

Experiment 5:

Student Name: Shashwat

Branch: BE-CSE

Semester: 6

Subject Name: Advanced Programming Lab-2

UID: 22BCS15112

Section/Group: 637-B

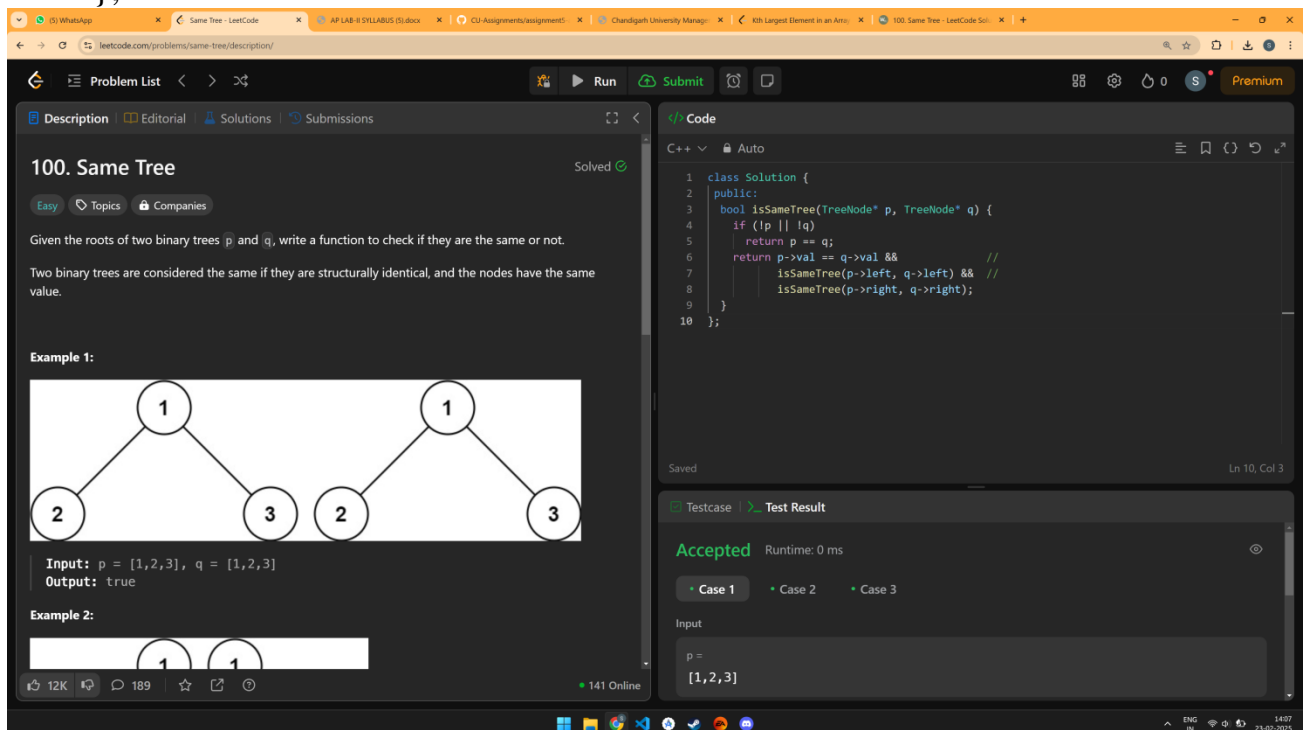
Date of Performance: 14/02/25

Subject Code: 22CSP-351

Aim(a) : Given the roots of two binary trees **p** and **q**, write a function to check if they are the same or not. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value

1. Code:

```
class Solution {  
public:  
    bool isSameTree(TreeNode* p, TreeNode* q) {  
        if (!p || !q)  
            return p == q;  
        return p->val == q->val &&          //  
            isSameTree(p->left, q->left) && //  
            isSameTree(p->right, q->right);  
    }  
};
```



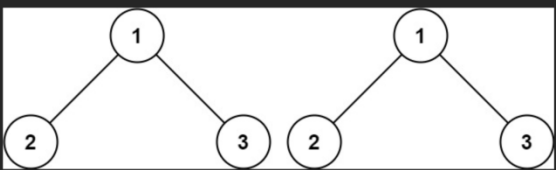
100. Same Tree Solved

Easy Topics Companies

Given the roots of two binary trees **p** and **q**, write a function to check if they are the same or not.


Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:



Input: **p** = [1,2,3], **q** = [1,2,3]
Output: true

Example 2:



12K 189 141 Online

Code

```
C++  
1 class Solution {  
2 public:  
3     bool isSameTree(TreeNode* p, TreeNode* q) {  
4         if (!p || !q)  
5             return p == q;  
6         return p->val == q->val &&          //  
            isSameTree(p->left, q->left) && //  
            isSameTree(p->right, q->right);  
7     }  
8 }  
9  
10 };
```

Saved Ln 10, Col 3

Testcase **Test Result**

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

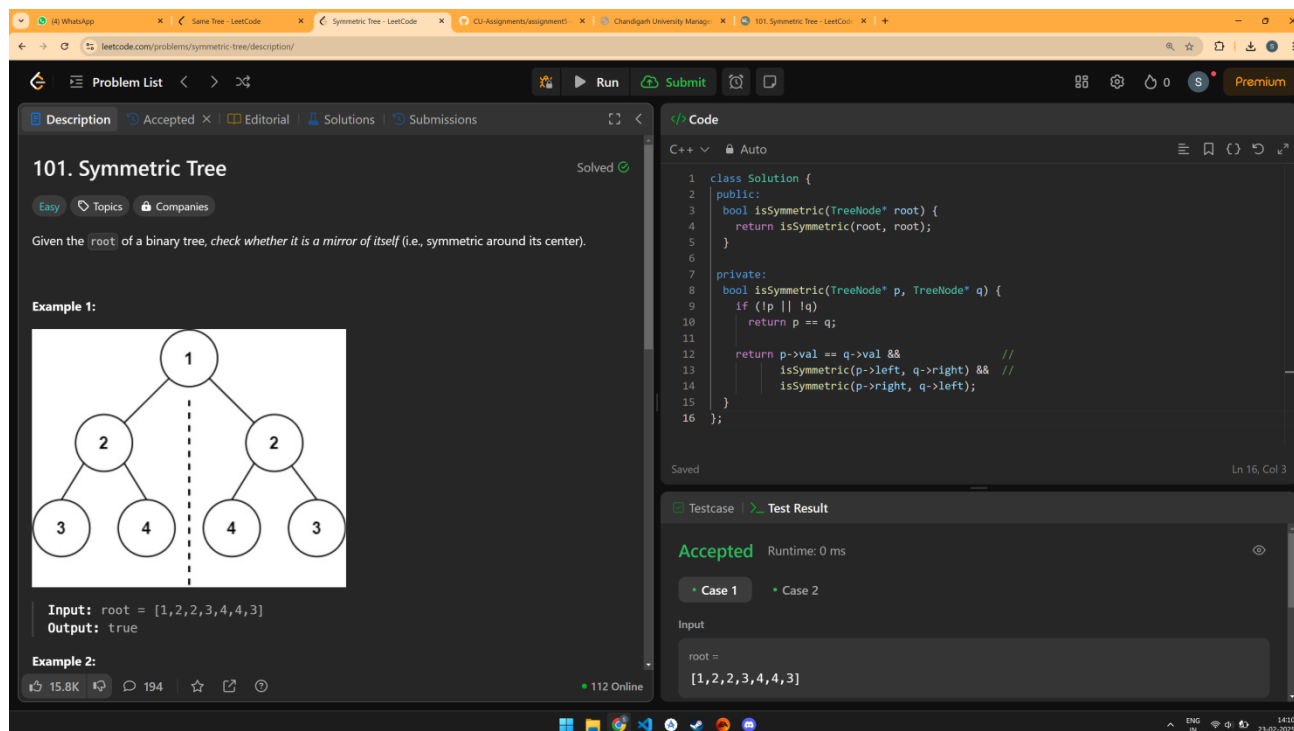
p = [1,2,3]

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

Code:

```
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);
    }
};
```



The screenshot shows a web browser with multiple tabs. The active tab is '101. Symmetric Tree - LeetCode'. The page displays the problem description, an example of a symmetric tree, and a C++ solution. The tree diagram shows a root node 1 with two children 2, which in turn have children 3 and 4 respectively, forming a mirror image. The input is [1,2,2,3,4,4,3] and the output is true. The C++ code is as follows:

```
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);
    }
};
```

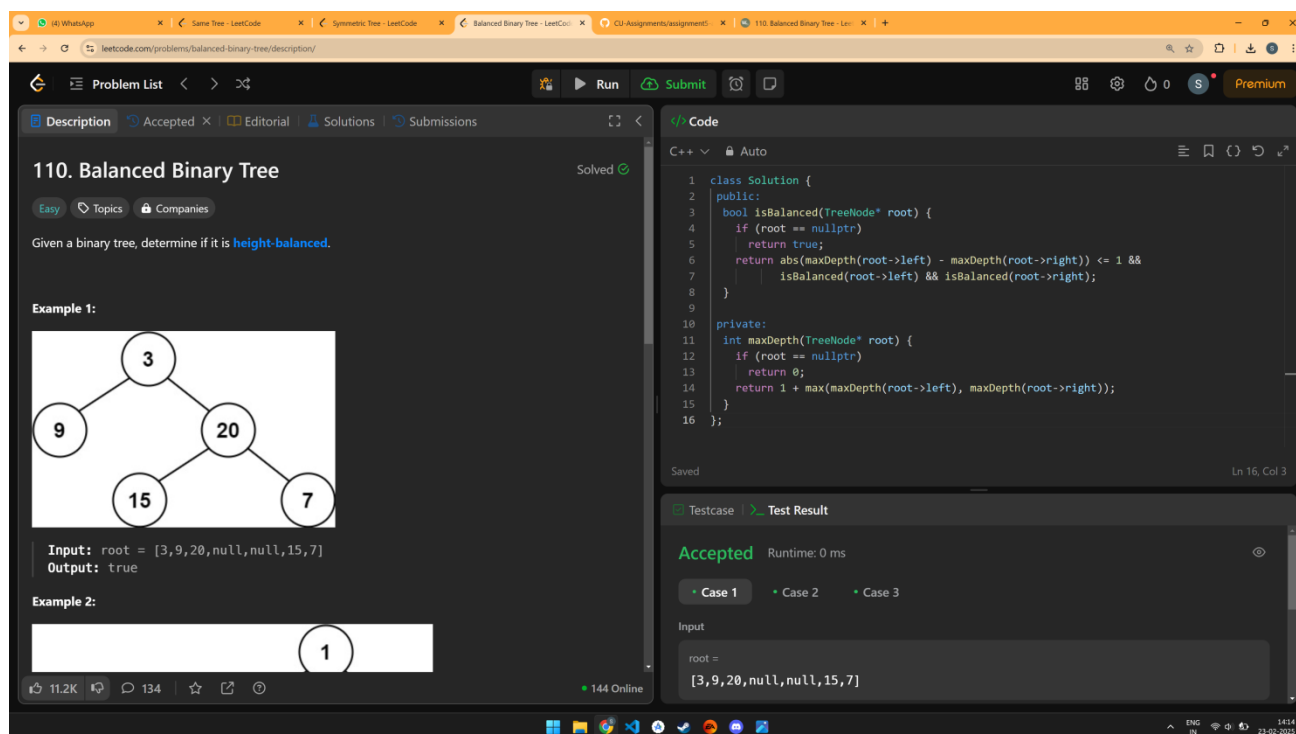
The test result shows 'Accepted' with a runtime of 0 ms. The input for the test case is [1,2,2,3,4,4,3].

Aim© : Given a binary tree, determine if it is height-balanced.

Code:

```
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        if (root == nullptr)
            return true;
        return abs(maxDepth(root->left) - maxDepth(root->right)) <= 1 &&
            isBalanced(root->left) && isBalanced(root->right);
    }

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

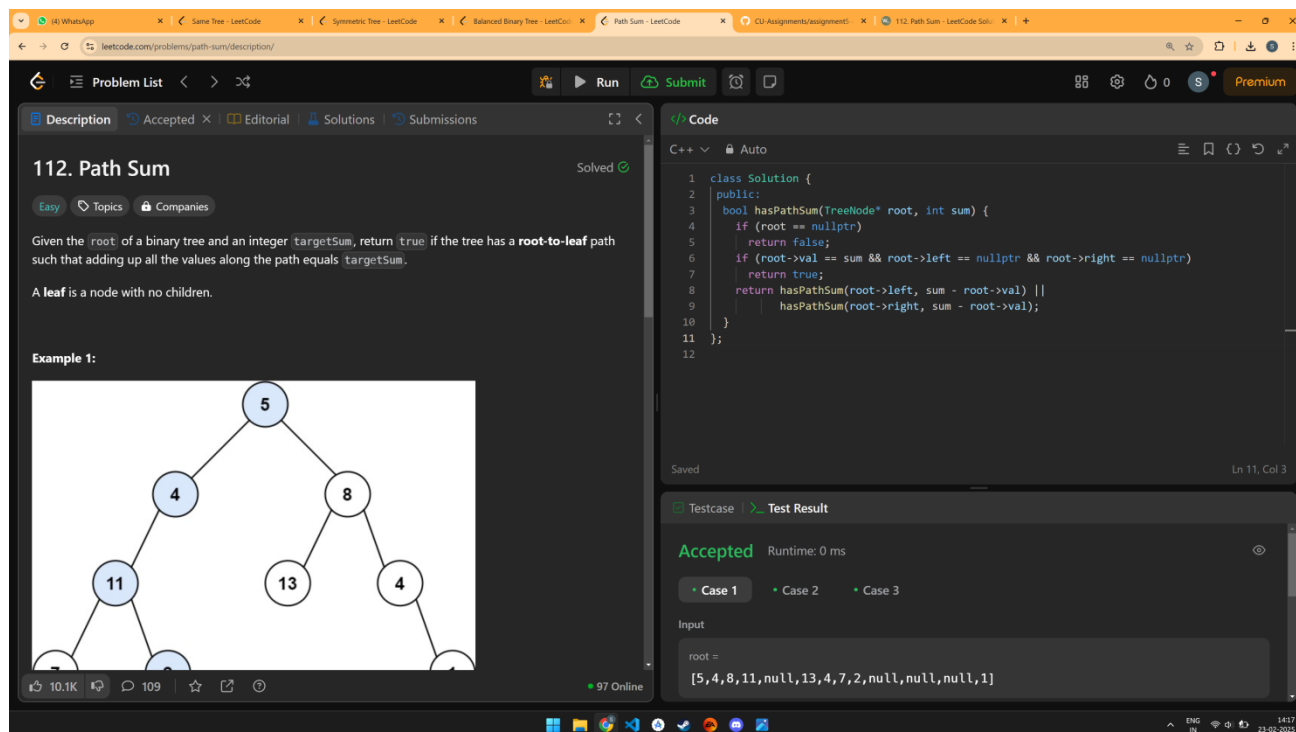


The screenshot shows a web browser with multiple tabs. The active tab is 'leetcode.com/problems/balanced-binary-tree/description/'. The page displays the problem '110. Balanced Binary Tree' with a 'Solved' status. The problem description states: 'Given a binary tree, determine if it is height-balanced.' Example 1 shows a binary tree with root 3, left child 9, right child 20, 9's left child 15, and 20's right child 7. The input is 'root = [3,9,20,null,null,15,7]' and the output is 'true'. Example 2 shows a single node 1. The C++ code is shown in the 'Code' editor, matching the code provided in the previous block. The 'Testcase' section shows 'Accepted' with a runtime of 0 ms. The 'Test Result' section shows 'Case 1' selected, with input 'root = [3,9,20,null,null,15,7]'.

Aim(d): Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum. A leaf is a node with no children.

Code:

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if (root == nullptr)
            return false;
        if (root->val == sum && root->left == nullptr && root->right == nullptr)
            return true;
        return hasPathSum(root->left, sum - root->val) ||
            hasPathSum(root->right, sum - root->val);
    }
};
```



The screenshot displays the LeetCode interface for the problem "112. Path Sum". The problem description states: "Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum. A leaf is a node with no children." An example tree is shown with root 5, left child 4, right child 8, and further children 11, 13, and 4. The solution code in C++ is provided, implementing a recursive function hasPathSum. The code is as follows:

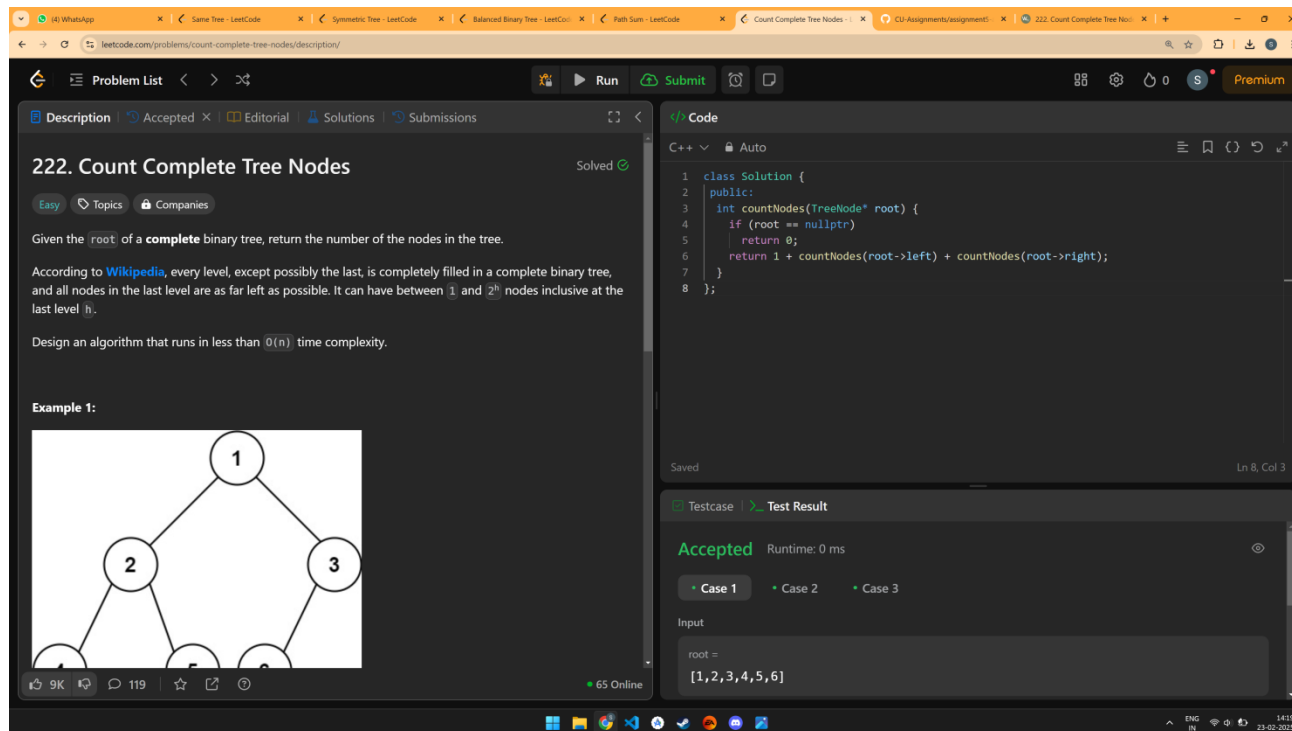
```
1 class Solution {
2 public:
3     bool hasPathSum(TreeNode* root, int sum) {
4         if (root == nullptr)
5             return false;
6         if (root->val == sum && root->left == nullptr && root->right == nullptr)
7             return true;
8         return hasPathSum(root->left, sum - root->val) ||
9             hasPathSum(root->right, sum - root->val);
10    }
11 };
12
```

The test result shows "Accepted" with a runtime of 0 ms. The input for the test case is: root = [5,4,8,11,null,13,4,7,2,null,null,null,1].

Aim(e): Given the root of a complete binary tree, return the number of the nodes in the tree. According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h . Design an algorithm that runs in less than $O(n)$ time complexity.

Code:

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



The screenshot displays a web browser window with the LeetCode problem '222. Count Complete Tree Nodes' open. The problem description states: "Given the root of a complete binary tree, return the number of the nodes in the tree. According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h . Design an algorithm that runs in less than $O(n)$ time complexity." An example tree diagram is shown with root 1, children 2 and 3, and further children 4, 5, 6, 7.

The code editor on the right shows the following C++ solution:

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

The test result section shows 'Accepted' with a runtime of 0 ms. The input is 'root = [1,2,3,4,5,6]'.

Aim(f): Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST. Basically, the deletion can be divided into two stages: Search for a node to remove. If the node is found, delete the node.

Code:

```
class Solution {
public:
    TreeNode* deleteNode(TreeNode* root, int key) {
        if (root == nullptr)
            return nullptr;
        if (root->val == key) {
            if (root->left == nullptr)
                return root->right;
            if (root->right == nullptr)
                return root->left;
            TreeNode* minNode = getMin(root->right);
            root->right = deleteNode(root->right, minNode->val);
            minNode->left = root->left;
            minNode->right = root->right;
            root = minNode;
        } else if (root->val < key) {
            root->right = deleteNode(root->right, key);
        } else { // root->val > key
            root->left = deleteNode(root->left, key);
        }
        return root;
    }

private:
    TreeNode* getMin(TreeNode* node) {
        while (node->left != nullptr)
            node = node->left;
        return node;
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

};

450. Delete Node in a BST

Medium

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the **root node reference** (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Example 1:

Input: root = [5,3,6,2,4,null,7], key = 3
Output: [5,4,6,2,null,null,7]
Explanation: Given key to delete is 3. So we find the node with value 3 and

```
private:
TreeNode* getMin(TreeNode* node) {
    while (node->left != nullptr) {
        node = node->left;
    }
    return node;
}
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

root = [5,3,6,2,4,null,7]

Aim(g): Given the root of a binary tree, return the length of the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root. The length of a path between two nodes is represented by the number of edges between them

Code:

```
class Solution {
public:
    int diameterOfBinaryTree(TreeNode* root) {
        int ans = 0;
        maxDepth(root, ans);
        return ans;
    }

private:
    int maxDepth(TreeNode* root, int& ans) {
        if (root == nullptr)
            return 0;

        const int l = maxDepth(root->left, ans);
        const int r = maxDepth(root->right, ans);
        ans = max(ans, l + r);
        return 1 + max(l, r);
    }
};
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

543. Diameter of Binary Tree

Easy Topics Companies

Given the `root` of a binary tree, return the *length of the diameter* of the tree.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.

The **length** of a path between two nodes is represented by the number of edges between them.

Example 1:

```
graph TD
    1((1)) --- 2((2))
    1 --- 3((3))
    2 --- 4((4))
    2 --- 5((5))
```

14.5K 289 214 Online

Code

```
C++ Auto
1 class Solution {
2 public:
3     int diameterOfBinaryTree(TreeNode* root) {
4         int ans = 0;
5         maxDepth(root, ans);
6         return ans;
7     }
8
9 private:
10    int maxDepth(TreeNode* root, int& ans) {
11        if (root == nullptr) {
12            return 0;
13        }
14        const int l = maxDepth(root->left, ans);
15        const int r = maxDepth(root->right, ans);
16        ans = max(ans, 1 + r);
17        return 1 + max(l, r);
18    }
19 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =

[1,2,3,4,5]