# *INFORME*

Sobre la presentación de esta tarea relacionado al tema AVL que es un tipo de árbol que está relacionado con un Factor de Equilibrio el cual se debe de hallar con la diferencia de altura del lado derecho menos el lado izquierdo esta diferencia debe de estar en un rango de (-1, 0, 1) de lo contrario se tiene que hacer rotaciones. Los tipos de rotaciones son 4:
A) Rotación Simple a la Izquierda (RSI)
B) Rotación Simple a la Derecha (RSD)
C) Rotación Doble Izquierda (RSD - RSI)
D) Rotación Doble Derecha (RSI - RSI)

Sobre mi código en la implementación de la tarea cuenta con tres clases

1) ALV.java
2) App.java
3) ItemDuplicated.java

Sobre la tercera clase, es una implementación para las excepciones en casos de error.

```java
package Avl;

public class ItemDuplicated extends Exception {
        public ItemDuplicated(String message) {
                super(message);
        }
}
```

La clase ALV es la más importante donde se encuentra todo el código.

```java
public class AVL<E extends Comparable<E>> {
        class Node {
                public E data;
                protected Node left, right;
                protected int bf;
                public Node(E data) {
                        this (data,null,null);
                }
                public Node(E data, Node left, Node right) {
                        this.data = data;
                        this.left = left;
                        this.right = right;
                        this.bf = 0;
                }
```

```
        public void setLeft(Node nodeL) {
                this.left = nodeL;
        }
        public void setRight(Node nodeR) {
                this.right = nodeR;
        }
        public void setData(E dat) {
                this.data = dat;
        }
        public E getData() {
                return this.data;
        }
    }
```

En la primera parte implementamos una clase Node que es una clase interna para mantener una relación entre los nodos. Los parámetros importantes vienen a ser el hijo izquierdo (left), hijo derecho (right), información del nodo (Data) y su factor de equilibrio (bf). Esta clase interna es el código que nos proporcionó la ingeniera. Sobre los cuatro métodos restantes fue implementacion mía que utilizare para el metodo delete.

También se hizo la implementación de métodos que faltaba completar este fue el primer avance que hice además de que en la tarea era lo primero que nos pedía.

```
private Node balanceToRight(Node node){
        Node hijo = node.left;
        switch(hijo.bf) {
        case -1:
                node.bf = 0;
                hijo.bf = 0;
                node = rotateSL(node);
                break;
        case 1:
                Node nieto = hijo.right;
                switch(nieto.bf) {
                        case 1: node.bf = 0;hijo.bf = 1; break;
                        case 0: node.bf = 0; hijo.bf = 0; break;
                        case -1: node.bf = -1; hijo.bf = 0; break;
                }
                nieto.bf =0;

                node.left = rotateSR(hijo);
                node = rotateSL(node);
        }

        return node;
}
```

```
private Node balanceToLeft(Node node){
        Node hijo = node.right;
        switch(hijo.bf) {
        case 1:
                node.bf = 0;
                hijo.bf = 0;
                node = rotateSL(node);
                break;
        case -1:
                Node nieto = hijo.left;
                switch(nieto.bf) {
                case -1: node.bf = 0;hijo.bf = -1; break;
                case 0: node.bf = 0; hijo.bf = 0; break;
                case 1: node.bf = 1; hijo.bf = 0; break;

                }
                nieto.bf =0;

                node.right = rotateSR(hijo);
                node = rotateSL(node);
        }
```

Para el método delete esta fue la implementación que hice el primer void delete es para que haya interacción con la clase main, el método Node delete es donde se hará la operación de eliminar el nodo y también el respectivo balanceo.

```
public void delete(E x)  throws ItemDuplicated {
        this.height = false;
        this.root = delete(this.root, x);
}

public Node delete(Node node, E data) {

        if(node == null) {
                return node;
        }
        int da1 = (int)data;
        int da2 = (int)node.getData();

        if(da1 < da2) {
                node.setLeft(delete(node.left, data));
        }else if(da1 > da2) {
                node.setRight(delete(node.right, data));
        }else {
                if(node.left == null && node.right == null) {
                        System.out.println("removing a leaf node ... ");
                        return null;
                }
```

```
                if(node.left == null) {
                        System.out.println("msm 2");
                        Node tempNode = node.right;
                        node = null;
                        return tempNode;
                } else if(node.right == null) {
                        System.out.println("msm 3");
                        Node tempNode = node.left;
                        node = null;
                        return tempNode;
                }

                System.out.println(" removing ...");
                Node tempNode = getPredecessor(node.left);

                node.setData(tempNode.getData());
                node.setLeft(delete(node.left, tempNode.getData()));
        }

        //node.setHeight(Math.max(height(node.left), height(node.right) +1 ));
        return settleDeletion(node);
}
```

Y por último sobre la clase App.java en esta clase implemente el método main con la resolución de los ejercicios 2 y 4 donde pide implementar cuatro ejemplos de árboles que son alv y hacer una eliminación.

```
public class App {
        public static void main(String [] args) throws ItemDuplicated {
                AVL<Integer> arbol = new AVL<Integer>();
                int [] a = {44, 70, 30, 2 , 72, 26, 81,55};
                insertar(arbol, a);

                AVL<Integer> arbol2 = new AVL<Integer>();
                int [] b = {80, 95, 66, 70, 98, 67};
                insertar(arbol2, b);

                AVL<Integer> arbol3 = new AVL<Integer>();
                int [] c = {80,40,60,120,100,30,130,140,125,150,20,15};
                insertar(arbol3, c);

                arbol3.delete(100);
                arbol3.inOrden();
```

```
                AVL<Integer> arbol4 = new AVL<Integer>();
                int [] d = {14, 1, 6, 17, 8, 9};
                insertar(arbol4, d);
        }

        public static void insertar(AVL<Integer> tree, int [] arr) throws ItemDuplicated {
                for(int i=0; i<arr.length; i++) {
                        tree.insert(arr[i]);
                }
                tree.inOrden();
        }
}
```

Se hizo una implementación de los cuatro ejemplos y también se agregó un método insertar que recibe un AVL y un arreglo en el cual estén todos los elementos del árbol. En este método hay un for que se repetirá hasta que la iteración (i) sea menor que la longitud del arreglo y cada vez llamará al método insertar mandando como parámetro al elemento del arreglo en esa posición y por último llama al método inorden para que nos muestre el árbol de menor a mayor.