Assignment 3

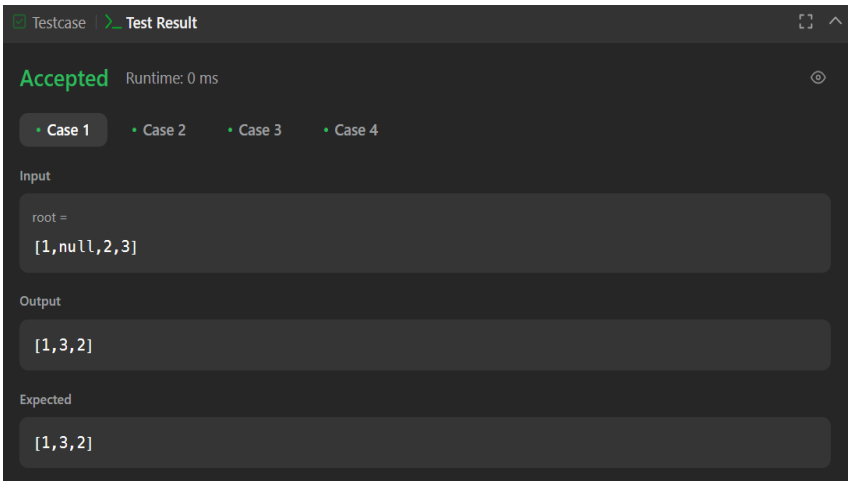
Student Name: Nishant Kumar
UID: 22BCS15009
Branch: CSE
Section/Group: 22BCS_FL_IOT-602 A
Semester: 6th
Date of Performance: 14/02/2025
Subject Name: Advanced Programming Lab - 2
Subject Code: 22CSP-351

Problem 94. Binary Tree Inorder Traversal

- **Implementation/Code:**

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        stack<TreeNode*> st;
        TreeNode* curr = root;

        while (curr || !st.empty()) {
            while (curr) {
                st.push(curr);
                curr = curr->left;
            }
            curr = st.top(); st.pop();
            result.push_back(curr->val);
            curr = curr->right;
        }
        return result;
    }
};
```

- **Output:**

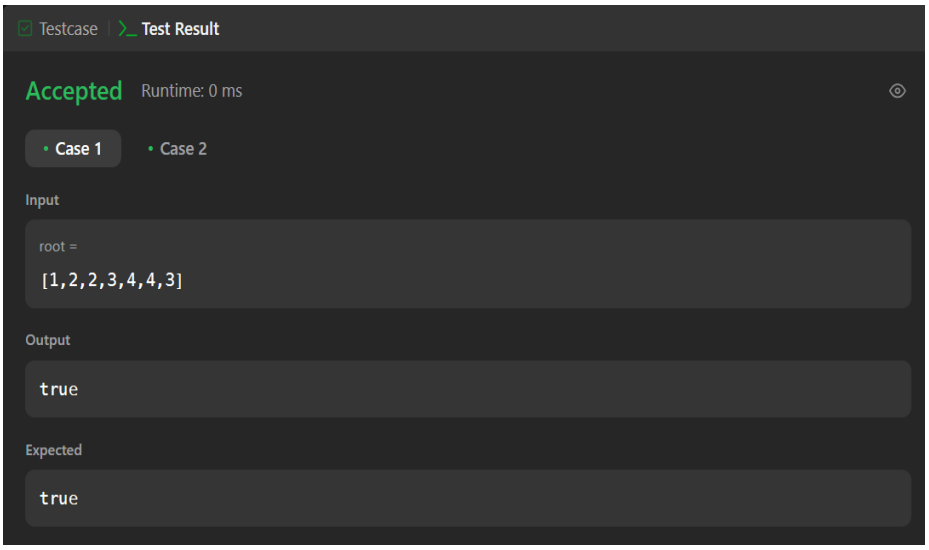


Problem 101. Symmetric Tree

- **Implementation/Code:**

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

        return isMirror(left->left, right->right) && isMirror(left->right, right->left);
    }

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

- **Output:**

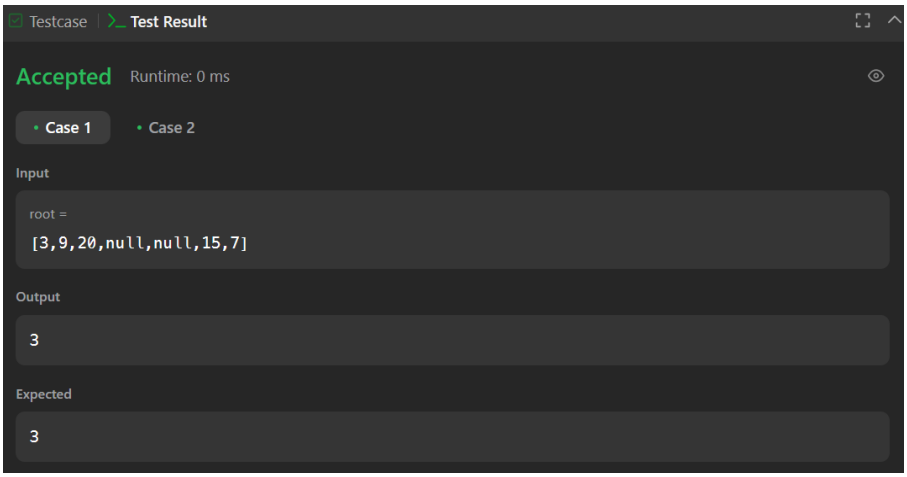


Problem 104. Maximum Depth of Binary Tree

- **Implementation/Code:**

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

- **Output:**

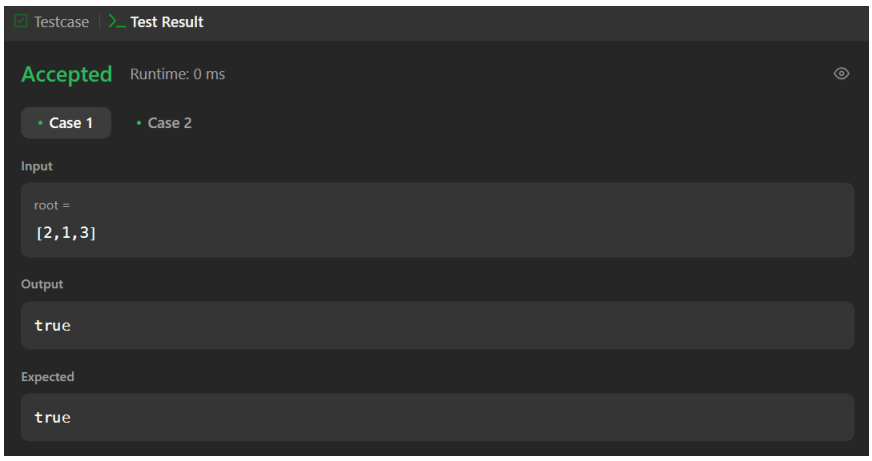


Problem 98. Validate Binary Search Tree

- **Implementation/Code:**

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return validate(root, LONG_MIN, LONG_MAX);
    }
private:
    bool validate(TreeNode* node, long minVal, long maxVal) {
        if (!node) return true;

        if (node->val <= minVal || node->val >= maxVal) return false;

        return validate(node->left, minVal, node->val) &&
               validate(node->right, node->val, maxVal);
    }
};
```

- **Output:**

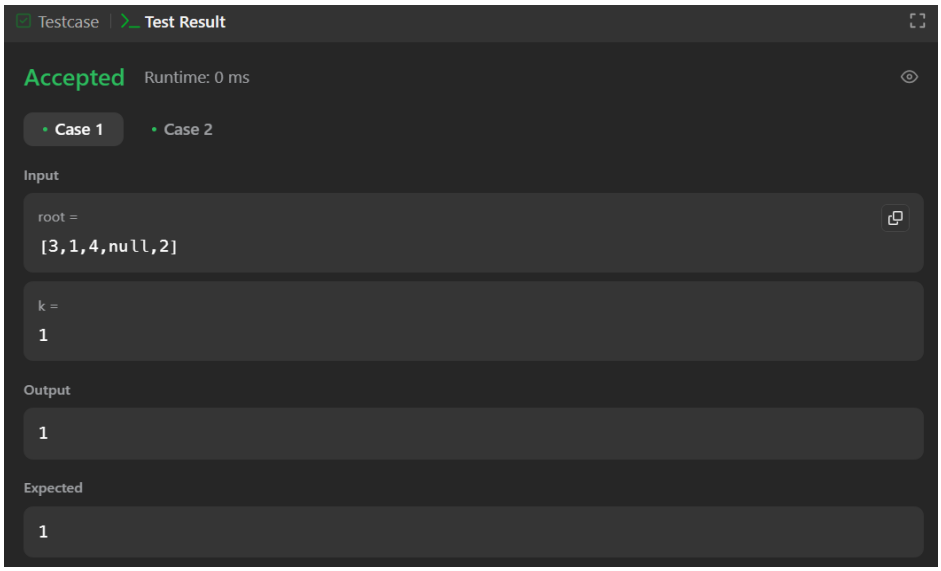


Problem 230. Kth Smallest Element in a BST

- **Implementation/Code:**

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> st;
        TreeNode* curr = root;

        while (curr || !st.empty()) {
            while (curr) {
                st.push(curr);
                curr = curr->left;
            }
            curr = st.top(); st.pop();
            k--;
            if (k == 0) return curr->val;
            curr = curr->right;
        }
        return -1;
    }
};
```

- **Output:**



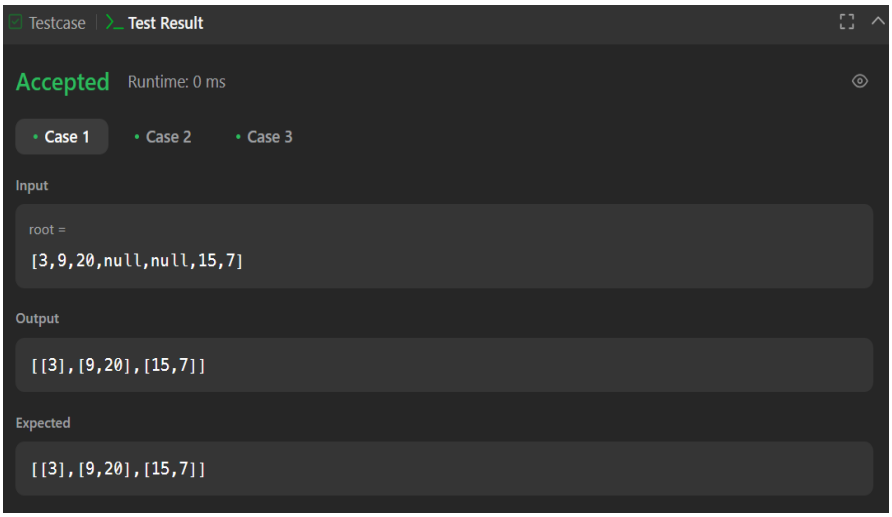
Problem 102. Binary Tree Level Order Traversal

- **Implementation/Code:**

```
class Solution {
public:
    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> level;

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

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level);
        }
        return result;
    }
};
```

- **Output:**



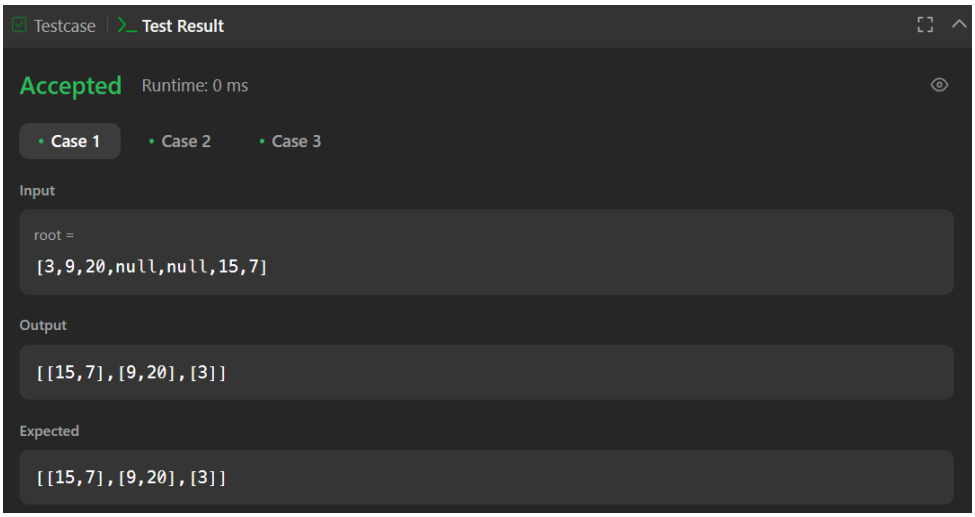
Problem 107. Binary Tree Level Order Traversal II

- **Implementation/Code:**

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(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> level;

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

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.insert(result.begin(), level);
        }
        return result;
    }
};
```

- **Output:**



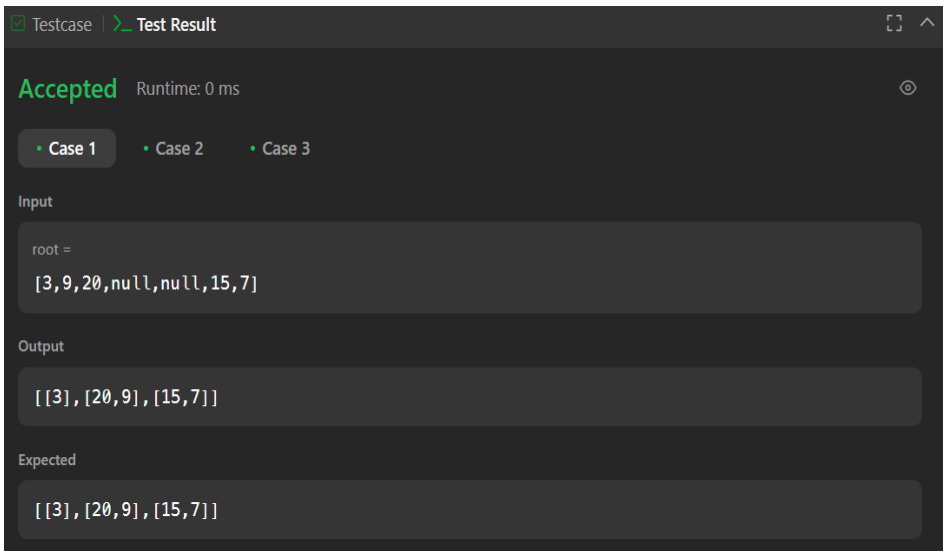
Problem 103. Binary Tree Zigzag Level Order Traversal

- **Implementation/Code:**

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (!root) return result;

        queue<TreeNode*> q;
```

```
        q.push(root);
        bool leftToRight = true;

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

            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();

                int index = leftToRight ? i : (size - 1 - i);
                level[index] = node->val;

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level);
            leftToRight = !leftToRight;
        }
        return result;
    }
};
```

- **Output:**



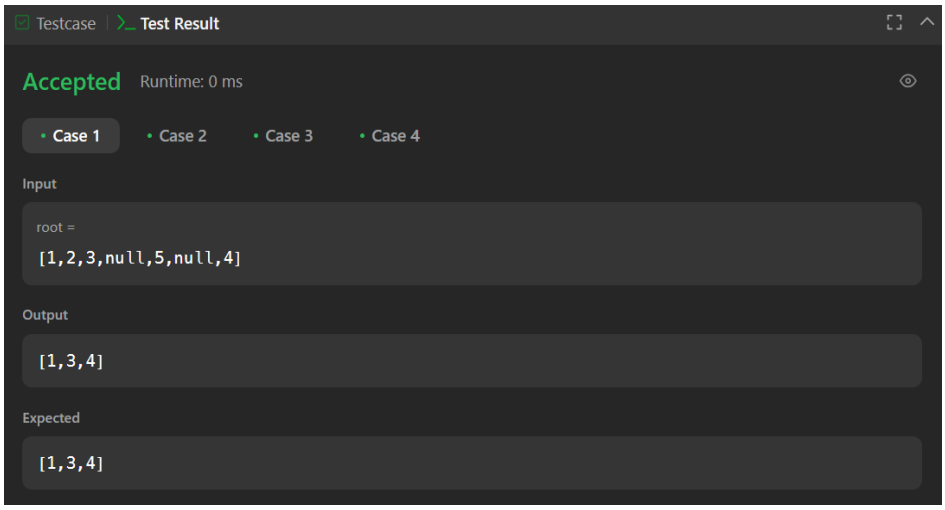
Problem 199. Binary Tree Right Side View

- **Implementation/Code:**

```
class Solution {
public:
    void dfs(TreeNode* node, int level, vector<int>& result) {
        if (!node) return;
```

```
        if (result.size() == level)
            result.push_back(node->val);
        dfs(node->right, level + 1, result);
        dfs(node->left, level + 1, result);
    }
    vector<int> rightSideView(TreeNode* root) {
        vector<int> result;
        dfs(root, 0, result);
        return result;
    }
};
```

- **Output:**



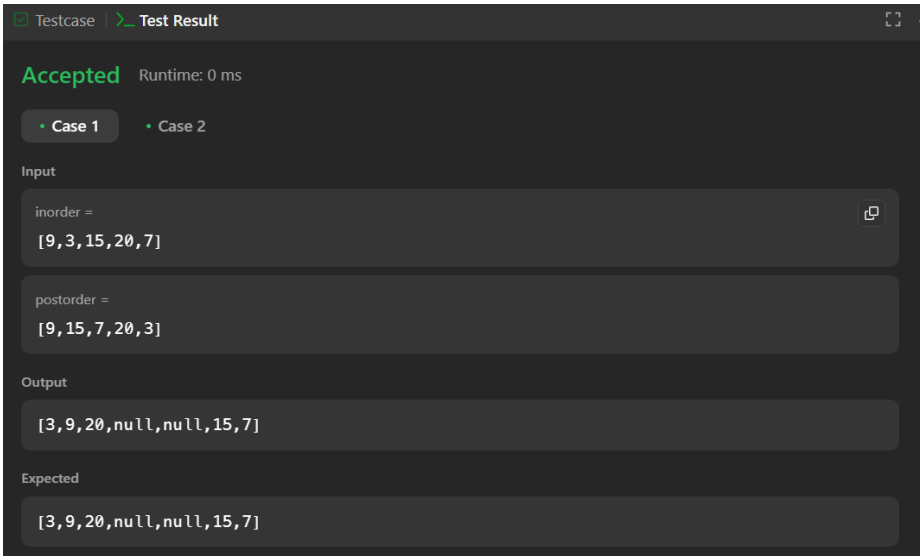
Problem 106. Construct Binary Tree from Inorder and Postorder Traversal

- **Implementation/Code:**

```
class Solution {
public:
    unordered_map<int, int> inorderMap;
    int postIndex;

    TreeNode* build(vector<int>& inorder, vector<int>& postorder, int left, int right) {
        if (left > right) return nullptr;

        int rootVal = postorder[postIndex--];
        TreeNode* root = new TreeNode(rootVal);

        int inorderIndex = inorderMap[rootVal];


        root->right = build(inorder, postorder, inorderIndex + 1, right);
        root->left = build(inorder, postorder, left, inorderIndex - 1);
        return root;
```

```
    }
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        postIndex = postorder.size() - 1;

        for (int i = 0; i < inorder.size(); i++) {
            inorderMap[inorder[i]] = i;
        }
        return build(inorder, postorder, 0, inorder.size() - 1);
    }
};
```

- **Output:**



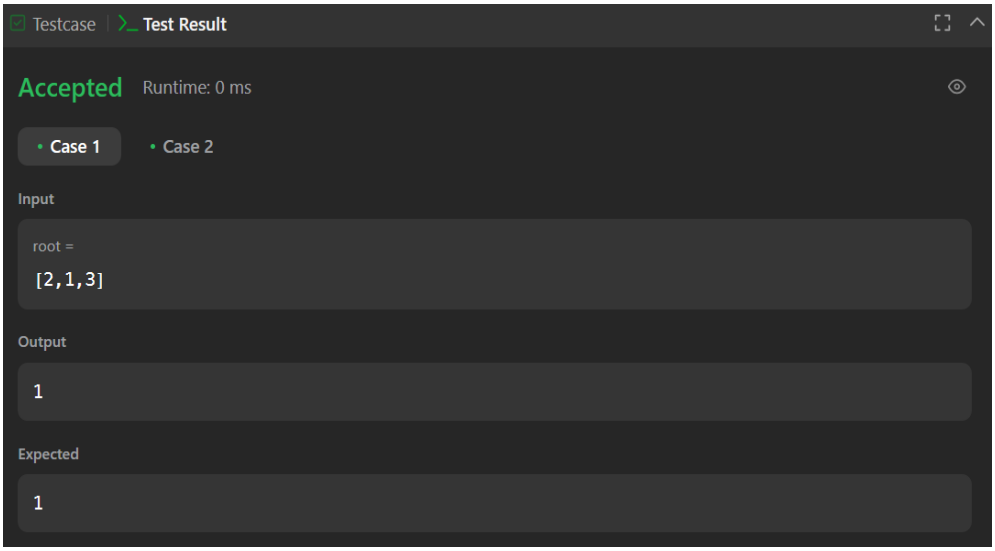
Problem 513. Find Bottom Left Tree View

- **Implementation/Code:**

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue<TreeNode*> q;
        q.push(root);
        int bottomLeftValue = root->val;

        while (!q.empty()) {
            int levelSize = q.size();
            bottomLeftValue = q.front()->val;

            for (int i = 0; i < levelSize; i++) {
                TreeNode* node = q.front();
                q.pop();

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
```

```
            }
        }
        return bottomLeftValue;
    }
};
```

- **Output:**

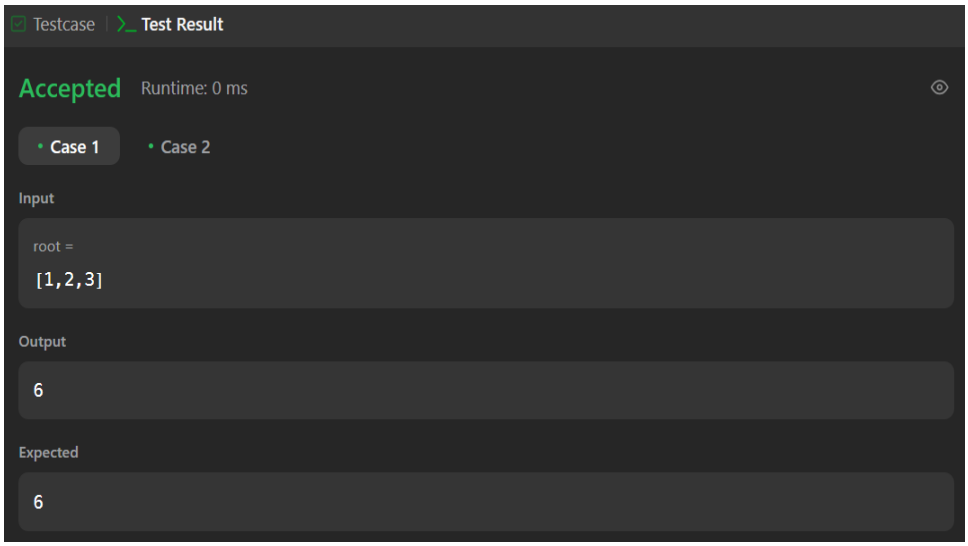


Problem 124. Binary Tree Maximum Path Sum

- **Implementation/Code:**

```
class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int ans = -1001;
        function<int(TreeNode*)> dfs = [&](TreeNode* root) {
            if (!root) {
                return 0;
            }
            int left = max(0, dfs(root->left));
            int right = max(0, dfs(root->right));
            ans = max(ans, left + right + root->val);
            return root->val + max(left, right);
        };
        dfs(root);
        return ans;
    }
};
```

- **Output:**



Problem 987. Vertical Order Traversal of a Binary Tree

- **Implementation/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 [node, pos] = q.front(); q.pop();
            int col = pos.first, row = pos.second;

            nodes[col][row].insert(node->val);

            if (node->left) q.push({node->left, {col - 1, row + 1}});
            if (node->right) q.push({node->right, {col + 1, row + 1}});
        }
        vector<vector<int>> result;
        for (auto& [col, rows] : nodes) {
            vector<int> column_values;
            for (auto& [row, values] : rows) {
                column_values.insert(column_values.end(), values.begin(), values.end());
            }
            result.push_back(column_values);
```

```
            }
            return result;
        }
    };
```

- **Output:**