

Grokking the Coding Interview: Patterns for Coding Questions

35% completed

Count Paths for a Sum (medium)

- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Count Paths for a Sum (medium)

We'll cover the following

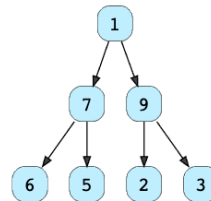
- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

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).

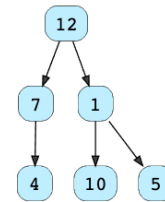
Example 1:

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:

```
1 class TreeNode {
2
3   constructor(value) {
4     this.value = value;
5     this.left = null;
6     this.right = null;
7   }
8 };
9
10
11 const count_paths = function(root, S) {
12   // TODO: Write your code here
13   return -1
14 };
15
16
17 var root = new TreeNode(12)
18 root.left = new TreeNode(7)
19 root.right = new TreeNode(1)
20 root.left.left = new TreeNode(4)
21 root.right.left = new TreeNode(10)
22 root.right.right = new TreeNode(5)
23 console.log('Tree has paths: ${count_paths(root, 11)}')
24
```

RUN

SAVE

RESET



Solution

This problem follows the [Binary Tree Path Sum](#) pattern. We can follow the same DFS approach. But there will

Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Top 'K' Elements

- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)
- Frequency Sort (medium)
- Kth Largest Number in a Stream (medium)
- 'K' Closest Numbers (medium)
- Maximum Distinct Elements (medium)
- Sum of Elements (medium)
- Rearrange String (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: K-way merge

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: 0/1 Knapsack (Dynamic Programming)

- Introduction
- 0/1 Knapsack (medium)
- Equal Subset Sum Partition (medium)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)

- Introduction

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:

```
1 const Deque = require('./collections/deque'); //http://www.collectionsjs.com
2
3 class TreeNode {
4   constructor(val, left = null, right = null) {
5     this.val = val;
6     this.left = left;
7     this.right = right;
8   }
9 }
10
11 function count_paths(root, S) {
12   return count_paths_recursive(root, S, new Deque());
13 }
14
15 function count_paths_recursive(currentNode, S, currentPath) {
16   if (currentNode === null) {
17     return 0;
18   }
19
20   // add the current node to the path
21   currentPath.push(currentNode.val);
22   let pathCount = 0,
23       pathSum = 0;
24   // find the sums of all sub-paths in the current path list
25   for (i = currentPath.length - 1; i >= 0; i--) {
26     pathSum += currentPath[i];
27     // if the sum of any sub-path is equal to 'S' we increment our path count.
28     if (pathSum === S) {
```

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 you, instead of the other way around. [See how](#)

✓ MARK AS COMPLETED

← Back

Path With Given Sequence (medium)

Next →

Problem Challenge 1

Report an Issue Ask a Question

