

## Grokking the Coding Interview: Patterns for Coding Questions

34% completed

Search Course

(medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

### Pattern: Tree Depth First Search

Introduction

Binary Tree Path Sum (easy)

All Paths for a Sum (medium)

Sum of Path Numbers (medium)

Path With Given Sequence (medium)

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 Greater Element

## All Paths for a Sum (medium)

We'll cover the following

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

### Problem Statement

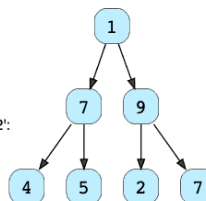
Given a binary tree and a number 'S', find all paths from root-to-leaf such that the sum of all the node values of each path equals 'S'.

Example 1:

S: 12

Output: 2

Explanation: There are the two paths with sum '12':  
1 -> 7 -> 4 and 1 -> 9 -> 2

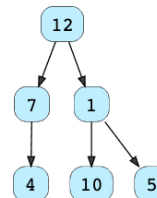


Example 2:

S: 23

Output: 2

Explanation: Here are the two paths with sum '23':  
12 -> 7 -> 4 and 12 -> 1 -> 10



### Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 import java.util.*;
2
3 class TreeNode {
4     int val;
5     TreeNode left;
6     TreeNode right;
7
8     TreeNode(int x) {
9         val = x;
10    }
11 }
12
13 class FindAllTreePaths {
14     public static List<List<Integer>> findPaths(TreeNode root, int sum) {
15         List<List<Integer>> allPaths = new ArrayList<>();
16         // TODO: Write your code here
17         return allPaths;
18     }
19
20     public static void main(String[] args) {
21         TreeNode root = new TreeNode(12);
22         root.left = new TreeNode(7);
23         root.right = new TreeNode(1);
24         root.left.left = new TreeNode(4);
25         root.right.left = new TreeNode(10);
26         root.right.right = new TreeNode(5);
27         int sum = 23;
28         List<List<Integer>> result = FindAllTreePaths.findPaths(root, sum);
```

RUN

SAVE

RESET

### Solution

- 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

## 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

This problem follows the [Binary Tree Path Sum](#) pattern. We can follow the same DFS approach. There will be two differences:

1. Every time we find a root-to-leaf path, we will store it in a list.
2. We will traverse all paths and will not stop processing after finding the first path.

## Code

Here is what our algorithm will look like:

```

Java Python3 C++ JS
28
29 // if the current node is a leaf and its value is equal to sum, save the current path
30 if (currentNode.val == sum && currentNode.left == null && currentNode.right == null) {
31     allPaths.add(new ArrayList<Integer>(currentPath));
32 } else {
33     // traverse the left sub-tree
34     findPathsRecursive(currentNode.left, sum - currentNode.val, currentPath, allPaths);
35     // traverse the right sub-tree
36     findPathsRecursive(currentNode.right, sum - currentNode.val, currentPath, allPaths);
37 }
38
39 // remove the current node from the path to backtrack,
40 // we need to remove the current node while we are going up the recursive call stack.
41 currentPath.remove(currentPath.size() - 1);
42 }
43
44 public static void main(String[] args) {
45     TreeNode root = new TreeNode(12);
46     root.left = new TreeNode(7);
47     root.right = new TreeNode(1);
48     root.left.left = new TreeNode(4);
49     root.right.left = new TreeNode(10);
50     root.right.right = new TreeNode(5);
51     int sum = 23;
52     List<List<Integer>> result = FindAllTreePaths.findPaths(root, sum);
53     System.out.println("Tree paths with sum " + sum + ": " + result);
54 }

```

## Time complexity

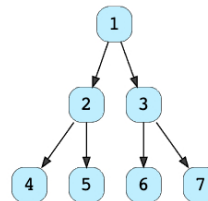
The time complexity of the above algorithm is  $O(N^2)$ , where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once (which will take  $O(N)$ ), and for every leaf node we might have to store its path which will take  $O(N)$ .

We can calculate a tighter time complexity of  $O(N \log N)$  from the space complexity discussion below.

## Space complexity

If we ignore the space required for the `allPaths` list, the space complexity of the above algorithm will be  $O(N)$  in the worst case. 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).

How can we estimate the space used for the `allPaths` array? Take the example of the following balanced tree:



Here we have seven nodes (i.e.,  $N = 7$ ). Since, for binary trees, there exists only one path to reach any leaf node, we can easily say that total root-to-leaf paths in a binary tree can't be more than the number of leaves. As we know that there can't be more than  $N/2$  leaves in a binary tree, therefore the maximum number of elements in `allPaths` will be  $O(N/2) = O(N)$ . Now, each of these paths can have many nodes in them. For a balanced binary tree (like above), each leaf node will be at maximum depth. As we know that the depth (or height) of a balanced binary tree is  $O(\log N)$  we can say that, at the most, each path can have  $\log N$  nodes in it. This means that the total size of the `allPaths` list will be  $O(N * \log N)$ . If the tree is not balanced, we will still have the same worst-case space complexity.

From the above discussion, we can conclude that the overall space complexity of our algorithm is  $O(N * \log N)$ .

Also from the above discussion, since for each leaf node, in the worst case, we have to copy  $\log N$  nodes to store its path, therefore the time complexity of our algorithm will also be  $O(N * \log N)$ .

## Similar Problems

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

(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

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

Alien Dictionary (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

**Problem 1:** Given a binary tree, return all root-to-leaf paths.

*Solution:* We can follow a similar approach. We just need to remove the “check for the path sum”.

**Problem 2:** Given a binary tree, find the root-to-leaf path with the maximum sum.

*Solution:* We need to find the path with the maximum sum. As we traverse all paths, we can keep track of the path with the maximum sum.

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around. [See how](#)

← Back

MARK AS COMPLETED

Next →

Binary Tree Path Sum (easy)

Sum of Path Numbers (medium)

Report an Issue

Ask a Question