

Grokking the Coding Interview: Patterns for Coding Questions

33% completed

Pattern: Fast & Slow pointers

- Introduction
- LinkedList Cycle (easy)
- Start of LinkedList Cycle (medium)
- Happy Number (medium)
- Middle of the LinkedList (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: Merge Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- 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: Cyclic Sort

- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (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: In-place Reversal of a LinkedList

- Introduction
- Reverse a LinkedList (easy)
- Reverse a Sub-list (medium)

Binary Tree Path Sum (easy)

We'll cover the following

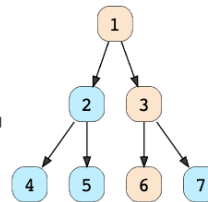
- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

Problem Statement

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

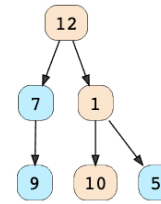
Example 1:

S: 10
Output: true
Explanation: The path with sum '10' is highlighted

**Example 2:**

S: 23
Output: true
Explanation: The path with sum '23' is highlighted

S: 16
Output: false
Explanation: There is no root-to-leaf path with sum '16'.



Try it yourself

Try solving this question here:

Java

Python3

JS

C++

```
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 has_path = function(root, sum) {
12   // TODO: Write your code here
13   return false;
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(9)
21 root.right.left = new TreeNode(10)
22 root.right.right = new TreeNode(5)
23 console.log('Tree has path: ${has_path(root, 23)}')
24 console.log('Tree has path: ${has_path(root, 16)}')
25
```

RUN

SAVE

RESET



Solution

As we are trying to search for a root-to-leaf path, we can use the **Depth First Search (DFS)** technique to solve this problem.

- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)
- Level Order Successor (easy)
- Connect Level Order Siblings (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

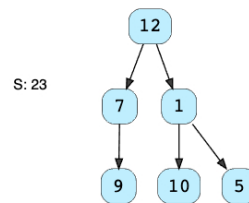
us problem.

To recursively traverse a binary tree in a DFS fashion, we can start from the root and at every step, make two recursive calls one for the left and one for the right child.

Here are the steps for our Binary Tree Path Sum problem:

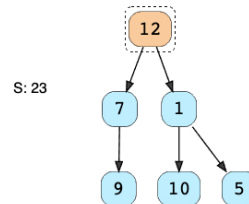
1. Start DFS with the root of the tree.
2. If the current node is not a leaf node, do two things:
 - Subtract the value of the current node from the given number to get a new sum => $S = S - \text{node.value}$
 - Make two recursive calls for both the children of the current node with the new number calculated in the previous step.
3. At every step, see if the current node being visited is a leaf node and if its value is equal to the given number 'S'. If both these conditions are true, we have found the required root-to-leaf path, therefore return **true**.
4. If the current node is a leaf but its value is not equal to the given number 'S', return false.

Let's take the example-2 mentioned above to visually see our algorithm:



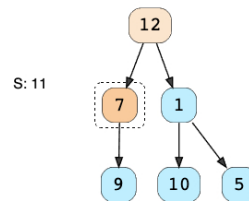
Let's start with the root node

1 of 10



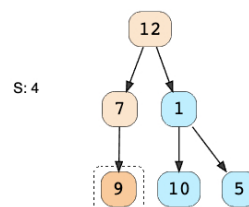
Not a leaf node, make recursive calls for children.

2 of 10



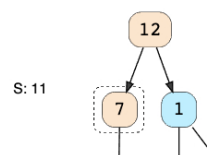
Not a leaf node, make recursive call for the left child.

3 of 10



Leaf node, but $S \neq 9$, therefore return false.

4 of 10



S: 11

- 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

Pattern: Bitwise XOR

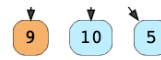
- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

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

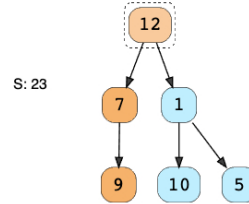
Pattern: K-way merge

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)



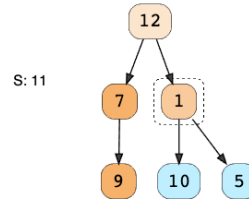
After traversing the left-child, make a recursive call for the right child. This recursive all will return false, as the right child is 'null'.

5 of 10



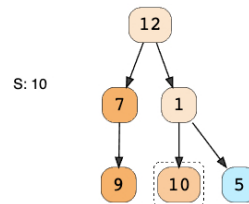
After traversing the left-child, make a recursive call for the right child, as the left-child failed in finding the path.

6 of 10



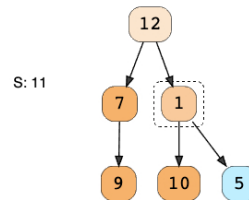
Not a leaf node, make recursive call for the left child.

7 of 10



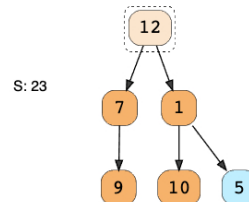
Leaf node, but $S == 10$, we have found a path; therefore return true.

8 of 10



As the left-child returned 'true', return 'true' without processing further.

9 of 10



As the right-child returned 'true', return 'true', we have found the path.

10 of 10



Code

Here is what our algorithm will look like:



Smallest Number Hange (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

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

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

Mark Course as Completed

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

```
7 }
8
9
10 function hasPath(root, sum) {
11     if (root === null) {
12         return false;
13     }
14
15     // if the current node is a leaf and its value is equal to the sum, we've found a path
16     if (root.val === sum && root.left === null && root.right === null) {
17         return true;
18     }
19
20     // recursively call to traverse the left and right sub-tree
21     // return true if any of the two recursive call return true
22     return hasPath(root.left, sum - root.val) || hasPath(root.right, sum - root.val);
23 }
24
25
26 const root = new TreeNode(12);
27 root.left = new TreeNode(7);
28 root.right = new TreeNode(1);
29 root.left.left = new TreeNode(9);
30 root.right.left = new TreeNode(10);
31 root.right.right = new TreeNode(5);
32 console.log('Tree has path: ${hasPath(root, 23)}');
33 console.log('Tree has path: ${hasPath(root, 16)}');
```

RUNSAVERESET

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](#)

← Back

MARK AS COMPLETED

Next →

Introduction

All Paths for a Sum (medium)