2021 省选模拟赛

solution

作业题

算法 1（$m \leq 20$）

良心送分子任务，直接枚举每条边是否在树上即可。

算法 2（图是一棵仙人掌）

不难发现，仙人掌上不在环中的边一定会出现在生成树上。接下来只需要考虑在环上的边。

不难发现仙人掌上每一个环都是独立的，而每一个环中需要恰好删去一条边。这样一来可以直接 DP 实现。

算法 3（$w_i = 1$）

当 n 是奇数时答案显然是 0 。

当 n 是偶数时显然每一棵生成树的权值都是 1 。

那么只需要写一个生成树计数即可。

算法 4（全部子任务）

既然是生成树计数，那么肯定需要用到矩阵树定理。

矩阵树定理求出来的最后答案为所有生成树的边权之积的和，即：$\sum_T \prod_{e \in T} w_e$

由于最后需要求的是异或和，那么我们统计异或和为某个特定值的树的个数，这也非常方便。

对于每一条边权为 w_i 的边 i，我们构造一个长度为 256 的只有第 w_i 为 1，其余位为 0 的多项式 F_i，用 $\mathrm{FWT}(F_i)$ 替换 w_i，然后套用矩阵树定理就解决了。

时间复杂度 $\mathrm{O}(n^3 w_i)$ 。

社会实践

算法 1（$n \leq 3$）

显然答案只有 0,1,2,3 四种情况，分类讨论即可。

算法 2（$n \leq 12$）

不难发现可能的局面只有 $\sum_{i=1}^{n} 3^i$ 种，那么可以先通过状压 DP 求解出所有局面的答案，然后直接回答。

算法 3（$n \leq 2000$）

先尝试解决一个汉诺塔问题。

这个问题从正面思考并不好下手，考虑从反面思考。也就是说，把问题转换成以下这个等价问题：

给定一个最终局面，初始局面为 n 个圆盘都在同一根柱子上（3 种初始局面任选一种），然后通过最少的移动步数把初始局面变成最终局面。

问题转换过后就可以通过贪心算法解决。

首先 n 号圆盘一定不动，所以初始在哪根柱子上是确定的。如果 $i+1 \sim n$ 号圆盘到达最终局面了，假设 $1 \sim i$ 都还在同一根柱子上，这时候需要把 i 移动到目标位置，那么先要使用 $2^{i-1}-1$ 步把 $1 \sim i-1$ 都搬到第三根柱子上，然后再使用 1 步把 i 移动到目标位置。

经过这一过程，总共花费 2^{i-1} 步，然后 $i \sim n$ 复位了，$1 \sim i-1$ 还在同一根柱子上。这是一个子问题，可以继续做。

因为，我们可以在 $O(n)$ 的时间内解决一个汉诺塔问题，那么总时间复杂度为 $O(n^2)$。

算法 4（$n \leq 3 \times 10^5$）

如果会算法 3，那么接下来就是套路了。

考虑算法 3 的模拟过程：从右往左枚举，假设枚举到第 i 位，如果 i 以复位，那么对答案无贡献，转移到 $i-1$；否则，将所在位置改为不同于 i 所在位置和目标位置的第三根柱子，答案加上 2^{i-1}，转移到 $i-1$。

很显然，上述过程相当于一个三个点的自动机。

那么可以直接用线段树维护一下区间 $[l,r]$，从哪个节点进入，在经过该区间后到哪个节点，答案加上多少。

时间复杂度 $O(n \log n)$。

旅行

题解索引：
算法 0 为暴力算法。
算法 1 可以通过子任务 2。
算法 2 ~ 3 为同一个算法，仅常数不同，根据常数可以通过子任务 2，3。
算法 2 补充和算法 3 都为对算法 2 的优化，并无实际意义，可以直接看算法 3。
算法 4 可以通过子任务 1，可能可以通过子任务 2，较难通过子任务 3。

算法 0（$limA = limB = n^2$）

这个子任务应该不难想到做法。

首先可以假设 1 为根，然后用 $n-1$ 次 B 操作询问出每个点的子树大小。

然后维护一个和 1 相连的连通块，每次把没有加入连通块的点中子树大小最大的加入连通块，不难发现这个点与连通块一定有边直接相连。

寻找到相连的点只需要询问一下以当前点为根，某个点的子树中是不是包含整个连通块即可。

最后会用 $n-1$ 次询问 B 和 $\frac{n^2}{2}$ 次询问 A 。

然而这种做法并没有分。

算法 1

如果知道了算法 0，那么这个做法也不难想了。

很明显只需要加速算 1 中寻找点的过程。因为这是一棵树，所以不难想到点分。

每次把重心找出来，不难发现，以当前点为根，重心的子树中仅不包含它周围节点中的某一个节点。如果存在这个节点，那么相当于找到了下次点分的方向，如果不存在这个节点，那么这个点就是答案了。

点分和二分答案各占一个 log，所以最后大概会用 $n\log^2 n$ 次询问 A 。但实际上因为两个 log 的常数都非常小，所以询问远远不需要 $n\log^2 n$ 次。经实际测试，最坏情况为 1.9×10^4 ，也可能是我不会卡。

当然，如果善于利用询问 B ，可以大大减少二分的次数。因为如果每次二分前先做一遍询问 B ，那么可以排除大部分干扰路径，这样一来可以大大减少询问 A 使用的次数，但是无法通过子任务 3。

算法 2

和算法 0 一样，可以先用 $n-1$ 次询问询问出以 1 为根的每个点的子树大小。但是同算法 0 不同的是，我们可以倒过来做，采用找叶子策略。

首先对于子树大小为 1 的点肯定是叶子节点。对于子树大小不为一的点，它至少有一个节点。那么我们可以在当且这些叶子集合中二分出所有是这个点的叶子节点。

由于每个叶子节点只有一个父亲，所有被二分出的叶子节点就会从集合中删除，而当前节点会变成一个新的叶子节点加入集合。

这样理论会使用 $n\log n$ 次询问 A 和 $n-1$ 次询问 B ，实际测试大概会使用 10^4 次询问

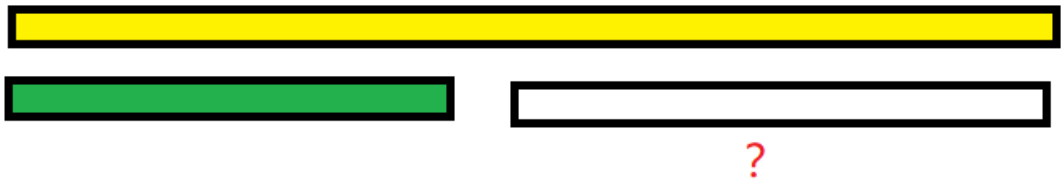
A。

算法 2 补充

算法 2 的剥叶子做法需要将查找的常数严格压缩，即需要利用已知信息将查询的常数减小。

最简单的一种查询方法就是直接分治，如果整个区间中的所有点都是或都不是叶子，那么直接退出查找，否则继续查找。但是不难发现上述做法是会带常数 2 的，因为在查找叶子的过程中另一半区间的失败也需要询问。

所以这时候需要将另一边的失败情况快速判断。



如图，一个核心问题是，如果我们知道了父亲区间属于部分有部分没有的情况，如果我们已经做完的左区间，那么右区间还需要一次询问吗？

情况 1

如果假设当且节点的所有叶子都已经被找到了，那么显然右区间中不可能有新的叶子节点，那么也没有必要再去失败一次了。

判断方法是每找到一个叶子就把当且节点的子树大小减去叶子节点的子树大小，减到大小为 1 时代表所有叶子已经找完。

情况 2

如果还有叶子没有被找到呢？

不妨假设当且右节点所对应的区间是唯一没有被查找的区间。

那么我们可以根据剩余子树和以及该区间内的叶子节点的子树和直接计算出该区间进行 check 后的结果，而无需调用 check。

情况 3

如果有区间不是唯一没有被查找的区间呢？

由于我们知道，父亲节点对应的区间中至少有一个叶子，那么如果左区间没有叶子，那么右区间中至少有一个叶子。

此时我们可以根据右区间的区间和来判断有没有可能有区间都是叶子，如果没有可能都是叶子，而右区间又至少有一个叶子，那么右区间 check 后的值也只有 3 一种了。

通过以上两种情况，我们可以发现，如果查找的点只有一个叶子节点，那么这种优化是完全可以把复杂度控制在 $\log n$ 内的。

如果有两个即以上的叶子，如果你在查询前先将查询的序列随机排列，通过计算，最后一个叶子查询次数严格控制在 $\log n$ 次，前面的点如果失败在左区间上，那么右区间是不用在查询的，也就是 $1.5 \times \log n$，这是一个草率估计的下界。

由于出题人比较菜，还不会使用数学方法证明，所以尝试写程序证明：

当叶子集合大小固定为 1000，

① 当查询的叶子数量为 1 时每个叶子平均期望查询 10.4855 次

② 当查询的叶子数量为 2 时每个叶子平均期望查询 11.6809 次

③ 当查询的叶子数量为 3 时每个叶子平均期望查询 11.7304 次

④ 当查询的叶子数量为 4 时每个叶子平均期望查询 11.5878 次

⑤ 当查询的叶子数量为 5 时每个叶子平均期望查询 11.4053 次

此后叶子数量越多时，查询的平均值越少。

不过这个证明并不严谨，如果有严谨的数学推导，请教教出题人。

然而，当你查询的节点越多时，叶子集合收缩的越多，那么 $\log n$ 也会从 10 开始渐渐变小，如果你查询的次数多，那么初始的叶子集合的大小就不会很大。因此期望值大概是 $n \log n$ 次查询并且有一个小于 1 的常数。

从实际效果来看，改进后的算法可以在最坏情况下只查询了 8945 次。

算法 3

考虑继续优化算法 2 的找叶子过程。

事实上，如果每次不是一起把所有都找出来，而是一个一个找，那么效率显然是严格 $\log n$ 的。

所以，该算法询问次数的下界为：

$$\sum_{i=1}^{n-1} \lceil \log_2 i \rceil = 8967$$

所以，使用该算法可以确定在 9100 次询问内询问出解。

然而一般情况下叶子集合不是满的，询问次数可能略小于 $\lceil \log_2 i \rceil$，所以正常在 8000 次询问内就能询问出解。

算法 4

我们可以先找出叶子节点，叶子节点既可以用询问 B 直接确定，也可以用询问 A 确定。

如果除了根以外其他节点都是叶子的话，那么树的形态自然也确定了。

否则随机选择一个非叶节点 x，通过二分找到 x 的所有节点。然后把 x 复制一份后和父亲断开，形成两棵新的树。依次类推的递归下去。

乍一看这是非常暴力的做法，尤其是找子树中所有节点这一部分，但是理性分析一似乎不那么坏。

不妨假设总节点个数为 n，x 中有 a 个节点，令 $t = \min(a, n-a)$，线段树查找次数充其量不过 $t \log t$。

不妨令 $T(n)$ 为一棵 n 个节点的树的询问 A 使用次数。

那么 $T(n) = \max_{i=1}^{\frac{n}{2}} (T(i) + T(n-i) + i \log n)$。

这个式子的复杂度分析与启发式分裂一致，所以最坏情况为 $n\log^2 n$ 。经计算，当 $n = 1000$ 时，代入上述式子后 $T(n) = 28956$ 。

但是，不难发现这两个 log 都是不满的，而且该算法剪枝空间非常大。经实际测试，最坏情况为 1.4×10^4 多一点点。