Experiment-5

Student Name: Anushree Alok **UID:** 22BCS16052
Branch: BE-CSE **Section/Group:** IOT 625-A
Semester: 6th **Date of Performance:**14/02/25
Subject Name: Advanced Programming Lab - 2 **Subject Code:** 22CSP-351

1. Aim:

a) Same Tree: To check if they are the same or not, given the roots of two binary trees p and q. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

b) Path Sum: To check if a tree has a root-to-leaf path such that adding up all the values along the path equals targetSum, given the root of the binary tree and an integer targetSum'

2. Objectives:

- To check if root node values of both trees are equal.
- To check for empty trees.
- To subtract node value from target sum all the way starting from the root.
- To return Boolean value if target sum becomes zero at the end.
- To perform recursion in both left and right subtrees.

3. Algorithm:

(a)

STEP-1: Check if both trees are the same node:

- If both **p** and **q** are the same object in memory, return true (both trees are identical).

- If one of the trees is **null** and the other is not, return false (trees are different).
- If the values of **p** and **q** are not equal, return false (trees are different).

STEP-2: Recursively compare left subtrees:
- Call the **isSameTree** method on the left child of both trees (**p.left** and **q.left**).
- If both left subtrees are identical, continue to the next step.

STEP-3: Recursively compare right subtrees
- Call the **isSameTree** method on the right child of both trees (**p.right** and **q.right**).
- If both right subtrees are identical, return true.

STEP-4: Return Boolean Value
- If both the left and right subtrees are the same, return true.
- If either the left or right subtrees are different, return false.

(b)

STEP-1: Check if root is null
- If the root is **null**, return false (no path exists).

STEP-2: Decrease target sum by current node's value
- Subtract the current node's value (**root.val**) from the target sum (**s**).

STEP-3: Check if it's a leaf node and if target sum is **0**
- If the current node is a leaf node (both left and right children are **null**) and the remaining sum (**s**) is **0**, return true (path exists with the given sum).

STEP-4: Recursively check left and right subtrees
- Recursively call the **dfs** method on the left child (**root.left**) and the right child (**root.right**) with the updated sum **s**.

STEP-5: Return Boolean Value
- Return the logical OR of the results from the left and right subtree checks (if either subtree has a valid path sum, return true; otherwise, return false)

4. Implementation/Code:

(a)

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if (p == q) return true;
        if (p == null || q == null || p.val != q.val) return false;
        return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```

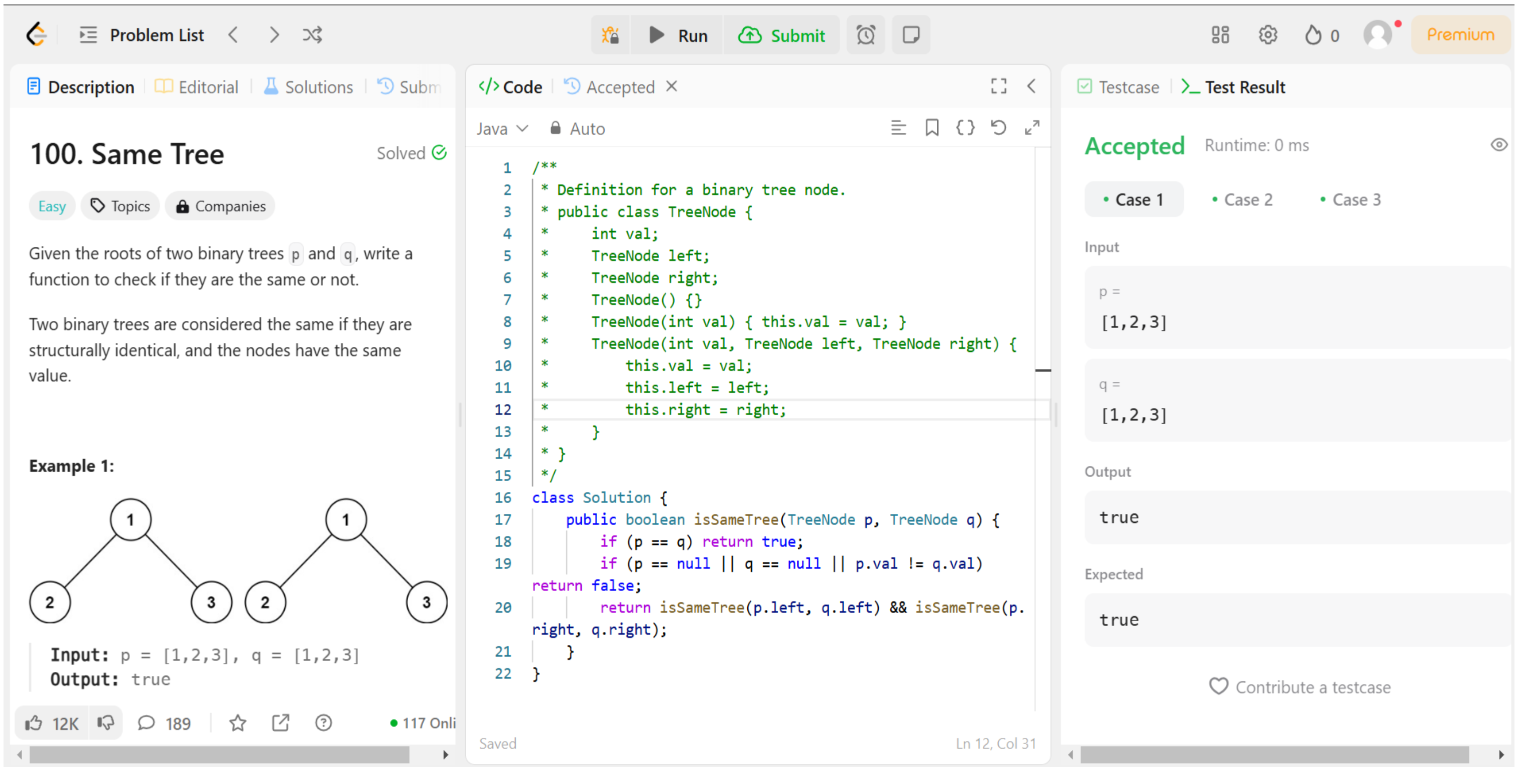
(b)

```
class
/**
 * Definition for a binary tree node.
 * public class TreeNode {
```

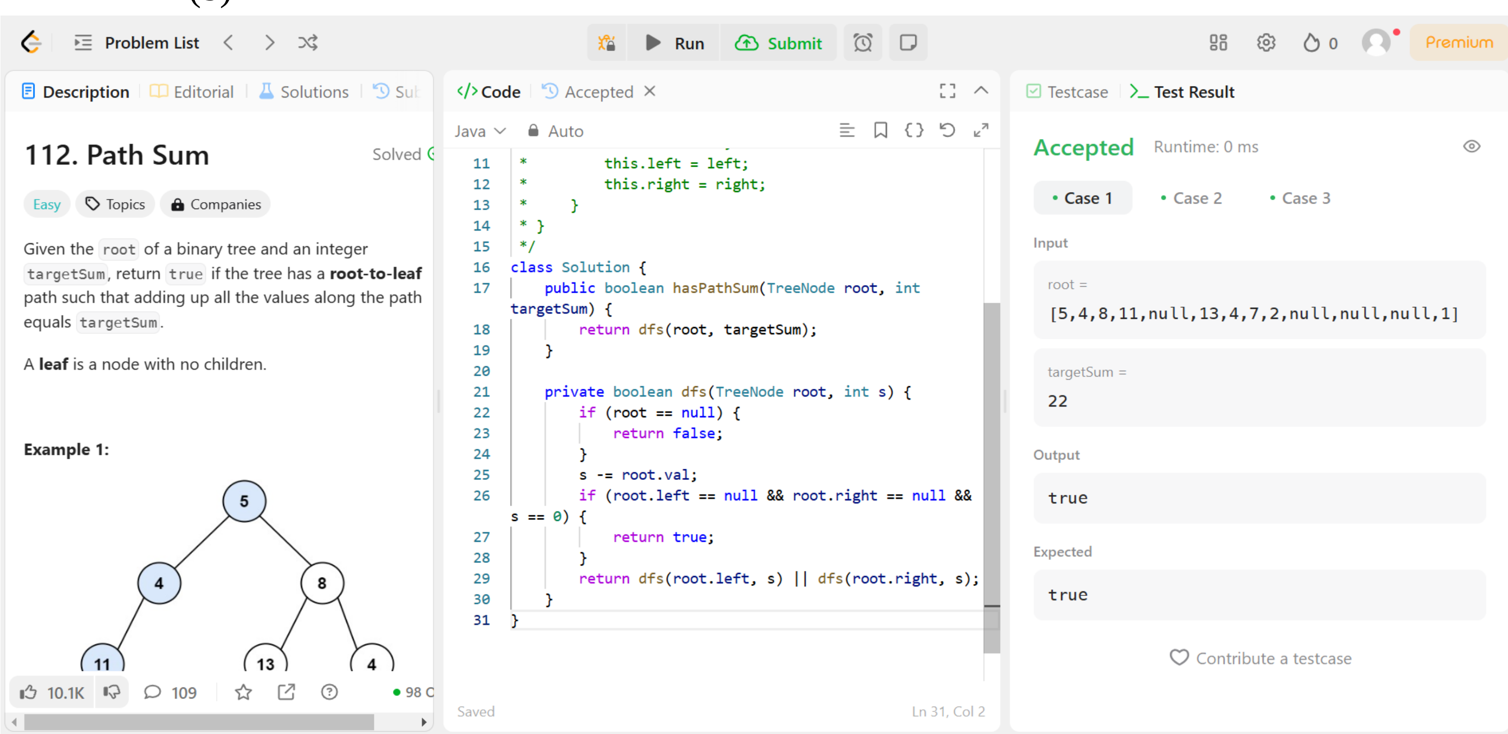
```
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean hasPathSum(TreeNode root, int targetSum) {
        return dfs(root, targetSum);
    }
    private boolean dfs(TreeNode root, int s) {
        if (root == null) {
            return false;
        }
        s -= root.val;
        if (root.left == null && root.right == null && s == 0) {
            return true;
        }
        return dfs(root.left, s) || dfs(root.right, s);
    }
}
```

5. Output:

(a)


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Code
Accepted
Testcase
Test Result
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
12K
189
Java
Auto
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public boolean isSameTree(TreeNode p, TreeNode q) {
if (p == q) return true;
if (p == null || q == null || p.val != q.val) return false;
return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
}
}
Saved
Ln 12, Col 31
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
p =
[1,2,3]
q =
[1,2,3]
Output
true
Expected
true
Contribute a testcase

(b)


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Code
Accepted
Testcase
Test Result
112. Path Sum
Solved
Easy
Topics
Companies
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.
Example 1:
10.1K
109
Java
Auto
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public boolean hasPathSum(TreeNode root, int targetSum) {
return dfs(root, targetSum);
}
private boolean dfs(TreeNode root, int s) {
if (root == null) {
return false;
}
s -= root.val;
if (root.left == null && root.right == null && s == 0) {
return true;
}
return dfs(root.left, s) || dfs(root.right, s);
}
}
Saved
Ln 31, Col 2
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =
[5,4,8,11,null,13,4,7,2,null,null,null,1]
targetSum =
22
Output
true
Expected
true
Contribute a testcase

6. Time Complexity:

(a) **O(n):** If both trees have n nodes, the time complexity will be proportional to the number of nodes **n**. In the worst case, we must visit every node in both trees therefore taking O(n) time.

(b) **O(n):** In the worst case, we need to visit every node in the binary tree to find the path that sums to the target therefore if the tree has **n** nodes, the time complexity will be O(n).

7. Space Complexity:

1. **O(h):** The recursion stack depth can go as deep as the height of the tree h. In the worst case, we need to store up to the maximum depth of the recursion stack therefore taking O(h) time where h is the height of the tree.
2. **O(h):** In the worst case (a skewed tree), the recursion stack depth can go as deep as the hieght **h** of the tree, therefore making it O(h).

8. Learning Outcomes:

- Learnt about trees binary trees and traversal and searching operations on it.
- Learnt how to compare two trees.
- Learnt to find the sum of nodes on a path in a tree.