链滴

日刷 leetcode-- 简单版（二）

作者：InkDP

原文链接：https://ld246.com/article/1564394374741

来源网站：链滴

返回总目录

日刷leetcode–简单版

---

26. 删除排序数组中的重复项

题目描述

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

- 题目中已说明，这是有序数组
- 采用双指针的方式，一个用于循环，一个用于记录不同的数

示例代码

```
func removeDuplicates(nums []int) int {
    i := 0
```

```
    for j :=  1; j < len(nums); j++ {
        if nums[i] != nums[j] {
            i++
            nums[i] = nums[j]
        }
    }
    return i + 1
}
```

运行结果

执行用时 :100 ms, 在所有 Go 提交中击败了 84.75%的用户

内存消耗 :7.9 MB, 在所有 Go 提交中击败了 66.91%的用户

---

27. 移除元素

题目描述

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);
// 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

- 因为　你不需要考虑数组中超出新长度后面的元素，所以遇到相同的移除就完事了

示例代码

```
func removeElement(nums []int, val int) int {
    for i := 0; i < len(nums); i++ {
        if nums[i] == val {
            nums = append(nums[:i], nums[i+1:]...)
            i--
        }
    }
    return len(nums)
}
```

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 :2.4 MB, 在所有 Go 提交中击败了 44.67%的用户

---

28. 实现 strStr()

题目描述

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
```

解题思路

- 定义两个数组指针 i 和 j，分别记录 haystack 和 needle
- i 递增，从左往右依次匹配，如果当前 haystack 和 needle 字符相等，则继续匹配下一位，直到 j 长度大于 needle 或者 haystack 和 needle 字符不相等
- 如果 haystack 和 needle 字符不相等，则 i 回到第一次匹配的位置，j 归 0 等待下一次匹配
- 判断 j 是否等于 needle 的长度，如果是这表示完全匹配

示例代码

```
func removeElement(nums []int, val int) int {
    for i := 0; i < len(nums); i++ {
        if nums[i] == val {
            nums = append(nums[:i], nums[i+1:]...)
            i--
        }
    }
    return len(nums)
}
```

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 53.10%的用户

35. 搜索插入位置

题目描述

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

- 这似乎是经典的二分了吧，所以微分就完事了

示例代码

```
func searchInsert(nums []int, target int) int {
    l, r := 0, len(nums)
    for ; l < r; {
        mid := (l + r) / 2
        if nums[mid] == target {
            return mid
        } else if nums[mid] < target {
            l = mid + 1
        } else {
            r = mid
        }
    }
    return l
}
```

运行结果

执行用时 :4 ms, 在所有 Go 提交中击败了 97.04%的用户

内存消耗 :3.1 MB, 在所有 Go 提交中击败了 50.79%的用户

38. 报数

题目描述

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

- 这道题其实并不难，难得是题目的描述可能让你弄不明白是怎么回事，

- 从第二行开始，后面的每一行都是数前面的 数字有几个连续的，比如第二行数第一行有1个1，所第二行对应11
- 依次类推，第二行有 2个1，所以第三行21
- 4: 1211
- 5: 111221(数连续的，1个1，1个2,2个1)
- ...
- 我们循环判断一个数字字符串，首先定义一个参照标准: this := str[0]，一个计数器count := 1,循和从str[1]开始，判断str[i] == this，为真则count ++ ，否则从新对this与count赋值。这样我们就成了一次报数操作
- 要求求到第 N 位，这个 N 可以循环也可以递归，看个人喜好

示例代码

```
func countStr(n int, str []byte) []byte {
    if n == 1 {
        return str
    }
    s := make([]byte, 0, len(str)*2)
    this := str[0]
    count := 1
    for i := 1; i < len(str); i++ {
        if this == str[i] {
            count++
        } else {
            s = append(s, byte(count+'0'), this) // count为int型，+'0'相当于+48，得到count对应的SCII码值
            this = str[i]
            count = 1
        }
    }
    s = append(s, byte(count+'0'), this)
    return countStr(n-1, s)
}
```

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 50.58%的用户

53. 最大子序列合

题目描述

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。

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 1

- 暴力法，时间复杂度为 O(n^2)使用双循环的形式累加即可，这里不过多阐述

解题思路 2

- 分治法，这道题分治法并不是最简方式，但还是大概讲下思路
- 最大子序和要么在左半边，要么在右半边，要么是穿过中间，对于左右边的序列，情况也是一样，此可以用递归处理。中间部分的则可以直接计算出来，3 个值取最大的即可，时间复杂度是 O(nlogn)

示例代码

```
func maxSubArray(nums []int) int {
    return maxSum(nums, 0, len(nums)-1)
}

func maxSum(nums []int, l, r int) int {
    if l == r {
        return nums[l]
    }
    mid := (l + r) / 2
    maxLeft := maxSum(nums, l, mid)
    maxRight := maxSum(nums, mid+1, r)
    lSum, sum := math.MinInt64, 0
    for i := mid; i >= l; i-- {
        sum += nums[i]
        if sum > lSum {
            lSum = sum
        }
    }
    rSum, sum := math.MinInt64, 0
    for i := mid + 1; i <= r; i++ {
        sum += nums[i]
        if sum > rSum {
            rSum = sum
        }
    }
    return max(max(maxLeft, maxRight), lSum+rSum)
}

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

```
        return b
    }
}
```

运行结果

执行用时 :8 ms, 在所有 Go 提交中击败了 92.80%的用户

内存消耗 :3.3 MB, 在所有 Go 提交中击败了 73.67%的用户

解题思路 2

- 动态规划，有关这个思路我找到了一段通熟易懂的理解方式

假设你是一个选择性遗忘的赌徒，数组表示你这几天来赢钱或者输钱，
你用 sum 来表示这几天来的输赢，
用 ans 来存储你手里赢到的最多的钱，

如果昨天你手上还是输钱（sum < 0），你忘记它，明天继续赌钱；
如果你手上是赢钱(sum > 0), 你记得，你继续赌钱；
你记得你手赢到的最多的钱

示例代码

```
func maxSubArrays(nums []int) int {
    sum, ans := math.MinInt64, math.MinInt64
    for _,v := range nums {
        if sum > 0 {
            sum += v
        }else{
            sum = v
        }
        if ans < sum {
            ans = sum
        }
    }
    return ans
}
```

运行结果

执行用时 :4 ms, 在所有 Go 提交中击败了 99.10%的用户
内存消耗 :3.3 MB, 在所有 Go 提交中击败了 80.78%的用户