Experiment 5A

Student Name: Rhythm Tyagi
UID: 22BCS17203
Branch: CSE
Section/Group:IOT_NTPP_602-A
Semester: 6th
Date of Performance:20-01-25
Subject Name: AP2
Subject Code: 22CSP-351

Aim: Maximum Depth of Binary Tree

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

Objective: To determine the maximum depth of a binary tree by finding the longest path from the root to a leaf node using recursive DFS (O(n)) or iterative BFS (O(n)) approaches.

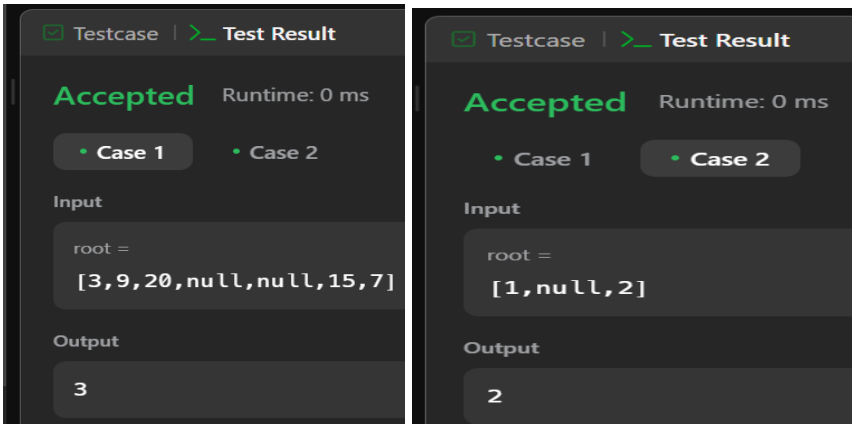
Algorithm:

1. Base Case: If the tree is empty (root == null), return 0.
2. Recursive Case: Compute the depth of the left and right subtrees.
3. Return: 1 + max(leftDepth, rightDepth).

Code:

```
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);
        return 1 + Math.max(leftDepth, rightDepth);
    }
}
```

Output:



Learning Outcomes:

1. Learn to traverse a binary tree using DFS (recursion) and BFS (iteration with a queue).
2. Compute the maximum depth efficiently using recursive divide-and-conquer or level-wise traversal.
3. Optimize tree operations with O(n) time complexity and understand recursion vs. iteration trade-offs.

Experiment 5 B

Aim: Binary Tree Level Order Traversal

Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level)

Objective: To perform level order traversal of a binary tree using Breadth-First Search (BFS) with a queue, returning node values level by level from left to right in O(n) time.

Algorithm:

1. Base Case: If root == null, return an empty list.
2. Initialize a Queue: Add the root node to the queue.
3. Process Levels: While the queue is not empty:
   - Get the size of the current level.
   - Process each node at the level:
     - Remove the node from the queue and store its value.
     - Add its left and right children to the queue (if they exist).
   - Add the current level's values to the result list.
4. Return the final result list.

CODE:

```
import java.util.*;
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> level = new ArrayList<>();
            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                level.add(node.val);
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
```

```
            result.add(level);
        }
        return result;
    }
}
```

Output:

Learning Outcomes:

1. Learn to traverse a binary tree **level by level** using a queue.
2. Utilize **FIFO** structure for processing tree nodes.
3. Optimize traversal with **O(n) time** and **O(n) space** efficiency.

Experiment 5 C

Aim: Symmetric Tree

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

Objective: To determine if a binary tree is symmetric by checking whether its left and right subtrees are mirror images using recursive DFS or iterative BFS (queue-based approach) in O(n) time

Algorithm:

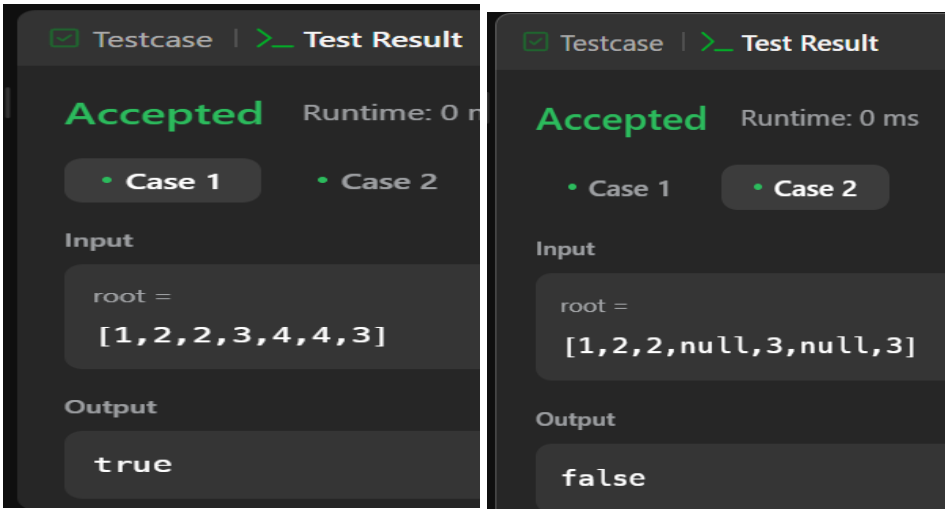
1. **Base Case:** If root == null, return true (an empty tree is symmetric).
2. **Recursive Approach (DFS):**
   - Check if the left and right subtrees are **mirror images**.
   - Both nodes should have the **same value**.
   - Left subtree's left child should match right subtree's right child, and vice versa.
3. **Iterative Approach (BFS using Queue):**
   - Use a **queue** to store nodes in pairs.
   - Compare corresponding nodes while processing the queue.
   - Enqueue children in **mirrored order** (left-right & right-left)

CODE:

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) return true;
        return isMirror(root.left, root.right);
    }
    private boolean isMirror(TreeNode t1, TreeNode t2) {
        if (t1 == null && t2 == null) return true;
        if (t1 == null || t2 == null) return false;
        return (t1.val == t2.val)
                && isMirror(t1.left, t2.right)
                && isMirror(t1.right, t2.left);
    }
```

}

Output:



Learning Outcomes:

1. Learn how to check if a tree is a mirror of itself using recursive DFS and iterative BFS.
2. Implement mirrored traversal using recursion (depth-first) and queue-based iteration (breadth-first).
3. Optimize symmetry checking with O(n) time and space-efficient approaches using recursion or a queue.