CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

Experiment-2.1

Student Name: Ayushi Sharma **UID:** 21BCS8973

Branch: BE-CSE **Section/Group:**21BCSCC-645-B

Semester: 6th **Date of Performance:**19-02-2024

Subject Name: Advanced Programming lab-2 **Subject Code:**21CSP-351

Aim:

1. To Solve the Same Tree Problem
2. To Solve the Diameter of the Binary Tree Problem

Objective:

- **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.**

- **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.**

Algorithm:

Same Tree Algorithm:

- Base Case Handling: Checks if both trees are empty. If so, returns true.
- Null Check: If one tree is empty while the other is not, returns false.
- Value Comparison: Compares the values of the current nodes in both trees.
- Recursive Calls: Recursively calls the function for left subtrees and right subtrees.
- Return: Returns true if all conditions hold true, indicating identical trees; otherwise, returns false.

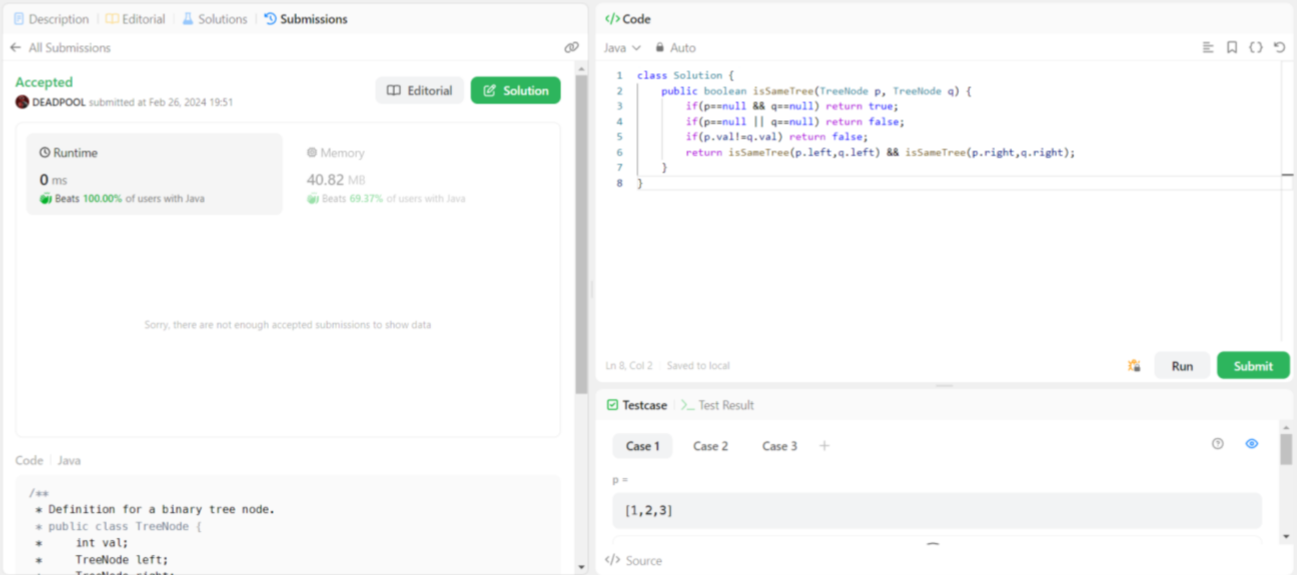
Diameter of Binary Tree Algorithm:

- Define Helper Function:
- Create a function named help that takes a node as input.
- In the helper function, check if the current node is null (empty). If it is, return 0.
- Recursively call the help function on the left and right child nodes of the current node.
- Update the global variable diameter with the maximum value between itself and the sum of the heights of the left and right subtrees. diameter = max(diameter, left + right);
- Return Height:
- Return the maximum height of the left and right subtrees plus 1 (to account for the current node). return max(left, right) + 1;

Code(A):

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

Output(A):



Accepted
DEADPOOL submitted at Feb 26, 2024 19:51

Runtime
0 ms
Beats 100.00% of users with Java

Memory
40.82 MB
Beats 69.37% of users with Java

Sorry, there are not enough accepted submissions to show data

Code | Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
```

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

Ln 8, Col 2 | Saved to local

p =

[1,2,3]

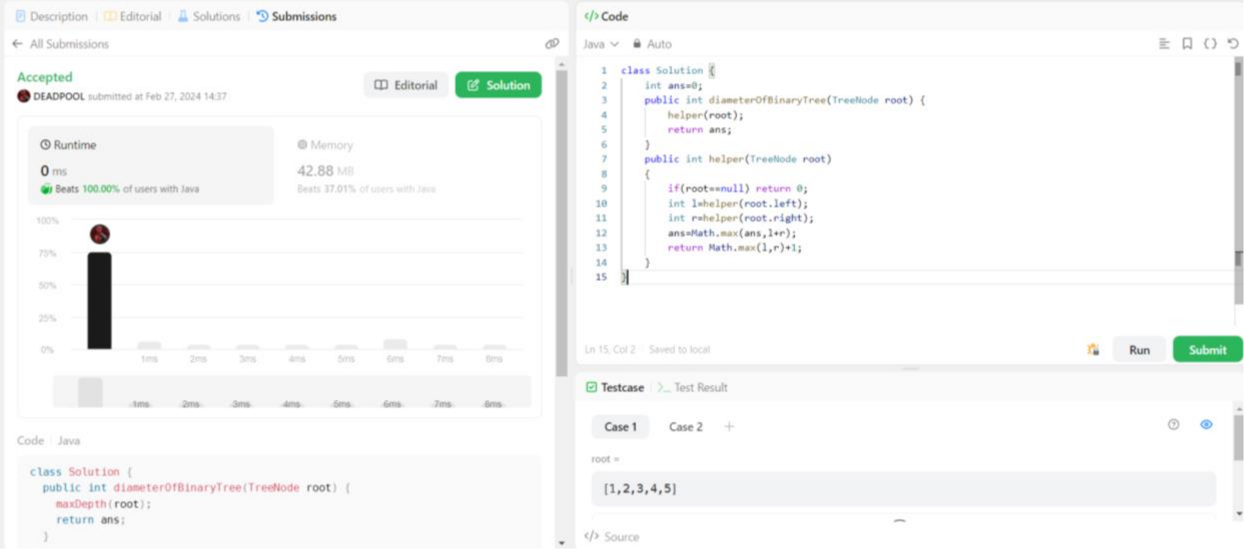
Time Complexity (A): O(N)

Space Complexity(A): O(N)

Code(B):

```
class Solution {
    int ans=0;
    public int diameterOfBinaryTree(TreeNode root) {
        helper(root);
        return ans;
    }
    public int helper(TreeNode root)
    {
        if(root==null) return 0;
        int l=helper(root.left);
        int r=helper(root.right);
        ans=Math.max(ans,l+r);
        return Math.max(l,r)+1;
    }
}
```

Output(B):



Accepted
DEADPOOL submitted at Feb 27, 2024 14:37

Runtime
0 ms
Beats 100.00% of users with Java

Memory
42.88 MB
Beats 37.01% of users with Java

Code | Java

```
class Solution {
  public int diameterOfBinaryTree(TreeNode root) {
    maxDepth(root);
    return ans;
  }
```

```
class Solution {
    int ans=0;
    public int diameterOfBinaryTree(TreeNode root) {
        helper(root);
        return ans;
    }
    public int helper(TreeNode root)
    {
        if(root==null) return 0;
        int l=helper(root.left);
        int r=helper(root.right);
        ans=Math.max(ans,l+r);
        return Math.max(l,r)+1;
    }
}
```

Ln 15, Col 2 | Saved to local

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

Time Complexity (B): O(N)

Space Complexity(B): O(N)