dp优化

Konata

海亮高级中学

842812488@qq.com

2024年2月26日

前言

- 需要提前了解的前置知识：李超树
- 由于本人特别菜，出现错误或者讲解有问题请直接指出
- 请不要喷讲题人/kel

P5574 任务分配问题

在某台有 k 个 CPU 的计算机中，有 n 个计算任务等待执行。
a_i 为第 i 个任务的优先级，方便起见， a 为一个排列。
现在，要将这些任务分配给 CPU 去解决。
由于内存等原因，一个 CPU 只能负责连续一段的任务，并且要按 (从左到右的) 顺序执行。
在某个 CPU 内，无序度定义为：前者先执行，而后者优先级高的任务对的个数。
请最小化每个 CPU 的无序度之和。
$n \leq 25000, k \leq 25$

P5574 任务分配问题

设$f[j][i]$表示把前i个数分成j段所需的最小代价。
转移：设$c(i,j)$为$(i,j]$之间的顺序对数,可得:

$$f[j][i]=\min_{t=1}^{i-1}\{f[j-1][t]+c(t,i)\}$$

如何计算$c(t,i)$?
注意到,类似莫队,我们能从$c(l,r)$使用树状数组$O(log)$推到相邻的区间。
那么我们就倒着来转移,不断加入数字,再更新逆序对数,就能做到转移$O(nlogn)$了。
总的复杂度是$O(n^2klogn)$

P5574 任务分配问题

我们发现c满足四边形不等式，所以具有决策单调性，用决策单调性分治即可。
还有一个问题就是怎么快速求c，我们可以用类似莫队的思路暴力跳指针
首先，从区间 $[L, R]$ 的最后一次计算递归到 $[L, mid]$ 的第一次计算最多会挪动 $O(R - L)$ 次。其次，同一层递归之间的区间挪动最多为 $O(n)$ 次（因为右端点和左端点都只会向右走）。这样均摊下来单次计算就是 $O(1)$ 的。
复杂度为$O(nk\log^2 n)$

CF671D Roads in Yusland

给定一棵 n 个点的以 1 为根的树。
有 m 条路径 (x, y)，保证 y 是 x 或 x 的祖先，每条路径有一个权值。
你要在这些路径中选择若干条路径，使它们能覆盖每条边，同时权值和最小。
$n, m \leq 3 \times 10^5$，空间限制$256MB$

CF671D Roads in Yusland

对于给出的路径 (u_i, v_i)（v_i 是 u_i 的祖先），我们在节点 u_i 处考虑是否选择它。以下称 u_i 为“起点”，v_i 为“终点”。
设 $\mathrm{dp}(u, j)$ 表示选择了若干条起点在 u 的子树内的路径，使得这些路径覆盖了 u 子树内的所有边，并且它们的终点的深度的最小值为 j（$j \le \mathrm{dep}(u)$），达到这种情况所需的最小花费。

CF671D Roads in Yusland

设 $f(u)=\min_{j=1}^{\mathrm{dep}(u)-1}\mathrm{dp}(u,j)$，也就是至少覆盖到 u 和父亲之间的边，所需的花费。

转移。先不考虑以 u 为起点的路径，则有：

$$\mathrm{dp}(u,j)=\min_{v\in\mathrm{son}(u)}\left\{\mathrm{dp}(v,j)+\sum_{w\in\mathrm{son}(u),w\neq v}f(w)\right\}$$

也就是枚举一个儿子 v，让它里面的路径，最小深度达到了 j。
然后其他儿子 w 就可以随便选了。
把它改写一下：

$$\mathrm{dp}(u,j)=\sum_{v\in\mathrm{son}(u)}f(v)+\min_{v\in\mathrm{son}(u)}\{\mathrm{dp}(v,j)-f(v)\}$$

CF671D Roads in Yusland

接下来考虑以 u 为起点的路径，设终点的深度为 j，价格为 c，则转移也是类似的：

$$\mathrm{dp}(u,j)=\sum_{v\in\mathrm{son}(u)} f(v)+c$$

答案就是 $\mathrm{dp}(1,1)$。这个朴素 DP 的时间复杂度为 $\mathcal{O}(n^2)$。

CF671D Roads in Yusland

考虑优化。容易想到，用线段树来维护 DP 的第二维。需要支持：

- 区间加：转移前把所有 $\mathrm{dp}(v,j)$ 加上 $-f(v)$；以及最后把所有 $\mathrm{dp}(u,j)$ 加上 $\sum_{v\in \mathrm{son}(u)} f(v)$。
- 查询区间最小值：也就是求出 $f(v)$。
- 线段树合并。合并时，对应位置取 min。发现这相当于要支持区间取 min。
- 单点对一个数取 min：在考虑以 u 为起点的路径时，单点对 c 取 min。发现这已经被上一条（区间取 min）包含。

时间、空间复杂度 $\mathcal{O}((n+m)\log n)$。因为本题空间限制较小，难以通过。

CF671D Roads in Yusland

我们需要挖掘更多性质。我们可以发现一个重要的单调性：对于 $j_1 < j_2$，若 $\mathrm{dp}(u, j_1) < \mathrm{dp}(u, j_2)$，则 $\mathrm{dp}(u, j_2)$ 不会对答案有任何贡献，可以删除。所以一个点有用的 dp 值序列是一个单调的序列。所以我们考虑对每个点，维护一个 `std::set`。里面存二元组：$(j, \mathrm{dp}(v, j))$。以 j 为关键字排序。

- 区间加法。发现只要我们把 $j > \mathrm{dep}(u)$ 的元素及时弹出，区间加就变为了全局加。因此只要维护一个全局的标记即可。
- 合并可以采用启发式合并。相同的 j 第二维取 min
- 单点取 min 相当于插入一个新元素。
- 查询最小值：根据单调性直接取最后一个元素即可

时间复杂度 $\mathcal{O}((n+m)\log^2 n)$，空间复杂度 $\mathcal{O}(n+m)$

CF1129D Isolation

给出一个长度为 n 的序列，把它划分成若干段，使得每一段中出现过恰好一次的元素个数 $\leq k$，求方案数对 998244353 取模后的结果。

$n \leq 10^5$

CF1129D Isolation

- 首先考虑 dp，dp_i 表示以 i 为结尾的划分的方式，那么显然有转移 $dp_i = \sum_{j=0}^{i-1} dp_j[uni(j+1,i) \le k]$，其中 $uni(l,r)$ 表示 $[l,r]$ 中出现恰好一次的数的个数。
- 考虑优化。我们实时维护一个数组 $f_j = uni(j+1,i)$，那么上式可写作 $dp_i = \sum_{j=0}^{i-1} dp_j[f_j \le k]$，而注意到每次 i 指针的右移引起 f 数组的变化应为两个区间的 $+1/-1$。
- 于是现在问题就转化为，有一个序列，每个元素都有一个权值 value 和键值 key，每次我们可以将一个区间的键值 $+1/-1$ 或者询问序列中键值 $\le k$ 的元素的权值之和。

CF1129D Isolation

上面的问题一般数据结构不好维护，考虑分块。我们设一个阈值 B 并将原序列分成 B 块，并对每一块建一个 BIT，每次我们执行区间 $+v$ 时，边角块就暴力修改 f_i 并修改 BIT 里的值，中间块就维护一个标记数组 tag_i 并令 tag_i 加上 v，查询时就枚举每一块，并在 BIT 中查询 $\le k - tag_i$ 所有数的和即可。时间复杂度 $n\sqrt{n\log n}$

CF1129D Isolation

注意到上面的解法并没有用到每次加的值为 ± 1 这一性质。我们考虑直接对于每一块维护一个前缀和数组 $sum_{i,j}$ 表示第 i 块中 $f_k \le j$ 的 dp_k 之和，那么由于加的值为 ± 1，故每次对区间中一个元素暴力加值的时候，最多会引起一个 $sum_{i,j}$ 的变化；因此修改单个元素的复杂度就变成了 $\mathcal{O}(1)$，总复杂度也就变成了 $n\sqrt{n}$

还有一个问题，就是当我们计算出 dp_i 之后，对应的 f_i 显然是 0，而此时该位置的权值也发生了改变，即由 0 变为了 dp_i，此时我们就要遍历 $j \in [0, n]$ 并令 $sum_{b,j}$ 加上 dp_i，其中 b 为 i 所在的块，这样复杂度又会退化到 n^2，我们只需将 $sum_{i,j}$ 的定义改为第 i 块中 $f_k \ge j$ 的 dp_k 之和，查询时维护一个 tot_i 表示这块中所有的 dp_j 之和，拿这块中所有的 dp_j 之和减去不合法的情况即可，这样修改还是 $\mathcal{O}(1)$ 的。

CF1175G Yet Another Partiton Problem

给定数组 $a_1, a_2 \cdots a_n$，你需要将它划分成 k 段（每个元素在且仅在一段中），某段 $a_l, a_{l+1} \cdots a_r$ 的权值为 $(r-l+1) \times \max_{l \le i \le r}\{a_i\}$，整个划分的权值是每段权值之和。求最小划分权值。

$n \le 2 \times 10^4$，$k \le 100$

CF1175G Yet Another Partiton Problem

$O(n^2k)$ 的暴力 dp 很简单，令 $f_{i,k}$ 表示将前 i 个数分成 k 段的最小值，转移方程是

$$f_{i,k} = \min_{j=k-1}^{i-1} f_{j,k-1} + (i-j) \times \max_{l=j+1}^{i} a_l$$

考虑优化。 max 不方便处理，注意到序列可以划分为若干段，其中每一段的 max 值都相等。这部分可以用单调栈维护。

CF1175G Yet Another Partiton Problem

- 在每一段内部，需要求出 $f_{j,k-1} - j \times \max$ 的最小值。这显然可以用斜率优化。所求即为经过点 $(j, f_{j,k-1})$ 且斜率为 max 的直线在 y 轴截距的最小值。凸包上二分即可。
- 每次向单调栈加入一个值时可能会将若干段合并为一段，用启发式合并，将小凸包的点插入大凸包。时间复杂度为 $O(n \log n)$ 。
- 令 $k = \max$ ， $b = f_{j,k-1} - j \times \max$ ，则 $f_{i,k}$ 即为每一段的直线 $y = kx + b$ 在 $x = i$ 时纵坐标的最小值。可以用李超树维护。因为插入的是直线，所以复杂度为 $(n \log n)$ 。注意有删除最新加入的若干条直线操作，所以李超树需要可回溯或者持久化。

时间复杂度 $O(nk \log n)$ ，空间复杂度 $O(n \log n)$

CF280E Sequence Transformation

给你一个非减序列$x_1, x_2, ..., x_n(1 \leq x_1 \leq x_2 \leq ... \leq x_n \leq q)$。你还有两个整数$a$和$b(a \leq b, a(n-1) < q)$

你要把序列变成$y_1, y_2, ..., y_n(1 \leq y_i \leq q, a \leq y_{i+1} - y_i \leq b)$

变换的代价为$\sum_{i=1}^{n}(x_i - y_i)^2$

最小化变换代价

原题$n \leq 6000$，可加强至10^5

CF280E Sequence Transformation

我们设 $dp_i(u)$ 表示对于所有的合法的 u，$y_i = u$ 时 $\sum_{j\le i}(y_j - x_j)^2$ 的最小值。有转移式为

$$dp_{i+1}(u) = \min_{d\in[a,b]} dp_i(u-d) + (u-x_j)^2$$

设 $dp_{i-1}(u)$ 的最小值取在 k_i，就有

$$dp_i(u) = dp_{i-1}(u-a) + (u-x_i)^2 (u < k_i + a)$$

$$dp_i(u) = dp_{i-1}(k_i) + (u-x_i)^2 (k_i + a \le u \le k_i + b)$$

$$dp_i(u) = dp_{i-1}(u-b) + (u-x_i)^2 (u > k_i + b)$$

很容易发现 $dp_i(u)$ 是一个凸函数，并且根据最小值点分为两半平移之后加上一个二次函数。我们联想到slope trick，考察其导函数，希望找到导函数的零点（使原函数取最小值的点）。

CF280E Sequence Transformation

首先，我们发现 $dp_i'(u)$ 是单调的，并且有：

$$dp_i'(u) = dp_{i-1}'(u-a) + 2u - 2x_i(u < k_i + a)$$

$$dp_i'(u) = 2u - 2x_i(k_i + a \le u \le k_i + b)$$

$$dp_i'(u) = dp_{i-1}'(u-b) + 2u - 2x_i(u > k_i + b)$$

我们发现，新的函数就是把原来的函数从中间断开，然后左边往右平移 a，右边往右平移 b，中间用 0 填满。我们每次只是平移，然后加上一次函数，所以导函数也一直是一次函数，这就很好分段维护了。考虑分段维护当前所有的 $dp_i'(x)$，每次只需要找到零点断开，左右分别平移，中间安排 0，然后整体加上 $2u - 2x_i$。每次最多增加两个段，所以最终的段个数不会超过 $2n$，复杂度 $O(n^2)$。

CF280E Sequence Transformation

最后我们以 $y_n = k_n$ 为基础倒推 y，因为 k_i 是使得当前取值最小的位置，而 $dp_i(u)$ 还是个凸函数，所以很显然，尽可能的让 y_{i-1} 在不违反 y_i 条件的情况下最接近 k_{i-1} 即可。然后直接用得到的 y 计算答案。

CF280E Sequence Transformation

如果要做到更好的话，我们发现我们需要动态找到一个极值点，也就找到一个跨过 0 点的段，而因为是凸函数，所以斜率单调，我们用平衡树维护所有斜率相同的段，每次找到跨过 0 点的段，这是容易的。
然后，我们把左边的段打一个平移 a 的标记，右边的段打一个 b 标记，加入中间的 0 段，最后整体加一个一次函数就好了。
也就是说，我们维护平移量，和一个一次函数的 tag 即可。

CF771E Bear and Rectangle Strips

给定 $2 \times n$ 的矩阵 t，求最多能切分出多少个和为 0 的连续子矩阵。

$n \le 3 \cdot 10^5$，$|t_{i,j}| \le 10^9$

CF771E Bear and Rectangle Strips

我们可以用map先预处理出每个位置开始只用第一行、第二行、两行的下一个合法矩阵右边界的最小值记为nxt。
我们可以很轻松设计出基础的dp：设$f_{i,j}$表示考虑第一行前i列和第二行前j列的答案。转移为不选矩阵转移到$f_{i+1,j}$, $f_{i,j+1}$和第一行（$f_{nxt_i,j}$）、第二行（f_{i,nxt_j}）、两行（$f_{nxt_{max(i,j)},nxt_{max(i,j)}}$）选择矩阵的转移。

CF771E Bear and Rectangle Strips

优化的核心思想是我们只要正解能被生成即可，不需要所有状态都得转移到。想象一个解，容易考虑一个类似归并的生成过程：当前如果接下来两行都是单行矩形，那么选右端点小的那个。也相当于不允许出现第一行右端点为i，第二行右端点为j，且$nxt_j < i$或者对称的情况。并且我们考虑对于一个i，固定$j \le i < nxt_j$，发现只需要记录使得$f_{i,j}$取到最大值的最小的j即可。因为如果没取到最大值，由于$nxt_j > i$，故该状态所有的导出状态都不优于取到最大值的状态的导出状态。因此只需 1D，记对称的两个情况，刷表法转移即可。

CF1530H Turing's Award

图灵大神站在一个向两边无限延伸的磁带上，一共有 n 个时刻，时刻 1 他站在 0 号格子，每一个时刻结束后他都可以选择向左/向右走一格或不动。

现在输入一个均匀随机生成的 n 阶排列 a_i ，在时刻 i 时他会把 a_i 写在当前格子上，如果当前格子上已经写有数字就会覆盖掉原有的数字。第 n 个时刻结束后，定义价值为将写有数的所有格子从左向右连起来形成的序列的最长上升子序列，求可能获得的最大价值。

$n \leq 15000$

CF1530H Turing's Award

你发现这个覆盖不太好考虑，考虑时间倒流，变成如下形式：

一开始，小 A 的位置上有一个数 a_n，然后对于接下来 $n-1$ 步，每次小 A 可以向左走/向右走/不动，然后如果此时小 A 所站的位置上没有数，就写上 a_i，求最后形成序列的最长上升子序列长度。

考虑到任意时刻的序列一定是包含原点的连续序列 b，并且此时不能覆盖有数的位置，那么 a_i 只可能插入到 b 的首尾两个位置。

CF1530H Turing's Award

我们将所有最终在 b 中的元素 a_i 成为「好」的元素，显然 a_n 一定是好元素，观察到：

- 我们并不关心小 A 走到 b 中间哪个位置，只关心他是否有时间在已经确定的连续好元素之间移动。例如，目前考虑了 $a_i, a_{i+1}, \cdots, a_n$，钦定 a_i 为好元素并且插入到 b 的首位（插入末尾的情况是对称的），目前 $a_i, a_{i+1}, \cdots, a_n$ 中一共有 k 个好元素，考虑第 $k+1$ 个好元素 $a_j (j < i)$：
  - 如果 a_j 插入在 b 的首位，这种情况没有限制。
  - 如果 a_j 插入在 b 的末尾，小 A 要走过 k 个好元素。那么需要满足 $i - j \geq k$。
- 考虑一个好元素 a_j（$j \neq n$）如果存在一种方案使得 a_j 在最终的 b 序列里不在 LIS 中，一定存在一种方案使得 a_j 最终不在 b 序列中，因为可以在时刻 j 留在原地不动。所以我们更改好元素的定义：最终出现在 LIS 中的元素（包括 a_n）。

CF1530H Turing's Award

然后我们可以讨论 a_n 是否属于 LIS，考虑 dp：
令 $f_L(k,i)$ 表示当前从 n 考虑到 i，a_i 插入到了 LIS 的首位，目前一共有 k 个好元素，LIS 末尾元素的最小值；$f_R(k,i)$ 表示当前从 n 考虑到 i，a_i 插入到了 LIS 的末尾，目前一共有 k 个好元素，LIS 首位元素的最大值。那么我们要求的就是最大的 k，使得存在一个 i，状态 $f_L(k,i)$ 或者 $f_R(k,i)$ 合法（属于 $[1,n]$）。
根据 a_n 是否属于 LIS，我们可以确定 dp 的初始状态。
考虑 $f_L(k,i)$ 可以贡献到的状态（$f_R(k,i)$ 类似），讨论新的好元素 a_j 放左还是放右即可：
$f_L(k,i) \to f_L(k+1,j) \quad j<i \wedge a_j<a_i$
$a_i \to f_R(k+1,j) \quad j<i-k \wedge a_j>f_L(k,i)$

CF1530H Turing's Award

那么不难得到 f_L, f_R 的转移：

$$f_L(k+1,i)=\min\left(\min_{j>i\wedge a_j>a_i} f_L(k,j), \min_{j\geq i+k\wedge f_R(k,j)\geq a_i} a_j\right)$$

$$f_R(k+1,i)=\max\left(\max_{j>i\wedge a_j<a_i} f_R(k,j), \max_{j\geq i+k\wedge f_L(k,j)\leq a_i} a_j\right)$$

显然可以优化，用两个数据结构支持查询前缀 max 和后缀 min 即可。复杂度 $O(kn\log n)$，k 为答案，考虑到均匀随机序列的 LIS 长度期望为 $O(\sqrt{n})$，k 取 $O(\sqrt{n})$，复杂度 $O(n\sqrt{n}\log n)$

其实这里还用了一个 dp 优化套路：将 dp 答案与状态的一维交换，例如我们也可以按照 LIS 的首尾位为状态 dp，但是复杂度更劣。所以把 LIS 长度列入 dp 状态。

P5044 meetings 会议

有 N 座山横着排成一行，从左到右编号为从 0 到 $N-1$。山的高度为 H_i（$0 \le i \le N-1$）。每座山的顶上恰好住着一个人。
你打算举行 Q 个会议。会议 j 的参加者为住在从山 L_j 到山 R_j。
对于该会议，你必须选择某个山 x 做为会议举办地。举办该会议的成本与你的选择有关，其计算方式如下：
- 来自每座山 y 的参会者的成本，等于在山 x 和 y 之间的所有山的最大高度。特别地，来自山 x 的参会者的成本是 H_x。
- 会议的成本等于其所有参会者的成本之和。
你想要用最低的成本来举办每个会议。询问之间独立。
$N, Q \le 750000$

P5044 meetings 会议

考虑一个 $\mathcal{O}(n^2)$ 暴力，设 $f_{l,r}$ 表示区间 $[l,r]$ 的答案，我们可以按照区间 max 分治下两边看看在哪边举办，这样另一边的贡献就已知了：

$$f_{l,r} = \min(f_{l,x-1} + (r-x+1)h_x, f_{x+1,r} + (x-l+1)h_x)$$

其中 $x = \max_{i=l}^{r}\{h_i\}$。

P5044 meetings 会议

容易发现这种按照区间极值向两边递归的结构类似笛卡尔树，所以我们考虑对于 h 数组建立笛卡尔树。但你发现建立了之后还是没法直接转移，主要问题在于这个区间实在是太没有规律了。
但观察到我们的 min 的两个参数中的 f，一定满足有一个端点是笛卡尔树某个节点的左右端点。由于两边的求解是独立的，我们不妨设均满足左端点在笛卡尔树上。（另一边可以把数组 reverse 之后再做）
然后对于一个笛卡尔树上的节点 $[l, r]$，设它对应的 max 位置为 mid，我们需要求解所有的 $f_{l,i}, l \le i \le r$。分治之后，所有的 $f_{l,i}, l \le i < mid$ 都已经求出。然后我们再向右分治，求出 $f_{mid+1,i}, mid < i \le r$。容易发现 $f_{l,mid} = f_{l,mid-1} + h_{mid}$。

P5044 meetings 会议

然后再来观察 $f_{l,i}, i > mid$ 的表达式：

$$f_{l,i} = \min(f_{l,mid} + (i - mid)h_{mid}, f_{mid+1,i} + (mid - l + 1)h_{mid})$$

和一开始 dp 方程的区别就是我们把 $f_{l,mid-1} + h_{mid}$ 合并了一下，这样出现了一个我们很好算的常数。其中的 $f_{l,mid}, f_{mid+1,i}$ 我们都已经求出。但问题是，如果我们还是枚举 i 求解并没有做本质上的优化，时间复杂度仍然是 $\mathcal{O}(n^2)$ 的。不过我们注意到，min 的两个参数都是随着 i 单调递增的，且前者每次恰好加 h_{mid}，而后者至多加 h_{mid}。这说明，这个 min 的取值是单调的！也就是说，我们可以通过二分的方式确定取前者后者的分界点。从而可以快速区间处理 $f_{l,i}$ 的求解。

P5044 meetings 会议

接下来我们来看看具体怎么实现这个过程。首先是存 f 的问题，我们需要对第二维支持较为复杂的区间操作，所以直接对第二维建立一棵线段树。我们递归计算左边右边后，第二维在 $[l, mid)$ 范围内的数表示的是 $f_{l,i}$，在 $(mid, r]$ 范围内的表示的是 $f_{mid+1,i}$。（因为每次计算的都是以这个节点左端点开始的 f）而 $f_{l,mid}$ 的计算是简单的。接下来我们考虑在线段树上二分，二分到一个被完全覆盖的区间时，只需要判断 min 取值的分界点在不在这个区间内，如果不在就直接更新（这里要么是区间加，要么是区间赋值一次函数），如果在就递归下去。容易发现我们只需要维护一个区间内最靠左的 f 和最靠右的 f 即可。而复杂度，显然相当于两次区间修改。然后把每个询问拆分成的其中一个询问挂在它左端点对应节点上即可。

时间复杂度 $\mathcal{O}(n \log n)$

CF1830F The Third Grace

给定一个数轴上的 n 个区间和 m 个点。第 i 个区间覆盖坐标 $[l_i, r_i]$，第 i 个点在坐标 i 处，并且具有系数 p_i。
最初，所有点都未激活。你需要选择一些点来激活。对于每个区间 i，我们定义它的代价为：
- 若区间内没有被激活的点，则代价为 0；
- 否则，代价为在区间内坐标最大的被激活点的系数。

你的任务是通过选择哪些点激活，使得所有区间的代价之和最大。
$n, m \leq 10^6$

KTT

首先介绍一个叫做KTT的算法。用下解决如下问题

考虑维护 n 个一次函数，其中第 i 个是 $a_i x_i + b_i$。维护一下三种操作：
- b 区间加：对 $i \in [l, r], b_i \leftarrow b_i + c$。
- x 区间加：对 $i \in [l, r], x_i \leftarrow x_i + c$。
- 求 f 区间 max：求 $\max_{l \le i \le r} a_i x_i + b_i$。

其中需要保证 $c > 0$。

KTT 的复杂度大概是 $\mathcal{O}((n+q)\text{polylog } n)$。其中 polylog 大概在实际上可以看做 $\log \sim \log^2$。跑得很快。我们下面仅围绕着 CF1830F 所需要的结构进行讨论，也就是我们这里不进行 b 的区间加，我们进行 a, b 的单点赋值。实际上做到 b 的区间加是不难的。

KTT

KTT 的本质还是一颗线段树。我们考虑线段树的节点维护一下的信息：

- x 的懒标记 $laz(x)$。
- f 的区间 max 所代表的函数 val_i。
- 使得 i 子树内某个节点维护最大值发生改变的最小增加的正整数 $intr_i$。

如果没看懂可以直接看下面每个操作的解释。

在构建 KTT 的过程中，如果访问到一个节点，我们就把其维护的最优线段的 x 归零，并加到 b_i 上面去。

KTT

pushdown函数：我们只需要更新一下两边儿子最优节点的 b 值，加一下懒标记，再右移一下最小更新点即可。

```
void pushdown(int id){
    for(int son:{id<<1,id<<1|1}){
        val[son].b+=laz[id]*val[son].a;
        intr[son]-=laz[id];
        laz[son]+=laz[id];
    }
    laz[id]=0;
    return;
}
```

KTT

pushup函数：其中inter函数是求两个线段的交点的纵坐标（向上取整，如果小于 0 返回无穷大）

```
void pushup(int id){
    val[id]=(val[id<<1]<val[id<<1|1]?val[id
        <<1|1]:val[id<<1]);
    intr[id]=min({intr[id<<1],intr[id<<1|1],
        inter(val[id<<1],val[id<<1|1])});
    return;
}
```

KTT

rebuild函数：考虑一个节点子树内某个节点的最大值发生变化的时候我们需要修改其子树的值。如果一个子树的节点 *intr* 的是 ≥ 0 的，也就是内部不会发生变化，我们应该继续递归。

```
void rebuild(int id){
    if(intr[id]>0) return;
    pushdown(id);
    rebuild(id<<1);rebuild(id<<1|1);
    pushup(id);
    return;
}
```

KTT

set函数：单点赋值。

```
void set(int x,line v,int id=1,int l=1,int r
    =m){
     if(l==r) {val[id]=v;return;}
     pushdown(id);
     int mid=l+r>>1;
     if(x<=mid) set(x,v,id<<1,l,mid);
     else set(x,v,id<<1|1,mid+1,r);
     pushup(id);
     return;
}
```

KTT

modify函数：这个函数是实现 x 的区间加的。

```
void modify(int L,int R,ll x,int id=1,int l
    =1,int r=m){
      if(L<=l&&r<=R){
            laz[id]+=x;val[id].b+=val[id].a*x;
                 intr[id]-=x;
            rebuild(id);return;
      }
      pushdown(id);
      int mid=l+r>>1;
      if(L<=mid) modify(L,R,x,id<<1,l,mid);
      if(mid<R) modify(L,R,x,id<<1|1,mid+1,r);
      pushup(id);
}
```

KTT

ask函数：同普通线段树查询

```
ll ask(int L,int R,int id=1,int l=1,int r=m)
    {
     if(L<=l&&r<=R) return val[id].b;
     pushdown(id);
     int mid=l+r>>1;ll ans=0;
     if(L<=mid) ans=max(ans,ask(L,R,id<<1,l,
        mid));
     if(mid<R) ans=max(ans,ask(L,R,id<<1|1,
        mid+1,r));
     pushup(id);
     return ans;
}
```

CF1830F The Third Grace

回到题目的解法：
设 f_i 表示以 i 为最后一个被选中的位置的方案的最大权值，其中不包含 i 的贡献，因为 i 的贡献和下一个被选中的位置有关。
枚举下一个位置 j，转移 $f_i + c_{i,j}p_i \to f_j$，其中 $c_{i,j}$ 表示 $l \le i \le r < j$ 的区间数量。
考虑在 j 增大的过程中实时维护所有 $g_i = f_i + c_{i,j}p_i$。当 $j \to j+1$ 时，考虑所有 $r = j$ 的区间 $[l, r]$，将区间内每个 $g_i \leftarrow g_i + p_i$。
相当于：单点修改 g_i，区间将 $g_i \leftarrow g_i + p_i$，全局查询 $\max g_i$。
用KTT即可解决。

CF1830F The Third Grace

上述做法是固定 j 考虑 i 的贡献，尝试固定 i 考虑它对所有 j 的贡献，即用 $f_i + c_{i,j}p_i$ 去更新 f_j。尝试对每个 j 维护当前 $h_j = c_{i,j}$，$i \to i+1$ 相当于对 h 进行后缀加减，那么问题相当于：

- 给定 k, b，用 $kh_j + b$ 更新 f_j。
- 后缀加减 h。
- 查询 f_j。

因为 h 单调不降，所以没有操作二是经典李超树。

CF1830F The Third Grace

对于操作二，相当于要维护一个取值点会变的李超树，并且注意到 $i \to i+1$ 不会对已经添加的直线产生影响（否则只能使用 KTT）。即保证之前添加的每条直线在位置 $l \sim r$ 处的取值不变，所以 $kh_l + b$ 需要变成 $k(h_l + 1) + (b - k)$。

将部分 h 被修改的区间的直线向两侧下放，这样就没有直线在边界处了，打懒标记处理对 h 的区间加法。

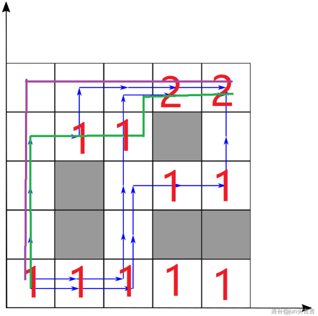
因为部分 h 被修改的区间数量为 $\mathcal{O}(\log m)$，所以总复杂度为 $\mathcal{O}(n\log^2 m + m\log m)$

CF720D Slalom

一个 $n \times m$ 的网格，其中有 k 个矩形障碍，保证这些障碍不重叠。求从 $(1,1)$ 走到 (n,m)，每步只能往右或往上走，不经过任何障碍的方案数。
两种方案被视为不同，当且仅当存在一个障碍，它在第一种方案里被从右侧绕过，而在第二种方案里被从左侧绕过（第一种左，第二种右同理）。
$n,m \le 10^6$，$k \le 10^5$。

CF720D Slalom

首先，这个不同的定义与我们平常接触到的略有不同。我们平常用$f_{i,j}$来dp的思路似乎不太可行。于是下面有本题第一个非常重要的转化：只记录最低的路径上的方案数。那么样例就是长这样的：

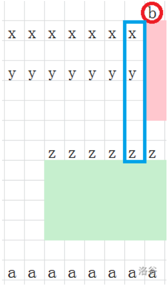


最后的答案即为$2+1=3$。

CF720D Slalom

先来讲一讲这个东西是如何得到的：

- 如果当前列没有新矩形，那么$f_{i,j} = f_{i-1,j}$
- 如果加入了新矩形，那么新矩形的上方一列会产生新的最低的路径。如下图：



那么b就是从x, y, z转移过来，与a半毛钱关系都没有。

CF720D Slalom

于是我们设矩形从$y1$到$y2$，那么就有：

$$f_{i,x} = 0, x \in [y1, y2]$$

$$f_{i,y2+1} = \sum_{x=low}^{y2+1} f_{i-1,x} \ (low\text{为从从y2下去能到的最远距离})$$

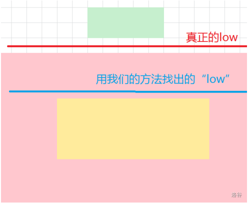
那么现在我们的时间复杂度为$O(nm^2)$，空间复杂度为$O(n)$，考虑优化。
首先，考虑此算法的瓶颈在哪里。主要还是两个地方：

- 求出 low
- 求 $\sum_{x=low}^{y2+1} f_{i-1,x}$

第二个东西求和用线段树维护十分简单。

CF720D Slalom

们用一个set来存储当前所有覆盖的线段。那么我们就可以找到最大（指按pair的排序方法）的一条第一条线段。那么*low*就可以取他的 $y2+1$ 。可能 $y2+1$ 并不是最远距离。看下面一张图片来理解一下：



两个之间的 f 值必定是 0 ，因此不会影响 dp。