The Xana coup (xanadu)

BY LUKAS MICHEL (GERMANY)

Abridged problem statement. You are given a tree with N vertices, where every node is initially either turned on or off. Every vertex has a button. Now you can execute the following operation several times: You press the button at vertex v which toggles vertex v and all of its direct neighbors. Toggling a vertex means turning it on if it's currently turned off and vice versa. Now you are to find the minimal number of button presses needed to turn off all vertices.

There are a few important insights in this problem.

- For every vertex, it's only important whether we toggled this vertex an even or an odd number off times. It's easy to see this: After toggling a vertex once, it changes its state, and if we toggle it a second time, it returns to its initial state.
- The order of the button presses doesn't matter.
- It doesn't make sense to press a button more than once. This would toggle the vertex and its neighbors an even number of times, causing no change.
- So we will press every button either 0 or 1 times. This means the solution will either be between 0 and N or it will be impossible to turn off all vertices.

Subtask 1. $N \le 20$

After observing that we have to press every button either 0 or 1 times, the first subtask is really straightforward. Just try all possible subsets of buttons to press. For every subset check if pressing those buttons turns off all nodes and then choose the subset with the minimal number of buttons that turns off all nodes. This yields an easy $O(N \cdot 2^N)$ solution that should easily fit into the time limit.

Subtask 2. $N \le 40$

There are several solutions for the second subtask. One possible solution is to first find a centroid. We root the tree at this centroid. Then, every subtree will have at most $N/2$ vertices. Let's first assume we don't press the centroid's button. We now run the brute force from subtask 1 on every subtree. This way, we determine the minimal number of button presses if we press the button of the root of this subtree and the minimal number of button presses if we don't press the button of the root of this subtree. Now, we have to combine the solutions of the subtrees. Obviously, if the centroid is turned on, we have to press an odd number of children's buttons, and an even number otherwise.

To combine the solutions efficiently, we can iterate over all children. We have two variables *even* and *odd* that represent the minimal number of button presses if we want to press an even or odd number of children's buttons. *even* is initially set to 0 and *odd* to ∞. Now let's assume we are currently processing a child. Let p_1 be the minimal number of button presses to turn off the subtree of the child if we want to press the child's button, and p_2 the same value if we don't want to press the child's button. Then we set $even = min(odd + p_1, even + p_2)$ and $odd = min(even + p_1, odd + p_2)$. The logic behind this: If we want to press an even number, we can either press an odd number before and then press the current child's button, or we press an even number before and we don't press the current child's button.

Of course, after having processed all children, our result is in *even* if the centroid was turned off and in *odd* if the centroid was turned on. Now, we press the button of the centroid and repeat the same procedure again, adding 1 at the end. This solution runs in $O(N \cdot 2^{N/2})$.

Another, perhaps more approachable solution starts by dividing the tree into its bipartite parts, i.e. dividing the set of vertices into two sets such that every edge has a vertex in each of them. Then, the smaller of these parts has at most $N/2$ vertices; let's call the set of vertices in this part V_1 and the other one V_2. Note that whether a vertex in V_2 is turned on only depends on this vertex's button and on buttons of vertices in V_1.

We can now brute force all possible button presses for V_1. With these fixed, the state of any vertex in V_2 only depends on its own button; this means that we know whether we have to press it so that the vertex is off in the end. Now that all buttons in V_2 are fixed, we can check whether the vertices in V_1 are all off. If they are, we have a possible solution. We then just take the minimal number of button presses over all solutions. This algorithm also runs in $O(N \cdot 2^{N/2})$.

Subtask 3. The graph is a line.

Vertex 1 will only have one neighbor, vertex 2. Imagine we press the button at vertex 1. Then, we already know whether we have to press the button at vertex 2 or not: If vertex 1 was previously turned off, we will have to press button 2, and if vertex 1 was previously turned on, we should not press button 2. The same holds for all other buttons.

So, deciding whether we press the first button already determines what buttons we press. This yields an $O(N)$ solution for this subtask: Try both options (pressing or not pressing the first button) and choose the option that turns off all vertices in the minimal number of button presses.

Subtask 4. The graph is a binary tree.

If every vertex is directly connected to at most 3 other vertices, we can choose a root r such that every vertex has at most 2 direct children.

We can observe that any vertex v only depends on its parent and its ≤ 2 children. Now we can do dynamic programming on the tree. For every vertex v we compute 4 values:

- $dp[0][0][v]$: Assuming we didn't press the button of v's parent and we don't want to press button v, what is the minimal number of button presses to turn off all nodes in v's subtree? This should be set to ∞ if it's impossible to turn off all vertices under these constraints.
- $dp[0][1][v]$: Now we assume that we didn't press the button of v's parent, but we want to press button v.
- $dp[1][0][v]$: Now we did press the button of v's parent and we don't want to press button v.
- $dp[1][1][v]$: Now we assume that we press both the button of v's parent and button v.

Now let's assume we have calculated dp for all children and now we want to calculate $dp[i][j][v]$. First we check whether after pressing the button of v's parent i times and button v j times (after toggling v $i+j$ times) vertex v is turned on or off. If v is already turned off, we have to press either both or none of the children's buttons. Otherwise we should press exactly one of the children's buttons. So, we simply use the dp-values that we calculated for the children. The overall solution to the problem then is $min(dp[0][0][r], dp[0][1][r])$ because we can either push button r or not push button r. This solution runs in $O(N)$.

Subtask 5. No further constraints.

For the full solution we can use the same dynamic programming that we used for subtask 4. We just have to change how we calculate $dp[i][j][v]$. Assume that v is turned off after toggling it $i + j$ times. Then, we have to press an even number of children's buttons. Otherwise, we should press an odd number of children's buttons.

There are multiple ways to calculate that efficiently, here is one: We use an idea similar to the one we used for solving subtask 2. For every node, we temporarily calculate 4 values:

- $tmp[0][0]$: This should be set to the minimal number of button presses to turn off all vertices in the children's subtrees if v is not pressed and we press an even number of children's buttons.
- $tmp[0][1]$: Same as before, but we press an odd number of children's buttons.
- $tmp[1][0]$: Now we want to press v and an even number of children's buttons.
- $tmp[1][1]$: Here we want to press v and an odd number of children's buttons.

How can we calculate those? We do almost the same as in subtask 2: We initialize the odd tmp-values with ∞ and the even ones with 0. Now let's iterate over all children and calculate their dp. Assume we are currently processing child c. Then the we change tmp in the following way:

- $tmp[0][0] = min(tmp[1][0] + dp[0][1][c], tmp[0][0] + dp[0][0][c])$
  We don't press v, so we use $dp[0]$ in both cases. We want to press an even number of children's buttons, so we can either press an odd number before ($tmp[1][0]$) and press c or we press an even number ($tmp[0][0]$) and don't press c.
- $tmp[0][1] = min(tmp[1][1] + dp[1][1][c], tmp[0][1] + dp[1][0][c])$
- $tmp[1][0] = min(tmp[0][0] + dp[0][1][c], tmp[1][0] + dp[0][0][c])$
- $tmp[1][1] = min(tmp[0][1] + dp[1][1][c], tmp[1][1] + dp[1][0][c])$

Now if vertex v is turned on after being toggled $i + j$ times, then $dp[i][j][v] = j + tmp[1][j]$, because we need an odd number of children's buttons pressed. Else, $dp[i][j][v] = j + tmp[0][j]$. This solution runs in $O(N)$.