## PREREQUISITES:

DFS, BFS and intuition for how it operates on a tree, finding lowest common ancestor

## PROBLEM:

Given a tree, let $(u_1, u_2, \ldots, u_N)$ be a bfs ordering of it with $u_1 = 1$.
Find the minimum possible value of $\sum_{i=1}^{N-1} dist(u_i, u_{i+1})$ across all possible bfs orderings.

## EXPLANATION:

Let's first define something that will be useful later: let $h_u$ denote the height of vertex $u$, i.e, the height of the subtree of which $u$ is the root. This can also be thought of as the length of the longest path from $u$ to a leaf in its subtree.
$h_u = 1$ if $u$ is a leaf.

Now, look at the problem a bit differently. Instead of thinking about distances, the final answer can also be thought of as the number of times we cross each edge, summed up across all the edges.

So, consider a vertex $u$ and its parent $p$. How many times do we cross this edge at least?
Well, we move down from $p \to u$ sometimes and up from $u \to p$ sometimes, so let's consider them separately.

▾ Moving down

Moving down means we enter $u$ from outside.

An arbitrary bfs process will visit $u$, then visit vertices at the same depth as $u$, then visit the children of $u$, then other vertices at the same depth as these children, and so on.

You can see that we will thus enter $u$ exactly $h_u$ times: once for every depth contained in the subtree of $u$.

▾ Moving up

The analysis here is the exact same as the moving down case: we move up across this edge exactly once for each depth contained in the subtree of $u$, making $h_u$ moves in total.

This gives us an immediate upper bound on the answer: $2 \sum_{i=2}^{N} h_i$.

However, this isn't the final answer by itself: there is one more thing to consider, which is the fact the we need not make all the moves described above.

In particular, depending on the bfs order chosen, there are some edges that we won't move up across, or won't move down across $h_i$ times — for example, we won't move up across the last edge we visit.
But which edges can satisfy this property?

Once again consider a vertex $u$ and its parent $p$. Suppose $u$ is at a depth of $k$. When can we ensure that we never move up along this edge?
The answer is quite simple: if there exists a vertex at depth $k + 1$ that is not a child of $u$, then we must move up along this edge to visit it.

On the other hand, if every vertex at depth $k+1$ is indeed a child of $u$, then there always exists a way to not have to move up from $u$ to $p$: simply choose the bfs order so that $u$ is the last depth-$k$ vertex visited.
Notice that this also ensures we only go down $p \to u$ exactly once.

Of course, this isn't the only case where we don't perform some moves, but nevertheless it is enough to form an intuition for the process.

This should immediately give a somewhat natural greedy solution: the deeper the subtree of a vertex is, the later we should visit it.

That is, suppose we compute the height $h_i$ of each vertex, as described above.
Then, start a bfs from $1$, but push children of $1$ into the queue in increasing of $h_i$ values.

This process gives us a bfs order $(u_1, u_2, \ldots, u_N)$, after which the final answer is $\sum_{i=1}^{N-1} d(u_i, u_{i+1})$.

Computing pairwise distances is a standard technique, and can be done in $\mathcal{O}(1)$ or $\mathcal{O}(\log N)$ by finding LCA for example.

## TIME COMPLEXITY

$\mathcal{O}(N \log N)$ per test case.