EGOI 2024 Editorial - Team Coding

Problem author: Yann Viegas.

The problem

You are a given a rooted tree (rooted at 0) with N nodes. Each node is assigned a programming language (numbered from 1 to K). l_i will denote the programming language assigned to node i. To complete a programming project, you will chose a node u, then you can apply the following operation any number of times:

- pick two nodes v and w having the same depth in the tree
- swap the programming language assigned to nodes v and w

The value of the project is the number of nodes in the subtree of u that share the same programming language as u. When the leader is u, we will call the value of the project $\text{val}(u)$. Also, in the rest of the editorial, $\text{dep}(u)$ will denote the depth of node u and for programming language c, $\text{cnt}(c)$ will denote the number of nodes u such that $l_u = c$.

Find the maximum value of a project that can be achieved and the minimum number of operations needed to achieve it.

We will first discuss how to find the maximum value of a project. The number of operations will then be easy to get.

Test group 1: Chain

In a chain, no two nodes share the same depth so we can not apply any operation. Thus, we fix u as the leader of the project and count the number of $j > i$ such that $l_j = l_i$. This can be implemented in $O(n)$.

Test group 4: $N \leq 2000$

Let us fix u as the leader of the project and compute $\text{val}(u)$. For a given depth d (depths are computed relatively to the root 0, not to u), $\text{atDep}(u, d)$ will denote the number of nodes at depth d in the subtree of u. We will also denote the number of nodes at depth d having programming language c and lying in the subtree of u as $\text{cAtDep}(u, d, c)$. I claim that:

$$\text{val}(u) = \sum_{d=\text{dep}(u)}^{N} \min(\text{atDep}(u, d), \text{cAtDep}(0, d, l_u))$$

.

Note that you can limit the sum to the depths that appear in the subtree of u which gives a way to compute $\text{val}(u)$ in $O(\text{size of the subtree of u})$.

Test group 2: $K \leq 2$

In this subtask there are only 2 different programming languages. Thus, we can fix the color of the leader of the project (let it be c). There are different ways to solve this subtask.

One possible idea is to reduce the number of leaders to consider to fewer candidates as leaders having the same color might share redundant information.

Indeed, if u and v are such that $l_u = l_v$ and u is an ancestor of v, then $\text{val}(u) \geq \text{val}(v)$. Essentially, all the operations you apply when you pick v as a leader will have the same effect when you pick u as a leader. This way, we can pick the node with the smallest depth (let it be u) and process it the same way we solved $N \leq 2000$. After computing $\text{val}(u)$, we do not need to consider nodes lying in the subtree of u anymore. Then, we pick another node u, which is the node with the smallest depth that is not yet marked.... The computations are amortised as all the subtrees we will do computations on are disjoint. This gives an $O(n)$ solution once the order of nodes is fixed (which costs $O(n\log(n))$ to precompute).

Another way to solve that subtask would be to use small to large merging. Our dfs function should return a map (can also be a vector or deque depending on implementation details) containing $\min(\text{atDep}(u, d), \text{cAtDep}(0, d, l_u))$ for all d that are actual depths in the subtree of u (otherwise the size of the map would explode). The function should also return $\text{val}(u)$. The details are left as an exercise to the reader (but the code is quite short). Note that this can be implemented in $O(n)$ as height merging results in linear time.

Test group 3: each programming language is assigned to at most 10 nodes

The constraints lead us to fix the color c of the project. Now, there are at most 10 leaders candidate because of the constraints, so we can fix the leader u. To compute $\text{val}(u)$, we can notice that there are at most 10 non zero terms in the sum. Indeed, $\text{cAtDep}(u, d, c) \neq 0$ for every d such that there is at least one node of color c at that depth (and there are at most 10 such nodes). The only thing left is to be able to compute each term of the sum efficiently. To compute $\text{atDep}(u, d)$: for each depth d, sort all nodes by their euler tour "in" time and use binary search to count nodes that lie in the subtree of u. Thus, we can query $\text{atDep}(u, d)$ in $O(\log(n))$. Finally, we can compute $\text{cAtDep}(u, d, c) \neq 0$ naively by iterating through all u such that $l_u = c$.

We can thus process color c in $O(\text{cnt}(c)^2\log(n))$. Note that it is also possible to do it in $O(\sum_{c=0}^{K} \text{cnt}(c)^2)$ by processing all the colors at the same time by doing a height merging, similar to what was discussed for $K \leq 2$.

Test group 5: full solution

Using what was previously discussed, we can process a color in $O(\min(N, \text{cnt}(c)^2)$. The total number of operations is going to be $O(N\sqrt{N})$. One way to see that is to notice that for all c such that $\text{cnt}(c) < \sqrt{N}$, we process them in $O(\text{cnt}(c)^2)$. Thus, we use $\sum_{\text{c}|\text{cnt}(c)<\sqrt{N}} \text{cnt}(c)^2$ operations to process them. However,

$$\sum_{\text{c}|\text{cnt}(c)<\sqrt{N}} \text{cnt}(c)^2 \leq \sum_{\text{c}|\text{cnt}(c)<\sqrt{N}} \text{cnt}(c)\sqrt{N} \leq N\sqrt{N}$$

as $\text{cnt}(c) < \sqrt{N}$ and $\sum_{\text{c}|\text{cnt}(c)<\sqrt{N}} \text{cnt}(c) \leq \sum_{c=0}^{K} \text{cnt}(c) = N$.

For c such that $\text{cnt}(c) \geq \sqrt{N}$, we run the $O(n)$ algorithm and there are at most $\sqrt{N}$ such c as $(\#\text{ of such c}) \cdot \sqrt{N} \leq \sum_{\text{c}|\text{cnt}(c)\geq\sqrt{N}} \text{cnt}(c) \leq N$

Note that $O(N\sqrt{N}\text{log}N)$ solutions also pass (in case we use the log algorithms from the previous setions) and such solutions are also easier to implement.

Finding number of operations

Assuming the leader is u, the number of operations performed at depth d is

$$\min(\text{atDep}(u, d), \text{cAtDep}(0, d, l_u)) - \text{cAtDep}(c, d, l_u)$$

. We can compute this the same way as we computed $\text{val}(u)$.