# Assignment – 3

# for

**ADVANCED DATA STRUCTURES**

**AND ALGORITHMS (UNC601)**

## Submitted By: Vishav Singla Roll number: 102115204 Group: 3NC8

**Submitted to**

# Dr. Ram Kishan Dewangan

# Assistant Professor



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB**

**INDIA**

**January – May**

**2024**

**Write a program for AVL tree having functions for the following operations:**

**Insert[1] an element (no duplicates are allowed),**

**Delete[1] an existing element,**

**Traverse the AVL (in-order, pre-order, and post-order),**

**Right rotation, and Left rotation.**

**The program should clearly state the unbalanced node as well as the rotation which is being performed on that node.**

**CODE:**

class AVLNode {

    int key, height;

    AVLNode left, right;

    AVLNode(int d) {

        key = d;

        height = 1;

    }

}

class AVLTree {

    AVLNode root;

    int height(AVLNode N) {

        if (N == null)

            return 0;

        return N.height;

    }

    int max(int a, int b) {

        return (a > b) ? a : b;

    }

**Right rotation**

    AVLNode rightRotate(AVLNode y) {

        AVLNode x = y.left;

        AVLNode T2 = x.right;

        x.right = y;

        y.left = T2;

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

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

        return x;

    }

**Left rotation**

    AVLNode leftRotate(AVLNode x) {

        AVLNode y = x.right;

        AVLNode T2 = y.left;

        y.left = x;

        x.right = T2;

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

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

        return y;

    }

    int getBalance(AVLNode N) {

        if (N == null)

            return 0;

        return height(N.left) - height(N.right);

    }

**Insert an element**

    AVLNode insert(AVLNode node, int key) {

        if (node == null)

            return (new AVLNode(key));

        if (key < node.key)

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

        else if (key > node.key)

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

        else

            return node;

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

        int balance = getBalance(node);

        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;

    }

    AVLNode minValueNode(AVLNode node) {

        AVLNode current = node;

        while (current.left != null)

            current = current.left;

        return current;

    }

**Delete an existing element**

    AVLNode deleteNode(AVLNode 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)) {

                AVLNode temp = null;

                if (temp == root.left)

                    temp = root.right;

                else

                    temp = root.left;

                if (temp == null) {

                    temp = root;

                    root = null;

                } else

                    root = temp;

            } else {

                AVLNode temp = minValueNode(root.right);

                root.key = temp.key;

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

            }

        }

        if (root == null)

            return root;

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

        int balance = getBalance(root);

        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;

    }

**pre-order**

    void preOrder(AVLNode node) {

        if (node != null) {

            System.out.print(node.key + " ");

            preOrder(node.left);

            preOrder(node.right);

        }

    }

**in-order**

    void inOrder(AVLNode node) {

        if (node != null) {

            inOrder(node.left);

            System.out.print(node.key + " ");

            inOrder(node.right);

        }

    }

**post-order**

    void postOrder(AVLNode node) {

        if (node != null) {

            postOrder(node.left);

            postOrder(node.right);

            System.out.print(node.key + " ");

        }

    }

    public static void main(String[] args) {

        AVLTree tree = new AVLTree();

        // Insertion test cases

        tree.root = tree.insert(tree.root, 10);

        tree.root = tree.insert(tree.root, 20);

        tree.root = tree.insert(tree.root, 30);

        tree.root = tree.insert(tree.root, 40);

        tree.root = tree.insert(tree.root, 50);

        tree.root = tree.insert(tree.root, 25);

        // Printing traversals

        System.out.println("Preorder traversal of constructed tree is : ");

        tree.preOrder(tree.root);

        System.out.println();

        System.out.println("Inorder traversal of constructed tree is : ");

        tree.inOrder(tree.root);

        System.out.println();

        System.out.println("Postorder traversal of constructed tree is : ");

        tree.postOrder(tree.root);

        System.out.println();

        // Deletion test cases

        System.out.println("Deleting 30 from the tree...");

        tree.root = tree.deleteNode(tree.root, 30);

        System.out.println("Inorder traversal after deletion : ");

        tree.inOrder(tree.root);

        System.out.println();

        System.out.println("Deleting 25 from the tree...");

        tree.root = tree.deleteNode(tree.root, 25);

        System.out.println("Inorder traversal after deletion : ");

        tree.inOrder(tree.root);

        System.out.println();

        // Test case for left rotation

        System.out.println("Performing left rotation on the tree...");

        tree.root = tree.insert(tree.root, 5);

        System.out.println("Inorder traversal after left rotation : ");

        tree.inOrder(tree.root);

        System.out.println();

        // Test case for right rotation

        System.out.println("Performing right rotation on the tree...");

        tree.root = tree.insert(tree.root, 45);

        System.out.println("Inorder traversal after right rotation : ");

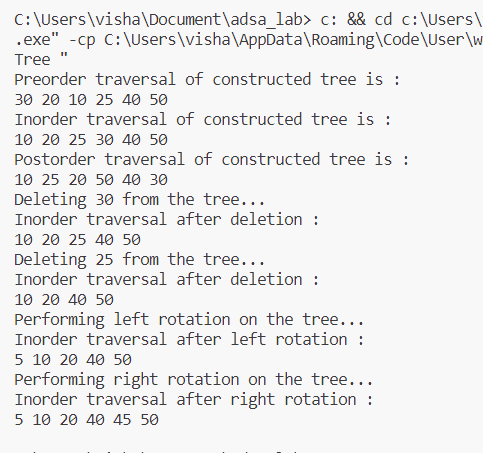
        tree.inOrder(tree.root);

        System.out.println();

    }

}

**OUTPUT:**

****