Finding Paths with Target Sum in Binary Tree LeetCode

Given the root of a binary tree and an integer targetSum, return *the number of paths where the sum of the values along the path equals* targetSum.

The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).

Example:

```
              10
             /   \
           5       -3
         /   \        \
       3       2        11
     /   \       \
   3      -2       1
```

TargetSum = 8

Output: The Number of Paths with Sum 8: 3 [(5 → 3), (5 → 2 → 1), (-3 → 11)]

Approach 1: Function to count the number of paths in the binary tree that sum to 'k'.

- Initialize an empty vector **path** to track the current path and **ans** to store the answer.
- Start traversal from the root.
- Recursively traverse the left and right subtrees.
- For each node, update the path with its value and calculate sums along the path.
- If the sum equals the target **k**, increment **ans**.
- **Time Complexity: O(N^2) in the worst case, where N is the number of nodes in the tree (each node is considered multiple times).**
- **Space Complexity: O(H) where H is the height of the tree (stack space for recursion).**

Approach 2: Optimized function to count the number of paths in the binary tree that sum to 'k'.

- Initialize a **path** map to store prefix sums and **ans** for the answer.

- Initialize **currSum** to 0 and set **path[0]** to 1 (to account for paths starting from the root).
- Start the optimized path sum calculation.
- Recursively traverse the tree while updating **currSum** and checking if there are paths with sum **k**.
- Use the **path** map to find the number of paths.
- **Time Complexity: O(N) as it visits each node exactly once.**
- **Space Complexity: O(N) for the path map and O(H) for recursion stack.**

Conclusion:

- Approach 2 is better in terms of time complexity (linear time) compared to Approach 1 (quadratic time).
- Approach 2 also uses extra space for the **path** map, but it significantly improves the time complexity.
- Therefore, Approach 2 is the better choice for counting paths with the given sum in a binary tree.