LOTFIZADEH OLYMPIAD

Problem Analysis

Christmas Tree

Rahim Mammadli

December 2021

1 Subtask 1

For this subtask it is enough to go through each subset S, and each candidate socket node c, and calculate the best possible (S, c) pair. Bitmasks can be used to iterate over all possible subsets. Expected complexity : $O(2^n * n^2)$

2 Subtask 2

For a given c and S , the given formula is equal to $|S|*n-\sum_{x\in S} dist(x,c)-max_{x\in S}(h_x)$, where $dist$ denotes shortest distance between the given 2 nodes in the tree. Given this formula it is clear that, if we denote node y as the node with maximum h (heat) value among the ones we have chosen, e.g. $max_{x\in S}(h_x) = h_y$ and $y \in S$, then it is optimal to also include all the nodes z whose heat value is less than or equal to the one of y's in our chosen subset S. Indeed each added node will add n to our expression, and subtract at most $n-1$ from our expression. So one can sweep over nodes' heat values, and pick all the nodes whose heat value is less than or equal to current value. We also have to find best possible socket node c to choose along with given S. In this subtask, this can be done by first, just doing bfs from each newly added node as we sweep over the heat values, and then going through all nodes and choosing the one with minimum sum of distances found by bfs. Expected complexity : $O(n^2)$

3 Subtask 3

From this subtask onwards, the general idea will always be to sweep over heat values, and choosing all nodes less than current heat value. Only thing that will change is the way we find socket node c. For this subtask the given tree represents "a path graph"(`https://en.wikipedia.org/wiki/Path_graph`). Let us pick one of the two degree 1 nodes as the root, and renumber the nodes as their distance from the root. Let m_i denote the new number of node i. In this case, given a chosen subset of nodes S the best choice for socket node c is the node corresponding to median of the set that contains all m_x, such that $x \in S$. This node and sum of distances from this particular node, can be kept track of with a simple set in $C++$. This is very similar to finding running median. One would need to keep track of two variables to keep track of sum from given socket node : $var1 = \sum_{y \in S, m_y < m_x} m_y$, and $var2 = \sum_{y \in S, m_y > m_x} m_y$. Expected complexity : $O(n * log(n))$

4 Subtask 4

Let us again sweep for heat value. Let us call a node "activated", if it has heat value less than or equal to current heat value that is being considered (e.g. it belongs to current subset S). It can be proven that as we sweep for heat values, if the optimal socket node in previous iteration was c_1, newly added node (as we increased the heat value exactly one new node was added) is cur , the new optimal socket node c_2 is somewhere on the simple path from c_1 to cur in the given tree. Let us assume $cur! = c1$. Then let x denote the first node on the path from c_1 to cur. Let us cut the tree into two parts along the edge (c_1, x). Now it is clear that as long as the part of the tree not containing c_1 has more activated nodes, compared to the part of the tree containing c_1, we should move the socket node towards cur. Since the distance between any two nodes in this tree is $O(log(n))$, we can move the socket node one by one. And to check number of activated nodes in a particular subtree we can just use the binary structure given to us. Whenever a node is activated add $+1$ to each one of its parents. Note that knowing the count of activated nodes in subtrees is enough to figure out the number of the nodes in either side of the tree whenever we consider a cut, where cut is defined as above. Expected complexity : $O(n * log(n))$

5 Subtask 5

Let us again sweep for heat value. Now given, c_1 and *cur* as defined above, instead of moving one by one for socket node, we will binary search for it implicitly. Let us create the binary jump table for the given tree, e.g. create a two dimensional array, where $lca[i][j]$ denotes the 2^jth parent of node i. First of, if the current number of nodes is even, it can be proven that the previous c_1 is still one of the optimal nodes to choose as a socket. Now if the number of nodes is odd, then we know the new c_2 is somewhere on path between c_1 and *cur*. Let x again denote the first node on the path from c_1 to *cur*. Let us make a cut along the edge $(c1, x)$. Here let us denote number of activated nodes in the part of the cut containing c_1 as k. if c_1 is non optimal , then the number of activated nodes in the part of the cut not containing c_1 is exactly 1 more than the number of activated nodes in the part of the cut containing c_1, e.g. it is $k + 1$. This can be simply proved by contradiction, e.g. if it would'nt be $k + 1$ that means c_1 was not optimal socket node for previous iteration. This information will ease our algorithm. . Given c_1 and *cur*, and the fact that number of activated nodes currently is odd, and c_1 is currently non-optimal, the new c_2 is one of ;

a) The ancestor of c_1 with maximum depth who has more than k activated nodes in its subtree.

b) The ancestor of *cur* with maximum depth who has more than k activated nodes in its subtree.

Compute both and take the one with maximum depth. One would need dfs timestamps, to calculate "tin", "tout"s, and use fenwick tree/segment tree on top of it. As mentioned before binary search can be done implicitly using the *lca* binary jump table. Expected complexity : $O(n * log^2(n))$