Experiment-5

Student Name: Ruchika Yadav
UID: 22BCS15921
Branch: BE-CSE
Section/Group: KPIT-901/B
Semester: 6th
Date of Performance: 18/01/25
Subject Name: Advanced Programming Lab - 2
Subject Code: 22CSP-351

1. Aim:

1. **Problem: 2.2.1:** 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. **Problem: 2.2.2:** Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

2. Objective:

Problem 2.2.1:
- Understand how to compare two binary trees recursively.
- Learn how to handle different edge cases, such as null nodes and value mismatches.

Problem 2.2.2: Find the index of the first occurrence of a substring (needle) in a given string (haystack) or return -1 if the substring is not found.

1. Implementation/Code:

1.)

```
class Solution {
public:
   bool isSameTree(TreeNode* p, TreeNode* q) {
      if (!p && !q) return true;

      if (!p || !q || p->val != q->val) return false;

      return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
   }
};
```

`

2.)

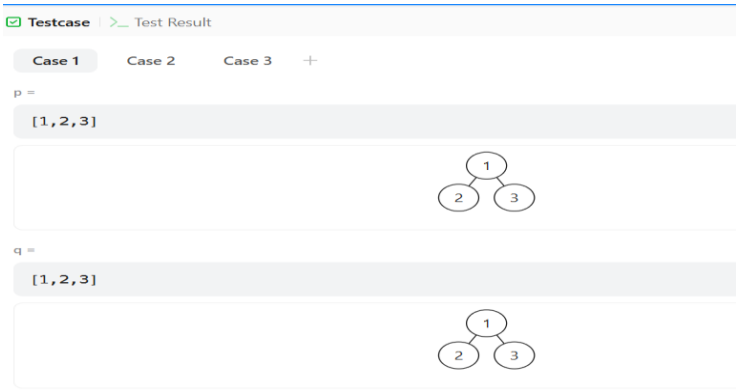
```
#include <iostream>
#include <string>
using namespace std;

int strStr(string haystack, string needle)
{
    if (needle.empty())
        return 0;
    for (int i = 0; i <= haystack.size() - needle.size(); i++)
    {
        if (haystack.substr(i, needle.size()) == needle)
        {
            return i;
        }
    }
    return -1;
}

int main()
{
    string haystack, needle;
    cout << "Enter the haystack string: ";
    cin >> haystack;
    cout << "Enter the needle string: ";
    cin >> needle;

    int index = strStr(haystack, needle);

    if (index != -1)
    {
```

`

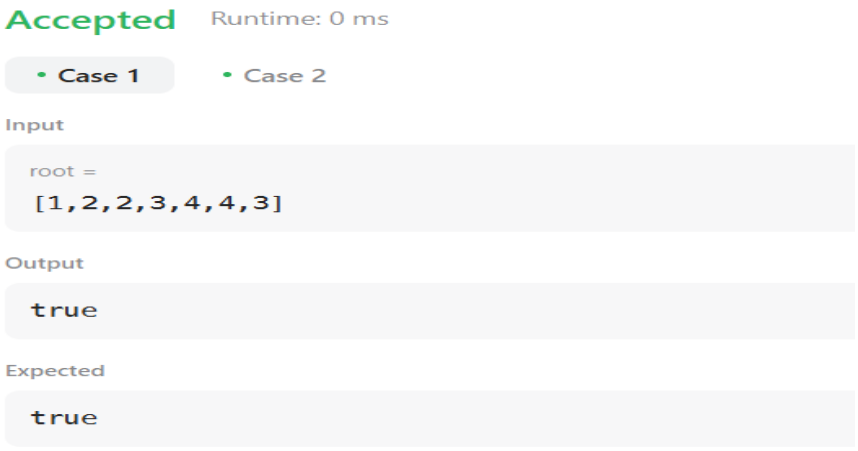
```
        cout << "The first occurrence of \"" << needle << "\" in \"" << haystack <<
"\" is at index: " << index << endl;
    }
    else
    {
        cout << "The substring \"" << needle << "\" is not found in \"" << haystack
<< "\"." << endl;
    }

    return 0;
}
```

2. Output:

1.



2.



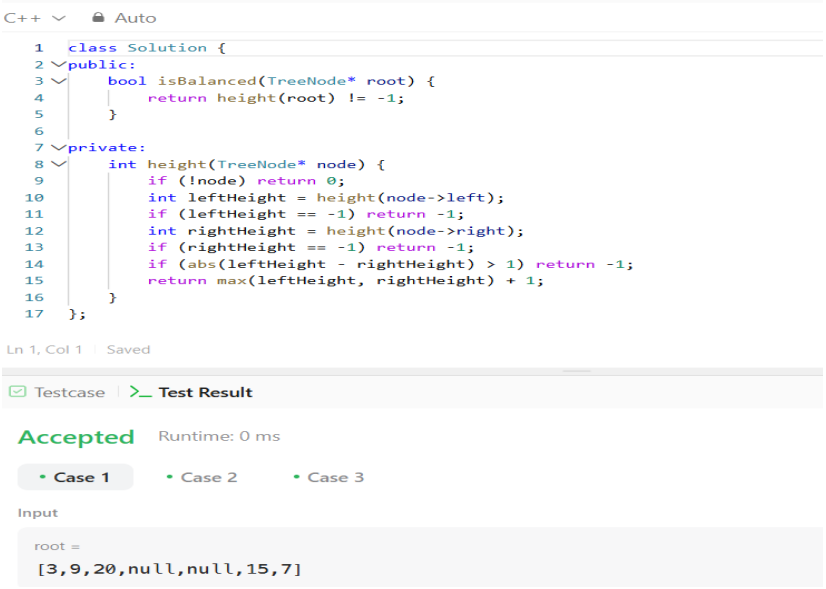
`

3. **Time Complexity:**
   1. O(n+m)
   2. O(n-m+1)

4. **Space Complexity:**
   1. O(n)
   2. O(1)

5. **Learning Outcome:**
   1. Understand string manipulations and rotations.
   2. Learn how to check for substrings efficiently.
   3. Develop problem-solving skills for string-related algorithms.
   4. Gain knowledge of substring search techniques.
   5. Practice using loops and conditionals for string traversal.
   6. Enhance skills in optimizing string operations.

2.2.3



`

2.2.4

C++ Auto

```cpp
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        if (!root) return false;
        if (!root->left && !root->right) {
            return targetSum == root->val;
        }
        return hasPathSum(root->left, targetSum - root->val) ||
               hasPathSum(root->right, targetSum - root->val);
    }
};
```

Ln 1, Col 1 Saved

Testcase | Test Result

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

2.2.5

C++ Auto

```cpp
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (!root) return 0;
        int leftHeight = getLeftHeight(root);
        int rightHeight = getRightHeight(root);
        if (leftHeight == rightHeight) {
            return (1 << leftHeight) - 1;
        } else {
            return 1 + countNodes(root->left) + countNodes(root->right);
        }
    }

private:
    int getLeftHeight(TreeNode* node) {
        int height = 0;
        while (node) {
```

Ln 1, Col 1 Saved

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

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

`