

Name: Yash Pandey

UID: 22BCS15020

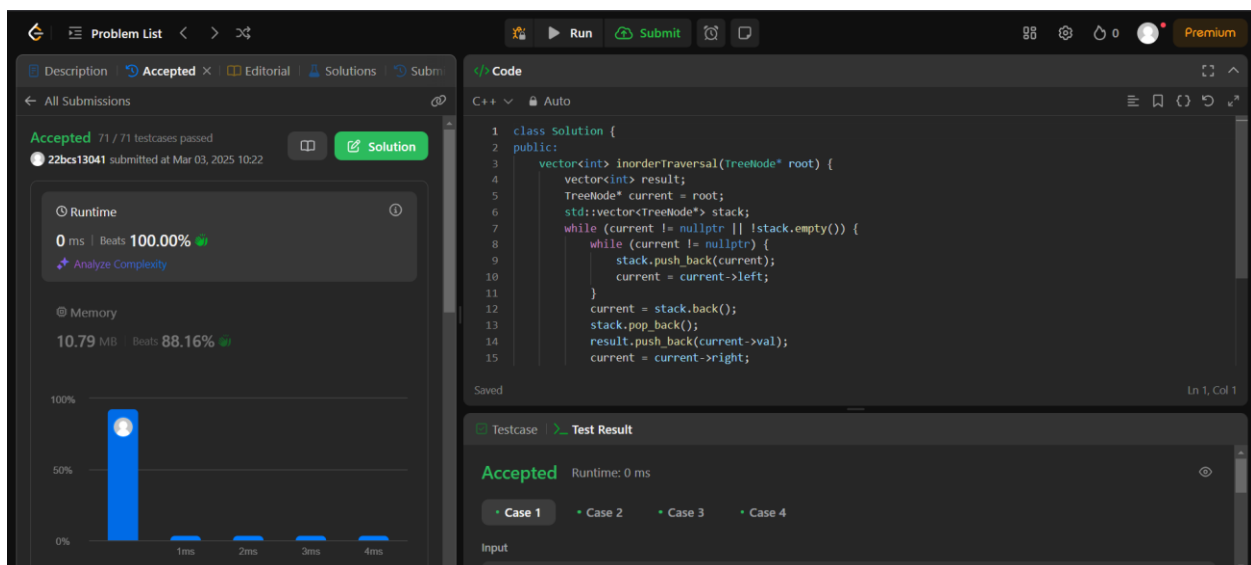
Batch: FL_IoT 601 'A'

1. Binary Tree Inorder Traversal

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        stack<TreeNode*> stack;

        while (root != nullptr || !stack.empty()) {
            while (root != nullptr) {
                stack.push(root);
                root = root->left;
            }
            root = stack.top(), stack.pop();
            ans.push_back(root->val);
            root = root->right;
        }

        return ans;
    }
};
```

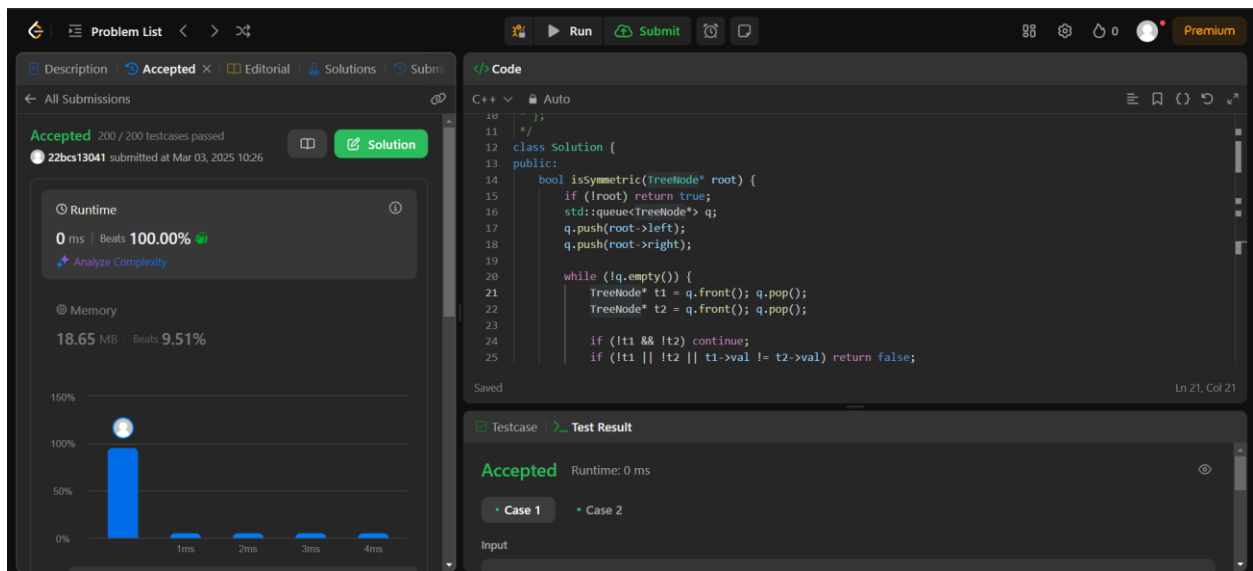


2. Symmetric Tree

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return isSymmetric(root, root);
    }

private:
    bool isSymmetric(TreeNode* p, TreeNode* q) {
        if (!p || !q)
            return p == q;

        return p->val == q->val && //
               isSymmetric(p->left, q->right) && //
               isSymmetric(p->right, q->left);
    }
};
```



3. Maximum Depth of Binary Tree

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

};

The screenshot displays a coding platform interface. On the left, the submission status is 'Accepted' with 39/39 testcases passed. The runtime is 0 ms, achieving 100.00% beats, and the memory usage is 19.09 MB, achieving 44.95% beats. A bar chart shows the runtime distribution. The right panel shows the C++ code for a binary tree structure and a maxDepth function. The test case section shows 'Case 1' with input 'root = 1 2 9 20 null null 15 71'.

4. [Validate Binary Search Tree](#)

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return isValidBST(root, nullptr, nullptr);
    }
private:
    bool isValidBST(TreeNode* root, TreeNode* minNode, TreeNode* maxNode) {
        if (root == nullptr)
            return true;
        if (minNode && root->val <= minNode->val)
            return false;
        if (maxNode && root->val >= maxNode->val)
            return false;

        return isValidBST(root->left, minNode, root) &&
            isValidBST(root->right, root, maxNode);
    }
};
```

The screenshot shows a LeetCode submission for the problem "Kth Smallest Element in a BST". The submission is "Accepted" with 86/86 testcases passed. The runtime is 0 ms, beating 100.00% of submissions. The memory is 22.00 MB, beating 48.36% of submissions. The code is a recursive function to find the kth smallest element in a BST.

```

14 bool isValidBST(TreeNode* root) {
15     return isValidBST(root, nullptr, nullptr);
16 }
17 private:
18 bool isValidBST(TreeNode* root, TreeNode* minNode, TreeNode* maxNode) {
19     if (root == nullptr)
20         return true;
21     if (minNode && root->val <= minNode->val)
22         return false;
23     if (maxNode && root->val >= maxNode->val)
24         return false;
25     return isValidBST(root->left, minNode, root) &&
26            isValidBST(root->right, root, maxNode);
27 }
28

```

5. Kth Smallest Element in a BST

```

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        const int leftCount = countNodes(root->left);

        if (leftCount == k - 1)
            return root->val;
        if (leftCount >= k)
            return kthSmallest(root->left, k);
        return kthSmallest(root->right, k - 1 - leftCount); // leftCount < k
    }

private:
    int countNodes(TreeNode* root) {
        if (root == nullptr)
            return 0;
        return 1 + countNodes(root->left) + countNodes(root->right);
    }
};

```

The screenshot displays a LeetCode submission for the problem "Kth Smallest Element in a BST". The submission is marked as "Accepted" with 93/93 test cases passed. The runtime is 0 ms, beating 100.00% of submissions, and the memory usage is 24.46 MB, beating 42.65%. The code is written in C++ and implements a recursive function to find the kth smallest element in a binary search tree.

```

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        const int leftCount = countNodes(root->left);

        if (leftCount == k - 1)
            return root->val;
        if (leftCount >= k)
            return kthSmallest(root->left, k);
        return kthSmallest(root->right, k - 1 - leftCount); // leftcount < k
    }

private:
    int countNodes(TreeNode* root) {
        if (root == nullptr)
            return 0;
    }
};

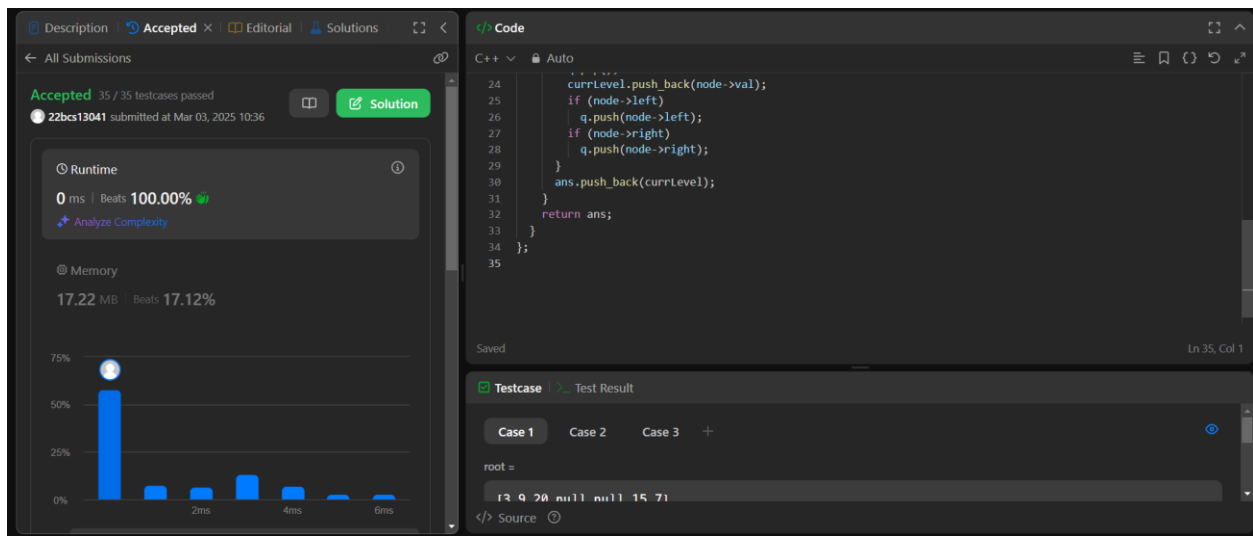
```

6. Binary Tree Level Order Traversal

```

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;
    }
};

```



7. Binary Tree Level Order Traversal II

```

class Solution {
public:
    vector<vector<int>> levelOrderBottom(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);
        }
    }
}

```

Description

Accepted

Editorial

Solutions

Submit

All Submissions

Accepted 34 / 34 testcases passed

22bcs13041 submitted at Mar 03, 2025 10:37

Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

15.82 MB | Beats 79.86%

Time Interval	Memory Usage (%)
0-1ms	75%
1ms-2ms	~5%
2ms-3ms	~5%
3ms-4ms	~10%
4ms-5ms	~5%

Code

```

C++
14 q->pop();
15 currLevel.push_back(node->val);
16 if (node->left)
17     q.push(node->left);
18 if (node->right)
19     q.push(node->right);
20 }
21 ans.push_back(currLevel);
22 }
23
24 ranges::reverse(ans);
25 return ans;
26 }
27 };
28

```

Saved

Testcase

Test Result

Case 1 Case 2 Case 3 +

root =

13 9 20 null null 15 71

Source

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

        vector<vector<int>> ans;
        deque<TreeNode*> dq{{root}};
        bool isLeftToRight = true;

        while (!dq.empty()) {
            vector<int> currLevel;
            for (int sz = dq.size(); sz > 0; --sz)
                if (isLeftToRight) {
                    TreeNode* node = dq.front();
                    dq.pop_front();
                    currLevel.push_back(node->val);
                }
            else {
                currLevel.push_back(dq.back()->val);
                dq.pop_back();
            }
            isLeftToRight = !isLeftToRight;
        }
        return ans;
    }
};
```

```

        if (node->left)
            dq.push_back(node->left);
        if (node->right)
            dq.push_back(node->right);
    } else {
        TreeNode* node = dq.back();
        dq.pop_back();
        currLevel.push_back(node->val);
        if (node->right)
            dq.push_front(node->right);
        if (node->left)
            dq.push_front(node->left);
    }
    ans.push_back(currLevel);
    isLeftToRight = !isLeftToRight;
}

return ans;
}
};

```

The screenshot shows a C++ code editor interface. On the left, a sidebar indicates the solution is 'Accepted' with '33 / 33 testcases passed'. It also shows performance metrics: Runtime is 2 ms (Beats 13.38%) and Memory is 15.28 MB (Beats 18.33%). The main editor displays the C++ code, which is a continuation of the previous code block. The bottom panel shows a 'Testcase' section with 'Case 1' selected, displaying the input 'root = 13 9 20 null null 15 71' and the output '13 9 20 null null 15 71'.

9. Binary Tree Right Side View

```

class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        if (root == nullptr)

```



```

return {};

vector<int> ans;
queue<TreeNode*> q{{root}};

while (!q.empty()) {
    const int size = q.size();
    for (int i = 0; i < size; ++i) {
        TreeNode* node = q.front();
        q.pop();
        if (i == size - 1)
            ans.push_back(node->val);
        if (node->left)
            q.push(node->left);
        if (node->right)
            q.push(node->right);
    }
}
return ans;
}
};

```

The screenshot displays the LeetCode submission interface for the problem "Right Side View". The submission is marked as "Accepted" with 217/217 testcases passed. The runtime is 0 ms, beating 100.00% of other submissions. The memory usage is 15.36 MB, beating 13.90% of other submissions. The code is written in C++ and implements a breadth-first search (BFS) approach using a queue to traverse the tree level by level. For each level, the value of the rightmost node (the last element in the queue for that level) is added to the answer vector.

```

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

        vector<int> ans;
        queue<TreeNode*> q{{root}};

        while (!q.empty()) {
            const int size = q.size();
            for (int i = 0; i < size; ++i) {
                TreeNode* node = q.front();
                q.pop();
                if (i == size - 1)
                    ans.push_back(node->val);
                if (node->left)
                    q.push(node->left);
                if (node->right)
                    q.push(node->right);
            }
        }
        return ans;
    }
};

```

The test case section shows a single case with the input tree structure: 1 2 3 null 5 null 4 1.

10. Construct Binary Tree from Inorder and Postorder Traversal

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

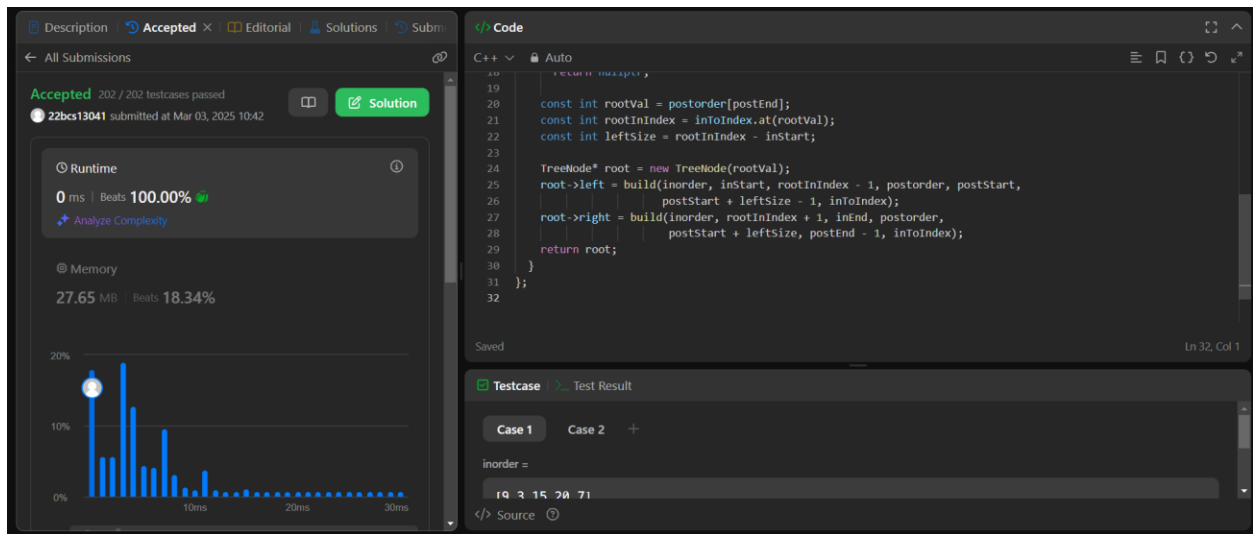
        for (int i = 0; i < inorder.size(); ++i)
            inToIndex[inorder[i]] = i;

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

private:
    TreeNode* build(const vector<int>& inorder, int inStart, int inEnd,
                    const vector<int>& postorder, int postStart, int postEnd,
                    const unordered_map<int, int>& inToIndex) {
        if (inStart > inEnd)
            return nullptr;

        const int rootVal = postorder[postEnd];
        const int rootInIndex = inToIndex.at(rootVal);
        const int leftSize = rootInIndex - inStart;

        TreeNode* root = new TreeNode(rootVal);
        root->left = build(inorder, inStart, rootInIndex - 1, postorder, postStart,
                        postStart + leftSize - 1, inToIndex);
        root->right = build(inorder, rootInIndex + 1, inEnd, postorder,
                        postStart + leftSize, postEnd - 1, inToIndex);
        return root;
    }
};
```

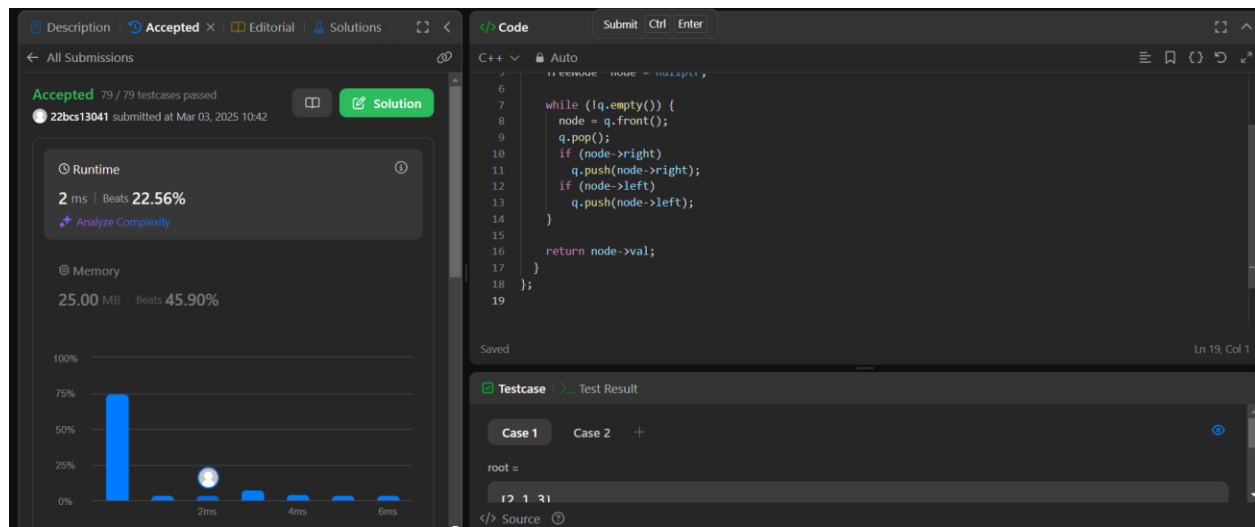


11. Find Bottom Left Tree Value

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue<TreeNode*> q{{root}};
        TreeNode* node = nullptr;

        while (!q.empty()) {
            node = q.front();
            q.pop();
            if (node->right)
                q.push(node->right);
            if (node->left)
                q.push(node->left);
        }

        return node->val;
    }
};
```



12. Binary Tree Maximum Path Sum

```
class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int ans = INT_MIN;
        maxPathSumDownFrom(root, ans);
        return ans;
    }
private:
    int maxPathSumDownFrom(TreeNode* root, int& ans) {
        if (root == nullptr)
            return 0;

        const int l = max(0, maxPathSumDownFrom(root->left, ans));
        const int r = max(0, maxPathSumDownFrom(root->right, ans));
        ans = max(ans, root->val + l + r);
        return root->val + max(l, r);
    }
};
```

The screenshot displays a coding platform interface. On the left, a sidebar indicates the solution is 'Accepted' with 96/96 testcases passed. It shows a runtime of 0ms (Beats 100.00%) and memory usage of 28.09 MB (Beats 25.54%). A bar chart shows the runtime distribution. The main editor contains C++ code for a binary tree problem. The right sidebar shows a test case with input 'root =' and output '1 2 3 1'.

13. Vertical Order Traversal of a Binary Tree

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        vector<vector<int>> ans;
        map<int, multiset<pair<int, int>>> xToSortedPairs;

        dfs(root, 0, 0, xToSortedPairs);
        for (const auto& [_ , pairs] : xToSortedPairs) {
            vector<int> vals;
            for (const pair<int, int>& pair : pairs)
                vals.push_back(pair.second);
            ans.push_back(vals);
        }
        return ans;
    }
private:
    void dfs(TreeNode* root, int x, int y,
              map<int, multiset<pair<int, int>>>& xToSortedPairs) {
        if (root == nullptr)
            return;
        xToSortedPairs[x].emplace(y, root->val);
        dfs(root->left, x - 1, y + 1, xToSortedPairs);
        dfs(root->right, x + 1, y + 1, xToSortedPairs);
    }
};
```

DescriptionAccepted x Editorial Solutions

All Submissions

Accepted 34 / 34 testcases passed
22bcs13041 submitted at Mar 03, 2025 10:44

Runtime

2 ms | Beats 51.53%

Analyze Complexity

Memory

16.56 MB | Beats 17.78%

Time (ms)	Percentage (%)
0-1	40
1-2	10
2-3	15
3-4	20
4-5	10
5-6	5

Code

Submit Ctrl Enter

```
12 ans.push_back(val);
13 }
14 return ans;
15 }
16 private:
17 void dfs(TreeNode* root, int x, int y,
18         map<int, multiset<pair<int, int>>& xToSortedPairs) {
19     if (root == nullptr)
20         return;
21     xToSortedPairs[x].emplace(y, root->val);
22     dfs(root->left, x - 1, y + 1, xToSortedPairs);
23     dfs(root->right, x + 1, y + 1, xToSortedPairs);
24 }
25 };
26
```

SavedLn 26, Col 1

Testcase

Test Result

Case 1Case 2Case 3+

root =

1 3 0 7 0 null null 15 71

</> Source