OI 计算几何 Part 2 二维凸包与单调性优化

Computational Geometry in OI

daklqw

Zhenhai High School

July 31, 2024

Outline

1 凸包

2 凸包上操作

3 凸包与数据结构

凸包

Definition (Convex Hull)

给定 n 个点 $S = \{x_1, x_2, \ldots, x_n\}$，其构成的凸包为

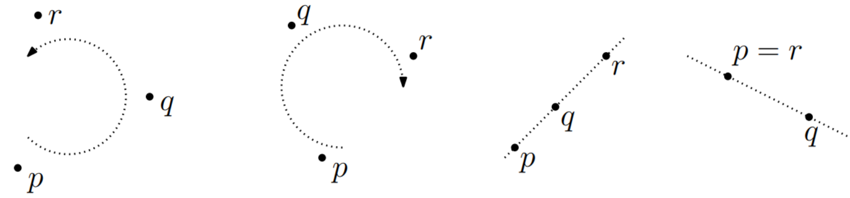
$$\mathrm{Conv}(S) = \{\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n : \alpha_1 + \alpha_2 + \ldots \alpha_n = 1, \alpha_i \geq 0\}.$$

也就是包含点集的最小凸多边形。一般按逆时针顺序存下凸包边界上的顶点。

定向

逆时针为正向，对应着叉积为正。

$\mathrm{orient}(p, q, r) > 0$ $\mathrm{orient}(p, q, r) < 0$ $\mathrm{orient}(p, q, r) = 0$



Graham 扫描法

Algorithm (Graham's Scan)

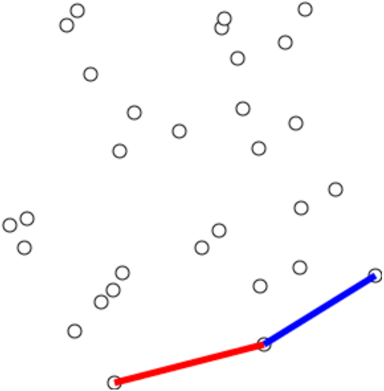
选择一个必在凸包上的点 v，此时存在一条直线，使得剩下点都在这条直线的同一侧，这样好做极角排序。

令 $S=\{v\}$，按照极角序将点一个个加入集合 S，同时实时维护 S 的凸包。

将凸包的顶点按极角序存储，那么每当新点 w 加入 S，w 会导致凸包顶点序列的一段后缀从凸包中移除。

即这个凸包是个<u>单调栈</u>：一旦在序列末尾追加点 w 会导致最后三个点不是逆时针的（即凸的），说明序列末尾的点再也不会出现在后来的凸包里。

Graham 扫描法



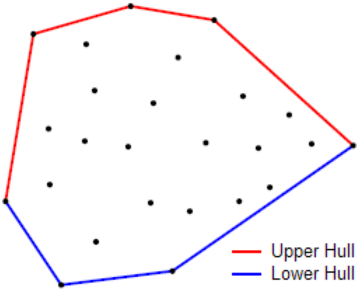
Graham 扫描法

```
// Assumptions:
// p :: std::vector<vec2d> // points
// p[0] is the bottommost and leftmost point
// c :: std::vector<vec2d> // convex hull
std::sort(p.begin() + 1, p.end(), polar_cmp);
for (vec2d w : p) {
  while (c.size() >= 2 && \
          cross(c[c.size() - 2], c.back(), w) <= 0) {
    c.pop_back();
  }
  c.push_back(w);
}
```

Andrew 算法

Algorithm (Andrew)

采用更加直接的排序：以 x 为第一关键字，y 为第二关键字排序。
此时如果使用单调栈可以分别得出凸包的上凸壳和下凸壳。



Codeforces 某一道题，但是我找不到了

Example

给定 n 个点，对于每个点询问，如果删去这个点，剩下的点集凸包会有多少点。

$n \leq 2 \times 10^5$。

Algorithm

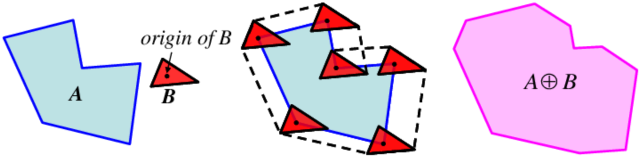
发现删除凸包里的点完全没有影响。删除凸包的顶点后，其相邻顶点仍在凸包上。
因此对凸包的定点进行奇偶标号：建两个凸包，一个是点集删去所有奇数点的，一个是删去所有偶数点的。
然后就可以根据两个凸包的结果分别查询出来答案。

Minkowski 和

Definition (Minkowski Sum)

给定位置向量（即原点到点 P 的向量）的集合 A, B，则 A 和 B 的 Minkowski 和定义为：

$$A + B = \{a + b : a \in A, b \in B\}.$$



(a) (b) (c)

凸包的 Minkowski 和

Example

给定两个凸包 A, B，求 $A+B$。

Proposition

两个凸集的 *Minkowski* 和是凸的。

Proof.

$\forall x, y \in A+B, \lambda \in [0,1]$，假设 $x = a_x + b_x, y = a_y + b_y$，则
$\lambda x + (1-\lambda) y = (\lambda a_x + (1-\lambda) a_y) + (\lambda b_x + (1-\lambda) b_y) \in A+B$。 □

凸包的 Minkowski 和

Proposition

凸包的 *Minkowski* 和为凸包顶点 *Minkowski* 和的凸包。即 $\mathrm{Conv}(A+B)=\mathrm{Conv}(A)+\mathrm{Conv}(B)$。

Proof.

凸包顶点 Minkowski **和为** A_i+B_j **的形式，所以** $\mathrm{Conv}(A+B) \subseteq \mathrm{Conv}(A)+\mathrm{Conv}(B)$。
对于凸包的 Minkowski **和，其内部点为** $\lambda_1 A_1+\lambda_2 A_2+\cdots+\lambda_n A_n + \gamma_1 B_1+\gamma_2 B_2+\cdots+\gamma_m B_m$ **的形式。**

凸包的 Minkowski 和

Proof (Cont.)

构造如下点集 C_i：维护 $i \in [n], j \in [m]$，起初 $i = j = 1$。将 $p = A_i + B_j \in A + B$ 加入点集，并将 λ_i 和 γ_j 共同减去 $w_p = \min(\lambda_i, \gamma_j)$。如果 $\lambda_i = 0$ 则令 i 加一，如果 $\gamma_j = 0$ 则令 j 加一。

因为 $\sum \lambda_i = \sum \gamma_j = 1$，如此构造得到的点集是一组 $A + B$ 点的线性组合，并且 $\sum w_p = 1$。

所以有 $\mathrm{Conv}(A) + \mathrm{Conv}(B) \subseteq \mathrm{Conv}(A + B)$。 $\square$

凸包的 Minkowski 和

Proposition

假设凸包的边集按顺序为 $a_1, a_2, \ldots, a_n$ 以及 $b_1, b_2, \ldots, b_m$，并且 A, B 的顶点 $A_i = A_1 + a_1 + a_2 + \cdots + a_{i-1}, B_i = B_1 + b_1 + b_2 + \ldots b_{i-1}$，则 $A + B = C$ 的顶点满足：

$$\begin{cases} C_1 = A_1 + B_1, \\ C_i = C_1 + c_1 + c_2 + \cdots + c_{i-1}. \end{cases}$$

其中 $c_1, c_2, \ldots, c_{n+m}$ 为 $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m$ 经过极角排序后的结果。

凸包的 Minkowski 和

Proof.

对于凸包 A, B，$A+B$ 的顶点一定是 A_i+B_j 的形式，其中 A_i 和 B_j 都是对应凸包上的顶点。
$A+B$ 上的相邻点一定是 A_i+B_j 和 $A_i+B_{j'}$ 或者 $A_{i'}+B_j$ 的形式。
这两点使用反证法都可以得到。因此 $A+B$ 只可能有 A 和 B 上的边。
同时因为 $A+B$ 的边斜率单调，因此只会有 $n+m$ 条边。 $\square$

凸包的 Minkowski 和

Proposition

凸包的 *Minkowski* 和可以在 $O(n)$ 内完成。

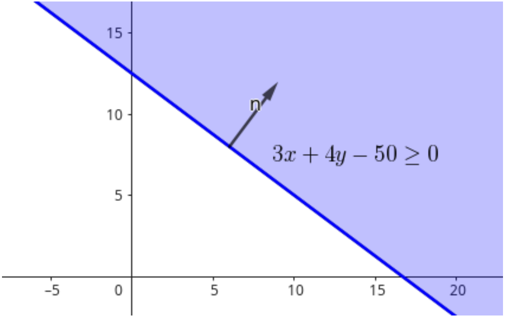
对凸包 A, B 的边进行归并排序，即可得到 $A+B$。

半平面

Definition

直线法向量一侧的区域为半平面。半平面的式子为:

$$Ax + By + C \geq 0.$$



半平面交

Proposition

有限个半平面的交是凸集。

Proof.

一个点 u 在半平面交内当且仅当 $\forall i, \langle u, n_i \rangle + c_i \geq 0$。
此时，若 u, v 都在半平面交内 $\lambda u + (1-\lambda)v$ 也在半平面交内，因为 $\langle \lambda u + (1-\lambda)v, n_i \rangle + c_i \geq 0$。所以半平面交为凸集。 $\square$

半平面交

Algorithm

类似凸包，将半平面按照极角排序并一个个加入，然后使用双端队列维护在半平面交里的直线。同时维护半平面交带来的凸包。
改变交集的半平面减小了凸包的大小。想象将凸包切了一刀，则若干边会被完全舍去，至多两条边会被砍掉半个。即加入这个半平面后，将队列首尾的若干半平面弹出，对应着被完全砍去的边。
实际为了方便实现，一般忽略最新加入的半平面是否真正切到了这个凸包，选择在最后处理这种情况。

半平面交

```cpp
bool check(hp a, hp b, hp c) {
  vec2d p = intersection(b, c);
  return dot(p, a.n) + a.C >= 0;
}
std::sort(half_planes.begin(), half_planes.end(), polar_cmp);
for (hp v : half_planes) {
  while (dq.size() >= 2 && !check(v, dq[-1], dq[-2])) dq.pop_back();
  while (dq.size() >= 2 && !check(v, dq[0], dq[1])) dq.pop_front();
}
while (dq.size() >= 2 && !check(dq[0], dq[-1], dq[-2])) dq.pop_back();
while (dq.size() >= 2 && !check(dq[-1], dq[0], dq[1])) dq.pop_front();
```

注：我没写过双端队列的半平面交，所以建议大家再去网上找找板子，我这个不保证细节正确。

点线对偶

Proposition (Point-Line Duality)

点	线
$P:(A,B)$	$\ell : Ax+By+1=0$

- 两点确定一条直线
- 三点共线
- 凸包

- 两条直线交于一点
- 三线共点
- 半平面交

凸包半平面交对偶

通过点线对偶，可以通过凸包算法求半平面交。即 $Ax + By + 1 \geq 0 \Leftrightarrow (A, B)$。

对所有半平面做对偶，但是不要忘记加入全平面对应的点 $(0, 0)$。

如果存在半平面 $C = 0$，可以通过给所有半平面平移来避免。

然后就可以用凸包求半平面交了。

线性规划

Definition (Linear Programming)

线性规划是一类问题，具有以下形式：

$$
\begin{aligned}
\textbf{Find } & x \in \mathbb{R}^n \\
\textbf{Maximize } & c^T x \\
\textbf{Subject to } & Ax \le b \\
\textbf{And } & x \ge 0.
\end{aligned}
$$

二维线性规划

Example

二维线性规划就是向量 x 只有两维。

对应着半平面 $Ax_1 + Bx_2 \leq C$。因此可以用单纯形法解决。

有解对应着半平面交为一个凸包。

无解对应着半平面交为空。

解无界对应着半平面交有无界区域。

凸包 DP

Example

给定 n 个点和 m 条线段，线段连接这这些点。问这些点能构成多少种凸多边形。$n \le 500$。

凸包 DP

Example

给定 n 个点和 m 条线段，线段连接这这些点。问这些点能构成多少种凸多边形。$n \leq 500$。

Algorithm

枚举凸包起点 S，记 DP 状态为 f_i，表示上个点是 i 的情况下凸壳的方案数。
按照极角序将线段排序，并按极角序枚举边 (u, v)，进行转移：

$$f_v += f_u.$$

由于枚举了极角序，不需要判断凸性。

[JSOI2007] 合金

Example

有三种元素。有 n 个原材料，每个材料的三种元素比例由 a_i, b_i, c_i 给出。原材料可以合成，合成后的材料元素比例是线性组合。
现在给出 m 个要求材料的元素比例 d_i, e_i, f_i，问最少要多少种不同的原材料能够合成出所有要求的材料。
保证 $a_i + b_i + c_i = d_i + e_i + f_i = 1$，$n, m \leq 500$。

Algorithm

可以发现，因为 $a+b+c=1$，所以可以把材料看作平面上的点 (a,b)。
材料的合成即是线性组合，因此原材料能合成的材料为一个凸包。
所以此题的任务是找出一个最小的点集，使得其凸包包含所有点。
找出所有可能在凸包上的边，求一个最小环即可。

旋转卡壳

Example (凸包直径)

给定凸包，在线性时间内求它的直径（内部最远点对）。

旋转卡壳

Example (凸包直径)

给定凸包，在线性时间内求它的直径（内部最远点对）。

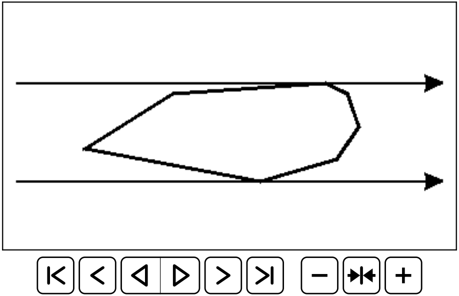
Algorithm

最远的两点肯定在凸包的顶点上。按斜率枚举一对平行线（对应着枚举最远两点连线的斜率），使其刚好卡住整个凸包。

此时平行线切到的点是不断在旋转的。旋转卡壳做的就是，枚举这个旋转的过程，找出所有被切到点对改变的时刻。

此时，我们能得到一段区间斜率的区间，使得切到的点是不变的。在这个区间内求极值是容易的。

旋转卡壳



旋转卡壳

Algorithm

假设逆时针旋转平行线，目前两点为 u, v，则下一个切到的点和 u, v 逆时针方向的边有关系。
比较两条边哪条更先被转到（使用叉积），来选择哪一个点被替换掉。
实际实现其实可以使用类似于双指针的方法，即从小到大枚举一个点，可以直接得到切到另一个点的区间。

最小矩形覆盖

Example

给定点集，求一个矩形，使得点集被矩形完全覆盖，且矩形面积最小。

最小矩形覆盖

Example

给定点集，求一个矩形，使得点集被矩形完全覆盖，且矩形面积最小。

Algorithm

因为矩形的边是垂直的，因此直接枚举矩形一条边的斜率，可以直接枚举到这条平行线和垂线切到的四个点。

[SCOI2007] 最大土地面积

Example

给定点集，从其中选四个点使其构成的多边形面积最大化。
点个数 $n \leq 2000$。

[SCOI2007] 最大土地面积

Algorithm

最优的点一定都在凸包上。可以发现，一旦枚举了对角线，剩下两个点便是与对角线法向量内积最大与最小的点。而这个点，是随着对角线斜率单调变化的。
所以枚举其中一个点，用类似旋转卡壳的方法，枚举对角线另一个点，用平行于这个对角线的直线去卡凸包得到剩下两个点。

Outline

1 凸包

2 凸包上操作

3 凸包与数据结构

凸包上三分/二分

对于凸函数，寻找其极值可以使用三分。如果能方便求得其导数，便可以用二分。

在凸包上由于其斜率容易计算，因此很好二分。

凸包上三分/二分

对于凸函数，寻找其极值可以使用三分。如果能方便求得其导数，便可以用二分。

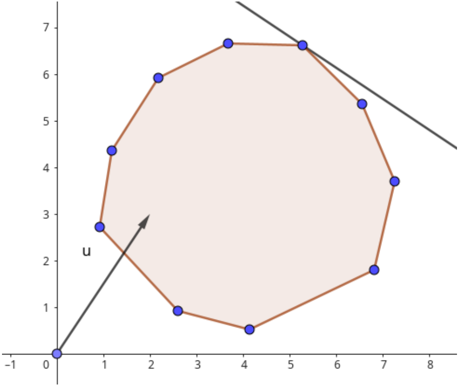
在凸包上由于其斜率容易计算，因此很好二分。

Example

给定向量 x 和凸包 C，求 C 上的点 p，使得内积 $\langle x, p\rangle$ 被最大化。

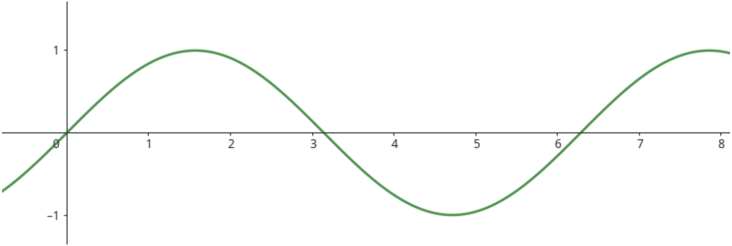
相当于找一个方向上的最远点。

凸包上三分/二分



凸包上三分/二分

凸包的形状类似圆，想象 $\sin$ 函数：如果随意截取一个周期，那么这段区间内函数很可能不是凸的，因为其导数在一个周期内产生了两次符号变化：



凸包上三分/二分

为了解决这个问题，截取半个周期，分两次二分可以保证函数是上凸/下凸的。

因此，凸包二分可以对着凸壳做。使用 Andrew 算法直接获取凸包的上下凸壳，并根据给定向量的 y 方向分量大小决定选择二分的凸壳。

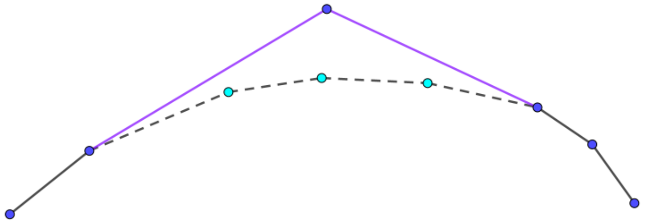
上下凸壳加点

Example

维护一个上凸壳，支持动态加点，支持查询使内积最大化的点。

Algorithm

使用平衡树维护上凸壳，每次加点时使用二分找到插入位置，然后检查是否需要删除其左右的点。查询时使用二分找到使内积最大化的点。



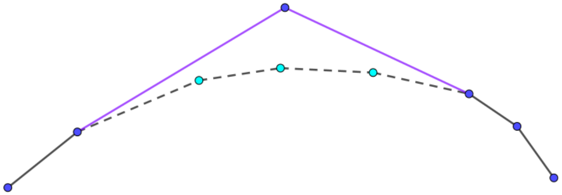
凸包切线

Example

给定凸包，问凸包外一个点到凸包的切线。

Algorithm

等价于将这个点加入点集，产生的新边。因此在平衡树上二分切线的位置即可。



凸包合并

Example

维护一堆点的集合 S_i，支持以下操作：

- 合并两个集合成一个新的集合；
- 对一个集合 S 查询，给定向量 x，最大化 $\langle x, p\rangle$ 的点 $p \in S$。

凸包合并

Example

维护一堆点的集合 S_i，支持以下操作：

- 合并两个集合成一个新的集合；
- 对一个集合 S 查询，给定向量 x，最大化 $\langle x, p\rangle$ 的点 $p \in S$。

Algorithm

使用平衡树维护集合的上下凸壳，此时问题在合并。直接使用启发式合并的复杂度是 $O(\log^2 n)$。

凸包合并

Algorithm

使用空间更多的做法：使用线段树合并，并且在边界处判断是否有点要删除。因为删点总次数是 $O(n)$，所以这么做仍然是 $O(\log n)$ 的。

使用平衡树启发式合并：可以做到线性空间。例如 *Splay* 这种同样可以在合并时判断删点条件：当插入一个点，被旋到根时，可以根据区间信息（每个点维护区间最前面和最后面两个点）来判断删点。

SOJ418.【SHR 1】忌蒜挤核

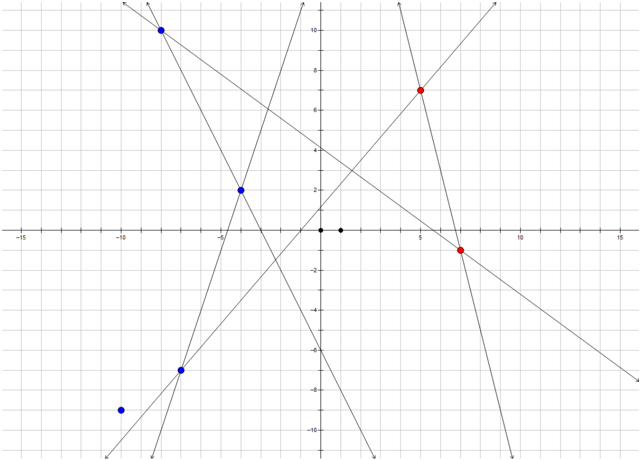
Example

维护一个点集，支持往里面加点。支持查询满足以下条件本质不同直线的数量：

- 经过点集里至少两个点。
- 将点分成两侧，一侧是 $x<0$ 的，另一侧是 $x>0$ 的。

点数 $n \leq 3 \times 10^5$。

SOJ418.【SHR 1】忌蒜挤核



SOJ418.【SHR 1】忌蒜挤核

Algorithm

需要建立凸壳。容易发现只有两条内公切线以及公切线中间的凸壳边是可用的。
因为凸壳是单调增长的，因此可以对凸壳中改变里的位置重新求内公切线。但是实现较阴间。

SOJ418.【SHR 1】忌蒜挤核

实际上做射影变换 $(x, y) \to (1/x, y/x)$ 后，发现答案就变成了所有点构成的凸壳的边。

原因是，题目本质上等价于，求点 i, j，使得对于所有 k，行列式 $\begin{vmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{vmatrix}$ 的符号与 x_k 相同。

根据行列式性质，相当于判断 $\begin{vmatrix} x_j/x_j & y_j/x_j & 1 \\ x_i/x_i & y_i/x_i & 1 \\ x_k/x_k & y_k/x_k & 1 \end{vmatrix}$ 的符号。

Outline

1 凸包

2 凸包上操作

3 凸包与数据结构

Example

要求维护一个关于二维点的栈，支持：

- 往序列尾加元素

Algorithm (势能线段树)

凸壳可以作为区间信息被维护。其合并即归并。但是其合并复杂度为线性。

实际上是一种基于势能的二进制分组。由于合并操作较慢，因此要求达到一定势能再合并。

李超线段树

Example

维护平面上线段的集合，支持：

- 动态插入线段。
- 给定横座标 x_0，求直线 $x = x_0$ 与所有线段交点的最高点。

Algorithm

使用线段树，每个节点 $[l, r]$ 上维护线段的集合 S，并保证最高的线段只有一条。同样，每条线段分布在线段树的若干节点上，用类似于 *Lazy Tag* 的方法维护。

李超线段树

Algorithm

插入线段是一个区间修改，当在区间 $[l, r]$ 插入线段 x 的时候，和原有的线段 y 相比会有三种情况：

- x 完全比 y 高，则把 y 换成 x。
- y 完全比 x 高，则不用管。
- x 和 y 有交点：
  - 如果 x 在上方的部分比 y 多，则交换 x 和 y。
  - 否则递归处理交点所在的那半边。

这样插入复杂度为 $O(\log^2 n)$。查询需要查从 $[x, x]$ 到根的所有节点所以复杂度为 $O(\log n)$。

斜率优化

在一类 DP 问题中，DP 的转移实际上是在点集查询内积最大的点。此时问题就可以转化为维护一个凸壳。

并且有时候用来查询的向量斜率也是单调的，因此可以单调地选取凸壳上的点。

斜率优化相关题目详解

见 单调性优化课件 （中的 P21 到 P58，或第 50 帧到第 173 帧）。

最大叉积问题

Example (最大叉积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ay - bx$。

$1 \leq |P|, Q \leq 10^6$。

最大叉积问题

Example (最大叉积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ay - bx$。

$1 \leq |P|, Q \leq 10^6$。

由叉积的几何意义，两个向量叉积是平行四边形的面积，也是 $2S_{\Delta OXY}$。

最大叉积问题

Example (最大叉积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ay - bx$。

$1 \leq |P|, Q \leq 10^6$。

由叉积的几何意义，两个向量叉积是平行四边形的面积，也是 $2S_{\Delta OXY}$。

那么根据公式 $S = \frac{1}{2}ah_a$，其中只有 h_a 是可变的，我们要最大化它。

最大叉积问题

Example (最大叉积问题)

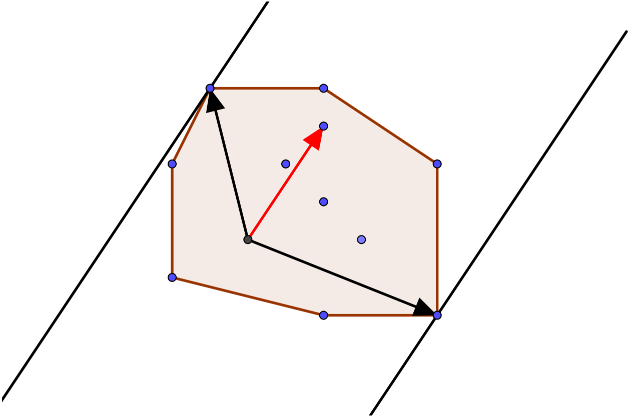
二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ay - bx$。

$1 \le |P|, Q \le 10^6$。

由叉积的几何意义，两个向量叉积是平行四边形的面积，也是 $2S_{\Delta OXY}$。

那么根据公式 $S = \frac{1}{2}ah_a$，其中只有 h_a 是可变的，我们要最大化它。
那么相当于拿一条直线取截这些点，因此最大的答案一定在凸包上。

最大叉积问题



最大点积问题

Example (最大点积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ax + by$。

$1 \leq |P|, Q \leq 10^6$。

最大点积问题

Example (最大点积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ax + by$。

$1 \leq |P|, Q \leq 10^6$。

和叉积一毛一样，只是令新的 $a' = -b, b' = a$。所以答案一定在凸包上。

最大点积问题

Example (最大点积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ax + by$。

$1 \leq |P|, Q \leq 10^6$。

和叉积一毛一样，只是令新的 $a' = -b, b' = a$。所以答案一定在凸包上。

所以回到单调性，不妨令 $a > 0, b > 0$，因为其他的情况形式都是类似的。

最大点积问题

Example (最大点积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ax + by$。

$1 \le |P|, Q \le 10^6$。

和叉积一毛一样，只是令新的 $a' = -b, b' = a$。所以答案一定在凸包上。

所以回到单调性，不妨令 $a > 0, b > 0$，因为其他的情况形式都是类似的。

我们发现，去掉不优点的方法已经没有用了！因为剩下的序列，并不存在一个 $\preceq$ 使得单调。

最大点积问题

Example (最大点积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a, b，求最大的 $ax + by$。

$1 \le |P|, Q \le 10^6$。

和叉积一毛一样，只是令新的 $a' = -b, b' = a$。所以答案一定在凸包上。

所以回到单调性，不妨令 $a > 0, b > 0$，因为其他的情况形式都是类似的。

我们发现，去掉不优点的方法已经没有用了！因为剩下的序列，并不存在一个 $\preceq$ 使得单调。

而由我们上面的经验，直接建出的凸壳，就是剩下的序列的子序列。

最大点积问题

Example (最大点积问题)

二维平面，给出一组点集 P，多组询问，每次询问给出两个数 a,b，求最大的 $ax+by$。

$1 \le |P|, Q \le 10^6$。

和叉积一毛一样，只是令新的 $a' = -b, b' = a$。所以答案一定在凸包上。

所以回到单调性，不妨令 $a > 0, b > 0$，因为其他的情况形式都是类似的。

我们发现，去掉不优点的方法已经没有用了！因为剩下的序列，并不存在一个 $\preceq$ 使得单调。

而由我们上面的经验，直接建出的凸壳，就是剩下的序列的子序列。

也就是，凸包的限制更加严格，因为凸包很多时候要求代价函数 $f(x,y)$ 是一个线性函数。

凸壳的性质

当我们限制了点集象限的时候（多半是 $y \geq 0$）的部分，得到的凸壳是有很多优美的性质的。然而，我们通过去掉不优元素得到的单调序列，一般不具有这些性质。

凸壳的性质

当我们限制了点集象限的时候（多半是 $y \geq 0$）的部分，得到的凸壳是有很多优美的性质的。然而，我们通过去掉不优元素得到的单调序列，一般不具有这些性质。

Property (函数的凸性)

对于上凸壳点序列 S 和第一或二象限点 p，令 $f(x) = x \cdot p$，则对于所有 $1 < i < |S|$，令 $a = P_{i-1}, b = P_i, c = P_{i+1}$，则有

$$\frac{f(b) - f(a)}{b_x - a_x} \geq \frac{f(c) - f(b)}{c_x - b_x}$$

函数是凸的，我们可以在函数上三分。（而一般的单调序列没这个性质）

凸壳的性质

函数的凸性.

$$\frac{b_y - a_y}{b_x - a_x} \geq \frac{c_y - b_y}{c_x - b_x}$$

$$\text{let } v = \frac{p_x}{p_y}$$

$$p_y \frac{(b_x - a_x)v + b_y - a_y}{b_x - a_x} \geq p_y \frac{(c_x - b_x)v + c_y - b_y}{c_x - b_x}$$

$$\frac{b_x p_x + b_y p_y - a_x p_x - a_y p_y}{b_x - a_x} \geq \frac{c_x p_x + c_y p_y - b_x p_x - b_y p_y}{c_x - b_x}$$

□ □

凸壳上二分

```
int getMax(vec * A, int n, vec p) {
    int ans = func(A[n], p);
    int l = 1, r = n - 1;
    while (l <= r) {
        int mid = l + r >> 1;
        int v1 = func(A[mid], p), v2 = func(A[mid + 1], p);
        ans = std::max(ans, v1);
        ans = std::max(ans, v2);
        if (v1 > v2) r = mid - 1; else l = mid + 1;
    }
    return ans;
}
```

凸壳的性质

Property (单调的询问点决策点的单调性)

对于上凸壳点序列 S 和单调不降序列 v_i。
令 $f(i,v) = S_i \cdot (v_i, 1)$，$g(v)$ 为 $f(i,v)$ 取到最大值时最大的 i。
那么有 $g(v_i)$ 形成的序列单调不降。

这个性质在后面的一些题目中可以用来优化，把二分变成暴力扫。

凸壳的性质

Proof (单调的询问点决策点的单调性).

不妨往凸壳最后面塞一个辅助点 $((S_n)_x + 1, -\infty)$。

对一个 v_i 从左往右扫，直到 $f(i+1, v) < f(i, v)$。

令 $a = S_j, b = S_{j+1}$，由于代入 v_i 后斜率是单调的，于是会在第一个 j 使得 $v + \frac{b_y - a_y}{b_x - a_x} < 0$

处停下。因为所有询问的 v 单调不降，所以后面要单调不增。

因为凸壳上从左到右斜率单调不增，所以得到最大化权值时得到的决策点是单调不降的。□ □

【SKR #3】要塞之山

Example (要塞之山)

你需要维护一个栈，栈里储存点集。同时有如下几种询问：

1. 在栈顶放入一个点。
2. 弹出栈顶的点。
3. 给出 a, b，求 $(ax + by)$ 的最大值。

保证所有点坐标非负且 $< 2^{31}$。保证总操作数 $\leq 5 \times 10^5$。

【SKR #3】要塞之山

Example (要塞之山)

你需要维护一个栈，栈里储存点集。同时有如下几种询问：

1. 在栈顶放入一个点。
2. 弹出栈顶的点。
3. 给出 a, b，求 $(ax + by)$ 的最大值。

保证所有点坐标非负且 $< 2^{31}$。保证总操作数 $\leq 5 \times 10^5$。

我们只需要实时维护栈里面内容的凸包即可。不考虑删除，像单调栈一样，我们考虑在插入的同时维护凸包。只需要不断弹出破坏凸包性质的点就好了。

【SKR #3】要塞之山

Example (要塞之山)

你需要维护一个栈，栈里储存点集。同时有如下几种询问：

1. 在栈顶放入一个点。
2. 弹出栈顶的点。
3. 给出 a, b，求 $(ax + by)$ 的最大值。

保证所有点坐标非负且 $< 2^{31}$。保证总操作数 $\leq 5 \times 10^5$。

我们只需要实时维护栈里面内容的凸包即可。不考虑删除，像单调栈一样，我们考虑在插入的同时维护凸包。只需要不断弹出破坏凸包性质的点就好了。

考虑删除，一次插入带来的操作过多，删除后再次操作同样会消耗那么多的时间，所以需要改进一下。

下面将讲述如何使用可回退栈来解决。

【SKR #3】要塞之山

考虑一个点的插入会带来从栈顶往栈底一段连续段的删除，并且带来 $O(1)$ 的修改。

【SKR #3】要塞之山

考虑一个点的插入会带来从栈顶往栈底一段连续段的删除，并且带来 $O(1)$ 的修改。

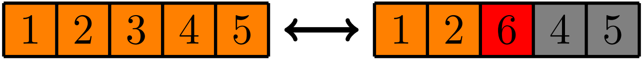
那么每次修改之前记下栈的 top，然后维护凸包的时候，并不真正地删除栈顶的点，而是改变 top 的指针。同时插入的时候，我们记录下插入的点替换了什么。

【SKR #3】要塞之山

考虑一个点的插入会带来从栈顶往栈底一段连续段的删除，并且带来 $O(1)$ 的修改。

那么每次修改之前记下栈的 top，然后维护凸包的时候，并不真正地删除栈顶的点，而是改变 top 的指针。同时插入的时候，我们记录下插入的点替换了什么。

这个时候弹出栈顶的点可以视为撤销插入操作，只需要把被替换的点换回来，并且改变 top 指针即可。



【SKR #3】要塞之山

下面列出栈操作的实现。

```
const int N = 5e5 + 10;
int tops[N], top, cnt;
vec st[N], repl[N];
void push(vec x) {
    int at = findLeft(x);
    // 找到要删除的凸壳段的左端点
    ++cnt;
    tops[cnt] = top;
    repl[cnt] = st[at];
    st[at] = x, top = at;
}
void pop() {
    st[top] = repl[cnt];
    top = tops[cnt];
    --cnt;
}
```

【ZJOI2007】仓库建设

Example (仓库建设)

有 n 个白点 $1\ldots n$，你要选择一些点染成黑色（n 一定要是黑色）。记 R_i 为最小且 $\geq i$ 的黑点。最小化

$$\sum_{i=1}^{n}\left([i\text{是黑点}]W_i+(V_{R_i}-V_i)P_i\right)$$

其中，W_i,P_i,V_i 已给定，且 $V_{i-1}<V_i$。$n\leq 10^6$。

【ZJOI2007】仓库建设

Example (仓库建设)

有 n 个白点 $1 \dots n$，你要选择一些点染成黑色（n 一定要是黑色）。记 R_i 为最小且 $\geq i$ 的黑点。最小化

$$\sum_{i=1}^{n} \left([i\text{是黑点}]W_i + (V_{R_i} - V_i)P_i\right)$$

其中，W_i, P_i, V_i 已给定，且 $V_{i-1} < V_i$。$n \leq 10^6$。

很容易发现，这道题符合分段转移的模型。

【ZJOI2007】仓库建设

Example (仓库建设)

有 n 个白点 $1 \ldots n$，你要选择一些点染成黑色（n 一定要是黑色）。
记 R_i 为最小且 $\geq i$ 的黑点。最小化

$$\sum_{i=1}^{n}\left([i\text{是黑点}]W_i + (V_{R_i} - V_i)P_i\right)$$

其中，W_i, P_i, V_i 已给定，且 $V_{i-1} < V_i$。$n \leq 10^6$。

很容易发现，这道题符合分段转移的模型。
记 P_i 的前缀和为 S，P_iV_i 的前缀和为 T。
记 f_i 为在前 i 个点中，i 是黑点的最小权值和。则转移方程为：

$$f_i = \min_{j<i}\{f_j + W_i + V_i(S_i - S_j) - (T_i - T_j)\}$$

【ZJOI2007】仓库建设

【ZJOI2007】仓库建设

整理式子得

$$f_i = \min_{j<i}\{f_j - V_iS_j + T_j\} + V_iS_i + W_i - T_i$$

【ZJOI2007】仓库建设

整理式子得

$$f_i = \min_{j<i}\{f_j - V_i S_j + T_j\} + V_i S_i + W_i - T_i$$

然后就化成了 $(-S_j, f_j + T_j)$ 与 $(V_i, 1)$ 的最小点积问题。
也就是 $(S_j, -f_j - T_j)$ 与 $(V_i, 1)$ 的最大点积问题。

【ZJOI2007】仓库建设

整理式子得

$$f_i = \min_{j<i}\{f_j - V_iS_j + T_j\} + V_iS_i + W_i - T_i$$

然后就化成了 $(-S_j, f_j + T_j)$ 与 $(V_i, 1)$ 的最小点积问题。
也就是 $(S_j, -f_j - T_j)$ 与 $(V_i, 1)$ 的最大点积问题。
所以有效的转移点一定在 $(S_j, -f_j - T_j)$ 构成的上凸壳上。

【ZJOI2007】仓库建设

整理式子得

$$f_i = \min_{j<i}\{f_j - V_iS_j + T_j\} + V_iS_i + W_i - T_i$$

然后就化成了 $(-S_j, f_j + T_j)$ 与 $(V_i, 1)$ 的最小点积问题。
也就是 $(S_j, -f_j - T_j)$ 与 $(V_i, 1)$ 的最大点积问题。
所以有效的转移点一定在 $(S_j, -f_j - T_j)$ 构成的上凸壳上。
现在我们移动右端点，那么就是不断的插入点。注意到我们插入的点的横坐标 S_i 是单调递增的，所以我们可以只在序列最右边插入。

【ZJOI2007】仓库建设

整理式子得

$$f_i = \min_{j<i}\{f_j - V_iS_j + T_j\} + V_iS_i + W_i - T_i$$

然后就化成了 $(-S_j, f_j + T_j)$ 与 $(V_i, 1)$ 的最小点积问题。
也就是 $(S_j, -f_j - T_j)$ 与 $(V_i, 1)$ 的最大点积问题。
所以有效的转移点一定在 $(S_j, -f_j - T_j)$ 构成的上凸壳上。

现在我们移动右端点，那么就是不断的插入点。注意到我们插入的点的横坐标 S_i 是单调递增的，所以我们可以只在序列最右边插入。

再看我们询问的点 V_i 也是单调递增的，所以我们的转移点是从左到右单调增的。

【ZJOI2007】仓库建设

整理式子得

$$f_i = \min_{j<i}\{f_j - V_i S_j + T_j\} + V_i S_i + W_i - T_i$$

然后就化成了 $(-S_j, f_j + T_j)$ 与 $(V_i, 1)$ 的最小点积问题。
也就是 $(S_j, -f_j - T_j)$ 与 $(V_i, 1)$ 的最大点积问题。
所以有效的转移点一定在 $(S_j, -f_j - T_j)$ 构成的上凸壳上。

现在我们移动右端点，那么就是不断的插入点。注意到我们插入的点的横坐标 S_i 是单调递增的，所以我们可以只在序列最右边插入。

再看我们询问的点 V_i 也是单调递增的，所以我们的转移点是从左到右单调增的。

为了快速地维护答案，我们需要在最左边弹出不再有用的节点。此时需要一个单调队列维护。

【NOI2007】货币兑换

Example (货币兑换)

你有 100\$，你可以保持两种券 A 和 B。下面你要进行 n 天操作。

每一天券的价格分别为 a_i 和 b_i，同时会给你一个比例 r_i。

你可以指定一个比例 $x \in [0,1]$，分别卖出 x 比例的 A ，和 x 比例的 B。你也可以用 $r_i:1$ 的比例买这两种券。

每天两种操作随便使用，要求在最后手上的 \$ 最多。$n \le 10^6$。

【NOI2007】货币兑换

Example (货币兑换)

你有 100$，你可以保持两种券 A 和 B。下面你要进行 n 天操作。

每一天券的价格分别为 a_i 和 b_i，同时会给你一个比例 r_i。

你可以指定一个比例 $x \in [0,1]$，分别卖出 x 比例的 A ，和 x 比例的 B。你也可以用 $r_i : 1$ 的比例买这两种券。

每天两种操作随便使用，要求在最后手上的 $ 最多。$n \le 10^6$。

首先**很容易**证明每次操作要么买完钱，要么卖完。

【NOI2007】货币兑换

Example (货币兑换)

你有 100$，你可以保持两种券 A 和 B。下面你要进行 n 天操作。

每一天券的价格分别为 a_i 和 b_i，同时会给你一个比例 r_i。

你可以指定一个比例 $x \in [0,1]$，分别卖出 x 比例的 A ，和 x 比例的 B。你也可以用 $r_i : 1$ 的比例买这两种券。

每天两种操作随便使用，要求在最后手上的 $ 最多。$n \le 10^6$。

首先**很容易**证明每次操作要么买完钱，要么卖完。

然后身上要么 $ 数为 0，要么券数为 0。

如果使用扫描线，就知道了对于每个右端点，对应的能获得的最大 $ 数，以及用这些 $ 能换多少 A，多少 B。

【NOI2007】货币兑换

下面只用考虑券到 \$ 的转移。对于一个右端点，相当于要做一个最大点积问题。

【NOI2007】货币兑换

下面只用考虑券到 $ 的转移。对于一个右端点，相当于要做一个最大点积问题。

因为插入的点横坐标不单调，所以需要在中间插入。

【NOI2007】货币兑换

下面只用考虑券到 $ 的转移。对于一个右端点，相当于要做一个最大点积问题。

因为插入的点横坐标不单调，所以需要在中间插入。

我们用平衡树维护这个凸壳。如果一个点插入后不可能更优，就放弃这个点。否则插入后尝试去掉相邻的破坏凸壳的点。

【NOI2007】货币兑换

下面只用考虑券到 \$ 的转移。对于一个右端点，相当于要做一个最大点积问题。

因为插入的点横坐标不单调，所以需要在中间插入。

我们用平衡树维护这个凸壳。如果一个点插入后不可能更优，就放弃这个点。否则插入后尝试去掉相邻的破坏凸壳的点。

然后查询在平衡树上二分即可，复杂度 $O(n\log n)$。

【NOI2007】货币兑换

下面只用考虑券到 $ 的转移。对于一个右端点，相当于要做一个最大点积问题。

因为插入的点横坐标不单调，所以需要在中间插入。

我们用平衡树维护这个凸壳。如果一个点插入后不可能更优，就放弃这个点。否则插入后尝试去掉相邻的破坏凸壳的点。

然后查询在平衡树上二分即可，复杂度 $O(n\log n)$。

当然平衡树太难写，我们考虑分治。这样可以用一个区间形成的凸包，这样就规避了中间删除。

因为凸包需要排序，简单的实现需要 $O(n\log^2 n)$。

【NOI2007】货币兑换

下面只用考虑券到 \$ 的转移。对于一个右端点，相当于要做一个最大点积问题。

因为插入的点横坐标不单调，所以需要在中间插入。

我们用平衡树维护这个凸壳。如果一个点插入后不可能更优，就放弃这个点。否则插入后尝试去掉相邻的破坏凸壳的点。

然后查询在平衡树上二分即可，复杂度 $O(n\log n)$。

当然平衡树太难写，我们考虑分治。这样可以用一个区间形成的凸包，这样就规避了中间删除。

因为凸包需要排序，简单的实现需要 $O(n\log^2 n)$。

于是考虑用归并排序代替直接排序。这样我们建凸包就可以线性。

【NOI2007】货币兑换

下面只用考虑券到 $ 的转移。对于一个右端点，相当于要做一个最大点积问题。

因为插入的点横坐标不单调，所以需要在中间插入。

我们用平衡树维护这个凸壳。如果一个点插入后不可能更优，就放弃这个点。否则插入后尝试去掉相邻的破坏凸壳的点。

然后查询在平衡树上二分即可，复杂度 $O(n\log n)$。

当然平衡树太难写，我们考虑分治。这样可以用一个区间形成的凸包，这样就规避了中间删除。

因为凸包需要排序，简单的实现需要 $O(n\log^2 n)$。

于是考虑用归并排序代替直接排序。这样我们建凸包就可以线性。

但是发现询问还是要二分。于是我们对询问也进行归并排序，这样也变成了线性。复杂度变成了 $O(n\log n)$。

【NOI2014】购票

Example (购票)

给你一棵带边权的根为 1 的树，每个点到根距离为 D_i，对于每个点都有三个参数 L_i, P_i, Q_i。

每个点 i 可以往距离它不超过 L_i 的祖先 j 上跳，花费为

$$(D_i - D_j)P_i + Q_i$$

你需要对每个点计算到 1 的花费最少的路径。

$n \leq 2 \times 10^5$。所有数都非负。

【NOI2014】购票

Example (购票)

给你一棵带边权的根为 1 的树，每个点到根距离为 D_i，对于每个点都有三个参数 L_i, P_i, Q_i。

每个点 i 可以往距离它不超过 L_i 的祖先 j 上跳，花费为

$$(D_i - D_j)P_i + Q_i$$

你需要对每个点计算到 1 的花费最少的路径。

$n \leq 2 \times 10^5$。所有数都非负。

发现和上一道题类似，都是类似下标小贡献给下标大的点积最大化问题。

【NOI2014】购票

Example (购票)

给你一棵带边权的根为 1 的树，每个点到根距离为 D_i，对于每个点都有三个参数 L_i, P_i, Q_i。

每个点 i 可以往距离它不超过 L_i 的祖先 j 上跳，花费为

$$(D_i - D_j)P_i + Q_i$$

你需要对每个点计算到 1 的花费最少的路径。

$n \le 2 \times 10^5$。所有数都非负。

发现和上一道题类似，都是类似下标小贡献给下标大的点积最大化问题。

类似的，我们使用点分治。每次分治时，我们先分治算出到根链的 DP 值，再贡献给分治中心的子树，再递归子树。

【NOI2014】购票

这样的话，直接实现就是 $O(n\log^2 n)$ 的，但是需要写平衡树，代码复杂度上天。

【NOI2014】购票

　　这样的话，直接实现就是 $O(n\log^2 n)$ 的，但是需要写平衡树，代码复杂度上天。

　　同样的，我们来分析这道题插入点和查询点的单调性。

　　首先插入的下标是单调的，而我们查询的是后缀凸壳，这样我们可以使用扫描线规避在中间插入。

【NOI2014】购票

这样的话，直接实现就是 $O(n\log^2 n)$ 的，但是需要写平衡树，代码复杂度上天。

同样的，我们来分析这道题插入点和查询点的单调性。

首先插入的下标是单调的，而我们查询的是后缀凸壳，这样我们可以使用扫描线规避在中间插入。

所以就倒着插入点维护凸壳，同时在凸壳上二分来查询，复杂度 $O(n\log^2 n)$。

【集训队作业 2018】line

Example (line)

有 n 个人要调题。你需要把他们分成若干连续段，这些段从前往后来调题。

每个人 i 有一个权值 C_i，一个段 S 需要调题的时间是 $\max_{i\in S} C_i$。

每个人 i 有一个代价 W_i，一个段 S 的代价是 $T \times \sum_{i\in S} W_i$，其中 T 是这段之前的所有段的用时之和。

每个人 i 有一个参数 L_i，表示 i 和 L_i 不能在同一段。

你需要最小化段的代价之和。

$n \le 10^5$，代价都非负，$L_i < i$。

【集训队作业 2018】line

首先如果把前缀用时和放到 DP 里面，状态会很大。所以我们考虑用时对后面的贡献。

【集训队作业 2018】line

首先如果把前缀用时和放到 DP 里面，状态会很大。所以我们考虑用时对后面的贡献。

记 S_i 为 W_i 的后缀和，则有方程：

$$f_i = \min_{L_i \le j < i} \{f_j + S_{i+1} \times \left(\max_{j < k \le i} C_k\right)\}$$

【集训队作业 2018】line

首先如果把前缀用时和放到 DP 里面，状态会很大。所以我们考虑用时对后面的贡献。

记 S_i 为 W_i 的后缀和，则有方程：

$$f_i = \min_{L_i \le j < i} \{f_j + S_{i+1} \times \left(\max_{j < k \le i} C_k\right)\}$$

显然如果知道 max 就是一个点积最大化问题。那么肯定是单调栈了！

【集训队作业 2018】line

首先如果把前缀用时和放到 DP 里面，状态会很大。所以我们考虑用时对后面的贡献。

记 S_i 为 W_i 的后缀和，则有方程：

$$f_i = \min_{L_i \le j < i} \{ f_j + S_{i+1} \times \left(\max_{j < k \le i} C_k \right) \}$$

显然如果知道 max 就是一个点积最大化问题。那么肯定是单调栈了！

使用扫描线，那么对于相同 max 的左端点区间，我们使用线段树或支持动态在末尾插入的 ST 表等数据结构查区间最小值。

【集训队作业 2018】line

首先如果把前缀用时和放到 DP 里面，状态会很大。所以我们考虑用时对后面的贡献。

记 S_i 为 W_i 的后缀和，则有方程：

$$f_i = \min_{L_i \le j < i} \{f_j + S_{i+1} \times \left(\max_{j < k \le i} C_k \right)\}$$

显然如果知道 max 就是一个点积最大化问题。那么肯定是单调栈了！

使用扫描线，那么对于相同 max 的左端点区间，我们使用线段树或支持动态在末尾插入的 ST 表等数据结构查区间最小值。

特判掉不完整的区间，那么剩下的就是一个动态在末尾插入，动态末尾删除，动态查询区间凸壳的问题。

回顾【NOI2014】购票

先回来看看这道题，如果我们把每个状态来自哪个状态，连一条边，那么状态的结构就变成了一棵树，我们可以在树上转移。

末尾插入是往子节点跳，末尾删除是往父亲跳，区间凸壳是树上的链查询。

所以，实际上两个问题是本质相同的。

回顾【NOI2014】购票

先回来看看这道题，如果我们把每个状态来自哪个状态，连一条边，那么状态的结构就变成了一棵树，我们可以在树上转移。

末尾插入是往子节点跳，末尾删除是往父亲跳，区间凸壳是树上的链查询。

所以，实际上两个问题是本质相同的。

扩展一下，增加两个操作：在开头插入，开头删除。那么我们只是变一下根的指针而已。

回顾【NOI2014】购票

先回来看看这道题，如果我们把每个状态来自哪个状态，连一条边，那么状态的结构就变成了一棵树，我们可以在树上转移。

末尾插入是往子节点跳，末尾删除是往父亲跳，区间凸壳是树上的链查询。

所以，实际上两个问题是本质相同的。

扩展一下，增加两个操作：在开头插入，开头删除。那么我们只是变一下根的指针而已。

所以，变成动态的，我们就可以使用动态点分治了。（相信神仙们人均写过紫荆花之恋）

然后就可以在点分树上跑链上的区间凸壳了。在线的序列区间凸壳怎么写？

回顾【NOI2014】购票

既然不带删除，我们可以存下分治的结果。我们只需要使用类似线段树的结构，归并排序合并子树凸壳，查询在线段树上查询，就变成了整区间凸壳问题。

回顾【NOI2014】购票

既然不带删除，我们可以存下分治的结果。我们只需要使用类似线段树的结构，归并排序合并子树凸壳，查询在线段树上查询，就变成了整区间凸壳问题。

但是再套一个点分树空间就两个 $\log$ 了，有没有好一点的办法？

回顾【NOI2014】购票

既然不带删除，我们可以存下分治的结果。我们只需要使用类似线段树的结构，归并排序合并子树凸壳，查询在线段树上查询，就变成了整区间凸壳问题。

但是再套一个点分树空间就两个 log 了，有没有好一点的办法？

显然我们可以通过一些技巧将链查询变为点分树上的整链，所以在点分树上就规避了区间查询凸壳的问题，细节换空间，变成单 log 空间。

回顾【NOI2014】购票

既然不带删除，我们可以存下分治的结果。我们只需要使用类似线段树的结构，归并排序合并子树凸壳，查询在线段树上查询，就变成了整区间凸壳问题。

但是再套一个点分树空间就两个 log 了，有没有好一点的办法？

显然我们可以通过一些技巧将链查询变为点分树上的整链，所以在点分树上就规避了区间查询凸壳的问题，细节换空间，变成单 log 空间。

啥你说链查询可以用树剖。的确用树剖加上区间凸壳能得到细节和常数同时小很多的做法。空间单 log，预处理单 log，查询两 log。

回顾【NOI2014】购票

既然不带删除，我们可以存下分治的结果。我们只需要使用类似线段树的结构，归并排序合并子树凸壳，查询在线段树上查询，就变成了整区间凸壳问题。

但是再套一个点分树空间就两个 $\log$ 了，有没有好一点的办法？

显然我们可以通过一些技巧将链查询变为点分树上的整链，所以在点分树上就规避了区间查询凸壳的问题，细节换空间，变成单 $\log$ 空间。

啥你说链查询可以用树剖。的确用树剖加上区间凸壳能得到细节和常数同时小很多的做法。空间单 $\log$，预处理单 $\log$，查询两 $\log$。

虽说可以动态树剖，但是貌似因为有链的重构不太好讲，所以不再讨论树剖在此类动态问题上的应用，一般来说是一种优秀的静态链上凸包的解决方案。

【集训队作业 2018】line

我们发现，动态点分治完全可以用在这道题里。但是代码复杂度和常数都贼大。

【集训队作业 2018】line

我们发现，动态点分治完全可以用在这道题里。但是代码复杂度和常数都贼大。

如果记下分治过程，插入时，为了不更新过多的节点，我们使用二进制分组。

【集训队作业 2018】line

我们发现，动态点分治完全可以用在这道题里。但是代码复杂度和常数都贼大。

如果记下分治过程，插入时，为了不更新过多的节点，我们使用二进制分组。

但是如果删除一个组的末尾，然后我们再加回去，会重新 $O(L)$ 的信息合并。如此反复横跳就爆了。

我们发现，正是因为组的拆分太容易，导致了复杂度的不平衡。

【集训队作业 2018】line

我们发现，动态点分治完全可以用在这道题里。但是代码复杂度和常数都贼大。

如果记下分治过程，插入时，为了不更新过多的节点，我们使用二进制分组。

但是如果删除一个组的末尾，然后我们再加回去，会重新 $O(L)$ 的信息合并。如此反复横跳就爆了。

我们发现，正是因为组的拆分太容易，导致了复杂度的不平衡。

于是类比插入的势能，我们给删除也加一组势能。对于每次拆分，我们也需要 L 的势能。

【集训队作业 2018】line

我们发现，动态点分治完全可以用在这道题里。但是代码复杂度和常数都贼大。

如果记下分治过程，插入时，为了不更新过多的节点，我们使用二进制分组。

但是如果删除一个组的末尾，然后我们再加回去，会重新 $O(L)$ 的信息合并。如此反复横跳就爆了。

我们发现，正是因为组的拆分太容易，导致了复杂度的不平衡。

于是类比插入的势能，我们给删除也加一组势能。对于每次拆分，我们也需要 L 的势能。

为了方便地描述算法，我们在长度为 2^M 的线段树上模拟这个二进制分组。子树满了的节点称为实节点，此时每个实节点的连通块就是每个二进制分组。

此时区间合并和区间拆分对应组合并和组拆分的意义就显然了。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

我们记每一层节点个数为 C，区间长度为 $L=2^k$，整个二进制分组维护的序列大小为 S。

我们定义插入势能 $\Phi_I=\max\{S-C\times L,0\}$。

我们定义删除势能 $\Phi_D=\max\{C\times L-S,0\}$。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

我们记每一层节点个数为 C，区间长度为 $L = 2^k$，整个二进制分组维护的序列大小为 S。

我们定义插入势能 $\Phi_I = \max\{S - C \times L, 0\}$。

我们定义删除势能 $\Phi_D = \max\{C \times L - S, 0\}$。

当插入势能 $\Phi_I \geq 2^k$ 时，我们进行区间合并，增加一个区间，并消耗 2^k 的插入势能。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

我们记每一层节点个数为 C，区间长度为 $L=2^k$，整个二进制分组维护的序列大小为 S。

我们定义插入势能 $\Phi_I=\max\{S-C\times L,0\}$。

我们定义删除势能 $\Phi_D=\max\{C\times L-S,0\}$。

当插入势能 $\Phi_I\geq 2^k$ 时，我们进行区间合并，增加一个区间，并消耗 2^k 的插入势能。

当删除势能 $\Phi_D\geq 2^k$ 时，我们进行区间拆分，减少一个区间，并消耗 2^k 的删除势能。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

我们记每一层节点个数为 C，区间长度为 $L = 2^k$，整个二进制分组维护的序列大小为 S。

我们定义插入势能 $\Phi_I = \max\{S - C \times L, 0\}$。

我们定义删除势能 $\Phi_D = \max\{C \times L - S, 0\}$。

当插入势能 $\Phi_I \geq 2^k$ 时，我们进行区间合并，增加一个区间，并消耗 2^k 的插入势能。

当删除势能 $\Phi_D \geq 2^k$ 时，我们进行区间拆分，减少一个区间，并消耗 2^k 的删除势能。

当插入时，如果删除势能 $\Phi_D > 0$，则删除势能 Φ_D 会消耗 1，否则插入势能 Φ_I 会增加 1。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

我们记每一层节点个数为 C，区间长度为 $L = 2^k$，整个二进制分组维护的序列大小为 S。

我们定义插入势能 $\Phi_I = \max\{S - C \times L, 0\}$。

我们定义删除势能 $\Phi_D = \max\{C \times L - S, 0\}$。

当插入势能 $\Phi_I \geq 2^k$ 时，我们进行区间合并，增加一个区间，并消耗 2^k 的插入势能。

当删除势能 $\Phi_D \geq 2^k$ 时，我们进行区间拆分，减少一个区间，并消耗 2^k 的删除势能。

当插入时，如果删除势能 $\Phi_D > 0$，则删除势能 Φ_D 会消耗 1，否则插入势能 Φ_I 会增加 1。

当删除时，如果插入势能 $\Phi_I > 0$，则插入势能 Φ_I 会消耗 1，否则删除势能 Φ_D 会增加 1。

【集训队作业 2018】line

我们发现删除和插入互相抵消。我们定义两个势能，插入势能和删除势能。每个时刻最多只有一种势能为正。

我们记每一层节点个数为 C，区间长度为 $L = 2^k$，整个二进制分组维护的序列大小为 S。

我们定义插入势能 $\Phi_I = \max\{S - C \times L, 0\}$。

我们定义删除势能 $\Phi_D = \max\{C \times L - S, 0\}$。

当插入势能 $\Phi_I \geq 2^k$ 时，我们进行区间合并，增加一个区间，并消耗 2^k 的插入势能。

当删除势能 $\Phi_D \geq 2^k$ 时，我们进行区间拆分，减少一个区间，并消耗 2^k 的删除势能。

当插入时，如果删除势能 $\Phi_D > 0$，则删除势能 Φ_D 会消耗 1，否则插入势能 Φ_I 会增加 1。

当删除时，如果插入势能 $\Phi_I > 0$，则插入势能 Φ_I 会消耗 1，否则删除势能 Φ_D 会增加 1。

此时区间合并的复杂度已经恢复正常。

【集训队作业 2018】line

注意到我们删除节点的时候，有时会产生一个区间没被拆分却存着过时的信息，称为坏区间。

【集训队作业 2018】line

注意到我们删除节点的时候，有时会产生一个区间没被拆分却存着过时的信息，称为坏区间。

每次拆分，一定存在一个坏区间，我们拆分它。

【集训队作业 2018】line

注意到我们删除节点的时候，有时会产生一个区间没被拆分却存着过时的信息，称为坏区间。

每次拆分，一定存在一个坏区间，我们拆分它。

每次插入，如果存在坏区间，就在坏区间上原地重建，并且把它右边那个变成新的坏区间。否则就在最右边新建一个区间。

【集训队作业 2018】line

注意到我们删除节点的时候，有时会产生一个区间没被拆分却存着过时的信息，称为坏区间。

每次拆分，一定存在一个坏区间，我们拆分它。

每次插入，如果存在坏区间，就在坏区间上原地重建，并且把它右边那个变成新的坏区间。否则就在最右边新建一个区间。

这样子，每层最多有一个坏区间，它一定在最右边。

【集训队作业 2018】line

注意到我们删除节点的时候，有时会产生一个区间没被拆分却存着过时的信息，称为坏区间。

每次拆分，一定存在一个坏区间，我们拆分它。

每次插入，如果存在坏区间，就在坏区间上原地重建，并且把它右边那个变成新的坏区间。否则就在最右边新建一个区间。

这样子，每层最多有一个坏区间，它一定在最右边。

考虑插入和删除，每次最多增加 $O(\log n)$ 的势能，所以每次操作均摊 $O(\log n)$。

【集训队作业 2018】line

注意到我们删除节点的时候，有时会产生一个区间没被拆分却存着过时的信息，称为坏区间。

每次拆分，一定存在一个坏区间，我们拆分它。

每次插入，如果存在坏区间，就在坏区间上原地重建，并且把它右边那个变成新的坏区间。否则就在最右边新建一个区间。

这样子，每层最多有一个坏区间，它一定在最右边。

考虑插入和删除，每次最多增加 $O(\log n)$ 的势能，所以每次操作均摊 $O(\log n)$。

考虑查询，最多访问 $O(\log n)$ 个节点，访问的坏区间也最多 $O(\log n)$ 个，加上这道题查询所用的二分，所以复杂度 $O(\log^2 n)$。

【集训队作业 2018】line

但是我们还是可以补救一下。我们发现转移方程中 S 是单调的!!!
于是对于每个节点维护一个指针，扫过去即可。

【集训队作业 2018】line

但是我们还是可以补救一下。我们发现转移方程中 S 是单调的!!!

于是对于每个节点维护一个指针，扫过去即可。

查询复杂度变为均摊 $O(\log n)$。

总复杂度时空 $O(n\log n)$，是个常数不大，代码复杂度不大的优秀做法。

【CF455E】Function

Example (Function)

你有一个函数 $f(x,y)(1\le x,y\le n)$：

$$f(x,y)=\begin{cases}A_y & x=1\\ \min\{f(x-1,y),f(x-1,y-1)\}+A_y & 2\le x\end{cases}$$

Q 组询问单点 $f(V,R)(V\le R)$ 。
$n,Q\le 10^5,0\le A_i\le 10^4$。

【CF455E】Function

Example (Function)

你有一个函数 $f(x,y)(1 \le x,y \le n)$：

$$f(x,y)=\begin{cases} A_y & x=1 \\ \min\{f(x-1,y),f(x-1,y-1)\}+A_y & 2\le x \end{cases}$$

Q 组询问单点 $f(V,R)(V\le R)$ 。
$n,Q\le 10^5, 0\le A_i\le 10^4$。

Obviously you can use EIT to solve it. By ignore2004

【CF455E】Function

转化一下这个函数的意义：取右端点为 $r = R$，选择一个左端点 $R - V + 1 = L \le l$，权值为

$$\sum_{l \le i \le r} A_i + (V - r + l) \left(\min_{l \le i \le r} A_i \right)$$

【CF455E】Function

转化一下这个函数的意义：取右端点为 $r = R$，选择一个左端点 $R - V + 1 = L \le l$，权值为

$$\sum_{l \le i \le r} A_i + (V - r + l)\left(\min_{l \le i \le r} A_i\right)$$

是不是差点去写单调栈？仔细分析一下，其实左端点一定是最小值，否则将左端点向右移到最小值处答案会更小。

【CF455E】Function

转化一下这个函数的意义：取右端点为 $r = R$，选择一个左端点 $R - V + 1 = L \le l$，权值为

$$\sum_{l \le i \le r} A_i + (V - r + l)\left(\min_{l \le i \le r} A_i\right)$$

是不是差点去写单调栈？仔细分析一下，其实左端点一定是最小值，否则将左端点向右移到最小值处答案会更小。

这样答案就是

$$\min_{L \le i \le R}\{\sum_{i \le j \le R} A_j + A_i(V - R + i)\}$$

【CF455E】Function

转化一下这个函数的意义：取右端点为 $r = R$，选择一个左端点 $R - V + 1 = L \le l$，权值为

$$\sum_{l \le i \le r} A_i + (V - r + l)\left(\min_{l \le i \le r} A_i\right)$$

是不是差点去写单调栈？仔细分析一下，其实左端点一定是最小值，否则将左端点向右移到最小值处答案会更小。

这样答案就是

$$\min_{L \le i \le R}\{\sum_{i \le j \le R} A_j + A_i(V - R + i)\}$$

将区间和改写成前缀和，就是一个点积最大化问题。于是直接调用分治区间凸包解决。

【CF455E】Function 加强版

Example (Function 加强版)

你有一个函数 $f(x,y)(1 \le x,y \le n)$:

$$f(x,y)=\begin{cases} A_y & x=1 \\ \min\{f(x-1,y),f(x-1,y-1)\}+A_y & 2 \le x \end{cases}$$

Q 组询问单点 $f(V,R)(V \le R)$ 。
$n,Q \le 10^6, 0 \le A_i \le 10^4$。

【CF455E】Function 加强版

Example (Function 加强版)

你有一个函数 $f(x,y)(1 \le x,y \le n)$：

$$f(x,y) = \begin{cases} A_y & x = 1 \\ \min\{f(x-1,y), f(x-1,y-1)\} + A_y & 2 \le x \end{cases}$$

Q 组询问单点 $f(V,R)(V \le R)$ 。
$n, Q \le 10^6, 0 \le A_i \le 10^4$。

Obviously you can use EIT to solve it. By ignore2004

【CF455E】Function 加强版

Example (Function 加强版)

你有一个函数 $f(x,y)(1 \le x,y \le n)$：

$$f(x,y)=\begin{cases} A_y & x=1 \\ \min\{f(x-1,y),f(x-1,y-1)\}+A_y & 2 \le x \end{cases}$$

Q 组询问单点 $f(V,R)(V \le R)$ 。
$n,Q \le 10^6, 0 \le A_i \le 10^4$。

Obviously you can use EIT to solve it. By ignore2004
据说这道题 KTT 是俩 log 的，但是常数小，ignore2004 觉得能过。

【CF455E】Function 加强版

Example (Function 加强版)

你有一个函数 $f(x,y)(1 \le x,y \le n)$：

$$f(x,y)=\begin{cases} A_y & x=1 \\ \min\{f(x-1,y),f(x-1,y-1)\}+A_y & 2 \le x \end{cases}$$

Q 组询问单点 $f(V,R)(V \le R)$ 。
$n,Q \le 10^6, 0 \le A_i \le 10^4$。

Obviously you can use EIT to solve it. By ignore2004
据说这道题 KTT 是俩 log 的，但是常数小，ignore2004 觉得能过。
我们可以区间凸包，询问排序后双指针做到 $O(n\log n)$ 时间，$O(n)$ 空间，但是常数并不小。我们还有常数更小的做法。

【CF455E】Function 加强版

$$\min_{L\le i\le R}\{\sum_{i\le j\le R} A_j + A_i(V-R+i)\}$$

记前缀和为 S_i，注意到点为 $(A_i, iA_i - S_i)$。
扫描线单调栈，可以得到有用点中当 $i<j$ 时 $A_i<A_j$。

【CF455E】Function 加强版

$$\min_{L\le i\le R}\{\sum_{i\le j\le R} A_j + A_i(V-R+i)\}$$

记前缀和为 S_i，注意到点为 $(A_i, iA_i - S_i)$。

扫描线单调栈，可以得到有用点中当 $i<j$ 时 $A_i<A_j$。

但是，貌似还是要区间凸包，还不能利用询问单调，我们只能从后缀凸壳中找性质。

【CF455E】Function 加强版

$$\min_{L\le i\le R}\{\sum_{i\le j\le R} A_j + A_i(V-R+i)\}$$

记前缀和为 S_i，注意到点为 $(A_i, iA_i - S_i)$。

扫描线单调栈，可以得到有用点中当 $i<j$ 时 $A_i<A_j$。

但是，貌似还是要区间凸包，还不能利用询问单调，我们只能从后缀凸壳中找性质。

一个想法就是凸壳的后缀等效于后缀的凸壳，这样可以方便维护。

直觉告诉我们，一般情况下这个东西不对。但是这道题模型十分特殊，我们选择暴力对拍，发现竟然是对的!!!

【CF455E】Function 加强版

为了方便叙述，我们将点的纵坐标取反，查询点的横坐标取反。注意到 $L-1=R-V$，问题变成：

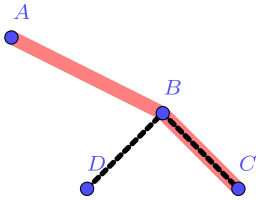
$$\max\{(A_i, S_i - iA_i) \cdot (L-1, 1)\}$$

【CF455E】Function 加强版

为了方便叙述，我们将点的纵坐标取反，查询点的横坐标取反。注意到 $L-1=R-V$，问题变成：

$$\max\{(A_i, S_i - iA_i)\cdot(L-1, 1)\}$$

一般地说，后缀的凸壳 DBC 与凸壳 ABC 的后缀 BC 并不等效。

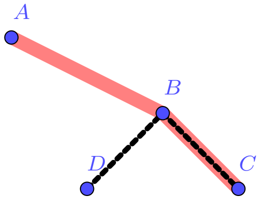


【CF455E】Function 加强版

为了方便叙述，我们将点的纵坐标取反，查询点的横坐标取反。注意到 $L-1=R-V$，问题变成：

$$\max\{(A_i, S_i - iA_i)\cdot(L-1,1)\}$$

一般地说，后缀的凸壳 DBC 与凸壳 ABC 的后缀 BC 并不等效。



所以，在这道题中，我们需要证明**给定相邻两个凸壳点** i **和** j**，对于任意点** $i<k<j$**，对于任意询问都有** k **不优于** j。

【CF455E】Function 加强版

使用反证法，假设 k 比 j 优，则有：

$$
\begin{aligned}
P &= (A_i, S_i - iA_i) \\
Q &= (A_k, S_k - kA_k) \\
R &= (A_j, S_j - jA_j) \\
H &= (L-1, 1) \\
A_i &< A_k < A_j \\
S_k - S_i &\ge (k-i)A_k \\
S_j - S_k &\ge (j-k)A_j \\
i &\le L-1 < k \\
(Q-P) \times (R-Q) &\ge 0 \\
(R-Q) \cdot H &< 0
\end{aligned}
$$

【CF455E】Function 加强版

$$
\begin{aligned}
&\text{let } k = i + \delta_k, j = k + \delta_j \\
&\text{let } A_k = A_i + \Delta_k, A_j = A_k + \Delta_j \\
&\text{let } S_k = S_i + (k-i)A_k + \gamma_k, \\
&\quad S_j = S_k + (j-k)A_j + \gamma_j \\
&\quad \delta_k, \delta_j, \Delta_k, \Delta_j, \gamma_k, \gamma_j > 0 \\
&\quad (Q-P) \times (R-Q) \\
&= \gamma_j \Delta_k - (\gamma_k + \delta_k \Delta_k)\Delta_j \\
&\geq 0 \\
&\quad (R-Q) \cdot H \\
&= \gamma_j - \Delta_j (k - L + 1) \\
&< 0
\end{aligned}
$$

【CF455E】Function 加强版

$$
\begin{aligned}
\gamma_j\Delta_k &\geq (\gamma_k+\delta_k\Delta_k)\Delta_j \\
\frac{\gamma_j\Delta_k}{\gamma_k+\delta_k\Delta_k} &\geq \Delta_j \\
\gamma_j &< \Delta_j(k-L+1) \\
\frac{\gamma_j}{k-L+1} &< \Delta_j \\
\frac{\gamma_j}{k-L+1} &< \frac{\gamma_j\Delta_k}{\gamma_k+\delta_k\Delta_k} \\
\gamma_k+\delta_k\Delta_k &< \Delta_k(k-L+1)
\end{aligned}
$$

【CF455E】Function 加强版

$$
\begin{aligned}
\gamma_j\Delta_k &\geq (\gamma_k+\delta_k\Delta_k)\Delta_j \\
\frac{\gamma_j\Delta_k}{\gamma_k+\delta_k\Delta_k} &\geq \Delta_j \\
\gamma_j &< \Delta_j(k-L+1) \\
\frac{\gamma_j}{k-L+1} &< \Delta_j \\
\frac{\gamma_j}{k-L+1} &< \frac{\gamma_j\Delta_k}{\gamma_k+\delta_k\Delta_k} \\
\gamma_k+\delta_k\Delta_k &< \Delta_k(k-L+1)
\end{aligned}
$$

因为有 $\delta_k \geq (k-L+1)$，矛盾，于是证毕。

【NOI2019】回家路线

Example (回家路线)

有 n 个点，m 种车，每种车在时刻 l_i 从点 u_i 出发，在时刻 r_i 到达点 v_i。

你有一个函数 $f(x) = ax^2 + bx + c$。

你需要找出满足下列条件的序列 $A_i(1 \le i \le |A| = k)$：

1. $u_{A_1} = 1,\ v_{A_k} = n,\ v_{A_i} = u_{A_{i+1}}(1 \le i < k)$。
2. $r_{A_i} \le l_{A_{i+1}}(1 \le i < k)$。

并且最小化：

$$r_{A_n} + \sum_{i=1}^{k} f(l_{A_i} - r_{A_{i-1}})$$

我们认为 $A_0 = r_0 = 0$。保证 $n \le 10^5, m \le 2 \times 10^5$。

保证 $0 \le a \le 10, 0 \le b, c \le 10^6, 0 \le l_i < r_i \le 10^3$。

【NOI2019】回家路线

Obviously you can use BFA to solve it. By ignore2004
(hint: BFA, Brute Force Algorithm)

【NOI2019】回家路线

Obviously you can use BFA to solve it. By ignore2004

(hint: BFA, Brute Force Algorithm) 这道题良心的出题人为了防止转移爆 int??? 反正听说场上把 r_i 的 10^6 改成 10^3。

那么我们真的就可以写暴力了。

【NOI2019】回家路线

Obviously you can use BFA to solve it. By ignore2004

(hint: BFA, Brute Force Algorithm) 这道题良心的出题人为了防止转移爆 int??? 反正听说场上把 r_i 的 10^6 改成 10^3。

那么我们真的就可以写暴力了。

显然状态和所在点以及时间有关系，我们刚好可以开一个 10^8 的 int 数组 (`381MB`)。

【NOI2019】回家路线

Obviously you can use BFA to solve it. By ignore2004

(hint: BFA, Brute Force Algorithm) 这道题良心的出题人为了防止转移爆 int??? 反正听说场上把 r_i 的 10^6 改成 10^3。

那么我们真的就可以写暴力了。

显然状态和所在点以及时间有关系，我们刚好可以开一个 10^8 的 int 数组 (381MB)。

那么按时间左端点枚举边，直接暴力转移就行，复杂度 $O(mT)$。

注意数组和循环的顺序以获得小常数。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g=\min\{f+a(y-x)^2+b(y-x)+c\}$ 是一个点积最大化问题。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g = \min\{f + a(y-x)^2 + b(y-x) + c\}$ 是一个点积最大化问题。

显然答案为 $(-2ax + b, f + ax^2 - bx + c) \cdot (y, 1)$ 的最小值。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g=\min\{f+a(y-x)^2+b(y-x)+c\}$ 是一个点积最大化问题。

显然答案为 $(-2ax+b,f+ax^2-bx+c)\cdot(y,1)$ 的最小值。

取反一下坐标变为最大化，发现横坐标 $2ax-b$ 关于 x 的递增而递增。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g = \min\{f + a(y-x)^2 + b(y-x) + c\}$ 是一个点积最大化问题。

显然答案为 $(-2ax + b, f + ax^2 - bx + c) \cdot (y, 1)$ 的最小值。

取反一下坐标变为最大化，发现横坐标 $2ax - b$ 关于 x 的递增而递增。

还是按照时间左端点枚举边，我们发现要给每个点维护一个点集。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g=\min\{f+a(y-x)^2+b(y-x)+c\}$ 是一个点积最大化问题。

显然答案为 $(-2ax+b,f+ax^2-bx+c)\cdot(y,1)$ 的最小值。

取反一下坐标变为最大化，发现横坐标 $2ax-b$ 关于 x 的递增而递增。

还是按照时间左端点枚举边，我们发现要给每个点维护一个点集。

此时查询的点是单调的，我们可以通过在凸壳上移动指针完成查询。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g = \min\{f + a(y-x)^2 + b(y-x) + c\}$ 是一个点积最大化问题。

显然答案为 $(-2ax + b, f + ax^2 - bx + c) \cdot (y, 1)$ 的最小值。

取反一下坐标变为最大化，发现横坐标 $2ax - b$ 关于 x 的递增而递增。

还是按照时间左端点枚举边，我们发现要给每个点维护一个点集。

此时查询的点是单调的，我们可以通过在凸壳上移动指针完成查询。

为了让修改的点单调，同时为了查询时询问的不是前缀凸壳而是全局凸壳，我们暂存转移后的点，在左端点变化的时候，再加进点集里。

【NOI2019】回家路线

作为优秀的共产主义接班人，我们不能止步于暴力。

显然这个转移式 $g = \min\{f + a(y-x)^2 + b(y-x) + c\}$ 是一个点积最大化问题。

显然答案为 $(-2ax+b, f+ax^2-bx+c) \cdot (y, 1)$ 的最小值。

取反一下坐标变为最大化，发现横坐标 $2ax-b$ 关于 x 的递增而递增。

还是按照时间左端点枚举边，我们发现要给每个点维护一个点集。

此时查询的点是单调的，我们可以通过在凸壳上移动指针完成查询。

为了让修改的点单调，同时为了查询时询问的不是前缀凸壳而是全局凸壳，我们暂存转移后的点，在左端点变化的时候，再加进点集里。

这样就是一个典型的单调队列维护凸壳模型了。

【ARC066F】Contest with Drinks Hard

Example (Contest with Drinks Hard)

给你一个序列 A，Q 次询问，每次询问修改一个元素，询问后复原。

每次询问，你需要选择若干个元素，记 B_i 为是否选了第 i 个，价值为连续区间数减去选择的元素权值和，即：

$$\sum_{l=1}^{n}\sum_{r=1}^{n}\prod_{k=l}^{r}B_k-\sum_{i=1}^{n}B_iA_i$$

最大化价值。保证 $|A|,Q\le 3\times 10^5$。

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

如果不经过，那么修改对答案没有影响。如果经过，答案的变化量和这个元素的变化量相等。那么两种方案取 $\max$ 即可。

然后问题就是求出经过/不经过一个点的最大答案。

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

如果不经过，那么修改对答案没有影响。如果经过，答案的变化量和这个元素的变化量相等。那么两种方案取 $\max$ 即可。

然后问题就是求出经过/不经过一个点的最大答案。

首先，这道题显然是一个点积最大化问题。于是可以求出前缀答案和后缀答案。于是不经过就可以算出了。

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

如果不经过，那么修改对答案没有影响。如果经过，答案的变化量和这个元素的变化量相等。那么两种方案取 $\max$ 即可。

然后问题就是求出经过/不经过一个点的最大答案。

首先，这道题显然是一个点积最大化问题。于是可以求出前缀答案和后缀答案。于是不经过就可以算出了。

对于经过的，我们分治枚举经过中点的区间，对于每个左端点和右端点都要计算。因为两个过程等价，下面我们只考虑右端点：

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

如果不经过，那么修改对答案没有影响。如果经过，答案的变化量和这个元素的变化量相等。那么两种方案取 max 即可。

然后问题就是求出经过/不经过一个点的最大答案。

首先，这道题显然是一个点积最大化问题。于是可以求出前缀答案和后缀答案。于是不经过就可以算出了。

对于经过的，我们分治枚举经过中点的区间，对于每个左端点和右端点都要计算。因为两个过程等价，下面我们只考虑右端点：

对于一个右端点，我们在右半边区间枚举它；预处理左半边区间形成的凸壳，在右端点找到取得最优答案的左端点。

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

如果不经过，那么修改对答案没有影响。如果经过，答案的变化量和这个元素的变化量相等。那么两种方案取 max 即可。

然后问题就是求出经过/不经过一个点的最大答案。

首先，这道题显然是一个点积最大化问题。于是可以求出前缀答案和后缀答案。于是不经过就可以算出了。

对于经过的，我们分治枚举经过中点的区间，对于每个左端点和右端点都要计算。因为两个过程等价，下面我们只考虑右端点：

对于一个右端点，我们在右半边区间枚举它；预处理左半边区间形成的凸壳，在右端点找到取得最优答案的左端点。

因为对于每个点要求出经过它最大的答案，所以在枚举右端点的同时，还要做一个区间取 max 操作。

【ARC066F】Contest with Drinks Hard

显然，只要讨论方案是否经过修改的这个元素。

如果不经过，那么修改对答案没有影响。如果经过，答案的变化量和这个元素的变化量相等。那么两种方案取 $\max$ 即可。

然后问题就是求出经过/不经过一个点的最大答案。

首先，这道题显然是一个点积最大化问题。于是可以求出前缀答案和后缀答案。于是不经过就可以算出了。

对于经过的，我们分治枚举经过中点的区间，对于每个左端点和右端点都要计算。因为两个过程等价，下面我们只考虑右端点：

对于一个右端点，我们在右半边区间枚举它；预处理左半边区间形成的凸壳，在右端点找到取得最优答案的左端点。

因为对于每个点要求出经过它最大的答案，所以在枚举右端点的同时，还要做一个区间取 $\max$ 操作。

对于这一点，使用前缀和标记即可。