Data Structures and Algorithms

LAB-12 **Issue Date: May 19, 2025**

Start Time: 1:45 PM

Total Marks: 50 **Completion Time: 3:00 PM**

Objective:

In this lab, students will practice the working of AVL trees and BST.

Instructions:

1) Follow the question instructions very carefully, no changes in function prototypes are allowed.
2) Your laptops must be on airplane mode.
3) Anyone caught in an act of plagiarism would be awarded an "F" grade in this Lab.

TASK-01: Split an AVL Tree at a Given Value [20 Marks]

You are given a pointer to the root of an AVL Tree that contains unique integers. You are also given an integer value x.

Your task is to split the AVL Tree into two balanced AVL Trees:

- The left tree should contain all nodes with values **strictly less than x.**
- The right tree should contain all nodes with values **greater than or equal to x.**

Both resulting trees must be **AVL-balanced** and maintain the binary search tree (BST) property.

Function Prototype:

```
void splitAVL(Node* root, int key, Node *&leftTree, Node* &rightTree);
```

Sample Run:

Input :

Key : 10

```
     10
    /  \
   5    15
  / \     \
 2   7     20
```

Output

Left AVL

```
  5
 / \
2   7
```

Right AVL

```
15
  \
   20
```

TASK-02: Isomorphic AVL Trees [15 Marks]

Write a function that takes two binary trees as input and:

1. Checks if the trees are **isomorphic**, allowing any number of left-right swaps.
2. Validates that both trees are **valid AVL trees** (height-balanced with balance factor ∈ {–1, 0, +1} for every node).

Two binary trees are Isomorphic if:

- They are structurally the same, OR
- You can make them structurally the same by **swapping** left and right children at some nodes

The key condition is that:

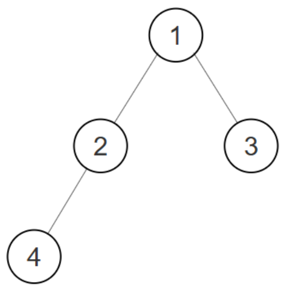
The structure and node values must match after any number of left-right swaps.

Function Prototype:

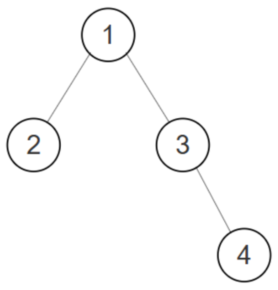
bool areIsomorphicAndAVL(Node root1, Node* root2);*

Sample Run:

Input:



t1



t2

Output: false

TASK-03: BST Compression Level Count [15 Marks]

You are given a Binary Search Tree (BST). Your task is to count the number of levels in the tree where every node at that level has exactly **one child**.

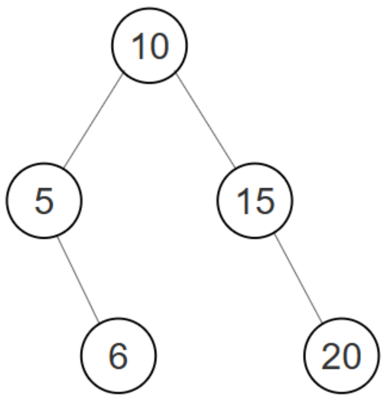
A **compression level** is a level L in the BST such that every node at that level has exactly one child (either left or right, but not both).

Function Prototype:

*int countCompressionLevels(TreeNode *root);*

Sample Run:

Input:



Output: 1

In level 1, both 5 and 15 have only one child. So, only level 1 is compression level.