Experiment 6

Student Name: Anshu Kumar **UID: 22BCS16672**

Branch: BE-CSE **Section/Group: 641/A**

Semester: 6th **Date of Performance: 04/03/2025**

Subject Name: AP LAB-II **Subject Code: 22CSP-351**

1.**Aim:**
The aim of this problem is to calculate the maximum depth (or height) of a binary tree.

Objective:

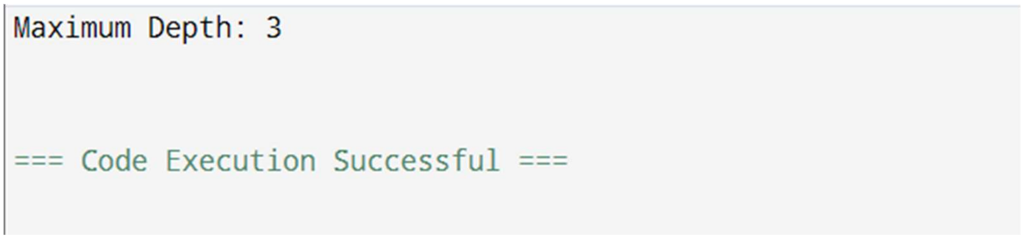
- Understand how to traverse a binary tree.
- Implement the depth calculation using recursion.

Problem Statement:
Given a binary tree, return its maximum depth.

Code Implementation:

```
#include <iostream>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int maxDepth(TreeNode* root) {
    if (!root) {
        return 0;
    }
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return max(leftDepth, rightDepth) + 1;
}

int main() {
    // Example usage:
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
    cout << "Maximum Depth: " << maxDepth(root) << endl; // Output: 3
    return 0;
}
```

Output:



```
Maximum Depth: 3


=== Code Execution Successful ===
```

2. Aim:
The aim of this problem is to validate if a given binary tree is a Binary Search Tree (BST).

Objective:

- Learn how to traverse a tree while validating BST properties.
- Implement an in-order traversal to check if the tree is sorted.

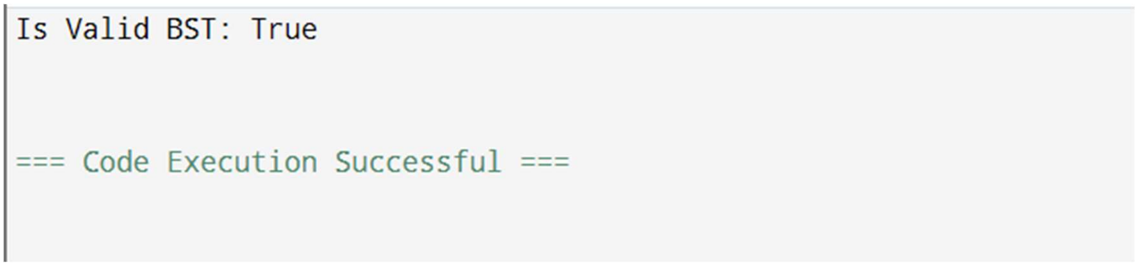
Problem Statement:
Given a binary tree, determine if it is a valid binary search tree (BST).

Code Implementation:

```
#include <iostream>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

bool isValidBSTHelper(TreeNode* root, long minVal, long maxVal) {
    if (!root) {
        return true;
    }
    if (root->val <= minVal || root->val >= maxVal) {
        return false;
    }
    return isValidBSTHelper(root->left, minVal, root->val) && isValidBSTHelper(root->right, root->val, maxVal);
}
```

```
bool isValidBST(TreeNode* root) {
    return isValidBSTHelper(root, LONG_MIN, LONG_MAX);
}

int main() {
    // Example usage:
    TreeNode* root = new TreeNode(2);
    root->left = new TreeNode(1);
    root->right = new TreeNode(3);

    cout << "Is Valid BST: " << isValidBST(root) << endl; // Output: 1 (True)
    return 0;
}
```

Output:



```
Is Valid BST: True


=== Code Execution Successful ===
```

3. Aim:
The aim of this problem is to perform a level order traversal (breadth-first search) on a binary tree.

Objective:

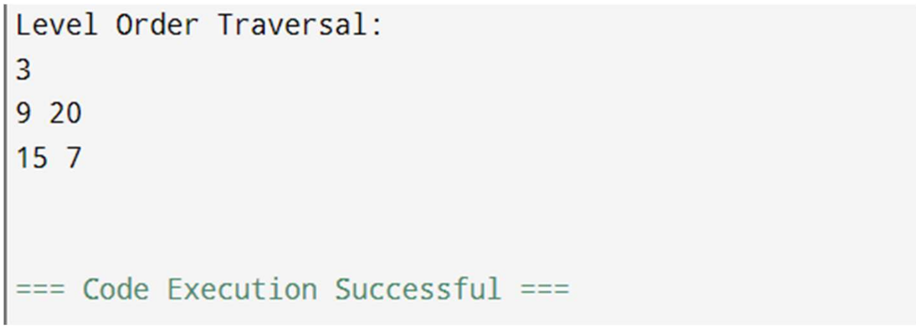
- Understand how to use a queue to traverse the tree level by level.
- Implement the level order traversal algorithm.

Problem Statement:
Given a binary tree, return the level order traversal of its nodes' values.

Code Implementation:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> result;
    if (!root) {
        return result;
    }

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int levelSize = q.size();
        vector<int> levelValues;

        for (int i = 0; i < levelSize; i++) {
            TreeNode* node = q.front();
            q.pop();
            levelValues.push_back(node->val);

            if (node->left) {
                q.push(node->left);
            }
            if (node->right) {
                q.push(node->right);
            }
        }
        result.push_back(levelValues);
    }

    return result;
}

int main() {
    // Example usage:
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

    vector<vector<int>> result = levelOrder(root);
    cout << "Level Order Traversal: " << endl;
    for (const auto& level : result) {
        for (int val : level) {
            cout << val << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:



```
Level Order Traversal:
3
9 20
15 7


=== Code Execution Successful ===
```

Learning Outcomes:

1. Tree Traversal Techniques:
   Learn the different ways to traverse a tree (pre-order, in-order, post-order, and level-order) and apply these techniques to solve real-world problems.
2. Recursive Tree Solutions:
   Develop the understanding of recursion in trees and how it can be used for problems like depth calculation, validation, and others.
3. Binary Search Tree Properties:
   Understand and validate the properties of a Binary Search Tree (BST), such as the left child being less than the parent node and the right child being greater than the parent node.
4. Queue in Tree Traversal:
   Learn the use of a queue data structure to implement breadth-first search (level-order traversal) in a binary tree.