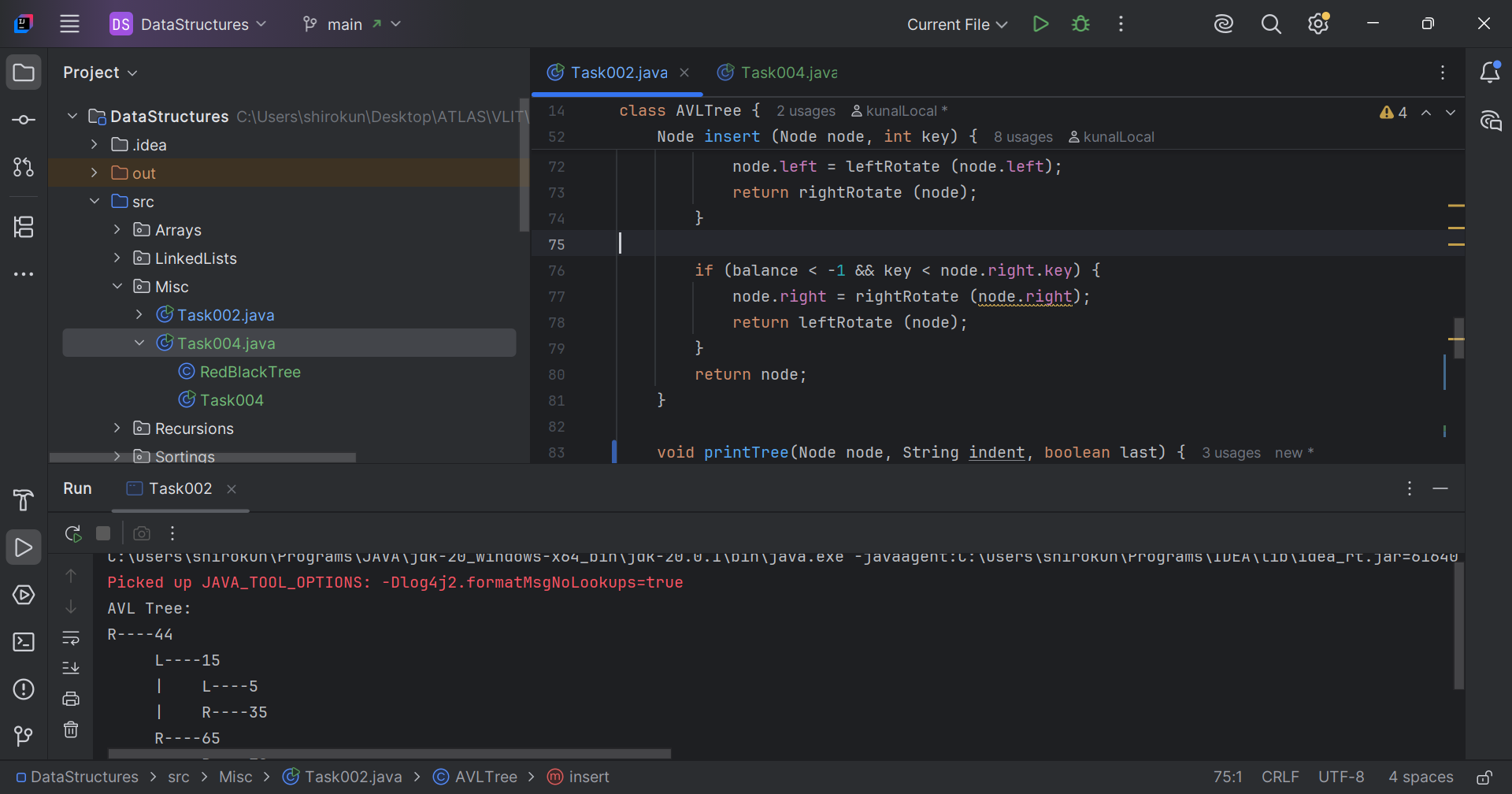
Day 17 – 12/07/2025

Q1. Write algorithm for AVL tree.   
Ans. Algorithm:

1. Insert a node into the AVL tree using a standard binary search tree insertion algorithm.
2. Calculate the balance factor of the node's ancestors.
3. If the balance factor becomes greater than 1 or less than -1, perform rotations to balance the tree.

// Task002: AVL tree insertion.  
  
package Misc;  
  
class Node {  
 int key, height;  
 Node left, right;  
 Node (int d) {  
 key = d;  
 height = 1;  
 }  
}  
  
class AVLTree {  
 Node root;  
 int height (Node N) {  
 if (N == null)  
 return 0;  
 return N.height;  
 }  
  
 int max (int a, int b) {  
 return Math.*max*(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 key) {  
 if (node == null)  
 return (new Node (key));  
  
 if (key < node.key)  
 node.left = insert (node.left, key);  
 else if (key > node.key)  
 node.right = insert (node.right, key);  
 else  
 return node;  
  
 node.height = 1 + max (height (node.left), height (node.right));  
 int balance = getBalance (node);  
 if (balance > 1 && key < node.left.key)  
 return rightRotate (node);  
  
 if (balance < -1 && key > node.right.key)  
 return leftRotate (node);  
  
 if (balance > 1 && key > node.left.key) {  
 node.left = leftRotate (node.left);  
 return rightRotate (node);  
 }  
  
 if (balance < -1 && key < node.right.key) {  
 node.right = rightRotate (node.right);  
 return leftRotate (node);  
 }  
 return node;  
 }  
  
 void printTree(Node node, String indent, boolean last) {  
 if (node != null) {  
 System.*out*.print(indent);  
 if (last) {  
 System.*out*.print("R----");  
 indent += " ";  
 } else {  
 System.*out*.print("L----");  
 indent += "| ";  
 }  
 System.*out*.println(node.key);  
 printTree(node.left, indent, false);  
 printTree(node.right, indent, true);  
 }  
 }  
}

public class Task002 {  
 public static void main(String[] args) {  
 AVLTree tree = new AVLTree();  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 15);  
 tree.root = tree.insert(tree.root, 44);  
 tree.root = tree.insert(tree.root, 35);  
 tree.root = tree.insert(tree.root, 65);  
 tree.root = tree.insert(tree.root, 78);  
  
 System.*out*.println("AVL Tree: ");  
 tree.printTree(tree.root, "", true);  
 }  
}



Q3. Write algorithm for Red-Black tree.  
Ans. Algorithm –

1. Insert a new node with a given key into the tree.
2. If the tree is empty, make the new node the root node and color it black.
3. Otherwise, insert the new node as a leaf node and color it red.
4. If the parent of the new node is black, the tree is still valid.
5. If the parent of the new node is red, we need to fix the tree by performing rotations and recoloring.

// Task004: Red-Black tree.  
  
package Misc;  
  
class RedBlackTree {  
 Node root;  
 private static class Node {  
 int data;  
 Node left;  
 Node right;  
 Node parent;  
 boolean isRed;  
 Node(int data) {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 this.parent = null;  
 this.isRed = true;  
 }  
 }  
   
 private void leftRotate(Node x) {  
 Node y = x.right;  
 x.right = y.left;  
 if (y.left != null) {  
 y.left.parent = x;  
 }  
 y.parent = x.parent;  
 if (x.parent == null) {  
 root = y;  
 } else if (x == x.parent.left) {  
 x.parent.left = y;  
 } else {  
 x.parent.right = y;  
 }  
 y.left = x;  
 x.parent = y;  
 }  
   
 private void rightRotate(Node y) {  
 Node x = y.left;  
 y.left = x.right;  
 if (x.right != null) {  
 x.right.parent = y;  
 }  
 x.parent = y.parent;  
 if (y.parent == null) {  
 root = x;  
 } else if (y == y.parent.right) {  
 y.parent.right = x;  
 } else {  
 y.parent.left = x;  
 }  
 x.right = y;  
 y.parent = x;  
 }  
  
  
 private void fixInsertion(Node k) {  
   
// No two adjacent nodes should be red  
 while (k.parent != null && k.parent.isRed) {  
// Left branch  
 if (k.parent == k.parent.parent.left) {  
 Node y = k.parent.parent.right;  
   
// Red Aunt Color Exchange  
 if (y != null && y.isRed) {  
 k.parent.isRed = false;  
 y.isRed = false;  
 k.parent.parent.isRed = true;  
 k = k.parent.parent;  
 }  
   
// Black Aunt Rotate  
 else {  
 if (k == k.parent.right) {  
 k = k.parent;  
 leftRotate(k);  
 }  
 k.parent.isRed = false;  
 k.parent.parent.isRed = true;  
 rightRotate(k.parent.parent);  
 }  
 }  
   
// Right branch  
 else {  
 Node y = k.parent.parent.left;  
   
// Red Aunt Color Exchange  
 if (y != null && y.isRed) {  
 k.parent.isRed = false;  
 y.isRed = false;  
 k.parent.parent.isRed = true;  
 k = k.parent.parent;  
 }  
   
// Black Aunt Rotate  
 else {  
 if (k == k.parent.left) {  
 k = k.parent;  
 rightRotate(k);  
 }  
 k.parent.isRed = false;  
 k.parent.parent.isRed = true;  
 leftRotate(k.parent.parent);  
 }  
 }  
 }  
 root.isRed = false;  
 }

public void insert(int data) {  
 Node node = new Node(data);  
 if (root == null) {  
 root = node;  
 }

else {  
 Node y = null;  
 Node x = root;  
 while (x != null) {  
 y = x;  
 if (node.data < x.data) {  
 x = x.left;  
 } else {  
 x = x.right;  
 }  
 }

node.parent = y;  
 if (node.data < y.data) {  
 y.left = node;  
 } else {  
 y.right = node;  
 }  
 }

// root is always black  
 if (node.parent == null) {  
 node.isRed = false;  
 return;  
 }  
 if (node.parent.parent == null) {  
 return;  
 }  
 fixInsertion(node);  
 }  
  
 public void printTree(Node node, String indent, boolean last) {  
 if (node != null) {  
 System.*out*.print(indent);  
 if (last) {  
 System.*out*.print("R----");  
 indent += " ";  
 } else {  
 System.*out*.print("L----");  
 indent += "| ";  
 }

String color = node.isRed ? "RED" : "BLACK";  
 System.*out*.println(node.data + "(" + color + ")");  
 printTree(node.left, indent, false);  
 printTree(node.right, indent, true);  
 }  
 }  
}

public class Task004 {  
 public static void main(String[] args) {  
 RedBlackTree tree = new RedBlackTree();  
 for (int i = 1; i < 8; i++) {  
 tree.insert(i);  
 if (i==4) break;  
 }  
 tree.insert(9);  
 tree.insert(6);  
 tree.insert(10);  
 tree.insert(5);  
 System.*out*.println("Red-black tree: ");  
 tree.printTree(tree.root, "", true);  
 }  
}

