



Experiment 3

Student Name: Nitesh Bamber

Branch: BE-CSE

Semester: 6th

Subject Name: Advanced Programming - II

UID: 22BCS13791

Section/Group: 602/A

Date of Performance: 14-02-25

Subject Code: 22CSP-351

1. Aim:

To solve the following problems on Leetcode, with the goal of optimizing solutions in terms of time complexity and space efficiency:

1) Binary Tree Inorder Traversal (94)

Aim: To perform inorder traversal (left, root, right) of a binary tree and return the sequence of visited nodes in a list. The traversal should be implemented using recursion or an iterative approach with a stack.

2) Symmetric Tree (101)

Aim: To determine whether a given binary tree is symmetric around its center, meaning it is a mirror reflection of itself when split from the root. The solution should explore both recursive and iterative approaches using queues.

3) Maximum Depth of Binary Tree (104)

Aim: To compute the maximum depth (height) of a binary tree, which is the longest path from the root to a leaf node. The depth is measured by the number of nodes along this path.

4) Validate Binary Search Tree (98)

Aim: To determine whether a given binary tree is a valid Binary Search Tree (BST), ensuring that for every node, the left subtree contains values less than the node, and the right subtree contains values greater than the node. The solution should consider edge cases, such as duplicate values and large depth constraints.

5) Kth Smallest Element in a BST (230)

Aim: To find the kth smallest element in a given Binary Search Tree (BST). The problem leverages the BST property, where an inorder traversal yields elements in sorted order, allowing an efficient solution.

6) Binary Tree Level Order Traversal (102)

Aim: To traverse a binary tree level by level (Breadth-First Search) and return a list of node values grouped by each level from top to bottom. The traversal should be implemented using a queue-based approach.

7) Binary Tree Level Order Traversal II (107)

Aim: To perform level order traversal on a binary tree and return the values of nodes at each level in bottom-up order, meaning the lowest level appears first in the output.

8) Binary Tree Zigzag Level Order Traversal (103)

Aim: To traverse a binary tree in a zigzag manner, where nodes at each level alternate between left-to-right and right-to-left traversal order. The traversal should be implemented using a queue or deque to facilitate the directional switching.

9) Binary Tree Right Side View (199)

Aim: To return a list of node values that are visible when looking at a binary tree from the right side, meaning only the rightmost node at each level should be included in the output.

10) Construct Binary Tree from Inorder and Postorder Traversal (106)

Aim: To construct a binary tree given its inorder and postorder traversal sequences. The solution should reconstruct the tree by identifying the root from the postorder list and partitioning the inorder list accordingly.

11) Find Bottom Left Tree Value (513)

Aim: To find the leftmost node value in the last row (deepest level) of a given binary tree. The solution should ensure the value returned belongs to the lowest possible depth and is the leftmost node at that depth.

12) Binary Tree Maximum Path Sum (124)

Aim: To find the maximum path sum in a binary tree, where a path is defined as any sequence of connected nodes, and the path sum is the sum of all node values in that sequence. The solution should account for paths that may or may not pass through the root.

13) Vertical Order Traversal of a Binary Tree (987)

Aim: To perform a vertical order traversal of a binary tree, where nodes are grouped by their vertical position and sorted based on horizontal position and value. The traversal follows a top-down, left-to-right approach for sorting nodes within the same vertical level.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

2. Objective:

The objective of these problems is to develop a deep understanding of various tree traversal techniques, binary tree properties, and Binary Search Tree (BST) operations. These problems cover fundamental and advanced concepts, including:

- Tree Traversal: Inorder, level order, zigzag, vertical, and postorder traversals.
- Tree Properties: Checking symmetry, depth calculation, and validation of BSTs.
- Tree Construction: Reconstructing a tree from traversal sequences.
- Path-Based Computations: Finding maximum path sums, right-side views, and bottom-left values.
- Search and Retrieval in BSTs: Finding the kth smallest element efficiently.

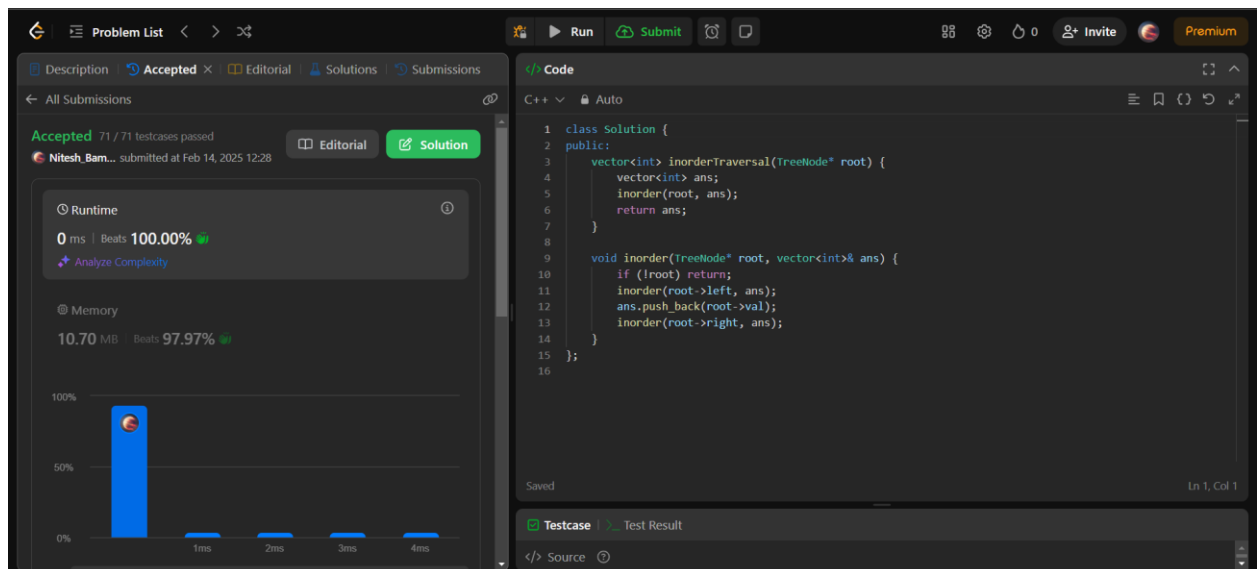
3. Implementation/Code:

Problem No. 94: Binary Tree Inorder Traversal

Code:

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

    void inorder(TreeNode* root, vector<int>& ans) {
        if (!root) return;
        inorder(root->left, ans);
        ans.push_back(root->val);
        inorder(root->right, ans);
    }
};
```

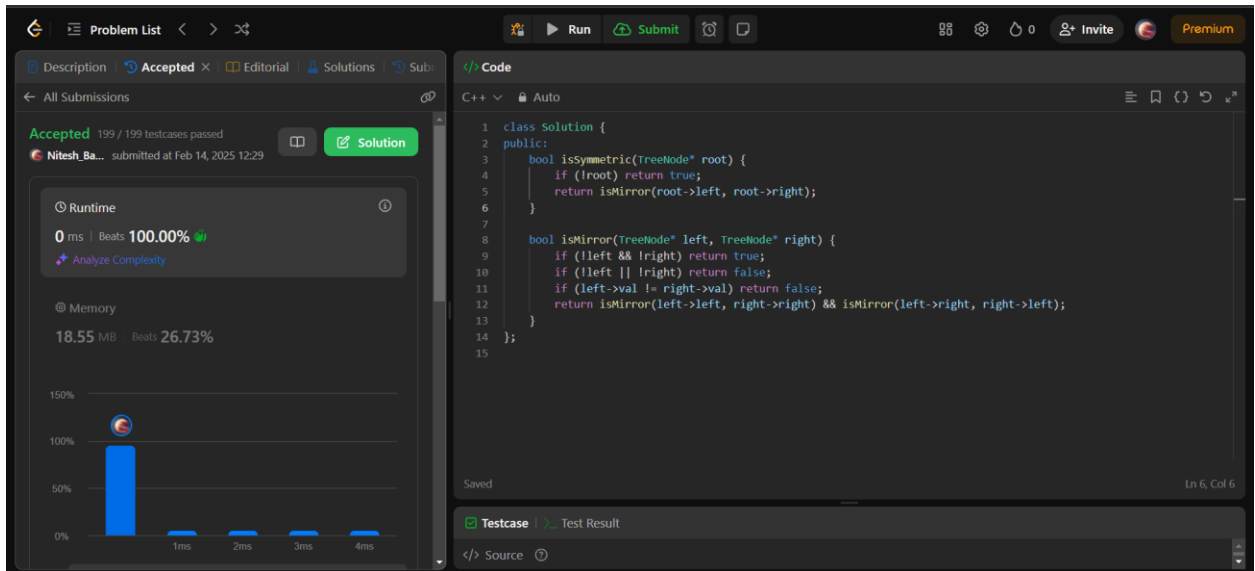


Problem No. 101: Symmetric Tree

Code:

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

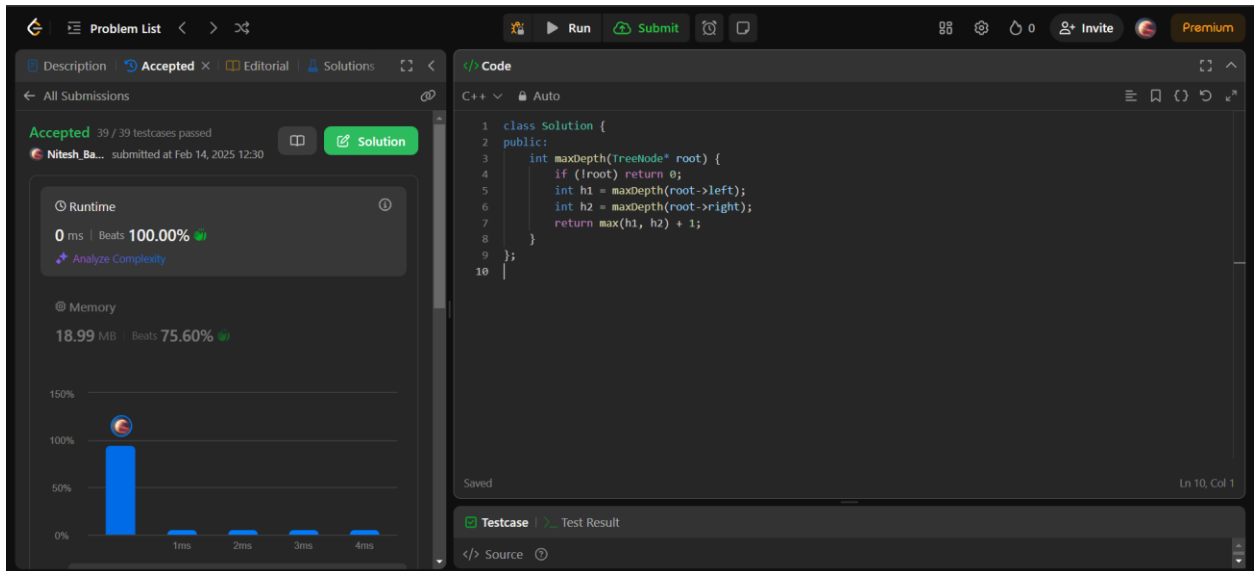
    bool isMirror(TreeNode* left, TreeNode* right) {
        if (!left && !right) return true;
        if (!left || !right) return false;
        if (left->val != right->val) return false;
        return isMirror(left->left, right->right) && isMirror(left->right, right->left);
    }
};
```



Problem No. 104: Maximum Depth of Binary Tree

Code:

```
class Solution {  
public:  
    int maxDepth(TreeNode* root) {  
        if (!root) return 0;  
        int h1 = maxDepth(root->left);  
        int h2 = maxDepth(root->right);  
        return max(h1, h2) + 1;  
    }  
};
```



The screenshot displays a coding platform interface with a dark theme. On the left, the 'Problem List' tab is active, showing 'Accepted' status for 39/39 testcases. The submission is by 'Nitesh_Ba...' and was submitted on Feb 14, 2025 at 12:30. The 'Runtime' section shows 0 ms and 100.00% beats. The 'Memory' section shows 18.99 MB and 75.60% beats. A bar chart below shows the runtime distribution. On the right, the 'Code' editor shows the C++ solution for the problem. The code is as follows:

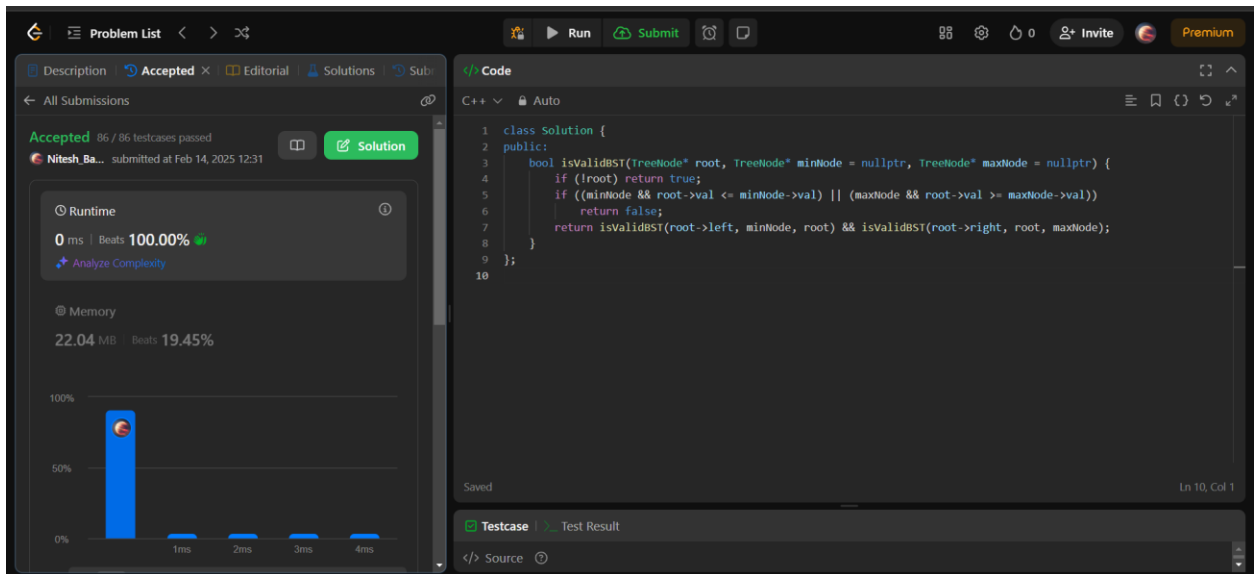
```
1 class Solution {  
2 public:  
3     int maxDepth(TreeNode* root) {  
4         if (!root) return 0;  
5         int h1 = maxDepth(root->left);  
6         int h2 = maxDepth(root->right);  
7         return max(h1, h2) + 1;  
8     }  
9 }  
10
```

The bottom of the interface shows a 'Testcase' tab and a 'Test Result' section.

Problem No. 98: Validate Binary Search Tree

Code:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem No. 230: Kth Smallest Element in a BST

Code:

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> st;
        while (true) {
            while (root) {
                st.push(root);
                root = root->left;
            }
            root = st.top();
            st.pop();
            if (--k == 0) return root->val;
            root = root->right;
        }
    }
};
```

The screenshot displays a code editor interface with a dark theme. The left sidebar shows the 'Problem List' and 'Accepted' status for the problem. The main editor area contains the C++ code for the solution. The right sidebar shows the 'Runtime' and 'Memory' performance metrics, along with a bar chart indicating the execution time relative to other solutions.

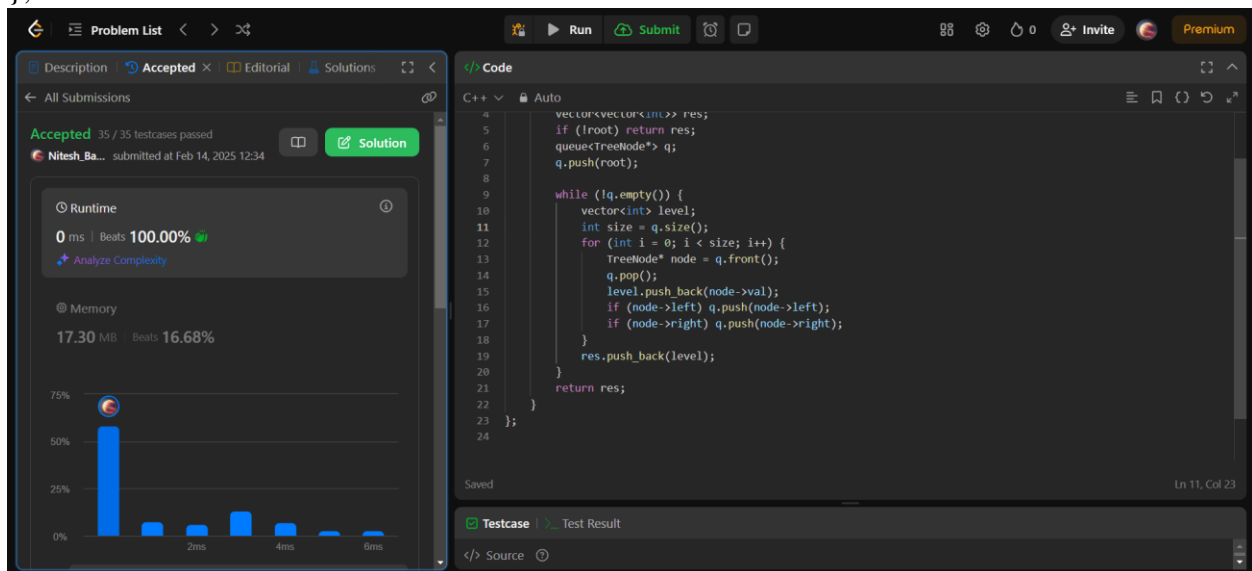
Runtime: 0 ms | Beats: 100.00%
Memory: 24.32 MB | Beats: 67.69%

```
1 class Solution {
2 public:
3     int kthSmallest(TreeNode* root, int k) {
4         stack<TreeNode*> st;
5         while (true) {
6             while (root) {
7                 st.push(root);
8                 root = root->left;
9             }
10            root = st.top();
11            st.pop();
12            if (--k == 0) return root->val;
13            root = root->right;
14        }
15    }
16 };
17
```


Problem No. 102: Binary Tree Level Order Traversal

Code:

```
class Solution {  
public:  
    vector<vector<int>> levelOrder(TreeNode* root) {  
        vector<vector<int>> res;  
        if (!root) return res;  
        queue<TreeNode*> q;  
        q.push(root);  
  
        while (!q.empty()) {  
            vector<int> level;  
            int size = q.size();  
            for (int i = 0; i < size; 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);  
            }  
            res.push_back(level);  
        }  
        return res;  
    }  
};
```



The screenshot displays a code editor interface for a problem titled "Problem No. 102: Binary Tree Level Order Traversal". The code is written in C++ and implements a BFS approach using a queue. The left sidebar shows the problem description, accepted status, and runtime/memory usage. The right sidebar shows the code editor with the solution code.

Runtime: 0 ms | Beats: 100.00%

Memory: 17.30 MB | Beats: 16.68%

Code:

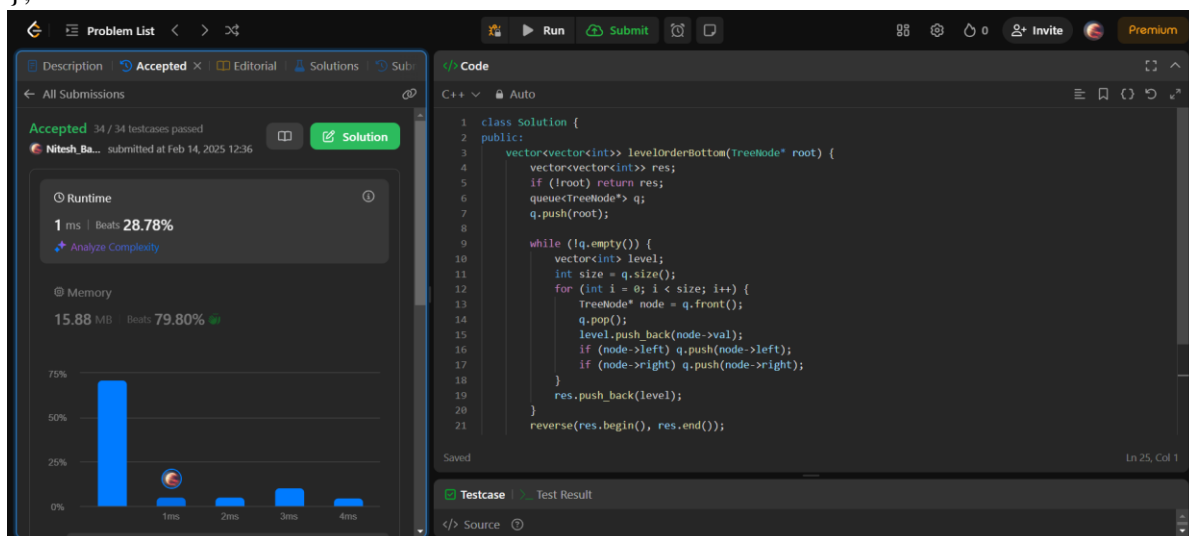
```
vector<vector<int>> res;  
if (!root) return res;  
queue<TreeNode*> q;  
q.push(root);  
  
while (!q.empty()) {  
    vector<int> level;  
    int size = q.size();  
    for (int i = 0; i < size; i++) {  
        TreeNode* node = q.front();  
        q.pop();  
        level.push_back(node->val);  
        if (node->left) q.push(node->left);  
        if (node->right) q.push(node->right);  
    }  
    res.push_back(level);  
}  
return res;
```

Problem No. 107: Binary Tree Level Order Traversal II

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            for (int i = 0; i < size; 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);
            }
            res.push_back(level);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```



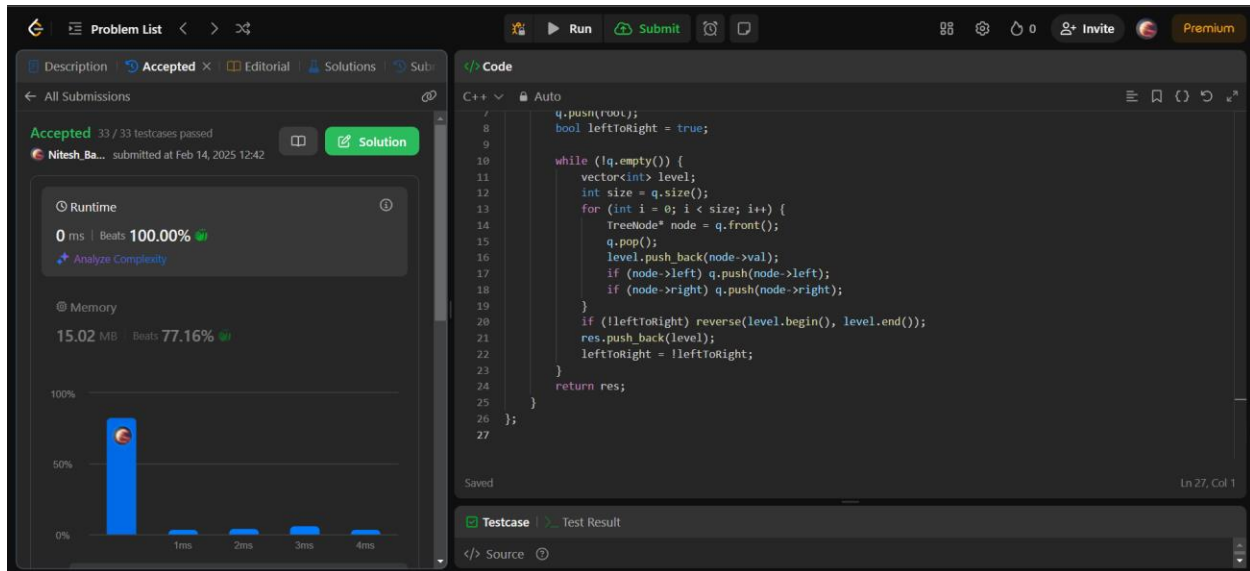
Problem No. 103: Binary Tree Zigzag Level Order Traversal

Code:

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);
        bool leftToRight = true;

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            for (int i = 0; i < size; 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);
            }
            if (!leftToRight) reverse(level.begin(), level.end());
            res.push_back(level);
            leftToRight = !leftToRight;
        }
        return res;
    }
};
```

```
}  
};
```



```
7 q.push(root);  
8 bool leftToRight = true;  
9  
10 while (!q.empty()) {  
11     vector<int> level;  
12     int size = q.size();  
13     for (int i = 0; i < size; i++) {  
14         TreeNode* node = q.front();  
15         q.pop();  
16         level.push_back(node->val);  
17         if (node->left) q.push(node->left);  
18         if (node->right) q.push(node->right);  
19     }  
20     if (!leftToRight) reverse(level.begin(), level.end());  
21     res.push_back(level);  
22     leftToRight = !leftToRight;  
23 }  
24 return res;  
25  
26 }  
27
```

Problem No. 199: Binary Tree Right Side View

Code:

```
class Solution {  
public:  
    vector<int> rightSideView(TreeNode* root) {  
        vector<int> res;  
        if (!root) return res;  
        queue<TreeNode*> q;  
        q.push(root);  
  
        while (!q.empty()) {  
            int size = q.size();  
            for (int i = 0; i < size; i++) {  
                TreeNode* node = q.front();
```

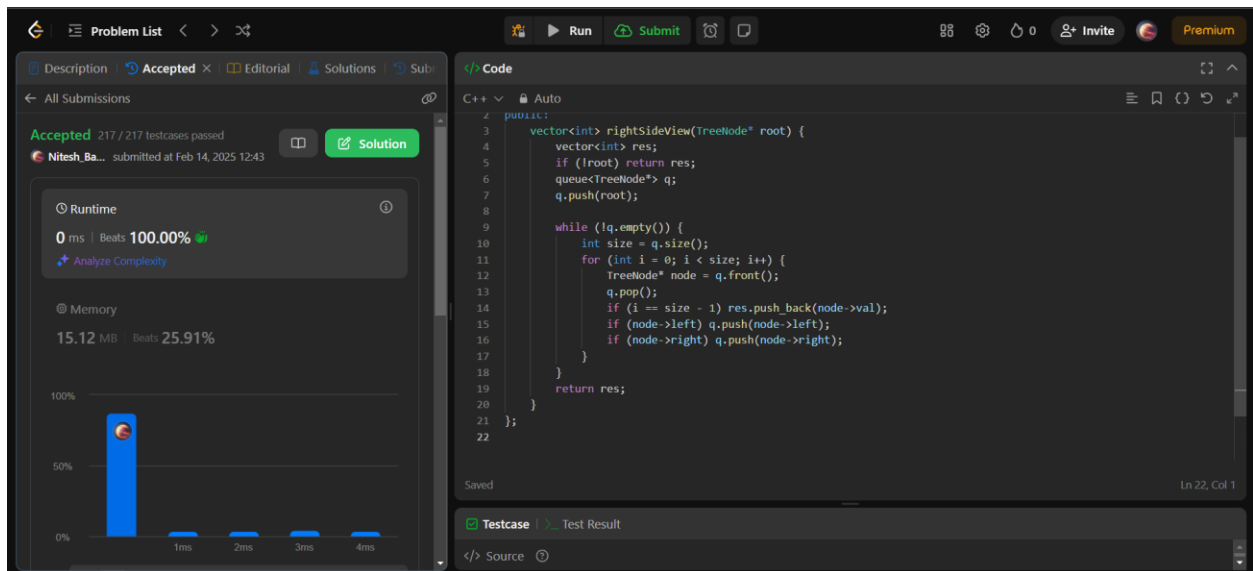
```

        q.pop();
        if (i == size - 1) res.push_back(node->val);
        if (node->left) q.push(node->left);
        if (node->right) q.push(node->right);
    }
}

return res;
}

};

```



```

1 public:
2
3     vector<int> rightSideView(TreeNode* root) {
4         vector<int> res;
5         if (!root) return res;
6         queue<TreeNode*> q;
7         q.push(root);
8
9         while (!q.empty()) {
10             int size = q.size();
11             for (int i = 0; i < size; i++) {
12                 TreeNode* node = q.front();
13                 q.pop();
14                 if (i == size - 1) res.push_back(node->val);
15                 if (node->left) q.push(node->left);
16                 if (node->right) q.push(node->right);
17             }
18         }
19         return res;
20     }
21 };
22

```

Problem No. 106: Construct Binary Tree from Inorder and Postorder Traversal

Code:

```

class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> indexMap;

```

```
for (int i = 0; i < inorder.size(); i++) indexMap[inorder[i]] = i;

return build(inorder, postorder, indexMap, 0, inorder.size() - 1, postorder.size() - 1);

}
```

```
TreeNode* build(vector<int>& inorder, vector<int>& postorder, unordered_map<int, int>&
indexMap, int inStart, int inEnd, int postIndex) {

    if (inStart > inEnd) return nullptr;

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

    int inRoot = indexMap[root->val];

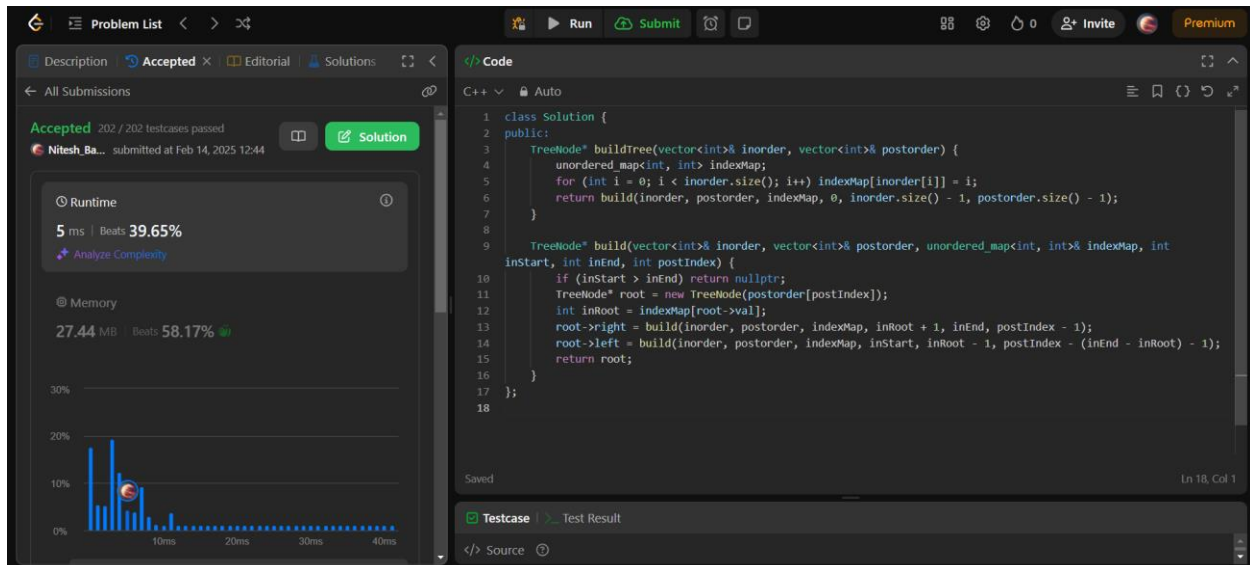
    root->right = build(inorder, postorder, indexMap, inRoot + 1, inEnd, postIndex - 1);

    root->left = build(inorder, postorder, indexMap, inStart, inRoot - 1, postIndex - (inEnd -
inRoot) - 1);

    return root;

}

};
```



The screenshot shows a C++ IDE with the following components:

- Problem List:** Shows the problem is "Accepted" with 202/202 testcases passed. The user "Nitesh, Ba..." submitted it on Feb 14, 2025 at 12:44.
- Runtime:** 5 ms | Beats: 39.65%.
- Memory:** 27.44 MB | Beats: 58.17%.
- Code Editor:** Contains the C++ code for building a binary tree from inorder and postorder traversals. The code is as follows:


```
1 class Solution {
2 public:
3     TreeNode* buildtree(vector<int>& inorder, vector<int>& postorder) {
4         unordered_map<int, int> indexMap;
5         for (int i = 0; i < inorder.size(); i++) indexMap[inorder[i]] = i;
6         return build(inorder, postorder, indexMap, 0, inorder.size() - 1, postorder.size() - 1);
7     }
8
9     TreeNode* build(vector<int>& inorder, vector<int>& postorder, unordered_map<int, int>& indexMap, int
inStart, int inEnd, int postIndex) {
10         if (inStart > inEnd) return nullptr;
11         TreeNode* root = new TreeNode(postorder[postIndex]);
12         int inRoot = indexMap[root->val];
13         root->right = build(inorder, postorder, indexMap, inRoot + 1, inEnd, postIndex - 1);
14         root->left = build(inorder, postorder, indexMap, inStart, inRoot - 1, postIndex - (inEnd - inRoot) - 1);
15         return root;
16     }
17 };
18
```
- Testcase:** Shows the test result for the code.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem No. 513: Find Bottom Left Tree Value

Code:

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

        while (!q.empty()) {
            res = q.front()->val;
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return res;
    }
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem List < >

Description **Accepted** **Editorial** **Solutions** **Submissions**

← All Submissions

Accepted 79 / 79 testcases passed

Nitesh_Ba... submitted at Feb 14, 2025 12:44

Runtime

0 ms | Beats 100.00%

[Analyze Complexity](#)

Memory

24.95 MB | Beats 63.66%

100%
75%
50%
25%
0%

2ms 4ms 6ms

Code

```
1 class Solution {
2 public:
3     int findBottomLeftValue(TreeNode* root) {
4         queue<TreeNode*> q;
5         q.push(root);
6         int res = root->val;
7
8         while (!q.empty()) {
9             res = q.front()->val;
10            int size = q.size();
11            for (int i = 0; i < size; i++) {
12                TreeNode* node = q.front();
13                q.pop();
14                if (node->left) q.push(node->left);
15                if (node->right) q.push(node->right);
16            }
17        }
18        return res;
19    };
20 };
21
```

Saved Ln 21, Col 1

Testcase | **Test Result**

</> Source