Documentation for the Problem: "Sum of Left Leaves"

1. Problem Statement

The task requires calculating the sum of all left leaves in a binary tree. A left leaf is defined as a node that is a leaf (i.e., it has no children) and is specifically the left child of another node. The input is the root of a binary tree, and the output is the sum of the values of all such left leaves. For example, in a binary tree with the root node 3, having a left child of 9 and a right subtree with a left leaf of 15, the sum of left leaves would be 24. If there are no left leaves, the result should be 0.

2. Intuition

The key observation to solve this problem is recognizing that we need to traverse the tree and accumulate the values of all left leaves. A left leaf can be identified when a node's left child exists and has no children (i.e., it is a leaf). By recursively traversing the tree, we can sum the values of all such left leaves. This problem is best solved using a depth-first search (DFS) approach, where we visit each node and check if its left child is a leaf. If so, we add its value to the sum.

3. Key Observations

To determine if a node's left child is a leaf, we check two conditions: first, that the left child exists, and second, that it has no left or right children. This condition ensures that the left child is indeed a leaf. We need to perform this check for every node as we traverse the tree. Additionally, we recursively calculate the sum for both the left and right subtrees, ensuring we explore all potential left leaves in the tree.

4. Approach

The approach involves using a depth-first search (DFS) to traverse the tree. Starting from the root, we recursively check if the left child of each node is a leaf. If it is, we add its value to the running sum. We also need to continue the traversal for both the left and right subtrees to ensure that we don't miss any left leaves deeper in the tree. The base case for the recursion is when the node is None, indicating the end of a branch. After the traversal is complete, we return the accumulated sum of all left leaves.

5. Edge Cases

There are several edge cases to consider:

- An empty tree (root = None) should return a sum of 0, as there are no nodes to process.
- A tree with only one node should also return 0 since no leaves are left.
- If the tree has no left leaves (e.g., all leaf nodes are right children), the result should be 0.
- If all leaves in the tree are left leaves, then the sum should be the sum of all left leaves.

6. Complexity Analysis

The time complexity of the solution is O(n), where n is the number of nodes in the tree. This is because we visit each node exactly once during the DFS traversal. The space complexity is O(h), where h is the height of the tree, due to the recursive call stack. In the worst case, where the tree is skewed, the height could be equal to the number of nodes, so the space complexity would be O(n).

7. Alternative Approaches

An alternative approach to solving this problem could involve using breadth-first search (BFS) with a queue. This approach would also visit each node level by level, checking if a node's left child is a leaf. While BFS can solve the problem, DFS is often more intuitive and has lower memory overhead, making it a more efficient choice for this particular problem. An iterative DFS approach using an explicit stack instead of recursion could also work, preventing potential issues with recursion depth in skewed trees.

8. Code Implementation

The solution relies on the recursive DFS method, where we check each node's left child to determine if it is a leaf and sum its value. The recursion continues until all nodes are processed. This method efficiently computes the desired sum of left leaves.

9. Test Cases

For example, in a tree with a structure where the root node is 3, and it has a left child of 9 and a right child with a further left leaf of 15, the output should be 24, as both 9 and 15 are left leaves. If the tree consists of a single node, the result would be 0. Other test cases should also account for trees with no left leaves, multiple left leaves, and trees where all leaf nodes are left, children.

10. Final Thoughts

This problem is an excellent exercise in tree traversal, particularly in using depth-first search to solve tree-related problems. The recursive nature of the problem makes it well-suited for DFS, allowing us to efficiently find and sum all left leaves. Understanding how to identify and accumulate values from left leaves can be applied to various tree-based problems. The solution works for a variety of edge cases and efficiently handles trees of varying sizes.