

1. <https://leetcode.com/problems/binary-tree-level-order-traversal/description/>

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

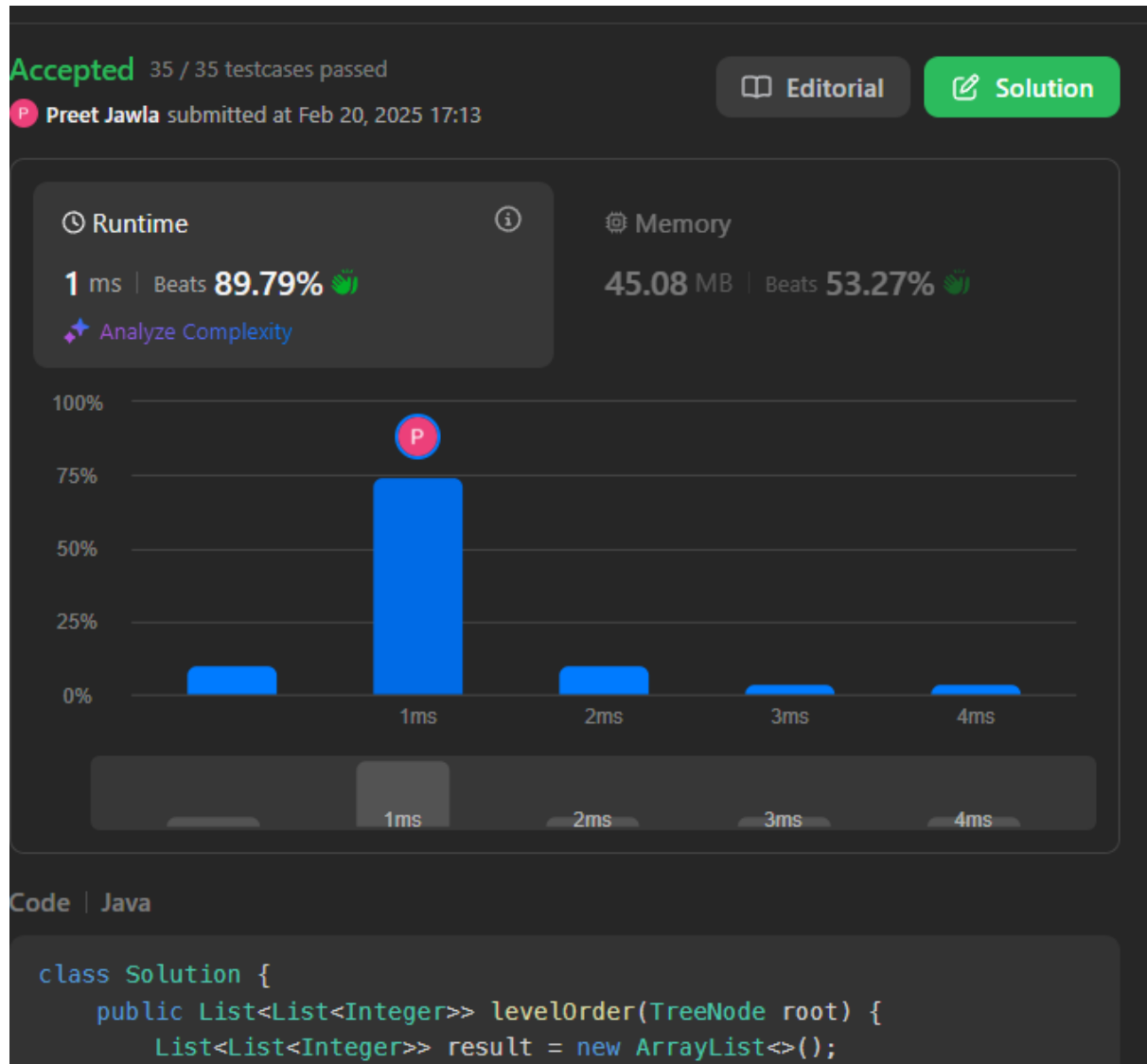
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> level = new ArrayList<>();

            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                level.add(node.val);

                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }

            result.add(level);
        }

        return result;
    }
}
```



2. <https://leetcode.com/problems/binary-tree-inorder-traversal/description/> (Iterative)

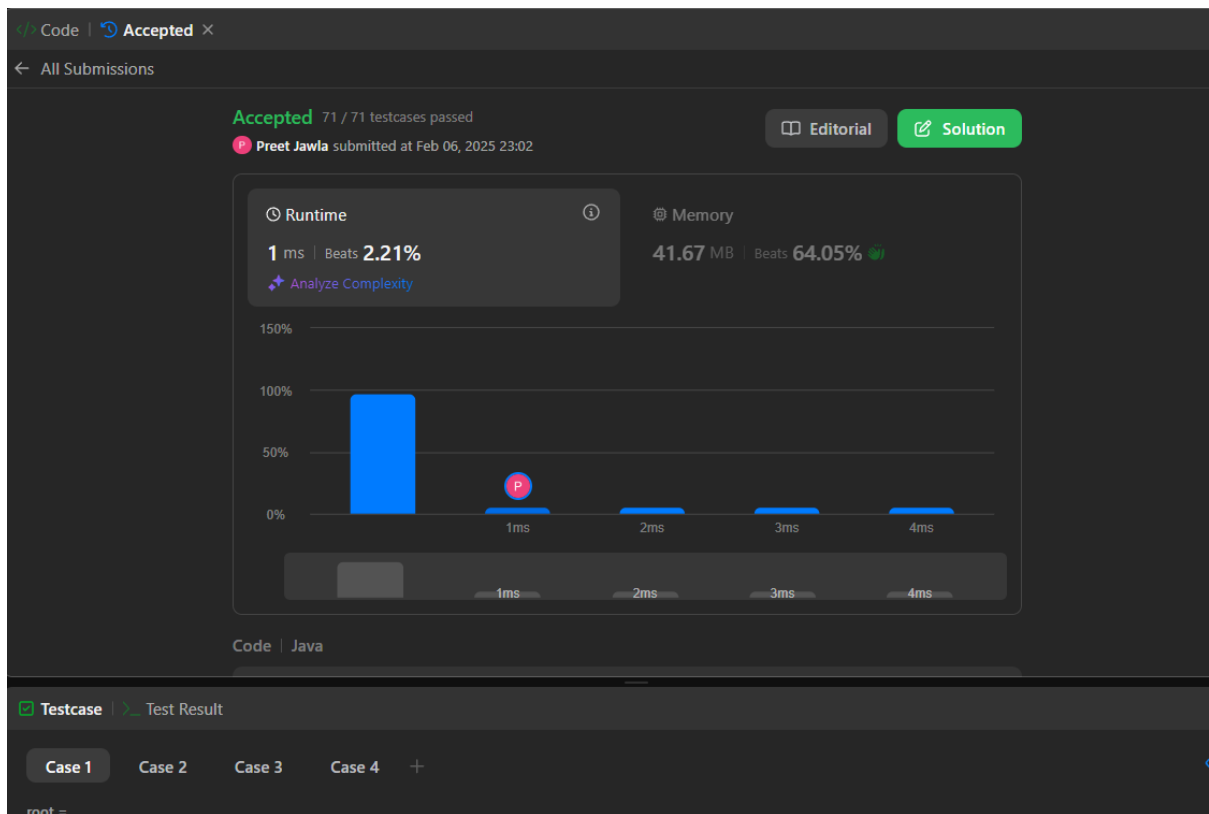
```
class Solution {  
    public List<Integer> inorderTraversal(TreeNode root) {  
        List<Integer> result = new ArrayList<>();  
        Stack<TreeNode> stack = new Stack<>();  
        TreeNode curr = root;  
  
        while (curr != null || !stack.isEmpty()) {  
            while (curr != null) {  
                stack.push(curr);  
            }  
        }  
    }  
}
```

```

        curr = curr.left;
    }
    curr = stack.pop();
    result.add(curr.val);
    curr = curr.right;
}

return result;
}
}

```



3. <https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/>

```

class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
    }
}

```

```

if (root == null) return result;

Queue<TreeNode> queue = new LinkedList<>();
queue.add(root);
boolean leftToRight = true;

while (!queue.isEmpty()) {
    int levelSize = queue.size();
    LinkedList<Integer> level = new LinkedList<>();

    for (int i = 0; i < levelSize; i++) {
        TreeNode node = queue.poll();

        if (leftToRight) {
            level.addLast(node.val);
        } else {
            level.addFirst(node.val);
        }

        if (node.left != null) queue.add(node.left);
        if (node.right != null) queue.add(node.right);
    }

    result.add(level);
    leftToRight = !leftToRight;
}

return result;
}
}

```

Description
Accepted
Editorial
Solutions
Submissions

All Submissions

Accepted 33 / 33 testcases passed
Preet Jawa submitted at Feb 20, 2025 17:15

Editorial
Solution

Runtime
0 ms | Beats 100.00%

Memory
42.72 MB | Beats 7.26%

Analyze Complexity

Code | Java

```

class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        boolean leftToRight = true;

        while (!queue.isEmpty()) {
            List<Integer> level = new ArrayList<>();
            int levelSize = queue.size();
            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                if (leftToRight) {
                    level.addLast(node.val);
                } else {
                    level.addFirst(node.val);
                }
                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }
            result.add(level);
            leftToRight = !leftToRight;
        }

        return result;
    }
}

```

Testcase
Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

- <https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/>

```

class Solution {
    private Map<Integer, Integer> inorderMap;
    private int preorderIndex;

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        inorderMap = new HashMap<>();
        preorderIndex = 0;

        for (int i = 0; i < inorder.length; i++) {
            inorderMap.put(inorder[i], i);
        }

        return constructTree(preorder, 0, inorder.length - 1);
    }
}

```

```

private TreeNode constructTree(int[] preorder, int left, int right) {
    if (left > right) return null;

    int rootValue = preorder[preorderIndex++];
    TreeNode root = new TreeNode(rootValue);

    int inorderIndex = inorderMap.get(rootValue);

    root.left = constructTree(preorder, left, inorderIndex - 1);
    root.right = constructTree(preorder, inorderIndex + 1, right);

    return root;
}
}

```

← All Submissions Java

Accepted 203 / 203 testcases passed

P Preet Jawla submitted at Feb 20, 2025 17:17

[Editorial](#) [Solution](#)

Runtime

2 ms | Beats 65.58%

[Analyze Complexity](#)

Memory

44.65 MB | Beats 31.53%

Runtime (ms)	Percentage (%)
1ms	~5%
2ms	~30%
3ms	~22%
4ms	~2%
5ms	~2%
6ms	~2%
7ms	~3%
8ms	~1%

Code | Java

```

class Solution {
    private Map<Integer, Integer> inorderMap;
    private int preorderIndex;

```

Saved

☒ Testcase

Accepted

5. <https://leetcode.com/problems/path-sum-ii/description/>

```
import java.util.*;
```

```
class TreeNode {
```

```
    int val;
```

```
    TreeNode left, right;
```

```
    TreeNode(int x) { val = x; }
```

```
}
```

```
class Solution {
```

```
    public List<List<Integer>> pathSum(TreeNode root, int targetSum) {
```

```
        List<List<Integer>> result = new ArrayList<>();
```

```
        findPaths(root, targetSum, new ArrayList<>(), result);
```

```
        return result;
```

```
    }
```

```
    private void findPaths(TreeNode node, int targetSum, List<Integer> path, List<List<Integer>> result)
```

```
{
```

```
        if (node == null) return;
```

```
        path.add(node.val);
```

```
        targetSum -= node.val;
```

```
        // If it's a leaf node and the sum matches, add the path to result
```

```
        if (node.left == null && node.right == null && targetSum == 0) {
```

```
            result.add(new ArrayList<>(path));
```

```
        } else {
```

```
            // Recur for left and right subtrees
```

```
            findPaths(node.left, targetSum, path, result);
```

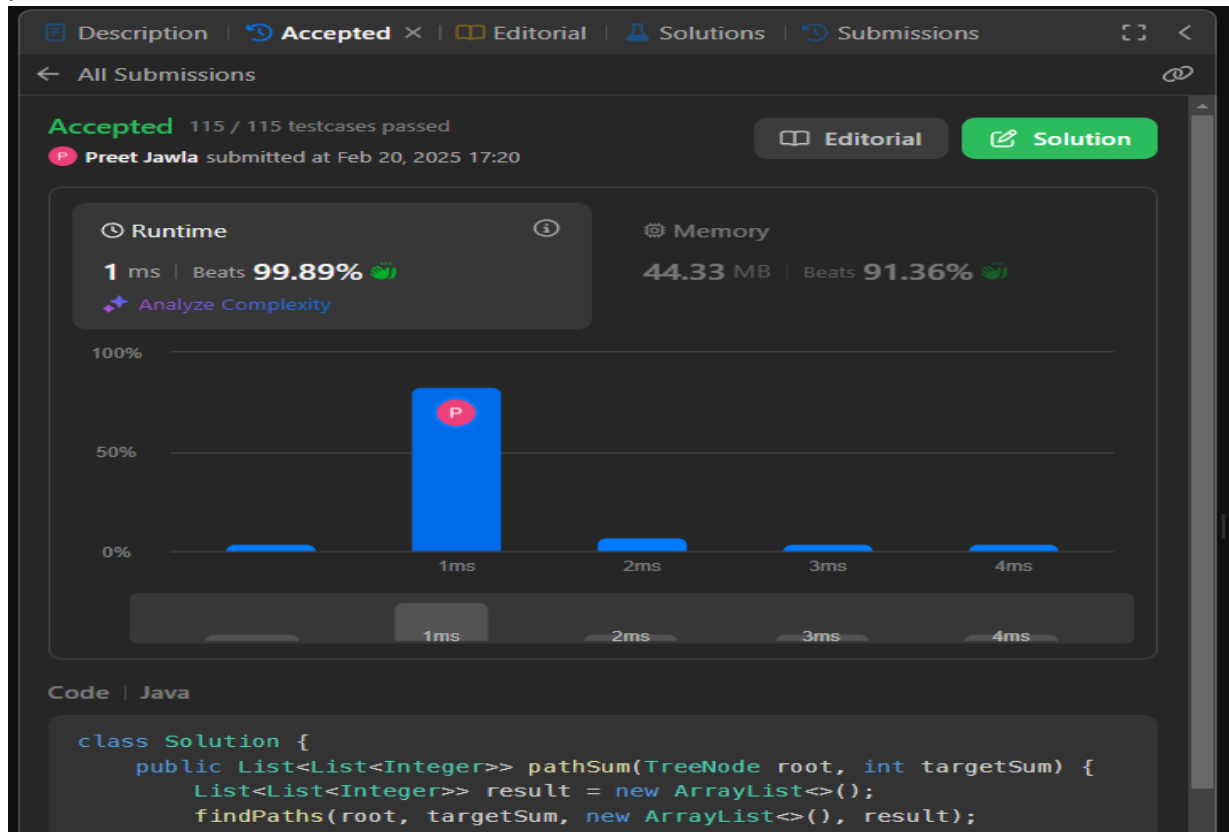
```
            findPaths(node.right, targetSum, path, result);
```

```
        }
```

```

// Backtrack: Remove the last added node before returning to the parent
path.remove(path.size() - 1);
}
}

```



6. <https://leetcode.com/problems/flatten-binary-tree-to-linked-list/description/>

```

class Solution {
    public void flatten(TreeNode root) {
        TreeNode curr = root;
        while (curr != null) {
            if (curr.left != null) {
                TreeNode prev = curr.left;
                while (prev.right != null) {
                    prev = prev.right;
                }
                prev.right = curr.right;
                curr.right = curr.left;
            }
            curr = curr.right;
        }
    }
}

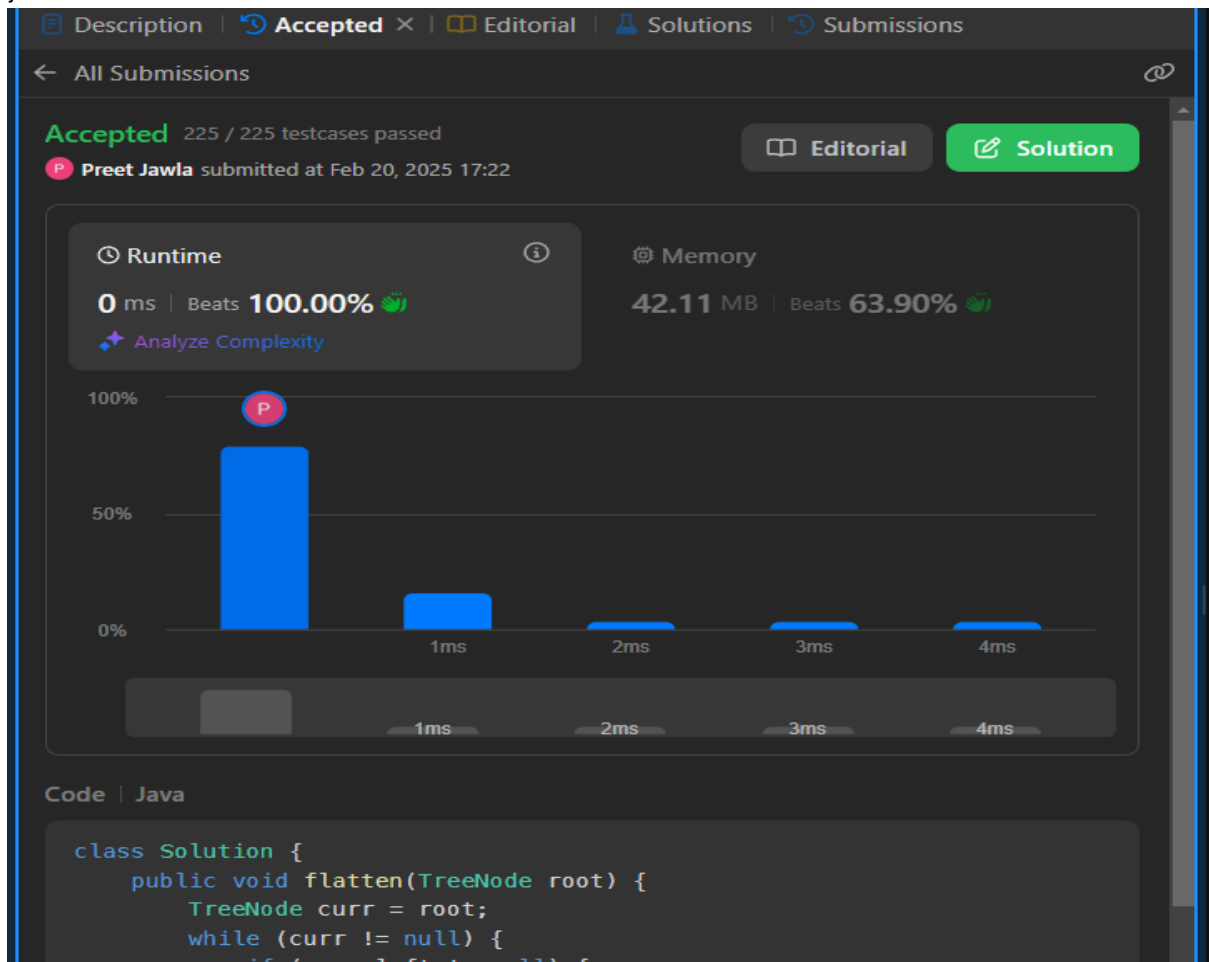
```



```

        curr.left = null;
    }
    curr = curr.right;
}
}
}

```



7. <https://leetcode.com/problems/binary-tree-preorder-traversal/description/> (iterative)

```

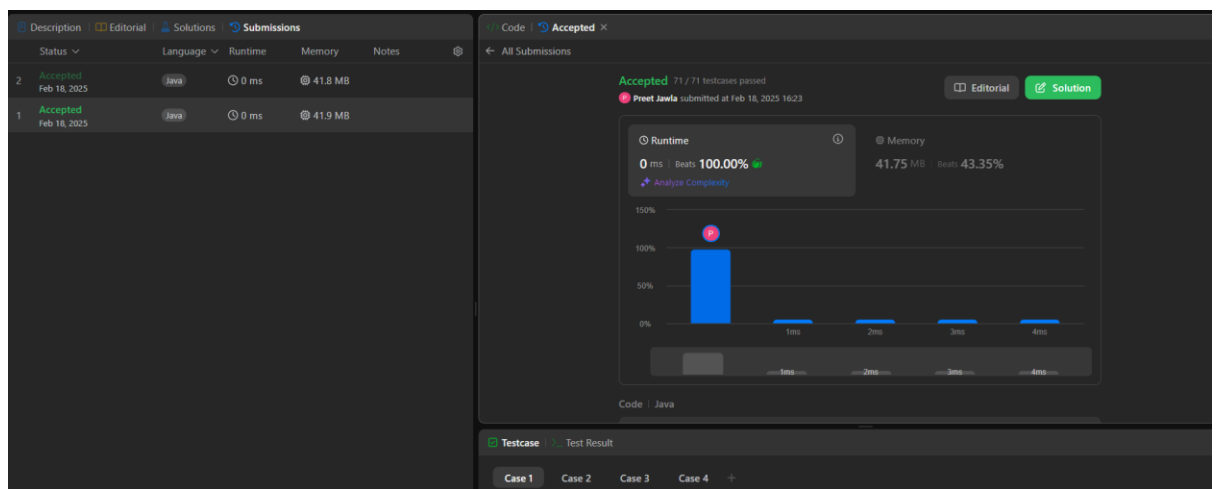
8. class Solution {
9.     public List<Integer> preorderTraversal(TreeNode root) {
10.         List<Integer> result = new ArrayList<>();
11.         preOrder(root, result);
12.         return result;
13.     }

```

```

14. private void preOrder(TreeNode node, List<Integer> result){
15.     if(node == null) return;
16.     result.add(node.val);
17.     preOrder(node.left, result);
18.     preOrder(node.right, result);
19. }
20.
21. }

```



22. <https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/>

```

class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null || root == p || root == q) return root;


        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);

        if (left != null && right != null) return root;
        return left != null ? left : right;
    }
}

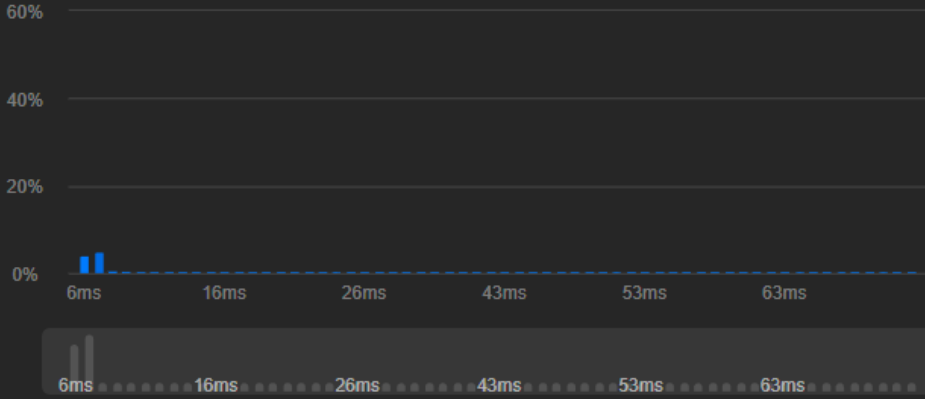
```

Accepted 32 / 32 testcases passed

Preet Jawa submitted at Feb 20, 2025 17:24

Runtime: 7 ms | Beats 63.51%  [Analyze Complexity](#)

Memory: 44.86 MB | Beats 46.66%



Code | Java

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeN
        if (root == null || root == p || root == q) return root;

    TreeNode left = lowestCommonAncestor(root.left, p, q);
    TreeNode right = lowestCommonAncestor(root.right, p, q);
}
```

23. <https://leetcode.com/problems/binary-tree-cameras/description/>

```
class Solution {
    private int cameras = 0;

    public int minCameraCover(TreeNode root) {
        return dfs(root) == 0 ? cameras + 1 : cameras;
    }

    private int dfs(TreeNode node) {
        if (node == null) return 2;

```

```

int left = dfs(node.left);

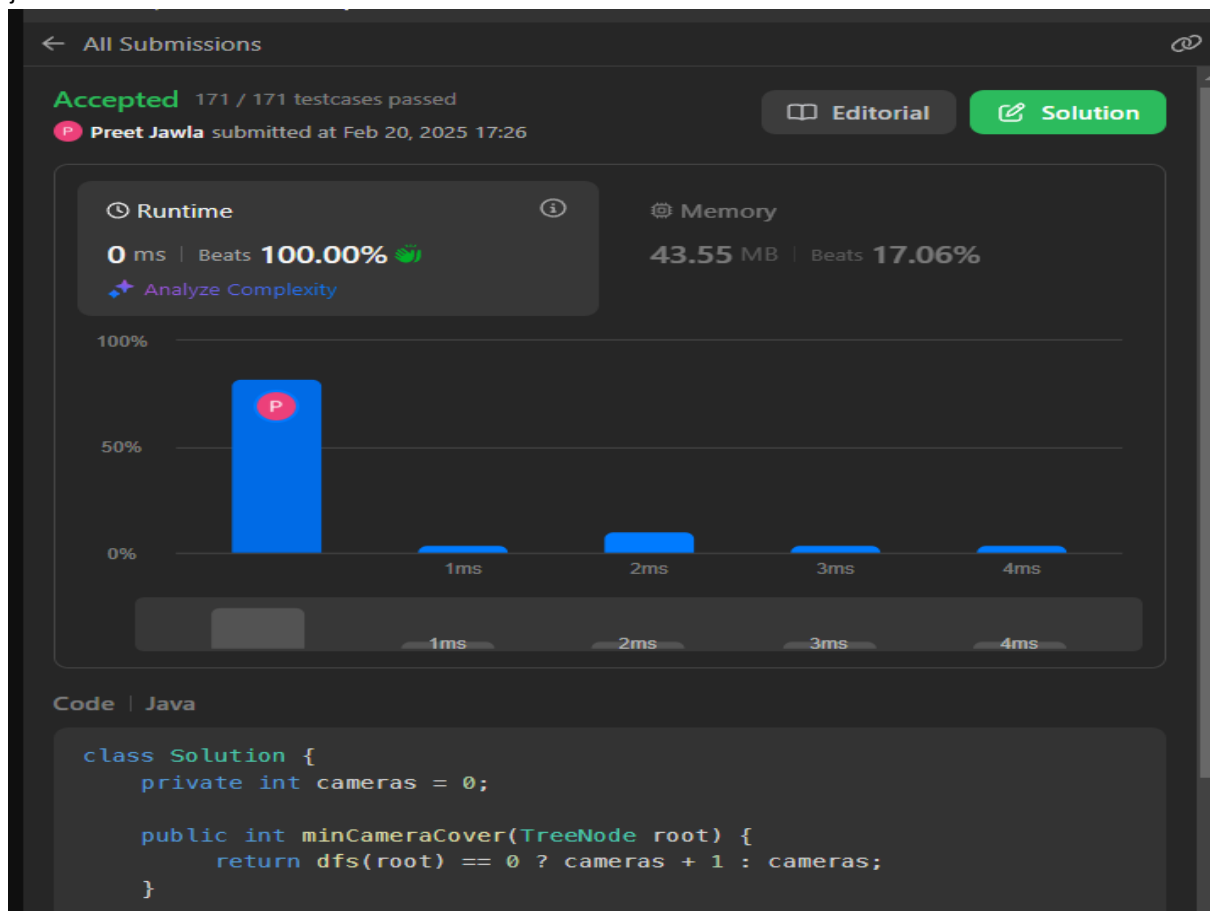
int right = dfs(node.right);


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

if (left == 1 || right == 1) {
    return 2;
}

return 0;
}
}

```



24. <https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/description/>

```
25. class Solution {
26.     public List<List<Integer>> verticalTraversal(TreeNode root) {
27.         TreeMap<Integer, List<int[]>> map = new TreeMap<>();
28.
29.         Queue<Tuple> queue = new LinkedList<>();
30.         queue.offer(new Tuple(root, 0, 0));
31.
32.         while (!queue.isEmpty()) {
33.             Tuple t = queue.poll();
34.             TreeNode node = t.node;
35.             int row = t.row, col = t.col;
36.
37.             map.putIfAbsent(col, new ArrayList<>());
38.             map.get(col).add(new int[]{row, node.val});
39.
40.             if (node.left != null) queue.offer(new Tuple(node.left, row + 1, col - 1));
41.             if (node.right != null) queue.offer(new Tuple(node.right, row + 1, col + 1));
42.         }
43.
44.         List<List<Integer>> result = new ArrayList<>();
45.
46.         for (List<int[]> nodes : map.values()) {
47.             nodes.sort((a, b) -> a[0] == b[0] ? a[1] - b[1] : a[0] - b[0]); // Sort by row, then value
48.             List<Integer> colList = new ArrayList<>();
49.             for (int[] node : nodes) colList.add(node[1]); // Extract values
50.             result.add(colList);
51.         }
52.
53.         return result;
}
```

```

54. }
55.
56. private static class Tuple {
57.     TreeNode node;
58.     int row, col;
59.     Tuple(TreeNode node, int row, int col) {
60.         this.node = node;
61.         this.row = row;
62.         this.col = col;
63.     }
64. }
65. }

```

The screenshot displays a LeetCode submission page for a problem involving binary tree traversal. The submission is marked as 'Accepted' and was submitted by 'Preet Java' on Feb 20, 2025, at 17:30. The runtime graph shows a peak memory usage of 42.57 MB and a runtime of 3 ms, achieving a 82.91% success rate. The code is written in Java and implements a vertical traversal of a binary tree using a queue and a TreeMap.

```

class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        TreeMap<Integer, List<Integer>> map = new TreeMap<>();
        Queue<Tuple> queue = new LinkedList<>();
        queue.offer(new Tuple(root, 0, 0));

        while (!queue.isEmpty()) {
            Tuple t = queue.poll();
            TreeNode node = t.node;
            int row = t.row, col = t.col;

            map.putIfAbsent(col, new ArrayList<>());
            map.get(col).add(new Integer(row, node.val));

            if (node.left != null) queue.offer(new Tuple(node.left, row + 1, col - 1));
            if (node.right != null) queue.offer(new Tuple(node.right, row + 1, col + 1));
        }

        List<List<Integer>> result = new ArrayList<>();
        for (List<Integer> nodes : map.values()) {
            nodes.sort((a, b) -> a[0] - b[0]);
            List<Integer> colList = new ArrayList<>();
            for (Integer[] node : nodes) colList.add(node[1]);
            result.add(colList);
        }
        return result;
    }
}

```

The code is saved and the test result is 'Accepted' with a runtime of 0 ms. The interface also shows tabs for 'Description', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'.