**《算法设计与分析》周练记录表-第2周**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **班学号** | **姓名** | **OJ账号** | **练习平台** | **完成题目数量** | **题号** |
| 11117233 | 徐鸿飞 | Xhofe | LeetCode | 8 | 376,392,435,948,910,860,881,135 |
| Xhofe | LintCode |  |  |
|  | 牛客网 |  |  |
|  | CCF-CSP |  |  |

LeetCode：https://leetcode.com/ 或https://leetcode-cn.com/

LintCode：https://www.lintcode.com/

牛客网（剑指Offer）：https://www.nowcoder.com/ta/coding-interviews?page=1

CCF-CSP：http://www.cspro.org/lead/application/ccf/login.jsp

\*注：本课程作业报告每周提交一次，请所有同学每周日晚21:00前交给各班学习委员，由学习委员统计并上传至微云，课堂上将随机抽取同学讲解自己的练习（5分钟）。

## 一、练习题目及代码

#### 【题目1】LeetCode376

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为**摆动序列。**第一个差（如果存在的话）可能是正数或负数。少于两个元素的序列也是摆动序列。

例如， [1,7,4,9,2,5] 是一个摆动序列，因为差值 (6,-3,5,-7,3) 是正负交替出现的。相反, [1,4,7,2,5] 和 [1,7,4,5,5] 不是摆动序列，第一个序列是因为它的前两个差值都是正数，第二个序列是因为它的最后一个差值为零。

给定一个整数序列，返回作为摆动序列的最长子序列的长度。 通过从原始序列中删除一些（也可以不删除）元素来获得子序列，剩下的元素保持其原始顺序。

**示例 1:**

**输入:** [1,7,4,9,2,5]

**输出:** 6

**解释:** 整个序列均为摆动序列。

**示例 2:**

**输入:** [1,17,5,10,13,15,10,5,16,8]

**输出:** 7

**解释:** 这个序列包含几个长度为 7 摆动序列，其中一个可为[1,17,10,13,10,16,8]。

**示例 3:**

**输入:** [1,2,3,4,5,6,7,8,9]

**输出:** 2

**进阶:**

你能否用 O(*n*) 时间复杂度完成此题?

**【代码】**

**package** LeetCode;

**public class** leetcode376 {

**public int** wiggleMaxLength(**int**[] nums) {

**int** len=nums.**length**;

**if** (len<2)**return** len;

**int** res=1;

**int** temp=nums[0];

**int** zf=0;

**for** (**int** i=1;i<len;i++){

**if** (zf==1){

**if** (temp>nums[i]){

res++;

zf=-1;

}

}**else if** (zf==-1){

**if** (temp<nums[i]){

res++;

zf=1;

}

}

**else** {

**if** (nums[i]>temp){res++;zf=1;}

**else if** (nums[i]<temp){res++;zf=-1;}

**else** {zf=0;}

}

temp=nums[i];

}

**return** res;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

用zf标记上一次摆动的方向，若下一次摆动不同则res++；无论是否摆动不同，都更新上一个数的值，第一次未考虑前两个数相同的情形。

**【对比学习心得】**

**public class** Solution {

**public int** wiggleMaxLength(**int**[] nums) {

**if** (nums == **null** || nums.**length** == 0)

**return** 0;

**int** up = 1, down = 1;

**for** (**int** i = 1; i < nums.**length**; i++) {

**if** (nums[i] > nums[i - 1])

up = down + 1;

**else if** (nums[i] < nums[i - 1])

down = up + 1;

}

**return** up > down ? up : down;

}

}

#### 【题目2】LeetCode392

给定字符串 **s** 和 **t** ，判断 **s** 是否为 **t** 的子序列。

你可以认为 **s** 和 **t** 中仅包含英文小写字母。字符串 **t** 可能会很长（长度 ~= 500,000），而 **s** 是个短字符串（长度 <=100）。

字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"ace"是"abcde"的一个子序列，而"aec"不是）。

**示例 1:**

**s** = "abc", **t** = "ahbgdc"

返回 true.

**示例 2:**

**s** = "axc", **t** = "ahbgdc"

返回 false.

**【代码】**

**package** LeetCode;

**public class** leetcode392 {

**public boolean** isSubsequence(String s, String t) {

**int** lens=s.length();

**int** lent=t.length();

**int** i=0,j=0;

**while** (i<lens&&j<lent){

**if** (s.charAt(i)==t.charAt(j)){

++i;++j;

}**else** {

++j;

}

}

**return** i==lens;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

遍历字符串t，依次找出s中的字符，找完跳出，遍历完仍未找到false；没发现和贪心的联系？？

**【对比学习心得】**

#### 【题目3】LeetCode435

给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。

**注意:**

1. 可以认为区间的终点总是大于它的起点。
2. 区间 [1,2] 和 [2,3] 的边界相互“接触”，但没有相互重叠。

**示例 1:**

**输入:** [ [1,2], [2,3], [3,4], [1,3] ]

**输出:** 1

**解释:** 移除 [1,3] 后，剩下的区间没有重叠。

**示例 2:**

**输入:** [ [1,2], [1,2], [1,2] ]

**输出:** 2

**解释:** 你需要移除两个 [1,2] 来使剩下的区间没有重叠。

**示例 3:**

**输入:** [ [1,2], [2,3] ]

**输出:** 0

**解释:** 你不需要移除任何区间，因为它们已经是无重叠的了。

**【代码】**

**class** Solution:

**def** eraseOverlapIntervals(self, intervals: list[Interval]) -> int:

**if** len(intervals)<2:

**return** 0

intervals.sort(key=**lambda** x:x.end,reverse=**False**)

length=len(intervals)-1

temp=intervals[0].end

**for** i **in** range(1,len(intervals)):

**if** intervals[i].start>=temp:

temp=intervals[i].end

length-=1

**return** length

**【提交记录截图】**



**【遇到的问题及解决办法】**

按照end排序，temp纪录第一个end，依次寻找start比上一个end大的项，找到更新temp=这一项的end；刚开始用java写不好排序。。。

**【对比学习心得】**

#### 【题目4】LeetCode948

你的初始能量为 P，初始分数为 0，只有一包令牌。

令牌的值为 token[i]，每个令牌最多只能使用一次，可能的两种使用方法如下：

* 如果你至少有 token[i] 点能量，可以将令牌置为正面朝上，失去 token[i] 点能量，并得到 1 分。
* 如果我们至少有 1 分，可以将令牌置为反面朝上，获得 token[i] 点能量，并失去 1 分。

在使用任意数量的令牌后，返回我们可以得到的最大分数。

**示例 1：**

**输入：**tokens = [100], P = 50

**输出：**0

**示例 2：**

**输入：**tokens = [100,200], P = 150

**输出：**1

**示例 3：**

**输入：**tokens = [100,200,300,400], P = 200

**输出：**2

**提示：**

1. tokens.length <= 1000
2. 0 <= tokens[i] < 10000
3. 0 <= P < 10000

**【代码】**

**package** LeetCode;

**import** java.util.Arrays;

**public class** leetcode948 {

**public int** bagOfTokensScore(**int**[] tokens, **int** P) {

**if** (tokens.**length**==0)**return** 0;

Arrays.*sort*(tokens);

**int** i=0,j=tokens.**length**-1;

**int** res=0;

**while** ((res>0||tokens[i]<=P)&&i<=j){

**if** (tokens[i]<=P){

res++;

P-=tokens[i];

i++;

}**else if** (i<j){

res--;

P+=tokens[j];

j--;

}**else break**;

}

**return** res;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

贪心的思路：先给令牌排序，然后循环的让小的正面朝上，大的反面朝上：当前最小令牌<=当前P，则正面朝上，若当前最小的指针小于当前最大指针且分数>0时，则让当前最大的反面朝上；其他情况则退出循环。

**【对比学习心得】**

#### 【题目5】LeetCode910

给定一个整数数组 A，对于每个整数 A[i]，我们可以选择**x = -K 或是 x = K**，并将 x 加到 A[i] 中。

在此过程之后，我们得到一些数组 B。

返回 B 的最大值和 B 的最小值之间可能存在的最小差值。

**示例 1：**

**输入：**A = [1], K = 0

**输出：**0

**解释：**B = [1]

**示例 2：**

**输入：**A = [0,10], K = 2

**输出：**6

**解释：**B = [2,8]

**示例 3：**

**输入：**A = [1,3,6], K = 3

**输出：**3

**解释：**B = [4,6,3]

**提示：**

1. 1 <= A.length <= 10000
2. 0 <= A[i] <= 10000
3. 0 <= K <= 10000

**【代码】**

**package** LeetCode;

**import** java.util.Arrays;

**public class** leetcode910 {

**public int** smallestRangeII(**int**[] A, **int** K) {

Arrays.*sort*(A);

**int** len = A.**length**;

**int** res = A[len-1] - A[0];

**for** (**int** i=1;i<len;i++) {

**int** min = Math.*min*(A[0]+K,A[i]-K);

**int** max = Math.*max*(A[len-1]-K,A[i-1]+K);

res = Math.*min*(max-min,res);

}

**return** res;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

先给A排序，找到一个分割点，之前的全部加K，后面的全部减K，使得过程之中的最大值和最小值的差值最小。

不能忽略全加和全减的情形（虽然一样）

**【对比学习心得】**

#### 【题目6】LeetCode860

在柠檬水摊上，每一杯柠檬水的售价为 5 美元。

顾客排队购买你的产品，（按账单 bills 支付的顺序）一次购买一杯。

每位顾客只买一杯柠檬水，然后向你付 5 美元、10 美元或 20 美元。你必须给每个顾客正确找零，也就是说净交易是每位顾客向你支付 5 美元。

注意，一开始你手头没有任何零钱。

如果你能给每位顾客正确找零，返回 true ，否则返回 false 。

**示例 1：**

**输入：**[5,5,5,10,20]

**输出：**true

**解释：**

前 3 位顾客那里，我们按顺序收取 3 张 5 美元的钞票。

第 4 位顾客那里，我们收取一张 10 美元的钞票，并返还 5 美元。

第 5 位顾客那里，我们找还一张 10 美元的钞票和一张 5 美元的钞票。

由于所有客户都得到了正确的找零，所以我们输出 true。

**示例 2：**

**输入：**[5,5,10]

**输出：**true

**示例 3：**

**输入：**[10,10]

**输出：**false

**示例 4：**

**输入：**[5,5,10,10,20]

**输出：**false

**解释：**

前 2 位顾客那里，我们按顺序收取 2 张 5 美元的钞票。

对于接下来的 2 位顾客，我们收取一张 10 美元的钞票，然后返还 5 美元。

对于最后一位顾客，我们无法退回 15 美元，因为我们现在只有两张 10 美元的钞票。

由于不是每位顾客都得到了正确的找零，所以答案是 false。

**提示：**

* 0 <= bills.length <= 10000
* bills[i] 不是 5 就是 10 或是 20

**【代码】**

**package** LeetCode;

**import** java.util.Arrays;

**public class** leetcode860 {

**public boolean** lemonadeChange(**int**[] bills) {

**int**[] coins=**new int**[2];

Arrays.*fill*(coins,0);

**for** (**int** i=0;i<bills.**length**;i++){

**if** (bills[i]==5){

coins[0]++;

}

**else if** (bills[i]==10){

**if** (coins[0]==0){

**return false**;

}**else** {

coins[0]--;

coins[1]++;

}

}**else** {

**if** (coins[0]==0){

**return false**;

}**else if** (coins[1]>0){

coins[1]--;

coins[0]--;

}**else if** (coins[0]>=3){

coins[0]-=3;

}**else** {

**return false**;

}

}

}

**return true**;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

贪心策略：优先找面值大的货币。

**【对比学习心得】**

#### 【题目7】LeetCode881

第 i 个人的体重为 people[i]，每艘船可以承载的最大重量为 limit。

每艘船最多可同时载两人，但条件是这些人的重量之和最多为 limit。

返回载到每一个人所需的最小船数。(保证每个人都能被船载)。

**示例 1：**

**输入：**people = [1,2], limit = 3

**输出：**1

**解释：**1 艘船载 (1, 2)

**示例 2：**

**输入：**people = [3,2,2,1], limit = 3

**输出：**3

**解释：**3 艘船分别载 (1, 2), (2) 和 (3)

**示例 3：**

**输入：**people = [3,5,3,4], limit = 5

**输出：**4

**解释：**4 艘船分别载 (3), (3), (4), (5)

**提示：**

* 1 <= people.length <= 50000
* 1 <= people[i] <= limit <= 30000

**【代码】**

**package** LeetCode;  
  
**import** java.util.Arrays;  
  
**public class** leetcode881 {  
 **public int** numRescueBoats(**int**[] people, **int** limit) {  
 **int** i=0,j=people.**length**-1;  
 Arrays.*sort*(people);  
 **int** ship=0;  
 **while** (i<=j){  
 **if** (people[i]+people[j]<=limit){  
 i++;j--;  
 ship++;  
 }**else** {  
 j--;  
 ship++;  
 }  
 }  
 **return** ship;  
 }  
}

**【提交记录截图】**



**【遇到的问题及解决办法】**

贪心策略：可以的话，让当前最重的人和当前最轻的人坐同一条船，否则当前最重的人一个人做一条船。

**【对比学习心得】**

#### 【题目8】LeetCode135

老师想给孩子们分发糖果，有 *N* 个孩子站成了一条直线，老师会根据每个孩子的表现，预先给他们评分。

你需要按照以下要求，帮助老师给这些孩子分发糖果：

* 每个孩子至少分配到 1 个糖果。
* 相邻的孩子中，评分高的孩子必须获得更多的糖果。

那么这样下来，老师至少需要准备多少颗糖果呢？

**示例 1:**

**输入:** [1,0,2]

**输出:** 5

**解释:** 你可以分别给这三个孩子分发 2、1、2 颗糖果。

**示例 2:**

**输入:** [1,2,2]

**输出:** 4

**解释:** 你可以分别给这三个孩子分发 1、2、1 颗糖果。

第三个孩子只得到 1 颗糖果，这已满足上述两个条件。

**【代码】**

**package** LeetCode;  
**public class** leetcode135 {  
 **public int** candy(**int**[] ratings) {  
 **int** count=ratings.**length**;  
 **if** (count<=1){  
 **return** count;  
 }  
 **int**[] candy=**new int**[count];  
 candy[0]=1;  
 **for** (**int** i=1;i<count;i++){  
 **if** (ratings[i]>ratings[i-1]){  
 candy[i]=candy[i-1]+1;  
 }**else** {  
 candy[i]=1;  
 }  
 }  
 **for** (**int** i=count-2;i>=0;i--){  
 **if** (ratings[i]>ratings[i+1]){  
 candy[i]=Math.*max*(candy[i],candy[i+1]+1);  
 }  
 }  
 **int** res=0;  
 **for** (**int** i=0;i<count;i++){  
 res+=candy[i];  
 }  
 **return** res;  
 }  
}

**【提交记录截图】**



**【遇到的问题及解决办法】**

给第一个孩子一个糖果，然后从左向右遍历，若果该孩子比上一个高，则给上一个孩子糖果数+1，否则给1个；这时只保证了从左往右满足，所以需要再从右往左遍历一次，若果发现不满足的则更新当前值。

**【对比学习心得】**

## 二、本周小结

本来是觉得贪心算法应该比动态规划简单一些的，但是在做题的时候就会觉得贪心也不简单，而且贪心很多也是不排序的，需要排序的可能还会简单一些。

感觉就是看到题目很快就有了一个思路，写完才发现这个思路行不通，能过测试但是过不了全部。

-------------------------------

成绩评定：

教师签名：

批改日期：