UA

Indexador

DanielAsensiRochDNI : **48776120C**

April 30, 2022

Contents

1 Introducción:

En la asignatura de Explotación de la información estudiamos diversos problemas relacionados a como se organiza la información que llega de manera masiva a los computadores y los diferentes algoritmos para procesarla, organizarla y presentarla, de manera que esta sea lo más accesible posible para los algoritmos de búsqueda y por consecuencia para el usuario final.

1.1 Presentación del problema

En esta práctica se nos presenta la problemática de realizar un Indexador, el cual será capaz de pasado un fichero o cadena de caracteres que represente una pregunta, guardar toda su información relevante, para más adelante en futuras prácticas utilizarla para obtener los ficheros relevantes de nuestro buscador, para ello nuestro indexador debe ser capaz de:

1. Obtener los términos de un documento
2. Filtrar los términos de un documento según las stopWords (palabras de parada)
3. Almacenar las posiciones en las que aparece dicha palabra en el documento
4. Obtener y incrementar la frecuencia en la que aparece la palabra en el documento o en general en la colección
5. Guardar la indexación realizada en un fichero de texto para su posterior recuperación
6. Realizar el stemming de cada una de las palabras indexadas

2 Análisis de la solución implementada:

Para la realización de todas estas acciones mencionadas anteriormente contaremos con las siguientes estructuras de datos, las cuales se encargarán de almacenar los datos en memoria:

2.1 InfDoc

Esta estructura de información se encargará de almacenar toda la información relevante de **un solo** documento, en concreto la información que se almacenará será la siguiente:

- Id del documento
- Número de palabras del documento
- Número de palabras sin parada
- Número de palabras diferentes
- Número de tamaño en bytes
- Fecha de modificación

Para el manejo de la información anteriormente nombrada al ser índole privada se propuso la idea de pasarla a pública para ahorrar en tiempo de llamadas a función, **esto fue descartado** aunque incrementaba la velocidad en aproximadamente un 30%, pasando directamente al manejo de getters y setters por referencias para un menor gasto de memoria. Esta información se obtendrá utilizando : **stat(nombreDocu.c_str(),infoDocumento);**

2.2 Fecha

Esta clase se encargará de proporcionar la fecha de modificación a los documentos, para la implementación de esta clase se ha optado por el uso de **struct tm *clock** una estructura proporcionada por c++ la cual nos proporcionará mediante llamadas al sistema el tiempo de reloj, esto se realizará para cada uno de los archivos.

2.3 InfColeccionDocs

Esta estructura de información se encargará de almacenar toda la información relevante de **todos** los documentos, en concreto la información que se almacenará será la siguiente:

- Número total de documentos
- Número total de palabras
- Número total de palabras sin parada
- Número total de palabras diferentes
- Número total de tamaño en bytes

Para el manejo de la información privada se han implementado getters y setters por referencia para un menor gasto de memoria.

2.4 InformacionTermino

Esta estructura de información almacenará la información relevante a cada uno de los términos indexados, la información almacenada será la siguiente:

- Frecuencia total del termino en todos los documentos
- Posición del termino en cada uno de los documentos

Para guardar la posición del termino en cada uno de los documentos utilizaremos un HashMap mediante clave **Id documento** y valor la estructura de información **InfTermDoc** que explicaré a continuación, para el manejo de la información se han utilizado getters y setters por referencia, además para buscar la información dentro del hashMap se han probado diferentes métodos pero se ha optado por la utilización de **.find** o el acceso mediante clave ya que su complejidad de acceso es **O(1)**

2.5 InfTermDoc

Esta estructura de información se encargará de manejar la información con respecto a un termino en específico dentro de un documento en específico, la información que se guardará será la siguiente:

- Frecuencia total del termino en el documento
- Lista de posiciones de un termino en un documento

Se ha mantenido la implementación original de la lista propuesta por el profesor, cabe destacar que se testeo el uso de un hashmap con el id del documento y una lista de posiciones asociadas, además se probo también la reserva de memoria fija para la lista, pero esto fue descartado ya que el tiempo empleado para esto era demasiado alto.

2.6 InformacionTerminoPregunta

Esta estructura de información se encargará de manejar la información con respecto a un termino en específico dentro de una pregunta en específico, la información que se guardará será la siguiente:

- Frecuencia total del termino en la pregunta
- Lista de posiciones de un termino en la pregunta

Se ha mantenido la implementación original de la lista propuesta por el profesor.

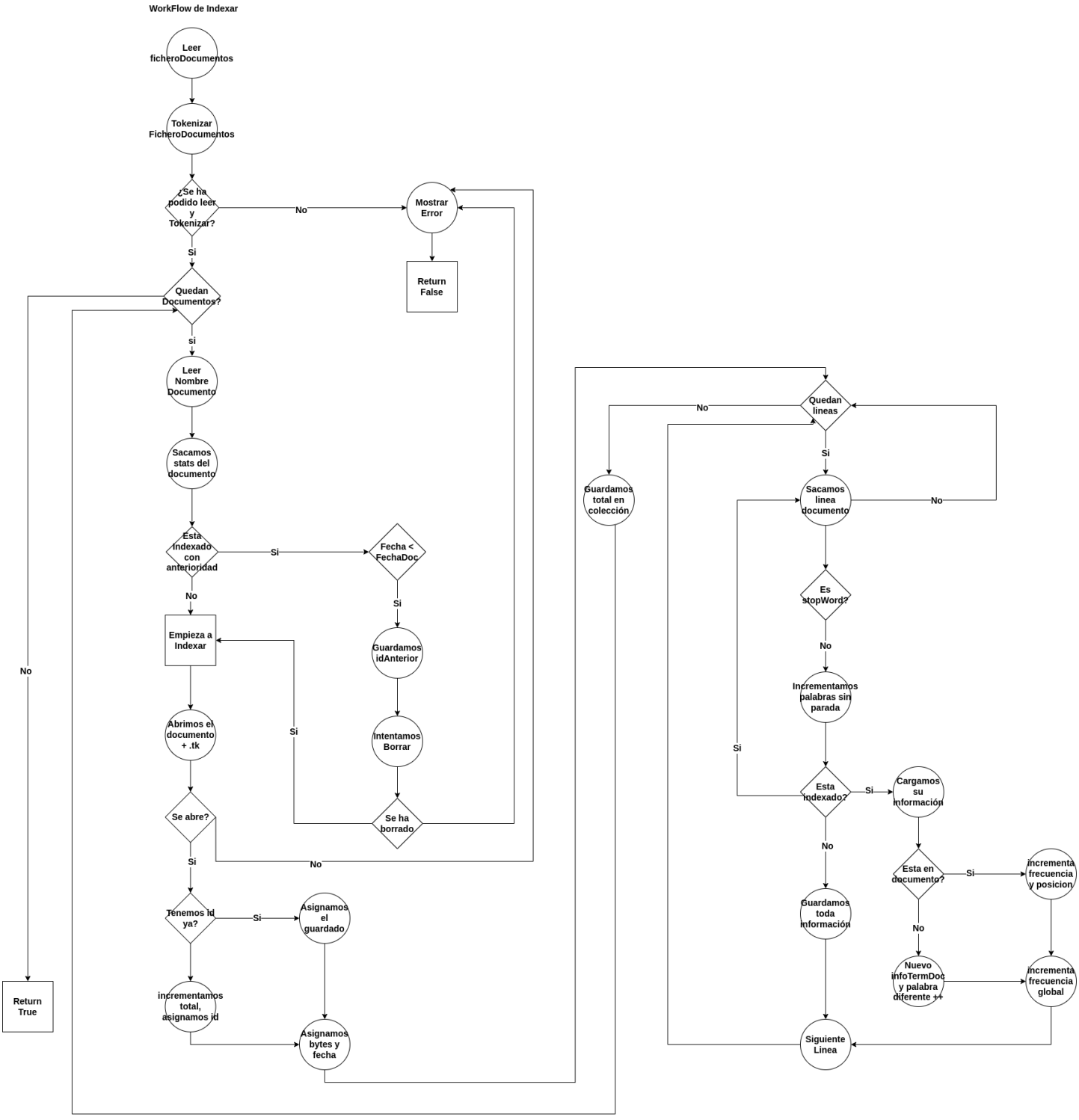
2.7 InformacionPregunta

Esta estructura de información se encargará de almacenar toda la información relevante de **una sola** pregunta, en concreto la información que se almacenará será la siguiente:

- Número de palabras de la pregunta
- Número de palabras sin parada en la pregunta
- Número de palabras diferentes en la pregunta

3 Algoritmo implementado para guardado en memoria

Para mayor entendimiento del funcionamiento de mi algoritmo para el guardado en memoria he realizado un pequeño grafo para mostrar el flujo de trabajo del mismo:


WorkFlow de Indexar
Leer ficheroDocumentos
Tokenizar FicheroDocumentos
¿Se ha podido leer y Tokenizar?
No
Mostrar Error
Return False
Si
Quedan Documentos?
No
Return True
si
Leer Nombre Documento
Sacamos stats del documento
Esta indexado con anterioridad
Si
Fecha < FechaDoc
Si
Guardamos idAnterior
Intentamos Borrar
Se ha borrado
Si
No
Empieza a Indexar
Abrimos el documento + .tk
Se abre?
No
Si
Tenemos id ya?
Si
Asignamos el guardado
incrementamos total, asignamos id
Asignamos bytes y fecha
Quedan lineas
No
Guardamos total en colección
Si
Sacamos linea documento
No
Es stopWord?
No
Incrementamos palabras sin parada
Si
Esta indexado?
Si
Cargamos su información
No
Guardamos toda información
Esta en documento?
Si
incrementa frecuencia y posicion
No
Nuevo infoTermDoc y palabra diferente ++
incrementa frecuencia global
Siguiente Linea

4 Justificación de la solución elegida

Antes de llegar al algoritmo de indexación presentado en la imagen anterior se probaron las siguientes soluciones:

La primera solución que se probo en el algoritmo fue tras abrir el fichero que contenía los nombres de los documentos, abrir el fichero a indexar y leer las lineas una a una, para cada una de las lineas se creaba una lista con los tokens generados que se indexarían con posterioridad, esta solución fue **descartada** ya que la generación y formación de las listas con los tokens ocupaba un gran espacio en la memoria y una gran cantidad de tiempo.

La siguiente solución probada fue la siguiente, tras abrir el fichero de documentos y abrir el fichero este se tokenizaría en su totalidad generando una lista con todos los términos a indexar, esta solución también fue **descartada** ya que el tamaño de la lista generado era demasiado y por lo tanto su tiempo de generación y lectura era masivo.

Una de las mejoras implementadas ha sido la creación de setter y getter por referencia, estos nos permitirán ahorrar tiempo y espacio durante la ejecución de nuestro programa, esto nos servirá sobretodo para buscar en las listas privadas de posiciones y en la resignación de hashmaps.

Se testeo el uso de un hashmap con el id del documento y una lista de posiciones asociadas, además se probo también la reserva de memoria fija para la lista, pero esto fue **descartado** ya que el tiempo empleado para esto era demasiado alto

Otra de las soluciones implementadas para mejorar la eficiencia fue en el momento de insertar las posiciones de una palabra dentro de los documento, ya que para comprobar si una palabra estaba en L_docs se ha usado una función auxiliar para insertarla y no la copia de todo el hashmap.

5 Solución de indexación en disco:

En cuanto a la indexación en disco para su posterior recuperación se ha optado por hacerlo todo en un mismo documento, para ello se expondrá un ejemplo con la estructura implementada:

```
1
1
./StopWordsEspanyol.txt
otras van consigo haces nuestras incluso mismo
```

La estructura de este primer parrafo es la siguiente:

1. Variable de almacenar la posción del termino
2. Variable de almacenar en disco
3. Fichero de almacenamiento de stopWords
4. Todas las stopWors en una mima linea

```
7
3
3
¿Quién es el presidente de la UE?
3
```

En el siguiente párrafo se puede apreciar como se guarda la información con respecto a la pregunta, la información se guarda en el siguiente orden:

1. Número total de palabras de la pregunta
2. Número total de palabras diferentes de la pregunta
3. Número total de palabras sin parada
4. La pregunta
5. Cantidad de palabras indexadas de la pregunta

Después se realiza un bucle que escribirá las palabras indexadas con sus posiciones y frencuencia en la pregunta, quedando de la siguiente manera:

```
presidente
1
3
UE?
1
6
¿Quién
1
0
```

La siguiente información que guardaremos será la relevante a la colección de documentos

```
2
11
4
7
53
```

Orden de información almacenada:

1. Cantidad de documento indexados
2. Número total de palabras de la colección
3. Número total de palabras diferentes de la colección
4. Número total de palabras sin parada
5. Número total de bytes de la colección

A continuación se mostrará la información relevante a los términos indexados tal como frecuencia de los mismos y sus posiciones:

```
4
pal1
2
1
1
2
0 3
```

Esta información se escribirá y leerá mediante dos bucles por lo tanto toda tendrá el mismo formato:

1. Cantidad de palabras en el índice
2. Termino
3. Frecuencia total del termino

4. Cantidad de documentos en los que aparece
5. Id del documento
6. Frecuencia del termino en el documento
7. Posiciones del termino en el documentos

A continuación se escribirá la información relevante a los documentos con el siguiente formato:

```
2
corpus_corto/fichero2.txt
2
5
2
3
23
122
3
23
12
39
14
```

Estos siguen la siguientes estructura:

1. La cantidad de documentos que tenemos en la colección
2. El nombre del documento
3. Número de palabras indexadas del documento
4. Cantidad de palabras diferentes del documento
5. Cantidad de palabras sin parada del documento
6. El tamaño en bytes del documento
7. Los siguientes 6 últimos campos son la fecha de modificación del mismo.

Por último se guarda la información relevante al tokenizado de los archivos de la siguiente manera:

```
0
0
. ,:
```

Estos valores representa:

1. El valor de casos especiales que parseará a true o false
2. El valor de pasar a minusculas y sin acentos que parseará a true o false
3. Los delimitadores que se emplearán

6 Soluciones descartadas de indexación en disco

En cuanto a la indexación en disco solo se probaron dos alternativas la que se encuentra implementada actualmente la siguiente. La solución **descartada** fue la de crear archivos para cada uno de los ficheros indexados, en el cual se guardaba la información relevante además de sus términos con frecuencias y posiciones pertinentes, además se creo un archivo con la información relevante de todos los términos y de que archivo extraer sus posiciones. El motivo de descarte de esta solución fue la demora en la creación de los archivos y la lectura de los mismos ya que se debían abrir los archivos a la vez para la reindexación de las posiciones. Otro de los motivos del descarte fue que tras la implementación y prueba de esta solución mi computador sufrió un cuelgue durante la lectura de los archivos ya que no podía manejar tantos a la vez.

7 Análisis de eficiencia computacional

7.1 Indexación sin guardado en disco

La ejecución con los siguientes parámetros:

```
IndexadorHash b("./StopWordsEspanyol.txt", ". ,:", false, false,
"./indicePruebaEspanyol", 0, false, true);

b.Indexar("listaFicheros.txt");
```

El tiempo medio tras la repetición de ejecuciones ha sido de **5.03s**

7.2 Indexación sin guardado en disco más guardado de pregunta

Ahora se probará guardando la indexación en el disco:

```
IndexadorHash b("./StopWordsEspanyol.txt", ". ,:", false, false,
"./indicePruebaEspanyol", 0, true, true);

b.Indexar("listaFicheros.txt");
```

El tiempo medio tras la repetición de ejecuciones ha sido de **7,22s**

7.3 Indexación con guardado en disco y guardado de pregunta

Ahora se probará sin guardar la indexación en disco pero indexando una pregunta

```
    long double aa = getcputime();
IndexadorHash b("./StopWordsEspanyol.txt", ". ,:", false, false,
"./indicePruebaEspanyol", 0, false, true);
b.Indexar("listaFicheros.txt");

b.IndexarPregunta("¿Quién es el presidente de la UE?");
```

El tiempo medio tras la repetición de ejecuciones ha sido de **5.03s**

7.4 Indexación con guardado en disco y recuperación de datos en otro indexador

La siguiente prueba de tiempo se realizará con dos indexadores el primero guardará los datos y el segundo los recuperará además se indexará una pregunta:

```
    IndexadorHash b("./StopWordsEspanyol.txt", ". ,:", false, false,
"./indicePruebaEspanyol", 0, true, true);
b.Indexar("listaFicheros.txt");

b.IndexarPregunta("¿Quién es el presidente de la UE?");

IndexadorHash a("./indicePruebaEspanyol");
```

El tiempo medio tras la repetición de ejecuciones ha sido de **12.8s**