University of Central Punjab

**Faculty of Information Technology**

**Data Structures and Algorithms**

# Topic

AVL Tree

# Objective

The basic purpose of this lab is to learn and implement AVL Trees — a self-balancing Binary Search Tree where the difference between heights of left and right subtrees cannot be more than one for all nodes.

## Task 1

Implement a class named AVLTree with insertion and deletion functionalities as provided, dry run the insertion and deletion functions of AVL Tree.

// Insert a node

Node insert(Node node, int key) {

if (node == null)

return new Node(key);

if (key < node.key)

node.left = insert(node.left, key);

else if (key > node.key)

node.right = insert(node.right, key);

else // duplicate keys not allowed

return node;

node.height = 1 + Math.max(height(node.left), height(node.right));

int balance = getBalance(node);

// Balancing cases

if (balance > 1 && key < node.left.key)

return rightRotate(node);

if (balance < -1 && key > node.right.key)

return leftRotate(node);

if (balance > 1 && key > node.left.key) {

node.left = leftRotate(node.left);

return rightRotate(node);

}

if (balance < -1 && key < node.right.key) {

node.right = rightRotate(node.right);

return leftRotate(node);

}

return node;

}

Node deleteNode(Node root, int key) {

if (root == null)

return root;

if (key < root.key)

root.left = deleteNode(root.left, key);

else if (key > root.key)

root.right = deleteNode(root.right, key);

else {

if ((root.left == null) || (root.right == null)) {

Node temp = null;

temp = (root.left != null) ? root.left : root.right;

if (temp == null) {

temp = root;

root = null;

} else

root = temp;

} else {

Node temp = minValueNode(root.right);

root.key = temp.key;

root.right = deleteNode(root.right, temp.key);

}

}

if (root == null)

return root;

root.height = Math.max(height(root.left), height(root.right)) + 1;

int balance = getBalance(root);

// Balance tree

if (balance > 1 && getBalance(root.left) >= 0)

return rightRotate(root);

if (balance > 1 && getBalance(root.left) < 0) {

root.left = leftRotate(root.left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root.right) <= 0)

return leftRotate(root);

if (balance < -1 && getBalance(root.right) > 0) {

root.right = rightRotate(root.right);

return leftRotate(root);

}

return root;

}

## Task 2

Create a sample tree using recursive insertNode() calls:  
AVLTree tree = new AVLTree();  
tree.insertNode(30);  
tree.insertNode(20);  
tree.insertNode(40);  
tree.insertNode(10);  
tree.insertNode(25);  
tree.inOrder();  
Perform dry-run on paper and show tree structure.

## Task 3

Display the AVL tree using:  
- Pre-Order Traversal  
- In-Order Traversal  
- Post-Order Traversal

## Task 4

Find Minimum and Maximum in AVL Tree:  
Use leftmost node for min and rightmost for max.

## Task 5

Write a function to compute and return the height of the AVL Tree.

## Task 6

Write functions to count:

* Total number of nodes.
* Number of leaf nodes (nodes with no children).
* Number of internal nodes (nodes with at least one child).