Informe de Laboratorio 05

Tema: Arboles AVL

Nota

Estudiante	Escuela	Asignatura
Diego Renato Llerena Tellez Mauricio Zegarra Puma Andre David Delgado Allpan Gonzalo Rail Layme Mamani Andre Hilacondo Begazo @unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Estructura de datos y algoritmos

Laboratorio	Tema	Duración
05	Arboles AVL	02 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 21 Junio 2023	Al 28 Junio 2023

1. Tarea

- Elabore un informe implementando Arboles AVL con toda la lista de operaciones search(),getMin(), getMax(), parent(), son(), insert(), remove().
- INPUT: Una sola palabra en mayúsculas.
- OUTPUT: Se debe contruir el árbol AVL considerando el valor decimal de su código ascii.
- Luego, pruebe todas sus operaciones implementadas.
- Estudie la libreria Graph Stream para obtener una salida gráfica de su implementación.
- Utilice todas las recomendaciones dadas por el docente

2. Equipos, materiales y temas utilizados

- Sistema Operativo (GNU/Linux de preferencia).
- GNU Vim.
- Git.
- Cuenta en GitHub con el correo institucional.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/MauricioZegarra/Laboratorio_05_EDA.git`

4. Actividad realizada

4.1. Clase Nodo y Clase AVL

- Para la realización de este laboratorio se nos pidio implementar los métodos de un Árbol AVL y mostar su funcionalidad. Un árbol AVL es un tipo especial de árbol de búsqueda binario que trata de estar lo más balanceado posible, es decir que la diferencia entre el sub-arbol derecho e izquierdo debe de ser igual a 0 o deber -1 o 1. A continuación se muestra el código generado para la resolución de los ejercicios.
- Primero se crea la clase Nodo que es la base para la implementación de las listas enlazadas.

Listing 1: Creando la Clase Nodo

```
  public class Nodo<E> {

private E data;
private Nodo<E> left;
private Nodo<E> right;
private int bf;

public Nodo(E data, Nodo<E> left, Nodo<E> right) {
    this.data = data;
    this.left = left;
    this.right = right;
    this.bf = 0;
}

public int getBf() {
    return bf;
}

public void setBf(int bf) {
    this.bf = bf;
}

public Nodo(E data) {
    this(data, null, null);
}

public E getData() {
    return this.data;
}

public void setData(E data) {
    this.data = data;
}

public Nodo<E> getLeft() {
```

```
        return this.left;
    }

    public void setLeft(Nodo<E> left) {
        this.left = left;
    }

    public Nodo<E> getRight() {
        return this.right;
    }

    public void setRight(Nodo<E> right) {
        this.right = right;
    }

    public String toString() {
        return this.data.toString() + "(" + this.bf + ")";
    }
}
```

- El código implementa una estructura de datos conocida como Arbol AVL.
- La clase AVL es el contenedor principal que representa el Arbol AVL.
- Está parametrizada con el tipo T, que debe ser un tipo comparable.
- La clase Nodo representa los nodos individuales del Arbol AVL. Cada nodo contiene un dato de tipo T, referencias al nodo izquierdo y derecho, y un campo para almacenar la altura del nodo

Listing 2: Clase AVL: Base

```
public class Avl<E extends Comparable<E>> {
     private Nodo<E> root;
     private boolean height;

 public Avl() {
    this.root = null;
 }

 public boolean isEmpty() {
    return this.root == null;
 }
```

- `public void insert(T input) throws ExceptionNoFound ...` - Inserta un elemento en el árbol AVL.
- `private Node<T>rotateRSR(Node<T>xNode) ...` - Realiza una rotación hacia la derecha en el árbol AVL.
- `private Node<T>rotateRSL(Node<T>yNode) ...` - Realiza una rotación hacia la izquierda en el árbol AVL.

Listing 3: Clase AVL: Metodos de rotación derecha e izquierda

```
private Nodo<E> rotateRSR(Nodo<E> node) {
     Nodo<E> son = node.getLeft();
     node.setLeft(son.getRight());
     son.setRight(node);
     node = son;
     return node;
 }

 private Nodo<E> rotateRSL(Nodo<E> node) {
     Nodo<E> son = node.getRight();
     node.setRight(son.getLeft());
     son.setLeft(node);
     node = son;
     return node;
 }
```

Listing 4: Clase AVL:Metodo de balance a la Derecha

```
private Nodo<E> balanceToRight(Nodo<E> node) {
     Nodo<E> son = node.getLeft();
     if (son.getBf() == -1) {
         node.setBf(0);
         son.setBf(0);
         node = rotateRSR(node);
     } else if (son.getBf() == 1) {
         Nodo<E> gSon = son.getRight();
         switch (gSon.getBf()) {
             case 1:
                 node.setBf(0);
                 son.setBf(1);
                 break;
             case 0:
                 node.setBf(0);
                 son.setBf(0);
                 break;
             case -1:
                 node.setBf(-1);
                 son.setBf(0);
                 break;
         }
         gSon.setBf(0);

         node.setLeft(rotateRSL(son));
         node = rotateRSR(node);
     }
     return node;
 }
```

Listing 5: Clase AVL: Método de balance a la izquierda

```
   private Nodo<E> balanceToLeft(Nodo<E> node) {
     Nodo<E> son = node.getRight();
     if (son.getBf() == 1) {
         node.setBf(0);
         son.setBf(0);
```

```
            node = rotateRSL(node);
        } else if (son.getBf() == -1) {
            Nodo<E> gSon = son.getLeft();
            switch (gSon.getBf()) {
                case -1:
                    node.setBf(0);
                    son.setBf(-1);
                    break;
                case 0:
                    node.setBf(0);
                    son.setBf(0);
                    break;
                case 1:
                    node.setBf(1);
                    son.setBf(0);
                    break;
            }
            gSon.setBf(0);

            node.setRight(rotateRSR(son));
            node = rotateRSL(node);
        }
        return node;
    }
```

Listing 6: Clase AVL: Método Search()

```
      public E search(E x) throws ExceptionNoFound {
        Nodo<E> aux = search(x, this.root);
        if (aux == null) {
            throw new ExceptionNoFound("Elemento no se encuentra en el arbol");
        }
        return aux.getData();
    }

         private Nodo<E> search(E x, Nodo<E> current) throws ExceptionNoFound {
        if (current == null) {
            return null;
        } else {
            int resC = current.getData().compareTo(x);
            if (resC == 0) {
                return current;
            }
            if (resC < 0) {
                return search(x, current.getRight());
            } else {
                return search(x, current.getLeft());
            }
        }
    }
```

Listing 7: Clase AVL: Método getMax()

```
      public E getMax() throws ExceptionNoFound {
        if (isEmpty()) {
```

```
            throw new ExceptionNoFound("El arbol esta vacio");
        }

        Nodo<E> maxNode = getMaxNode(root);
        return maxNode.getData();
    }

    private Nodo<E> getMaxNode(Nodo<E> current) {
        if (current.getRight() == null) {
            return current;
        }
        return getMaxNode(current.getRight());
    }
```

Listing 8: Clase AVL: Método getMin()

```
      public E getMin() throws ExceptionNoFound {
        if (isEmpty()) {
            throw new ExceptionNoFound("El arbol esta vacio");
        }

        Nodo<E> minNode = getMinNode(root);
        return minNode.getData();
    }

    private Nodo<E> getMinNode(Nodo<E> current) {
        if (current.getLeft() == null) {
            return current;
        }
        return getMinNode(current.getLeft());
    }
```

Listing 9: Clase AVL: Método getMin()

```
      public E getMin() throws ExceptionNoFound {
        if (isEmpty()) {
            throw new ExceptionNoFound("El arbol esta vacio");
        }

        Nodo<E> minNode = getMinNode(root);
        return minNode.getData();
    }

    private Nodo<E> getMinNode(Nodo<E> current) {
        if (current.getLeft() == null) {
            return current;
        }
        return getMinNode(current.getLeft());
    }
```

Listing 10: Clase AVL: Método son()

```
    //  son busca si ambos elementos estan presentes y si a es padre de b,
     //de lo contrario, devuelve false.
     public boolean son(E a, E b) throws ExceptionNoFound {
```

```
    return son(a, b, this.root);
}

private boolean son(E a, E b, Nodo<E> current) throws ExceptionNoFound {
    if (current == null) {
        return false;
    }

    boolean aEncontrado = false;
    boolean bEncontrado = false;

    if (current.getData().equals(a)) {
        aEncontrado = true;
    } else if (current.getData().equals(b)) {
        bEncontrado = true;
    }

    if (aEncontrado && bEncontrado) {
        return true;
    }

    boolean hijoEnIzquierda = son(a, b, current.getLeft());
    boolean hijoEnDerecha = son(a, b, current.getRight());

    if (aEncontrado || bEncontrado || hijoEnIzquierda || hijoEnDerecha) {
        return true;
    }

    return false;
}
```

Listing 11: Clase AVL: Método parent()

```
//metodo parent busca el padre mas cercano entre dos nodos
 public E parent(E a, E b) throws ExceptionNoFound {
 return parent(a, b, this.root);
 }

 private E parent(E a, E b, Nodo<E> current) throws ExceptionNoFound {
    if (current == null) {
        throw new ExceptionNoFound("Elemento no se encuentra en el arbol");
    }

    if ((current.getLeft() != null && current.getLeft().getData().equals(a) &&
         current.getRight() != null && current.getRight().getData().equals(b))
            || (current.getRight() != null && current.getRight().getData().equals(a) &&
                current.getLeft() != null && current.getLeft().getData().equals(b))) {
        return current.getData();
    }

    E parentEnIzquierda = parent(a, b, current.getLeft());
    E parentEnDerecha = parent(a, b, current.getRight());

    if (parentEnIzquierda != null) {
        return parentEnIzquierda;
    } else if (parentEnDerecha != null) {
```

```
            return parentEnDerecha;
        }

        return null;
    }
```

4.2. Clase GraphAVLWord.java

- Esta clase se genera graficamente el árbol de busqueda binaria.
- Establece la propiedad del sistema .ºrg.graphstream.ui.[en] "swing"para utilizar la interfaz de usuario de Swing para mostrar el grafo.
- Crea un objeto Graph llamado graph utilizando la implementación SingleGraph.

Listing 12: Clase GraphAVLWord: Main

```
public static void main(String[] args) throws ExceptionNoFound {

    System.setProperty("org.graphstream.ui", "swing");

    Scanner sc = new Scanner(System.in);

    System.out.print("Introduzca la palabra: ");
    String word = sc.next();

    Avl<Character> avl = new Avl<Character>();

    for (int i = 0; i < word.length(); i++) {
        avl.insert(word.charAt(i));
    }

    System.out.println("\nLa cabeza del AVL es: " + avl.getRoot());

    GraphAVLWord<Character> graphPrinter = new GraphAVLWord<Character>(avl);
    graphPrinter.print();
}

public void print() {
    addNodes(this.avl.getRoot(), null);
    addEdges(this.avl.getRoot());

    this.graph.display();
}
```

Listing 13: Clase GraphAVLWord: addNodes()

```
private void addNodes(Nodo<E> node, Node parentNode) {
    if (node != null) {
        Node graphNode = this.graph.addNode(node.getData().toString());
        graphNode.setAttribute("ui.label", node.getData().toString());
        if (parentNode != null) {
            this.graph.addEdge(parentNode.getId() + "_" + graphNode.getId(), parentNode,
                graphNode);
        }
```

```
            addNodes(node.getLeft(), graphNode);
            addNodes(node.getRight(), graphNode);
        }
    }
```

Listing 14: Clase GraphAVLWord: addEdges()

```
private void addEdges(Nodo<E> current) {
        if (current.getLeft() != null) {
            String edgeId = current.getData().toString() + "_" +
                 current.getLeft().getData().toString();
            if (this.graph.getEdge(edgeId) == null) {
                this.graph.addEdge(edgeId, current.getData().toString(),
                     current.getLeft().getData().toString());
            }
            addEdges(current.getLeft());
        }

        if (current.getRight() != null) {
            String edgeId = current.getData().toString() + "_" +
                 current.getRight().getData().toString();
            if (this.graph.getEdge(edgeId) == null) {
                this.graph.addEdge(edgeId, current.getData().toString(),
                     current.getRight().getData().toString());
            }
            addEdges(current.getRight());
        }
    }
```

4.3. Ejecuciones

- Ejecución del árbol con los nombres Diego y Andre

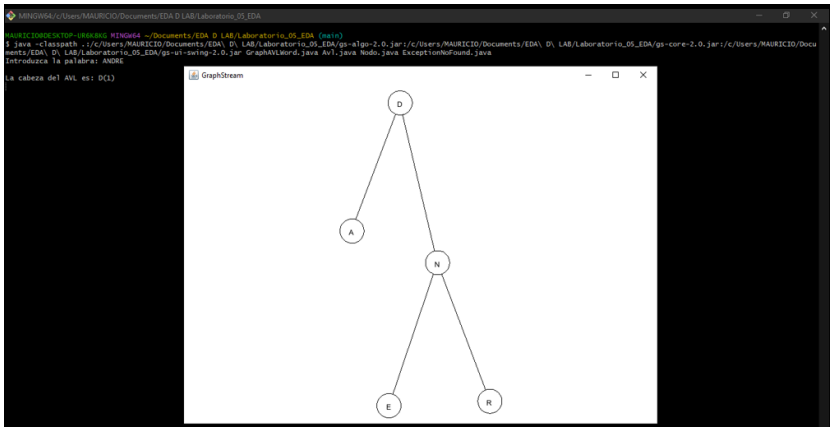

Figura 1: ejecuciones: ejecucion de la clase Aplicacion con .ANDRE"

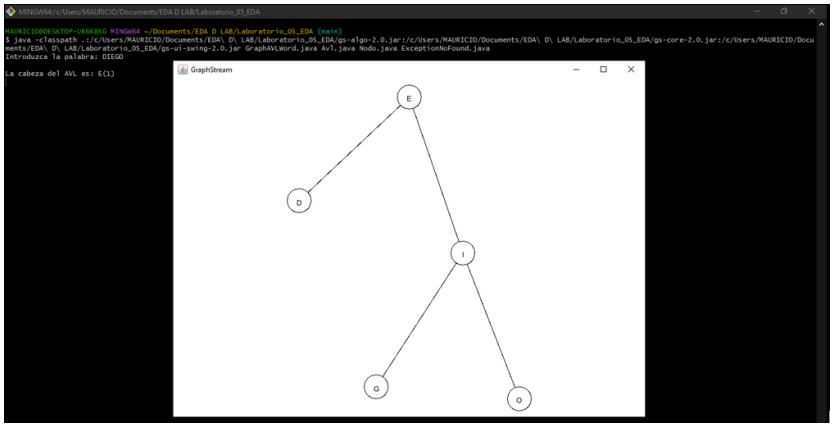

Figura 2: ejecuciones: ejecucion de la clase Aplicacion con "DIEGO"

4.4. Estructura de laboratorio 05

- El contenido que se entrega en este laboratorio es el siguiente:

```
Laboratorio_05_EDA/
|---Avl.java
|---ExceptionNoFound.java
|---GraphAVLWord.java
|---Nodo.java
|---gs-algo-2.0.jar
|---gs-core-2.0.jar
|---gs-ui-swing-2.0.jar
|--- latex
    |--- img
    |   |--- logo_abet.png
    |   |--- logo_episunsa.png
    |   |--- logo_unsa.jpg
    |   |--- ejecucion_word_ANDRE.jpg
    |   |--- ejecucion_word_DIEGO.jpg
    |   |--- Pas.png
    |--- Lab05 Arboles AVL.pdf
    |--- Lab05 Arboles AVL.tex
```

5. Pregunta:

- ¿Explique como es el algoritmo que implementó para obtener el factor de equilibrio de un nodo?
- El factor de equilibrio (bf) se basa en la diferencia de alturas entre subárbol izquierdo y el subárbol derecho del nodo. Este se utiliza para determinar si un nodo está balanceado o si se requieren rotaciones para mantener balanceado el árbol.

  Sobre el algoritmo que implementamos (‘insert‘ y ‘remove‘), primero se realiza un recorrido hasta llegar al nodo afectado (en el bf), después se calcula la diferencia de alturas, esta diferencia se almacena en ‘bf‘ del nodo.

  Ahora, dependiendo del valor se realiza la acción siguiente:

- Si el factor de balance es -1, 0 o 1, el nodo se considera balanceado y no se requieren ajustes. - Si el factor de balance es -2 o 2, indica que el nodo está desbalanceado y se requieren rotaciones y ajustes para restaurar el balance. - En el caso de una inserción, si el factor de balance es 2, se llama al método 'balanceToLeft' para realizar rotaciones y ajustes hacia la izquierda. - En el caso de una eliminación, si el factor de balance es -2, se llama al método 'balanceToRight' para realizar rotaciones y ajustes hacia la derecha.

6. Rúbricas

6.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

6.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	1	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		19	

7. Referencias

- https://www.geeksforgeeks.org/introduction-to-avl-tree/
- https://www.geeksforgeeks.org/insertion-in-an-avl-tree/
- https://www.geeksforgeeks.org/deletion-in-an-avl-tree/
- https://www.javatpoint.com/avl-tree
- https://docs.oracle.com/javase/tutorial/java/generics/types.html
- https://algorithmtutor.com/Data-Structures/Tree/AVL-Trees/