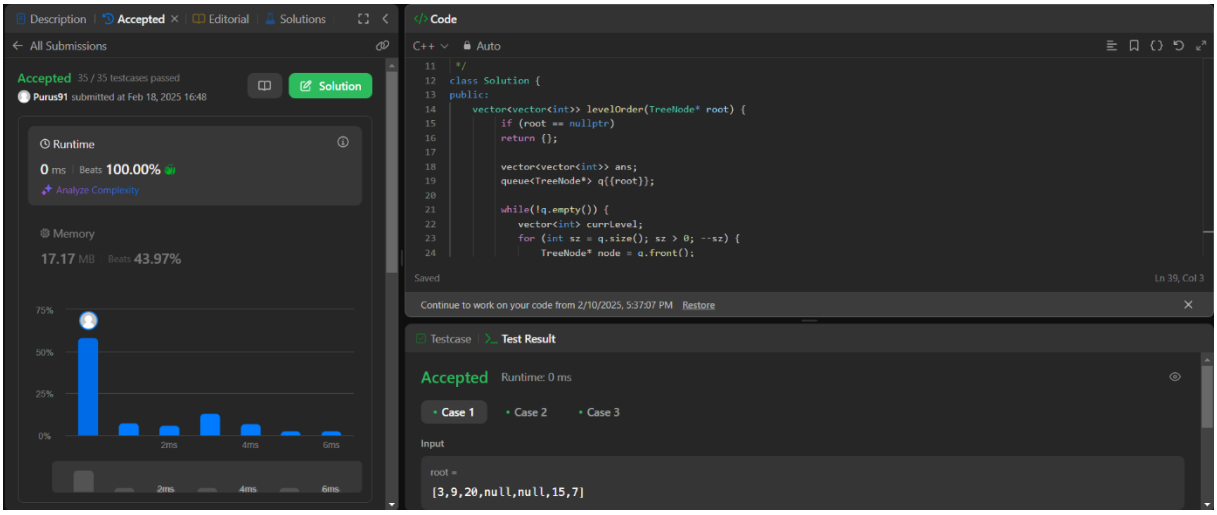
Name: Purushottam Kumar
UID: 22BCS10794
Section: 22BCS_FL_IOT_602-B
Subject: AP LAB

Experiment - 04

Question 1: Binary Tree Level Order Traversal

Code: class Solution {

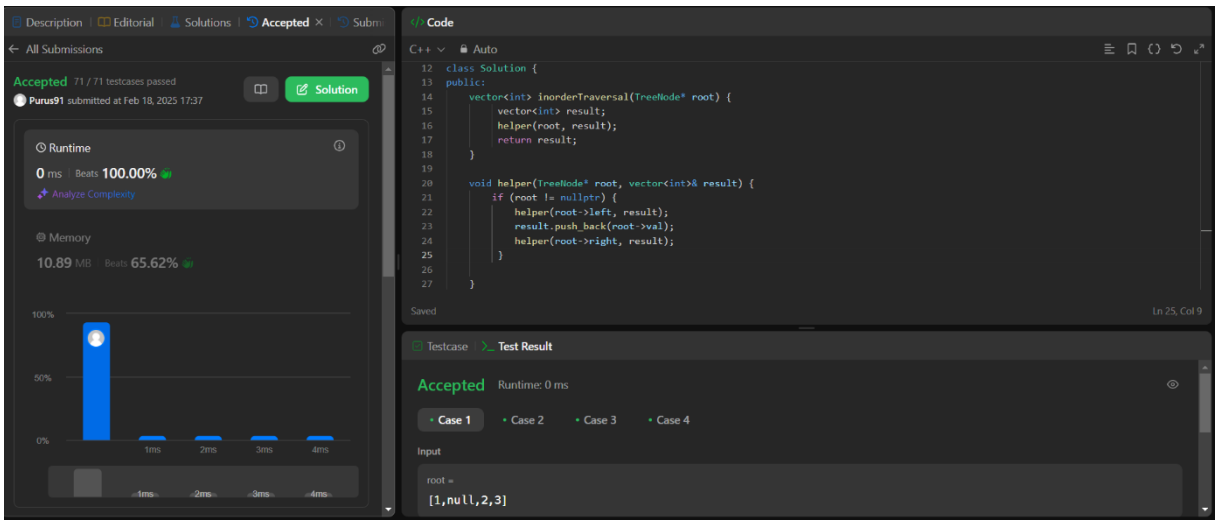
```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        if (root == nullptr)
        return {};

        vector<vector<int>> ans;
        queue<TreeNode*> q{{root}};
                q.push(node->right);

          }
                ans.push_back(currLevel);

        }
         return ans;

    }
};
```



Question 2 :- Binary Tree Inorder Traversal

Code:-

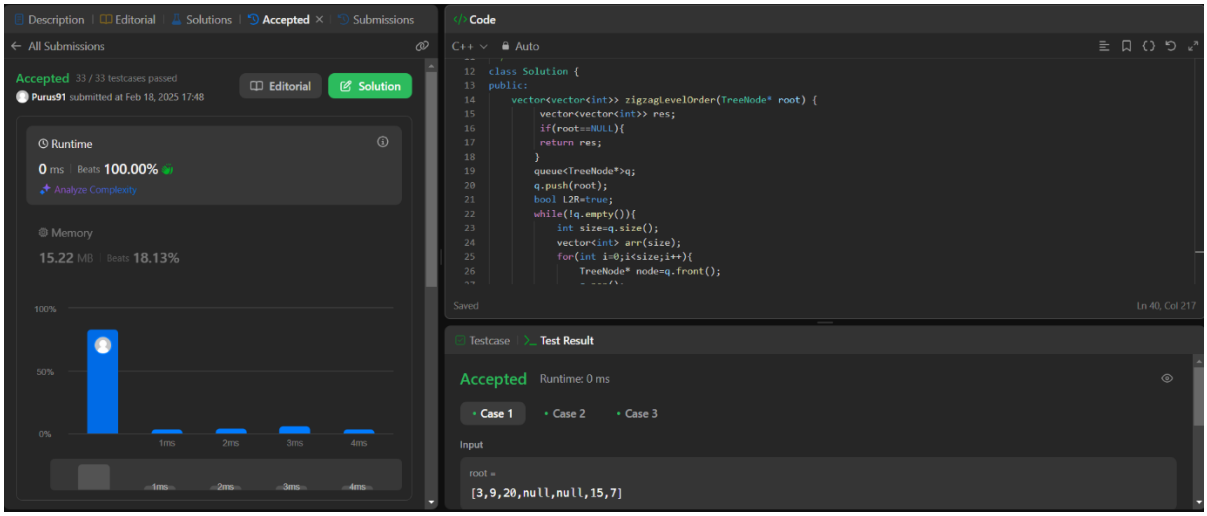
```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        helper(root, result);
        return result;
    }

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

    }
};
```



Question 3 :- Binary Tree Zigzag Level Order

Code :-

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if(root==NULL){
            return res;
        }
        queue<TreeNode*>q;
        q.push(root);
        bool L2R=true;
        while(!q.empty()){
            int size=q.size();
            vector<int> arr(size);
            for(int i=0;i<size;i++){
                TreeNode* node=q.front();
                q.pop();
                int ind=(L2R)?i:(size-i-1);
                arr[ind]=node->val;
                if(node->left){
                    q.push(node->left);
                }
                if(node->right){
                    q.push(node->right);
                }
            }
            L2R=!L2R;
            res.push_back(arr);
        }
        return res;
    }
}
```

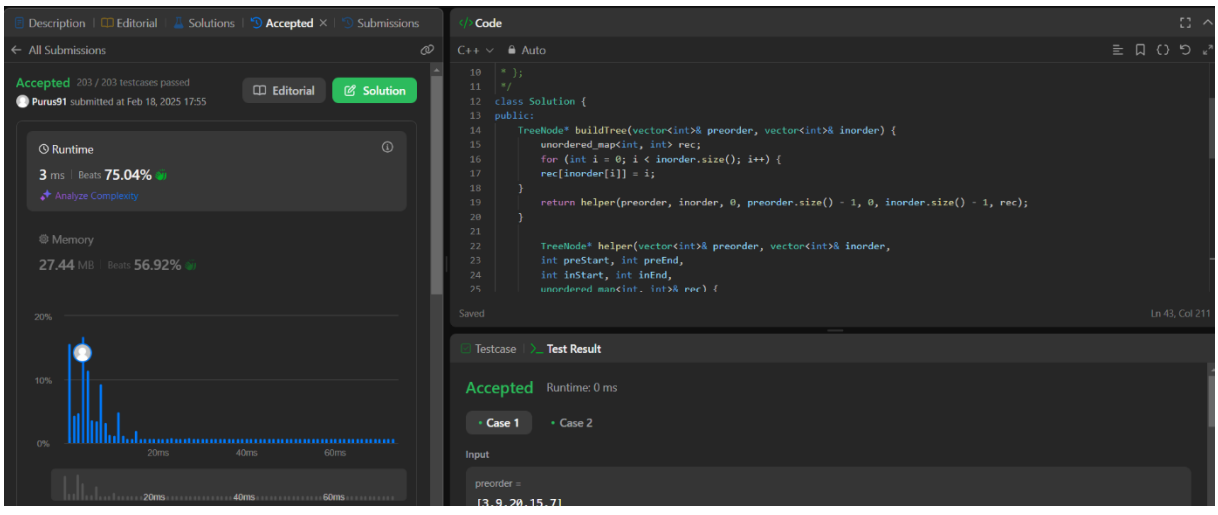


Question 4:- .Binary Tree Level Order Traversal

Code:-

```
#include <bits/stdc++.h>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        unordered_map<int, int> rec;
        for (int i = 0; i < inorder.size(); i++) {
            rec[inorder[i]] = i;
        }
        return helper(preorder, inorder, 0, preorder.size() - 1, 0, inorder.size() - 1,
rec);
    }
```

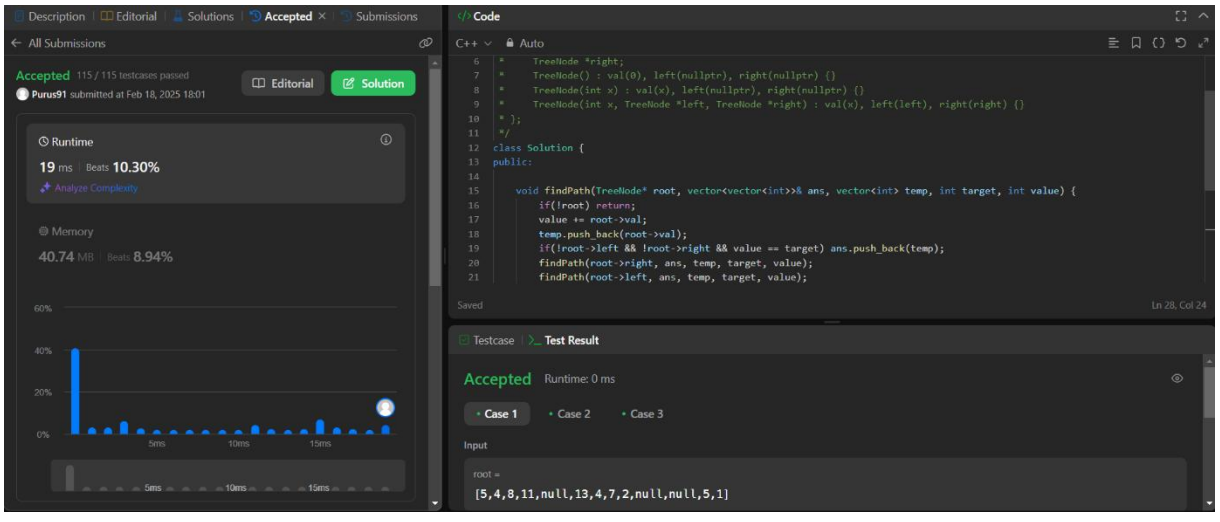
```
    TreeNode* helper(vector<int>& preorder, vector<int>& inorder,
                     int preStart, int preEnd,
                     int inStart, int inEnd,
                     unordered_map<int, int>& rec) {
        if (preStart > preEnd || inStart > inEnd) return nullptr;

        int val = preorder[preStart];
        TreeNode* root = new TreeNode(val);
        int idx = rec[val];
        int leftSubtreeSize = idx - inStart;

        root->left = helper(preorder, inorder,
                            preStart + 1, preStart + leftSubtreeSize,
                            inStart, idx - 1,
                            rec);

        root->right = helper(preorder, inorder,
                             preStart + leftSubtreeSize + 1, preEnd,
                             idx + 1, inEnd,
                             rec);

        return root;
    }
};
```



Question 5 :- Path Sum ||

Code:-

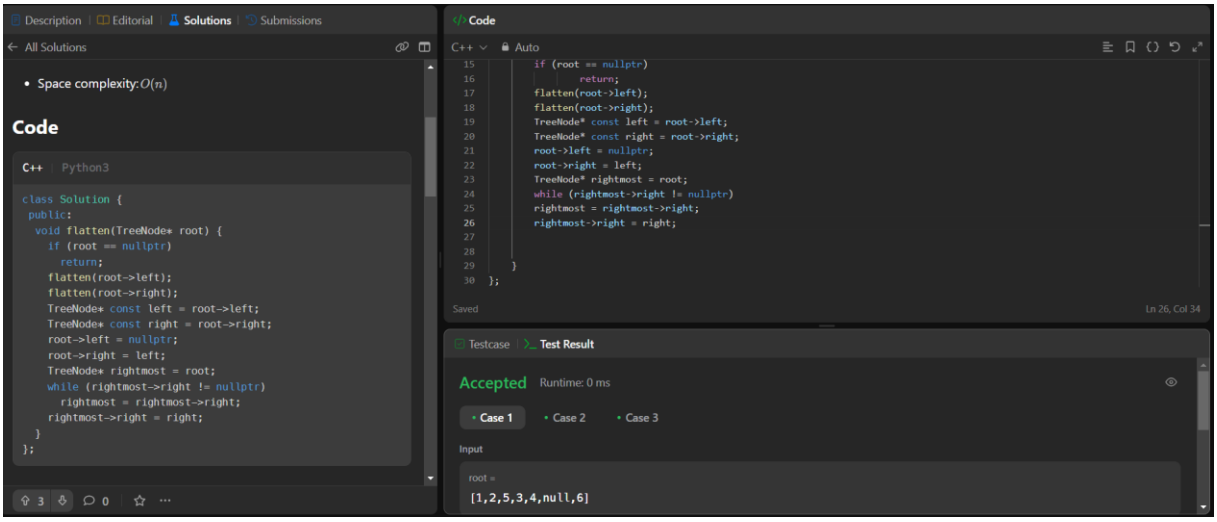
```
class Solution {
public:
    void findPath(TreeNode* root, vector<vector<int>>& ans, vector<int> temp, int target, int value) {
        if(!root) return;
        value += root->val;
        temp.push_back(root->val);
        if(!root->left && !root->right && value == target) ans.push_back(temp);
        findPath(root->right, ans, temp, target, value);
        findPath(root->left, ans, temp, target, value);
    }

    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        vector<vector<int>> ans;
        vector<int> temp;
        findPath(root, ans, temp, targetSum, 0);
        return ans;
    }
};
```



Question 6 :- Flatten Binary Tree to Linked List

Code:-

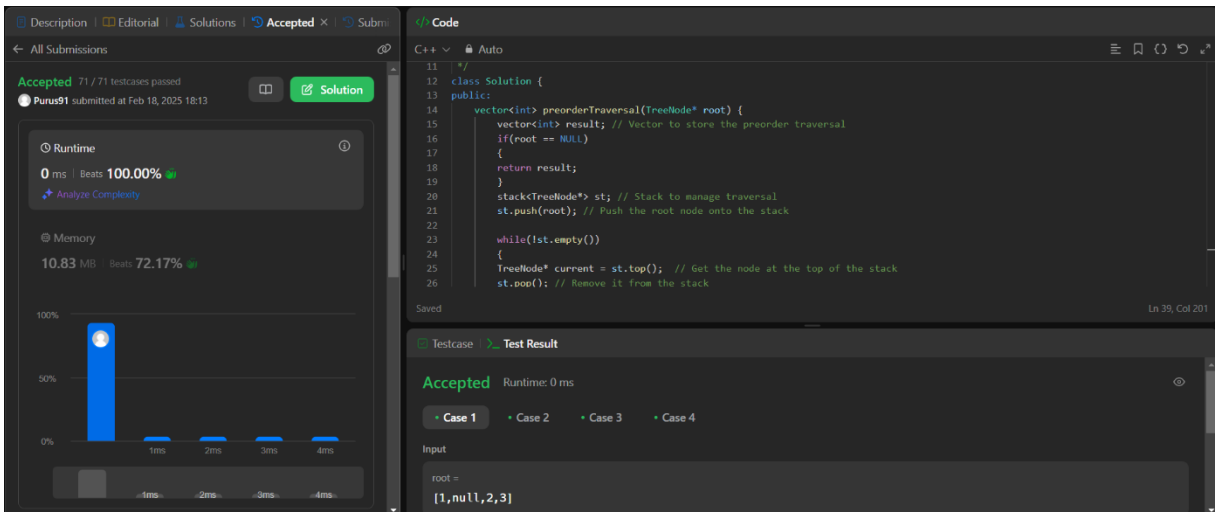
```
class Solution {
 public:
  void flatten(TreeNode* root) {
    if (root == nullptr)
      return;
    flatten(root->left);
    flatten(root->right);
    TreeNode* const left = root->left;
    TreeNode* const right = root->right;
    root->left = nullptr;
    root->right = left;
    TreeNode* rightmost = root;
    while (rightmost->right != nullptr)
      rightmost = rightmost->right;
    rightmost->right = right;
  }
};
```



Question 7 :- Binary tree Preorder Traversal

Code:-

```
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> result; // Vector to store the preorder traversal
        if(root == NULL)
        {
            return result;
        }
        stack<TreeNode*> st; // Stack to manage traversal
        st.push(root); // Push the root node onto the stack

        while(!st.empty())
        {
            TreeNode* current = st.top();  // Get the node at the top of the stack
            st.pop(); // Remove it from the stack
            result.push_back(current->val); // Add the node's value to the result
        //will first insert right node so at the time of insertion the left will pop out
            if(current->right)
            {
                st.push(current->right);
            }
            if(current->left)
            {
                st.push(current->left);
            }
        }
        return result;
    }
};
```

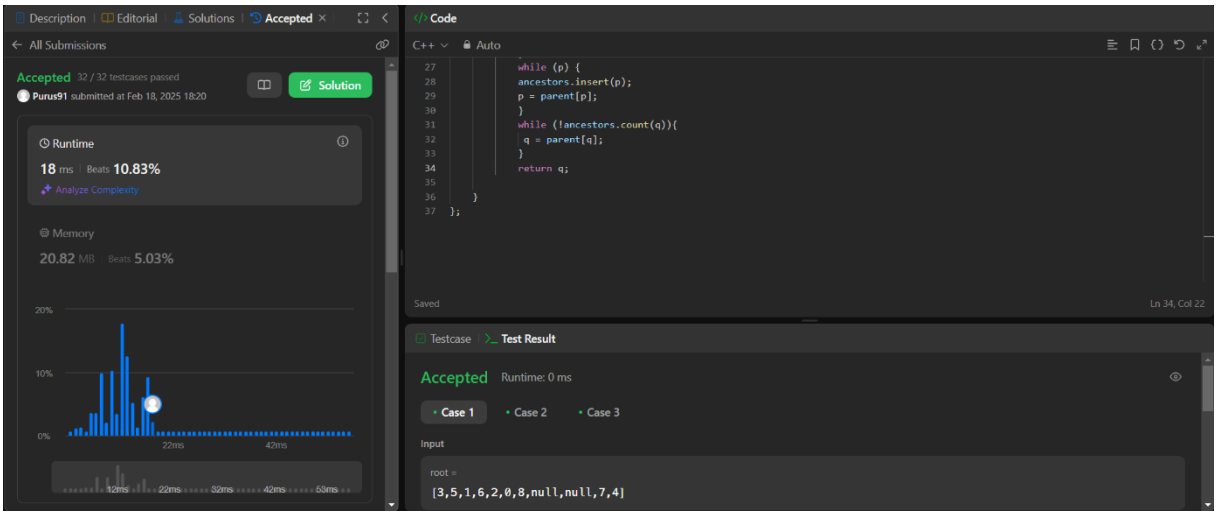


Question 8 :- Lowest Common Ancestor of a Binary Tree

Code:-

```
class Solution {
 public:
  TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode*
p, TreeNode* q) {
    queue<TreeNode*> q_{{root}};
    unordered_map<TreeNode*, TreeNode*> parent{{root,
nullptr}};
    unordered_set<TreeNode*> ancestors;
    while (!parent.count(p) || !parent.count(q)) {
      root = q_.front(), q_.pop();
      if (root->left) {
        parent[root->left] = root;
        q_.push(root->left);
      }
      if (root->right) {
        parent[root->right] = root;
        q_.push(root->right);
      }
    }
    while (p) {
      ancestors.insert(p);
      p = parent[p];
    }
```

```
    while (!ancestors.count(q)){
      q = parent[q];
    }
    return q;
  }
};
```

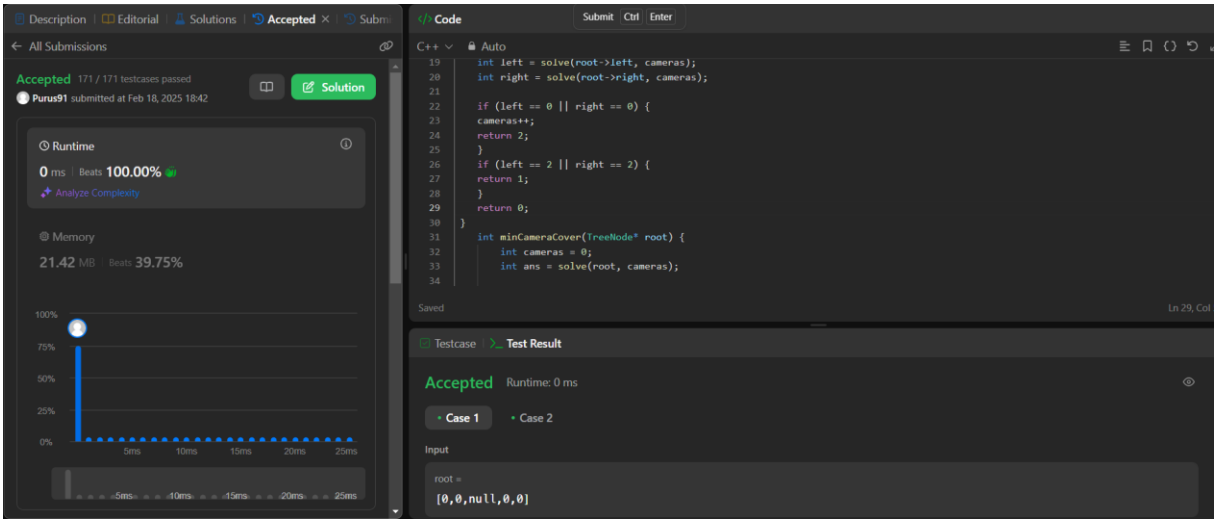


Question 9 :- Binary Tree Cameras

Code:-

```
class Solution {
public:
    int solve(TreeNode* root, int &cameras) {
        if (!root) {
            return 1;
        }

        int left = solve(root->left, cameras);
        int right = solve(root->right, cameras);

        if (left == 0 || right == 0) {
            cameras++;
            return 2;
        }
        if (left == 2 || right == 2) {
            return 1;
        }
```

```
        return 0;
    }

    int minCameraCover(TreeNode* root) {
        int cameras = 0;
        int ans = solve(root, cameras);

        if (ans == 0) {
            cameras++;
        }

        return cameras;
    }
};
```



Question 10 :- Vertical Order Traversal of a Binary Tree

Code:-

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int,map<int,multiset<int>>> nodes;
        queue<pair<TreeNode*,pair<int,int>>> q;
        q.push({root,{0,0}});
        while(!q.empty()){
            auto p=q.front();
```

```
            q.pop();
            TreeNode* node=p.first;
            int x=p.second.first,y=p.second.second;
            nodes[x][y].insert(node->val);
            if(node->left){
                q.push({node->left,{x-1,y+1}});
            }
            if(node->right){
                q.push({node->right,{x+1,y+1}});
            }
        }
        vector<vector<int>> ans;
        for(auto p:nodes){
            vector<int>col;
            for(auto q:p.second){
                for(auto it : q.second){
                    col.push_back(it);
                }
            }
            ans.push_back(col);
        }
        return ans;
    }
};
```