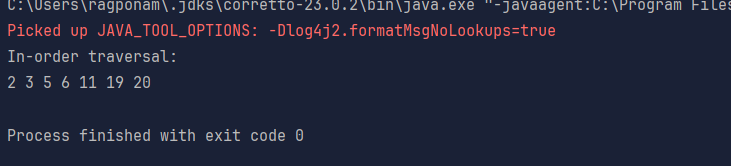
Task1-4:

class **Node**{  
  
 int value;  
 **Node** left;  
 **Node** right;  
 public Node(int value){  
 this.value = value;  
 this.left=null;  
 this.right=null;  
 }  
}  
class **Tree**{  
 **Node** root;  
 public Tree(){  
 root=null;  
 }  
 public void insert(int value){  
 root = insertRec(root, value);  
 }  
 private **Node** insertRec(**Node** root, int value){  
 if (root==null){  
 root=new Node(value);  
 return root;  
 }  
 if (value< root.value){  
 root.left = insertRec(root.left, value);  
  
 }  
 else if (value> root.value){  
 root.right = insertRec(root.right, value);  
  
 }  
 return root;  
 }  
 public void inorder() {  
 inorderRec(root);  
 **System**.*out*.println();  
 }  
  
 private void inorderRec(**Node** root) {  
 if (root != null) {  
 inorderRec(root.left);  
 **System**.*out*.print(root.value + " ");  
 inorderRec(root.right);  
 }  
 }  
  
 }  
 public class **Task1to4** {  
 public static void main(**String**[] args) {  
 **Tree** tree = new Tree();  
 tree.insert(3);  
 tree.insert(3);  
 tree.insert(5);  
 tree.insert(2);  
 tree.insert(11);  
 tree.insert(6);  
 tree.insert(20);  
 tree.insert(19);  
 **System**.*out*.println("In-order traversal: ");  
  
 tree.inorder();  
 }  
}



Task5:

Applications of trees:

Hierarchal data representation

Binary search

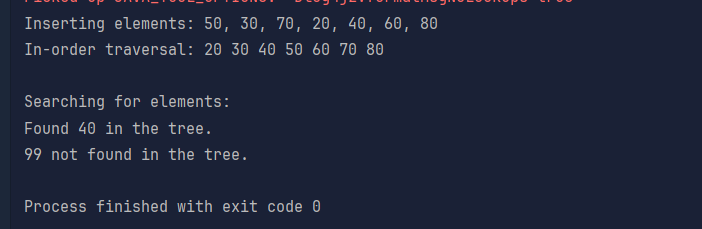
Syntax trees

Network routing algorithms and spanning trees

Decision making trees

Task6:

class **TreeNode** {  
 int item;  
 **TreeNode** left, right;  
  
 TreeNode(int item) {  
 this.item = item;  
 left = right = null;  
 }  
}  
  
class **BinarySearchTreeOp02** {  
 **TreeNode** root;  
  
 public BinarySearchTreeOp02() {  
 this.root = null;  
 }  
  
 public void insert(int item) {  
 root = insertRec(root, item);  
 }  
  
 private **TreeNode** insertRec(**TreeNode** root, int item) {  
 if (root == null) {  
 root = new TreeNode(item);  
 return root;  
 }  
  
 if (item < root.item) {  
 root.left = insertRec(root.left, item);  
 } else if (item > root.item) {  
 root.right = insertRec(root.right, item);  
 }  
  
 return root;  
 }  
  
 public **TreeNode** search(int key) {  
 **TreeNode** current = root;  
 while (current != null) {  
 if (key == current.item) {  
 return current;  
 } else if (key < current.item) {  
 current = current.left;  
 } else {  
 current = current.right;  
 }  
 }  
 return null;  
 }  
  
 public void inorder() {  
 inorderRec(root);  
 **System**.*out*.println();  
 }  
  
 private void inorderRec(**TreeNode** root) {  
 if (root != null) {  
 inorderRec(root.left);  
 **System**.*out*.print(root.item + " ");  
 inorderRec(root.right);  
 }  
 }  
  
 public static void main(**String**[] args) {  
 **BinarySearchTreeOp02** bst = new BinarySearchTreeOp02();  
  
 **System**.*out*.println("Inserting elements: 50, 30, 70, 20, 40, 60, 80");  
 bst.insert(50);  
 bst.insert(30);  
 bst.insert(70);  
 bst.insert(20);  
 bst.insert(40);  
 bst.insert(60);  
 bst.insert(80);  
  
 **System**.*out*.print("In-order traversal: ");  
 bst.inorder();  
  
 **System**.*out*.println("**\n**Searching for elements:");  
 int searchKey1 = 40;  
 **TreeNode** result1 = bst.search(searchKey1);  
 if (result1 != null) {  
 **System**.*out*.println("Found " + searchKey1 + " in the tree.");  
 } else {  
 **System**.*out*.println(searchKey1 + " not found in the tree.");  
 }  
  
 int searchKey2 = 99;  
 **TreeNode** result2 = bst.search(searchKey2);  
 if (result2 != null) {  
 **System**.*out*.println("Found " + searchKey2 + " in the tree.");  
 } else {  
 **System**.*out*.println(searchKey2 + " not found in the tree.");  
 }  
 }  
}



Task7:

Full Binary tree

Complete Binary tree

Perfect Binary tree

Skewed Binary tree

Balanced Binary tree

Task8:

Social Networks

Mapping Navigation

Computer Networks

World Wide Web

Logistics and Supply chain

Scheduling

Resource allocation

Biology and genetics

Chemical structures

Task9:

Undirected graph

Directed graph

Weighted graph

Unweighted graphs

Cyclic Graph

Acyclic graph

Task10:

class **Graph01** {  
 class **Edge** {  
 int src, dest;  
 }  
 int vertices, edges;  
  
 **Edge**[] edge;  
  
 Graph01(int vertices, int edges) {  
 this.vertices = vertices;  
 this.edges = edges;  
  
 edge = new Edge[edges];  
 for(int i = 0; i < edges; i++) {  
 edge[i] = new Edge();  
 }  
 }  
  
 public static void main(**String**[] args) {  
 int noVertices = 5;  
 int noEdges = 8;  
 **Graph01** gObj = new Graph01(noVertices, noEdges);   
  
 gObj.edge[0].src = 1;  
 gObj.edge[0].dest = 2;  
 gObj.edge[1].src = 1;  
 gObj.edge[1].dest = 3;  
 gObj.edge[2].src = 1;  
 gObj.edge[2].dest = 4;  
 gObj.edge[3].src = 2;  
 gObj.edge[3].dest = 4;  
 gObj.edge[4].src = 2;  
 gObj.edge[4].dest = 5;  
 gObj.edge[5].src = 3;  
 gObj.edge[5].dest = 4;  
 gObj.edge[6].src = 3;  
 gObj.edge[6].dest = 5;  
 gObj.edge[7].src = 4;  
 gObj.edge[7].dest = 5;  
  
 for(int i =0; i < noEdges; i++) {  
 **System**.*out*.println(gObj.edge[i].src+ " - " + gObj.edge[i].dest);  
 }  
 }  
}

