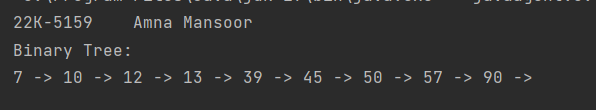
22K-5159 Amna Mansoor BSE-3B DS LAB-8

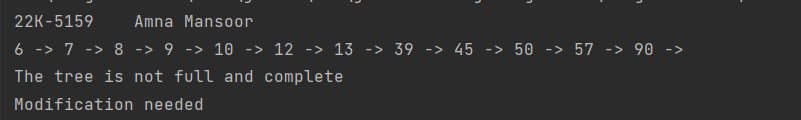
Task 1:

public class Task1 {  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 BST bst=new BST();  
 int[] array={45,10,7,90,12,50,13,39,57};  
 for (int i = 0; i < array.length; i++) {  
 bst.insert(array[i]);  
 }  
 System.*out*.println("Binary Tree: ");  
 bst.inorder();  
  
 }  
}  
class Node{  
 int data;  
 Node left, right;  
 public Node(int item){  
 data=item;  
 left=right=null;  
 }  
  
}  
class BST{  
 Node root;  
 public BST(){  
 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);  
 System.*out*.println();  
 }  
 void inorderVisit(Node root){  
 if(root!=null){  
 inorderVisit(root.left);  
 System.*out*.print(root.data+ " -> ");  
 inorderVisit(root.right);  
 }  
 }  
}

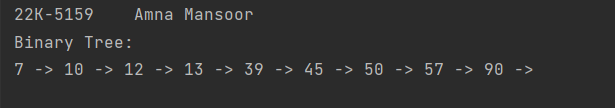


Task 2:

public class Task2 {  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 BST2 bst=new BST2();  
 int[] array={45,10,7,90,12,50,13,39,57};  
 for (int i = 0; i < array.length; i++) {  
 bst.insert(array[i]);  
 }  
  
 bst.insert(6);  
 bst.insert(8);  
 bst.insert(9);  
 bst.inorder();  
  
 if (bst.isTreeFull(bst.root)&& bst.isComplete()) {  
 System.*out*.println("The tree is full and complete");  
 }  
 else{  
 System.*out*.println("The tree is not full and complete");  
 System.*out*.println("Modification needed");  
 bst.makeFullAndComplete();  
 System.*out*.println("After modification");  
 bst.inorder();  
 }  
  
 }  
}  
class Node2{  
 int data;  
 Node2 left, right;  
 public Node2(int item){  
 data=item;  
 left=right=null;  
 }  
  
}  
class BST2{  
 Node2 root;  
 public BST2(){  
 root=null;  
 }  
  
 void insert(int data){  
 root=insertVal(root,data);  
 }  
 Node2 insertVal(Node2 root, int data){  
 if (root==null){  
 root=new Node2(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);  
 System.*out*.println();  
 }  
 void inorderVisit(Node2 root){  
 if(root!=null){  
 inorderVisit(root.left);  
 System.*out*.print(root.data+ " -> ");  
 inorderVisit(root.right);  
 }  
 }  
 public boolean isTreeFull(Node2 node){  
 if(node==null){  
 return true;  
 }  
 if(node.left==null&&node.right==null){  
 return true;  
 }  
 if((node.left!=null)&&(node.right!=null)){  
 return (isTreeFull(node.left)&&isTreeFull(node.right));  
 }  
 return false;  
 }  
 public boolean isComplete(){  
 int index=0;  
 int size=getSize(this.root);  
 return isComplete(this.root,index,size);  
 }  
 private boolean isComplete(Node2 node, int index,int size){  
 if(node==null){  
 return true;  
 }  
 if(index>=size){  
 return false;  
 }  
 return (isComplete(node.left,2\*index+1,size)&&isComplete(node.right,2\*index+2,size));  
 }  
 private int getSize(Node2 node){  
 if(node==null){  
 return 0;  
 }  
 return 1+getSize(node.left)+getSize(node.right);  
 }  
 public void makeFullAndComplete(){  
 int size=getSize(root);  
 Node2[] nodeArray=new Node2[size];  
 int index=0;  
 storeInArray(root,nodeArray,index);  
 root=buildTree(nodeArray,0,size-1);  
 }  
 private void storeInArray(Node2 node, Node2[] nodeArray, int index){  
 if(node!=null){  
 nodeArray[index]=node;  
 storeInArray(node.left,nodeArray,2\*index+1);  
 storeInArray(node.right,nodeArray,2\*index+2);  
 }  
 }  
 private Node2 buildTree(Node2[] nodeArray, int start, int end){  
 if(start>end){  
 return null;  
 }  
 int mid=(start+end)/2;  
 Node2 node=nodeArray[mid];  
 node.left=buildTree(nodeArray,start,mid-1);  
 node.right=buildTree(nodeArray,mid+1,end);  
 return node;  
 }  
}

  
  
Task 3:

import java.util.Scanner;  
  
public class Task3 {  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 BST3 bst=new BST3();  
 int[] array= new int[]{45, 10, 7, 90, 12, 50, 13, 39, 57};  
 for (int i = 0; i < array.length; i++) {  
 bst.insert(array[i]);  
 }  
 System.*out*.println("Binary Tree: ");  
 bst.inorder();  
 System.*out*.println("\nEnter a value to be searched: ");  
 Scanner sc=new Scanner(System.*in*);  
 int key= sc.nextInt();  
  
 int level=bst.getLevel(bst.root,key);  
 if(level!=0) {  
 System.*out*.println("\nNode: " + key + " found at LEVEL= " + bst.getLevel(bst.root, key));  
 }  
 else {  
 bst.insert(key);  
 System.*out*.println("New Tree: ");  
 bst.inorder();  
 System.*out*.println("New node placed at level: "+bst.getLevel(bst.root, key));  
 }  
 }  
}  
class Node3{  
 int data;  
 Node3 left, right;  
 public Node3(int item){  
 data=item;  
 left=right=null;  
 }  
  
}  
class BST3{  
 Node3 root;  
 public BST3(){  
 root=null;  
 }  
  
 void insert(int data){  
 root=insertVal(root,data);  
 }  
 Node3 insertVal(Node3 root, int data){  
 if (root==null){  
 root=new Node3(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);  
 System.*out*.println();  
 }  
 void inorderVisit(Node3 root){  
 if(root!=null){  
 inorderVisit(root.left);  
 System.*out*.print(root.data+ " -> ");  
 inorderVisit(root.right);  
 }  
 }  
  
 public int getLevel(Node3 node, int data) {  
 return getLevelNum(node,data,0);  
 }  
  
 int getLevelNum(Node3 node, int data, int level) {  
 if(node==null){  
 return 0;  
 }  
 if(node.data==data){  
 return level;  
 }  
 int downlevel=getLevelNum(node.left,data,level+1);  
 if(downlevel!=0){  
 return downlevel;  
 }  
 downlevel=getLevelNum(node.right,data,level+1);  
 return downlevel;  
 }  
}



Task 4:

import java.util.LinkedList;  
import java.util.Queue;  
  
class Node4 {  
 int data;  
 Node4 left, right;  
  
 public Node4(int data) {  
 this.data = data;  
 left = right = null;  
 }  
}  
  
public class Task4 {  
 Node4 root;  
  
 public Task4() {  
 root = null;  
 }  
  
 // Function to find the closest value in the tree based on ceil or floor  
 public void findClosestValue(int x, boolean isCeil) {  
 if (root == null) {  
 System.*out*.println("The tree is empty.");  
 return;  
 }  
  
 int closest = Integer.*MAX\_VALUE*;  
 Queue<Node4> queue = new LinkedList<>();  
 queue.offer(root);  
  
 while (!queue.isEmpty()) {  
 Node4 current = queue.poll();  
  
 // Update closest value based on the chosen operation (ceil or floor)  
 if (isCeil) {  
 if (current.data >= x && current.data < closest) {  
 closest = current.data;  
 }  
 } else {  
 if (current.data <= x && current.data > closest) {  
 closest = current.data;  
 }  
 }  
  
 // Enqueue left and right children if they exist  
 if (current.left != null) {  
 queue.offer(current.left);  
 }  
 if (current.right != null) {  
 queue.offer(current.right);  
 }  
 }  
  
 System.*out*.println("Closest value: " + closest);  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 Task4 tree = new Task4();  
 tree.root = new Node4(10);  
 tree.root.left = new Node4(5);  
 tree.root.right = new Node4(11);  
 tree.root.left.left = new Node4(4);  
 tree.root.left.right = new Node4(7);  
 tree.root.right.right = new Node4(8);  
  
 int x = 6; // Value to find closest to  
 boolean isCeil = true; // true for ceil, false for floor  
  
 tree.findClosestValue(x, isCeil);  
 }  
}



Task 5:

class Node5 {  
 int data;  
 Node5 left, right;  
  
 public Node5(int data) {  
 this.data = data;  
 left = right = null;  
 }  
}  
  
public class Task5 {  
 Node5 root;  
  
 public Task5() {  
 root = null;  
 }  
  
 // Function to merge two sorted arrays into a single sorted array  
 public int[] mergeSortedArrays(int[] arr1, int[] arr2) {  
 int n = arr1.length;  
 int m = arr2.length;  
 int[] result = new int[n + m];  
  
 int i = 0, j = 0, k = 0;  
  
 while (i < n && j < m) {  
 if (arr1[i] < arr2[j]) {  
 result[k++] = arr1[i++];  
 } else {  
 result[k++] = arr2[j++];  
 }  
 }  
  
 while (i < n) {  
 result[k++] = arr1[i++];  
 }  
  
 while (j < m) {  
 result[k++] = arr2[j++];  
 }  
  
 return result;  
 }  
  
 // Function to create a BST from a sorted array  
 public Node5 sortedArrayToBST(int[] arr, int start, int end) {  
 if (start > end) {  
 return null;  
 }  
  
 int mid = (start + end) / 2;  
 Node5 newNode = new Node5(arr[mid]);  
  
 newNode.left = sortedArrayToBST(arr, start, mid - 1);  
 newNode.right = sortedArrayToBST(arr, mid + 1, end);  
  
 return newNode;  
 }  
  
 // Function to merge two BSTs  
 public void mergeBSTs(Task5 bst1, Task5 bst2) {  
 int[] arr1 = bst1.inOrderTraversal(bst1.root);  
 int[] arr2 = bst2.inOrderTraversal(bst2.root);  
  
 int[] mergedArray = mergeSortedArrays(arr1, arr2);  
  
 root = sortedArrayToBST(mergedArray, 0, mergedArray.length - 1);  
 }  
  
 // In-order traversal to get sorted elements from a BST  
 public int[] inOrderTraversal(Node5 node) {  
 int[] result = new int[0];  
 if (node != null) {  
 int[] left = inOrderTraversal(node.left);  
 int[] right = inOrderTraversal(node.right);  
  
 // Merge the left, current, and right arrays  
 result = new int[left.length + 1 + right.length];  
 System.*arraycopy*(left, 0, result, 0, left.length);  
 result[left.length] = node.data;  
 System.*arraycopy*(right, 0, result, left.length + 1, right.length);  
 }  
 return result;  
 }  
  
 // Function to print the BST in-order  
 public void printInOrder(Node5 node) {  
 if (node == null) {  
 return;  
 }  
 printInOrder(node.left);  
 System.*out*.print(node.data + " ");  
 printInOrder(node.right);  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
  
 Task5 bst1 = new Task5();  
 bst1.root = new Node5(5);  
 bst1.root.left = new Node5(3);  
 bst1.root.right = new Node5(6);  
 bst1.root.left.left = new Node5(2);  
 bst1.root.left.right = new Node5(4);  
  
 Task5 bst2 = new Task5();  
 bst2.root = new Node5(2);  
 bst2.root.left = new Node5(1);  
 bst2.root.right = new Node5(3);  
 bst2.root.right.right = new Node5(7);  
 bst2.root.right.right.left = new Node5(6);  
  
 Task5 mergedBST = new Task5();  
 mergedBST.mergeBSTs(bst1, bst2);  
  
 System.*out*.println("Merged BST (In-order traversal):");  
 mergedBST.printInOrder(mergedBST.root);  
 }  
}

