ESTRUCTURAS DE DATOS

2024

TRABAJO PRACTICO N°4
AVL

Profesores:
HECTOR REINAGA
FERNANDA DANIELA OYARZO
MIRTHA FABIANA MIRANDA

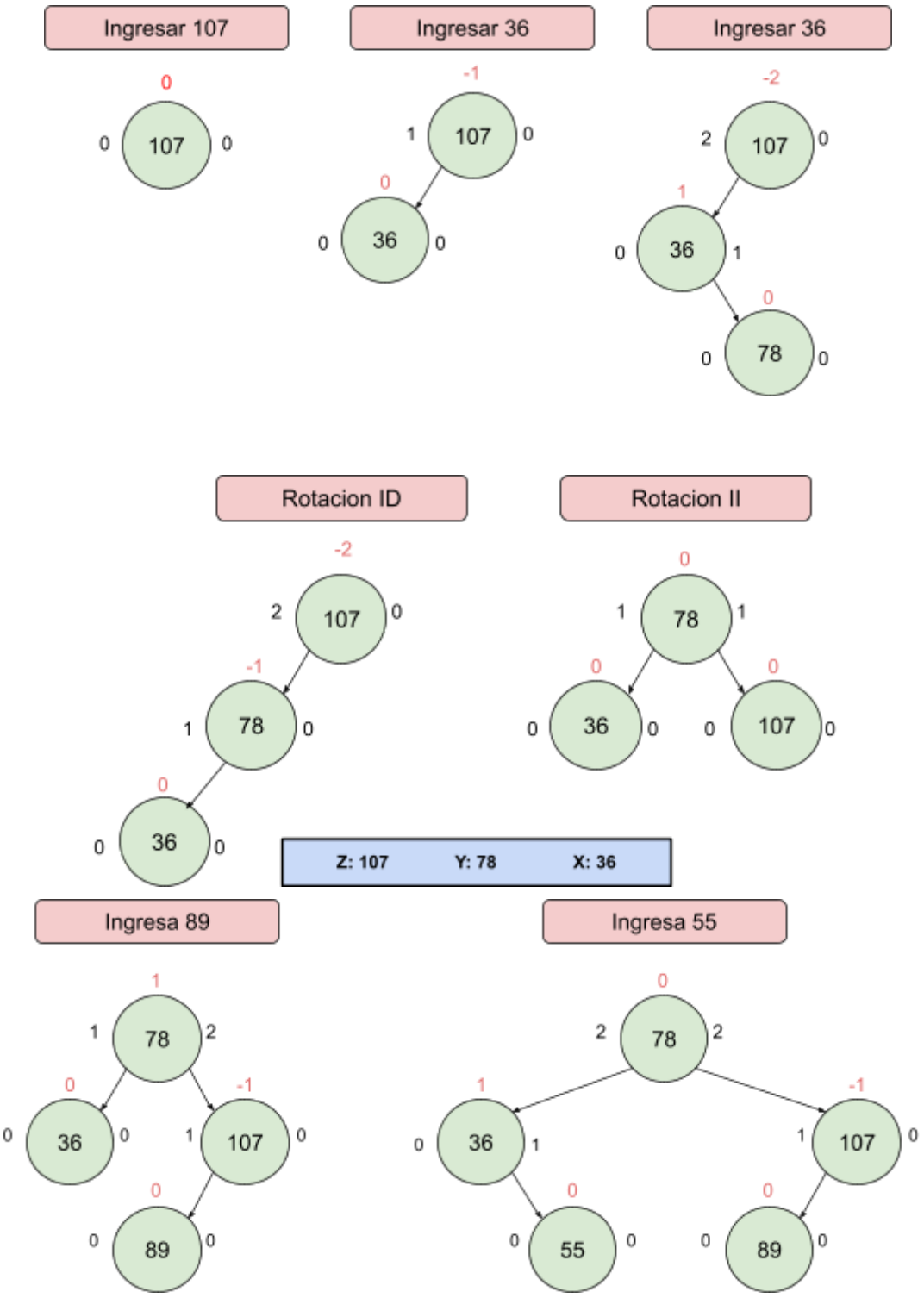
Alumno:
GONZALO ALEJANDRO ULLOA

Índice

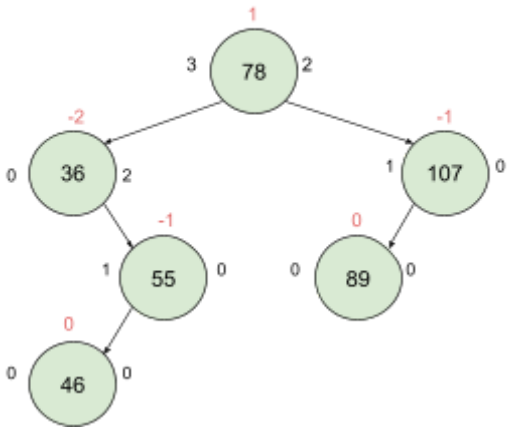
Desarrollo

3.

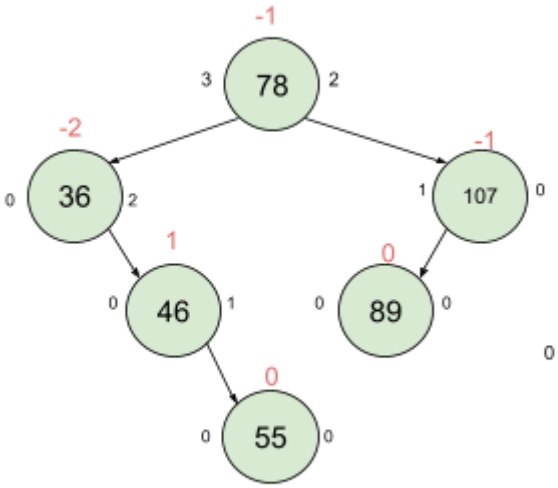
Inserción



Ingresa 46

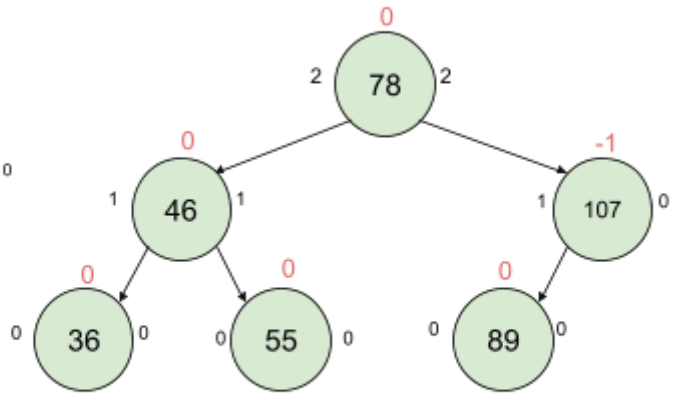


Rotación DI



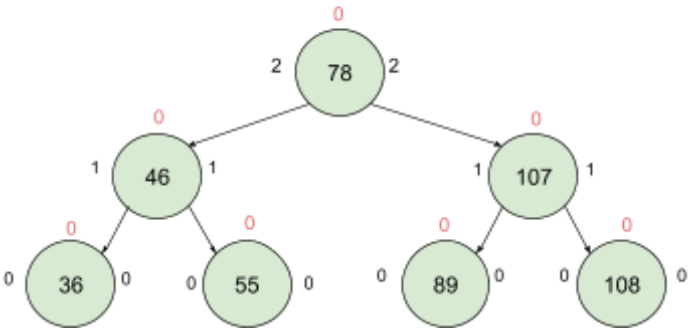
Z: 36 Y: 55 X: 46

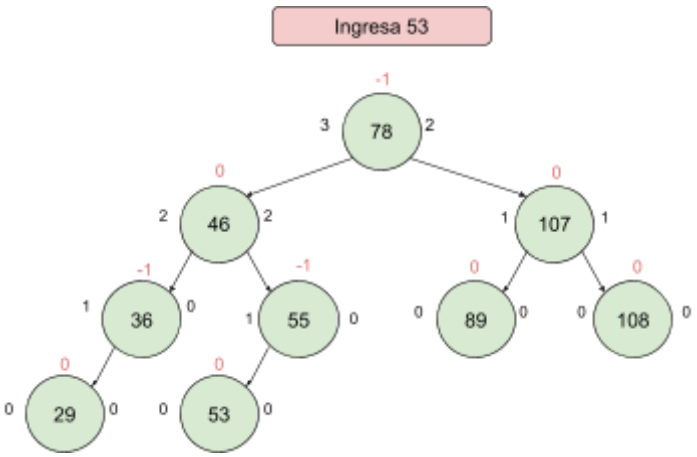
Rotación DD

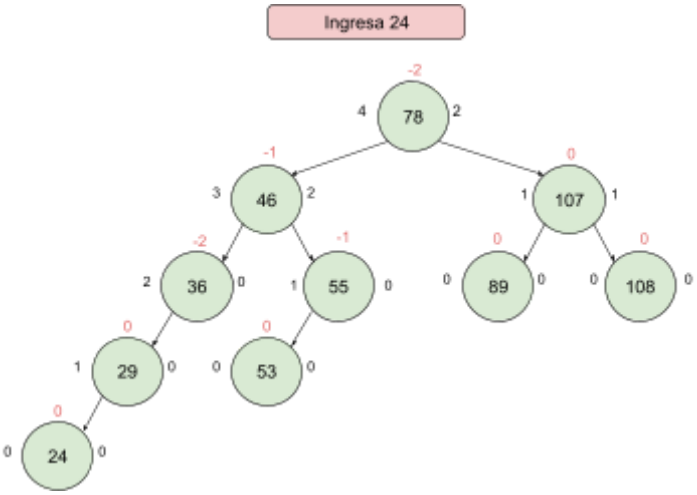


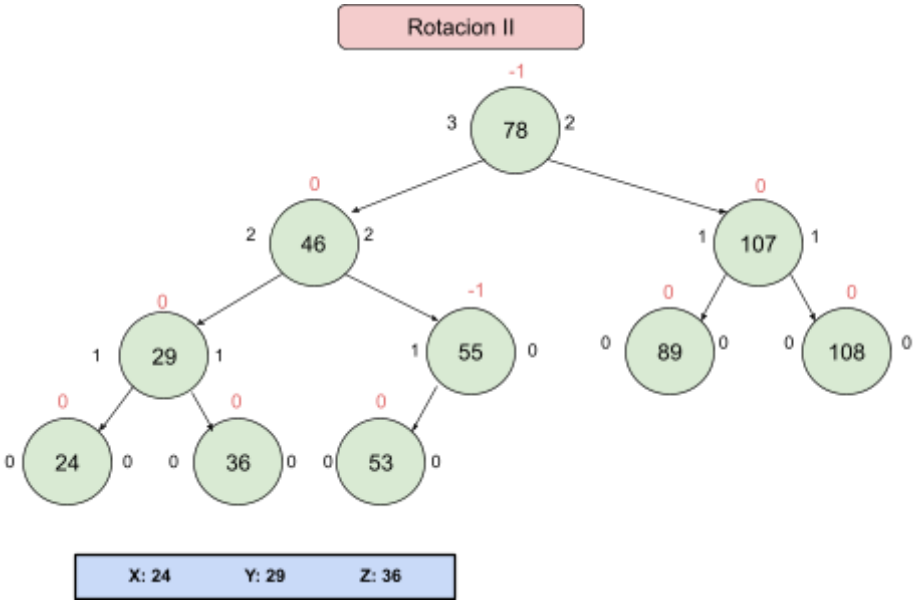
Z: 36 Y: 46 X: 55

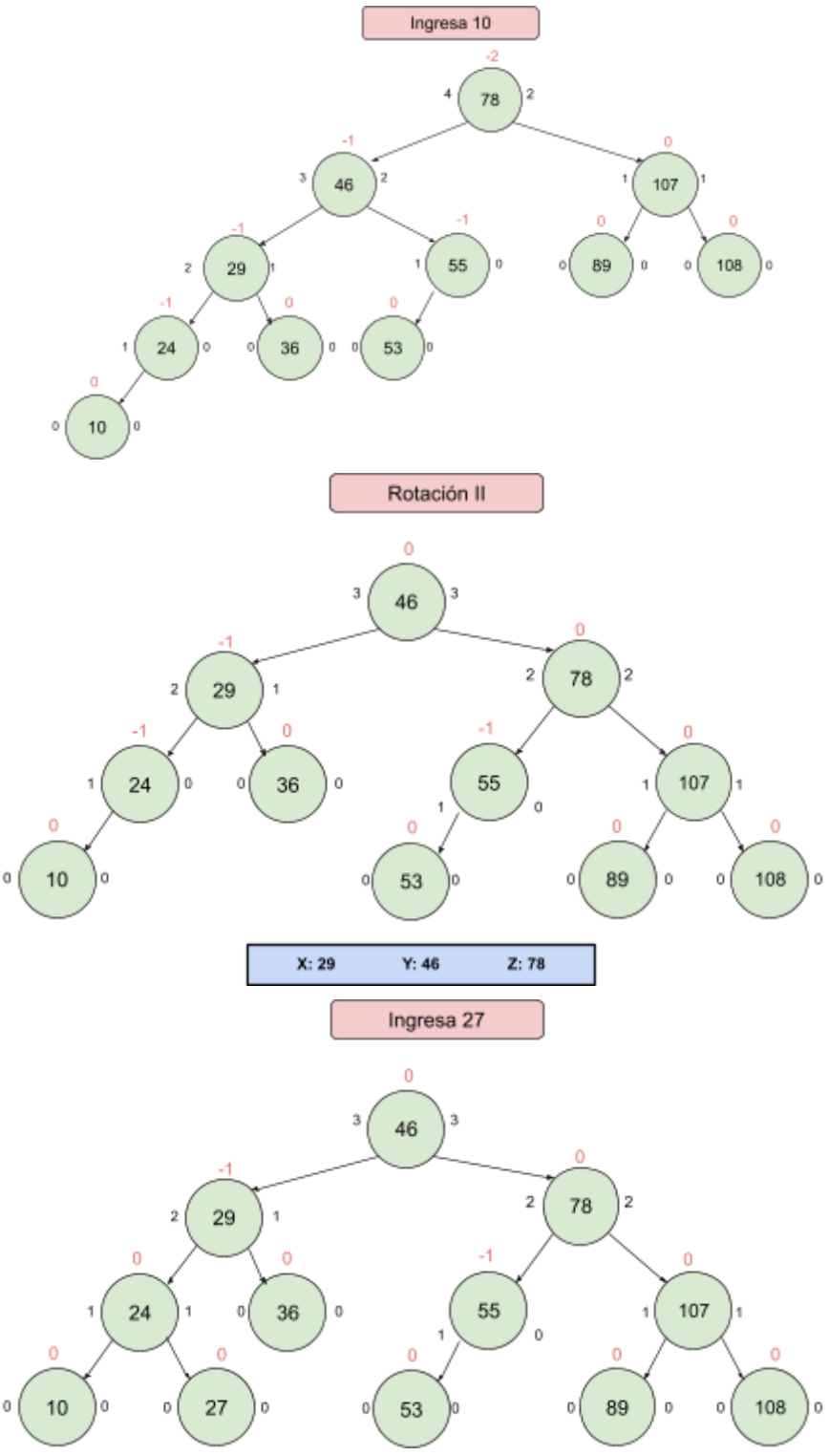
Ingresa 108

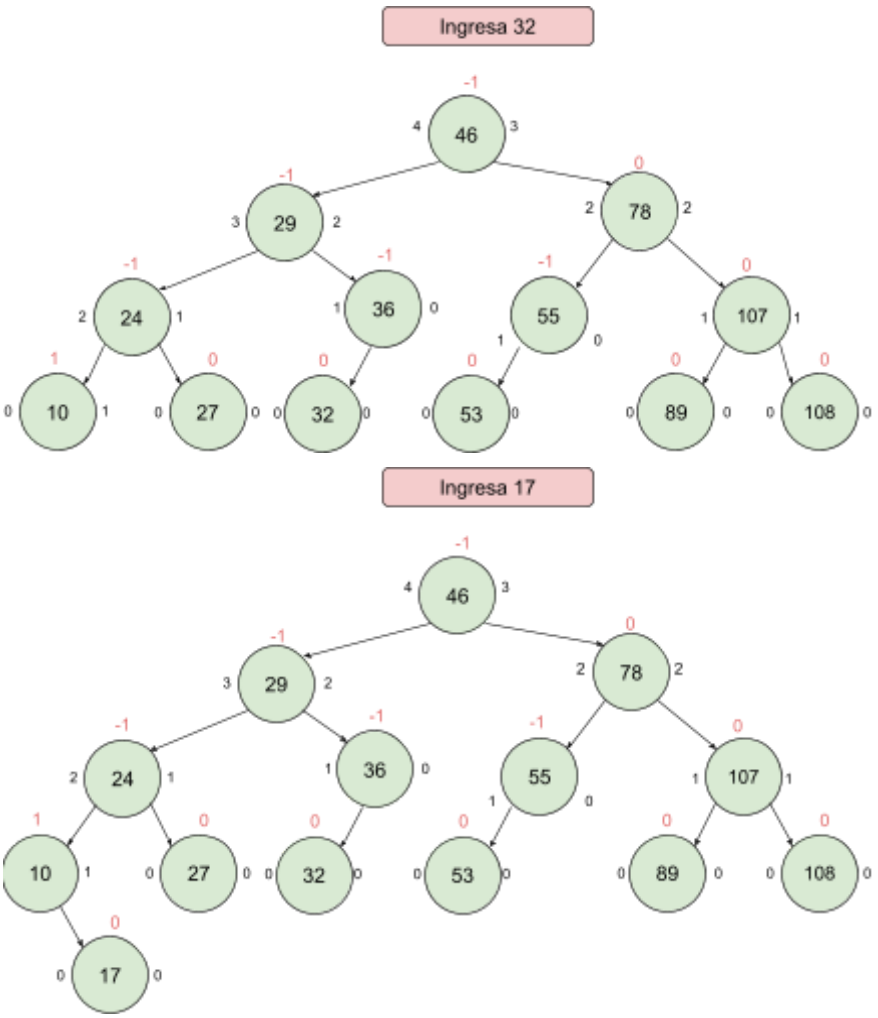



Ingresa 53
78
46
107
36
55
89
108
29
53

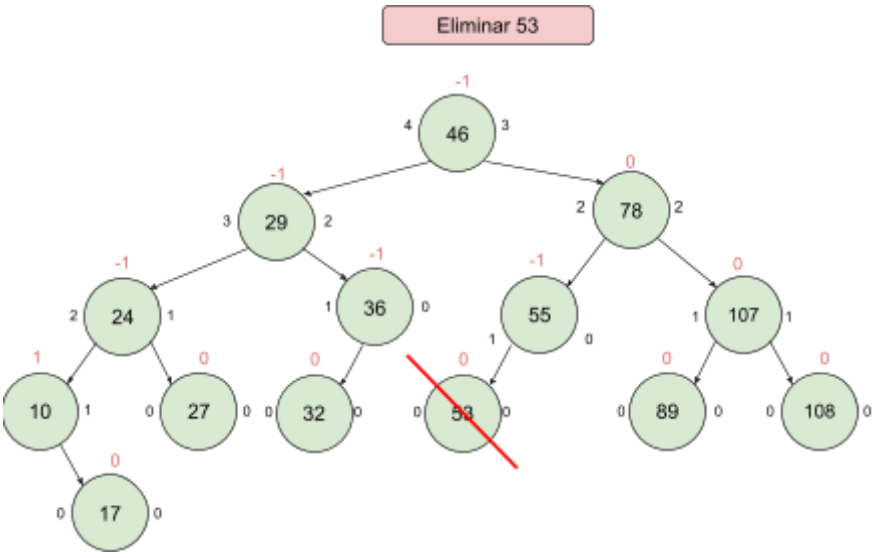

Ingresa 24
78
46
107
36
55
89
108
29
53
24

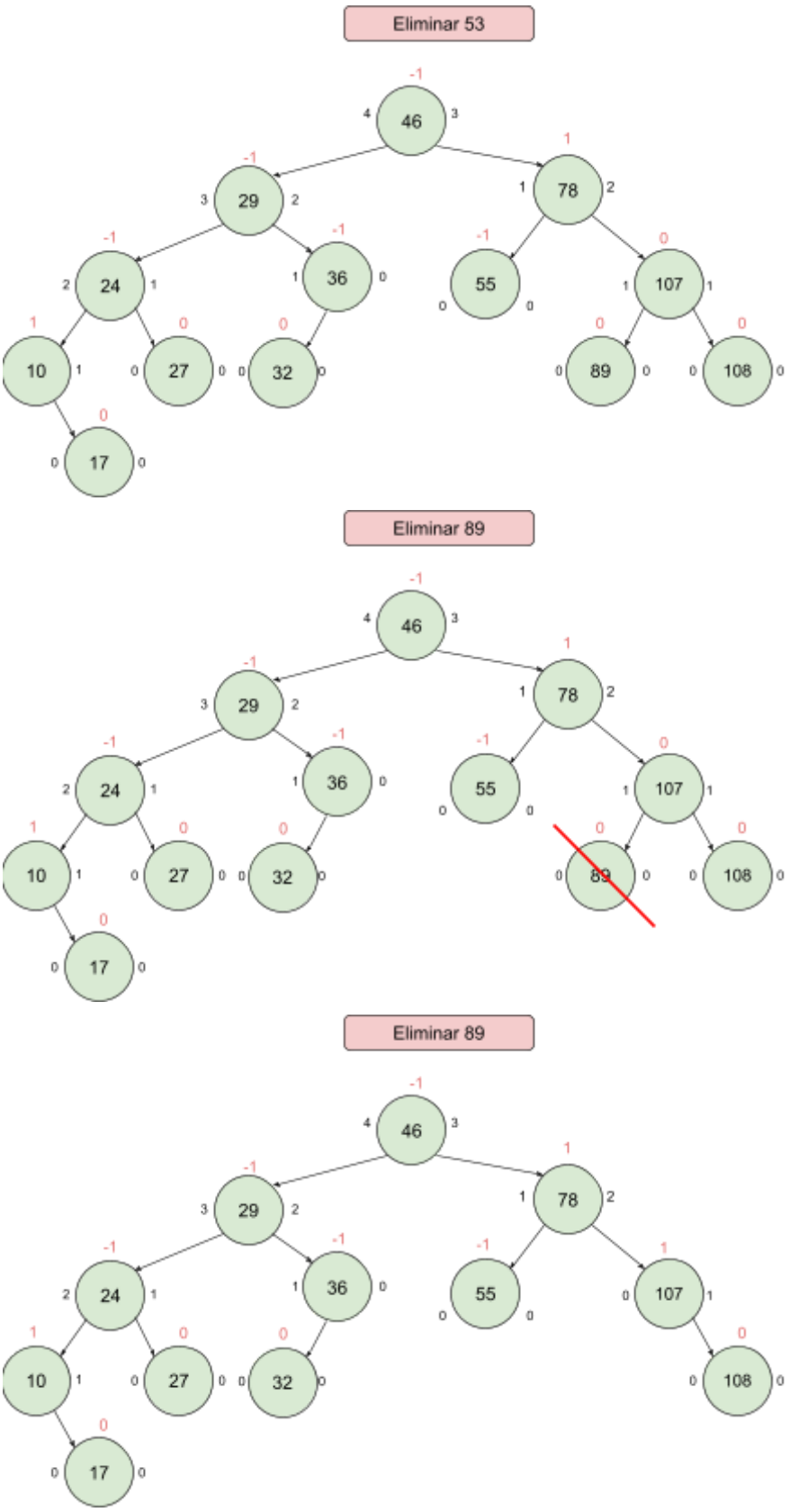

Rotacion II
78
46
107
29
55
89
108
24
36
53
X: 24
Y: 29
Z: 36

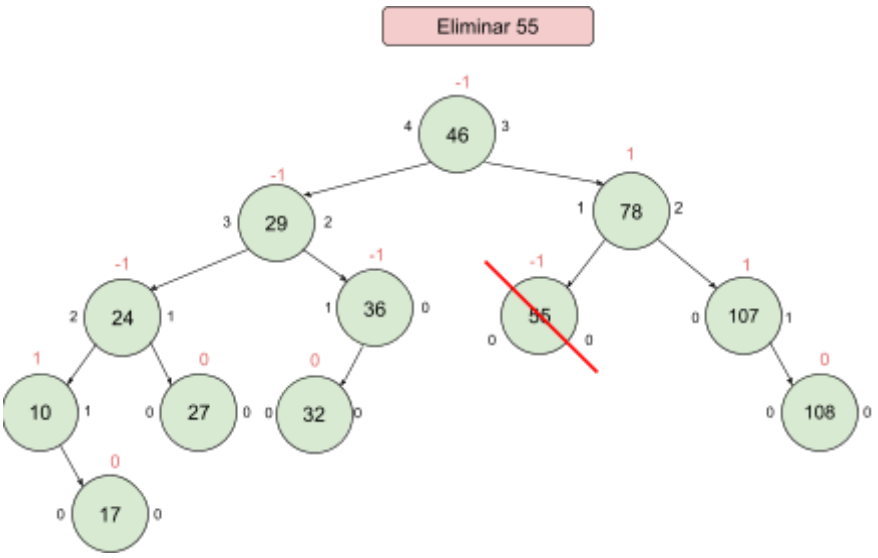

Ingresa 10
78
46
107
29
55
89
108
24
36
53
10
Rotación II
46
29
78
24
36
55
107
10
53
89
108
X: 29
Y: 46
Z: 78
Ingresa 27
46
29
78
24
36
55
107
10
27
53
89
108

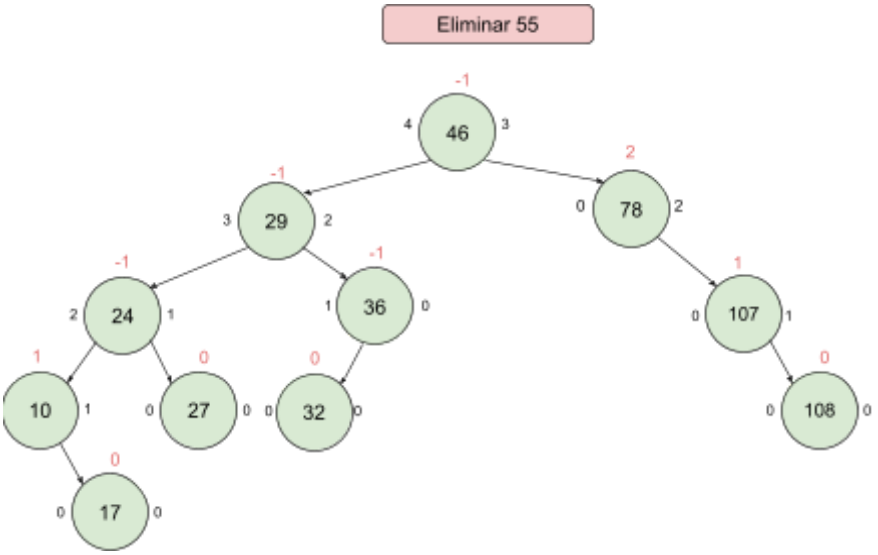


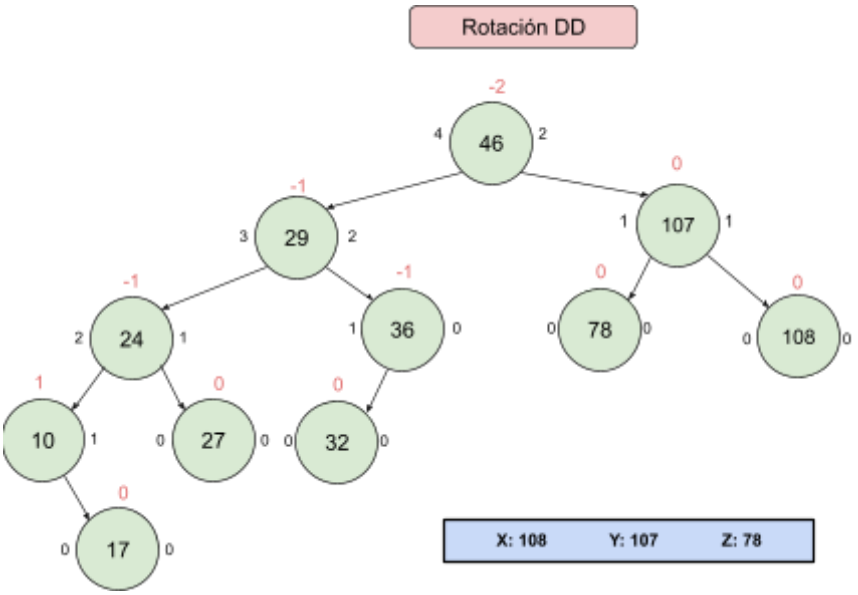
Eliminación

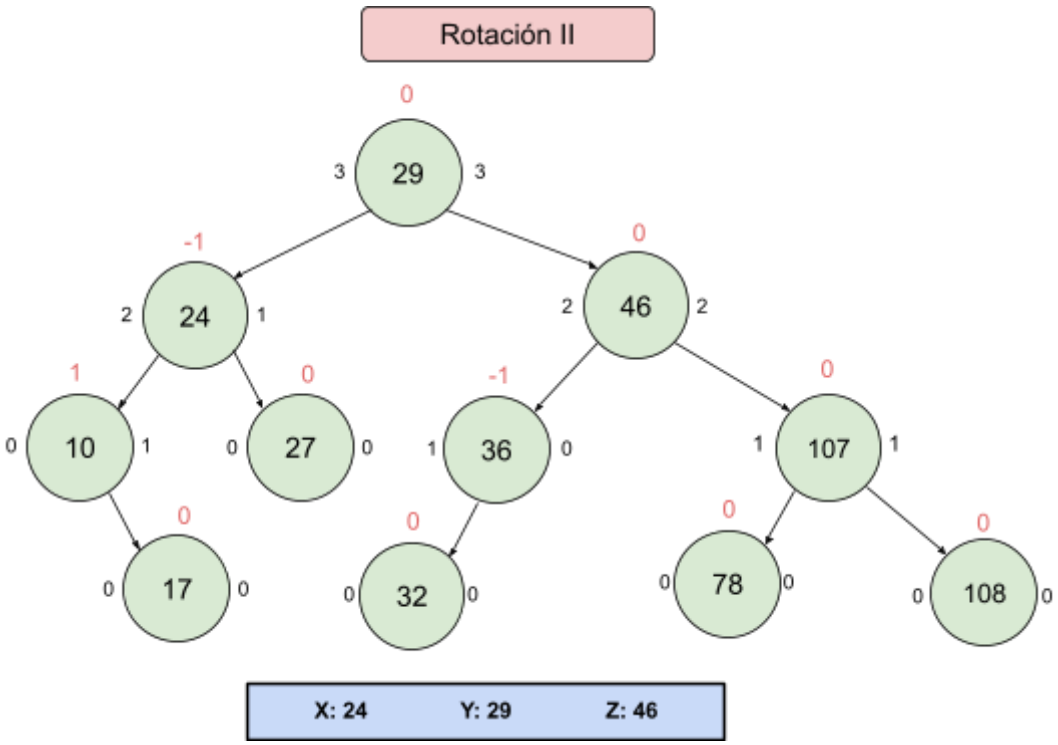



Eliminar 53
46
29
78
24
36
55
107
10
27
32
89
108
17
Eliminar 89
46
29
78
24
36
55
107
10
27
32
89
108
17
Eliminar 89
46
29
78
24
36
55
107
10
27
32
108
17


Eliminar 55
46
29
78
24
36
55
107
10
27
32
108
17


Eliminar 55
46
29
78
24
36
107
10
27
32
108
17


Rotación DD
46
29
107
24
36
78
108
10
27
32
17
X: 108
Y: 107
Z: 78



5.

Clase Nodo String

```
public class AVLNodeString {
    private String content;
    private byte balance;
    private AVLNodeString left, right;
    public AVLNodeString(String content) {
        this.content = content;
        this.balance = 0;
        this.left = null;
        this.right = null;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public byte getBalance() {
        return balance;
    }
```

```
    public void setBalance(int balance) {
        this.balance = (byte) balance;
    }
    public AVLNodeString getLeft() {
        return left;
    }
    public void setLeft(AVLNodeString left) {
        this.left = left;
    }
    public AVLNodeString getRight() {
        return right;
    }
    public void setRight(AVLNodeString right) {
        this.right = right;
    }
}
```

Clase Arbol AVL

```
public class AVLTreeString {
    private AVLNodeString root;
    private boolean grown, shrunk, found;
    public AVLTreeString() {
        root = null;
        grown = false;
        shrunk = false;
        found = false;
    }
    public boolean search(String word) {
        AVLNodeString node = root;
        while (node != null) {
            int comparison = word.compareTo(node.getContent());
            if (comparison == 0) {
                return true;
            }
            if (comparison < 0) {
                node = node.getLeft();
            } else {
                node = node.getRight();
            }
```

```
        }
        return false;
    }
    public boolean insert(String word) {
        found = false;
        grown = true;
        root = insert(root, word);
        return !found;
    }
    private AVLNodeString insert(AVLNodeString node, String word)
{
        if (node == null) {
            return new AVLNodeString(word);
        }
        int comparison = word.compareTo(node.getContent());
        if (comparison == 0) {
            found = true;
            grown = false;
        } else if (comparison < 0) {
            node.setLeft(insert(node.getLeft(), word));
            if (grown) node.setBalance(node.getBalance() - 1);
        } else {
            node.setRight(insert(node.getRight(), word));
            if (grown) node.setBalance(node.getBalance() + 1);
        }
        return balanceNode(node);
    }
    private AVLNodeString balanceNode(AVLNodeString node) {
        switch (node.getBalance()) {
            case -2:
                if (node.getLeft().getBalance() == 1)
rotateLeft(node.getLeft());
                node = rotateRight(node);
                grown = false;
                break;
            case -1:
                break;
            case 0:
                grown = false;
                break;
            case 1:
```

```
                break;
            case 2:
                if (node.getRight().getBalance() == -1)
rotateRight(node.getRight());
                node = rotateLeft(node);
                grown = false;
                break;
            default:
                balanceError(node);
                break;
        }
        return node;
    }
    private void balanceError(AVLNodeString node) {
        System.out.println("Error en el valor de equilibrio: " +
node.getBalance() + " y en el contenido: " + node.getContent());
    }
    private AVLNodeString rotateRight(AVLNodeString node) {
        AVLNodeString leftNode = node.getLeft();
        node.setLeft(leftNode.getRight());
        leftNode.setRight(node);

        node.setBalance(node.getBalance() + 1 -
Math.min(leftNode.getBalance(), 0));
        leftNode.setBalance(leftNode.getBalance() + 1 +
Math.max(node.getBalance(), 0));

        return leftNode;
    }
    private AVLNodeString rotateLeft(AVLNodeString node) {
        AVLNodeString rightNode = node.getRight();
        node.setRight(rightNode.getLeft());
        rightNode.setLeft(node);

        node.setBalance(node.getBalance() - 1 -
Math.max(rightNode.getBalance(), 0));
        rightNode.setBalance(rightNode.getBalance() - 1 +
Math.min(node.getBalance(), 0));

        return rightNode;
    }
```

```
    public boolean delete(String word) {
        found = true;
        shrunk = true;
        root = delete(root, word);
        return found;
    }
    private AVLNodeString delete(AVLNodeString node, String word)
{
        if (node == null) {
            found = false;
            shrunk = false;
            return null;
        }
        int comparison = word.compareTo(node.getContent());
        if (comparison == 0) {
            if (node.getLeft() == null) {
                return node.getRight();
            }
            if (node.getRight() == null) {
                return node.getLeft();
            }
            String minValue = minValue(node.getRight());
            node.setContent(minValue);
            node.setRight(delete(node.getRight(), minValue));
            if (shrunk) node.setBalance(node.getBalance() - 1);
        } else if (comparison < 0) {
            node.setLeft(delete(node.getLeft(), word));
            if (shrunk) node.setBalance(node.getBalance() + 1);
        } else {
            node.setRight(delete(node.getRight(), word));
            if (shrunk) node.setBalance(node.getBalance() - 1);
        }
        return balanceNodeAfterDeletion(node);
    }
    private AVLNodeString balanceNodeAfterDeletion(AVLNodeString
node) {
        switch (node.getBalance()) {
            case -2:
                if (node.getLeft().getBalance() == 1)
rotateLeft(node.getLeft());
                node = rotateRight(node);
```

```
                break;
            case -1:
                shrunk = false;
                break;
            case 0:
                break;
            case 1:
                shrunk = false;
                break;
            case 2:
                if (node.getRight().getBalance() == -1)
rotateRight(node.getRight());
                node = rotateLeft(node);
                break;
            default:
                balanceError(node);
                break;
        }
        return node;
    }
    private String minValue(AVLNodeString node) {
        while (node.getLeft() != null) {
            node = node.getLeft();
        }
        return node.getContent();
    }
    public int height() {
        return height(root);
    }
    private int height(AVLNodeString node) {
        if (node == null) {
            return 0;
        }
        return 1 + Math.max(height(node.getLeft()),
height(node.getRight()));
    }
    public int internal() {
        return internal(root, 0);
    }
    private int internal(AVLNodeString node, int height) {
        if (node == null) {
```

```
            return 0;
        }
        return height + internal(node.getLeft(), height + 1) +
internal(node.getRight(), height + 1);
    }
}
```

Main

```
class Main2{
    public static void main(String[] args) {
        AVLTreeString avl= new AVLTreeString();
        avl.insert("RIO");
        avl.insert("Rio");
        avl.insert("rio");
        System.out.println("Esta 'rio': "+avl.search("rio"));
        System.out.println("Esta 'RIO': "+avl.search("rio"));
        System.out.println("Esta 'Rio': "+avl.search("rio"));
        System.out.println("\nBorrando rio y Rio");
        avl.delete("rio");
        avl.delete("Rio");
        System.out.println("Esta 'rio': "+avl.search("rio"));
        System.out.println("Esta 'RIO': "+avl.search("RIO"));
        System.out.println("Esta 'Rio': "+avl.search("Rio"));
    }
}
```

Resultado de consola.

```
'Main2'
Esta 'rio': true
Esta 'RIO': true
Esta 'Rio': true

Borrando rio y Rio
Esta 'rio': false
Esta 'RIO': true
Esta 'Rio': false
PS C:\Users\GonzaloUlloa\Desktop\gon\EDA>
```