



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 5

Student Name: Amit Kumar

Branch: BE-CSE

Semester:6

Subject Name: AP LAB

UID:22BCS14056

Section/Group:637/B

Date of Performance:14-2-25

Subject Code:22CSH-359

- 1. Problem statement-** 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.

```
</> Code
Java ▾ 🔒 Auto
Ln 1, Col 1

2 class TreeNode {
3     int val;
4     TreeNode left;
5     TreeNode right;
6     TreeNode() {
7     }
8     TreeNode(int val) {
9         this.val = val;
10    }
11    TreeNode(int val, TreeNode left, TreeNode right) {
12        this.val = val;
13        this.left = left;
14        this.right = right;
15    }
16 }
17 class Solution {
18     public boolean isSameTree(TreeNode p, TreeNode q) {

Saved
```

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

p =
[1,2,3]

2. Problem statement- Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center)

```

</> Code
Java ▾ Auto
1
2 class TreeNode {
3     int val;
4     TreeNode left;
5     TreeNode right;
6     TreeNode() {}
7     TreeNode(int val) { this.val = val; }
8     TreeNode(int val, TreeNode left, TreeNode right) {
9         this.val = val;
10        this.left = left;
11        this.right = right;
12    }
13 }
14 class Solution {
15     public boolean isSymmetric(TreeNode root) {
16         return isMirror(root, root);
17     }
18     private boolean isMirror(TreeNode t1, TreeNode t2) {
19         if (t1 == null && t2 == null) {
20             return true;
21         }
22         if (t1 == null || t2 == null) {
23             return false;
24         }
25         return (t1.val == t2.val) && isMirror(t1.left, t2.right) && isMirror(t1.right, t2.left);
26     }
27 }

```

3. Problem statement- Given a binary tree, determine if it is height-balanced.

```

</> Code
Java ▾ Auto
1
2 class TreeNode {
3     int val;
4     TreeNode left;
5     TreeNode right;
6     TreeNode() {}
7     TreeNode(int val) { this.val = val; }
8     TreeNode(int val, TreeNode left, TreeNode right) {
9         this.val = val;
10        this.left = left;
11        this.right = right;
12    }
13 }
14
15 class Solution {
16     public boolean isBalanced(TreeNode root) {
17         return isBalanced(root.left) && isBalanced(root.right) &&

```

Saved Ln 1, Col 1

Testcase | Test Result

Input







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



Output







false

Expected

4. Problem statement- 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.


Submit      0  Premium


</> Code  

Java  Auto     

```
1 class TreeNode {
2     int val;
3     TreeNode left;
4     TreeNode right;
5     TreeNode() {}
6     TreeNode(int val) { this.val = val; }
7     TreeNode(int val, TreeNode left, TreeNode right) {
8         this.val = val;
9         this.left = left;
10        this.right = right;
11    }
12 }
13 class Solution {
14     public boolean hasPathSum(TreeNode root, int targetSum) {
15         if (root == null) {
16             return false;
17         }
18     }
19 }
```

Saved Ln 12, Col 2

☒ Testcase  Test Result

Accepted Runtime: 0 ms 

- Case 1
- Case 2
- Case 3

Input

root =
[1,2,3]

5. Problem statement- 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

Java ▾ Auto

```
2   int val;
3   TreeNode left;
4   TreeNode right;
5   TreeNode(int x) { val = x; }
6 }
7 public class Solution {
8     public int countNodes(TreeNode root) {
9         if (root == null) {
10             return 0;
11         }
12         int height = getHeight(root);
13         if (height < 0) {
14             return 0;
15         }
16         int left = 0, right = (1 << height) - 1;
17         while (left <= right) {
18             int mid = left + (right - left) / 2;
19             if (exists(mid, height, root)) {
```

Saved

Ln 34, Col 6

☒ Testcase | [> Test Result](#)

Accepted Runtime: 0 ms



• Case 1 • Case 2 • Case 3

Input

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

6. Problem statement- 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.

Java ▾ 🔒 Auto

```
1 class TreeNode {
2     int val;
3     TreeNode left;
4     TreeNode right;
5     TreeNode(int x) { val = x; }
6 }
7 public class Solution {
8     public TreeNode deleteNode(TreeNode root, int key) {
9         if (root == null) {
10             return null;
11         }
12         if (key < root.val) {
13             root.left = deleteNode(root.left, key);
14         }
15         else if (key > root.val) {
16             root.right = deleteNode(root.right, key);
17         }
18         if (key == root.val) {
19             // Node to be deleted
20             if (root.left == null) {
21                 return root.right;
22             }
23             if (root.right == null) {
24                 return root.left;
25             }
26             // Node has both left and right children
27             // Find the In-order Successor (Minimum in the right subtree)
28             TreeNode minNode = root.right;
29             while (minNode.left != null) {
30                 minNode = minNode.left;
31             }
32             // Replace the value of the current node with the In-order Successor
33             root.val = minNode.val;
34             // Delete the In-order Successor from the right subtree
35             root.right = deleteNode(root.right, minNode.val);
36         }
37     }
38 }
```

Saved

Ln 6, Col 2

☒ Testcase

☐ Test Result

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

key =
3

Output
[5,4,6,2,null,null,7]

7. Problem statement- 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

Java ▾ 🔒 Auto

```
1 class TreeNode {
2     int val;
3     TreeNode left;
4     TreeNode right;
5     TreeNode(int x) { val = x; }
6 }
7 public class Solution {
8     private int diameter = 0; // This will hold the maximum diameter found
9     public int diameterOfBinaryTree(TreeNode root) {
10         depth(root); // Start the depth-first search
11         return diameter; // Return the maximum diameter found
12     }
13     private int depth(TreeNode node) {
14         if (node == null) {
15             return 0; // Base case: the depth of a null node is 0
16         }
17     }
```

Saved

Ln 12, Col 6

☒ Testcase | ☒ Test Result

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

Output

3

Expected

3