浅谈斜率优化

I_LOVE_MATH

2025 年 2 月 21 日

目录 I

目录 II

概论

列出状态转移方程，如果能化简为以下的形式：

$$dp[i] = \min / \max(c[i] + d[j] + C)$$

此时我们就可以利用单调队列优化从做 $O(n^2)$ 到 $O(n)$ 的复杂度。
现在考虑更一般的情况，如果化简为以下形式：

$$dp[i] = \min / \max(a[i] \cdot b[j] + c[i] + d[j] + C)$$

此时单调队列优化就不再适用，就需要使用**斜率优化**。

概论

斜率优化的思想就是通过剔除一些不可能成为最优解的决策点，使剩下的决策点形成一个**凸包**，再使用单调队列、二分、平衡树、CDQ 分治或者李超线段树来快速转移。

代数法

设决策点 j_2 优于 j_1。
则有：（$<$ 对应 max，$>$ 对应 min）

$$a[i] \cdot b[j_1] + c[i] + d[j_1] + C < / > a[i] \cdot b[j_2] + c[i] + d[j_2] + C$$

化简得：

$$a[i] \cdot b[j_1] + d[j_1] < / > a[i] \cdot b[j_2] + d[j_2]$$

代数法

令 $Y(x)=d[x]$，$X(x)=b[x]$，$k=-a[i]$，再让 $X(j_2)>X(j_1)$。

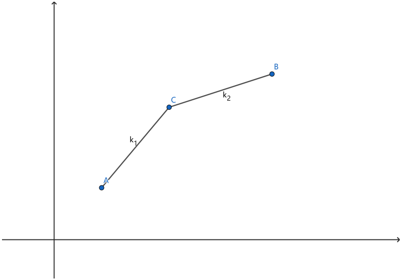
移项得：

$$k</> \frac{Y(j_2)-Y(j_1)}{X(j_2)-X(j_1)}$$

这就是说，将两个决策点在平面直角坐标系中用两点表示出来，如果两点间斜率小于（或大于）k，那么后面的点优于前面的点，下面我们讨论 max 的情况。

代数法

现在我们考虑有如下位置关系的三个点（$k_2 < k_1$）：

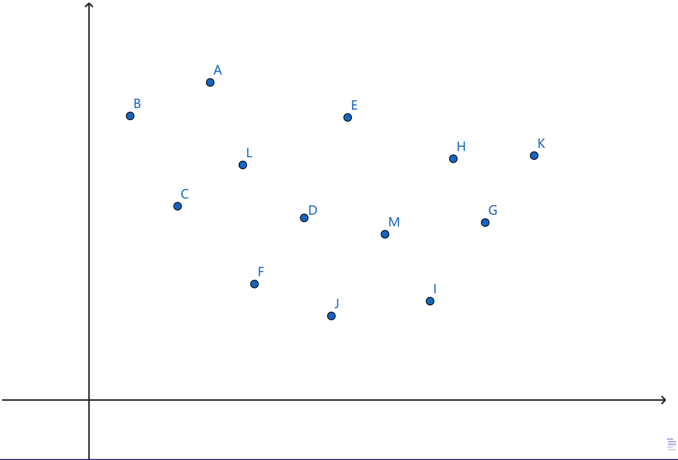


- 当 $k < k_2 < k_1$ 时，则 C 比 A 优，B 比 C 优，B 最优。
- 当 $k_2 \le k \le k_1$ 时，则 A、B 至少有一个比 C 优。
- 当 $k_2 < k_1 < k$ 时，则 A 比 C 优，C 比 B 优，A 最优。

综上，**无论何种情况，C 都不可能最优**，可以删去。

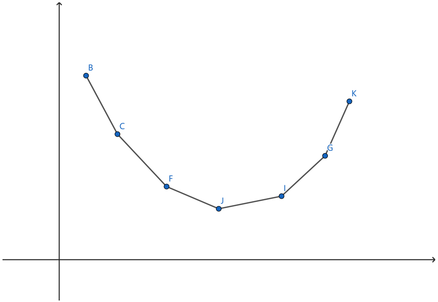
代数法

推广以上的性质，对于所有的决策点：



代数法

删去所有不可能为最优的决策点，发现可能为最优解的点会形成如下的一个**下凸包**：



同理，对于 min 的情况，会形成一个上凸包。

几何法

我们先讨论 max 的情况：

$$dp[i] = \max(a[i] \cdot b[j] + c[i] + d[j] + C)$$

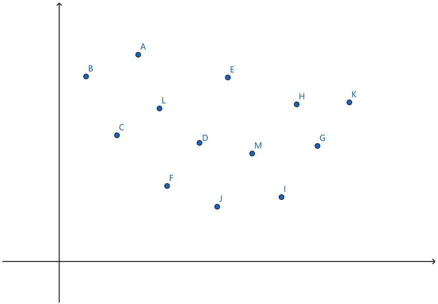
去掉 max，再移项得：

$$-d[j] = a[i] \cdot b[j] + c[i] - dp[i] + C$$

我们可以将 $-d[j]$ 看作纵坐标，$a[i]$ 看作斜率，$b[j]$ 看作横坐标，$c[i] - dp[i] + C$ 看作常数。
那么此时就可以把状态转移方程**看作一个形如 $y = kx + b$ 的直线方程**，其中对于同一个 i 斜率 $k = a[i]$ 是不变的。
要使 $dp[i]$ 最大，就要使方程的截距 b 尽可能小。

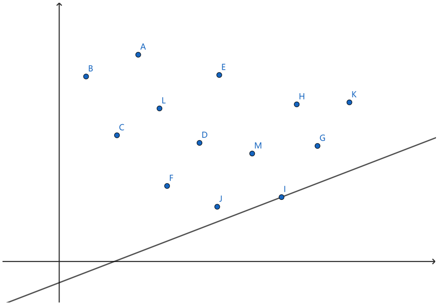
几何法

将所有决策点 $(b[j], -d[j])$ 在平面直角坐标系中表示出来:



几何法

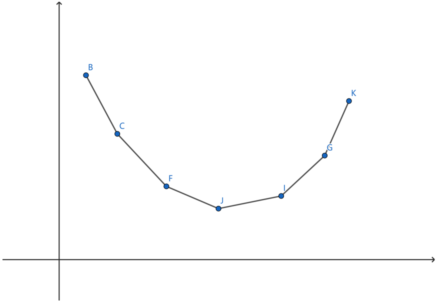
寻找最优决策点的过程就可以看作**平移一条斜率不变的直线**，**自下往上遇到的第一个点必然使截距最小**，如图：



事实上，这个点就是和前面点斜率小于等于当前斜率，和后面点斜率大于等于当前斜率的点。

几何法

现在我们考虑改变直线斜率，发现可能是最优解的决策点当且仅当在下面的**下凸包**上：



不在下面的下凸包上的，无论斜率如何改变，都不可能是从下往上平移最小的点。

几何法

同理，对于 min 的情况，会形成一个上凸包。
个人更喜欢几何法，比较直观，所以下面的例题都会采用几何法论证。

代码实现

对于不同的情况，我们有不同的插入决策点及查询最优解的方式（具体的可以看下面的例题）：

决策点横坐标（$b[j]$）严格增，直线方程斜率（$a[i]$）严格增

- 插入：可以维护一个队列存储决策点编号 j，由于决策点横坐标（$b[j]$）严格增，再根据凸包斜率严格增（或减）的性质，**在插入时只要队列倒数第二元素和队尾的斜率小于（或大于）队尾和要插入的点的斜率，就弹出队尾，直到保证斜率严格增（或减）**。
- 查询：根据最优决策点和前面点斜率小于（或大于）等于当前斜率，和后面点斜率大于（或小于）等于当前斜率的点，再根据直线方程斜率（$a[i]$）严格增，所以最优决策点横坐标严格增，不用保存前面的决策点，因此**只要队首的两个元素斜率小于（或大于）当前斜率，就弹出队首，直到不能弹为止，队首就是最优决策点**。
- 复杂度：$O(n)$。

决策点横坐标（$b[j]$）严格增，直线方程斜率（$a[i]$）严格增

- 插入：由于决策点横坐标（$b[j]$）严格增不变，同上。
- 查询：由于直线方程斜率（$a[i]$）不严格增，所以最优决策点没有单调性了，此时只能根据凸包斜率的单调性二分找到和前面点斜率小于（或大于）等于当前斜率，和后面点斜率大于（或小于）等于当前斜率的点的点。
- 复杂度：$O(n \log n)$。

决策点横坐标（$b[j]$）不严格增

由于横坐标都不严格增，也就是说可能加的点在两点之间，上面的方法就行不通了。
可以考虑利用平衡树动态维护凸包或者采用 CDQ 分治离线等方法，在此先不予陈述。
这里介绍一种思维含量低、代码量小、常数小的优秀算法：**李超线段树**。

概论

要求在平面直角坐标系下维护两个操作：
1. 在平面上加入一条线段。
2. 给定一个数 k，询问与直线 $x = k$ 相交的线段的交点的纵坐标最值。
李超线段树就是能够维护以上两个操作的数据结构。

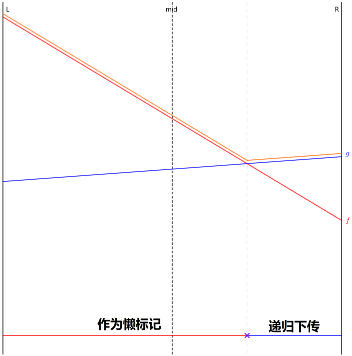
基本概念

首先需要明确：李超树是一种线段树，它的一个节点存储的是一个区间 $[l, r]$ **上值最大的线段的编号的懒标**。
为什么说是懒标记？就是指这条线段并不一定最大（小），而是可能取到最值。
这就需要谈到，李超线段树是一种基于**标记永久**的线段树。
标记永久化就是指修改时一路更改被影响到的节点，询问时则一路累加路上的标记，从而省去标记上传下传的操作。

基本概念

也就是说，我们不保证存储 $[l,r]$ 的节点的编号一定在整个区间上取到最值，但保证对于任意 $x=p$ ，结合所有包含 p 的区间 $[l,r]$ 取最值一定可以取到 p 的最值。
举个例子，假设现在横坐标范围是 $[1,4]$，对于 $x=2$，我们需要保证区间 $[1,4]$、$[1,2]$，$[2,2]$ 中存储的线段编号至少有一个在 $x=2$ 可以取到最值，也就是说取这三个区间的最值就是这个点的最值。
以上就是标记永久化的思想，可以方便修改，也不会使查询复杂度升高。

插入直线



我们先来考虑插入直线，即覆盖整个区间的线，下面默认取最大值，最小值同理。
分类讨论，对于每一个区间 $[l, r]$:

插入直线

- 若区间 $[l, r]$ 没有直线覆盖，直接将标记改为这条直线的编号然后返回即可。
- 若区间 $[l, r]$ 有直线覆盖，考虑此区间的懒标记所表示的直线 f 和插入的直线 g，不妨令原来的懒标记直线 f 在中点 mid 处取值大，即 $f(mid) \geq g(mid)$（代码上，如果插入线段在中点处取值大就和懒标记 swap 即可）。
  - 如果 $f(l) \geq g(l)$ 则在左区间 $[l, mid]$ 上 f 一定不比 g 劣，也就是说 g 对左区间最值不可能有影响，**不需要再递归处理左区间**，右区间同理。
  - 如果 $f(l) < g(l)$ 则在左区间 $[l, mid]$ 上 f 可能比 g 劣，由于 f 不一定就是这段区间的最值，那么我们**递归左区间下传标记** g，右区间同理。

插入直线

我们可以发现，如果这么做，对于每一个 $x = p$，结合所有包含这个点的区间，一定能取到最值。
因为对于每一条插入的线段，如果它在当前区间中点的取值大，那么原来的标记就会被替换，这也就使在这条线段上取到最值的点都被覆盖，又为了保证不影响在原线段取到最值的点，又下传原线段，使在原线段取到最值的点都被原线段覆盖，这样就满足的标记永久化的要求。
时间复杂度：$O(\log n)$。

决策点横坐标（$b[j]$）不严格增

```
void update(int x, int l, int r, int k)
{
    if (!dat[x])
    {
        dat[x] = k;
        return;
    }
    if (p[k].calc(mid) - p[dat[x]].calc(mid) > eps)
    {
        swap(k, dat[x]);
    }
    if (p[k].calc(l) - p[dat[x]].calc(l) > eps)
    {
        update(ls, l, mid, k);
    }
    if (p[k].calc(r) - p[dat[x]].calc(r) > eps)
    {
        update(rs, mid + 1, r, k);
    }
}
```

插入线段

与插入直线不同的是，插入线段有定义域的限制。
与普通线段树一样，我们可以把要插入的区间分成至多 $\log n$ 个在线段树上的区间，然后对每一个被插入区间包含的区间像插入直线一样下传标记即可。
线段树区间操作的本质就是将一个区间分成至多 $\log n$ 个在线段树上的区间—— KevinLikesCoding 大神。

决策点横坐标（$b[j]$）不严格增

```
void update(int x, int l, int r, int ql, int qr, int k)
{
    if (r < ql || l > qr)
    {
        return;
    }
    if (ql <= l && r <= qr)
    {
        if (!dat[x])
        {
            dat[x] = k;
            return;
        }
        if (p[k].calc(mid) - p[dat[x]].calc(mid) > eps)
        {
            swap(k, dat[x]);
        }
        if (p[k].calc(l) - p[dat[x]].calc(l) > eps)
        {
            update(ls, l, mid, ql, qr, k);
        }
        if (p[k].calc(r) - p[dat[x]].calc(r) > eps)
        {
            update(rs, mid + 1, r, ql, qr, k);
        }
        return;
    }
    update(ls, l, mid, ql, qr, k);
    update(rs, mid + 1, r, ql, qr, k);
}
```

插入线段

根据标记永久化的性质，我们只需递归遍历所有包含这个点的区间再取最大值即可。
代码（需要说明的是，我这里建了一个虚拟线段 0，如果 $dat[x]=0$，res 就会被赋值为虚拟线段在 k 处的值，最大值就赋值为负无限，最小值赋值为正无限即可）：

```
double query(int x, int l, int r, int k)
{
    if (r < k || l > k)
    {
        return INT_MIN;
    }
    double res = p[dat[x]].calc(k);
    if (l == r)
    {
        return res;
    }
    return max(res, max(query(ls, l, mid, k), query(rs, mid + 1, r, k)));
}
```

算法应用

事实上，李超线段树并不是维护动态凸包，而是利用本身的特性直接维护答案。
将一些不变量提出：

$$dp[i] = c[i] + C + \min / \max(b[j] \cdot a[i] + d[j])$$

我们将 $\min / \max$ 中的式子看作一个直线方程 $y = kx + b$，其中 $b[j]$ 是斜率，$a[i]$ 是此时横坐标，$d[j]$ 是截距。
发现此时就是要**求若干个直线** $y = b[j] \cdot x + d[j]$ **在** $x = a[i]$ **时的最值**，而这恰好是李超线段树所维护的东西。
因此，每次我们只需将直线 $y = b[j] \cdot x + d[j]$ 插入李超树，再查询所有直线在 $x = a[i]$ 时的最值更新答案即可。
时间复杂度：$O(n \log n)$。

题目描述

机器上有 n 个需要处理的任务，它们构成了一个序列。这些任务被标号为 1 到 n，因此序列的排列为 $1, 2, 3 \cdots n$。这 n 个任务被分成若干批，每批包含相邻的若干任务。从时刻 0 开始，这些任务被分批加工，第 i 个任务单独完成所需的时间是 T_i。在每批任务开始前，机器需要启动时间 s，而完成这批任务所需的时间是各个任务需要时间的总和。

注意，同一批任务将在同一时刻完成。每个任务的费用是它的完成时刻乘以一个费用系数 C_i。

请确定一个分组方案，使得总费用最小。

$T1 : 1 \le n \le 5000, 0 \le S \le 50, 1 \le T_i, C_i \le 100$

$T2 : 1 \le n \le 300000, 0 \le S \le 512, 1 \le T_i, C_i \le 100$

$T3 : 1 \le n \le 300000, 0 \le S \le 512, 0 \le C_i \le 512, -512 \le T_i \le 512$

解题思路

$T1$:
朴素 DP，不需要优化。
我们对 c 和 t 做前缀和，用前缀和替代原来的数组。

- 状态表示：$dp[i]$ 表示完成前 i 个任务的最小费用。
- 初始化：$dp[0]=0$。
- 状态转移：考虑分配成的最后一组，设最后一组为 $j+1\sim i$，首先要在 $dp[j]$ 的基础上加上 $t[i]\cdot(c[i]-c[j])$，但是 s 貌似比较难处理，发现 s 会对后面所有的任务产生影响，因此为方便处理，我们可以提前把它对后面的总贡献 $s\cdot(c[n]-c[j])$ 直接算进当前区间，因此有状态转移方程：

$$dp[i]=\min(dp[j]+t[i]\cdot(c[i]-c[j])+s\cdot(c[n]-c[j]))$$

- 答案：$dp[n]$

时间复杂度：$O(n^2)$。

$T2$:
因为 $1 \le n \le 300000$，$O(n^2)$ 无法通过，发现状态转移方程符合斜率优化的条件，考虑斜率优化。
化简、移项得:

$$dp[j] = (t[i] + s) \cdot c[j] + dp[i] - t[i] \cdot c[j] - s \cdot c[n]$$

要使 $dp[i]$ 更小，那么就要使截距尽可能小，因此决策点会形成一个下凸包。
由于 $T_i, C_i \ge 1$，因此前缀和严格增，所以**横坐标 $c[j]$ 严格增，截距 $t[i] + s$ 严格增**，应用第一种情况即可。

$T3:$
此时 $-512 \le T_i \le 512$，所以截距不严格增，其他不变，在查询时二分找决策点即可。
编码技巧：在处理斜率时为避免误差，**可以写作乘积的形式**。

题目描述

有 n 根柱子依次排列，每根柱子都有一个高度。第 i 根柱子的高度为 h_i。
现在想要建造若干座桥，如果一座桥架在第 i 根柱子和第 j 根柱子之间，那么需要 $(h_i - h_j)^2$的代价。
在造桥前，所有用不到的柱子都会被拆除，因为他们会干扰造桥进程。第 i 根柱子被拆除的代价为 w_i，注意 w_i 不一定非负，因为可能政府希望拆除某些柱子。
现在政府想要知道，通过桥梁把第 1 根柱子和第 n 根柱子连接的最小代价。注意桥梁不能在端点以外的任何地方相交。
$2 \leq n \leq 10^5, 0 \leq h_i, |w_i| \leq 10^6$。

解题思路

先用 w 的前缀和替代原来的序列。
考虑先列出 $O(n^2)$ 的状态转移方程：

- 状态表示：$dp[i]$ 表示用桥梁把前 i 根柱子连接的最小代价。
- 初始化：$dp[1]=0$
- 状态转移：考虑柱子 i 和 j 相接，则有：

$$dp[i]=\min(dp[j]+(h[i]-h[j])^2+w[i-1]-w[j])$$

- 答案：$dp[n]$。

解题思路

现在考虑斜率优化，移项：

$$dp[j] = 2h[i]h[j] - h[j]^2 - w[j] + dp[i] - h[i]^2 + w[i-1]$$

发现由于原来的 $h[i]$ 可以为 0，因此横坐标 $h[j]$ 不严格增，只能用李超线段树优化。
将式子化为：

$$dp[i] = h[i]^2 + w[i-1] + \min(-2h[j]h[i] + dp[j] + h[j]^2 - w[j])$$

问题转化为：每次插入直线 $y = -2h[j] \cdot x + dp[j] + h[j]^2 - w[j]$，求在 $x = h[i]$ 时的最小值。
用李超线段树维护即可，时间复杂度 $O(n \log n)$。

题目描述

Pine 开始了从 S 地到 T 地的征途。
从 S 地到 T 地的路可以划分成 n 段，相邻两段路的分界点设有休息站。
Pine 计划用 m 天到达 T 地。除第 m 天外，每一天晚上 Pine 都必须在休息站过夜。所以，一段路必须在同一天中走完。
Pine 希望每一天走的路长度尽可能相近，所以他希望每一天走的路的长度的方差尽可能小。
帮助 Pine 求出最小方差是多少。
设方差是 v，可以证明，$v \times m^2$ 是一个整数。为了避免精度误差，输出结果时输出 $v \times m^2$。
$1 \leq n \leq 3000$。
保证从 S 到 T 的总路程不超过 3×10^4。
$2 \leq m \leq n+1$，每段路的长度为不超过 3×10^4 的正整数。

解题思路

考虑先化简最后答案的式子（a 表示输入的每段路的长度）：

$$s^2=\frac{(\bar{a}-a_1)^2+(\bar{a}-a_2)^2+\cdots+(\bar{a}-a_m)^2}{m}$$

$$s^2=\frac{m\bar{a}^2-2\bar{a}(a_1+a_2+\cdots+a_m)+{a_1}^2+{a_2}^2+\cdots+{a_m}^2}{m}$$

$$s^2=\frac{m(\frac{\sum_{i=1}^m a_i}{m})^2-2(\frac{\sum_{i=1}^m a_i}{m})(\sum_{i=1}^m a_i)+\sum_{i=1}^m {a_i}^2}{m}$$

$$s^2=-\frac{(\sum_{i=1}^m a_i)^2}{m^2}+\frac{\sum_{i=1}^m {a_i}^2}{m}$$

$$s^2\cdot m^2=-(\sum_{i=1}^m a_i)^2+m\sum_{i=1}^m {a_i}^2$$

解题思路

发现右边第一项是定值，因此只要右边第二项最小即可，也就是问题转化为**求最小平方和**。

不妨用 a 的前缀和替代原来的序列。

于是开始动态规划：

- 状态表示：由于分成几段也会影响答案，因此要开两维表示，$dp[i][j]$ 表示前 i 项分成 j 段的最小平方和。
- 初始化：$dp[0][i]=0$。
- 状态转移：设最后一段为 $k+1\sim i$，则有：

$$dp[i][j]=\min(dp[k][j-1]+(a[i]-a[k])^2)$$

- 答案：把 $dp[n][m]$ 代入上面的式子即可。

时间复杂度 $O(n^3)$，无法通过，考虑斜率优化。

解题思路

将式子化为：

$$dp[k][j-1]+a[k]^2=2a[i]a[k]+dp[i][j]-a[i]^2$$

要使 $dp[i][j]$ 更小，则截距就要更小，所以最优决策点组成一个**下凸包**。
由于 $a[i]$ 严格增，所以**横坐标和斜率都严格增**，应用第一种情况即可。
比较值得注意的是，因为这道题是二维的，我们优化的是第一维，所以可以先枚举第二维，然后跑 m 遍斜率优化。
时间复杂度 $O(m\log n)$。

题目描述

小 Y 最近在一家金券交易所工作。该金券交易所只发行交易两种金券：A 纪念券（以下简称 A 券）和 B 纪念券（以下简称 B 券）。每个持有金券的顾客都有一个自己的帐户。金券的数目可以是一个实数。
每天随着市场的起伏波动，两种金券都有自己当时的价值，即每一单位金券当天可以兑换的人民币数目。我们记录第 K 天中 A 券和 B 券的价值分别为 A_K 和 B_K（元/单位金券）。
为了方便顾客，金券交易所提供了一种非常方便的交易方式：比例交易法。
比例交易法分为两个方面：

题目描述

a) 卖出金券：顾客提供一个 $[0, 100]$ 内的实数 OP 作为卖出比例，其意义为：将 $OP\%$ 的 A 券和 $OP\%$ 的 B 券以当时的价值兑换为人民币；
b) 买入金券：顾客支付 IP 元人民币，交易所将会兑换给用户总价值为 IP 的金券，并且，满足提供给顾客的 A 券和 B 券的比例在第 K 天恰好为 Rate_K；
注意到，同一天内可以进行多次操作。
小 Y 是一个很有经济头脑的员工，通过较长时间的运作和行情测算，他已经知道了未来 N 天内的 A 券和 B 券的价值以及 Rate。他还希望能够计算出来，如果开始时拥有 S 元钱，那么 N 天后最多能够获得多少元钱。
$N \le 10^5$，$0 < A_K \le 10$，$0 < B_K \le 10$，$0 < \mathrm{Rate}_K \le 100$，$\mathrm{MaxProfit} \le 10^9$。

解题思路

考虑 DP。

- 状态表示：$dp[i]$ 表示前 i 天最多能获得多少钱。
- 初始化：$dp[0] = S$。
- 答案：$dp[n]$。

解题思路

- 状态转移：设现在是第 i 天，要使钱最多那么一定不买金券。
  - 不卖金券：$dp[i] = dp[i-1]$。
  - 卖金券：设现在卖出的金券是在第 j 天买入的，如果卖出有利润，那么当时就应该花光所有钱买金券以获得最大收益。设第 i 天最多能获得 x_i 张 A 券，y_i 张 B 券，则有：

$$x_i = dp_i \frac{R_i}{A_i R_i + B_i}$$

$$y_i = dp_i \frac{1}{A_i R_i + B_i}$$

所以：$dp[i] = \max(x_j A_i + y_j B_i)$。

解题思路

综上，状态转移方程为:

$$dp[i] = \max(dp[i-1], \max(x_j A_i + y_j B_i))$$

解题思路

现在来考虑斜率优化，我们把它化成斜率优化能处理的式子（先不考虑不卖，最后把它算上即可）：

$$y_j = -\frac{A_i}{B_i}x_j + \frac{1}{B_i}dp[i]$$

发现 x_j 没有单调性，因此考虑使用李超线段树。
变形得：

$$dp[i] = b[i]\max(x_j\frac{a_i}{b_i} + y_j)$$

问题转化为：每次插入直线 $y = x_j \cdot x + y_j$，求在 $x = \frac{a_i}{b_i}$ 时的最大值。
用李超线段树维护即可，时间复杂度 $O(n \log n)$。

解题思路

值得注意的是，这道题需要查询实数位置的最小值，可以离散化，再将李超线段树的 clac() 函数改写即可。

题目描述

猫国的铁路系统中有 n 个站点，从 $1-n$ 编号。小猫准备从 1 号站点出发，乘坐列车回到猫窝所在的 n 号站点。它查询了能够乘坐的列车，这些列车共 m 班，从 $1-m$ 编号。小猫将在 0 时刻到达 1 号站点。对于 i 号列车，它将在时刻 p_i 从站点 x_i 出发，在时刻 q_i 直达站点 y_i，小猫只能在时刻 p_i 上 i 号列车，也只能在时刻 q_i 下 i 号列车。小猫可以通过多次换乘到达 n 号站点。一次换乘是指对于两班列车，假设分别为 u 号与 v 号列车，若 $y_u = x_v$ 并且 $q_u \le p_v$，那么小猫可以乘坐完 u 号列车后在 y_u 号站点等待 $p_v - q_u$ 个时刻，并在时刻 p_v 乘坐 v 号列车。
小猫只想回到猫窝并且减少途中的麻烦，对此它用烦躁值来衡量。

题目描述

1.小猫在站点等待时将增加烦躁值，对于一次 $t(t \geq 0)$ 个时刻的等待，烦躁值将增加 $At^2 + Bt + C$，其中 A, B, C 是给定的常数。注意：小猫登上第一班列车前，即从 0 时刻起停留在 1 号站点的那些时刻也算作一次等待。
2.若小猫最终在时刻 z 到达 n 号站点，则烦躁值将再增加 z。
形式化地说，若小猫共乘坐了 k 班列车，依次乘坐的列车编号可用序列 $s_1, s_2, \cdots, s_k$ 表示。该方案被称作一条可行的回家路线，当且仅当它满足下列两个条件：
1.$x_{s1} = 1, y_{sk} = n$
2.对于所有 $j(1 \leq j < k)$，满足 $y_{sj} = x_{s_{j+1}}$ 且 $q_{sj} \leq p_{s_{j+1}}$

对于该回家路线，小猫得到的烦躁值将为：

$$q_{s_k}+(A\times p_{s_1}^2+B\times p_{s_1}+C)+\sum_{j=1}^{k-1}(A(p_{s_{j+1}}-q_{s_j})^2+B(p_{s_{j+1}}-q_{s_j})+C)$$

小猫想让自己的烦躁值尽量小，请你帮它求出所有可行的回家路线中，能得到的最小的烦躁值。题目保证至少存在一条可行的回家路线。

对于所有的测试点，保证 $2\le n\le 10^5$，$1\le m\le 10^6$，$0\le A\le 10$，$0\le B,C\le 10^7$，$1\le x_i,y_i\le n$，$x_i\ne y_i$，$0\le p_i<q_i\le 4\times 10^4$。

解题思路

考虑 DP。

首先我们可以先不考虑到达 n 号站点增加的烦躁值，最后把它算上即可，因此下文指的最小烦躁值都不包含这一项。

- 状态表示：这道题要在状态表示上做点文章，如果用 $dp[i]$ 表示到达 i 站点的最小烦躁值，发现难以转移，而再加一维时间，时间和空间复杂度都难以接受，因此我们可以用 $dp[i]$ **表示乘坐 i 号列车到达 y_i 号站点的最小烦躁值**，用一维就可以包含时间和空间。
- 初始化：$dp[0]=0$。
- 状态转移：考虑从 j 列车换乘到 i 列车，则有：

$$dp[i]=\min_{x_i=y_j,p_i>q_j}(dp[j]+A(p_i-q_j)^2+B(p_i-q_j)+C)$$

- 答案：$\min_{y_i=n}(dp[i]+q_i)$。

解题思路

时间复杂度 $O(m^2)$，考虑斜率优化：

$$dp[j] + A{q_j}^2 - Bq_j = 2Ap_iq_j + dp[i] - A{p_i}^2 - Bp_i - C$$

我们先考虑两个限制如何处理：

- 对于 $x_i = y_j$，我们只需开 n **个单调队列**，查询从 x_i 号单调队列中取，插入到第 y_i 号单调队列即可。
- 对于 $p_i > q_j$，我们可以考虑在查询时让单调队列中所有 q_j 都小于等于 p_i，具体地，由于 $0 \le p_i, q_i \le 4 \times 10^4$，因此我们可以**用桶存**代替堆，插入时先不插进单调队列，把它放进键值为 q_i 的桶中，然后在查询的时候统一把键值小于 p_i 的桶中的元素都插入单调队列。

解题思路

发现对于第二个限制的处理方法，要求 p_i 不降，于是我们一开始便可以将 p_i 排序。
由于我们插入是按桶的顺序插入的，所以 q_j 也顺带着不降了，因此斜率和横坐标都不降。
要使 $dp[i]$ 更小，就要使截距更小，因此最优决策点构成一个下凸，应用第一种情况即可。
时间复杂度 $O(m\log m)$。