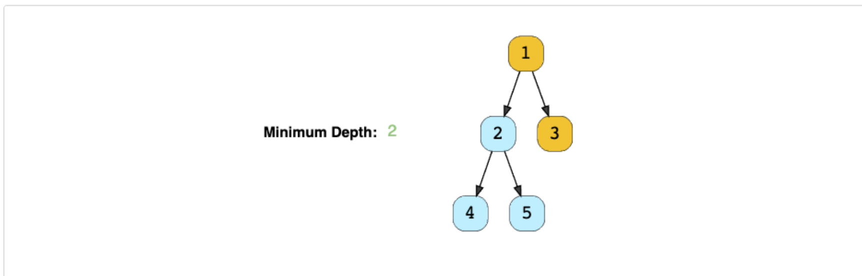
Minimum Depth of a Binary Tree (easy)

We'll cover the following

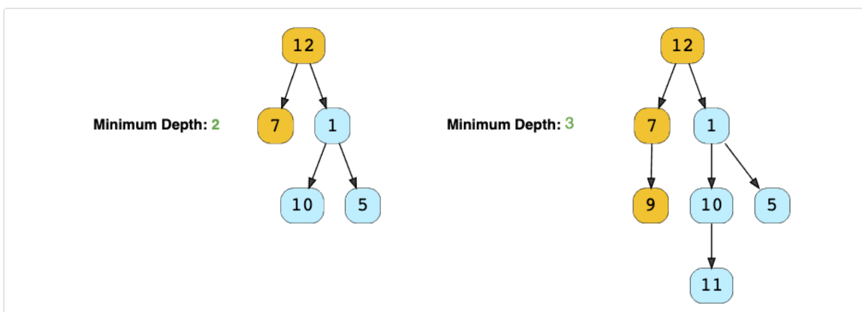
Problem Statement

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:

Java | Python3 | JS | C++

```
class TreeNode {

  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }
};


const find_minimum_depth = function(root) {
  // TODO: Write your code here
  return -1;
};



var root = new TreeNode(12)
root.left = new TreeNode(7)
root.right = new TreeNode(1)
root.right.left = new TreeNode(10)
root.right.right = new TreeNode(5)
console.log(`Tree Minimum Depth: ${find_minimum_depth(root)}`)
root.left.left = new TreeNode(9)
root.right.left.left = new TreeNode(11)
console.log(`Tree Minimum Depth: ${find_minimum_depth(root)}`)

```

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:

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com


class TreeNode {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
  }
}

function find_minimum_depth(root) {
  if (root === null) {
    return 0;
  }

  const queue = new Deque();
  queue.push(root);
  let minimumTreeDepth = 0;
  while (queue.length > 0) {
    minimumTreeDepth += 1;
    levelSize = queue.length;
    for (i = 0; i < levelSize; i++) {
      currentNode = queue.shift();

      // check if this is a leaf node
      if (currentNode.left === null && currentNode.right === null) {
        return minimumTreeDepth;
```

RUN SAVE RESET

Close

Output 3.877s

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:

Java | Python3 | C++ | JS

```
    this.val = val;
    this.left = null;
    this.right = null;
  }
}


function find_maximum_depth(root) {
  if (root === null) {
    return 0;
  }

  const queue = new Deque();
  queue.push(root);
  let maximumTreeDepth = 0;
  while (queue.length > 0) {
    maximumTreeDepth += 1;
    const levelSize = queue.length;
    for (i = 0; i < levelSize; i++) {
```

- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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)

```
    currentNode = queue.shift();

    // insert the children of current node in the queue
    if (currentNode.left !== null) {
      queue.push(currentNode.left);
    }
    if (currentNode.right !== null) {
      queue.push(currentNode.right);
```

RUN SAVE RESET

Close

Output 3.664s

```
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

MARK AS COMPLETED

← Back — Level Averages in a Binary Tree (easy)

Next → — Level Order Successor (easy)

Report an Issue