Atcoder Educational DP Contest 题解

张晴川

March 14, 2020

Contents

前言

本文为 **Atcoder** 上的 *Educational DP Contest* 的简要题解。该场比赛共有 26 题，从易到难，是练习动态规划的优秀之选。

提交地址为: `https://atcoder.jp/contests/dp`

本文会在我的 github 上 持续更新。如发现错误或有修改建议，请在 Github 上提 issue。如觉得有帮助，不妨 star 一下以支持我持续写作。

A Frog 1

题意

有 n 块石头排成一排，第 i 块石头的高度为 h_i。青蛙现在站在第 1 块石头上。它可以从第 i 块石头跳到第 $i+1$ 或者 $i+2$ 块石头上。从第 i 块石头跳到第 j 块需要消耗 $|h_j - h_i|$ 点体力。请求出跳到第 n 块石头所需的最小体力值。

数据范围

- $1 \le n \le 10^5$

题解

设 $dp[i]$ 为跳到第 i 块石头上所需的最小体力值，则 $dp[n]$ 为题之所求。根据题意，不难发现有：

$$dp[i] = \begin{cases} 0 & i = 1 \\ |h_2 - h_1| & i = 2 \\ \min(dp[i-1] + |h_i - h_{i-1}|, dp[i-2] + |h_i - h_{i-2}|) & i \ge 3 \end{cases}$$

$O(N)$ 递推一下即可。

核心代码

```
dp[1] = 0
dp[2] = abs(h[2]-h[1]);
for(int i = 2;i<=n;i++){
    dp[i] = min(dp[i-1]+abs(h[i]-h[i-1]),dp[i-2]+abs(h[i]-h[i-2]));
}
```

B　Frog 2

题意

有 n 块石头排成一排，第 i 块石头的高度为 h_i。青蛙现在站在第 1 块石头上。它可以从第 i 块石头跳到第 $i+1, i+2, \ldots, i+k$ 块石头上。从第 i 块石头跳到第 j 块需要消耗 $|h_j - h_i|$ 点体力。请求出跳到第 n 块石头所需的最小体力值。

数据范围

- $1 \le n \le 10^5$
- $1 \le k \le 100$

题解

设 $dp[i]$ 为跳到第 i 块石头上所需的最小体力值，则 $dp[n]$ 为题之所求。根据题意，不难发现有：

$$dp[i] = \begin{cases} 0 & i = 1 \\ \min\{dp[j] + |h_i - h_j| \mid i - k \le j < i\} & i \ge 2 \end{cases}$$

$O(nk)$ 递推一下即可。

核心代码

```
dp[1] = 0
for(int i = 2;i<=n;i++){
    for(int j = max(i-k,1);j<i;j++){
        dp[i] = min(dp[i],dp[j]+abs(h[i]-h[j]));
    }
}
```

C Vacation

题意

太郎的暑假有 n 天，第 i 天他可以选择以下三种活动之一：

- 游泳，获得 a_i 点幸福值。
- 捉虫，获得 b_i 点幸福值。
- 写作业，获得 c_i 点幸福值。

但他不能连续两天进行同一种活动，请求出最多可以获得多少幸福值。

数据范围

- $1 \le n \le 10^5$

题解

设 $A[i], B[i], C[i]$ 分别表示在第 i 天进行三种活动的前提下，前 i 天的最大幸福值。那么可以得到，

$$A[i] = \begin{cases} a_1 & i = 1 \\ \max(B[i-1], C[i-1]) + a_i & i \ge 2 \end{cases}$$

$B[i]$ 和 $C[i]$ 的求法类似，故不赘述。实现时可以开一个大小为 $n \times 3$ 的二维数组。

核心代码

```
//v[d][i] 表示第 d 天进行第 i 项活动获得的幸福值
for(int i = 0;i<3;i++){
    dp[1][i] = v[1][i];
}
for(int day = 2;day<=n;day++){
    for(int cur = 0;cur<3;cur++){
        dp[day][cur] = 0;
        for(int last = 0;last<3;last++){
            if(cur != last){
                dp[day][cur] = max(dp[day][cur],dp[day-1][last] + v[day][cur]);
            }
        }
    }
}
```

D Knapsack 1

题意

经典 01 背包问题。有 n 个物品，每个物品有重量 w_i 和价值 v_i。背包的容量为 W，换句话说，可以放进背包的物品的重量总和不能超过 W。请最大化被选取物品的价值总和。

数据范围

- $1 \le n \le 100$
- $1 \le W \le 10^5$
- $1 \le v_i \le 10^9$

题解

我们设 $dp[i][j]$ 表示 **"只考虑前 i 个物品的情况下，背包容量是 j 所能凑出的最大价值之和"**。那么在计算 $dp[i][j]$ 的时候，所有情况可以分成两类考虑，

1. **拿第 i 件物品**，那么现在背包容量只剩下 $j - w_i$，而由于每样物品只能拿一件，所以我们只需要考虑前 $i-1$ 件物品的最优选取方式，即最终价值为 $dp[i-1][j-w_i] + v_i$。
2. **不拿第 i 件物品**，那么背包容量仍然是 j。由于我们选择不拿第 i 件物品，现在只需要考虑前 $i-1$ 件物品的最优选取方式，即最终价值为 $dp[i-1][j]$。

所以可以得到转移方程为：

$$dp[i][j] = \begin{cases} 0 & i = 0 \\ \max(dp[i-1][j-w_i] + v_i, dp[i-1][j]) & i \ge 1 \end{cases}$$

递推或记忆化搜索的复杂度均为 $O(nW)$。

核心代码

```
//注意要开 long long
for(int i = 1;i<=n;i++)
    for(int j = 0;j<=W;j++)
        if(j < w[i])
        //容量不够
            dp[i][j] = dp[i-1][j];
        else
        //容量足够
            dp[i][j] = max(dp[i-1][j-w[i]] + v[i], dp[i-1][j]);
```

E　Knapsack 2

题意

同上题，仅数据范围不同：

数据范围

- $1 \le n \le 100$
- $1 \le W \le \mathbf{10^9}$
- $1 \le v_i \le \mathbf{10^3}$

题解

由于 W 达到了 10^9 的量级，之前的 $O(nW)$ 算法无法通过，但由于每样物品的价值上限仅为 10^3，我们可以另辟蹊径。设 $dp[i][j]$ 表示 **"只考虑前 i 个物品的情况下，总价值是 j 所需的最小容量"**。那么在计算 $dp[i][j]$ 的时候，所有情况依然可以分成两类考虑：

1. **拿第 i 件物品**，那么别的物品的总价值需要凑出 $j - v_i$，而由于每样物品只能拿一件，所以我们只需要考虑前 $i - 1$ 件物品的最优选取方式，即最终重量为 $dp[i-1][j-v_i] + w_i$。
2. **不拿第 i 件物品**，那么别的物品需要凑出 j 的价值。由于我们选择不拿第 i 件物品，现在只需要考虑前 $i - 1$ 件物品的最优选取方式，即最小重量为 $dp[i-1][j]$。

计算完所有状态的值后，只需要选取满足重量上限的最大价值即可。

核心代码

```
int bound = n*1000;//最大价值
for(int v = 0;v<=bound;v++)
    if(v == 0) dp[0][v] = 0;
    else dp[0][v] = inf;
for(int i = 1;i<=n;i++)
    for(int tot = 0;tot<=bound;tot++)
        if(tot < v[i])
            dp[i][tot] = dp[i-1][tot]
        else
            dp[i][tot] = min(dp[i-1][tot],dp[i-1][tot-v[i]] + w[i]);
answer = 0
for(int tot = 0;tot<=bound;tot++)
    if(dp[n][tot] <= W)
        answer = tot
```

F LCS

题意

给定字符串 s 和 t，求最长公共子序列。

数据范围

- $1 \le |s|, |t| \le 3000$

题解

首先注意到一般认为子序列是**不需要**连续的。

设 s, t 的长度分别为 n, m。现在考虑两个串的最后字符，一共有两种情况：

1. $s_n = t_m$，那么答案最末字符显然等于也等于它们俩。
2. $s_n \ne t_m$，那么两者必有其一不在答案中，所以答案为 $LCS(s_{1\dots n}, t_{1\dots m-1})$ 和 $LCS(s_{1\dots n-1}, t_{1\dots m})$ 的较长者。

用 $dp[i][j]$ 表示 s 的前 i 个字符和 t 的前 j 个字符的 LCS 长度。那么可以得到转移方程

$$dp[i][j] = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ dp[i-1][j-1] + 1 & s[i] == t[j] \\ \max(dp[i-1][j], dp[i][j-1]) & \text{otherwise} \end{cases}$$

答案本身只需要通过 $dp[i][j]$ 的转移来源恢复即可，参考以下代码。

核心代码

```
// 计算最优长度
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= m; j++)
        if(s[i] == t[j])
            dp[i][j] = dp[i - 1][j - 1] + 1;
        else
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
// 恢复答案
int i = n, j = m;
while(dp[i][j] > 0)
    if(s[i] == t[j]) {
        ans[dp[i][j]] = s[i];
        i--;
        j--;
    } else {
        if(dp[i][j] == dp[i - 1][j])i--;
```

```
        else j--;
    }
```

G Longest Path

题意

给定一个 n 个点，m 条边的**有向无环图**，求图中最长路径的长度。定义路径长度为边的数量。

数据范围

- $2 \le N \le 10^5$
- $1 \le M \le 10^5$

题解

本题是**所有**动态规划模型的**理论**基础。抽象地说，所有动态规划的本质都是在一张（带权）有向无环图上求最长路，之前做的背包，最长公共子序列皆是如此。对应关系如下：

动态规划	**有向无环图**
状态	节点
转移方程	边
无后效性	图中无环
$\cdots$	$\cdots$

这题的转移方程比较显然，即 $dp[i]$ 表示以节点 i 为终点的路径中最长的长度。转移方程为：

$$dp[i] = \max\{dp[j] + 1 \mid \text{存在边 } j \to i\}$$

有向无环图最重要的性质就是没有环，这使得我们可以对节点进行拓扑排序，并按照拓扑序的顺序计算 dp 状态的值。相信你对于宽度优先搜索（BFS）没有什么问题，以下我给出一种用深度优先搜索（DFS）的做法，又称记忆化搜索。

核心代码

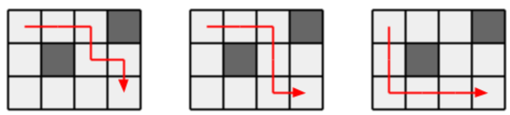
```
void calc(int i) {
    if(vis[i])
        return dp[i];
    else {
        dp[i] = 0;
        vis[i] = true;
        for(auto j:reversed_edges[i]) {
            dp[i] = max(dp[i], calc(j) + 1);
        }
        return dp[i];
    }
}
```

H　Grid 1

题意

给定一个 h 行 w 列的网格，现在位于 $(1,1)$，每次可以向下或右走一格，求有多少种方式（对 10^9+7 取模）走到 (h,w)。

网格中有若干个不能经过的障碍，但不会出现在起点和终点。



数据范围

- $2 \le h,w \le 1000$

题解

这题比较简单，用 $dp[i][j]$ 表示走到 (i,j) 的方案数，只需要枚举来源即可。转移方程为：

$$dp[i][j] = \begin{cases} 0 & (i,j) \text{ 是障碍} \\ dp[i-1][j] + dp[i][j-1] & \text{otherwise} \end{cases}$$

顺带一提，如果没有障碍，这个 dp 表格就是 杨辉三角，即 $dp[i][j] = \binom{i+j}{i}$ 。

核心代码

```
dp[1][1] = 1;
for(int i = 1; i <= h; i++) {
    for(int j = 1; j <= w; j++) {
        if(i == 1 and j == 1)continue;
        if(dp[i][j] == '.') //如果可以经过
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
    }
}
```

I Coins

题意

有 n 枚硬币排成一排。现在同时抛所有硬币，第 i 枚硬币向上的概率是 p_i，向下的概率是 $1-p_i$，求向上的硬币数量比向下的多的概率。

数据范围

- $1 \le n \le 2999$
- n 是奇数

题解

这题依旧很简单，设 $dp[i][j]$ 表示前 i 个元素有 j 个向上的概率。只需要枚举当前硬币是向上还是向下即可，与背包类似。转移方程为：

$$dp[i][j] = dp[i-1][j-1] \times p_i + dp[i-1][j] \times (1-p_i)$$

计算完毕后，枚举有多少枚硬币向上就做完了。

核心代码

```
prob[0][0] = 1;
for(int i = 1;i<=n;i++){
    dp[i][0] = dp[i-1][0] * (1-p[i]);
    for(int j = 1;j<=i;j++){
        dp[i][j] = dp[i-1][j-1] * p[i] + dp[i-1][j] * (1-p[i]);
    }
}

ans = 0;
for(int j = 0;j<=n;j++){
    if(j > n-j){
        ans += dp[n][j];
    }
}
```

J Sushi

题意

现在有 n 盘寿司在桌上排成一排，第 i 盘寿司里面有 a_i 个寿司。直到吃完所有寿司为止，你都会持续进行下述操作：

- 从 $\{1, 2, \cdots, n\}$ 中**均匀随机**选取一个数 i。如果第 i 盘中仍有寿司，那就吃一块，否则什么也不做。

现在求期望操作次数是多少。

数据范围

- $1 \le n \le 300$
- $1 \le a_i \le 3$

题解

暴力 DP

首先我们考虑最暴力的做法，用 $dp[c_1][c_2]\cdots[c_n]$ 表示第 i 盘还剩 c_i 个寿司的期望次数。那么枚举随机到的数 i 就可以得到方程：

$$dp[c_1][c_2]\cdots[c_n] = 1 + \sum_{i=1}^{n} \frac{1}{n} dp[c_1][c_2]\cdots[\max(c_i - 1, 0)]\cdots[c_n]$$

合并状态

（注意该方程并不能构成**转移**方程，因为当第 i 盘已经为空的时候，状态不变）

我们现在考虑**合并等价状态**（题外话：动态规划优化的初等方法无非就那么几种，合并状态就是最重要的思想之一）。

注意到由于随机数是均匀选取的，那么盘子的位置是无关紧要的，而只有剩余数量有影响。于是相同数值的不同排列的期望次数必然是相同的。例如 $dp[1][2][3] = dp[3][2][1]$。

所以只需要考虑每种数值出现的次数。因为 $a_i \le 3$，所以至多有四种不同数值：$\{0, 1, 2, 3\}$。我们重新定义状态 $dp[a][b][c][d]$ 表示当前还剩下 $a/b/c/d$ 盘有 $0/1/2/3$ 个寿司。

有了状态，我们通过枚举当前随机到的盘子里还剩几只寿司得到如下方程：

$$\begin{aligned}dp[a][b][c][d] = 1&+\frac{a}{n}dp[a][b][c][d]\\&+\frac{b}{n}dp[a+1][b-1][c][d]\\&+\frac{c}{n}dp[a][b+1][c-1][d]\\&+\frac{d}{n}dp[a][b][c+1][d-1]\end{aligned}$$

移项整理得到**转移**方程：

$$\begin{aligned}dp[a][b][c][d] =&\frac{n}{b+c+d}\\&+\frac{b}{b+c+d}dp[a+1][b-1][c][d]\\&+\frac{c}{b+c+d}dp[a][b+1][c-1][d]\\&+\frac{d}{b+c+d}dp[a][b][c+1][d-1]\end{aligned}$$

消除无用状态

但由于 $n \leq 300$，总状态数高达 $300^4 \approx 8 \times 10^9$ 无法通过。我们需要进一步优化。

注意到任意时刻下，$a+b+c+d$ 的值都应该等于 n：因为不管进行多少次操作，盘子总数总是 n。所以其实只需要知道 b, c, d 就可以反推出 a 的值：$a = n-(b+c+d)$。那么现在只保留后面三维，得到转移方程：

$$\begin{aligned}dp[b][c][d] =&\frac{n}{b+c+d}\\&+\frac{b}{b+c+d}dp[b-1][c][d]\\&+\frac{c}{b+c+d}dp[b+1][c-1][d]\\&+\frac{d}{b+c+d}dp[b][c+1][d-1]\end{aligned}$$

至此，足矣。

核心代码

```
double calc(int b, int c, int d) {
    if(min({b,c,d}) < 0)return 0;
    if(vis[b][c][d])
        return dp[b][c][d];
    else {
        vis[b][c][d] = true;
        double sum = b + c + d;
        dp[b][c][d] = n / sum
                    + b / sum * calc(b - 1, c, d)
                    + c / sum * calc(b + 1, c - 1, d)
                    + d / sum * calc(b, c + 1, d - 1);
        return dp[b][c][d];
    }
}
```

K

题意

题解

核心代码

```
K
```

L

题意

题解

核心代码

```
L
```

M

题意

题解

核心代码

```
M
```

N

题意

题解

核心代码

```
N
```

O

题意

题解

核心代码

```
0
```

P

题意

题解

核心代码

```
P
```

Q

题意

题解

核心代码

```
Q
```

R

题意

题解

核心代码

```
R
```

S

题意

题解

核心代码

```
S
```

T

题意

题解

核心代码

```
T
```

U

题意

题解

核心代码

```
U
```

V

题意

题解

核心代码

```
V
```

W

题意

题解

核心代码

```
W
```

X

题意

题解

核心代码

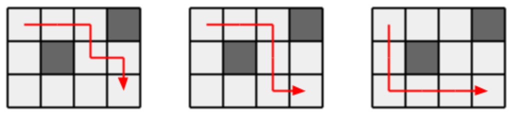
```
X
```

Y　Grid 2

题意

给定一个 h 行 w 列的网格，现在位于 $(1,1)$，每次可以向下或右走一格，求有多少种方式（对 10^9+7 取模）走到 (h,w)。

网格中有 n 个不能经过的障碍，但不会出现在起点和终点。



数据范围

- $2 \le h, w \le 10^5$
- $1 \le n \le 3000$

题解

(注：由于非此题重点，本题解不包含计算组合数取模的教程，请自行上网搜板子）本题是 *Grid 1* 的加强版。由于 $h \times w$ 高达 10^{10}，无法使用之前的方法。接下来我们来使用常见于~~小学奥数~~的**容斥原理**解决这一题。

没有障碍　那么答案就是组合数 $\binom{(h-1)+(w-1)}{h-1}$ ——因为一共要走 $(h-1)+(w-1)$ 步，选择其中的 $h-1$ 步向下。

一个障碍　设障碍格为 (a,b)，那么答案应该是总方案数减去经过障碍格的不合法方案数，即:

$$\underbrace{\binom{(h-1)+(w-1)}{h-1}}_{\text{总方案数}} - \underbrace{\binom{(a-1)+(b-1)}{a-1}\binom{(h-a)+(w-b)}{h-a}}_{\text{经过 (a,b) 的非法方案数}}$$

两个障碍　设障碍格为 $A:(a,b)$ 和 $B:(c,d)$，那么答案等于总方案数减去经过 A 的不合法方案数减去经过 B 的不合法方案数最后加回被计算了两次的同时经过 A 和 B 的不合法方案数。关于最后一部分，这里有两种情况。

1. 不存在同时经过的路径，如上图所示。那么数量就是 0，即答案为

$$
\begin{aligned}
&\binom{(h-1)+(w-1)}{h-1}\\
-&\binom{(a-1)+(b-1)}{a-1}\binom{(h-a)+(w-b)}{h-a}\\
-&\binom{(c-1)+(d-1)}{a-1}\binom{(h-c)+(w-d)}{h-c}
\end{aligned}
$$

2. 存在同时经过的路径。假设先经过 A 然后经过 B，那么方案数为

$$
\begin{aligned}
&\binom{(h-1)+(w-1)}{h-1}\\
-&\binom{(a-1)+(b-1)}{a-1}\binom{(h-a)+(w-b)}{h-a}\\
-&\binom{(c-1)+(d-1)}{a-1}\binom{(h-c)+(w-d)}{h-c}\\
+&\binom{(a-1)+(b-1)}{a-1}\binom{(c-a)+(d-b)}{c-a}\binom{(h-c)+(w-d)}{h-c}
\end{aligned}
$$

多个障碍 总的来说，容斥原理告诉我们，对于障碍的每个子集，如果有奇数个就减去，有偶数个就加上。但 n 个障碍就意味着有 2^n 个子集需要计算，这是无法承受的，我们需要加速。

注意到以下事实：

1. 如果一个子集中存在一个格子在另一个格子的**严格**右上方，那么可以直接忽略，因为我们只能向右下方走，无法同时经过。
2. 设障碍的一个子集**从左上到右下**依次为 $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$，那么这个子集的贡献为：

$$
(-1)^m\binom{(x_1-1)+(y_1-1)}{x_1-1}\prod_{i=2}^{m}\binom{(x_i-x_{i-1})+(y_i-y_{i-1})}{x_i-x_{i-1}}\binom{(h-x_m)+(w-y_m)}{h-x_m}
$$

3. 如果把起点和终点也看作的障碍格，那么形式更加简洁：

$$
(-1)^m\prod_{i=2}^{m}\binom{(x_i-x_{i-1})+(y_i-y_{i-1})}{x_i-x_{i-1}}
$$

经过以上铺垫，我们大致可以窥到完整解法的轮廓了。首先把所有障碍（以及起点和终点）按左上到右下的顺序排序。那么我们设 $dp[i][0/1]$ 表示走到第 i 个

格子一共经过了奇数/偶数个障碍。那么转移方程为:

$$dp[i][0/1] = \sum_{\text{j 在 i 的左上方}} dp[j][1/0]\binom{(x_i - x_j) + (y_i - y_j)}{x_i - x_j}$$

初始条件为:

$$dp[(1,1)][1] = 1$$

答案为:

$$dp[(h,w)][0] - dp[(h,w)][1]$$

总复杂度是 $O(n^2)$。

核心代码

```
dp[0][1] = 1;
for(int i=1;i<n+2;i++) {
    for(int j = 0;j<i;j++){
        if(x[j]<=x[i] and y[j]<=y[i]){
            for(int parity = 0;parity<2;parity++){
                dp[i][0] += dp[j][1] * binomial(x[i]-x[j]+y[i]-y[j],x[i]-x[j])
            }
        }
    }
}
ans = dp[n-1][0] - dp[n-1][1]
```

Z　Frog 3

题意

有 n 块石头（编号为 1 到 n）排成一排，第 i 块石头的高度为 h_i，并满足 $h_1 < h_2 < \cdots < h_N$。青蛙现在站在第 1 块石头上。它可以从第 i 块石头跳到第 $i+1, i+2, \ldots, N$ 块石头上。从第 i 块石头跳到第 j 块需要消耗 $(h_j - h_i)^2 + C$ 点体力。请求出跳到第 n 块石头所需的最小体力值。

数据范围

- $2 \le N \le 2 \times 10^5$
- $1 \le C \le 10^{12}$
- $1 \le h_1 < h_2 < \cdots < h_N \le 10^6$

题解

这题用到的技巧被称为**决策单调性**。这是一个貌似高端，但本质简单的知识点。下面我来分别介绍一下什么是**决策**和**单调性**。

什么是决策　首先，我们依旧设 $dp[i]$ 为跳到第 i 块石头上所需的最小体力值。其次，用 $cost(i,k)$ 表示从 i 转移到 k 得到的值，表达式为 $dp[i]+(h_k-h_i)^2+C$。显然有 $dp[k] = \min\{cost(i,k) \mid 1 \le i < k\}$。在计算 $dp[k]$ 的时候，每一个转移源头 i 被称为一个**决策**。

枚举所有的决策的总复杂度是 $O(n^2)$。这显然是不可承受的，于是我们需要探索更多性质。

什么是单调性　假设在考虑 $dp[k]$ 的时候，决策 j 要优于决策 i，即 $cost(i,k) > cost(j,k)$，我们看看可以为计算 $dp[k+1]$ 提供什么便利。根据 $cost(i,k)$ 的定义，我们可以得到决策 i 的成本增幅为：

$$\begin{aligned}
\Delta_i &= cost(i,k+1) - cost(i,k) \\
&= \underbrace{(dp[i] + (h_{k+1}-h_i)^2 + C)}_{cost(i,k+1)} - \underbrace{(dp[i] + (h_k - h_i)^2 + C)}_{cost(i,k)} \\
&= (h_{k+1}-h_i)^2 - (h_k - h_i)^2 \\
&= (h_{k+1} + h_k - 2h_i)(h_{k+1} - h_k)\ \textbf{(平方差公式)}
\end{aligned}$$

同理可得 $\Delta_j = (h_{k+1} + h_k - 2h_j)(h_{k+1} - h_k)$

注意到 $h_i < h_j < h_k < h_{k+1}$ 是**单调递增**的，不难计算 $\Delta_i - \Delta_j = (2h_j - 2h_i)(h_{k+1} - h_k) > 0$，即决策 i 的成本上升得比决策 j 更多。由于决策 j 本来

就是更优的决策，它将一直保持优势下去。于是我们可以得出结论：**一旦一个决策被后来者打败，那么再也不需要被考虑**。

在计算一个状态的值的时候，我们定义成本最小的决策为最优决策，如有多种取最左的。举个例子：

i	1	2	3	4	5
最优决策	N/A	1	1	2	3

由于之前的结论，最优决策从左到右一定是单调不下降的：只有可能出现 111222333，而不会有 111333222。因为一旦 2 被 3 打败，就永远不会比 3 更优了。这就是所谓的**单调性**。

由于决策具有单调性，我们只需要二分出每个决策作为最优决策的起点即可。我们这里用`pair<int,int>`的数组实现，每个元素`(i,pos)`表示决策 `i` 从 `pos`。开始成为最优策略。例如我们用 `[(1,1),(2,4),(3,7)]` 表示 111222333。

鉴于文字描述会过于冗长，直接观察以下伪代码：

核心代码

```
dp = [0,inf,inf,...] # 下标从 0 开始
optimal_strategy = [(1,2)] # 一开始 0 是所有状态的最优策略
for j in [2..n]:
# 首先二分出当前位置的最优策略
    l = 1, r = optimal_strategy.length()
    while l < r:
        mid = (l + r) / 2 + 1
        if optimal_strategy[mid][0] <= j:
            l = mid
        else:
            r = mid-1
    dp[j] = cost(optimal_strategy[l][0],j)
# 然后我们计算成为最优决策的起点
    # 首先去除完全劣于当前策略的策略
    i,pos = optimal_strategy.back()
    while cost(i,pos) > cost(j,pos):
        optimal_strategy.pop()
        i,pos = optimal_strategy.back()
    # 开始二分起点
    l = pos, r = n
    while l < r:
        mid = (l + r) / 2
        if cost(i,mid) <= cost(j,mid):
            l = mid + 1
        else:
            r = mid
    optimal_strategy.push((j,l)); # 二分后 l 就是想求的起点
```