Code Jam 2008 - Round 2

Analysis: Cheating a Boolean Tree

This is an easy exercise in dynamic programming.

Let us define for each node v in the tree, F(v, x) to be the smallest number of gates we need to flip in order to make the output of v be x (0 or 1). The value F(v, x) can be computed using dynamic programming as follows.
If v is a leaf with input value 0, then F(v, 0) = 0 -- no gate needs to be changed; and F(v, 1) can be assigned to -1 or some really big value to indicate "mission impossible".
If v has two children, u and w, and the gate at v is OR, then F(v, 0) can be computed by taking the better of the following options.

1. Do not change the gate and use the plan for F(u, 0) and F(w, 0);
2. Change the gate to AND and use the plan for F(u, 0);
3. Change the gate to AND and use the plan for F(w, 0).

So

$$F(v, 0) = \min\{ F(u, 0) + F(w, 0),\ 1 + F(u, 0),\ 1 + F(w, 0) \}.$$

The other cases are similar. One can compute the F values iteratively bottom-up or recursively top-down.

In addition to the solution above, we introduce some interesting observations. Look closely at the formula above. Suppose we want the output 0 at the top, when you start a top-down computation, you will only use values F(_, 0) and never use any F(_, 1). Similarly, if the desired output at the top is 1, you will never need to care about the values for F(_, 0). Furthermore, in computing F(v, 1), you never want to change an OR gate to an AND gate. The deep reason for these is that, when there is no negation gate involved, the circuit computes a monotone function, and you will never want to change an output from 1 to 0 in the middle.
By de Morgan's law, if we interchange all the input values, change all the gates to the opposite type, and change the desired output from 0 to 1 (or 1 to 0), we obtain a dual problem, and the minimum number of gates one needs to change remains the same. So we can assume that the desired value is 1 (or 0, depends on your taste), and forget about implementing half of the cases.