积木

【题意简述】

n 个数的序列，你可以给每个数加上 t(t 为任意自然数)，费用是 t^2， 每个数只能增加一次。另有费用为 c*相邻两数差的绝对值的和。求最小总费用。

【数据范围】

20%的数据满足：n、h≤100
40%的数据满足：n、h≤1000
60%的数据满足：n≤1000
80%的数据满足：n≤100000
100%的数据满足：1≤n、h、c≤1000000

【算法分析】

算法 1：

显然，h 最大的不会再增加。我们可以通过搜索枚举最终序列，并计算花费，更新答案。

时间复杂度: $O(h^n)$

空间复杂度:$O(n)$

期望得分：0~10 分

算法 2：

动态规划，f[i][j]表示将第 i 个数变为 j，前 i 个数的最小花费和是多少，通过枚举 f[i-1][k]转移得到。

时间复杂度:$O(nh^2)$

空间复杂度:$O(nh)$

期望得分：20 分

算法 3：

观察 f[i][j]的转移方程： $f[i][j]=\min\{f[i-1][k]+c*|j-k|+(j-h[i])^2\}$，即

$$f[i][j]=\min(\min\{f[i-1][k]-c*k\mid k\le j\}+c*j,\min\{f[i-1][k]+c*k\mid k>j\}-c*j)+(j-h[i])^2$$

用两个数组分别维护 $\min\{f[i][k]-c*k\mid k\le j\}$ 和 $\min\{f[i][k]+c*k\mid k>j\}$，便能

做到 O(1)转移。

时间复杂度:*O(nh)*

空间复杂度:*O(nh)*

期望得分：40 分

注：算法 2、3 利用滚动数组都可以将空间复杂度降为 *O(h)*

算法 4：

算法 2、3 都建立在 h 的基础上，然而 h 非常大，想要拿到更高的分数，必须设计一个与 h 无关的算法。

先考虑这样一个问题：相邻三个数 i、j、k，i、k 固定不动，那么只有当 j<min(i,k)时，j 才可能增加。设 j 增加 x，则花费是关于 x 的二次函数：

$$f(x)=x^2-2cx+c*(i+k-2*j),x\in[0,\min(i,k)-j]且x\in N$$

我们可以在 O(1)的时间内求出函数的最小值。

类似的，若是 h[i]、h[j](i<j)固定不动，i+1~j-1 的 h 值全都改变。设 k 为 i+1~j-1 中 h 最大的，若 h[k]≥min(h[i],h[j])，k 显然不会增加。因此我们只需考虑确定 h[i]、h[j]不动，且 h[i+1]~h[j-1]都小于 min(h[i],h[j])的情况。此时中间这一段会怎么变呢？会变成同一个高度。

证明：若最终不在同一高度上，将修改后 h 最大的减小 1，差的绝对值之和不增，修改代价减小，总花费会减少(注意考虑边界情况，连续一段 h 都为最大时我们应选则两端的来减小)。

这样，花费仍是一个二次函数：$f(x)=\sum_{k=i+1}^{j-1}(x-h[k])^2+c*(h[i]-x+h[j]-x)$

设 $s1[i]=\sum_{j=1}^{i}h[i]$，$s2[i]=\sum_{j=1}^{i}h[i]^2$，则:

$$f(x)=(j-i-1)x^2-2(s1[j-1]-s1[i]+c)x+s2[j-1]-s2[i]+c*(h[i]+h[j])$$

其中，$x\in[\max\{h[i+1]\sim h[j-1]\},\min(h[i],h[j])]$ 且 $x\in N$。

据此，我们可以得出一个时间复杂度与 h 无关的算法：用 f[i]表示前 i 个数，第 i 个不变的最小花费，通过枚举上一个不变的数 j 来转移，f[j]加上上述二次函数求出的最小值即可得出 f[i]。j 从后往前枚举，方便求出 j+1~i-1 的最大值。注意当 j=0 时上述二次函数需要适当修改。同样的，再从后往前做一次，用 g[i]表示后 i 个数，倒数第 i 个不变的最小花费。$\min\{f[i]+g[n-i+1]\}$ 即为答案。

时间复杂度:$O(n^2)$

空间复杂度:*O(n)*

期望得分：60 分

算法 5：

算法 4 有很多冗余，我们来仔细分析一下转移。考虑决策 j，对于 h[j]>h[i]的转移只有一个，即 h[j]是 i 之前第一个比 h[i]大的，再往前走就不满足 h[j+1]~h[i-1]都小于 min(h[j],h[i])了。而对于 h[j]≤h[i]的转移，注意到 h[j+1]~h[i-1]都小于 h[j]，我们若是维护一个递减序列，j 一定在序列中，且在其转移给 f[i]后会被弹出。

因此，我们维护一个堆栈，在放入 h[i]时，被弹出的元素(h[j]≤h[i])可以转移给 f[i]，第一个未被弹出的元素(h[j]>h[i])也可以转移给 f[i]。二次函数 x 的左边界即 RMQ 问题，可以在 O(nlogn)的时间内求解。

时间复杂度:*O(nlogn)*

空间复杂度:*O(nlogn)*

期望得分：80 分

算法 6：

现在，问题的瓶颈变为了查询 h[j+1]~h[i-1]的最大值。我们可不可以利用维护的堆栈来快速查询呢？答案是肯定的，若 j=i-1，即堆栈栈顶元素，很容易求出转移方程。否则，显然，j 在堆栈中的上一个元素即为 h[j+1]~h[i-1]的最大值。

至此，我们完美的解决了本题。

时间复杂度:*O(n)*

空间复杂度:*O(n)*

期望得分：100 分