Estructura de datos para buscar en el contenido de un directorio

Alejandro Cano Munera
Universidad Eafit
Colombia
acanom@eafit.edu.co

Jorge Luis Herrera Chamat
Universidad Eafit
Colombia
jlherrerac@eafit.edu.co

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

RESUMEN
El problema de buscar en el contenido de un directorio consiste en encontrar eficientemente, los archivos y subdirectorios que se encuentran en un directorio. Este problema juega un papel muy importante en la mayoría de las aplicaciones y sistemas operativos actuales ya que en gran parte la eficiencia y desempeño de estos depende directamente de la capacidad que tenga una estructura de datos implementada en estos sistemas de hacer búsquedas y/o modificaciones. A lo largo de la historia se han presentado diversos problemas que corresponden a las nuevas exigencias de los mercados de manejar un volumen de datos cada vez más grande, un claro ejemplo de ello es el sistema de archivos ext2, que debido a las desventajas de almacenamiento que presenta fue desplazado por ext3, sin embargo, este último también presento diversas desventajas frente a las exigencias de manejo de información siendo reemplazado por ext4, y así cada vez más surgen nuevas necesidades que deben ser suplidas por sistemas más eficientes y de mayor capacidad.

La solución que generamos a dicho problema fue una estructura de datos diseñada específicamente para la búsqueda eficiente de un contenido dado. Los resultados luego de la implementación y pruebas fueron muy buenos, ya que en poco tiempo de búsqueda se logran obtener los resultados esperados. En dicho proceso de análisis y diseño logramos determinar cuáles son las diferencias entre las diferentes estructuras de datos y como cada una cumple un papel fundamental en diferentes tareas.

Palabras clave
Estructuras de datos, eficiencia, buscador, archivos, Árbol de búsqueda.

Palabras clave de la clasificación de la ACM

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

1. INTRODUCCIÓN
A lo largo de la historia han sido varias las estructuras de datos que han nacido con la finalidad de darle un buen manejo a la información y permitir cada vez de forma más rápida y eficiente realizar búsquedas y modificaciones en un conjunto de archivos, y abarcar un número más grande de información, algunos ejemplos de estructuras de datos que han respondido a estas necesidades son el Árbol-B*, Árbol-AVL, Árbol-B+, entre otros. En la actualidad el volumen de datos almacenado por las diferentes aplicaciones y sistemas operativos va en aumento, debido a esto las exigencias acerca de una estructura de datos que funcione de manera más eficiente y que abarque un volumen más grande de datos también aumentan. El objetivo de este documento es hacer una recopilación de toda la información recogida durante la implementación y pruebas de la estructura de datos que lista el contenido de un directorio eficientemente.

2. PROBLEMA
El problema de buscar en el contenido de un directorio consiste en encontrar eficientemente, los archivos y subdirectorios que se encuentran en un directorio, la importancia de la solución de este problema está en que una estructura de datos que maneje los archivos de una manera más eficiente supone un mejor desempeño de las aplicaciones y sistemas operativos y en efecto una mejor experiencia para los usuarios.

3. TRABAJOS RELACIONADOS

3.1 Árbol-B*
El "Árbol-B*" es una estructura de datos propuesta por Donald Knuth en 1973, con el fin de proponer nuevas reglas para realizar el mantenimiento de los Árboles-B; de forma que no se realiza una división de un nodo en dos ya que eso hace que los nodos resultantes tengan la mitad de claves, sino que se realizan divisiones de dos nodos completos a tres de forma que los nodos resultantes tienen dos tercios del total. Por consiguiente, este tipo de árboles son muy similares a los Árboles-B, diferenciándose únicamente en la cantidad de llaves repartidas en cada nodo [1].

3.2 Árbol binario

Los árboles a diferencia de las listas son una estructura de datos no lineal, atendiendo más a una estructura de tipo jerárquico. En los arboles binarios se pueden realizar principalmente cuatro tipos de recorridos: Preorden: (raíz, izquierdo, derecho), inorden: (izquierdo, raíz, derecho), postorden: (izquierdo, derecho, raíz). Entre otras aplicaciones, los árboles se emplean para analizar circuitos eléctricos y para representar la estructura de fórmulas matemáticas, así como para organizar la información y realizar búsquedas [2].

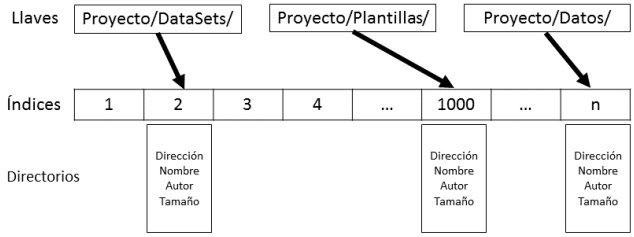
3.3 Árbol-AVL

El "Árbol-AVL" es una estructura de datos creada por Adelson-Velskii y Landis en 1962, Los árboles AVL están siempre equilibrados de tal modo que, para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles [3].

3.4 Árbol-B+

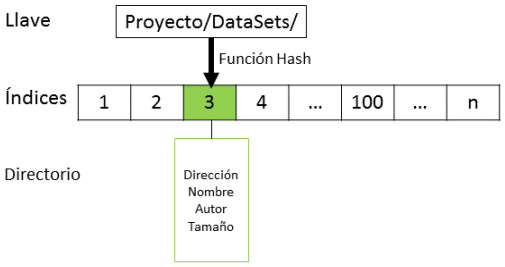
Los "Árboles B+" constituyen otra mejora sobre los árboles B, pues conservan la propiedad de acceso aleatorio rápido y permiten además un recorrido secuencial rápido. En un árbol B+ todas las claves se encuentran en hojas, duplicándose en la raíz y nodos interiores aquellas que resulten necesarias para definir los caminos de búsqueda. Para facilitar el recorrido secuencial rápido las hojas se pueden vincular, obteniéndose, de esta forma, una trayectoria secuencial para recorrer las claves del árbol. Su principal característica es que todas las claves se encuentran en las hojas. Los árboles B+ ocupan algo más de espacio que los árboles B, pues existe duplicidad en algunas claves. En los árboles B+ las claves de las páginas raíz e interiores se utilizan únicamente como índices [4].

4. Tabla de Hash

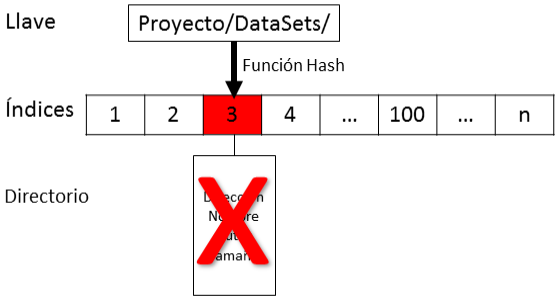


Gráfica 1: Tabla de Hash de directorios. Un directorio es una clase que contiene dirección, nombre, autor y tamaño. La llave principal para guardar en la tabla son las rutas.

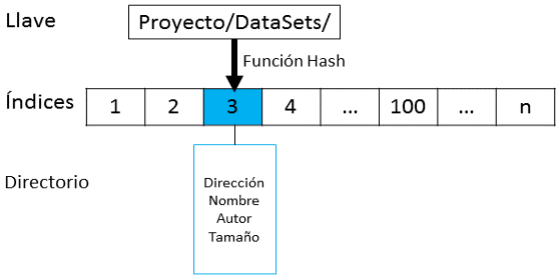
4.1 Operaciones de la estructura de datos



Gráfica 2: Imagen de una operación de insertar de una tabla de Hash. Se le aplica la función hash a la ruta y se inserta en la tabla.



Gráfica 3: Imagen de una operación de borrado de una tabla de Hash. Se le aplica la función hash a la ruta y se inserta en la tabla.



Gráfica 4: Imagen de una operación de búsqueda de una tabla de Hash. Se le aplica la función hash a la ruta y se inserta en la tabla.

4.2 Criterios de diseño de la estructura de datos

Elegimos una tabla de Hash como estructura de datos principalmente por su excelente eficiencia a la hora de realizar búsquedas, ya que las búsquedas son la actividad principal por realizar durante el tiempo de ejecución, además las otras operaciones (Borrar e insertar) también presentan óptimos desempeños, por otra parte, esta estructura de datos tuvo un satisfactorio rendimiento en cuanto al uso de la memoria. Estos datos los corroboramos experimental y teóricamente luego de indagar e implementar diversas estructuras de datos tales como arreglos, listas enlazadas, arboles, entre otros.

4.3 Análisis de Complejidad

Método	Complejidad
Insertar un directorio	O(n)
Eliminar un directorio	O(n)
Buscar un directorio	O(n)

Tabla 1: Tabla para reportar la complejidad para el peor de los casos en las operaciones principales de la tabla Hash diseñada.

4.4 Tiempos de Ejecución

La siguiente tabla muestra el tiempo en milisegundos que le tomo a la estructura de datos realizar las operaciones de insertar, buscar y eliminar con los diferentes conjuntos de datos.

	Conjunto de datos 1 (3 dir, 18 fic)	Conjunto de datos 2 (425 dir, 3225 fic)	Conjunto de datos 3 (2609 dir, 62901 fic)
Leer txt	1000 ms	3000 ms	8000 ms
Insertar	0,2 ms	0,5 ms	1 ms
Borrar	0,2 ms	0,5 ms	1 ms
Buscar	0,2 ms	0,5 ms	1 ms

Tabla 2: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos

4.5 Memoria

La siguiente tabla muestra la cantidad de memoria en Megabytes que le tomo a la estructura de datos realizar las operaciones de insertar, buscar y eliminar con los diferentes conjuntos de datos.

	Conjunto de datos 1 (3 dir, 18 fic)	Conjunto de datos 2 (425 dir, 3225 fic)	Conjunto de datos 3 (2609 dir, 62901 fic)
consumo de memoria	15 MB	40 MB	70 MB

Tabla 3: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

4.6 Análisis de los resultados

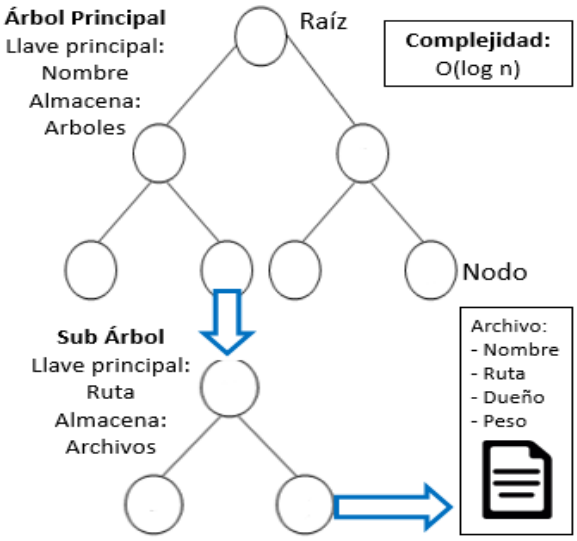
Los datos que se muestran a continuación son el resultado de las diferentes estructuras de datos implementadas, donde podemos evidenciar, que el mejor desempeño tanto en memoria como el tiempo de ejecución lo obtuvo la tabla de Hash

Estructuras de datos	Lista de Arreglos (ArrayList)	Lista enlazada (LinkedList)	Tabla de Hash
Tiempo de ejecución	10000 ms	25000 ms	8000 ms
Memoria	80 MB	45 MB	70 MB
Insertar	6000 ms	1000 ms	1 ms
Eliminar	6000 ms	1000 ms	1 ms
Búsqueda	6000 ms	500 ms	1 ms

Tabla 4: Análisis de los resultados obtenidos con la implementación de la estructura de datos.

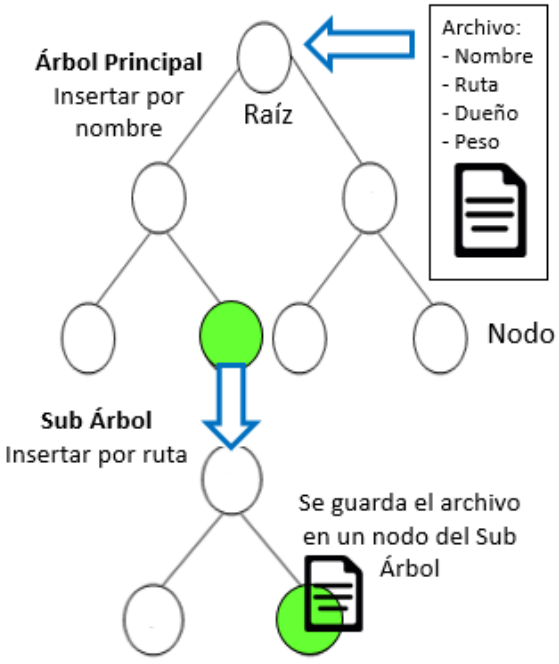
5. Árbol de Arboles

(Utilizando la implementación de Java 'TreeMap').

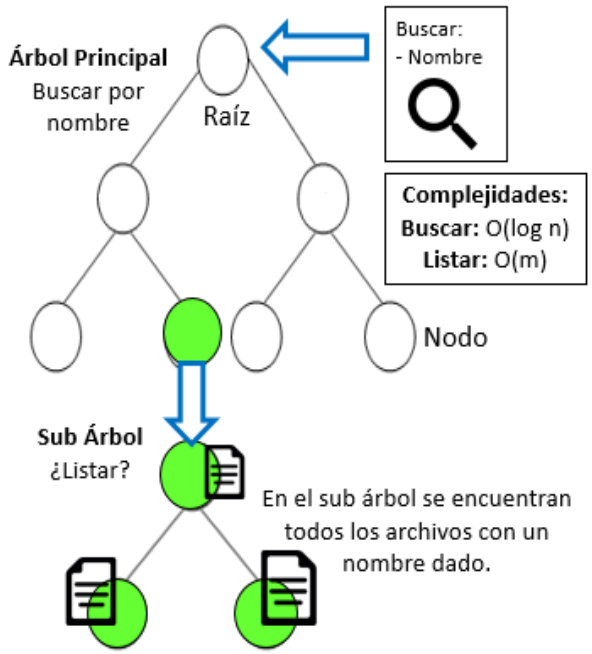


Gráfica 1: Árbol de Árboles. Cada nodo del árbol principal contiene un sub árbol, donde se almacenan ya sea los ficheros o directorios, ambos con atributos en común tales como nombre, ruta, dueño y peso.

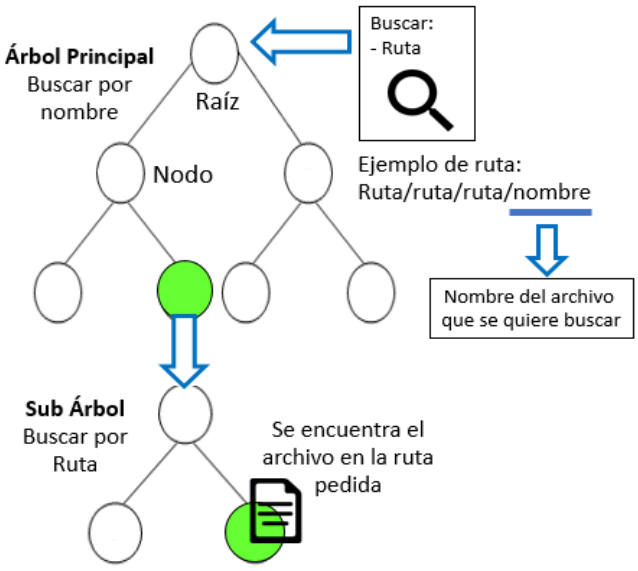
4.1 Operaciones de la estructura de datos



Gráfica 2: Operación de insertar Archivo en la estructura de datos. Primero se piden todos los datos al usuario (Nombre del archivo, ruta, dueño y peso), luego se busca en el árbol principal por el nombre y se almacena el archivo en el subárbol por la ruta.



Gráfica 3: Operación buscar por nombre en la estructura de datos. Basados en el nombre del archivo se hace una búsqueda del nodo que contiene el subárbol con los archivos con dicho nombre y se da la opción de listar todos los archivos con dicho nombre, lo cual tendría complejidad O(m) siendo m el número de archivos con el nombre dado.



Gráfica 4: Operación buscar por ruta en la estructura de datos. Basados en el nombre del archivo se hace una búsqueda del nodo que contiene el subárbol con los archivos con dicho nombre y luego se busca el elemento con la ruta completa en este subárbol.

4.2 Criterios de diseño de la estructura de datos

Elegimos un árbol de árboles como estructura de datos principalmente por su excelente eficiencia a la hora de realizar búsquedas, ya que las búsquedas son la actividad principal por realizar durante el tiempo de ejecución, además las otras operaciones como insertar también presentan óptimos desempeños. Esta estructura de Árbol nos garantiza una complejidad en el peor de los casos de O(log n) en las operaciones más importantes, gracias al balanceo de sus nodos. Por otra parte, esta estructura de datos tuvo un satisfactorio rendimiento en cuanto al uso de la memoria. Estos datos los corroboramos experimental y teóricamente luego de indagar e implementar diversas estructuras de datos tales como arreglos, listas enlazadas, tablas de Hash, entre otros.

4.3 Análisis de Complejidad

Método	Complejidad
Insertar un directorio	O(log n)
Insertar un archivo	O(log n)
Buscar por nombre	O(log n)
Buscar por ruta	O(log n)
Listar elementos	O(m)

Tabla 1: Tabla para reportar la complejidad en el peor de los casos, en esta tabla n representa el número total de elementos en la estructura de datos y m el número de archivos en una búsqueda con un mismo nombre

4.4 Tiempos de Ejecución

La siguiente tabla muestra el tiempo promedio en milisegundos que le tomo a la estructura de datos realizar las operaciones de leer e insertar los datos por primera vez, insertar un dato nuevo y buscar con los diferentes conjuntos de datos.

	Conjunto de datos 1 (3 dir, 18 fic)	Conjunto de datos 2 (425 dir, 3225 fic)	Conjunto de datos 3 (2609 dir, 62901 fic)
Leer texto	0,001 ms	105,7 ms	985 ms
Insertar	0,0001 ms	0,003 ms	0,05 ms
Buscar	0,0001 ms	0,003 ms	0,05 ms
Listar	0,0001 ms	0,003 ms	0,05 ms

Tabla 2: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos en su caso promedio.

4.5 Memoria

La siguiente tabla muestra la cantidad de memoria en MB que le tomo a la estructura de datos realizar las operaciones de insertar, buscar y eliminar con los diferentes conjuntos de datos.

	Conjunto de datos 1 (3 dir, 18 fic)	Conjunto de datos 2 (425 dir, 3225 fic)	Conjunto de datos 3 (2609 dir, 62901 fic)
consumo de memoria	13 MB	24 MB	30 MB

Tabla 3: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

4.6 Análisis de los resultados

Los datos que se muestran a continuación son el resultado de las diferentes estructuras de datos implementadas, donde podemos evidenciar, que el mejor desempeño tanto en memoria como el tiempo de ejecución lo obtuvo el árbol de árboles.

Estructuras de datos	Lista enlazada (LinkedList)	Tabla de Hash	Árbol de Arboles
Cargar archivo	25000 ms	8000 ms	985 ms
Memoria	45 MB	70 MB	30 MB
Insertar	1000 ms	1 ms	0,05 ms
Buscar	1000 ms	1 ms	0,05 ms
Listar	500 ms	1 ms	0,05 ms

Tabla 4: Análisis y comparación de los resultados obtenidos con la implementación de diferentes estructuras de datos en su caso promedio.

5. CONCLUSIONES

1. Esta estructura de Árbol nos garantiza una complejidad en el peor de los casos de O(log n) en las operaciones más importantes, gracias al balanceo de sus nodos. Por otra parte, esta estructura de datos tuvo un buen rendimiento en cuanto al uso de memoria. Estos datos los corroboramos experimental y teóricamente.
2. Obtuvimos una estructura de datos en la que se puede buscar e insertar de manera rápida cualquier archivo o directorio que esté contenido en ésta. De igual manera se puede leer un archivo de texto que contenga archivos y carpetas previamente ordenados y añadirlos con el mismo orden para posteriormente buscarlos.
3. Cambiamos la estructura de la primera solución por completo a fin de optimizar lo máximo posible todas las funciones que debíamos implementar y obtuvimos una eficiencia mucho mayor tanto en las funciones como en la facilidad para utilizar la estructura y las operaciones que se podían realizar dentro de ella.

5.1 Trabajos futuros

Lo que nos gustaría mejorar en el futuro serían los tiempos de búsqueda y el espacio ocupado en memoria, es decir, hacer diferentes cambios que nos ayuden a mejorar la estructura y a hacerla más optima, por otra parte, agregarle más funcionalidades o servicios para tener una gama más amplia de búsquedas por diferentes parámetros. Más adelante buscar que la estructura de datos creada sea una de las más eficientes en el mercado.

AGRADECIMIENTOS

Esta investigación fue soportada parcialmente por el Icetex – Programa Ser Pilo Paga 3

Nosotros agradecemos por su ayuda con una técnica de programación a Luisa María Vásquez Gómez, Luis Javier Palacios Mesa, Sebastián Giraldo Gómez y Santiago Escobar Mejía, estudiantes de ingeniería de sistemas de la Universidad EAFIT.

REFERENCIAS

[1] Fernández, J. Árboles B*. Granada, España. [En línea] Disponible en: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B2.htm [Accedido el 30 de septiembre de 2017]

[2] Wikipedia. Árbol Binario. [En línea] Disponible en: https://es.wikipedia.org/wiki/%C3%81rbol_binario [Accedido el 30 de septiembre de 2017]

[3] Wikipedia. Árbol AVL. [En línea] Disponible en: https://es.wikipedia.org/wiki/%C3%81rbol_AVL [Accedido el 30 de septiembre de 2017]

[4] Fernández, J. Árboles B+. Granada, España. [En línea] Disponible en: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B3.htm [Accedido el 30 de septiembre de 2017]