



## Experiment 5

**Student Name:** Akash Kumar Singh

**UID:** 22BCS15769

**Branch:** B.E-C.S.E

**Section/Group:** 638/B

**Semester:** 6th

**Date of Performance:** 18/2/25

**Subject Name:** A.P LAB II

**Subject Code:** 22CSP-351

### **Experiment-2.1.1(Same Tree)**

#### **1. Aim:**

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.

#### **2. Implementation/Code:**

```
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if (p == null && q == null) {
            return true;
        }
        if (p == null || q == null || p.val != q.val) {
            return false;
        }
        return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```

#### **3. Output**



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

☒ Testcase | [> Test Result](#)

• Case 1

• Case 2

• Case 3

Input

p =  
[1,2,3]

q =  
[1,2,3]

Output

true

Leetcode Link: <https://leetcode.com/problems/same-tree/>

## Experiment-2.1.2(Symmetric Tree)

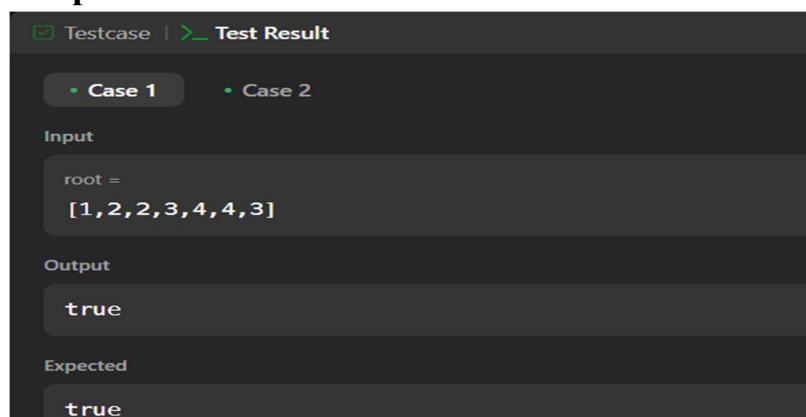
### 1. Aim:

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

### 2. Implementation /Code:

```
class Solution {  
    public boolean isSymmetric(TreeNode root) {  
        if (root == null) return true;  
        return isMirror(root.left, root.right);  
    }  
  
    private boolean isMirror(TreeNode t1, TreeNode t2) {  
        if (t1 == null && t2 == null) return true;  
        if (t1 == null || t2 == null) return false;  
        return (t1.val == t2.val) && isMirror(t1.left, t2.right) &&  
isMirror(t1.right, t2.left);  
    }  
}
```

### 3. Output



Leetcode Link : <https://leetcode.com/problems/symmetric-tree/>

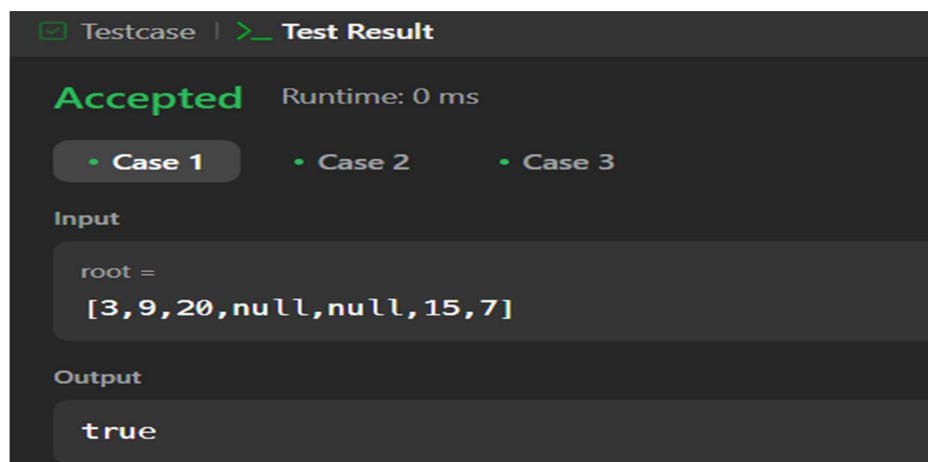
## Experiment-2.1.3(Balanced Binary Tree)

1. **Aim:** Given a binary tree, determine if it is height-balanced. A binary tree is height-balanced if the difference between the heights of the left and right subtrees of any node is no more than 1.

2. **Implementation/Code:**

```
class Solution {  
    public boolean isBalanced(TreeNode root) {  
        return height(root) != -1;  
    }  
    private int height(TreeNode node) {  
        if (node == null) return 0;  
        int leftHeight = height(node.left);  
        if (leftHeight == -1) return -1;  
        int rightHeight = height(node.right);  
        if (rightHeight == -1) return -1;  
        if (Math.abs(leftHeight - rightHeight) > 1) return -1;  
        return Math.max(leftHeight, rightHeight) + 1;  
    }  
}
```

3. **Output:**



LeetCode Link: <https://leetcode.com/problems/balanced-binary-tree/>

## Experiment-2.1.4(Path Sum)

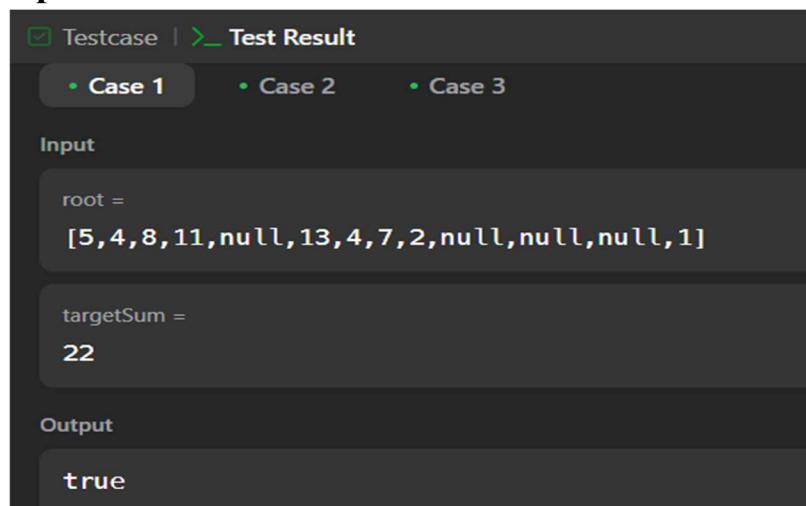
### 1. Aim:

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.

### 2. Code:

```
class Solution {  
    public boolean hasPathSum(TreeNode root, int targetSum) {  
        if (root == null) return false;  
        if (root.left == null && root.right == null) {  
            return targetSum == root.val;  
        }  
        return hasPathSum(root.left, targetSum - root.val) ||  
            hasPathSum(root.right, targetSum - root.val);  
    }  
}
```

### 3. Output:



LeetCode Link: <https://leetcode.com/problems/path-sum/submissions/1554716818/>

**Experiment-2.1.5(Delete Node in a BST)**

**1. Aim:** Given the root of a BST and a key, delete the node with the given key in the BST.

**2. Code:**

```
class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if (root == null) return null;

        if (key < root.val) {
            root.left = deleteNode(root.left, key);
        } else if (key > root.val) {
            root.right = deleteNode(root.right, key);
        } else {
            if (root.left == null) return root.right;

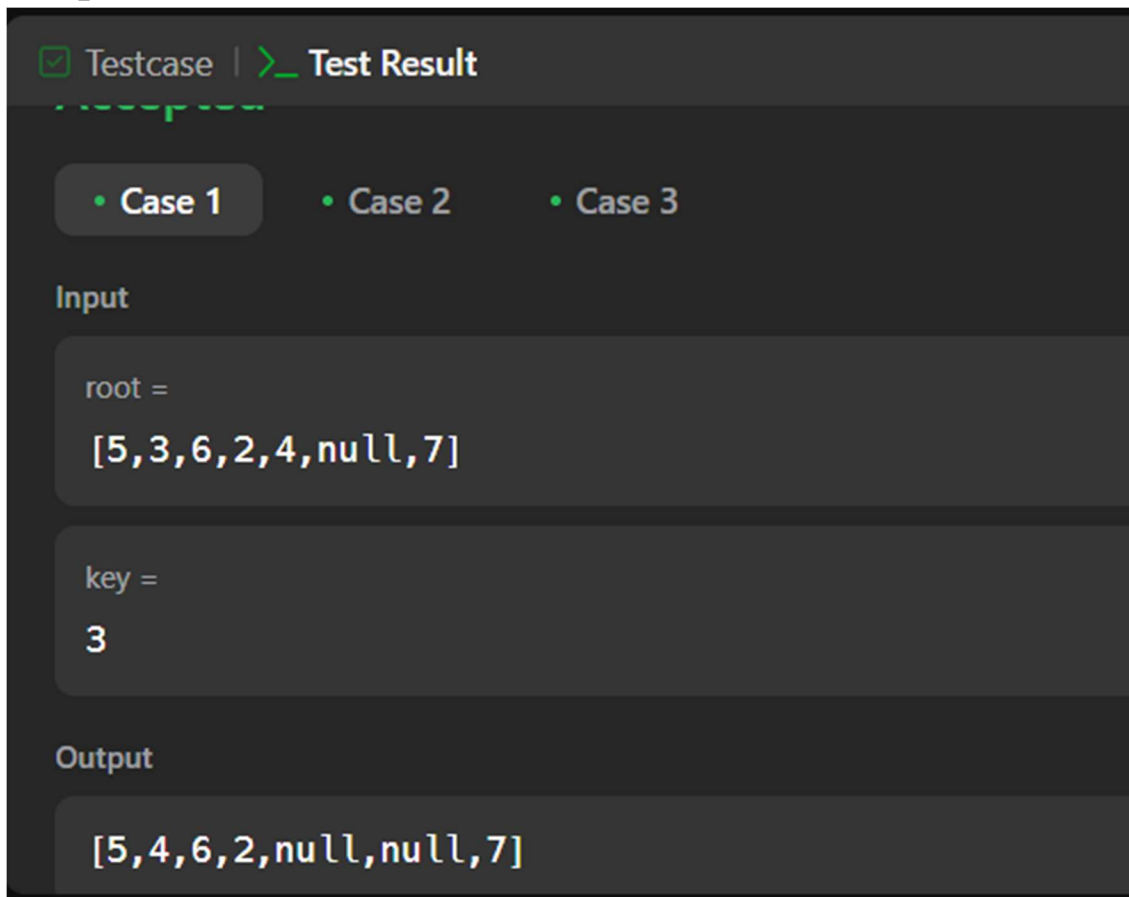
            if (root.right == null) return root.left;

            TreeNode minNode = getMin(root.right);
            root.val = minNode.val;
            root.right = deleteNode(root.right, minNode.val);
        }
        return root;
    }

    private TreeNode getMin(TreeNode node) {
```

```
        while (node.left != null) {  
            node = node.left;  
        }  
        return node;  
    }  
}
```

### 3. Output:



The screenshot shows a coding platform interface with a dark theme. At the top, there is a tab labeled 'Testcase' with a green checkmark and a tab labeled 'Test Result' with a green arrow. Below the tabs, there are three buttons: 'Case 1', 'Case 2', and 'Case 3'. The 'Case 1' button is selected. Under the 'Input' section, there are two text boxes: 'root =' followed by '[5,3,6,2,4,null,7]' and 'key =' followed by '3'. Under the 'Output' section, there is a text box showing '[5,4,6,2,null,null,7]'. The interface is clean and modern, with a focus on the test results.

Link: <https://leetcode.com/problems/delete-node-in-a-bst/submissions/1554723767/>

### Experiment-2.1.6(Count Complete Tree Nodes)

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

2. **Code:**

```
class Solution {
    public int countNodes(TreeNode root) {
        if (root == null) return 0;

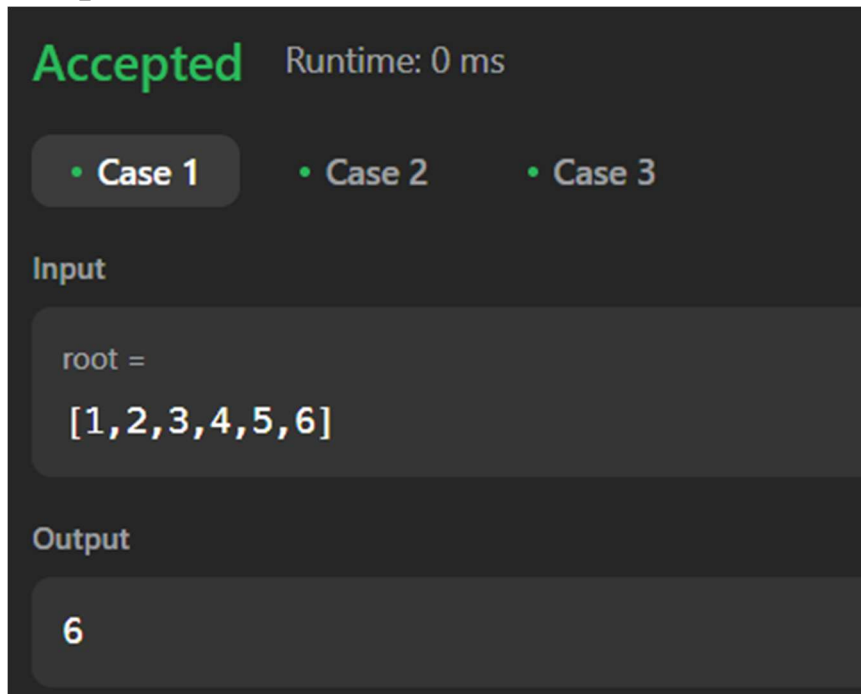
        int leftDepth = getDepth(root.left);
        int rightDepth = getDepth(root.right);

        if (leftDepth == rightDepth) {
            return (1 << leftDepth) + countNodes(root.right);
        } else {
            return (1 << rightDepth) + countNodes(root.left);
        }
    }

    private int getDepth(TreeNode node) {
        int depth = 0;
        while (node != null) {
            depth++;
            node = node.left;
        }
        return depth;
    }
}
```



### 3. Output:



Link: <https://leetcode.com/problems/count-complete-tree-nodes/>

### Experiment-2.1.7(Diameter of Binary Tree)

1. **Aim:** 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.
2. **Code:**

```
class Solution {  
    private int diameter = 0; // Stores the maximum diameter found  
  
    public int diameterOfBinaryTree(TreeNode root) {  
        height(root);  
        return diameter;  
    }  
    private int height(TreeNode node) {
```

```
        if (node == null) return 0;
        int leftHeight = height(node.left);
        int rightHeight = height(node.right);
        diameter = Math.max(diameter, leftHeight + rightHeight);
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```

### 3. Output:

☒ Testcase | >\_ Test Result

• Case 1

• Case 2

Input

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

Output

3

Expected

3

Link: <https://leetcode.com/problems/diameter-of-binary-tree/submissions/1554742412/>