## **Experiment 6**

Name: Aryan Anand UID: 22BET10056

Branch: BE-IT Section/Group: 22BET\_702-B
Semester: 6 Date of Performance: 07-03-25

Subject Name: Advanced Programming Lab-2 Subject Code: 22ITP-351

**Problem 1.** Maximum Depth of Binary Tree- 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.

#### Code:

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

```
Accepted Runtime: 0 ms

• Case 1
• Case 2

Input

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

Output

3

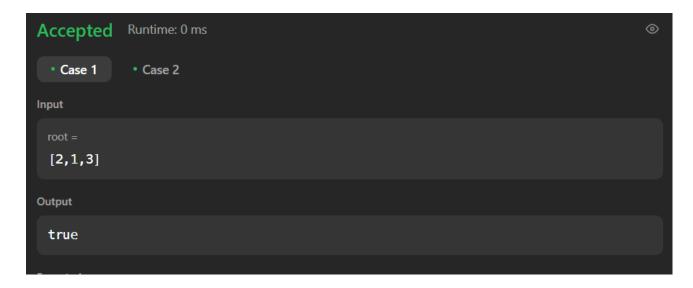
Expected

3
```

**Problem 2.** Validate Binary Search Tree- Given the root of a binary tree, determine if it is a valid binary search tree (BST).

#### Code:

```
class Solution {
public:
  void findInorder(TreeNode* root, vector<int> &inorder) {
     if (!root) return;
     findInorder(root->left, inorder);
     inorder.push_back(root->val);
     findInorder(root->right, inorder);
  }
  bool isValidBST(TreeNode* root) {
     vector<int> inorder;
     findInorder(root, inorder);
     for (int i = 1; i < inorder.size(); i++) {
       if (inorder[i - 1] >= inorder[i]) return false;
     }
     return true;
};
```



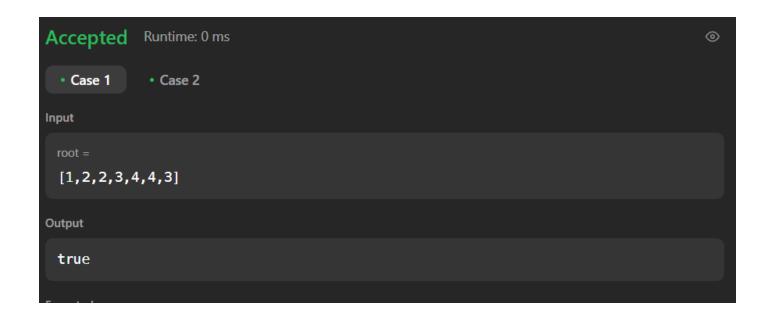
**Problem 3.** Symmetric Tree. - Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

### Code:

```
class Solution {
public:
    bool isMirror(TreeNode* left, TreeNode* right) {
    if (!left && !right) return true;
    if (!left || !right) return false;
    return (left->val == right->val) && isMirror(left->left, right->right) && isMirror(left->right, right->left);
}

bool isSymmetric(TreeNode* root) {
    if (!root) return true;
    return isMirror(root->left, root->right);
}

};
```



**Problem 4.** 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).

#### 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;
 }
};
```

```
Accepted Runtime: 0 ms

• Case 1
• Case 2
• Case 3

Input

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

Output

[[3],[9,20],[15,7]]
```

**Problem 5.** Convert Sorted Array to Binary Search Tree- Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree

#### Code:

```
#include <vector>
using namespace std;
class Solution {
public:
  TreeNode* sortedArrayToBST(vector<int>& nums) {
     return helper(nums, 0, nums.size() - 1);
  }
private:
  TreeNode* helper(vector<int>& nums, int left, int right) {
     if (left > right) return nullptr;
     int mid = left + (right - left) / 2;
     TreeNode* root = new TreeNode(nums[mid]);
     root->left = helper(nums, left, mid - 1);
     root->right = helper(nums, mid + 1, right);
     return root;
  }
};
```

```
Accepted Runtime: 0 ms

• Case 1
• Case 2

Input

nums = [-10, -3, 0, 5, 9]

Output

[0, -10, 5, null, -3, null, 9]
```

**Problem 6.** Binary Tree Inorder Traversal-Given the root of a binary tree, return the inorder traversal of its nodes' values.

```
Code:
```

```
class Solution {
  public:
    vector<int> inorderTraversal(TreeNode* root) {
       vector<int> result;
       helper(root, result);
       return result;
    }

    void helper(TreeNode* root, vector<int>& result) {
       if (root != nullptr) {
         helper(root->left, result);
         result.push_back(root->val);
         helper(root->right, result);
       }
    }
}
```

```
Accepted Runtime: 0 ms

• Case 1
• Case 2
• Case 3
• Case 4

Input

root =
[1,null,2,3]

Output

[1,3,2]

Expected

[1,3,2]
```

**Problem 7.** Binary Zigzag Level Order Traversal- Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between)..

#### Code:

```
class Solution {
public:
  void solve(vector<vector<int>>& ans, TreeNode* temp, int level) {
     if (temp == NULL) return;
     if (ans.size() <= level) ans.push_back({});</pre>
     if (level % 2 == 0) ans[level].push_back(temp->val);
     else ans[level].insert(ans[level].begin(), temp->val);
     solve(ans, temp->left, level + 1);
     solve(ans, temp->right, level + 1);
  }
  vector<vector<int>>> zigzagLevelOrder(TreeNode* root) {
     vector<vector<int>> ans;
     solve(ans, root, 0);
     return ans;
   }
}
```

```
Accepted Runtime: 0 ms

• Case 1
• Case 2
• Case 3

Input

root =

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

Output

[[3],[20,9],[15,7]]
```

**Problem 8.** Construct Binary Tree from Inorder and Postorder Traversal- Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.

### **Code:**

```
class Solution {
public:
  TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
     unordered map<int, int> inorderIndexMap;
     for (int i = 0; i < inorder.size(); ++i) {
       inorderIndexMap[inorder[i]] = i;
     int postIndex = postorder.size() - 1;
     return constructTree(inorder, postorder, inorderIndexMap, postIndex, 0, inorder.size() - 1);
  TreeNode* constructTree(vector<int>& inorder, vector<int>& postorder, unordered_map<int, int>&
inorderIndexMap, int& postIndex, int inStart, int inEnd) {
     if (inStart > inEnd) return nullptr;
     int rootVal = postorder[postIndex--];
     TreeNode* root = new TreeNode(rootVal);
     int rootIndex = inorderIndexMap[rootVal];
     root->right = constructTree(inorder, postorder, inorderIndexMap, postIndex, rootIndex + 1, inEnd);
     root->left = constructTree(inorder, postorder, inorderIndexMap, postIndex, inStart, rootIndex - 1);
     return root;
};
```

```
Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

inorder = [9,3,15,20,7]

postorder = [9,15,7,20,3]

Output

[3,9,20,null,null,15,7]
```

**Problem 9.** Kth Smallest element in a BST- Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.

#### Code:

```
class Solution {
public:
  void solve(TreeNode* root, int &cnt, int &ans, int k){
     if(root == NULL) return;
     //left, root, right
     solve(root->left, cnt, ans, k);
     cnt++;
     if(cnt == k){
       ans = root->val;
       return;
     }
     solve(root->right, cnt, ans, k);
  int kthSmallest(TreeNode* root, int k) {
     int cnt = 0;
     int ans;
     solve(root, cnt, ans, k);
     return ans;
};
```

```
Accepted Runtime: 0 ms

• Case 1
• Case 2

Input

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

k = 1

Output

1
```

**Problem 10.** Populating Next Right Pointers in Each Node-You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

### **Code:**

```
class Solution {
public:
  Node* connect(Node* root) {
     if(root==nullptr) return { };
     queue<Node*>q;
     q.push(root);
     while(!q.empty()){
       int n = q.size();
        for(int i=0;i< n;i++)
          Node* t = q.front();
          q.pop();
          if(i!=n-1){
             t->next=q.front();
          if(t->left) q.push(t->left);
          if(t->right) q.push(t->right);
        }
     }
     return root;
  }
};
```

```
Accepted Runtime: 0 ms

• Case 1
• Case 2

Input

root = [1,2,3,4,5,6,7]

Output

[1,#,2,3,#,4,5,6,7,#]
```