ESTRUCTURA DE DATOS PARA UNA EFICIENTE BÚSQUEDA DE ARCHIVOS.

Luisa María Vásquez Gómez
Universidad Eafit
Colombia
lmvasquezg@eafit.edu.co

Santiago Escobar Mejía
Universidad Eafit
Colombia
sescobarm@eafit.edu.com

Sebastián Giraldo Gómez
Universidad Eafit
Colombia
sgiraldog@eafit.edu.co

RESUMEN
En este entregable se analizan y comparan las diferentes soluciones planteadas al problema de la eficiente búsqueda de archivos y subdirectorios en un directorio especifico, para lo cual se presentan varias alternativas de solución, y después de plantear una solución parcial y analizar sus resultados se llega a una solución final que resuelva eficazmente la dificultad propuesta.

Palabras claves

Estructura de datos; listas; búsqueda; inserción; directorio; árbol binario; tiempo; operación; complejidad; diseño; árbol rojo-negro.

Palabras claves de la clasificación ACM

- Theory of computation ~ Design and analysis of algorithm
- Theory of computation ~ Analysis of algorithms and problem complexity
- Information systems ~ Data management systems
- Software ~ Software notations and tools

1. INTRODUCCIÓN

Desde hace siglos el hombre ha buscado comprender su entorno para poder desarrollarse más eficientemente en este, y ha buscado una forma de guardar esta información de manera que otros tuvieran acceso a ella en el futuro; con la revolución tecnológica dada a finales del siglo XX apareció una nueva forma: las estructuras de datos; estas permitían guardar grandes cantidades de información sin necesidad de utilizar un gran espacio físico ni realizar un considerable gasto de recursos ; además llegaron en el momento justo, ya que junto con ellas se dio una "revolución de la información" lo cual permitió a la gran mayoría de la población mundial tener acceso a cualquier tipo de conocimiento con tan solo un click; en tiempos como estos, una manera de organizar esta gran cantidad de datos era urgentemente necesaria .

2. PROBLEMA

En una era donde la información está al alcance de cualquier persona con acceso a internet, se encuentra la necesidad de permitir a los usuarios ver de manera organizada todos los datos que se encuentran en determinado sistema y asimismo poder buscar de manera rápida y eficiente por un archivo especifico; logrando así que el manejo de la información sea más fácil y estructurado para quien desee acceder a ella.

3. PROBLEMAS RELACIONADOS Y POSIBLES SOLUCIONES

A continuación, se mencionan y explican diferentes tipos de estructuras de datos que han sido utilizadas para la resolución de problemas similares al anteriormente mencionado.

3.1 Listas lineales

Una lista es una secuencia de elementos del mismo tipo, de cada uno de los cuales se puede decir cuál es su siguiente (en caso de existir). Son de las estructuras de datos más simples y están compuestas por valores "clave " que representan cada elemento que se encuentra dentro de sí; son usadas para representar directorios pequeños ya que su única forma de encontrar entradas es mediante la búsqueda lineal (pasando por cada elemento de la lista). Pueden ser implementadas mediante las siguientes estructuras:

3.2 Árboles B+

Los arboles B+ nacen de la necesidad de organizar la información de manera equilibrada, es decir, que, así como los arboles B controlan su alargamiento o acortamiento dependiendo de los elementos que son insertados o borrados de estos, lo cual permite que cualquier camino desde la raíz hasta un valor clave sea de la misma longitud.

El "orden " de un árbol B+ indica que cada nodo puede contener "n" claves y "n-1" referencias a otros nodos, como ejemplo se presenta a continuación un árbol B+ de orden 5:

3.3 Árbol rojo-negro

Son arboles que nacen (al igual que los arboles B) de la necesidad de hacer que cada ruta desde la raíz hasta cualquier nodo hoja tenga la misma eficiencia, este árbol en particular, resuelve el problema mediante una notación por colores de cada nodo, cumpliendo con las siguientes condiciones:

- La raíz y los nodos hoja son negros
- Un nodo rojo tiene dos hijos negros
- La ruta entre la raíz y cualquier hoja pasa por la misma cantidad de nodos negros

Esta estructura de datos será retomada más adelante en este entregable.

3.4 Tabla Hash

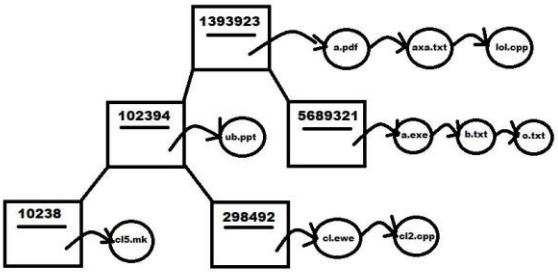
Las Tablas hash son un tipo de estructura de datos que permiten al sistema acceder a determinada entrada de un directorio sin necesidad de mirar todas las existentes en él.

La función "hash" es la que se encarga de convertir un valor clave en un número con el cual se identifica cada entrada particular.

Las tablas hash solucionan la problemática de hacer cada vez más eficiente la búsqueda de determinada entrada en un directorio ya que no se debe hacer una revisión lineal de cada elemento en esta, lo cual impide tener acceso basado en el orden en el que se encuentran las entradas.

4. ÁRBOL DE BÚSQUEDA BINARIA CON LISTAS ENLAZADAS

Para el problema planteado diseñamos una combinación de arboles binarios y listas enlazadas; cada fichero o carpeta se almacena en un Árbol Binario de Búsqueda de acuerdo a una "llave" que se genera de acuerdo a su nombre y a su vez como atributo posee una lista enlazada que contiene los archivos o ficheros que van dentro de ella.

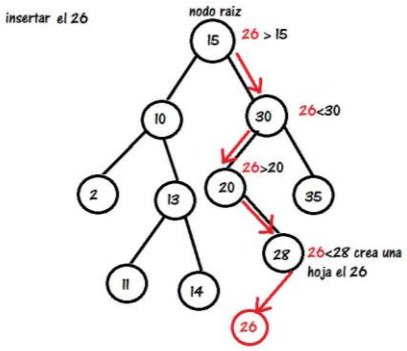


Gráfica 1. Estructura de datos diseñada, basada en árbol de búsqueda binaria y lista enlazada.

4.1 Operaciones

4.1.1 Inserción

Para insertar un fichero, el programa genera una "clave" según su nombre y basado en si esta es mayor o menor a las existentes en el árbol la ubica en este último, por otro lado, para insertar un archivo se debe primero usar la función "buscar" del árbol para encontrar la carpeta a la cual se desea añadir el archivo, una vez encontrada, se añadirá a la lista enlazada de la carpeta.



Gráfica 2. Inserción en árbol de búsqueda binaria.

4.1.2 Búsqueda

Para buscar una carpeta en el árbol B solo es necesario en nombre de esta, ya que con este se puede generar la clave y observando si esta es mayor o menor a los valores ya existentes en el arbolo se puede determinar la ruta directa a esta .Para buscar un archivo se debe en primer lugar buscar la carpeta a la cual pertenece con el método mencionado anteriormente, después, proporcionando el nombre del archivo se procederá a buscar en la lista enlazada de la carpeta el archivo deseado pasando por cada uno de los existentes.

4.1.3 Eliminación

Para eliminar una carpeta del árbol B se deben tener en cuenta los posibles 3 casos: que sea un nodo hoja, con un hijo o con dos hijos:

- Si es un nodo hoja simplemente se elimina su referencia con el nodo padre
- Si tiene un hijo, este toma el lugar del nodo a ser eliminado
- Si tiene dos o más hijos, el hijo de menor valor en el sub-árbol que quedará sin nodo padre tomará el lugar de este último, conservando así la jerarquía de menor-mayor del árbol.

4.2 Criterios de diseño de la estructura de datos

Para el diseño de esta estructura de datos nos fijamos principalmente en la eficiencia de las operaciones y encontramos que el árbol de búsqueda binaria tenía una complejidad de O(log n) para el promedio de los casos, pero debido a que no íbamos a manejar solo un tipo de dato vimos la necesidad de utilizar otra estructura de datos como las listas enlazadas, ya que estas últimas nos permiten separar y agrupar los archivos de una carpeta determinada en lugar de tenerlos todos desordenados como sucedería en varios árboles, y trabajando la idea llegamos a la desarrollada en este entregable.

4.3 Complejidad y tiempo de operaciones

Operación	Caso promedio	Peor caso	Tiempo 1	Tiempo 2	Tiempo promedio
Inserción	O(log n)	O(n)	5 seg	10 seg	8 seg
Búsqueda	O(log n)	O(n)	3 seg	9 seg	5 seg
Eliminación	O(log n)	O(n)	7 seg	15 seg	9 seg

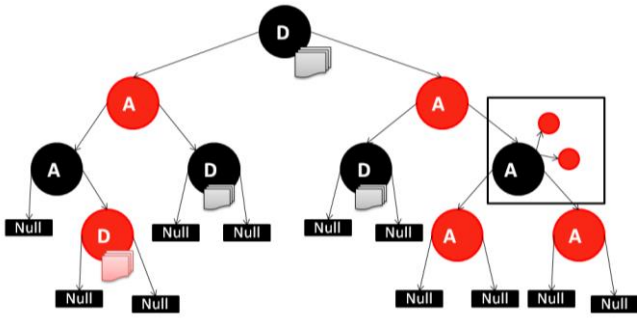
Gráfica 3.Complejidad algorítmica de las operaciones de la estructura de datos y tiempos con dos diferentes conjuntos de datos .

4.5 Análisis de resultados

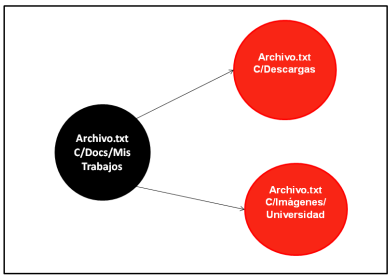
Basados en los resultados obtenidos concluimos que, a pesar de ser una estructura de datos eficiente, se podría realizar de manera que tome menos tiempo realizar las operaciones con grandes cantidades de datos y asimismo sea mas fácil el acceso a archivos sin necesidad de pasar por dos búsquedas.

5. ESTRUCTURA FINAL: ÁRBOL ROJO-NEGRO

Con el fin de mejorar la complejidad de la estructura de datos con respecto a la solución planteada anteriormente, se diseño un árbol rojo negro que contenga todos los archivos deseados y pueda realizar una búsqueda rápida y eficiente, una inserción con manejo de colisiones y un eficaz listado de contenidos .



Gráfica 4. Representación grafica de la estructura de datos diseñada; las "A" representan ficheros y las "D" directorios.



Gráfica 5. Árbol que maneja las colisiones en la estructura de datos.

5.1 Operaciones en la estructura de datos

5.1.1 Inserción

La inserción en el árbol rojo negro es similar a la de un árbol binario de búsqueda (trabajo en el numeral anterior); se identifica si el elemento a insertar es mayor o menor que la raíz, si es menor, se ubica al lado izquierdo de esta, de lo contrario, a la derecha, y así recursivamente. Sin embargo, el árbol rojo negro tiene la particularidad de balancear el árbol de manera que la altura del sub árbol derecho no difiera en más de una unidad de la del sub árbol izquierda, manteniendo las siguientes condiciones:

- La raíz y los nodos hoja (incluyendo las nulas) son negros.
- Un nodo rojo tiene dos hijos negros
- La ruta entre la raíz y cualquier hoja pasa por la misma cantidad de nodos negros

En la estructura diseñada, además de lo anteriormente mencionado, se añadió la funcionalidad de que cada vez que se añada un fichero o directorio, se agrega su nombre a la lista de contenidos de su directorio padre, de modo que al listar su contenido no sea necesario recorrer el árbol en su totalidad. También, para evitar problemas al momento de agregar dos archivos con igual nombre, cada nodo que el árbol posee es raíz de un "sub árbol" que contiene todos los ficheros o directorios con su mismo nombre, de manera que al buscar un archivo con ese nombre, se puedan diferenciar por medio de su ruta.

5.1.2 Búsqueda

Al igual que en un árbol binario de búsqueda, para encontrar un elemento en el árbol se identifica si el elemento a buscar es igual a la raíz, si así es, se concluye la búsqueda, de lo contrario, si es menor a esta se procede a buscar en el subárbol izquierdo, o si es mayor, se busca en el subárbol derecho; así se logra describir una función logarítmica, lo cual hace que encontrar un elemento se haga de la manera más rápida posible.

En el caso de haber más de un archivo con el nombre buscado, se listarán los existentes con sus respectivas rutas de manera que el usuario sepa cuáles son los que existen en la estructura de datos.

También se puede realizar una búsqueda por ruta, que funciona análogamente a la búsqueda por nombre y da la opción a usuario de listar los elementos en dicho directorio, sin embargo, se le advierte que realizar esta acción implicaría una complejidad de O(n) y podría tardar más tiempo, pero si el usuario no lo desea, la complejidad de la búsqueda continuaría siendo O (log n).

5.2 Criterios de diseño de la estructura de datos

La nueva estructura de datos fue diseñada con el fin de reducir los tiempo de ejecución y la complejidad respecto a la anterior, por lo cual buscamos un árbol similar al de búsqueda binaria pero que fuera más eficaz a la hora de buscar elementos y que fuera balanceado; varias opciones que surgieron fueron Árbol B, Árbol B+, Árbol AVL y Árbol KD,casi todas ellas nos permitían una mayor eficacia en la búsqueda y menor complejidad en las operaciones, sin embargo, decidimos escoger una que nos permitiera continuar con un proceso gradual y no empezar desde cero, y así decidimos implementar el árbol Rojo Negro. Nos ayudamos de la herramienta TreeMap implementada en java y le añadimos funcionalidades que consideramos necesarias como el manejo de colisiones y el listado de contenidos para que la estructura fuera más completa.

5.3 Análisis de complejidad

Operación	Caso promedio	Peor de los casos
Inserción	O(log n)	O(log n)
Búsqueda	O(log n)	O(log n)

Gráfica 6. Complejidad algorítmica de las operaciones de la nueva estructura de datos.

5.4 Tiempos de ejecución y memoria

	Datos 1	Datos 2	Promedio
Inserción(Tiempo)	0.006 s	0.015 s	0.0105 s
Búsqueda(Tiempo)	0.008 s	0.012 s	0.0235 s
Consumo de memoria	1,21 MB	9,53 MB	5,37 MB

Gráfica 7. Tiempos de ejecución de la estructura de datos junto con su consumo en memoria.

5.5 Análisis de resultados

Observando los resultados obtenidos, evidenciamos un gran avance con respecto a la estructura de datos anteriormente diseñada ya que funciona más rápida y eficientemente para grandes cantidades de datos tanto en la lectura del archivo como en la búsqueda, esto se debe a la disminución de la complejidad algorítmica de la estructura, lograda al implementar el árbol rojo-negro. Además, esta nueva estructura permite más opciones para buscar (por nombre o ruta) y además maneja colisiones y listado de contenidos de directorios.

6. CONCLUSIONES

En este proyecto nos formamos altamente en los diferentes tipos de estructuras de datos existentes, sus ventajas, desventajas y posibles casos de uso, además aprendimos a diseñar una estructura basada en varias ya existentes utilizando los conocimientos adquiridos en el curso de Estructuras de Datos y Algoritmos I. Al finalizar este proyecto obtuvimos una gran satisfacción al lograr que nuestra estructura de datos manejara grandes cantidades de archivos de entrada sin afectar en gran parte su tiempo de ejecución, tanto en la inserción como en la búsqueda, a diferencia de la primera estructura que se demoraba bastante en estos procesos y no era una solución optima al problema planteado. Por último, nos gustaría continuar desarrollando la estructura de datos para hacerla cada vez más apta para las personas que la podrían utiliza en el día a día, ya sea añadiéndole interfaz grafica, autocompletación en la búsqueda de archivos, entre otros. Propiciando así un software de mayor calidad a los usuarios.

6.1 Trabajos futuros

Tenemos miras a convertir esta estructura de datos en algo más completo y aplicable al mundo real, añadiéndole búsquedas por otros atributos, filtros, organización de datos por orden alfabético o peso, entre otras. De manera que logremos una estructura que pueda ser de gran provecho para cualquier tipo de datos que se deseen almacenar, ya sean documentos, imágenes, programas, videos, etc. y pueda ser usada por cualquier usuario para su vida cotidiana.

AGRADECIMIENTOS

Agradecemos a Alejandro Cano, Luis Javier Palacios y Jorge Luis Herrera por su ayuda en la comprensión de conceptos nuevos y en el diseño de la estructura de datos.

Este entregable fue parcialmente soportada por: Alcaldía de Medellín, Fondo EPM y MinTIC.

REFERENCIAS

1. Adkins, A. and Gonzalez-Rivero, J. Directory and Indez Data Structures. Franklin W Olin College of Engineering. Recuperado de: https://www.youtube.com/watch?v=1ZZV9QhGUmQ
2. Anderson-Fredd, S. B+ Trees. Baidu. Recuperado de: https://www.sci.unich.it/~acciaro/bpiutrees.pdf
3. Franco, E. Estructuras de Datos: Tema 5: Tablas Hash. Instituto Politecnico Nacional. Recuperado de: www.eafranco.com/docencia/estructurasdedatos/files/05/Tema05.pdf
4. Martinez, P., Sanchez, J., and Gallardo, C.Listas. Estructuras de datos,92-142.Recuperado de: http://ocw.upm.es/lenguajes-y-sistemas-informaticos/estructuras-de-datos/contenidos/tema3nuevo/Listas.pdf
5. Vaca, C..Estructuras de datos y algoritmos. Universidad de Valladolid. Recuperado de: https://www.infor.uva.es/~cvaca/asigs/doceda/rojonegro.pdf
6. Wikipedia, The Free Encyclopedia. "Hash Tables ". Recuperado de: https://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=5139350
7. Burnett, C. Red-Black Tree Example. Wikipedia: The Free Encyclopedia. Recuperado de: https://commons.wikimedia.org/wiki/File:Red-black_tree_example.svg