UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias


# Estructuras de Datos

*Práctica 13: Árboles AVL*

Profesora:
Amparo López Gaona
Ayudante: Adrián Aguilera Moreno
Ayudante de Laboratorio: Kevin Jair Torres Valencia

# Objetivos

- Comprender e implementar la estructura de datos Árbol AVL.
- Implementar dos tipos de recorrido: PostOrden y PreOrden.
- Calcular la altura de un árbol y utilizar esta información para el balanceo.
- Aplicar algoritmos de rotación para mantener el balanceo del árbol.

# Introducción

Un árbol AVL es un tipo de árbol binario de búsqueda autobalanceado, donde la diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo no puede ser mayor que 1. Este equilibrio garantiza que las operaciones de búsqueda, inserción y eliminación se realicen en tiempo $O(\log n)$, incluso en el peor de los casos.

El nombre *AVL* proviene de sus creadores, Adelson-Velsky y Landis, quienes introdujeron esta estructura en 1962. La característica principal que distingue a un árbol AVL de un árbol binario de búsqueda común es su capacidad para autobalancearse mediante rotaciones cuando se realizan operaciones de inserción o eliminación.

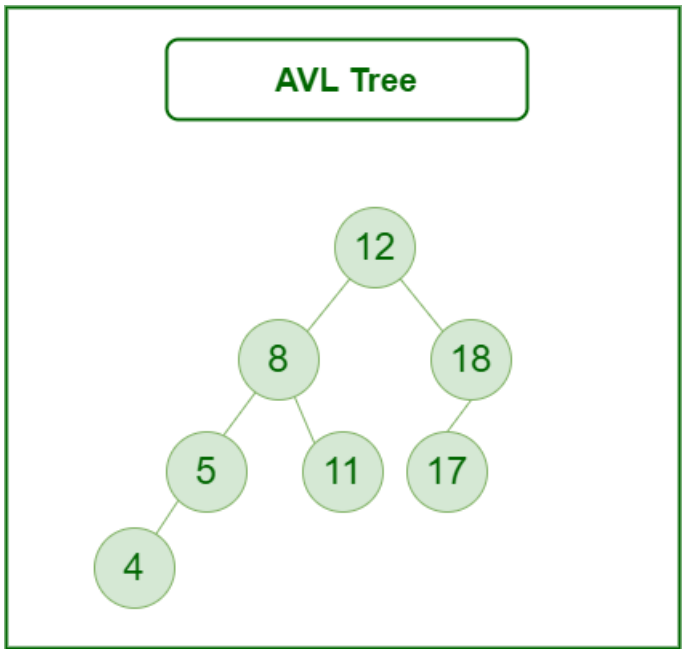


Figure 1: Representación de un Árbol AVL

## Propiedades de un árbol AVL

1. Propiedad de árbol binario de búsqueda: Para cada nodo, todos los elementos en su subárbol izquierdo son menores que él, y todos los elementos en su subárbol derecho son mayores.

2. Propiedad de balanceo: Para cada nodo, la diferencia de altura entre sus subárboles izquierdo y derecho (conocida como factor de balanceo) debe ser -1, 0 o 1.

## Rotaciones en árboles AVL

Para mantener la propiedad de balanceo, se utilizan rotaciones cuando un nodo se vuelve desbalanceado:

- Rotación simple a la derecha: Se utiliza cuando el subárbol izquierdo de un nodo es más alto que el derecho, y el desbalance está en el subárbol izquierdo del hijo izquierdo.

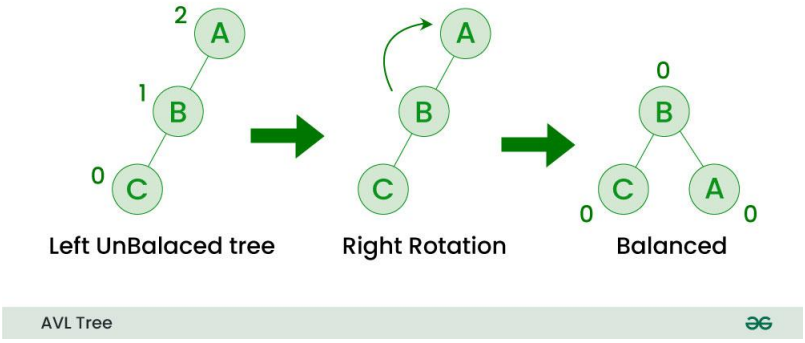


Figure 2: Rotacion simple a la derecha

- Rotación simple a la izquierda: Se utiliza cuando el subárbol derecho de un nodo es más alto que el izquierdo, y el desbalance está en el subárbol derecho del hijo derecho.

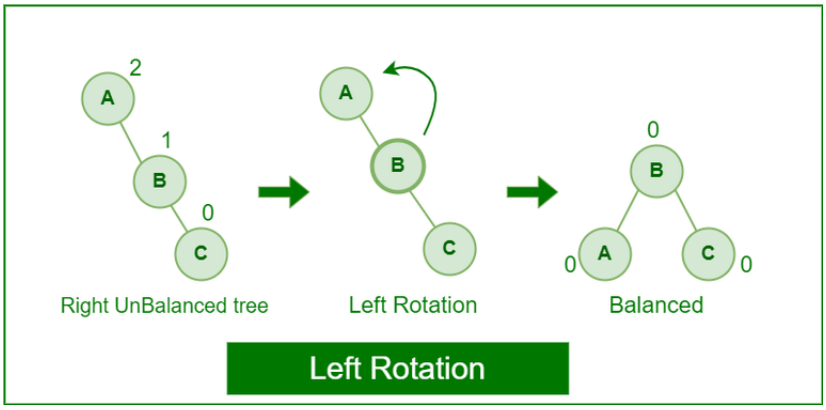


Figure 3: Rotacion simple a la izquierda

- Rotación doble izquierda-derecha: Se utiliza cuando el subárbol izquierdo de un nodo es más alto que el derecho, pero el desbalance está en el subárbol derecho del hijo izquierdo.

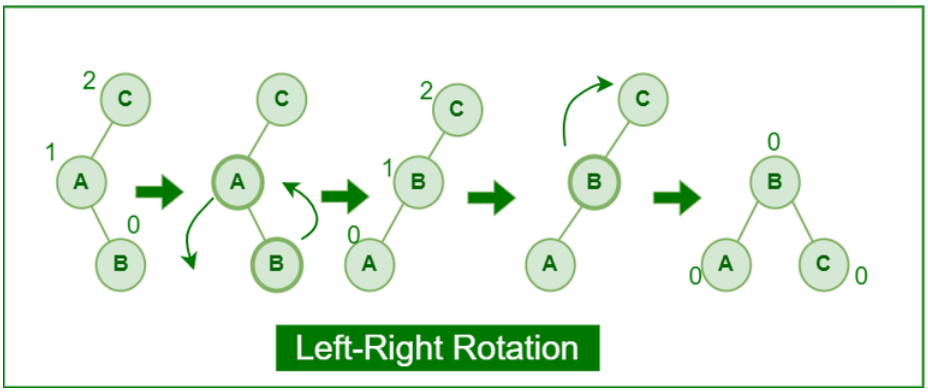


Figure 4: Rotación doble izquierda-derecha

- Rotación doble derecha-izquierda: Se utiliza cuando el subárbol derecho de un nodo es más alto que el izquierdo, pero el desbalance está en el subárbol izquierdo del hijo derecho.

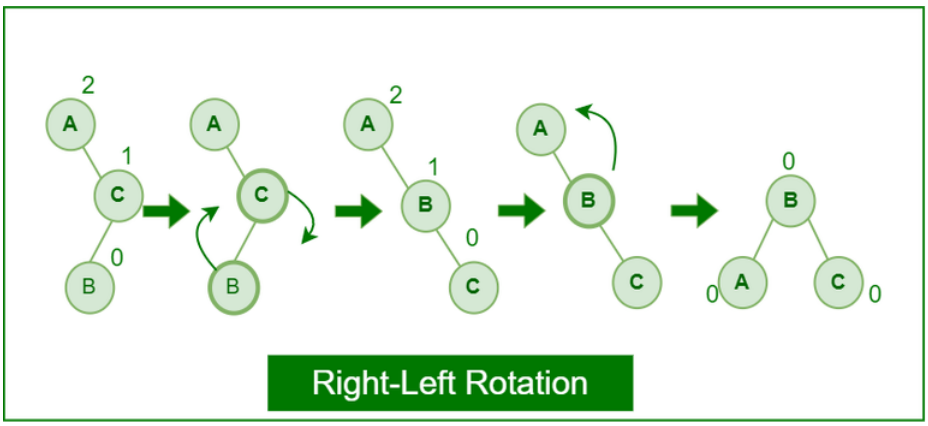


Figure 5: Rotación doble derecha-izquierda

## Iterador

Recordemos que un `Iterator` proporciona una forma de acceder secuencialmente a los elementos de una colección sin exponer su estructura interna. Para nuestra implementación el iterador nos permite recorrer los nodos del árbol de diferentes maneras (PreOrden, PostOrden) sin modificar la implementación del árbol.

- Recorrido Pre-Orden (raíz-izquierda-derecha): Se visita primero el nodo actual, luego su subárbol izquierdo y finalmente su subárbol derecho.
- Recorrido Post-Orden (izquierda-derecha-raíz): Se visita primero el subárbol izquierdo, luego el subárbol derecho y finalmente el nodo actual.

### Nodo

La clase Nodo representa los nodos del árbol. Cada nodo almacena

- Un dato de tipo `T`
- Una referencia al hijo izquierdo.
- Una referencia al hijo derecho.
- La altura del árbol.

## Desarrollo

En esta práctica, deberán completar la implementación de un Árbol AVL genérico, siguiendo el principio de separación de interfaz e implementación. Se te proporcionarán algunos archivos base que deberás extender.

Para ello tendrán que implementar los métodos en las siguientes clases:

- ArbolAVLImpl:
  - Debe implementar todos los métodos de la interfaz `ArbolAVL`, es decir, las operaciones de inserción, búsqueda y eliminación.
  - Para el caso de la eliminación, considerar los tres casos (nodo sin hijos, con un hijo, con dos hijos) y las rotaciones necesarias si el nodo está desbalanceado.
- IteradorPreOrden:
  - Deben usar una pila para implementar el recorrido preorden.
  - Implementar los métodos de `hasNext` y `next`, de acuerdo a lo anterior.
- IteradorPostOrden
  - Deben usar una pila para implementar el recorrido postorden.
  - Implementar los métodos de `hasNext` y `next`, de acuerdo a lo anterior.

### Archivos proporcionados

- ArbolAVL: Interfaz que define el comportamiento de un Árbol AVL.
- Nodo: Representa un nodo del Árbol AVL.
- ArbolAVLImpl: Implementación de la interfaz de un Árbol AVL.
- Iterador: Interfaz para el Iterador

- IteradorPostOrden: Implementación de un iterador para recorrer un Árbol AVL en orden postorden.
- IteradorPreOrden: Implementación de un iterador para recorrer un Árbol AVL en orden preorden.
- Main: Clase para probar la implementación de la estructura.

# Formato de Entrega

1. Las prácticas se entregarán en parejas.
2. NO incluir los archivos .class dentro de la carpeta.
3. Los archivos de código fuente deben estar documentados.
4. Se pueden discutir y resolver dudas entre los integrantes del grupo. Pero cualquier práctica plagiada total o parcialmente será penalizada con cero para los involucrados.
5. La práctica se debe subir al Github Classroom correspondiente.
6. La entrega en classroom debe contener el link HTTPS y SSH de su repositorio y es lo único que se debe entregar.
7. El horario y día de entrega se acordará en la clase de laboratorio y no deberá sobrepasar 2 clases de laboratorio.