# Assignment – 1

# 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 binary search tree (BST) having functions for the following operations:**

**Insert an element (no duplicates are allowed),**

**Delete an existing element,**

**Traverse the BST (in-order, pre-order, and post-order), Maximum depth, and Minimum depth.**

**NOTE:**

**Maximum depth: Number of nodes along the longest path from the root node down to the**

**farthest leaf node.**

**Minimum depth: Number of nodes along the shortest path from the root node down to the nearest leaf node.**

**CODE:**

class TreeNode {

    int key;

    TreeNode left, right;

    public TreeNode(int key) {

        this.key = key;

        this.left = this.right = null;

    }

**1. Insert an Element**

    public TreeNode insert(TreeNode root, int key) {

        if (root == null) {

            return new TreeNode(key);

        }

        if (key < root.key) {

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

        } else if (key > root.key) {

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

        }

        return root;

    }

**2. Delete an Element**

    public TreeNode delete(TreeNode root, int key) {

        if (root == null) {

            return root;

        }

        if (key < root.key) {

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

        } else if (key > root.key) {

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

        } else {

            if (root.left == null) {

                return root.right;

            } else if (root.right == null) {

                return root.left;

            }

            root.key = minValue(root.right);

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

        }

        return root;

    }

    private int minValue(TreeNode node) {

        int minValue = node.key;

        while (node.left != null) {

            minValue = node.left.key;

            node = node.left;

        }

        return minValue;

    }

**3. Inorder Traversal**

    public void inOrderTraversal(TreeNode root) {

        if (root != null) {

            inOrderTraversal(root.left);

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

            inOrderTraversal(root.right);

        }

    }

**4. Preorder Traversal**

    public void preOrderTraversal(TreeNode root) {

        if (root != null) {

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

            preOrderTraversal(root.left);

            preOrderTraversal(root.right);

        }

    }

**5. Postorder Traversal**

    public void postOrderTraversal(TreeNode root) {

        if (root != null) {

            postOrderTraversal(root.left);

            postOrderTraversal(root.right);

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

        }

    }

**5. Max depth Traversal**

    public int maxDepth(TreeNode root) {

        if (root == null) {

            return 0;

        } else {

            int leftDepth = maxDepth(root.left);

            int rightDepth = maxDepth(root.right);

            return Math.max(leftDepth, rightDepth) + 1;

        }

    }

**6. Min depth Traversal**

    public int minDepth(TreeNode root) {

        if (root == null) {

            return 0;

        } else {

            int leftDepth = minDepth(root.left);

            int rightDepth = minDepth(root.right);

            return (leftDepth == 0 || rightDepth == 0) ? leftDepth + rightDepth + 1 : Math.min(leftDepth, rightDepth) + 1;

        }

    }

}

public class Main {

    public static void main(String[] args) {

        TreeNode root = null;

        TreeNode treeNode = new TreeNode(0);

        // Inserting elements

        int[] keysToInsert = {10, 5, 15, 3, 7, 12, 18, 8};

        for (int key : keysToInsert) {

            root = treeNode.insert(root, key);

        }

        // In-order traversal

        System.out.println("In-order traversal:");

        treeNode.inOrderTraversal(root);

        System.out.println();

        // Pre-order traversal

        System.out.println("Pre-order traversal:");

        treeNode.preOrderTraversal(root);

        System.out.println();

        // Post-order traversal

        System.out.println("Post-order traversal:");

        treeNode.postOrderTraversal(root);

        System.out.println();

        // Maximum depth

        System.out.println("Maximum Depth: " + treeNode.maxDepth(root));

        // Minimum depth

        System.out.println("Minimum Depth: " + treeNode.minDepth(root));

        // Deleting elements

        int keyToDelete = 10;

        root = treeNode.delete(root, keyToDelete);

        // In-order traversal after deletion

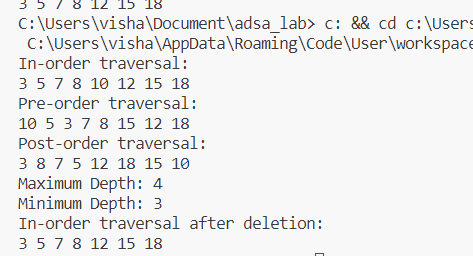
        System.out.println("In-order traversal after deletion:");

        treeNode.inOrderTraversal(root);

    }

}

**OUTPUT:**

****