Segundo Estudio de Caso

Curso: Estructuras de Datos

Código: SOFT-10

Periodo: 2025-C3

Estudiante: Tatiana Solis Quesada

Profesor: Romario Salas Cerdas

Fecha de entrega: 30 de noviembre del 2025

Introducción

Los árboles AVL forman parte de las estructuras de datos más importantes dentro de la informática, ya que permiten mantener la información ordenada y balanceada sin importar cómo se ingresen los valores. A diferencia de un árbol binario de búsqueda tradicional, donde una mala secuencia de inserciones puede ocasionar que el árbol se “incline” hacia un lado y pierda eficiencia, los árboles AVL utilizan el cálculo del factor de balance y una serie de rotaciones para evitar este problema.

En este estudio se analizan los conceptos fundamentales que garantizan el comportamiento equilibrado de un árbol AVL, explicando cómo se mide el balance de cada nodo, cómo se determina si existe un desbalance y cuáles son los algoritmos utilizados para corregirlo. Además, se incluyen representaciones gráficas que ilustran paso a paso estos procesos junto con una descripción de aplicaciones reales donde este tipo de estructura resulta indispensable por su desempeño y estabilidad.

¿Qué es el factor de balance?

El **factor de balance (FB)** es un valor numérico que indica si un nodo se encuentra equilibrado. Se obtiene mediante:

FB = altura del subárbol izquierdo – altura del subárbol derecho

Este valor siempre debe mantenerse dentro del rango:

$$-1 \leq FB \leq 1$$

Si un nodo presenta un factor fuera de ese intervalo, se considera desbalanceado y requiere una rotación para corregirse.

Cálculo de la altura de un nodo

La **altura** de un nodo es la distancia desde ese nodo hasta su hoja más profunda. Se determina así:

- *Un nodo sin hijos → altura = 1*
- *Un nodo interno →*

altura = 1 + max(altura del hijo izquierdo, altura del hijo derecho)

Después de cada inserción o eliminación, es obligatorio recalcular las alturas desde abajo hacia arriba.

Ejemplo gráfico del factor de balance

Vamos a insertar los valores **50, 40 y 30** en ese orden.

Paso 1: Insertamos 50

Alturas:

h(50) = 1

FB(50) = 0

El árbol está balanceado.

Paso 2: Insertamos 40

50

/

40

Alturas:

h(40) = 1

h(50) = 2

FB(50) = 1

FB(40) = 0

Todavía está balanceado.

Paso 3: Insertamos 30

50

/

40

/

30

Alturas:

h(30) = 1

h(40) = 2

h(50) = 3

Factores:

FB(30) = 0

FB(40) = 1

FB(50) = 2 → DESBALANCE LL

Es un caso Izquierda–Izquierda (LL)

Se corrige con rotación simple a la derecha sobre el nodo 50.

Rotaciones en un árbol AVL

Las rotaciones son operaciones que reestructuran el árbol para devolverle su balance.

Existen cuatro tipos principales:

1. Rotación simple a la derecha (LL)

2. Rotación simple a la izquierda (RR)
3. Rotación doble izquierda–derecha (LR)
4. Rotación doble derecha–izquierda (RL)

Cada una responde a un patrón específico de desbalance.

Rotación simple a la derecha (LL)

Se usa cuando el desbalance está en el hijo izquierdo del hijo izquierdo.

Ejemplo visual:

Antes:

Después:

Rotación simple a la izquierda (RR)

Se utiliza cuando el hijo derecho del nodo también tiene un subárbol derecho más alto.

Antes:

Después:

Y

**/ **

X Z

Rotación doble izquierda–derecha (LR)

Ocurre cuando el desbalance está en el subárbol derecho del hijo izquierdo.

Es equivalente a:

1. Rotación simple a la izquierda en el hijo izquierdo
2. Rotación simple a la derecha en el nodo desbalanceado

Antes:

X

/

Y

****

Z

Después:

Z

**/ **

Y X

Rotación doble derecha–izquierda (RL)

SucedE cuando el desbalance está en el subárbol izquierdo del hijo derecho

Es equivalente a:

1. Rotación simple a la derecha en el hijo derecho
2. Rotación simple a la izquierda en el nodo desbalanceado

Antes:

Después:

Aplicaciones de los árboles AVL (versión nueva)

Los árboles AVL se utilizan en entornos donde la velocidad de acceso a los datos debe mantenerse constante incluso cuando la cantidad de información aumenta o cuando las inserciones son aleatorias. Algunas de sus aplicaciones más comunes son:

1. Bases de datos

Los índices requieren estructuras estables que mantengan búsquedas rápidas.

Los AVL evitan el peor caso de un ABB y mantienen tiempos de acceso logarítmicos.

Búsquedas eficientes

No dependen del orden de inserción

2. Compiladores

Los compiladores administran símbolos, variables, clases y funciones.

Los AVL son útiles porque:

Mantienen tiempos constantes durante el análisis sintáctico

Evitan la degradación del rendimiento al manejar cientos de elementos

3. Sistemas de archivos

Muchos sistemas requieren encontrar y organizar archivos con rapidez.

AVL garantiza que el acceso sea uniforme

Ideal para búsquedas por nombre, hash o metadatos

4. Motores de videojuegos

En juegos 2D/3D se maneja:

- entidades
- colisiones
- activación de objetos

Los AVL facilitan actualizar información en tiempo real sin perder rendimiento.

Conclusiones

Los árboles AVL representan una estructura eficiente y confiable para trabajar con información que debe mantenerse ordenada y accesible rápidamente. Su algoritmo de balanceo automático evita que el rendimiento se degrade, a diferencia de otras estructuras más simples como los ABB tradicionales.

El análisis realizado permite comprender cómo se calcula el factor de balance, cómo se identifican los desbalances y de qué forma las rotaciones restauran la estabilidad del árbol. Así mismo, se evidenció que los AVL tienen aplicaciones en áreas críticas como bases de datos, compiladores, sistemas de archivos y videojuegos, demostrando su versatilidad y relevancia práctica.

Esta investigación y su implementación en Java facilitan no solo la comprensión teórica de los árboles AVL, sino también el desarrollo de habilidades en programación de estructuras avanzadas.