Estructura de Datos - Árboles

**Andres Iván González Guerrero - 1077294332**

17 de mayo de 2025

# ****1. Introducción****

Los árboles binarios son estructuras de datos fundamentales en computación, esenciales para organizar información de manera jerárquica y eficiente. Este informe unificado combina el análisis de dos recursos clave:

1. **El artículo de Oscar Blancarte** sobre conceptos teóricos y tipos de árboles (ABB, AVL).
2. **La guía práctica de Hackers de la Programación**, que aborda la implementación en Java de ABB.

**Objetivo**:

* Proporcionar una visión integral que va desde la **teoría** (definiciones, propiedades, complejidades) hasta la **práctica** (código Java, ejemplos ejecutables).
* Destacar aplicaciones reales y oportunidades de mejora en las implementaciones existentes.

**Alcance**:

* ABB y AVL.
* Operaciones básicas (inserción, búsqueda, recorridos).
* Análisis comparativo de recursos educativos.

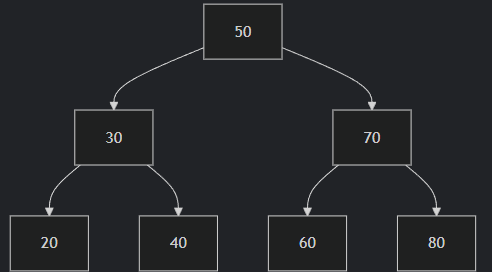
# 2. ****Conceptos Básicos****

**Definición formal**:  
Un árbol es un conjunto de **nodos** conectados por **aristas**, donde:

* **Raíz**: Nodo superior sin padre.
* **Hojas**: Nodos sin hijos.
* **Altura**: Número de niveles desde la raíz hasta la hoja más profunda.

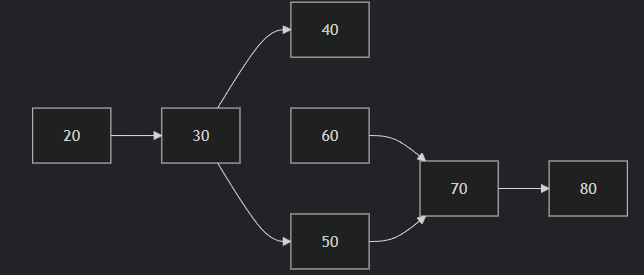
# 3. ****Tipos de Árboles****

* **Árbol binario**:
  + Estructura y propiedades (ej: máximo 2 hijos por nodo).



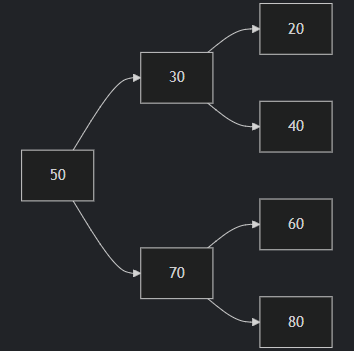
* **Árbol binario de búsqueda (ABB)**:

#### **In-Order (Izq → Raíz → Der)**



**Secuencia**: 50 → 30 → 20 → 40 → 70 → 60 → 80

#### **Pre-Order (Raíz → Izq → Der)**



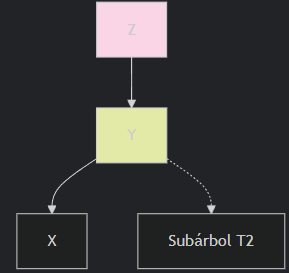
**Secuencia**: 50 → 30 → 20 → 40 → 70 → 60 → 80

# Codigo:

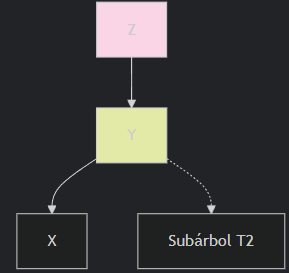
# 

# 

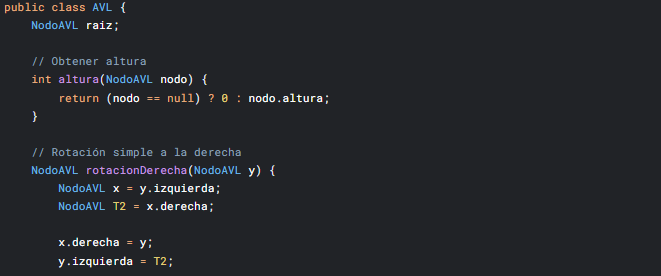
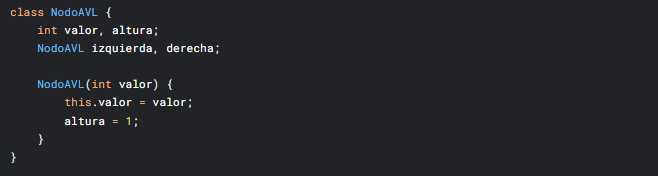
* **Árboles balanceados** (AVL, B-tree):
  + Importancia del balanceo para eficiencia.

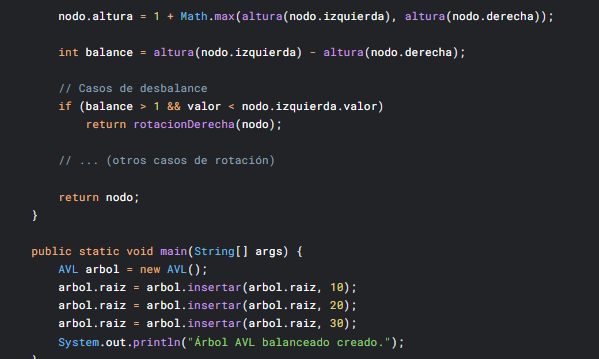
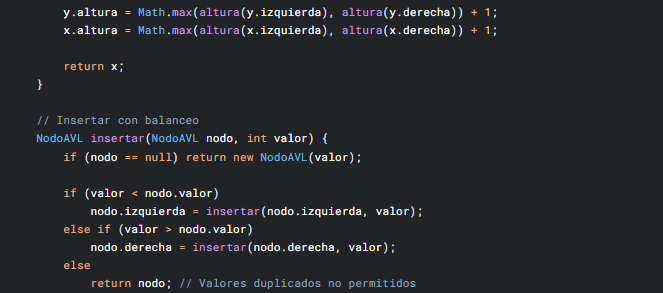


**Después de rotar**:



# Codigo:





# 4. ****Operaciones Principales****

#### **Inserción en Árbol Binario de Búsqueda (ABB)**

La inserción en un ABB sigue la propiedad fundamental:

* **Valores menores** que el nodo actual se colocan en el **subárbol izquierdo**.
* **Valores mayores** van al **subárbol derecho**.

**Algoritmo paso a paso**:

1. Si el árbol está vacío, el nuevo nodo se convierte en la raíz.
2. Compara el valor a insertar con el nodo actual.
3. Si es menor, avanza al hijo izquierdo; si es mayor, al derecho.
4. Repite hasta encontrar una posición vacía.

**Ejemplo en C++**:

# 5. ****Aplicaciones Prácticas****

* Ejemplos mencionados en el artículo:
  + Índices de bases de datos, sistemas jerárquicos (ej: DOM en HTML).

# 6. ****Código de Ejemplo****

* Lenguaje usado (ej: Java, C#, Python).
* Fragmento relevante (adaptable a tu lenguaje preferido):

java

## ****Conclusiones y Recomendaciones****

### ****4.1 Conclusiones****

* **Complementariedad**: Mientras el artículo de Oscar Blancarte profundiza en la teoría y el balanceo (AVL), la guía de Hackers ofrece una **implementación clara en Java** para ABB.
* **Brechas identificadas**:
  + Falta de cobertura sobre **eliminación** en ambos recursos.
  + El artículo práctico no aborda optimizaciones para casos desbalanceados.

### ****4.2 Recomendaciones****

1. **Para principiantes**:
   * Comenzar con la implementación de Hackers para entender ABB.
   * Usar los diagramas y explicaciones de Blancarte para reforzar conceptos.
2. **Para avanzados**:
   * Extender el código con eliminación y balanceo AVL.
   * Realizar pruebas de rendimiento para comparar ABB vs AVL.

### ****4.3 Trabajo Futuro****

* Implementar un **árbol rojo-negro** para comparar rendimiento.
* Desarrollar un visualizador interactivo de operaciones.