

Grokking the Coding Interview: Patterns for Coding Questions

37% 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

Solution Review: Problem Challenge 2

We'll cover the following

- Path with Maximum Sum (hard)
- Solution
- Code
 - Time complexity
 - Space complexity

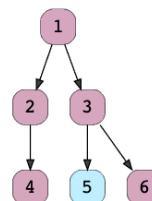
Path with Maximum Sum (hard)

Find the path with the maximum sum in a given binary tree. Write a function that returns the maximum sum. A path can be defined as a **sequence of nodes between any two nodes** and doesn't necessarily pass through the root.

Example 1:

Output: 16

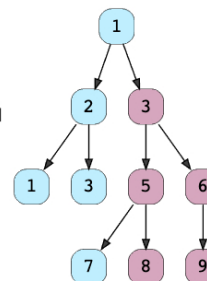
Explanation: The path with maximum sum is: [4, 2, 1, 3, 6]



Example 2:

Output: 31

Explanation: The path with maximum sum is: [8, 5, 3, 6, 9]



Solution

This problem follows the [Binary Tree Path Sum](#) pattern and shares the algorithmic logic with [Tree Diameter](#). We can follow the same [DFS](#) approach. The only difference will be to ignore the paths with negative sums. Since we need to find the overall maximum sum, we should ignore any path which has an overall negative sum.

Code

Here is what our algorithm will look like, the most important changes are in the highlighted lines:

```
Java Python3 C++ JS JS
1 class TreeNode {
2   constructor(val, left = null, right = null) {
3     this.val = val;
4     this.left = left;
5     this.right = right;
6   }
7 }
8
9
10 class MaximumPathSum {
11   find_maximum_path_sum(root) {
12     this.globalMaximumSum = -Infinity;
13     this.find_maximum_path_sum_recursive(root);
14     return this.globalMaximumSum;
15   }
16
17   find_maximum_path_sum_recursive(currentNode) {
18     if (currentNode === null) {
19       return 0;
20     }
21
22     let maxPathSumFromLeft = this.find_maximum_path_sum_recursive(currentNode.left);
23     let maxPathSumFromRight = this.find_maximum_path_sum_recursive(currentNode.right);
24
25     // ignore paths with negative sums, since we need to find the maximum sum we should
```

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

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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

26 // ignore any path which has an overall negative sum.
27 maxPathSumFromLeft = Math.max(maxPathSumFromLeft, 0);
28 maxPathSumFromRight = Math.max(maxPathSumFromRight, 0);

RUN

SAVE

RESET

Close

2.413s

Output

Maximum Path Sum: 6
Maximum Path Sum: 31
Maximum Path Sum: -1

Time complexity

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

Space complexity

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).

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

Next →

Problem Challenge 2

Introduction