d %d", &a, &b, &c)
                var result int
                if a > b {
                        result = a - c + b
                } else {
                        result = b - c + a
                }
                if result > 0 {
                        fmt.Print("Case #", i+1, ": ", true, "\n")
                } else {
                        fmt.Print("Case #", i+1, ": ", false, "\n")
                }
        }
}
```

83.12 1012. 数字分类 (1)

- 题目

```
给定一系列正整数，请按要求对数字进行分类，并输出以下 5 个数字：
    A1 = 能被 5 整除的数字中所有偶数的和；
    A2 = 将被 5 除后余 1 的数字按给出顺序进行交错求和，即计算 n1-n2+n3-n4…；
    A3 = 被 5 除后余 2 的数字的个数；
    A4 = 被 5 除后余 3 的数字的平均数，精确到小数点后 1 位；
    A5 = 被 5 除后余 4 的数字中最大数字。
输入格式：
每个输入包含 1 个测试用例。每个测试用例先给出一个不超过 1000 的正整数 N，
随后给出 N 个不超过 1000 的待分类的正整数。数字间以空格分隔。
输出格式：
对给定的 N 个正整数，按题目要求计算 A1~A5 并在一行中顺序输出。
数字间以空格分隔，但行末不得有多余空格。
若其中某一类数字不存在，则在相应位置输出 N。
输入样例 1：13 1 2 3 4 5 6 7 8 9 10 20 16 18
```

(续下页)

(接上页)

```
输出样例 1: 30 11 2 9.7 9
输入样例 2: 8 1 2 4 5 6 7 9 16
输出样例 2: N 11 2 N 9
```

- 解题思路

```
package main

import (
        "fmt"
)

func main() {
        var n int
        fmt.Scanf("%d", &n)

        var A1, A2, A5 = 0, 0, 0
        var A4 = 0
        mapArr := make(map[int][]int)
        for i := 0; i < n; i++ {
                var num int
                fmt.Scanf("%d", &num)
                mapArr[num%5] = append(mapArr[num%5], num)
        }

        for i := 0; i < n; i++ {
                for j := 0; j < len(mapArr[i]); j++ {
                        if i == 0 && mapArr[i][j]%2 == 0 {
                                A1 += mapArr[i][j]
                        }
                        if i == 1 && j%2 == 0 {
                                A2 += mapArr[i][j]
                        }
                        if i == 1 && j%2 == 1 {
                                A2 -= mapArr[i][j]
                        }
                        if i == 3 {
                                A4 += mapArr[i][j]
                        }
                        if i == 4 && mapArr[i][j] > A5 {
                                A5 = mapArr[i][j]
                        }
                }
        }
```

(续下页)

(接上页)

```
        for i := 0; i < 5; i++ {
                if i != 0 {
                        fmt.Print(" ")
                }
                if i == 0 && A1 == 0 || i != 0 && len(mapArr[i]) == 0 {
                        fmt.Print("N")
                        continue
                }
                if i == 0 {
                        fmt.Print(A1)
                } else if i == 1 {
                        fmt.Print(A2)
                } else if i == 2 {
                        fmt.Print(len(mapArr[2]))
                } else if i == 3 {
                        fmt.Print(fmt.Sprintf("%.1f", float64(A4)/
↪float64(len(mapArr[i]))))
                } else if i == 4 {
                        fmt.Print(A5)
                }
        }
}
```

83.13 1013. 数素数 (1)

- 题目

```
令 Pi 表示第 i 个素数。现任给两个正整数 M≤N≤10^4，请输出 PM 到 PN 的所有素数。
输入格式：输入在一行中给出 M 和 N，其间以空格分隔。
输出格式：输出从 PM 到 PN 的所有素数，每 10 个数字占 1␣
↪行，其间以空格分隔，但行末不得有多余空格。
输入样例：5 27
输出样例：
11 13 17 19 23 29 31 37 41 43
47 53 59 61 67 71 73 79 83 89
97 101 103
```

- 解题思路

```
package main

import "fmt"
```

(续下页)

(接上页)

```
func main() {
        var N, M int
        var num = 2
        var count int
        fmt.Scanf("%d %d", &M, &N)
        result := make([]int, 0)
        for {
                if count < N {
                        if isPrime(num) {
                                count++
                                if count >= M {
                                        result = append(result, num)
                                }
                        }
                        num++
                } else {
                        break
                }
        }

        for i := 0; i < len(result); i++ {
                if i%10 != 0 {
                        fmt.Print(" ")
                }
                fmt.Print(result[i])
                if i%10 == 9 {
                        fmt.Println()
                }
        }
}

func isPrime(a int) bool {
        for i := 2; i*i <= a; i++ {
                if a%i == 0 {
                        return false
                }
        }
        return true
}
```

83.14 1014. 福尔摩斯的约会 (1)

- 题目

```
大侦探福尔摩斯接到一张奇怪的字条：我们约会吧！
3485djDkxh4hhGE 2984akDfkkkkggEdsb s&hgsfdk d&Hyscvnm。
大侦探很快就明白了，字条上奇怪的乱码实际上就是约会的时间星期四 14:04，
因为前面两字符串中第 1 对相同的大写英文字母（大小写有区分）是第 4 个字母D，代表星期四；
第 2 对相同的字符是 E ，那是第 5 个英文字母，
代表一天里的第 14 个钟头（于是一天的 0 点到 23 点由数字 0 到 9、以及大写字母 A 到 N表示）；
后面两字符串第 1 对相同的英文字母 s 出现在第 4 个位置（从 0 开始计数）上，代表第 4分钟。
现给定两对字符串，请帮助福尔摩斯解码得到约会的时间。
输入格式：输入在 4 行中分别给出 4 个非空、不包含空格、且长度不超过 60 的字符串。
输出格式：
在一行中输出约会的时间，格式为 DAY HH:MM，其中 DAY 是某星期的 3 字符缩写，即 MON表示星期一，
TUE 表示星期二，WED 表示星期三，THU 表示星期四，FRI 表示星期五，SAT 表示星期六，SUN表示星期日。
题目输入保证每个测试存在唯一解。
输入样例：
3485djDkxh4hhGE
2984akDfkkkkggEdsb
s&hgsfdk
d&Hyscvnm
输出样例：THU 14:04
```

- 解题思路

```
package main

import (
        "bufio"
        "fmt"
        "os"
)

var week = []string{
        "MON",
        "TUE",
        "WED",
        "THU",
```

(续下页)

(接上页)

```
        "FRI",
        "SAT",
        "SUN",
}

func main() {
        var a, b, c, d string
        reader := bufio.NewReader(os.Stdin)
        data, _, _ := reader.ReadLine()
        a = string(data)
        data, _, _ = reader.ReadLine()
        b = string(data)
        data, _, _ = reader.ReadLine()
        c = string(data)
        data, _, _ = reader.ReadLine()
        d = string(data)
        len0 := len(a)
        len2 := len(c)
        if len(b) > len0 {
                len0 = len(b)
        }
        if len(d) > len2 {
                len2 = len(d)
        }
        flag := true
        for i := 0; i < len0; i++ {
                if a[i] == b[i] {
                        if flag == true {
                                if a[i] >= 'A' && a[i] <= 'G' {
                                        fmt.Printf("%s ", week[a[i]-'A'])
                                        flag = false
                                }
                        } else {
                                if a[i] >= 'A' && a[i] <= 'N' {
                                        fmt.Printf("%02d:", int(a[i]-'A')+10)
                                        break
                                } else if a[i] >= '0' && a[i] <= '9' {
                                        fmt.Printf("%02d:", int(a[i]-'0'))
                                        break
                                }
                        }
                }
        }
```

(续下页)

(接上页)

```
        for i := 0; i < len2; i++ {
                if c[i] == d[i] && ((c[i] >= 'A' && c[i] <= 'Z') || (c[i] >= 'a' &&␣
↪c[i] <= 'z')) {
                        fmt.Printf("%02d", i)
                        break
                }
        }
}
```

83.15 1015. 德才论 (1)

- 题目

```
宋代史学家司马光在《资治通鉴》中有一段著名的“德才论”：
“是故才德全尽谓之圣人，才德兼亡谓之愚人，德胜才谓之君子，才胜德谓之小人。凡取人之术，苟不得圣人，
君子而与之，与其得小人，不若得愚人。”
现给出一批考生的德才分数，请根据司马光的理论给出录取排名。
输入格式：
输入第一行给出 3 个正整数，分别为：
N（≤10^5），即考生总数；L（≥60），为录取最低分数线，即德分和才分均不低于 L␣
↪的考生才有资格被考虑录取；
H（
↪<100），为优先录取线——德分和才分均不低于此线的被定义为“才德全尽”，此类考生按德才总分从高到低排
才分不到但德分到线的一类考生属于“德胜才”，也按总分排序，但排在第一类考生之后；
德才分均低于␣
↪H，但是德分不低于才分的考生属于“才德兼亡”但尚有“德胜才”者，按总分排序，但排在第二类考生之后；
其他达到最低线 L 的考生也按总分排序，但排在第三类考生之后。

随后 N 行，每行给出一位考生的信息，
包括：准考证号 德分 才分，其中准考证号为 8 位整数，德才分为区间 [0, 100]␣
↪内的整数。数字间以空格分隔。
输出格式：
输出第一行首先给出达到最低分数线的考生人数 M，随后 M␣
↪行，每行按照输入格式输出一位考生的信息，
考生按输入中说明的规则从高到低排序。
当某类考生中有多人总分相同时，按其德分降序排列；若德分也并列，则按准考证号的升序输出。
输入样例：
14 60 80
10000001 64 90
10000002 90 60
10000011 85 80
10000003 85 80
```

(续下页)

(接上页)

```
10000004 80 85
10000005 82 77
10000006 83 76
10000007 90 78
10000008 75 79
10000009 59 90
10000010 88 45
10000012 80 100
10000013 90 99
10000014 66 60
输出样例：
12
10000013 90 99
10000012 80 100
10000003 85 80
10000011 85 80
10000004 80 85
10000007 90 78
10000006 83 76
10000005 82 77
10000002 90 60
10000014 66 60
10000008 75 79
10000001 64 90
```

- 解题思路

```
package main

import (
        "bufio"
        "fmt"
        "os"
        "sort"
        "strconv"
        "strings"
)

type Student struct {
        id      int
        d, c, t int
        sortNum int
}
```

(续下页)

(接上页)

```
type Students []Student

func (s Students) Len() int      { return len(s) }
func (s Students) Swap(i, j int) { s[i], s[j] = s[j], s[i] }
func (s Students) Less(i, j int) bool {
        if s[i].sortNum == s[j].sortNum {
                if s[i].t == s[j].t {
                        if s[i].d == s[j].d {
                                return s[i].id < s[j].id
                        }
                        return s[i].d > s[j].d
                }
                return s[i].t > s[j].t
        }
        return s[i].sortNum > s[j].sortNum
}

func main() {
        reader := bufio.NewReader(os.Stdin)
        var N, L, H int
        _, _ = fmt.Scanf("%d %d %d", &N, &L, &H)
        arr := make(Students, N)
        count := 0
        for i := 0; i < N; i++ {
                var id int
                var d, c int
                str, _ := reader.ReadString('\n')
                strArray := strings.Fields(str)
                id, _ = strconv.Atoi(strArray[0])
                d, _ = strconv.Atoi(strArray[1])
                c, _ = strconv.Atoi(strArray[2])
                if c < L || d < L {
                        continue
                }
                total := d + c
                arr[i].t = total
                arr[i].id = id
                arr[i].d = d
                arr[i].c = c
                sortNum := 0
                if c >= H && d >= H {
                        sortNum = 4
                } else if d >= H && c < H {
```

(续下页)

(接上页)

```
                        sortNum = 3
                } else if d < H && c < H && d >= c {
                        sortNum = 2
                } else {
                        sortNum = 1
                }
                arr[i].sortNum = sortNum
                count++
        }
        fmt.Println(count)
        sort.Sort(arr)
        for i := 0; i < count; i++ {
                fmt.Printf("%d %d %d\n", arr[i].id, arr[i].d, arr[i].c)
        }
}
```

CHAPTER 84

Indices and tables

- genindex
- modindex
- search