NOI 2024 tree 题解

unzcjouhi

题意简介

给定一棵树和其上的 m 条点对 (a_i, b_i)，要求你给树上每条边定向，使得 a_i 不能到达 b_i，求字典序最小的定向。

吐槽环节

算法一

对于 $n \leq 15, m \leq 50$ 的测试点，我们可以暴力枚举每一条边的定向，并计算答案。

算法二

先考虑特殊性质 A。容易发现注意到如果 (a_i, b_i) 中满足 a_i 与 b_i 在树上始终不相邻，则我们可以将树上的顶点染为红色和蓝色，使得同色顶点均不相邻。

容易发现所有好的定向只有两种：将所有红色点连向蓝色点，或者将所有蓝色点连向红色点。

因此，我们只要令第一条边的定向为 0，进而确定其它边的定向。

算法三

特殊性质 B 是供 2-SAT 算法通过的。令 $x_i = 1$ 表示 $1 - i$ 边定向为 $1 \to i$。则所有的限制形如：$x_i = 0, x_i = 1$，或 $x_i = 1 \to x_j = 1, x_j = 0 \to x_i = 0$。

直接用字典序贪心将它转化为 $O(n)$ 次判定操作，可以得到 $O(n(n+m))$ 的做法。但注意到我们可以先用 dfs 搜出所有 x_i 必须为 1 或者必须为 0 的 i。之后，我们每次找到一条输入编号最小的值还未确定的 x_i，确定 x_i，并 dfs 得到由它确定的其它变量的值。这样，复杂度可以做到 $O(n+m)$。

算法四

考虑如果固定一些边的定向之后，如何（多项式地）判断是否存在一个符合条件的定向。

如果一个点对 (a_i, b_i) 满足：a_i 到 b_i 的路径上已经有一条反向的边了，a_i 已经无法到达 b_i，删去这一点对。

如果一个点对 (a_i, b_i) 不满足条件，就说明不存在一个符合条件的定向。

如果一个点对 (a_i, b_i) 满足：a_i 到 b_i 的路径上没有反向的边，且恰好有一条边没有被定向，那么这条边的定向必定与 $a_i \to b_i$ 有向路径上的定向相反。我们确定这条边的定向，并删去这个点对。

不断这样做删除点对的操作，直到不能操作了为止。那么此时，我们将所有已经定向的边缩点，就转化到了特殊性质 A 的情况。因此，符合条件的定向是存在的。

算法四

对每个点对 (a_i, b_i)，我们维护 $a_i \to b_i$ 上定向未确定的边和反向的边的个数。我们每次确定一条边的定向之后，更新所有包含这条边的点对。这样，我们可以 $O(nm)$ 的解决判定问题。

用 $O(n)$ 次判定，我们可以求出字典序最小的定向。我们自己得到了 $O(n^2m)$ 的算法。

算法五

注意到我们不用每次都推倒重做，而是一边做删除点对的操作，一边决定字典序最小的定向。

如果一个点对 (a_i, b_i) 满足：a_i 到 b_i 的路径上已经有一条反向的边了，那么不必考虑这条路径。

如果一个点对 (a_i, b_i) 满足：a_i 到 b_i 的路径上没有反向的边，且恰好有一条边没有被定向，那么这条边的定向必定与 $a_i \to b_i$ 有向路径上的定向相反。我们确定这条边的定向，并删去这条路径。

如果没有找到可以删除的点对，则我们找到第一条定向未确定的边，将其确定为 0。（类似特殊性质 A 的分析，可以证明这样好的定向依然存在）

这样复杂度可以优化到 $O(nm)$。

进一步优化的目标

我们维护一个“待处理”边的队列。我们需要做到：每次确定一条边的定向之后，找出所有新的待处理的点对 (a_i, b_i)，使得：$a_i \to b_i$ 恰好有一条边定向未确定，其它边定向与 $a_i \to b_i$ 的定向的相同。

可能的优化方法有很多。可能有分块、启发式合并、树链剖分、动态点分治，倍增，甚至随机化。后面几个个都能导向 $O((n+m)poly(\log n))$ 的算法。篇幅所限，这里我们仅介绍两种方法。

算法六

我们可以把该算法看成一个不断合并、缩点的过程。具体来说，假设 (a_i, b_i) 已经在树上相邻，则我们将其定向为 $b_i \to a_i$，并按上述操作将 a_i, b_i 两个点缩成一个点，并删去已经满足条件的点对。

如果不存在这样的 (a_i, b_i)，那么在字典序最小的方案中，第一条未确定的边定向一定是 $u_i \to v_i$ 的形式。

算法六

考虑使用启发式合并优化此算法。我们需要对每个点（或之后合并产生的“大点”）记录 (比如用 set)：在缩点后的树中所有与它相邻的点，在缩点后的树中所有含有此点的组对 (a_i, b_i)。

每次合并时，我们需要找到新的相邻 (a_i, b_i)。假设我们将 (u, v) 合并，且 u 所代表大小不小于 v。则我们将 v 的信息合并至 u 的信息。且我们需要一一检验所有含有 v 的点对 (a_i, b_i) 是否在树上相邻。此外，我们还要枚举 v 在树上所有的邻接点，判断 v 与其邻点是否需要合并。

我们很难在每次缩点后“删去”已经满足条件的边。但实际上，这个过程可以替换为：在每次处理一条边的时候，用并查集检验它在原树上是否能决定一条定向未固定的边的定向。

算法七

使用树链剖分 + 线段树。将 $a_i \to b_i$ 拆分成 $O(\log n)$ 条轻边和 $O(\log n)$ 条重链，每条重链再拆成 $O(\log n)$ 条线段树上的节点。

对线段树上每个节点代表的链，维护它含有的定向未确定的边的个数。每当个数 ≤ 1 时，检查所有 $a_i \to b_i$ 是否需要删去。

这样，每条路径被检查了 $O(\log^2 n)$。这样时空复杂度均为 $O((n+m)\log^2 n)$ 的。使用全局平衡二叉树可以强行做到 $O((n+m)\log n)$。

算法八

前述的做法可以换成倍增。对路径 $a_i \to b_i$，我们拆成从 LCA 出发或到达 LCA 的两条路径，再拆分成两条长度为 2 的幂的路径。

对 (u, k)，记录 u 到它的 2^k 级祖先的的路径上是否只有 ≤ 1 条边的定向未确定，确定的边的定向是是全向上/全向下/混合。如果有 $= 1$ 条边未确定，还有记录这条边的编号。

每个 (u, k) 由 $(u, k-1)$ 和 $(anc_{u,k-1}, k-1)$ 转移而来。每次确定一条边的定向之后，自下而上用记忆化 dfs 更新 (u, k) 的信息。(u, k) 的信息更新后，检验含有 (u, k) 的点对。

这样，每个 (u, k) 最多被更新 $O(1)$ 次，且每个点对也最多被处理 $O(1)$ 次。我们的时空复杂度可以做到 $O(n \log n + m)$。

这些个做法常数都比较大。如果谁现场想出更好的做法，欢迎现在就上台分享！

祝大家 D2 考出好成绩！谢谢大家！