|  |  |
| --- | --- |
| Student Name | Uzair Hussain |
| Roll Number | 21SW085 |
| Section # | 3rd or III |
| Lab # | 12th – Binary Search Trees |

**Task#01**

**Code: Numeric Binary Search Tree**

class Task1\_Numeric\_Tree {

    static class BinaryTree {

        int key;

        BinaryTree left, right;

        public BinaryTree(int item) {

            key = item;

            left = right = null;

        }

    }

    static BinaryTree root;

    Task1\_Numeric\_Tree() {

        root = null;

    }

    public static int size(BinaryTree root){

        if (root == null){

            return 0;

        }

        int leftNodes=size(root.left);

        int rightNodes=size(root.right);

        return (leftNodes + 1 + rightNodes);    // adding +1 of root

    }

    public int height(BinaryTree root){

        if(root==null){

            return 0;

        }

        int leftsize=0;

        int rightsize=0;

        BinaryTree temp=root;

        while(root.left!=null){

            leftsize++;

            root=root.left;

        }while(temp.right!=null){

            rightsize++;

            temp=temp.right;

        }

        return Math.max(leftsize,rightsize);

    }

    void insert(int key) {

        root = insertRec(root, key);

    }

    BinaryTree insertRec(BinaryTree root, int key) {

        if (root == null) {

            root = new BinaryTree(key);

            return root;

        }

        if (key < root.key)

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

        else if (key > root.key)

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

        return root;

    }

    void inorder() {

        inorderRec(root);

    }

    void inorderRec(BinaryTree root) {

        if (root != null) {

            inorderRec(root.left);

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

            inorderRec(root.right);

        }

    }

    void preorder() {

        preorderRec(root);

    }

    void preorderRec(BinaryTree root) {

        if (root != null) {

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

            preorderRec(root.left);

            preorderRec(root.right);

        }

    }

    void postorder() {

        postorderRec(root);

    }

    void postorderRec(BinaryTree root) {

        if (root != null) {

            postorderRec(root.left);

            postorderRec(root.right);

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

        }

    }

    public boolean isFull(BinaryTree root){

        int nodes=1;

        int power=height(root)+1;

        Math.pow(nodes, power);

        nodes--;

        if(size(root)==nodes){

            return true;

        }

        else{

            return false;

        }

    }

    public static void main(String[] args) {

        Task1\_Numeric\_Tree tree1 = new Task1\_Numeric\_Tree();

        int[] keys = {6, 8, 22, 3, 7, 5, 12, 10, 9, 20, 35, 40, 42};

        for (int key : keys) {

            tree1.insert(key);

        }

        System.out.println("\n\t\t \*\*\*\*\*   Numeric Binary Tree   \*\*\*\*\*\*\*\n\n");

        System.out.println("Size of tree is : "+ tree1.size(root));

        System.out.println("Height of tree is : "+ tree1.height(root));

        System.out.println("Is Full status of tree is : "+tree1.isFull(root));

        System.out.print("Inorder traversal: ");

        tree1.inorder();

        System.out.print("\nPreorder traversal: ");

        tree1.preorder();

        System.out.print("\nPostorder traversal: ");

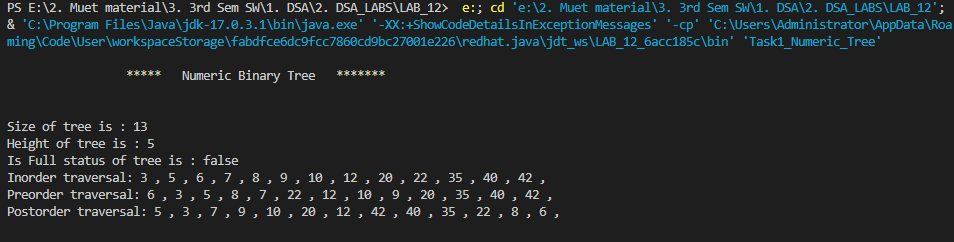
        tree1.postorder();

        System.out.println("\n\n\n");

    }

}

**Output 1:**

****

**Task 2:**

**Code: Alphabetic Binary Search Tree**

class Task2\_Alphabetic\_Tree {

    static class BinaryTree {

        int key2;

        BinaryTree left, right;

        public BinaryTree(int item) {

            key2 = item;

            left = right = null;

        }

    }

    static BinaryTree root2;

    Task2\_Alphabetic\_Tree() {

        root2 = null;

    }

    public static int size(BinaryTree root2){

        if (root2 == null){

            return 0;

        }

        int leftNodes=size(root2.left);

        int rightNodes=size(root2.right);

        return (leftNodes + 1 + rightNodes);    // adding +1 of root

    }

    public int height(BinaryTree root2){

        if(root2==null){

            return 0;

        }

        int leftsize=0;

        int rightsize=0;

        BinaryTree temp=root2;

        while(root2.left!=null){

            leftsize++;

            root2=root2.left;

        }while(temp.right!=null){

            rightsize++;

            temp=temp.right;

        }

        return Math.max(leftsize,rightsize);

    }

    void insert(int key) {

        root2 = insertRec(root2, key);

    }

    BinaryTree insertRec(BinaryTree root, int key) {

        if (root == null) {

            root = new BinaryTree(key);

            return root;

        }

        if (key < root.key2)

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

        else if (key > root.key2)

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

        return root;

    }

    void inorder() {

        inorderRec(root2);

    }

    void inorderRec(BinaryTree root) {

        if (root != null) {

            inorderRec(root.left);

            System.out.print((char)root.key2 + " , ");

            inorderRec(root.right);

        }

    }

    void preorder() {

        preorderRec(root2);

    }

    void preorderRec(BinaryTree root) {

        if (root != null) {

            System.out.print((char)root.key2 + " , ");

            preorderRec(root.left);

            preorderRec(root.right);

        }

    }

    void postorder() {

        postorderRec(root2);

    }

    void postorderRec(BinaryTree root) {

        if (root != null) {

            postorderRec(root.left);

            postorderRec(root.right);

            System.out.print((char)root.key2 + " , ");

        }

    }

    public boolean isFull(BinaryTree root){

        int nodes=1;

        int power=height(root)+1;

        Math.pow(nodes, power);

        nodes--;

        if(size(root)==nodes){

            return true;

        }

        else{

            return false;

        }

    }

    public static void main(String[] args) {

        Task2\_Alphabetic\_Tree tree2 = new Task2\_Alphabetic\_Tree();

        char[] keys2 = {'A','B','C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P'};

        for (int key : keys2) {

            tree2.insert((int)key);

        }

        System.out.println("\n\t\t \*\*\*\*\*   Alphabetic Binary Tree   \*\*\*\*\*\*\*\n\n");

        System.out.println("Size of tree is : "+ tree2.size(root2));

        System.out.println("Height of tree is : "+ tree2.height(tree2.root2));

        System.out.println("Is Full status of tree is : "+tree2.isFull(root2));

        System.out.print("Inorder traversal: ");

        tree2.inorder();

        System.out.print("\nPreorder traversal: ");

        tree2.preorder();

        System.out.print("\nPostorder traversal: ");

        tree2.postorder();

        System.out.println("\n\n");

    }

}

**Output:**

**Graphical user interface

Description automatically generated with low confidence**

**GitHub Repository for all Lab Tasks: (from lab 1 to continue)**

[**https://github.com/UzairHussain193/DSA\_LABS\_21SW**](https://github.com/UzairHussain193/DSA_LABS_21SW)

**The End!**