UMAG Universidad de Magallanes | Facultad de Ingeniería

Fly: Motor de busqueda

Franco Aguilar[†], **Milton Hernandez**[†], **Iván Mansilla**[†] **y Ayrton Morrison**[†]

[†] *Universidad de Magallanes*

Este informe fue compilado el 17 de noviembre de 2024

Resumen

En el presente informe se presentará un sistema de recuperación de información tipo motor de búsqueda, el cual indexa documentos web con hipervínculos a otros documentos a través de las técnicas de índice invertido y PageRank.

Keywords:

Índice

1. Introducción

Fly es un motor de búsqueda que indexa documentos web con hipervínculos a otros documentos, utilizando técnicas como el **índice invertido** y **PageRank**. Un sistema de recuperación de información es una herramienta que permite buscar y recuperar datos en una colección de archivos. Esto se utiliza en aplicaciones como buscadores web y bases de datos.

Las principales funciones de **Fly** incluyen:

- Procesar documentos de un directorio y extraer palabras clave.
- Crear un índice de los archivos que contienen las palabras clave.
- Calcular el nivel de importancia de cada archivo (PageRank).
- Listar los archivos que contienen la palabra buscada, ordenados por importancia.

Para implementar estas funciones, se utilizaron estructuras de datos como grafos, listas enlazadas y tablas hash: Los grafos representan las relaciones entre documentos, las listas enlazadas permiten una navegación sencilla entre elementos relacionados, las tablas hash son esenciales para la búsqueda eficiente de palabras clave.

Este informe describe el proceso de desarrollo e implementación de Fly, detallando las decisiones de diseño y las técnicas utilizadas para crear un sistema eficiente de recuperación de información.

1.1. Índice invertido

El índice invertido es una estructura de datos que almacena los archivos que contienen un contenido específico, en lugar de listar los contenidos de un archivo. Esto es útil para trabajar con grandes volúmenes de información, ya que facilita la búsqueda y la relación entre los archivos y su contenido.

1.2. PageRank

PageRank es un algoritmo que calcula la importancia de un archivo dentro de un conjunto de archivos. La idea principal es que los archivos más importantes tendrán más enlaces desde otros archivos. La fórmula para calcular la importancia es:

$$PR(A) = (1 - d) + d \sum_{i=1}^{N} \frac{PR(L_i)}{C(L_i)} \tag{1}$$

Donde:

- A : Archivo para el cual se calcula el PageRank
- d : Factor de amortiguación (0.85)
- L_i : Páginas que enlazan a A
- $C(L_i)$: Número de enlaces en L_i

1.3. Algoritmos

1.4. Otros ejecutables

2. Objetivos

En este proyecto, se tiene el objetivo de crear un programa eficiente y rápido en la búsqueda y recuperación de información, utilizando estructuras de datos como grafos, listas enlazadas y tablas hash, y además aplicar técnicas de investigación para implementar un sistema de búsqueda eficiente.

2.1. Objetivos secundarios

1. **Utilización de estructura de datos:** Crear estructuras de datos eficientes para almacenar y procesar información mediante el uso de listas enlazadas, tablas hash, grafos y manejo de archivos.
2. **Optimización de código:** Optimizar el código para mejorar su eficiencia y rendimiento.
3. **Trabajo en equipo:** Reforzar el trabajo en equipo y la comunicación entre los miembros del equipo para asignar tareas y resolver conflictos.
4. **Eficiencia:** Conseguir tiempos cortos de ejecución mediante algoritmos de búsqueda eficientes.

3. Planteamiento del desarrollo del proyecto

Para iniciar con el desarrollo del proyecto, se establecieron cuatro características principales:

- **Index invertido:** Crear un índice invertido para almacenar los archivos que contienen una palabra específica, en lugar de listar los contenidos de un archivo. Esta característica resulta fundamental para el funcionamiento del programa, ya que es lo que permite identificar los archivos que contienen una palabra específica. Para implementar esto, surgieron dudas en cómo manejar los archivos y palabras, cómo almacenar que un archivo pertenece a una palabra, y cómo realizar la búsqueda de la forma más eficiente posible.
- **PageRank:** Calcular el nivel de importancia de cada archivo (PageRank) utilizando el algoritmo de PageRank.
- **Grafos:** Crear un grafo para representar las relaciones entre los archivos y sus contenidos.
- **Manejo de archivos:** Investigar como crear una función de manejo de archivos para identificar los archivos regulares de texto plano en el directorio de entrada y en aquellos sub-directorios de éste y procesarlos de manera eficiente, evitando la lectura de archivos que no sean de este tipo y continuar con el archivo siguiente.

Con esto en mente, entonces se planteó el desarrollo del proyecto, dividiéndolo en cuatro etapas: Creación de la estructura de datos, creación de las funciones y algoritmos, el ensamblaje del programa y la optimización del código.

4. Implementación

La implementación del programa se realizó mediante el lenguaje de programación C. Las funciones necesarias fueron implementadas en archivos separados según su funcionalidad. El código fue compilado a través de un `Makefile`, el cual se encuentra en el directorio raíz.

4.1. Estructura de directorios

Dentro del directorio `src` se encuentran los siguientes archivos:

- `errors.c`: Contiene las funciones de gestión de errores.
- `files.c`: Contiene las funciones de gestión de archivos.
- `graph.c`: Contiene las funciones de gestión de grafos.
- `hash.c`: Contiene las funciones de gestión de tablas hash.
- `link_list.c`: Contiene las funciones de gestión de listas enlazadas.
- `main.c`: Contiene el código principal del programa.
- `page_rank.c`: Contiene las funciones de calculación de PageRank.
- `reverse_index.c`: Contiene las funciones de gestión del índice invertido.
- `stop_words.c`: Contiene las funciones de gestión de stop words.
- `timer.c`: Contiene las funciones de gestión de tiempo de ejecución.
- `utilities.c`: Contiene las funciones de utilidad del programa.

Por otro lado, en la carpeta `testing` se encuentran archivos para el testeo del programa:

- `spanish.txt`: Contiene una lista de stop words en español.
- `test.txt`: Script de *Python* que recupera de *Wikipedia* artículos aleatorios.

4.2. Estructuras de datos

4.2.1. Índice invertido

Para el índice invertido se optó finalmente por utilizar una **tabla hash** para almacenar las palabras, permitiendo así una búsqueda de orden $O(1)$. Sin embargo, acá surgieron algunos inconvenientes, como la necesidad de manejar posibles colisiones y el límite de memoria que trae consigo los arreglos. Para solucionar esto, se optó que cada celda de esta tabla hash contenga una **lista enlazada simple**, en la cual será en donde realmente se almacenarán las palabras asociadas a un hash key. Esto permite una búsqueda más eficiente y sin un límite de memoria.

Por otro lado, para poder relacionar un archivo con una palabra se hizo que cada palabra además almacene consigo otra **lista enlazada simple**, en la cual se almacenarán los archivos que contienen dicha palabra.

4.2.2. Grafos

4.3. PageRank

4.4. Manejo de archivos

Para el manejo de archivos se optó por utilizar una **lista enlazada simple** para almacenar los archivos, en la cual cada archivo se almacena en una celda de la lista. Esto permite una búsqueda más eficiente y sin un límite de memoria. Además, para poder identificar y procesar los archivos

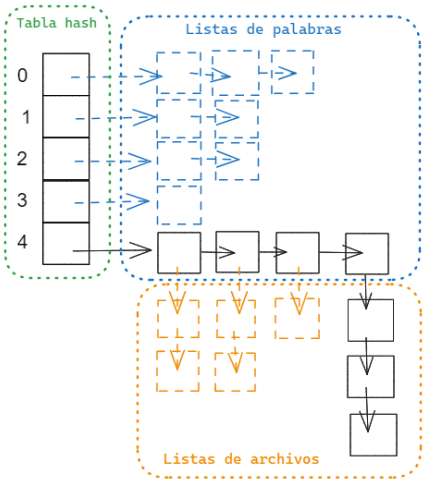

Figura 1. Esquema de la estructura de datos del índice invertido

del directorio y sus sub-directorios se utilizó la librería **dirent.h**.

La librería **dirent.h** es una librería de C que permite leer los archivos de un directorio, y que proporciona una estructura llamada **struct dirent** que contiene información sobre cada archivo, como por ejemplo su nombre (con extensión), su tipo (archivo o directorio) y su identificador (ID).

En el caso de existir un sub-directorio dentro del directorio de entrada, se procesará recursivamente, añadiendo cada archivo que se encuentre en dicho directorio a la lista de archivos.

Por último para poder identificar los archivos que contienen texto plano, se utilizó una función para extraer el nombre sin extensión y a su vez verificar si el archivo es de texto plano a traves de su extensión.

4.5. implementaciones extras

Se implementó un temporizador para medir los tiempos de ejecución de las funciones y así analizar y mejorar el código(solo para testing).

También se utilizó **mergeSort** para ordenar (de mayor a menor) el PageRank de los archivos, para tener una mejor visualización de los resultados.

5. Gestión del equipo de trabajo

El equipo consto de 4 personas, donde entre éstas se tomaron decisiones y se distribuyeron los puntos importante del motor de busqueda para realizar esta tarea. Para mejorar el orden y la eficiencia implementamos **normas de codificación**, establecer fechas para tareas pendientes, y por consecuencia reuniones para acordar las siguientes tareas.

5.1. Normas de codificación

Con respecto al nombramiento de las constantes, estas fueron llamadas con el formato **SCREAMING_SNAKE_CASE**, y las variables se llamaron respectivamente con el formato **camelCase**. También a las funciones se acordó utilizar el formato **snake_case**, y colocar las llaves de apertura en la siguiente linea, pero por otro lado aquellas llaves de apertura de otros bloques de codigo se colocarán justo al lado de su linea final, evitando omitir llaves en caso de existir únicamente una sentencia.

ejemplo de referencia: **https://github.com/ayrvanmo/ForKing/blob/master/docs/forKingDoc.pdf**

5.2. Lista de tareas y organización

Teniendo ya realizada las normas de codificación, se establecieron objetivos principales (Puntos importantes del programa), y objetivos secundarios o a corto plazo, los cuales fueron señalados para mantener un control sosbre el flujo de trabajo y repartirlos equitativamente pensando en los puntos fuertes de cada integrante del equipo; Por ejemplo los objetivos principales que designamos fueron asignados a cada integrante la codificación de cada una de ellas.

5.3. Reuniones semanales

Durante cada semana, se hacia por lo menos un día de reunión para conversar y gestionar los trabajos listos, pendientes y mejoras sobre estos mismos, y también para trabajar en aquellos puntos más complicados del programa. También se acordo que durante cada semana puedan realizarse reuniones casuales para abordar problemas sobre codificación simples y solucionarlos en conjunto. Las reuniones ya designadas previamente serian de manera presencial y en caso de que un integrante no pueda asistir éste debe ver la posibilidad de conectarse mediante llamada o videollamada para mantenerse actualizado sobre los objetivos y lo conversado en aquella reunión. En el caso de las reuniones casuales, estás pueden ser realizadas entre solo dos integrantes con la previa disponibilidad de ambos.

6. Posibles Mejoras a futuro

7. Conclusiones

Referencias

[1] A. S. Tanenbaum y A. S. Woodhull, *Sistemas Operativos: Diseño e implementación*, Segunda edición, trad. por R. Escalona. México: Prentice Hall, 1997.

[2] T. O. Group. «fork() - create a new process». (2004), dirección: https://pubs.opengroup.org/onlinepubs/009696799/functions/fork.html (visitado 12-10-2024).

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, *Introduction to Algorithms*, 3rd. Cambridge, MA: MIT Press, 2009, ISBN: 978-0-262-03384-8.

[4] geeksforgeeks. «Buddy System – Memory Allocation Technique». (sep. de 2024), dirección: https://www.geeksforgeeks.org/buddy-system-memory-allocation-technique/.