算法设计与分析课程作业作业 5

162140224 刘昊文

2023 年 5 月 2 日

1 零钱兑换

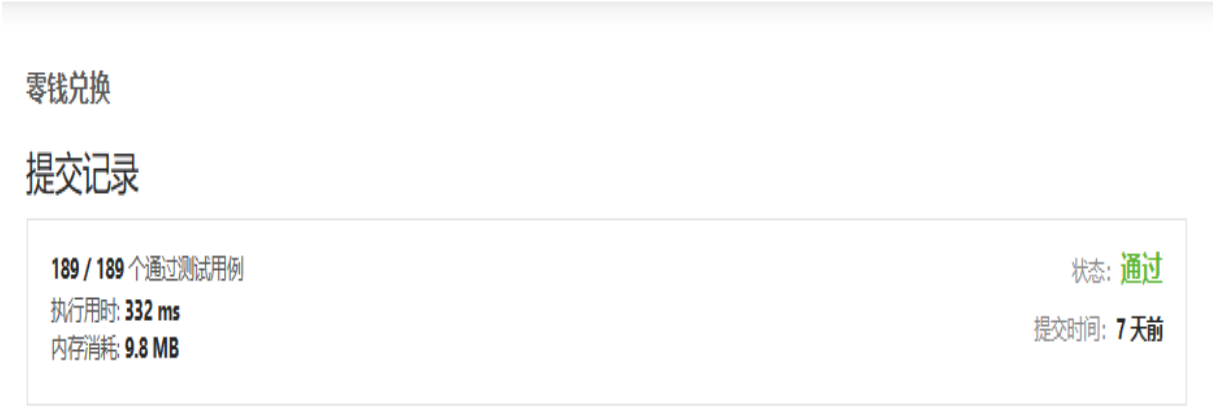
这道题可以看作一个完全背包问题，状态 dp[i][j] 表示前 i 个物品金额为 j 时需要的最少的硬币个数, 而这个状态只能由 dp[i-1][j-coins[i]]+1 或 dp[i-1][j] 状态转移得来所以易得状态转移方程

$$dp[i][j] = max(dp[i-1][j-k\times coins[i]] + k \times coins[i], dp[i-1][j]) \qquad k \le \frac{j}{coins[i]}$$

这道题可以减去物品的纬度，压缩一维但需要注意的是在枚举内层当前状态金额时, 需要从 coins[i] 枚举到总金额，进行正向状态转移, 使得当前状态可以利用当前这一层的状态, 这代表硬币能重复拿取，符合题意

一维状态转移方程如下

$$dp[j] = min(dp[j-coins[i]] + 1, dp[j])$$



2 分割等和子集

这道题需要将一个数组分为两个和相等的子数组，假设这两个子数组为 num1 和 num2，数组总和为 tot

两个子数组和需要相等，易得这两个子数组和都要为 tot/2。

易得当 tot 为奇数时，不可能成立当 tot 为偶数时，可转换为 01 背包问题，判断能否凑成 tot/2，原问题得解

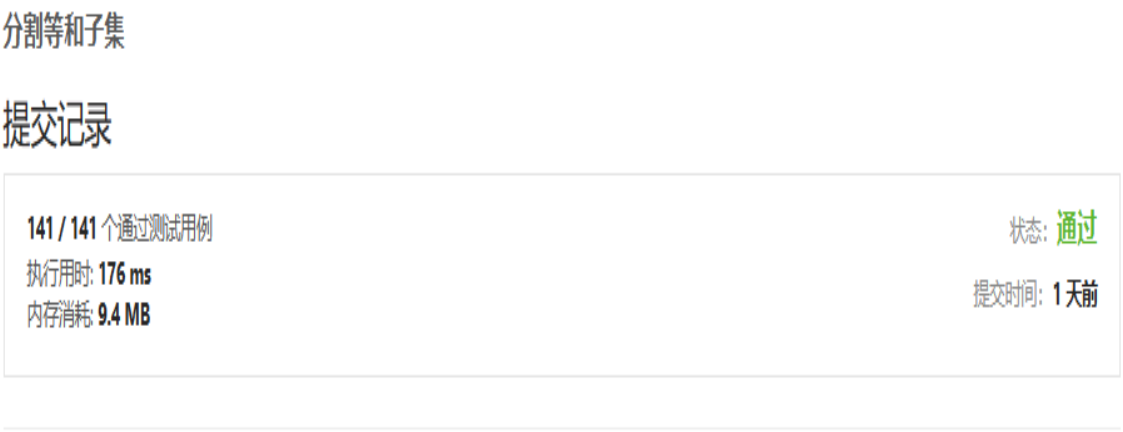
在 01 背包问题中，递归方程为

$$dp[i][j] = max(dp[i-1][j-nums[i]] + nums[i], dp[i-1][j])$$

事实上，我们可以减去物品的纬度，将其压缩至一维递归方程如下

$$dp[j] = max(dp[j], dp[j-nums[i]] + nums[i])$$

需要注意的是，此时外层 i 循环不同物品时，内层 j 需要倒序循环，改变状态更新次序，使当前状态不可以利用当前这一层的状态，防止一个数字多次被拿，与第一题正好相反



3 最长回文子序列

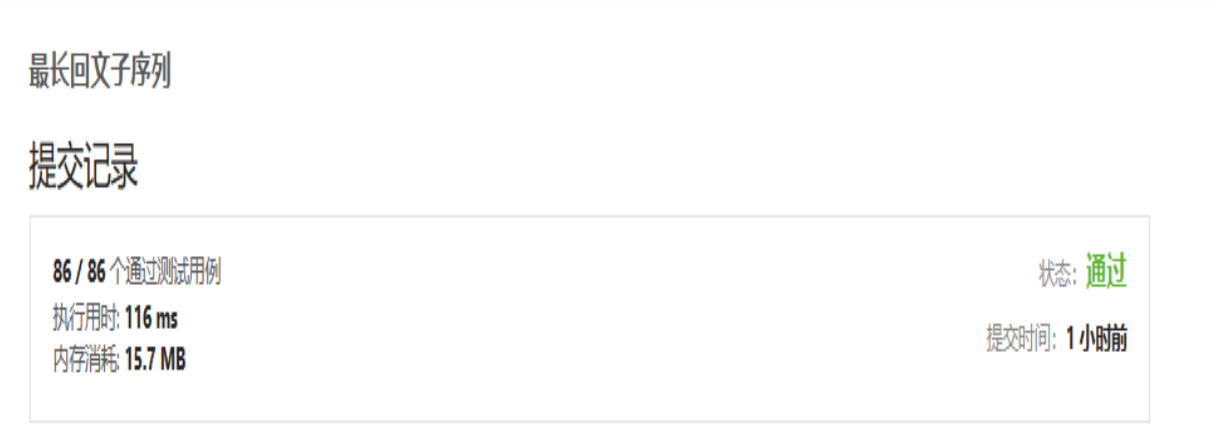
这道题需要求出序列中的最长回文子序列，通过观察可得，对于一个序列，最长回文子序列等于它与它的逆序列的最长公共子序列

因此这题可以转换成一个求最长公共子序列的问题

对于求最长公共子序列，状态 dp[i][j] 表示当前第一个序列从 1 到 i，第二个序列从 1 到 j，最长回文子序列的长度所以容易得到状态转移方程为

$$dp[i][j] = \begin{cases} dp[i-1][j-1]+1 & \text{if } s1[i] = s2[j] \\ max(dp[i-1][j], dp[i][j-1]) & \text{if } s1[i] \neq s2[j], \end{cases}$$

原问题得解



4 打家劫舍

在这道题中, 对于一个房屋, 只有它旁边的房子未被闯入时, 才能闯入该房间

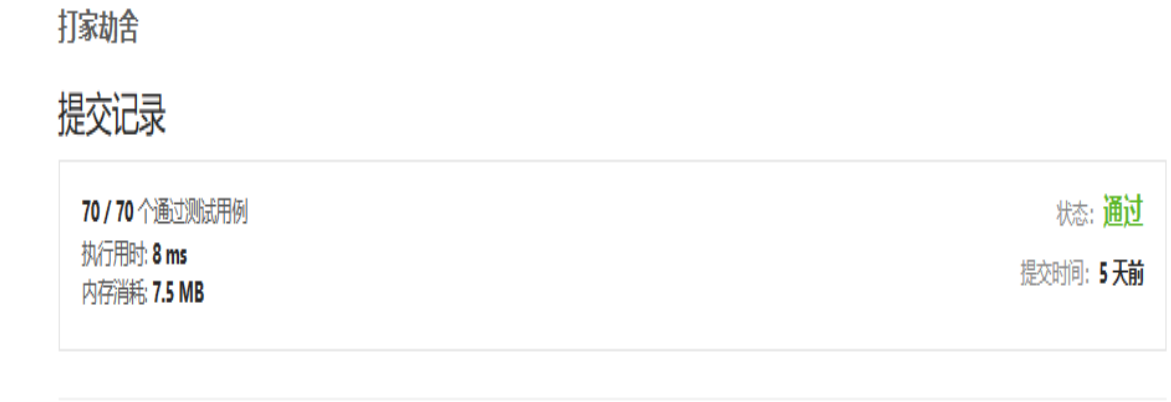
对于状态 $dp[i][0]$ 第一维 i 表示前 i 个房间，第二维 0 表示第 i 个房间未被闯入，1 表示被闯入

当第 i 个房间未被闯入时，第 i-1 个房间是否闯入都符合题意，所以选择其中较大值即 $max(dp[i-1][1], dp[i-1][0])$

当第 i 个房间被闯入时，第 i-1 个房间只能是未被闯入，所以取前 i-1 个房间未被闯入的值加上第 i 个房间的金额即 $dp[i-1][0] + nums[i]$

所以易得状态转移方程

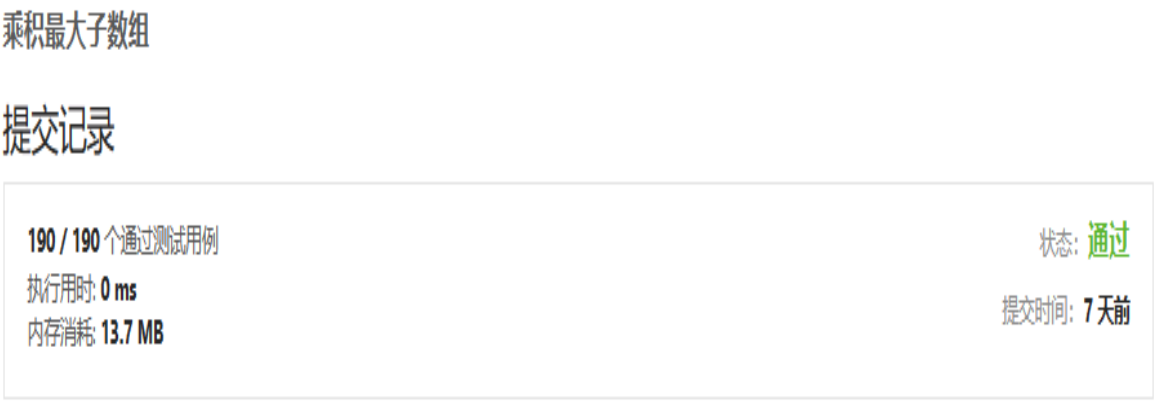
$$\begin{cases} dp[i][1] = dp[i-1][0] + nums[i] \\ dp[i][0] = max(dp[i-1][1], dp[i-1][0]) \end{cases}$$



5 乘积最大子数组

这道题需要找到乘积最大的子数组，数组中即存在正数，也存在负数。所以存在乘积为负数的时候，此时可能会负负得正得出更大值得情况所以需要维护出子数组的最大值与最小值，设 $dp[i][0]$ 表示第 i 个元素结尾的乘积最大子数组 $dp[i][1]$ 表示第 i 个元素结尾的乘积最小子数组 dp[i][1] 和 dp[i][0] 可由 dp[i-1][0] 和 dp[i-1][1] 得到，除此之外，还可以不选择前 i-1 个元素，只选择 num[i]，所以可得状态转移方程如下:

$$\begin{cases} dp[i][1] = max(nums[i], max(dp[i-1][1] * nums[i], dp[i-1][0] * nums[i])) \\ dp[i][0] = min(nums[i], min(dp[i-1][1] * nums[i], dp[i-1][0] * nums[i])) \end{cases}$$



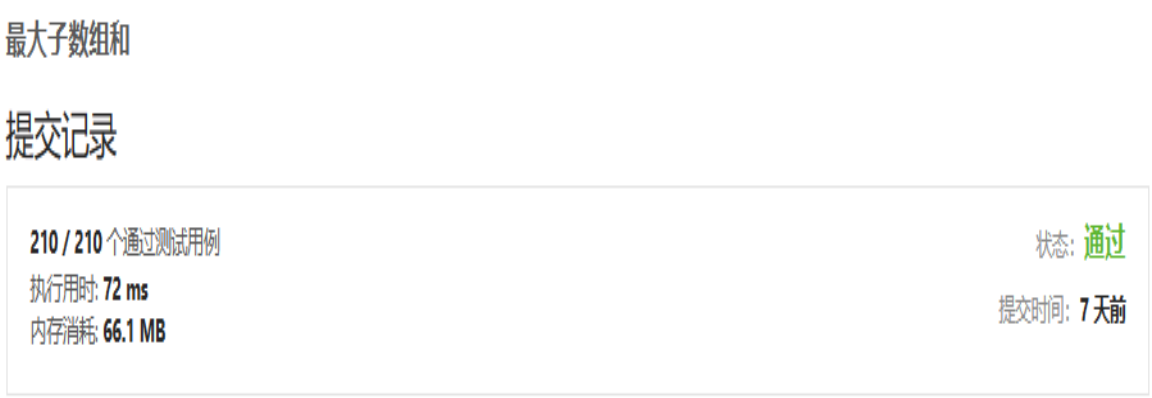
6 最大子数组和

这道题需要找出和最大的子数组，状态 dp[i] 表示从 1 到 i 最大的连续子数组和，易得当 dp[i] 大于 0 时则其对后面的答案有贡献

而当 dp[i] 小于 0 时，则不计算其的负贡献，所以易得递归方程如下

$$dp[i] = \begin{cases} 0 & \text{if } dp[i-1] < 0, \\ dp[i-1] + num[i] & \text{if } dp[i-1] > 0, \end{cases}$$

事实上，观察递归方程，我们可以通过一个变量 tot 来遍历整个数组，遍历整个数组 tot=tot+num[i] 当 tot<0 时则将 tot 清零，否则继续循环



7　三角形最小路径和

对于这道题，从上往下走，分析较为复杂，考虑从下往上走
一个点只能来自其左下与右下的点即 triangle[i+1][j] 与 triangle[i+1][j+1] 即

$$triangle[1][1] = triangle[1][1] + min(triangle[2][1], triangle[2][2])$$
$$triangle[2][1] = triangle[2][1] + min(triangle[3][1], triangle[3][2])$$
$$triangle[2][2] = triangle[2][2] + min(triangle[3][2], triangle[3][3])$$
$$.....$$

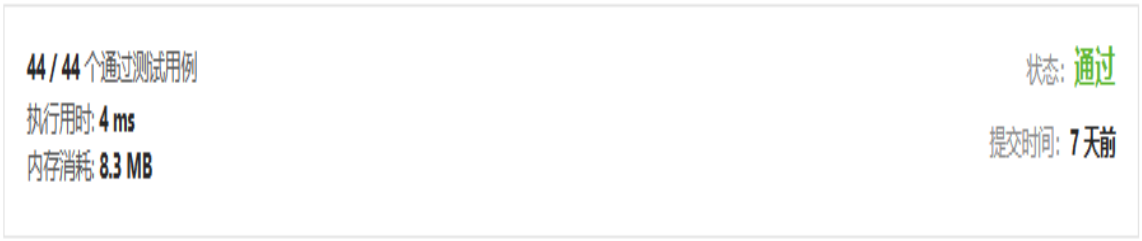
可推出方程 triangle[i][j]=triangle[i][j]+min(triangle[i+1][j],triangle[i+1][j+1])
状态转移方程如下

$$triangle[i][j] = triangle[i][j] + min(triangle[i+1][j], triangle[i+1][j+1])$$

遍历方向为: 从下到上从左到右

三角形最小路径和

提交记录



8 最长递增子序列

记 dp[i] 表示对于前 i 个数，在选择第 i 个数的情况下, 得到最长的单调不升子序列的长度,

当第 i 个数是子序列的第一项,dp[i]=1

第 i 个数不是子序列的第一项, 选择的第 i 个数, 假设其前一项为第 j 个数，即第 i 个数接在第 j 个数结尾的序列后面枚举这样的 j，可以得到状态转移方程:dp[i]=dp[j]+1

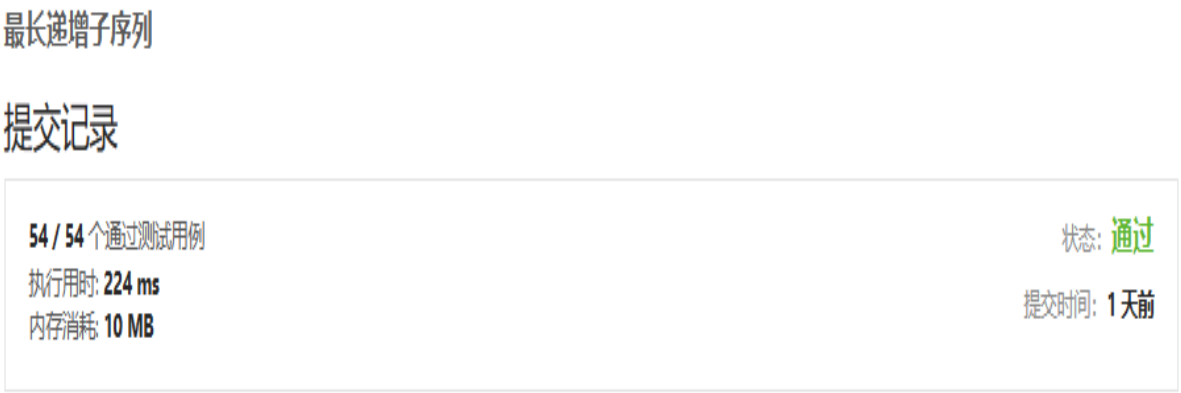
$$dp[i] = max(dp[i], dp[j] + 1) \qquad j < i \qquad if(nums[i] > nums[j])$$

上述算法时间复杂度为 O(n^2)

事实上存在着 O$(nlog(n))$ 的解法

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int lengt=nums.size();
        int len=0,dp[2501];
        dp[0]=nums[0];
        for(int i=1;i<lengt;i++)
        {
            if(dp[len]<nums[i]) dp[++len]=nums[i];
            else dp[lower_bound(dp,dp+len,nums[i])-dp]=nums[i];
        }
        return len+1;
    }
};
```

O(n^2) 解法耗时如下



O$(nlog(n))$ 解法耗时如下

最长递增子序列

提交记录

54 / 54 个通过测试用例
执行用时: **8 ms**
内存消耗: **10 MB**

状态: 通过
提交时间: **9 天前**