ESTRUCTURA DE DATOS PARA UBICAR DIRECTORIOS Y ARCHIVOS EN UN ALMACENAMIENTO

Luis Carlos Rodríguez Zúñiga
Universidad Eafit
Colombia
lcrodriguz@eafit.edu.co

Laura Sánchez Córdoba
Universidad Eafit
Colombia
lsanchezc@eafit.edu.co

Manuela Valencia Toro
Universidad Eafit
Colombia
mvalenciat@eafit.edu.co

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

RESUMEN

Ante el creciente avance de la tecnología actualmente, surge la posibilidad de tener acceso a una cantidad exorbitante de información, por ende, resulta completamente necesario generar herramientas que permitan un uso fácil y óptimo de todos estos datos para dar solución a algunos problemas como: organizar tanta información, cuál es la cantidad de información que se permite almacenar, cómo hallar un archivo específico en medio de la información almacenada, etc. Para suplir esta necesidad se utilizará como herramienta una estructura de datos que almacene la información, un Árbol Rojo-Negro (T*reeMap* de Java), ya que esta estructura garantiza que los datos puedan ser almacenados y buscados de manera fácil y eficiente, dadas las propiedades que tiene. Con la implementación de éste se obtuvieron muy buenos resultados respecto al tiempo en las operaciones, y a pesar de que a diferencia de otras estructuras, tiene un costo mayor de memoria, los resultados nos indican que es conveniente usar este tipo de estructura para la organización y búsqueda de datos.

Palabras clave

Estructura de datos, búsqueda, arboles binarios, archivos, árbol rojo-negro, complejidad, datos.

Palabras clave de la clasificación de la ACM

Software and its engineering → Software creation and management → Search-based software engineering

Theory of computation → Design and analysis of algorithms → Data structures design and analysis → Sorting and searching

Information systems → Data management systems → Data structures → Data access methods

1. INTRODUCCIÓN

En la presente era de la información, y gracias a avances tecnológicos que van desde la creación de las computadoras, el nacimiento de la internet, y más; los seres humanos tenemos la facilidad de acceder a grandes cantidades de información (datos) y guardarlos para posteriormente usarlos en caso de que sea necesario, pero ¿Cómo evitar que la búsqueda de un archivo necesario no sea difícil para nosotros? ¿Cómo encontrar un solo archivo en el mar de información en el que podemos estar inmersos? Para responder a estas preguntas, surgen lo que hoy en día se conocen como Estructuras de datos, las cuales son un conjunto datos que tienen una relación estructural promoviendo así un mejor procesamiento de los datos[1] para en nuestro caso, ubicar archivos de una manera más fácil, por ende, en el presente trabajo se presentarán cuatro estructuras que permitirán solucionar nuestro problema de manejo de archivos.

2. PROBLEMA

Dada la cantidad de información que podemos tener almacenada, se requiere generar una herramienta (estructura de datos) que permita manejar de manera fácil y eficiente los directorios y archivos que tenemos almacenados en un computador, de manera que sea más sencillo encontrar la información que se pueda requerir.

3. TRABAJOS RELACIONADOS

Para realizar el proyecto se requieren algunas bases previas que nos peritan llevar a cabo la realización de nuestra propia estructura de datos, de manera que aquí se explicarán cuatro de ellas que han permitido el manejo óptimo de la gran cantidad de datos que se tengan.

3.1. Árbol rojo-negro[2]

Un árbol rojo-negro es un árbol binario de búsqueda con un bit extra de almacenamiento por nodo: su color, que puede ser ROJO o NEGRO.

Al restringir los colores de los nodos en cualquier ruta simple desde la raíz a una hoja, los árboles rojo-negro aseguran que ninguna de esas trayectorias sea más del doble que cualquier otra, de modo que el árbol esté aproximadamente equilibrado.

Cada nodo del árbol ahora contiene los atributos color, clave, izquierda, derecha. Si un hijo o el padre de un nodo no existe, el atributo de puntero correspondiente del nodo contiene el valor nulo. Consideraremos estos nulos como indicadores para hojas (nodos externos) del árbol binario de búsqueda y los nodos normales, portadores de claves como siendo nodos internos del árbol.

Un árbol rojo-negro es un árbol binario que satisface las siguientes propiedades rojo-negro:

1. Cada nodo es rojo o negro.

2. La raíz es negra.

3. Cada hoja (nula) es negra.

4. Si un nodo es rojo, entonces sus dos hijos son negros.

5. Para cada nodo, todas las rutas simples desde el nodo hasta las hojas descendentes contienen mismo número de nodos negros.

3.2. Árbol binario[3]

Es un árbol que solamente puede tener dos hijos por nodo.

Los dos hijos de cada nodo en un árbol binario se llaman el niño izquierdo y el niño derecho, que corresponde a sus posiciones cuando dibuja una imagen de un árbol, un nodo en un árbol binario no tiene necesariamente el máximo de dos niños; Puede tener solamente un niño izquierdo, o solamente un niño derecho, o no puede tener ningún niño en absoluto (en cuyo caso es una hoja).

Árbol binario de búsqueda:

La característica de definición de un árbol binario de búsqueda es la siguiente: El nodo izquierdo de un nodo debe tener una clave menor que su padre y el nodo derecho de un nodo debe tener una clave mayor o igual que su padre, esta clave indica el campo por el cual se realiza la ordenación de los datos.

3.3. Árbol-b[3]

Los árboles B son árboles de búsqueda equilibrados diseñados para funcionar bien en discos u otros dispositivos de almacenamiento secundario de acceso directo

Los árboles B son similares a los árboles rojo-negros, pero son mejores para minimizar las operaciones de E / S de disco. Muchos sistemas de bases de datos utilizar árboles B, o variantes de B-árboles, para almacenar información.

Difieren de los árboles rojo-negros en que los nodos de los árboles B pueden tener muchos hijos, de unos pocos a miles. Es decir, el "factor de ramificación" de un árbol-B puede ser bastante grande, aunque generalmente depende de las características de la unidad de disco utilizada.

La altura exacta de un árbol B puede ser considerablemente menor que la de un árbol rojo-negro, sin embargo, debido a que su factor de ramificación, y por lo tanto la base del logaritmo que expresa su altura, puede ser mucho mayor.

Cabe agregar que un árbol B que está diseñado esencialmente para memorias de acceso aleatorio.

3.4. Tablas de hash[4]

Una tabla Hash es un contenedor asociativo parecido a un diccionario el cual permite un almacenamiento y posterior recuperación eficientes de elementos (denominados valores) a partir de otros objetos, llamados claves.

Esta es un conjunto de entradas, cada una de las cuales tiene una clave única, ésta y la información que hay, son los componentes de las entradas de las tablas.

Esta estructura es bastan ante eficiente en la búsqueda de información ya que además de su clave, las tablas poseen un tiempo de recuperación contantes, es decir, no está sujeto al tamaño de la información que está en la entrada. Estas tablas están compuestas por un arreglo y una función de dispersión que almacena la información en la entrada con sus respectivas claves. En el manejo de éstas existe un riesgo de colisión, en caso de que se asigne una misma clave. Existen dos tipos de tablas de hash según se solucione la colisión:

Encadenamiento separado: Las colisiones se resuelven insertándolas en una lista. De esa forma tendríamos como estructura un vector de listas.

Direccionamiento abierto: Utilizamos un vector como representación y cuando se produzca una colisión se reasigna una clave hasta hallar un espacio.

4. La estructura: Árbol Rojo-Negro

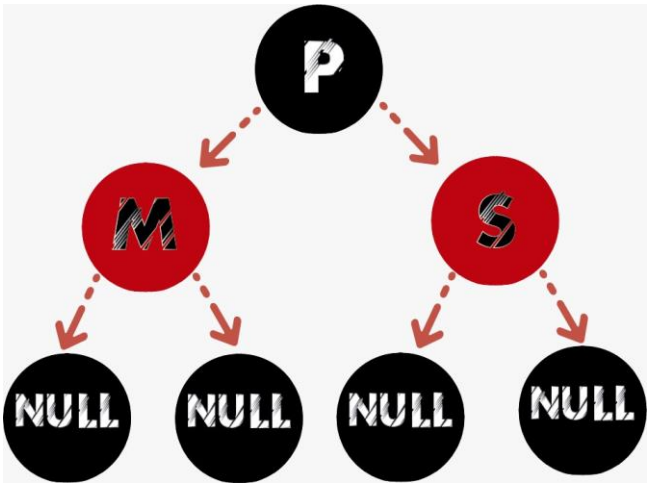

Gráfico 1: Ejemplo de la organización de los datos tipo String, de esta manera se organizan los archivos basados en el nombre, cada nodo tiene el nombre de la carpeta como llave (son los que se organizan alfabéticamente) y como valor el dato que el usuario quiera buscar (tamaño, dueño, ubicación).

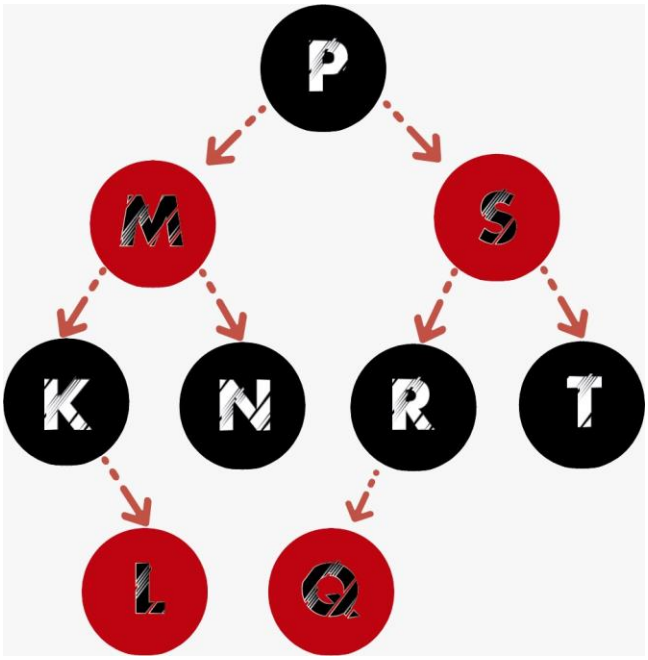

Gráfico 2: Así se van almacenado los datos y se balancea el árbol según las propiedades del árbol (ver numeral 3.1.), cada nodo puede ser archivo o carpeta.

4.1 Operaciones de la estructura[5]

Rotación: para el auto balanceo según las propiedades

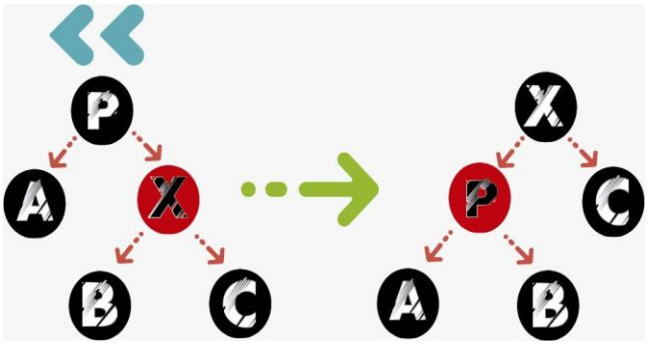

Gráfico 3: Simple rotación a la izquierda**.**

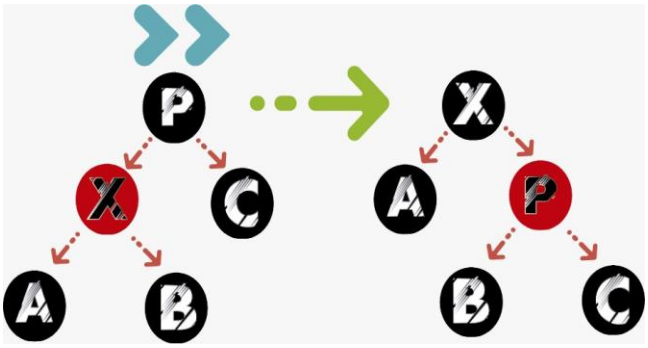

Gráfico 4: Simple rotación a la derecha.

Inserción

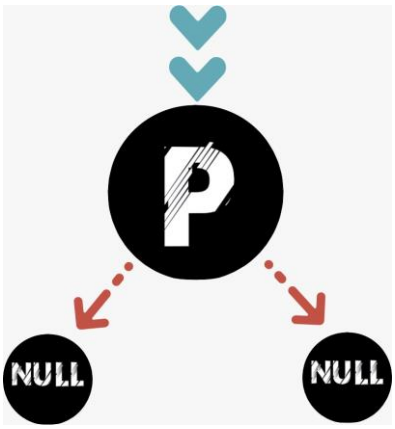

Gráfico 5: El primer caso de inserción, cuando el nodo a ingresar es la raíz.

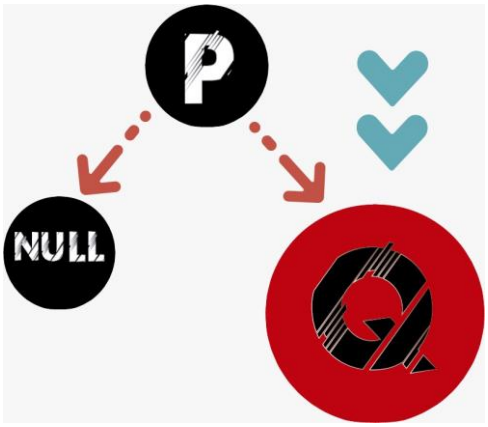

Gráfico 6: Segundo caso, cuando el nodo a insertar es hijo de la raíz.

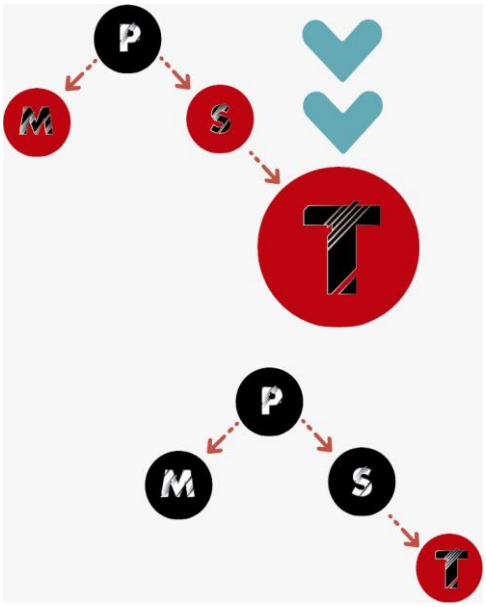

Gráfico 7: Tercer caso, cuando modifica los colores para que las 4 propiedades se cumplan.

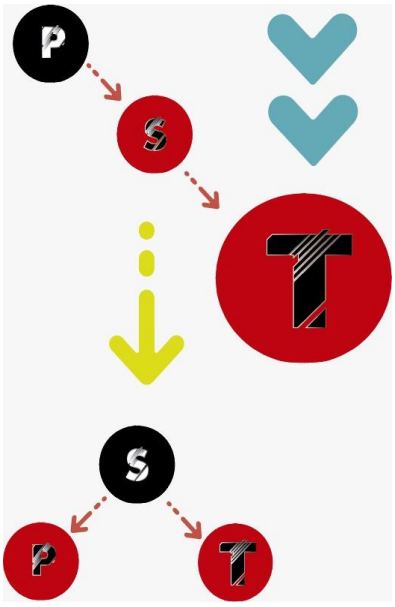

Gráfico 8: Cuarto caso, rotación para balancear el árbol

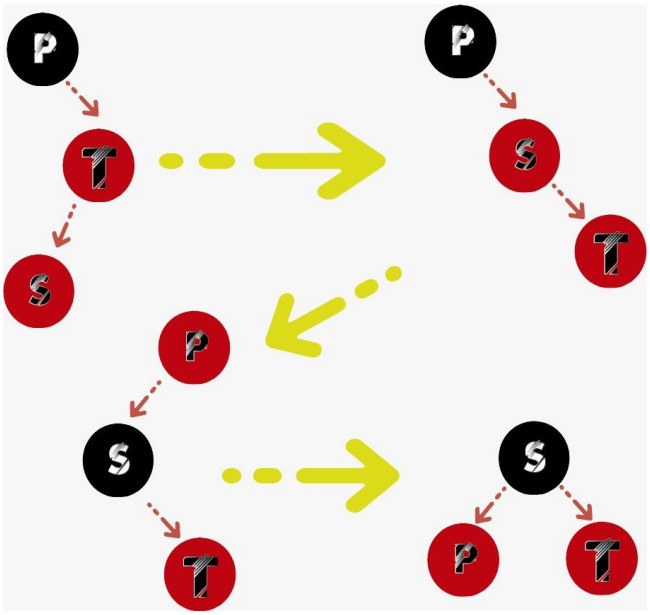

Gráfico 9: Quinto caso, una doble rotación para poder equilibrar el árbol, y cumplir las 4 propiedades.

4.2 Criterios de diseño de la estructura de datos

La decisión de utilizar el árbol rojo-negro radica en que es una estructura de datos, que permite organizar de forma eficiente los datos a almacenar en la estructura y posteriormente buscarlos. El árbol rojo-negro usado (*treeMap*[6] de Java) nos garantiza que para los métodos simples de inserción y búsqueda tengan una complejidad O(log n)[7] y para los que nosotros colocamos como devolver la ruta del archivo y leerlo, el tiempo que se tome también sea poco como se mostrará en las tablas de tiempos de los métodos, esto se da gracias a qué es un árbol auto balanceado que organiza los datos leídos (tipo String) en orden alfabético, garantizando así, por las reglas explicadas en el numeral 3.1. que los recorridos que hace al ejecutar los métodos sean los más óptimos posibles, registrando un mejor tiempo. Sin embargo, dado que éste árbol trabaja con "nodos centinelas" para indicar cuando un nodo no tiene hijos, ocupa más memoria que otros árboles (como el AVL)[8] mas dado que realiza menos rotaciones, la inserción de los datos para su posterior búsqueda es mucho más rápida.

La búsqueda se da por medio de dos maneras según solicite el usuario, por tamaño (que devuelve el tamaño del archivo buscado), por nombre del archivo (que devuelve la ruta del archivo), o por dueño (devuelve el nombre del dueño del archivo) y se hace por tres árboles distintos según el caso, en el primero se recibe el nombre del archivo como llave y el tamaño como valor; mientras que, en el caso de buscar por nombre, se guarda el nombre del archivo como llave y la carpeta de dónde proviene facilitando así la devolución de la ruta mediante la obtención de los valores, finalmente por dueño, se almacena como clave el nombre de la carpeta y su dueño como el valor a devolver.

4.3 Análisis de complejidad

Método	**Complejidad**
Lectura y almacena-miento	O(n Log n)
Búsqueda por nombres	O(n Log n)
Búsqueda por tamaño	O(Log n)
Búsqueda por dueño	O(Log n)

Tabla1:

4.4 Tiempos de ejecución

Métodos	**Mayor tiempo**	**Menor tiempo**	**Tiempo promedio**
Lectura y almacena-miento	147 ms	133 ms	140 ms
Búsqueda por nombres	592559 ns	117115 ms	354837 ns
Búsqueda por tamaño			

	76022 ns	68625 ns	72323 ns
Búsqueda por dueño			

Métodos	**Mayor tiempo**	**Menor tiempo**	**Tiempo promedio**
Lectura y almacena-miento	5 ms	1 ms	3 ms
Búsqueda por nombres	436406 ns	86295 ms	261351 ns
Búsqueda por tamaño	18697 ns	11362 ns	65030 ns
Búsqueda por dueño			

Tabla 2: Tiempos promedio del algoritmo al ejecutar los diferentes métodos de la estructura.

4.5 Memoria

	Conjunto de Datos 1	**Conjunto de Datos 2**
Consumo de Memoria	**16 mb**	**36 mb**

Tabla 2: Consumo de memoria promedio del algoritmo al ejecutar los diferentes métodos de la estructura.

CONCLUSIÓN

Las estructuras de datos son herramientas que permiten manejar de manera más eficiente los datos según lo que se quiera, en el caso del proyecto, usar un árbol rojo-negro, nos permitió manejar los datos de una manera más fácil y rápida, de igual forma, garantizó un manejo ordenado y más eficiente para la búsqueda e inserción.

Este manejo eficiente se puede ver con las tablas de los datos obtenidos, puesto que podemos evidenciar que los procesos no llevan demasiado tiempo en ser realizados, además, el manejo de memoria también es bastante eficiente como podemos evidenciar en las tablas para los conjuntos de datos.

Cabe resaltar que la decisión de usar el árbol rojo negro subyace bajo la búsqueda de una estructura que específicamente útil para este proyecto, puesto que, a diferencia de las otras estructuras; ésta es auto balanceada, con complejidades de O(log n) y no tiene que establecerse un tamaño predeterminado, su funcionamiento es adecuado.

Finalmente, se evidencia la importancia de la realización de este tipo de proyecto que, como futuros ingenieros, nos permiten saber cómo aplicar soluciones por medio del manejo óptimo de los datos a trabajos futuros como búsquedas en internet, programas eficientes para bases de datos, entre otros.

REFERENCIAS

1. Richard Johnsonbaug and Marcus Schaefer, ALGORITHMS, 2004. Consultado agosto 12, 2017.

2. Thomas Cormen, Introduction to Algorithms (3th edition), 2009. Consultado agosto 12, 2017.

3. Robert Lafore, Data Structures and Algorithms in Java (2nd edition), 2002. Consultado agosto 12, 2017.

4. Sergio Sama, DataStructures Tool. Consultado agosto 13, 2017. http://www.hci.uniovi.es/Products/DSTool/hash/hash-queSon.html

5. Jessica Castillo and Magaly Martínez, Árboles rojos-negros, 2016. Consultado octubre 15, 2017. https://www.slideshare.net/whaleejaa/arbol-rojo-y-negro-57693151

6. Class TreeMap, consultado octubre 15, 2017. https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html

7. Big O Cheat Sheet, consultado octubre 20, 2017. http://bigocheatsheet.com/

8. Red Black Tree Over AVL Tree, 2012. StackOverflow, consultado octubre 28, 2017. https://stackoverflow.com/questions/13852870/red-blacktree-over-avl-tree