序列加法机

首先不同位置之间是独立的，证明在后面，也就是我们只管把每个 a_i 向着 b_i 的方向调整。如果设对 i 的调整步数为 y 的话，那么显然平均分配最优，而且 y 越大总代价越小(是凸的)。即考虑初始 $y=1$，然后逐渐变大为 $|b_i-a_i|$ 这样一个过程。每次 y 增加 1 会带来代价减少的一个差量，考虑所有 i 中差量最多的那个 i'，显然在 i' 上增加一个调整步数是最优的。

那么直接维护一个堆就好了，比较套路。

考虑我们知道了每个 i 的调整步数了以后怎么构造出方案。如果一个 i 要调整，但被 j 挡住了，那就先调整 j，以此类推。由于 a,b 都是单调不降的，因此不会出现最后又绕了个圈回到 i 从而矛盾的情况。

摸鱼军训

记 $\operatorname{rev}[x]=x$ 之前 $>x$ 数的数量，$\operatorname{rev}[x]$ 在 k 趟冒泡之后会变为 $\max(0,\operatorname{rev}[x]-k)$。

考虑 k 趟冒泡排序之后数组的样子：

- 若 $\operatorname{rev}[a[i]]\geq k$，则 $a[i]$ 每次都向前移动 1 格，去到 $i-k$ 位置
- 否则，剩下所有 $\operatorname{rev}[a[i]]<k$ 的数有序排列，依次放入剩余的位置中

用逆序对数量稍加分析，可以得到上述结论。

对于一次询问 (k,x)，若 $\operatorname{rev}[x]<k$，需要解决 2 个问题：

- x 在所有 $\operatorname{rev}[x]<k$ 的数中排第几，设为 rk
- 第 rk 个空位在什么位置

由于本题可以离线，这两个问题都可以求出 rev 数组后使用支持单点置 1、区间求和、查询第 k 个 $0/1$ 位置的数据结构（树状数组或者线段树）实现。
综上，总复杂度为 $(n+q)\times\log n$。

皮卡丘

50 pts

每次询问做一遍【NOI2010 超级钢琴】

复杂度 $O\left(mn\log n+\left(\sum k\right)\log\left(\sum k\right)\right)$

另 15 pts: k=1

考虑线段树解决

对于每个点我们维护：区间最大值 $\max$ 、区间最小值 $\min$ 、区间的最大答案 ans

这样对于一个节点 p，$ans_p=\min\left(ans_{p->ls},ans_{p->rs},\max_{p->ls}-\min_{p->rs}\right)$

区间 query 方法类似

对于区间加， tag 对 $\max, \min$ 的影响都是直接加，而对 ans 没有影响

(注意到这个和动态最大子段和求解的方法非常像，实际上是因为把 a 进行差分之后我们所求的基本就是最大子段和.......)

复杂度 $O(m \log n)$

100 pts

我们假设你已经会了【NOI2010 超级钢琴】

注意到这道题的正解当中采用了“候补答案集合”的思想: 把左端点 $= x$ ，右端点 $\in [l, r]$ 的所有答案视作 一个 node 放进优先队列，每次取答案最小的一个进行累加，然后按最优解的位置把这个“候补答案集合” 分裂成两个

这显然不够带劲。我们能不能直接用一个 node 表示一个矩形（左端点属于一个区间，右端点也属于一 个区间）的答案?

一般的矩形是不好求解最优答案的。但是有两种可以：左右端点属于的区间完全重合（也就是 $k = 1$ 的 做法）和左右端点属于的区间完全相离（在左边取最大值，再在右边取最小值，二者相减）

我们的目标是求解这两种矩形，然后在分裂的时候也保证得到的“候补答案集合”也属于这两种矩形，事实上这是可以做到的

先看第一种: 左端点、右端点都 $\in [l, r]$ ，假设最优解位于 (x, y)

我们把它分裂成如下矩形:

左端点 $\in [l, x-1]$，右端点 $\in [l, x-1] (x > l)$

左端点 $\in [l, x-1]$，右端点 $\in [x, r] (x > l)$

左端点 $\in [x, x]$，右端点 $\in [x, x] (x \neq y)$

左端点 $\in [x, x]$，右端点 $\in [x+1, y-1] (x < y-1)$

左端点 $\in [x, x]$，右端点 $\in [y+1, r] (y < r)$

左端点 $\in [x+1, r]$，右端点 $\in [x+1, r] (x < r)$

再看第二种: 左端点 $\in [l_0, r_0]$，右端点 $\in [l_1, r_1], l_1 > r_0$，假设最优解位于 (x, y)

我们把它分裂成如下矩形:

左端点 $\in [l_0, x-1]$，右端点 $\in [l_1, r_1]\ (x > l_0)$

左端点 $\in [x, x]$，右端点 $\in [l_1, y-1]\ (l_1 < y)$

左端点 $\in [x, x]$，右端点 $\in [y+1, r_1]\ (y < r_1)$

左端点 $\in [x+1, r_0]$，右端点 $\in [l_1, r_1]\ (x < r_0)$

就好了

复杂度 $O\left(n \log n + \left(\sum k\right)\left(\log n + \log\left(\sum k\right)\right)\right)$

银行的源起

问题简化

先来考虑一个简单版本的问题：我们只有一个银行。答案是：

$$\sum_{(x,fa)\in E} w\cdot\min\left(size_x, S-size_x\right) \tag{1}$$

证明：
不妨设树根为 1。在上面的表达式中，$size_x$ 表示的是在 x 子树中居民的数量，S 表示树中居民的总数量。考虑如何计算所有居民到达银行的总时间：对于每一条边，它的贡献为 $w*$ **通过这条边的居民的数量的总和**，即对于每一条边 (x, fa, w)，我们有两种选择：一是选择子树内的所有居民通过该边（此时银行在子树外），故此部分贡献为 $w\cdot size_x$；二是选择 x 的子树外的所有点上的居民通过该边进人 x 的子树（此时银行在子树内）。两种方式取 min 即可。

这个简化问题可以通过一遍 DFS 在 $O(n)$ 的时间内解决。

Subtask1

暴力枚举两个银行的位置

Subtask2

在链上直接算即可

Subtask3

考虑最终银行放置的位置，会有一条没有居民过的边。那么可以枚举该边并将此边断开，使原树分成两个树，再通过上面的方法对两棵树的答案独立计算。时间复杂度为 $O\left(n^2\right)$。

Subtask4

定义 in_v 为 DFS 时进人 v 点的时间戳，out v 为 DFS 时离开 v 点的时间戳。

假设只放置一个银行，我们就应该找到对于所有边 (v, to, w) : $\sum_{e\in E} w\cdot min\left(size_{to}, S-size_{to}\right)$ 的总和。考虑 $size_{to}$ 和 $S-size_{to}$ 什么时候会算到贡献里：$w\cdot min\left(size_{to}, S-size_{to}\right)$ 为 $w\cdot size_{to}$ 当且仅当 $size_{to}\leq\frac{S}{2}$, 为 $w\cdot\left(S-size_{to}\right)$ 当且仅当 $size_{to}>\frac{S}{2}$。

回到 $O\left(n^2\right)$ 的解法。我们在 DFS 时在树中移去边 (v, to, w) 。现在有两棵子树, 大小分别为:
$X=size_{to}, Y=size_1-size_{to}$ 。考虑分别在 X, Y 两棵树上解决问题。

第一部分先计算 v 子树中的点的答案： $w\cdot\min\left(size_{to}, X-size_{to}\right)$ 的值的和。通过上述的小技巧，我们可 以计算出 $w\cdot\mathrm{size}_{to}\left(\forall size_{to}\leq\frac{X}{2}\right)+w\cdot X\left(\forall size_{to}>\frac{X}{2}\right)-w\cdot size_{to}\left(\forall size_{to}>\frac{X}{2}\right)$ 。

剩余的部分就是要求出区间 $[l, r]$ 内所有数字 $\leq K$ 的和。可以把区间 $\left[in_{to}, out_{to}\right]$ 内的点的 w 和 $w\cdot size_{to}$ 丢进两个树状数组, 分别统计两类的答案即可。

to 的子树以外的部分做法也很类似。注意要将这些操作在区间 $\left[1, in_{to}-1\right]$ 和 $\left[out_{to}+1, n\right]$ 和子树 Y 上完成。除了从根节点开始的链到 v (to 的子树中包括的节点)。在这条链上 $size_u$ 单调递减, 所以用双指针并且记录前缀和。可以减去我们要为 $\leq\frac{Y}{2}$ 计算的部分。同样对 size $_1-$ size $_u$ 做这样的步骤。最后加上我们需要的 $>\frac{Y}{2}$ 的部分, 这部分在这条链上递增。

时间复杂度和空间复杂度均为 $O(n\log n)$。