

0001-0100

0001-0100

- 1. 两数之和
 - 题目
 - 解答思路
- 7. 整数反转
 - 题目
 - 解答思路
- 9.回文数
 - 题目
 - 解答思路
- 13.罗马数字转整数
 - 题目
 - 解答思路
- 14.最长公共前缀
 - 题目
 - 解答思路
- 20.有效的括号
 - 题目
 - 解题思路
- 21.合并两个有序链表
 - 题目
 - 解题思路
- 26.删除排序数组中的重复项
 - 题目
 - 解题思路
- 27.移除元素
 - 题目
 - 解题思路
- 28.实现strStr()
 - 题目
 - 解题思路
- 35.搜索插入位置
 - 题目
 - 解题思路
- 38.报数
 - 题目
 - 解题思路

1. 两数之和

题目

```
1 给定一个整数数组 nums 和一个目标值 target，
2 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。
3 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。
4
5 示例：
6 给定 nums = [2, 7, 11, 15], target = 9
7
8 因为 nums[0] + nums[1] = 2 + 7 = 9
9 所以返回 [0, 1]
```

解答思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|-------------|----------|--------|
| 01 | 暴力法: 2层循环遍历 | $O(n^2)$ | $O(1)$ |
| 02 | 两遍哈希遍历 | $O(n)$ | $O(n)$ |
| 03(最优) | 一遍哈希遍历 | $O(n)$ | $O(n)$ |

```
1  # 暴力法: 2层循环遍历
2  func twoSum(nums []int, target int) []int {
3      for i := 0; i < len(nums); i++ {
4          for j := i + 1; j < len(nums); j++ {
5              if nums[i]+nums[j] == target {
6                  return []int{i, j}
7              }
8          }
9      }
10     return []int{}
11 }
12
13 # 两遍哈希遍历
14 func twoSum(nums []int, target int) []int {
15     m := make(map[int]int, len(nums))
16     for k, v := range nums {
17         m[v] = k
18     }
19
20     for i := 0; i < len(nums); i++ {
21         b := target - nums[i]
22         if num, ok := m[b]; ok && num != i {
23             return []int{i, m[b]}
24         }
25     }
26     return []int{}
```

```

27 }
28
29 # 一遍哈希遍历
30 func twoSum(nums []int, target int) []int {
31     m := make(map[int]int, len(nums))
32     for i, b := range nums {
33         if j, ok := m[target-b]; ok {
34             return []int{j, i}
35         }
36         m[b] = i
37     }
38     return nil
39 }

```

7. 整数反转

题目

```

1  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。
2
3  示例 1：
4  输入：123
5  输出：321
6
7  示例 2：
8  输入：-123
9  输出：-321
10
11 示例 3：
12 输入：120
13 输出：21
14
15 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  $[-2^{31}, 2^{31} - 1]$ 。
16 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

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|------------------------------|--------------|--------|
| 01 | 使用符号标记，转成正数，循环得到%10的余数，再加上符号 | $O(\log(x))$ | $O(1)$ |
| 02(最优) | 对x进行逐个%10取个位，一旦溢出，直接跳出循环 | $O(\log(x))$ | $O(1)$ |

```

1 // 使用符号标记，转成正数，循环得到%10的余数，再加上符号
2 func reverse(x int) int {
3     flag := 1
4     if x < 0 {
5         flag = -1
6         x = -1 * x
7     }
8
9     result := 0
10    for x > 0 {
11        temp := x % 10
12        x = x / 10
13
14        result = result*10 + temp
15    }
16
17    result = flag * result
18    if result > math.MaxInt32 || result < math.MinInt32 {
19        result = 0
20    }
21    return result
22 }
23
24 // 对x进行逐个%10取个位，一旦溢出，直接跳出循环
25 func reverse(x int) int {
26     result := 0
27     for x != 0 {
28         temp := x % 10
29         result = result*10 + temp
30         if result > math.MaxInt32 || result < math.MinInt32 {
31             return 0
32         }
33         x = x / 10
34     }
35     return result
36 }

```

9.回文数

题目

1 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。

2

3 示例 1：输入：121 输出：true

4

5 示例 2：输入：-121 输出：false

6 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。

7

8 示例 3：输入：10 输出：false

9 解释：从右向左读，为 01 。因此它不是一个回文数。

10

11 进阶：

12 你能不将整数转为字符串来解决这个问题吗？

解答思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|---------------------------|--------------|--------------|
| 01(最优) | 数学解法，取出后半段数字进行翻转，然后判断是否相等 | $O(\log(x))$ | $O(1)$ |
| 02 | 转成字符串，依次判断 | $O(\log(x))$ | $O(\log(x))$ |
| 03 | 转成byte数组，依次判断，同2 | $O(\log(x))$ | $O(\log(x))$ |

```
1 // 数学解法，取出后半段数字进行翻转，然后判断是否相等
2 func isPalindrome(x int) bool {
3     if x < 0 || (x%10 == 0 && x != 0) {
4         return false
5     }
6
7     revertedNumber := 0
8     for x > revertedNumber {
9         temp := x % 10
10        revertedNumber = revertedNumber*10 + temp
11        x = x / 10
12    }
13    // for example:
14    // x = 1221 => x = 12 revertedNumber = 12
15    // x = 12321 => x = 12 revertedNumber = 123
16    return x == revertedNumber || x == revertedNumber/10
17 }
18
19 // 转成字符串，依次判断
20 func isPalindrome(x int) bool {
21     if x < 0 {
22         return false
23     }
```

```

24
25     s := strconv.Itoa(x)
26     for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
27         if s[i] != s[j] {
28             return false
29         }
30     }
31     return true
32 }
33
34 // 转成byte数组, 依次判断, 同2
35 func isPalindrome(x int) bool {
36     if x < 0 {
37         return false
38     }
39     arrs := []byte(strconv.Itoa(x))
40     Len := len(arrs)
41     for i := 0; i < Len/2; i++ {
42         if arrs[i] != arrs[Len-i-1] {
43             return false
44         }
45     }
46     return true
47 }

```

13.罗马数字转整数

题目

```

1  罗马数字包含以下七种字符：I， V， X， L， C， D 和 M。
2
3  字符          数值
4  I             1
5  V             5
6  X             10
7  L             50
8  C             100
9  D             500
10 M             1000
11 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 。
12 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
13 9 表示为 IX。这个特殊的规则只适用于以下六种情况：
14     I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
15     X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
16     C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

```

```
17 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。
18
19 示例 1:输入: "III" 输出: 3
20
21 示例 2: 输入: "IV" 输出: 4
22
23 示例 3: 输入: "IX" 输出: 9
24
25 示例 4: 输入: "LVIII" 输出: 58
26 解释: L = 50, V= 5, III = 3.
27
28 示例 5:
29 输入: "MCMXCIV" 输出: 1994
30 解释: M = 1000, CM = 900, XC = 90, IV = 4.
```

解答思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|---|-------|-------|
| 01 | 本质上其实就是全部累加，然后遇到特殊的就做判断。使用一个字段记录递增 | O(n) | O(1) |
| 02(最优) | 从右到左遍历字符串，如果当前字符代表的值不小于其右边，就加上该值；否则就减去该值。 | O(n) | O(1) |

```
1 // 带标记位
2 func romanToInt(s string) int {
3     m := map[byte]int{
4         'I': 1,
5         'V': 5,
6         'X': 10,
7         'L': 50,
8         'C': 100,
9         'D': 500,
10        'M': 1000,
11    }
12    result := 0
13    last := 0
14
15    for i := len(s) - 1; i >= 0; i-- {
16        current := m[s[i]]
17        flag := 1
18        if current < last {
19            flag = -1
```

```

20     }
21     result = result + flag*current
22     last = current
23 }
24 return result
25 }
26
27 // 不带标记位, 小于则减去2倍数
28 func romanToInt(s string) int {
29     m := map[byte]int{
30         'I': 1,
31         'V': 5,
32         'X': 10,
33         'L': 50,
34         'C': 100,
35         'D': 500,
36         'M': 1000,
37     }
38     result := 0
39     last := 0
40
41     for i := len(s) - 1; i >= 0; i-- {
42         current := m[s[i]]
43         if current < last {
44             result = result - current
45         }else {
46             result = result + current
47         }
48         last = current
49     }
50     return result
51 }

```

14.最长公共前缀

题目

```

1  编写一个函数来查找字符串数组中的最长公共前缀。
2  如果不存在公共前缀，返回空字符串 ""。
3
4  示例 1:
5  输入: ["flower","flow","flight"]
6  输出: "fl"
7
8  示例 2:
9  输入: ["dog","racecar","car"]
10 输出: ""
11 解释: 输入不存在公共前缀。

```


12
13 说明：
14 所有输入只包含小写字母 a-z 。

解答思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|---|-----------------|----------|
| 01 | 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 | $O(n^2)/O(n*m)$ | $O(1)$ |
| 02 | 纵向扫描(暴力法):直接取第一个字符串作为最长公共前缀，将其每个字符遍历过一次 | $O(n^2)/O(n*m)$ | $O(1)$ |
| 03(最优) | 排序后，然后计算第一个，和最后一个字符串的最长前缀 | $O(n\log(n))$ | $O(1)$ |
| 04 | trie树 | $O(n^2)$ | $O(n^2)$ |
| 05 | 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 | $O(n^2)/O(n*m)$ | $O(1)$ |
| 06 | 分治法 | $O(n^2)$ | $O(1)$ |

```
1 // 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
2 func longestCommonPrefix(strs []string) string {
3     if len(strs) == 0{
4         return ""
5     }
6     if len(strs) == 1{
7         return strs[0]
8     }
9
10    short := strs[0]
11    for _, s := range strs{
12        if len(short) > len(s){
13            short = s
14        }
15    }
16
17    for i := range short{
18        shortest := short[:i+1]
19        for _,str := range strs{
20            if strings.Index(str,shortest) != 0{
21                return short[:i]
22            }
23        }
24    }
25 }
```

```

23     }
24 }
25 return short
26 }
27
28 // 暴力法:直接依次遍历
29 func longestCommonPrefix(strs []string) string {
30     if len(strs) == 0 {
31         return ""
32     }
33     if len(strs) == 1 {
34         return strs[0]
35     }
36
37     length := 0
38
39     for i := 0; i < len(strs[0]); i++ {
40         char := strs[0][i]
41         for j := 1; j < len(strs); j++ {
42             if i >= len(strs[j]) || char != strs[j][i] {
43                 return strs[0][:length]
44             }
45         }
46         length++
47     }
48     return strs[0][:length]
49 }
50
51 // 排序后, 遍历比较第一个, 和最后一个字符串
52 func longestCommonPrefix(strs []string) string {
53     if len(strs) == 0 {
54         return ""
55     }
56     if len(strs) == 1 {
57         return strs[0]
58     }
59
60     sort.Strings(strs)
61     first := strs[0]
62     last := strs[len(strs)-1]
63     i := 0
64     length := len(first)
65     if len(last) < length {
66         length = len(last)
67     }
68     for i < length {
69         if first[i] != last[i] {
70             return first[:i]
71         }

```

```
72     i++
73 }
74
75     return first[:i]
76 }
77
78
79 // trie树
80 var trie [][]int
81 var index int
82
83 func longestCommonPrefix(strs []string) string {
84     if len(strs) == 0 {
85         return ""
86     }
87     if len(strs) == 1 {
88         return strs[0]
89     }
90
91     trie = make([][]int, 2000)
92     for k := range trie {
93         value := make([]int, 26)
94         trie[k] = value
95     }
96     insert(strs[0])
97
98     minValue := math.MaxInt32
99     for i := 1; i < len(strs); i++ {
100         retValue := insert(strs[i])
101         if minValue > retValue {
102             minValue = retValue
103         }
104     }
105     return strs[0][:minValue]
106 }
107
108 func insert(str string) int {
109     p := 0
110     count := 0
111     for i := 0; i < len(str); i++ {
112         ch := str[i] - 'a'
113         // fmt.Println(string(str[i]),p,ch,trie[p][ch])
114         if value := trie[p][ch]; value == 0 {
115             index++
116             trie[p][ch] = index
117         } else {
118             count++
119         }
120         p = trie[p][ch]
```

```

121     }
122     return count
123 }
124
125 // 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
126 func longestCommonPrefix(strs []string) string {
127     if len(strs) == 0 {
128         return ""
129     }
130     if len(strs) == 1 {
131         return strs[0]
132     }
133
134     commonStr := common(strs[0], strs[1])
135     if commonStr == "" {
136         return ""
137     }
138     for i := 2; i < len(strs); i++ {
139         if commonStr == "" {
140             return ""
141         }
142         commonStr = common(commonStr, strs[i])
143     }
144     return commonStr
145 }
146
147 func common(str1, str2 string) string {
148     length := 0
149     for i := 0; i < len(str1); i++ {
150         char := str1[i]
151         if i >= len(str2) || char != str2[i] {
152             return str1[:length]
153         }
154         length++
155     }
156     return str1[:length]
157 }
158
159 // 分治法
160 func longestCommonPrefix(strs []string) string {
161     if len(strs) == 0 {
162         return ""
163     }
164     if len(strs) == 1 {
165         return strs[0]
166     }
167
168     return commonPrefix(strs, 0, len(strs)-1)

```

```

169 }
170
171 func commonPrefix(strs []string, left, right int) string {
172     if left == right {
173         return strs[left]
174     }
175
176     middle := (left + right) / 2
177     leftStr := commonPrefix(strs, left, middle)
178     rightStr := commonPrefix(strs, middle+1, right)
179     return commonPrefixWord(leftStr, rightStr)
180 }
181
182 func commonPrefixWord(leftStr, rightStr string) string {
183     if len(leftStr) > len(rightStr) {
184         leftStr = leftStr[:len(rightStr)]
185     }
186
187     if len(leftStr) < 1 {
188         return leftStr
189     }
190
191     for i := 0; i < len(leftStr); i++ {
192         if leftStr[i] != rightStr[i] {
193             return leftStr[:i]
194         }
195     }
196     return leftStr
197 }

```

20.有效的括号

题目

```

1  给定一个只包括 '(' , ')' , '{' , '}' , '[' , ']' 的字符串，判断字符串是否有效。
2  有效字符串需满足：
3      左括号必须用相同类型的右括号闭合。
4      左括号必须以正确的顺序闭合。
5  注意空字符串可被认为是有效字符串。
6
7  示例 1：
8  输入："()"
9  输出：true
10
11 示例 2：
12 输入："()[{}]"
13 输出：true
14

```

```
15 示例 3:
16 输入: "()"
17 输出: false
18
19 示例 4:
20 输入: "([)]"
21 输出: false
22
23 示例 5:
24 输入: "{[]}"
25 输出: true
```

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|-----|-------------------|-------|-------|
| 01 | 使用栈结构实现栈 | O(n) | O(n) |
| 02 | 借助数组实现栈 | O(n) | O(n) |
| 03 | 借助数组实现栈，使用数字表示来匹配 | O(n) | O(n) |

```
1  // 使用栈结构实现
2  func isValid(s string) bool {
3      st := new(stack)
4      for _, char := range s {
5          switch char {
6              case '(', '[', '{':
7                  st.push(char)
8              case ')', ']', '}':
9                  ret, ok := st.pop()
10                 if !ok || ret != match[char] {
11                     return false
12                 }
13             }
14         }
15
16         if len(*st) > 0 {
17             return false
18         }
19         return true
20     }
21
22     var match = map[rune]rune{
23         ')': '(',
24         ']': '[',
25         '}': '{',
26     }
```

```

27
28 type stack []rune
29
30 func (s *stack) push(b rune) {
31     *s = append(*s, b)
32 }
33 func (s *stack) pop() (rune, bool) {
34     if len(*s) > 0 {
35         res := (*s)[len(*s)-1]
36         *s = (*s)[:len(*s)-1]
37         return res, true
38     }
39     return 0, false
40 }
41
42 // 借助数组实现栈
43 func isValid(s string) bool {
44     if s == "" {
45         return true
46     }
47
48     stack := make([]rune, len(s))
49     length := 0
50     var match = map[rune]rune{
51         ')': '(',
52         ']': '[',
53         '}': '{',
54     }
55
56     for _, char := range s {
57         switch char {
58             case '(', '[', '{':
59                 stack[length] = char
60                 length++
61             case ')', ']', '}':
62                 if length == 0 {
63                     return false
64                 }
65                 if stack[length-1] != match[char]{
66                     return false
67                 } else {
68                     length--
69                 }
70         }
71     }
72     return length == 0
73 }
74
75 // 借助数组实现栈，使用数字表示来匹配

```

```

76 func isValid(s string) bool {
77     if s == "" {
78         return true
79     }
80
81     stack := make([]int, len(s))
82     length := 0
83     var match = map[rune]int{
84         ')': 1,
85         '(': -1,
86         ']': 2,
87         '[': -2,
88         '}': 3,
89         '{': -3,
90     }
91
92     for _, char := range s {
93         switch char {
94             case '(', '[', '{':
95                 stack[length] = match[char]
96                 length++
97             case ')', ']', '}':
98                 if length == 0 {
99                     return false
100                 }
101                 if stack[length-1]+match[char] != 0 {
102                     return false
103                 } else {
104                     length--
105                 }
106             }
107         }
108     return length == 0
109 }

```

21.合并两个有序链表

题目

- 1 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。
- 2
- 3 示例：
- 4 输入：1->2->4, 1->3->4
- 5 输出：1->1->2->3->4->4

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|------|--------|--------|
| 01(最优) | 迭代遍历 | $O(n)$ | $O(1)$ |
| 02 | 递归实现 | $O(n)$ | $O(n)$ |

```
1 // 迭代遍历
2 func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
3     if l1 == nil {
4         return l2
5     }
6     if l2 == nil {
7         return l1
8     }
9
10    var head, node *ListNode
11    if l1.Val < l2.Val {
12        head = l1
13        node = l1
14        l1 = l1.Next
15    } else {
16        head = l2
17        node = l2
18        l2 = l2.Next
19    }
20
21    for l1 != nil && l2 != nil {
22        if l1.Val < l2.Val {
23            node.Next = l1
24            l1 = l1.Next
25        } else {
26            node.Next = l2
27            l2 = l2.Next
28        }
29        node = node.Next
30    }
31    if l1 != nil {
32        node.Next = l1
33    }
34    if l2 != nil {
35        node.Next = l2
36    }
37    return head
38 }
39
40
41 // 递归遍历
42 func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
43     if l1 == nil {
```

```

44     return l2
45 }
46 if l2 == nil {
47     return l1
48 }
49
50 if l1.Val < l2.Val{
51     l1.Next = mergeTwoLists(l1.Next,l2)
52     return l1
53 }else {
54     l2.Next = mergeTwoLists(l1,l2.Next)
55     return l2
56 }
57 }

```

26.删除排序数组中的重复项

题目

```

1  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。
2  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。
3
4  示例 1：
5  给定数组 nums = [1,1,2],
6  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。
7  你不需要考虑数组中超出新长度后面的元素。
8
9  示例 2：
10 给定 nums = [0,0,1,1,1,2,2,3,3,4],
11 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。
12 你不需要考虑数组中超出新长度后面的元素。
13
14 说明：
15 为什么返回数值是整数，但输出的答案是数组呢？
16 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。
17 你可以想象内部操作如下：
18 // nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
19 int len = removeDuplicates(nums);
20
21 // 在函数里修改输入数组对于调用者是可见的。
22 // 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。
23 for (int i = 0; i < len; i++) {
24     print(nums[i]);
25 }

```

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|------|--------|--------|
| 01 | 双指针法 | $O(n)$ | $O(1)$ |
| 02(最优) | 计数法 | $O(n)$ | $O(1)$ |

```

1 // 双指针法
2 func removeDuplicates(nums []int) int {
3     i, j, length := 0, 1, len(nums)
4     for ; j < length; j++{
5         if nums[i] == nums[j]{
6             continue
7         }
8         i++
9         nums[i] = nums[j]
10    }
11    return i+1
12 }
13
14 // 计数法
15 func removeDuplicates(nums []int) int {
16     count := 1
17     for i := 0; i < len(nums)-1; i++ {
18         if nums[i] != nums[i+1] {
19             nums[count] = nums[i+1]
20             count++
21         }
22     }
23     return count
24 }

```

27.移除元素

题目

```

1 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。
2 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。
3 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。
4
5 示例 1：
6 给定 nums = [3,2,2,3], val = 3,
7 函数应该返回新的长度 2，并且 nums 中的前两个元素均为 2。
8 你不需要考虑数组中超出新长度后面的元素。
9
10 示例 2：
11 给定 nums = [0,1,2,2,3,0,4,2], val = 2,

```

```
12 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。
13 注意这五个元素可为任意顺序。
14 你不需要考虑数组中超出新长度后面的元素。
15 说明：
16 为什么返回数值是整数，但输出的答案是数组呢？
17 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。
18 你可以想象内部操作如下：
19 // nums 是以“引用”方式传递的。也就是说，不对实参作任何拷贝
20 int len = removeElement(nums, val);
21 // 在函数里修改输入数组对于调用者是可见的。
22 // 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。
23 for (int i = 0; i < len; i++) {
24     print(nums[i]);
25 }
```

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|----------------|-------|-------|
| 01(最优) | 双指针，数字前移 | O(n) | O(1) |
| 02 | 双指针，出现重复最后数字前移 | O(n) | O(1) |
| 03 | 首位指针法 | O(n) | O(1) |

```
1 // 双指针，数字前移
2 func removeElement(nums []int, val int) int {
3     i := 0
4     for j := 0; j < len(nums); j++){
5         if nums[j] != val{
6             nums[i] = nums[j]
7             i++
8         }
9     }
10    return i
11 }
12
13 // 双指针，出现重复最后数字前移
14 func removeElement(nums []int, val int) int {
15     i := 0
16     n := len(nums)
17     for i < n{
18         if nums[i] == val{
19             nums[i] = nums[n-1]
20             n--
21         }else {
22             i++
23         }
24     }
25 }
```

```

24     }
25     return n
26 }
27
28 // 首位指针法
29 func removeElement(nums []int, val int) int {
30     i, j := 0, len(nums)-1
31     for {
32         // 从左向右找到等于 val 的位置
33         for i < len(nums) && nums[i] != val {
34             i++
35         }
36         // 从右向左找到不等于 val 的位置
37         for j >= 0 && nums[j] == val {
38             j--
39         }
40         if i >= j {
41             break
42         }
43         // fmt.Println(i,j)
44         nums[i], nums[j] = nums[j], nums[i]
45     }
46     return i
47 }

```

28.实现strStr()

题目

```

1  实现 strStr() 函数。
2  给定一个 haystack 字符串和一个 needle 字符串，
3  在 haystack 字符串中找出 needle 字符串出现的第一个位置（从0开始）。
4  如果不存在，则返回-1。
5
6  示例 1：
7  输入：haystack = "hello", needle = "ll"
8  输出：2
9
10 示例 2：
11 输入：haystack = "aaaaa", needle = "bba"
12 输出：-1
13
14 说明：
15 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。
16 对于本题而言，当 needle 是空字符串时我们应当返回 0 。
17 这与C语言的 strstr() 以及 Java的 indexOf() 定义相符。

```

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|-----|----------|-------|-------|
| 01 | Sunday算法 | O(n) | O(1) |
| | | | |

- Sunday算法

1 |

```
1 // Sunday算法
2 func strStr(haystack string, needle string) int {
3     if needle == ""{
4         return 0
5     }
6     if len(needle) > len(haystack){
7         return -1
8     }
9     // 计算模式串needle的偏移量
10    m := make(map[int32]int)
11    for k,v := range needle{
12        m[v] = len(needle)-k
13    }
14
15    index := 0
16    for index+len(needle) <= len(haystack){
17        // 匹配字符串
18        str := haystack[index:index+len(needle)]
19        if str == needle{
20            return index
21        }else {
22            if index + len(needle) >= len(haystack){
23                return -1
24            }
25            // 后一位字符串
26            next := haystack[index+len(needle)]
27            if nextStep,ok := m[int32(next)];ok{
28                index = index+nextStep
29            }else {
30                index = index+len(needle)+1
31            }
32        }
33    }
34    if index + len(needle) >= len(haystack){
35        return -1
36    }else {
```

```
37     return index
38 }
39 }
```

35.搜索插入位置

题目

```
1  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
2  中，返回它将会被按顺序插入的位置。
3  你可以假设数组中无重复元素。
4
5  示例 1：
6  输入：[1,3,5,6], 5
7  输出：2
8
9  示例 2：
10 输入：[1,3,5,6], 2
11 输出：1
12
13 示例 3：
14 输入：[1,3,5,6], 7
15 输出：4
16
17 示例 4：
18 输入：[1,3,5,6], 0
19 输出：0
```

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|--------|------|--------------|--------|
| 01(最优) | 二分查找 | $O(\log(n))$ | $O(1)$ |
| 02 | 顺序查找 | $O(n)$ | $O(1)$ |
| 03 | 顺序查找 | $O(n)$ | $O(1)$ |

```
1  // 二分查找
2  func searchInsert(nums []int, target int) int {
3      low, high := 0, len(nums)-1
4      for low <= high {
5          mid := (low + high) / 2
6          switch {
7              case nums[mid] < target:
8                  low = mid + 1
```

```

9      case nums[mid] > target:
10         high = mid - 1
11     default:
12         return mid
13     }
14 }
15 return low
16 }
17
18 // 顺序查找
19 func searchInsert(nums []int, target int) int {
20     i := 0
21     for i < len(nums) && nums[i] < target {
22         if nums[i] == target {
23             return i
24         }
25         i++
26     }
27     return i
28 }
29
30 // 顺序查找
31 func searchInsert(nums []int, target int) int {
32     for i := 0; i < len(nums); i++ {
33         if nums[i] >= target {
34             return i
35         }
36     }
37     return len(nums)
38 }

```

38.报数

题目

```

1 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：
2 1.      1
3 2.     11
4 3.     21
5 4.    1211
6 5.    111221
7
8 1 被读作 "one 1" ("一个一")，即 11。
9 11 被读作 "two 1s" ("两个一")，即 21。
10 21 被读作 "one 2", "one 1" ("一个二"，"一个一")，即 1211。
11
12 给定一个正整数 n ( $1 \leq n \leq 30$ )，输出报数序列的第 n 项。
13 注意：整数顺序将表示为一个字符串。

```



```
14
15
16
17 示例 1:
18 输入: 1
19 输出: "1"
20
21 示例 2:
22 输入: 4
23 输出: "1211"
```

解题思路

| No. | 思路 | 时间复杂度 | 空间复杂度 |
|---------|----------|----------|--------|
| 01 (最优) | 递推+双指针计数 | $O(n^2)$ | $O(1)$ |
| 02 | 递归+双指针计数 | $O(n^2)$ | $O(n)$ |

```
1 // 递推+双指针计数
2 func countAndSay(n int) string {
3     strs := []byte{'1'}
4     for i := 1; i < n; i++ {
5         strs = say(strs)
6     }
7     return string(strs)
8 }
9
10 func say(strs []byte) []byte {
11     result := make([]byte, 0, len(strs)*2)
12
13     i, j := 0, 1
14     for i < len(strs) {
15         for j < len(strs) && strs[i] == strs[j] {
16             j++
17         }
18         // 几个几
19         result = append(result, byte(j-i+'0'))
20         result = append(result, strs[i])
21         i = j
22     }
23     return result
24 }
25
26 // 递归+双指针计数
27 func countAndSay(n int) string {
28     if n == 1 {
29         return "1"
```

```
30     }
31     strs := countAndSay(n - 1)
32
33     result := make([]byte, 0, len(strs)*2)
34
35     i, j := 0, 1
36     for i < len(strs) {
37         for j < len(strs) && strs[i] == strs[j] {
38             j++
39         }
40         // 几个几
41         result = append(result, byte(j-i+'0'))
42         result = append(result, strs[i])
43         i = j
44     }
45     return string(result)
46 }
```