Universidad Cenfotec

Autor(es):

-Isabel Galeano Hernández, Cédula: 1-1888-0968

-Daniel Zúñiga Rojas, Cédula: 1-1811-0097

-Isaías Arce Rodríguez, Cédula: 3-0525.0698

Tema:
Investigación Árboles Binarios

Curso:
BISOFT-20 Estructuras de datos 2

Profesor:
Christian Sibaja Fernández

Fecha de entrega:
18/07/2021

Cuatrimestre:
2021-3

Índice

Árbol B

Los árboles-B son estructuras de datos de tipo árbol que se encuentran comúnmente en las implementaciones de bases de datos y sistemas de archivos. Son árboles binarios de búsqueda en los cuales cada nodo puede poseer más de dos hijos.

La idea principal de los árboles binarios es que los nodos internos deben tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten.

Características importantes de los árboles-B:

Un árbol-B de orden M (el máximo número de hijos que puede tener cada nodo) es un árbol que satisface las siguientes propiedades:

1. Cada nodo tiene como máximo M hijos.
2. Cada nodo (excepto raíz y hojas) tiene como mínimo M/2 hijos.
3. La raíz tiene al menos 2 hijos si no es un nodo hoja.
4. Todos los nodos hoja aparecen al mismo nivel.
5. Un nodo no hoja con k hijos contiene k-1 elementos almacenados.
6. Los hijos que cuelgan de la raíz (r1, ⋯, rm) tienen que cumplir ciertas condiciones:
    1. El primero tiene valor menor que r1.
    2. El segundo tiene valor mayor que r1 y menor que r2, etc.
    3. El último hijo tiene valor mayor que rm.

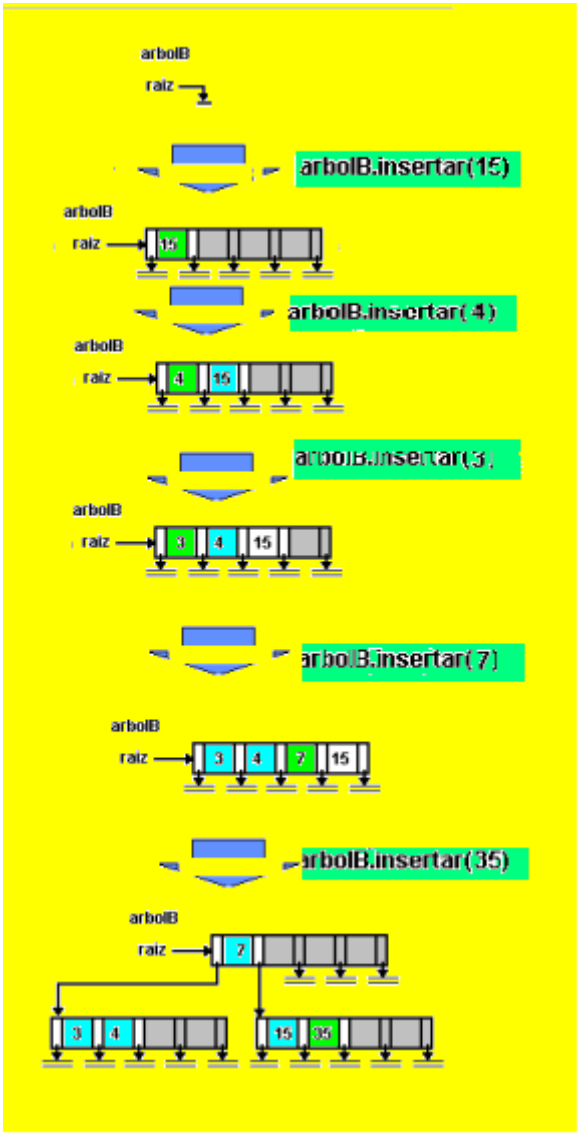
Árbol-B Altura: Con el objetivo de tener una acotación de la altura, h, del árbol B en función del número total de clave almacenados, n, y del orden del árbol B, m, calculamos inicialmente el número mínimo de enlaces por nivel en un árbol B.

Nivel	Número mínimo de enlaces
1	2

2	$2*(m/2)$
3	$2*(m/2)^2$
h-1	$2*(m/2)^{h-2}$

Ejemplo de búsqueda en un árbol-B: Se empieza en la raíz, y se recorre el árbol hacia abajo, escogiendo el sub-nodo de acuerdo a la posición relativa del valor buscado respecto a los valores de cada nodo.

1. Situarse en el nodo raíz.
2. (*) Comprobar si contiene la clave a buscar.
    1. Encontrada fin de procedimiento.
    2. No encontrada:
        1. Si es hoja no existe la clave.
        2. En otro caso el nodo actual es el hijo que corresponde:
            1. La clave a buscar k < k1: hijo izquierdo.
            2. La clave a buscar k > ki y k < ki+1 hijo iésimo.
            3. Volver a paso 2(*).


arbolB
raiz
arbolB.insertar(15)
arbolB
raiz
15
arbolB.insertar(4)
arbolB
raiz
4
15
arbolB.insertar(3)
arbolB
raiz
3
4
15
arbolB.insertar(7)
arbolB
raiz
3
4
7
15
arbolB.insertar(35)
arbolB
raiz
7
3
4
15
35

Árbol AVL

Es un árbol de búsqueda binario que trata de mantenerlo lo más balanceado posible, conforme se realizan inserciones y eliminaciones. Fueron propuestos en 1962 por los matemáticos rusos Adelson, Velskii y Landis, de ahí su nombre. En cualquier nodo del árbol, la diferencia entre las alturas de los subárboles no debe exceder una unidad.

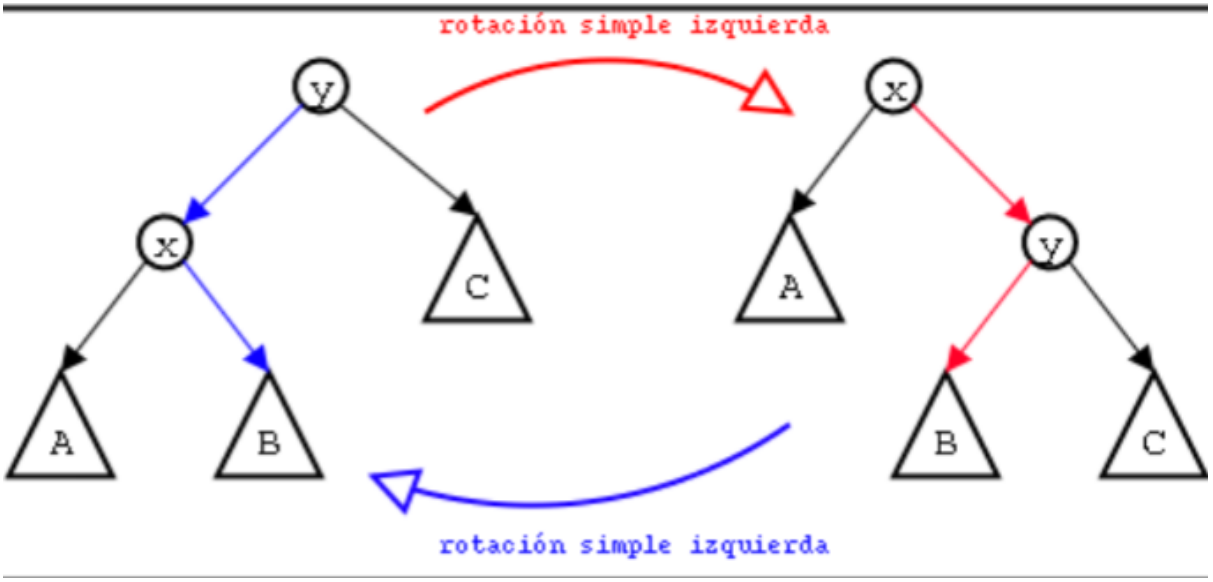
Cada nodo de un árbol AVL tiene un valor de -1, 0 o 1 que es su factor de balanceo y representa las alturas de las diferencias de alturas de los subárboles de ese nodo. Cuando el valor es 0 significa que las alturas son iguales, 1 que la altura del subárbol derecho es superior a la del izquierdo y -1 que el izquierdo es mayor al derecho.

Inserción

```
ALGORITMO INSERTARAUX ENTRADA/SALIDA I : Iterador; Crece: Integer;
c : Item ;
VAR CreceIz, CreceDe : Integer ; B : Arbol ;
METODO
    si EsVacioArbIt ( I ) entonces
        B = Enraizar ( c ) ; Mover ( I, B ) ; Crece = TRUE ; sino
        Crece = CreceIz = CreceDe = FALSE;
        si ( c < Obtener ( I ) ) entonces
        INSERTARAUX ( HijoIzq ( I ), c, CreceIz ) ;
        Crece = CreceIz ;
        sino si ( c > Obtener ( I ) ) entonces
        INSERTARAUX ( HijoDer ( I ), c, CreceDe ) ;
        Crece = CreceDe ;
      fsi
    fsi
    si Crece entonces
      caso de:
        1) ( CreceIz y FE ( I ) = 1 ) ó ( CreceDe y FE ( I ) = -1 ) : Crece =
```

```
            FALSE ; FE ( I ) = 0 ;
                2) CreceIz y FE ( I ) = 0 : FE ( I ) = -1 ;
                3) CreceDe y FE ( I ) = 0 : FE ( I ) = 1 ;
                4) CreceIz y FE ( I ) = -1 : EquilibrarIzquierda ( I, Crece ) ;
                5 ) CreceDe y FE ( I ) = 1 : EquilibrarDerecha ( I, Crece ) ;
            fcaso
        fsi
    fsi
fMETOD
```

Rotaciones



Árbol B+

Un árbol B+ es un árbol de búsqueda balanceado implementado comúnmente en bases de datos.

Esta estructura se mantiene perfectamente balanceada ya que todas las hojas se encuentran a la misma profundidad. El árbol B+ puede ser de orden m, donde m es la cantidad de nodos hijos que cada nodo puede poseer. Dentro de cada nodo interno pueden existir entre m y m*2 entradas de datos, exceptuando la raíz que puede poseer una sola llave si así se desea. Los datos están indexados en el disco y siempre se apunta a ellos desde las hojas.

Propiedades

- Solo las hojas apuntan a datos.
- Pueden existir llaves duplicadas.
- Todas las hojas se encuentran al mismo nivel
- El proceso de eliminación es sencillo ya que la data se encuentra en las hojas
- Cada nodo contiene solo llaves(no llaves-datos).
- El proceso de búsqueda puede ser más eficiente que el de un árbol binario, esto se da por la alta cantidad de hijos dentro de cada nodo significa que se requieren menos operaciones.

Algoritmo de Búsqueda

- **Función de llamado:**

```
function buscar(k) is
    return busqueda_Arbol(k, root)
```

- **Función recursiva:**

```
func: busqueda_Arbol(k, nodo) is
   if nodo is hoja entonces
      return nodo
   switch k do
   case k ≤ k_0
      return busqueda_Arbol(k, p_0)
   case k_i < k ≤ k_{i+1}
      return busqueda_Arbol(k, p_{i+1})
   case k_d < k
      return busqueda_Arbol(k, p_{d})
```

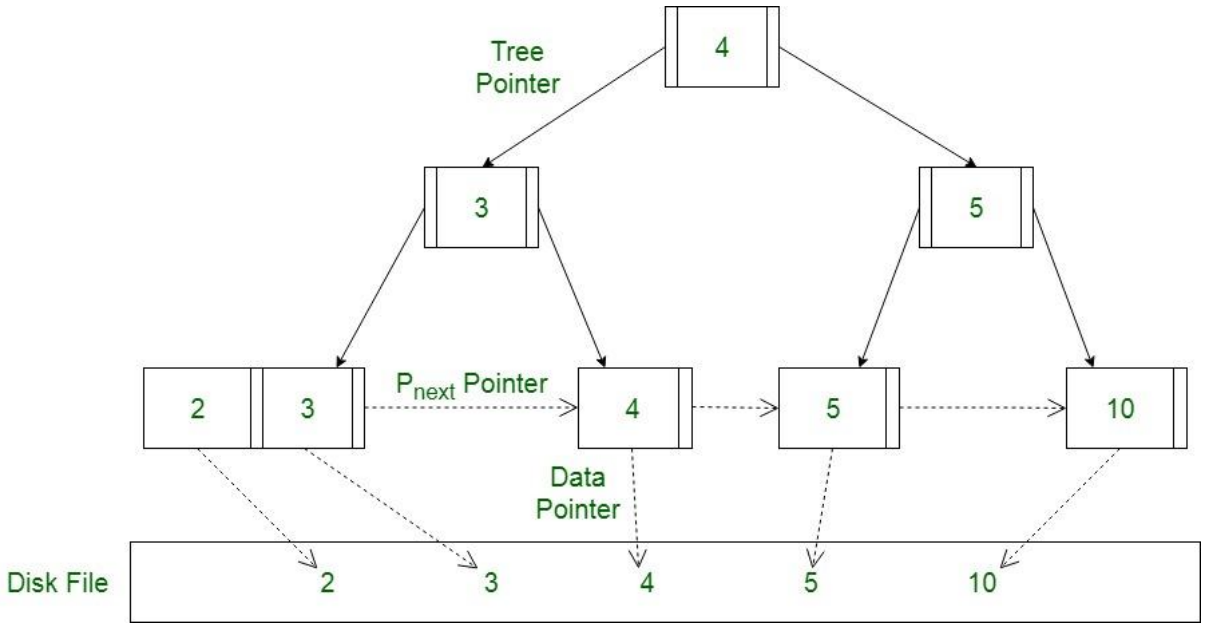
- **Inserción**

Buscar la hoja correspondiente según la llave. Dentro de la hoja seguir la secuencia y si no hay desbordamiento se inserta la llave.
En caso de que exista desbordamiento en una hoja se siguen los siguientes pasos:

1. Separar la hoja en dos nodos.
2. El primer nodo va a contener los valores de máximo((m-1)/2) y el segundo nodo los valores restantes
3. Se copia el menor valor del segundo nodo y se inserta en el nodo padre.

En caso de desbordamiento en un nodo-no hoja se siguen los siguientes pasos:

1. Separar el nodo en dos nodos.
2. El primer nodo contiene los valores de máximo(m/2)-1.
3. Se copia el valor más pequeño de los restantes al nodo padre.
4. El segundo nodo contiene los valores restantes.



Árbol Rojo-Negro

Un árbol Rojo-Negro es una representación en árbol binario de un árbol 2-3-4. Los hijos de un nodo en un árbol Rojo-Negro son de dos tipos: Rojos y Negros. Si el hijo ya existía en el árbol 2-3-4 original será Negro, sino será Rojo.

Es un árbol binario estricto en el que sus nodos nulos se tienen en cuenta en la definición de las operaciones todo nodo hoja es nulo. Cada nodo tiene estado rojo o negro. Los nodos hoja (nulos) son negros. La raíz es negra, esto solo para simplificar algunas operaciones.

Propiedades

- Es un árbol binario de búsqueda
- Cada camino desde la raíz hasta las hojas tiene el mismo número de hijos negros (esto es debido a que todos los nodos externos en un árbol 2-3-4 están en el mismo nivel y los hijos negros representan los hijos originales)
- Ningún camino desde la raíz a las hojas tiene dos o más hijos rojos consecutivos
- Cambiar un nodo de rojo a negro no afecta a la condición 1, pero afecta a la

condición 2 (altura negra se incrementa en todos los nodos ascendientes).

- Cambiar un nodo de negro a rojo puede afectar a la condición 1 (si el padre o alguno de los hijos es rojo) y también a la condición 2 (altura negra se decrementa en todos los nodos ascendientes).
- Si como resultado de una operación la raíz pasa a ser rojo, se puede cambiar a negro directamente sin afectar a las condiciones.
- Borrar un nodo rojo no afecta a las condiciones, pero borrar un nodo negro sí (altura negra decrece en los ascendientes).

Se necesitan cumplir las siguiente condiciones para un árbol rojo y negro:

1. Un nodo rojo tiene dos hijos negros.
2. Todo camino de la raíz a cualquier hoja pasa por el mismo número de nodos negros.

Altura negra de un nodo (denotada por la letra H, para diferenciarla de la altura normal, h). La altura negra de un nodo es igual a la altura del nodo cuando sólo se tienen en cuenta los nodos negros del subárbol cuya raíz es el nodo. La altura negra de un nodo se puede calcular de la siguiente forma:

$$H(n) = \left\{ \begin{array}{ll} máx\,(H(n.izq), H(n.der))+1 & si\ n\ es\ negro \\ máx\,(H(n.izq), H(n.der)) & si\ n\ es\ rojo \end{array} \right\}$$

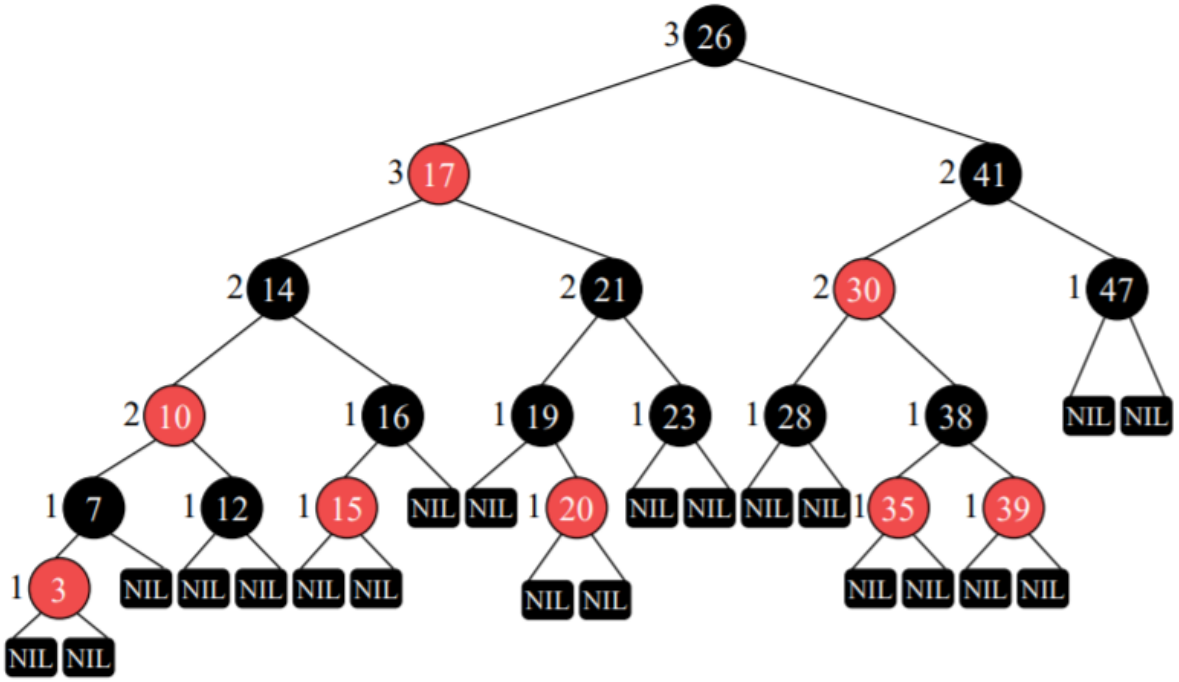
La segunda condición de un árbol rojo-negro se puede expresar de otra forma, tal como que para todo nodo interno (que no sea hoja, es decir nulo), la altura negra de su hijo izquierdo es igual a la altura negra de su hijo derecho.

Inserción de un nodo

La inserción de un nodo se realiza igual que en un árbol binario de búsqueda al principio. Al nuevo nodo se le da el color rojo. De ésta manera no se viola la segunda condición (altura negra), aunque se puede violar la primera (el padre del nodo insertado puede ser rojo).

Puede existir un caso trivial: Si el padre del nodo insertado es negro, no se realiza ningún ajuste (árbol correcto).

En caso contrario se entra en un bucle donde x representa el nodo que se está comprobando. x es un nodo rojo, y los casos se refieren a situaciones en que su padre existe y es también rojo: Cuando el padre no exista, simplemente se cambia el color de x (que es la raíz) a negro y se termina el bucle, y si el padre existe y es negro se termina directamente el bucle. Dependiendo del caso, se realizan una serie de operaciones y o bien continuará el ciclo (x pasa a ser otro nodo) o bien se terminará. En cada caso se supone que el nodo que se comprueba es rojo, su padre existe y es rojo, y su abuelo existe (no puede ser raíz un nodo rojo). El nodo hermano del padre se denomina tío.



Diferencia entre los árboles

Árbol AVL	Árbol B	Arbol B+	Arbol Rojo-Negro
Cada árbol AVL es también un árbol binario porque el árbol AVL también tiene los dos hijos máximos.	Sus nodos no pueden tener más de una clave y más de dos hijos, según el valor de m . En el árbol B, los datos se especifican en un orden clasificado con valores más bajos en el subárbol izquierdo y valores más altos en el subárbol derecho.	Elimina el inconveniente del árbol B utilizado para la indexación al almacenar punteros de datos sólo en los nodos de las hojas del árbol.	sus nodos nulos se tienen en cuenta en la definición de las operaciones todo nodo hoja es nulo. Cada nodo tiene estado rojo o negro. Los nodos hoja (nulos) son negros.
Los árboles AVL proporcionan una búsqueda eficiente ya que es un árbol estrictamente equilibrado.	La búsqueda es ineficaz en BST cuando hay una gran cantidad de nodos disponibles en el árbol porque la altura no está equilibrada.	Todas las claves están en los nodos hoja, por lo que la búsqueda es más rápida y precisa.	El árbol rojo negro no proporciona una búsqueda eficiente ya que los árboles rojos negros están aproximadamente equilibrados.
La inserción y eliminación son complejas en el árbol AVL ya que requieren múltiples rotaciones para equilibrar el árbol.	La eliminación de nodos internos es muy lenta y un proceso que requiere mucho tiempo, ya que también se debe considerar el elemento secundario de la clave eliminada.	La eliminación en el árbol B + es muy rápida porque todos los registros se almacenan en los nodos hoja, por lo que no se tiene que considerar al hijo del nodo.	La inserción y eliminación son más fáciles en el árbol rojo negro, ya que requiere menos rotaciones para equilibrar el árbol.
	En el árbol B, los nodos de hojas no están vinculados entre sí.	En el árbol B +, los nodos hoja están vinculados entre sí para proporcionar el acceso secuencial.	
Cada nodo tiene un factor de equilibrio en el árbol AVL cuyo valor puede ser 1, 0 o -1. Requiere espacio adicional para almacenar el factor de balance por nodo.			No contiene ningún factor de equilibrio. Almacena solo un bit de información que denota el color rojo o negro del nodo.
	En el árbol B, todas las claves y registros se almacenan tanto en los nodos internos como en los de hoja.	En el árbol B +, las claves son los índices almacenados en los nodos internos y los registros se almacenan en los nodos hoja	

Referencias

1. *Arboles B - Arboles B*. (s. f.). Google Sites. Recuperado 19 de octubre de 2021, de https://sites.google.com/site/clasearbolesb/arboles-b

2. *El árbol de búsqueda equilibrado (B-Tree) en las bases de datos SQL*. (s. f.). USE THE INDEX, LUKE. Recuperado 19 de octubre de 2021, de https://use-the-index-luke.com/es/sql/i%CC%81ndice-anatomi%CC%81a/b-tree

3. Universidad de Valladolid. (s.f). Árbol Rojinegro. Recuperado 29 de octubre de 2021, de https://www.infor.uva.es/~cvaca/asigs/doceda/rojonegro.pdf

4. Campos, J. (s.f). Técnicas Avanzadas de Programación. Universidad de Zaragoza. Recuperado 29 de octubre de 2021, de http://webdiis.unizar.es/asignaturas/TAP/material/1.3.rojinegros.pdf

5. GeeksforGeeks. (2020, April 9). *Introduction of B+ Tree*. Recuperado 29 de Octubre de 2021, de https://www.geeksforgeeks.org/introduction-of-b-tree/

6. GeeksforGeeks. (2020b, July 8). *Insertion in a B+ tree*. Recuperado 29 de octubre de 202,1 de https://www.geeksforgeeks.org/insertion-in-a-b-tree/ http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf https://rua.ua.es/dspace/bitstream/10045/16037/4/ped-09_10-tema3_2.pdf

7. Escuela Computación. (2019). Árboles Balanceados AVL. Universidad Don Bosco. Recuperado 1 de noviembre de 2021, de https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-con-estructuras-de-datos/2019/i/guia-7.pdf

8. *Red Black Tree vs AVL tree - javatpoint*. (s. f.). Www.Javatpoint.Com. Recuperado 1 de noviembre de 2021, de

https://www.javatpoint.com/red-black-tree-vs-avl-tree

9. Gurin, S. (2004). Árboles AVL. TDLP. Recuperado 1 de noviembre de 2021, de http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf

10. *B tree vs B+ tree - javatpoint*. (s. f.). Www.Javatpoint.Com. Recuperado 1 de noviembre de 2021, de https://www.javatpoint.com/b-tree-vs-bplus-tree

11. *Binary Search tree vs AVL tree - javatpoint*. (s. f.). Www.Javatpoint.Com. Recuperado 1 de noviembre de 2021, de https://www.javatpoint.com/binary-search-tree-vs-avl-tree