NOI25 Task Solution
Xoracle (SWE)

Problem Authors:
Raunak Redkar, redkar@kth.se, KTH Royal Institute of Technology
Harry Zhang, harry.zhang@kodsport.se, KTH Royal Institute of Technology

Solutions

Subtask 1

Fix a node that we call a.

Query a with all other $N-1$ nodes.

Now the results we will get back will numbers within exactly one of these sets:

- $[0, 2, 3]$
- $[0, 1, 3]$
- $[0, 1, 2]$

If it is the first set, then we know $\deg(a) = 1$. Or if it is in the second set, then $deg(a) = 1$, and so on. Now we know what degree a is, and can use the information from the queries to figure out what degrees all other nodes has.

Subtask 2

Similar to subtask 1.

- $[0, 2, 3, 5]$
- $[0, 1, 3, 6]$
- $[0, 1, 2, 7]$
- $[0, 5, 6, 7]$

By knowing atleast 3 elements in one of these sets, $deg(a)$ will be uniquely defined.

Subtask 3

For every pair of nodes, query them. If the query result i 0, then the two nodes must have the same degree.

We divide the nodes into groups, where the nodes with the same degree are in the same group.

Since the graph is a tree, there must exist a group that represents the nodes of degree 1. We assume for each group that the nodes have degree 1, then we can calculate the degrees for every other node using the queries we used. If the sum of degrees of all nodes add upp to $2(N-1)$, then we have found a valid solution. A proof this is mentioned in the full solution.

Subtask 4

It can be realised that less queries can be used.

We query in a BFS-fashion. When we find that two nodes x, y have the same degree, it is easy to see that the result of querying x with any other node compared to querying y with any other node will be the same.

There are at most $\sqrt{N}$ different degrees that could show up, which means that using this optimization, we only have to spend $\approx N\sqrt{N} \approx 32000$ queries.

Subtask 5 (Solution Idea by Joshua Andersson)

$N \leq 10^3$.

Query any spanning tree of the graph, for example query 1 with 2, and then 2 with 3, and so on. This uses exactly $N - 1$ queries.

Since a degree in this graph is bounded by $N \leq 10^3$, it has at most 10 bits.

Now fix a node and call it x. For each bit j from 1 to 10, assume bit j of $deg(a)$ either on or not. Now for each combination of bit-configuration, we can calculate the rest of the degrees for all other nodes, using the XOR-values from the queries. We do this until we find a configuration where the sum of degrees is $2(N - 1)$ (proof mentioned in full solution).

This results in a time complexity of $O(N2^{\log N}) = O(N^2)$.

Equvialently, we can just iterate $deg(a) = 1, \ldots, N$, and calculate degrees for all other nodes.

Small Comment

We will eventually realize that only $N - 1$ queries is actually needed to actually get information of the degree XOR of any pair of nodes, because of the property of XOR. We can calculate the answers to the rest of the queries offline without using more than $N - 1$ queries.

But to able to do this efficiently, then either we need to query in some nice way and use a data structure on top of it, or we query the first $N - 1$ queries in a even nicer way as seen in the full solution.

Additionally, we had originally thought of subtasks $q = 3000, N \leq 10^3$, and $q = 1100, N \leq 10^3$ that suits the problem solver if they did not gain this insight of the property of XOR. However, we forgor what the specific solution for these subtasks were.

Full Solution (Solution Idea by Harry Zhang)

To make the solution run in linear time or $O(N \log N)$, we do the following:

The solution begins by querying node 1 with every other node in the tree. This allows grouping nodes based on their XOR values, since nodes with the same degree will have the same XOR result. If two nodes have an XOR result of 0, they must be the same node. So now we have divided all N nodes into m groups, where all nodes in each group has the same degree, and no two different groups has the same degree. Also store all XOR values from the queries, these are needed later to construct all nodes' real degrees.

Since the structure is a tree, there is always at least one node with degree 1. Moreover, the number of nodes with degree larger than 2 is always less than the number of nodes with degree 1. A handwavey proof is: Start with any node with degree $k \geq 3$, then it has exactly k nodes with degree 1. For every node with degree $k \geq 3$ we add to this tree, we we will also need to add $k-1$ nodes with degree 1, because it is a tree.

This means that among the two largest groups (in terms of the number of nodes in the group), at least one must consist of nodes with degree 1. (If the largest group is not degree 1, then it has to be degree 2).

To determine which group corresponds to degree 1, the solution first assumes that the largest group consists of nodes with degree 1 and checks whether the sum of all degrees equals $2(N-1)$. If this holds, the assumption is correct. If not, the second-largest group is assumed to be degree 1, and the sum is checked again. The correct group is the one that makes the sum equal to $2(N-1)$, and once it is identified, the degrees of all nodes can be determined. (Because the group with degree 1 is the largest or the next largest, if we assume a smaller group is degree 1, then the sum of degrees will not add up to $2(N-1)$).

Finally, if both groups result in the correct sum, it means that the second group consists of nodes with degree 2. In that case, it does not matter which group is assigned as degree 1, as the final answer will remain the same. For example, a line graph with $n = 4$ results in the degree sequence $1, 2, 2, 1$, which is ambiguous with $2, 1, 1, 2$. Another example is a graph where one node has degree 7, another has degree 4, and the remaining nodes are arranged such that the number of nodes with degree 2 matches the number of nodes with degree 1. This happens because the only possible ambiguities arise when distinguishing between degree 1 and degree 2 nodes, and switching between these cases is equivalent to XOR-ing with 3. Therefore, ambiguities occur only when every node in the graph has their own corresponding node such that their degree XOR equals 3.

This method efficiently determines the degrees of all nodes using $N-1$ queries and runs in $O(N)$ time.

Alternative Solution (Solution Idea by Theodor Beskow)

Query in the same way as above.

We use the fact that there are a lot of nodes of degree 1 and degree 2.

Now we randomly pick a node, and assume that node has degree 1. Using this assumption, all other degrees will be uniquely determined.

Do this until the sum of degrees add up to $2(N-1)$.

However, we can see that this algorithm would not be sufficient in for example a line graph with $N = 10^5$ nodes, as the probability of picking the correct degree 1 node is relatively small.

To solve that issue, we use the fact that there are a lot of nodes of degree 1 **and** degree 2. In fact, the majority of the nodes in a tree has degree 1 or degree 2. (Proof attached further down.)

Now we can randomly pick a node, assume first that it has degree 1. If that doesn't work, assume it has degree 2. Repeat.

Since majority of nodes has degree 1 or 2, the expected number of times we would have to randomly pick nodes is less than 2, resulting in a running time of $O(N)$.

Formal proof of that majority of nodes in a tree has degree 1 or degree 2

Proof. Let A be the number of nodes with degree 1, B = number of nodes with degree 2, C = number of nodes with degree 3 or higher, which implies that $N = A + B + C$.

The sum of degrees is $2(N-1)$, but it can also be written as $A+2B+\sum_{deg(v)\geq 3} \deg(v) = 2(N-1)$.

We substitute $N = A + B + C$, resulting in

$$A + 2B + \sum_{deg(v)\geq 3} \deg(v) = 2(A + B + C - 1)$$

$$\sum_{deg(v)\geq 3} \deg(v) = A + 2C - 2.$$

We can lower bound $\sum_{deg(v)\geq 3} \deg(v) \geq 3C$, which gives us

$$3C \leq A + 2C - 2$$

$$C \leq A - 2$$

$$2 + C \leq A$$

and since B is non-negative, we get

$$C < 2 + C \leq A \leq A + B,$$

resulting in $C < A + B$, which implies that there are more nodes of degree 1 and 2 combined than nodes that has degree larger than 2.

□