Experiment 3 A

Student Name: Deepanshu **UID: 22BCS16692**

Branch:BE-CSE **Section/Group: NTPP 602-A**

Semester:6TH **Date of Performance:10/02/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Maximum Depth of Binary Tree

2. AIM:

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.

3. Algorithm

- Start DFS with the root node at depth 0.
- If the node is null, return the current depth.
- Recursively explore left and right children, increasing depth by 1.
- Return the maximum depth from left or right subtree.

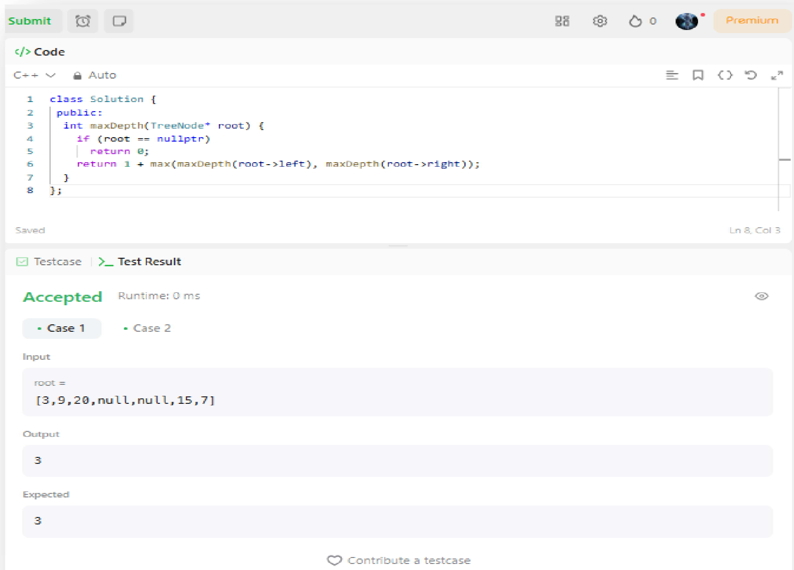
Implemetation/Code

```
class Solution {
public:
int maxDepth(TreeNode* root) {
if (root == nullptr)
return 0;
return 1 + max(maxDepth(root->left), maxDepth(root->right));
}
};
```

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Output



Time Complexity : O(n)

Space Complexity : O(h)

Learning Outcomes:-

- Understand how to use depth-first search for tree traversal.
- Gain skills in calculating the depth or height of binary trees.

Experiment 3 B

Student Name: Deepanshu **UID: 22BCS16443**
Branch:BE-CSE **Section/Group: NTPP- 602(A)**
Semester:6TH **Date of Performance:10/02/25**
Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Binary Tree Level Order Traversal

2. AIM:

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).

3. Algorithm

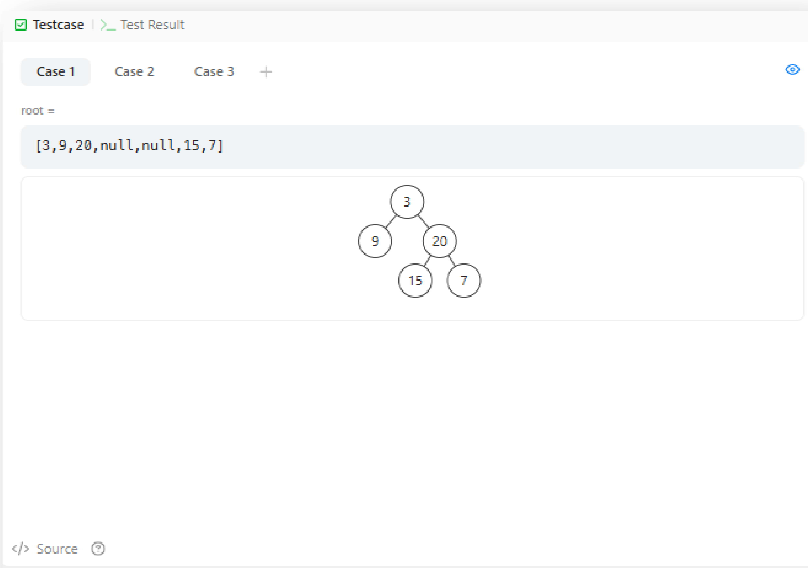
- Create a **queue** and enqueue the root node..
- Create an empty **result list** to store the final level order traversal.
- Append the `level` list to `result`.
- This list contains all levels of the binary tree.

Implemetation/Code:

```
class Solution {
public:
vector<vector<int>> levelOrder(TreeNode* root) {
if (root == nullptr)
return {};
vector<vector<int>> ans;
queue<TreeNode*> q{{root}};
while (!q.empty()) {
vector<int> currLevel;
for (int sz = q.size(); sz > 0; --sz) {
TreeNode* node = q.front();
```

```
q.pop();
currLevel.push_back(node->val);
if (node->left)
q.push(node->left);
if (node->right)
q.push(node->right);
}
ans.push_back(currLevel);
}
return ans;
}
};
```

Output:



Time Complexity : O(k)

Space Complexity : O(h)

Learning Outcomes:-

- Learn how to perform and apply in-order traversal in binary trees to solve problems.
- Python generators to manage state and produce results on demand during tree traversal.