

Grokking the Coding Interview: Patterns for Coding Questions

32% completed

Search Course

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

Minimum Depth of a Binary Tree (easy)

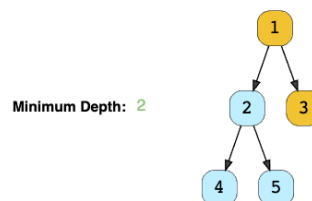
We'll cover the following

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

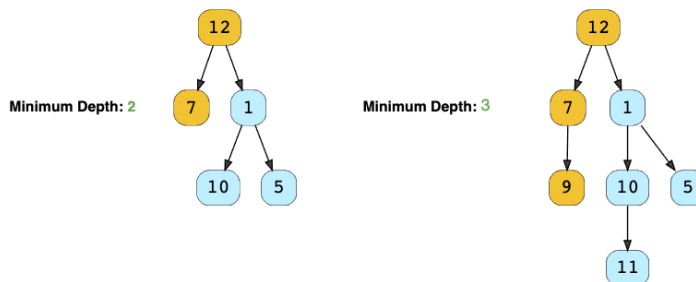
Problem Statement

Find the minimum depth of a binary tree. The minimum depth is the number of nodes along the **shortest path from the root node to the nearest leaf node**.

Example 1:



Example 2:



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 find_minimum_depth = function(root) {
12   // TODO: Write your code here
13   return -1;
14 };
15
16
17
18 var root = new TreeNode(12)
19 root.left = new TreeNode(7)
20 root.right = new TreeNode(1)
21 root.right.left = new TreeNode(10)
22 root.right.right = new TreeNode(5)
23 console.log('Tree Minimum Depth: ${find_minimum_depth(root)}')
24 root.left.left = new TreeNode(9)
25 root.right.left.left = new TreeNode(11)
26 console.log('Tree Minimum Depth: ${find_minimum_depth(root)}')
27
```

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

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)

RUN

SAVE

RESET

Solution

This problem follows the [Binary Tree Level Order Traversal](#) pattern. We can follow the same **BFS** approach. The only difference will be, instead of keeping track of all the nodes in a level, we will only track the depth of the tree. As soon as we find our first leaf node, that level will represent the minimum depth of the tree.

Code

Here is what our algorithm will look like, only the highlighted lines have changed:

JavaPython3C++JS

```
1  const Deque = require('./collections/deque'); //http://www.collectionsjs.com
2
3
4  class TreeNode {
5      constructor(val) {
6          this.val = val;
7          this.left = null;
8          this.right = null;
9      }
10 }
11
12 function find_minimum_depth(root) {
13     if (root === null) {
14         return 0;
15     }
16
17     const queue = new Deque();
18     queue.push(root);
19     let minimumTreeDepth = 0;
20     while (queue.length > 0) {
21         minimumTreeDepth += 1;
22         levelSize = queue.length;
23         for (i = 0; i < levelSize; i++) {
24             currentNode = queue.shift();
25
26             // check if this is a leaf node
27             if (currentNode.left === null && currentNode.right === null) {
28                 return minimumTreeDepth;
29             }
30         }
31     }
32 }
```

RUN

SAVE

RESET

Output

3.877s

Close

Tree Minimum Depth: 2
Tree Minimum Depth: 3

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 $O(N)$ which is required for the queue. Since we can have a maximum of $N/2$ nodes at any level (this could happen only at the lowest level), therefore we will need $O(N)$ space to store them in the queue.

Similar Problems

Problem 1: Given a binary tree, find its maximum depth (or height).

Solution: We will follow a similar approach. Instead of returning as soon as we find a leaf node, we will keep traversing for all the levels, incrementing **maximumDepth** each time we complete a level. Here is what the code will look like:

JavaPython3C++JS

```
6      this.val = val;
7      this.left = null;
8      this.right = null;
9  }
10 }
11
12
13 function find_maximum_depth(root) {
14     if (root === null) {
15         return 0;
16     }
17
18     const queue = new Deque();
19     queue.push(root);
20     let maximumTreeDepth = 0;
21     while (queue.length > 0) {
22         maximumTreeDepth += 1;
23         const levelSize = queue.length;
24         for (i = 0; i < levelSize; i++) {
```

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

25currentNode = queue.shift();

26

27// insert the children of current node in the queue

28if (currentNode.left !== null) {

29queue.push(currentNode.left);

30}

31if (currentNode.right !== null) {

32queue.push(currentNode.right);

33}

RUN

SAVE

RESET

Close

Output3.664s

Tree Maximum Depth: 3

Tree Maximum Depth: 4

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 →

Level Averages in a Binary Tree (easy)

Level Order Successor (easy)

Report an Issue