111. Minimum Depth of Binary Tree

Solved

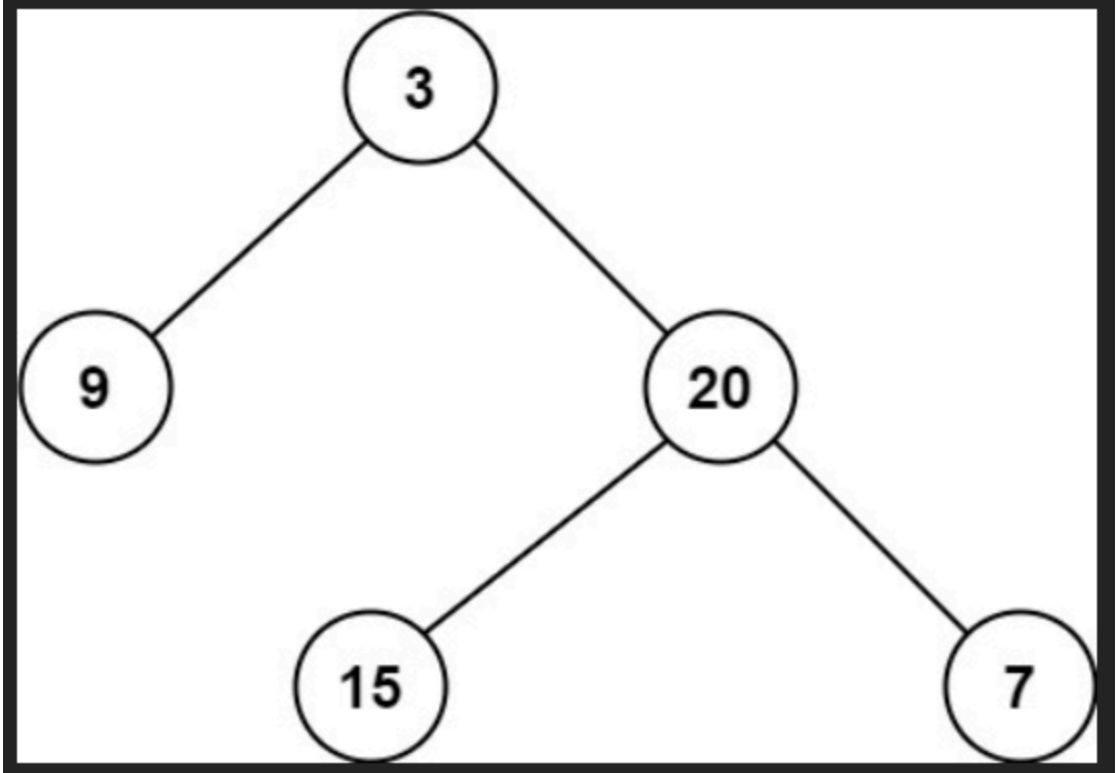
Easy | Topics | Companies

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 2
```

Example 2:

```
Input: root = [2,null,3,null,4,null,5,null,6]
Output: 5
```

Constraints:

- The number of nodes in the tree is in the range $[0, 10^5]$.
- $-1000 <= Node.val <= 1000$

Python:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
```

```
#         self.left = left
#         self.right = right
from collections import deque
from typing import Optional

class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0   # empty tree has depth 0

        queue = deque([(root, 1)])  # (node, current_depth)

        while queue:
            node, depth = queue.popleft()

            # if it's a leaf node -> return the depth
            if not node.left and not node.right:
                return depth

            # otherwise keep going
            if node.left:
                queue.append((node.left, depth + 1))
            if node.right:
                queue.append((node.right, depth + 1))
```

JavaScript:

```
var minDepth = function(root) {
    if (!root) return 0;

    let queue = [[root, 1]]; // store node and depth

    while (queue.length > 0) {
        let [node, depth] = queue.shift();

        // check if it's a leaf node
        if (!node.left && !node.right) {
            return depth;
        }

        if (node.left) queue.push([node.left, depth + 1]);
        if (node.right) queue.push([node.right, depth + 1]);
    }
};
```

Java:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int minDepth(TreeNode root) {
        if (root == null) {
            return 0; // Empty tree
        }

        // If one child is null, we must go down the other child
        if (root.left == null) {
            return 1 + minDepth(root.right);
        }
        if (root.right == null) {
            return 1 + minDepth(root.left);
        }

        // Both children exist → take the smaller depth
        return 1 + Math.min(minDepth(root.left), minDepth(root.right));
    }
}
```