0001-0100

1. 两数之和

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

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)

```go
# 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{}
}

# 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
}
```

7. 整数反转

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

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)

```go
// 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
}

```

```go
// 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
}
```

9.回文数

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

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))

```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 {
```

```go
  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
    }
  }
  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
}
```

13.罗马数字转整数

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

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

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)

```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,
  }
  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
}
```

14.最长公共前缀

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

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(nlog(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)

```go
// 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
    }
  }

  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 ""
  }
```

```go
  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]
  }

  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
```

```go

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 {
      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 ""
  }
```

```go
  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])
  }
  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)]
  }

  if len(leftStr) < 1 {
    return leftStr
  }

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