Documentation

The problem of counting nodes in a complete binary tree can be optimized by leveraging the unique properties of such trees. A full binary tree is defined as one in which every level, except possibly the last, is filled. Additionally, all nodes in the last level are positioned as far left as possible. The challenge is to devise an algorithm that can count the nodes more efficiently than a straightforward traversal of all nodes, with a time complexity of (O(n)).

The key insight in solving this problem is that because a complete binary tree is "almost" a perfect binary tree, we can use the height of the tree to reduce the number of nodes we need to examine. The tree's height can be computed by traversing the leftmost path from the root to the deepest leaf node. For a perfect binary tree, the number of nodes is determined by its height, and this characteristic allows us to avoid directly counting each node.

The strategy involves recursively comparing the heights of the left and right subtrees. If the height of the left subtree is equal to that of the right subtree, this indicates that the left subtree forms a perfect binary tree. In this case, the number of nodes in the left subtree can be computed directly using the formula ($2^{height} - 1$), and the algorithm can then proceed to count the nodes in the right subtree.

If the heights of the left and right subtrees differ, this means the right subtree is perfect but one level shorter than the left subtree. In this case, the number of nodes in the right subtree can be computed, and the algorithm proceeds to count the nodes in the left subtree.

By using this recursive approach and leveraging the properties of complete binary trees, we can avoid traversing every node in the tree. Instead, we can skip large parts of the tree when subtrees are identified as perfect. This approach results in a time complexity of ($O(\log^2 n)$), where (n) is the total number of nodes in the tree. The logarithmic factor comes from the need to calculate the height of the tree and from the recursive steps. This significantly improves the efficiency compared to a brute-force (O(n)) solution, making it suitable for large trees with up to ($5 * 10^4$) nodes, as described in the problem's constraints.

This solution efficiently handles the characteristics of a complete binary tree, allowing for node counting in a more optimized and scalable way.