IMPLEMENTACIÓN DE UN ALGORITMO QUE PERMITA OPTIMIZAR LA BÚSQUEDA DE ARCHIVOS AL LISTAR UN DIRECTORIO.

Alejandro Murillo González
Universidad Eafit
Colombia
amurillog@eafit.edu.co

Juan Pablo Vidal Correa
Universidad Eafit
Colombia
jpvidalc@eafit.edu.co

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

RESUMEN
Actualmente el ser humano maneja grandes flujos de información por medio de dispositivos tecnológicos que emplean sistemas de archivos para dicha tarea, pero estos sistemas presentan complicaciones al momento de enfrentarse a grandes cantidades de datos, lo que ha llevado a que se genere la problemática de mejorar la eficiencia en los procesos de almacenamiento y búsqueda de la información al momento de listar un directorio que contiene la información. Existen varias alternativas viables para mejorar los procesos de los sistemas de archivos, las cuales abarcan desde estructuras de datos hasta sistemas de asignación.

Palabras clave: Nodo, clave, valor hash, raíz (root), complejidad, árbol binario.

Palabras clave de la clasificación de la ACM:

Teoría de la computación → Diseño y análisis de algoritmos → Diseño y análisis de estructuras de datos → Operaciones optimas

Theory of computation → Design and analysis of algorithms → Data structures design and analysis → shortest paths

1. INTRODUCCIÓN

Los avances tecnológicos se aceleraron exponencialmente durante el siglo 20, gracias al impulso producido por dos guerras mundiales y por la posterior guerra fría, estos conflictos permitieron que nuestro dominio de la ciencia y la tecnología creciera a pasos agigantados. Dicha evolución, le permitió al ser humano de la actualidad adentrarse en una época que se conoce como "la era de la información", en donde la humanidad diariamente se ve asediada por masivas cantidades de información. Lo anterior ha hecho necesario que nuestros dispositivos tecnológicos tengan que afrontar el desafío de controlar grandes cantidades de datos, producto de la información que constantemente se recibe. Esto en un corto lapso debido a él gran tráfico de información que se maneja.

Dicha información es manipulada por medio de sistemas de archivos, los cuales son elementos encargados de asignar espacio a los archivos, administrar el espacio libre y permitir el acceso a los datos guardados, esto agrupando la información en bloques.

A partir de lo anterior surge el inconveniente de que algunos sistemas de archivos se ven agobiados por el inmenso flujo de información, ya que no soportan la cantidad de información o su tiempo de respuesta es muy limitado para lo que se necesita actualmente. Es por esto, por lo que sistemas de archivos actuales deben ser reemplazados para garantizar la eficiencia, este es el caso del *ext2,* el cual, al mantener un sistema de asignación por lista enlazada, la cual necesita revisar cada bloque para encontrar los datos, muestra problemas al momento de manejar grandes cantidades de información y es por esto por lo que se debió reemplazar por el *ext4*.

2. PROBLEMA

El problema consiste en generar un sistema de archivos que optimice los procesos de búsqueda y asignación de información para mejorar la eficiencia a la hora de encontrar los archivos y subdirectorios en un directorio con una gran cantidad de información, además de mejorar la administración de los bloques libres, los cuales son uno de los mayores inconvenientes a mejorar en los sistemas de archivos por su mal manejo y la importancia que tienen para reducir los tiempos de respuesta al momento de listar directorios. La resolución de este problema le permitiría al sistema de archivos actualizarse para afrontar los nuevos retos que se avecinan entorno al manejo y la recolección de la información.

3. TRABAJOS RELACIONADOS

Es importante tener en cuenta las siguientes nociones:

Clave: es un campo o una combinación de campos que identifica de forma única a cada fila de una tabla en un fichero o directorio.

Nodo: es un punto de intersecciónn o unión de varios elementos que confluyen en el mismo lugar.

Posibles soluciones:

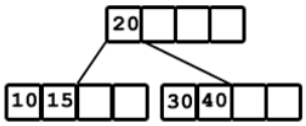
3.1 Árboles B:

Son árboles binarios de búsqueda en los cuales cada nodo interno debe tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos

dentro del rango predefinido, los nodos internos se juntan o se parten.

Búsqueda:

Localizar una clave en un B-árbol es una operación simple pues consiste en situarse en el nodo raíz del árbol, si la clave se encuentra ahí hemos terminado y si no es así seleccionamos de entre los hijos el que se encuentra entre dos valores de clave que son menor y mayor que la buscada respectivamente y repetimos el proceso hasta que la encontremos. En caso de que se llegue a una hoja y no podamos proseguir la búsqueda, la clave no se encuentra en el árbol.

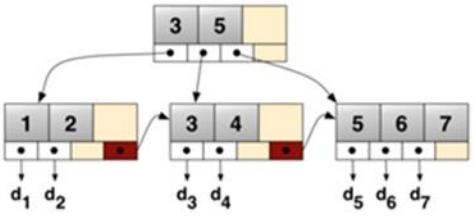


Grafica 1: Árboles B

3.2 Árboles 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. El problema con estos árboles es que ocupan algo más de espacio que los árboles B, pues existe duplicidad en algunas claves, por lo que el árbol B tiene menos nodos, pero debe hacer mucho más recorrido. Además, en los árboles B+ las claves de las páginas raíz e interiores se utilizan únicamente como índices.

Búsqueda:

En este caso, la búsqueda no debe detenerse cuando se encuentre la clave en la página raíz o en una página interior, sino que debe proseguir en la página apuntada por la rama derecha de dicha clave.



Grafica 2: Árboles B+:

Este tipo de árboles de índices en forma de archivos secuenciales indexados tienen un problema: el rendimiento, tanto para buscar en el índice como para buscar registros, se degrada a medida que crece el archivo.

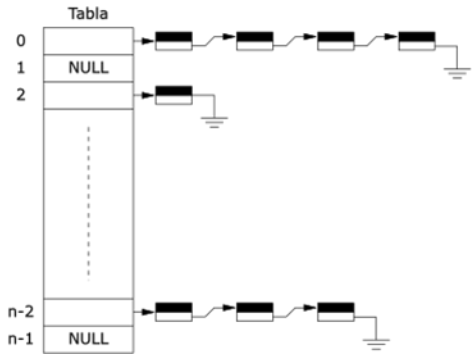
3.3 Tablas hash:
Las tablas hash son estructuras de datos que se utilizan para almacenar un número elevado de datos sobre los que se necesitan operaciones de búsqueda e inserción muy eficientes. Una tabla hash almacena un conjunto de pares "(clave, valor)". La clave es única para cada elemento de la tabla y es el dato que se utiliza para buscar un determinado valor.

Búsqueda:

Se crea una tabla de un tamaño razonable para almacenar los pares (clave, valor) y una función "hash" que recibe la clave y devuelve un índice para acceder a una posición de la tabla. (Existen múltiples librerías en casi todos los lenguajes de programación que proporcionan implementaciones muy eficientes de estas tablas). Puede ocurrir el caso en el que la función devuelve el mismo índice para dos claves distintas, esto se conoce como **colisión.**

Solución para la colisión:

Se puede combinar la estructura de tabla con la de lista encadenada, lo que permite que cada posición de la tabla no almacene un único elemento sino la cabeza de una lista encadenada que a su vez contiene todos aquellos elementos cuya función de hash ha devuelto idéntico resultado. Si en la posición no hay elemento, entonces contiene un puntero a NULL.



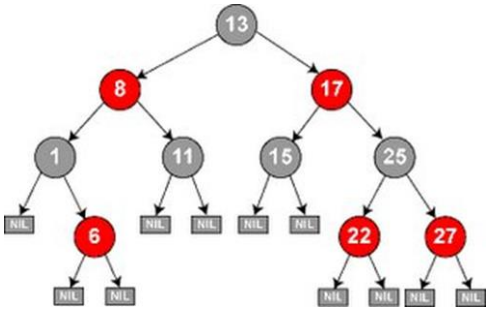
Grafica 3: Tablas hash

3.4 Árbol rojo-negro:
Un Árbol rojo-negro es un árbol binario en el que cada nodo tiene un color como atributo extra, ya sea rojo o negro. El color de los nodos asegura que la trayectoria más larga de la raíz a una hoja no es más larga que el doble del largo de la más corta. Esto significa que el árbol está fuertemente balanceado.

Búsqueda:

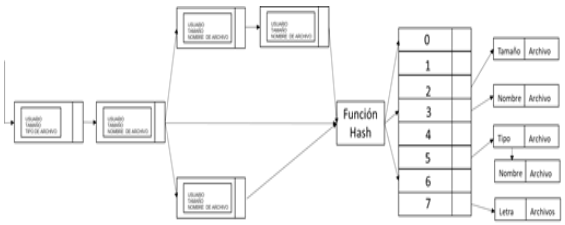
Consiste en acceder a la raíz del árbol y comparar su valor con el valor buscado. Si el elemento a localizar coincide con el de la raíz, la búsqueda ha concluido con éxito. Si el elemento es menor, se busca en el subárbol izquierdo; si es mayor, en el derecho. Si se alcanza un nodo hoja y el elemento no ha sido encontrado se supone que no existe en

el árbol. Cabe destacar que la búsqueda en este tipo de árboles es muy eficiente y representa una función logarítmica.



Grafica 4: Árbol rojo-negro

4. PRIMERA ESTRUCTURA DE DATOS: TREEHASH

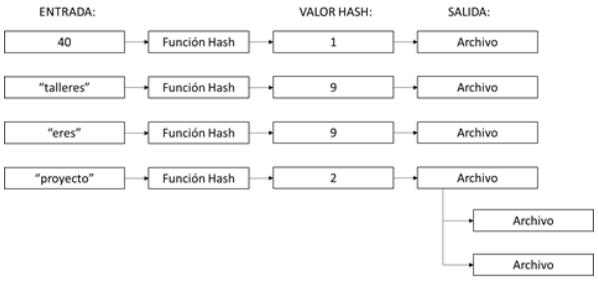


Grafica 5: Lista simplemente enlazada que recopila la información del conjunto de datos, en la cual cada nodo contiene el usuario, tamaño y tipo o nombre del archivo; los datos de la lista pasan a una función hash que les agrega un valor hash para clasificarlos y facilitar la búsqueda.

Enlace Código: https://github.com/AlejandroMllo/ST0245-032/tree/master/proyecto/codigo

4.1 Operaciones de la estructura de datos:

Búsqueda:



Grafica 6: Imagen de operaciones de búsqueda, se puede ver que la búsqueda queda reducida a buscar en los valores hash, y se tiene en cuenta archivos que contienen otros archivos.

4.2 Criterios de diseño de la estructura de datos:

Para diseñar la estructura de datos tuvimos en cuenta la eficiencia en el tiempo, archivos que contiene otros archivos y la comparación entre palabras, es decir palabras o nombres que contengan caracteres parecidos, pero que correspondan a archivos distintos, como por ejemplo buscar la "t" y que en el resultado aparezca tanto "taller" como "trabajo" debido a que ambos empiezan con la misma letra. Debido a lo anterior se implementó una lista simplemente enlazada cuyo nodo final empieza otras listas simplemente enlazadas (ver grafica 5),las cuales recogen todo los archivos del directorio y los jerarquiza, teniendo en cuenta relaciones de herencia, en donde un archivo contiene otros archivos, además cada nodo contiene el usuario, el tamaño y el nombre del archivo; lo anterior permite tratar el problema de la herencia entre archivos, y además optimiza el tiempo al recorrer recolectar y recorrer todo el conjunto de datos, esto permite pasar los datos ya organizados a una función hash, la cual le asigna un valor hash a cada archivo según su tamaño, nombre de archivo y/o tipo de archivo (en el caso de archivos Root), lo cual permite abarcar el problema de archivos que tengan nombres parecidos, porque les asignara un mismo valor hash, lo cual permitirá mostrar los archivos con el mismo nombre, permitiendo abarcar una búsqueda más parecida a los algoritmos utilizados por los buscadores, los cuales muestran las búsquedas parecidas. La anterior combinación entre una lista enlazada y una tabla hash permite optimizar el tiempo, ya que organiza los datos, catalogándolos para una búsqueda más rápida.

4.3 Análisis de Complejidad:

Metodo	Complejidad
Recopilar conjunto de datos	O(1)
Insertar palabras en tabla Hash	O(1)
Busqueda	O(n)

Tabla 1: Reporte de complejidad.

n: Es el parámetro de búsqueda, el cual puede ser usuario, tamaño del archivo, nombre del archivo y tipo de archivo.

4.4 Tiempos de Ejecución:

	Conjunto de Datos 1	*Conjunto de Datos 2*
Creacion	0.0013179243639414946 ms	0.1978782839961042 ms
Operación de Busqueda	2.7441E+11 ms	5.63E+10 ms

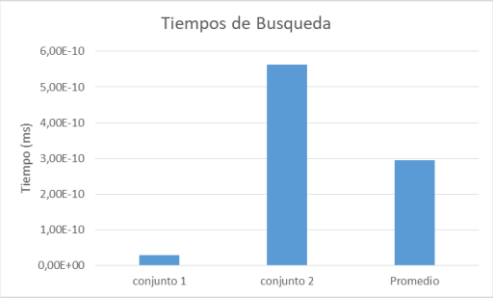
Tabla 2: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos.

4.5 Memoria:

	Conjunto de Datos 1	*Conjunto de Datos 2*
Consumo de memoria	23,5 MB	44,7 MB

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

4.6 Análisis de los resultados:



Grafica 7: Comparación de los tiempo de búsqueda entre conjuntos y su promedio.

Observando la gráfica, podemos concluir que el algoritmo logro el objetivo de optimizar el tiempo de búsqueda.

La estructura de datos generada cumplió con los criterios planteados, ya que como se pudo observar con los datos obtenidos acerca del tiempo de ejecución para los conjuntos de datos de prueba se puede evidenciar un tiempo mínimo para la búsqueda (ver grafica 7), el cual es el principal objetivo para el problema que estamos resolviendo, por lo que se puede decir que el algoritmo optimiza los tiempos de búsqueda. Por otro lado los datos que arrojo el consumo de memoria demuestran un problema, ya que el algoritmo consume mucho recurso para su implementación, esto debido al uso de listas enlazadas, que a su vez generan otras listas enlazadas lo cual genera un gasto de memoria muy grande, pese a la rapidez, por lo que es necesario considerar el gasto de memoria para próximas implementaciones.

REFERENCIAS

1. Agustín J. González. Tablas HASH, ELO320: Estructura de Datos y Algoritmos, (1). Retrieved August 10, 2017, from: http://profesores.elo.utfsm.cl/~agv/elo320/01and02/dataStructures/hashing.pdf
2. Abraham García Soto, Martin Gomez and Antonio Jose. Arboles B, Tutores de Estructuras de Datos Interactivo, (5). Retrieved August 10, 2017, from university of Granada: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arbB.htm
3. Estructura de datos 2. Available from: http://datastruct2.blogspot.com.co/2015/08/arboles-b.html (2015); accessed 10 August 2017.
4. Tutorias arboles. Avaible from: https://sites.google.com/site/tutoriasarboles/arboles-b-y-b (2010); accessed 11 August 2017.
5. DataStructures Tool. Available from: http://www.hci.uniovi.es/Products/DSTool/b/b-queSon.html (2012); accessed 10 August 2017.