Universidad de las Fuerzas Armadas – ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Software

Estructura de Datos – NRC28436

Tema: Concepto de Árbol
Clasificación de Árboles (Binarios, AVL, B, B+)

Número de Grupo 1

- Ampuero Jonathan
- Bustos Ruben
- Guano Victor
- Manosalvas Gabriel

Docente: Ing. Washington Loza H. Mgs.

Quito, Quito, 10 de diciembre de 2025

Resumen

En el presente informe exploraremos la estructura de datos fundamental del Árbol, destacando la principal organización jerárquica de la información en la programación y como este interactúa y como realiza sus combinaciones hasta hacerlo más efectivo e eficiente. Se estudio un catálogo que incluye los Árboles Binarios, Árboles AVL, y los Árboles B y B +, agilizando la gestión eficiente de datos y la utilización en bases de datos. El objetivo es puntualizar su actividad, estructura y su uso práctico en la Ingeniería de Software, distinguiendo cómo estas estructuras comprometiendo operaciones ágiles y escalables.

1. Introducción

Esta investigación planteamos abordar una de las estructuras de datos no lineal más básico en la informática, como son los árboles. Su diferencia de las estructuras lineales como son las listas y pilas, los árboles simbolizan jerarquías y relaciones de manera eficaz, lo que optimiza significativamente las ejecuciones de búsqueda, inserción y eliminación de datos.

2. Objetivo General

El objetivo central de esta presentación es ahondar en el concepto principal del Árbol y analizar su funcionamiento. Asimismo, se indagará una clasificación de árboles, enfocándonos en su estructura como son los (Árboles Binarios, Árboles AVL, Árboles B, Árboles B +).

3. Desarrollo

3.1. Árboles Binarios

3.1.1. Definición

Es un árbol ordenado en el cual sigue un cierto orden en la distribución de sus ramas. Además, que los árboles ya ordenados de grado 2 es esencial en el área de computación ya que puede representar la información relacionada con la solución de muchos problemas. Un árbol binario se dice que cada nodo puede tener máximo dos subárboles (subárbol izquierdo y subárbol derecho) según la ubicación del nodo Raíz.

Los árboles binarios tienen múltiples aplicaciones ya que se les puede utilizar en soluciones de un problema los cuales existen 2 alternativas árbol de decisiones para el cual puede representar el árbol genealógico construido de forma ascendente.

3.1.2 Representación Gráfica

Existen diferentes estructuras de Árbol Binario como lo son:

- **Árbol binario de búsqueda**

Se trata de una de las estructuras de datos organizados como lo que es el árbol binario. Cada nodo va a tener una clave, mientras que todos los valores en el subárbol izquierdo de un nodo son menores que su clave.

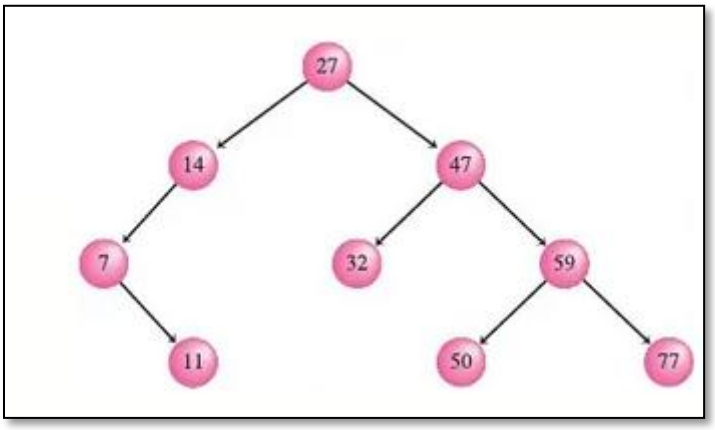


Figura 1. Árbol binario de búsqueda.

- **Árboles binarios Distintos**

Se puede decir que dos árboles binarios son distintos cuando sus estructuras como lo son los arcos y la distribución de nodos son diferentes.

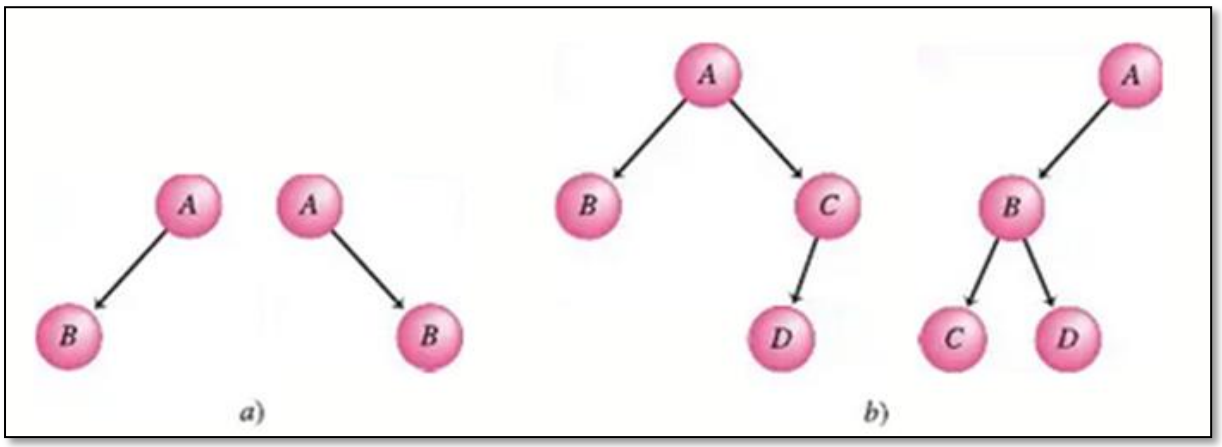


Figura 2. Árboles binarios distintos**.**

- **Árboles Similares**

Se dice que son iguales cuando sus estructuras son iguales o idénticas pero la información que contiene sus nodos difiere entre sí, se presentan dos ejemplos de árboles binarios similares.

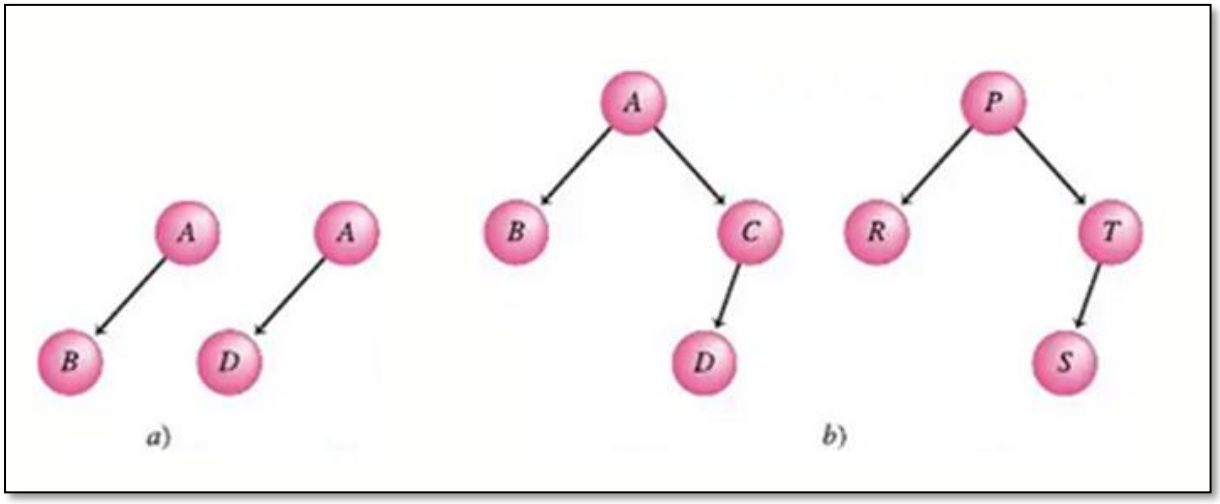


Figura 3. Árboles Similares.

- **Árboles equivalentes**

Son aquellos que son similares y los nodos contienen la misma información.

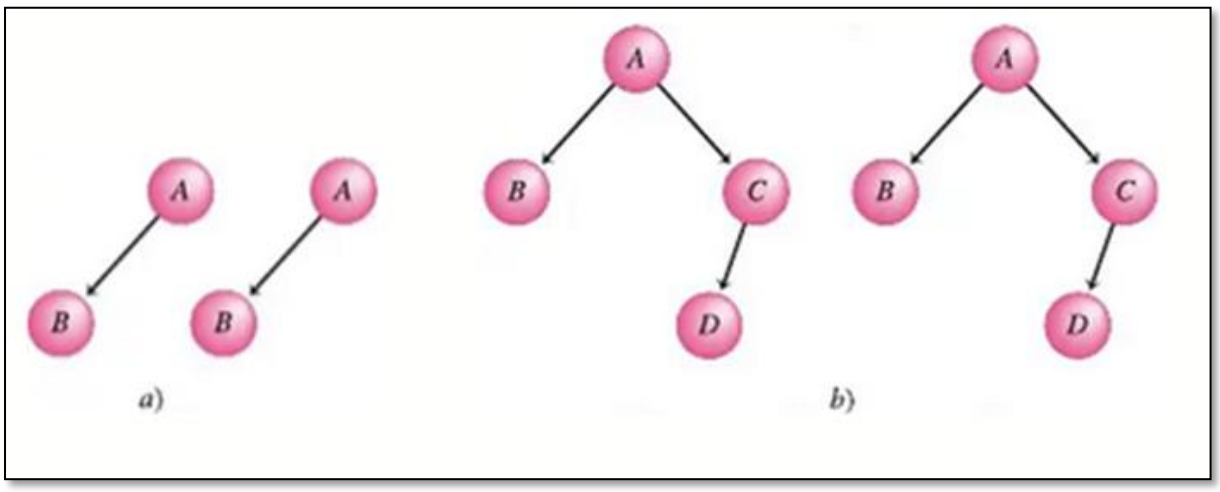


Figura 4. Árboles equivalentes.

- **Árboles Binarios completos**

Todos sus nodos excepto los del último nivel, tienen sus dos hijos el subárbol izquierdo y el subárbol derecho.

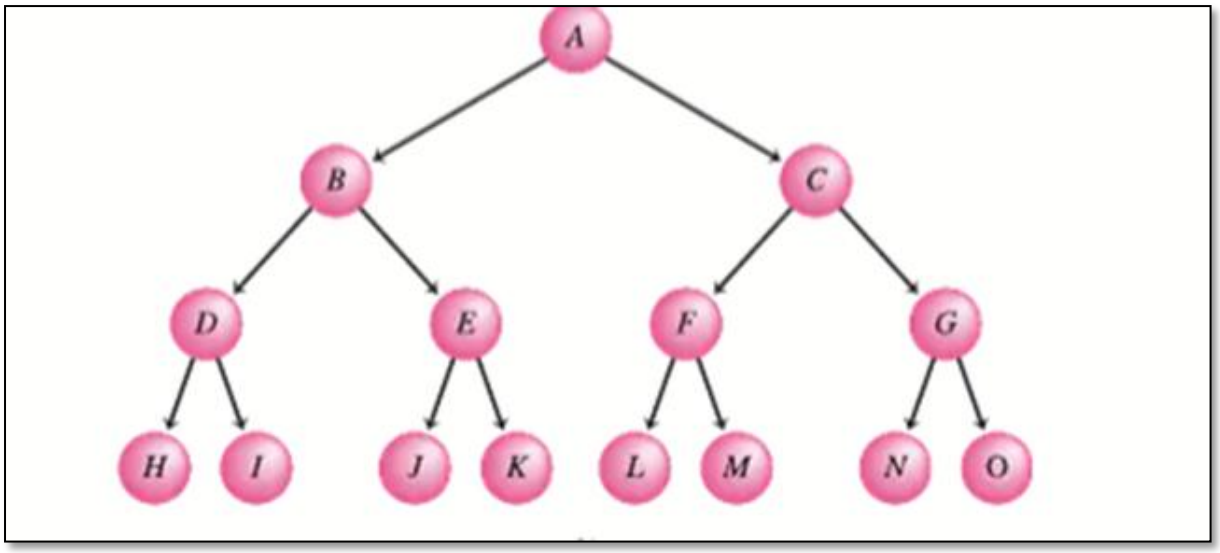


Figura 5. Árboles Binarios completos.

3.1.3 Sintaxis General

La estructura de un árbol se la define de la siguiente manera una estructura recursiva teniendo en cuenta que cada nodo del árbol, junto con todos sus descendientes, y manteniendo la ordenación original constituye también un árbol o subárbol del árbol principal. (P, 2017).

Se puede decir que el movimiento que se da en los árboles será siempre el inicio el nodo raíz hacia un nodo hoja solo si no se implementa punteros al nodo padre.

```
struct Arbol {
        int dato;
        struct Arbol *rama1;
        struct Arbol *rama2;
        };
```

Figura 6. Sintaxis Árboles Binarios.

3.1.4 Ventajas y Desventajas

Ventajas	Desventajas
El número de accesos al árbol es menor que en una lista enlazada.	Los nodos que se van insertando deben tener un orden aleatorio y ser equilibrado.
Las búsquedas son más eficientes cuando el árbol está equilibrado	Si los datos no son de un mismo tipo no ingresaran
Permite recorrer todos los datos, en el orden que se desee o se requiera	Se debe seguir la estructura.
Permite el ordenamiento de manera continua.	El orden dentro de la programación tiene un cierto grado de dificultad.

Tabla 1. Ventajas y Desventajas.

3.1.5 Comparación

Características	Árbol Binario	Árbol B+
Número de hijos por nodo	Máximo 2 hijos.	Entre n y 2n claves por nodo.
Altura del árbol	Desbalancear y crecer mucho.	Siempre balanceado ya que todas las hojas están al mismo nivel.
Eficiencia de búsqueda	Peor caso O(n).	Garantizado O (log n).
Inserción	Puede desbalancear.	Divide nodos llenos
Eliminación	Complicada en nodos con dos hijos.	Requiere fusiones (merge) y redistribución.
Ventaja principal	Sencillez y flexibilidad.	Alta eficiencia en búsqueda y almacenamiento masivo.

Tabla 2. Comparación.

3.1.6 Aplicaciones

Los árboles Binarios pueden tener una gran cantidad de aplicaciones como lo es en el área informática.

1. **Almacenar y recuperar datos de manera eficiente:** Se puede implementar con árboles de búsqueda binaria.

2. **Inteligencia Artificial**

Un claro ejemplo es cuando tenemos un árbol genealógico donde cada persona tiene obligatoriamente un Padre donde cada nodo va a representar un punto de decisión y las ramas lo que van a representar son los resultados.

Para el desarrollo de Android los árboles binarios los podríamos usar en:

1. **Ver jerarquía:** Para este diseño sabemos que las interfaces de los usuarios en cualquier aplicación móvil se los representa como un árbol además que pueden ser optimizados en su diseño.
2. **Navegación:** En las aplicaciones móviles los gráficos de navegación también son representados como árboles binarios ya que se los puede optimizar las rutas de la navegación.

3.1.7 Conclusiones

- Existen varios tipos de árboles binarios los cuales nos ayudan a satisfacer las necesidades dependiendo la estructura requerida.
- Existen ventajas importantes como los recorridos ordenados y los accesos rápidos, aunque también presentan limitaciones como la necesidad de mantener un equilibrio adecuado.
- La eficiencia de un árbol binario va a depender de su equilibrio.

3.2. Árboles AVL

3.2.1. Definición

Un árbol AVL agrega un engranaje de auto balance, principalmente se define como un sobrante entre el máximo (Altura) del subárbol izquierdo y derecho de un Nodo, su principal particularidad es realizar reacomodos, después de sus inserciones o eliminación de elementos.

En algunos libros este concepto se encuentra como una definición de árbol balanceado o como un árbol de búsqueda donde para todo nodo T de un árbol la altura del subárbol derecho e izquierdo no deben diferir en lo suma de una unidad.

3.2.2. Características

- La principal diferencia entre las alturas de los subárboles no debe excederse en más de 1.
- Cada Nodo tiene asignado un peso de acuerdo con la altura.
- La inserción y eliminación en un árbol AVL es la misma que en un ABB.

3.2.3. Ejemplo

En el presente ejemplo el árbol de la izquierda es AVL, mientras que el de la derecha infringe la condición de equilibrio en el Nodo 6, ya que su subárbol izquierdo tiene altura 3 y su subárbol derecho tiene altura 1.

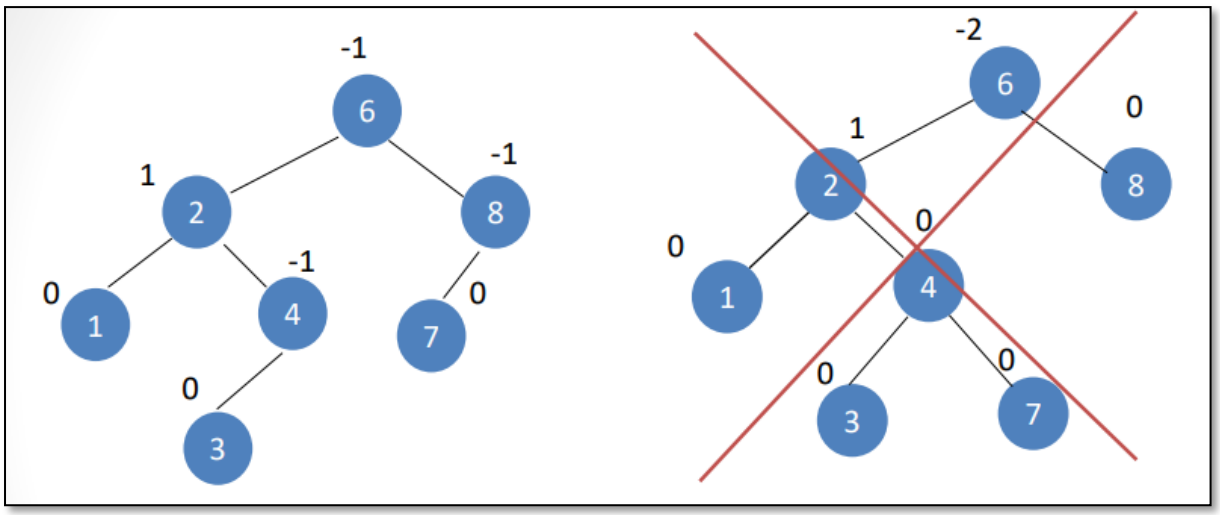


Figura 7. Ejemplo Árbol AVL.

EQUILIBRIO = ALTURA DERECHA - ALTURA IZQUIERDA

Donde:

+ (Positivo) = Más alto en la parte derecha [Profundo]
- (Negativo) = Más alto en la parte izquierda [Profundo]

El árbol binario es un AVL si y sólo si cada uno de sus nodos tiene un equilibrio de -1, 0 + 1. Si alguno de los pesos se modifica en un valor no válido debe seguir un esquema de rotación.

- Desequilibrio hacia la izquierda (EQUILIBRIO > +1)

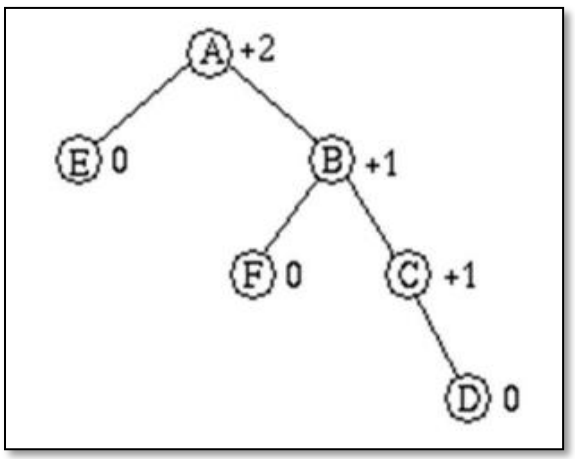


Figura 8. Ejemplo Árbol AVL.

- Desequilibrio hacia la derecha (EQUILIBRIO < -1)

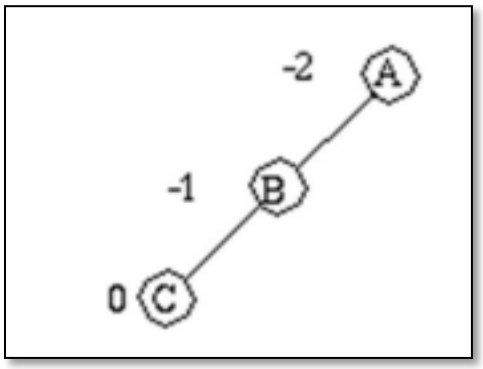


Figura 9. Ejemplo Árbol AVL.

3.2.4. Operaciones sobre un AVL

1.- Insertar Nodo
2.- Balancear
- Caso 1 - Rotación simple izquierda RSI
- Caso 2 - Rotación simple derecha RSD
- Caso 3 - Rotación simple izquierda RDI
- Caso 4 - Rotación doble derecha RDD

3.- Eliminar Nodo
4.- Calcular Altura

3.2.4. Tipos de Balanceos

- **Caso 1.- Rotación simple izquierda RSI**

En este caso se considera si está desequilibrado a la izquierda (E>+1) y su hijo derecho tiene el mismo signo (+) hacemos rotación sencilla izquierda.

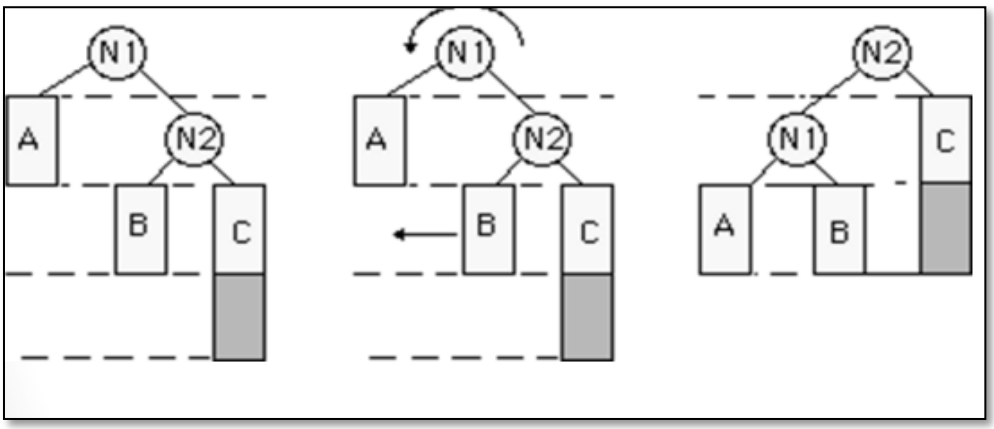


Figura 10. Rotación simple izquierda.

- **Caso 2.- Rotación simple derecha RSD**

En este caso se considera si está desequilibrado a la derecha (E<-1) y su hijo izquierdo tiene el mismo signo (-0) hacemos rotación sencilla derecha.

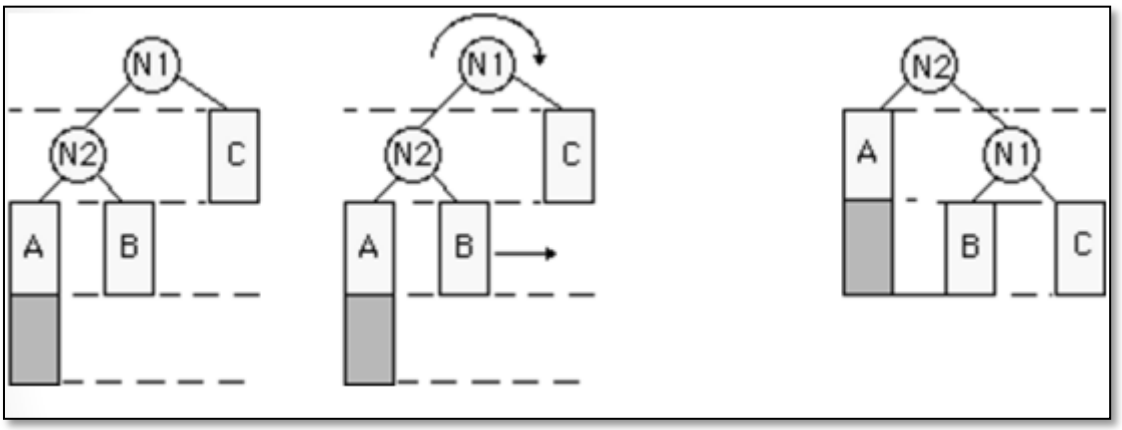


Figura 11. Rotación simple derecha.

- **Caso 3.- Rotación doble izquierda RDI**

En este caso se considera si está desequilibrado a la derecha (E < -1), y su hijo izquierdo tiene distinto signo (+) hacemos rotación doble izquierda-derecha.

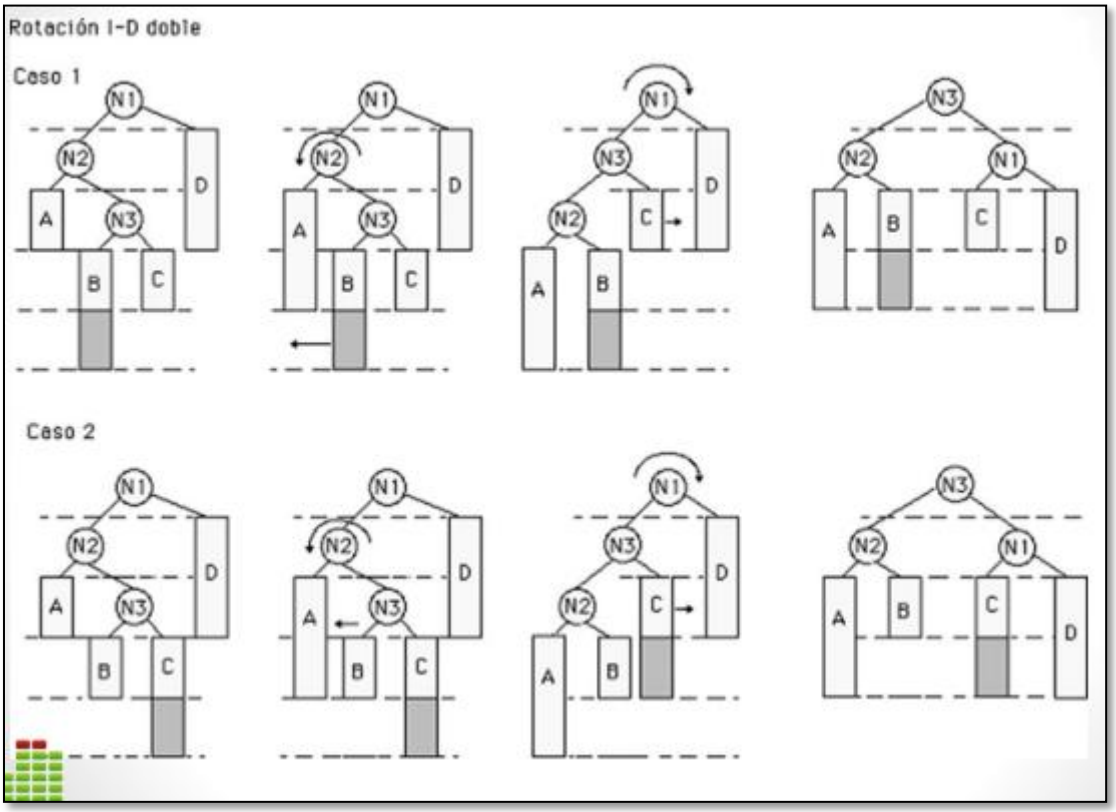


Figura 12. Rotación doble izquierda.

- **Caso 4.- Rotación doble derecha RDD**

En este caso se considera si está desequilibrado a la izquierda (E >+1), y su hijo derecho tiene distinto signo (-) hacemos rotación doble derecha - izquierda.

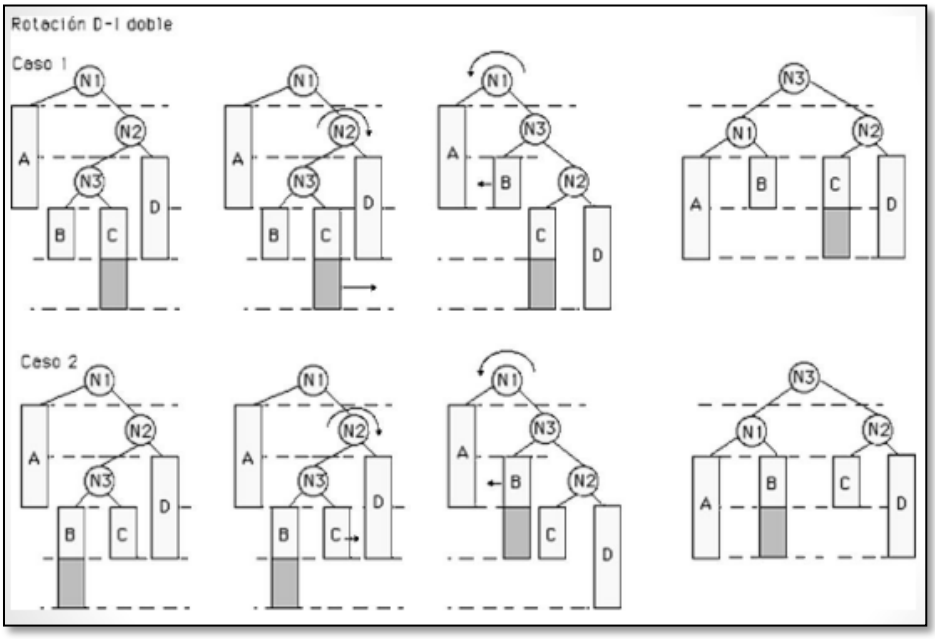


Figura 13. Rotación doble derecha.

3.2.5. Complejidad de búsqueda

Una vez conocido los árboles AVL debemos tener en cuenta que estos siempre están equilibrados de tal modo que, para todos los nodos, su altura de la rama izquierda no toma ninguna decisión en más de una unidad de la altura de la rama derecha.

Estos árboles mantienen un orden de complejidad O(log2n).

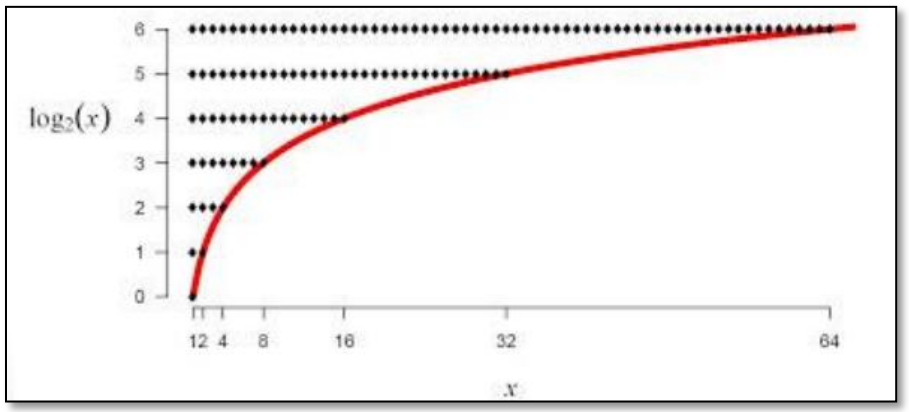


Figura 13. Complejidad AVL.

3.3. Árboles B

3.3.1. Definición

Un árbol B es un árbol balanceado de búsqueda multinivel y de múltiples hijos. A diferencia de los árboles binarios, es una estructura multi ramificada (m - way) que se mantiene balanceada de forma automática.

- Funcionan de manera eficiente en sistemas que leen y escriben grandes bloques de datos.
- Su diseño minimiza las operaciones.
- Se pueden utilizar en bases de datos a gran escala como MySQL.

3.3.2. Representación gráfica

Un árbol B destaca por su gran estructura y poca altura, y esto permite almacenar muchos elementos en pocos niveles.

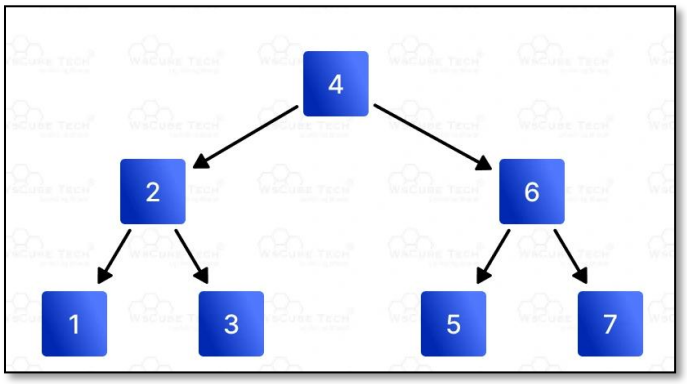


Figura 13. Representación gráfica Árbol B.

En un árbol B:

- Cada nodo puede contener múltiples valores.
- Cada nodo puede tener varios hijos.
- El árbol se mantiene equilibrado asegurándose de que todas las hojas (los nodos en la parte inferior) estén al mismo nivel.

La idea principal de un árbol B es mantener los datos organizados de manera que sea rápido encontrar, agregar o eliminar información. Esto es especialmente importante cuando se trabaja con grandes cantidades de datos, como en bases de datos o sistemas de archivos.

Propiedades del árbol B

- **Árbol equilibrado**

Los árboles B están equilibrados, lo que significa que todos los nodos de las hojas (los nodos en la parte inferior del árbol) están al mismo nivel. Esto garantiza que el árbol permanezca corto, lo que ayuda a una recuperación de datos eficiente.

- **Estructura del nodo**

Cada nodo de un árbol B puede contener múltiples claves (valores) e hijos. La cantidad de claves que puede contener un nodo está determinada por el orden del árbol B, denominado 'm'.

- **Orden m**

El orden 'm' de un árbol B define el rango de claves e hijos que puede tener cada nodo:

- Un nodo puede tener como máximo 'm' hijos.
- Un nodo puede tener como máximo claves 'm-1'.

Cada nodo, excepto la raíz, debe tener al menos ⌈m/2⌉ hijos y ⌈m/2⌉-1 claves. El nodo raíz debe tener al menos dos hijos si no es un nodo hoja.

- **Claves en nodos**

Las claves dentro de un nodo se almacenan de forma ordenada. Esto ayuda a realizar una búsqueda eficiente dentro del nodo.

- **Punteros**

Cada nodo tiene punteros a sus nodos secundarios. El número de punteros secundarios es siempre uno más que el número de claves en el nodo.

- **Rango de datos en subárboles**

Para cualquier nodo dado:

- Todas las claves del subárbol izquierdo son más pequeñas que las claves del nodo.
- Todas las claves del subárbol derecho son mayores que las claves del nodo.

- **Altura del árbol**

La altura de un árbol B se mantiene baja al permitir que los nodos tengan varios hijos. Esto garantiza que operaciones como búsqueda, inserción y eliminación se puedan realizar de manera eficiente.

3.3.3. Sintaxis General

En C + +, siguiendo a Deitel & Deitel, la representación del árbol B es mediante una clase que gestiona arreglos dinámicos para almacenar múltiples claves por nodo.

Estructura Técnica

- Nodos: Contienen un arreglo de claves y un arreglo de punteros a hijos.
- Orden(m): Define el número máximo de hijos, un nodo tiene máximo m-1 claves.
- Balanceo: Todas las hojas deben estar en el mismo nivel.

3.3.4. Ventajas y Desventajas

Ventajas	Desventajas
Altura reducida: Permite búsquedas extremadamente rápidas en discos lentos.	**Uso de Memoria:** Los nodos pueden tener espacios vacíos (memoria no utilizada).
Balanceo Perfecto: No existe el riesgo de que el árbol se degrade a una lista.	**Complejidad Algorítmica:** El borrado de nodos requiere reestructuraciones complejas.
Operaciones O (log n): Eficiencia constante para búsqueda e inserción.	**Sobrecarga en RAM:** Para datos pequeños, un árbol AVL es más ágil.

Tabla 3. Ventajas y Desventajas.

3.3.5. Comparación Árbol B vs Árbol Binario de Búsqueda

Característica	Árbol Binario	Árbol B
Capacidad por Nodo	1 clave	Múltiples Claves
Factor de Ramificación	Máximo 2 hijos	Hasta m hijos
Altura del Árbol	Alta	Muy baja
Crecimiento	Hacia abajo (hojas)	Hacia arriba (raíz)

Tabla 4. Comparación Árbol B vs ABB .

3.3.6. Aplicaciones

Motores de Búsqueda

- Los buscadores internos de plataformas grandes como eBay, utilizan variantes denominadas Block Trees, para organizar el diccionario de términos que permite encontrar palabras clave.
- Y es útil porque en una plataforma como eBay que permite buscar celulares entre 200$ y 500$, el sistema navega por el árbol directamente hasta el nodo del precio 200 y lee secuencialmente hasta el 500, entregando todos los resultados posibles en milisegundos.

3.3.7. Conclusión y Desafío

- El árbol B es una estructura de árbol óptima para el manejo de Big Data y almacenamiento.
- La implementación de la división de nodos y la gestión de punteros al momento de utilizarlos en un entorno multi ramificado.

3.4. Árboles B+

3.4.1. Definiciones fundamentales

Árbol B+

Un árbol B + es un tipo de árbol de búsqueda multidireccional que mantiene datos ordenados y permite operaciones eficientes de inserción, eliminación y búsqueda. A diferencia de un árbol B, que almacena tanto claves como datos en nodos internos y hojas, un árbol B + solo almacena claves en sus nodos internos y conserva todos los datos en las hojas.

Esta distinción permite una mayor eficiencia en la búsqueda y el acceso secuencial, lo que convierte a los árboles B en la opción preferida para la indexación de bases de datos y sistemas de archivos.

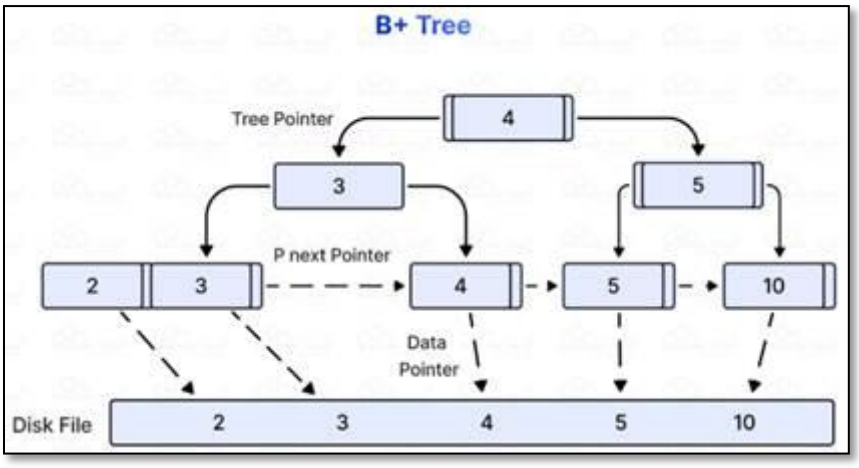


Figura 14. Representación gráfica Árbol B +.

Los árboles B + se han convertido en la técnica más utilizada para la organización de archivos indizados. La principal característica de estos árboles es que toda la información se encuentra en las hojas, mientras que los nodos raíz e interiores almacenan claves que se utilizan como índices. Debido a esta característica de los árboles B +, todos los caminos desde la raíz hasta cualquiera de los datos tienen la misma longitud.

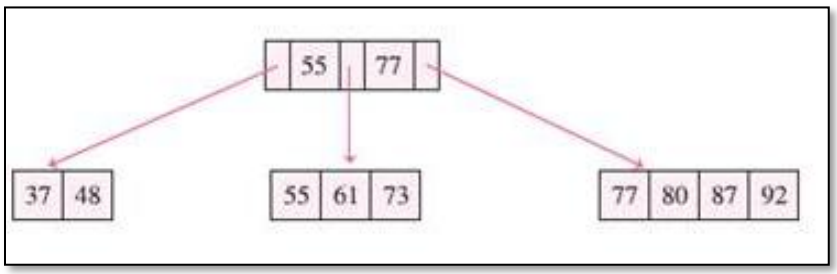


Figura 15. Representación gráfica Árbol B +.

Es de notar que los árboles B + ocupan un poco más de espacio que los árboles B, y esto ocurre al existir duplicidad en algunas claves. Sin embargo, esto es aceptable si el archivo se módica frecuentemente, puesto que se evita la operación de reorganización del árbol que es tan costosa en los árboles B.

3.4.2. Representaciones gráficas e imágenes:

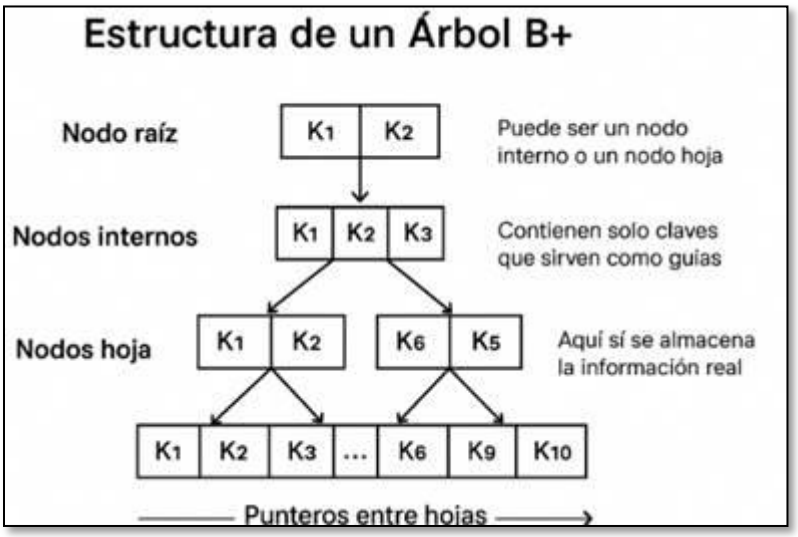


Figura 16. Estructura Árbol B +.

3.4.3. Sintaxis general y estructura técnica

Inserción en árboles B+

El proceso de inserción en árboles B + es relativamente simple, similar al proceso de inserción en árboles B. La dificultad se presenta cuando se desea insertar una clave en una página que se encuentra llena (m = 2d) donde m es claves o número de elementos y d es el orden del árbol.

En este caso, la página (o nodo) afectada se divide en 2, distribuyéndose las m + 1 claves de la siguiente forma: "las d primeras claves en la página de la izquierda y las d + 1 restantes claves en la página de la derecha". Una copia de la clave del medio sube a la página antecesora.

En la figura se muestran dos diagramas que ilustran cómo funciona este caso. Puede suceder que la página antecesora se desborde nuevamente, en dicho caso se debe repetir el proceso anterior. Es importante notar que el desbordamiento en una página que no es hoja no produce duplicidad de claves. El proceso de propagación puede llegar hasta la raíz, en cuyo caso la altura del árbol se puede incrementar en una unidad.

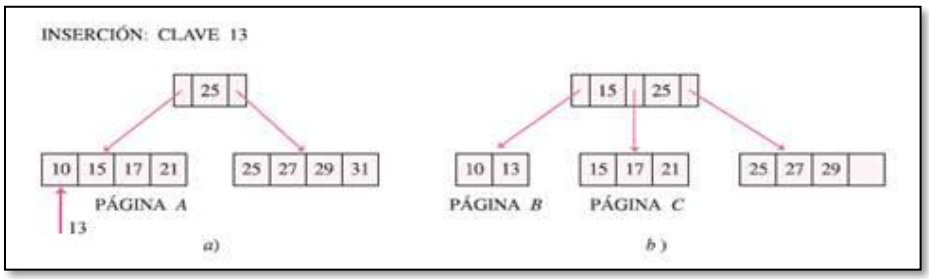


Figura 17. Inserción Árbol B +.

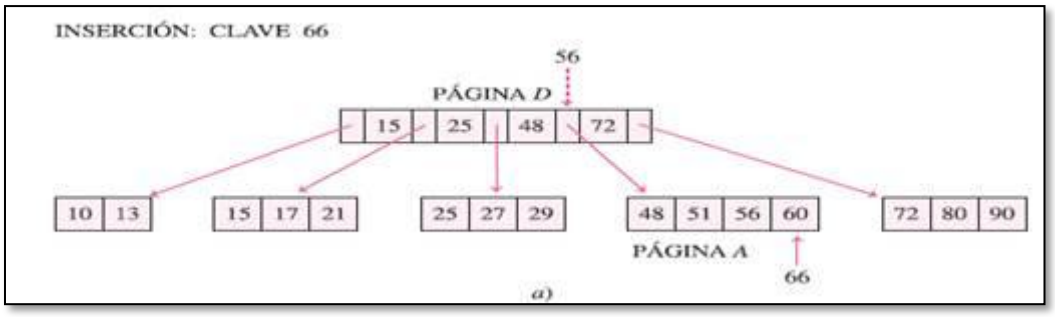


Figura 18. Inserción Árbol B +.

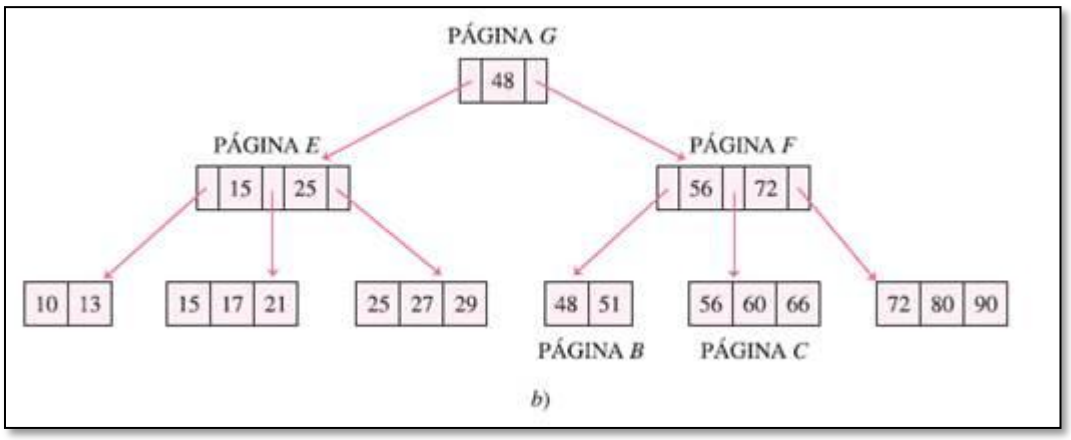


Figura 18. Inserción Árbol B +.

3. **Ventajas y desventajas**

Ventajas	Desventajas
Alta ramificación en abanico: El alto factor de ramificación reduce la altura del árbol y disminuye operaciones de E/S de disco, mejorando el rendimiento.	Sobrecarga de espacio: Puede haber mayor uso de memoria por almacenar niveles de claves y punteros, afectando conjuntos de datos grandes.
Acceso secuencial y consultas de rango: Los nodos hoja enlazados permiten recorrido secuencial eficiente y consultas de rango sin volver a nodos internos.	Complejidad del equilibrio: Inserciones y eliminaciones requieren equilibrar, lo que añade sobrecarga en actualizaciones frecuentes.
Optimización para Entrada y Salida de Disco (I/O): El tamaño de nodos alineado a páginas de disco minimiza lecturas/escrituras, ideal cuando los datos no caben en memoria.	Rendimiento de la caché: Puede presentar accesos no contiguos causantes de fallos de caché y ligera pérdida de rendimiento.

Escalabilidad: Crece en altura de forma logarítmica, manteniendo rendimiento con grandes volúmenes de datos.	Gestión de concurrencia: Requiere mecanismos avanzados para mantener consistencia y evitar cuellos de botella en altos niveles de concurrencia.

Tabla 5. Ventajas y Desventajas.

7. Aplicación práctica o caso de uso

Bases de datos NoSQL y NewSQL

Los árboles B + se utilizan en bases de datos NoSQL (como MongoDB) y MySQL (como Google Spanner) para gestionar la indexación y realizar consultas complejas. Su eficiencia al gestionar grandes volúmenes de datos y proporcionar un acceso rápido los convierte en una opción popular para estos sistemas de bases de datos modernos.

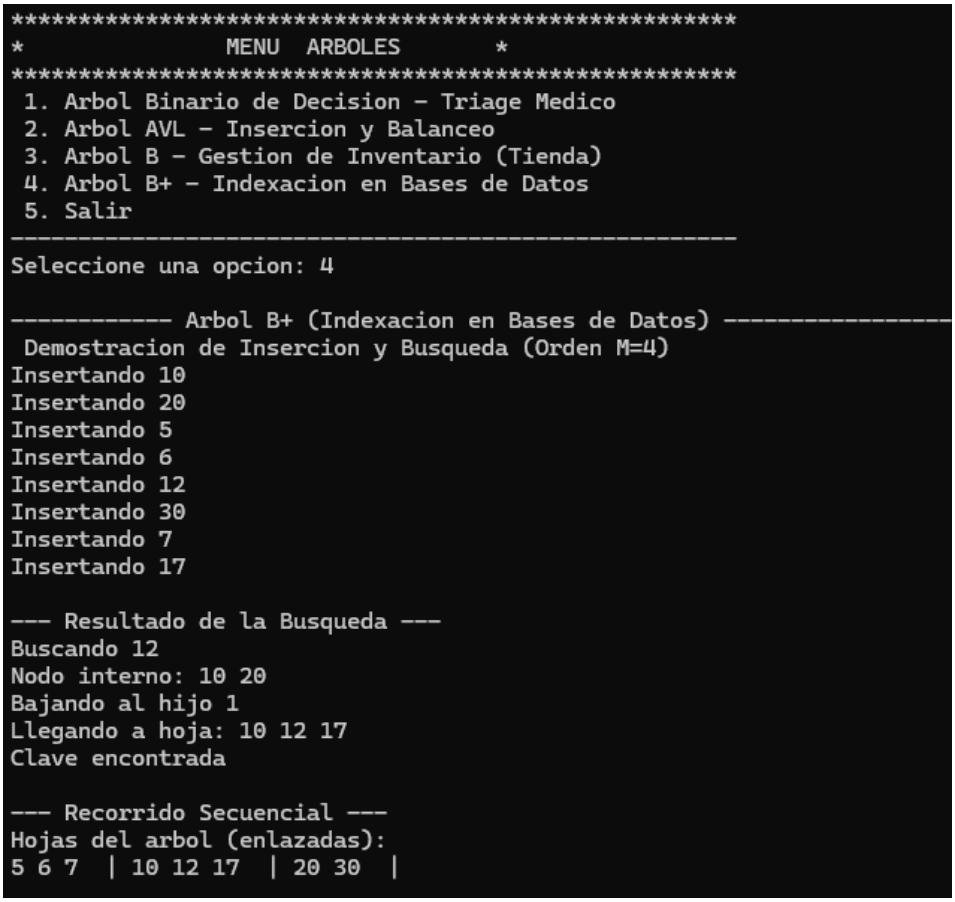
4. Caso práctico

```
******************************************************
*               MENU  ARBOLES        *
******************************************************
 1. Arbol Binario de Decision - Triage Medico
 2. Arbol AVL - Insercion y Balanceo
 3. Arbol B - Gestion de Inventario (Tienda)
 4. Arbol B+ - Indexacion en Bases de Datos
 5. Salir
------------------------------------------------------
Seleccione una opcion: 1

------------ Triage Medico (Arbol Binario de Decision) ------
Tiene fiebre? (si/no): si
Tiene dificultad para respirar? (si/no): no
Tiene presion arterial alta? (si/no): si
Diagnostico final: OBSERVACION INMEDIATA
```

```
*****************************************************
*               MENU  ARBOLES       *
*****************************************************
 1. Arbol Binario de Decision - Triage Medico
 2. Arbol AVL - Insercion y Balanceo
 3. Arbol B - Gestion de Inventario (Tienda)
 4. Arbol B+ - Indexacion en Bases de Datos
 5. Salir
-----------------------------------------------------
Seleccione una opcion: 2

------------ Arbol AVL (Balanceo Automatico) -----------------
 Demostracion de Insercion y Rotacion (30, 20, 10)
Insertando: 30
Insertando: 20

Nodo 30 FE=1
Insertando: 10

Nodo 20 FE=1

Nodo 30 FE=2
-> Desbalance Detectado (RR)
-> Rotacion Derecha (RR) sobre nodo: 30

Arbol final en InOrden: 10 20 30

Arbol AVL balanceado correctamente.
```

```
--- MENU ARBOL B (Inventario) ---
1. Insertar producto
2. Buscar producto por ID
3. Mostrar inventario (recorrido)
4. Volver al menu principal
Seleccione: 1
ID del producto: 23
Nombre del producto: lentes
Producto insertado.
```

```
--- MENU ARBOL B (Inventario) ---
1. Insertar producto
2. Buscar producto por ID
3. Mostrar inventario (recorrido)
4. Volver al menu principal
Seleccione: 3
Inventario (clave - nombre):
[2 - gorra] [3 - billetera] [23 - lentes]
```



5. Conclusiones

- Concluimos que los Árboles son una estructura de datos esencial para la estructurar una jerarquía. Los Árboles Binarios y AVL garantizan búsquedas eficientes en memoria gracias al auto balanceo que mantiene el rebuscamiento de O(log2n).
- Los Árboles B y B + son estructuras optimizadas para el almacenamiento colectivo, minimizando las operaciones al reducir la altura a través del factor de ramificación.

6. Bibliografía

[1] Eduardo, J. (s. f.). *Estructuras de datos - Osvaldo Cairó, Silvia Guardati - 3ra edición*. Scribd. https://es.scribd.com/document/626046903/Estructuras-de-Datos-Osvaldo-Cairo-Silvia-Guardati-3ra-Edicion

[2] *Docencia*. (2010). Docencia eaFranco. https://docencia.eafranco.com/materiales/estructurasdedatos/11/Tema11.pdf.

[3] Di Mare, A. (15 de Febrero de 2006). Una clase C + + completa para conocer y usar árboles. Obtenido de Una clase C++ completa para conocer y usar árboles: http://www.di-mare.com/adolfo/p/TreeTdef.htm

[4] Make Computer Science Great Again. (s. f.). *Understanding B-Trees: A Comprehensive Guide*. https://medium.com/@MakeComputerScienceGreatAgain/understanding-b-trees-a-comprehensive-guide-8b4e12486677

[5] Deitel, H. M., & Deitel, P. J. (2004). *Cómo programar en C/C ++ y Java*. Pearson Educación.

[6] Koffman, E. B., & Wolfgang, P. A. T. (2005). *Objects, Abstraction, Data Structures and Design: Using C++*. John Wiley & Sons.