Árboles B+

Bruno S. López Fernandez *

* *Instituto Tecnológico Autónomo de México, matrícula 153318*
(Profesor: Carlos Fernando Esponda Darlington).

Abstract: Investigación acerca de las estructuras de datos conocidas como árboles B+, sus diferencias con los árboles B, qué son, sus bondades y algunas desventajas. Además dar ejemplos de su utilización actualmente.

Keywords: programación, estructuras, datos, árboles

1. INTRODUCCIÓN

Este documento pretende explicar un poco acerca del funcionamiento de las estructuras de datos conocidas como árboles B+, tratándolas como un caso especial de los árboles B, así como exponer algunos de sus usos actuales en la industria.

2. ÁRBOLES B

Los árboles-B son una clase de árboles de búsqueda de multicamino, en este tipo de árboles se especifica el número de hijos de cada nodo como el orden del árbol. Además tienen las siguientes propiedades, según Chase y Lewis (2014):

- La raíz del árbol tiene almenos dos sub-árboles a menos que sea una hoja.
- Cada nodo interno que no es raíz contiene $k-1$ elementos y k hijos, donde $\lceil m/2 \rceil \leq k \leq m$
- Cada hoja n contiene $k-1$, donde $\lceil m/2 \rceil \leq k \leq m$
- Todas las hojas se encuentran al mismo nivel

La creación de estas estructuras de datos se debe a la presmisa de que la memoria principal no es lo suficientemente grande como para alojar grandes colecciones, por lo que se tendría que utilizar la memoria del disco o alguna otra memoria externa, lo cual muy probablemente aumentaría el tiempo de procesamiento, esto debido a que la complejidad del tiempo ya no se basa solo en el numero de comparaciones para encontrar el elemento, sino tambien el tiempo de acceso a esta memoria de apoyo, además de cuántos accesos se realizar a ésta.

Añadir este tiempo puede ser muy costoso para el acceso a una colección, por lo que los árboles-B fueron desarroollados con el objetivo de minimizar la cantidad de veces que se necesita accesar la memoria secundaria. Regularmente están hechos con el objetivo que un nodo sea del mismo tamaño que un bloque en la memoria secundaria. De esta manera se obtiene la mayor cantidad de información por cada acceso a esta memoria.

En comparación con los árboles binarios, este tipo de árboles puede tener muchos más elementos por nodo.

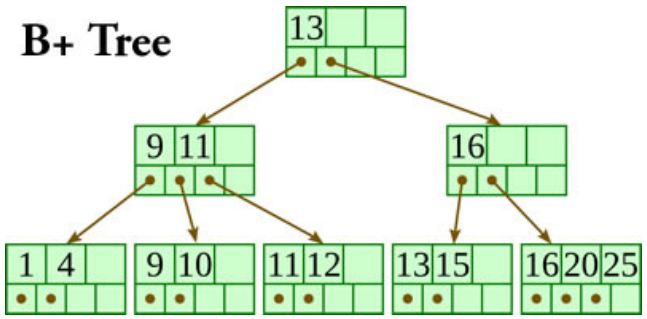

Fig. 1. Ejemplo de árbol B+

3. ÁRBOLES B+

Uno de los principales problemas con los árboles B puede ser el acceso secuencial, se puede utilizar el recorrido inorden, pero no se estarían aprovechando las bondades de este tipo de árboles, esto se debe a que en este tipo de árboles cada elemento aparece una sola vez, sin importar si es en un nodo interno o en un hoja. En cambio en los árboles B+ cada elemento aparece en una hoja, sin importar si aparece en un nodo interno, esto quiere decir que los elementos que aparezcan como nodos internos o como raices, apareceran tambien como sucesor inorden de su posición, por lo que los nodos que no son hojas, solo sirven como índices hacia los datos..

Además de esto, otra de las diferencias clave es que cada hoja tiene un apuntador a la siguiente y gracias a esto los árboles B+ brindan acceso a la estructura del árbol de manera secuencial por medio de una lista ligada de hojas.

3.1 Inserción

(1) Ubicarse en la raíz del árbol
(2) Ver si donde se encuentra es hoja.
  (a) Si es hoja
    (i) Ver si queda espacio en el nodo para insertar el dato, en caso de que haya lugar, insertarlo en el lugar correspondiente.
    (ii) Si no hay lugar en el nodo, dividir la hoja en dos, subir una copia del dato en medio a la raiz, si la raiz se encuentra llena repetir este paso hasta donde sea necesario.
  (b) Si no es hoja

(i) Comparar el elemento con cada uno de los apuntadores para encontrar el hijo del nodo correspondiente para proseguir la búsqueda. Regresar a paso 2.

3.2 Eliminación

(1) Remover elemento y en caso de que exista su referencia de la raíz.
  (a) Si el numero de elementos es mayor que $[m/2]$, entonces no se modifican las raices u hojas vecinas.
  (b) Si el número de elementos es menor que $[m/2]$, entonces deben redistribuirse los elementos de ese nodo, tanto en otras hojas como en el índice.
    (i) Si el hermano mayor o menor (nodo a la derecha o noddo a la izquierda) tiene numero de elementos mayor que $[m/2]$, y se pueden redistribuir los elementos entre sus hermanos hacerlo, cuidando reparar el índice en el nivel superior.
    (ii) Si los hermanos (nodo a la derecha o noddo a la izquierda) apenas cumplen $[m/2]$ hay que unir el nodo con su hermano, pero si no es una hoja, se debe incorporar el índice que se tenía antes en esta unión. En cualquier caso se deberá aplicar este algoritmo al nodo raiz que antes separaba lo que se unió, con la excepción que sea la raíz del árbol y en dado caso la unión se convierte en la nueva raíz

3.3 Usos y aplicaciones

Debido a las ventajas descritas anteriormente, principalmente que el último nivel del árbol funciona como una lista ordenada y ligada, los árboles B+ son utilizados en distintas aplicaciones, pero principalmente en la búsqueda de archivos asegurando que la distancia de la raiz a cada uno sea la misma, y por lo tanto el tiempo de obtención de estos sea igual.

La principal aplicación de los árboles B+ es el indexado de archivos, actualmente la mayoría de los índices de archivos en las computadoras o en dispositivos inteligentes utilizan esta estructura debido a que aseguran el tiempo de acceso y como se comentó anteriormente minimizan la cantidad de veces que se accesa una memoria secundaria. También se utiliza con el mismo fin en las memorias flash.

También se aplica esta estructura en los DBMS (Data Base Management System), esto debido a su gran utilidad cuando se utilizan grandes cantidades de datos. Distintos servidores de bases de datos como SQL utilizan este tipo de estricturas de datos para el manejo de información.

Otra aplicación que ha crecido mucho en los últimos años es el almacenamiento y procesamiento de información en la nube, esto debido a que se necesitan algortimos eficientes de busqueda, inserción y eliminación para poder satisfacer las necesidades de la cantidad creciente de usuarios y eficientar los procesos.

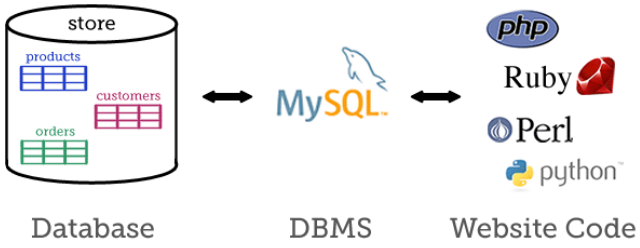

Fig. 2. Ilustración de un Data Base Management System

4. CONCLUSIÓN

Se puede apreciar como estas estructuras de datos son realmente útiles para grandes cantidades de datos y considero que al ritmo que está creciendo la generación de datos, con cada vez más sensores que recuperan más información, más datos que los usuarios están dispuestos a proporcionar y otros que proporcionan sin darse cuenta, cada vez serán más utilizadas estas estructuras para el manejo de la nueva era de la información

REFERENCES

J. Chase and J. Lewis. (2014). *Java Software Structures: Designing and Using Data Structures*. Estados Unidos: Pearson.

Hendrix College. (s.f.). *B+-trees*. Recuperado de: www.cburch.com/cs/340/reading/btree/index.html

Tutorial Cup (s.f.). *Concepts of B+ Tree and Extensions - B+ and B Tree index files in DBMS*. Recuperado de: https://www.tutorialcup.com/dbms/b-tree.htm

P. Kieseberg, S. Schrittwieser, L. Morgan, M. Mulazzani, M. Huber and E. Weippl. (s.f.) *Using the Structure of B+-Trees for Enhancing Logging Mechanisms of Databases*. Recuperado de: www.sba-research.org/.../iiWAS2011_133_Short_Kieseberg.pdf