Experiment-6

Student Name: Abhiraj Patel
UID: 22BCS11329
Branch: BE-CSE
Section/Group: KRG-IOT-1A
Semester: 6th
Date of Performance:03/03/25
Subject Name: Advanced Programming Lab - 2
Subject Code: 22CSP-351

1. Aim:

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

2. Implementation/Code:

1.)

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

bool isSameTree(TreeNode* p, TreeNode* q) {
    if (!p || !q) return p == q;
    return p->val == q->val && isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}

int main() {
```

```
    TreeNode* p = new TreeNode(1);
    p->left = new TreeNode(2);
    p->right = new TreeNode(3);

    TreeNode* q = new TreeNode(1);
    q->left = new TreeNode(2);
    q->right = new TreeNode(3);

    cout << (isSameTree(p, q) ? "Same" : "Not Same") << endl;
    return 0;
}
```

2.)

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

bool isMirror(TreeNode* t1, TreeNode* t2) {
    if (!t1 || !t2) return t1 == t2;
    return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1-
>right, t2->left);
}

bool isSymmetric(TreeNode* root) {
    return !root || isMirror(root->left, root->right);
}

int main() {
```

```
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(2);
    root->left->left = new TreeNode(3);
    root->left->right = new TreeNode(4);
    root->right->left = new TreeNode(4);
    root->right->right = new TreeNode(3);

    cout << (isSymmetric(root) ? "Symmetric" : "Not Symmetric") <<endl;

    return 0;
}
```

3. **Output:**

1.

```
Same
```

2.

```
Symmetric
```

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

5. **Space Complexity:**
   1. O(n)

2. O(n)

6. Learning Outcome:

1. Understanding tree structures, node relationships, and recursion-based traversal.
2. Applying recursion effectively to traverse and compare tree nodes.
3. Identifying symmetric structures using a recursive mirror-checking approach.
4. Using DFS for tree comparison and symmetry checks.