

Experiment 6

Student Name: shubhang
Branch: BE-IT
Semester: 6
Subject Name: AP LAB-II

UID:22BET10325
Section/Group: IOT-702(A)
Date of Performance:06/03/25
Subject Code: 22ITP-351

PROBLEM 1:

Aim:

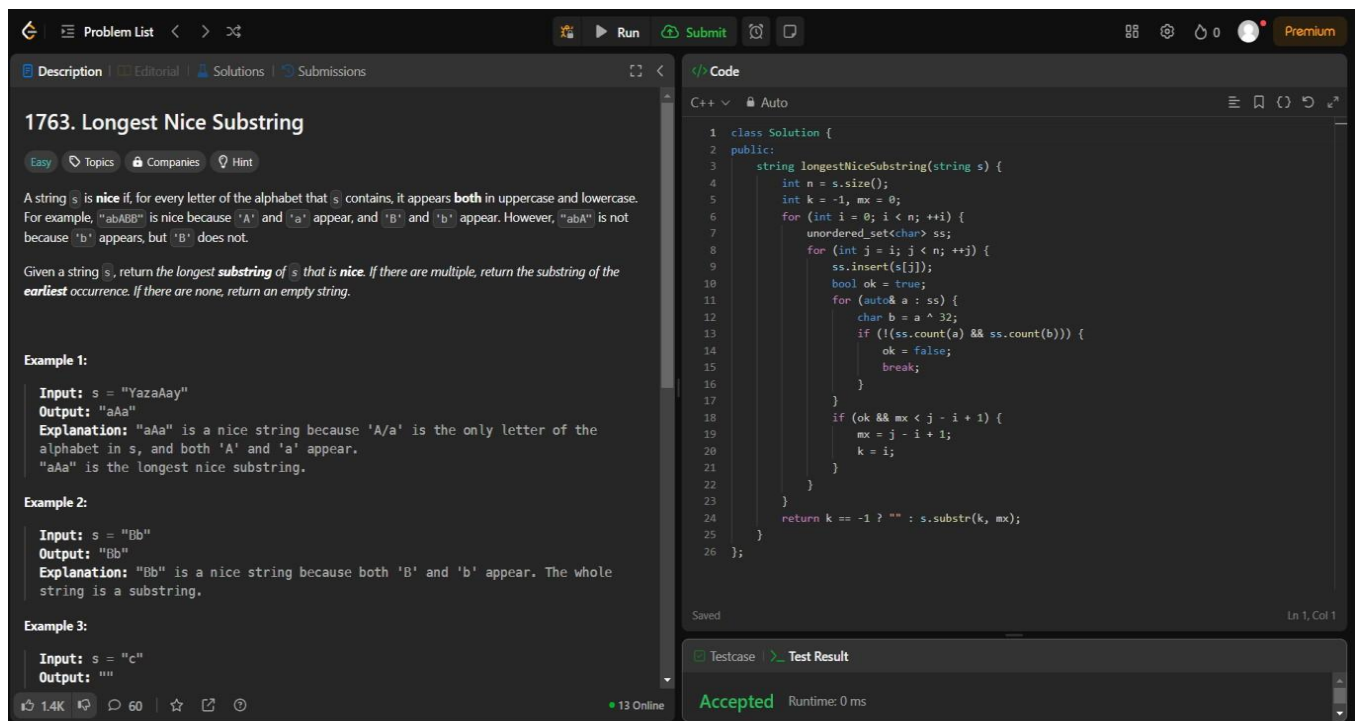
Given the root of a binary tree, return *its maximum depth*.

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.

Code:

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

Output:



1763. Longest Nice Substring

Easy Topics Companies Hint

A string *s* is **nice** if, for every letter of the alphabet that *s* contains, it appears **both** in uppercase and lowercase. For example, "aBb" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "aBbA" is not because 'b' appears, but 'B' does not.

Given a string *s*, return the **longest substring** of *s* that is **nice**. If there are multiple, return the substring of the **earliest** occurrence. If there are none, return an empty string.

Example 1:

Input: *s* = "YazaAay"
Output: "aAa"
Explanation: "aAa" is a nice string because 'A/a' is the only letter of the alphabet in *s*, and both 'A' and 'a' appear. "aAa" is the longest nice substring.

Example 2:

Input: *s* = "Bb"
Output: "Bb"
Explanation: "Bb" is a nice string because both 'B' and 'b' appear. The whole string is a substring.

Example 3:

Input: *s* = "c"
Output: ""

```
1 class Solution {
2 public:
3     string longestNiceSubstring(string s) {
4         int n = s.size();
5         int k = -1, mx = 0;
6         for (int i = 0; i < n; ++i) {
7             unordered_set<char> ss;
8             for (int j = i; j < n; ++j) {
9                 ss.insert(s[j]);
10                bool ok = true;
11                for (auto& a : ss) {
12                    char b = a ^ 32;
13                    if (!ss.count(a) && ss.count(b)) {
14                        ok = false;
15                        break;
16                    }
17                }
18                if (ok && mx < j - i + 1) {
19                    mx = j - i + 1;
20                    k = i;
21                }
22            }
23        }
24        return k == -1 ? "" : s.substr(k, mx);
25    }
26 };
```

Accepted Runtime: 0 ms



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.



PROBLEM 2:

Aim: 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.

Code:

```
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);
    }
};
```

Output:

Description
Editorial
Solutions
Submissions

98. Validate Binary Search Tree

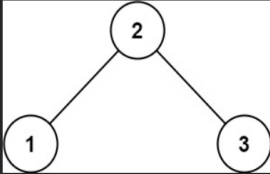
Medium Topics Companies

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:



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

Example 2:

Code

```

1 class Solution {
2 public:
3     bool isValidBST(TreeNode* root) {
4         return isValidBST(root, nullptr, nullptr);
5     }
6
7 private:
8     bool isValidBST(TreeNode* root, TreeNode* minNode, TreeNode* maxNode) {
9         if (root == nullptr)
10            return true;

```

Saved Ln 2, Col 9

Testcase

Test Result


Case 1

Case 2

+

root =

[2,1,3]



PROBLEM 3:

Aim: 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 && //
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
isSymmetric(p->left, q->right) && //
```

```
isSymmetric(p->right, q->left);
```

```
}
```

```
};
```

Output:

Description | Editorial | Solutions | Submissions

101. Symmetric Tree

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

Example 2:

Code

```
1 class Solution {
2 public:
3     bool isSymmetric(TreeNode* root) {
4         return isSymmetric(root, root);
5     }
6
7 private:
8     bool isSymmetric(TreeNode* p, TreeNode* q) {
9         if (!p || !q)
10            return p == q;
11     }
12 }
```

Saved | Ln 16, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =
[1,2,2,3,4,4,3]

Output

true

Expected

true

15.9K | 197 | 129 Online

PROBLEM 4:

Aim: 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).

Code:

```
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;
    }
};
```

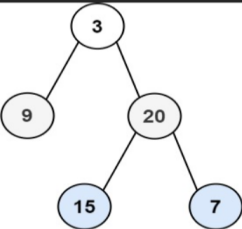
OUTPUT:

102. Binary Tree Level Order Traversal

Medium

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]`
Output: `[[3],[9,20],[15,7]]`

Example 2:

Code

```
1 class Solution {
2 public:
3     vector<vector<int>> levelOrder(TreeNode* root) {
4         if (root == nullptr)
5             return {};
6
7         vector<vector<int>> ans;
8         queue<TreeNode*> q{{root}};
9
10        while (!q.empty()) {
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

Output

[[3],[9,20],[15,7]]

Expected

[[3],[9,20],[15,7]]

PROBLEM 5:

Aim: Given an integer array `nums` where the elements are sorted in ascending order, convert *it to a height-balanced binary search tree*.

Code:

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return build(nums, 0, nums.size() - 1);
    }

private:
    TreeNode* build(const vector<int>& nums, int l, int r) {
        if (l > r)
            return nullptr;
        const int m = (l + r) / 2;
        return new TreeNode(nums[m], build(nums, l, m - 1), build(nums, m + 1, r));
    }
};
```

Output:

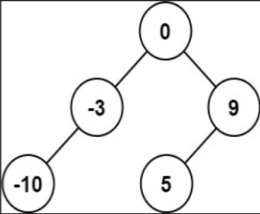
Description Editorial Solutions Submissions

108. Convert Sorted Array to Binary Search Tree


Easy Topics Companies

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a height-balanced binary search tree*.

Example 1:



Input: `nums = [-10,-3,0,5,9]`
Output: `[0,-3,9,-10,null,5]`
Explanation: `[0,-10,5,null,-3,null,9]` is also accepted:



11.3K 124 69 Online

Code

```

private:
    TreeNode* build(const vector<int>& nums, int l, int r) {
        if (l > r)
            return nullptr;
        const int m = (l + r) / 2;
        return new TreeNode(nums[m], build(nums, l, m - 1), build(nums, m + 1, r));
    }
};

```

Saved Ln 13, Col 4

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`nums =`
`[-10,-3,0,5,9]`

Output

`[0,-10,5,null,-3,null,9]`

Expected

`[0,-3,9,-10,null,5]`

PROBLEM 6:

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

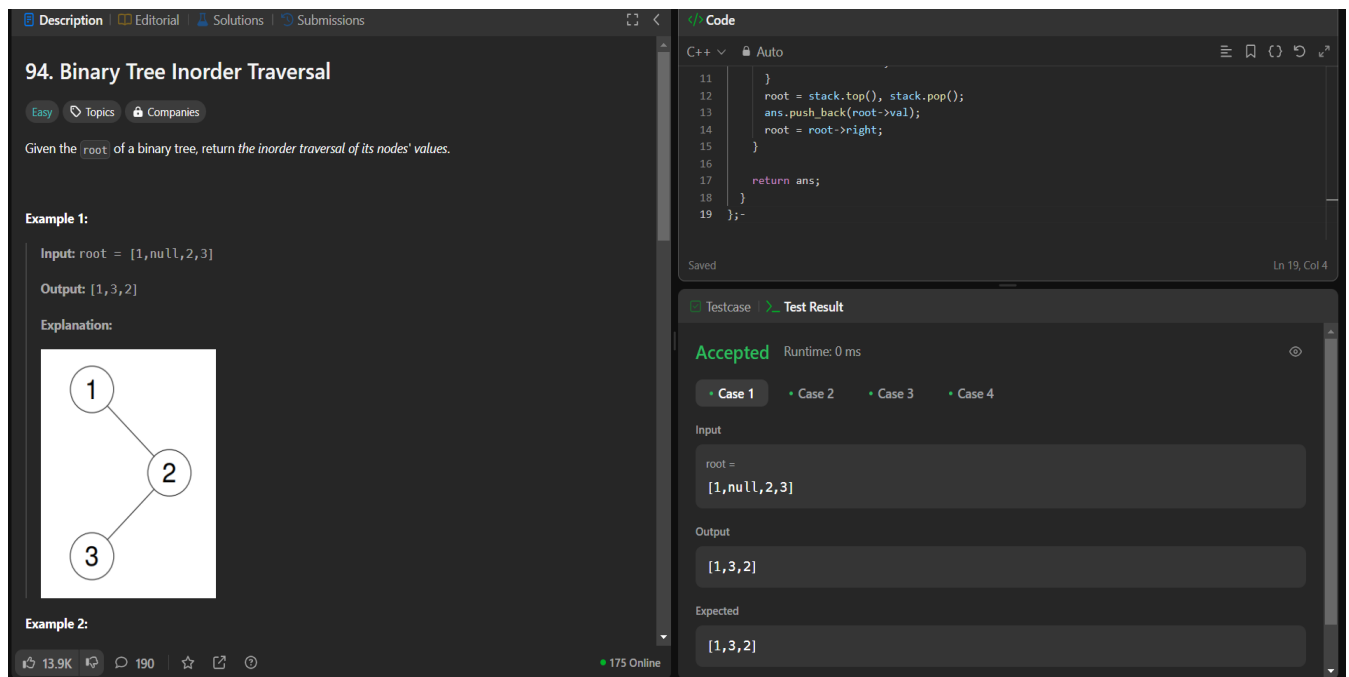
Code:

```
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;
    }
};
```

Output:



The screenshot displays a coding platform interface for the problem "94. Binary Tree Inorder Traversal". The left panel shows the problem description, which states: "Given the `root` of a binary tree, return the *inorder traversal of its nodes' values*." It includes an example with input `root = [1,null,2,3]` and output `[1,3,2]`. A diagram of the binary tree is shown: a root node 1 with a right child 2, and node 2 has a left child 3. The right panel shows a C++ code editor with the following code:

```
11     }
12     root = stack.top(), stack.pop();
13     ans.push_back(root->val);
14     root = root->right;
15 }
16
17 return ans;
18 }
19 };
```

Below the code editor, the "Test Result" section shows "Accepted" with a runtime of 0 ms. It lists four test cases, with "Case 1" selected. The input for Case 1 is `root = [1,null,2,3]`, the output is `[1,3,2]`, and the expected output is `[1,3,2]`.

PROBLEM 7:

Aim: Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return *the binary tree*.

Code:

```
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;
    }
};
```

Output:

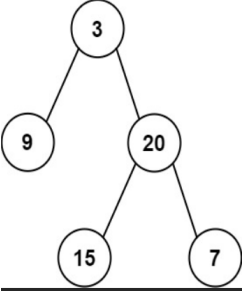
Description | Editor | Solutions | Submissions

106. Construct Binary Tree from Inorder and Postorder Traversal

Medium | Topics | Companies

Given two integer arrays `inorder` and `postorder` where `inorder` is the inorder traversal of a binary tree and `postorder` is the postorder traversal of the same tree, construct and return *the binary tree*.

Example 1:



Input: `inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]`
Output: `[3,9,20,null,null,15,7]`

Example 2:

8.3K | 78 | 47 Online

Code

```
C++
1 class Solution {
2 public:
3     TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
```

Saved | Ln 30, Col 4

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

`inorder =`
`[9,3,15,20,7]`

`postorder =`
`[9,15,7,20,3]`

Output

`[3,9,20,null,null,15,7]`

Expected

`[3,9,20,null,null,15,7]`

Contribute a testcase

PROBLEM 9:

Aim: 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.*

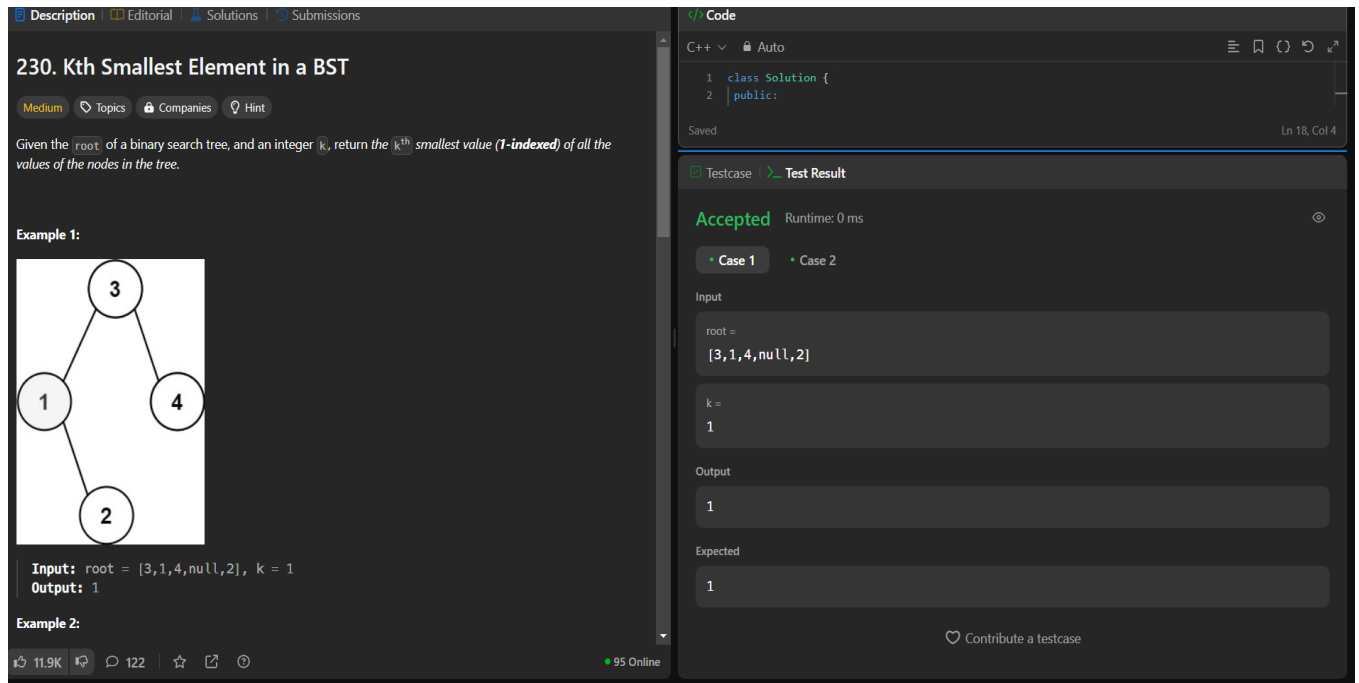
Code:

```
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);
    }
};
```

Output:

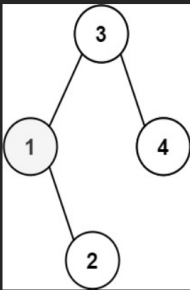


230. Kth Smallest Element in a BST

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:



```

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

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

Example 2:

11.9K 122 95 Online

Code

```

C++ Auto
1 class Solution {
2 public:

```

Saved Ln 18, Col 4

Testcase **Test Result**

Accepted Runtime: 0 ms

Case 1 **Case 2**

Input

`root =`
`[3,1,4,null,2]`

`k =`
`1`

Output

`1`

Expected

`1`

Contribute a testcase

PROBLEM 9:

Aim: 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.

Code:

```

class Solution {
public:
    Node* connect(Node* root) {
        if (root == nullptr)
            return nullptr;
        connectTwoNodes(root->left, root->right);
        return root;
    }

private:
    void connectTwoNodes(Node* p, Node* q) {
        if (p == nullptr)

```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
return;  
p->next = q;  
connectTwoNodes(p->left, p->right);  
connectTwoNodes(q->left, q->right);  
connectTwoNodes(p->right, q->left);  
}  
};
```

Output:

Description

Editorial

Solutions

Submissions

116. Populating Next Right Pointers in Each Node

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:

Figure 1: Initial tree structure. Figure 2: Tree structure after populating next pointers.

Code

C++

Auto

Ln 18, Col 3

Ln 19, Col 3

Saved

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Input

root =
[1,2,3,4,5,6,7]

Output

[1,#,2,3,#,4,5,6,7,#]

Expected

[1,#,2,3,#,4,5,6,7,#]

Contribute a testcase



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.