Segundo Estudio de Caso.

Implementación de Algoritmos de Evaluación del Factor de Balance y Rotaciones en árboles AVL

Dylan Josue Chaves Hernández

Universidad CENFOTEC

Estructuras de Datos

Romario Salas Cerdas

Fecha: 11, 2025

Resumen Ejecutivo

Este trabajo investiga en profundidad los algoritmos encargados de evaluar el factor de balance en nodos de árboles AVL y los algoritmos de rotación que corrigen los desbalances detectados. Se presenta la teoría necesaria (definición de altura, factor de balance y condiciones de desbalance), se detallan y representan paso a paso las cuatro rotaciones básicas (rotarIzquierda, rotarDerecha, rotarIzquierdaDerecha y rotarDerechaIzquierda), se expone la lógica general de inserción y eliminación modificadas para mantener las alturas y se enlistan las principales aplicaciones donde los árboles AVL resultan la estructura adecuada. El documento incluye diagramas explicativos y pseudocódigo para las operaciones críticas.

Tabla de Contenido

Introducción

Las estructuras de datos jerárquicas, específicamente los árboles binarios de búsqueda (BST), son fundamentales en las ciencias de la computación para el almacenamiento y recuperación eficiente de información. Sin embargo, los BST tradicionales sufren de una vulnerabilidad crítica: su rendimiento está intrínsecamente ligado al orden de inserción de los datos. En el peor de los casos, cuando los datos ingresan de forma ordenada (ascendente o descendente), el árbol se degenera en una lista enlazada, degradando la complejidad temporal de las operaciones de búsqueda, inserción y eliminación de un óptimo $O(log\ n)$ a un ineficiente $O(n)$.

Para mitigar este problema, en 1962, los matemáticos soviéticos Adelson-Velsky y Landis propusieron los árboles AVL, la primera estructura de datos de árbol binario de búsqueda autobalanceada. La premisa central de un árbol AVL es mantener la altura del árbol lo más compacta posible mediante la propiedad de altura: para cualquier nodo en el árbol, la diferencia de alturas entre su subárbol izquierdo y su subárbol derecho no puede ser superior a uno. Este trabajo de investigación documenta los algoritmos críticos que permiten esta propiedad: el cálculo del factor de balance y las operaciones de rotación (simples y dobles) necesarias para corregir desequilibrios.

Desarrollo

Algoritmos de Evaluación del Factor de Balance

El núcleo del mecanismo de autobalanceo reside en la evaluación constante del estado de cada nodo tras una operación de modificación (inserción o eliminación). Esta evaluación se cuantifica mediante una métrica conocida como el Factor de Balance (FB).

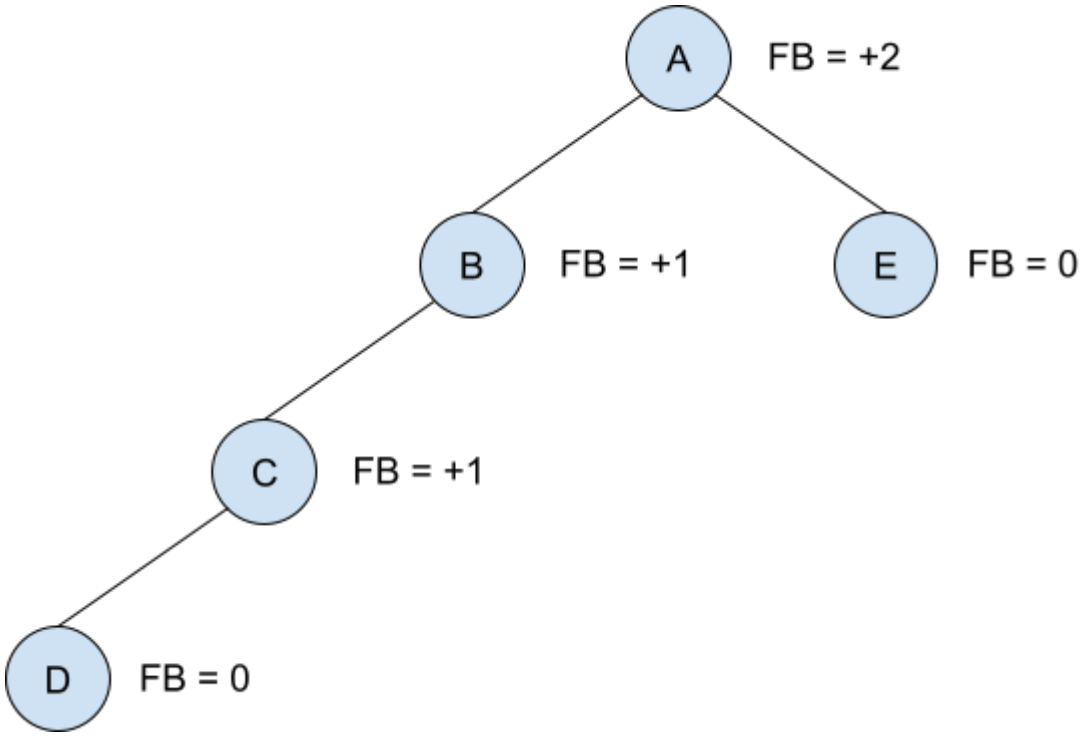
El algoritmo para calcular el factor de balance de un nodo N se define formalmente como la diferencia entre la altura de su subárbol izquierdo y la altura de su subárbol derecho. Matemáticamente, se expresa como:

$$FB(N) = Altura(N_{izquierda}) - Altura(N_{derecha})$$

Donde la altura de un nodo nulo se considera -1 (o 0, dependiendo de la convención, pero manteniendo la consistencia). Para que un nodo se considere "balanceado", su FB debe pertenecer al conjunto $\{-1, 0, 1\}$.

- $FB = 0$: El nodo está perfectamente equilibrado (ambos subárboles tienen la misma altura).
- $FB = 1$: El subárbol izquierdo es un nivel más alto que el derecho (cargado a la izquierda).
- $FB = -1$: El subárbol derecho es un nivel más alto que el izquierdo (cargado a la derecha).

Si tras una operación, el cálculo resulta en $|FB| \geq 2$, el algoritmo detecta un desbalance que requiere intervención inmediata. La implementación eficiente de este algoritmo implica que cada nodo debe almacenar su altura actual o calcular recursivamente, actualizándose desde las hojas hacia la raíz (bottom-up) tras cada inserción.



Algoritmos de Rotación para la Corrección de Desbalances

Una vez detectado un desbalance (cuando FB es 2 o -2), se utilizan operaciones locales llamadas rotaciones para restaurar la propiedad AVL sin violar la propiedad de orden del árbol binario de búsqueda. Existen cuatro casos de desbalance que se resuelven con cuatro algoritmos de rotación específicos.

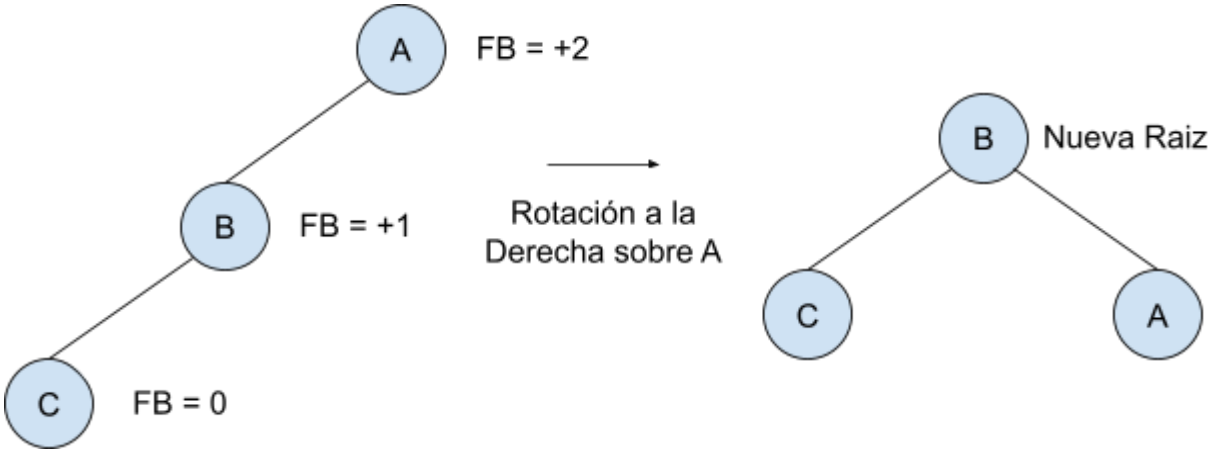
Rotación Simple a la Derecha (Caso Izquierda-Izquierda)

Este algoritmo se aplica cuando el desbalance proviene del subárbol izquierdo del hijo izquierdo de un nodo (el nodo raíz tiene $FB = +2$ y su hijo izquierdo tiene $FB = +1$ o 0).

Imaginemos un nodo raíz A desbalanceado hacia la izquierda, con un hijo izquierdo B.

1. El nodo B asciende para convertirse en la nueva raíz del subárbol.
2. El nodo A desciende para convertirse en el hijo derecho de B.
3. Si B tenía un hijo derecho (digamos, subárbol $T2$), este se convierte en el hijo izquierdo de A.

Esta operación reduce la altura del lado izquierdo y aumenta la del derecho, equilibrando el árbol.

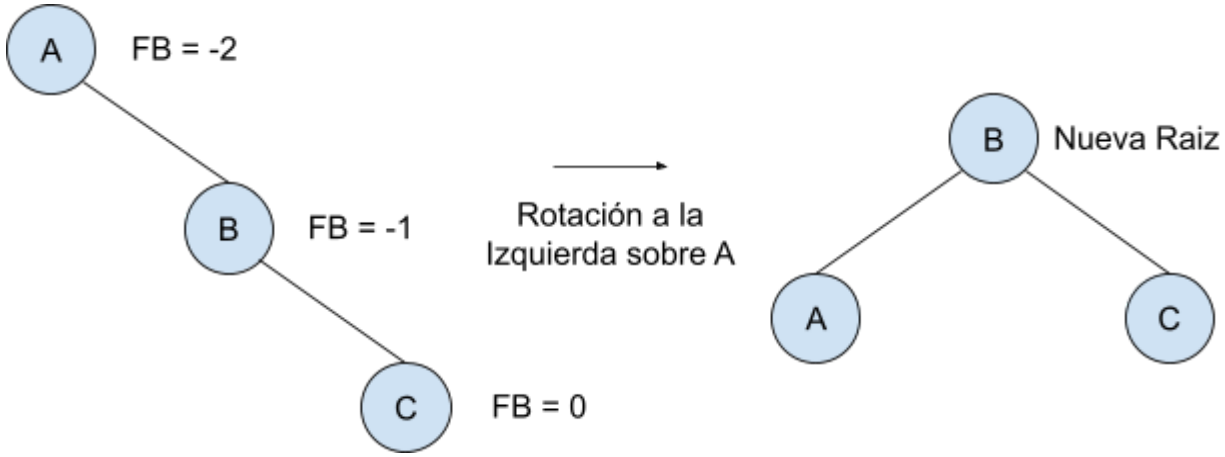


Rotación Simple a la Izquierda (Caso Derecha-Derecha)

Es el caso simétrico al anterior (el nodo raíz tiene $FB = -2$ y su hijo derecho tiene $FB = -1$ o 0). Se utiliza cuando el desbalance está en el subárbol derecho del hijo derecho.

Dado un nodo A desbalanceado hacia la derecha con un hijo derecho B:

1. B asciende para ser la nueva raíz.
2. A se convierte en el hijo izquierdo de B.
3. El subárbol izquierdo original de B pasa a ser el hijo derecho de A.

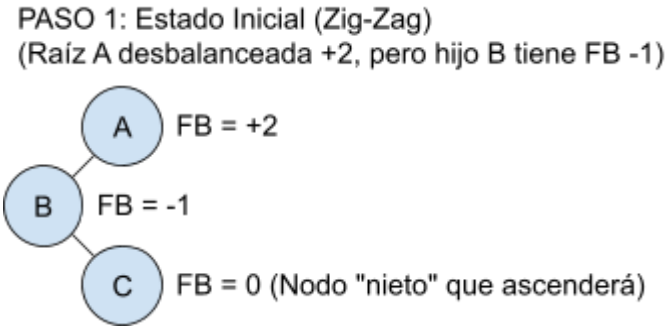


Rotación Doble Derecha-Izquierda (Caso Derecha-Izquierda)

Es el simétrico del caso anterior. Ocurre cuando la raíz A tiene $FB = -2$ y su hijo derecho B tiene $FB = +1$.

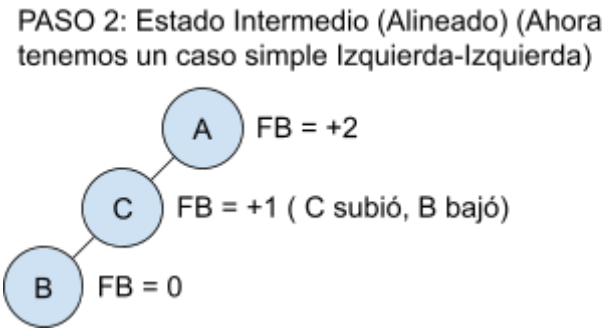
1. Se aplica una *Rotación a la Derecha* sobre el hijo derecho (B). Esto alinea los nodos en una estructura "Derecha-Derecha".
2. Se aplica una *Rotación a la Izquierda* sobre la raíz (A).

Al igual que en el caso anterior, el "nieto" asciende dos niveles para convertirse en la nueva raíz, asegurando que la altura global disminuya y el equilibrio se restaure.



Aplicaciones de los Árboles AVL

La estricta política de balanceo de los árboles AVL los convierte en estructuras de datos ideales para escenarios donde la velocidad de búsqueda es



prioritaria frente a la velocidad de actualización. A continuación, se detallan sus principales aplicaciones

Bases de Datos y Sistemas de Indexación

Muchos sistemas de bases de datos utilizan variantes de árboles balanceados para mantener índices de registros (aunque a menudo se usan Árboles B, los AVL son aplicables en índices en memoria).

En una base de datos con millones de registros (ej. cédulas de identidad), una búsqueda lineal es inaceptable. Un árbol AVL garantiza que buscar un registro entre mil millones de datos tomará aproximadamente 30 comparaciones ($log_2 10^9 \approx 30$), independientemente de si los datos se insertaron ordenadamente.

Conjuntos (Sets) y Diccionarios (Maps) en Librerías de Software

Implementación de estructuras de datos abstractas que requieren unicidad de elementos y búsquedas rápidas, como los std::set en C++ o diccionarios ordenados.

Estas estructuras requieren operaciones de búsqueda (contains o lookup) extremadamente frecuentes. Aunque la inserción en un AVL es ligeramente más lenta que en un árbol Rojo-Negro debido a las rotaciones más frecuentes, el AVL genera árboles más compactos (menor altura), lo que resulta en tiempos de búsqueda más rápidos para cargas de trabajo de "lectura intensiva".

Gestión de Memoria (Memory Allocators)

Sistemas operativos que necesitan rastrear bloques de memoria libres y ocupados según su dirección o tamaño.

El sistema operativo necesita encontrar el "mejor ajuste" (best fit) para un bloque de memoria rápidamente. Un árbol AVL permite buscar el bloque de tamaño adecuado en tiempo logarítmico garantizado, evitando la fragmentación y la latencia en la asignación de recursos.

Conclusión

La estructura de datos de pila, a pesar de su simplicidad conceptual, es una herramienta indispensable en la ciencia de la computación. Su principio fundamental LIFO proporciona una solución elegante, eficiente y segura para el problema del análisis de expresiones aritméticas. Al permitir el aplazamiento y la priorización de operaciones de manera sistemática, las pilas forman la base sobre la cual los compiladores e intérpretes modernos, como la Máquina Virtual de Java, son capaces de traducir la lógica matemática legible por humanos a un formato ejecutable por máquinas.

La comprensión de este mecanismo no solo es crucial para el estudio de algoritmos y estructuras de datos, sino que también ofrece una visión clara del funcionamiento interno de las tecnologías de software que utilizamos a diario.

Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Weiss, M. A. (2012). Data Structures and Algorithm Analysis in Java (3rd ed.). Pearson.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.