Proyecto de Recuperación de la Información

Alejandro Díaz Gómez y Juan Luis Sena Cárdenas

Decisiones Generales

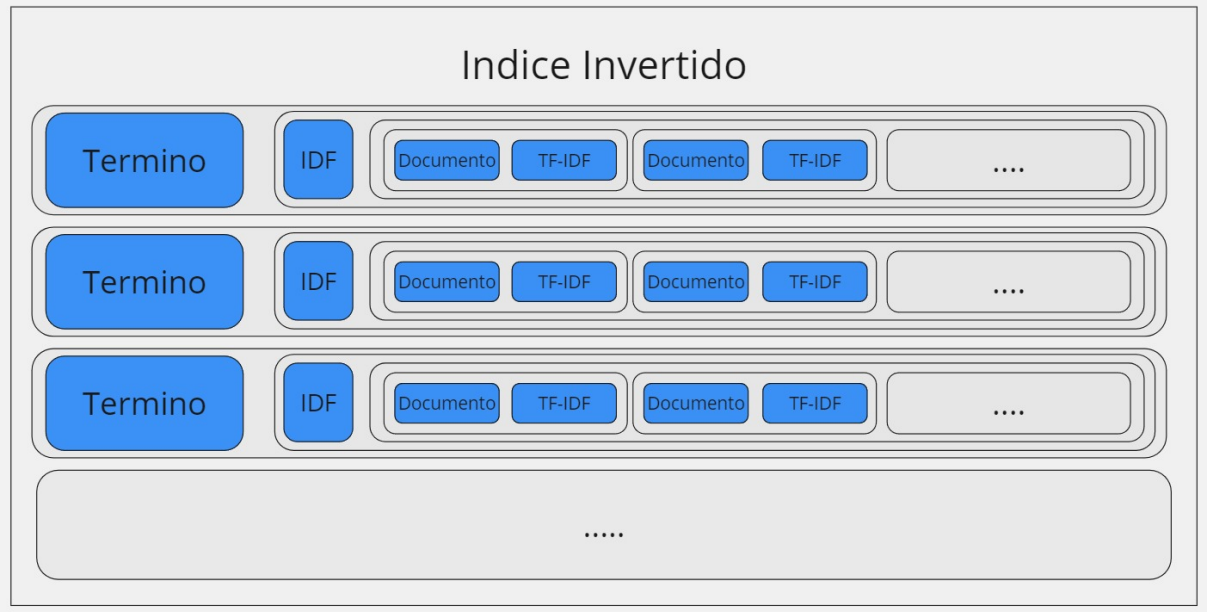
El sistema ha sido implementado totalmente en java debido principalmente a su velocidad de ejecución y a que es un lenguaje con el que ambos teníamos experiencia previa, por lo que el desarrollo no tuvo demasiadas complicaciones en ese sentido. Inicialmente se probó a utilizar Python debido a su mayor sencillez y las facilidades que aporta a la hora del desarrollo, sin embargo, tuvo que ser rápidamente descartado al ver su bajo rendimiento cuando se enfrentaba a la gran cantidad de documentos de los que dispone el corpus a utilizar (unos casi 30.000 documentos).

A la hora de implementar el sistema, dividimos el trabajo de forma que una persona realizaría el módulo de indexación, mientras que la otra implementaría el de búsqueda, para más tarde poder enlazar y comunicar ambos módulos correctamente, formando el sistema completo y realizando múltiples pruebas finales sobre el mismo. Aunque a pesar de esta división del trabajo, fue necesaria una comunicación constante durante todo el desarrollo de los módulos, en especial para la toma de decisiones comunes, como la estructura de datos a utilizar, el preprocesado específico a realizar, o el tipo de fichero a utilizar para exportar las distintas estructuras necesarias.

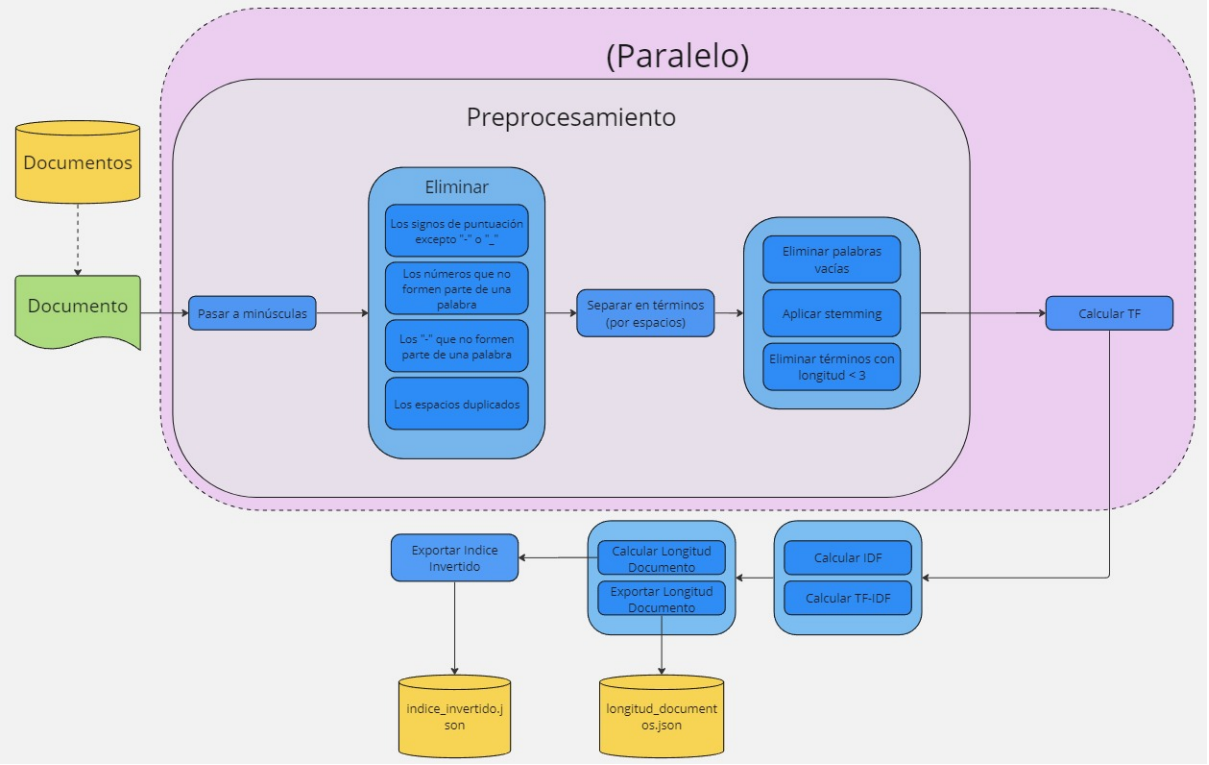
Indexación

Antes de explicar el proceso de indexación, es necesario explicar la **estructura principal** a utilizar para el **índice invertido**.

Esta estructura se trata de un HashMap<String, Object[]>, donde cada String representa un término, mientras que el vector de Object, el cual siempre tiene dos elementos, contiene como primer elemento el IDF del término correspondiente, y como segundo término otro HashMap<String, Float>, donde el String representa el nombre de cada documento y el Float su TF-IDF con el término en cuestión. La estructura del índice se aprecia mejor con el siguiente esquema:



Pasando al proceso de indexación como tal de nuestro sistema de recuperación de la información, este sigue el siguiente esquema:



Por cada documento de nuestra colección realiza el siguiente **preprocesamiento**:

- Se pasa todo el texto del documento a **minúsculas**

- Se eliminan los siguientes elementos
    - Todos los signos de puntuación, ya que se consideran irrelevantes para este sistema en concreto. Los únicos a mantener son el “-” y “_”.
    - Los números que no forman parte de ninguna palabra
    - Los “-” y “_” que no forman parte de ninguna palabra
    - Los posibles espacios duplicados
- Se separan los términos del texto. El criterio de separación es el espacio.
- De los términos obtenidos:
    - Se eliminan aquellos que se aparezcan en nuestra colección de **términos vacíos**, la cual hemos obtenido de: https://www.ranks.nl/stopwords
    - Se les aplica el algoritmo de Porter para **Stemming**. El sistema para la aplicación de este algoritmo es el siguiente: http://snowball.tartarus.org/, y la implementación concreta para java se puede encontrar en el siguiente enlace: https://github.com/snowballstem/snowball/tree/master/java
    - Se eliminan términos cuya longitud sea menor a 3 caracteres. Es crucial que este paso se realice después del Stemming, ya que este proceso puede reducir términos a otros de longitud muy reducida, los cuales no nos interesan.

Tras todo el preprocesamiento disponemos de una lista de términos con la cual podemos comenzar a calcular el TF. Para ello simplemente se itera por cada término y se añade su valor correspondiente con el documento actual. Al mismo tiempo, se guardan los términos en un set de todos los términos del conjunto de documentos, además de llevar la cuenta de en cuantos documentos aparece cada término en un HashMap. Estas dos estructuras serán esenciales a la hora de calcular el IDF.

Es de vital importancia mencionar que tanto el preprocesamiento como el cálculo del TF están paralelizados para reducir drásticamente el tiempo que tardan en realizarse. Para ello se utiliza un pool de threads de tamaño fijo y configurable por el usuario como parámetro, y en el código concurrente únicamente se protege el acceso a las estructuras de datos mediante un cerrojo y bloques synchronized.

Esto nos permite pasar de 5.5 segundos a 1.5 segundos para el cálculo del TF de la colección completa de documentos. Nota: estos tiempos de ejecución pueden variar drásticamente entre la primera ejecución del indexador (~15 seg) y las siguientes (~1.5 seg), probablemente debido a temas de acceso a caché.

Lo siguiente es calcular el IDF esto se hace iterando por cada uno de los términos del set global que hemos mencionado anteriormente. Nada más calcular el IDF de cada término, se

multiplica por el TF de cada documento con ese término, para así obtener el TF-IDF y completar el índice invertido.

Por último, queda extraer los datos. Lo primero a extraer será la longitud de cada documento, la cual calculamos con la siguiente fórmula:

$$|\vec{d}_j| = \sqrt{\sum_{i}^{t} w_{i,j}^2}$$

Donde t es el número de términos del documento, y $w_{i,j}$ es el TF-IDF del término i con el documento j

Tras el sencillo cálculo de la longitud de cada documento en un HashMap, se extrae en formato json al archivo “longitud_documentos.json”, para ello se utiliza un BufferedWriter y se escribe secuencialmente en el archivo a partir del contenido del HashMap.

El proceso de exportación es el mismo para el índice invertido, lo único que cambia es la iteración sobre la estructura a la hora de escribir en el json, ya que esta estructura es bastante más compleja.

Se utiliza json como formato de archivos debido a que se adapta perfectamente al tipo de estructuras que queremos exportar sobre estos archivos, sin embargo, java no tiene el mejor soporte para trabajar con este tipo de archivos, por lo que para la lectura de estos archivos es necesario el apoyo en una librería externa, en nuestro caso usamos Gson: https://mvnrepository.com/artifact/com.google.code.gson/gson

Búsqueda

El módulo de búsqueda parte de los archivos devueltos por el módulo de indexación con el índice invertido y las longitudes de los documentos, los carga al inicio en los mismos tipos de estructuras que los utilizados en el módulo de indexación. A partir de ellos devuelve los documentos más relevantes para la consulta introducida por el usuario por línea de comandos. Esta consulta puede ser de distintos tipos:

- **Búsqueda de términos:** El usuario podrá introducir uno o varios términos, en el segundo caso también se dará la opción de escoger entre la aplicación de conjunción (AND) o disyunción (OR).

  En este tipo de consultas, se aplica un preprocesado idéntico al realizado por el módulo de indexación para asegurar compatibilidad entre los términos de las consultas y los términos en el índice invertido.

Tras realizar el preprocesado y obtener la lista de términos que forman la consulta, se buscan todos los documentos en los que aparezca al menos un término de la consulta. En el caso de que la consulta sea de tipo AND, se da el paso extra de filtrar estos documentos para quedarnos únicamente con aquellos que contengan todos los términos de la consulta.

Lo último es ordenar los documentos basados en un ranking. Este estará basado en la similitud entre el cada documento y la consulta, la cual se calcula de la siguiente manera:

$$cos\theta = sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Donde j es el índice del documento, de modo que $|\vec{d_j}|$ y $|\vec{q}|$ es el módulo de los vectores de la consulta y documento, mientras que $\vec{d_j} \cdot \vec{q}$ es el producto de ambos vectores.

- **Búsqueda de frases:** Cuando el usuario introduce una frase entre comillas, el sistema devolverá únicamente los documentos donde aparecen esas frases.

  Para este tipo de consultas no se realiza ningún tipo de preprocesado, ya que se supone que al introducir una frase entre comillas, el usuario tiene la intención de realizar una búsqueda exacta carácter a carácter, en caso contrario se usaría una búsqueda de términos disyuntiva.

  A pesar de no aplicarse preprocesado directamente en este tipo de búsqueda, este se utiliza para incrementar la eficiencia del proceso. El procedimiento consiste en preprocesar los términos de la consulta para buscar únicamente en los documentos que contengan estos términos, y de este modo reducir considerablemente el rango de la búsqueda, y a su vez el tiempo de ejecución. Una vez se han obtenido los documentos con los términos procesados con ayuda del índice invertido, se pasa a buscar dentro del propio contenido de cada documento con el objetivo de encontrar ocurrencias exactas con respecto a la consulta original. Aquellos documentos que contengan una o más ocurrencias serán mostrados como resultado de la consulta junto con los índices del primer y el último término de la consulta para cada ocurrencia.

  Un ejemplo más claro de lo que se devolvería para este caso es el siguiente: Consideremos el documento D1 cuyo contenido se muestra a continuación:

  ```
  D1: The formation and early evolution of the Milky Way Galaxy
  Búsqueda: "Milky Way Galaxy"
  Resultado: D1: [ [7, 9] ]
  ```