Documentation for Problem 113: Path Sum II

Problem Description

The task is to find all root-to-leaf paths in a binary tree where the sum of the node values equals a given target sum. Each path should be returned as a list of node values, not references.

Definitions

- **Root-to-leaf path:** A path that starts at the root node and ends at a leaf node. A leaf node is a node with no children.
- **Binary tree:** A tree data structure in which each node has at most two children.

Input

- *root:* The root of the binary tree.
- *targetSum:* The integer target sum.

Output

- A list of lists, where each list represents a root-to-leaf path whose node values sum to targetSum.

Example 1

Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22

Output: [[5,4,11,2],[5,8,4,5]]

Explanation: *There are two paths whose sum equals targetSum:*

- 5 + 4 + 11 + 2 = 22
- 5 + 8 + 4 + 5 = 22

Example 2

Input: root = [1,2,3], targetSum = 5

Output: []

Example 3

Input: root = [1,2], targetSum = 0

Output: []

Constraints

- The number of nodes in the tree is in the range [0, 5000].
- -1000 <= Node.val <= 1000
- -1000 <= targetSum <= 1000

Solution

The solution uses Depth-First Search (DFS) to traverse the binary tree and find all paths that sum to the targetSum.

Detailed Explanation

1. **TreeNode Definition:**
   - A TreeNode class is defined to represent each node in the binary tree. It contains the node value (val), a left child (left), and a right child (right).

2. **DFS Helper Function:**
   - The dfs function is defined within the pathSum method. It takes three arguments: the current node (node), the current path (current_path), and the current sum of node values (current_sum).

3. **Base Case:**
   - If the current node is None, the function returns immediately.

4. **Path and Sum Update:**
   - The current node's value is added to the current_path, and its value is added to the current_sum.

5. **Leaf Node Check:**
   - If the current node is a leaf (no left or right children) and current_sum equals targetSum, the current path is added to the result list.

6. **Recursive Traversal:**
   - The function recursively calls itself for the left and right children of the current node.

7. **Backtracking:**
   - After traversing the left and right children, the current node is removed from the current_path to backtrack.

8. **Initialization and Result:**
   - An empty list result is initialized to store the valid paths. The dfs function is called with the root node, an empty list for the current path, and 0 for the initial sum.
   - The result list is returned as the output.

Time Complexity

- The time complexity of this solution is O(N), where N is the number of nodes in the tree. This is because each node is visited once.

Space Complexity

- The space complexity is O(N) due to the recursion stack and the space required to store the paths in the result list.