Description | Editorial | Solutions | Submissions

104. Maximum Depth of Binary Tree

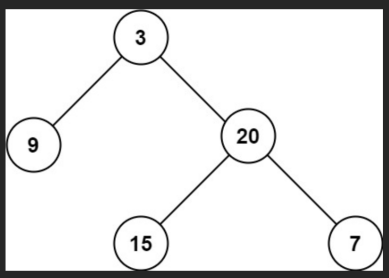
Solved

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.

Example 1:



Input: root = [3,9,20,null,null,15,7]

13.4K | 159 | 229 Online

Code | Accepted

C++ | Auto

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root == NULL) return 0;
        int maxLeft = maxDepth(root->left);
        int maxRight = maxDepth(root->right);
        return max(maxLeft , maxRight) + 1;
    }
};
```

Saved | Ln 20, Col 3

Testcase | Test Result

Description | Editorial | Solutions | Submissions

98. Validate Binary Search Tree

Solved

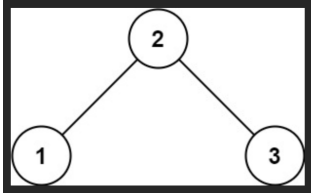
Medium | Topics | Companies

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 | Accepted

C++ | Auto

```cpp
 * };
 */
class Solution {

private:
    void inorder(TreeNode* root, vector<int> &a,set<int> &s){
        if(root == nullptr) return;

        inorder(root->left, a,s);
        a.push_back(root->val);
        s.insert(root->val);
        inorder(root->right, a,s);
    }
public:
    bool isValidBST(TreeNode* root) {
        vector<int> a;
        set<int> s;
        inorder(root, a,s);
        if(a.size() == 1) return true;
        if(s.size() != a.size()) return false;
        if(a[0] == a[a.size() - 1]) return false;
        return is_sorted(a.begin(), a.end());
    }
};
```

Saved | Ln 33, Col 3

Description | Editorial | Solutions | Submissions

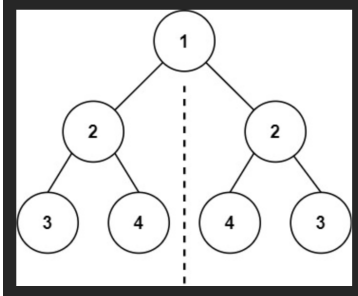
101. Symmetric Tree

Solved

Easy | Topics | Companies

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

Example 1:



Input: root = [1,2,2,3,4,4,3]
Output: true

15.9K | 197 | 130 Online

Code | Accepted

C++ | Auto

```cpp
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return isCheck(root->left , root->right);
    }
    bool isCheck(TreeNode* leftNode , TreeNode* rightNode){
        if(leftNode == NULL && rightNode == NULL) return true;
        if (leftNode == NULL || rightNode == NULL) return false;
        if (leftNode->val != rightNode->val) return false;
        return isCheck(leftNode->left, rightNode->right) &&
           isCheck(leftNode->right, rightNode->left);

    }

};
```

Saved | Ln 26, Col 3

Testcase | Test Result

Description | Editorial | Solutions | Submissions

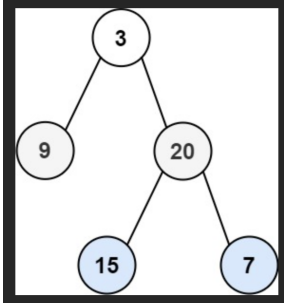
103. Binary Tree Zigzag Level Order Traversal

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:




```
Input: root = [3,9,20,null,null,15,7]
```

11.3K | 136 | 89 Online

Code | Accepted

C++ | Auto

```cpp
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> ans;
    if (root == NULL) return ans;

    queue<TreeNode*> q;
    q.push(root);
    bool leftToRight = true;

    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 != NULL) q.push(node->left);
            if (node->right != NULL) q.push(node->right);
        }
        if (!leftToRight) {
            reverse(level.begin(), level.end());
        }

        ans.push_back(level);
```

Saved | Ln 55, Col 3

Testcase | Test Result

Problem List | Run | Submit | 38 | Premium

Description | Editorial | Solutions | Submissions

236. Lowest Common Ancestor of a Binary Tree

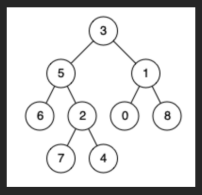
Solved

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:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
```

17.3K | 116 | 238 Online

Code | Accepted

C++ | Auto

```cpp
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p,
TreeNode* q) {
        if (!root || root == p || root == q) return root;

        TreeNode* leftLCA = lowestCommonAncestor(root->left, p, q);
        TreeNode* rightLCA = lowestCommonAncestor(root->right, p, q);

        if (leftLCA && rightLCA) return root;
        return leftLCA ? leftLCA : rightLCA;
    }
};
```

Saved | Ln 12, Col 3

Testcase | Test Result

Problem List | Run | Submit | 38 | Premium

Description | Editorial | Solutions | Submissions

94. Binary Tree Inorder Traversal

Solved

Easy | Topics | Companies

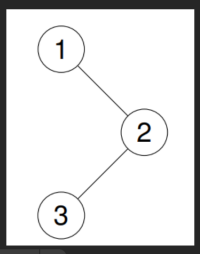
Given the `root` of a binary tree, return *the inorder traversal of its nodes' values*.

Example 1:

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

Output: `[1,3,2]`

Explanation:



14K | 190 | 178 Online

Code | Accepted

C++ | Auto

```cpp
*/
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        if(!root) return ans;
        TreeNode* node = root;
        stack<TreeNode*> st;
        while(1){
            if(node != NULL){
                st.push(node);
                node = node->left;
            }
            else{
                if(st.empty())break;
                node = st.top();
                st.pop();
                ans.push_back(node->val);
                node = node->right;
            }
        }

        return ans;
    }
};
```

Saved | Ln 35, Col 3

Testcase | Test Result

Problem List | Run | Submit | 38 | Premium

Description | Editorial | Solutions | Submissions

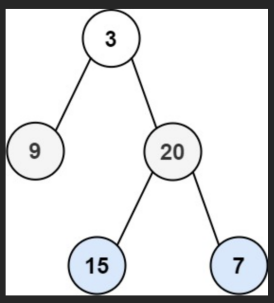
102. Binary Tree Level Order Traversal

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:



Input: `root = [3,9,20,null,null,15,7]`

16K | 123 | 151 Online

Code | Accepted

C++ | Auto

```cpp
* };
*/
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root == NULL)return ans;
        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);
            }
            ans.push_back(level);
        }
        return ans;
    }
};
```

Saved | Ln 33, Col 3

Testcase | Test Result

Problem List | Run | Submit | 38 | Premium

Description | Editorial | Solutions | Submissions

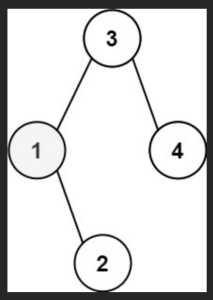
230. Kth Smallest Element in a BST

Solved

Medium | Topics | Companies | Hint

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.*

Example 1:




```
Input: root = [3,1,4,null,2], k = 1
```

11.9K | 122 | 129 Online

Code | Accepted

All Submissions

Accepted 93 / 93 testcases passed

siddharth46 submitted at Mar 10, 2025 19:12

Editorial | Solution

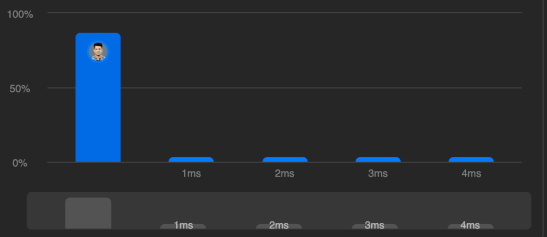
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

24.60 MB | Beats 18.34%



Code | C++

```
/**
 * Definition for a binary tree node.
```

Testcase | Test Result

Problem List | Run | Submit | 38 | Premium

Description | Accepted | Editorial | Solutions | Submissions

116. Populating Next Right Pointers in Each Node

Solved

Medium | Topics | Companies

You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

Example 1:

10K | 73 | 53 Online

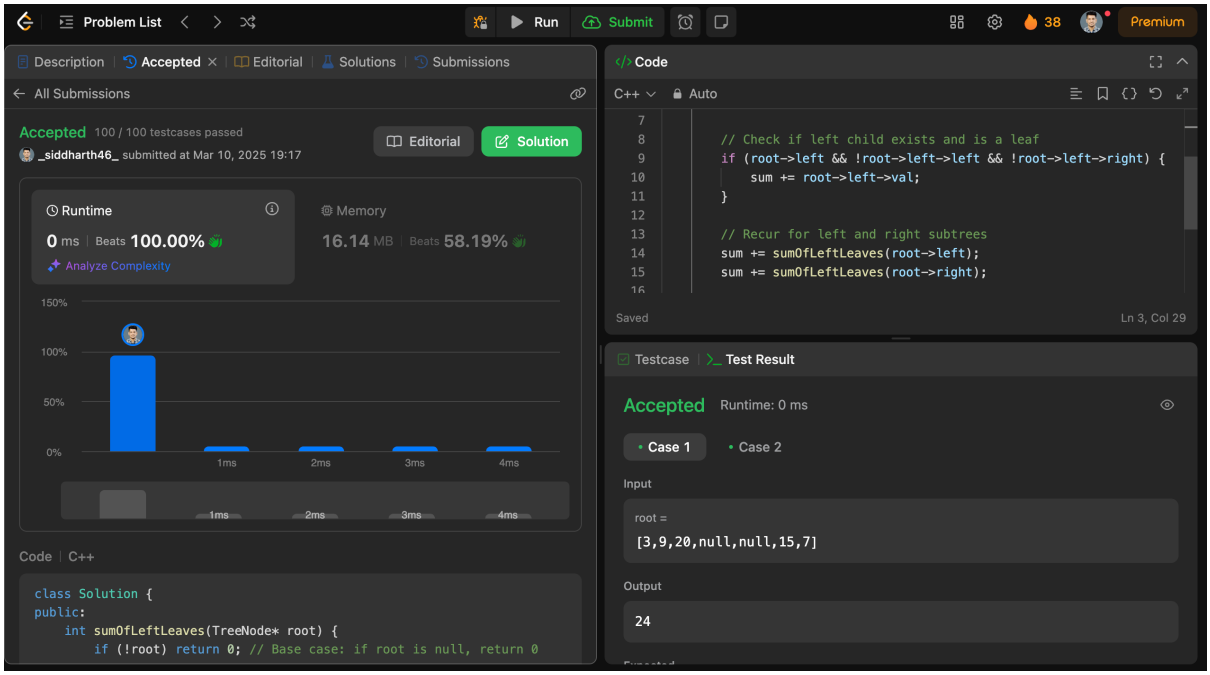
Code

C++ | Auto

```
level
            Node* current = leftmost;

            while (current) {
                // Connect left child to right child
                current->left->next = current->right;

                // Connect right child to the next node's left child
                if (current->next) {
                    current->right->next = current->next->left;
                }

                // Move to the next node in the same level
                current = current->next;
            }

            // Move to the next level
            leftmost = leftmost->left;
        }

        return root;
    }
};
```

Saved | Ln 31, Col 1

Testcase | Test Result


Problem List
Run
Submit
38
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 100 / 100 testcases passed
siddharth46 submitted at Mar 10, 2025 19:17
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
16.14 MB | Beats 58.19%
150%
100%
50%
0%
1ms
2ms
3ms
4ms
Code | C++
class Solution {
public:
int sumOfLeftLeaves(TreeNode* root) {
if (!root) return 0; // Base case: if root is null, return 0
Code
C++
Auto
// Check if left child exists and is a leaf
if (root->left && !root->left->left && !root->left->right) {
sum += root->left->val;
}
// Recur for left and right subtrees
sum += sumOfLeftLeaves(root->left);
sum += sumOfLeftLeaves(root->right);
Saved
Ln 3, Col 29
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[3,9,20,null,null,15,7]
Output
24