UNIVERSIDAD DE GRANADA

Práctica 3. Desarrollo de un Sistema de Recuperación de Información con Apache Lucene.

Gestión de Información en la Web

Máster Profesional en Ingeniería Informática

Curso académico 2022/2023

Autor

José Alberto Gómez García

modej@correo.ugr.es

Índice

1. Introducción.

En esta primera práctica de la asignatura "Gestión de Información en la Web" se va a realizar el desarrollo de un Sistema de Recuperación de Información (SRI). Para ello, nos apoyaremos en la API de recuperación de información Apache Lucene.

Esta biblioteca cuenta con soporte para multitud de lenguajes de programación, como Python (PyLucene), Ruby y Perl (Plucene) entre otros, aunque está escrita originalmente en Java. Es por este motivo, y por la mayor cantidad de documentación disponible, que usaré Java para desarrollar el proyecto. En concreto, haré uso de la versión 17 del JDK de Java, y la versión 9.5 de Apache Lucene (al ser la versión más reciente disponible, aunque la versión más reciente del paquete "lucene-analyzers-common" es la 8.11.2). Como IDE usaré IntelliJ IDEA en su versión 2022.2.3.

2. Desarrollo del software.

El software desarrollado se divide en 3 paquetes, correspondientes al indexador de la colección documental, al motor de búsqueda y a las utilidades necesarias por los dos paquetes anteriormente mencionados. Además, se adjunta un pequeño script en Python para obtener la colección documental y descomprimir los ficheros de texto en una única carpeta.

2.1. Indexador

Para representar la funcionalidad del indexador de nuestra colección documental se ha creado una clase con tal fin, cuyo nombre, para sorpresa de nadie, es "Indexador". Internamente, esta hará uso de un objeto de la clase *IndexWriter* de Lucene para crear el sistema de ficheros propio del índice y añadir el resultado de indexar documentos a dicha estructura de datos.

El constructor de la clase *Indexador* se encarga de inicializar el objeto de la clase *IndexWriter* mencionado anteriormente. En particular:

- Lee el fichero de palabras vacías y crea la estructura de datos propia de Lucene para almacenar esta información (objeto de la clase *CharArraySet*). Para ello, recurre a una función del paquete de utilidades.
- A partir del idioma especificado en la configuración, decide que analizador debe utilizar. De acuerdo con los requerimientos del guion se incorpora el analizado en inglés y español (*EnglishAnalyzer* y *SpanishAnalyzer*). Estos proporcionan mecanismos de extracción de tokens, lematización, stemming y demás propios del idioma. En el caso de que se introduzca un idioma distinto de los anteriores, se utiliza el analizador genérico de Lucene (*StandardAnalyzer*).

```
List<String> words = StopWordsReader.readStopWords();
CharArraySet stopWords = StopFilter.makeStopSet(words, true);
Analyzer analyzer;  // Especializado en función del idioma
if (Configuration.LANGUAGE.equalsIgnoreCase("ES"))
    analyzer = new SpanishAnalyzer(stopWords);
else if (Configuration.LANGUAGE.equalsIgnoreCase("EN"))
    analyzer = new EnglishAnalyzer(stopWords);
else
    analyzer = new StandardAnalyzer(stopWords);
```

- Se comprueba la existencia del directorio donde se guardará el índice. Si este ya existe, se vaciará su contenido; en caso contrario se creará el directorio.

```
File directorio = new File(Configuration.INDEX_DIR);
if (directorio.exists() && directorio.isDirectory()) {
    System.out.println("Borrando directorio de índice existente: " + Configuration.INDEX_DIR);
    borrarDirectorio(directorio);
}
if (!directorio.exists()) {
    directorio.mkdir();
    System.out.println("Directorio para índice creado: " + Configuration.INDEX_DIR);
}
```

- Finalmente, se abrirá el directorio donde se guardará el índice (objeto de la clase *Directory*), se crea la configuración del indexador (mediante un objeto de la clase *IndexWriterConfig*) y se inicializa el indexador creando un objeto al que se le pasan como parámetro los objetos mencionados anteriormente.

```
Directory indexDirectory = FSDirectory.open(Paths.get(Configuration.INDEX_DIR));
IndexWriterConfig config = new IndexWriterConfig(analyzer);
writer = new IndexWriter(indexDirectory, config);
```

Una vez inicializado el objeto propio que actúa como indexador, se expone al usuario el método *doIndexing*, el cual añade al índice el resultado de procesar los documentos de texto presentes en el directorio especificado en la configuración del programa.

Este método lista todos los ficheros en el directorio de interés, comprueba que son visibles, existen, se pueden leer y pasan un proceso de filtrado (en nuestro caso, asegurar que son ficheros de texto plano). De cumplirse estos requisitos, se llama una función privada encargada de indexar dicho fichero.

```
public Long doIndexing(FileFilter filter) throws IOException {
    File[] files = new File(Configuration.DOCUMENTS_DIR).listFiles();
    assert files ≠ null;
    for (File file : files) {
        if (!file.isDirectory() && !file.isHidden() && file.exists() && file.canRead() && filter.accept(file))
            indexFile(file);
        else
            System.out.println("No se ha indexado " + file.getCanonicalPath());
    }
    return writer.getMaxCompletedSequenceNumber();
}
```

Esta función privada es *indexFile*, la cual recibe un objeto de tipo *File* (representación del fichero a procesar). Esta función es un "wrapper" del método *addDocument* del objeto de la clase *IndexWriter*. A su vez, esta función llama a otra función privada,

getDocumentToIndex, cuyo cometido es crear la estructura de datos propia de Lucene (objeto de la clase *Document*) para representar el documento y los campos que vamos a considerar. En nuestro caso serán 3, el contenido del fichero de texto en sí mismo, su nombre de fichero, y la ruta completa en el sistema de archivos.

```
private void indexFile(File file) throws IOException {
    System.out.println("Indexando " + file.getCanonicalPath());
    try {
        Document document = getDocumentToIndex(file);
        writer.addDocument(document);
    } catch (IOException e) {
        String errorWhere = "Error durante la indexación " + file.getCanonicalPath() + "\n";
        System.out.println(errorWhere + e.getMessage());
    }
}
```

```
private Document getDocumentToIndex(File file) throws IOException {
    Document document = new Document();
    TextField contentField = new TextField("contents", new FileReader(file));
    TextField fileNameField = new TextField("filename", file.getName(), TextField.Store.YES);
    TextField filePathField = new TextField("filepath", file.getCanonicalPath(), TextField.Store.YES);
    document.add(contentField);
    document.add(fileNameField);
    document.add(filePathField);
    return document;
}
```

Finalmente, se tiene un método *close* para cerrar el indexador. Cabe destacar que estamos tratando con ficheros y estructuras que los manejan, por lo que tenemos que asegurarnos de que ambas se cierran correctamente.

En este paquete se adjunta también un pequeño programa principal para que el usuario pueda especificar diferentes parámetros necesarios a la hora de crear el indexador y realizar el propio proceso de indexado de la colección documental. Este programa informará al usuario de los documentos que se han indexado y del tiempo que ha tomado dicho procesamiento.

2.2. Buscador.

Este paquete, siguiendo la misma filosofía que el anterior, proporciona una clase propia para abstraer el funcionamiento del buscador y un pequeño programa principal que permite al usuario especificar los parámetros necesarios para el funcionamiento del buscador y realizar las búsquedas en sí mismas.

El constructor de la clase propia se encarga de abrir el directorio donde se encuentra el índice que generamos gracias al módulo del indexador. Con esta información, se creará un lector del índice (objeto de la clase *IndexReader*) y un buscador dentro del mismo (objeto de la clase *IndexSearcher*).

Posteriormente, se sigue el mismo proceso de lectura del fichero de palabras vacías y creación del analizador en función del idioma especificado que se realizaba en el

constructor del indexador. Finalmente, se crea un objeto de la clase *QueryParser*, responsable de realizar el análisis sintáctico de las consultas que introducirá el usuario.

```
public Buscador() throws IOException {
    // Obtener directorio del índice del sistema de archivos
    Directory indexDirectory = FSDirectory.open(Paths.get(Configuration.INDEX_DIR));
    try {
        IndexReader indexReader = DirectoryReader.open(indexDirectory);
        indexSearcher = new IndexSearcher(indexReader);

        // El fichero de las palabras vacías viene de la configuración (ya sea del fichero o del usuario)
        List<String> words = StopWordsReader.readStopWords();
        CharArraySet stopWords = StopFilter.makeStopSet(words, true);

        Analyzer analyzer;  // Especializaremos en función del idioma
        if (Configuration.LANGUAGE.equalsIgnoreCase("ES"))
            analyzer = new SpanishAnalyzer(stopWords);
        else if (Configuration.LANGUAGE.equalsIgnoreCase("EN"))
            analyzer = new EnglishAnalyzer(stopWords);
        else
            analyzer = new StandardAnalyzer(stopWords);

        queryParser = new QueryParser("contents", analyzer);

    } catch (IOException e) {
        System.out.println("Error en la creación del buscador. \n" + e.getMessage());
    }
}
```

La clase *Buscador* proporciona adicionalmente un método para realizar las búsquedas en sí mismas (gracias al *QueryParser* y el *IndexSearcher*), y otro método para obtener el documento (como estructura de datos propia de Lucene), de manera que podamos consultar datos como la ruta completa del fichero o su nombre.

El programa principal adjunto en este módulo se encarga de inicializar la configuración del buscador y el propio objeto. Posteriormente, entrará en un bucle en el que permitirá al usuario introducir consultas y obtener resultados, hasta que dicho usuario indique que no quiere realizar más búsquedas.

Dicho programa principal hace uso de una función de esta misma clase, “wrapper” de la función de búsqueda de la clase del buscador, en la que se realiza un pequeño procesado para poder mostrar el número de resultados, mostrar la ruta completa o nombre del fichero (según configuración), mostrar el valor de similitud de cada documento devuelto y realizar un filtrado en función de dicho valor de así especificarse en la configuración.

2.3. Otras utilidades.

Este paquete está compuesto por 5 clases que sirven de apoyo a la funcionalidad principal de la aplicación, contenida en los paquetes anteriores. Comentemos brevemente el cometido de cada una de las clases:

- *DeleteDirectory*: proporciona la capacidad de borrar los ficheros dentro de un directorio y el propio directorio en sí. Se utiliza en la creación del indexador, si la carpeta destino del índice ya existe.

- *StopWordsReader*: dada una ruta en el sistema de archivos, abre el fichero correspondiente y lee las palabras vacías presentes en el mismo. Este método se crea teniendo en mente los documentos de palabras vacías que se nos ha proporcionado, en los cuales hay una única palabra por fila.
- *TextFileFilter*: pequeña clase cuyo cometido es comprobar si un fichero tiene la extensión .txt, y por tanto es un fichero de texto plano.
- *Configuration*: en esta clase se almacenan las variables que necesitan tanto indexador como buscador para funcionar correctamente. Estas son las rutas a los directorios del índice y la colección documental, el idioma, la ruta al fichero de palabras vacías, el criterio de búsqueda, el número máximo de resultados y si se desea filtrar los documentos recuperados por el buscador haciendo uso de un determinado umbral, así como el valor del propio umbral.
- *ModifyConfiguration*: en esta clase se cuenta con varios métodos que nos permiten modificar el valor de los parámetros descritos en el apartado anterior mediante un menú textual mostrado al usuario. No realiza persistencia de la configuración especificada por el usuario.

2.4. Obtención de la colección documental.

Para probar el correcto funcionamiento del software desarrollado se ha decidido utilizar una colección de libros, que he obtenido de Project Gutenberg.

La web está pensada para ser usada por humanos, por lo que los libros deben de ser descargados uno a uno. Para poder obtener un número considerable de ellos se desarrolló un pequeño script Python que hacía varias peticiones y guardaba los ficheros.

Sin embargo, los términos y condiciones de la página web prohíben este comportamiento, y como resultado, mi dirección IP fue baneada durante un par de horas. Así pues, tuve que descartar dicho script y hacer uso del “robot” propio de la página web, único método permitido para la descarga automática de ficheros.

Se usó la orden:

wget -w 2 -m -H "http://www.gutenberg.org/robot/harvest?filetypes[]=txt&langs[]=en"

en el terminal de Windows para descargar ficheros de texto plano. La ejecución toma un tiempo considerable, así que se detuvo una vez se habían descargado algo más de 450 ficheros.

Sin embargo, los documentos descargados se encontraban comprimidos en ficheros ZIP, y había redundancia de información, pues algunos hacían contenían el mismo libro. Por tanto, se desarrolló el script Python *get_collection.py,* adjunto en la entrega, que toma los ficheros comprimidos, extrae el libro de ellos, los coloca todos en una misma carpeta y elimina los repetidos.

Finalmente, hacemos uso de una colección documental compuesta por 227 libros aleatorios de diferentes temáticas. En particular, hay varios pasajes de la Biblia.

3. Mejoras e innovaciones.

Además de cumplir con los requisitos especificados en el guion de la práctica, se incorporan las siguientes mejoras e "innovaciones".

- Se proporciona una configuración por defecto, de manera que el usuario puede ejecutar el software directamente, sin preocuparse por esta.
- En caso de querer utilizar otros valores, distintos a los por defecto, se proporciona un menú textual donde se puede indicar de todos aquellos parámetros que sea necesarios por indexador y/o buscador.
- Se proporcionan analizadores específicos para inglés y español, de manera que realizan procesamientos de los textos particulares para cada idioma, los cuales permiten mejorar los resultados obtenidos. Dada la estructura del código, es muy sencillo incorporar analizadores para francés, italiano u otros idiomas en un futuro, usándose por el momento el analizador genérico en estos casos. Téngase en cuenta que idiomas como el chino, coreano o japonés requieren del uso de bibliotecas de Lucene específicas, no incluidas en este proyecto.
- Se ordenan los documentos devueltos por el buscador ante una consulta de mayor a menor relevancia.
- Se proporciona la capacidad de filtrar los documentos devueltos por el buscador haciendo uso de un umbral mínimo de relevancia (configurable).
- Se permite obtener el nombre del documento o la ruta completa de los documentos devueltos por el buscador.
- Se tiene la capacidad de indicar el número de resultados máximo que el buscador debe devolver al usuario.

4. Manual de usuario.

Para poder hacer uso del software desarrollado, este debe ser importado en el IDE que estemos utilizando. En el caso de IntelliJ IDEA, simplemente deberemos darle al botón de *Open* que hay en el menú que se lanza para seleccionar el proyecto antes de entrar al propio IDE.

Se aconseja importar la carpeta completa resultado de descomprimir el fichero adjunto en la entrega. De esta manera, se tendrán dos carpetas hijas, *documentación* y *proyecto.* De importarse solamente la carpeta *proyecto,* se deberán modificar las rutas por defecto especificadas en el fichero de configuración, o limitarnos a usar siempre la configuración por la línea de comandos.

Para crear el índice que posteriormente usará el buscador simplemente deberemos ejecutar el programa *MainIndexador*, el cual nos preguntará si deseamos hacer uso de la configuración por defecto.

En caso afirmativo, el proceso de creación del índice se realizará de forma automática, sin más intervención por parte del usuario. Al finalizar la ejecución, se mostrará el número de documentos indexados y el tiempo que ha tomado dicho procesamiento. El directorio con el índice se encontrará en *proyecto/index.*

```
¿Desea utilizar la configuración por defecto? (S/N)
S
Directorio para índice creado: ./proyecto/index
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10001.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10002.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10003.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10004.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10005.txt
```

```
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\G-Philemon.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\G-Philippians.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\G-Revelation.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\G-Romans.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\G-Titus.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\Introduction_and_Copyright.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\phys-0.txt
Tiempo de indexación de 227 documentos: 6056 milisegundos
```

En caso de que el usuario decida no utilizar la configuración por defecto se pedirán los datos necesarios por el indexador. Estos son el directorio donde crear el índice, el directorio de donde leer la colección documental, el idioma de los documentos y el fichero con las palabras vacías.

En caso de introducir datos erroneos, como un directorio para la colección documental no existe, o un fichero de palabras vacías no existente, el programa reconocerá el error y volverá a pedir el dato.

```
¿Desea utilizar la configuración por defecto? (S/N)
N
Configuración de los parámetros del indexador:

Introduzca el directorio donde crear el índice:
./proyecto/index
Introduzca el directorio con los documentos:
./proyecto/collection
Introduzca el idioma de los documentos (EN o ES):
EN
Introduzca el fichero de stopwords (txt):
./proyecto/data/stopwords_en.txt
Borrando directorio de índice existente: C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\.\proyecto\index
Directorio para índice creado: C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\.\proyecto\index
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10001.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10002.txt
Indexando C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10003.txt
```

Creado el índice, podemos a pasar a usar el buscador. Una vez más, se pedirá al usuario que confirme si desea utilizar la configuración ya establecida o la suya propia. Si acepta usar la configuración por defecto se entrará en un bucle en el que se le pedirá que introduzca la consulta a realizar, se le devolverá el resultado correspondiente y se le preguntará si desea seguir buscando.

```
¿Desea utilizar la configuración por defecto? (S/N)
S
Introduzca una consulta para realizar la búsqueda:
matthew

Tiempo de búsqueda: 50 milisegundos
Número de resultados: 15

Documento: 10100.txt  --> Score: 1.7259666
Documento: 12372.txt  --> Score: 1.7130105
Documento: G-Matthew.txt  --> Score: 1.6518188
Documento: 10116.txt  --> Score: 1.572145
Documento: 10058.txt  --> Score: 1.4995128
Documento: 10139.txt  --> Score: 1.4995128
Documento: 10433.txt  --> Score: 1.4288944
Documento: 10161.txt  --> Score: 1.3639534
Documento: 10129.txt  --> Score: 1.3446372
Documento: 10251.txt  --> Score: 1.2839718
Documento: G-Mark.txt  --> Score: 1.2695391
Documento: 10375.txt  --> Score: 1.2310181
Documento: 10099.txt  --> Score: 1.2215962
Documento: 10219.txt  --> Score: 1.2207849
Documento: 10531.txt  --> Score: 1.1921116

¿Desea realizar otra búsqueda? (S/N)
N
Gracias por usar el buscador.
```

En este caso se ha usado como término de búsqueda *matthew,* dado que he podido comprobar que aparece en varios documentos varias veces. En concreto, aparece en 33 documentos, aunque se mostrarán solo 15 resultados.

Podemos ver como se muestra el tiempo que ha demorado la búsqueda, los nombres de los documentos y el *score* de cada uno de ellos.

En el caso de usar la configuración manual, se pedirán los mismos parámetros que para el caso del indexador (los directorios de donde leer el índice y la colección documental, el idioma de los documentos y el fichero con las palabras vacías). Además, se le pedirá el número máximo de resultados a mostrar, si desea visualizar el nombre o ruta completa de los documentos que se devuelvan como resultado, si se desea utilizar el umbral de similiud (y su valor, en caso afirmativo).

Posteriormente, se entrará en el bucle en que se pide la consulta y se muestran los resultados, hasta que el usuario decida que no quiere buscar nada más.

```
¿Desea utilizar la configuración por defecto? (S/N)
N
Configuración de los parámetros de la búsqueda:

Introduzca el directorio del índice:
./proyecto/index
Introduzca el directorio con los documentos:
./proyecto/collection
Introduzca el idioma de los documentos (EN o ES):
EN
Introduzca el fichero de stopwords (txt):
./proyecto/data/stopwords_en.txt
Introduzca el número máximo de resultados a mostrar:
12
¿Desea buscar por nombre de fichero o por ruta de fichero? (filename/filepath)
filepath
¿Desea utilizar un umbral de similitud? (S/N)
S
Introduzca el umbral de similitud [0-100]:
1.5
Introduzca una consulta para realizar la búsqueda:
matthew

Tiempo de búsqueda: 42 milisegundos
Número de resultados: 4

Documento: C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10100.txt  --> Score: 1.7259666
Documento: C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\12372.txt  --> Score: 1.7130105
Documento: C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\G-Matthew.txt  --> Score: 1.6518188
Documento: C:\Users\Usuario\Desktop\MASTER\GIW\Practicas\Desarrollo\PracticaLucene\proyecto\collection\10116.txt  --> Score: 1.572145
```

En esta ocasión, solo 4 documentos cumplen los requisitos especificados.

5. Bibliografía.

Durante la realización se han consultado las siguientes fuentes de información:

- [1] Apache Lucene 9.5.0 Documentation. Apache Lucene Documentation
- [2] Lucene Tutorial – Index and Search Examples. HowToDoInJava
- [3] Introduction to Apache Lucene. Baledung
- [4] Lucene Tutorials. TutorialsPoint.
- [5] Searching and Indexing With Apache Lucene. DZone.
- [6] Lucene in 5 minutes. LuceneTutorial.
- [7] Project Gutenberg.