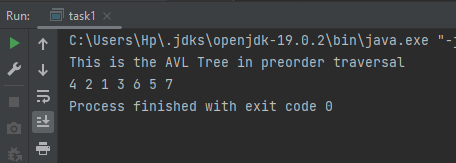
**22k-5195 Lab 9**

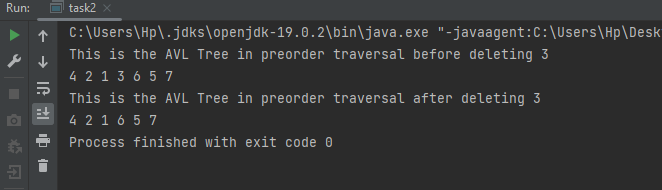
Task1:

class Node {  
 int data, height;  
 Node left, right;  
  
 Node(int d) {  
 data = d;  
 height = 1;  
 }  
}  
public class task1 {  
 Node root;  
  
  
 int height(Node N) {  
 if (N == null)  
 return 0;  
  
 return N.height;  
 }  
  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
  
 Node rightRotate(Node y) {  
 Node x = y.left;  
 Node T = x.right;  
  
  
 x.right = y;  
 y.left = T;  
  
  
 y.height = max(height(y.left), height(y.right)) + 1;  
 x.height = max(height(x.left), height(x.right)) + 1;  
  
  
 return x;  
 }  
  
 Node leftRotate(Node x) {  
 Node y = x.right;  
 Node T = y.left;  
  
  
 y.left = x;  
 x.right = T;  
  
  
 x.height = max(height(x.left), height(x.right)) + 1;  
 y.height = max(height(y.left), height(y.right)) + 1;  
  
  
 return y;  
 }  
  
  
 int getBalance(Node N) {  
 if (N == null)  
 return 0;  
  
 return height(N.left) - height(N.right);  
 }  
  
 Node insert(Node node, int data) {  
  
 if (node == null)  
 return (new Node(data));  
  
 if (data < node.data)  
 node.left = insert(node.left, data);  
 else if (data > node.data)  
 node.right = insert(node.right, data);  
 else  
 return node;  
  
  
 node.height = 1 + max(height(node.left),  
 height(node.right));  
  
 int balance = getBalance(node);  
  
 if (balance > 1 && data < node.left.data)  
 return rightRotate(node);  
  
 if (balance < -1 && data > node.right.data)  
 return leftRotate(node);  
  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
 void preOrder(Node node) {  
 if (node != null) {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 public static void main(String[] args) {  
 task1 tree = new task1();  
  
  
 tree.root = tree.insert(tree.root, 1);  
 tree.root = tree.insert(tree.root, 2);  
 tree.root = tree.insert(tree.root, 3);  
 tree.root = tree.insert(tree.root, 4);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 6);  
 tree.root = tree.insert(tree.root, 7);  
  
 System.*out*.println("This is the AVL Tree in preorder traversal");  
 tree.preOrder(tree.root);  
 }  
 }



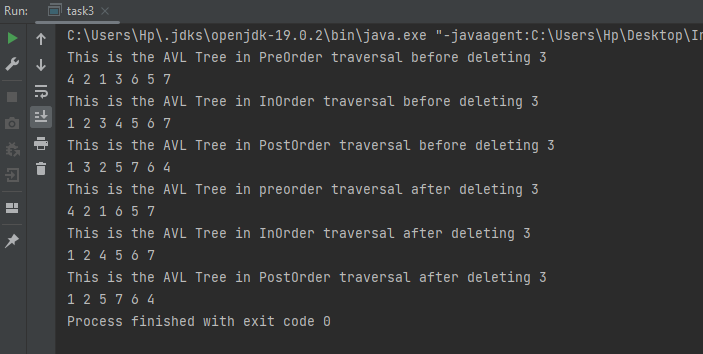
Task 2:

class Node  
{  
 int data, height;  
 Node left, right;  
  
 Node(int d)  
 {  
 data = d;  
 height = 1;  
 }  
}  
  
public class task2  
{  
 Node root;  
  
 int height(Node N)  
 {  
 if (N == null)  
 return 0;  
 return N.height;  
 }  
  
 int max(int a, int b)  
 {  
 return (a > b) ? a : b;  
 }  
  
 Node rightRotate(Node y)  
 {  
 Node x = y.left;  
 Node T2 = x.right;  
  
  
 x.right = y;  
 y.left = T2;  
  
  
 y.height = max(height(y.left), height(y.right)) + 1;  
 x.height = max(height(x.left), height(x.right)) + 1;  
  
  
 return x;  
 }  
  
  
 Node leftRotate(Node x)  
 {  
 Node y = x.right;  
 Node T2 = y.left;  
  
  
 y.left = x;  
 x.right = T2;  
  
  
 x.height = max(height(x.left), height(x.right)) + 1;  
 y.height = max(height(y.left), height(y.right)) + 1;  
  
  
 return y;  
 }  
  
  
 int getBalance(Node N)  
 {  
 if (N == null)  
 return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node insert(Node node, int data)  
 {  
  
 if (node == null)  
 return (new Node(data));  
  
 if (data < node.data)  
 node.left = insert(node.left, data);  
 else if (data > node.data)  
 node.right = insert(node.right, data);  
 else  
 return node;  
  
  
 node.height = 1 + max(height(node.left),  
 height(node.right));  
  
  
 int balance = getBalance(node);  
  
  
 if (balance > 1 && data < node.left.data)  
 return rightRotate(node);  
  
  
 if (balance < -1 && data > node.right.data)  
 return leftRotate(node);  
  
 if (balance > 1 && data > node.left.data)  
 {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
  
  
 if (balance < -1 && data < node.right.data)  
 {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
  
 Node minValueNode(Node min)  
 {  
 Node current = min;  
  
  
 while (current.left != null)  
 current = current.left;  
  
 return current;  
 }  
  
 Node deleteNode(Node root, int data)  
 {  
  
 if (root == null)  
 return root;  
  
  
 if (data < root.data)  
 root.left = deleteNode(root.left, data);  
  
  
 else if (data > root.data)  
 root.right = deleteNode(root.right, data);  
  
  
 else  
 {  
  
 if ((root.left == null) || (root.right == null))  
 {  
 Node temp = null;  
 if (temp == root.left)  
 temp = root.right;  
 else  
 temp = root.left;  
  
  
 if (temp == null)  
 {  
 temp = root;  
 root = null;  
 }  
 else  
 root = temp;  
  
 }  
 else  
 {  
  
 Node temp = minValueNode(root.right);  
  
  
 root.data = temp.data;  
  
 root.right = deleteNode(root.right, temp.data);  
 }  
 }  
  
  
 if (root == null)  
 return root;  
  
  
 root.height = max(height(root.left), height(root.right)) + 1;  
  
  
 int balance = getBalance(root);  
  
  
 if (balance > 1 && getBalance(root.left) >= 0)  
 return rightRotate(root);  
  
 if (balance > 1 && getBalance(root.left) < 0)  
 {  
 root.left = leftRotate(root.left);  
 return rightRotate(root);  
 }  
  
  
 if (balance < -1 && getBalance(root.right) <= 0)  
 return leftRotate(root);  
  
  
 if (balance < -1 && getBalance(root.right) > 0)  
 {  
 root.right = rightRotate(root.right);  
 return leftRotate(root);  
 }  
  
 return root;  
 }  
  
 void preOrder(Node node)  
 {  
 if (node != null)  
 {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 public static void main(String[] args)  
 {  
 task2 tree = new task2();  
  
 tree.root = tree.insert(tree.root, 1);  
 tree.root = tree.insert(tree.root, 2);  
 tree.root = tree.insert(tree.root, 3);  
 tree.root = tree.insert(tree.root, 4);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 6);  
 tree.root = tree.insert(tree.root, 7);  
  
 System.*out*.println("This is the AVL Tree in preorder traversal before deleting 3");  
 tree.preOrder(tree.root);  
  
 tree.root = tree.deleteNode(tree.root, 3);  
  
 System.*out*.println("");  
 System.*out*.println("This is the AVL Tree in preorder traversal after deleting 3");  
 tree.preOrder(tree.root);  
 }  
}



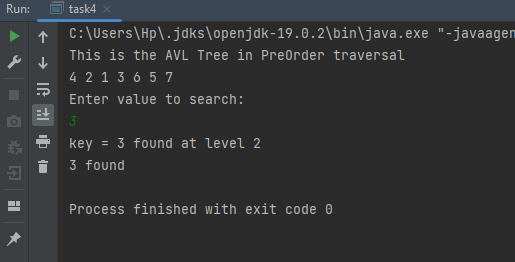
Task3:

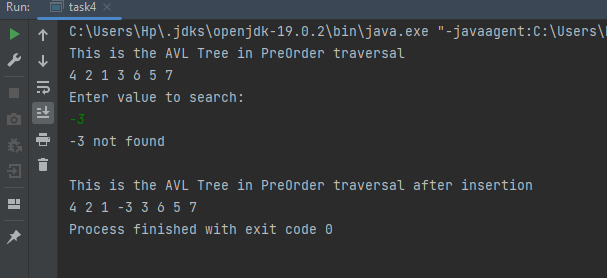
class Node  
{  
 int data, height;  
 Node left, right;  
  
 Node(int d)  
 {  
 data = d;  
 height = 1;  
 }  
}  
public class task3 {  
  
  
 Node root;  
  
 int height(Node N)  
 {  
 if (N == null)  
 return 0;  
 return N.height;  
 }  
  
 int max(int a, int b)  
 {  
 return (a > b) ? a : b;  
 }  
  
 Node rightRotate(Node y)  
 {  
 Node x = y.left;  
 Node T2 = x.right;  
  
  
 x.right = y;  
 y.left = T2;  
  
  
 y.height = max(height(y.left), height(y.right)) + 1;  
 x.height = max(height(x.left), height(x.right)) + 1;  
  
  
 return x;  
 }  
  
  
 Node leftRotate(Node x)  
 {  
 Node y = x.right;  
 Node T2 = y.left;  
  
  
 y.left = x;  
 x.right = T2;  
  
  
 x.height = max(height(x.left), height(x.right)) + 1;  
 y.height = max(height(y.left), height(y.right)) + 1;  
  
  
 return y;  
 }  
  
  
 int getBalance(Node N)  
 {  
 if (N == null)  
 return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node insert(Node node, int data)  
 {  
  
 if (node == null)  
 return (new Node(data));  
  
 if (data < node.data)  
 node.left = insert(node.left, data);  
 else if (data > node.data)  
 node.right = insert(node.right, data);  
 else  
 return node;  
  
  
 node.height = 1 + max(height(node.left),  
 height(node.right));  
  
  
 int balance = getBalance(node);  
  
  
 if (balance > 1 && data < node.left.data)  
 return rightRotate(node);  
  
  
 if (balance < -1 && data > node.right.data)  
 return leftRotate(node);  
  
 if (balance > 1 && data > node.left.data)  
 {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
  
  
 if (balance < -1 && data < node.right.data)  
 {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
  
 Node minValueNode(Node min)  
 {  
 Node current = min;  
  
  
 while (current.left != null)  
 current = current.left;  
  
 return current;  
 }  
  
 Node deleteNode(Node root, int data)  
 {  
  
 if (root == null)  
 return root;  
  
  
 if (data < root.data)  
 root.left = deleteNode(root.left, data);  
  
  
 else if (data > root.data)  
 root.right = deleteNode(root.right, data);  
  
  
 else  
 {  
  
 if ((root.left == null) || (root.right == null))  
 {  
 Node temp = null;  
 if (temp == root.left)  
 temp = root.right;  
 else  
 temp = root.left;  
  
  
 if (temp == null)  
 {  
 temp = root;  
 root = null;  
 }  
 else  
 root = temp;  
  
 }  
 else  
 {  
  
 Node temp = minValueNode(root.right);  
  
  
 root.data = temp.data;  
  
 root.right = deleteNode(root.right, temp.data);  
 }  
 }  
  
  
 if (root == null)  
 return root;  
  
  
 root.height = max(height(root.left), height(root.right)) + 1;  
  
  
 int balance = getBalance(root);  
  
  
 if (balance > 1 && getBalance(root.left) >= 0)  
 return rightRotate(root);  
  
 if (balance > 1 && getBalance(root.left) < 0)  
 {  
 root.left = leftRotate(root.left);  
 return rightRotate(root);  
 }  
  
  
 if (balance < -1 && getBalance(root.right) <= 0)  
 return leftRotate(root);  
  
  
 if (balance < -1 && getBalance(root.right) > 0)  
 {  
 root.right = rightRotate(root.right);  
 return leftRotate(root);  
 }  
  
 return root;  
 }  
  
 void Inorder(Node node)  
 {  
 if (node == null)  
 return;  
  
 Inorder(node.left);  
 System.*out*.print(node.data + " ");  
 Inorder(node.right);  
 }  
 void preOrder(Node node)  
 {  
 if (node != null)  
 {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
 void PostOrder(Node node)  
 {  
 if (node == null)  
 return;  
  
 PostOrder(node.left);  
 PostOrder(node.right);  
 System.*out*.print(node.data + " ");  
  
 }  
  
  
 public static void main(String[] args)  
 {  
 task3 tree = new task3();  
  
 tree.root = tree.insert(tree.root, 1);  
 tree.root = tree.insert(tree.root, 2);  
 tree.root = tree.insert(tree.root, 3);  
 tree.root = tree.insert(tree.root, 4);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 6);  
 tree.root = tree.insert(tree.root, 7);  
  
 System.*out*.println("This is the AVL Tree in PreOrder traversal before deleting 3");  
 tree.preOrder(tree.root);  
 System.*out*.println();  
 System.*out*.println("This is the AVL Tree in InOrder traversal before deleting 3");  
 tree.Inorder(tree.root);  
 System.*out*.println();  
 System.*out*.println("This is the AVL Tree in PostOrder traversal before deleting 3");  
 tree.PostOrder(tree.root);  
  
 tree.root = tree.deleteNode(tree.root, 3);  
  
 System.*out*.println();  
 System.*out*.println("This is the AVL Tree in preorder traversal after deleting 3");  
 tree.preOrder(tree.root);  
 System.*out*.println();  
 System.*out*.println("This is the AVL Tree in InOrder traversal after deleting 3");  
 tree.Inorder(tree.root);  
 System.*out*.println();  
 System.*out*.println("This is the AVL Tree in PostOrder traversal after deleting 3");  
 tree.PostOrder(tree.root);  
 }  
 }



Task 4

import java.util.Scanner;  
class Node  
{  
 int data, height;  
 Node left, right;  
  
 Node(int d)  
 {  
 data = d;  
 height = 1;  
 }  
}  
public class task4 {  
  
 Node root;  
  
 int height(Node N) {  
 if (N == null)  
 return 0;  
 return N.height;  
 }  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
 Node rightRotate(Node y) {  
 Node x = y.left;  
 Node T2 = x.right;  
  
  
 x.right = y;  
 y.left = T2;  
  
  
 y.height = max(height(y.left), height(y.right)) + 1;  
 x.height = max(height(x.left), height(x.right)) + 1;  
  
  
 return x;  
 }  
  
  
 Node leftRotate(Node x) {  
 Node y = x.right;  
 Node T2 = y.left;  
  
  
 y.left = x;  
 x.right = T2;  
  
  
 x.height = max(height(x.left), height(x.right)) + 1;  
 y.height = max(height(y.left), height(y.right)) + 1;  
  
  
 return y;  
 }  
  
  
 int getBalance(Node N) {  
 if (N == null)  
 return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node insert(Node node, int data) {  
  
 if (node == null)  
 return (new Node(data));  
  
 if (data < node.data)  
 node.left = insert(node.left, data);  
 else if (data > node.data)  
 node.right = insert(node.right, data);  
 else  
 return node;  
  
  
 node.height = 1 + max(height(node.left),  
 height(node.right));  
  
  
 int balance = getBalance(node);  
  
  
 if (balance > 1 && data < node.left.data)  
 return rightRotate(node);  
  
  
 if (balance < -1 && data > node.right.data)  
 return leftRotate(node);  
  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
  
  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
  
 Node search(Node root, int key, int count) {  
 if (root == null) {  
 return null;  
 }  
 if (root.data == key) {  
 System.*out*.println("key = " + key + " found at level " + count);  
 return root;  
 }  
 if (root.data < key) {  
 return search(root.right, key, count + 1);  
 } else {  
 return search(root.left, key, count + 1);  
 }  
 }  
  
  
 void preOrder(Node node) {  
 if (node != null) {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
  
 public static void main(String[] args) {  
 Scanner scan = new Scanner(System.*in*);  
  
 task4 tree = new task4();  
  
 tree.root = tree.insert(tree.root, 1);  
 tree.root = tree.insert(tree.root, 2);  
 tree.root = tree.insert(tree.root, 3);  
 tree.root = tree.insert(tree.root, 4);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 6);  
 tree.root = tree.insert(tree.root, 7);  
  
 System.*out*.println("This is the AVL Tree in PreOrder traversal");  
 tree.preOrder(tree.root);  
 System.*out*.println();  
 System.*out*.println("Enter value to search:");  
 int x = scan.nextInt();  
 if (tree.search(tree.root, x,0) == null) {  
 System.*out*.println(x + " not found");  
 tree.root = tree.insert(tree.root, x);  
 System.*out*.println();  
 System.*out*.println("This is the AVL Tree in PreOrder traversal after insertion");  
 tree.preOrder(tree.root);  
 }  
 else {  
 System.*out*.println(x + " found");  
 }  
  
  
 }  
}





Task 5:

import java.util.Scanner;  
class Node {  
 int data, height;  
 Node left, right;  
  
 Node(int d) {  
 data = d;  
 height = 1;  
 }  
}  
public class task5 {  
  
 Node root;  
  
 int height(Node N) {  
 if (N == null)  
 return 0;  
 return N.height;  
 }  
  
 int getBalance(Node n) {  
 if (n == null)  
 return 0;  
 return height(n.left) - height(n.right);  
 }  
  
 Node rightRotate(Node y) {  
 Node x = y.left;  
 Node T2 = x.right;  
  
 x.right = y;  
 y.left = T2;  
  
 y.height = Math.*max*(height(y.left), height(y.right)) + 1;  
 x.height = Math.*max*(height(x.left), height(x.right)) + 1;  
  
 return x;  
 }  
  
 Node leftRotate(Node x) {  
 Node y = x.right;  
 Node T2 = y.left;  
  
 y.left = x;  
 x.right = T2;  
  
 x.height = Math.*max*(height(x.left), height(x.right)) + 1;  
 y.height = Math.*max*(height(y.left), height(y.right)) + 1;  
  
 return y;  
 }  
  
 Node insert(Node node, int data) {  
 if (node == null)  
 return (new Node(data));  
  
 if (data < node.data)  
 node.left = insert(node.left, data);  
 else if (data > node.data)  
 node.right = insert(node.right, data);  
 else  
 return node;  
  
 node.height = 1 + Math.*max*(height(node.left), height(node.right));  
  
 int balance = getBalance(node);  
  
 if (balance > 1 && data < node.left.data)  
 return rightRotate(node);  
  
 if (balance < -1 && data > node.right.data)  
 return leftRotate(node);  
  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
 void getHeights(Node node) {  
 if (node == null)  
 return;  
  
 int leftHeight = height(node.left);  
 int rightHeight = height(node.right);  
  
 System.*out*.println("Left side height: " + leftHeight);  
 System.*out*.println("Right side height: " + rightHeight);  
 }  
 void kthSmallestAndLargest(int k) {  
 if (root == null) {  
 System.*out*.println("Tree is empty");  
 return;  
 }  
  
 int[] count = {0};  
  
 kthSmallest(root, k, count);  
  
 count[0] = 0;  
 kthLargest(root, k, count);  
  
 getHeights(root);  
 }  
 void kthSmallest(Node node, int k, int[] count) {  
 if (node == null || count[0] >= k)  
 return;  
  
 kthSmallest(node.left, k, count);  
  
 count[0]++;  
  
 if (count[0] == k) {  
 System.*out*.println(k + "th smallest element is " + node.data);  
 return;  
 }  
  
 kthSmallest(node.right, k, count);  
 }  
  
  
 void kthLargest(Node node, int k, int[] count) {  
 if (node == null || count[0] >= k)  
 return;  
  
 kthLargest(node.right, k, count);  
  
 count[0]++;  
  
 if (count[0] == k) {  
 System.*out*.println(k + "th largest element is " + node.data);  
 return;  
 }  
  
 kthLargest(node.left, k, count);  
 }  
  
   
  
   
 void preOrder(Node node) {  
 if (node != null) {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner scan = new Scanner(System.*in*);  
 task5 tree = new task5();  
  
 tree.root = tree.insert(tree.root, 1);  
 tree.root = tree.insert(tree.root, 2);  
 tree.root = tree.insert(tree.root, 3);  
 tree.root = tree.insert(tree.root, 4);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 6);  
 tree.root = tree.insert(tree.root, 7);  
 System.*out*.println("This is the AVL Tree in PreOrder traversal");  
 tree.preOrder(tree.root);  
 System.*out*.println();  
  
 System.*out*.println("Enter the value of k (which kth smallest and largest element):");  
 int k = scan.nextInt();  
  
 tree.kthSmallestAndLargest(k);  
 }  
 }

