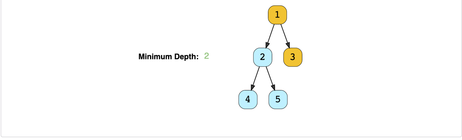
Back To Course Home

Minimum Depth of a Binary Tree (easy)

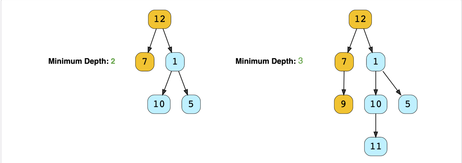
Problem Statement

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++
  int val;
  TreeNode left;
  TreeNode right;

  TreeNode(int x) {
    val = x;
  }
};

class MinimumBinaryTreeDepth {
  public static int findDepth(TreeNode root) {
    // TODO: Write your code here
    return -1;
  }

  public static void main(String[] args) {
    TreeNode 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);
    System.out.println("Tree Minimum Depth: " + MinimumBinaryTreeDepth.findDepth(root));
    root.left.left = new TreeNode(9);
    root.right.left.left = new TreeNode(11);
    System.out.println("Tree Minimum Depth: " + MinimumBinaryTreeDepth.findDepth(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
import java.util.*;

class TreeNode {
  int val;
  TreeNode left;
  TreeNode right;

  TreeNode(int x) {
    val = x;
  }
};

class MinimumBinaryTreeDepth {
  public static int findDepth(TreeNode root) {
    if (root == null)
      return 0;

    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(root);
    int minimumTreeDepth = 0;
    while (!queue.isEmpty()) {
      minimumTreeDepth++;
      int levelSize = queue.size();
      for (int i = 0; i < levelSize; i++) {
        TreeNode currentNode = queue.poll();

        // check if this is a leaf node
        if (currentNode.left == null && currentNode.right == null)

Run
Save
Reset
```

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
import java.util.*;

class TreeNode {
  int val;
  TreeNode left;
  TreeNode right;

  TreeNode(int x) {
    val = x;
  }
};

class MaximumBinaryTreeDepth {
  public static int findDepth(TreeNode root) {
    if (root == null)
      return 0;

    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(root);
    int maximumTreeDepth = 0;
    while (!queue.isEmpty()) {
      maximumTreeDepth++;
      int levelSize = queue.size();
      for (int i = 0; i < levelSize; i++) {
        TreeNode currentNode = queue.poll();
        // insert the children of current node in the queue
        if (currentNode.left != null)
          queue.add(currentNode.left);

Run
Save
Reset
```