



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

WORKSHEET 6

Student Name: Shubham Chauhan

UID:22BCS11888

Branch: CSE

Section/Group: NTPP 603/B

Semester: 06

Date of Performance: 27/02/2025

Subject Name: AP Lab II

Subject Code: 22CSP-351

1. Aim:

A. Maximum Depth of Binary Tree

B. Validate Binary Search Tree

C. Symmetric Tree

2. Source Code:

a.

```
/**  
 * 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;  
 *     }  
 * }
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
*/  
  
// Definition for a binary tree node.  
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;  
    }  
}  
  
public class Solution {  
    public int maxDepth(TreeNode root) {  
        // Base case: if the tree is empty, the depth is 0  
        if (root == null) {  
            return 0;  
        }  
  
        // Recursively find the depth of left and right subtrees  
        int leftDepth = maxDepth(root.left);  
        int rightDepth = maxDepth(root.right);  
  
        // Return the maximum depth plus 1 (to count the current node)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return Math.max(leftDepth, rightDepth) + 1;
    }
}
```

b.

```
/**
 * 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;
 *     }
 * }
 */
// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
TreeNode() {}

TreeNode(int val) { this.val = val; }

TreeNode(int val, TreeNode left, TreeNode right) {
    this.val = val;
    this.left = left;
    this.right = right;
}

}

public class Solution {
    public boolean isValidBST(TreeNode root) {
        // Call the helper function with the full valid range for the root
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    // Helper function to check the validity of the BST with valid range
    private boolean isValidBST(TreeNode node, long min, long max) {
        // Base case: if the node is null, it's valid (an empty tree is a valid BST)
        if (node == null) {
            return true;
        }

        // Check if the current node's value is within the valid range
        if (node.val <= min || node.val >= max) {
            return false;
        }

        // Recursively check the left subtree and right subtree
    }
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return isValidBST(node.left, min, node.val) && isValidBST(node.right, node.val,
max);
    }
}
```

C.

```
/**
 * 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;
 *     }
 * }
 */

// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
TreeNode() {}

TreeNode(int val) { this.val = val; }

TreeNode(int val, TreeNode left, TreeNode right) {
    this.val = val;
    this.left = left;
    this.right = right;
}

}

public class Solution {
    public boolean isSymmetric(TreeNode root) {
        // If the root is null, the tree is symmetric
        if (root == null) {
            return true;
        }

        // Check if the left and right subtrees are mirrors of each other
        return isMirror(root.left, root.right);
    }

    // Helper function to check if two trees are mirrors of each other
    private boolean isMirror(TreeNode left, TreeNode right) {
        // Base case: both are null, so they are mirrors of each other
        if (left == null && right == null) {
            return true;
        }

        // If one of them is null or the values are different, they are not mirrors
    }
```

```

        if (left == null || right == null || left.val != right.val) {
            return false;
        }

        // Recursively check if the left child of the left tree and the right child of
        // the right tree

        // are mirrors, and if the right child of the left tree and the left child of the
        // right tree are mirrors

        return isMirror(left.left, right.right) && isMirror(left.right, right.left);
    }
}

```

3. Screenshot of Outputs:

a.

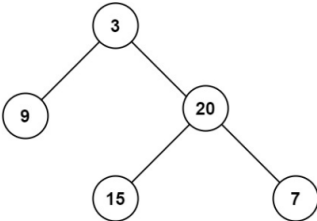
104. Maximum Depth of Binary Tree

Easy Topics Companies

Given the `root` of a binary tree, return its maximum depth.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:



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

13.3K 158 264 Online

Solved

```

36     }
37
38     // Recursively find the depth of left and right subtrees
39     int leftDepth = maxDepth(root.left);
40     int rightDepth = maxDepth(root.right);
41
42     // Return the maximum depth plus 1 (to count the current node)
43     return Math.max(leftDepth, rightDepth) + 1;
44 }
45
46

```

Saved Ln 45, Col 2

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =
[3,9,20,null,null,15,7]

Output

3

Expected

b.

98. Validate Binary Search Tree

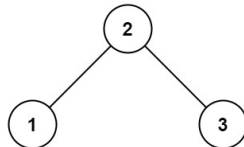
Medium Topics Companies

Given the `root` of a binary tree, determine if it is a valid binary search tree (BST).

A **valid BST** is defined as follows:

- The left **subtree** of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:



Input: `root = [2,1,3]`
Output: `true`

Example 2:

17.4K 226 200 Online

Solved

```

43
44 // Check if the current node's value is within the valid range
45 if (node.val <= min || node.val >= max) {
46     return false;
47 }
48
49 // Recursively check the left subtree and right subtree
50 return isValidBST(node.left, min, node.val) && isValidBST(node.right, node.val, max);
51
52
53
  
```

Saved

Ln 53, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =
[2,1,3]

Output

true

Expected

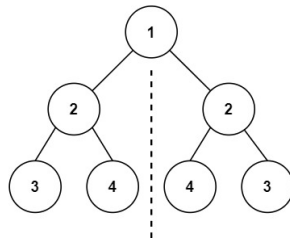
C.

101. Symmetric Tree

Easy Topics Companies

Given the `root` of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Example 1:



Input: `root = [1,2,2,3,4,4,3]`
Output: `true`

Example 2:

15.9K 195 115 Online

Solved

```

51 .....return false;
52 .....}
53 .....}
54 .....// Recursively check if the left child of the left tree and the right child of the right
55 .....// are mirrors, and if the right child of the left tree and the left child of the right
56 .....// tree are mirrors
57 .....return isMirror(left.left, right.right) && isMirror(left.right, right.left);
58 .....}
59 .....}
  
```

Saved

Ln 59, Co

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =
[1,2,2,3,4,4,3]

Output

true

Expected

4. Learning Outcomes

- Learned about trees .
- Learned about types of trees.
- Learned about tree traversal methods.