

Experiment-5

Student Name: Kundan

UID: 22BCS10726

Branch: BE-CSE

Section/Group: IOT_637 (B)

Semester: 6th

Date of Performance: 19/02/2025

Subject Name: AP LAB-II

Subject Code: 22CSP-351

Problem- 1

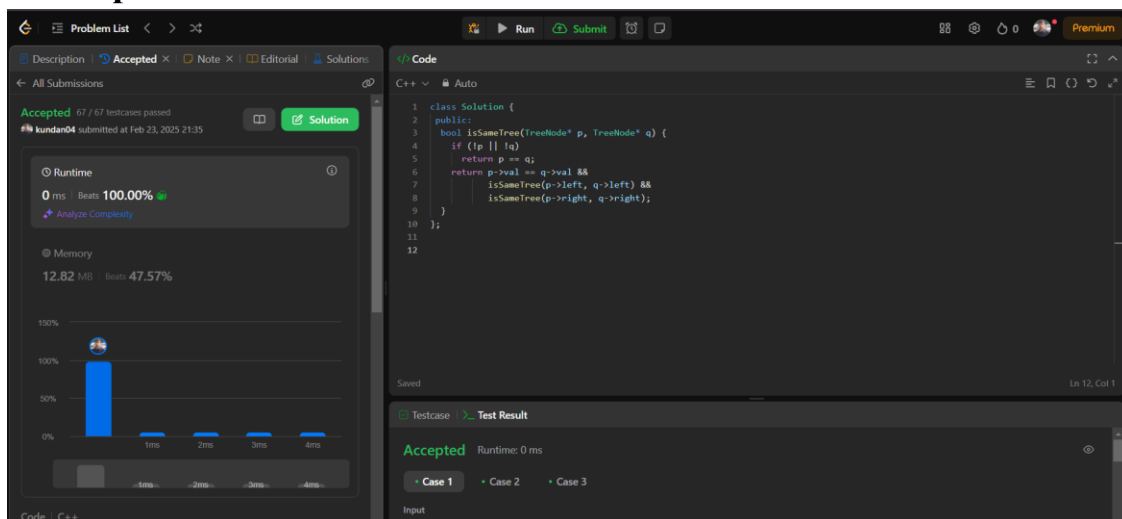
1. Aim:

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: Backend:

```
class Solution {  
public:  
    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);  
    }  
};
```

3. Output:



Problem- 2

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

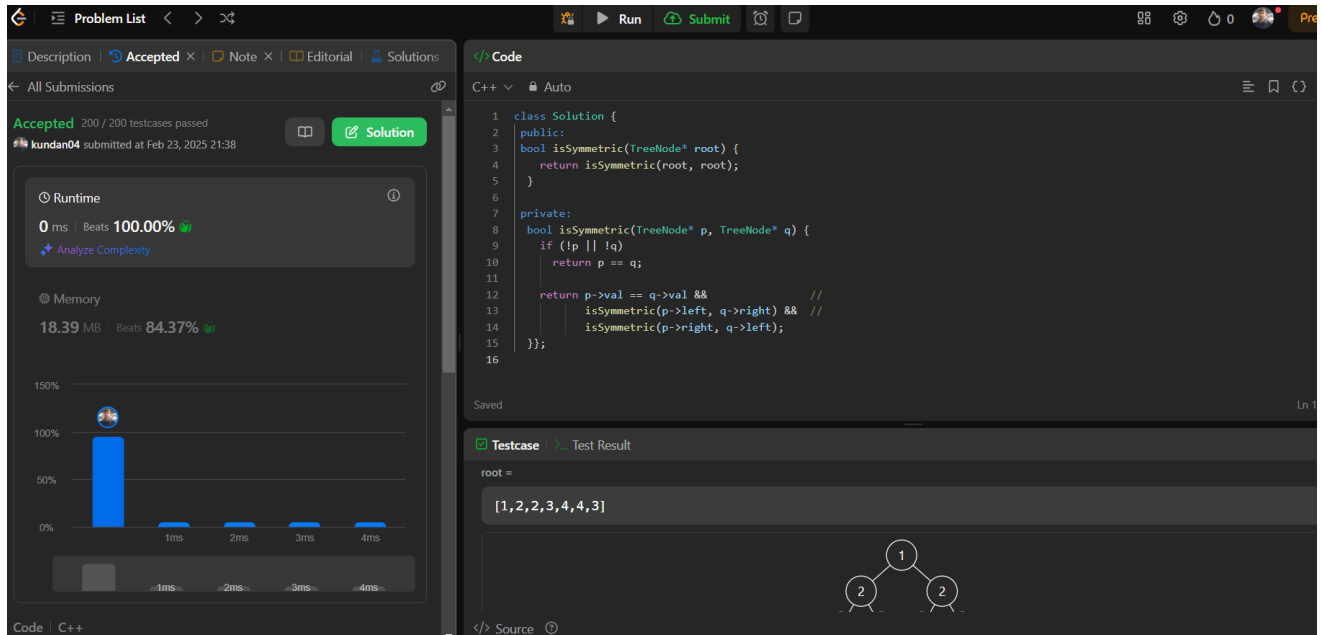
2. **Implementation/Code: Backend:**

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return isSymmetric(root, root);
    }

private:
    bool isSymmetric(TreeNode* p, TreeNode* q) {
        if (!p || !q)
            return p == q;

        return p->val == q->val &&
               isSymmetric(p->left, q->right) &&
               isSymmetric(p->right, q->left);
    }
};
```

3. **Output:**



The screenshot shows a coding platform interface. On the left, the 'Accepted' status is confirmed with 200/200 testcases passed. Performance metrics are displayed: Runtime is 0 ms (Beats 100.00%) and Memory is 18.39 MB (Beats 84.37%). A bar chart shows the user's performance compared to others. The right panel displays the C++ code for the solution. Below the code, a test case is shown with the input array [1, 2, 2, 3, 4, 4, 3] and a corresponding binary tree diagram where the root node 1 has two children, both labeled 2, which are symmetric.

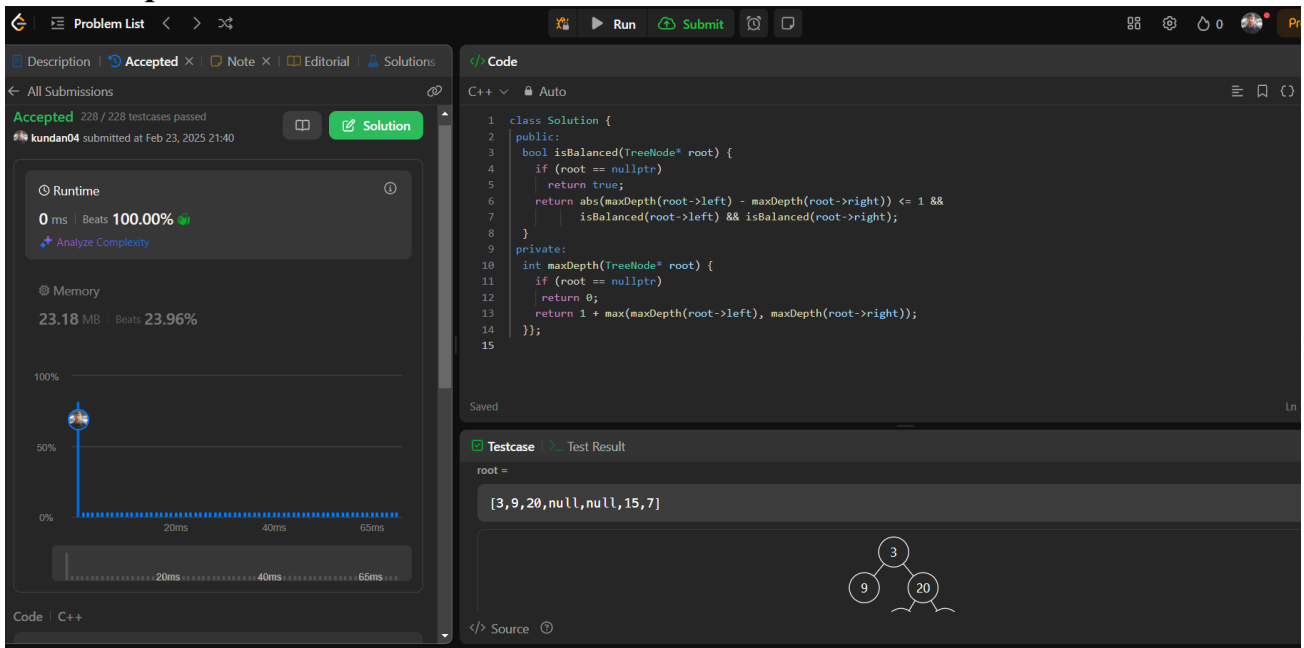
Problem- 3

1. **Aim:** Given a binary tree, determine if it is height - balanced.

2. **Implementation/Code: Backend:**

```
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        if (root == nullptr)
            return true;
        return abs(maxDepth(root->left) - maxDepth(root->right)) <= 1 &&
            isBalanced(root->left) && isBalanced(root->right);
    }
private:
    int maxDepth(TreeNode* root) {
        if (root == nullptr)
            return 0;
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
};
```

3. **Output:**



The screenshot displays a coding platform interface. On the left, the 'Problem List' tab is active, showing 'Accepted 228 / 228 testcases passed' and a submission by 'kundan04' at 'Feb 23, 2025 21:40'. Below this, the 'Runtime' section shows '0 ms | Beats 100.00%' and the 'Memory' section shows '23.18 MB | Beats 23.96%'. A graph shows the runtime performance across 65ms. The right panel shows the 'Code' editor with the C++ solution for the 'isBalanced' problem. The code defines a 'Solution' class with a public 'isBalanced' method and a private 'maxDepth' method. The 'Testcase' section shows the input 'root = [3,9,20,null,null,15,7]' and a corresponding binary tree diagram with root 3, left child 9, and right child 20.

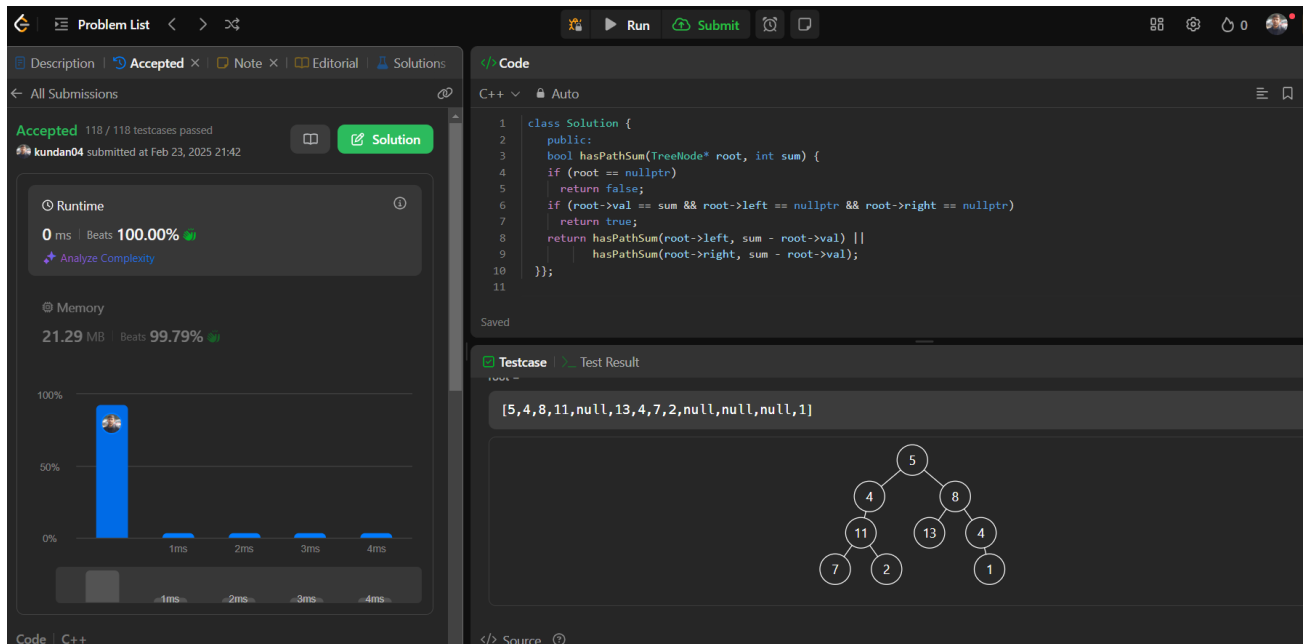
Problem- 4

1. Aim: Given the root of a binary tree and an integer target Sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals target Sum. A leaf is a node with no children

2. Implementation/Code: Backend:

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

3. Output:



The screenshot shows a C++ IDE with the following components:

- Problem List:** Shows the problem is "Accepted" with 118 / 118 testcases passed. The user "kundan04" submitted it on Feb 23, 2025 at 21:42.
- Runtime:** 0 ms, Beats 100.00%.
- Memory:** 21.29 MB, Beats 99.79%.
- Code:** The C++ code for the solution is displayed.
- Testcase:** The input is [5,4,8,11,null,13,4,7,2,null,null,null,1].
- Test Result:** A binary tree diagram is shown, representing the input sequence. The root is 5, with left child 4 and right child 8. Node 4 has left child 11 and right child 13. Node 11 has left child 7 and right child 2. Node 8 has left child 13 and right child 4. Node 13 has left child 1.

Problem- 5

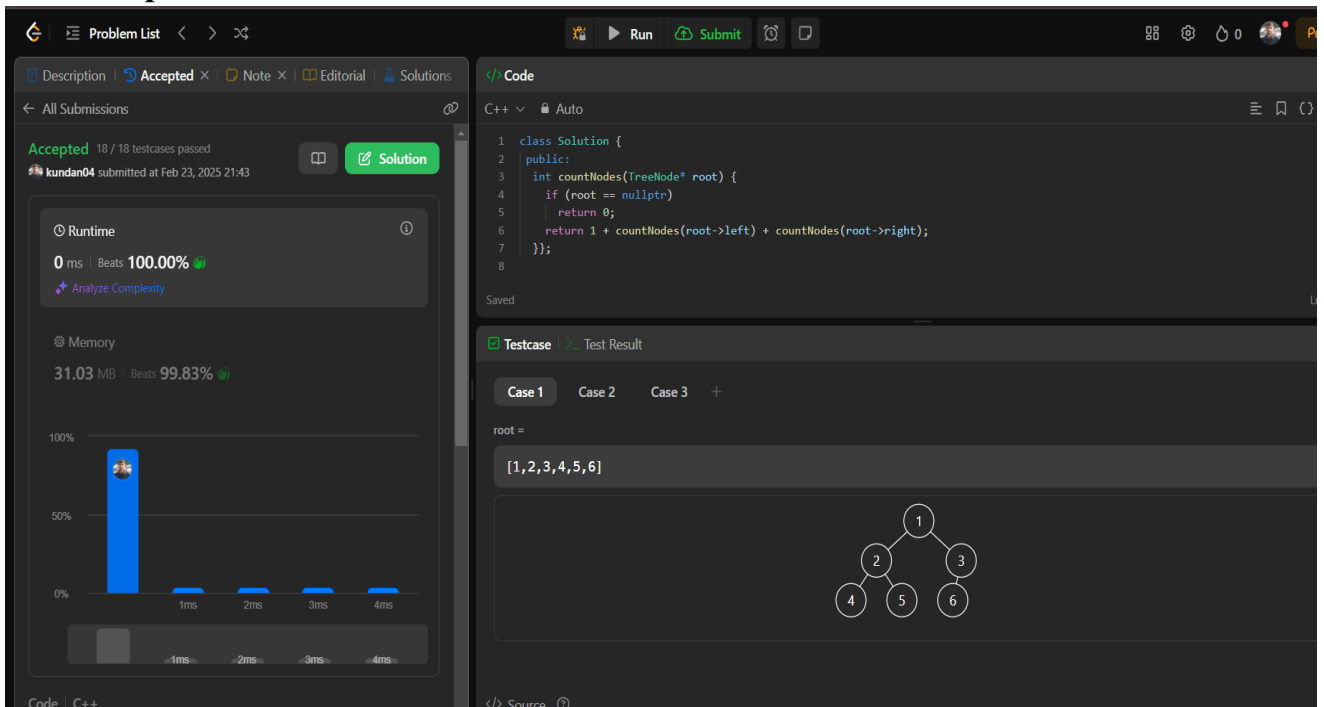
1. Aim:

Given the root of a complete binary tree, return the number of the nodes in the tree. According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h . Design an algorithm that runs in less than $O(n)$ time complexity.

2. Implementation/Code: Backend:

```
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (root == nullptr)
            return 0;
        return 1 + countNodes(root->left) + countNodes(root->right);
    }
};
```

3. Output:



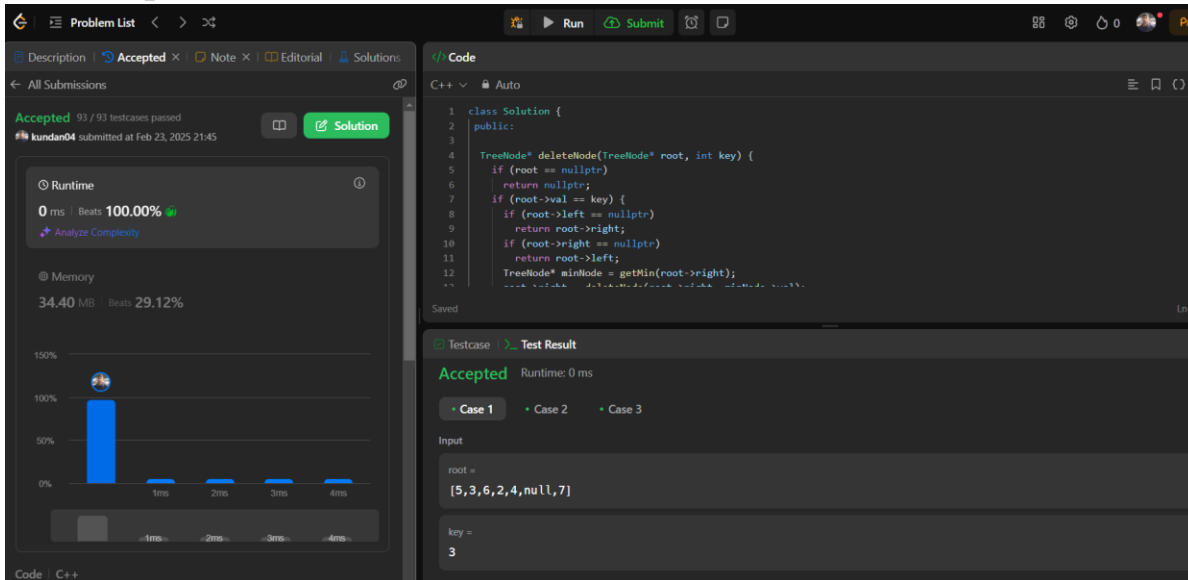
Problem- 6**1. Aim:**

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST. Basically, the deletion can be divided into two stages: Search for a node to remove. If the node is found, delete the node.

2. Implementation/Code: Backend:

```
class Solution {
public:
    TreeNode* deleteNode(TreeNode* root, int key) {
        if (root == nullptr)
            return nullptr;
        if (root->val == key) {
            if (root->left == nullptr)
                return root->right;
            if (root->right == nullptr)
                return root->left;
            TreeNode* minNode = getMin(root->right);
            root->right = deleteNode(root->right, minNode->val);
            minNode->left = root->left;
            minNode->right = root->right;
            root = minNode;
        } else if (root->val < key) {
            root->right = deleteNode(root->right, key);
        } else { // root->val > key
            root->left = deleteNode(root->left, key);
        }
        return root;
    }
private:
    TreeNode* getMin(TreeNode* node) {
        while (node->left != nullptr)
            node = node->left;
        return node;
    }
};
```

3. Output:



Problem- 7

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. 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.

2. Implementation/Code: Backend:

```

class Solution {
public:
    int diameterOfBinaryTree(TreeNode* root) {
        int ans = 0;
        maxDepth(root, ans);
        return ans;
    }

private:
    int maxDepth(TreeNode* root, int& ans) {
        if (root == nullptr)
            return 0;
    }
}

```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
const int l = maxDepth(root->left, ans);  
const int r = maxDepth(root->right, ans);  
ans = max(ans, 1 + r);  
return 1 + max(l, r);  
}  
};
```

3. Output:

The screenshot displays a coding platform interface for a C++ solution. The top navigation bar includes links for Problem List, Accepted, Note, Editorial, and Solutions. The main content area shows the solution status as 'Accepted' with 106/106 testcases passed. The runtime is 0ms (Beats 100.00%) and memory usage is 23.60 MB (Beats 92.78%). The test case input is [1, 2, 3, 4, 5], and the output is a binary tree diagram with root 1 and children 2 and 3.

Runtime: 0ms | Beats 100.00%
Memory: 23.60 MB | Beats 92.78%

Testcase 1: Test Result

Case 1 Case 2 +

root =

[1, 2, 3, 4, 5]

1
2 3