

# 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

## 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
    9 表示为 IX。这个特殊的规则只适用于以下六种情况：
13      I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
14      X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
15      C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。
16
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(最优)	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	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    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 }
```