《世界沉睡童话》解题报告

一、题目描述

泠珞给了你一棵大小为 n 的有标号有根树 T，T 中的结点从 $1 \sim n$ 标号，其中标号为 Rt 的结点是它的根。

在最初的时候，T 的每个结点上写有一个 $1 \sim n$ 之间的整数，称作这个结点的点权。保证 n 个结点的点权构成一个 $1 \sim n$ 的排列。设标号为 i 的结点的点权是 a_i，那么 $a_1 \sim a_n$ 构成一个 $1 \sim n$ 的排列。

需要注意的是，对于 T 中的每个非叶子结点 u，u 的儿子 v_i 之间都是有顺序的。设 u 有 k 个儿子，那么 u 的 k 个儿子从左到右依次称为 $\mathrm{ch}(u,1), \mathrm{ch}(u,2), \cdots\cdots, \mathrm{ch}(u,k)$。

现在以 Rt 为根 DFS 这棵树 T。当访问到一个结点 u 的时候，我们按照 $\mathrm{ch}(u,1), \mathrm{ch}(u,2), \cdots\cdots, \mathrm{ch}(u,k)$ 的顺序，依次递归地 DFS 其未被访问过的 k 个儿子结点。这个过程将会得到唯一的一个 DFS 序列 s，其中 s_i 表示第 i 个被访问到的结点的标号是 s_i。

接下来你需要进行恰好 $n-1$ 次删边操作，第 i 次操作将会删去 s_{i+1} 与 s_{i+1} 的父亲 $\mathrm{Fa}(s_{i+1})$ 所连的边 $\big(s_{i+1}, \mathrm{Fa}(s_{i+1})\big)$。

在第 i 次删边操作进行之前，你需要决定是否交换 s_{i+1} 的点权与 $\mathrm{Fa}(s_{i+1})$ 的点权。如果你选择交换，那么 $a_{s_{i+1}}$ 与 $a_{\mathrm{Fa}(s_{i+1})}$ 的值将会互换，然后 $\big(s_{i+1}, \mathrm{Fa}(s_{i+1})\big)$ 将被删去；否则 $\big(s_{i+1}, \mathrm{Fa}(s_{i+1})\big)$ 将被直接删去，不改变点权。

你一共需要做出 $n-1$ 次选择，做出这 $n-1$ 次选择的方案数共有 2^{n-1} 种。

当 $n-1$ 条边尽数被删去以后，设 a'_i 表示最终标号为 i 的点的点权。我们定义集合 S 为所有通过交换操作可以得到的最终点权的排列构成的集合。换言之，一个 $1 \sim n$ 的排列 $p \in S$ 当且仅当存在至少一种做出选择的方案使得最终每个 a'_i 都 $= p_i$。

最后泠珞给了你一个 $1 \sim n$ 的排列 p，你需要在下列三个小问中**选择一个**进行回答：

- 第 1 小问：判定 p 是否 $\in S$。
- 第 2 小问：求出 p 在 S 中的后继。即求出字典序大于 p 并且 $\in S$ 的排列 q 中，字典序最小的那一个。如果 S 中存在字典序大于 p 的排列 q，你需要报告存在并输出字典序最小的那一个 q；否则你只需要报告不存在。
- 第 3 小问：求出 p 在 S 中的排名。即求出一共有多少个排列 $q \in S$，满足 q 的字典序小于 p，对大质数 998244353 取模。

二、数据范围

对于所有数据，保证 $2 \le n \le 2 \times 10^5$；$n$ 个结点的儿子信息确实描述了一棵树 T；$a_1 \sim a_n$，$p_1 \sim p_n$ 均构成一个 $1 \sim n$ 的排列。

以下是各子任务的具体情况以及特殊性质：

子任务编号	分值	$n =$	树的形态	特殊性质	子任务依赖
1	5	20			
2	5	8×10^4	完全二叉树	DFS 序 $1 \sim n$	
3	5	8×10^4	完全二叉树		2
4	5	8×10^4	二叉树	DFS 序 $1 \sim n$	2
5	10	8×10^4	二叉树		$2 \sim 4$
6	10	8×10^4		DFS 序 $1 \sim n$	$2, 4$
7	15	8×10^4			$1 \sim 6$
8	15	1.2×10^5			$1 \sim 7$
9	15	1.6×10^5			$1 \sim 8$
10	15	2×10^5			$1 \sim 9$

- 『二叉树』：保证 T 中每个结点的儿子个数都 ≤ 2。
- 『完全二叉树』：保证 T 与一棵大小同样为 n 的完全二叉树同构，并且 Rt 对应完全二叉树的根。
- 『DFS 序 $1 \sim n$』：按照题目描述中所述方法对 T 进行 DFS，第 i 个被访问到的结点一定是标号为 i 的结点，即 $s_i = i$。

三、解题过程

一、解决第 1 小问

- 记 b_x 表示初始时点权为 x 的结点的标号，则 b 与 a 互为逆排列。
- 再记 t_i 表示标号为 i 的结点在 DFS 的过程中第几个被访问到，则 t 与 s 也互为逆排列。
- 如果一条边在被删去之前交换了两端点的点权，则将这条边染黑；否则将这条边染白。做出 $n-1$ 次选择一共有 2^{n-1} 种方案，一一对应 2^{n-1} 种不同的染色方案。根据一种染色方案进行交换，称最终点权构成的排列 p 为该种染色方案得到的结果。

- 结论 $1.\ |S|=2^{n-1}$。即任意两种不同的染色方案得到的结果也一定不同。

两种染色方案是不同的，说明至少存在一条边 $\big(u,\mathrm{Fa}(u)\big)$，在第一种方案中这条边染白，在第二种方案中这条边染黑。

连接 u 子树和外部的途径只有 $\big(u,\mathrm{Fa}(u)\big)$ 一条。我们将 u 子树内结点的初始点权标为 0，u 子树外的标为 1。如果交换 $\big(u,\mathrm{Fa}(u)\big)$，最终 u 子树内就会恰有一个 1；否则就会全部是 0。因此交换 $\big(u,\mathrm{Fa}(u)\big)$ 与不交换 $\big(u,\mathrm{Fa}(u)\big)$，最终点权肯定不会完全相同。

我们可以由此导出第 1 小问的做法：给定一个排列 p，我们先 DFS 一遍，对每个结点 u 求出 u 子树内所有 $t_{b_{p_i}}$ 的最大值、最小值。若最小值 $=t_u$ 且最大值 $=t_u+\mathrm{siz}(u)-1$，则说明 $\big(u,\mathrm{Fa}(u)\big)$ 必须染白；否则说明 $\big(u,\mathrm{Fa}(u)\big)$ 必须染黑。

得到每条边的颜色之后，我们再检查一遍交换过后的结果是否确实 $=p$ 即可。时空复杂度均为 $\mathcal{O}(n)$，可以获得 10 分。

我们再定义一个『部分染色方案』：一条边可以被确定染白，也可以被确定染黑，或者尚未确定被染成什么颜色。定义一个部分染色方案的自由度等于未染色的边的总个数 c，那么将该部分染色方案完善成完整的染色方案的方式共有 2^c 种，且这 2^c 种完整的染色方案所得到的最终点权构成的排列 p 一定两两不同。

二、刻画权值在点权交换的过程中运输的路线

假设 $n-1$ 条边是否交换都已经固定下来。

考虑权值 a_u 在点权交换的过程中运输的路线：初始时 a_u 就在 u 的位置，经过一系列枢纽结点的中转，a_u 最终将到达另一个结点 v，我们称 u 为起始地，称 v 为目的地，称 $u\rightsquigarrow v$ 这条树链上所有的结点为枢纽结点。注意 v 可能 $=u$。

首先结点 u 是第 1 个枢纽结点。第 1 次我们找 u 周围 k_u+1 条邻边中的第一条黑边，该条黑边所指向的结点就是第 2 个枢纽结点。从第 2 次开始往后，第 i 次我们从某条黑边进入第 i 个枢纽结点 h_i，我们找 h_i 周围 $k_{h_i}+1$ 条邻边中该条黑边的下一条黑边，则该下一条黑边所指向的结点就是第 $i+1$ 个枢纽结点。不断这样找下去，直到某次 **“下一条黑边”** 不存在，即进来的黑边后面全是白边为止，说明我们已经到达了最后一个枢纽结点，也就是目的地。

一个枢纽结点 u 周围 k_u+1 条邻边被交换的顺序是：先交换 $\big(u,\mathrm{Fa}(u)\big)$，然后从左到右依次交换 $(v_1,u),(v_2,u),\cdots\cdots,(v_{k_u},u)$。我们所谓寻找 **“第一条黑边”** 以及寻找 **“下一条黑边”** 的顺序同样是这个顺序。

现在我们要反过来想，固定一个目的地 u，我们要找到 u 的起始地是哪个结点 v。我们反过来考虑这个过程，倒着找到所有的枢纽。最后 1 个枢纽一定是 u 本身。第 1 次我们找 u 的邻边中最后一条黑边，它所指向的是倒数第 2 个枢纽结点。从第 2 次往后，我们从进来的黑边开始找上一条黑边，它所指向的正是上一个枢纽结点。我们这样一直顺藤摸瓜，最终不存在 **“上一条黑边”** 的枢纽结点就是起始地。

注意到从倒数的 h_2 开始，在这个过程中绝大部分的时候，都是从 h_i 的某个儿子到 h_i 的边出发，找到的上一条黑边就是 $\big(h_i,\mathrm{Fa}(h_i)\big)$ 本身。因为一旦找到的黑边不是 $\big(h_i,\mathrm{Fa}(h_i)\big)$，意味着对于 h_{i+1} 而言是从它的父亲 h_i 到它的边 (h_i,h_{i+1}) 进来的，追溯的过程在 h_{i+1} 就结束了。同理如果 $h_2\neq\mathrm{Fa}(h_1)$ 而 $\in\mathrm{son}(h_1)$，那么追溯的过程在 h_2 就结束了。

我们可以整理从 u 开始往前追溯的过程一路上访问的所有边，这里我们用结点的标号 u 来表示 $\big(u,\mathrm{Fa}(u)\big)$ 这条边：首先从右到左倒序访问 u 的所有儿子，然后访问 u 自己，然后倒序访问 u 左侧的所有兄弟，然后访问 $\mathrm{Fa}(u)$ 以及 $\mathrm{Fa}(u)$ 的所有左兄弟，以此类推……

当我们访问的是一个儿子或者一个兄弟的时候，如果访问到的边是黑色，那么追溯的过程立刻就结束了，目的地 u 的起始地就是这个儿子或者这个兄弟本身；否则如果是白色则过程继续。当我们访问的是一个祖先的时候，如果访问到的边是白色，那么追溯的过程也立刻就结束了，目的地 u 的起始地即是这个祖先本身；否则如果是黑色则还要继续。

我们把上一个自然段所做出的观察称作结论 2。结论 2 意味着我们可以把一路上访问到的所有边倒序罗列下来，对于每条边我们都有一个预期颜色，一旦找到第一条不符合预期颜色的边，整个追溯的过程就结束了，而 u 的起始地也就是这条不符合预期颜色的边靠下的端点；否则追溯的过程还要继续。由于最后一条边 $\big(\mathrm{Rt},\mathrm{Fa}(\mathrm{Rt})\big)$ 一定是白色，不符合预期颜色黑色，因此追溯的过程不可能一直继续下去。

三、解决第 2 小问

现在我们来解决第 2 小问。我们维护一个『部分染色方案』，实时记录当前还剩下的自由度 cur。

- 操作 $\mathrm{fix}(u,x)$ 表示：固定 $p_u=x$，即固定 u 的起始地为初始时点权为 x 的结点 b_x。

当我们执行 $\mathrm{fix}(u,x)$ 的时候，一定会将一个子集里的边钦定染黑，将另一个子集里的边钦定染白。如果先前被染黑的边今天又被钦定染白，或者如果先前被染白的边今天又被钦定染黑，那么就是矛盾的。一个 $\mathrm{fix}(u,x)$ 是矛盾的，说明在之前的所有其他 fix 都满足的前提之下，不可能使得 $p_u=x$，即使得 u 的起始地为初始时点权为 x 的结点 b_x。

解决第 2 小问和第 3 小问的共同流程是从 1 到 n 扫一遍，扫到 i 的时候，令 S_i 表示所有执行 $\mathrm{fix}(i,x)$ 不会导致矛盾的 x 组成的集合。

对于第 2 小问的解决，我们首先要求出 p 和 q 不相等的最高位是哪一位。我们希望这个最高位越靠后越好，当我们扫到 i 的时候我们求出 $\max S_i$，若 $\max S_i>p_i$ 说明 $p_i\neq q_i$ 的最高位可以是 i 这一位。然后我们执行 $\mathrm{fix}(i,p_i)$，固定 $q_i=p_i$，再进入下一个循环 $i+1$。如果扫到 i 的时候发现 $\mathrm{fix}(i,p_i)$ 矛盾，那么扫完 i 我们就应该停止扫描。

我们通过上述过程找到所有可以不相等的最高位里最靠后的那一位 λ，然后我们清空所有染色的限制再正着扫一遍。我们依次执行 $\mathrm{fix}(1,p_1),\mathrm{fix}(2,p_2),\cdots\cdots,\mathrm{fix}(\lambda-1,p_{\lambda-1})$，然后我们求出 S_λ 中 $>p_\lambda$ 的最小元素 x 执行 $\mathrm{fix}(\lambda,x)$，然后我们从 $\lambda+1$ 开始不断求出 $\min S_i$ 并且 $\mathrm{fix}(i,\min S_i)$，直到 $i=n$ 为止。每次执行 $\mathrm{fix}(i,x)$ 意味着 $q_i=x$。

解决第 3 小问则直接很多：设 $\mathrm{reduce}(i,x)$ 表示固定 $p_i=x$ 将导致多少条先前未染色的边被确定为染黑或者染白之一，则我们只需要正着扫一遍，扫到 i 的时候，我们先将 $\sum\limits_{x\in S_i\cap[1,p_i)}2^{\mathrm{cur}-\mathrm{reduce}(i,x)}$ 的贡献计入答案，然后再执行 $\mathrm{fix}(i,p_i)$ 即可。

现在我们考虑一个暴力的算法。扫到 u 的时候，我们需要完成两件任务：执行 $\mathrm{fix}(u,x)$ 以及求出 S_u。

我们先把以 u 为目的地追溯的过程中一路上访问到的所有边拿出来，以下简称『沿路边』。按照追溯的顺序，我们将所有沿路边依次命名为 $e_1,e_2,\cdots,e_m$，我们这样解读一条沿路边的颜色：绿色表示它的黑白颜色等于预期颜色，红色表示它的黑白颜色不等于预期颜色。

- 定义第 i 条沿路边 e_i 是合法的，当且仅当它自己是红色或者未染色，并且 $e_1\sim e_{i-1}$ 都是绿色或者未染色。记 a_{e_i} 表示 e_i 靠下的端点的初始权值，则 e_i 是合法的意味着 e_i 靠下的端点可以作为目的地 u 的起始地，也就意味着 p_u 可以 $=a_{e_i}$。
- 求出 S_u：先找到编号最小的红色沿路边 e_i，那么 S_u 就是 $e_1\sim e_{i-1}$ 中所有的未染色边 j 对应的 a_{e_j} 所组成的集合。
- 执行 $\mathrm{fix}(u,x)$：设第 i 条沿路边对应的 $a_{e_i}=x$，那么将 e_i 染成红色，将 $e_1\sim e_{i-1}$ 全都染成绿色。在这一步中如果先前被染红的边今天又被钦定染绿，或者如果先前被染绿的边今天又被钦定染红，那么 $\mathrm{fix}(u,x)$ 就是矛盾的。甚至可能不存在任何一条沿路边 e_i 对应的 $a_{e_i}=x$，此时意味着 b_x 不可能是目的地 u 的来源地，因此 $\mathrm{fix}(u,x)$ 也是矛盾的。

注意所谓的 **“染红”** 实际上指的是将这条沿路边染成与它的预期颜色不同的那个黑白颜色，而 **“染绿”** 指的就是把它染成预期颜色。

以上暴力的算法能够在 $\mathcal{O}(n\log n)$ 的时间和 $\mathcal{O}(n)$ 的空间之内通过完全二叉树的数据。对于 DFS 序 $1\sim n$ 的数据，我们可以用一个栈来维护从 i 开始追溯的过程中一路上按顺序访问到的边所组成的序列，随着 i 的扫描该序列的总变化量是 $\mathcal{O}(n)$ 的且所有变化都一定发生在末尾，稍作简单的实现可以导出一个时空复杂度均为 $\mathcal{O}(n)$ 的做法，可以获得 30 分，结合第 1 小问的 $\mathcal{O}(n)$ 做法可以获得 37 分。

我们用重链剖分可以将上述 $\mathcal{O}(n^2)$ 的暴力优化到 $\mathcal{O}(n\log^2 n)$。

首先注意到每条边至多被染色一次，因此可以在每次染色的时候暴力定位到所有需要从灰边变成黑白边之一的那些边，并执行染色操作。

- 对于每个结点 u，我们开一个数据结构 $\mathbb{S}_u$ 倒序维护 u 的所有儿子。
- 对于每条以 u 为顶端的重链，我们开一个数据结构 $\mathbb{T}_u$ 自下而上维护以这条重链为虫身的一个毛毛虫。我们从重链底端自下而上遍历到重链顶端，当我们遍历到 v 时，我们从 v 的重儿子开始从右到左将 v 一个前缀中的所有儿子加入数据结构 $\mathbb{T}_u$。

最后我们在所有 $\mathbb{S}_u,\mathbb{T}_u$ 的结尾处追加一条边 $\big(u,\mathrm{Fa}(u)\big)$。在数据结构 $\mathbb{S}_u$ 内除了 $\big(u,\mathrm{Fa}(u)\big)$ 的预期颜色是黑色以外，所有儿子的预期颜色都是白色。在数据结构 $\mathbb{T}_u$ 内所有重边以及 $\big(u,\mathrm{Fa}(u)\big)$ 的预期颜色都是黑色，而所有轻边的预期颜色都是白色。

$\mathbb{S}_u,\mathbb{T}_u$ 维护的是红绿而不是黑白。当我们决定为一条边染上黑白颜色的时候，我们要在 ≤ 4 个数据结构里将其染成对应的红绿颜色。

现在我们考虑跳重链的过程。当我们从 u 一路跳重链上来的时候，会经过 $\mathcal{O}(\log n)$ 条轻边和被这些轻边所分隔的等量重链前缀。当经过一条轻边 (u,v) 时我们在 $\mathbb{S}_u$ 里查询，而当经过一个重链前缀 $u\rightsquigarrow v$ 时我们在 $\mathbb{T}_u$ 里查询。

假设我们从 $\mathbb{T}_u$ 中的第 l 条边切进这个数据结构 $\mathbb{T}_u$。我们首先用 `set` 找到 $[l,m]$ 中 l 后面的第一条红边 r，如果不存在则 $r=m+1$。我们要求的是 $[l,r)$ 内所有未染色边靠下的端点初始权值的 $\max$ 和 $\min$，可以直接使用线段树维护区间 $\max$ 和区间 $\min$。在中间的 $i=\lambda$ 那一步我们需要求出 S_λ 中 $>p_\lambda$ 的最小元素 x 执行 $\mathrm{fix}(\lambda,x)$，由于只有这一步，我们可以直接暴力扫一遍。

该算法的时间复杂度为 $\mathcal{O}(n\log^2 n)$，空间复杂度为 $\mathcal{O}(n)$，可以获得 70 分。结合第 3 小问的 $\mathcal{O}(n^2)$ 暴力可以获得 79 分。

四、解决第 3 小问

现在我们来解决第 3 小问。每个数据结构 $\mathbb{S}_u, \mathbb{T}_u$ 所干的事都是一样的：以某个维护 m 条边的 $\mathbb{T}_u$ 为例，每次我们从某个位置 l 切进这个数据结构 $\mathbb{T}_u$，用 `set` 找到 $[l,m]$ 中 l 后面的第一条红边 r，我们实际上是对 $[l,r]$ 区间进行查询。

- $\mathbb{T}_u$ 维护一个长度为 m、字符集为 $\{\texttt{0},\texttt{1},\texttt{?}\}$ 的序列 $c_1 \sim c_m$，初始时所有 $c_i = \texttt{?}$。
- 每个修改操作给出一个位置 i，你需要支持把 $c_i = \texttt{?}$ 修改为 `0`，或者把 $c_i = \texttt{?}$ 修改为 `1`。
- 每个查询操作给出一个区间 $[l,r]$ 和一个权值 x，保证 $c_l \sim c_{r-1}$ 中只有 `0` 和 `?`，并且 $c_r = \texttt{1}$。对于 $[l,r]$ 中每个 $c_i = \texttt{?}$ 或者 $c_i = \texttt{1}$ 的位置 i，如果 $a_{e_i} < x$ 满足，你需要将 $\frac{1}{2}$ 的『$c_l \sim c_i$ 中 `?` 的总个数』次幂的贡献计入答案，其中 a_{e_i} 永远不会改变。

单独一个数据结构 $\mathbb{T}_u$ 所干的事能够归约所谓的『$x < A$ 乘 $y < B$ 和』问题，因此我们应该追求一个 $\mathcal{O}(n\sqrt{n})$ 的做法。

我们先考虑数点问题，不考虑权值问题，即计数 $[l,r)$ 中满足 $a_{e_i} < x$ 的 $c_i = \texttt{?}$ 的总个数。

假如只有查询操作，我们可以开一棵线段树维护序列 $a_{e_1} \sim a_{e_m}$，在每个线段树结点 u 上开一个有序的 `vector` M_u，存放 $a_{e_{l_u}} \sim a_{e_{r_u}}$ 排序后的结果。设线段树结点 u 的两个子结点分别为 $\mathrm{lch}(u)$ 和 $\mathrm{rch}(u)$，则 M_u 可以直接由 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 二者归并得到。

当我们查询一个区间 $[l,r]$ 时，首先在线段树上定位到 $[l,r]$ 被分解为的 $\mathcal{O}(\log n)$ 个线段树结点，这 $\mathcal{O}(\log n)$ 个线段树结点管辖区间的无交并等于 $[l,r]$，我们在每个线段树结点 u 的 M_u 上进行二分即可查出 $a_{e_i} < x$ 的个数。

现在我们要应对修改操作。当 $c_i = \texttt{?}$ 被修改为 `0` 时，相当于删除了 a_{e_i} 这个元素。如果在 `push_up()` 的过程中不断向上归并 `vector`，单次操作 $\mathcal{O}(n)$ 不能承受。因此我们设置一个阈值 B，只对于那些 $r_u - l_u + 1 \le B$ 的线段树结点 u 维护出 M_u，这样向上 `push_up()` 的过程中重构的总区间长度是 $B + \frac{B}{2} + \frac{B}{4} + \cdots\cdots = \mathcal{O}(B)$，一次修改的时间复杂度也为 $\mathcal{O}(B)$。

当然查询操作的时间复杂度会相应增大。当我们递归到一个 $[l_u,r_u] \subseteq [l,r]$ 的线段树结点 u 的时候，如果 $r_u - l_u + 1 > B$，那么我们不能直接在 M_u 里查询，只能继续递归下去直到长度 $\le B$ 的结点。这样一来我们只能把 $[l,r]$ 分解为 $\mathcal{O}(\frac{n}{B} + \log n)$ 个管辖区间，乘上每个 M_u 内二分的 $\mathcal{O}(\log n)$，一次查询的时间复杂度为 $\mathcal{O}(\frac{n\log n}{B} + \log^2 n)$。

更令人无法接受的是我们外面还套了一层重链剖分！好在由于所有 $\mathbb{S}_u, \mathbb{T}_u$ 所维护的序列总长度之和 $\le 4n$，满足 $[l_u,r_u] \subseteq [l,r]$ 并且 $r_u - l_u + 1 > B$ 需要暴力递归的线段树结点 u 仍只有 $\mathcal{O}(\frac{n}{B})$ 个。一次查询的时间复杂度变为了 $\mathcal{O}(\frac{n\log n}{B} + \log^3 n)$。

取 $B = \mathcal{O}(\sqrt{n\log n})$ 平衡得该算法的时间复杂度为 $\mathcal{O}(n\sqrt{n\log n} + n\log^3 n)$，空间复杂度为 $\mathcal{O}(n\log n)$，并不十分令人满意。

以上线段树结构叫做『AKEE 线段树』。只是解决数点问题其实可以直接树套树，然而 AKEE 线段树可以无缝衔接到权值问题。

对于 M_u 中的每一个 x，我们记 $[l_u,r_u]$ 中所有满足 $a_{e_i} \le x$ 的 $c_i = \texttt{?}$ 的贡献，即 $\frac{1}{2}$ 的『$c_{l_u} \sim c_i$ 中 `?` 的总个数』之和。只需要在每个线段树结点 u 上开一个与 M_u 等长的 `vector` F_u 存放贡献之和，再额外记录 $[l_u,r_u]$ 中 $c_i = \texttt{?}$ 的总个数，即可支持 `push_up()` 归并。

最后我们可以利用分散层叠算法将 AKEE 线段树优化到 $\mathcal{O}(n\sqrt{n} + n\log^2 n)$。

具体来说，对于 $r_u - l_u + 1 \le B$ 的线段树结点 u，我们再开两个与 M_u 等长的 `vector` L_u, R_u，分别用来记录 M_u 中的每一个 x 在 $M_{\mathrm{lch}(u)}$ 以及 $M_{\mathrm{rch}(u)}$ 中二分查找的结果。对于 $r_u - l_u + 1 > B$ 的线段树结点 u 我们同样如此做，但是此时 M_u 并不是 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 直接归并的结果，而是 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 各自均匀的每隔 3 个元素取最后 1 个元素，再将二者归并起来的结果。

对于长度 $> B$ 的部分，我们归并重构的总区间长度是 $B + \frac{2B}{3} + \frac{4B}{9} + \cdots\cdots = \mathcal{O}(B)$，不会增大一次修改的时间复杂度 $\mathcal{O}(B)$。

在一次查询操作当中，我们可以先在线段树的根结点 M_{root} 中进行二分查找，然后根据在一个线段树结点 M_u 中二分查找的结果 $\mathcal{O}(1)$ 确定出在两个子结点 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 中二分查找的结果，自上而下递归直到长度 $\le B$ 的结点。这样一来，由于去掉了暴力二分所带来的 $\mathcal{O}(\log n)$，一次查询的时间复杂度降到了 $\mathcal{O}(\frac{n}{B} + \log^2 n)$。

取 $B = \mathcal{O}(\sqrt{n})$ 平衡得该算法的时间复杂度为 $\mathcal{O}(n\sqrt{n} + n\log^2 n)$，空间复杂度为 $\mathcal{O}(n\log n)$，可以获得 100 分。

四、参考资料

本题的灵感来源于 [BalticOI 2016 Day 2] C. Swap。由于在先前的信息学竞赛中笔者做到过 [CSP-S 2019 Day 2] C. 树上的数 以及 [统一省选 2022 Day 2] C. 最大权独立集问题 两道题目，因而笔者对本题中逐一删边交换点权的模型有一定的认识和理解。在笔者成功解决 [BalticOI 2016 Day 2] C. Swap 一道题目之后，立刻认识到了此题具有进一步加强的巨大潜力，于是命制了《世界沉睡童话》一题。

笔者解决第 3 小问利用了分散层叠算法优化 AKEE 线段树。所谓『AKEE 线段树』的说法来源于中国人民大学附属中学陈于思学长，经过向昱帆学长在 https://www.cnblogs.com/dmoransky/p/14957063.html 中的介绍被笔者所了解。而本题中对于长度 $> B$ 的线段树结点所使用的分散层叠算法则是使用了蒋明润 2020 年信息学奥林匹克中国国家集训队论文《浅谈利用分散层叠算法对经典分块问题的优化》中第 3.2 小节所阐述的对动态区间排名问题所介绍的分散层叠算法，将原问题优化到了 $\mathcal{O}(n\sqrt{n})$ 的时间复杂度。