

# 动态规划习题课

## 动态规划习题课

### 一、海贼 OJ-46-切割回文

状态定义

状态转移

### 二、海贼-OJ-47-01背包

状态定义

状态转移

### 三、海贼 OJ-48-完全背包

状态定义

状态转移

### 四、海贼 OJ-49-多重背包

问题模型转换

状态定义

状态转移

### 五、海贼 OJ-50-扔鸡蛋

状态定义

状态转移

## 一、海贼 OJ-46-切割回文

---

### 状态定义

$dp[i]$  代表取字符串的前  $i$  位，最少分成多少段回文串

### 状态转移

$$dp[i] = \min(dp[j]) + 1 \mid s[j+1, i] \text{ is palindrome}$$

1. 根据状态转移，算法时间复杂度  $O(n^2)$
2. 所以，我们需要对转移阶段进行优化
3. 动态规划优化章节的时候，重点解决

Chrome 文件 编辑 视图 历史记录 书签 用户 标签页 窗口 帮助

apen Central Ms Ltd (BMO) 14 2 97 164421 70%

OJ - Online Judge

不安全 | oj.haizeix.com/problem/46

应用 苹果中国 work url 前端技术 技术博客 信息学 笔试面试相关 MAC相关 生活相关 工商税务 动漫娱乐 人工智能 网站开发 科技信息 iCloud 新浪微博 腾讯微博 W 维基百科 百度 中国雅虎 其他书签

## #46. 练习题3：切割回文

描述 提交 在线 IDE 管理 题解视频 上一题 下一题 统计

**题目描述**

给出一个字符串S，问对字符串S最少切几刀，使得分成的每一部分都是一个回文串（注意：单一字符是回文串）

输入 (1 ≤ n ≤ 500000) 只包含小写字母。

输出 输出一个整数，代表所切的最少刀数。

**样例输入**

sehuhezexe

**样例输出**

4

dp[i][j] 以 i 位置为结尾，最少为多少段  
 $dp[i][j] = \min(dp[j][i]) + 1 \quad | \quad j+1 \leftarrow i \text{ 回文}$

dp[i][j] 1

## 二、海贼-OJ-47-01背包

### 状态定义

$dp[i][j]$  前  $i$  个物品，背包最大承重为  $j$  的情况下，最大价值

### 状态转移

$$dp[i][j] = \max \begin{cases} dp[i-1][j] & \text{没选第 } i \text{ 件} \\ dp[i-1][j-v[i]] + w[i] & \text{选了第 } i \text{ 件} \end{cases}$$

1. 第一种程序实现，状态如何定义的，程序就如何实现
2. 第二种程序实现，使用滚动数组，对代码进行了空间优化
3. 第三种程序实现，将程序中的  $dp$  数组变成一维的，并且修改了更新顺序

```
19 #define MAX_N 100
20 #define MAX_V 10000
21 int dp[MAX_V + 5];
22
23 int main() {
24     int V, n, v, w;
25     cin >> V >> n;
26     for (int i = 1; i <= n; i++) {
27         cin >> v >> w;
28         for (int j = V; j >= v; j--) {
29             dp[j] = max(dp[j], dp[j - v] + w);
30         }
31     }
32     cout << dp[V] << endl;
33     return 0;
34 }
```

Handwritten notes on the code:

- Diagram:  $dp[j-v]$  —  $i-1, j \rightarrow dp[j]$  with a box containing  $i, j$ .
- Text: 动态 (Dynamic)
- Text:  $dp[j]$  更新之前 (Before updating  $dp[j]$ )
- Text:  $dp[j]$  更新之后 (After updating  $dp[j]$ )

Terminal output:

```
<动路线/6.海贼班/3.直播课内容/X.直播课/4.HZ0J47-2.cpp [FORMAT=unix] [TYPE=CPP] [POS=29,13][85%]
"4.HZ0J47-2.cpp" 34L, 752C 已写入
```

### 三、海贼 OJ-48-完全背包

#### 状态定义

$dp[i][j]$  前  $i$  个物品，背包最大承重为  $j$  的情况下，最大价值

#### 状态转移

$$dp[i][j] = \max \begin{cases} dp[i-1][j] & \text{没选第 } i \text{ 件} \\ dp[i][j-v[i]] + w[i] & \text{选了若干个第 } i \text{ 件} \end{cases}$$

程序实现的时候，参考01背包的程序实现，将逆向刷表，改成正向刷表

### 四、海贼 OJ-49-多重背包

## 问题模型转换

1. 多重背包，每类物品多了一个数量限制
2. 01背包，每种物品只有一个
3. 将多重背包中的数量限制，当做多个单一物品来处理
4. 至此就将多重背包，转成了0/1背包问题

## 状态定义

$dp[i][j]$  前  $i$  个物品，背包最大承重为  $j$  的情况下，最大价值

## 状态转移

$$dp[i][j] = \max \begin{cases} dp[i-1][j] & \text{没选第 } i \text{ 件} \\ dp[i-1][j-v[i]] + w[i] & \text{选了第 } i \text{ 件} \end{cases}$$

## 五、海贼 OJ-50-扔鸡蛋

---

### 状态定义

$dp[n][m]$  用  $n$  个鸡蛋，测  $m$  层楼，最坏情况下最少测多少次

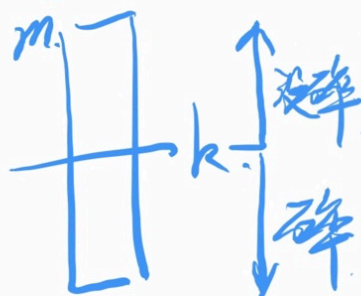
### 状态转移

$$dp[n][m] = \min(\max \begin{cases} dp[n-1][k-1] + 1 & \text{鸡蛋碎了} \\ dp[n][m-k] + 1 & \text{鸡蛋没碎} \end{cases})$$

1. 程序所使用的存储空间与楼层数量强相关
2. 楼层数量达到了  $2^{31}$ ，所以在这种状态定一下不可行
3. 状态定义不可行，我们就需要优化状态定义
4. 时间复杂度  $O(n \times m^2)$ ，当  $m$  过大的时候，无法通过时间限制

# 状态转移

$n, m-k$



$[m, k-1]$

$$dp[i][j] = \max \begin{cases} dp[i-1][j] & \text{没选第 } i \text{ 件} \\ dp[i-1][j-v[i]] + w[i] & \text{选了第 } i \text{ 件} \end{cases}$$

$dp[n][m-k]$   $n$  个鸡蛋, 测  $m$  层楼, 最少测试

$$dp[n][m-k] = \min_{+1} (\max (dp[n-1][k-1], dp[n-1][m-k]))$$

