《世界沉睡童话》命题报告

叶李蹊

中国人民大学附属中学

January 14, 2024

题目大意

给定一棵大小为 n 的有标号有根树 T，其中结点 Rt 是 T 的根。
设最初结点 u 的点权是 a_u，保证 $a_1 \sim a_n$ 构成一个 $1 \sim n$ 的排列。
注意一个结点的儿子之间是有顺序的。设结点 u 有 k_u 个儿子，它们从左到右依次称为 $\mathrm{ch}(u,1), \mathrm{ch}(u,2), \cdots\cdots, \mathrm{ch}(u,k_u)$。
由于结点的儿子之间有顺序，以 Rt 为根 DFS 这棵树将得到唯一一个 DFS 序列 s，其中结点 s_i 第 i 个被访问到。
接下来进行 $n-1$ 次删边操作，第 i 次删边操作删去 $\left(s_{i+1}, \mathrm{Fa}(s_{i+1})\right)$。
在此之前，你需要选择是否交换 $a_{s_{i+1}}$ 与 $a_{\mathrm{Fa}(s_{i+1})}$。
当所有边尽数被删去以后，设最终结点 u 的点权是 a'_u。定义 S 表示所有可能的排列 $a'_1 \sim a'_n$ 构成的集合。
最后给定一个 $1 \sim n$ 的排列 p，你需要在下列三个小问中选择一个回答：

1. 判定 p 是否 $\in S$；
2. 求出 S 中字典序大于 p 的最小排列 q；
3. 求出 S 中字典序小于 p 的排列个数，对 998244353 取模。

数据范围

以下是各个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	10	20			
2	10	8×10^4	完全二叉树	DFS 序 $1 \sim n$	
3	10	8×10^4	完全二叉树		2
4	10	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	10	8×10^4			$1 \sim 6$
8	10	1.2×10^5			$1 \sim 7$
9	10	1.6×10^5			$1 \sim 8$
10	10	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。

评分标准

在一个测试点中：

- 如果你正确回答了第 1 小问，将获得该测试点满分 10% 的分数；
- 如果你正确回答了第 2 小问，无论第 1 小问是否回答、回答正确与否，都将获得该测试点满分 70% 的分数；
- 如果你正确回答了第 3 小问，无论第 1, 2 小问是否回答、回答正确与否，都将获得该测试点满分 100% 的分数。

定义与记号

- 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。

Definition

染色方案 假如一条边在被删去之前交换了两端点的点权，我们把这条边染黑；否则把这条边染白。我们记 c_u 表示 $\left(u, \mathrm{Fa}(u)\right)$ 这条边的颜色：$c_u = 1$ 表示黑色，$c_u = 0$ 表示白色。

- 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。我们用染色方案代指选择方案。
- 根据一种染色方案交换点权，称最终点权构成的排列 p 为该种染色方案导出的结果。

不同染色方案导出的结果一定不同

Theorem

$|S| = 2^{n-1}$。即任意两种不同的染色方案导出的结果也一定不同。

Proof.

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 $c_u = 0$，在第二种方案中 $c_u = 1$。

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。 □

第一小问的做法

我们可以由此得出第 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 $= t_u + \mathsf{siz}(u) - 1$，则说明 $\left(u, \mathsf{Fa}(u)\right)$ 必须染白；否则说明 $\left(u, \mathsf{Fa}(u)\right)$ 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。时间复杂度 $O(n)$，可以获得 10 分。

Definition (部分染色方案)

在一个部分染色方案当中，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。对于第三种情况，我们记 $c_u = ?$。

Definition (自由度)

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 cur，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^{cur} 种，且这 2^{cur} 种完整的染色方案导出的结果一定两两不同。

点权的运输

固定一个染色方案，考虑交换点权的过程。

Definition (始发地和目的地)

对于同一个权值 x，如果最初结点 u 拥有点权 x，最终结点 v 拥有点权 x，我们称权值 x 从 u 运输到了 v，称 v 是 u 的目的地，u 是 v 的始发地，记 $\mathsf{destination}(u) = v$，$\mathsf{origin}(v) = u$。

点权的运输

Definition (运输路线)

在交换点权的过程中，x 一定依次经过从 u 到 v 的树链上所有的结点各一次，我们称树链 $u \rightsquigarrow v$ 是权值 x 的运输路线。

设树链 $u \rightsquigarrow v$ 上的结点依次为 $h_1 \sim h_m$。拿出其中一个 h_i 来看：

- h_i 的 $k_{h_i}+1$ 条邻边被删去的顺序为：

$\big(\mathrm{h}_i, \mathrm{Fa}(h_i)\big), \big(h_i, \mathrm{ch}(h_i,1)\big), \big(h_i, \mathrm{ch}(h_i,2)\big), \cdots\cdots, \big(h_i, \mathrm{ch}(h_i,k_{h_i})\big)$。

- h_i 作为一个枢纽结点，起到中转权值的作用：在权值 x 首次到达结点 h_i 之后，找到 h_i 的邻边中第一条尚未被删去的黑边，那么 x 将被运往这条边的另一个端点。

运输路线的正序描述

钦定 $\text{destination}(u) = v$，以下是 $\text{destination}(u) = v$ 对染色方案的限制：

- 所有 (h_i, h_{i+1}) 必须染黑；
- 对于 h_1，(h_1, h_2) 是 h_1 的邻边中第一条黑边；
- 对于 h_m，(h_{m-1}, h_m) 是 h_m 的邻边中最后一条黑边；
- 对于中间的 h_i，(h_{i-1}, h_i) 和 (h_i, h_{i+1}) 是 h_i 的邻边中相邻两条黑边，其中 (h_{i-1}, h_i) 在前，(h_i, h_{i+1}) 在后。

例外：当 $u = v$ 时要求 u 的所有邻边染白。

运输路线的倒序描述

钦定 $\text{origin}(u) = v$，以下是 $\text{origin}(u) = v$ 对染色方案的限制：

- 所有 (h_i, h_{i+1}) 必须染黑；
- 对于 h_1，(h_1, h_2) 是 h_1 的邻边中最后一条黑边；
- 对于 h_m，(h_{m-1}, h_m) 是 h_m 的邻边中第一条黑边；
- 对于中间的 h_i，(h_{i-1}, h_i) 和 (h_i, h_{i+1}) 是 h_i 的邻边中相邻两条黑边，其中 (h_i, h_{i+1}) 在前，(h_{i-1}, h_i) 在后。

例外：当 $u = v$ 时要求 u 的所有邻边染白。

Theorem

钦定 $\mathit{origin}(u) = v$ 对染色方案的限制是两个集合 E_0, E_1，要求 E_0 中的边全部染白，E_1 中的边全部染黑。E_0, E_1 的限制既是充分条件，也是必要条件。

追溯过程的描述

固定一个染色方案，我们追溯出 origin(u) 的值。
我们知道 (h_i, h_{i+1}) 是 h_i 的邻边中位于 (h_{i-1}, h_i) 之前的第一条黑边。
利用这一点，我们可以从 $h_1 = u$ 开始，顺藤摸瓜追溯出 $h_1 \sim h_m$。
注意到如果 h_{i+1} 是 h_i 的儿子，那么追溯到 h_{i+1} 就结束了！因此在 h_m 之前，h_{i+1} 一直是 h_i 的父亲。

追溯过程的描述

经过整理，我们得到以下描述：

- 倒序访问 u 到其所有儿子的连边，然后访问 $\big(u, \mathrm{Fa}(u)\big)$；
- 倒序访问 $\mathrm{Fa}(u)$ 到其所有位于 u 左侧儿子的连边，然后访问 $\big(\mathrm{Fa}(u), \mathrm{anc}(u, 2)\big)$；
- 倒序访问 $\mathrm{anc}(u, 2)$ 到其所有位于 $\mathrm{Fa}(u)$ 左侧儿子的连边，然后访问 $\big(\mathrm{anc}(u, 2), \mathrm{anc}(u, 3)\big)$；

以此类推。

追溯过程的描述

追溯过程结束的条件有两个：

1. 访问一个结点到其儿子的连边 (u, v) 时，如果 $c_v = 1$，则追溯过程在 v 结束；
2. 访问一个结点到其父亲的连边 $\big(u, \mathrm{Fa}(u)\big)$ 时，如果 $c_u = 0$，则追溯过程在 u 结束。

称第一种边的预期颜色是白色，第二种边的预期颜色是黑色。

追溯过程的描述

设将要访问的边依次为 $e_1 \sim e_m$，它们的预期颜色分别为 $c^*_{e_1} \sim c^*_{e_m}$。访问到第一条 $c_{e_i} \neq c^*_{e_i}$ 的边时，追溯过程结束，$\mathrm{origin}(u)$ 是这条边靠下的端点。

由于 $\big(\mathrm{Rt}, \mathrm{Fa}(\mathrm{Rt})\big)$ 一定是白边，不符合预期颜色黑色，故而追溯过程一定会结束。

Definition (追溯过程)

总结上述的讨论，我们定义针对 $\mathrm{origin}(u)$ 的追溯过程是一个二元组序列 $(e_1, c^*_{e_1}) \sim (e_m, c^*_{e_m})$，按照访问的顺序依次表示每条边靠下的端点以及每条边的预期颜色。

逐步确定染色方案

固定 $p_u = x$ 意味着钦定 $\text{origin}(u) = b_x$。
我们维护一个部分染色方案，初始时所有 $c_u = ?$。随着对排列 p 的扫描，逐步确定染色方案。
我们需要提供以下接口：

- 操作 $\text{fix}(u, c)$ 表示将 $\big(u, \text{Fa}(u)\big)$ 染成颜色 c；
- 操作 $\text{assign}(u, x)$ 表示固定 $p_u = x$，即钦定 $\text{origin}(u) = b_x$。
  $\text{assign}(u, x)$ 可能与现有的部分染色方案矛盾。假如 $\text{assign}(u, x)$ 希望将确定的白边染黑，或者将确定的黑边染白，那么 $\text{assign}(u, x)$ 就是矛盾的。
  对于没有矛盾的 $\text{assign}(u, x)$，记 $\text{reduce}(u, x)$ 表示 $\text{assign}(u, x)$ 将会确定的无色边个数，即自由度的减小量。
- 记 $\text{valid}(u)$ 表示当前执行 $\text{assign}(u, x)$ 不会导致矛盾的 x 构成的集合。

第二小问的做法

首先求出 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。

从 1 到 n 依次考虑每个结点。考虑到 u 的时候：

1. 如果 $\max \mathrm{valid}(u) > p_u$，说明 p_u 可以 $\neq q_u$；
2. 执行 $\mathrm{assign}(u, p_u)$。假如 $\mathrm{assign}(u, p_u)$ 矛盾，立刻跳出循环。

记 λ 表示最大的 u 使得 p_u 可以 $\neq q_u$（假如 λ 不存在则输出 -1）。重置所有 $c_u = ?$，再次从 1 到 n 依次考虑每个结点。考虑到 u 的时候：

1. 对于 $u < \lambda$，直接执行 $\mathrm{assign}(u, p_u)$；
2. 对于 $u = \lambda$，求出 $\mathrm{valid}(u)$ 中 $> p_u$ 的最小元素 x，然后执行 $\mathrm{assign}(u, x)$；
3. 对于 $u > \lambda$，记 $x = \min \mathrm{valid}(u)$，执行 $\mathrm{assign}(u, x)$。

唯一 $u = \lambda$ 的那一步可以直接暴力。因此后续数据结构优化需要解决的问题有三个：

1. 执行 $\mathrm{assign}(u, x)$；
2. 求出 $x = \min \mathrm{valid}(u)$；
3. 求出 $x = \max \mathrm{valid}(u)$。

第三小问的做法

维护当前自由度 cur 和累计答案 ans，初始时 $\text{cur} = n - 1$，$\text{ans} = 0$。从 1 到 n 依次考虑每个结点。考虑到 u 的时候：

1. 对于 $\text{valid}(u)$ 中每个 $< p_u$ 的 x，将 $2^{\text{cur}-\text{reduce}(u,x)}$ 计入答案；
2. 执行 $\text{assign}(u, p_u)$。假如 $\text{assign}(u, p_u)$ 矛盾，立刻跳出循环。

对于第三小问，后续数据结构优化需要解决的问题有两个：

1. 执行 $\text{assign}(u, x)$；
2. 求出 $\text{contribution}(u) = \sum_{x \in \text{valid}(u)} [x < p_u] 2^{\text{cur}-\text{reduce}(u,x)}$。

上述做法的暴力实现

针对 $\mathrm{origin}(u)$ 进行追溯。设访问到的边靠下的端点依次为 $e_1 \sim e_m$，它们的预期颜色分别为 $c^*_{e_1} \sim c^*_{e_m}$。

- 如果 $c_{e_i} \neq ?$ 并且 $c_{e_i} = c^*_{e_i}$，那么称 e_i 是绿色的；
- 如果 $c_{e_i} \neq ?$ 并且 $c_{e_i} \neq c^*_{e_i}$，那么称 e_i 是红色的；
- 如果 $c_{e_i} = ?$，那么称 e_i 是未染色的；

暴力找出 $e_1 \sim e_m$。如果追溯过程在 e_i 结束，说明 $\mathrm{origin}(u) = e_i$，即 $a'_u = a_{e_i}$。

- $\mathrm{assign}(u, x)$ 设 $a_{e_i} = x$。如果这样的 i 不存在则 $\mathrm{assign}(u, x)$ 是矛盾的。否则将 e_i 染成红色，并将 $e_1 \sim e_{i-1}$ 染成绿色。
- $\mathrm{reduce}(u, x)$ 等于 $e_1 \sim e_i$ 这 i 条边中未染色的个数。
- $\mathrm{valid}(u)$ 找到第一条红色边 e_i。那么 $\mathrm{valid}(u)$ 等于 $e_1 \sim e_i$ 中所有未染色边构成的集合，再并上 e_i。

这里染红一条边指的是通过 fix 操作将其染成不等于预期颜色的那种黑白颜色。染绿指的是染成预期颜色。

对于完全二叉树的特殊情形，时间复杂度 $O(n \log n)$，可以获得 30 分。

DFS 序 $1 \sim n$ 的情况

对于 DFS 序 $1 \sim n$ 的特殊情况，我们可以随着 $1 \sim n$ 的扫描，实时维护针对 $\text{origin}(u)$ 进行追溯的过程。

Definition (追溯过程的差异度)

我们认为针对 $\text{origin}(u)$ 的追溯过程是一个二元组序列 $(e_1, c^*_{e_1}) \sim (e_m, c^*_{e_m})$。定义针对 $\text{origin}(u)$ 的追溯过程 A 和针对 $\text{origin}(u+1)$ 的追溯过程 B 的差异度 $\text{diff}(u, u+1)$ 等于这两个序列不完全一致的前缀长度之和，即 $|A| + |B| - 2|\text{LCS}(A, B)|$。给定两个栈 A, B，通过弹出栈顶和压入栈中两种操作从 A 得到 B 所需要的最小操作次数等于 $\text{diff}(A, B)$。

DFS 序 $1 \sim n$ 的情况

Theorem

$\sum_{u=1}^{n-1} diff(u, u+1) = O(n)$。

Proof.

我们可以用下列三种操作按 DFS 序遍历一棵树：

- 下降操作：从 u 走到 $\mathrm{ch}(u,1)$。变化量是 $k_u + (k_{\mathrm{ch}(u,1)} + 1)$。
- 转换操作：从 $\mathrm{ch}(u,i)$ 走到 $\mathrm{ch}(u,i+1)$。变化量是 $(k_{\mathrm{ch}(u,i)} + 1) + (k_{\mathrm{ch}(u,i)} + 2)$。
- 上升操作：从 $\mathrm{ch}(u,k_u)$ 走到 u。变化量是 $(k_{\mathrm{ch}(u,k_u)} + 1) + 1$。

由于每个结点的度数对总差异度之和的贡献次数 ≤ 3，因此总差异度之和是 $O(n)$ 的。 □

直接使用线段树维护对 $\mathrm{origin}(u)$ 进行追溯的过程，时间复杂度 $O(n \log n)$，可以获得 30 分，结合完全二叉树 50 分。

定义与记号

记 top(u) 表示 u 所在的重链顶端，bottom(u) 表示 u 所在的重链底端。

Definition (二元组序列 circle(u))

我们定义 circle(u) 是一个长度为 $k_u + 1$ 的二元组序列。按照如下的过程：

- 倒序访问 u 到其所有儿子的连边，然后访问 $(u, \mathrm{Fa}(u))$；

其中 u 到其所有儿子的连边的预期颜色都是白色，而 $(u, \mathrm{Fa}(u))$ 的预期颜色是黑色。

定义与记号

Definition (二元组序列 $\mathrm{trace}(u, v)$)

对于树上的一条从下到上的直链 $u \rightsquigarrow v$，我们定义 $\mathrm{trace}(u, v)$ 是一个二元组序列。按照如下的过程：

- 倒序访问 $\mathrm{Fa}(u)$ 到其所有位于 u 左侧儿子的连边，然后访问 $\big(\mathrm{Fa}(u), \mathrm{anc}(u, 2)\big)$；
- 倒序访问 $\mathrm{anc}(u, 2)$ 到其所有位于 $\mathrm{Fa}(u)$ 左侧儿子的连边，然后访问 $\big(\mathrm{anc}(u, 2), \mathrm{anc}(u, 3)\big)$；
- 以此类推，直到访问完 $\big(v, \mathrm{Fa}(v)\big)$ 为止。

在这个过程中，所有结点到其儿子的连边预期颜色都是白色，所有结点到其父亲的连边预期颜色都是黑色。

Definition (二元组序列 $\mathrm{chain}(u)$)

最后我们对于每个重链顶端结点 u，定义 $\mathrm{chain}(u) = \mathrm{trace}\big(\mathrm{bottom}(u), u\big)$。

用重链剖分拆分追溯过程

针对 $\mathrm{origin}(u)$ 的追溯过程可以拆成两个部分：$\mathrm{circle}(u)$ 和 $\mathrm{trace}(u, \mathrm{Rt})$。我们知道 $u \rightsquigarrow \mathrm{Rt}$ 的直链可以划分成 $\leq \log n$ 条轻边和重链前缀。按照重链剖分，我们把 $\mathrm{trace}(u, \mathrm{Rt})$ 拆分成 $\leq \log n$ 段 $\mathrm{trace}(u, p)$ 和 $\mathrm{trace}(p, q)$ 的并，其中 $p = \mathrm{Fa}(u)$，$q = \mathrm{top}(u)$。我们有：

- $\mathrm{trace}(u, p)$ 一定是 $\mathrm{circle}(p)$ 的一个后缀，并且所有边的预期颜色也和 $\mathrm{circle}(p)$ 相一致；
- $\mathrm{trace}(p, q)$ 一定是 $\mathrm{chain}(q)$ 的一个后缀，并且所有边的预期颜色也和 $\mathrm{chain}(u)$ 相一致；

我们称这样的后缀叫做重合子段，于是我们把针对 $\mathrm{origin}(u)$ 的追溯过程拆成了 $O(\log n)$ 个重合子段。

用数据结构维护重合子段

对于每个 circle(u) 和每个重链顶端的 chain(u)，我们各开一个数据结构来维护。

由于每条边只会有效染色一次，我们可以暴力定位那些需要被染色的边进行染色。所有 n 个过程总共只会染色 $n-1$ 条边。

以 trace(p, q) 为例，假设 trace(p, q) 的第一条边在 chain(q) 中位于位置 l，那么这个重合子段就是后缀 $[l, m]$。既然预期颜色一致，我们可以直接称呼边的红色和绿色，用一条边的红色和绿色代指这条边的黑色和白色。

用数据结构维护重合子段

我们用数据结构维护重合子段：

- 我们开一个 set 维护所有无色边的位置，用于 $\mathsf{assign}(u, x)$ 定位所有无色边进行染色；
- 我们开一个 set 维护所有红色边的位置，用于寻找 l 之后第一条红色边的位置；
- 再开一棵线段树维护满足 $c_{e_i} = 1$ 或 $c_{e_i} = ?$ 之一的 a_{e_i} 的区间 max、区间 min，用于求出 $\max \mathsf{valid}(u)$ 和 $\min \mathsf{valid}(u)$。

处理一个重合子段的时间复杂度是 $O(\log n)$。由于重合子段的个数也是 $O(\log n)$ 的，单次操作的时间复杂度是 $O(\log^2 n)$。我们可以正确回答第二小问，时间复杂度 $O(n \log^2 n)$，可以获得 70 分。
使用全局平衡二叉树可以优化到 $O(n \log n)$。

单个数据结构的任务

我们将单个数据结构的任务做一个总结：

- 维护两个序列 $a_{e_1} \sim a_{e_m}$、$c_{e_1} \sim c_{e_m}$，其中 $a_{e_1} \sim a_{e_m}$ 的值不会变化，所有 c_{e_i} 初值为 ?。
- 每次修改操作会将某个 c_{e_i} 变为 0/1。
- 每次查询操作一个后缀 $[l, m]$ 和一个权值 x，查询 $\sum_{i=l}^{m} [a_{e_i} < x] w'_i \prod_{j=l}^{i-1} w_j$。其中：
  - 对于 $c_{e_i} = ?$：$w_i = w'_i = \frac{1}{2}$；
  - 对于 $c_{e_i} = 0$：$w_i = 1$，$w'_i = 0$；
  - 对于 $c_{e_i} = 1$：$w_i = 0$，$w'_i = 1$。
- 另外还需要返回 $\prod_{j=l}^{m} w_j$，以供后续查询使用。

注意到单个数据结构的任务可以归约 $x < A$ 乘 $y < B$ 和问题，因此追求一个 $O(n\sqrt{n})$ 算法是明智的。

静态情况

数点问题

静态情况指的是 c_{e_i} 具有 0/1/? 初值，但是不会变化的情况。
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 上记一个 vector M_u 表示 $a_{e_{l_u}} \sim a_{e_{r_u}}$ 排序后的结果，那么 M_u 可以由 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 直接归并得到。
查询时在 $[l, m]$ 分解为的 $O(\log n)$ 个线段树结点上各自二分即可。
假如采用主席树的办法，这种办法没有希望扩展到动态情况。

带权问题

定义一个 i 在 $[l_u, r_u]$ 中的贡献是 $w_i' \prod_{j=l_u}^{i-1} w_j$。

那么再开一个与 M_u 等长的 vector F_u 表示 $[l_u, r_u]$ 中所有 $a_{e_i} < x$ 的 i 的贡献之和，再记录 $\prod_{i=l_u}^{r_u} w_i$ 的值，即可支持 pushup。

可以看出线段树的结构处理带权问题游刃有余。

动态情况

Lemma (重链剖分的性质)

对于任意一个结点 $u \rightsquigarrow Rt$ 的直链，我们可以把 $u \rightsquigarrow Rt$ 分解为 $k \leq \log n$ 段重链前缀。假设这 k 段重链的顶端按照从上到下的顺序依次为 $z_1, z_2, \cdots, z_k$，那么一定有 $siz(z_i) \leq \frac{n}{2^{i-1}}$。

根据这个性质我们可以得到，跳到从上往下数第 i 条重链的时候，所访问的两个数据结构所维护的序列长度必然分别 $\leq \frac{n}{2^{i-1}}$。因此为了得到一个 $O(n\sqrt{n})$ 做法，只需要能在 $O(\sqrt{n})$ 的时间复杂度内处理单个数据结构的修改查询即可。

AKEE 线段树

以下线段树结构的名字叫 AKEE 线段树[1]：

- 记一个线段树结点的长度 $\mathrm{len}(u) = r_u - l_u + 1$；
- 设置一个阈值 B，只对 $\mathrm{len}(u) \le B$ 的线段树结点维护出 M_u。
- 修改的时间消耗来源于不断向上 pushup。由于单次修改重构的区间长度和 $= B + \frac{B}{2} + \frac{B}{4} + \cdots = 2B$，总复杂度 $O(B)$。
- 查询只在当前节点完全包含于查询区间且 $\mathrm{len}(u) \le B$ 时，在该点上相应查询。访问的结点分为两种：
  1. 父亲已经完全包含于查询区间，但是父亲长度 $> B$ 暴力递归下来的结点。至多 $O(\frac{n}{B})$ 个。
  2. 位于查询区间的两侧，父亲并不属于查询区间的结点。至多 $O(\log n)$ 个。

[1] https://www.cnblogs.com/dmoransky/p/14957063.html

AKEE 线段树

我们把访问到的结点个数 $O(\frac{n}{B}+\log n)$ 乘上暴力二分的 $O(\log n)$ 得到在单个数据结构上查询的时间复杂度 $O(\frac{n\log n}{B}+\log^2 n)$。

在 $O(\log n)$ 个数据结构上查询的总时间复杂度 $O(\frac{n\log n}{B}+\log^3 n)$。取 $B=O(\sqrt{n\log n})$ 平衡得时间复杂度 $O(n\sqrt{n\log n})$，并不令人满意。

AKEE 线段树是很好实现的，只需要在 pushup 和 query 的时候加一个 if 条件就行。相比于直接分块，按照线段树结点维护的区间 len 根号分治并不会带来任何复杂度优化，这样的线段树结构唯一的作用在于简化代码。

常数正常的话可以获得 $70\sim 90$ 分。

分散层叠技术

最后我们用分散层叠技术优化 AKEE 线段树。
我们希望利用 x 在父亲结点 u 上二分的结果 $O(1)$ 确定出在两个子结点 $\mathrm{lch}(u)$ 和 $\mathrm{rch}(u)$ 上二分的结果。

- 对于长度 $\leq B$ 的结点，我们只需要保留归并的过程即可做到。具体来说我们在归并 M_u 的时候顺便记录两个 vector L_u, R_u，表示归并到这个位置的时候左右子结点两个指针分别指到哪了，就可以 $O(1)$ 确定。
- 对于长度 $> B$ 的结点，我们扩展定义 M_u 表示 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 各取 1/3 的元素归并起来的结果。我们将 $M_{\mathrm{lch}(u)}$ 和 $M_{\mathrm{rch}(u)}$ 分别每隔 3 个元素取一个元素，然后归并起来。这样只需要暴力往后找 ≤ 2 个元素就可以定位到，仍然是 $O(1)$ 的。

于是只需要在线段树的根处做一次二分即可。

分散层叠技术

- 对于 $\text{len} > B$ 的部分，单次修改重构的区间长度和 $= B + \frac{2B}{3} + \frac{4B}{9} + \cdots = 3B$，时间复杂度 $O(B)$。
- 查询操作递归到长度 $\leq B$ 的结点时应该已经知道了二分位置，单个复杂度 $O(\frac{n}{B} + \log n)$，一次询问总计 $O(\frac{n}{B} + \log^2 n)$。

取 $B = O(\sqrt{n})$ 平衡得时间复杂度 $O(n\sqrt{n})$，可以获得满分 100 分。

随着对排列 p 的扫描，上述做法在线统计出了每个 i 的贡献（一个 i 的贡献指的是满足 $q_i < p_i$ 且 $\forall j < i$ 都有 $q_j = p_j$ 的合法排列 q 的个数）。

同时也有一种离线的办法可以避免分散层叠。既然已经知道了完整的排列 p，我们可以把整个统计过程中所有的染色操作和查询操作全部离线下来。这样一来，每个数据结构只需要用相对简单的序列分块实现即可。

感谢大家的聆听！

感谢大家的聆听！