树的直径与最近公共祖先

李淳风

长郡中学

2024 年 9 月 28 日

树的直径

给定一棵树，树中每条边都有一个权值，树中两点之间的距离定义为连接两点的路径上的边权之和。树中最远的两个节点之间的距离被称为树的直径，连接这两点的路径被称为树的最长链。后者通常也可称为直径，即直径是一个数值概念，也可以代指一条路径。
树的直径一般有两种做法，时间复杂度都是 $O(n)$。这里先考虑用树形 DP 来求树的直径。

树的直径

给定一棵树，树中每条边都有一个权值，树中两点之间的距离定义为连接两点的路径上的边权之和。树中最远的两个节点之间的距离被称为树的直径，连接这两点的路径被称为树的最长链。后者通常也可称为直径，即直径是一个数值概念，也可以代指一条路径。
树的直径一般有两种做法，时间复杂度都是 $O(n)$。这里先考虑用树形 DP 来求树的直径。

不妨设 1 号节点为根，“N 个点 $N-1$ 条边的无向图”就可以看作有根树。设 $D[x]$ 表示从 x 出发走向以 x 为根的子树，所能到达的最远距离，转移就是考虑 x 的每个儿子节点，枚举 x 往哪个儿子走并统计最大值。
接下来我们考虑对于每个节点 x 求出经过点 x 的最长链的长度 $F[x]$。
显然，经过 x 的最长链的长度由四个部分组成：从 y_i 到 y_i 子树的最长距离，边 (x, y_i)，边 (x, y_j)，从 y_j 到 y_j 子树的最长距离。
需要注意的是，我们并不需要枚举 i, j。回顾计算 $D[x]$ 的过程，容易发现在枚举到 y_j 时，$D[x]$ 恰好保存了从 x 到之前遍历过的子树中的最远距离，我们可以直接使用 $D[x] + D[y_j] + edge(x, y_j)$ 来更新 $F[x]$，再用 $D[y_j] + edge(x, y_j)$ 来更新 $D[x]$。

树的直径

我们还可以通过两次 DFS 或 BFS 来求出树的直径，并且更方便计算出直径上的具体节点。这种做法包括两步：

- 从任意一个节点出发，通过 BFS 或 DFS 对树进行一次遍历，求出与出发点距离最远的节点，记为 p。
- 从节点 p 出发，通过 BFS 或 DFS 再进行一次模拟，求出与 p 距离最远的节点，记为 q。

从 p 到 q 的路径就是树的一条直径。这是因为对于树上的一个节点，距离它最远的节点一定是某条树的直径的某个端点。因此 p 一定是直径的一个端点，那么距离 p 最远的节点 q 自然就是另一个端点了。

需要注意的是，两次 DFS 求直径的方法只适用于边权均为非负的情况。如果树上存在负权边，那么刚才的结论就不再成立，根据第一次 DFS 选择的起点不同，可能会找到不同的“端点 p”，进而得到不同的直径，可能得到错误解。此时只能使用树形 DP 算法来求树的直径。

例题

巡逻

在一个地区中有 n 个村庄，编号为 $1, 2, \cdots, n$。有 $n-1$ 条道路连接着这些村庄，每条道路刚好连接两个村庄，从任何一个村庄，都可以通过这些道路到达其他任何一个村庄。每条道路的长度均为 1 个单位。
为保证该地区的安全，巡警车每天要到所有的道路上巡逻。警察局设在编号为 1 的村庄里，每天巡警车总是从警察局出发，最终又回到警察局。
为了减少总的巡逻距离，该地区准备在这些村庄之间建立 K 条新的道路，每条新道路可以连接任意两个村庄，甚至可以是一个环。同时，为了不浪费资金，每天巡警车必须经过新建的道路正好一次。
现在给定村庄间道路的信息和需要新建的道路数，请你计算出最佳的新建道路的方案，使得总的巡逻距离最小，并输出这个最小的巡逻距离。$3 \leq n \leq 10^5, 1 \leq K \leq 2$。

例题

巡逻

在一个地区中有 n 个村庄，编号为 $1, 2, \cdots, n$。有 $n-1$ 条道路连接着这些村庄，每条道路刚好连接两个村庄，从任何一个村庄，都可以通过这些道路到达其他任何一个村庄。每条道路的长度均为 1 个单位。
为保证该地区的安全，巡警车每天要到所有的道路上巡逻。警察局设在编号为 1 的村庄里，每天巡警车总是从警察局出发，最终又回到警察局。
为了减少总的巡逻距离，该地区准备在这些村庄之间建立 K 条新的道路，每条新道路可以连接任意两个村庄，甚至可以是一个环。同时，为了不浪费资金，每天巡警车必须经过新建的道路正好一次。
现在给定村庄间道路的信息和需要新建的道路数，请你计算出最佳的新建道路的方案，使得总的巡逻距离最小，并输出这个最小的巡逻距离。$3 \leq n \leq 10^5, 1 \leq K \leq 2$。

不建立新的道路时，按照题意会恰好经过每条边两次，路线总长度为 $2(n-1)$，因为我们在 DFS 时每条边必然被递归一次、回溯一次。

巡逻

建立一条新道路之后，因为新道路必须经过正好一次，所以在沿着新道路 (x, y) 巡逻之后，要返回 x，就必须沿着树上从 y 到 x 的路径巡逻一遍，最终形成一个环。

因此当 $K = 1$ 时，我们找到树的最长链，在两个端点之间加一条新道路，就能让总距离最小。若树的直径为 L，答案就是 $2(n-1) - L + 1$。

巡逻

建立一条新道路之后，因为新道路必须经过正好一次，所以在沿着新道路 (x, y) 巡逻之后，要返回 x，就必须沿着树上从 y 到 x 的路径巡逻一遍，最终形成一个环。

因此当 $K = 1$ 时，我们找到树的最长链，在两个端点之间加一条新道路，就能让总距离最小。若树的直径为 L，答案就是 $2(n-1) - L + 1$。

建立第二条新道路 (u, v) 之后，又会形成一个环。若两条新道路形成的环不重叠，则树上 u, v 之间的路径只会经过一次，答案继续减小。否则，在两个环重叠的情况下，如果我们还按照刚才的方法把第 2 个环与建立 1 条新道路的情况相结合，两个环重叠的部分就不会被巡逻到。因此我们不得不让巡逻车在适当的时候重新巡逻这些边两次。

综上所述，我们得到了如下算法：

- 在最初的树上求直径，设直径为 L_1。然后把直径上的边权取反（从 1 变为 -1）。
- 在最长链边权取反之后的树上再次求直径，设直径为 L_2。

最终的答案就是 $2(n-1) - (L_1 - 1) - (L_2 - 1) = 2n - L_1 - L_2$。若 L_2 包含 L_1 取反的部分，就相当于两个环重叠。减掉 $(L_1 - 1)$ 后，重叠的部分变为只经过一次，减掉 $(L_2 - 1)$ 后，相当于把重叠的部分加回来，变为需要经过两次。

时间复杂度为 $O(n)$。注意第二次求直径有负权边，应该使用树形 DP。

例题

消防

某个国家有 n 个城市，这 n 个城市中任意两个都连通且有唯一一条路径，每条连通两个城市的道路的长度为 z_i 。
这个国家的人对火焰有超越宇宙的热情，所以这个国家最兴旺的行业是消防业。由于政府对国民的热情忍无可忍（大量的消防经费开销）可是却又无可奈何（总统竞选的国民支持率），所以只能想尽方法提高消防能力。
现在这个国家的经费足以在一条边长度和不超过 s 的路径（两端都是城市）上建立消防枢纽，为了尽量提高枢纽的利用率，要求其他所有城市到这条路径的距离的最大值最小。
你受命监管这个项目，你当然需要知道应该把枢纽建立在什么位置上。
$1 \leq n \leq 3 * 10^5, 1 \leq z_i \leq 10^3$ 。

例题

消防

某个国家有 n 个城市，这 n 个城市中任意两个都连通且有唯一一条路径，每条连通两个城市的道路的长度为 z_i 。
这个国家的人对火焰有超越宇宙的热情，所以这个国家最兴旺的行业是消防业。由于政府对国民的热情忍无可忍（大量的消防经费开销）可是却又无可奈何（总统竞选的国民支持率），所以只能想尽方法提高消防能力。
现在这个国家的经费足以在一条边长度和不超过 s 的路径（两端都是城市）上建立消防枢纽，为了尽量提高枢纽的利用率，要求其他所有城市到这条路径的距离的最大值最小。
你受命监管这个项目，你当然需要知道应该把枢纽建立在什么位置上。
$1 \leq n \leq 3 * 10^5, 1 \leq z_i \leq 10^3$ 。

容易发现这道题就是“树网的核”的加强版。因为这条路径需要建立在直径上才能使答案最小，同时如果有多条直径，在任意一条直径上都一样。

消防

$O(n^3)$ 的枚举直接掠过。考虑当路径的一个端点 p 固定后，另一个端点 q 在距离不超过 s 的前提下，显然越远越好。因此我们只需在直径上枚举 p，然后直接确定 q 的位置，再通过 DFS$O(n)$ 算一遍所有点到这条路径的距离最大值，就在 $O(n^2)$ 的时间内解决的这道题。

但这还不够，我们接着考虑能否接着优化。容易发现，本题的答案具有单调性，可以二分答案，把问题转化为“验证是否存在一条路径，使所有点中最远距离不超过 mid”。设直径的两个端点为 u, v，我们在直径上找到与 u 距离不超过 mid 的情况下，距离 u 最远的节点作为路径的一个端点 p，以及同理找到另一个端点 q。

这样找到的两个端点在满足直径两侧的那部分节点到路径距离不超过 mid 的前提下，尽量靠近直径的中心。再判断一下 p, q 之间的距离是否超过 s，以及从 p, q 这条路径出发进行 DFS 判断是否存在距离路径大于 mid 的节点即可。该算法时间复杂度为 $O(n \log SUM)$。

消防

我们还有一种不使用二分的做法，通过单调队列把时间复杂度优化到 $O(n)$。

设直径上的节点为 $u_1, u_2, \cdots, u_t$。与前面几个解法类似，先把这 t 个节点标记为“已访问”，然后通过深度优先遍历，求出 $d[u_i]$，表示从 u_i 出发，不经过直径上的其它节点，能够到达的最远点的距离。

这样一来，以 $u_i, u_j (i \leq j)$ 为端点的路径，所有点到它的距离最大值为：

$$\max\left(\max_{i \leq k \leq j}\{d[u_k]\}, dist(u_1, u_i), dist(u_j, u_t)\right)$$

此时用单调队列维护区间最大值，已经能够做到 $O(n)$ 了。但是还不够完美，根据直径的最长性，还可以简化为：

$$\max\left(\max_{1 \leq k \leq t}\{d[u_k]\}, dist(u_1, u_i), dist(u_j, u_t)\right)$$

$\max_{1 \leq k \leq t}\{d[u_k]\}$ 是一个定值，因此我们只需要枚举直径上的每个点 u_i，在距离不超过 s 的情况下，维护距离 u_i 最远的点 u_j。对于每个 i，j 是单调递增的，使用单调指针维护即可。总复杂度 $O(n)$。

最近公共祖先（LCA）

给定一棵有根树，若节点 z 既是节点 x 的祖先，又是节点 y 的祖先，则称 z 是 x, y 的公共祖先。在 x, y 的所有公共祖先中，深度最大的一个称为 x, y 的最近公共祖先，记为 $LCA(x, y)$。
$LCA(x, y)$ 是 x 到根的路径和 y 到根的路径的交汇点。它也是 x 和 y 之间的路径上的深度最小的点。求最近公共祖先的方法通常有以下几种：

向上标记法
从 x 向上走到根节点，并标记所有的节点。
从 y 向上走到根节点，当第一遇到已标记的节点时，就找到了 $LCA(x, y)$。
对于每个询问，向上标记法的时间复杂度最坏为 $O(n)$。

最近公共祖先（LCA）

树上倍增法

树上倍增法是一个很重要的算法，除了求 LCA 之外，还在很多问题中都有广泛的应用。设 $f[x][k]$ 表示 x 的 2^k 倍祖先，即从 x 出发向根节点走 2^k 步到达的节点。特别地，若该节点不存在，则 $f[x][k] = 0$。$f[x][0]$ 就是 x 的父节点，$f[x][k] = f[f[x][k-1]][k-1]$。

以上部分是预处理，时间复杂度为 $O(n \log n)$，之后可以多次对不同的 x, y 计算 LCA，每次询问的时间复杂度是 $O(\log n)$。

基于 f 数组计算 $LCA(x, y)$ 分为以下几步：

- 设 $d[x]$ 表示 x 的深度。不妨设 $d[x] \geq d[y]$（否则交换 x, y）。
- 用二进制拆分思想，把 x 向上调整到与 y 同一深度。
- 若此时 $x = y$，说明已经找到了 LCA，LCA 就等于 y。
- 用二进制拆分思想，把 x, y 同时向上调整，并保证二者深度一致且二者不相会。
- 此时 x, y 只差一步就相会了，它们的父节点 $f[x][0]$ 就是 LCA。

最近公共祖先（LCA）

LCA 的 Tarjan 算法

Tarjan 算法本质上是使用并查集对“向上标记法”的优化。它是一个离线算法，需要把 m 个询问一次性读入，统一计算，最后统一输出。

在深度优先遍历的任意时刻，树中结点分为三类：

- 已经访问完毕并且回溯的节点。在这些节点上标记一个整数 2。
- 已经开始递归，但尚未回溯的节点。这些节点就是当前正在访问的节点 x 以及 x 的祖先。在这些节点上标记一个整数 1。
- 尚未访问的节点。这些节点没有标记。

对于正在访问的节点 x，它到根节点的路径已经标记为 1。若 y 是已经访问完毕并且回溯的节点，则 $LCA(x, y)$ 就是从 y 向上走到根，第一个遇到的标记为 1 的节点。

因此我们可以使用并查集进行优化，当一个节点获得整数 2 的标记时，把它所在的集合合并到它的父节点所在的集合中（合并时它的父节点标记一定为 1 且单独构成一个集合）。

这样相当于每个完成回溯的节点都有一个标记指向它的父节点。当我们 DFS 到 x 时，如果 y 已经回溯，只需查询 y 所在集合的代表元素，就等价于从 y 向上一直到走到一个开始递归但尚未回溯的节点（具有标记 1），即 $LCA(x, y)$。此时扫描与 x 相关的所有询问并进行处理。

树上差分

LCA 的 Tarjan 算法在之前的学习当中，我们定义了一个序列的前缀和序列和差分序列，并通过差分的技巧，把“区间”的增减转化为“左端点加 1，右端点减 1”。根据“差分序列的前缀和是原序列”这一原理，在树上可以进行类似的简化，其中“区间操作”对应为“路径操作”，“前缀和”对应为“子树和”。

例题

暗の連鎖

传说中的暗之连锁被人们称为 Dark。
Dark 是人类内心的黑暗的产物，古今中外的勇者们都试图打倒它。
经过研究，你发现 Dark 呈现无向图的结构，图中有 N 个节点和两类边，一类边被称为主要边，而另一类被称为附加边。
Dark 有 $N-1$ 条主要边，并且 Dark 的任意两个节点之间都存在一条只由主要边构成的路径。另外，Dark 还有 M 条附加边。
你的任务是把 Dark 斩为不连通的两部分。
一开始 Dark 的附加边都处于无敌状态，你只能选择一条主要边切断。
一旦你切断了一条主要边，Dark 就会进入防御模式，主要边会变为无敌的而附加边可以被切断。
但是你的能力只能再切断 Dark 的一条附加边。
现在你想要知道，一共有多少种方案可以击败 Dark。
注意，就算你第一步切断主要边之后就已经把 Dark 斩为两截，你也需要切断一条附加边才算击败了 Dark。
$N \leq 10^5, M \leq 2 * 10^5$。保证答案不超过 $2^{31} - 1$。

暗之连锁

根据题意，“主要边”构成一棵树，“附加边”则是非树边。把一条附加边 (x, y) 添加到主要边构成的树中，会与树上 x, y 之间的路径一起形成一个环。如果第一步选择切断 x, y 之间的某条边，那么第二步就必须切断附加边 (x, y)，才能令 Dark 被斩断为不连通的两部分。

因此，我们称每条附加边 (x, y) 都把树上 x, y 之间的路径“覆盖了一次”。我们只需统计出每条“主要边”被覆盖了多少次。若第一步把被覆盖 0 次的主要边切断，则第二步可以任意切断一条附加边。若第一步把被覆盖次数为 0 的主要边切断，则第二步方法唯一。若第一步把被覆盖 2 次或 2 次以上的主要边切断，则第二步无论如何操作都不能击败 Dark。

综上所述，我们要解决的问题是：给定一张无向图和一棵树，求每条树边被非树边“覆盖”了多少次。解决此问题的经典方法就是“树上差分”。我们给树上每个节点一个初始为 0 的权值，对于每条非树边 (x, y)，我们把 x, y 的权值加一，把 $LCA(x, y)$ 的权值减二，最后对整棵树进行一次 DFS，求出 $F[x]$ 表示以 x 为根的子树内节点的权值之和，$F[x]$ 就是 x 到 x 父亲的边被覆盖的次数。时间复杂度为 $O(n + m)$。

例题

雨天的尾巴

村落里的一共有 n 座房屋，并形成一个树状结构。然后救济粮分 m 次发放，每次选择两个房屋 (x, y)，然后对于 x 到 y 的路径上（含 x 和 y）每座房子里发放一袋 z 类型的救济粮。现在想知道，当所有的救济粮发放完毕后，每座房子里存放的最多的是哪种救济粮。$1 \leq n, m, z \leq 10^5$。

例题

雨天的尾巴

村落里的一共有 n 座房屋，并形成一个树状结构。然后救济粮分 m 次发放，每次选择两个房屋 (x, y)，然后对于 x 到 y 的路径上（含 x 和 y）每座房子里发放一袋 z 类型的救济粮。现在想知道，当所有的救济粮发放完毕后，每座房子里存放的最多的是哪种救济粮。$1 \leq n, m, z \leq 10^5$。

对每个点 x 建立一个大小为 m 的计数数组 $c[x][1 \sim m]$，记录第 x 个节点收到的每种物品的数量。我们同样可以使用树上差分的思想，与上一题不同的是，我们是对点进行操作，而不是对边进行操作了。因此我们转化为在差分数组 b 中，在节点 x, y 处产生 z，在节点 $LCA(x, y)$ 和 $father(LCA(x, y))$ 处消失 z。同样最后再对于每个节点统计 b 的子树和，就能得到该点的所有信息了。
为了节省空间并快速合并两个计数数组，我们可以使用“线段树合并”算法。对于每个节点 x 建立一棵动态开点的线段树代替差分数组 b，支持修改某个位置的值，并统计区间最大值以及最大值所在的位置。执行完 m 次发放操作后，进行一次 DFS，对于每个点在 DFS 完儿子节点后，把代表儿子子树的线段树进行合并即可。
时间和空间复杂度均为 $O((n+m)\log(n+m))$。

例题

天天爱跑步

小 C 同学认为跑步非常有趣，于是决定制作一款叫做《天天爱跑步》的游戏。《天天爱跑步》是一个养成类游戏，需要玩家每天按时上线，完成打卡任务。这个游戏的地图可以看作一棵包含 n 个结点和 $n-1$ 条边的树，每条边连接两个结点，且任意两个结点存在一条路径互相可达。树上结点编号为从 1 到 n 的连续正整数。

现在有 m 个玩家，第 i 个玩家的起点为 s_i ，终点为 t_i。每天打卡任务开始时，所有玩家在第 0 秒同时从自己的起点出发，以每秒跑一条边的速度，不间断地沿着最短路径向着自己的终点跑去，跑到终点后该玩家就算完成了打卡任务。(由于地图是一棵树，所以每个人的路径是唯一的)

小 C 想知道游戏的活跃度，所以在每个结点上都放置了一个观察员。在结点 j 的观察员会选择在第 w_j 秒观察玩家，一个玩家能被这个观察员观察到当且仅当该玩家在第 w_j 秒也正好到达了结点 j。小 C 想知道每个观察员会观察到多少人?

注意：我们认为一个玩家到达自己的终点后该玩家就会结束游戏，他不能等待一段时间后再被观察员观察到。即对于把结点 j 作为终点的玩家：若他在第 w_j 秒前到达终点，则在结点 j 的观察员不能观察到该玩家；若他正好在第 w_j 秒到达终点，则在结点 j 的观察员可以观察到这个玩家。

$n, m \leq 3 * 10^5$ 。

天天爱跑步

首先，每个玩家跑步的路线可以拆成两段：从 s_i 到 $lca(s_i, t_i)$，从 $lca(s_i, t_i)$ 到 t_i，其中后一段不包括 $lca(s_i, t_i)$ 这个端点。不难发现，位于节点 x 的观察员能观察到第 i 个玩家，当且仅当满足下列条件之一：

- x 处于 s_i 到 $lca(s_i, t_i)$ 的路径上，$d[s_i] - d[x] = w[x]$，其中数组 d 表示节点在树中的深度。
- x 处于 $lca(s_i, t_i)$ 到 t_i 的路径上，$d[s_i] + d[x] - 2d[lca(x_i, y_i)] = w[x]$。

因为两个条件都包含 x 对所在路径的限制且互不重叠，所以可以分开计算满足每个条件的玩家数量再进行相加。先考虑第一个条件的计算：把式子移项可以得到：$d[s_i] = d[x] + w[x]$。类比之前的题目，我们可以把问题转化为，由 m 个玩家，每个玩家给 (s_i 到 $lca(s_i, t_i)$ 路径上每个点添加一个类型为 s_i 的物品，最终求每个点 x 处类型为 $d[x] + w[x]$ 的物品数量有多少个。通过树上差分，可以视作在点 s_i 处产生物品 $d[s_i]$，在 $father(lca(s_i, t_i))$ 处物品消失。

天天爱跑步

我们同样可以使用动态开点的权值线段树维护计数数组，并用线段树合并维护子树和。但由于这道题的信息满足“可减性”，我们可以使用复杂度更优的做法。

我们对于每个点开一个 vector 存储这个节点加入/消失了哪些物品，并用一个全局数组 c 对每种类型的物品进行计数，初始全为 0。

对整棵树进行 DFS，在递归进入每个点 x 时，用一个局部变量 cnt 记录 $c[w[x]+d[x]]$。然后扫描点 x 的 vector 数组，修改 c 数组中对应的物品数量，接着继续递归 x 的所有子树。从 x 回溯之前，用新的 $c[w[x]+d[x]]$ 减去 cnt 就是“子树和”，即点 x 处类型为 $w[x]+d[x]$ 的物品的数量。这是因为我们只要查询子树内区间内类型为 $w[x]+d[x]$ 的物品的数量，而数量是可以由两个前缀和相减得到的。

对于第二个条件，只需改为物品 $d[s_i]-2d[lca(s_i,t_i)]$ 在点 t_i 处产生，在点 $lca(s_i,t_i)$ 处消失，最后求每个点 x 处类型为 $w[x]-d[x]$ 的物品数量。注意此时数组下标可能为负数，需要对下标范围进行平移或离散化。两个条件得到的结果相加，就是最终的答案。

例题

异象石

Adera 是 Microsoft 应用商店中的一款解谜游戏。
异象石是进入 Adera 中异时空的引导物，在 Adera 的异时空中有一张地图。
这张地图上有 N 个点，有 $N-1$ 条双向边把它们连通起来。
起初地图上没有任何异象石，在接下来的 M 个时刻中，每个时刻会发生以下三种类型的事件之一：

- 地图的某个点上出现了异象石（已经出现的不会再次出现）；
- 地图某个点上的异象石被摧毁（不会摧毁没有异象石的点）；
- 向玩家询问使所有异象石所在的点连通的边集的总长度最小是多少。

请你作为玩家回答这些问题。$1 \leq N, M \leq 10^5$。

异象石

根据题意，这张地图显然构成一棵树。先对这棵树进行 DFS，求出每个点的时间戳。仔细思考可以发现，如果我们按照时间戳从小到大的顺序，把出现异象石的节点排成一圈（首尾相连），并且累加相邻两个节点之间的路径长度，最后得到的结果恰好是所求答案的两倍。

因此，我们可以用一个数据结构（例如 set），按照时间戳递增的顺序，维护出现异象石的节点序列，并用一个变量 ans 来记录序列中相邻两个节点之间的路径长度之和（序列首尾也算相邻）。

设 $path(x, y)$ 表示树上 x, y 之间的路径长度，设 $d[x]$ 表示 x 到根节点的路径长度。我们有 $path(x, y) = d[x] + d[y] - 2d[lca(x, y)]$。$d$ 数组可以通过 DFS 求出，lca 可以使用树上倍增法进行计算。

若一个节点出现的异象石，就依据时间戳，把它插入到对应的位置；如果一个节点的异象石被摧毁就把它从序列中删除。并且在插入和删除时维护 ans 的值，询问时直接输出即可。

时间复杂度为 $O((N + M) \log N)$。

例题

次小生成树

给定一张 N 个点 M 条边的无向图，求它的严格次小生成树。设最小生成树的边权之和为 sum，严格次小生成树就是指边权值和大于 sum 的生成树中边权和最小的一个。
$N \leq 10^5, M \leq 3 * 10^5$。

例题

次小生成树

给定一张 N 个点 M 条边的无向图，求它的严格次小生成树。设最小生成树的边权之和为 sum，严格次小生成树就是指边权值和大于 sum 的生成树中边权和最小的一个。
$N \leq 10^5, M \leq 3 * 10^5$。

先求出任意一棵最小生成树，设边权值和为 sum。我们称在这棵最小生成树中的 $N-1$ 条边为“树边”，其他 $M-N+1$ 条边为“非树边”。
把一条非树边 (x, y, z) 添加到最小生成树中，会与树上 x, y 之间的路径一起形成一个环。
设树上 x, y 之间的最大边权为 val_1，严格次大边权为 $val_2 (val_1 > val_2)$：
若 $z > val_1$，则把 val_1 对应的那条边替换成 (x, y, z) 这条边，就得到了严格次小生成树的一个候选答案，边权值和为 $sum - val_1 + z$;
若 $z = val_1$，则把 val_2 对应的那条边替换成 (x, y, z) 这条边，就得到了严格次小生成树的一个候选答案，边权值和为 $sum - val_2 + z$。

次小生成树

因此我们只需要枚举每条“非树边”，计算出上述所有“候选答案”，找出一个最小值即可得到整张图的严格次小生成树。因此，我们要解决的问题就是，如何快速求出一条路径上的最大边权与严格次大边权。
我们同样可以使用树上倍增来进行预处理。用 $f[x][k]$ 来表示 x 的 2^k 辈祖先，$g[x][k][0]$ 与 $g[x][k][1]$ 分别表示从 x 到 $f[x][k]$ 路径上的最大边权和严格次大边权。在求 $f[x][k]$ 的时候把四个对应的 $g[][k-1][0/1]$ 拿出来比较一下，就能求出 $g[x][k][0/1]$ 了。
之后需要计算路径 (x, y) 上的最大值和严格次大值，照样采用倍增求 lca 的框架，每次 x/y 向上移动时把对应的最大值、严格次大值仿照预处理时的操作合并到一起，即可得到 (x, y) 路径上的最大边权和严格次大边权。
整个算法的时间复杂度为 $O(M \log N)$。

例题

疫情控制

H 国有 n 个城市，这 n 个城市用 $n-1$ 条双向道路相互连通构成一棵树，1 号城市是首都，也是树中的根节点。
H 国的首都爆发了一种危害性极高的传染病。当局为了控制疫情，不让疫情扩散到边境城市（叶子节点所表示的城市），决定动用军队在一些城市建立检查点，使得从首都到边境城市的每一条路径上都至少有一个检查点，边境城市也可以建立检查点。但特别要注意的是，首都是不能建立检查点的。
现在，在 H 国的一些城市中已经驻扎有总共 m 支军队，且一个城市可以驻扎多个军队。一支军队可以在有道路连接的城市间移动，并在除首都以外的任意一个城市建立检查点，且只能在一个城市建立检查点。一支军队经过一条道路从一个城市移动到另一个城市所需要的时间等于道路的长度（单位：小时）。
请问最少需要多少个小时才能控制疫情。注意：不同的军队可以同时移动。
$2 \leq m \leq n \leq 5 * 10^4$。

疫情控制

为了叙述方便，若从首都（根节点）到边境城市（叶子节点）x 的图中经过了军队 i，我们就称军队 i 管辖了节点 x。本题要求每个叶子节点都有军队管辖。显然，在不移动到根节点的情况下，军队所在的节点深度越浅，能管辖的叶子节点就越多。

本题的答案也满足单调性——若 ans 个小时能控制已经，那么花费更多的时间当然也能控制疫情。因此可以考虑二分答案 mid，再判定能否在 mid 小时内控制疫情。

这样我们就可以把军队分为两类。第一类是在 mid 小时内无法到达根节点的军队。对于这些军队，就让他们尽量往根节点走，mid 小时内能走到哪里，就驻扎在哪里。

记根节点的子节点集合为 $son(root)$。处理完第一类军队后，对每个节点 $s \in son(root)$，统计以 s 为根的子树中是否还有叶子节点尚未被管辖。设 H 是 $son(root)$ 中还有叶子节点尚未被管辖的节点组成的集合。

第二类是 mid 小时内能够到达根节点的军队。我们先让这些军队移动到根节点的子节点上（差一步到根节点），用三元组 $(i, s, rest)$ 代表。其中 i 表示军队编号，s 表示军队来自根节点的哪一个子节点，$rest$ 表示军队 i 移动到 s 后还剩余多少时间。这些军队要么驻扎在 s，要么跨过根节点去管辖其它的子树。

疫情控制

首先，若存在一支军队 $(i, s, rest)$ 满足 $s \in H$ 且军队 i 在 $rest$ 时间内无法从 s 移动到根再返回 s，则在最优解中，s 一定被自己子树内部的一支军队驻扎，不可能由其它子树的军队跨过根节点来驻扎。

假设 s 由来自另外一棵子树 s' 的一支军队 i' 来驻扎。因为军队 i 无法从 s 到根再返回 s，所以这支军队要么限制，要么跨过根节点驻扎在比 s 离根更近的节点 s''。但对于这种情况，军队 i' 也一定有能力去驻扎在 s''。

因此，对于这种情况，让 i 驻扎在自己的子树 s，把 i' 留作他用，未来的可能性更多，答案不会更差。

所以我们可以考虑所有满足上述条件的军队，并从 H 中删除对应的节点。现在，仍未确定驻扎地点的军队就都可以先到根节点，再去管辖其它子树。我们把剩余的军队按照“到达根节点后还剩余的时间”排序，按照到根节点的距离从小到大枚举 H 中剩余的所有节点，每次选择一个能够驻扎过来的、剩余时间最短的军队驻扎过来。这样就可以根据 H 中的节点是否全部被管辖来判断 mid 是否可行了。

整个算法的时间复杂度为 $O((n+m)\log n \log SUM)$，其中 SUM 表示所有边的长度之和。