

# Informe de Laboratorio 05

## Tema: Adelson-Velskii y Landis (AVL)

**Nota**

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca, Armando Steven Cuno Cahuari	Escuela Profesional de Ingeniería de Sistemas	Estructura de Datos y Algoritmos (EDA) Semestre: III Código: 1702122

Tarea	Tema	Duración
05	Adelson-Velskii y Landis (AVL)	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 13/06/2024	Al 13/06/2024 - 23:59pm

## 1. Introducción

Los árboles AVL son una estructura de datos de árbol binario de búsqueda auto-balanceado, nombrados en honor a sus inventores, Georgy Adelson-Velsky y Evgenii Landis, quienes los introdujeron en 1962. Estos árboles se diseñaron para mantener un equilibrio dinámico que garantiza que las operaciones de búsqueda, inserción y eliminación puedan realizarse en tiempo logarítmico  $O(\log n)$ . Este equilibrio se logra mediante la restricción de la diferencia de altura (balanceo) entre los subárboles izquierdo y derecho de cualquier nodo a un máximo de uno.

## 2. Objetivos

- Entender los conceptos básicos de los árboles AVL.
- Implementar correctamente las operaciones básicas.
- Mantener el equilibrio del árbol
- Optimizar el tiempo de búsqueda.
- Fomentar buenas prácticas de programación.
- Mejorar la eficiencia en la inserción y eliminación.

### 3. Tarea

Elabore un informe implementando Árboles AVL con toda la lista de operaciones(adapte el código de las diapositivas) :

- search(),
- getMin(),
- getMax(),
- parent(),
- son(),
- insert()

**INPUT:** Una sólo palabra en mayúsculas.

**OUTPUT:** Se debe construir el árbol AVL considerando el valor decimal de su código ascii. Luego, pruebe todas sus operaciones implementadas.

**Pregunta:** ¿Explique como es el algoritmo que implementó para obtener el factor de equilibrio de un nodo?

### 4. Entregables

- Informe hecho en latex.
- Archivos java.
- Responder la pregunta dejada.
- URL: Repositorio GitHub.

### 5. Equipos, materiales y temas utilizados

- AVL
- Java Development Kit
- URL: Repositorio GitHub

### 6. Pregunta

1. **¿Explique como es el algoritmo que implementó para obtener el factor de equilibrio de un nodo?**

El método de inserción en un árbol AVL se encarga de agregar un nuevo nodo con una clave específica y luego equilibrar el árbol si es necesario. El algoritmo sigue estos pasos:

**a) Inserción normal de un BST:**

- Se verifica si el nodo actual existe; si no, se crea un nuevo nodo con la clave proporcionada.
- Luego, se compara la clave con el valor del nodo actual.
- Si la clave es menor, se agrega al nodo izquierdo; si es mayor, al nodo derecho.

- Si se encuentra una clave duplicada, se retorna el árbol sin cambios adicionales.

**b) Actualización de la altura:**

- Después de insertar un nodo, se actualiza la altura del nodo raíz.
- Se utiliza el método max para determinar la mayor altura entre los dos nodos hijos y se le suma 1.

**c) Verificación del balance:**

- Se verifica el balance del nodo mediante el método getBalance para asegurarse de que no esté desequilibrado.

**d) Rotaciones para equilibrar el árbol:**

- Si el balance es mayor que 1 y la clave a agregar es menor que el valor del nodo izquierdo, se realiza una rotación a la derecha (rightRotate).
- Si el balance es menor que -1 y la clave es mayor que el valor del nodo derecho, se realiza una rotación a la izquierda (leftRotate).
- Si el balance es mayor que 1 y la clave es mayor que el valor del nodo izquierdo, se realiza primero una rotación a la izquierda en el nodo izquierdo (leftRotate), seguida de una rotación a la derecha (rightRotate).
- Si el balance es menor que -1 y la clave es menor que el valor del nodo derecho, se realiza primero una rotación a la derecha en el nodo derecho (rightRotate), seguida de una rotación a la izquierda (leftRotate).

Finalmente, se retorna el nodo actual.

## 7. URL de Repositorio Github

- URL del Repositorio GitHub
- [https://github.com/Victor-Gonzalo-Maldonado-Vilca/EDA\\_lab05.git](https://github.com/Victor-Gonzalo-Maldonado-Vilca/EDA_lab05.git)

## 8. Desarrollo del trabajo

### 8.1. Desarrollo de La clase Node

La clase Node representa un nodo en un árbol AVL en Java. Tiene atributos como key para el valor del nodo, height para la altura del nodo, left y right para los nodos hijos izquierdo y derecho respectivamente. El constructor Node(int d) inicializa el nodo con un valor d y establece la altura inicial en 1.

Listing 1: Ejemplo de código Java

```
class Node {  
    int key, height;  
    Node left, right;  
  
    Node(int d) {  
        key = d;  
        height = 1;  
    }  
}
```

## 8.2. Desarrollo de La clase AVLTree

Esta sección describe el desarrollo de la clase AVLTree en el contexto de implementar un árbol AVL en Java. Incluye los métodos fundamentales para el funcionamiento del árbol AVL, como la inserción de nodos, rotaciones para el balanceo, búsqueda de nodos, y obtención de valores máximo y mínimo dentro del árbol.

Listing 2: Ejemplo de código Java

```
class AVLTree {  
    Node root;
```

El árbol AVL nos otorgará los siguiente métodos:

### 8.2.1. Método height:

Este método devolverá la altura de un nodo. Tomará el nodo como argumento y primero verificará si el nodo existe; en caso de que no exista, retornará 0. De lo contrario, devolverá la altura del nodo mediante su atributo 'height'.

Listing 3: Ejemplo de código Java

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

### 8.2.2. Método max:

Esta función devolverá el valor máximo entre dos números. Verificará si 'a' es mayor que 'b'; en ese caso, retornará 'a'. En caso contrario, retornará 'b'.

Listing 4: Ejemplo de código Java

```
int max(int a, int b) {  
    return (a > b) ? a : b;  
}
```

### 8.2.3. Método rightRotate:

Este método se aplicará en caso de que el nodo izquierdo no esté balanceado. Realizará una rotación simple hacia la derecha, intercambiando valores entre tres nodos. Además, actualizaremos las alturas correspondientes de los nodos izquierdo y derecho utilizando el método 'max'. Finalmente, el método retornará el nodo después de haber aplicado la rotación hacia la derecha.

Listing 5: Ejemplo de código Java

```
Node rightRotate(Node y) {  
    Node x = y.left;  
    Node T2 = x.right;  
  
    // Realiza la rotacion  
    x.right = y;  
    y.left = T2;
```

```
// Actualiza las alturas
y.height = max(height(y.left), height(y.right)) + 1;
x.height = max(height(x.left), height(x.right)) + 1;

// Retorna la nueva raiz
return x;
}
```

#### 8.2.4. Método leftRotate:

Este método se aplicará en caso de que el nodo derecho no esté balanceado. Realizará una rotación simple hacia la izquierda, intercambiando los valores de tres nodos. Asimismo, actualizaremos las alturas correspondientes de los nodos izquierdo y derecho utilizando el método 'max', y retornaremos el nodo final después de aplicar la rotación hacia la izquierda.

Listing 6: Ejemplo de código Java

```
Node leftRotate(Node x) {
    Node y = x.right;
    Node T2 = y.left;

    // Realiza la rotacion
    y.left = x;
    x.right = T2;

    // Actualiza las alturas
    x.height = max(height(x.left), height(x.right)) + 1;
    y.height = max(height(y.left), height(y.right)) + 1;

    // Retorna la nueva raiz
    return y;
}
```

#### 8.2.5. Método getBalance:

Este método determinará si se cumple el factor de balance de un nodo dado. Tomará el nodo actual como argumento y primero verificará si el nodo existe; si no, retornará 0. De lo contrario, retornará la diferencia entre la altura del hijo izquierdo y la altura del hijo derecho.

Listing 7: Ejemplo de código Java

```
int getBalance(Node N) {
    if (N == null)
        return 0;
    return height(N.left) - height(N.right);
}
```

#### 8.2.6. Método Insertar:

Este método se encargará de insertar los valores correspondientes y luego balancearlos. Primero, realizará la inserción normal de un BST. Este método tendrá como argumentos el nodo actual y la clave (key) correspondiente a agregarse en un nodo. Primero, verificará si el nodo actual existe; si no, instanciará un nuevo nodo. Luego, verificará el valor del nodo actual: si la clave es menor que el

valor del nodo actual, la clave será agregada al nodo izquierdo; en caso contrario, al nodo derecho. Si encuentra una clave duplicada, simplemente retornará el árbol sin hacer cambios adicionales.

Después de insertar un nodo, actualizaremos la altura del nodo raíz. Utilizaremos el método 'max' para determinar la mayor altura entre los dos nodos hijos y le sumaremos 1. Luego, comprobaremos el balance del nodo para asegurarnos de que no esté desequilibrado mediante el método 'getBalance'. Si el balance es mayor que 1 y la clave a agregar es menor que el valor del nodo izquierdo, realizaremos una rotación a la derecha ('rightRotate'). Si el balance es menor que -1 y la clave es mayor que el valor del nodo derecho, realizaremos una rotación a la izquierda ('leftRotate').

Si el balance es mayor que 1 y la clave es mayor que el valor del nodo izquierdo, primero realizaremos una rotación a la izquierda en el nodo izquierdo ('leftRotate'), seguida de una rotación a la derecha ('rightRotate'). Si el balance es menor que -1 y la clave es menor que el valor del nodo derecho, primero realizaremos una rotación a la derecha en el nodo derecho ('rightRotate'), seguida de una rotación a la izquierda ('leftRotate').

Finalmente, retornaremos el nodo actual.

Listing 8: Ejemplo de código Java

```
Node insert(Node node, int key) {
    /* 1. Realiza la insercion normal de un BST */
    if (node == null)
        return (new Node(key));

    if (key < node.key)
        node.left = insert(node.left, key);
    else if (key > node.key)
        node.right = insert(node.right, key);
    else // Claves duplicadas no son permitidas en un AVL
        return node;

    /* 2. Actualiza la altura de este nodo ancestral */
    node.height = 1 + max(height(node.left), height(node.right));
    /* 3. Obtiene el factor de equilibrio de este nodo ancestral para comprobar
    si se ha desequilibrado */
    int balance = getBalance(node);
    // Si el nodo se ha desequilibrado, entonces hay 4 casos
    // Caso Izquierda-Izquierda
    if (balance > 1 && key < node.left.key)
        return rightRotate(node);
    // Caso Derecha-Derecha
    if (balance < -1 && key > node.right.key)
        return leftRotate(node);
    // Caso Izquierda-Derecha
    if (balance > 1 && key > node.left.key) {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }
    // Caso Derecha-Izquierda
    if (balance < -1 && key < node.right.key) {
        node.right = rightRotate(node.right);
        return leftRotate(node);
    }
    // Retorna el nodo (sin cambios)
    return node;
}

void insert(int key) {
    root = insert(root, key);
}
```

```
}
```

#### 8.2.7. Método minValueNode:

Este algoritmo toma como argumento el nodo actual y recorre, mediante una iteración, los hijos izquierdos hasta encontrar el último nodo que haya sido instanciado con un valor.

Listing 9: Ejemplo de código Java

```
Node minValueNode(Node node) {  
    Node current = node;  
  
    /* Recorre hacia la izquierda para encontrar el nodo hoja mas pequeno */  
    while (current.left != null)  
        current = current.left;  
  
    return current;  
}
```

#### 8.2.8. Método maxValueNode:

Este algoritmo funciona de manera similar al método 'minValueNode', pero en lugar de recorrer los hijos izquierdos, recorre los hijos derechos hasta encontrar el último nodo que haya sido instanciado con un valor.

Listing 10: Ejemplo de código Java

```
Node maxValueNode(Node node) {  
    Node current = node;  
  
    /* Recorre hacia la derecha para encontrar el nodo hoja mas grande */  
    while (current.right != null)  
        current = current.right;  
  
    return current;  
}
```

#### 8.2.9. Método search:

Este método realiza una búsqueda en el árbol AVL para encontrar un nodo con un valor específico. Compara el valor dado con el valor de los nodos y se mueve hacia el subárbol correspondiente según la relación de orden.

Listing 11: Ejemplo de código Java

```
Node search(Node root, int key) {  
    // Base Cases: root is null or key is present at root  
    if (root == null || root.key == key)  
        return root;  
  
    // Key is greater than root's key  
    if (root.key < key)  
        return search(root.right, key);  
}
```

```
// Key is smaller than root's key
return search(root.left, key);
}
```

#### 8.2.10. Método getMin:

Similar al método getMax, este método se utiliza en un árbol AVL para obtener el valor mínimo dentro del árbol. Verifica si el nodo raíz es nulo; si lo es, devuelve Integer.MINVALUE para indicar que el árbol está vacío. En caso contrario, llama al método minValueNode para encontrar el nodo con el valor mínimo en el árbol y devuelve el valor contenido en ese nodo.

Listing 12: Ejemplo de código Java

```
int getMin() {
    if (root == null)
        return Integer.MIN_VALUE;
    Node minNode = minValueNode(root);
    return minNode.key;
}
```

#### 8.2.11. Método getMax:

Este método se utiliza en un árbol AVL para obtener el valor máximo dentro del árbol. Verifica si el nodo raíz es nulo; si lo es, devuelve Integer.MAXVALUE para indicar que el árbol está vacío. En caso contrario, llama al método maxValueNode para encontrar el nodo con el valor máximo en el árbol y devuelve el valor contenido en ese nodo.

Listing 13: Ejemplo de código Java

```
int getMax() {
    if (root == null)
        return Integer.MAX_VALUE;
    Node maxNode = maxValueNode(root);
    return maxNode.key;
}
```

#### 8.2.12. Método parent:

Este método encuentra y devuelve el nodo padre de un nodo con un valor dado en el árbol AVL. Utiliza una función recursiva para buscar el nodo y su padre.

Listing 14: Ejemplo de código Java

```
Node parent(Node node, int key) {
    if (node == null || node.key == key)
        return null;

    // Caso especial cuando el nodo raíz es el padre
    if ((node.left != null && node.left.key == key) || (node.right != null &&
        node.right.key == key))
        return node;

    // Busca en los subárboles izquierdo y derecho
    Node leftSearch = parent(node.left, key);
```



```
Node rightSearch = parent(node.right, key);

return (leftSearch != null) ? leftSearch : rightSearch;
}
```

### 8.2.13. Método son:

Este método encuentra y devuelve los hijos de un nodo con un valor dado en el árbol AVL. Utiliza una función recursiva para buscar los hijos del nodo dado.

Listing 15: Ejemplo de código Java

```
Node son(Node node, int key) {
    if (node == null)
        return null;

    if (node.left != null && node.left.key == key)
        return node.left;
    else if (node.right != null && node.right.key == key)
        return node.right;

    Node leftSon = son(node.left, key);
    Node rightSon = son(node.right, key);

    return (leftSon != null) ? leftSon : rightSon;
}
```

## 8.3. Clase Principal Main

El fragmento de código Java muestra la clase Main con el método main. En este método, se instancia un objeto AVLTree, se insertan valores ASCII que representan las letras de "JAVA" en el árbol, y luego se llevan a cabo operaciones como búsqueda, obtención del mínimo y máximo, así como la identificación del padre e hijo de nodos específicos en el árbol AVL.

Listing 16: Ejemplo de código Java

```
public class Main {
    public static void main(String[] args) {
        AVLTree tree = new AVLTree();

        // Inserta las palabras en mayusculas como sus valores decimales de ASCII
        String inputWord = "JAVA";
        for (char c : inputWord.toCharArray()) {
            tree.insert((int) c);
        }

        // Realiza las operaciones de busqueda, obtener el minimo y maximo, el padre y el hijo
        int searchKey = (int) 'J';
        Node searchResult = tree.search(tree.root, searchKey);
        System.out.println("Search Result: " + searchResult.key);

        int min = tree.getMin();
        System.out.println("Minimum Value: " + min);

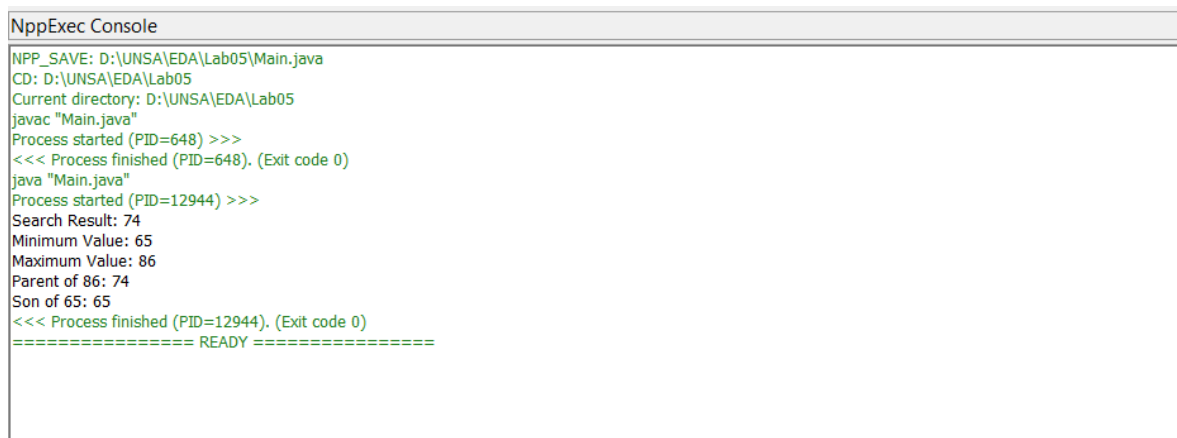
        int max = tree.getMax();
    }
}
```

```
System.out.println("Maximum Value: " + max);

int parentKey = (int) 'V';
Node parentNode = tree.parent(tree.root, parentKey);
if (parentNode != null)
    System.out.println("Parent of " + parentKey + ": " + parentNode.key);
else
    System.out.println("Parent of " + parentKey + " not found.");

int sonKey = (int) 'A';
Node sonNode = tree.son(tree.root, sonKey);
if (sonNode != null)
    System.out.println("Son of " + sonKey + ": " + sonNode.key);
else
    System.out.println("Son of " + sonKey + " not found.");
}
}
```

### 8.3.1. Ejecución



```
NppExec Console
NPP_SAVE: D:\UNSA\EDA\Lab05\Main.java
CD: D:\UNSA\EDA\Lab05
Current directory: D:\UNSA\EDA\Lab05
javac "Main.java"
Process started (PID=648) >>>
<<< Process finished (PID=648). (Exit code 0)
java "Main.java"
Process started (PID=12944) >>>
Search Result: 74
Minimum Value: 65
Maximum Value: 86
Parent of 86: 74
Son of 65: 65
<<< Process finished (PID=12944). (Exit code 0)
===== READY =====
```

Figura 1: Ejecución

## 9. Recomendaciones

- Se recomienda utilizar la clase AVLTree en aplicaciones donde se requiera un almacenamiento eficiente de datos que necesiten operaciones de inserción, búsqueda y eliminación frecuentes.
- Es importante entender y considerar los principios del balanceo en árboles AVL al realizar modificaciones en el código o adaptarlo a necesidades específicas, para garantizar que el árbol se mantenga balanceado en todo momento.
- Se sugiere realizar pruebas exhaustivas con diferentes conjuntos de datos y situaciones límite para evaluar el rendimiento y la robustez de la implementación del árbol AVL.
- Para un mejor seguimiento y comprensión del funcionamiento del árbol AVL, se recomienda utilizar herramientas de visualización que permitan observar el proceso de balanceo y la estructura del árbol en diferentes etapas.

- Mantener una buena documentación del código, incluyendo comentarios claros y explicativos, facilitará el mantenimiento y la colaboración en proyectos que utilicen la clase AVLTree.

## 10. Conclusiones

- La utilización de rotaciones simples y dobles permitió mantener el equilibrio del árbol AVL durante las operaciones de inserción y eliminación, garantizando así su eficiencia en términos de tiempo.
- La función de búsqueda implementada en el árbol AVL demostró su capacidad para encontrar nodos con rapidez, incluso en árboles de gran tamaño, gracias al balanceo automático que ofrece esta estructura.
- La visualización y comprensión de los conceptos relacionados con árboles AVL se vieron facilitadas mediante la representación gráfica de estos árboles, lo que ayudó en el proceso de desarrollo y depuración del código.
- En general, la experiencia de trabajar con árboles AVL ha sido educativa y valiosa para comprender en profundidad la importancia de las estructuras de datos balanceadas en la programación.

### 10.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple 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 los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: 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 2: Rúbrica para contenido del Informe y demostración

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

## 11. Referencias

- <https://www.w3schools.com/java/>
- <https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-ide-enterprise-java-and-we>
- <https://docs.oracle.com/javase/7/docs/api/java/util/List.html>
- <https://docs.oracle.com/javase/tutorial/java/generics/types.html>
- <https://www.baeldung.com/java-avl-trees>