**《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 | X-cool | LeetCode | 8 | 70，121，5，474，416，53，139，322 |
| 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】LeetCode70

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

**【代码】**

**public class** leetcode70 {

**public int** climbStairs(**int** n) {

**if** (n==1)**return** 1;*//刚开始未加*

**int** []array=**new int**[n];

array[0]=1;

array[1]=2;

**for** (**int** i = 2;i<n;i++){

array[i]=array[i-1]+array[i-2];

}

**return** array[n-1];

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

斐波那契数列，刚开始未加输入为1的情形。

**【对比学习心得】**

#### 【题目2】LeetCode121

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

如果你最多只允许完成一笔交易（即买入和卖出一支股票），设计一个算法来计算你所能获取的最大利润。

注意你不能在买入股票前卖出股票。

**示例 1:**

**输入:** [7,1,5,3,6,4]

**输出:** 5

**解释:** 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。

注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格。

**示例 2:**

**输入:** [7,6,4,3,1]

**输出:** 0

**解释:** 在这种情况下, 没有交易完成, 所以最大利润为 0。

**【代码】**

**import** java.lang.Integer;

**public class** leetcode121 {

**public int** maxProfit(**int**[] prices) {

**int** min=Integer.***MAX\_VALUE***;

**int** max=Integer.***MIN\_VALUE***;

**int** value=0;

**int** temp;

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

temp=min;

min=Integer.*min*(min,prices[i]);

**if** (temp!=min){

max=Integer.***MIN\_VALUE***;

}

max=Integer.*max*(max,prices[i]);

value=Integer.*max*(value,max-min);

}

**return** value;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

遍历数组，分别用min和max记录最小值和最大值，以及value纪录利润并更新，然后就是每次min改变时，max必须改变。

**【对比学习心得】**

#### 【题目3】LeetCode5

给定一个字符串 s，找到 s 中最长的回文子串。你可以假设 s 的最大长度为 1000。

**示例 1：**

**输入:** "babad"

**输出:** "bab"

**注意:** "aba" 也是一个有效答案。

**示例 2：**

**输入:** "cbbd"

**输出:** "bb"

**【代码】**

**class** Solution:

**def** longestPalindrome(self, s):

l = len(s)

**if** l == 0 **or** l == 1:

**return** s

res = s[0]

dp = [([0] \* l) **for** i **in** range(l)]

**for** i **in** range(l - 1):

dp[i][i] = **True**

**if** (s[i] == s[i + 1]):

dp[i][i + 1] = **True**

res2 = s[i:i + 2]

**else**:

dp[i][i + 1] = **False**

dp[l - 1][l - 1] = **True**

step = 3

run = **True**

**while True**:

**if** (step > l):

**break**

**for** i **in** range(l - step+1):

**if** s[i] == s[i + step - 1] **and** dp[i + 1][i + step - 2]:

dp[i][i + step - 1] = **True**

res = s[i:i + step]

**else**:

dp[i][i + step - 1] = **False**

step += 1

**return** res

**【提交记录截图】**



**【遇到的问题及解决办法】**

刚开始理解错了题意，认为子串可以不连续，就用上课所给的求两个字符串公共子串的方法求str与str[::-1]的公共子串（后来发现即使是不连续也是错的），之后的思想是依次求出其中的长度为i的回文串，i依次递增，就有递归方程：,然后就可以根据这个依次求出，但是时间复杂度好像有点高。

**【对比学习心得】**

#### 【题目4】LeetCode474

在计算机界中，我们总是追求用有限的资源获取最大的收益。

现在，假设你分别支配着 **m** 个 0 和 **n** 个 1。另外，还有一个仅包含 0 和 1 字符串的数组。

你的任务是使用给定的 **m** 个 0 和 **n** 个 1 ，找到能拼出存在于数组中的字符串的最大数量。每个 0 和 1 至多被使用**一次**。

**注意:**

1. 给定 0 和 1 的数量都不会超过 100。
2. 给定字符串数组的长度不会超过 600。

**示例 1:**

**输入:** Array = {"10", "0001", "111001", "1", "0"}, m = 5, n = 3

**输出:** 4

**解释:** 总共 4 个字符串可以通过 5 个 0 和 3 个 1 拼出，即 "10","0001","1","0" 。

**示例 2:**

**输入:** Array = {"10", "0", "1"}, m = 1, n = 1

**输出:** 2

**解释:** 你可以拼出 "10"，但之后就没有剩余数字了。更好的选择是拼出 "0" 和 "1" 。

**【代码】**

#include<vector>

#include<string>

#include<algorithm>

using namespace std;

class Solution {

public:

int findMaxForm(vector<string>& strs, int m, int n) {

int len = strs.size();

int \*\*\* arr;

int i, j, k;

arr = new int\*\*[len+1];

for (i = 0; i < len+1; i++) {

arr[i] = new int\*[m+1];

for (j = 0; j < m+1; j++) {

arr[i][j] = new int[n+1];

}

}

int \*count0 = new int[len];

int \*count1 = new int[len];

for (i = 0; i < len; i++) {

int a = 0; int b = 0;

for (j = 0; j < strs[i].length(); j++)

{

if (strs[i][j]=='0')

{

a++;

}

else

{

b++;

}

}

count0[i] = a;

count1[i] = b;

}

for (i = 0; i < len + 1; i++) {

arr[i][0][0] = 0;

}

for (i = 0; i < m + 1; i++) {

for (j = 0; j < n + 1; j++) {

arr[0][i][j] = 0;

}

}

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

for (j = 0; j < m + 1; j++) {

for (k = 0; k < n + 1; k++) {

if (j - count0[i - 1] < 0 || k - count1[i - 1] < 0) {

arr[i][j][k] = arr[i - 1][j][k];

}

else

{

arr[i][j][k] = max(arr[i - 1][j][k], arr[i - 1][j - count0[i-1]][k - count1[i - 1]]+1);

}

}

}

}

int res = arr[len][m][n];

for (i = 0; i < len+1; i++) {

for (j = 0; j < m+1; j++) {

delete[] arr[i][j];

}

delete[] arr[i];

}

delete[] arr;

delete count0, count1;

return res;

}

};

**【提交记录截图】**



**【遇到的问题及解决办法】**

与背包问题类似，每个字符串的价值为1，消耗的0和1即为背包的重量，动态规划求解，arr[i][j][k]纪录前i个字符串，j个0和k个1可以拼凑的最大数量，就有递推公式：

Arr[i][j][k]=

问题：时间复杂度过大，刚开始用python写的没有通过，然后用C++重写了一遍。

**【对比学习心得】**

#### 【题目5】LeetCode416

给定一个**只包含正整数**的**非空**数组。是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

**注意:**

1. 每个数组中的元素不会超过 100
2. 数组的大小不会超过 200

**示例 1:**

输入: [1, 5, 11, 5]

输出: true

解释: 数组可以分割成 [1, 5, 5] 和 [11].

**示例 2:**

输入: [1, 2, 3, 5]

输出: false

解释: 数组不能分割成两个元素和相等的子集.

**【代码】**

**package** LeetCode;

**import** java.lang.Integer;

**public class** leetcode416 {

**public boolean** canPartition(**int**[] nums) {

**int** sum=0;

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

**for** (**int** i=0;i<len;i++){

sum+=nums[i];

}

**if** (sum%2!=0){

**return false**;

}

**int** aims=sum/2;*//背包的重量*

**int**[][] dp=**new int**[len][aims+1];

dp[0][0]=0;

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

dp[0][i]=Integer.***MIN\_VALUE***;

}

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

**for**(**int** j=nums[i];j<=aims;j++){

dp[i][j]=Integer.*max*(dp[i-1][j],dp[i-1][j-nums[i]]+nums[i]);

}

}

**return** dp[len-1][aims]==aims;

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

0-1背包问题，在此基础上增加一个新的条件：将背包装满，在原有算法上改进初始化即可，将dp[0][1->W]改为-∞；因为初始化的dp数组事实上就是在没有任何物品可以放入背包时的合法状态.如果要求背包恰好装满，那么此时只有容量为0的背包可以在什么也不装且价值为0的情况下被“恰好装满”，其它容量的背包均没有合法的解，属于未定义的状态，应该被赋值为-∞了。

**【对比学习心得】**

#### 【题目6】LeetCode53

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），返回其最大和。

**【代码】**

#pragma once

#include<vector>

using namespace std;

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int len = nums.size();

if (len == 1) {

return nums[0];

}

else

{

return maxSubArray(nums, 0, len - 1);

}

}

int maxSubArray(vector<int>&nums, int left, int right) {

if (left == right) {

return nums[left];

}

int mid = (right - left) / 2 + left;

int sumL = maxSubArray(nums, left, mid);

int sumR = maxSubArray(nums, mid + 1, right);

int maxL = nums[mid];

int temp = maxL;

for (int i = 0; i < mid; i++)

{

temp += nums[mid - i - 1];

if (temp>maxL)

{

maxL = temp;

}

}

int maxR = nums[mid + 1];

temp = maxR;

for (int i = mid + 2; i < right + 1; i++) {

temp += nums[i];

if (temp > maxR) {

maxR = temp;

}

}

int sum = maxL + maxR;

if (sumL > sum) {

sum = sumL;

}

if (sumR > sum) {

sum = sumR;

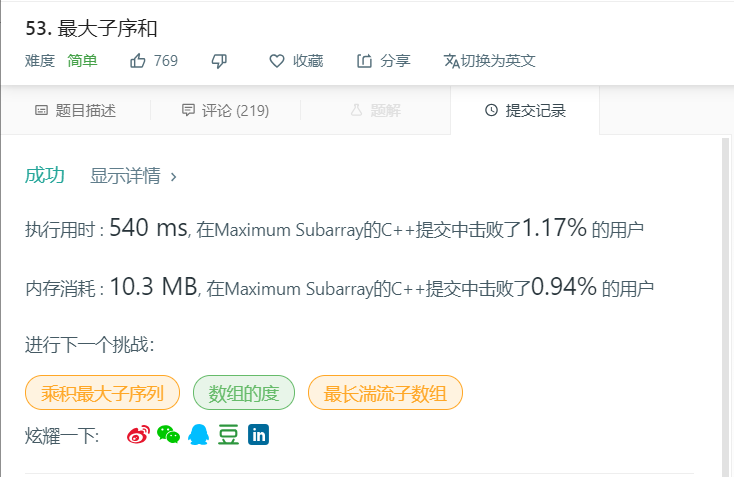
}

return sum;

}

};

**【提交记录截图】**



**【遇到的问题及解决办法】**

上次已经写过分治了，但是python没有通过，改用c++后就可以了。

**【对比学习心得】**

#### 【题目7】LeetCode139

给定一个**非空**字符串 *s* 和一个包含**非空**单词列表的字典 *wordDict*，判定 *s* 是否可以被空格拆分为一个或多个在字典中出现的单词。

**说明：**

* 拆分时可以重复使用字典中的单词。
* 你可以假设字典中没有重复的单词。

**示例 1：**

**输入:** s = "leetcode", wordDict = ["leet", "code"]

**输出:** true

**解释:** 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。

**示例 2：**

**输入:** s = "applepenapple", wordDict = ["apple", "pen"]

**输出:** true

**解释:** 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。

  注意你可以重复使用字典中的单词。

**示例 3：**

**输入:** s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]

**输出:** false

**【代码】**

**package** LeetCode;

**import** java.util.List;

**public class** leetcode139 {

**public boolean** wordBreak(String s, List<String> wordDict) {

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

**boolean**[] dp=**new boolean**[lens+1];

dp[0]=**true**;

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

**for** (**int** j=0;j<i;j++){

**if**(dp[j]&&wordDict.contains(s.substring(j,i))){

dp[i]=**true**;

**break**;

}**else** {

dp[i]=**false**;

}

}

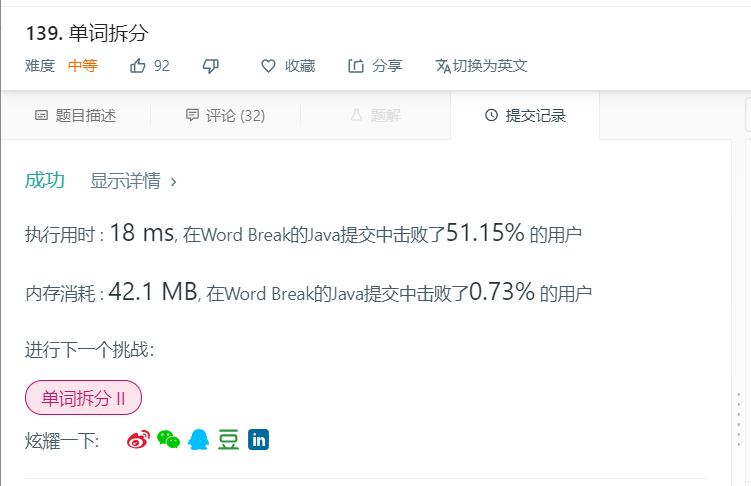
}

**return** dp[lens];

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

遍历字符串，并分割，dp[i]纪录前i个字符是否包含在字典中，则有转移方程：

最后的dp[lens]即为结果。

**【对比学习心得】**

#### 【题目8】LeetCode322

给定不同面额的硬币 coins 和一个总金额 amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 -1。

**示例 1:**

**输入:** coins = [1, 2, 5], amount = 11

**输出:** 3

**解释:** 11 = 5 + 5 + 1

**示例 2:**

**输入:** coins = [2], amount = 3

**输出:** -1

**说明**:

你可以认为每种硬币的数量是无限的。

**【代码】**

**public class** leetcode322v {

**public int** coinChange(**int**[] coins, **int** amount) {

**int** len=coins.**length**;

**int**[] dp=**new int**[amount+1];

dp[0]=0;

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

dp[i]=Integer.***MAX\_VALUE***/2;

}

**for** (**int** i=0;i<len;i++){

**for** (**int** j=coins[i];j<amount+1;j++){

dp[j]=Integer.*min*(dp[j],dp[j-coins[i]]+1);

}

}

**if** (dp[amount]>=amount+1){

**return** -1;

}

**return** dp[amount];

}

}

**【提交记录截图】**



**【遇到的问题及解决办法】**

完全背包问题。

Dp[j]表示拼凑j钱时最小零钱的数量，则有转移方程：dp[j]=min(dp[j],dp[j-coins[i]]+1);中间算出每种钱的最小零钱数量。

出现的问题：用Integer.***MAX\_VALUE***表示最大数，得出的结果是负的，发现是在加一后溢出了，变成了负数，改为Integer.***MAX\_VALUE/2***后正常。

**【对比学习心得】**

## 二、本周小结

动态规划真的很难，想不出来递推方程无从下手。有的题目实在想不出来参考了大佬的想法。有的题虽然过了，但是时间复杂度贼高，空间复杂度也有待优化。要多学习大佬的写法。

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

成绩评定：

教师签名：

批改日期：