Spring cleaning Solution

Topics

graph, tree, DFS, ordering, LCA, dynamic, smaller-to-larger

Problem Author

Máté Busa

Cleanable tree

It is obvious that a tree is not cleanable if it has an odd number of leaves.
Also, every tree with an even number of leaves is cleanable.

Subtask 2

$Q = 1$, there is an edge between node 1 and i for every i $(2 \leq i \leq N)$
Flóra can't add extra leaf to node 1

If we add 2 extra leaves to an inner node (not a leaf), it's a good strategy to clean the path between these extra leaves. So if we add an even number of extra leaves to an inner node, we can pair all these extra leaves. Otherwise there is an extra leaf without a pair. This leaf will be added to the tree.
A similar thing happens if we add leaves to an original leaf.
In this subtask we know that every path cleaning will go throw through node 1 (except those that we already paired). So we just have to add the distances from node 1.

This subtask can be solved in $O(N + D_1)$

Subtask 3

$Q = 1$, there is an edge between nodes i and $i + 1$ for all i $(1 \leq i < N)$
Flóra can't add extra leaf to node 1 nor node N

It's a good idea to clean the path between node 1 and node N. After this we should just simply pair the extra leaves in an optimal way.

This subtask can be solved in $O(N + sum(D_i))$

Even-odd nodes

It can be proved that in the optimal solution, all edges are cleaned at most twice.

So our task is to minimize the number of edges cleaned twice.
Let's root the tree in an inner node r (It's possible since $N > 2$).
We denote the parent of a node u by $P(u)$ in this rooted tree.
Let's call a node u even if in it's subtree there are even number of leaves. Call it odd otherwise (all leaves are odd nodes).
It can be proved that we clean the edge from u to $P(u)$ twice iff u is an even node (where $u \neq r$). Let E be the set of even nodes. In this case, the minimum required cost for the original tree is $N + |E| - 2$ if r is an even node. We can compute which nodes are even in

$O(N)$ time with a single dfs.

Subtask 4

$N \leq 20000$, $Q \leq 300$

We can build up every tree variation and calculate the number of even nodes for it.

This subtask can be solved in $O(N \cdot Q + sum(D_i))$

Path to the root

For every node u let's denote its distance from the root r by $D(u)$. Let $S(u)$ denote the number of even nodes on the path from u to r.
This means that the number of odd nodes on the path from u to r is $D(u) + 1 - S(u)$.
The previous values can be computed in $O(N)$ time for the original tree using a single dfs.

Subtask 5

The original tree is a complete binary tree rooted at node 1

In this case $D(u) <= logN$ for every u. When adding a leaf to node u, we change all parities from u to r, which takes at most $O(logN)$ time.

This subtask can be solved in $O(N + sum(D_I) \cdot logN)$.

Subtask 6

$D_i = 1$

We calculate the minimum required cost for the original tree in $O(N)$.

If we add an extra leaf to an original leaf the cleaning cost increases by 1, but nothing else happens.
If we add an extra leaf to an inner node u, then every node on the path from u to r will change parity.
So the answer is $N + |E| - S(u) + (D(u) - S(u))$ if r was and odd node in the original tree.

This subtask can be solved in $O(N + Q)$

Subtask 7

Original constraints.

Virtual tree approach

Adding an extra leaf to node u may cause parity change on the path from u to r. If we add an extra leaf to node v too then the parities from node $LCA(u, v)$ (LCA=lowest common ancestor) to r will change twice, i.e. it doesn't change at all.
If we add more leaves then there will be paths where the parity has changed odd times and where it has changed even times.
If we added a new leaf to a node u let's call it a critical node.
Let's define L as a subset of original nodes where:
If we add a node to node u, then $u \in L$. If $u, v \in L$ then $LCA(u, v) \in L$ too.
It can be proved that $|L| \leq min(N, 2 \cdot D_i - 1)$ for the ith variation. We can form a new tree from the nodes of L in the following way:
In this tree, the parent of node $u \in L$ is node $v \in L$ if v is an ancestor of u in the original tree and $D(v)$ is maximal.
Let's denote the parent of node u in the new tree by $P_L(u)$.
In the new tree, for all nodes u we calculate the number of critical nodes in the subtree rooted in u. (This can be computed in $O(|L|)$ using a single dfs.) If this number is even then nothing happens. If it's odd, the parity from node u to $P_L(u)$ will change in the original tree (but will not change in $P_L(u)$. We can say that the parity has changed from u to r, and then from $P_L(u)$ to r. These can be handled by using $S(u)$ and $S(P_L(u))$ the same way we described in subtask 5.

Note that we don't have to know the $i + 1$th variation before answering the ith one. So this solution can solve this problem "online".

This subtask can be solved in $O((N + sum(D_i))logN)$.

We remark that the problem can also be solved by utilizing the Heavy-Light Decomposition (HLD) of the original tree, this solution was also passing if the implementation was not too messy.

Dynamic offline solution

Instead of computing the value changes for every variation online, we can preread and store for each individual node which variations we add leaves to them. Then, using a dfs in a bottom-up DP manner, starting from the leaves for each node we pair the unpaired leaves in its subtree. We can store the unpaired leaves (only the variation's and parent node's identifiers are interesting for us) in some collections (e.g. set/map) and at each node, we merge the collections of all of its children.

If we encounter two children having an unpaired leaf for the same variation, it means the current node is the LCA for those two additions, and we can compute the change in the cost of cleaning for that variation and store it.

In order to maintain low complexity (i.e. logarithmic in terms of sums of D_i, we must make sure to not copy any collections needlessly, and always insert the elements of the smaller into

the bigger one.

After computing the cost changes for all variations simultaneously by a single dfs, we can contruct the answer for each variation by checking if it has any unpaired leaves in the final collection, and adding the cost change to the basic cleaning cost of the original tree.