Problem B. Where Is the Root?

Author: Ray Bai
Developer: Roman Bilyi
Editorialist: Anton Trygub

Subtask 1. You can just ask about each possible subset of nodes. As we will show later, for each two root candidates there is a query for which their answers would be different, so we can determine the root uniquely. It's enough to ask $2^n - 1$ queries.

Subtask 2. Let's ask a query for each pair of nodes. Clearly, if root is v, then the answer to each query (r, v) for $v \neq r$ is `YES`. If there is only one node that gives `YES` for each query, it must be the root. Suppose that some node $r_1 \neq r$ also gives `YES` for each query in which it's included. If r is not a lead, then there is a node $v \neq r_1, v \neq r$ such that $LCA(r_1, v) = r$, contradiction. So, r must be a leaf.

Suppose that there is another leaf r_1 that gives `YES` for all queries. Then consider any leaf v different from r, r_1 (in a graph in which at least one degree is at least 3, there are at least 3 leaves). Then, $LCA(r_1, v)$ can't be equal to r_1 and v, so the answer would be `NO`. Contradiction.

So, if there is only one such node, it's the root, otherwise it's the unique leaf among the nodes which gave `YES` for each query in which they were involved. It's enough to ask $\frac{n(n-1)}{2}$ queries.

Subtask 3. There are many different approaches that achieve various bounds. We will describe the solutions which work for $n = 500$ in 10 queries, and in 9 queries.

10 queries: Suppose that r is the root. First, let's ask a query about all leaves. Their LCA has to be r (if r is a leaf, it's clear; otherwise, there is a leaf in each of the subtrees of r). So, we can determine if r is a leaf with this query.

Now if r is **not a leaf**, let's ask queries for sets, which contain all leaves. We will get `YES` iff we have included r into the set. So, we can do binary search on the non-leaf nodes, discarding roughly half at each time. This way, we will need at most 9 queries.

Suppose now that r **is a leaf**. If we ask a question about some subset of the leaves of size ≥ 2, the answer will be `YES` if and only if r is in the chosen set. So, we can once again do binary search on the leaves, until we get at most 2 candidates r_1, r_2. At that point, we can ask a query for nodes r_2, r_3, where r_3 is any other leaf, to determine if r_2 is the root. This way, we will also need at most 9 queries.

Adding the initial query about all leaves, we are done in 10 queries.

9 queries: Our algorithm loses one query at the first step: it might be the case that the number of remaining candidates is $n-$ some small number (when the root is a leaf, and almost all nodes are leaves, for example). Let's try to solve this issue.

Let's arrange our nodes in the following fashion: first all leaves, then all non-leaves. What happens if we ask a query about the some prefix of this order of length at least 2? Let's consider two cases.

- Our prefix contains all the leaves, and, maybe, some other nodes. Then their LCA is r, and we will get `YES` if and only if r is in that prefix.
- Our prefix is some subset of the leaves. Then, if r is among them, we will get `YES`, otherwise their LCA would be some node different from any of them, so we will get `NO`. Once again, we will get `YES` if and only if r is in that prefix.

So, we can do binary search on this order of nodes! The only exception is when we end up with a prefix of length 2. Then, as in previous subtask, we should ask one of these leaves with one of other leaves.

This solves the problem in 9 queries.