19/20

Indexador

EXPLOTACIÓN DE LA INFORMACIÓN

Esteban Antón Sellés
48788593G
Ingeniería Informática

Índice

40% por la corrección automática con los ficheros de prueba.

- Control de tiempos de ejecución

15% por el cálculo de la eficiencia temporal “en disco” (almacenarEnDisco == true).
15% por el cálculo de la eficiencia temporal “en memoria” (almacenarEnDisco == false).

- Cálculo de eficiencia espacial

15% por el cálculo de la eficiencia espacial “en memoria” (almacenarEnDisco == false).

- Memoria

15% por la revisión presencial de la práctica durante las prácticas de laboratorio.

Introducción

En la siguiente práctica se ha llevado a cabo un indexador de ficheros de texto. El código para conseguirlo se divide en cuatro ficheros, *indexadorInformacion.h/.cpp* y *indexadorHash.h/.cpp*.

Estructura

indexadorInformacion

En los archivos con este nombre, se encuentran las clases con las que tener estructurada la información de la indexación de directorios, documentos y palabras, ya sean de un documento o de una pregunta. Se componen de siete clases:

- Fecha: Se ha usado el tipo *time_t*.
- InfTermDoc: En esta clase se encuentra la información sobre la **frecuencia de un término en un documento** con la variable *ft* y el **número de palabra en el que aparece el término**, es decir, la posición del término en el texto, en la lista *posTerm*.
- InformacionTermino: En esta clase se contiene la información sobre la **frecuencia del término en la colección** con la variable *ftc* y la **información del término en cada documento** en la tabla hash *l_docs*.
- InfDoc: Contiene la información sobre un documento, es decir, la **id** (*idDoc*), el **número de palabras** del documento (*numPal*), el **número de palabras sin las stop-words** del documento (*numPalSinParada*), el **número de palabras diferentes** (*numPalDiferentes*), el **tamaño del documento en bytes** (*tamBytes*) y la **fecha de modificación** (*fechaModificacion*).
- InfColeccionDocs: contiene la información total de la colección de documentos que ha sido indexada.
- InformacionTerminoPregunta: Contiene información sobre la **frecuencia de un término en una pregunta** con la variable *ft* y la **posición en la que se encuentra en la pregunta** con la variable *posTerm*.
- InformacionPregunta: Contiene información sobre el **número total de palabras de la pregunta** en la variable *numTotalPal*, el **número total de palabras que no son stop-words** con la variable *numTotalPalSinParada* y el **número total de palabras diferentes de la pregunta** con la variable *numTotalPalDiferentes*.

Cada clase cuenta con la implementación de su forma canónica y métodos get y set, según han ido siendo necesarios para realizar la práctica. En busca de eficiencia, esta vez se han implementado métodos ToString con el objetivo de evitar el sobreuso del operador *operator*<<.

indexadorHash

En esta clase se encuentra la mayor parte de carga de la práctica. En ella se encuentran los algoritmos de indexación y manejo de las variables de los atributos de la clase.
En cuanto a la estructura de esta clase, sus atributos son los siguiente:

- indice: Siendo una tabla hash, en el que la clave es un string, es decir, la **palabra indexada**, y el valor es la **información del término indexado**.
- indiceDocs: Siendo también una tabla hash, en el que la clave es el **nombre del documento** y el valor es la **información del documento**.
- informacionColeccionDocs: De tipo InfColeccionDocs, como se ha comentado anteriormente, almacenando **información general sobre la colección de documentos** indexados.
- pregunta: Contiene la pregunta tal cual se recibe, sin ningún tipo de tratamiento previo.
- indicePregunta: Siendo una tabla hash compuesta por una clave de tipo string, que es las **palabras que no son stop-words** de la pregunta, y como valor la **información de cada término** de la pregunta.
- infPregunta: Contiene información general sobre la pregunta.
- stopWords: Lista de palabras que son stop-words.
- ficheroStopWords: **Ruta** donde se encuentra el fichero con las stop-words que se van a usar para la indexación.
- tok: Tokenizador usado en la indexación.
- directorioIndice: Directorio donde se guardaría la indexación en caso de que así fuese indicado.
- tipoStemmer: Tipo de stemming que se aplica a cada palabra que no es stop-word en la indexación.
- almacenarEnDisco: Flag que indica si la indexación se almacena en disco.
- almacenarPosTerm: Flag que indica si la posición de los términos se debe almacenar en las clases InfTermDoc o InformacionTerminoPregunta.

Algoritmos

Como se ha comentado antes, los algoritmos realmente importantes, o que tienen mayor carga, están en la clase IndexadorHash.
Siempre que se ha podido, se ha hecho uso de iteradores para recorrer tanto listas como tablas hash.

Indexar

Este método se puede considerar el centro de toda la práctica, al fin y al cabo es desde donde se organiza el código para obtener los resultados esperados.
Pero dentro de esta función, que se dirige a indexar documentos, se ha decidido separar la indexación de un nuevo documento con la reindexación, aunque al final, ambos caminos se junten.
Antes de nada, se cargan los documentos mencionados en el fichero de documentos, cuya ruta se pasa por parámetro a la función *Indexar*. Seguidamente, por cada documento que ha sido cargado en la lista, se comprueba si el documento ya existe en la indexación, además de otros cálculos, y es aquí donde se bifurca el algoritmo.

indexarDocumento

A este método se accede cuando el documento es nuevo en el índice de documentos o para finalizar la reindexación.
Lo primero que se hace es tokenizar el documento y cargarlo en una lista. Siguiendo con el algoritmo de indexación de un documento, se establece el tamaño del documento y el número de palabras del documento. Después, palabra por palabra de la lista, se comprueba que no pertenezca al grupo de stop-words y se inserta en el índice del indexador. Y finalmente, se actualizan campos tanto de la información del documento, como de la colección de documentos del indexador.

reindexarDocumento

Se ha querido destacar ambos métodos, porque en este caso se debe eliminar la información referente al documento en cuestión, provocando una mayor carga de trabajo que la indexación de un nuevo documento, aunque en la reindexación se haga uso del método *indexarDocumento*.
A diferencia de lo comentado en el apartado anterior, esta vez se busca toda la información sobre el documento y las palabras que se encuentran en él y se elimina de forma que se mantienen actualizados los datos del índice y la información de la colección de los documentos para que mantengan la coherencia. A la hora de borrar el documento y toda la información relacionada con él, se busca en el índice de documentos y se actualizan los campos de la información de la colección de documentos. Posteriormente, se recorre cada término del índice y se comprueba si el término aparece en el documento que se quiere eliminar, en caso afirmativo, después de la eliminación de este de la información del término, se comprueba si el término ya no aparece en ningún otro documento más, por lo que si es así, debe ser eliminado del índice.
Después de esto, ocurre lo mismo que cuando se indexa un nuevo documento, exceptuando que para este, al ser indexado de nuevo, se le ha guardado el id que poseía anteriormente.

IndexarDirectorio

Para la indexación de un directorio, sabiendo que la ruta existe, se recupera la lista de todos los ficheros del directorio a través del comando “find [nombre_directorio] -type f | sort”.

IndexarPregunta

En este caso, el algoritmo es parecido al de indexar un documento, pero se puede considerar incluso más sencillo, ya que no hay que almacenar información más allá de las palabras que componen la pregunta y no pertenecen a las stop-words. Por lo que en el algoritmo aparece el borrado de la información de cualquier indexación de una pregunta anterior, la tokenización de la pregunta, la indexación de las palabras que la componen y la actualización de la información de la pregunta.

Eficiencia

En cuanto a la eficiencia del programa, se ha calculado de forma teórica, contando los bucles dependientes del tamaño del problema, y práctica, con la ejecución de un ejemplo de tamaño considerable.

En primer lugar, por su importancia en esta práctica, se ha calculado la eficiencia y cota del método indexar. Tanto a la hora de indexar un nuevo documento como volver a indexar uno de nuevo, se lleva a cabo la carga de los documentos, lo cual depende de la **cantidad de documentos** que haya (*d*). Con esto como punto de partida, cuando se indexa un nuevo documento, se extrae la tokenización de cada documento, devolviendo una lista de *p* palabras. Por lo que en el peor de los casos, donde todos los documentos existen y ninguna de las palabras es stopword (*s*), la cota superior sería $d \cdot p \cdot s$. Y en cuanto al mejor caso, sería que ninguno de los documentos existiese, por lo que solamente se recorrería un bucle d veces.

Por otro lado, el indexar de nuevo un documento, sumaría el proceso de borrado de la información del documento, en el que se recorre todo el índice de términos del indexador de tamaño *i*. Así pues, el peor caso sería $i+d \cdot p \cdot s$. Y como anteriormente, el mejor caso sería que solo se cargasen los documentos, y que ninguno de ellos existiese, por lo que se haría un bucle leyendo los d documentos.

En cuanto a las pruebas empíricas, estos son los resultados tanto en eficiencia temporal como en eficiencia espacial.