CS162 DS Lab Assignment 11

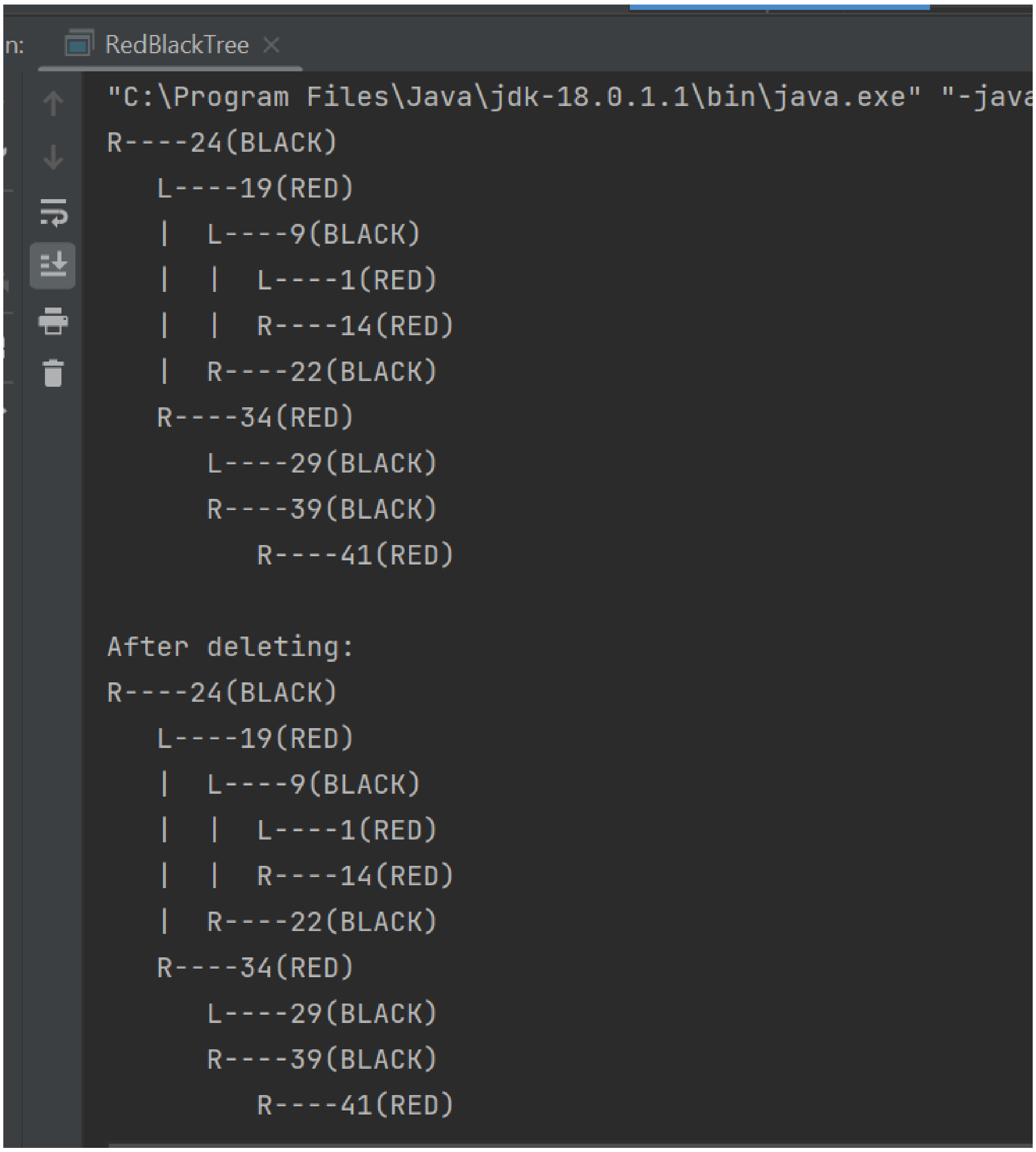
Name - Karan Patel

ID - 202152315

Q1)

```java
// Implementing Red-Black Tree in Java

class TreeNode {
    int data;
    TreeNode prt;
    TreeNode left;
    TreeNode right;
    int color;
}

class RedBlackTree {
    private TreeNode root;
    private TreeNode TNULL;

    // Preorder
    private void preOrderHelper(TreeNode node) {
        if (node != TNULL) {
            System.out.print(node.data + " ");
            preOrderHelper(node.left);
            preOrderHelper(node.right);
        }
    }

    // Inorder
    private void inOrderHelper(TreeNode node) {
        if (node != TNULL) {
            inOrderHelper(node.left);
            System.out.print(node.data + " ");
            inOrderHelper(node.right);
        }
    }

    // Post order
    private void postOrderHelper(TreeNode node) {
        if (node != TNULL) {
            postOrderHelper(node.left);
            postOrderHelper(node.right);
            System.out.print(node.data + " ");
        }
    }

    // Search the tree
    private TreeNode searchTreeHelper(TreeNode node, int key) {
        if (node == TNULL || key == node.data) {
            return node;
        }

        if (key < node.data) {
            return searchTreeHelper(node.left, key);
        }
        return searchTreeHelper(node.right, key);
    }

    // Balance the tree after deletion of a node
    private void fixDelete(TreeNode x) {
        TreeNode s;
        while (x != root && x.color == 0) {
            if (x == x.prt.left) {
                s = x.prt.right;
                if (s.color == 1) {
                    s.color = 0;
                    x.prt.color = 1;
                    leftRotate(x.prt);
                    s = x.prt.right;
                }

                if (s.left.color == 0 && s.right.color == 0) {
                    s.color = 1;
                    x = x.prt;
                } else {
                    if (s.right.color == 0) {
                        s.left.color = 0;
                        s.color = 1;
                        rightRotate(s);
                        s = x.prt.right;
                    }

                    s.color = x.prt.color;
                    x.prt.color = 0;
                    s.right.color = 0;
                    leftRotate(x.prt);
                    x = root;
                }
            } else {
                s = x.prt.left;
                if (s.color == 1) {
                    s.color = 0;
                    x.prt.color = 1;
                    rightRotate(x.prt);
                    s = x.prt.left;
                }

                if (s.right.color == 0 && s.right.color == 0) {
                    s.color = 1;
                    x = x.prt;
                } else {
                    if (s.left.color == 0) {
                        s.right.color = 0;
                        s.color = 1;
                        leftRotate(s);
                        s = x.prt.left;
                    }
                    s.color = x.prt.color;
                    x.prt.color = 0;
                    s.left.color = 0;
                    rightRotate(x.prt);
                    x = root;
                }
            }
        }
        x.color = 0;
    }

    private void rbTransplant(TreeNode u, TreeNode v) {
        if (u.prt == null) {
            root = v;
        } else if (u == u.prt.left) {
            u.prt.left = v;
        } else {
            u.prt.right = v;
        }
        v.prt = u.prt;
    }

    private void deleteNodeHelper(TreeNode node, int key) {
        TreeNode z = TNULL;
        TreeNode x, y;
        while (node != TNULL) {
            if (node.data == key) {
                z = node;
            }

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

        if (z == TNULL) {
            System.out.println("Couldn't find key in the tree");
            return;
        }

        y = z;
        int yOriginalColor = y.color;
        if (z.left == TNULL) {
            x = z.right;
            rbTransplant(z, z.right);
        } else if (z.right == TNULL) {
            x = z.left;
            rbTransplant(z, z.left);
        } else {
            y = minimum(z.right);
            yOriginalColor = y.color;
            x = y.right;
            if (y.prt == z) {
                x.prt = y;
            } else {
                rbTransplant(y, y.right);
                y.right = z.right;
                y.right.prt = y;
            }

            rbTransplant(z, y);
            y.left = z.left;
            y.left.prt = y;
            y.color = z.color;
        }
        if (yOriginalColor == 0) {
            fixDelete(x);
        }
    }

    // Balance the node after insertion
    private void fixInsert(TreeNode k) {
        TreeNode u;
        while (k.prt.color == 1) {
            if (k.prt == k.prt.prt.right) {
                u = k.prt.prt.left;
                if (u.color == 1) {
                    u.color = 0;
                    k.prt.color = 0;
                    k.prt.prt.color = 1;
                    k = k.prt.prt;
                } else {
                    if (k == k.prt.left) {
                        k = k.prt;
                        rightRotate(k);
                    }
                    k.prt.color = 0;
                    k.prt.prt.color = 1;
                    leftRotate(k.prt.prt);
                }
            } else {
                u = k.prt.prt.right;

                if (u.color == 1) {
                    u.color = 0;
                    k.prt.color = 0;
                    k.prt.prt.color = 1;
                    k = k.prt.prt;
                } else {
                    if (k == k.prt.right) {
                        k = k.prt;
                        leftRotate(k);
                    }
                    k.prt.color = 0;
                    k.prt.prt.color = 1;
                    rightRotate(k.prt.prt);
                }
            }
            if (k == root) {
                break;
            }
        }
        root.color = 0;
    }

    private void printHelper(TreeNode root, String indent, boolean last) {
        if (root != TNULL) {
            System.out.print(indent);
            if (last) {
                System.out.print("R----");
                indent += "    ";
            } else {
                System.out.print("L----");
                indent += "|   ";
            }

            String sColor = root.color == 1 ? "RED" : "BLACK";
            System.out.println(root.data + "(" + sColor + ")");
            printHelper(root.left, indent, false);
            printHelper(root.right, indent, true);
        }
    }

    public RedBlackTree() {
        TNULL = new TreeNode();
        TNULL.color = 0;
        TNULL.left = null;
        TNULL.right = null;
        root = TNULL;
    }

    public void preorder() {
        preOrderHelper(this.root);
    }

    public void inorder() {
        inOrderHelper(this.root);
    }

    public void postorder() {
        postOrderHelper(this.root);
    }

    public TreeNode searchTree(int k) {
        return searchTreeHelper(this.root, k);
    }

    public TreeNode minimum(TreeNode node) {
        while (node.left != TNULL) {
            node = node.left;
        }
        return node;
    }

    public TreeNode maximum(TreeNode node) {
        while (node.right != TNULL) {
            node = node.right;
        }
        return node;
    }

    public TreeNode successor(TreeNode x) {
        if (x.right != TNULL) {
            return minimum(x.right);
        }

        TreeNode y = x.prt;
        while (y != TNULL && x == y.right) {
            x = y;
            y = y.prt;
        }
        return y;
    }

    public TreeNode predecessor(TreeNode x) {
        if (x.left != TNULL) {
            return maximum(x.left);
        }

        TreeNode y = x.prt;
        while (y != TNULL && x == y.left) {
            x = y;
            y = y.prt;
        }

        return y;
    }

    public void leftRotate(TreeNode x) {
        TreeNode y = x.right;
        x.right = y.left;
        if (y.left != TNULL) {
            y.left.prt = x;
        }
        y.prt = x.prt;
        if (x.prt == null) {
            this.root = y;
        } else if (x == x.prt.left) {
            x.prt.left = y;
        } else {
            x.prt.right = y;
        }
        y.left = x;
        x.prt = y;
    }

    public void rightRotate(TreeNode x) {
        TreeNode y = x.left;
        x.left = y.right;
        if (y.right != TNULL) {
            y.right.prt = x;
        }
        y.prt = x.prt;
        if (x.prt == null) {
            this.root = y;
        } else if (x == x.prt.right) {
            x.prt.right = y;
        } else {
            x.prt.left = y;
        }
        y.right = x;
        x.prt = y;
    }

    public void insert(int key) {
        TreeNode node = new TreeNode();
        node.prt = null;
        node.data = key;
        node.left = TNULL;
        node.right = TNULL;
        node.color = 1;

        TreeNode y = null;
        TreeNode x = this.root;

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

        node.prt = y;
        if (y == null) {
            root = node;
        } else if (node.data < y.data) {
            y.left = node;
        } else {
            y.right = node;
        }

        if (node.prt == null) {
            node.color = 0;
            return;
        }

        if (node.prt.prt == null) {
            return;
        }

        fixInsert(node);
    }

    public TreeNode getRoot() {
        return this.root;
    }

    public void deleteNode(int data) {
        deleteNodeHelper(this.root, data);
    }
    public void printTree() {
        printHelper(this.root, "", true);
    }

    public static void main(String[] args) {
        RedBlackTree bt = new RedBlackTree();
        bt.insert(29);
        bt.insert(19);
        bt.insert(9);
        bt.insert(14);
        bt.insert(24);
        bt.insert(22);
        bt.insert(39);
        bt.insert(34);
        bt.insert(41);
        bt.insert(1);
        bt.printTree();

        System.out.println("\nAfter deleting:");
//         bt.deleteNode(23);
//        bt.deleteNode(15);
//        bt.deleteNode(1);
        bt.printTree();
    }
}
```

Result



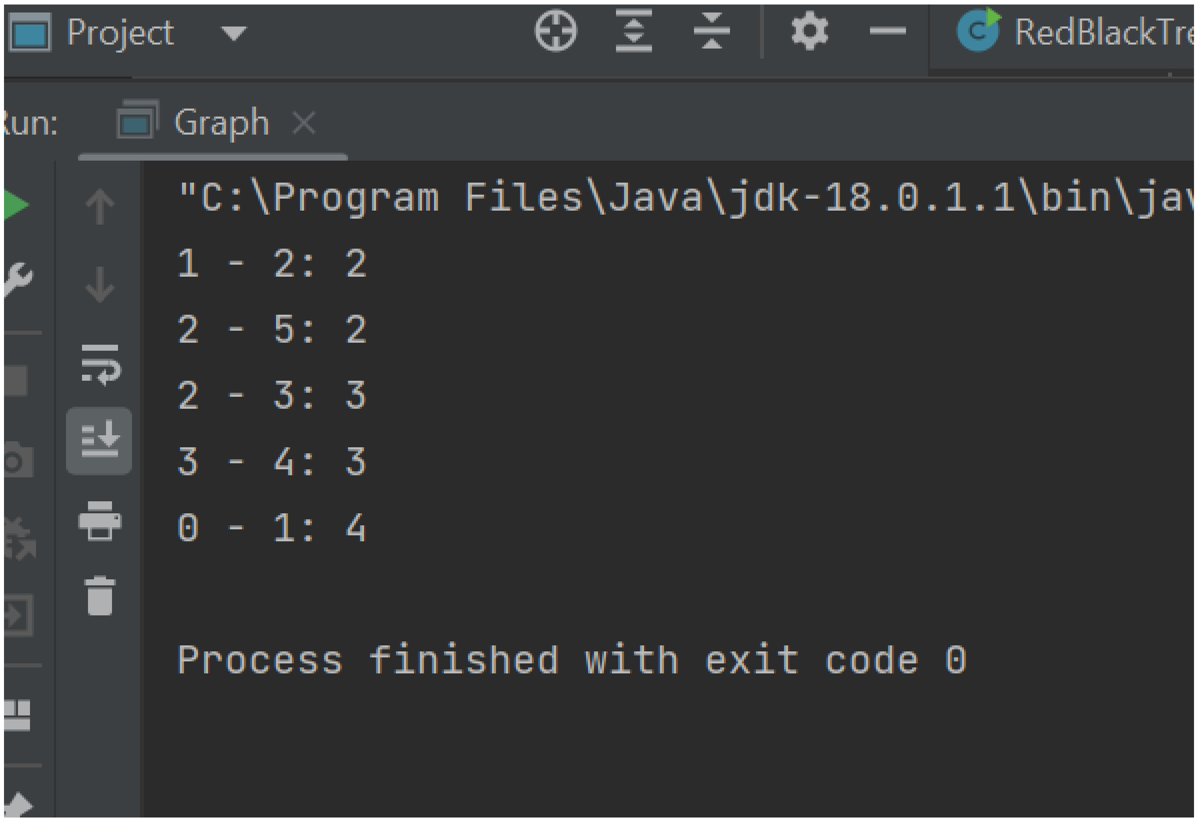
n: RedBlackTree

```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" "-java
R----24(BLACK)
   L----19(RED)
   |  L----9(BLACK)
   |  |  L----1(RED)
   |  |  R----14(RED)
   |  R----22(BLACK)
   R----34(RED)
      L----29(BLACK)
      R----39(BLACK)
         R----41(RED)

After deleting:
R----24(BLACK)
   L----19(RED)
   |  L----9(BLACK)
   |  |  L----1(RED)
   |  |  R----14(RED)
   |  R----22(BLACK)
   R----34(RED)
      L----29(BLACK)
      R----39(BLACK)
         R----41(RED)
```

Q2)

```java
import java.util.*;
class Graph {
    class Edge implements Comparable<Edge> {
        int src, dest, weight;

        public int compareTo(Edge compareEdge) {
            return this.weight - compareEdge.weight;
        }
    };

    // Union
    class subset {
        int parent, rank;
    };

    int vertices, edges;
    Edge edge[];

    // Graph creation
    Graph(int v, int e) {
        vertices = v;
        edges = e;
        edge = new Edge[edges];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }

    int find(subset subsets[], int i) {
        if (subsets[i].parent != i)
            subsets[i].parent = find(subsets, subsets[i].parent);
        return subsets[i].parent;
    }

    void Union(subset subsets[], int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        if (subsets[xroot].rank < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    // Applying Kruskal Algorithm
    void KruskalAlgo() {
        Edge result[] = new Edge[vertices];
        int e = 0;
        int i = 0;
        for (i = 0; i < vertices; ++i)
            result[i] = new Edge();
        // Sorting the edges
        Arrays.sort(edge);
        subset subsets[] = new subset[vertices];
        for (i = 0; i < vertices; ++i)
            subsets[i] = new subset();

        for (int v = 0; v < vertices; ++v) {
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }
        i = 0;
        while (e < vertices - 1) {
            Edge next_edge = new Edge();
            next_edge = edge[i++];
            int x = find(subsets, next_edge.src);
            int y = find(subsets, next_edge.dest);
            if (x != y) {
                result[e++] = next_edge;
                Union(subsets, x, y);
            }
        }
        for (i = 0; i < e; ++i)
            System.out.println(result[i].src + " - " + result[i].dest + ": "
+ result[i].weight);
    }

    public static void main(String[] args) {
        int vertices = 6; // Number of vertices
        int edges = 8; // Number of edges
        Graph G = new Graph(vertices, edges);

        G.edge[0].src = 0;
        G.edge[0].dest = 1;
        G.edge[0].weight = 4;

        G.edge[1].src = 0;
        G.edge[1].dest = 2;
        G.edge[1].weight = 4;

        G.edge[2].src = 1;
        G.edge[2].dest = 2;
        G.edge[2].weight = 2;

        G.edge[3].src = 2;
        G.edge[3].dest = 3;
        G.edge[3].weight = 3;

        G.edge[4].src = 2;
        G.edge[4].dest = 5;
        G.edge[4].weight = 2;

        G.edge[5].src = 2;
        G.edge[5].dest = 4;
        G.edge[5].weight = 4;

        G.edge[6].src = 3;
        G.edge[6].dest = 4;
        G.edge[6].weight = 3;

        G.edge[7].src = 5;
        G.edge[7].dest = 4;
        G.edge[7].weight = 3;
        G.KruskalAlgo();
    }
}
```

Result



```
"C:\Program Files\Java\jdk-18.0.1.1\bin\jav
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4

Process finished with exit code 0
```