Estructuras de Datos no Lineales

Versión 1.0/AVL y ARN

CAMAVINGAGII

2025

ÍNDICE GENERAL

CAPÍTULO 1

AVL

1.1 ¿De dónde surgen los AVL?

Definición 1.1.1

Un **AVL** surge básicamente por los problemas que puede poseer un ABB:

- En un ABB, las sucesivas **inserciones** y **eliminaciones** pueden **alterar el grado de equilibrio del árbol**, es decir, puede quedar una rama que tenga muchos nodos y otra rama que no tenga apenas ninguno (véase la imagen).
- El **tiempo de las operaciones** sobre un ABB **depende de la altura**, por lo que el grado de desequilibrio del árbol puede llegar a ser **O(n) en el peor caso**, que es básicamente una lista.
- Para **garantizar un tiempo proporcional a la mínima altura posible, es decir, logarítmico, es necesario mantener el árbol tan equilibrado posible, y por esto surge el denominado AVL**

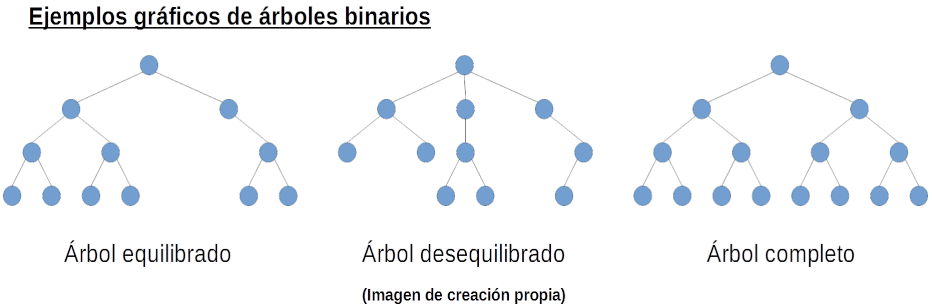

Figura 1.1: Ejemplos ABB

1.2 Definición de AVL

Definición 1.2.1

Un árbol AVL es un **tipo de árbol binario de búsqueda equilibrado** del que tenemos que saber las siguientes propiedades:

- **Factor de equilibrio** de un nodo: altura del subárbol derecho menos la altura del subárbol izquierdo del nodo.
- **Árbol binario equilibrado**: Aquel que el factor de equilibrio de **todos los nodos es -1,0 o 1**.
- La propiedad de equilibrio **garantiza que la altura de un AVL** es de orden logarítmico.
- Los algoritmos de inserción y eliminación en un AVL pueden **verificar la condición de equilibrio** y si es necesario, **reequilibran** el árbol mediante rotaciones de sus nodos, en un tiempo **proporcional a su altura**.
- En consecuencia, los tiempos de **búsqueda inserción y eliminación** en un AVL están en el orden logarítmico en el peor caso.

CAPÍTULO 2

ÁRBOL ROJINEGRO

Definición 2.0.1

Un árbol rojinegro (ARN) es un ABB que representa un árbol B de orden 3 (cada nodo 2 claves como máximo y una como mínimo), el cual está equilibrado (todas las hojas en el mismo nivel). Es una manera de implementar árboles B 2,3 mediante árboles binarios de búsqueda

- Esta representación permite una implementación **más sencilla** de las inserciones y eliminaciones, y también evita el sobrecoste de manejar una estructura de nodos más compleja, con más claves y punteros.

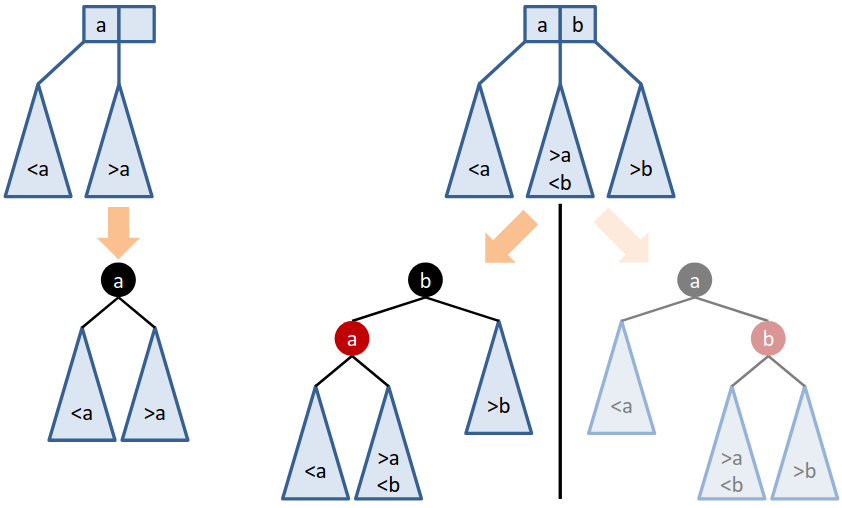

Figura 2.1: Ejemplo visual ARN

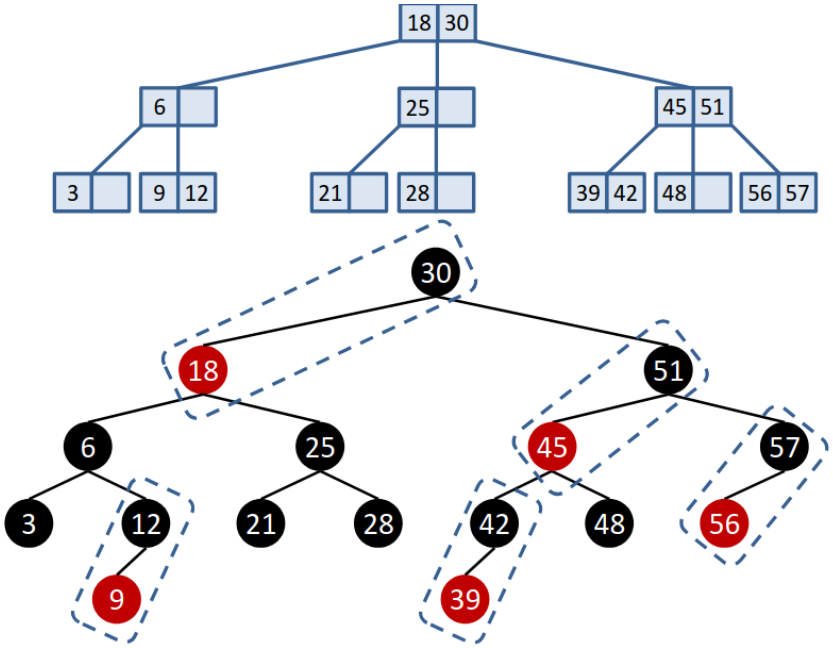

Figura 2.2: Ejemplo visual ARN

2.1 Propiedades

- Un ARN es un ABB con nodos rojos y negros que cumplen tres condiciones:
  - Cualquier **nodo rojo es hijo izquierdo** (implica que la raíz es de color negro).
  - Todo **nodo rojo tiene sus hijos negros**
  - **Toda rama desde la raíz tiene el mismo número de nodos negros**

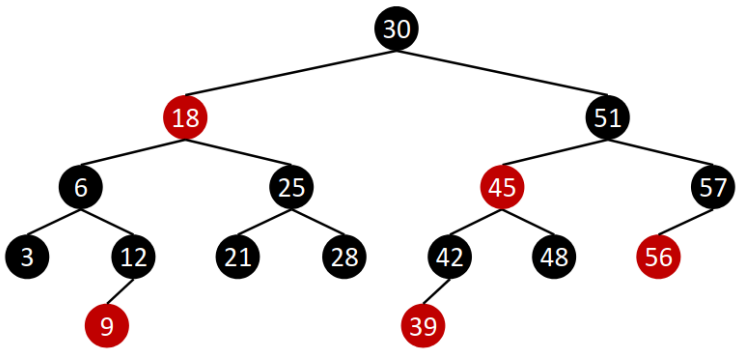

Figura 2.3: Ejemplo visual ARN

2.2 Consideraciones importantes

- **Inserción y eliminación en un árbol ARN**: Se realizan como en un ABB y a continuación se retrocede por el camino de búsqueda para restaurar, si es necesario, las propiedades de ARN (**debe representar el árbol B 2-3 subyacente**). Por lo tanto, como en un ABB, **los tiempos de estas operaciones son del ORDEN DE LA ALTURA DEL ÁRBOL (siempre es cierto**
- Operaciones internas:
  - **Rotaciones**: Recolocan los nodos del árbol conservando la propiedad de orden/búsqueda.
  - **Repintado**: Cambian el color de algunos nodos.
- **Eficiencia**: Un árbol rojinegro con n nodos tiene una **altura h** $<= 2log_2 n$ y, dado que las operaciones de búsqueda, inserción y eliminación tienen un peor tiempo de ejecución proporcional a la altura del árbol, entonces dichas operaciones son de $O(logn)$
- Se podría optar por representar los árboles ARN como árboles B de orden 4 y usar versiones más complicadas de algoritmos básicos que requieren menos rotaciones.

CAPÍTULO 3

AVL VS ARN

- Los AVL están **más equilibrados**, por lo que la **búsqueda es algo más rápida**, pero **las inserciones y eliminaciones son más lentas** debido a que al insertar hay que actualizar el factor de equilibrio de los ascendientes del nodo insertado, y si es necesario, realizar una rotación. Por otro lado, al eliminar, también hay que actualizar el factor de equilibrio y además pueden **necesitarse varias rotaciones extra para equilibrar el AVL**.
- En un ARN, la ventaja es que las inserciones y eliminaciones son algo más rápidas pero las búsquedas son algo más lentas que los AVL
- En la práctica la diferencia entre AVL y ARN respecto al rendimiento es poco apreciable.

3.0.1. ¿Cuándo utilizo uno u otro?

¿Cuándo usar AVL?

- Cuando las operaciones de **búsqueda son mucho más frecuentes** que inserciones o eliminaciones.
- Ideal para sistemas que **requieren búsquedas rápidas**, como:
  - Sistemas de archivos en memoria
  - Aplicaciones donde los datos no cambian mucho, pero se acceden constantemente
- AVL mantiene el árbol más compacto, lo que **reduce el número de comparaciones** en búsquedas.

¿Cuándo usar ARN?

- Cuando **hay muchas inserciones y eliminaciones** junto con búsquedas.
- Ideal para sistemas dinámicos donde **los datos cambian constantemente,** como:
  - Implementaciones internas de estructuras de alto rendimiento (por ejemplo, TreeMap en Java, std::map en C++)
  - Sistemas de bases de datos
  - Sistemas concurrentes donde las operaciones deben ser rápidas y constantes
- ARN **sacrifica** algo de **rendimiento en búsquedas**, pero compensa con operaciones de actualización más baratas.

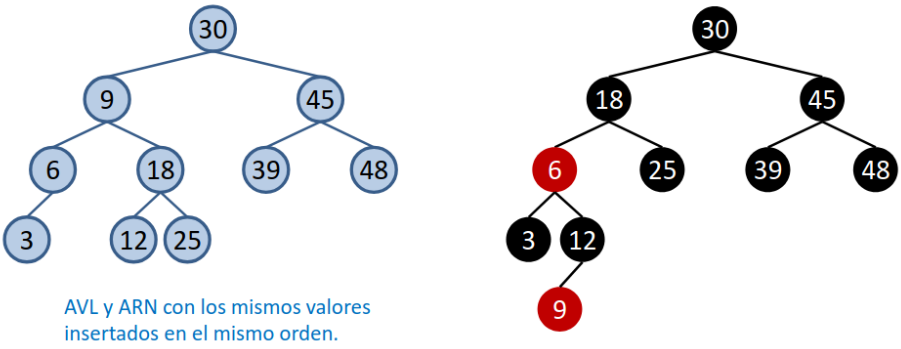

Figura 3.1: Ejemplo de los dos árboles