日刷 leetcode-- 简单版（三）

作者：InkDP
原文链接：https://ld246.com/article/1567159181734
来源网站：链滴

返回总目录

日刷leetcode–简单版

---

58. 最后一个单词的长度

题目描述

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

- 定义一个变量统计，从前往后遍历，遇到空格归零就可以了，注意处理最后几个个字符全为空格的况
- 定义一个变量统计，从后往前便利，虽然时间复杂度同为 O(n)，但是第二个明显快很多

示例代码

```
func lengthOfLastWord(s string) int {
    var count int
```

```
        for i:= len(s)-1; i >= 0; i-- {
            if s[i] == 32{
                if count == 0 {
                    continue
                }else{
                    break
                }
            }
            count ++
        }
    return count
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了 100.00%的用户

内存消耗 :2.2 MB, 在所有 Go 提交中击败了 39.13%的用户

66.加一

题目描述

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

- 从后面往前面循环，最后以一位加 1 即可，处理好末尾 9与999

```
func plusOne(digits []int) []int {
    c := 1 // 定义一个变量用来进位，进位归零则程序结束
    for i := len(digits) - 1; i >= 0; i-- {
        digits[i] += c
        c--
        if digits[i] == 10 {
            digits[i] = 0
            c = 1
        }
```

```
            if c == 0 {
                return digits
            }
        }
        if c != 0 { // 循环完后依旧存在进位则表示遇到了999
            digits = append([]int{1}, digits...)
        }
        return digits
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了 100.00%的用户

内存消耗 :2.2 MB, 在所有 Go 提交中击败了 30.57%的用户

67.二进制求和

题目描述

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

- 判断两个字符串的大小，保证 a 为较长的一个
- 从后往前循环相加，先循环较短的字符串，再循环长字符串剩余的，定义一个变量记录是否需要进，两个字符串相加是需要加上进位的值
- 最后判断进位值是否为 0，不为 0 则在相加后字符串最前面一位加 1

示例代码

```
func addBinary(a string, b string) string {
    la, lb := len(a), len(b)
    if la < lb {
        la, lb = lb, la
        a, b = b, a
    }
    var carry, s byte
    str := make([]byte, la+1)
```

```
    for lb > 0 {
        la--
        lb--
        s = byte(a[la]-'0') + byte(b[lb]-'0') + carry
        carry = s / 2
        s = s % 2
        str[la+1] = byte(s + '0')
    }
    for la > 0 {
        la--
        s = byte(a[la]-'0') + carry
        carry = s / 2
        s = s % 2
        str[la+1] = byte(s + '0')
    }
    if carry == 1 {
        str[la] = carry + '0'
    } else {
        str = str[la+1:]
    }
    return string(str[la:])
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了 100.00%的用户

内存消耗 :2.3 MB, 在所有 Go 提交中击败了 68.18%的用户

69. x 的平方根

题目描述

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 1

- 使用官方包 math.Sqrt，然后提取整数部分即可(不提供代码)

解题思路 2

- 使用二分法，判断中位数的积是否大于 x，是则右边向左移，否则左边直接等于中位数
- 注意的是取中位数是要取右中位数，也就是要加 1，不然会死循环

示例代码 2

```
func mySqrt(x int) int {
    l, r := 0, x/2+1
    for l < r {
        mid := (l + r + 1) / 2
        sqrt := mid * mid
        if sqrt > x {
            r = mid - 1
        } else {
            l = mid
        }
        fmt.Println(l, r, mid)
    }

    return l
}
```

运行结果

执行用时 :8 ms, 在所有 Go 提交中击败了 29.75%的用户

内存消耗 :2.8 MB, 在所有 Go 提交中击败了 5.23%的用户

70. 爬楼梯

题目描述

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

- 这是一个标准的斐波拉切数列，所以就不多说

示例代码

```
func climbStairs(n int) int {
    if n < 3 {
        return n
    }

    a, b := 1, 2
    res := 0
    for i := 2; i < n; i++ {
        res = a + b
        a, b = b, res
    }
    return res
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了 100.00%的用户

内存消耗 :2 MB, 在所有 Go 提交中击败了 52.61%的用户

83. 删除排序链表中的重复

题目描述

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

- 因为是排序了的，所以就相对来说比较简单，判断是否与下一个相等，相等即后移即可，讲 Next向Next.Next

示例代码

```
func deleteDuplicates(head *ListNode) *ListNode {
    if head == nil || head.Next == nil {
```

```
        return head
    }

    carry := head

    for carry != nil && carry.Next != nil {
        if carry.Val == carry.Next.Val {
            carry.Next = carry.Next.Next
        }else{
            carry = carry.Next
        }
    }
    return head
}
```

运行结果

执行用时 :4 ms, 在所有 Go 提交中击败了 96.46%的用户

内存消耗 :3.2 MB, 在所有 Go 提交中击败了 48.18%的用户