Estructura de datos para listar contenido de un directorio.

Kevin Arley Parra
Universidad EAFIT
Colombia
kaparrah@eafit.edu.co

Daniel Alejandro Mesa Arango
Universidad EAFIT
Colombia
damesaa@eafit.edu.co

Mauricio Toro
Universidad EAFIT
Colombia
mtorobe@eafit.edu.co

RESUMEN

El problema se resume a un sistema que pueda listar los contenidos de un directorios de una forma eficiente asi estos sean demasiados, su importancia radica en la gran cantidad de datos que se manejan en la actualidad en un mundo digitalizado y donde se busca dar la mejor atención y producto a los usuarios, algunos problemas que pueden relacionar son por ejemplo, que tan eficiente es una estructura de datos es la manera en la cual se comparan datos para dar una respuesta como buscar una placa en muchas, donde se debe de usar una estructura de datos adecuada para que el programa no tarde demasiado en encontrar la que se quiere.

1. INTRODUCCIÓN

Hoy en un mundo tan digitalizado se hace necesario tener los datos almacenados y que puedan ser consultados en cualquier momento por un usuario, sin embargo con el creciente volumen de datos muchas de las tecnologías y estructuras de datos han quedado obsoletas causando que los tiempos de espera para guardar algo en determinado subdirectorio o consultar algo sea mayor y consuma muchos recursos de la máquina, por esto han surgido nuevas soluciones que hacen mucho mejor el trabajo, varias son las soluciones que se pueden implementar las cuales traen sus ventajas y desventajas, sistemas como la búsqueda que ofrece Windows al usuario son ejemplos de implementación de una estructuras de datos, el usado por Microsoft es NTFS, implementa los árboles-B[1]. Se muestra entonces cuatro posibles soluciones y una por la cual optar considerándola la mejor para el caso.

2. PROBLEMA

El problema consiste en listar los archivos de un directorio de manera eficiente, de manera que se pueda encontrar los archivos y subdirectorios fácilmente.

sin que el usuario tenga que esperar mucho para que se encuentre lo que desea.

3. TRABAJOS RELACIONADOS

Para el actual informe se realizó una investigación sobre las diferentes estructuras de datos que pueden dar una solución efectiva al problema planteado, a continuación, una descripción de cada una de ellas y en la Figura 7 se encuentran varias características de complejidad de estos árboles y algunos otros.

3.1. Árbol rojo-negro [2]

Concretamente es un árbol binario de búsqueda equilibrado, la estructura original fue creada por Rudolf Bayer en 1972, es complejo, pero tiene un buen peor caso de tiempo de ejecución para sus operaciones y es eficiente. Puede buscar, insertar y borrar en un tiempo O(log n), donde n es el número de elementos del árbol.

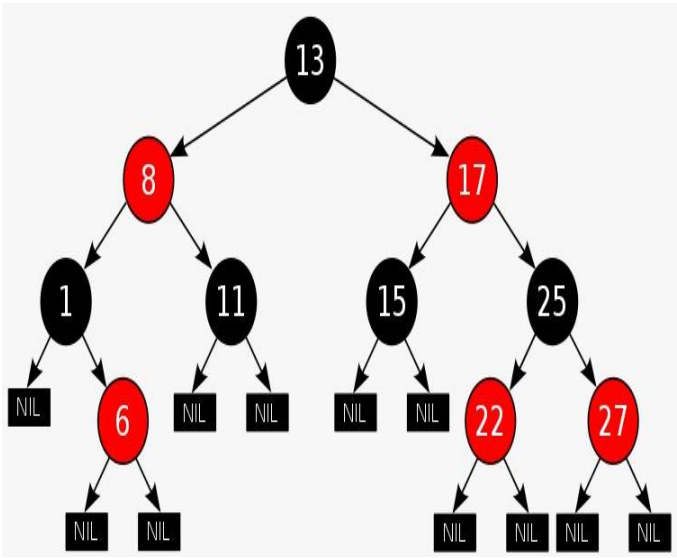

Figura 1

Además de los requisitos impuestos a los árboles binarios de búsqueda convencionales, se deben satisfacer las siguientes reglas para tener un árbol rojo-negro válido:

1. Todo nodo es o bien rojo o bien negro.
2. La raíz es negra.
3. Todas las hojas (NULL) son negras.
4. Todo nodo rojo debe tener dos nodos hijos negros.
5. Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

Estas reglas producen una regla crucial para los árboles rojo-negro: el camino más largo desde la raíz hasta una hoja no es más largo que dos veces el camino más corto desde la raíz a una hoja. El resultado es que dicho árbol está aproximadamente equilibrado.

Dado que las operaciones básicas como insertar, borrar y encontrar valores tienen un peor tiempo de ejecución proporcional a la altura del árbol, esta cota superior de la altura permite a los árboles rojo-negro ser eficientes en el peor caso, a diferencia de los árboles binarios de búsqueda.

Para comprobarlo basta ver que ningún camino puede tener dos nodos rojos seguidos debido a la propiedad 4. El camino más corto posible tiene todos sus nodos negros, y el más largo alterna entre nodos rojos y negros. Dado que todos los caminos máximos tienen el mismo número de nodos negros por la propiedad 5, no hay ningún camino que pueda tener longitud mayor que el doble de la longitud de otro camino.

3.2. Árbol-B [3]

Los B-árboles surgieron en 1972 creados por R.Bayer y E.McCreight. El problema original comienza con la necesidad de mantener índices en almacenamiento externo para acceso a bases de datos, es decir, con el grave problema de la lentitud de estos dispositivos se pretende aprovechar la gran capacidad de almacenamiento para mantener una cantidad de información muy alta organizada de forma que el acceso a una clave sea lo más rápido posible. Los B-árboles son árboles cuyos nodos pueden tener un número múltiple de hijos tal como muestra el esquema de uno de ellos en la figura 2.

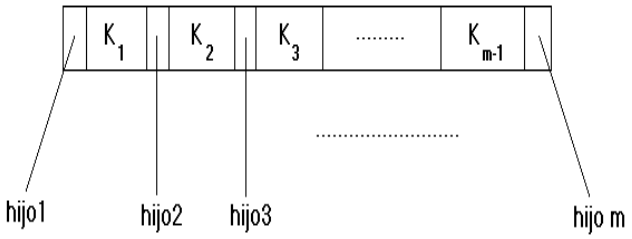

Figura 2

Como se puede observar en la figura 2, un B-árbol se dice que es de orden m si sus nodos pueden contener hasta un máximo de m hijos. En la literatura también aparece que si un árbol es de orden m significa que el mínimo número de hijos que puede tener es m+1(m claves).

1. Seleccionar como nodo actual la raíz del árbol.

2. Comprobar si la clave se encuentra en el nodo actual:

    1. Si la clave está, fin.

2. Si la clave no está:

    - Si estamos en una hoja, no se encuentra la clave. Fin.

    - Si no estamos en una hoja, hacer nodo actual igual al hijo que corresponde según el valor de la clave a buscar y los valores de las claves del nodo actual (i buscamos la clave K en un nodo con *n* claves: el hijo izquierdo si $K<K_1$,el hijo derecho si $K>K_n$ y el hijo i-ésimo si $K_i<K<K_{i+1}$)y volver al segundo paso.

Breve explicación de la inserción

Las inserciones se hacen en los nodos hoja. **[4]**

1. Realizando una búsqueda en el árbol, se halla el nodo hoja en el cual debería ubicarse el nuevo elemento.

2. Si el nodo hoja tiene menos elementos que el máximo número de elementos legales, entonces hay lugar para uno más. Inserte el nuevo elemento en el nodo, respetando el orden de los elementos.

3. De otra forma, el nodo debe ser dividido en dos nodos. La división se realiza de la siguiente manera:

1. Se escoge el valor medio entre los elementos del nodo y el nuevo elemento.

2. Los valores menores que el valor medio se colocan en el nuevo nodo izquierdo, y los valores mayores que el valor medio se colocan en el nuevo nodo derecho; el valor medio actúa como valor separador.

3. El valor separador se debe colocar en el nodo padre, lo que puede provocar que el padre sea dividido en dos, y así sucesivamente.

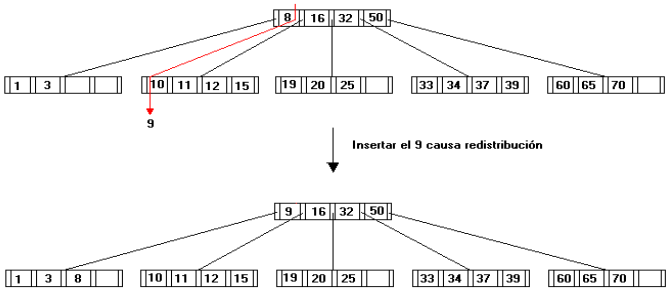

Figura 3

3.3. Árbol AVL [5]

Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1. La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis). Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz. Recordamos también que el tiempo de las operaciones sobre un árbol binario de búsqueda son O(log n) promedio, pero el peor caso es O(n), donde n es el número de elementos.

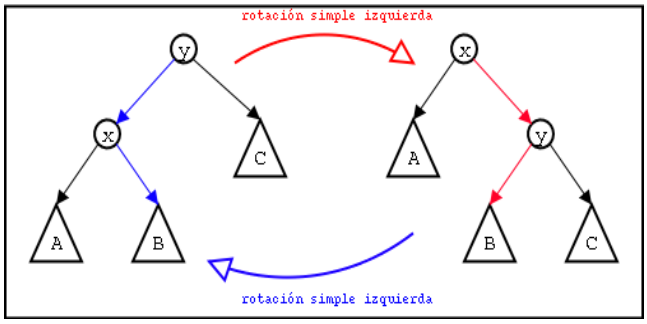

Figura 4

El árbol de Figure 4. es un árbol de búsqueda, se debe cumplir x < y y además todos los nodos del subárbol A deben ser menores que x y y; todos los nodos del subárbol B deben ser mayores que x pero menores que y; y todos los nodos del subárbol C deben ser mayores que y y por lo tanto que x.

En Figure 4 se ha modificado sencillamente el árbol. Como puede verificarse fácilmente por las desigualdades descriptas en el párrafo anterior, el nuevo árbol sigue manteniendo el orden entre sus nodos, es decir, sigue siendo un árbol binario de búsqueda. A esta transformación se le denomina rotación simple (o sencilla).

A continuación de muestra un ejemplo sencillo de un árbol AVL:

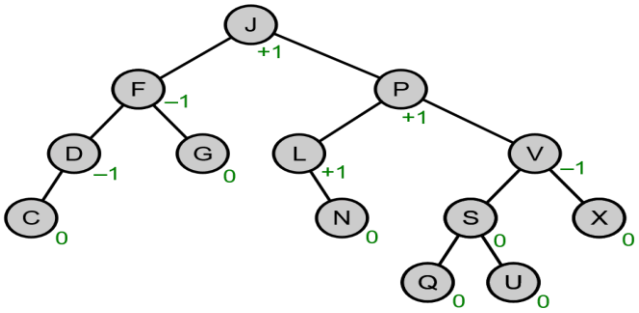

Figura 5

3.4. Árbol biselado [6]

Un Árbol biselado o Árbol Splay es un Árbol binario de búsqueda auto-balanceable, con la propiedad adicional de que a los elementos accedidos recientemente se accederá más rápidamente en accesos posteriores. Realiza operaciones básicas como pueden ser la inserción, la búsqueda y el borrado en un tiempo del orden de O(log n). Para muchas secuencias no uniformes de operaciones, el árbol biselado se comporta mejor que otros árboles de búsqueda, incluso cuando el patrón específico de la secuencia es desconocido. Esta estructura de datos fue inventada por Robert Tarjan y Daniel Sleator.

Todas las operaciones normales de un árbol binario de búsqueda son combinadas con una operación básica, llamada biselación. Esta operación consiste en reorganizar el árbol para un cierto elemento, colocando éste en la raíz. Una manera de hacerlo es realizando primero una búsqueda binaria en el árbol para encontrar el elemento en cuestión y, a continuación, usar rotaciones de árboles de una manera específica para traer el elemento a la cima. Alternativamente, un algoritmo "de arriba abajo" puede combinar la búsqueda y la reorganización del árbol en una sola fase.

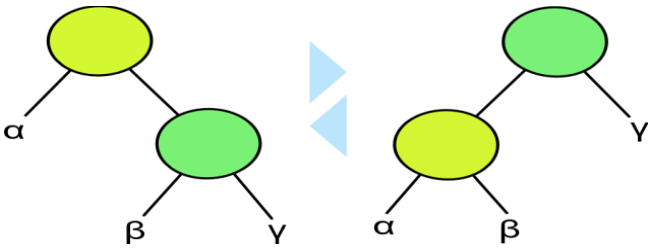

Figura 6

4. ELECCIÓN DE LA ESTRUCTURA DE DATOS

Elegimos la estructura de datos B-tree para empezar, ya que nos parece que puede servir, y no decidimos intentar crear una nueva ya que como dice un principio del libro 201 principles of software development [7], “no invente la rueda”. Sin embargo, dejamos abiertas las posibilidades de cambiar más delante de estructura o incluso tratar de implementar alguna si vemos que la elegida no funciona adecuadamente. A continuación, exponemos las razones por las cuales elegimos la estructura B-tree.

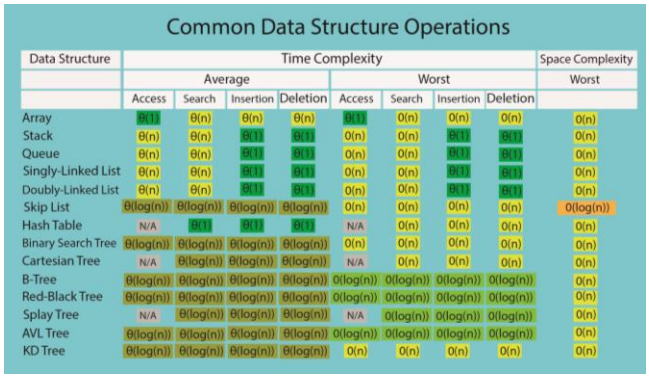

Figura 7

Basados en la figura 7, podemos ver que hay varias estructuras cuya complejidad es bastante notable, entre ellas está la B-tree, por eso decidimos elegirla, por su nivel de complejidad y porque tiene muchas características, ya que otras estructuras en la tabla incluso tenían mejor complejidad en algunos aspectos que la B-tree pero en otros su complejidad no era tan buena. Esa es otra razón por la cual escogimos el B-tree, ya que está bastante nivelado en todos los aspectos.

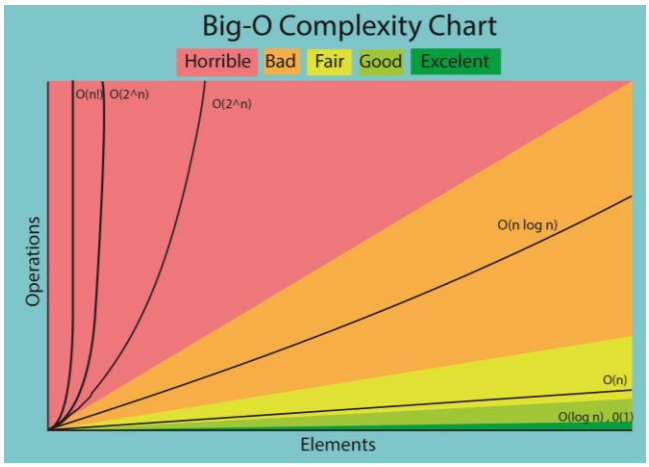

Figura 8

La figura 8 nos da una idea bastante aceptable para entender la complejidad y nos apoyamos en ésta también para elegir la estructura de árbol B-tree.

REFERENCIAS

[1] NTFS, 2017. Consultado el 12 de agosto de 2017, Wikipedia, La enciclopedia libre. Recuperado de: https://es.wikipedia.org/wiki/NTFS

[2]Árbol rojo-negro, 2017. Consultado el 12 de agosto de 2017, Wikipedia, La enciclopedia libre. Recuperado de: https://es.wikipedia.org/w/index.php?title=%C3%81rbol_rojo-negro&oldid=99055928

[3]B-Árbol, Tutor de Estructuras de Datos Interactivo: sección: inserción en un b-árbol, Exposito Lopez Daniel, Abraham García Soto, Martin Gomez Antonio Jose, director proyecto: Joaquín Fernández Valdivia, Universidad de granada: consultado el 12 de agosto de 2017, recuperado de: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B.htm

[4]Árbol-B, 2017. Consultado el 12 de agosto de 2017, Wikipedia, La enciclopedia libre. Recuperado de: https://es.wikipedia.org/w/index.php?title=%C3%81rbol-B&oldid=100262033

[5] Gurin, S. Árboles AVL. Consultado el 11 de agosto de 2017, recuperado: http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf

[6] Árbol biselado, 2017. Consultado el 11 de agosto de 2017, Wikipedia, La enciclopedia libre. Recuperado de: https://es.wikipedia.org/w/index.php?title=%C3%81rbol_biselado&oldid=98936910

[7] Alan M. Davis. 1995. 201 Principles of Software Development. McGraw-Hill, Inc., New York, NY, USA.

Programa utilizado para vectorizar las imágenes: Adobe Illustrator. http://www.adobe.com/Illustrator

Ilustraciones:

Figura 1:

https://en.wikipedia.org/wiki/Red%E2%80%93black_tree#/media/File:Red-black_tree_example.svg

Figura 2, 3:

http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B.htm

Figura 4:

https://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/doc-programacion-arboles-avl/htmls/rotacion_simple.html

Figura 5:

https://en.wikipedia.org/wiki/AVL_tree

Figura 6:

https://es.wikipedia.org/wiki/%C3%81rbol_biselado#/media/File:BinaryTreeRotations.svg

Figura 7, 8:

http://bigocheatsheet.com/