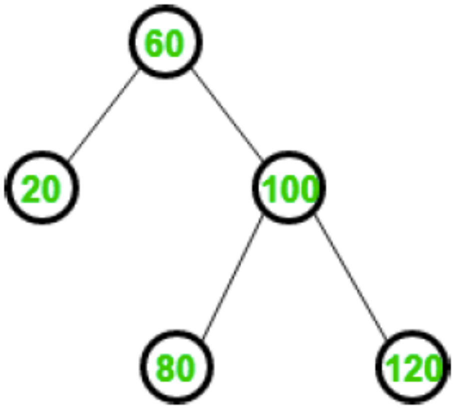
LAB ASSESSMENT 16 OCTOBER 2020

Submitted By:
Abhijeet Chakravorty
Reg No: 2019272002

```
}
```

Q1. Consider the following AVL tree. Write a java program to updated AVL tree after insertion of 70. Display the tree also.



The Mark Split up:

• After inserting all the values and displaying the tree	(10)
• After inserting 70, what type of rotation it undergoes	(5)
• Final tree after inserting 70	(5)

Ans 1.

Code:

Filename: DataOrderAVL.java

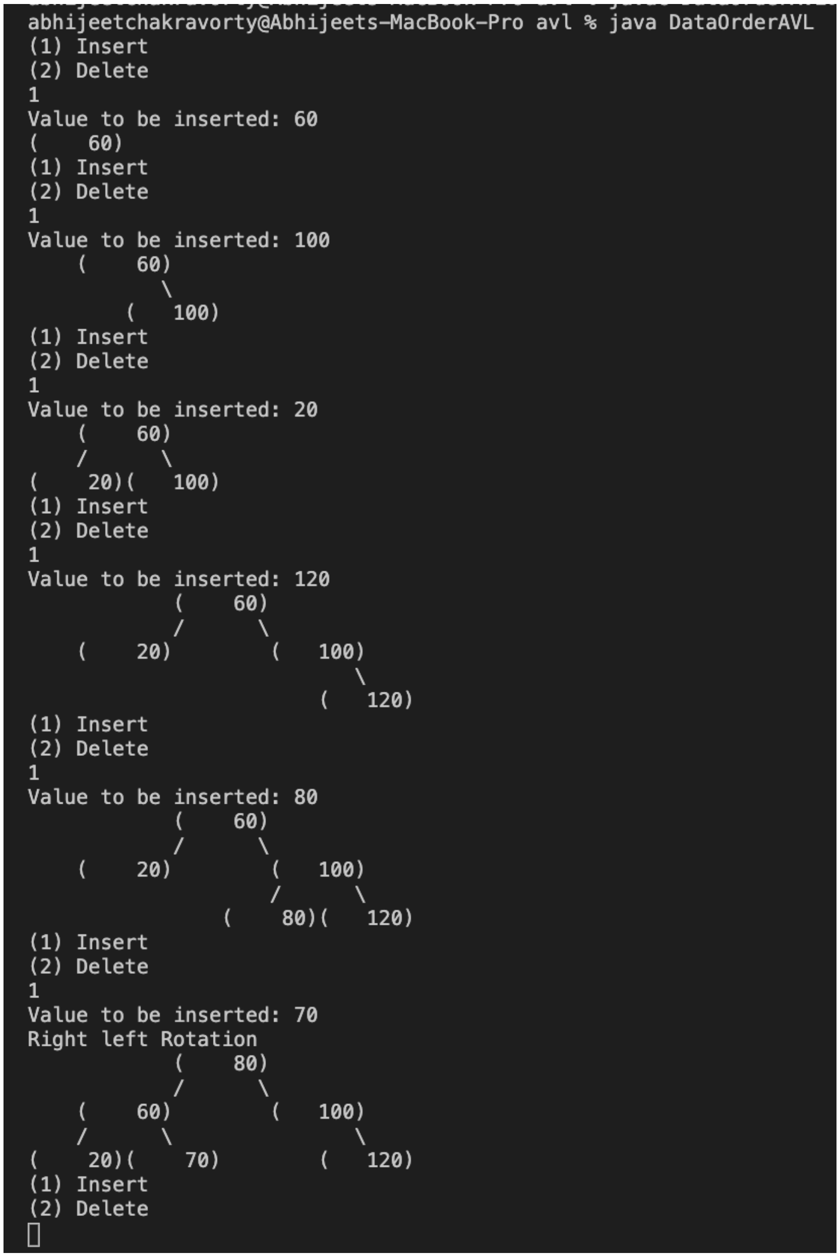
```
import java.util.*;
import java.io.*;

class DataOrderAVL {

    private int calcheight(Node N) {
        if (N == null) {
            return 0;
        }
        return N.height;
    }
```

```
private Node insertData(Node node, int value) {
    if (node == null) {
        return(new Node(value));
    }

    if (value < node.value) {
            node.left  = insertData(node.left, value);
    } else {
            node.right = insertData(node.right, value);
    }

    node.height = Math.max(calcheight(node.left), calcheight(node.right)) + 1;

    int balance = getTreeBalance(node);

    // Left Left Rotation
    if (balance > 1 && value < node.left.value) {
        return rotateRight(node);
    }


    // Right Right Rotation
    if (balance < -1 && value > node.right.value) {
        return rotateLeft(node);
    }

    // Left Right Rotation
    if (balance > 1 && value > node.left.value) {
        node.left =  rotateLeft(node.left);
        return rotateRight(node);
    }

    // Right Left Rotation
    if (balance < -1 && value < node.right.value) {
        node.right = rotateRight(node.right);
        return rotateLeft(node);
    }

    return node;
}

private Node rotateRight(Node y) {
    Node x = y.left;
    Node data = x.right;
    x.right = y;
    y.left = data;
    y.height = Math.max(calcheight(y.left), calcheight(y.right))+1;
    x.height = Math.max(calcheight(x.left), calcheight(x.right))+1;
    return x; // return new root
}


private Node rotateLeft(Node x) {
    Node y = x.right;
    Node data = y.left;
    // Perform rotation
    y.left = x;
    x.right = data;
    //  Update heights
    x.height = Math.max(calcheight(x.left), calcheight(x.right))+1;
    y.height = Math.max(calcheight(y.left), calcheight(y.right))+1;
    // Return new root
    return y;
}

public class Node {
    private Node left, right, parent;
    private int height = 1;
    private int value;

    private Node (int val) {
        this.value = val;
    }
}

private int getTreeBalance(Node N) {
    if (N == null) {
        return 0;
    }
    return calcheight(N.left) - calcheight(N.right);
}

public void traversePreOrder(Node root) {
    if (root != null) {
        traversePreOrder(root.left);
        System.out.printf("%d ", root.value);
        traversePreOrder(root.right);
    }
}

private Node minValueNode(Node node) {
    Node current = node;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}

private Node deleteNodeData(Node root, int value) {
    if (root == null) {
        return root;
    }

    if ( value < root.value ) {
        root.left = deleteNodeData(root.left, value);
    } else if( value > root.value ) {
        root.right = deleteNodeData(root.right, value);
    } else {
        if( (root.left == null) || (root.right == null) ) {
            Node temp;
            if (root.left != null) {
                temp = root.left;
            } else {
                temp = root.right;
            }

            if(temp == null) {
                temp = root;
                root = null;
            } else {
                root = temp;
            }
            temp = null;
        } else {
            Node temp = minValueNode(root.right);
            root.value = temp.value;
            root.right = deleteNodeData(root.right, temp.value);
        }
    }

    if (root == null) {
        return root;
    }

    root.height = Math.max(calcheight(root.left), calcheight(root.right)) + 1;
    int balance = getTreeBalance(root);
    if (balance > 1 && getTreeBalance(root.left) >= 0) {
        return rotateRight(root);
    }

    if (balance > 1 && getTreeBalance(root.left) < 0) {
        root.left =  rotateLeft(root.left);
        return rotateRight(root);
    }

    if (balance < -1 && getTreeBalance(root.right) <= 0) {
        return rotateLeft(root);
    }

    if (balance < -1 && getTreeBalance(root.right) > 0) {
        root.right = rotateRight(root.right);
        return rotateLeft(root);
    }

    return root;
}

public void print(Node root) {

    if(root == null) {
        System.out.println("(XXXXXX)");
        return;
    }

    int height = root.height,
        width = (int)Math.pow(2, height-1);


    List<Node> current = new ArrayList<Node>(1),
        next = new ArrayList<Node>(2);
    current.add(root);

    final int maxHalfLength = 4;
    int elements = 1;

    StringBuilder sb = new StringBuilder(maxHalfLength*width);
    for(int i = 0; i < maxHalfLength*width; i++) {
        sb.append(' ');
    }

    String textBuffer;
```

```
// Iterating through height levels.
for(int i = 0; i < height; i++) {

    sb.setLength(maxHalfLength * ((int)Math.pow(2, height-1-i) - 1));

    // Creating spacer space indicator.
    textBuffer = sb.toString();

    // Print tree node elements
    for(Node n : current) {

        System.out.print(textBuffer);

        if(n == null) {
            System.out.print("        ");
            next.add(null);
            next.add(null);
        } else {

            System.out.printf("(%6d)", n.value);
            next.add(n.left);
            next.add(n.right);

        }

        System.out.print(textBuffer);

    }

    System.out.println();
    // Print tree node extensions for next level.
    if(i < height - 1) {

        for(Node n : current) {

            System.out.print(textBuffer);

            if(n == null)
                System.out.print("        ");
            else
                System.out.printf("%s      %s",
                        n.left == null ? " " : "/", n.right == null ? " " : "\\");

            System.out.print(textBuffer);

        }
```

```
                System.out.println();

            }

            elements *= 2;
            current = next;
            next = new ArrayList<Node>(elements);
        }
    }

    public static void main(String args[]) {
        DataOrderAVL t = new DataOrderAVL();
        Node root = null;
        while (true) {
            System.out.println("(1) Insert");
            System.out.println("(2) Delete");

            try {
                BufferedReader bufferRead = new BufferedReader(new
InputStreamReader(System.in));
                String s = bufferRead.readLine();

                if (Integer.parseInt(s) == 1) {
                    System.out.print("Value to be inserted: ");
                    root = t.insertData(root, Integer.parseInt(bufferRead.readLine()));
                }
                else if (Integer.parseInt(s) == 2) {
                    System.out.print("Value to be deleted: ");
                    root = t.deleteNodeData(root, Integer.parseInt(bufferRead.readLine()));
                }
                else {
                    System.out.println("Invalid choice, try again!");
                    continue;
                }

                t.print(root);
            }
            catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

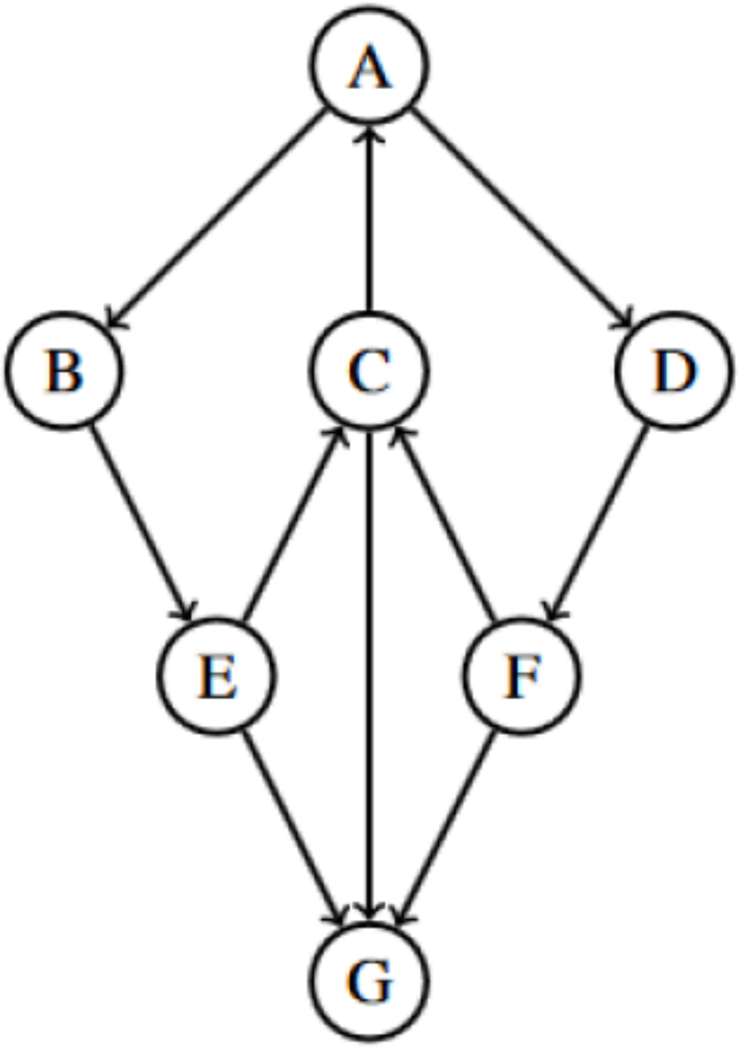
Output:



```
abhijeetchakravorty@Abhijeets-MacBook-Pro avl % java DataOrderAVL
(1) Insert
(2) Delete
1
Value to be inserted: 60
(    60)
(1) Insert
(2) Delete
1
Value to be inserted: 100
    (    60)
           \
        (   100)
(1) Insert
(2) Delete
1
Value to be inserted: 20
    (    60)
    /      \
(    20)(   100)
(1) Insert
(2) Delete
1
Value to be inserted: 120
            (    60)
            /      \
    (    20)        (   100)
                           \
                        (   120)
(1) Insert
(2) Delete
1
Value to be inserted: 80
            (    60)
            /      \
    (    20)        (   100)
                    /      \
                (    80)(   120)
(1) Insert
(2) Delete
1
Value to be inserted: 70
Right left Rotation
            (    80)
            /      \
    (    60)        (   100)
    /      \               \
(    20)(    70)        (   120)
(1) Insert
(2) Delete
```

a. **Displayed tree after inserting all values**
b. **After inserting 70 it undergoes Right Left Rotation**
c. **Final tree after inserting 70 is mentioned as well in the image**

Q2. Perform a depth first search on the following graph starting at A.



Ans 2.

<u>Code:</u>

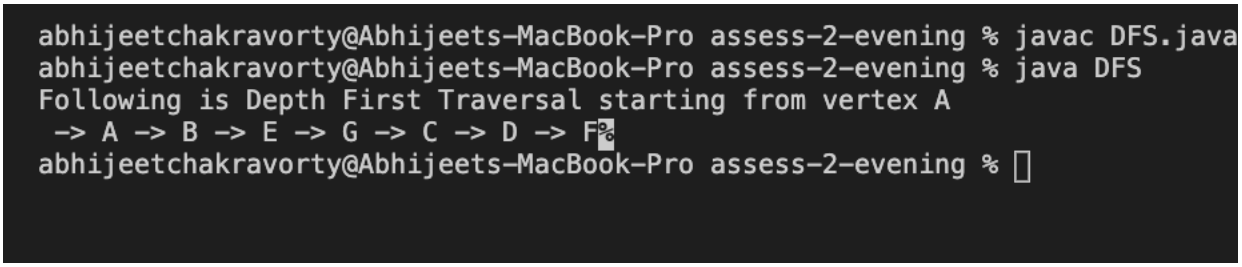
File Name: DFS.java

```
import java.util.*;
class DFS {
	private int V;
	private LinkedList < Integer > adj[];
```

```
@SuppressWarnings("unchecked")
DFS(int v) {
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
        adj[i] = new LinkedList();
}

void addFinalEdge(int v, int w) {
    adj[v].add(w);
}

void Utility(int v, boolean visited[]) {
    visited[v] = true;
    System.out.print(" -> " + (char) v);

    Iterator < Integer > i = adj[v].listIterator();
    while (i.hasNext()) {
        int n = i.next();
        if (!visited[n])
            Utility(n, visited);
    }
}

void DFS(int v) {
    boolean visited[] = new boolean[V];
    Utility(v, visited);
}

public static void main(String args[]) {
    DFS g = new DFS((int)'H');

    g.addFinalEdge('A', 'B');
    g.addFinalEdge('A', 'D');
    g.addFinalEdge('B', 'E');
    g.addFinalEdge('E', 'G');
    g.addFinalEdge('E', 'C');
    g.addFinalEdge('D', 'F');
    g.addFinalEdge('F', 'G');
    g.addFinalEdge('F', 'C');
    g.addFinalEdge('C', 'A');
    g.addFinalEdge('C', 'G');

    System.out.println("Following is Depth First Traversal starting from vertex A");
```

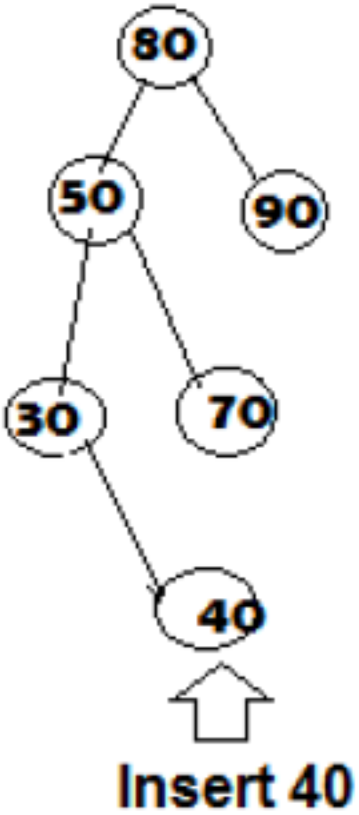
```
        g.DFS((int)'A');
    }
}
```

Output:



```
abhijeetchakravorty@Abhijeets-MacBook-Pro assess-2-evening % javac DFS.java
abhijeetchakravorty@Abhijeets-MacBook-Pro assess-2-evening % java DFS
Following is Depth First Traversal starting from vertex A
 -> A -> B -> E -> G -> C -> D -> F%
abhijeetchakravorty@Abhijeets-MacBook-Pro assess-2-evening %
```

Q 3. Write a java program to insert 40 in a given splay tree? (20)



Ans 3.

Code:

```
import java.util.*;
import java.io.*;
class SplayTree {

    public class Node {
        private Node left, right, parent;
        private int height = 1;
        private int value;

        private Node(int val) {
            this.value = val;
        }
    }

    private int calcheight(Node N) {
        if (N == null) {
            return 0;
        }
        return N.height;
    }

    private Node insertData(Node node, int value) {
        if (node == null) {
            return (new Node(value));
        }

        if (value < node.value) {
            node.left = insertData(node.left, value);
        } else {
            node.right = insertData(node.right, value);
        }

        node.height = Math.max(calcheight(node.left), calcheight(node.right)) +
1;

        return node;
    }
```

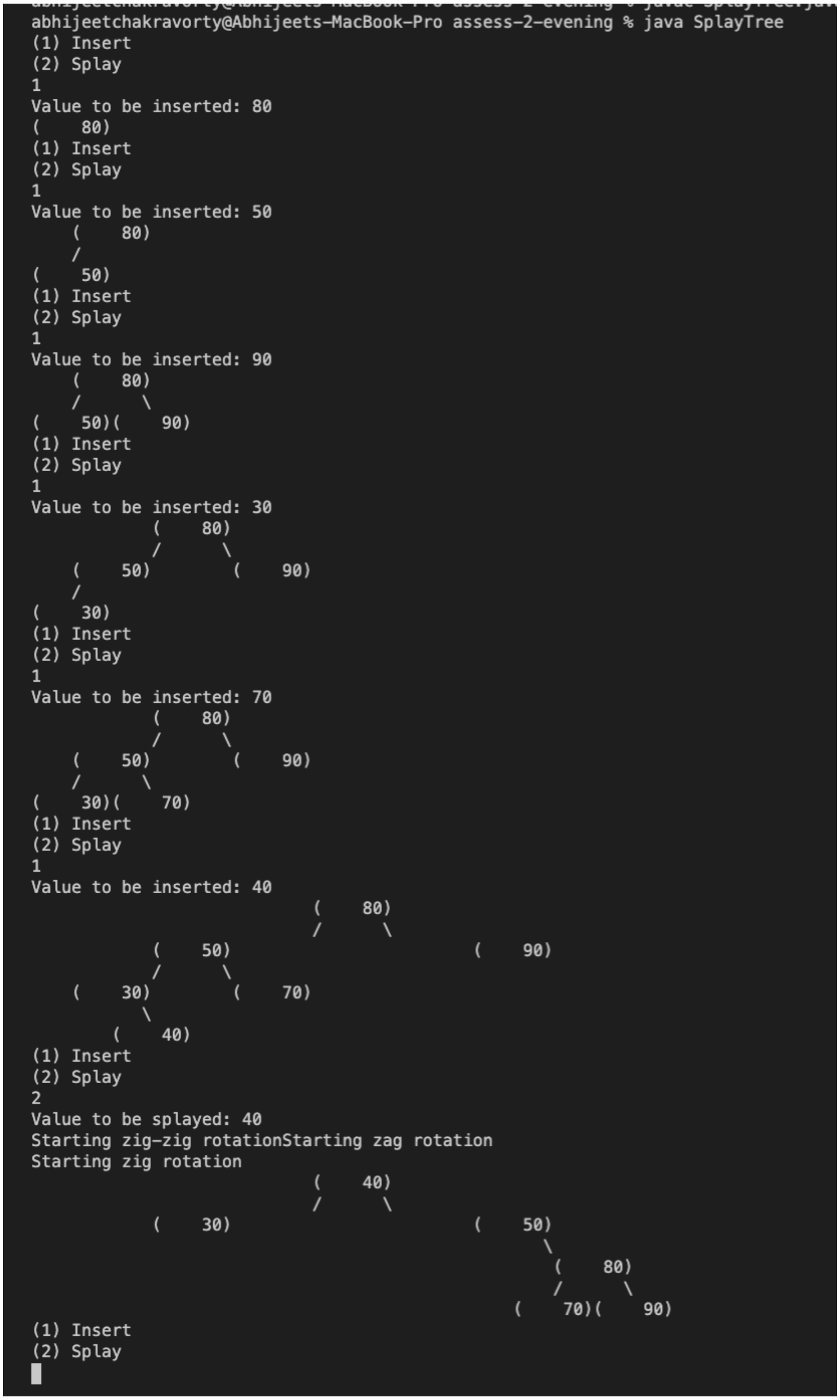
```
private Node rotateRight(Node y) {
    Node x = y.left;
    Node data = x.right;
    x.right = y;
    y.left = data;
    y.height = Math.max(calcheight(y.left), calcheight(y.right)) + 1;
    x.height = Math.max(calcheight(x.left), calcheight(x.right)) + 1;
    return x; // return new root
}


private Node rotateLeft(Node x) {
    Node y = x.right;
    Node data = y.left;
    // Perform rotation
    y.left = x;
    x.right = data;
    //  Update heights
    x.height = Math.max(calcheight(x.left), calcheight(x.right)) + 1;
    y.height = Math.max(calcheight(y.left), calcheight(y.right)) + 1;
    // Return new root
    return y;
}

private Node minValueNode(Node node) {
    Node current = node;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}

private Node deleteNodeData(Node root, int value) {
    if (root == null) {
        return root;
    }

    if (value < root.value) {
        root.left = deleteNodeData(root.left, value);
    } else if (value > root.value) {
        root.right = deleteNodeData(root.right, value);
    } else {
        if ((root.left == null) || (root.right == null)) {
                Node temp;
                if (root.left != null) {
                    temp = root.left;
                } else {
                    temp = root.right;
                }

                if (temp == null) {
                    temp = root;
                    root = null;
                } else {
                    root = temp;
                }
                temp = null;
            } else {
                Node temp = minValueNode(root.right);
                root.value = temp.value;
                root.right = deleteNodeData(root.right, temp.value);
            }
        }

        if (root == null) {
            return root;
        }

        root.height = Math.max(calcheight(root.left), calcheight(root.right)) + 1;
        return root;
    }

    public void print(Node root) {

        if (root == null) {
            System.out.println("(XXXXXX)");
            return;
        }

        int height = root.height,
            width = (int) Math.pow(2, height - 1);


        List < Node > current = new ArrayList < Node > (1),
            next = new ArrayList < Node > (2);
        current.add(root);
```

```
final int maxHalfLength = 4;
int elements = 1;

StringBuilder sb = new StringBuilder(maxHalfLength * width);
for (int i = 0; i < maxHalfLength * width; i++) {
    sb.append(' ');
}

String textBuffer;

// Iterating through height levels.
for (int i = 0; i < height; i++) {

    sb.setLength(maxHalfLength * ((int) Math.pow(2, height - 1 - i) - 1));

    // Creating spacer space indicator.
    textBuffer = sb.toString();

    // Print tree node elements
    for (Node n: current) {

        System.out.print(textBuffer);

        if (n == null) {
            System.out.print("        ");
            next.add(null);
            next.add(null);
        } else {

            System.out.printf("(%6d)", n.value);
            next.add(n.left);
            next.add(n.right);

        }

        System.out.print(textBuffer);

    }

    System.out.println();
    // Print tree node extensions for next level.
    if (i < height - 1) {
```

```
                for (Node n: current) {

                    System.out.print(textBuffer);

                    if (n == null) {
                        System.out.print("        ");
                    } else {
                        System.out.printf("%s      %s", n.left == null ? " " : "/",
n.right == null ? " " : "\\");
                    }

                    System.out.print(textBuffer);

                }

                System.out.println();

            }

            elements *= 2;
            current = next;
            next = new ArrayList < Node > (elements);
        }
    }

    private Node splayNodeData(Node root, int value) {
        // Base cases: root is null or
        // key is present at root
        if (root == null || root.value == value) {
            return root;
        }
        root.height = Math.max(calcheight(root.left), calcheight(root.right)) + 1;

        // value lies in left subtree
        if (root.value > value) {
            // value is not in tree, we are done
            if (root.left == null) return root;

            // Zig-Zig (Left Left)
            if (root.left.value > value) {
                // First recursively bring the
                // value as root of left-left
        System.out.print("Starting zig-zig rotation");
        root.left.left = splayNodeData(root.left.left, value);

        // Do first rotation for root,
        // second rotation is done after else
        root = rotateRight(root);
    } else if (root.left.value < value) // Zig-Zag (Left Right)
    {
        // First recursively bring
        // the value as root of left-right
        System.out.print("Starting zig-zag rotation");
        root.left.right = splayNodeData(root.left.right, value);

        // Do first rotation for root.left
        if (root.left.right != null) {
            root.left = rotateLeft(root.left);
        }
    }

    if (root.left != null) {
        System.out.println("Starting zig rotation");
    }
    // Do second rotation for root
    return (root.left == null) ? root : rotateRight(root);
} else { // value lies in right subtree
    // value is not in tree, we are done
    if (root.right == null) {
        return root;
    }

    // Zag-Zig (Right Left)
    if (root.right.value > value) {
        System.out.println("Starting Zag-Zig rotation");
        // Bring the value as root of right-left
        root.right.left = splayNodeData(root.right.left, value);

        // Do first rotation for root.right
        if (root.right.left != null) {
            root.right = rotateRight(root.right);
        }
    } else if (root.right.value < value) // Zag-Zag (Right Right)
    {
        // Bring the value as root of
```

```
                // right-right and do first rotation
                System.out.println("Starting Zag-Zag rotation");
                root.right.right = splayNodeData(root.right.right, value);
                root = rotateLeft(root);
            }

            if (root.right != null) {
                System.out.println("Starting zag rotation");
            }
            // Do second rotation for root
            return (root.right == null) ? root : rotateLeft(root);
        }
    }

    public static void main(String args[]) {
        SplayTree t = new SplayTree();
        Node root = null;
        while (true) {
            System.out.println("(1) Insert");
            System.out.println("(2) Splay");
            try {
                BufferedReader bufferRead = new BufferedReader(new
InputStreamReader(System.in));
                String s = bufferRead.readLine();

                if (Integer.parseInt(s) == 1) {
                    System.out.print("Value to be inserted: ");
                    root = t.insertData(root,
Integer.parseInt(bufferRead.readLine()));
                } else if (Integer.parseInt(s) == 2) {
                    System.out.print("Value to be splayed: ");
                    root = t.splayNodeData(root,
Integer.parseInt(bufferRead.readLine()));
                } else {
                    System.out.println("Invalid choice, try again!");
                    continue;
                }

                t.print(root);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output:



```
abhijeetchakravorty@Abhijeets-MacBook-Pro assess-2-evening % javac SplayTree.jav
abhijeetchakravorty@Abhijeets-MacBook-Pro assess-2-evening % java SplayTree
(1) Insert
(2) Splay
1
Value to be inserted: 80
(    80)
(1) Insert
(2) Splay
1
Value to be inserted: 50
    (    80)
    /
(    50)
(1) Insert
(2) Splay
1
Value to be inserted: 90
    (    80)
    /      \
(    50)(    90)
(1) Insert
(2) Splay
1
Value to be inserted: 30
            (    80)
            /      \
    (    50)        (    90)
    /
(    30)
(1) Insert
(2) Splay
1
Value to be inserted: 70
            (    80)
            /      \
    (    50)        (    90)
    /      \
(    30)(    70)
(1) Insert
(2) Splay
1
Value to be inserted: 40
                            (    80)
                            /      \
            (    50)                        (    90)
            /      \
    (    30)        (    70)
           \
        (    40)
(1) Insert
(2) Splay
2
Value to be splayed: 40
Starting zig-zig rotationStarting zag rotation
Starting zig rotation
                            (    40)
                            /      \
            (    30)                        (    50)
                                                   \
                                                (    80)
                                                /      \
                                            (    70)(    90)

(1) Insert
(2) Splay
```