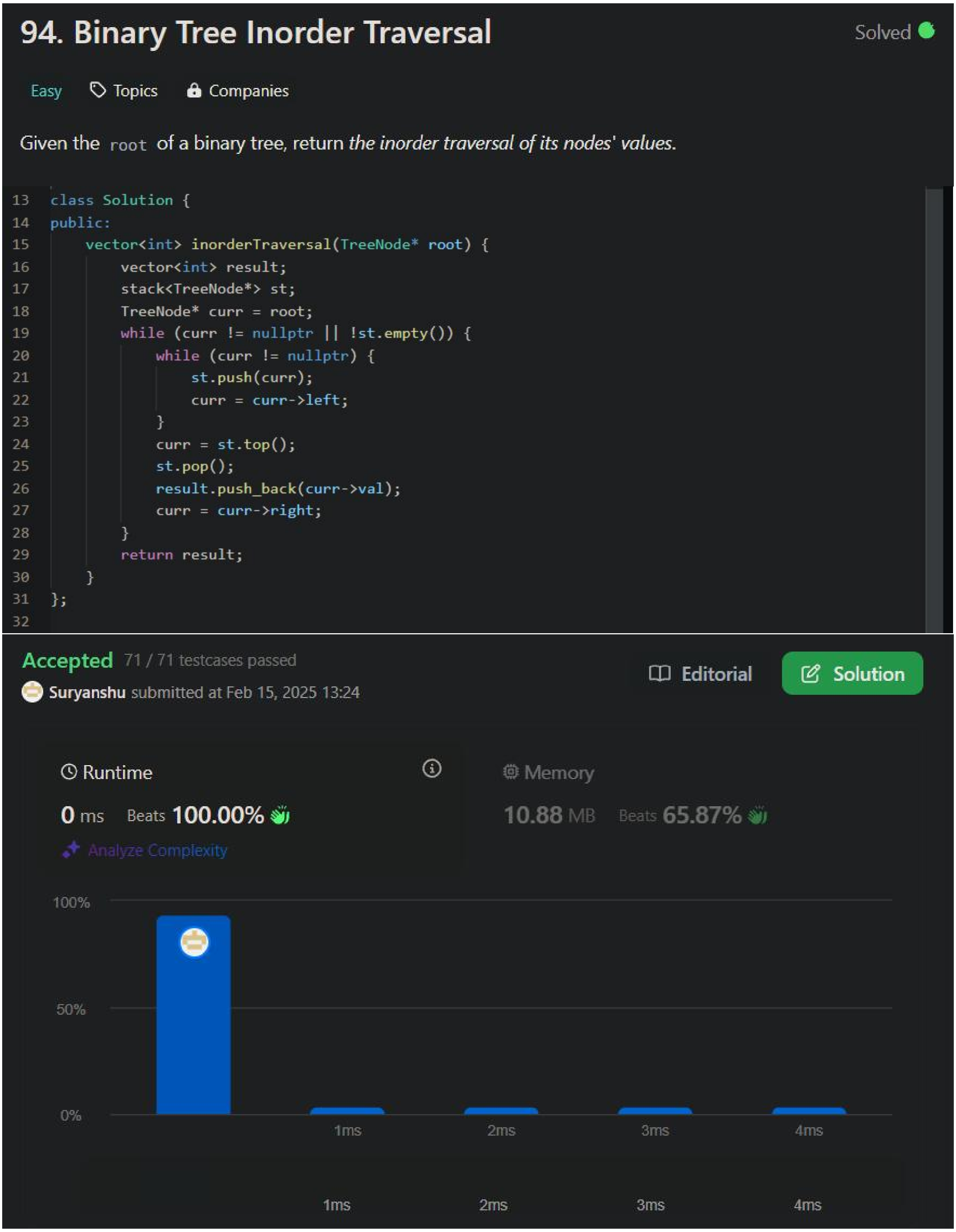
Advanced Programming Lab 2
Assignment 2

Suryanshu
22BCS12106
605 - B

Question 1



94. Binary Tree Inorder Traversal

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        stack<TreeNode*> st;
        TreeNode* curr = root;
        while (curr != nullptr || !st.empty()) {
            while (curr != nullptr) {
                st.push(curr);
                curr = curr->left;
            }
            curr = st.top();
            st.pop();
            result.push_back(curr->val);
            curr = curr->right;
        }
        return result;
    }
};
```

Accepted 71 / 71 testcases passed

Suryanshu submitted at Feb 15, 2025 13:24

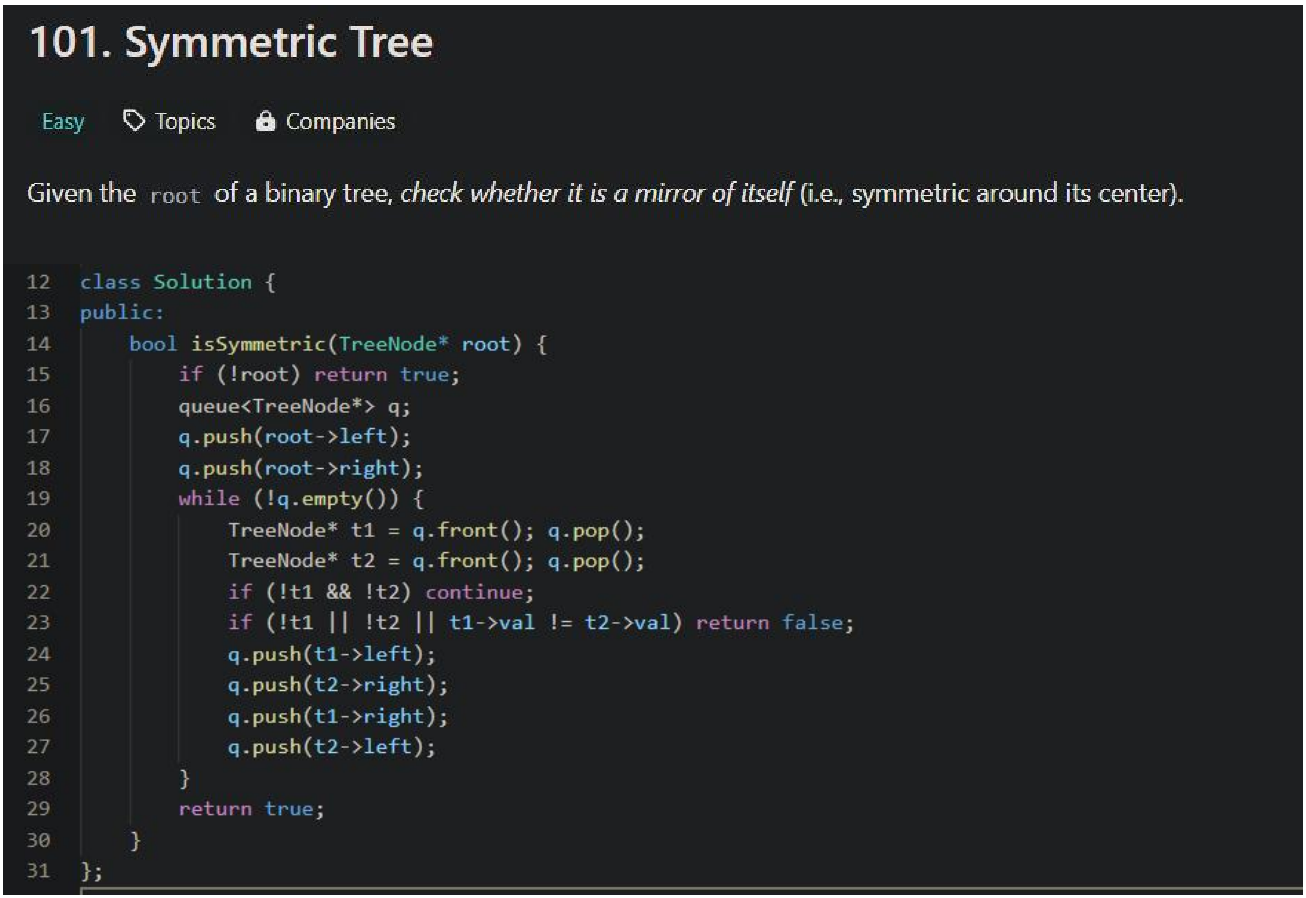
Runtime

0 ms Beats 100.00%

Memory

10.88 MB Beats 65.87%

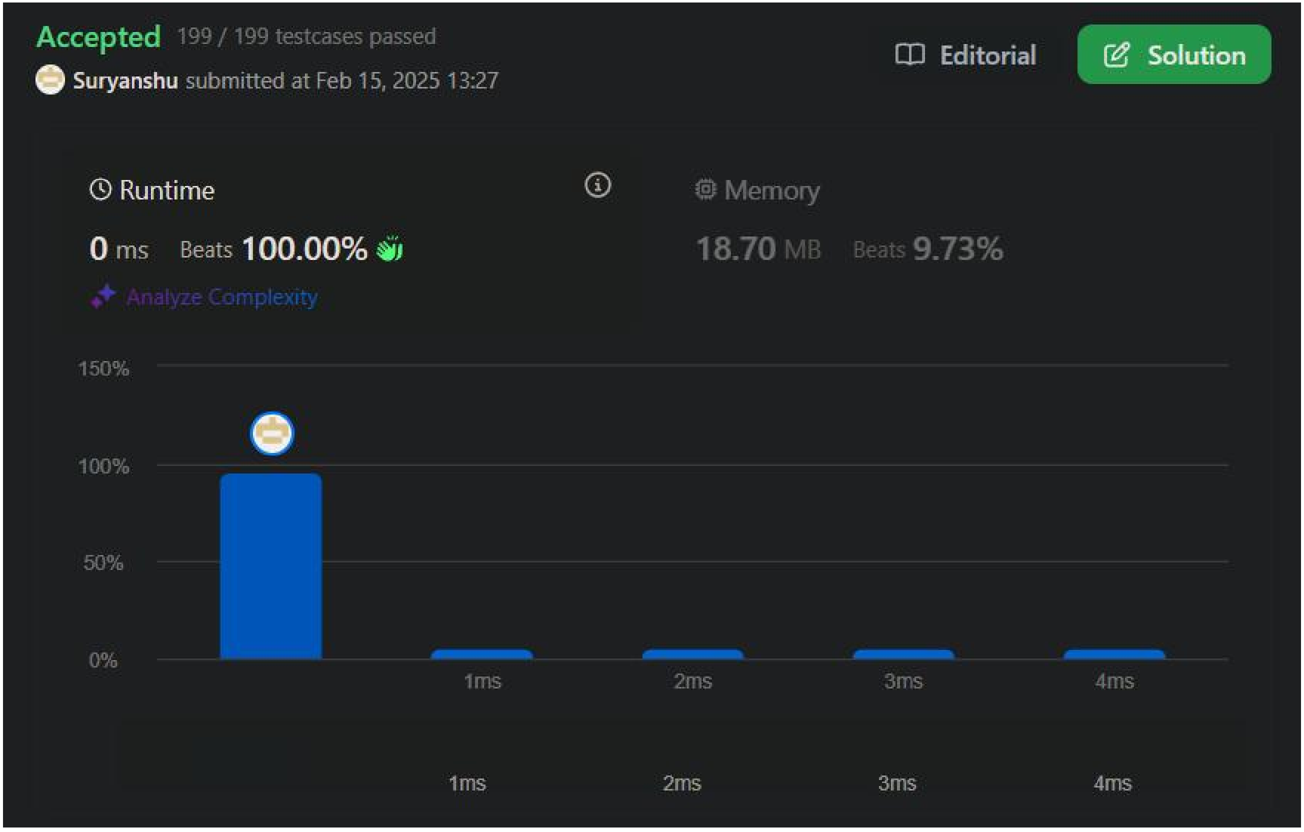
Question 2



101. Symmetric Tree

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

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        queue<TreeNode*> q;
        q.push(root->left);
        q.push(root->right);
        while (!q.empty()) {
            TreeNode* t1 = q.front(); q.pop();
            TreeNode* t2 = q.front(); q.pop();
            if (!t1 && !t2) continue;
            if (!t1 || !t2 || t1->val != t2->val) return false;
            q.push(t1->left);
            q.push(t2->right);
            q.push(t1->right);
            q.push(t2->left);
        }
        return true;
    }
};
```



Accepted 199 / 199 testcases passed

Suryanshu submitted at Feb 15, 2025 13:27

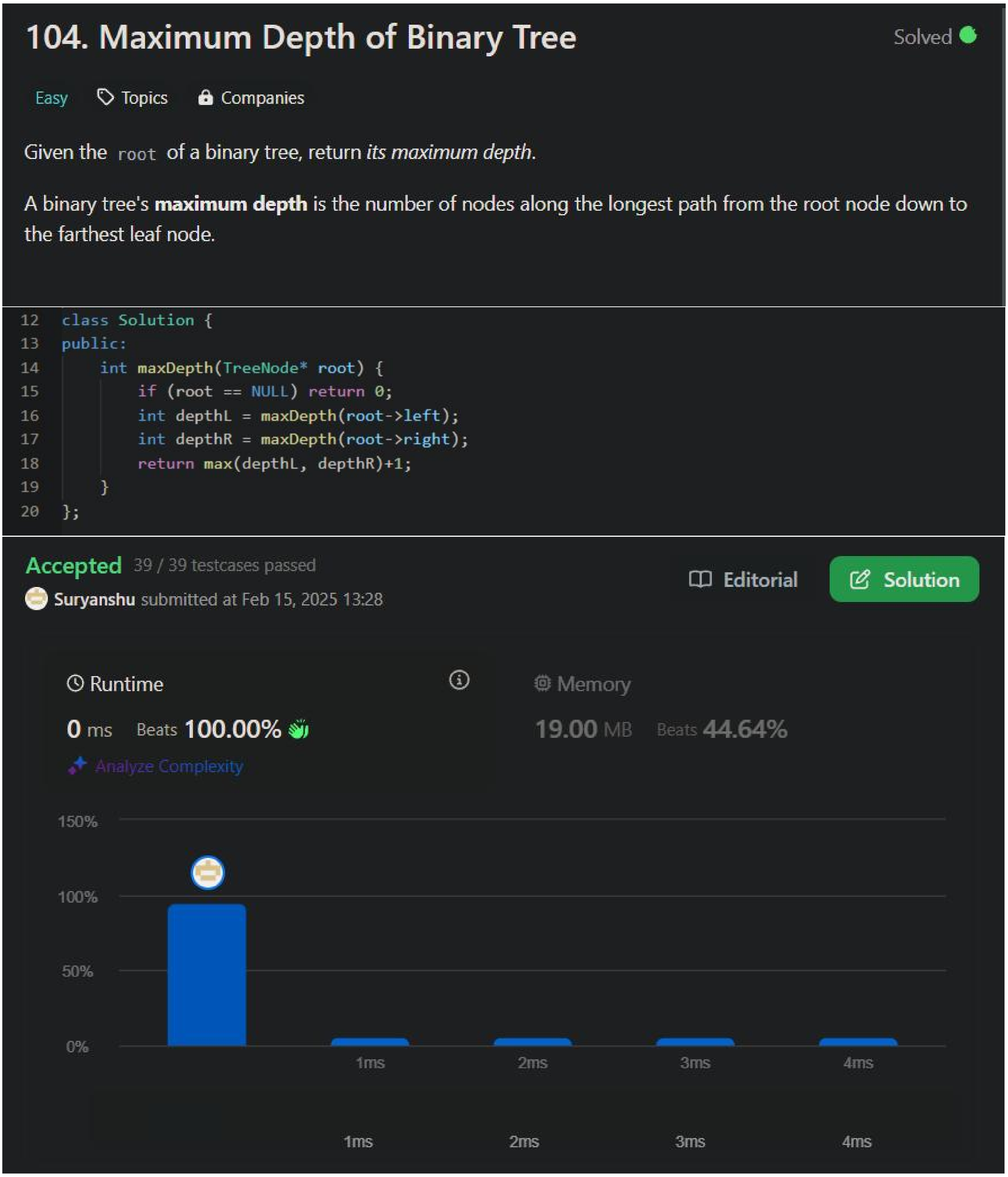
Runtime

0 ms Beats 100.00%

Memory

18.70 MB Beats 9.73%

Question 3



104. Maximum Depth of Binary Tree

Solved ●

Easy Topics Companies

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.

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

Accepted 39 / 39 testcases passed

Suryanshu submitted at Feb 15, 2025 13:28

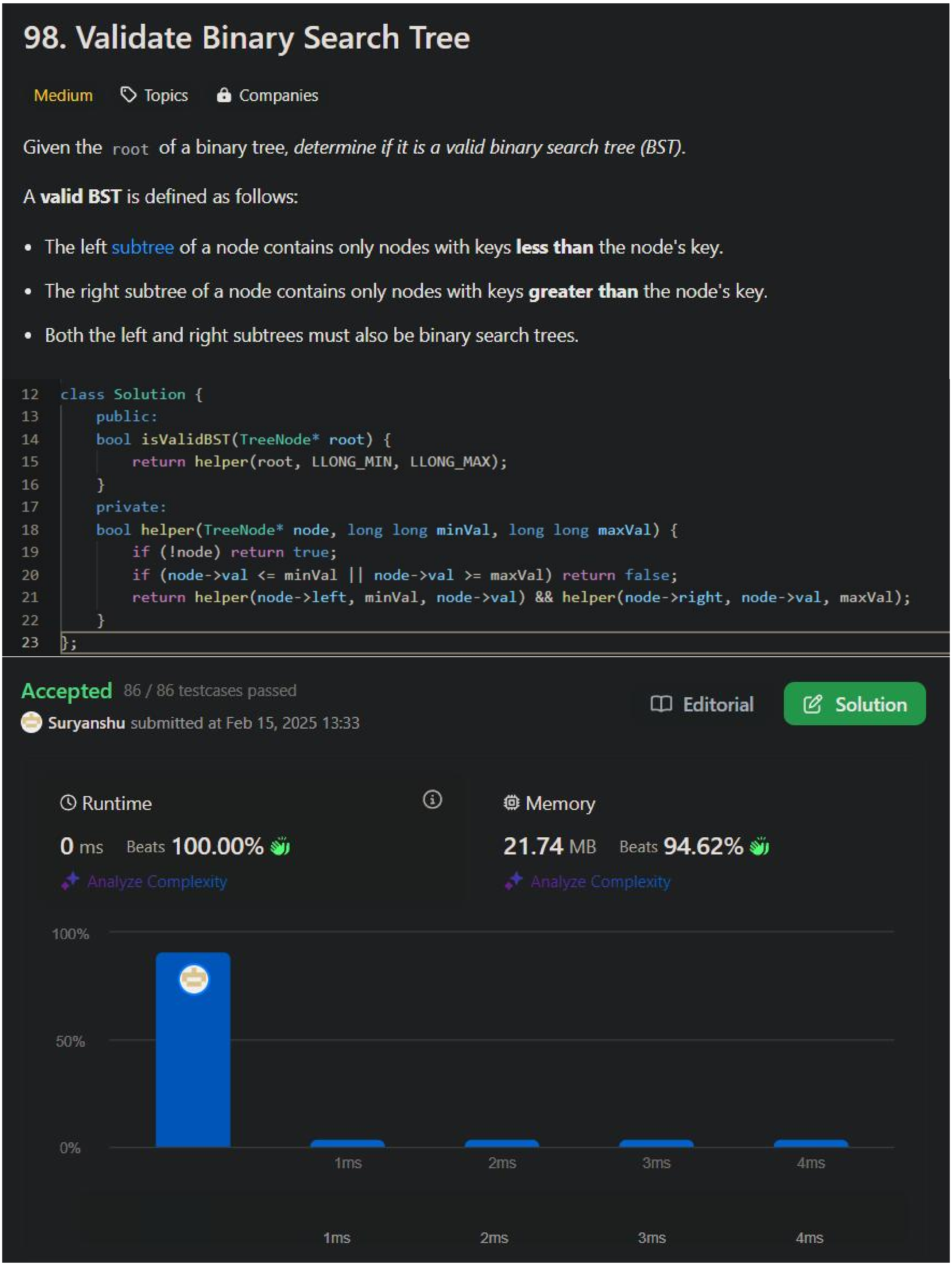
Runtime

0 ms Beats 100.00%

Memory

19.00 MB Beats 44.64%

Question 4



98. Validate Binary Search Tree

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST)*.

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

```cpp
class Solution {
    public:
    bool isValidBST(TreeNode* root) {
        return helper(root, LLONG_MIN, LLONG_MAX);
    }
    private:
    bool helper(TreeNode* node, long long minVal, long long maxVal) {
        if (!node) return true;
        if (node->val <= minVal || node->val >= maxVal) return false;
        return helper(node->left, minVal, node->val) && helper(node->right, node->val, maxVal);
    }
};
```

Accepted 86 / 86 testcases passed

Suryanshu submitted at Feb 15, 2025 13:33

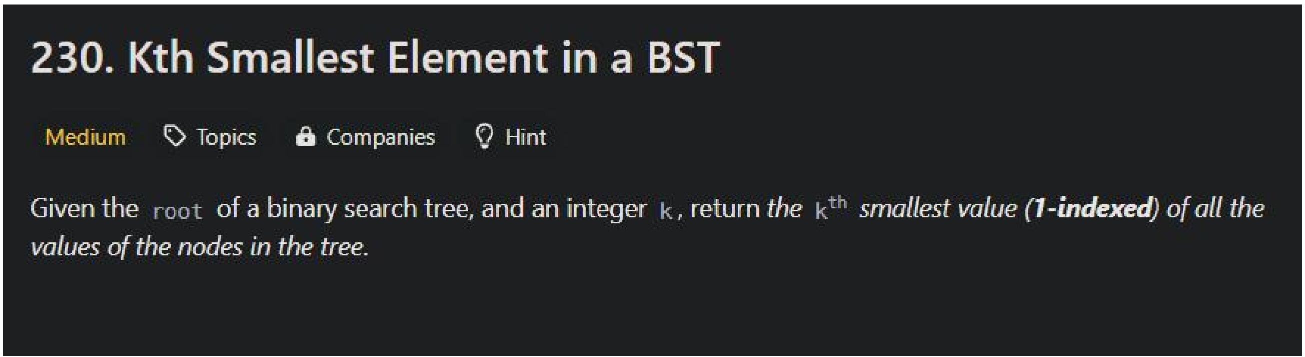
Runtime

0 ms Beats 100.00%

Memory

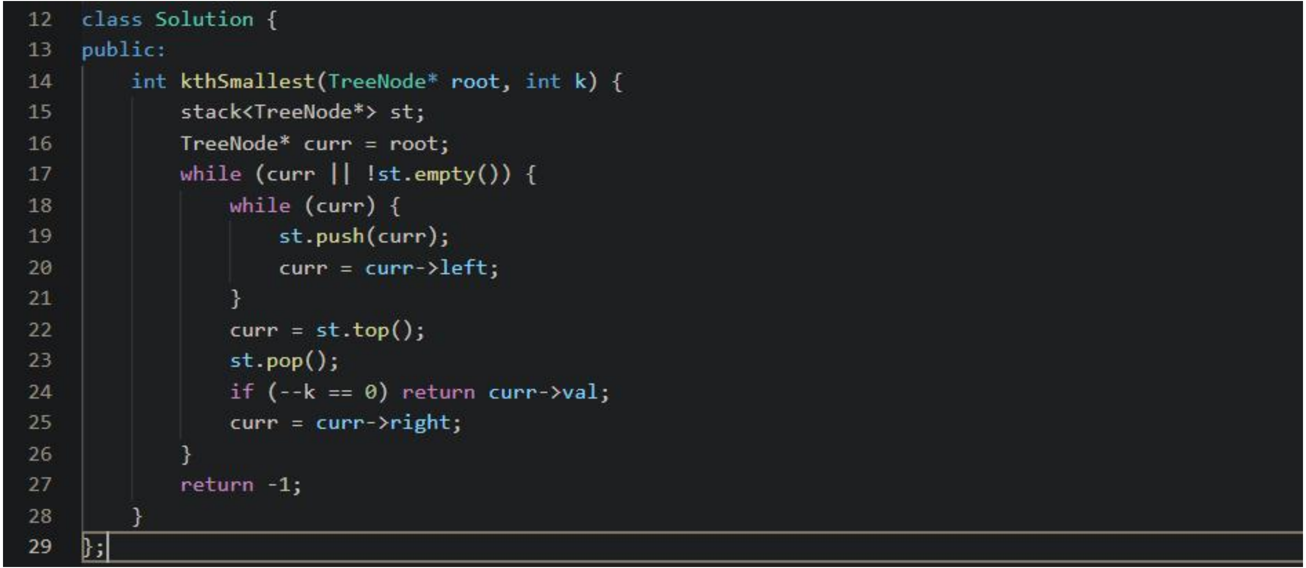
21.74 MB Beats 94.62%

Question 5

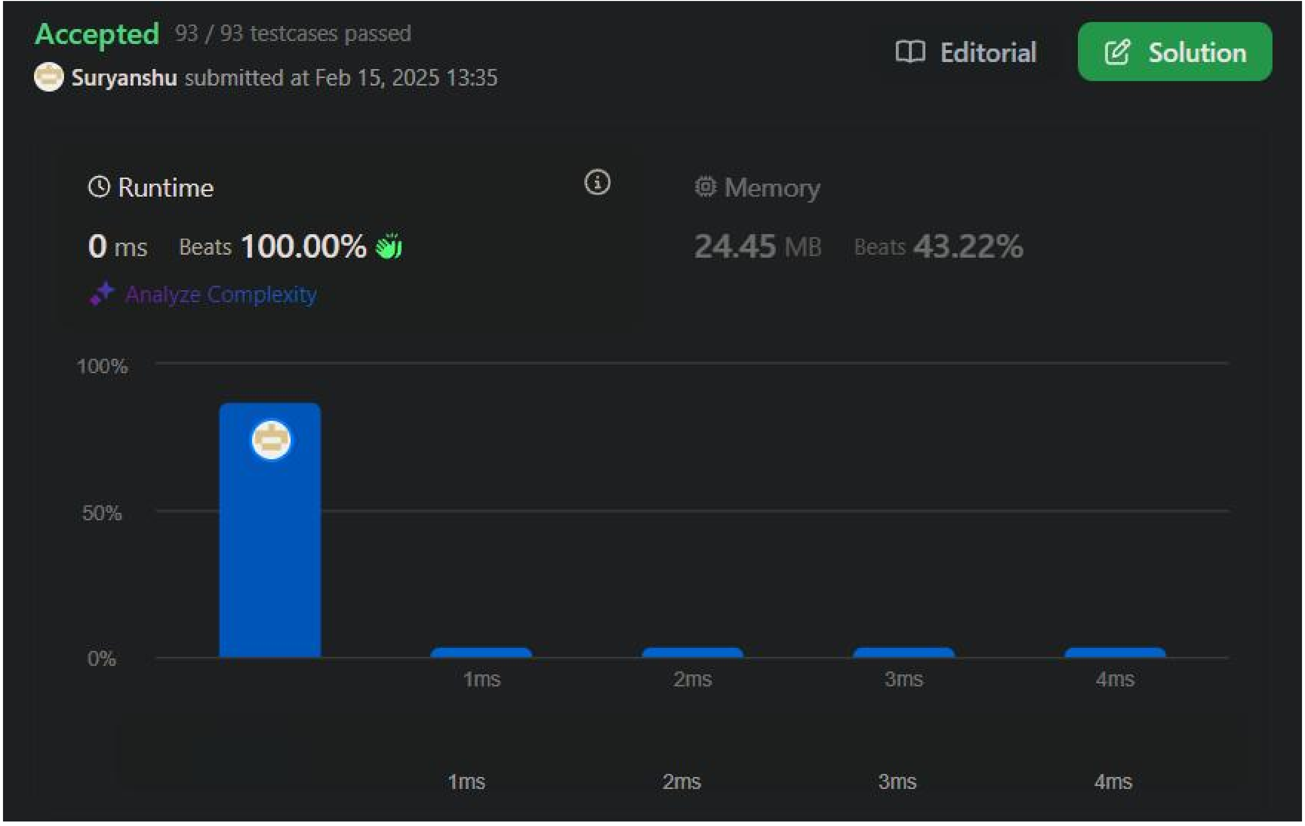


230. Kth Smallest Element in a BST

Given the `root` of a binary search tree, and an integer `k`, return *the* `k`th *smallest value (**1-indexed**) of all the values of the nodes in the tree.*



```cpp
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();
            if (--k == 0) return curr->val;
            curr = curr->right;
        }
        return -1;
    }
};
```



Accepted 93 / 93 testcases passed

Suryanshu submitted at Feb 15, 2025 13:35

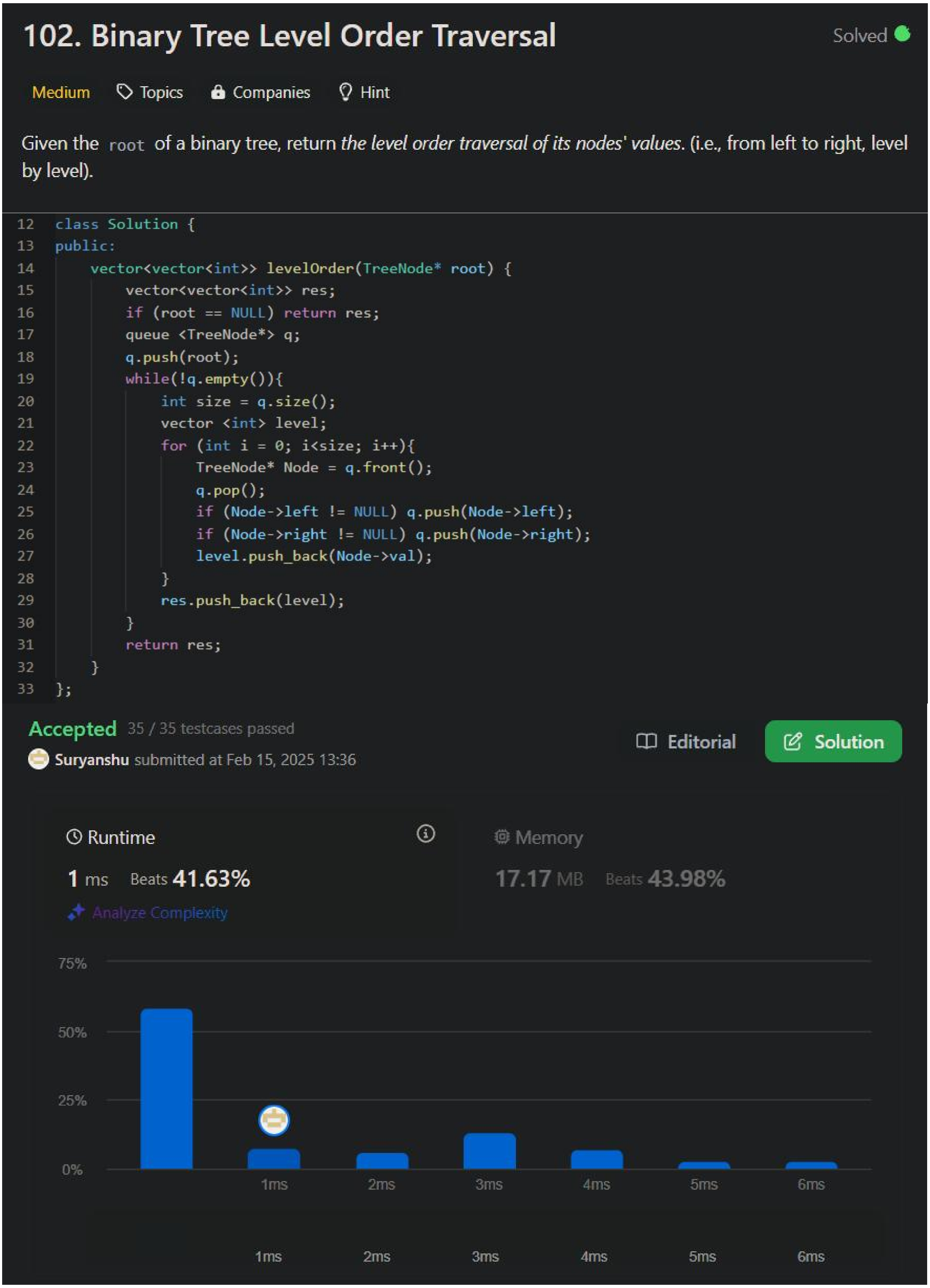
Runtime

0 ms Beats 100.00%

Memory

24.45 MB Beats 43.22%

Question 6



102. 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).

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

Accepted 35 / 35 testcases passed

Suryanshu submitted at Feb 15, 2025 13:36

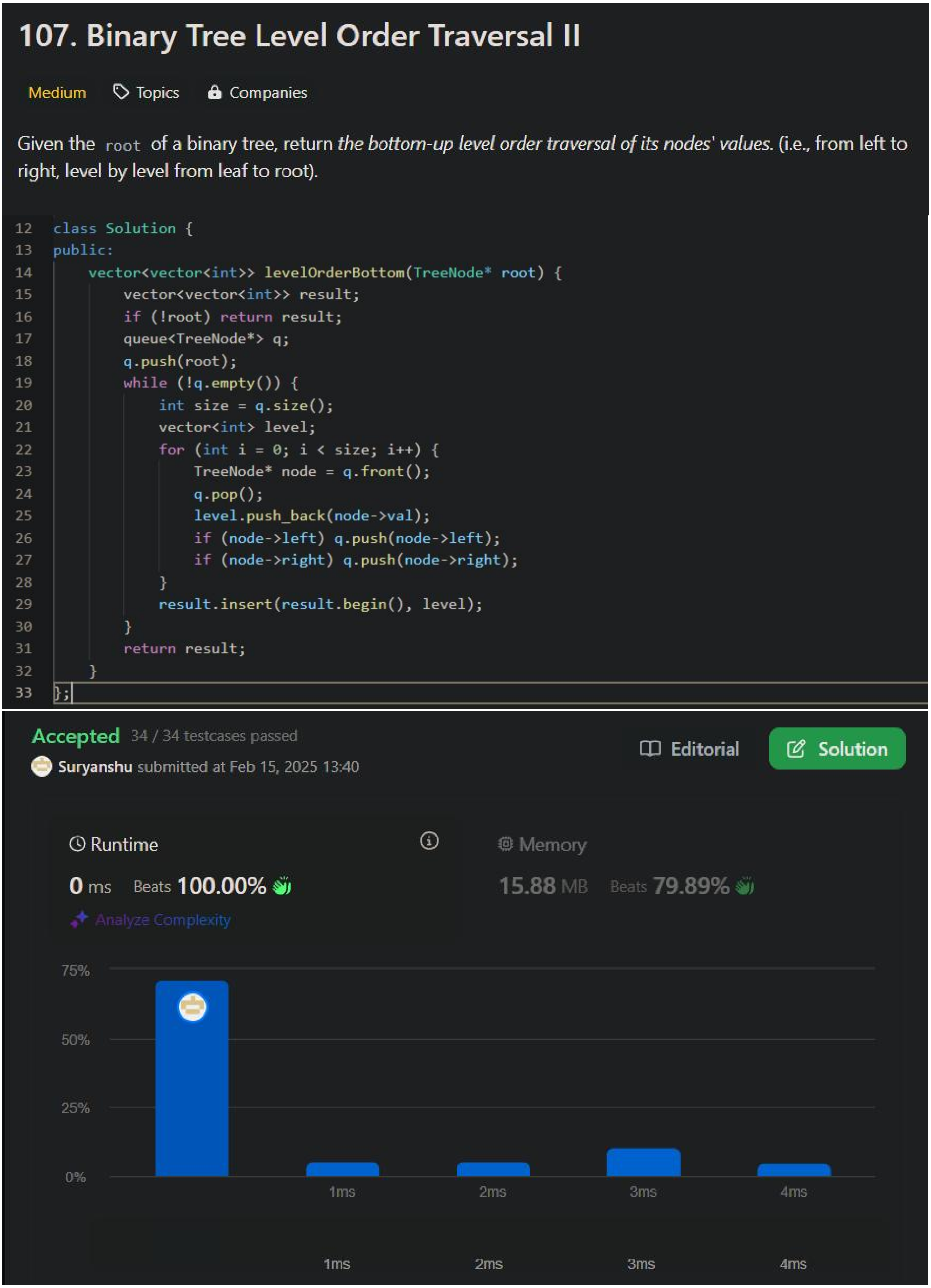
Runtime

1 ms Beats 41.63%

Memory

17.17 MB Beats 43.98%

Question 7



107. Binary Tree Level Order Traversal II

Medium Topics Companies

Given the `root` of a binary tree, return *the bottom-up level order traversal of its nodes' values*. (i.e., from left to right, level by level from leaf to root).

```cpp
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 size = q.size();
            vector<int> level;
            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);
            }
            result.insert(result.begin(), level);
        }
        return result;
    }
};
```

Accepted 34 / 34 testcases passed

Suryanshu submitted at Feb 15, 2025 13:40

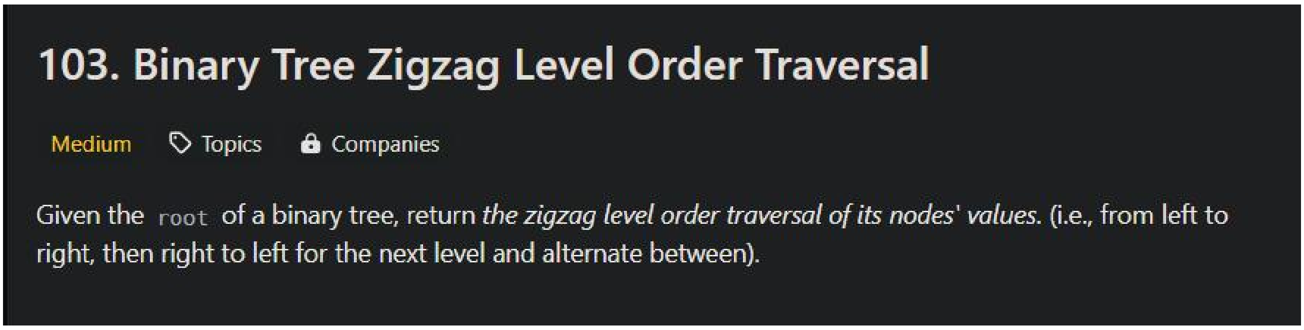
Runtime

0 ms Beats 100.00%

Memory

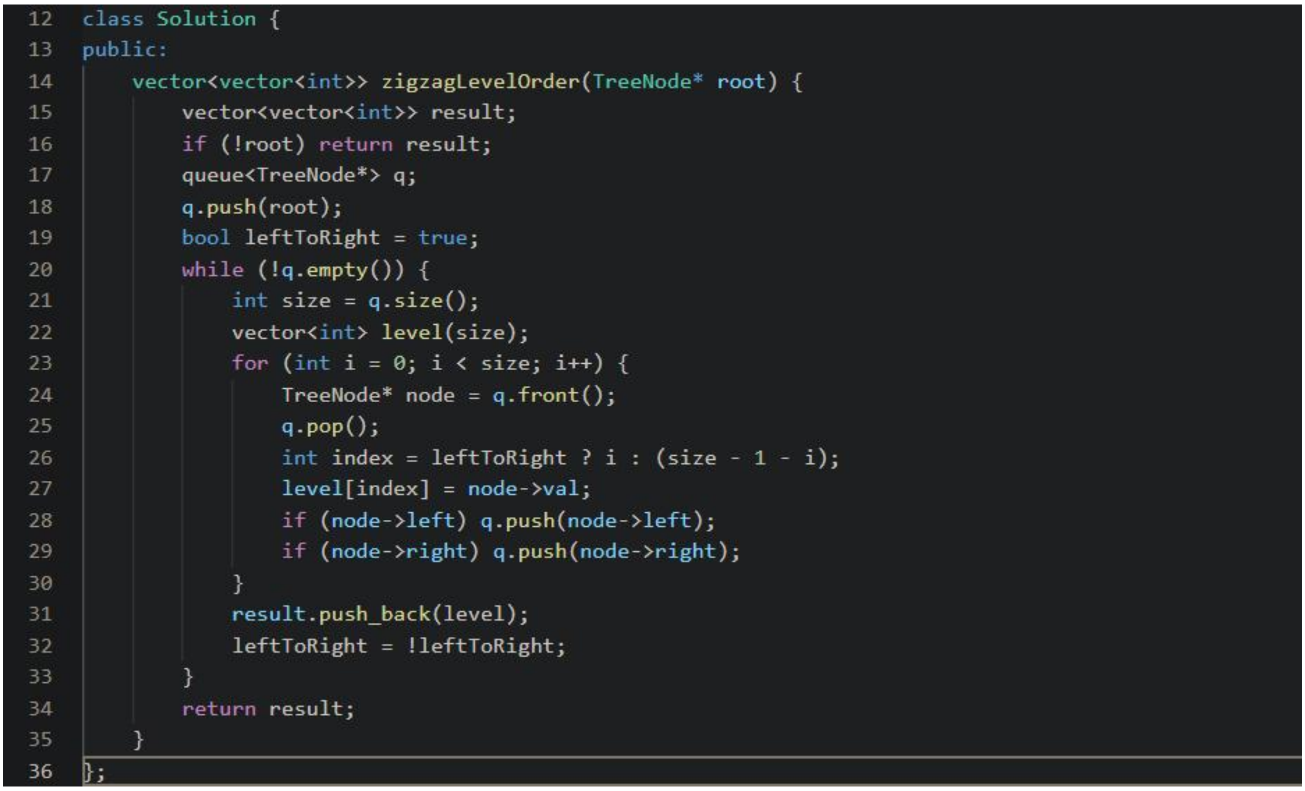
15.88 MB Beats 79.89%

Question 8

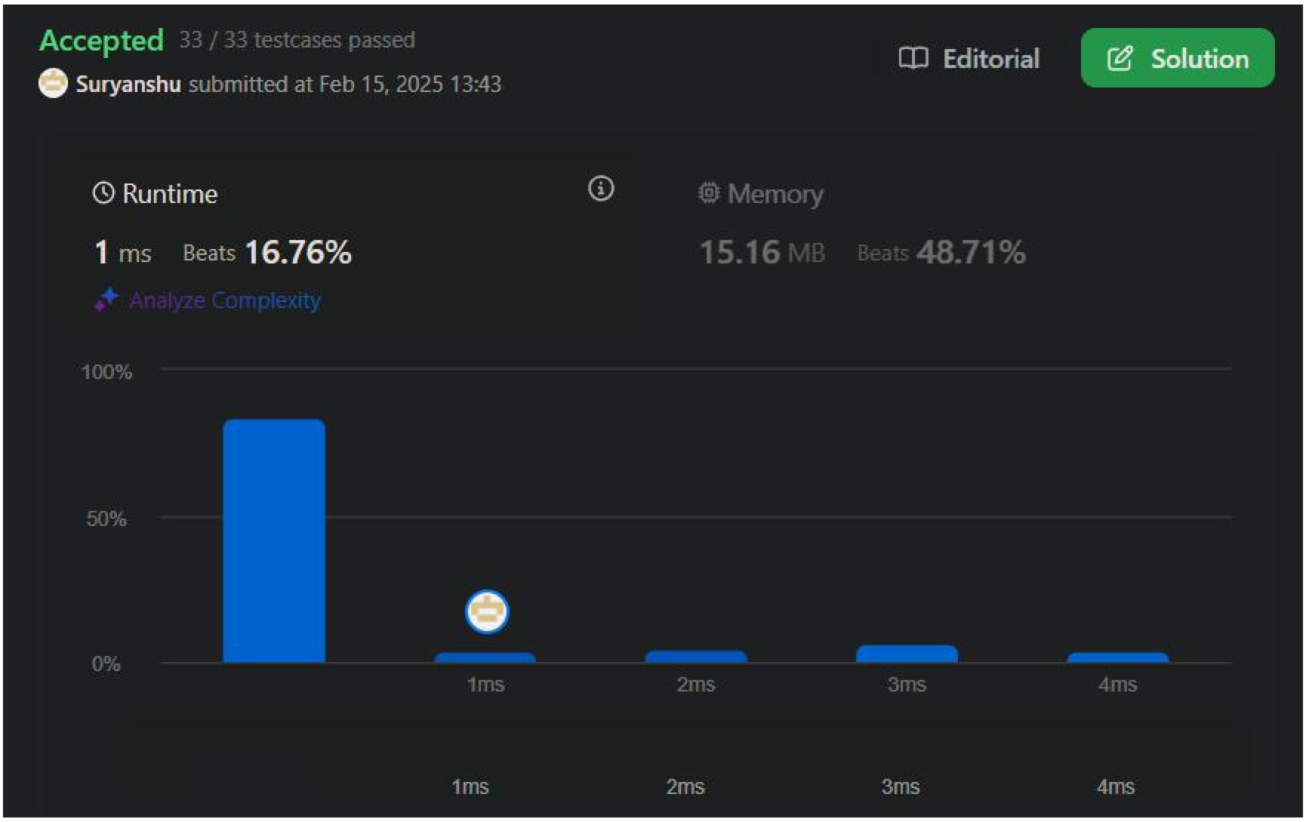


103. Binary Tree Zigzag Level Order Traversal

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).



```cpp
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;
    }
};
```



Accepted 33 / 33 testcases passed

Suryanshu submitted at Feb 15, 2025 13:43

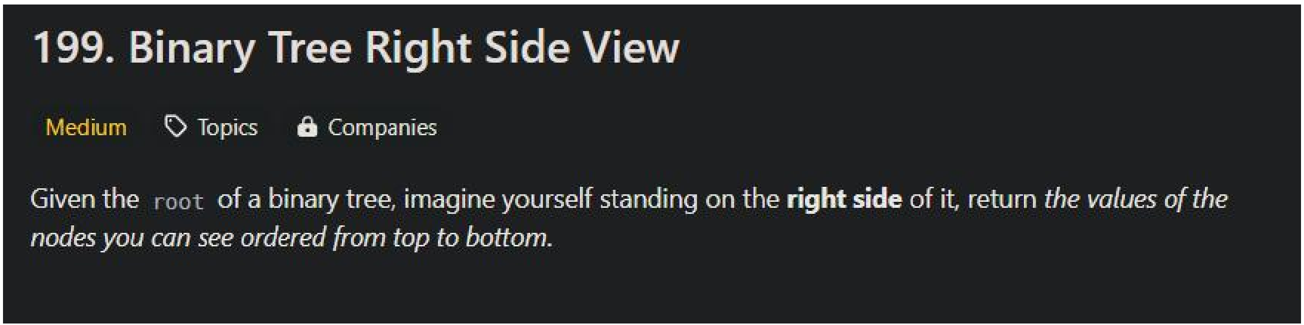
Runtime

1 ms Beats 16.76%

Memory

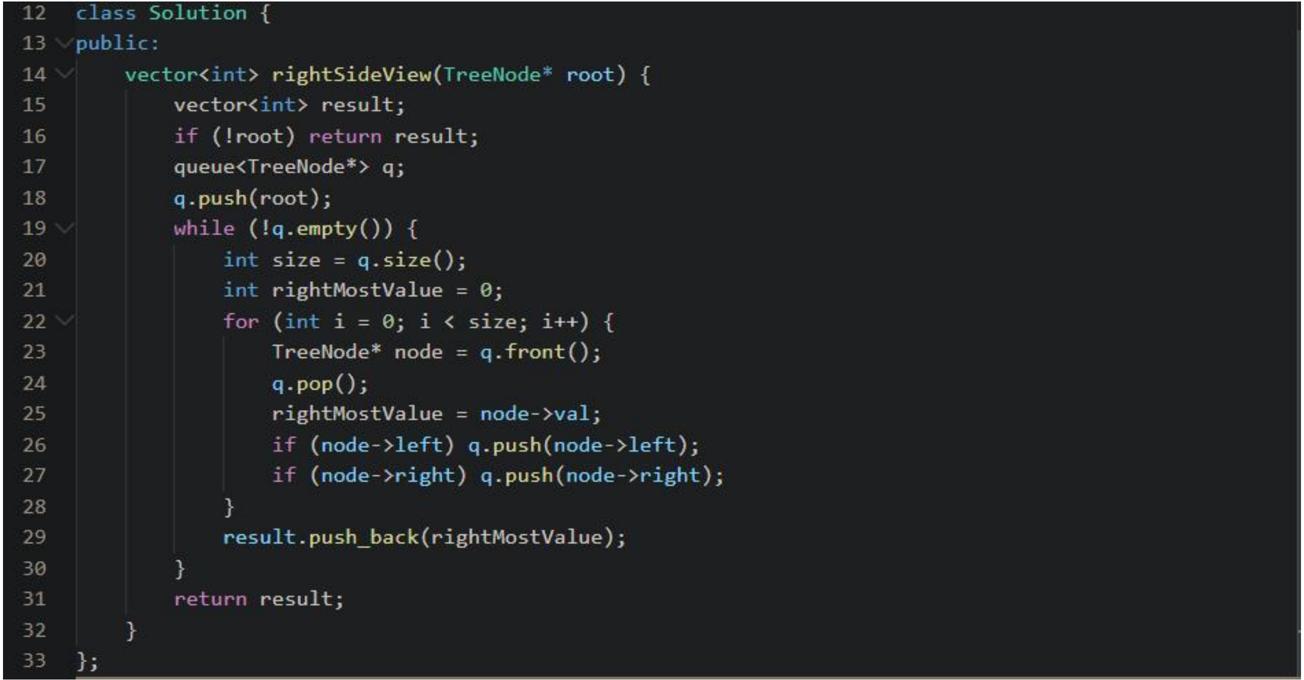
15.16 MB Beats 48.71%

Question 9

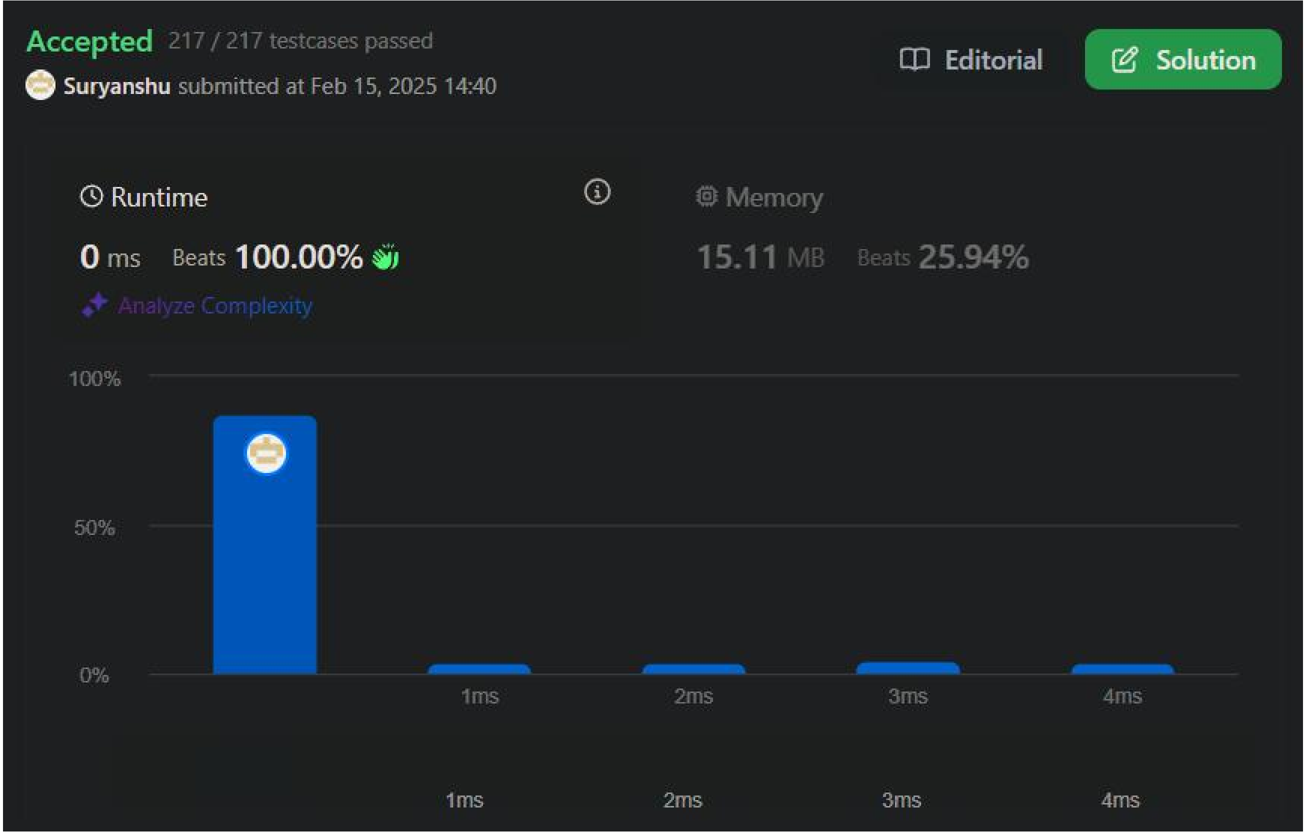


199. Binary Tree Right Side View

Given the `root` of a binary tree, imagine yourself standing on the **right side** of it, return *the values of the nodes you can see ordered from top to bottom.*



```cpp
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> result;
        if (!root) return result;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            int rightMostValue = 0;
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                rightMostValue = node->val;
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(rightMostValue);
        }
        return result;
    }
};
```



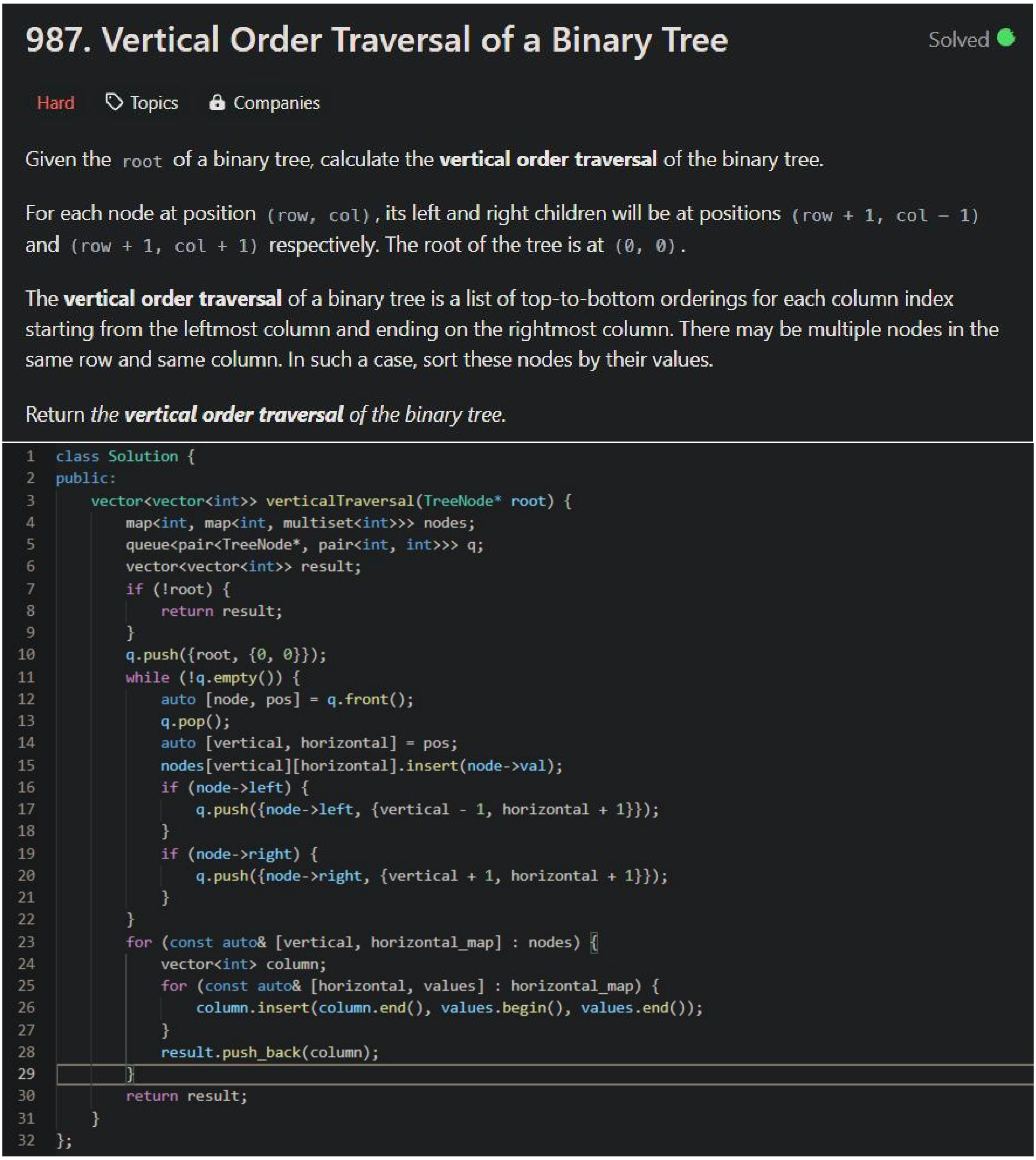
Accepted 217 / 217 testcases passed

Suryanshu submitted at Feb 15, 2025 14:40

Runtime: 0 ms Beats 100.00%

Memory: 15.11 MB Beats 25.94%

Question 10



987. Vertical Order Traversal of a Binary Tree

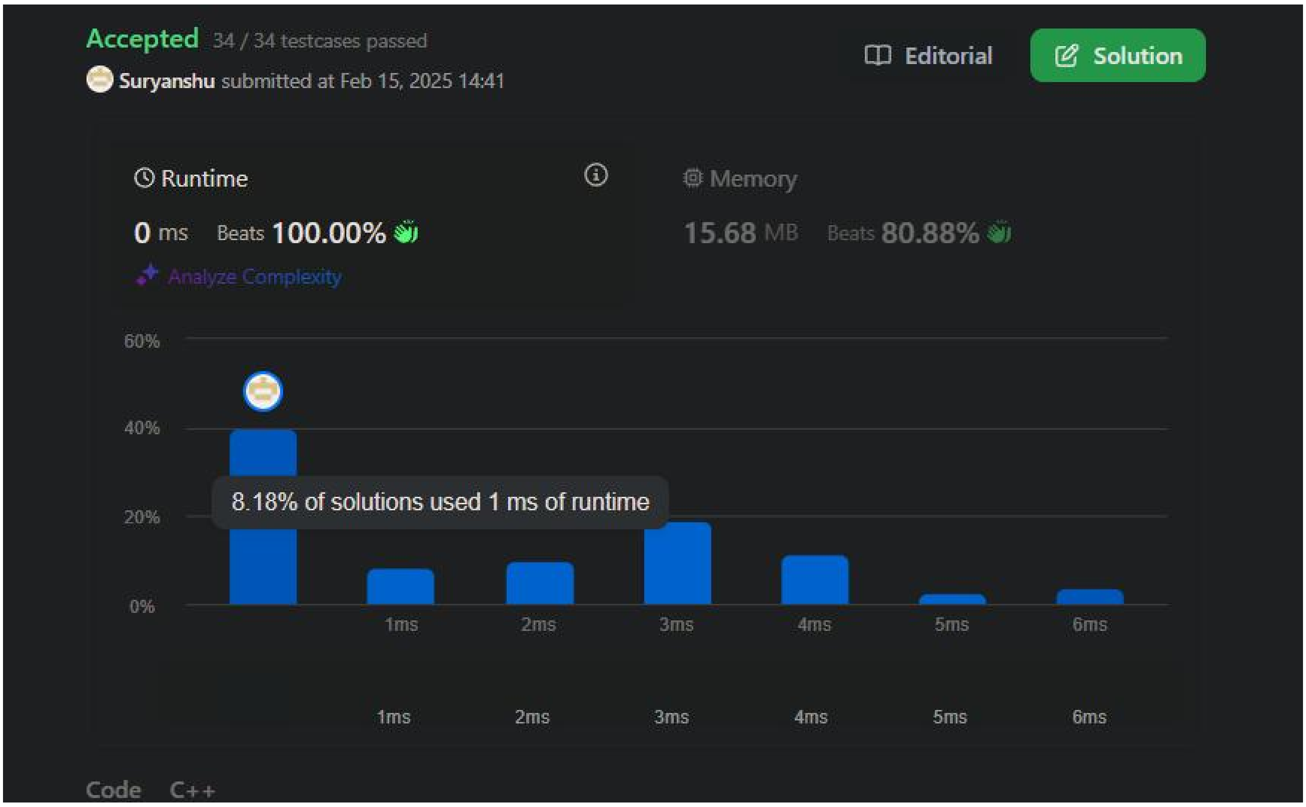
Given the `root` of a binary tree, calculate the **vertical order traversal** of the binary tree.

For each node at position `(row, col)`, its left and right children will be at positions `(row + 1, col - 1)` and `(row + 1, col + 1)` respectively. The root of the tree is at `(0, 0)`.

The **vertical order traversal** of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values.

Return *the **vertical order traversal** of the binary tree.*

```cpp
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int, map<int, multiset<int>>> nodes;
        queue<pair<TreeNode*, pair<int, int>>> q;
        vector<vector<int>> result;
        if (!root) {
            return result;
        }
        q.push({root, {0, 0}});
        while (!q.empty()) {
            auto [node, pos] = q.front();
            q.pop();
            auto [vertical, horizontal] = pos;
            nodes[vertical][horizontal].insert(node->val);
            if (node->left) {
                q.push({node->left, {vertical - 1, horizontal + 1}});
            }
            if (node->right) {
                q.push({node->right, {vertical + 1, horizontal + 1}});
            }
        }
        for (const auto& [vertical, horizontal_map] : nodes) {
            vector<int> column;
            for (const auto& [horizontal, values] : horizontal_map) {
                column.insert(column.end(), values.begin(), values.end());
            }
            result.push_back(column);
        }
        return result;
    }
};
```



Accepted 34 / 34 testcases passed

Suryanshu submitted at Feb 15, 2025 14:41

Runtime

0 ms Beats 100.00%

Memory

15.68 MB Beats 80.88%

8.18% of solutions used 1 ms of runtime

Code C++