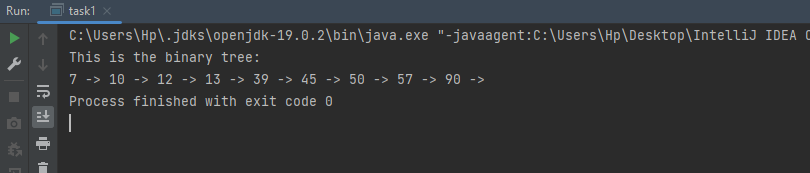
**22k-5195 lab 8**

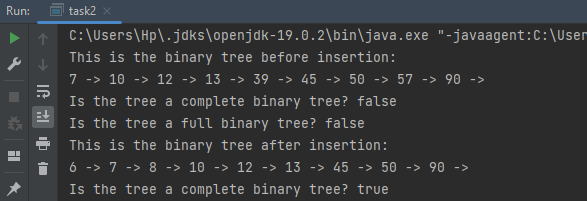
Task 1

class Node {  
 int data;  
 Node left, right;  
  
 public Node(int item) {  
 data = item;  
 left = right = null;  
 }  
}  
public class task1 {  
 Node root;  
  
 task1(){  
 root = null;  
 }  
 void insert(int data) {  
  
 root = insertVal(root, data);  
 }  
  
 Node insertVal(Node root, int data) {  
 if (root == null) {  
 root = new Node(data);  
 return root;  
 }  
  
 if (data < root.data)  
 root.left = insertVal(root.left, data);  
  
 else if (data > root.data)  
 root.right = insertVal(root.right, data);  
  
 return root;  
  
 }  
 void inorder() {  
 inorderVisit(root);  
 }  
  
 void inorderVisit(Node root) {  
 if (root != null) {  
 inorderVisit(root.left);  
 System.*out*.print(root.data + " -> ");  
 inorderVisit(root.right);  
 }  
 }  
 public static void main(String[] args) {  
 int[] a = {45, 10, 7, 90, 12, 50, 13, 39, 57};  
  
 task1 tree = new task1();  
 for(int x : a){  
 tree.insert(x);  
 }  
  
 System.*out*.println("This is the binary tree: ");  
 tree.inorderVisit(tree.root);  
}  
  
  
  
  
}



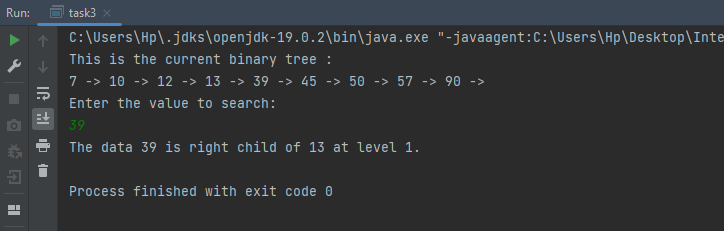
Task 2:

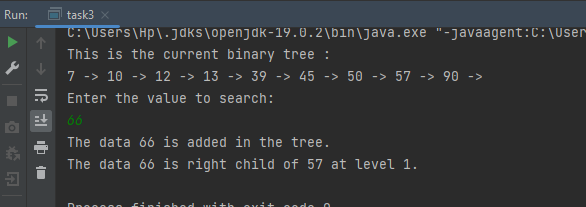
class Node {  
 int data;  
 Node left, right;  
  
 public Node(int item) {  
 data = item;  
 left = right = null;  
 }  
}  
public class task2 {  
 Node root;  
  
 task2(){  
 root = null;  
 }  
 void insert(int data) {  
  
 root = insertVal(root, data);  
 }  
  
 Node insertVal(Node root, int data) {  
 if (root == null) {  
 root = new Node(data);  
 return root;  
 }  
  
 if (data < root.data)  
 root.left = insertVal(root.left, data);  
  
 else if (data > root.data)  
 root.right = insertVal(root.right, data);  
  
 return root;  
  
 }  
 boolean isCompleteBT(Node root, int i, int Count) {  
 if (root == null)  
 return true;  
  
 if (i >= Count)  
 return false;  
  
 return (isCompleteBT(root.left, 2 \* i + 1, Count) &&  
 isCompleteBT(root.right, 2 \* i + 2, Count));  
 }  
 int countNode(Node root) {  
 if (root == null)  
 return 0;  
 return 1 + countNode(root.left) + countNode(root.right);  
 }  
  
 boolean isFullBT(Node root) {  
 if (root == null)  
 return true;  
  
 if (root.left == null && root.right == null)  
 return true;  
  
 if (root.left != null && root.right != null)  
 return (isFullBT(root.left) && isFullBT(root.right));  
  
 return false;  
 }  
 Node deleteNodes(Node root, int index, int nodeCount) {  
 if (root == null)  
 return null;  
  
 root.left = deleteNodes(root.left, 2 \* index + 1, nodeCount);  
 root.right = deleteNodes(root.right, 2 \* index + 2, nodeCount);  
  
 if (index >= nodeCount) {  
 return null; // Deleting the nodes that exceed the count  
 }  
  
 return root;  
 }  
  
 void inorder() {  
 inorderVisit(root);  
 }  
  
 void inorderVisit(Node root) {  
 if (root != null) {  
 inorderVisit(root.left);  
 System.*out*.print(root.data + " -> ");  
 inorderVisit(root.right);  
 }  
 }  
 public static void main(String[] args) {  
 int[] a = {45, 10, 7, 90, 12, 50, 13, 39, 57};  
  
 task2 tree = new task2();  
 for(int x : a){  
 tree.insert(x);  
 }  
  
 System.*out*.println("This is the binary tree before insertion: ");  
 tree.inorderVisit(tree.root);  
 System.*out*.println();  
  
 tree.insert(6);  
 tree.insert(8);  
 tree.insert(9);  
  
 int nodeCount = tree.countNode(tree.root);  
  
 System.*out*.println("Is the tree a complete binary tree? " + tree.isCompleteBT(tree.root, 0, nodeCount));  
 System.*out*.println("Is the tree a full binary tree? " + tree.isFullBT(tree.root));  
  
  
  
 if (!tree.isCompleteBT(tree.root, 0, nodeCount) || !tree.isFullBT(tree.root)) {  
 tree.root = tree.deleteNodes(tree.root, 0, nodeCount);  
 }  
  
 System.*out*.println("This is the binary tree after insertion:");  
 tree.inorderVisit(tree.root);  
 System.*out*.println();  
  
 System.*out*.println("Is the tree a complete binary tree? " + tree.isCompleteBT(tree.root, 0, nodeCount));  
 System.*out*.println("Is the tree a full binary tree? " + tree.isFullBT(tree.root));  
  
  
  
 }  
  
  
  
  
  
}



Task 3

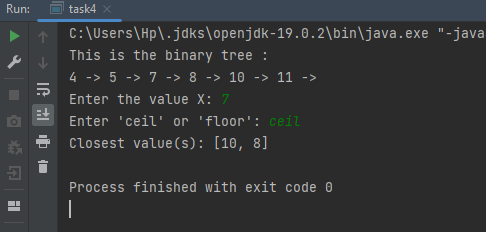
import java.util.Scanner;  
  
class Node {  
 int data;  
 Node left, right;  
  
 public Node(int item) {  
 data = item;  
 left = right = null;  
 }  
}  
public class task3 {  
 Node root;  
  
 task3(){  
 root = null;  
 }  
 void insert(int data) {  
  
 root = insertVal(root, data);  
 }  
  
 Node insertVal(Node root, int data) {  
 if (root == null) {  
 root = new Node(data);  
 return root;  
 }  
  
 if (data < root.data)  
 root.left = insertVal(root.left, data);  
  
 else if (data > root.data)  
 root.right = insertVal(root.right, data);  
  
 return root;  
  
 }  
 Node searchVal(int val, Node root) {  
 if (root == null || root.data == val) {  
 if (root != null && root.data == val) {  
 return root;  
 }  
 return null;  
 }  
  
 if (val < root.data) {  
 return searchVal(val, root.left);  
 } else {  
 return searchVal(val, root.right);  
 }  
 }  
 void Location(int val, Node root, int level) {  
 Node node = searchVal(val, root);  
 if (node == null) {  
 insert(val);  
 node = searchVal(val, root);  
 System.*out*.println("The data " + val + " is added in the tree.");  
  
 }  
  
 if (node == root) {  
 System.*out*.println("The data " + val + " is the root of the tree at level " + level + ".");  
 } else {  
 Node p = findP(root, node);  
 if (p.left == node) {  
 System.*out*.println("The data " + val + " is left child of " + p.data + " at level " + (level + 1) + ".");  
 } else {  
 System.*out*.println("The data " + val + " is right child of " + p.data + " at level " + (level + 1) + ".");  
 }  
 }  
 }  
 Node findP(Node root, Node node) {  
 if (root == null || node == null || root == node) {  
 return null;  
 }  
  
 if ((root.left == node) || (root.right == node)) {  
 return root;  
 }  
  
 Node left = findP(root.left, node);  
 if (left != null) {  
 return left;  
 }  
  
 return findP(root.right, node);  
 }  
  
  
  
 void inorderVisit(Node root) {  
 if (root != null) {  
 inorderVisit(root.left);  
 System.*out*.print(root.data + " -> ");  
 inorderVisit(root.right);  
 }  
 }  
 public static void main(String[] args) {  
 int[] a = {45, 10, 7, 90, 12, 50, 13, 39, 57};  
  
 task3 tree = new task3();  
 for(int x : a){  
 tree.insert(x);  
 }  
  
 System.*out*.println("This is the current binary tree : ");  
 tree.inorderVisit(tree.root);  
 System.*out*.println();  
  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.println("Enter the value to search:");  
 int n = scan.nextInt();  
  
 tree.Location(n, tree.root, 0);  
  
 }  
}





Task 4:

import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
class Node {  
 int data;  
 Node left, right;  
  
 public Node(int value) {  
 data = value;  
 left = null;  
 right = null;  
 }  
}  
  
public class task4 {  
 Node root;  
 task4() {  
 root = null;  
 }  
  
 void insert(int value) {  
 root = insertVal(root, value);  
 }  
 Node insertVal(Node root, int value) {  
 if (root == null) {  
 root = new Node(value);  
 return root;  
 }  
  
 if (value < root.data) {  
 root.left = insertVal(root.left, value);  
 } else if (value > root.data) {  
 root.right = insertVal(root.right, value);  
 }  
  
 return root;  
 }  
  
 void findClosestValue(int x, String operation) {  
 List<Integer> closestValues = new ArrayList<>();  
 findClosestValue(root, x, operation, closestValues);  
 if (closestValues.isEmpty()) {  
 System.*out*.println("No closest value found.");  
 } else {  
 System.*out*.println("Closest value(s): " + closestValues);  
 }  
 }  
  
 void findClosestValue(Node root, int x, String operation, List<Integer> closestValues) {  
 if (root == null) {  
 return;  
 }  
 if (operation.equalsIgnoreCase("ceil")) {  
 if (root.data == x + 1) {  
 closestValues.add(root.data);  
 } else if (root.data > x + 1) {  
 closestValues.add(root.data);  
 findClosestValue(root.left, x, operation, closestValues);  
 } else {  
 findClosestValue(root.right, x, operation, closestValues);  
 }  
 } else if (operation.equalsIgnoreCase("floor")) {  
 if (root.data == x - 1) {  
 closestValues.add(root.data);  
 } else if (root.data < x - 1) {  
 closestValues.add(root.data);  
 findClosestValue(root.right, x, operation, closestValues);  
 } else {  
 findClosestValue(root.left, x, operation, closestValues);  
 }  
 }  
 }  
 void inorderVisit(Node root) {  
 if (root != null) {  
 inorderVisit(root.left);  
 System.*out*.print(root.data + " -> ");  
 inorderVisit(root.right);  
 }  
 }  
 public static void main(String[] args) {  
 Scanner a = new Scanner(System.*in*);  
 task4 tree = new task4();  
  
 int[] data = {10, 5, 11, 4, 7, 8};  
 for (int i : data) {  
 tree.insert(i);  
 }  
 System.*out*.println("This is the binary tree : ");  
 tree.inorderVisit(tree.root);  
 System.*out*.println();  
  
  
 System.*out*.print("Enter the value X: ");  
 int x = a.nextInt();  
 a.nextLine();  
  
 System.*out*.print("Enter 'ceil' or 'floor': ");  
 String choice = a.nextLine();  
  
 tree.findClosestValue(x, choice);  
  
 }  
  
}



Task 5

class Node {  
 int data;  
 Node left, right;  
  
 public Node(int value) {  
 data = value;  
 left = null;  
 right = null;  
 }  
}  
  
public class task5 {  
  
 Node root;  
  
 task5() {  
 root = null;  
 }  
  
 void insert(int value) {  
 root = insertVal(root, value);  
 }  
  
 Node insertVal(Node root, int value) {  
 if (root == null) {  
 root = new Node(value);  
 return root;  
 }  
  
 if (value < root.data) {  
 root.left = insertVal(root.left, value);  
 } else if (value > root.data) {  
 root.right = insertVal(root.right, value);  
 }  
  
 return root;  
 }  
  
 Node mergeTrees(Node root1, Node root2) {  
 return merge(root1, root2);  
 }  
  
 Node merge(Node root1, Node root2) {  
 if (root1 == null) {  
 return root2;  
 }  
 if (root2 == null) {  
 return root1;  
 }  
  
 if (root1.data < root2.data) {  
 root1.right = merge(root1.right, root2);  
 return root1;  
 } else {  
 root2.left = merge(root1, root2.left);  
 return root2;  
 }  
 }  
  
 public void inOrderTraversal(Node root) {  
 if (root != null) {  
 inOrderTraversal(root.left);  
 System.*out*.print(root.data + " ");  
 inOrderTraversal(root.right);  
 }  
 }  
  
  
 public static void main(String[] args) {  
  
 task5 Bst1 = new task5();  
 int[] values1 = {5, 3, 2, 4, 6};  
 for (int value : values1) {  
 Bst1.insert(value);  
 }  
 task5 Bst2 = new task5();  
 int[] values2 = {2, 1, 3, 7, 6};  
 for (int value : values2) {  
 Bst2.insert(value);  
 }  
 Node mergedRoot = Bst1.mergeTrees(Bst1.root, Bst2.root);  
  
 System.*out*.println("Merged BST in preorder :");  
  
 *PreOrder*(mergedRoot);  
  
 System.*out*.println();  
 System.*out*.println("Merged BST in inorder :");  
 *InOrder*(mergedRoot);  
  
 System.*out*.println();  
  
 System.*out*.println("Merged BST in postorder :");  
 *PostOrder*(mergedRoot);  
 }  
  
  
 static void PreOrder(Node root) {  
 if (root == null)  
 return;  
 System.*out*.print(root.data + " ");  
 *PreOrder*(root.left);  
 *PreOrder*(root.right);  
 }  
  
 static void InOrder(Node root) {  
 if (root == null)  
 return;  
 *InOrder*(root.left);  
 System.*out*.print(root.data + " ");  
 *InOrder*(root.right);  
 }  
 static void PostOrder(Node root) {  
 if (root == null)  
 return;  
 *PostOrder*(root.left);  
 *PostOrder*(root.right);  
 System.*out*.print(root.data);  
  
 }  
  
}

