

## Informe de Laboratorio 06

### Tema: Arbol AVL

#### INFORMACIÓN BÁSICA

<b>ASIGNATURA:</b>	Estructura de Datos y Algoritmos				
<b>TÍTULO DEL TRABAJO:</b>	Arbol AVL				
<b>NÚMERO DE TRABAJO:</b>	04	<b>AÑO LECTIVO:</b>	2023-A	<b>NRO. SEMESTRE:</b>	III
<b>FECHA DE PRESENTACIÓN:</b>	26/06/23	<b>HORA DE PRESENTACIÓN:</b>	23:59		
<b>INTEGRANTE (s)</b>				<b>NOTA (0-20)</b>	
Hidalgo Chinchay, Paulo Andre Betanzos Rosas, Taylor Anthony Villafuerte Ccapira Frank Alexis					
<b>DOCENTE(s):</b> Mg. Edith Giovanna Cano Mamani					

Tabla 1: Mi tabla extendida

#### INTRODUCCIÓN

Se implementaran los diferentes metodos con los que cuentan los AVL como son obtener el minimo, el maximo, buscar un elemento, eliminar un elemento, entre otros que seran explicados a continuacion.

#### MARCO CONCEPTUAL

Los árboles AVL son un tipo especial de árbol binario de búsqueda que se caracteriza por estar balanceado. Fue inventado por los matemáticos rusos Adelson-Velskii y Landis. En un árbol AVL, todas las claves en su subárbol izquierdo son menores que la clave del nodo y todas las claves en el subárbol derecho son mayores. La diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho, 1.[1,2]

#### SOLUCIONES Y PRUEBAS

Métodos search, getMin, getMax

Para el método search se da como argumento un nodo, el cual sirve para redirigirlo a otro método search privado donde se pasan 2 nodos, con la finalidad de compararlos, se el 1er nodo que pasamos como argumento es mayor retorna la búsqueda del nodo derecho, en caso no sea así, retorna la búsqueda del izquierdo

```

133  public NodeAVL<T> search(NodeAVL<T> nodo){
134      return search(nodo,root);
135  }
136  private NodeAVL<T> search(NodeAVL<T> nodo,NodeAVL<T> otro){
137      if (nodo.getData().compareTo(otro.getData())==0)return otro;
138      else if (nodo.getData().compareTo(otro.getData())>0) {//nodo es mayor
139          return search(nodo, (NodeAVL<T>) otro.getRigth());
140      }else return search(nodo, (NodeAVL<T>) otro.getLeft());
141  }
```

Para obtener el máximo solo se retorna la raíz, ya que es un AVL máximo.

```

130  public NodeAVL<T> getMax(){
131      return this.root;
132  }
```

Para obtener el mínimo se usa otro método privado `getMin` pasando como argumento un nodo, inicialmente la raíz en la cual se valida que la izquierda no sea nula, en caso sea así se retorna el nodo paso por argumento o si no entra recursivamente la función hasta que ya no haya más nodos a la izquierda.

```
123  ✓ public NodeAVL<T> getMin(){
124      return getMin(root);
125  }
126  ✓ private NodeAVL<T> getMin(NodeAVL<T> nodo){
127      if (nodo.getLeft()==null)return nodo;
128      return getMin((NodeAVL<T>) nodo.getLeft());
129  }
```

Para implementar la clase son aprovechamos el método `search` para encontrar el nodo del cual obtendremos sus nodos hijos:

```
public String son(T elemento) throws ExceptionNotFound {
    String hijos="";
    NodeAVL<T> aux=search((NodeAVL<T>) new NodeAVL<T>(elemento));
    if(aux.getLeft()==null&&aux.getRigth()==null) {
        throw new ExceptionNotFound("El nodo no tiene hijos");
    }
    else {
        if(aux.getLeft()!=null) {
            hijos+="Left: "+aux.getLeft().getData()+" ";
        }
        if(aux.getRigth()!=null) {
            hijos+="Rigth: "+aux.getRigth().getData();
        }
    }

    return hijos;
}
```

El método `parent` es un poco mas complicado, puesto que no podemos acceder a un nodo padre desde un nodo hijo, por lo cual será necesario buscar desde la raíz

```

public T parent(T elemento) throws ExceptionNotFound {
    NodeAVL<T> aux=root;
    NodeAVL<T> padre;
    if(isEmpty()) {
        throw new ExceptionNotFound("Arbol vacío");
    }
    else if(elemento.compareTo(root.getData())==0) {
        throw new ExceptionNotFound("Elemento ingresado es la raíz");
    }
    else {
        while(aux!=null) {
            if(aux.getLeft().getData().compareTo(elemento)==0
                || aux.getRight().getData().compareTo(elemento)==0) {
                return aux.getData();
            }
            else {
                if(elemento.compareTo(aux.getData())==-1) {
                    aux=(NodeAVL<T>) aux.getLeft();
                }
                else {
                    aux=(NodeAVL<T>) aux.getRight();
                }
            }
        }
    }
    throw new ExceptionNotFound("No se encontró el elemento");
}

```

### LECCIONES APRENDIDAS Y CONCLUSIONES

Se aprendió a como implementar los AVL al igual que sus metodos, co las clases auxiliares Node y NodeAVL, las cuales permitian tener nodos con derecha e izquierda y hallar sus resectivos padres e hijos.

### REFERENCIAS Y BIBLIOGRAFÍA

[1][https://es.wikipedia.org/wiki/rbol\\_AVL](https://es.wikipedia.org/wiki/rbol_AVL)

[2][https://ccia.ugr.es/~jfv/ed1/c++/cdrom5/informacion/extras/arboles\\_avl.pdf](https://ccia.ugr.es/~jfv/ed1/c++/cdrom5/informacion/extras/arboles_avl.pdf)