Introduction to Monotonic Optimization

daklqw

Zhenhai High School

February 8, 2022

Frequently Asked Questions

- Q：这课件好简单啊。

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。
- Q：那我从哪里开始听？

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。
- Q：那我从哪里开始听？
- A：我好菜啊。

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。
- Q：那我从哪里开始听？
- A：我好菜啊。
- Q：咋写的这么不严谨啊？

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。
- Q：那我从哪里开始听？
- A：我好菜啊。
- Q：咋写的这么不严谨啊？
- A：我好菜啊。

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。
- Q：那我从哪里开始听？
- A：我好菜啊。
- Q：咋写的这么不严谨啊？
- A：我好菜啊。
- Q：楼下在假。

Frequently Asked Questions

- Q：这课件好简单啊。
- A：我好菜啊。
- Q：那我从哪里开始听？
- A：我好菜啊。
- Q：咋写的这么不严谨啊？
- A：我好菜啊。
- Q：楼下在假。
- A：我好菜啊。

前言

大家好，我是 daklqw，我来讲单调性优化。
当然我不是单调性优化带师，所以我讲得比较浅显。
据说在第一页全写英文能显得高级。
相信大家 OI 水平能吊打我，所以大家不要自闭。
不会的尽管问我。不过我相信我也不会。
ZJOI rk2 高一 rk1 $E 就不要提前上来秒题了。

全序

Definition (Total Order)

定义集合 S 的二元关系 $\preceq$，如果 $\preceq$ 满足：

- $\forall a, b \in S$，有 $a \preceq b$ 或 $b \preceq a$。（完全性）
- 若对 $a, b \in S$ 有 $a \preceq b$ 且 $b \preceq a$，则有 $a = b$。（反对称性）
- 若对 $a, b, c \in S$ 有 $a \preceq b$ 且 $b \preceq c$，则有 $a \preceq c$。（传递性）

则 $\preceq$ 是集合 S 上的全序 (Total Order) 关系。

全序

Definition (Total Order)

定义集合 S 的二元关系 $\preceq$，如果 $\preceq$ 满足：

- $\forall a, b \in S$，有 $a \preceq b$ 或 $b \preceq a$。（完全性）
- 若对 $a, b \in S$ 有 $a \preceq b$ 且 $b \preceq a$，则有 $a = b$。（反对称性）
- 若对 $a, b, c \in S$ 有 $a \preceq b$ 且 $b \preceq c$，则有 $a \preceq c$。（传递性）

则 $\preceq$ 是集合 S 上的全序 (Total Order) 关系。

显然，我们熟知的实数的二元关系 $\leq$ 在实数集 R 上就是一个全序关系。

偏序

Definition (Partial Order)

定义集合 S 的二元关系 $\preceq$，如果 $\preceq$ 满足：

- $\forall a \in S$，有 $a \preceq a$。（自反性）
- 若对 $a, b \in S$ 有 $a \preceq b$ 且 $b \preceq a$，则有 $a = b$。（反对称性）
- 若对 $a, b, c \in S$ 有 $a \preceq b$ 且 $b \preceq c$，则有 $a \preceq c$。（传递性）

则 $\preceq$ 是集合 S 上的偏序 (Partial Order) 关系。

偏序

Definition (Partial Order)

定义集合 S 的二元关系 $\preceq$，如果 $\preceq$ 满足：

- $\forall a \in S$，有 $a \preceq a$。（自反性）
- 若对 $a, b \in S$ 有 $a \preceq b$ 且 $b \preceq a$，则有 $a = b$。（反对称性）
- 若对 $a, b, c \in S$ 有 $a \preceq b$ 且 $b \preceq c$，则有 $a \preceq c$。（传递性）

则 $\preceq$ 是集合 S 上的偏序 (Partial Order) 关系。

为什么全序没有写自反性？因为完全性包含自反性。当然全序也是一种偏序。

偏序

Definition (Partial Order)

定义集合 S 的二元关系 $\preceq$，如果 $\preceq$ 满足：

- $\forall a \in S$，有 $a \preceq a$。（自反性）
- 若对 $a, b \in S$ 有 $a \preceq b$ 且 $b \preceq a$，则有 $a = b$。（反对称性）
- 若对 $a, b, c \in S$ 有 $a \preceq b$ 且 $b \preceq c$，则有 $a \preceq c$。（传递性）

则 $\preceq$ 是集合 S 上的偏序 (Partial Order) 关系。

为什么全序没有写自反性？因为完全性包含自反性。当然全序也是一种偏序。

定义二维平面点 $p = (p_x, p_y) \in R^2$ 的二元关系 $a \preceq b$ 当且仅当 $a_x \leq b_x$ 且 $a_y \leq b_y$。

则 $\preceq$ 是点集合 R^2 上的偏序关系。

这就是我们熟悉的二维偏序，也就是二维数点。

单调性

Definition (Monotonicity)

对于一个关于 S 的序列 P，如果对于一个二元关系 $\preceq$，满足 $\forall i \in [1, |S|)$，有 $P_i \preceq P_{i+1}$，则称 P 关于 $\preceq$ 具有单调性 (Monotonicity)。也可以说 P 是关于 $\preceq$ 单调的。

单调性

Definition (Monotonicity)

对于一个关于 S 的序列 P，如果对于一个二元关系 $\preceq$，满足 $\forall i \in [1, |S|)$，有 $P_i \preceq P_{i+1}$，则称 P 关于 $\preceq$ 具有单调性 (Monotonicity)。也可以说 P 是关于 $\preceq$ 单调的。

针对题目的性质，在定义全序关系之后，能够用全序关系排除一些多余的元素。

同时使剩下元素在另一个全序关系上单调，把二维问题降为一维，变成序列问题。

这时可以对单调性的序列套用二分等技巧达到优化的目的。

单调栈

Definition (Monotonic Stack)

定义单调栈 (Monotonic Stack) 是满足如下性质的三元组 $(M, S, \preceq)$:

- M 是一个关于 S 的栈。
- $\preceq$ 是 S 的全序关系。
- 对于 M 的栈底到栈顶的元素序列 P，P 是关于 $\preceq$ 单调的。

单调队列

Definition (Monotonic Queue)

定义单调队列 (Monotonic Queue) 是满足如下性质的三元组 $(M, S, \preceq)$：

- M 是一个关于 S 的队列。
- $\preceq$ 是 S 的全序关系。
- 对于 M 的队列头到队列尾的元素序列 P，P 是关于 $\preceq$ 单调的。

单调栈、单调队列

两个数据结构差别不大，需要根据题目的模型进行选择。

这两个数据结构虽然运用广泛，但是也不是万能的。

当然，学会维护一个单调序列之后，这两种数据结构也会变得显然了。

【经典题】最长上升子序列

Example (最长上升子序列)

给你一个序列 P，求出 P 的最长上升子序列。$1 \le |P| \le 10^6$。

【经典题】最长上升子序列

Example (最长上升子序列)

给你一个序列 P，求出 P 的最长上升子序列。$1 \leq |P| \leq 10^6$。

首先有一个简单的 DP：$f_i = \max_{j<i, a_j<a_i} f_j + 1$。

【经典题】最长上升子序列

Example (最长上升子序列)

给你一个序列 P，求出 P 的最长上升子序列。$1 \le |P| \le 10^6$。

首先有一个简单的 DP：$f_i = \max_{j<i,a_j<a_i} f_j + 1$。

此时，如果从小到大枚举 i，不断地往有序序列 S 里插入元素，求 S 中前缀最大值就可以。

【经典题】最长上升子序列

Example (最长上升子序列)

给你一个序列 P，求出 P 的最长上升子序列。$1 \leq |P| \leq 10^6$。

首先有一个简单的 DP：$f_i = \max_{j<i, a_j<a_i} f_j + 1$。

此时，如果从小到大枚举 i，不断地往有序序列 S 里插入元素，求 S 中前缀最大值就可以。

如果定义 (x_a, x_f) 的偏序为 $x_a \leq y_a, x_f \geq y_f$，如果 $a \preceq b$，则 b 无论如何都不比 a 优秀。

此时如果去掉所有不优秀的元素，则 $x_a \leq y_a, x_f \leq y_f$ 是留下来的元素集合的全序。

此时若维护一个有序数组，则两维分别都是一个单调的数列。

【经典题】最长上升子序列

Example (最长上升子序列)

给你一个序列 P，求出 P 的最长上升子序列。$1 \le |P| \le 10^6$。

首先有一个简单的 DP：$f_i = \max_{j<i, a_j<a_i} f_j + 1$。

此时，如果从小到大枚举 i，不断地往有序序列 S 里插入元素，求 S 中前缀最大值就可以。

如果定义 (x_a, x_f) 的偏序为 $x_a \le y_a, x_f \ge y_f$，如果 $a \preceq b$，则 b 无论如何都不比 a 优秀。

此时如果去掉所有不优秀的元素，则 $x_a \le y_a, x_f \le y_f$ 是留下来的元素集合的全序。

此时若维护一个有序数组，则两维分别都是一个单调的数列。

在得到转移点 p 后，就得到了 $f_i = p_f + 1$ 后，如何维护序列？

【经典题】最长上升子序列

Example (最长上升子序列)

给你一个序列 P，求出 P 的最长上升子序列。$1 \le |P| \le 10^6$。

首先有一个简单的 DP：$f_i = \max_{j<i, a_j<a_i} f_j + 1$。

此时，如果从小到大枚举 i，不断地往有序序列 S 里插入元素，求 S 中前缀最大值就可以。

如果定义 (x_a, x_f) 的偏序为 $x_a \le y_a, x_f \ge y_f$，如果 $a \preceq b$，则 b 无论如何都不比 a 优秀。

此时如果去掉所有不优秀的元素，则 $x_a \le y_a, x_f \le y_f$ 是留下来的元素集合的全序。

此时若维护一个有序数组，则两维分别都是一个单调的数列。

在得到转移点 p 后，就得到了 $f_i = p_f + 1$ 后，如何维护序列？

此时 $p_a < a_i, p_a < p_{a+1}$，则令新的 $p'_{a+1} = \min\{p_{a+1}, a_i\}$ 即可。

【经典题】最长上升子序列

∞	∞	∞	∞	∞	Initial State.
1	∞	∞	∞	∞	Insert: 1.
1	4	∞	∞	∞	Insert: 4.
1	3	∞	∞	∞	Insert: 3.
1	3	6	∞	∞	Insert: 6.
1	2	6	∞	∞	Insert: 2.
1	2	5	∞	∞	Insert: 5.
1	2	5	7	∞	Insert: 7.

【经典题】最长上升子序列

```
const int N = 1e6 + 10;
int calcLIS(int * A, int n) {
    static int f[N];
    int len = 0;
    for (int i = 1; i <= n; ++i) {
        int at = std::lower_bound(f + 1, f + 1 + len, A[i]) - f;
        // 二分找到转移点 (即 at - 1)
        if (len < at) {
            // 加长序列，此时仍满足单调性
            f[at] = A[i];
            len = at;
        } else {
            // 更新序列，此时仍满足单调性
            f[at] = std::min(f[at], A[i]);
        }
    }
    return len;
}
```

【经典题】滑动窗口

Example (滑动窗口)

给你一个序列 A，以及一个整数 $K(1 \le K \le |A|)$。
对所有 $i \in [1, |A| - K + 1]$ 求出 $\min_{j \in [i, i+K)} A_j$。
$1 \le |A| \le 10^6$。

【经典题】滑动窗口

Example (滑动窗口)

给你一个序列 A，以及一个整数 $K(1 \leq K \leq |A|)$。
对所有 $i \in [1, |A| - K + 1]$ 求出 $\min_{j \in [i, i+K)} A_j$。
$1 \leq |A| \leq 10^6$。

使用扫描线，枚举右端点 r，先不管长度为 K 的询问的限制。
我们考虑 r 左边的元素的优劣。

【经典题】滑动窗口

Example (滑动窗口)

给你一个序列 A，以及一个整数 $K(1 \le K \le |A|)$。
对所有 $i \in [1, |A| - K + 1]$ 求出 $\min_{j \in [i, i+K)} A_j$。
$1 \le |A| \le 10^6$。

使用扫描线，枚举右端点 r，先不管长度为 K 的询问的限制。
我们考虑 r 左边的元素的优劣。
那么定义二元组 (i, A_i) 的偏序 $i \le j, A_i \ge A_j$，此时 i 一定不比 j 优秀。去掉这些不优的元素，得到一个 $i \le j, A_i \le A_j$ 的全序。

【经典题】滑动窗口

Example (滑动窗口)

给你一个序列 A，以及一个整数 $K(1 \le K \le |A|)$。
对所有 $i \in [1, |A| - K + 1]$ 求出 $\min_{j \in [i, i+K)} A_j$。
$1 \le |A| \le 10^6$。

使用扫描线，枚举右端点 r，先不管长度为 K 的询问的限制。

我们考虑 r 左边的元素的优劣。

那么定义二元组 (i, A_i) 的偏序 $i \le j, A_i \ge A_j$，此时 i 一定不比 j 优秀。去掉这些不优的元素，得到一个 $i \le j, A_i \le A_j$ 的全序。

因为我们是从右边插入的，每次维护单调性质时，弹出的也必定是最右边。此时用一个单调栈维护即可。

【经典题】滑动窗口

Example (滑动窗口)

给你一个序列 A，以及一个整数 $K(1 \le K \le |A|)$。
对所有 $i \in [1, |A| - K + 1]$ 求出 $\min_{j\in[i,i+K)} A_j$。
$1 \le |A| \le 10^6$。

使用扫描线，枚举右端点 r，先不管长度为 K 的询问的限制。

我们考虑 r 左边的元素的优劣。

那么定义二元组 (i, A_i) 的偏序 $i \le j, A_i \ge A_j$，此时 i 一定不比 j 优秀。去掉这些不优的元素，得到一个 $i \le j, A_i \le A_j$ 的全序。

因为我们是从右边插入的，每次维护单调性质时，弹出的也必定是最右边。此时用一个单调栈维护即可。

但是每次查询都需要二分左端点。考虑到我们所有询问按右端点排序后左端点是单调不降的，因此栈底的元素一旦失效就永远失效了。

【经典题】滑动窗口

Example (滑动窗口)

给你一个序列 A，以及一个整数 $K(1 \le K \le |A|)$。
对所有 $i \in [1, |A| - K + 1]$ 求出 $\min_{j \in [i, i+K)} A_j$。
$1 \le |A| \le 10^6$。

使用扫描线，枚举右端点 r，先不管长度为 K 的询问的限制。
我们考虑 r 左边的元素的优劣。
那么定义二元组 (i, A_i) 的偏序 $i \le j, A_i \ge A_j$，此时 i 一定不比 j 优秀。去掉这些不优的元素，得到一个 $i \le j, A_i \le A_j$ 的全序。
因为我们是从右边插入的，每次维护单调性质时，弹出的也必定是最右边。此时用一个单调栈维护即可。
但是每次查询都需要二分左端点。考虑到我们所有询问按右端点排序后左端点是单调不降的，因此栈底的元素一旦失效就永远失效了。
此时可以用一个双端单调队列，询问时弹出队头所有不再用到的元素，插入时弹出队尾会破坏单调性的元素即可。

【经典题】滑动窗口

1	4	3	6	2	5	7
1	4	3	6	2	5	7
1	4	3	6	2	5	7
1	4	3	6	2	5	7
1	4	3	6	2	5	7

【经典题】滑动窗口

```
const int N = 1e6 + 10;
void solve(int * A, int * ans, int n, int K) {
    std::deque<int> que;
    for (int i = 1; i <= n; ++i) {
        while (!que.empty() && A[que.back()] >= A[i])
            que.pop_back();
        // 维护队尾单调性
        que.push_back(i);
        if (i >= K) {
            while (que.front() < i - K + 1)
                que.pop_front();
            // 去掉不再可能用到的数
            ans[i] = A[que.front()];
        }
    }
}
```

【ARC081F】Flip and Rectangles

Example (Flip and Rectangles)

给你一个 $n \times m$ 的 01 矩阵 A，你可以任意取反一些行或列，求可能出现的面积最大的全 0 子矩形。

$1 \le n, m \le 2000$。

【ARC081F】Flip and Rectangles

Example (Flip and Rectangles)

给你一个 $n \times m$ 的 01 矩阵 A，你可以任意取反一些行或列，求可能出现的面积最大的全 0 子矩形。

$1 \leq n, m \leq 2000$。

很容易得到，如果你要搞出一个子矩形 B，我们考虑第一行，推出列取反的策略是固定的，此时行取反的策略也固定了。

【ARC081F】Flip and Rectangles

Example (Flip and Rectangles)

给你一个 $n \times m$ 的 01 矩阵 A，你可以任意取反一些行或列，求可能出现的面积最大的全 0 子矩形。

$1 \le n, m \le 2000$。

很容易得到，如果你要搞出一个子矩形 B，我们考虑第一行，推出列取反的策略是固定的，此时行取反的策略也固定了。

于是对于子矩形 B 中的每一行 i ，有 $B_{i,j} \oplus B_{1,j} = B_{i,j-1} \oplus B_{1,j-1}$

【ARC081F】Flip and Rectangles

Example (Flip and Rectangles)

给你一个 $n \times m$ 的 01 矩阵 A，你可以任意取反一些行或列，求可能出现的面积最大的全 0 子矩形。

$1 \leq n, m \leq 2000$。

很容易得到，如果你要搞出一个子矩形 B，我们考虑第一行，推出列取反的策略是固定的，此时行取反的策略也固定了。

于是对于子矩形 B 中的每一行 i，有 $B_{i,j} \oplus B_{1,j} = B_{i,j-1} \oplus B_{1,j-1}$

移项得到 $B_{i,j} \oplus B_{i,j-1} = B_{1,j} \oplus B_{1,j-1}$。

【ARC081F】Flip and Rectangles

Example (Flip and Rectangles)

给你一个 $n \times m$ 的 01 矩阵 A，你可以任意取反一些行或列，求可能出现的面积最大的全 0 子矩形。

$1 \leq n, m \leq 2000$。

很容易得到，如果你要搞出一个子矩形 B，我们考虑第一行，推出列取反的策略是固定的，此时行取反的策略也固定了。

于是对于子矩形 B 中的每一行 i ，有 $B_{i,j} \oplus B_{1,j} = B_{i,j-1} \oplus B_{1,j-1}$

移项得到 $B_{i,j} \oplus B_{i,j-1} = B_{1,j} \oplus B_{1,j-1}$。

特判掉一维长度是 1 的情况（答案是 $\max\{n, m\}$），剩下的可以通过上面的式子得到对于所有 $i, j > 1$，都有

$B_{i,j} \oplus B_{i,j-1} = B_{i-1,j} \oplus B_{i-1,j-1}$。

【ARC081F】Flip and Rectangles

Example (Flip and Rectangles)

给你一个 $n \times m$ 的 01 矩阵 A，你可以任意取反一些行或列，求可能出现的面积最大的全 0 子矩形。

$1 \leq n, m \leq 2000$。

很容易得到，如果你要搞出一个子矩形 B，我们考虑第一行，推出列取反的策略是固定的，此时行取反的策略也固定了。

于是对于子矩形 B 中的每一行 i ，有 $B_{i,j} \oplus B_{1,j} = B_{i,j-1} \oplus B_{1,j-1}$

移项得到 $B_{i,j} \oplus B_{i,j-1} = B_{1,j} \oplus B_{1,j-1}$。

特判掉一维长度是 1 的情况（答案是 $\max\{n, m\}$），剩下的可以通过上面的式子得到对于所有 $i, j > 1$，都有

$B_{i,j} \oplus B_{i,j-1} = B_{i-1,j} \oplus B_{i-1,j-1}$。

发现只和 i, j 相关，于是先预处理这个条件得到 $(n-1) \times (m-1)$ 的矩阵 C，转化成求最大子矩形。

【ARC081F】Flip and Rectangles

对于最大子矩形，我们可以先处理每个点向上最多延伸多长，然后枚举矩形的下边界，变成序列问题。

即求 $\max\limits_{1\le i\le j\le m}(\min\limits_{i\le k\le j} A_k+1)(j-i+1)$。

【ARC081F】Flip and Rectangles

对于最大子矩形，我们可以先处理每个点向上最多延伸多长，然后枚举矩形的下边界，变成序列问题。

即求 $\max_{1\le i\le j\le m}(\min_{i\le k\le j} A_k+1)(j-i+1)$。

使用扫描线。然后对于一个右端点有若干可取的左端点，当做暂存点，在他们被删除的时候更新答案。

【ARC081F】Flip and Rectangles

```
const int N = 2010;
int st[N], at[N], top, ans;
void pop(int x) {
    ans = std::max((st[top] + 1) * (x - at[top] + 1), ans);
    --top;
}
int calcRow(int * A, int n) {
    top = ans = 0;
    for (int i = 1; i <= n; ++i) {
        int t = A[i], an = i;
        while (t < st[top]) an = std::min(an, at[top]), pop(i);
        if (t != st[top]) st[++top] = t, at[top] = an;
    }
    while (top) pop(n);
    return ans;
}
```

【ARC081F】Flip and Rectangles

当然也有一个更好写、常数更大、本质相同的方法

【ARC081F】Flip and Rectangles

当然也有一个更好写、常数更大、本质相同的方法

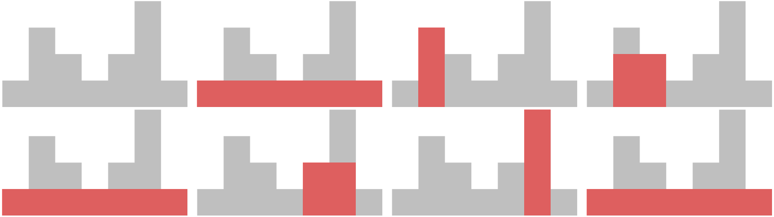
对于每一个元素，求出它最多能左右延伸多少。我们只需要求出右边第一个比它小的在哪，跑两遍就能知道一个位置的最大扩展区间。

【ARC081F】Flip and Rectangles

当然也有一个更好写、常数更大、本质相同的方法

对于每一个元素，求出它最多能左右延伸多少。我们只需要求出右边第一个比它小的在哪，跑两遍就能知道一个位置的最大扩展区间。

我们使用单调栈，从左到右插入元素，当一个元素被弹出的时候，我们就找到了这个元素的右端点。相比起来少了很多奇怪的讨论。



【ARC081F】Flip and Rectangles

```
const int N = 2010;
void calcRPoint(int * A, int * B, int n) {
    static int S[N]; int top = 0;
    for (int i = 1; i <= n; ++i) {
        while (top > 0 && A[S[top]] > A[i])
            B[S[top--]] = i - 1;
        S[++top] = i;
    }
    for (int i = 1; i <= top; ++i) B[S[i]] = n;
}
int calcRow(int * A ,int n) {
    static int L[N], R[N];
    calcRPoint(A, R, n), std::reverse(A + 1, A + 1 + n);
    calcRPoint(A, L, n), std::reverse(A + 1, A + 1 + n);
    int ans = 0;
    for (int i = 1, j = n; i <= n; ++i, --j)
        ans = std::max(ans, (A[i] + 1) * (R[i] - (n - L[j] + 1) + 2));
    return ans;
}
```

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 $ay - bx$。

$1 \le |P|, Q \le 10^6$。

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 $2S_{\Delta OXY}$。

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 $ay - bx$。

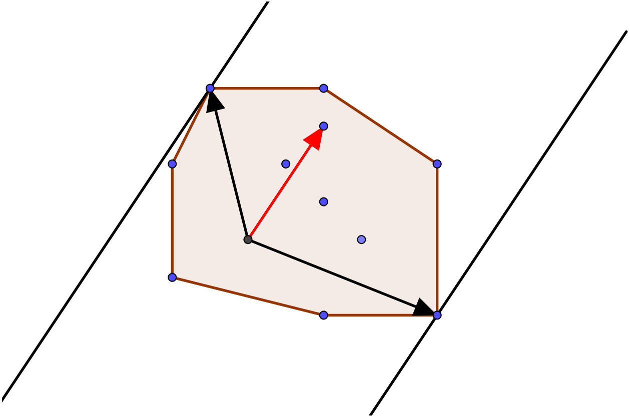
$1 \le |P|, Q \le 10^6$。

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 $ax + by$。

$1 \le |P|, Q \le 10^6$。

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.

$$
\begin{aligned}
\frac{b_y - a_y}{b_x - a_x} &\geq \frac{c_y - b_y}{c_x - b_x} \\
\text{let } v &= \frac{p_x}{p_y} \\
p_y \frac{(b_x - a_x)v + b_y - a_y}{b_x - a_x} &\geq p_y \frac{(c_x - b_x)v + c_y - b_y}{c_x - b_x} \\
\frac{b_x p_x + b_y p_y - a_x p_x - a_y p_y}{b_x - a_x} &\geq \frac{c_x p_x + c_y p_y - b_x p_x - b_y p_y}{c_x - b_x}
\end{aligned}
$$

□ □

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。

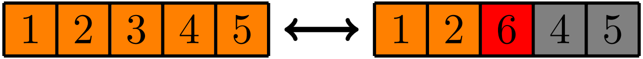
那么每次修改之前记下栈的 top，然后维护凸包的时候，并不真正地删除栈顶的点，而是改变 top 的指针。同时插入的时候，我们记录下插入的点替换了什么。

【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 $1 \dots n$，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。
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

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

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。

【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

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

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。

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。

$n \le 2 \times 10^5$。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 $\Phi_I\ge 2^k$ 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 $\Phi_D\ge 2^k$ 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 $R - V + 1 = L \leq l$，权值为

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

【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：

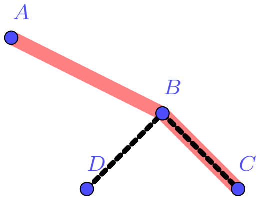
$$\max\{(A_i, S_i - iA_i) \cdot (L-1, 1)\}$$

【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。

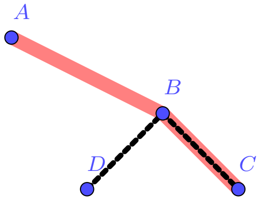


【CF455E】Function 加强版

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：

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

【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。

其他凸优化问题

凸优化不一定是点积最大化问题。对于凸的函数，可以利用一阶导数单调性来解题。

带权二分

在一类 xD/yD DP 中，若答案是 $f[n][m][k][\ldots]$，我们任意钦定一维，记 $g_x = f[x][m][k][\ldots]$。

如果 g 是一个凸函数，且可以很容易地计算 g 的最小值，以及取到最小值时的 x 时，那么一切就好办了。

因为凸函数的导数单调，则最值取在符号交错点处。

带权二分

在一类 xD/yD DP 中，若答案是 $f[n][m][k][\dots]$，我们任意钦定一维，记 $g_x = f[x][m][k][\dots]$。

如果 g 是一个凸函数，且可以很容易地计算 g 的最小值，以及取到最小值时的 x 时，那么一切就好办了。

因为凸函数的导数单调，则最值取在符号交错点处。

但是，我们要求的 n 处的导数并不一定在这个位置怎么办？

带权二分

在一类 xD/yD DP 中，若答案是 $f[n][m][k][\dots]$，我们任意钦定一维，记 $g_x = f[x][m][k][\dots]$。

如果 g 是一个凸函数，且可以很容易地计算 g 的最小值，以及取到最小值时的 x 时，那么一切就好办了。

因为凸函数的导数单调，则最值取在符号交错点处。

但是，我们要求的 n 处的导数并不一定在这个位置怎么办?

因为凸函数的和还是凸函数，我们给 DP 叠加一个一次函数 $y = kx$，即在 x 这一维增加 d 的时候，DP 值增加 dk。

带权二分

在一类 xD/yD DP 中，若答案是 $f[n][m][k][\dots]$，我们任意钦定一维，记 $g_x = f[x][m][k][\dots]$。

如果 g 是一个凸函数，且可以很容易地计算 g 的最小值，以及取到最小值时的 x 时，那么一切就好办了。

因为凸函数的导数单调，则最值取在符号交错点处。

但是，我们要求的 n 处的导数并不一定在这个位置怎么办？

因为凸函数的和还是凸函数，我们给 DP 叠加一个一次函数 $y = kx$，即在 x 这一维增加 d 的时候，DP 值增加 dk。

这样就相当于把原函数的导数平移了。我们二分这个 k，就能得到 g_x 除的值（只需要在最后去掉这个一次函数的贡献即可)。

【CF739E】Gosha is hunting

Example (Gosha is hunting)

有 n 个 pokemon，你有 A 个 Poke Ball 和 B 个 Ultra Ball。

对于每只 pokemon，Poke Ball 有 p_i 的概率抓住，Ultra Ball 有 q_i 的概率抓住。

你要分配球的使用，使期望最大，注意一个 pokemon 可以用多个球去抓，但是每种球每只最多丢一个。

保证 $A, B \le n \le 2000$。

【CF739E】Gosha is hunting

显然费用流建图完后就得到是一个凸函数，然后暴力 DP 即可。

【八省联考 2018】林克卡特树

Example (林克卡特树)

给一棵有正负权的 n 个点的树，你需要砍掉 k 条边，再连上 k 条 0 边。最大化直径。

保证 $n \leq 3 \times 10^5$。

【八省联考 2018】林克卡特树

既然是连 k 条边，则一定存在一种方案，选择 $k+1$ 条点不相交路径，然后再用 0 边连起来。

【八省联考 2018】林克卡特树

既然是连 k 条边，则一定存在一种方案，选择 $k+1$ 条点不相交路径，然后再用 0 边连起来。

显然选择前 k 大的链是凸的，可以费用流建图证明。注意可能会有退化成点的路径。暴力 DP 即可。

【SOJ973】我会彻查

Example (我会彻查)

你有 n 个数，你要指定一个 m，然后给出一个整数数组 $A_{m,n}$ 使得：1. $A_{m,n} > A_{m,n-1}$。2. $A_{m,1} > A_{m-1,n}$。3. $A_{m,n} < H$。4. $\sum A_{i,k} \leq B_k$。

最大化 A 的和，输出这个和。$n \leq 10^6, H \leq 10^9$。

【SOJ973】我会彻查

显然我们要榨干 B 数组。在没有任何性质的情况下，我们先枚举 m。

【SOJ973】我会彻查

显然我们要榨干 B 数组。在没有任何性质的情况下，我们先枚举 m。

我们可以把 A 拉成一个序列。但是下标从小到大枚举下标会导致难以确定当前的 $A_{i,j}$。

【SOJ973】我会彻查

显然我们要榨干 B 数组。在没有任何性质的情况下，我们先枚举 m。

我们可以把 A 拉成一个序列。但是下标从小到大枚举下标会导致难以确定当前的 $A_{i,j}$。

那么从大到小枚举下标。发现只要每次都取最紧的界，那么对后来的限制是最松的。显然这样是最大化的。

【SOJ973】我会彻查

显然我们要榨干 B 数组。在没有任何性质的情况下，我们先枚举 m。

我们可以把 A 拉成一个序列。但是下标从小到大枚举下标会导致难以确定当前的 $A_{i,j}$。

那么从大到小枚举下标。发现只要每次都取最紧的界，那么对后来的限制是最松的。显然这样是最大化的。

先处理一下最紧的下界 C_i，剩下要计算的只是增量。

从后往前扫，每个位置会有一个可行的增量上界。

【SOJ973】我会彻查

显然我们要榨干 B 数组。在没有任何性质的情况下，我们先枚举 m。

我们可以把 A 拉成一个序列。但是下标从小到大枚举下标会导致难以确定当前的 $A_{i,j}$。

那么从大到小枚举下标。发现只要每次都取最紧的界，那么对后来的限制是最松的。显然这样是最大化的。

先处理一下最紧的下界 C_i，剩下要计算的只是增量。

从后往前扫，每个位置会有一个可行的增量上界。

通过列式子容易发现，这个增量上界只是不断地对着 $B_i - C_i$ 取 $\min$。

很容易推出，在一段时间后，这个上界会变为 0。

很容易发现，当增量上界小于任何一个 $B - C$ 的时候，相当于全部减去 $B - C$。

【SOJ973】我会彻查

显然我们要榨干 B 数组。在没有任何性质的情况下，我们先枚举 m。

我们可以把 A 拉成一个序列。但是下标从小到大枚举下标会导致难以确定当前的 $A_{i,j}$。

那么从大到小枚举下标。发现只要每次都取最紧的界，那么对后来的限制是最松的。显然这样是最大化的。

先处理一下最紧的下界 C_i，剩下要计算的只是增量。

从后往前扫，每个位置会有一个可行的增量上界。

通过列式子容易发现，这个增量上界只是不断地对着 $B_i - C_i$ 取 $\min$。

很容易推出，在一段时间后，这个上界会变为 0。

很容易发现，当增量上界小于任何一个 $B - C$ 的时候，相当于全部减去 $B - C$。

则我们快速模拟这个过程，直到有一个小于 $B - C$，剩下的只需要 $O(n)$ 的暴力。

【SOJ973】我会彻查

这道题对于一定的 m 太难算，显然会去想凸性（因为其他的好像没太多用）。

打表容易发现是对的，但是现在我还是不会证这个凸性（对我来说太难了）

那我们就鸽了吧。

决策单调链

决策单调链是什么???（我瞎编的名字)
决策单调性大家是不是都听说过?
（总有一些魔鬼把斜率优化和决策单调性放在一起讲
（所以我是不是很良心?
但是这好难啊，所以我们下面讲三个东西吧。

最优决策点单调性

Definition (Optimal Decision Point)

对于一个矩阵 A，记一行的最优决策点 D_i 为：
最小的 j 使得 $A_{i,j}$ 最小。

最优决策点单调性

Definition (Optimal Decision Point Monotonicity)

对于一个矩阵 A，如果满足：
对于所有 $i < j$，则有 $D_i \leq D_j$。
则称 A 有最优决策点单调性，A 为单调矩阵 (Monotonic Matrix)。

决策单调性

Definition (Decision Monotonicity)

对于一个矩阵 A，如果满足：

对于任意一个 i 和两个下标 $x < y$，如果有 $A_{i,x} > A_{i,y}$，则对所有 $j > i$ 都有 $A_{j,x} > A_{j,y}$。

则称 A 有决策单调性，A 为完全单调矩阵 (Fully Monotonic Matrix)。

决策单调链第一定理

Theorem (DMC's 1st)

矩阵 A 是完全单调矩阵当且仅当 A 的任意一个子矩阵都是单调矩阵。

这是完全单调矩阵的另一种定义。

决策单调链第一定理

DMC's 1st Theorem.

- **充分性：**

由完全单调矩阵的定义，完全单调矩阵的所有子矩阵的都是一个完全单调矩阵。此时只需要证明任意一个完全单调矩阵都是单调矩阵。

考虑一列一列地向右填充元素，从上往下地填充。如果新加入的元素成了一整行唯一的最小值，才会更新这一行的 D。此时，在后面几行的 D 全部会更新，此时最优决策点单调性是保持着的。

决策单调链第一定理

DMC's 1st Theorem.

- **充分性：**

  由完全单调矩阵的定义，完全单调矩阵的所有子矩阵的都是一个完全单调矩阵。此时只需要证明任意一个完全单调矩阵都是单调矩阵。

  考虑一列一列地向右填充元素，从上往下地填充。如果新加入的元素成了一整行唯一的最小值，才会更新这一行的 D。此时，在后面几行的 D 全部会更新，此时最优决策点单调性是保持着的。

- **必要性：**

  假设存在一组下标 $x < y$，有 $A_{i,x} > A_{i,y}$ 但是存在一个 $j > i$ 使得 $A_{j,x} \le A_{j,y}$

  则子矩阵 $A[i,j;x,y]$ 不满足最优决策点单调性，矛盾。□ □

四边形不等式

Definition (Quadrangle Inequality)

对于一个 $n \times m$ 矩阵 A，如果满足：
对于所有 $1 \leq i < n, 1 \leq j < m$，有

$$a_{i+1,j+1} + a_{i,j} \leq a_{i+1,j} + a_{i,j+1}$$

则称方阵 A 满足四边形不等式，A 为凸矩阵 (Convex Matrix)。

四边形不等式

Definition (Quadrangle Inequality)

对于一个 $n \times m$ 矩阵 A，如果满足：
对于所有 $1 \leq i < n, 1 \leq j < m$，有

$$a_{i+1,j+1} + a_{i,j} \leq a_{i+1,j} + a_{i,j+1}$$

则称方阵 A 满足四边形不等式，A 为凸矩阵 (Convex Matrix)。

移项，可以得到两种形式：

$$a_{i,j} - a_{i+1,j} \leq a_{i,j+1} - a_{i+1,j+1}$$
$$a_{i,j} - a_{i,j+1} \leq a_{i+1,j} - a_{i+1,j+1}$$

然后就发现，这个实际上就是一个分量上斜率在另一分量上的单调性。

四边形不等式

Definition (Quadrangle Inequality)

对于一个 n 阶方阵 A，如果满足：
对于所有 $1 \le i_1 \le i_2 \le n, 1 \le j_1 \le j_2 \le m$，有

$$a_{i_1,j_1} + a_{i_2,j_2} \le a_{i_1,j_2} + a_{i_2,j_1}$$

则称方阵 A 满足四边形不等式，A 为凸矩阵 (Convex Matrix)。

四边形不等式

Definition (Quadrangle Inequality)

对于一个 n 阶方阵 A，如果满足：
对于所有 $1 \le i_1 \le i_2 \le n, 1 \le j_1 \le j_2 \le m$，有

$$a_{i_1,j_1} + a_{i_2,j_2} \le a_{i_1,j_2} + a_{i_2,j_1}$$

则称方阵 A 满足四边形不等式，A 为凸矩阵 (Convex Matrix)。

显然，用之前得到的差分的式子，首尾衔接可以得到这个形式与 ± 1 形式本质相同。

如何记住呢？对于 $\min$ 意义下的四边形不等式左边是中等大小，右边是一小一大。

决策单调链第二定理

Theorem (DMC's 2nd)

如果一个矩阵是凸矩阵，则这个矩阵是完全单调矩阵。

决策单调链第二定理

Theorem (DMC's 2nd)

如果一个矩阵是凸矩阵，则这个矩阵是完全单调矩阵。

DMC's 2nd Theorem.

设下标为 $x < y$，行标 $i < j$，若 $A_{i,x} > A_{i,y}$，则有：

$$A_{i,x} + A_{j,y} \le A_{i,y} + A_{j,x}$$

则

$$A_{i,x} - A_{i,y} \le A_{j,x} - A_{j,y}$$

显然 $A_{i,x} - A_{i,y} > 0$，则 $A_{j,x} - A_{j,y} > 0$，即 $A_{j,x} > A_{j,y}$。□ □

决策单调链

我们介绍完了这三个性质，它们从严格到宽松，形成了一条链：
四边形不等式 $\rightarrow$ 决策单调性 $\rightarrow$ 最优决策点单调性。
我们称它为决策单调链 (Decision Monotonicity Chain)。
这里面越严格的越好用，也越常见。
同时为了方便，我们把能用到决策单调链内容的问题统称为决策单调性问题 (Decision Monotonicity Problems)。

保凸原理

这个名字还是我编的。当然好像我们的结论还不够我们做题，所以下面还有一个十分简单的原理：

Principle (Convexity Preserving)

令 $g : R \to R$ 为任意一个函数，
若矩阵 W 为单调矩阵。则矩阵 $u_{i,j} = w_{i,j} + g(i)$ 也是单调矩阵。
若矩阵 W 为完全单调矩阵。则矩阵 $u_{i,j} = w_{i,j} + g(i)$ 也是完全单调矩阵。
若矩阵 W 为凸矩阵。则矩阵 $u_{i,j} = w_{i,j} + g(i)$ 也是凸矩阵。

保凸原理

Convexity Preserving Principle.

- **对于单调矩阵**

  每一行加上一个值后，最优决策点位置不变。

保凸原理

Convexity Preserving Principle.

- **对于单调矩阵**
  每一行加上一个值后，最优决策点位置不变。
- **对于完全单调矩阵**
  每一行加上一个值后，元素之间大小关系不变。

保凸原理

Convexity Preserving Principle.

- **对于单调矩阵**
  每一行加上一个值后，最优决策点位置不变。
- **对于完全单调矩阵**
  每一行加上一个值后，元素之间大小关系不变。
- **对于凸矩阵**
  令下标 $1 \le a, b \le n, 1 \le x, y \le m$，则有

$$
\begin{aligned}
u_{a,x} + u_{b,y} &= w_{a,x} + w_{b,y} + g(a) + g(b) \\
u_{a,y} + u_{b,x} &= w_{a,y} + w_{b,x} + g(b) + g(a)
\end{aligned}
$$

显然在两项比较之后，后面的 g 消掉了，矩阵 U 的凸性和 W 的完全一样。□

□

保凸原理

虽然我们干了一件像是证明 $1 = 1$ 的事，但是这正是我们做这类 DP 的正确性保证：

$$f_{i,j} = \min_k \{g(f_{i-1,k}) + w_{j,k}\}$$

也就是在我们做多次 DP 仍能保证决策单调链的证据来源。

保凸原理

虽然我们干了一件像是证明 $1 = 1$ 的事，但是这正是我们做这类 DP 的正确性保证：

$$f_{i,j} = \min_{k} \{g(f_{i-1,k}) + w_{j,k}\}$$

也就是在我们做多次 DP 仍能保证决策单调链的证据来源。
那对于下面这个全在线的 DP 决策单调性问题呢？

$$f_i = \min_{j<i} \{g(f_j) + w_{i,j}\}$$

保凸原理

虽然我们干了一件像是证明 $1=1$ 的事，但是这正是我们做这类 DP 的正确性保证：

$$f_{i,j}=\min_{k}\{g(f_{i-1,k})+w_{j,k}\}$$

也就是在我们做多次 DP 仍能保证决策单调链的证据来源。
那对于下面这个全在线的 DP 决策单调性问题呢？

$$f_i=\min_{j<i}\{g(f_j)+w_{i,j}\}$$

我们对于每个端点 i，则每个端点看做一个权值矩阵的阶段，即给权值叠加了 $g(f_j)$ 的列向量。因此还是满足保凸原理的。

划分凸定理

这个名字也是我编的。

四边形不等式很多时候应用在序列划分问题，即找 k 个分界点，最小化权值和，即：

$$f_{k,i} = \min_{j<i} \{f_{k-1,j} + w_{i,j}\}$$

我们有一个很好的结论：

Theorem (Convexity of Partition)

对于一个序列划分 *DP*，设 $g(x) = f_{x,n}$，则有：
序列 g 是下凸的，即 $g(i) - g(i-1) \le g(i+1) - g(i)$。

划分凸定理

Convexity of Partition Theorem.

考虑 $g(i-1)$ 的最优解方案为

$$[1, x_1), [x_1, x_2), \ldots, [x_{i-2}, x_{i-1})$$

考虑 $g(i+1)$ 的最优解方案为

$$[1, y_1), [y_1, y_2), \ldots, [y_i, y_{i+1})$$

划分凸定理

Convexity of Partition Theorem.

考虑 $g(i-1)$ 的最优解方案为

$$[1,x_1),[x_1,x_2),\dots,[x_{i-2},x_{i-1})$$

考虑 $g(i+1)$ 的最优解方案为

$$[1,y_1),[y_1,y_2),\dots,[y_i,y_{i+1})$$

由鸽子原理，得到一定存在一组下标使得

$$x_i \le y_j < y_{j+1} \le x_{i+1}(j=i+1)$$

划分凸定理

Convexity of Partition Theorem.

考虑 $g(i-1)$ 的最优解方案为

$$[1, x_1), [x_1, x_2), \ldots, [x_{i-2}, x_{i-1})$$

考虑 $g(i+1)$ 的最优解方案为

$$[1, y_1), [y_1, y_2), \ldots, [y_i, y_{i+1})$$

由鸽子原理，得到一定存在一组下标使得

$$x_i \leq y_j < y_{j+1} \leq x_{i+1} (j = i+1)$$

考虑 $pre(y, j) + suc(x, i+1)$ 和 $pre(x, i) + suc(y, j+1)$，我们交换了两个部分，得到一个新的方案，这两个方案都是长度为 i 的合法划分。 □

划分凸定理

Convexity of Partition Theorem.

考虑交换后的权值变化：

$$\Delta = w_{y_j,x_{i+1}} + w_{x_i,y_{j+1}} - w_{y_j,y_{j+1}} - w_{x_i,x_{i+1}}$$

划分凸定理

Convexity of Partition Theorem.

考虑交换后的权值变化：

$$\Delta = w_{y_j,x_{i+1}} + w_{x_i,y_{j+1}} - w_{y_j,y_{j+1}} - w_{x_i,x_{i+1}}$$

根据四边形不等式，我们交换后的权值和

$$v = \Delta + g(i-1) + g(i+1) \leq g(i-1) + g(i+1)$$

划分凸定理

Convexity of Partition Theorem.

考虑交换后的权值变化：

$$\Delta = w_{y_j,x_{i+1}} + w_{x_i,y_{j+1}} - w_{y_j,y_{j+1}} - w_{x_i,x_{i+1}}$$

根据四边形不等式，我们交换后的权值和

$$v = \Delta + g(i-1) + g(i+1) \leq g(i-1) + g(i+1)$$

因为 $g(i)$ 是最优解，所以有

$$g(i) + g(i) \leq v \leq g(i-1) + g(i+1)$$

□ □

【经典题】石子合并

Example (石子合并)

序列上有 n 堆石子，有各自重量 w_i。每次你可以合并相邻两堆，代价为重量之和。

求合并成一堆的最小代价和。

$n \le 5000$，$1 \le w_i \le 10^9$。

【经典题】石子合并

Example (石子合并)

序列上有 n 堆石子，有各自重量 w_i。每次你可以合并相邻两堆，代价为重量之和。

求合并成一堆的最小代价和。

$n \le 5000$，$1 \le w_i \le 10^9$。

……这不是区间 DP 吗？但是背过百万结论的大家一定都知道这个有最优决策点单调性。大家也一定都知道有 $O(n \log n)$ 做法，但是我不会……而且和这节课没什么关系。

所以这道就当做决策单调性问题的第一道例题了吧。

【经典题】石子合并

我们定义 $f'_{i,j}=f_{j,i}$，当 $j\geq i$ 时，$w_{j,i}=f_{j,i}=\infty$。
则实际上转移式是：

$$f_{i,j}=\min_{k}\left\{f_{i,k}+f'_{j,k}\right\}+w_{i,j}$$

【经典题】石子合并

我们定义 $f'_{i,j}=f_{j,i}$，当 $j\geq i$ 时，$w_{j,i}=f_{j,i}=\infty$。
则实际上转移式是：

$$f_{i,j}=\min_{k}\left\{f_{i,k}+f'_{j,k}\right\}+w_{i,j}$$

显然 w 有四边形不等式，我们要证明 f 的四边形不等式，这样 f 的转置 f' 自然也有了。

【经典题】石子合并

我们定义 $f'_{i,j} = f_{j,i}$，当 $j \geq i$ 时，$w_{j,i} = f_{j,i} = \infty$。
则实际上转移式是：

$$f_{i,j} = \min_k \left\{ f_{i,k} + f'_{j,k} \right\} + w_{i,j}$$

显然 w 有四边形不等式，我们要证明 f 的四边形不等式，这样 f 的转置 f' 自然也有了。
首先，如果不等式中任意一处带有 ∞，显然是符合的。
接下来是归纳法证明 $f_{i,j} + f_{i+1,j+1} \leq f_{i,j+1} + f_{i+1,j}$。

【经典题】石子合并

2D/1D DP 值的凸性.

显然 $L=1$ 成立。假设长度为 L 的都已经证明，现在证 $L+1$：

$$f_{i,j}+f_{i+1,j+1}\le f_{i,j+1}+f_{i+1,j}$$

假设 $f_{i,j+1}$ 的最优决策点为 x，$f_{i+1,j}$ 的最优决策点为 y，则：

$$f_{i,j}+f_{i+1,j+1}\le f_{i,x}+f_{x,j}+w_{i,j}+f_{i+1,x}+f_{x,j+1}+w_{i+1,j+1}$$
$$f_{i,j}+f_{i+1,j+1}\le f_{i,y}+f_{y,j}+w_{i,j}+f_{i+1,y}+f_{y,j+1}+w_{i+1,j+1}$$

□

【经典题】石子合并

2D/1D DP 值的凸性.

$$
\begin{aligned}
f_{i,j} + f_{i+1,j+1} &\le f_{i,x} + f_{x,j} + w_{i,j} + f_{i+1,x} + f_{x,j+1} + w_{i+1,j+1} \\
f_{i,j} + f_{i+1,j+1} &\le f_{i,y} + f_{y,j} + w_{i,j} + f_{i+1,y} + f_{y,j+1} + w_{i+1,j+1} \\
&f_{i,x} + f_{x,j} + f_{i+1,x} + f_{x,j+1} + \\
&f_{i,y} + f_{y,j} + f_{i+1,y} + f_{y,j+1} + \\
2w_{i,j} + 2w_{i+1,j+1} &\le 2w_{i,j+1} + 2w_{i+1,j} \\
&+ f_{i,x} + f_{x,j+1} + f_{i+1,y} + f_{y,j} \\
&+ f_{i,x} + f_{x,j+1} + f_{i+1,y} + f_{y,j}
\end{aligned}
$$

其中，第三行是四边形不等式，第四行是照抄，第五行是最优决策点的对应。□ □

【经典题】石子合并

证明完了 f 的四边形不等式，再来看看转移式：

$$f_{i,j} = \min_{k} \left\{ f_{i,k} + f'_{j,k} \right\} + w_{i,j}$$

【经典题】石子合并

证明完了 f 的四边形不等式，再来看看转移式：

$$f_{i,j} = \min_{k} \left\{ f_{i,k} + f'_{j,k} \right\} + w_{i,j}$$

固定一个 i，相当于 f 的第 i 行的行向量和 f' 叠加，得到凸矩阵，所以有 $D_{i,j} \leq D_{i,j+1}$。

【经典题】石子合并

证明完了 f 的四边形不等式，再来看看转移式：

$$f_{i,j} = \min_{k} \left\{ f_{i,k} + f'_{j,k} \right\} + w_{i,j}$$

固定一个 i，相当于 f 的第 i 行的行向量和 f' 叠加，得到凸矩阵，所以有 $D_{i,j} \leq D_{i,j+1}$。

同理，也有 $D_{i-1,j} \leq D_{i,j}$。

【经典题】石子合并

证明完了 f 的四边形不等式，再来看看转移式：

$$f_{i,j}=\min_{k}\left\{f_{i,k}+f'_{j,k}\right\}+w_{i,j}$$

固定一个 i，相当于 f 的第 i 行的行向量和 f' 叠加，得到凸矩阵，所以有 $D_{i,j}\le D_{i,j+1}$。

同理，也有 $D_{i-1,j}\le D_{i,j}$。

放在一起，就是 2D/1D DP 的最优决策点单调性：

$$D_{i-1,j}\le D_{i,j}\le D_{i,j+1}$$

【经典题】石子合并

证明完了 f 的四边形不等式，再来看看转移式：

$$f_{i,j} = \min_{k} \left\{ f_{i,k} + f'_{j,k} \right\} + w_{i,j}$$

固定一个 i，相当于 f 的第 i 行的行向量和 f' 叠加，得到凸矩阵，所以有 $D_{i,j} \le D_{i,j+1}$。

同理，也有 $D_{i-1,j} \le D_{i,j}$。

放在一起，就是 2D/1D DP 的最优决策点单调性：

$$D_{i-1,j} \le D_{i,j} \le D_{i,j+1}$$

但是这是大区间夹小区间，看起来不太实用，化一下式子，得到：

$$D_{i,j-1} \le D_{i,j} \le D_{i+1,j}$$

【经典题】石子合并

$$D_{i,j-1} \le D_{i,j} \le D_{i+1,j}$$

因为长度 L 的决策点是单调的，然后对长度 $L+1$ 进行了夹逼，所以每一层需要枚举的决策点只有 $O(n)$。

直接优化复杂度就是 $O(n^2)$。

【NAIPC2016】Jewel Thief

Example (Jewel Thief)

有 n 个物品，每个物品有一个体积 w_i 和价值 v_i，现在要求对 $V \in [1, m]$，求出体积为 V 的背包能够装下的最大价值。

保证 $n \le 10^6, m \le 10^5, 1 \le w_i \le 300, 1 \le v_i \le 10^9$

【NAIPC2016】Jewel Thief

Example (Jewel Thief)

有 n 个物品，每个物品有一个体积 w_i 和价值 v_i，现在要求对 $V \in [1, m]$，求出体积为 V 的背包能够装下的最大价值。

保证 $n \le 10^6, m \le 10^5, 1 \le w_i \le 300, 1 \le v_i \le 10^9$

这会儿我真没说啥. By ignore2004

【NAIPC2016】Jewel Thief

看到 w_i 很小，显然枚举体积，然后模意义分组。剩下的转移数组 $w_{i,j}$ 是前 $j-i$ 大的和。

【NAIPC2016】Jewel Thief

看到 w_i 很小，显然枚举体积，然后模意义分组。剩下的转移数组 $w_{i,j}$ 是前 $j-i$ 大的和。

显然证明四边形不等式。因为是取最大值，所以我们的四边形不等式要反向。

【NAIPC2016】Jewel Thief

看到 w_i 很小，显然枚举体积，然后模意义分组。剩下的转移数组 $w_{i,j}$ 是前 $j-i$ 大的和。

显然证明四边形不等式。因为是取最大值，所以我们的四边形不等式要反向。

然后四边形不等式一列，消掉相同元素后，剩下两种元素 $A \geq B$，然后就是证明 $A + A \geq A + B$。

【NAIPC2016】Jewel Thief

看到 w_i 很小，显然枚举体积，然后模意义分组。剩下的转移数组 $w_{i,j}$ 是前 $j-i$ 大的和。

显然证明四边形不等式。因为是取最大值，所以我们的四边形不等式要反向。

然后四边形不等式一列，消掉相同元素后，剩下两种元素 $A \geq B$，然后就是证明 $A+A \geq A+B$。

这不是显然吗？但是，我们还想知道怎么利用整个决策单调链。

$m \leq 10^5$，提示我们使用不高于 $O(n \log n)$ 的做法。

【NAIPC2016】Jewel Thief

考虑分治。我们算出中间点的决策点后，夹逼左右端点的决策点，这样递归下去。每层的区间不交，所以计算次数是 $O(n \log n)$。

```
int getd(int u, int l, int r) {
    int at = l;
    for (int i = l; i <= r; ++i)
        if (calc(u, i) < calc(u, at))
            at = i;
    return at;
}
void solve(int l, int r, int dl, int dr) {
    int mid = l + r >> 1, at = getd(mid, dl, dr);
    f[mid] = calc(mid, at);
    if (l < mid) solve(l, mid - 1, dl, at);
    if (mid < r) solve(mid + 1, r, at, dr);
}
```

【NAIPC2016】Jewel Thief

说到夹逼，可能会有另一个想法：算出一个点左右两个点，那就可以快速算出它的了。

那么如果我们奇偶分类，每次提取出下标为奇数的点，算出它们的决策点，那么每次都可以夹逼了。

$T(x) = T(x/2) + O(n)$，则复杂度 $O(n\log n)$。

【NAIPC2016】Jewel Thief

当然，为什么一定要夹逼呢？我们好像还没对决策单调性做过实际应用。

【NAIPC2016】Jewel Thief

当然，为什么一定要夹逼呢？我们好像还没对决策单调性做过实际应用。

在第一层，我们可以得到一个单调序列：$A_i < A_j, w_{1,A_i} \le w_{1,A_j}$。

【NAIPC2016】Jewel Thief

当然，为什么一定要夹逼呢？我们好像还没对决策单调性做过实际应用。

在第一层，我们可以得到一个单调序列：$A_i < A_j, w_{1,A_i} \le w_{1,A_j}$。

由最优决策点单调性，得到在 A_i 取到最优的 x 集合是一个区间，称为管辖区间。

【NAIPC2016】Jewel Thief

当然，为什么一定要夹逼呢？我们好像还没对决策单调性做过实际应用。

在第一层，我们可以得到一个单调序列：$A_i < A_j, w_{1,A_i} \le w_{1,A_j}$。

由最优决策点单调性，得到在 A_i 取到最优的 x 集合是一个区间，称为管辖区间。

那么我们就同时维护区间的单调队列，弹出不再有用的，同时维护管辖区间的单调性即可。

【NAIPC2016】Jewel Thief

当然，为什么一定要夹逼呢？我们好像还没对决策单调性做过实际应用。

在第一层，我们可以得到一个单调序列：$A_i < A_j, w_{1,A_i} \le w_{1,A_j}$。

由最优决策点单调性，得到在 A_i 取到最优的 x 集合是一个区间，称为管辖区间。

那么我们就同时维护区间的单调队列，弹出不再有用的，同时维护管辖区间的单调性即可。

这个做法有一个好处，就是可以做一些全在线的决策单调性问题！

【北大集训 2018】游览计划

Example (游览计划)

你有一个排列 P，你需要把它分为 K 个连续段，每个连续段的权值为这个连续段里所有点并上 1 形成的虚树的边数。

求最大权值和。保证 $|P|K \leq 2 \times 10^5$。

【北大集训 2018】游览计划

Example (游览计划)

你有一个排列 P，你需要把它分为 K 个连续段，每个连续段的权值为这个连续段里所有点并上 1 形成的虚树的边数。

求最大权值和。保证 $|P|K \leq 2 \times 10^5$。

啥？你说用 veb ??

Obviously you can use 压位-trie to solve it. By ignore2004

【北大集训 2018】游览计划

我们列出 DP 后，一个显然的想法，是直接去四边形不等式打表。然后发现真的有！

$$w_{r,l} + w_{r+1,l+1} \geq w_{r,l+1} + w_{r+1,l}$$

【北大集训 2018】游览计划

我们列出 DP 后，一个显然的想法，是直接去四边形不等式打表。然后发现真的有！

$$w_{r,l} + w_{r+1,l+1} \geq w_{r,l+1} + w_{r+1,l}$$

像证明前 k 大一样，考虑在 $w_{r,l+1}$ 基础上的增量。

只需要考虑 $l, r+1$ 分别都是树上新的分支的情况。如果一起插入，新的分支可能出现重叠，权值不比分别插入大。

【北大集训 2018】游览计划

我们列出 DP 后，一个显然的想法，是直接去四边形不等式打表。然后发现真的有！

$$w_{r,l}+w_{r+1,l+1}\geq w_{r,l+1}+w_{r+1,l}$$

像证明前 k 大一样，考虑在 $w_{r,l+1}$ 基础上的增量。

只需要考虑 $l,r+1$ 分别都是树上新的分支的情况。如果一起插入，新的分支可能出现重叠，权值不比分别插入大。

证完四边形不等式，保凸原理保证了多次 DP 的正确性。

【北大集训 2018】游览计划

再加上 $O(n\log n)$ 的决策点单调性的工具，就可以过掉了。过掉了??

【北大集训 2018】游览计划

再加上 $O(n\log n)$ 的决策点单调性的工具，就可以过掉了。过掉了??

但是对于计算区间虚树，我们没有特别好的办法，只能指针移动，每次插入和删除要查询前去后继，即单次 $O(\log\log n)$。

【北大集训 2018】游览计划

再加上 $O(n\log n)$ 的决策点单调性的工具，就可以过掉了。过掉了??

但是对于计算区间虚树，我们没有特别好的办法，只能指针移动，每次插入和删除要查询前去后继，即单次 $O(\log\log n)$。

再看看我们之前的三种做法：单调队列跨度太大不能用了，奇偶分类可以双指针，那分治呢？

【北大集训 2018】游览计划

再加上 $O(n\log n)$ 的决策点单调性的工具，就可以过掉了。过掉了??

但是对于计算区间虚树，我们没有特别好的办法，只能指针移动，每次插入和删除要查询前去后继，即单次 $O(\log\log n)$。

再看看我们之前的三种做法：单调队列跨度太大不能用了，奇偶分类可以双指针，那分治呢？

貌似直接 dfs 不是那么显然，那么换成 bfs 吧！

【北大集训 2018】游览计划

再加上 $O(n\log n)$ 的决策点单调性的工具，就可以过掉了。过掉了??

但是对于计算区间虚树，我们没有特别好的办法，只能指针移动，每次插入和删除要查询前去后继，即单次 $O(\log\log n)$。

再看看我们之前的三种做法：单调队列跨度太大不能用了，奇偶分类可以双指针，那分治呢？

貌似直接 dfs 不是那么显然，那么换成 bfs 吧！

实际上 dfs 的暴力移动指针是对的。分别考虑左右指针。

【北大集训 2018】游览计划

再加上 $O(n\log n)$ 的决策点单调性的工具，就可以过掉了。过掉了??

但是对于计算区间虚树，我们没有特别好的办法，只能指针移动，每次插入和删除要查询前去后继，即单次 $O(\log\log n)$。

再看看我们之前的三种做法：单调队列跨度太大不能用了，奇偶分类可以双指针，那分治呢？

貌似直接 dfs 不是那么显然，那么换成 bfs 吧！

实际上 dfs 的暴力移动指针是对的。分别考虑左右指针。

右指针只能是区间中点。考虑在分治树上父亲处计算跨子树移动的上界。

显然算完中点后去左区间中点，遍历左区间后从左区间右端点到右区间中点，总共当前区间一半长度。

左指针在右指针的基础上，只多了遍历整个区间。所以左右指针的上界都是 $O(n\log n)$。

【北大集训 2018】游览计划

于是这道题就做完了，复杂度 $O(kn\log n\log\log n)$。

【北大集训 2018】游览计划

于是这道题就做完了，复杂度 $O(kn\log n\log\log n)$。

当然，由划分凸定理，我们可以使用凸优化。

剩下的问题就是一个全在线决策单调性问题，由上面的经验，我们使用分治套分治，单次复杂度 $O(n\log^2 n\log\log n)$。

总复杂度 $O(n\log^3 n\log\log n)$。

【北大集训 2018】游览计划

于是这道题就做完了，复杂度 $O(kn\log n\log\log n)$。

当然，由划分凸定理，我们可以使用凸优化。

剩下的问题就是一个全在线决策单调性问题，由上面的经验，我们使用分治套分治，单次复杂度 $O(n\log^2 n\log\log n)$。

总复杂度 $O(n\log^3 n\log\log n)$。

当然我们可以做到 $O(n\log^2 n)$ 预处理，用单调队列 $O(n\log^2 n)$ 单次 DP，总复杂度 $O(n\log^3 n)$。

【北大集训 2018】游览计划

于是这道题就做完了，复杂度 $O(kn\log n\log\log n)$。

当然，由划分凸定理，我们可以使用凸优化。

剩下的问题就是一个全在线决策单调性问题，由上面的经验，我们使用分治套分治，单次复杂度 $O(n\log^2 n\log\log n)$。

总复杂度 $O(n\log^3 n\log\log n)$。

当然我们可以做到 $O(n\log^2 n)$ 预处理，用单调队列 $O(n\log^2 n)$ 单次 DP，总复杂度 $O(n\log^3 n)$。

当然后面还有一种做法（我不会），可以把 DP 优化到 $O(n\log n)$，总复杂度 $O(n\log^2 n)$。

【NOI2016】国王饮水记

Example (国王饮水记)

有 n 个相同的容器，标号为 $1 \dots n$。一开始每个容器都有一个水深度。初始水深度两两不同。

你每次可以选择若干个容器，使得里面的水变为深度平均值。你最多可以这样操作 k 次。

最大化容器 1 最终的深度。

你需要输出至多 $2P$ 位小数，checker 精度为 10^{-P}。

$n \leq 8000, k \leq 10^9, P \leq 3000$。

【NOI2016】国王饮水记

显然，容器 1 的深度是不降的。所以我们可以先将容器排序，深度比 1 低的去掉。

【NOI2016】国王饮水记

显然，容器 1 的深度是不降的。所以我们可以先将容器排序，深度比 1 低的去掉。

我们将所有和 1 一起操作后的容器拿出。显然这些容器每个都不可能用超过一次。

【NOI2016】国王饮水记

显然，容器 1 的深度是不降的。所以我们可以先将容器排序，深度比 1 低的去掉。

我们将所有和 1 一起操作后的容器拿出。显然这些容器每个都不可能用超过一次。

那么，能够证明每次选择的容器集合下标不会交错（单调）。因为如果有两个集合交错，调整法可以得到更优解。例如 S 和 T 元素混合排序后，前 $|S|$ 个给 S'，根据元素贡献这样子更优。

【NOI2016】国王饮水记

显然，容器 1 的深度是不降的。所以我们可以先将容器排序，深度比 1 低的去掉。

我们将所有和 1 一起操作后的容器拿出。显然这些容器每个都不可能用超过一次。

那么，能够证明每次选择的容器集合下标不会交错（单调）。因为如果有两个集合交错，调整法可以得到更优解。例如 S 和 T 元素混合排序后，前 $|S|$ 个给 S'，根据元素贡献这样子更优。

同时，也不会有元素不用。记最右边用过的容器为 x，则 1 深度小于等于 x。则后面的都可以用。若三个相邻区间 x, y, z，用过 x, z，没用 y，则显然用完 x 后再用 y 更优。

【NOI2016】国王饮水记

显然，容器 1 的深度是不降的。所以我们可以先将容器排序，深度比 1 低的去掉。

我们将所有和 1 一起操作后的容器拿出。显然这些容器每个都不可能用超过一次。

那么，能够证明每次选择的容器集合下标不会交错（单调）。因为如果有两个集合交错，调整法可以得到更优解。例如 S 和 T 元素混合排序后，前 $|S|$ 个给 S'，根据元素贡献这样子更优。

同时，也不会有元素不用。记最右边用过的容器为 x，则 1 深度小于等于 x。则后面的都可以用。若三个相邻区间 x, y, z，用过 x, z，没用 y，则显然用完 x 后再用 y 更优。

显然每次使用都要带上容器 1。如果存在一个操作，不带 1，那么匀了之后，前面的贡献系数不大于后面的，所以匀过去答案不会更优。

【NOI2016】国王饮水记

显然，容器 1 的深度是不降的。所以我们可以先将容器排序，深度比 1 低的去掉。

我们将所有和 1 一起操作后的容器拿出。显然这些容器每个都不可能用超过一次。

那么，能够证明每次选择的容器集合下标不会交错（单调）。因为如果有两个集合交错，调整法可以得到更优解。例如 S 和 T 元素混合排序后，前 $|S|$ 个给 S'，根据元素贡献这样子更优。

同时，也不会有元素不用。记最右边用过的容器为 x，则 1 深度小于等于 x。则后面的都可以用。若三个相邻区间 x, y, z，用过 x, z，没用 y，则显然用完 x 后再用 y 更优。

显然每次使用都要带上容器 1。如果存在一个操作，不带 1，那么匀了之后，前面的贡献系数不大于后面的，所以匀过去答案不会更优。

同理，每次选取的区间长度也不会是递增的，否则也可以通过调整得到更优。

【NOI2016】国王饮水记

所以我们证明了答案是选择一堆分割点，然后从左到右把区间喂给 1。那么方程显然。

$$f_{i,j}=\max_{k<j}\left\{\frac{f_{i-1,k}+S_j-S_k}{j-k+1}\right\}$$

【NOI2016】国王饮水记

所以我们证明了答案是选择一堆分割点，然后从左到右把区间喂给 1。那么方程显然。

$$f_{i,j} = \max_{k<j}\left\{\frac{f_{i-1,k} + S_j - S_k}{j - k + 1}\right\}$$

一看，竟然不是点积最大化???
原来这是斜率最大化，还是可以在凸包上二分。
于是现在有了一个 $O(nKP\log n)$ 的做法。

【NOI2016】国王饮水记

所以我们证明了答案是选择一堆分割点，然后从左到右把区间喂给 1。那么方程显然。

$$f_{i,j} = \max_{k<j}\left\{\frac{f_{i-1,k} + S_j - S_k}{j - k + 1}\right\}$$

一看，竟然不是点积最大化???
原来这是斜率最大化，还是可以在凸包上二分。
于是现在有了一个 $O(nKP\log n)$ 的做法。
但是打表一打（大胆猜想），还有最优决策点单调性???
那我们尝试从四边形不等式开始证。

【NOI2016】国王饮水记

所以我们证明了答案是选择一堆分割点，然后从左到右把区间喂给 1。那么方程显然。

$$f_{i,j}=\max_{k<j}\left\{\frac{f_{i-1,k}+S_j-S_k}{j-k+1}\right\}$$

一看，竟然不是点积最大化???
原来这是斜率最大化，还是可以在凸包上二分。
于是现在有了一个 $O(nKP\log n)$ 的做法。
但是打表一打（大胆猜想），还有最优决策点单调性???
那我们尝试从四边形不等式开始证。项好多……好难证……

【NOI2016】国王饮水记

所以我们证明了答案是选择一堆分割点，然后从左到右把区间喂给1。那么方程显然。

$$f_{i,j} = \max_{k<j} \left\{ \frac{f_{i-1,k} + S_j - S_k}{j - k + 1} \right\}$$

一看，竟然不是点积最大化???
原来这是斜率最大化，还是可以在凸包上二分。
于是现在有了一个 $O(nKP \log n)$ 的做法。
但是打表一打（大胆猜想），还有最优决策点单调性???
那我们尝试从四边形不等式开始证。项好多……好难证……
那就尝试决策单调性吧，毕竟只需要最有决策点单调。

【NOI2016】国王饮水记

We have $\dfrac{f_{i-1,k}+S_j-S_k}{j-k+1} < \dfrac{f_{i-1,l}+S_j-S_l}{j-l+1}(k<l)$

$$(j-l+1)(f_{i-1,k}+S_j-S_k) < (j-k+1)(f_{i-1,l}+S_j-S_l)$$

【NOI2016】国王饮水记

We have $\frac{f_{i-1,k}+S_j-S_k}{j-k+1} < \frac{f_{i-1,l}+S_j-S_l}{j-l+1}(k<l)$

$(j-l+1)(f_{i-1,k}+S_j-S_k) < (j-k+1)(f_{i-1,l}+S_j-S_l)$

考虑 $((j+1)-l+1)(f_{i-1,k}+S_{j+1}-S_k)$ 的增量

$$
\begin{aligned}
&(S_{j+1}-S_j)(j-k+1)+f_{i-1,l}+S_{j+1}-S_l- \\
&((S_{j+1}-S_j)(j-l+1)+f_{i-1,k}+S_{j+1}-S_k) \\
=&(S_{j+1}-S_j)(l-k)-(S_l-S_k)+f_{i-1,l}-f_{i-1,k}
\end{aligned}
$$

$\square$

【NOI2016】国王饮水记

$$(S_{j+1}-S_j)(l-k)-(S_l-S_k)+f_{i-1,l}-f_{i-1,k}$$

显然有：

- $f_{i-1,l} \geq f_{i-1,k}$。
- $(S_{j+1}-S_j)(l-k)-(S_l-S_k)$。

将增量累加到第二行的不等式，得到 $j+1$ 形式的不等式。□ □

于是现在有一个 $O(nKP)$ 的做法了，但是还是不足以通过。

【NOI2016】国王饮水记

似乎这个区间想要不增的条件有点苛刻，打表发现，我们区间划分到后来会变为一堆长度为 1 的。

【NOI2016】国王饮水记

似乎这个区间想要不增的条件有点苛刻，打表发现，我们区间划分到后来会变为一堆长度为 1 的。

还记得之前推出过的区间长度不增的性质吗？经过一系列推导（这回我也不会），然后发现前面最多有 14 段 >1 的。得到一个 $O(14nP)$ 的做法，还是过不了。

【NOI2016】国王饮水记

似乎这个区间想要不增的条件有点苛刻，打表发现，我们区间划分到后来会变为一堆长度为 1 的。

还记得之前推出过的区间长度不增的性质吗？经过一系列推导（这回我也不会），然后发现前面最多有 14 段 >1 的。得到一个 $O(14nP)$ 的做法，还是过不了。

原因在于高精度小数还是太慢了，但是前面才这么几层，哪用得到这么高精度？

【NOI2016】国王饮水记

似乎这个区间想要不增的条件有点苛刻，打表发现，我们区间划分到后来会变为一堆长度为 1 的。

还记得之前推出过的区间长度不增的性质吗？经过一系列推导（这回我也不会），然后发现前面最多有 14 段 >1 的。得到一个 $O(14nP)$ 的做法，还是过不了。

原因在于高精度小数还是太慢了，但是前面才这么几层，哪用得到这么高精度？

于是前面的 DP 可以把精度设小一点，或者使用高精度分数。后面的 1 直接暴力。

然后就能跑过去了。使用高精度分数更稳一点。设高精度分数精度为 S，则复杂度为 $O(nS^3+nP)$。

SMAWK 算法

~~下面的讲题顺序按照 17 年论文~~ 在我们的实践中，发现奇偶分类做法貌似还有很大的优化空间，因为每次算决策点的点就那么多，为什么还要有这么多决策点要去判断？

所以说，之前的递归式 $T(x) = T(x/2) + O(n)$，不太优美。

如果能改成 $T(x) = T(x/2) + O(x)$ 就好了。

所以我们要用上决策单调链，去缩减可行下标集合的大小。

SMAWK 算法

```
vector reduce(vector A, int m) { // reduce |A| to m
    int p = 1; // A : 1-index
    while (A.size() > m) {
        if (p != m && calc(p, A[p]) > calc(p, A[p + 1]))
            ++p;
        else {
            if (p == m && calc(p, A[p]) > calc(p, A[p + 1]))
                A.erase(p + 1);
            else {
                A.erase(p);
                --p;
            }
        }
    }
    return A;
}
```

SMAWK 算法

我们维护一个栈，它不是严格单调的。我们只拿出了所有可能成为最优的点。

SMAWK 算法

我们维护一个栈，它不是严格单调的。我们只拿出了所有可能成为最优的点。

当下一个端点更优时，由决策单调性，我们这个端点是没用的，把顶部不优的全部弹出。

SMAWK 算法

我们维护一个栈，它不是严格单调的。我们只拿出了所有可能成为最优的点。

当下一个端点更优时，由决策单调性，我们这个端点是没用的，把顶部不优的全部弹出。

在下一个端点不优时，当且仅当它是序列尾可以删掉，否则显然我们可以把它加入序列。

SMAWK 算法

我们维护一个栈，它不是严格单调的。我们只拿出了所有可能成为最优的点。

当下一个端点更优时，由决策单调性，我们这个端点是没用的，把顶部不优的全部弹出。

在下一个端点不优时，当且仅当它是序列尾可以删掉，否则显然我们可以把它加入序列。

于是我们就得到了可能的端点集合。我们在每一层调用一次 reduce，复杂度就变成了线性了。

SMAWK 算法

我们维护一个栈，它不是严格单调的。我们只拿出了所有可能成为最优的点。

当下一个端点更优时，由决策单调性，我们这个端点是没用的，把顶部不优的全部弹出。

在下一个端点不优时，当且仅当它是序列尾可以删掉，否则显然我们可以把它加入序列。

于是我们就得到了可能的端点集合。我们在每一层调用一次 reduce，复杂度就变成了线性了。

之前的奇偶分类加上这个 reduce，就是 SMAWK 算法。

可以发现，它唯一的需求就是决策单调性。

【NOI2009】诗人小 G

Example (诗人小 G)

你有一个序列 A，你要将它分为若干段 S_i，记每段和为 $f(S_i)$。
你需要最小化 $\sum |f(S_i) - L|^P$
保证 $|A| \le 10^5, A_i \le 30, L \le 3 \times 10^6, 1 \le P \le 10$。

显然不能点积最大化（你 k 维凸壳咋整？）

【NOI2009】诗人小 G

Example (诗人小 G)

你有一个序列 A，你要将它分为若干段 S_i，记每段和为 $f(S_i)$。
你需要最小化 $\sum |f(S_i) - L|^P$
保证 $|A| \le 10^5, A_i \le 30, L \le 3 \times 10^6, 1 \le P \le 10$。

显然不能点积最大化（你 k 维凸壳咋整?）
Obviously you can use anything to solve it. By ignore2004

【NOI2009】诗人小 G

但是，我们发现一个不对劲的：这个 P 次幂是带绝对值的！

【NOI2009】诗人小 G

但是，我们发现一个不对劲的：这个 P 次幂是带绝对值的！

先尝试四边形不等式。对于权值和，我们干脆把矩阵放到整数集上，$w_{r,l} = |r - l - L|^P$ 的 l 坐标平移。

【NOI2009】诗人小 G

但是，我们发现一个不对劲的：这个 P 次幂是带绝对值的！

先尝试四边形不等式。对于权值和，我们干脆把矩阵放到整数集上，$w_{r,l} = |r-l-L|^P$ 的 l 坐标平移。

这样子，只要我们证明了 $|r-l|^P$ 的四边形不等式就可以了。

即 $|w+2|^P + |w|^P - 2|w+1|^P \geq 0$。

【NOI2009】诗人小 G

但是，我们发现一个不对劲的：这个 P 次幂是带绝对值的！

先尝试四边形不等式。对于权值和，我们干脆把矩阵放到整数集上，$w_{r,l} = |r - l - L|^P$ 的 l 坐标平移。

这样子，只要我们证明了 $|r - l|^P$ 的四边形不等式就可以了。

即 $|w + 2|^P + |w|^P - 2|w + 1|^P \geq 0$。

讨论完 $w \in \{-2, -1, 0\}$ 后发现 w 的正负都没关系，所以讨论 $w > 0$。

【NOI2009】诗人小 G

但是，我们发现一个不对劲的：这个 P 次幂是带绝对值的！

先尝试四边形不等式。对于权值和，我们干脆把矩阵放到整数集上，$w_{r,l} = |r - l - L|^P$ 的 l 坐标平移。

这样子，只要我们证明了 $|r - l|^P$ 的四边形不等式就可以了。

即 $|w + 2|^P + |w|^P - 2|w + 1|^P \geq 0$。

讨论完 $w \in \{-2, -1, 0\}$ 后发现 w 的正负都没关系，所以讨论 $w > 0$。

$$(w + 2)^P + w^P - 2(w + 1)^P = \sum_{i=0}^{P} w^{P-i}(2^i - 2) + w^P \geq w^P - w^P = 0$$

【NOI2009】诗人小 G

但是，我们发现一个不对劲的：这个 P 次幂是带绝对值的！

先尝试四边形不等式。对于权值和，我们干脆把矩阵放到整数集上，$w_{r,l}=|r-l-L|^P$ 的 l 坐标平移。

这样子，只要我们证明了 $|r-l|^P$ 的四边形不等式就可以了。

即 $|w+2|^P+|w|^P-2|w+1|^P\geq 0$。

讨论完 $w\in\{-2,-1,0\}$ 后发现 w 的正负都没关系，所以讨论 $w>0$。

$$(w+2)^P+w^P-2(w+1)^P=\sum_{i=0}^{P}w^{P-i}(2^i-2)+w^P\geq w^P-w^P=0$$

所以说为什么要这个时候讲这道题呢？

因为 SMAWK 算法需要一个性质：代价函数任意一点都能方便算出。然后这道题就可以套 SMAWK 了。

【UOJ285】数据分块鸡

Example (数据分块鸡)

给你一个长度为 n 序列和 Q 个询问区间，让你分块，求最小代价。设询问为 $[l,r)$，块为 $[a_i,a_{i+1})$ 一个询问的代价为：

1. 若 $[l,r)=[a_i,a_{i+1})$，则代价为 1。
2. 若 $[l,r)\subseteq[a_i,a_{i+1})$，则代价为 $(l-a_i)+(a_{i+1}-r)$。
3. 其余情况，整块 $+1$，散块 $+\min\{L,B-L\}$，B 为块大小，L 为查询与块的交集大小。

保证 $n\le 5\times 10^4, Q\le 10^5$。

【UOJ285】数据分块鸡

显然对于任意一个块 $[l,r)$ 我们都能快速计算它对答案的贡献。

然后就可以全在线单调性 DP。当然我们需要去尝试证明四边形不等式。

【UOJ285】数据分块鸡

显然对于任意一个块 $[l, r)$ 我们都能快速计算它对答案的贡献。

然后就可以全在线单调性 DP。当然我们需要去尝试证明四边形不等式。

考虑最极端的情况，即只有单个区间。凸函数线性组合还是凸函数，所以单区间凸代表着总代价函数凸。

【UOJ285】数据分块鸡

显然对于任意一个块 $[l,r)$ 我们都能快速计算它对答案的贡献。

然后就可以全在线单调性 DP。当然我们需要去尝试证明四边形不等式。

考虑最极端的情况，即只有单个区间。凸函数线性组合还是凸函数，所以单区间凸代表着总代价函数凸。

令询问区间为 $[l,r)$，四个分块区间为 $[x,y)$, $[x+1,y+1)$, $[x+1,y)$, $[x,y+1)$。

【UOJ285】数据分块鸡

显然对于任意一个块 $[l,r)$ 我们都能快速计算它对答案的贡献。

然后就可以全在线单调性 DP。当然我们需要去尝试证明四边形不等式。

考虑最极端的情况，即只有单个区间。凸函数线性组合还是凸函数，所以单区间凸代表着总代价函数凸。

令询问区间为 $[l,r)$，四个分块区间为 $[x,y)$, $[x+1,y+1)$, $[x+1,y)$, $[x,y+1)$。

但是四个区间，根据题面，每个区间有多种和询问区间的关系，会有大量的讨论。

【UOJ285】数据分块鸡

显然对于任意一个块 $[l,r)$ 我们都能快速计算它对答案的贡献。

然后就可以全在线单调性 DP。当然我们需要去尝试证明四边形不等式。

考虑最极端的情况，即只有单个区间。凸函数线性组合还是凸函数，所以单区间凸代表着总代价函数凸。

令询问区间为 $[l,r)$，四个分块区间为 $[x,y)$, $[x+1,y+1)$, $[x+1,y)$, $[x,y+1)$。

但是四个区间，根据题面，每个区间有多种和询问区间的关系，会有大量的讨论。

但是我们能够断言，关系序列（比如包含，包含，包含，相等）确定了一类等价类，即单调性全部一样。

于是我们可以写程序机械化地证明，只要给出一个 n，并且跑 $O(n^4)$ 的检验算法即可。

【UOJ285】数据分块鸡

```cpp
int calc(int l, int r, int L, int R) {
    if (L <= l && r <= R) return std::max(l - L + R - r, 1);
    if (l <= L && R <= r) return 1;
    l = std::max(L, l), r = std::min(r, R);
    if (l > r) return 0;
    return std::min(r - l, (R - L) - (r - l));
}
bool ineq(int l, int r, int x, int y) {
    return calc(l, r, x, y) + calc(l, r, x + 1, y + 1) <=
           calc(l, r, x + 1, y) + calc(l, r, x, y + 1);
}
bool check(int n) {
    for (int l = 0; l < n; ++l)
        for (int r = l + 1; r <= n; ++r)
            for (int x = 0; x < n; ++x)
                for (int y = x + 2; y + 1 <= n; ++y)
                    if (!ineq(l, r, x, y))
                        return false;
    return true;
}
```

【UOJ285】数据分块鸡

所以最后就只剩下一个全在线的板子，用单调队列就好了。
至于线性的做法呢……我还不会……
谁会了来教教我啊。

`std::Thanks();`

感谢带家耐心地听我讲课。下面真没东西了……所以刷题去吧……

还有，这个 Smawk 还挺简单的，希望大家都能在课后写一遍。