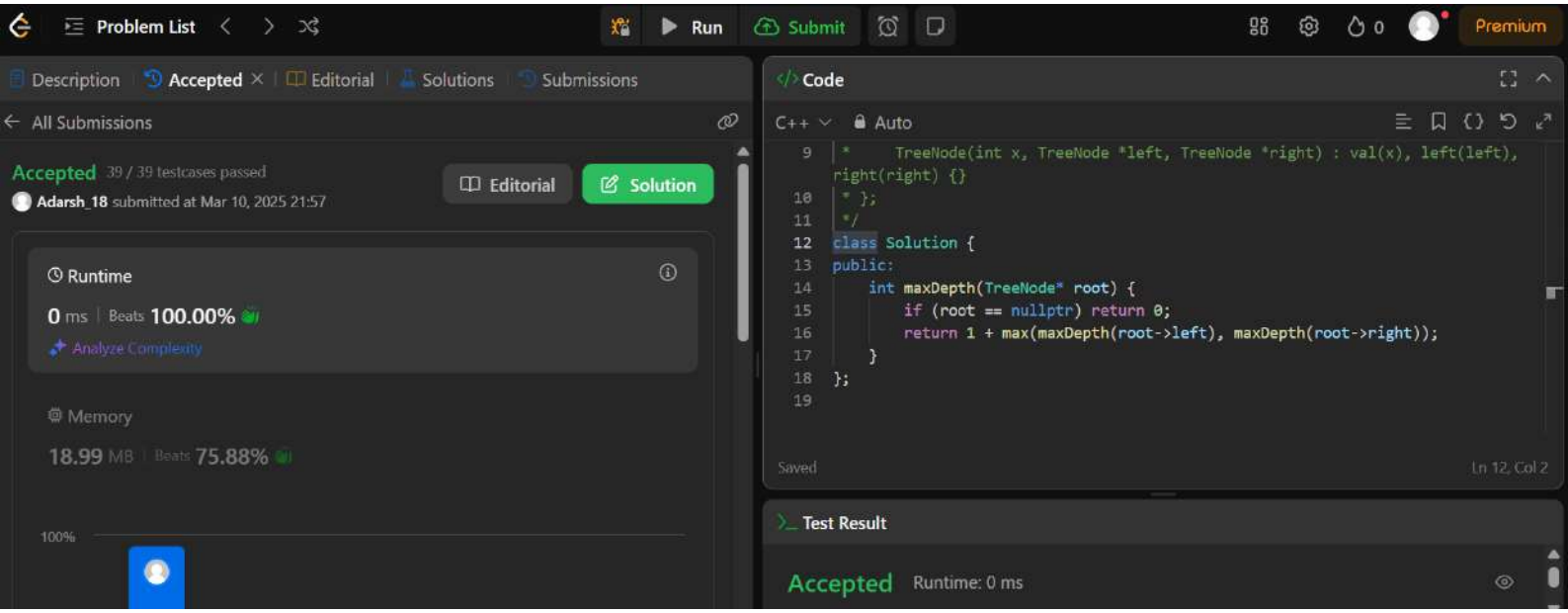

Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 39 / 39 testcases passed
Adarsh_18 submitted at Mar 10, 2025 21:57
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
18.99 MB | Beats 75.88%
100%
Code
C++
Auto
9 * TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 class Solution {
13 public:
14 int maxDepth(TreeNode* root) {
15 if (root == nullptr) return 0;
16 return 1 + max(maxDepth(root->left), maxDepth(root->right));
17 }
18 };
19
Saved
Ln 12, Col 2
Test Result
Accepted Runtime: 0 ms

Active Courses | Chrome Web Store... | {RECORDINGS} - DS... | LinkedIn Learning:... | SHL | Telegram | workattech | Problems - LeetCode | All Bookmarks

Problem List | Run | Submit | Premium

Description | Accepted | Editorial | Solutions | Submissions

98. Validate Binary Search Tree

Solved

Medium | Topics | Companies

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.

Example 1:

Code

C++ | Auto

```
10  * };
11  */
12 class Solution {
13 public:
14     bool isValidBST(TreeNode* root, long minVal = LONG_MIN, long maxVal =
    LONG_MAX) {
15         if (!root) return true;
16         if (root->val <= minVal || root->val >= maxVal) return false;
17         return isValidBST(root->left, minVal, root->val) && isValidBST
    (root->right, root->val, maxVal);
18     }
19 };
20
```

Saved | Ln 20, Col 1

Test Result

Accepted Runtime: 0 ms

Active Courses | Chrome Web Store... | {RECORDINGS} - DS... | LinkedIn Learning:... | SHL | Telegram | workattech | Problems - LeetCode | All Bookmarks

Problem List | Run | Submit | Premium

Description | Accepted | Editorial | Solutions | Submissions

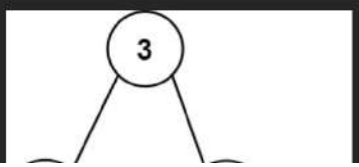
103. Binary Tree Zigzag Level Order Traversal

Solved

Medium | Topics | Companies

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).

Example 1:



Code

C++ | Auto

```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; // Flag to track traversal direction

        while (!q.empty()) {
            int size = q.size();
```

Saved | Ln 49, Col 1

Test Result

Accepted Runtime: 0 ms

Active Courses | Chrome Web Store... | {RECORDINGS} - DS... | LinkedIn Learning:... | SHL | Telegram | workattech | Problems - LeetCode | All Bookma

Problem List | Run | Submit | 0 | Premiu

Description | Accepted | Editorial | Solutions | Submissions

236. Lowest Common Ancestor of a Binary Tree

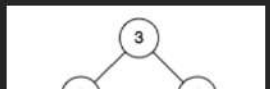
Solved

Medium | Topics | Companies

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:



Code

C++ | Auto

```
12      TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
13          if (!root || root == p || root == q) return root; // Base case
14
15          TreeNode* left = lowestCommonAncestor(root->left, p, q);
16          TreeNode* right = lowestCommonAncestor(root->right, p, q);
17
18          if (left && right) return root; // If p and q are found in left and
    right, root is LCA
19          return left ? left : right; // Else, return the non-null result
20      }
21  };
22
```

Saved | Ln 22, Co

Test Result

Accepted Runtime: 0 ms

Active Courses | Chrome Web Store... | {RECORDINGS} - DS... | LinkedIn Learning:... | SHL | Telegram | workattech | Problems - LeetCode | All Bookmarks

Problem List | Run | Submit | 0 | Premium

Description | Accepted | Editorial | Solutions | Submissions

94. Binary Tree Inorder Traversal

Solved

Easy | Topics | Companies

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

Example 1:

Input: `root = [1,null,2,3]`

Output: `[1,3,2]`

Explanation:

Code

C++ | Auto

```
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root, result);
        return result;
    }
};

```

Saved | Ln 27, Col 1

Test Result

Accepted Runtime: 0 ms

Active Courses | Chrome Web Store... | {RECORDINGS} - DS... | LinkedIn Learning:... | SHL | Telegram | workattech | Problems - LeetCode | All Bookmarks

Problem List | Run | Submit | Premium

Description | Accepted | Editorial | Solutions | Submissions

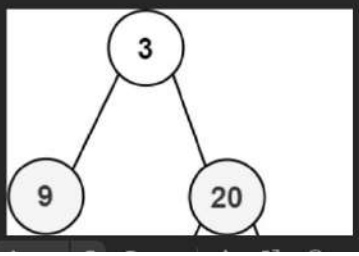
102. Binary Tree Level Order Traversal

Solved

Medium | Topics | Companies | Hint

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).

Example 1:



Code

C++ | Auto

```
31                 q.pop();
32                 level.push_back(node->val);
33
34                 if (node->left) q.push(node->left);
35                 if (node->right) q.push(node->right);
36             }
37             result.push_back(level);
38         }
39         return result;
40     }
41 };
42
```

Saved | Ln 42, Col 1

Test Result

Accepted Runtime: 0 ms