ESTRUCTURA DE DATOS PARA LA BÚSQUEDA DEL CONTENIDO DE UN DIRECTORIO

Santiago Soto
Universidad Eafit
Colombia
ssotom@eafit.edu.co

Andrés Sánchez
Universidad Eafit
Colombia
asanchezc@eafit.edu.co

Jamerson Correa
Universidad Eafit
Colombia
jscorreac@eafit.edu.co

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

RESUMEN

El problema de buscar el contenido de un directorio consiste en encontrar eficientemente los archivos y subdirectorios que se encuentran en un directorio. Por ello y por la importancia de la búsqueda de información en los sistemas que manejan grandes cantidades de datos se ha propuesto, como proyecto de clase, encontrar una estructura de datos eficiente para la solución de dicho problema.

Para solucionar este problema se implementó una estructura de datos de tipo árbol. Después de comparar los tipos de árboles y sus complejidades en tiempo de inserción y búsqueda, se llegó a la conclusión de que la mejor opción era un árbol autobalanceable AVL.

Al comenzar la ejecución del programa se lee un archivo de texto que represente el contenido de un directorio cualquiera y se genera un árbol balanceado organizado lexicográficamente, lo que facilita la búsqueda en gran manera reduciendo las posibilidades de recorrer subárboles innecesariamente.

Palabras clave

Estructuras de datos, Lectura de archivos, Árbol de búsqueda , Recorrido recursivo.

Palabras clave de la clasificación de la ACM

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

1. INTRODUCCIÓN

Los sistemas actuales manejan grandes y extensas cantidades de información contenida en archivos. El reto de generar una estructura de datos que distribuya y acomode ésta cantidad de archivos se ha convertido en un problema para los desarrolladores de software para lograr la mayor eficiencia en todos sus sistemas.
En el desarrollo de este proyecto se buscará desarrollar un eficiente motor de búsqueda de archivos y subdirectorios de un directorio determinado.

Para la solución de la búsqueda de archivos durante el avance de la tecnología, se diseñó el modelo ext2 y se desarrolló hasta llegar hasta el ext4. El problema con la primera versión de ésta solución no soportaba las cantidades de archivos de los sistemas actuales. Lo que generó su reemplazo por el ext3, pero nuevamente, el avance tecnológico requería utilizar nuevas características no compatibles con el ext3, dando origen al ext4, desarrollado por Google.

2. PROBLEMA

En un sistema donde se requiere manejar grandes cantidades de información almacenada en archivos o subdirectorios, buscar la eficacia a la hora de encontrar uno de estos archivos permitiendo mayores facilidades al usuario, tales como filtros de búsqueda, es el reto que se busca solucionar para optimizar todos los sistemas de información para cualquier cliente, ya sea una compañía que maneje un gran volumen de información o un usuario con un computador personal doméstico.

3. TRABAJOS RELACIONADOS

3.1 Árbol AA

En informática un árbol AA es un tipo de árbol binario de búsqueda auto-balanceable utilizado para almacenar y recuperar información ordenada de manera eficiente. Los árboles AA reciben el nombre de su inventor, Arne Andersson.
Los árboles AA son una variación del árbol rojo-negro, que a su vez es una mejora del árbol binario de búsqueda. A diferencia de los árboles rojo-negro, los nodos rojos en un árbol AA sólo pueden añadirse como un hijo derecho. En otras palabras, ningún nodo rojo puede ser un hijo izquierdo

3.2 Árbol de segmentos

Es una estructura de datos en forma de árbol para guardar intervalos o segmentos. Permite consultar cuál de los segmentos guardados contiene un punto. Este es, en principio, una estructura estática; es decir, su contenido no puede ser modificado una vez que su estructura es construida.

3.3 Árbol AVL

Es un tipo especial de árbol binario ideado por los matemáticos rusos Adelson-Velskii y Landis. Fue el primer árbol de búsqueda binario auto-balanceable que se ideó.

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.

3.4 Árbol k ario

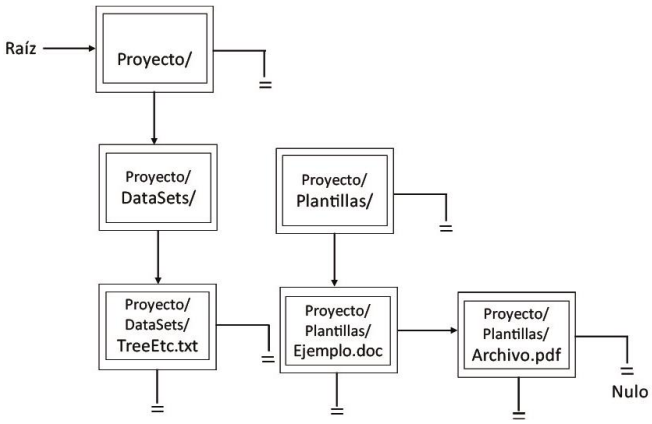
un árbol k-ario es un arraigado árbol en el que cada nodo no tiene más que hijos k. También es conocido a veces como una manera de árbol-k, un árbol N-ario, o un árbol M-ario.

Un árbol k-ario completo es un árbol k-ario donde cada nodo en el mismo nivel 0 tiene hijos k.

4. ESTRUCTURA 1: ÁRBOL GENERAL

Implementación personal de una estructura tipo árbol conformada por un conjunto de datos de tipo nodo que representará cada directorio o fichero dentro de la estructura, donde cada nodo tendrá relaciones con otros nodos, nodo padre, que será el directorio contenedor del nodo actual; el nodo hijo, que será un subdirectorio o un fichero del nodo actual; y un nodo hermano, que son directorios o ficheros dentro del mismo directorio. En esta estructura el único nodo sin nodo padre o hermano será el nodo raíz.

Diseño de la estructura de datos:

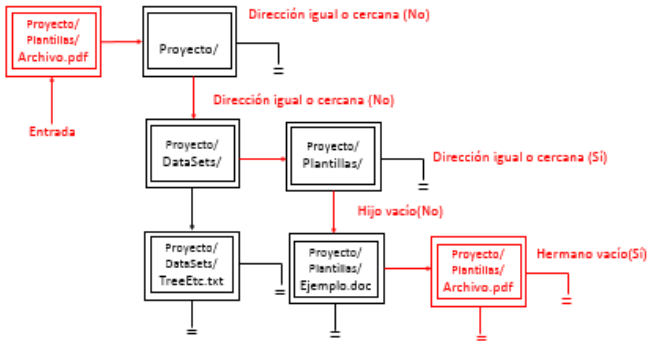


Gráfica 1: Árbol general de archivos. Un archivo es una clase que contiene nombre y dirección.

4.1 Operaciones de la estructura de datos

4.1.1 añadir nodo: añadir fichero o directorio a la estructura.

Esta operación verifica si un nodo, en este caso iniciando desde el nodo padre, posee o no hijos, en caso de no poseerlos éste se encarga de añadirlos haciendo un llamado recursivo.



Gráfica 2: Imagen de la operación de agregar nodo a el árbol general.

4.2 Criterios de la estructura

La propiedad de las estructuras tipo árbol de jerarquizar los datos, hacen de la estructura utilizada la apropiada para generar la correcta asignación de subdirectorios y ficheros a cada directorio, por otro lado, se asemeja a la estructura de la distribución dada por un sistema operativo de cada dato dentro del disco duro. Además, con el llamado recursivo para las funciones de nuestra estructura, se reducen notablemente la complejidad y por ende el tiempo de ejecución de cada funcionalidad de la estructura.

4.3 Análisis de las complejidades

Funcion	Complejidad
addNodo	O(n)

Tabla 1. Complejidad de las operaciones de la estructura de datos 1.

4.4 Tiempo de ejecución

Funcion	ejemplito.txt	treeEtc.txt	juegos.txt
Creación	0,3 ms	33,19 ms	26 s

Tabla 2. Tiempos de ejecución de las operaciones de la estructura de datos 1 con diferentes conjuntos de datos.

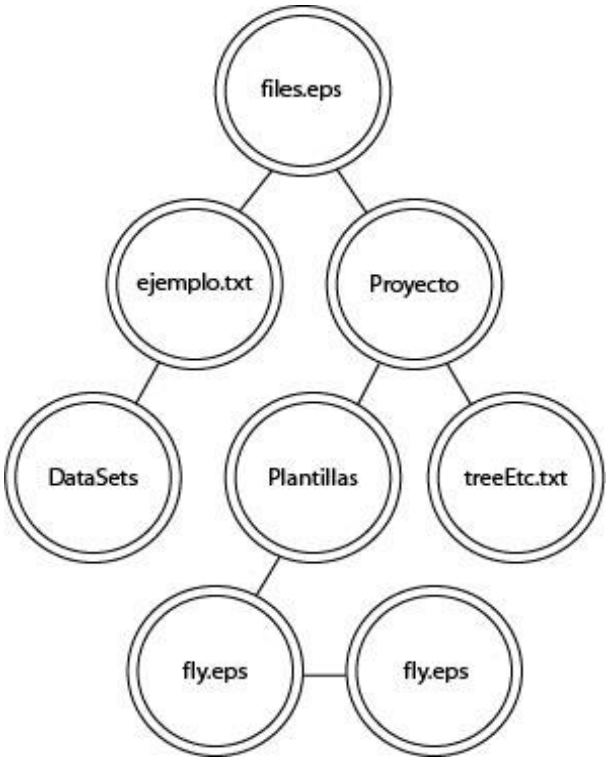
4.5 Memoria usada

Estructura	ejemplito.txt	treeEtc.txt	juegos.txt
Memoria	4.7 Mb	13.3 Mb	87.3 Mb

Tabla 3. Memoria del equipo ocupada por la estructura de datos 1 con diferentes conjuntos de datos.

5. ESTRUCTURA FINAL: ÁRBOL AVL

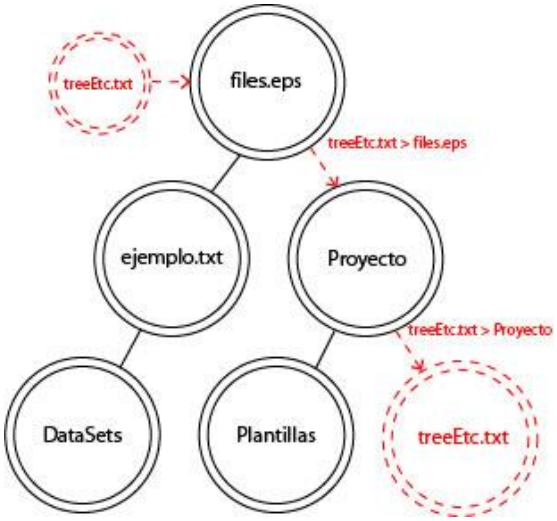
Ésta es una implementación genérica de un árbol binario de búsqueda auto-balanceable en donde se almacena la información (nombre y dirección) de cada archivo en orden alfabético respecto a su nombre, teniendo en cuenta la posibilidad de varios archivos con el mismo, en dicho caso se hace uso de un nodo auxiliar adyacente al nodo con el mismo nombre.



Gráfica 3: Árbol AVL de archivos. Un archivo es una clase que contiene nombre y dirección.

5.1 Operaciones de la estructura de datos

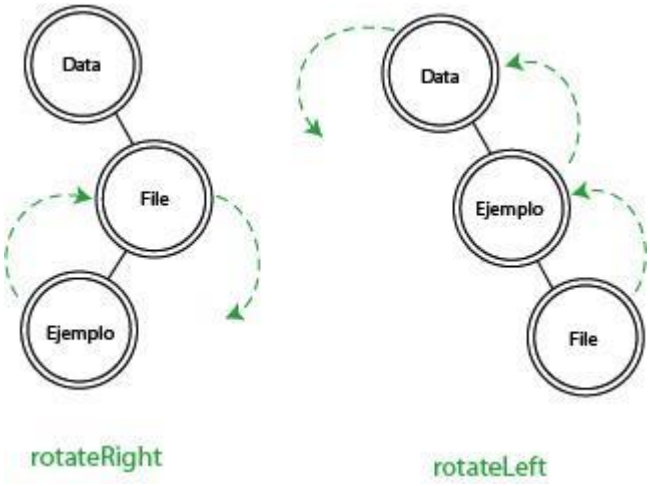
5.1.1 Añadir nodo: Archivo a la estructura. En ésta operación se analiza el nombre archivo ubicado en la raíz del árbol a analizar y se compara con el nombre del archivo a insertar, si se cumple que lexicográficamente el nombre a insertar es menor que el nombre de la raíz, se procederá a recorrer recursivamente el subárbol izquierdo de la raíz, de lo contrario se cumple que lexicográficamente es mayor se recorrerá recursivamente el árbol derecho de la raíz, y se encuentra algún fichero o directorio con el mismo nombre lo agregara adyacente al nodo.



Gráfica 4: Imagen de la operación de insertar nodo en el árbol AVL.

5.1.2 Rotaciones:

Una rotación restablece el equilibrio del árbol. Pueden darse dos casos: rotación simple o rotación doble; a su vez ambos casos pueden ser hacia la derecha o hacia la izquierda.



Gráfica 5: Imagen de las operaciones de rotación en el árbol AVL.

5.2 Criterios de diseño de la estructura de datos

Los árboles AVL son árboles de búsqueda binarios con excelentes garantías de rendimiento gracias a la restricción en la ubicación de cada nodo, lo que permite un árbol en todo momento equilibrado y balanceado respecto a su orden lexicográfico lo que permite lograr un tiempo de ejecución correspondiente a la complejidad O(log(n)).

Respecto a otros árboles auto-balanceables, la profundidad del árbol AVL es mucho menor lo que genera más eficacia al momento de realizar operaciones en volúmenes de datos más grandes como insertar o eliminar un dato. Y teniendo en cuenta que toda la memoria se puede asignar dinámicamente no es necesario conocer la cantidad de datos antes de la creación de la estructura de datos.

5.3 Análisis de la Complejidad

Funcion	Complejidad
rotateLeft	O (1)
rotateRight	O (1)
insert	O(log n)
search	O(log n)

Tabla 4. Complejidad de las operaciones de la estructura de datos 2.

5.4 Tiempos de Ejecución

Función	ejemplito.txt	treeEtc.txt	juegos.txt
Creación	1,6 ms	14,7 ms	0,24 s

Tabla 5. Tiempos de ejecución de las operaciones de la estructura de datos 2 con diferentes conjuntos de datos.

5.5 Memoria

Estructura	ejemplito.txt	treeEtc.txt	juegos.txt
Memoria	4.3 Mb	11.6 Mb	77.4 Mb

Tabla 6. Memoria del equipo ocupada por la estructura de datos 2 con diferentes conjuntos de datos.

5.6 Análisis de los resultados

Búsqueda	ejemplito.txt	treeEtc.txt	juegos.txt
“fly.eps”	0,021 ms	0,013 ms	0,019 ms
“log.xml”	0,010 ms	0,014 ms	0,014 ms
“SDL.h”	0,014 ms	0,012 ms	0,017 ms

Tabla 5. Tiempos de búsqueda de la estructura de datos 2 con diferentes conjuntos de datos.

6. CONCLUSIONES

Este documento presenta el proceso y evolución de la solución en el problema de buscar información en grandes volúmenes de datos en un computador realizando un análisis para determinar el tipo de estructura de datos más óptima para el desarrollo de la problemática.

Luego de concluir que la mejor adaptación para la búsqueda de ficheros y directorios es una estructura tipo árbol, se comenzaron a implementar varias soluciones con distintos tipos de esta estructura y realizando comparaciones entre tiempo de ejecución, consumo de memoria y complejidades de las operaciones de búsqueda e inserción de cada implementación. llegando al resultado final, un árbol auto-balanceable AVL, que redujo notablemente los tiempos de ejecución de los algoritmos.

6.1 Trabajos futuros

A pesar de los favorables resultados de la implementación para los conjuntos de datos dados establecidos para el desarrollo de proyecto, para realizar un trabajo óptimo que se acomode a las necesidades de cada usuario, se podría implementar más opciones de búsqueda y filtro, para el usuario, por ejemplo, busque por tamaños y dueño, tipo de archivo entre otros.

REFERENCIAS

1. Árbol AA. Retrieved August 20, 2017 from http://www.wikiwand.com/es/%C3%81rbol_AA
2. Árbol de segmento. (September 2017). Retrieved September 20, 2017 from https://es.wikipedia.org/wiki/%C3%81rbol_de_segmento
3. Árbol AVL. Retrieved September 20, 2017 from https://es.wikipedia.org/wiki/%C3%81rbol_AVL
4. Árbol k-ario. Retrieved September 20, 2017 from. https://es.wikipedia.org/wiki/%C3%81rbol_k-ario