Assignment #B: dp

Updated 1448 GMT+8 Nov 18, 2025

2025 fall, Complied by **韩旭 元培学院**

说明：

1）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

2）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

3）如果不能在截止前提交作业，请写明原因。

1. 题目

LuoguP1255 数楼梯

dp, bfs, https://www.luogu.com.cn/problem/P1255

用时：2min

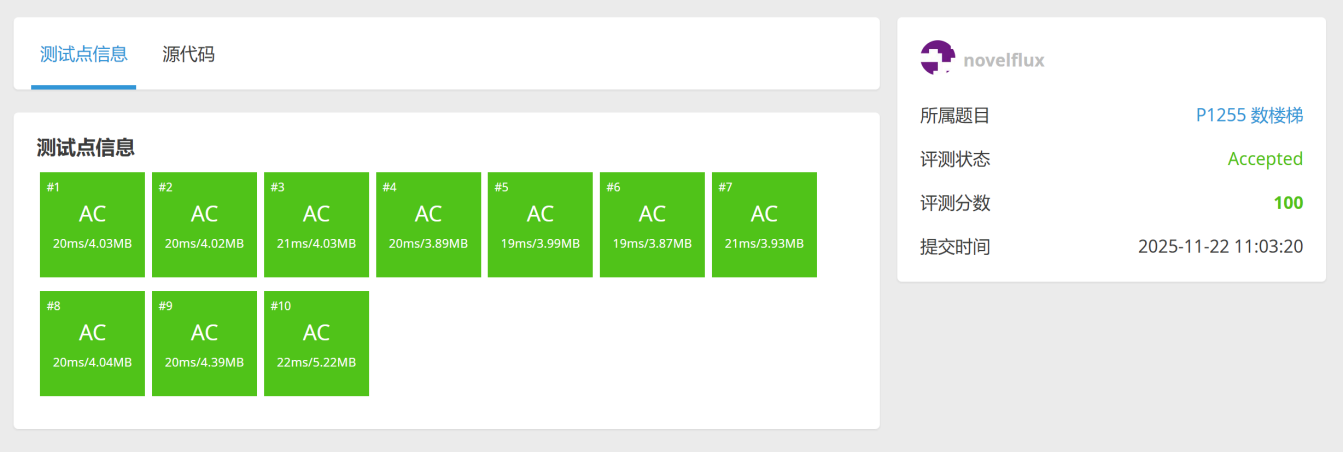
思路

状态变量为楼梯数，由于每一个状态都有走一步和走两步两种解法，所以对于任意一个楼梯数i，其总走法等于i-1对应的走法加上i-2对应的走法。按照这个状态转移方程进行DP即可。

代码：

```
N=int(input())
dp=[0]*(N+1)
dp[0]=1
dp[1]=1
for i in range(2,N+1):
    dp[i]=dp[i-1]+dp[i-2]
print(dp[N])
```

代码运行截图



27528: 跳台阶

dp, http://cs101.openjudge.cn/practice/27528/

用时：3min

思路

在这个题里，每一个状态i的总走法都是先走1,2,...,i步共计i种解法的走法和，按照这个状态转移方程进行DP即可。

代码：

```
N=int(input())
dp=[0]*(N+1)
dp[0]=1
dp[1]=1
for i in range(2,N+1):
    for j in range(1,i+1):
        dp[i]+=dp[i-j]
print(dp[N])
```

代码运行截图

#50941982提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
N=int(input())
dp=[0]*(N+1)
dp[0]=1
dp[1]=1
for i in range(2,N+1):
    for j in range(1,i+1):
        dp[i]+=dp[i-j]
print(dp[N])
```

基本信息

#: 50941982
题目: 27528
提交人: 22n2200017737
内存: 3624kB
时间: 23ms
语言: Python3
提交时间: 2025-11-22 11:07:35

English 帮助 关于

M23421:《算法图解》小偷背包问题

dp, http://cs101.openjudge.cn/pctbook/M23421/

用时：20min

思路

小偷背包是一道二维动态规划的题目，因为如果只考虑背包载重这一个维度的话，当小偷偷走某一个物品后，就不能再选择该物品了，子问题的结构就会发生变化，所以还需要纳入商品的维度。`dp[n][b]`表示商场里只有前n件物品，背包载重为b的时候，所能装下物品的最高价值。边缘状态为：只考虑第1件物品，那么只需要讨论这件物品本身能不能装进背包即可。状态转移方程为：如果第n件物品自己不能装进背包，那么`dp[n][b]=dp[n-1][b]`；如果可以，那么有两种可能，要么先把第n件物品装进背包，然后用剩余空间装前n-1件物品，要么不装第n件物品。`dp[n][b]`就是这两者中的较大值。

代码：

```
N,B=map(int,input().split())
price=list(map(int,input().split()))
weight=list(map(int,input().split()))
dp=[[0]*(B+1) for _ in range(N)]
for i in range(N):
    for j in range(B+1):
        if i==1:
            if weight[i]<=j:
                dp[i][j]=price[i]
        else:
            if weight[i]>j:
                dp[i][j]=dp[i-1][j]
            else:
                dp[i][j]=max(price[i]+dp[i-1][j-weight[i]],dp[i-1][j])
print(dp[N-1][B])
```

代码运行截图

#50981786提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
N,B=map(int,input().split())
price=list(map(int,input().split()))
weight=list(map(int,input().split()))
dp=[[0]*(B+1) for _ in range(N)]
for i in range(N):
    for j in range(B+1):
        if i==1:
            if weight[i]<=j:
                dp[i][j]=price[i]
        else:
            if weight[i]>j:
                dp[i][j]=dp[i-1][j]
            else:
                dp[i][j]=max(price[i]+dp[i-1][j-weight[i]],dp[i-1][j])
print(dp[N-1][B])
```

基本信息

#: 50981786
题目: M23421
提交人: 22n2200017737
内存: 3612kB
时间: 23ms
语言: Python3
提交时间: 2025-11-24 20:27:44

English 帮助 关于

M5.最长回文子串

dp, two pointers, string, https://leetcode.cn/problems/longest-palindromic-substring/

用时：30min

思路

这道题告诉我们，使用DP的题也未必要把最终结果存储到dp数组里面，不然就会超时。这道题的dp数组只需要存储从l到r的子序列是不是回文子序列即可，状态转移方程为：如果`s[l]==s[r]`并且从l+1到r-1也是回文子序列（或者l和r之差小于等于2），那么从l到r也是回文子序列。然后再额外维护一个最长回文子序列的开头和长度（这里最好直接记录长度，所以不维护开头结尾），每次发现一个回文子序列，就比较它和目前最长的长度，如果新的回文子序列更长，就依照它来更新最长回文子序列的开头和长度。最后按此输出即可。另外这里有一个小细节，由于dp的状态转移方程需要用到`dp[l+1][r-1]`，所以在循环的时候，需要要求`dp[l+1][r-1]`比`dp[l][r]`先得到更新。所以外层循环是r，内层循环是l。

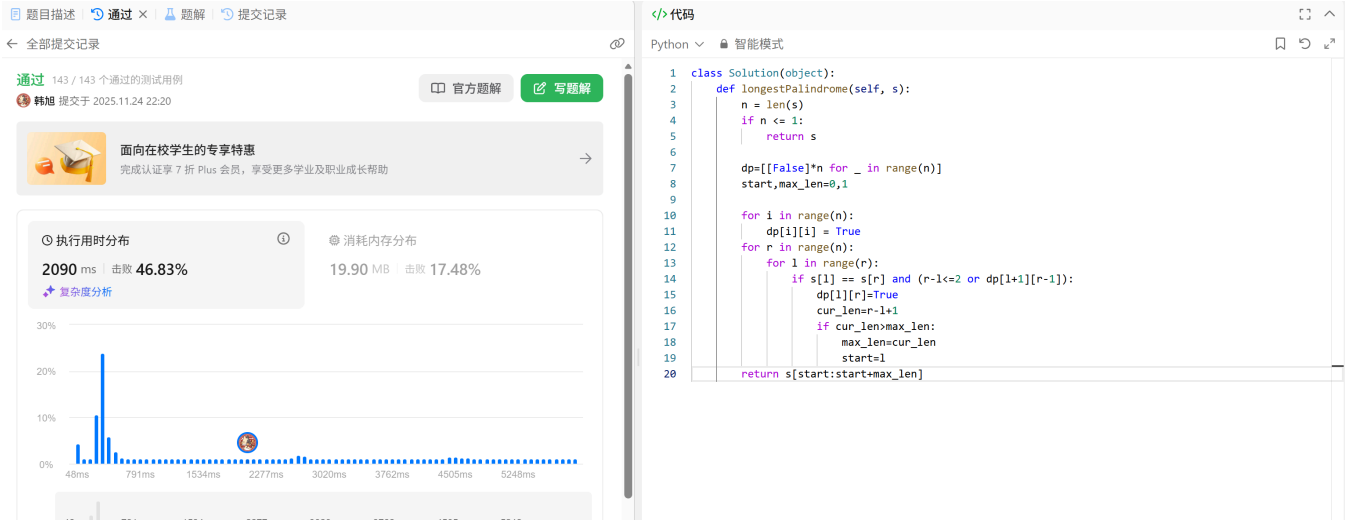
代码：

```
class Solution(object):
    def longestPalindrome(self, s):
        n = len(s)
        if n <= 1:
            return s

        dp=[[False]*n for _ in range(n)]
        start,max_len=0,1
```

```
        for i in range(n):
            dp[i][i] = True
        for r in range(n):
            for l in range(r):
                if s[l] == s[r] and (r-l<=2 or dp[l+1][r-1]):
                    dp[l][r]=True
                    cur_len=r-l+1
                    if cur_len>max_len:
                        max_len=cur_len
                        start=l
        print(s[start:start+max_len])
```

代码运行截图



474D. Flowers

dp, 1700 https://codeforces.com/problemset/problem/474/D

本题暂时跳过。

M198.打家劫舍

dp, https://leetcode.cn/problems/house-robber/

本题暂时跳过。

2. 学习总结和收获

这周其他作业有点多，先只做四个题。这四个题都是比较基本的dp思路，dp在代码实现上的基本套路比较好学，关键是想清楚dp的状态和值分别对应什么，以及状态转移方程的关系。最长回文子串那个题中可以看出，dp并非一定要把最终答案存在 dp 数组中，有时只需要 dp 来判断可行性，再额外维护答案（例如回文子串的起点与长度），从而保证效率。另外，dp一般通过循环来实现，循环的顺序决定了dp的表格记录能不能成功，这就取决于状态转移方程的依赖关系。