DataStructures and algorithms

Lab 9

Name:Pranjal bajaj

Cms:023-21-0022

Task 1

class Node {

    int value;

    Node left, right;

    Node(int value) {

        this.value = value;

        left = right = null;

    }

}

class tree {

    Node root;

    tree(int value) {

        root = new Node(value);

    }

    tree() {

        root = null;

    }

    void add(int value) {

        root = addRecursive(root, value);

    }

    Node addRecursive(Node node, int value) {

        if (node == null) {

            return new Node(value);

        }

        if (value < node.value) {

            node.left = addRecursive(node.left, value);

        } else if (value > node.value) {

            node.right = addRecursive(node.right, value);

        }

        return node;

    }

    void inOrderTraversal(Node node) {

        if (node != null) {

            inOrderTraversal(node.left);

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

            inOrderTraversal(node.right);

        }

    }

    void preOrderTraversal(Node node) {

        if (node != null) {

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

            preOrderTraversal(node.left);

            preOrderTraversal(node.right);

        }

    }

    void postOrderTraversal(Node node) {

        if (node != null) {

            postOrderTraversal(node.left);

            postOrderTraversal(node.right);

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

        }

    }

    public static void main(String[] args) {

        tree tree = new tree();

        tree.add(5);

        tree.add(3);

        tree.add(7);

        tree.add(2);

        tree.add(4);

        tree.add(6);

        tree.add(8);

        System.out.print("In-order Traversal: ");

        tree.inOrderTraversal(tree.root);

        System.out.print("\nPre-order Traversal: ");

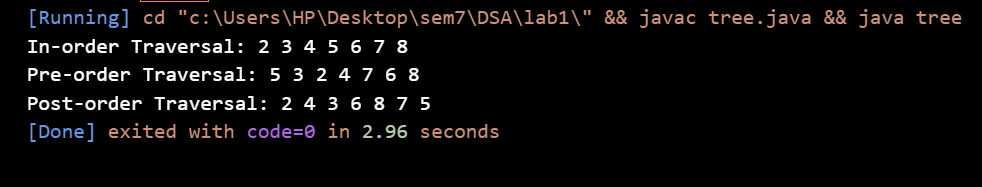
        tree.preOrderTraversal(tree.root);

        System.out.print("\nPost-order Traversal: ");

        tree.postOrderTraversal(tree.root);

    }

}



Task 2

class Node {

    int value;

    Node left, right;

    Node(int value) {

        this.value = value;

        left = right = null;

    }

}

class tree {

    Node root;

    // Constructors

    tree(int value) {

        root = new Node(value);

    }

    tree() {

        root = null;

    }

    // Add a value to the tree

    void add(int value) {

        root = addRecursive(root, value);

    }

    // Helper method for adding a value recursively

    Node addRecursive(Node node, int value) {

        if (node == null) {

            return new Node(value);

        }

        if (value < node.value) {

            node.left = addRecursive(node.left, value);

        } else if (value > node.value) {

            node.right = addRecursive(node.right, value);

        }

        return node;

    }

    // In-order traversal: left, root, right

    void inOrderTraversal(Node node) {

        if (node != null) {

            inOrderTraversal(node.left);

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

            inOrderTraversal(node.right);

        }

    }

    // Pre-order traversal: root, left, right

    void preOrderTraversal(Node node) {

        if (node != null) {

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

            preOrderTraversal(node.left);

            preOrderTraversal(node.right);

        }

    }

    // Post-order traversal: left, right, root

    void postOrderTraversal(Node node) {

        if (node != null) {

            postOrderTraversal(node.left);

            postOrderTraversal(node.right);

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

        }

    }

    public static void main(String[] args) {

        tree tree = new tree();

        // Add nodes to the tree

        tree.add(5);

        tree.add(3);

        tree.add(7);

        tree.add(2);

        tree.add(4);

        tree.add(6);

        tree.add(8);

        // Call the traversal methods and display the output

        System.out.print("In-order Traversal: ");

        tree.inOrderTraversal(tree.root);

        System.out.print("\nPre-order Traversal: ");

        tree.preOrderTraversal(tree.root);

        System.out.print("\nPost-order Traversal: ");

        tree.postOrderTraversal(tree.root);

    }

}



Task 3

class Node {

    int value;

    Node left, right;

    Node(int value) {

        this.value = value;

        left = right = null;

    }

}

class tree {

    Node root;

    tree() {

        root = null;

    }

    void add(int value) {

        root = addRecursive(root, value);

    }

    Node addRecursive(Node node, int value) {

        if (node == null) {

            return new Node(value);

        }

        if (value < node.value) {

            node.left = addRecursive(node.left, value);

        } else if (value > node.value) {

            node.right = addRecursive(node.right, value);

        }

        return node;

    }

    Node searchtree(Node root, int val) {

        if (root == null || root.value == val) {

            return root;

        }

        if (val < root.value) {

            return searchtree(root.left, val);

        }

        return searchtree(root.right, val);

    }

    public static void main(String[] args) {

        tree tree = new tree();

        tree.add(4);

        tree.add(2);

        tree.add(7);

        tree.add(1);

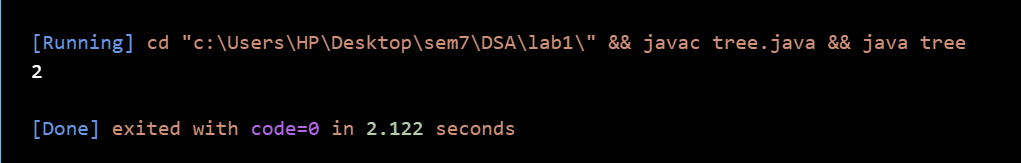
        tree.add(3);

        Node result = tree.searchtree(tree.root, 2);

        System.out.println(result != null ? result.value : "null");

    }

}



Task 4

import java.util.HashMap;

import java.util.Map;

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) {

        this.val = val;

        this.left = this.right = null;

    }

}

class tree {

    private int preorderIndex = 0;

    private Map<Integer, Integer> inorderIndexMap;

    public TreeNode buildTree(int[] preorder, int[] inorder) {

        inorderIndexMap = new HashMap<>();

        for (int i = 0; i < inorder.length; i++) {

            inorderIndexMap.put(inorder[i], i);

        }

        return buildTreeRecursive(preorder, 0, inorder.length - 1);

    }

    private TreeNode buildTreeRecursive(int[] preorder, int left, int right) {

        if (left > right) {

            return null;

        }

        int rootValue = preorder[preorderIndex++];

        TreeNode root = new TreeNode(rootValue);

        int inorderIndex = inorderIndexMap.get(rootValue);

        root.left = buildTreeRecursive(preorder, left, inorderIndex - 1);

        root.right = buildTreeRecursive(preorder, inorderIndex + 1, right);

        return root;

    }

    public static void main(String[] args) {

        tree builder = new tree();

        int[] preorder = {3, 9, 20, 15, 7};

        int[] inorder = {9, 3, 15, 20, 7};

        TreeNode root = builder.buildTree(preorder, inorder);

        printInOrder(root); // Verify the constructed tree

    }

    public static void printInOrder(TreeNode node) {

        if (node == null) return;

        printInOrder(node.left);

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

        printInOrder(node.right);

    }

}

