Buscador eficiente de archivos y subdirectorios

Eduard Damiam Londoño
EAFIT
Colombia
edlondonog@eafit.edu.co

Esteban Osorio
EAFIT
Colombia
eosorio@eafit.edu.co

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

RESUMEN

El problema es crear una estructura de datos, con la cual podamos por medio de una palabra clave obtener o que se nos brinde información objetiva sobre relacionados de la misma palabra; gracias esto podemos ahorrarnos tiempo de trabajo y de búsqueda al no tener que entrar a cada página para encontrar un dato; varios problemas relacionados son:

La autocompletacíon de texto a la hora de digitar; la predicción de texto en los celulares; el reconocimiento de palabras atreves de audio.

Nuestro trabajo está enfocado al problema de dado el nombre de un archivo o carpeta, verificar que ese archivo exista e imprimir la ruta de dicho archivo, para esto usamos una Tabla de Hash, la cual fue bastante eficiente con tiempos de ejecución menores a 1 seg y una cantidad de memoria gastada inferior a un MB.

Palabras clave
Tabla – Hash – Complejidad- Buscar- Árbol -

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

Un sistema de búsqueda se necesita porque de esta manera nos ahorramos mucho tiempo en la búsqueda de archivos en una base de datos muy grande

En este documento, nos vamos a centrar en esta necesidad del hombre desde hace mucho tiempo atrás para hallar/crear una forma de ahorrarnos más tiempo

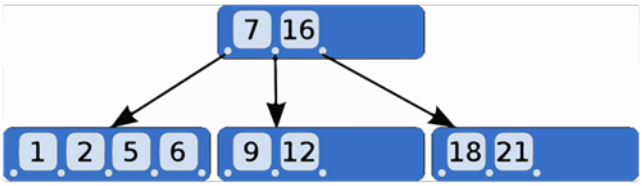
La historia de este problema viene desde 1945, fecha en la cual ya se empieza a tener una idea de la búsqueda automatizada de datos, ya para 1950 esta idea se convierte en descripciones más concretas de cuantos archivos de texto pueden ser buscados automáticamente, emergiendo varios trabajos sobre la idea básica de buscar en un computador. Uno de los métodos más importantes fue usando unidades de palabras e indexación para documento. Posteriormente, en los 60's se creó el Smartsystem, el cual permitió a los investigadores, experimentar que ideas tenían la mejor calidad. Actualmente los sistemas aprovechan las conexiones que hay entre un dato y otro, sobre todo en la web.

2. PROBLEMA

El problema es reducir el tiempo de búsqueda de un archivo usando un algoritmo que use palabras iguales. La razón de solución de este problema es disminuir el tiempo de búsqueda de archivos aumentando la productividad

3. TRABAJOS RELACIONADOS

3.1 Árbol B



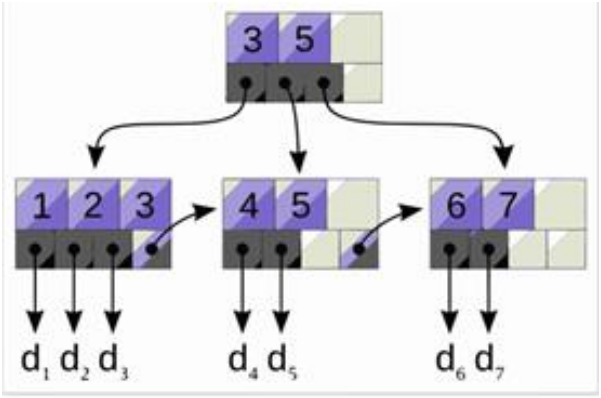
Grafica 1: representación gráfica de un árbol B

El árbol B es un árbol de búsqueda cuyos nodos pueden tener varios hijos, teniendo una relación de orden

entre cada uno o que puede estar vacío, en el peor de los casos la altura del árbol B es n.

A la hora de buscar se empieza en la raíz, y se recorre el árbol hacia abajo, escogiendo el sub-nodo de acuerdo con la posición relativa del valor buscado respecto a los valores de cada nodo

3.2 Árbol B+



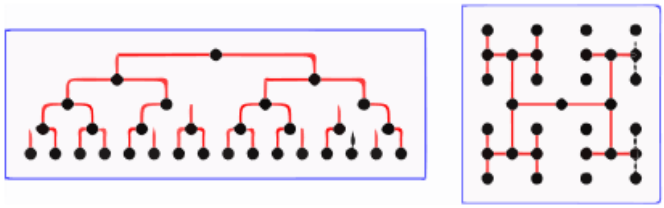
Grafica 2: representación gráfica de un árbol B+

Es una variación del árbol B, en un árbol B+ toda la información se guarda en las hojas. Los nodos internos sólo contienen claves y punteros. Todas las hojas se encuentran

en el mismo nivel, que corresponde al más bajo, su altura en el peor caso es logMn

A la hora de buscar se empieza desde la raíz buscando la hoja que contenga el valor, en cada nodo se busca el puntero interno que se debe seguir, se selecciona el nodo correspondiente mediante la búsqueda en los valores claves del nodo.

3.3 Árbol H

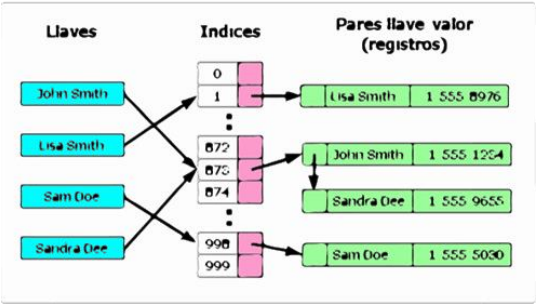


Grafica 3: representación gráfica de 2 árboles H

Es un árbol de datos similar al B Tree con una profundidad constante de 1 o 2 niveles, usan el Hash del nombre del archivo y no requiere balance, sus

algoritmos se diferencian de los algoritmos de los B Tree por la forma en la que tratan las colisiones de hash, que pueden desbordarse a través de varios bloques de hojas.

3.4 Tabla Hash



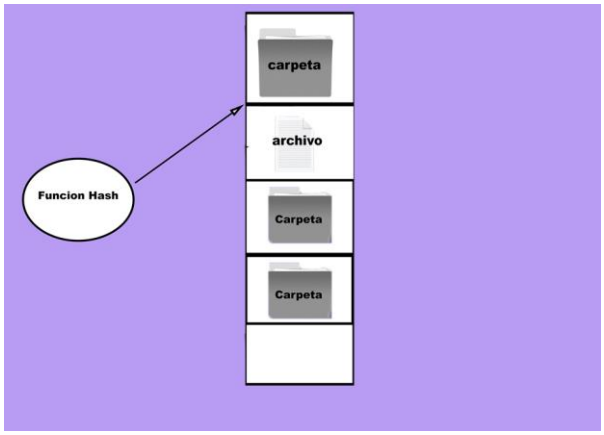
Grafica 4: representación gráfica de una tabla Hash

Asocia llaves o claves con valores, permite el acceso a los elementos almacenados a partir de una clave generada, funciona transformando la clave con una función hash en un hash, un número que identifica la posición donde la tabla hash localiza el valor deseado.

A la hora de buscar es necesario únicamente conocer la clave del elemento, a la cual se le aplica la función resumen, el valor obtenido se mapea al espacio de direcciones de la tabla, si el elemento existente en la posición indicada en el paso anterior tiene la misma clave que la empleada en la búsqueda, entonces es el deseado. Si la clave es distinta, se ha de buscar el elemento según la técnica empleada para resolver el problema de las colisiones al almacenar el elemento.

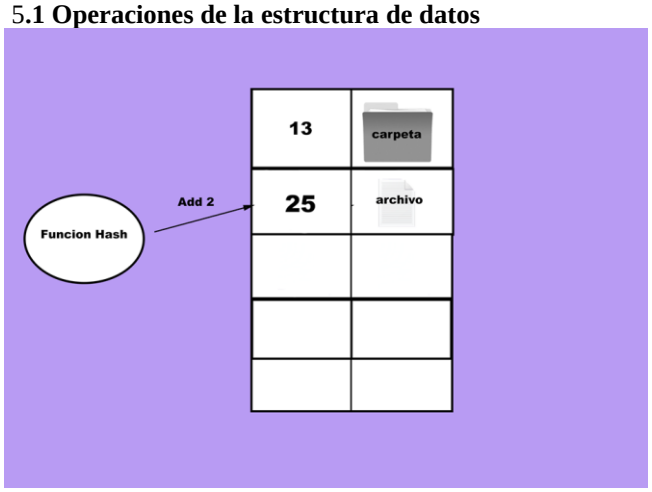
4. Tabla Hash

La estructura que usamos para solucionar el problema es una Tabla Hash, en concreto usamos la por defecto de java.

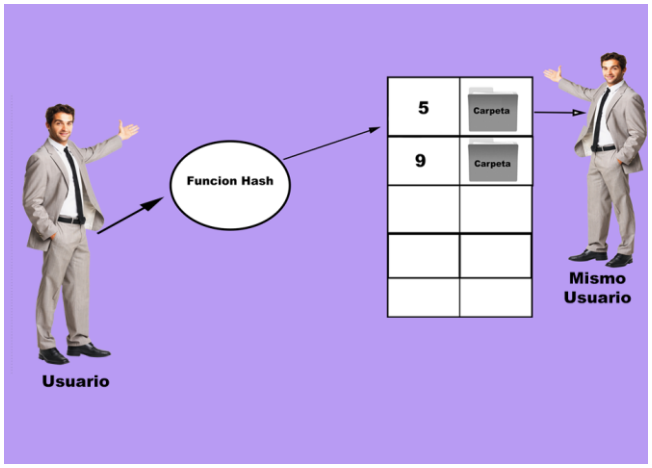


Gráfica 5: Tabla Hash de carpetas y archivos.

5.1 Operaciones de la estructura de datos



Grafica 6: Representación gráfica de la inserción de un archivo o carpeta en una tabla hash, nótese la key a la izquierda.



Grafica 7: búsqueda en una tabla hash

5.2 Criterios de diseño de la estructura de datos

Decidimos diseñar la estructura de datos como una tabla hash, la tabla guarda objetos de tipo Carpeta, a su vez los objetos de Carpeta guardan como atributo la key de su Carpeta padre (siendo un String vacío si no tienen carpeta Padre), de esta manera cuando se obtiene una carpeta, también se obtiene su carpeta padre. Hicimos la estructura así porque pensamos que es la que mejor eficiencia puede alcanzar, las listas por ejemplo simplemente para buscar una sola carpeta tendrían complejidad O(n) y ni hablar de si esta carpeta estuviera dentro de varias carpetas, los mismo para las colas y las pilas, en la tabla hash (estamos asumiendo que la tabla no se deja llenar nunca) buscar una sola carpeta seria O(1) y si esta está dentro de varias carpetas seria O(log (n)).

Cuando la comparamos con los árboles, el acceso en un árbol seria O(log (n)) , y no importa si la carpeta buscada está dentro de otras carpetas, pero donde le gana la tabla hash al árbol es cuando hay archivos o carpetas con el mismo nombre pero en distintas carpetas, en un árbol se tendría que recorrer todo, lo cual es O(n), en la tabla hash lo que hicimos fue que solo cambiamos la forma en la que terminaba la key(la cual es un String), de esta manera con una sola key de un archivo , podemos obtener la de todos los archivos con el mismo nombre y de esta manera no tenemos que recorrer toda la tabla por lo que su complejidad sería menor a O(n) en nuestro caso O(log (n)), más adelante explicamos cómo llegamos a la conclusión de que sería O(log(n)).

5.3 Análisis de la Complejidad

complejidad de las operaciones de la estructura de datos para el peor de los casos.

Sea m la cantidad de elementos con el mismo nombre, n la cantidad total de elementos en la tabla y s la cantidad de caracteres en un String.

Método	Complejidad
Insertar carpeta	O(s+m)
Acceder	O(1)
Imprimir ruta	O(log(n))^2

Tabla 5: Tabla para reportar la complejidad

5.4 Tiempos de Ejecución

	Conjunto de Datos 1	Conjunto de Datos 2	Conjunto de Datos 3
Creación	0.154 sg	0.003 sg	0.018 sg
Buscar	1 sg	0 seg	0 seg

Tabla 6: Tiempos de ejecución de las operaciones de la estructura de datos con diferentes conjuntos de datos

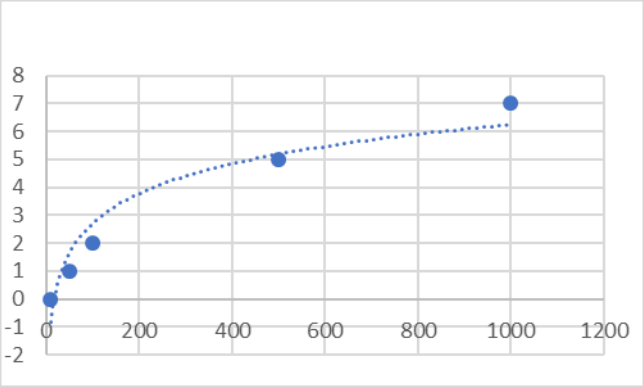
5.5 Memoria

	Conjunto de datos 1	Conjunto de Datos 2	Conjunto de Datos 3
Consumo de memoria	40 KB	29 KB	28 KB

Tabla 7: Consumo de memoria de la estructura de datos con diferentes conjuntos de datos

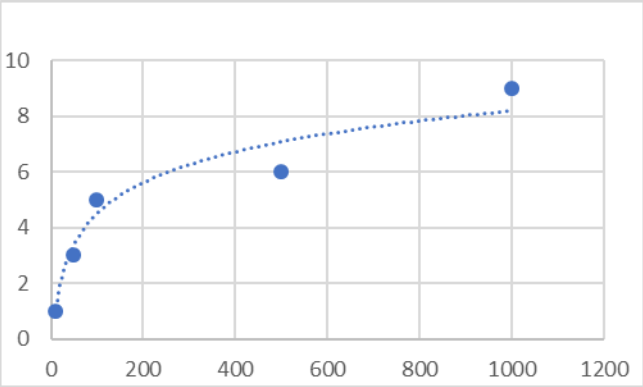
5.6 Análisis de los resultados

Hay 2 cosas que afectan nuestra estructura de datos, la primera es la cantidad de carpetas o archivos con el mismo nombre, pero con distinta carpeta padre (el sistema operativo no deja tener dos archivos con la mismo nombre en una misma carpeta), ya que como la función hash se saca con el nombre, lo archivos con un mismo nombre tendrían la misma función hash y causarían problemas, para solucionar esto, ya que la función hash es un String si se intenta insertar 2 objetos con el mismo hash, se le agrega un numero al final del hash de alguno de los dos, de esta manera si hay dos objetos con el hash 210 uno seria 2101 y el otro 210 y si se intenta agregar un tercero seria 2102. Esto sería un problema ya que buscar así en el peor de los casos seria O(n) siendo n la cantidad total de elementos en la tabla, pero esto solo sería así si por cada carpeta hubiera otra con su mismo nombre, cosa bastante improbable aplicado en la vida diaria. Para verificar eso sacamos el promedio de carpetas u archivos con el mismo nombre en función de la cantidad de carpetas en un pc normal lo cual nos dio la siguiente gráfica:



La cual nos dio log(n), por lo tanto buscar archivos o carpetas repetidas en nuestra estructura seria O(log (n)).

Lo otro que afecta nuestra estructura es la profundidad de archivos, como nuestra búsqueda guarda la carpeta padre de un archivo o de una carpeta como atributo, para obtener la ruta de dicho archivo se necesitaría recorrer todas las carpetas de dicha ruta, para saber cuál sería la complejidad de esto en términos de n, hicimos lo mismo que con la cantidad de archivos repetidos y nos dio la siguiente gráfica:



la cual podemos ver también es logarítmica, por lo tanto, la complejidad de cualquier búsqueda también seria O(log(n)),

ahora en el peor de lo casos ósea teniendo archivos o carpetas con nombres repetidos la complejidad total de imprimir la ruta seria O(log(n)) ^2.

Esto se puede ver en los tiempos de ejecución del conjunto de datos 1 el cual es bastante grande con 2609 carpetas y 62901 archivos, y aun así nos dio 0 seg la ejecución en casi todos los casos y 1 seg en un solo caso.

6. CONCLUSIONES

En conclusión lograr búsquedas cortas es algo necesario en el mundo de hoy en día, cada vez la cantidad de datos que se manejan comúnmente son más numerosos y se necesita rapidez a la hora de consultarlos, por lo que se buscan soluciones a este tipo de problemas, hoy en dia hay muchas estructuras de datos que facilitan esto y pensamos después de lo que logramos alcanzar con la tabla de Hash implementada, que esta es una estructura que puede ser fácilmente implementada y es bastante eficiente para buscar en poco tiempo.

Los que más nos interesaba era el tiempo de ejecución y vimos que estos eran bastante rápidos incluso para grandes cantidades de archivos y para nuestra sorpresa la cantidad de memoria que se gastaba era menor de lo que creímos que Gastaría no alcanzando ni al mega, por lo que gracias a estos resultados concluimos que logramos una muy buena implementación de la tabla Hash

6.1 Trabajos futuros

Hay 2 aspectos principales que nos gustaría mejorar, la primera es intentar de lograr un O(log(n)) de alguna manera ya que pensamos que debería ser posible bajar ese O(log (n))^2, implementando un mejor método para manejar las carpetas repetidas.

El segundo aspecto es la lectura del archivo y la creación de la tabla, creemos que nuestro algoritmo para leer los archivos y guardarlos es bastante deficiente, y también que debería ser posible saber según el peso del archivo .txt cuantas carpetas aproximadamente serán creadas, de esta manera podemos darle un tamaño aprox a la tabla desde el principio y no tener que estar aumentando su tamaño.

AGRADECIMIENTOS

Esta investigación fue soportada parcialmente por el ICETEX.

REFERENCIAS

1. Anon. 2017. HTree. (June 2017). Retrieved August 13, 2017 from https://en.wikipedia.org/wiki/HTree
2. Anon. 2017. Tabla hash. (July 2017). Retrieved August 13, 2017 from https://es.wikipedia.org/wiki/Tabla_hash
3. Anon. 2017. B tree. (August 2017). Retrieved August 13, 2017 from https://en.wikipedia.org/wiki/B%2B_tree
4. Anon. 2017. Árbol-B. (July 2017). Retrieved August 13, 2017 from https://es.wikipedia.org/wiki/%C3%81rbol-B