



Count Paths for a Sum (medium)

We'll cover the following



- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

Problem Statement#

Given a binary tree and a number 'S', find all paths in the tree such that the sum of all the node values of each path equals 'S'. Please note that the paths can start or end at any node but all paths must follow direction from parent to child (top to bottom).

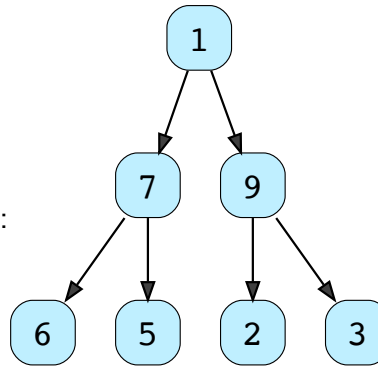
Example 1:

S: 12

Output: 3

Explanation: There are three paths with sum '12':

7 -> 5, 1 -> 9 -> 2, and 9 -> 3

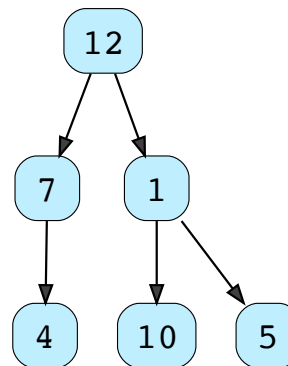
**Example 2:**

S: 11

Output: 2

Explanation: Here are the two paths with sum '11':

7 -> 4 . and 1 -> 10.



Try it yourself#

Try solving this question here:



Java



Python3



JS



C++

```
class TreeNode {  
  
    constructor(value) {  
        this.value = value;  
        this.left = null;  
        this.right = null;  
    }  
};  
  
const count_paths = function(root, S) {  
    // TODO: Write your code here  
    return -1  
};  
  
var root = new TreeNode(12)  
root.left = new TreeNode(7)  
root.right = new TreeNode(1)  
root.left.left = new TreeNode(4)  
root.right.left = new TreeNode(10)  
root.right.right = new TreeNode(5)  
console.log(`Tree has paths: ${count_paths(root, 11)}`)
```

RunSaveReset

Solution#

This problem follows the Binary Tree Path Sum



(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5642684278505472/>) pattern. We can follow the same **DFS** approach. But there will be four differences:

1. We will keep track of the current path in a list which will be passed to every recursive call.
2. Whenever we traverse a node we will do two things:

- Add the current node to the current path.
 - As we added a new node to the current path, we should find the sums of all sub-paths ending at the current node. If the sum of any sub-path is equal to 'S' we will increment our path count.
3. We will traverse all paths and will not stop processing after finding the first path.
 4. Remove the current node from the current path before returning from the function. This is needed to **Backtrack** while we are going up the recursive call stack to process other paths.

Code#

Here is what our algorithm will look like:

 Java	 Python3	 C++	 JS
--	---	---	--

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com

class TreeNode {
  constructor(val, left = null, right = null) {
    this.val = val;
    this.left = left;
    this.right = right;
  }
}

function count_paths(root, S) {
  return count_paths_recursive(root, S, new Deque());
}

function count_paths_recursive(currentNode, S, currentPath) {
  if (currentNode === null) {
    return 0;
  }
```

```
// add the current node to the path
currentPath.push(currentNode.val);
let pathCount = 0,
    pathSum = 0;
// find the sums of all sub-paths in the current path list
for (i = currentPath.length - 1; i >= 0; i--) {
    pathSum += currentPath[i];
    // if the sum of any sub-path is equal to 'S' we increment our path count.
    if (pathSum === S) {
        pathCount += 1;
    }
}
// traverse the left sub-tree
pathCount += count_paths_recursive(currentNode.left, S, currentPath);
// traverse the right sub-tree
pathCount += count_paths_recursive(currentNode.right, S, currentPath);

// remove the current node from the path to backtrack
// we need to remove the current node while we are going up the recursive call
currentPath.pop();
return pathCount;
}

const root = new TreeNode(12);
root.left = new TreeNode(7);
root.right = new TreeNode(1);
root.left.left = new TreeNode(4);
root.right.left = new TreeNode(10);
root.right.right = new TreeNode(5);
console.log(`Tree has paths: ${count_paths(root, 11)}`);
```

[Run](#)[Save](#)[Reset](#)

Time complexity#

The time complexity of the above algorithm is $O(N^2)$ in the worst case, where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once, but for every node, we iterate the

current path. The current path, in the worst case, can be $O(N)$ (in the case of a skewed tree). But, if the tree is balanced, then the current path will be equal to the height of the tree, i.e., $O(\log N)$. So the best case of our algorithm will be $O(N \log N)$.

Space complexity#

The space complexity of the above algorithm will be $O(N)$. This space will be used to store the recursion stack. The worst case will happen when the given tree is a linked list (i.e., every node has only one child). We also need $O(N)$ space for storing the `currentPath` in the worst case.

Overall space complexity of our algorithm is $O(N)$.

Interviewing soon? We've partnered with Hired so that companies apply to [utm_source=educative&utm_medium=lesson&utm_location=US&utm_campaign=hired](https://www.educative.io/courses/grokking-the-coding-interview/xV2J7jvN1or?utm_source=educative&utm_medium=lesson&utm_location=US&utm_campaign=hired)

[← Back](#)[Next →](#)[Path With Given Sequence \(medium\)](#)[Problem Challenge 1](#)[Mark as Completed](#)[Report an Issue](#)

