ESTRUCTURA DE DATOS PARA EL ALMACENAMIENTO DE UN SISTEMA DE DIRECTORIOS

Agustín Nieto García
Universidad Eafit
Colombia
anietog1@eafit.edu.co

David Trefftz Restrepo
Universidad Eafit
Colombia
ditrefftzr@eafit.edu.co

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

RESUMEN

La industria y, en general, el mundo y los estilos de vida actuales requieren del manejo de enormes volúmenes de datos, de manera que no se pierdan u ocurran errores mientras se manipulan. Así, son necesarias estructuras que permitan un manejo y administración eficientes de estos.

Para solucionar este problema se decidió hacer uso de una estructura de datos en forma de árbol, de manera que se permitiera una búsqueda logarítmica alrededor de cierto parámetro, la cual evolucionó a hacer uso de varios árboles para permitir la búsqueda por múltiples parámetros a gran velocidad.

Los resultados en cuanto a tiempo de procesamiento de las búsquedas e inserciones fueron positivos, dejando todas las operaciones ofrecidas en tiempos logarítmicos, en el peor de los casos (que, cabe resaltar, no suele existir el peor caso). Sin embargo, para lograrlo se tuvo que sacrificar cierta cantidad de espacio en cuanto a memoria, lo cual, si se buscara usar esta estructura para una cantidad de datos enormes, no tendría relevancia, al verse desde las ventajas de búsqueda ofrecidas. Durante la finalización del proyecto, fue esa una de las conclusiones a las que se llegó: el hecho de que siempre hay que sacrificar algún requisito por otro, por ejemplo, para este caso se cedió memoria para ofrecerle al usuario mejor rendimiento en otras operaciones.

Palabras clave

Árbol rojo-negro; búsqueda de archivos; estructuras de datos.

Palabras clave de la clasificación de la ACM

Information systems → Directory structures; Theory of computation → Data structures design and analysis; Information systems → Record storage alternatives.

1. INTRODUCCIÓN

Debido al mundo tan globalizado en cual vivimos hoy en día, la industria usualmente debe lidiar con cantidades masivas de datos constantemente. Esta necesidad lleva a los ingenieros de sistemas a buscar maneras más eficientes de almacenar y acceder estos datos.

2. PROBLEMA

El problema consiste en cómo administrar cantidades masivas de datos, manteniendo un orden que permita buscar, agregar, borrar o modificar la información. Esto, debido a, la gran cantidad de información manejada en las industrias, que cada vez solicitan sistemas más veloces y eficientes. Así, es necesario una estructura de datos que represente eficientemente los directorios y archivos y permita acceder a estos.

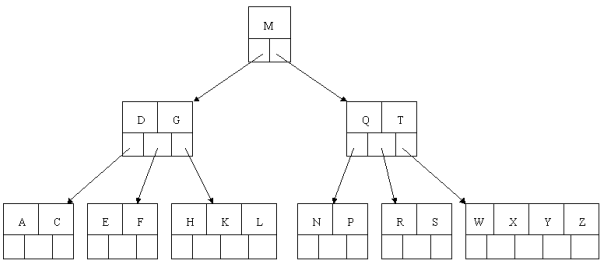
3. TRABAJOS RELACIONADOS

Algunos trabajos relacionados con la eficiente administración de datos y que podrían o han sido usados para el ordenamiento de ficheros se presentan a continuación:

3.1 Árbol B

Un Árbol B es una estructura de datos desarrollada por Rudolf Bayer y Ed McCreight, que almacena nodos con una cantidad x de referencias a nodos hijos y una cantidad x-1 de claves o pares (valores que almacena), y agrega como condición de balanceo que todas las hojas se encuentren al mismo nivel en el árbol. Así, al exceder la cantidad máxima de elementos dentro de un nodo, este puede dividirse, haciendo que el árbol deba volver a balancearse, formando nuevos nodos o agregando el valor excedente a otro nodo.

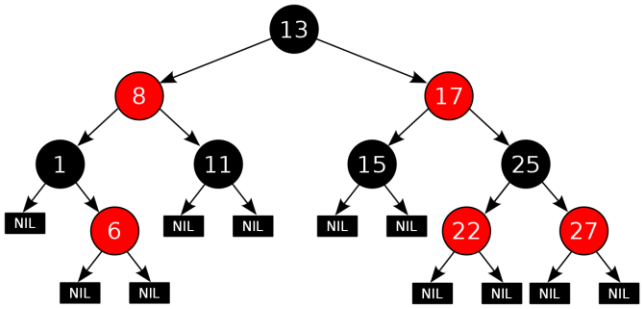
La ventaja que tienen los árboles B sobre los árboles binarios y otras estructuras de almacenamiento de datos es que conforme aumentan la cantidad de nodos hijos máximos de los nodos internos, la altura del árbol decrece (hay menos nodos y más datos), las operaciones de balanceo o búsqueda son menores y la eficiencia de la estructura aumenta. Sin embargo, el principal problema que presenta es que puede exceder en consumo de memoria debido a posibles espacios en memoria apartados (debido al tamaño mínimo y máximo predeterminado de claves) y en desuso. Por ello, la utilización de los Árboles B se realiza usualmente en dispositivos de almacenamiento secundario, como pueden ser discos rígidos [1].



Ejemplo de un Árbol B [2].

3.2 Árbol rojo y negro

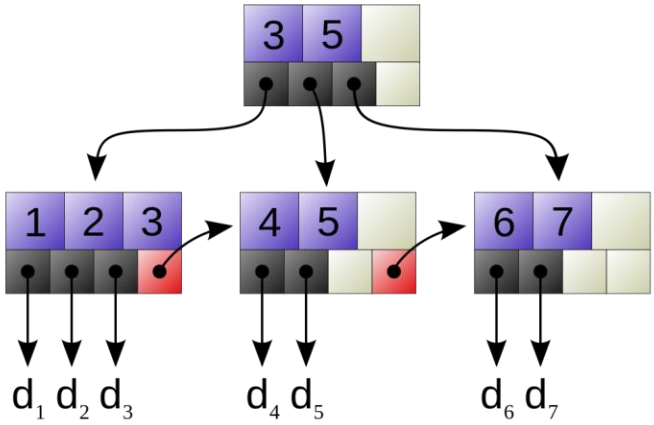
El árbol rojo-negro es un "árbol binario de búsqueda equilibrado, una estructura de datos utilizada en informática y ciencias de la computación". Es una versión mejorada del árbol AVL, donde utilizan ciertas marcas (rojas y negras) para facilitar la identificación de la raíz, nodos y hojas, en vez de enteros (optimización de espacio). Las cuales permiten saber el tiempo en que requiere una modificación en el árbol, ya que se tiene como condiciones que la raíz del árbol siempre debe ser negra, al igual que las hojas; se debe tener exactamente dos nodos negros debajo de un nodo rojo (de allí el nombre del algoritmo, ya que genera una secuencia de negro, rojo, negro continuamente) y que la distancia de la raíz a cualquier hoja del árbol siempre contendrá la misma cantidad de nodos negros. De tal manera, al no cumplirse alguna de estas condiciones, el árbol deberá balancearse. Este hecho permite que las operaciones de inserción, búsqueda y borrado tengan complejidades logarítmicas, es decir O(log(n)) [3].



Ejemplo de un árbol rojo-negro [3].

3.3 Árbol B+

Un árbol B+ es una variación de un árbol B, que añade a la estructura el hecho de que la información se guarda en los nodos hoja, que además se hallan conectados como una lista enlazada. La ventaja que contrae en comparación con un árbol B es que permite una permite la recuperación en rango dentro de los nodos hoja, gracias a la búsqueda secuencial. Sin embargo, como desventaja se hallan, además de las que tenía el árbol B, el hecho de que la altura del árbol se ve incrementada en el mejor de los casos, lo cual se compensa con el hecho de que se reduce la altura para el peor de los casos [4].



Ejemplo de un árbol B+ de grado 4 [4].

5. Árboles de archivos ordenados por parámetros

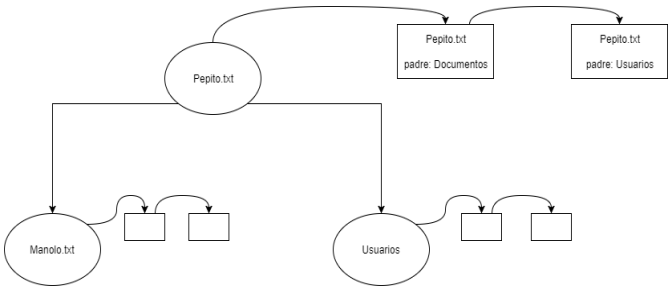
La estructura de datos final implementada, a diferencia de la anterior que era similar a simular el árbol de directorios; consta de varios árboles, cada uno organizado según cierto parámetro, de manera que se consigue un acceso logarítmico a cada uno de estos árboles.

5.1 Estructura de datos principal

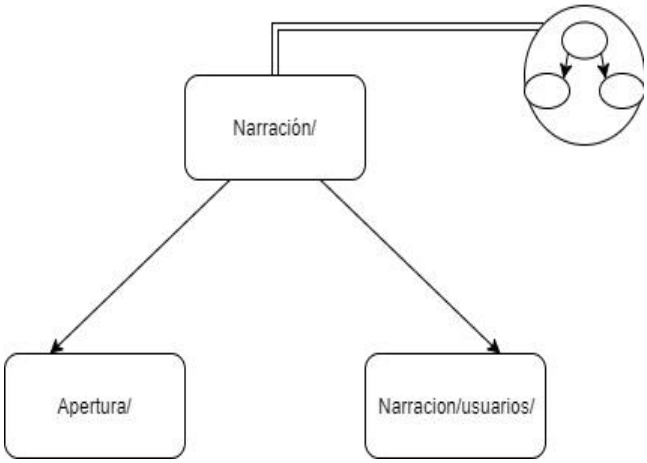
La estructura de datos es un árbol rojo-negro el cual es ofrecido por el API de Java, con la clase TreeMap. Este permite operaciones logarítmicas de búsqueda e inserción.

5.1 Operaciones de la estructura de datos

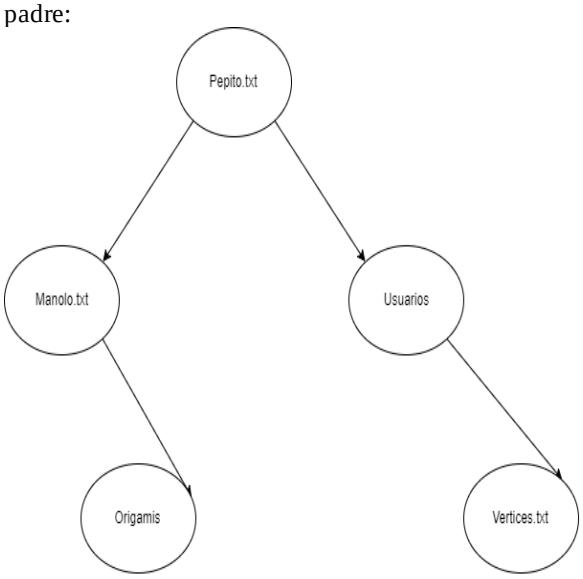
Búsqueda: En la siguiente imagen se muestra cómo se vería un árbol de aquellos, en este caso el árbol de ordenamiento por nombre, que contiene todos los archivos con el mismo nombre en una lista enlazada dentro del nodo. Se puede ver que cada nombre de archivo guarda todas las referencias a aquellos que tengan el mismo nombre, se diferencian por distintas razones, pero en este caso escogimos por cuál es su folder padre, la cual es la diferencia esencial. La búsqueda sobre todos los árboles es una búsqueda binaria, ya que estos están ordenados de manera lexicográfica. Se muestra únicamente la siguiente gráfica debido a que los árboles de los demás, excepto de las carpetas son totalmente iguales.



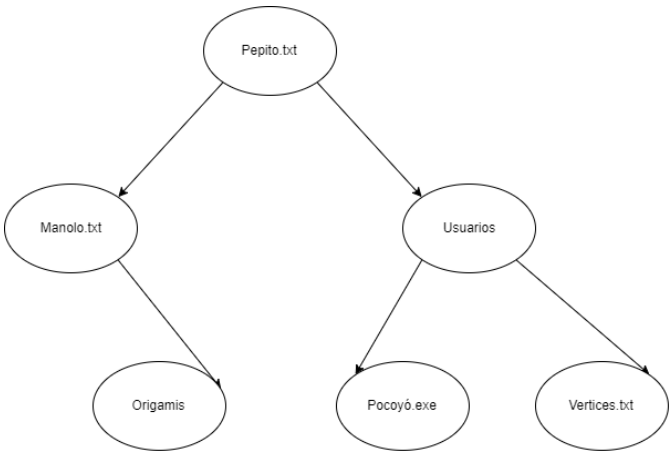
Árbol de carpetas: Cada carpeta tiene su propio conjunto de archivos que lo tienen como padre. Este está implementado usando la clase TreeSet, de manera que las búsquedas e inserciones en este también son logarítmicas y no se permitan archivos repetidos. Gráficamente es algo similar a lo siguiente:



Por otra parte, el siguiente es un ejemplo del conjunto de archivos posible dentro de la carpeta Narración. Lo que tendrían en común los archivos sería, en esencia, su carpeta padre:



Inserción: La inserción se realiza por orden natural del parámetro deseado (Nombre, tamaño, padre, extensión o usuario). Por ejemplo, para el conjunto anterior después de insertar Pocoyó.exe, quedaría así:



5.2 Criterios de diseño de la estructura de datos

Se decidió, para la estructura, maximizar la eficiencia de la búsqueda e inserción, mientras se le ofrecía al usuario la mayor cantidad de opciones posibles. Para ello hubo que sacrificar un poco el espacio de memoria. Se utilizo el concepto de manejar cinco arboles distintos para facilitar la búsqueda dependiendo del parámetro que se pase. Esto permite que el usuario tenga más opciones para poder restringir la búsqueda, como se puede ver en los distintos métodos ofrecidos por la estructura. Los tiempos, al utilizar esta aproximación son reducidos a cantidades absurdamente pequeñas. En el caso de que la cantidad de archivos sea de un billón al cuadrado, se reduce a solo 80 operaciones, lo cual es lo suficientemente veloz en la mayoría de los casos. A la conclusión de que debía realizarse de tal manera, se llegó después de haber implementado este con tablas de hash, las cuales posteriormente descartamos debido a su peor caso de O(n) y posteriormente con una estructura similar a los árboles de directorios, la cual además de restringir bastantes operaciones, podía llegar a tener complejidades O(n), debidas a la necesidad de una búsqueda recursiva de carpeta en carpeta.

5.3 Análisis de la Complejidad

La complejidad de las operaciones, para la estructura de datos actual sería, en el peor de los casos, como se muestra en la siguiente tabla, con n siendo el número de archivos en la estructura. Sin embargo, estas situaciones no suelen suceder y, en la mayoría de los casos la complejidad es supremamente menor, como se mostrará en la toma de tiempos para las operaciones de inserción del set de datos juegos.txt.

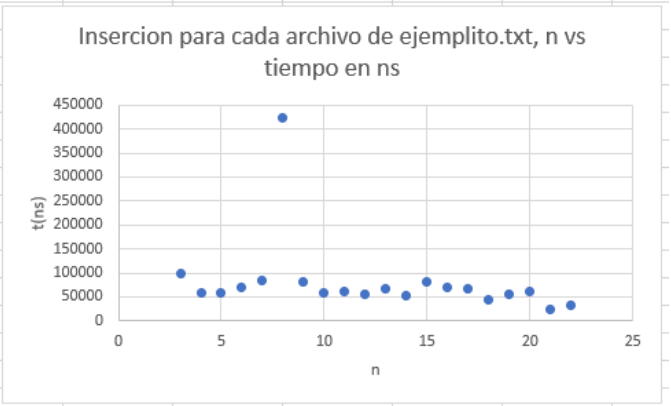
Método	Complejidad
Inserción por usuario	O(log(n))
Inserción por carpeta	O(log(n^2))
Inserción por tamaño	O(log(n))
Inserción por extensión	O(log(n))
Inserción por nombre	O(log(n))
Búsqueda por usuario	O(log(n))
Búsqueda por carpeta	O(log(n))
Búsqueda por nombre	O(log(n))
Búsqueda por tamaño	O(log(n))
Búsqueda por extensión	O(log(n))
Impresión del árbol de directorios	O(n*log(n))

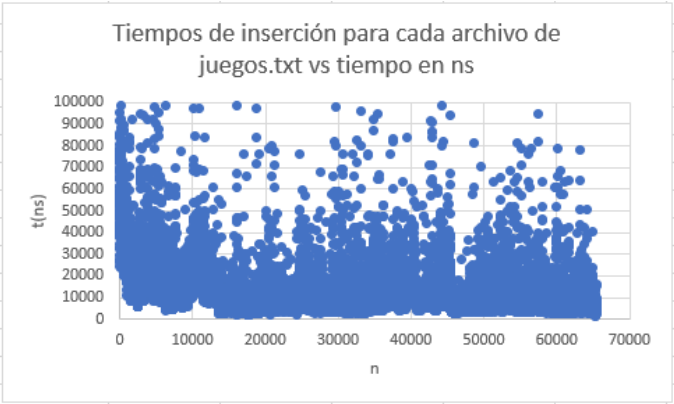
5.4 Tiempos de Ejecución

Los tiempos de ejecución de la siguiente tabla fueron tomados obteniendo el promedio de cada operación realizada 10000 veces, por lo que fue influido un tanto por el caché de las operaciones. Sin embargo, responde a lo que se esperaba obtener después de haber realizado el cálculo de complejidades.

Método	ejemplito.txt(n = 21)	juegos.txt(n = 65510)
Búsqueda por usuario	435	286
Búsqueda por carpeta	490	494
Búsqueda por nombre	444	693
Búsqueda por tamaño	415	436
Búsqueda por extensión	452	522

A continuación se muestran dos tablas, que reflejan la inserción de cada archivo hallado en los set de datos uno por uno respecto al n que correspondiente. Con juegos.txt se ve algo muy interesante, y es lo que se enunciaba anteriormente en los cálculos de complejidades: para estos sets de pruebas, el peor caso para todo casi que no existe y como la inserción se debe realizar en todos los árboles simultáneamente, ocurre que en la mayoría de los casos la complejidad será muy baja, además de que por ser logarítmica no se distinga mucho la diferencia, como ocurre en ejemplito.txt y en los tiempos anteriores.





5.5 Memoria

La memoria consumida para almacenar los datos depende de múltiples factores, como son la longitud de los nombres, la cantidad de carpetas anidadas, el estado actual de la JVM, entre otros. Para los casos de prueba juegos.txt y ejemplito.txt se obtuvo los siguientes resultados:

Set de datos	Memoria
juegos.txt	221.2 MB
ejemplito.txt	1.3 MB

5.6 Análisis de los resultados

Los resultados obtenidos no responden, ciertamente, al cálculo de complejidades anterior, debido a motivos anteriormente mencionados. Por ello, se propone el siguiente cálculo de complejidades, el cual no correspondería al peor de los casos, pero sí a los casos más reales:

Método	Complejidad
Inserción por usuario	Θ(log(u))
Inserción por carpeta	Θ(log(f))
Inserción por tamaño	Θ(log(s))
Inserción por extensión	Θ(log(x))
Inserción por nombre	Θ(log(m))
Búsqueda por usuario	Θ(log(u))
Búsqueda por carpeta	Θ(log(f))
Búsqueda por nombre	Θ(log(m))
Búsqueda por tamaño	Θ(log(s))
Búsqueda por extensión	Θ(log(x))
Impresión del árbol de directorios	Θ(n*log(f))

Para el cual las variables serían:
n: número de archivos
u: número de usuarios distintos
f: número de carpetas existentes
s: número de distintos tamaños de archivos, esta es posiblemente la variable que más cercana a n estaría.
x: número de extensiones distintas.
m: cantidad de nombres diferentes.

6. CONCLUSIONES

No existe una estructura de datos perfecta. A medida que íbamos decidiendo que aproximación tomar para el proyecto, nos dimos cuenta de que siempre se tenía que sacrificar algo en el proceso, ya fuera eficiencia, seguridad, memoria o la cantidad de operaciones que se ofreciera al usuario. El enfoque que se tomó, finalmente, fue completamente hacia la eficiencia de la estructura de datos, ignorando que tan seguro podría ser el acceso de las personas a estos. Se podría decir que se creó una estructura que permite al usuario experimentar formas de búsqueda de información en bases de datos de gran tamaño. En cuanto a los cálculos de complejidad queda resaltar el hecho de que, son únicamente una guía de los peores, comunes y mejores casos, ya que, por ejemplo, para la estructura actual, la complejidad llegó a ser casi que imposible que llegara al peor de los casos.

6.1 Trabajos futuros

Se podría implementar una estructura propia, de manera que se combinen todos los parámetros en un solo árbol y se reduzca un poco la cantidad de nodos presentes, lo cual optimizaría un poco la memoria. También se puede contemplar la posibilidad de optimizar la memoria reduciendo las opciones ofrecidas, ya que por ahora se desconoce si sí son todas necesarias también estaría muy bien optimizar operaciones como el borrado y la modificación que por este momento tendrían complejidades mayores a lo deseado.

AGRADECIMIENTOS

Se quiere dar gracias a:

El estudiante Kevin Parra por haber colaborado con la realización del proyecto en cuanto a la lectura de los casos de prueba respecta.

El profesor Helmuth Trefftz por ayudar en la investigación de distintos métodos para la creación de la estructura de datos.

La estudiante Lisset Arzola por proporcionar equipos que facilitaron la finalización del proyecto.

REFERENCIAS

1. Árbol-B. Es.wikipedia.org, 2017. https://goo.gl/7kEm8c.

2. B Tree. Nptel.ac.in, 2017. https://goo.gl/N5ecPr.

3. Árbol rojo-negro. Es.wikipedia.org, 2017. https://goo.gl/qLp9z2.

4. Árbol B+. Es.wikipedia.org, 2017. https://goo.gl/sy6xP3
.