Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Máster Profesional en Ingeniería Informática

Curso 2020/2021

PRÁCTICA 1

Gestión de Información en la Web

Breve descripción

Desarrollo de un Sistema de Recuperación de Información con Lucene, Solr o ElasticSearch

Autor

Álvaro de la Flor Bonilla (alvdebon@correo.ugr.es) 15408846-L

RESUMEN

En algún momento nos puede surgir la necesidad de desarrollar un sistema de recuperación de información. Aunque podemos partir de cero, existe un gran número de bibliotecas en diferentes lenguajes de programación que nos pueden permitir montar el software de búsqueda de una forma rápida y sencilla.

Este es el caso de Lucene, el cual es un conjunto de bibliotecas, escritas en Java(nativo), C++, C#, PHP, Python, Ruby o Perl, que nos facilitará la confección de aplicaciones para realizar la indexación de grandes volúmenes de documentos y recuperación a partir de consultas suministradas por los usuarios del sistema.

En esta práctica se construirá un sistema de recuperación de información empleando la biblioteca Lucene (en su última versión publicada), compuesto de dos programas:

1. Un indexador, el cual recibirá como argumentos la ruta de la colección documental a indexar, el fichero de palabras vacías a emplear y la ruta donde alojar los índices, y llevará a cabo la indexación, creando los índices oportunos y ficheros auxiliares necesarios para la recuperación. Esta aplicación se ejecutará en la línea de mandatos y no tendrá ningún componente gráfico. Este software realizará las tareas de tokenización, eliminación de palabras vacías y extracción de raíces antes de crear el índice.
2. Un motor de búsqueda, que al ejecutarse recibirá como argumento la ruta donde está alojado el índice de la colección y permitirá que un usuario realice una consulta de texto y obtenga el conjunto de documentos relevantes a dicha consulta. En este caso, el programa sí será gráfico. Sobre la consulta se realizarán los mismos procesos que sobre los documentos en el indexador.

ÍNDICE DEL PROYECTO

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Colecciones

Para la realización de este trabajo se han utilizado tres colecciones de datos diferentes.

Enlace a las colecciones: https://drive.google.com/drive/folders/1s6rxEcYjG4YLrdBJLmrWYQHbdBAOUnuR?usp=sharing

1.1.1 NSF Research Awards Abstracts 1990-2003

En concreto se ha utilizado el paquete *"Abstracts_Part1.zip"* y dentro de este el directorio *"./Part1\awards_1994\awd_1994_96"*.

Esta colección tiene 288 elementos.

1.1.2 Wikipedia

Colección de artículos, disponibles en el directorio *"http://www.search-engines-book.com/collections/"*, en concreto se ha utilizado el archivo *"Wiki Large (tar.gz)"*.

Esta colección tiene 2000 elementos.

1.1.3 Project Gutenberg eBook

Se ha realizado un pequeño script para descargar los últimos libros disponibles en "*https://www.gutenberg.org/browse/scores/top"*. En concreto, en la respectiva sección será explicado este pequeño programa.

Esta colección tiene 100.

1.2 Bibliotecas y librerías

Para la realización de este proyecto ha sido utilizado *"Java"* en su versión 14 y *"Lucene"* en su versión 8.8.1 siendo las bibliotecas utilizadas:

1. lucene-analyzers-*-8.8.1.jar
2. lucene-core-8.8.1.jar
3. lucene-queryparser-8.8.1.jar

1.2.1 IDE de desarrollo

Para el desarrollo de la práctica hemos utilizado IntelliJ IDEA 2020.2.

2 DESCRIPCIÓN DEL DESARROLLO

Podemos destacar 3 secciones principales en el desarrollo. En concreto señalamos la construcción del módulo **indexador**, encargado de generar el índice a partir de la colección de datos que seleccione el usuario. Este indexador es utilizado por el módulo **buscador**, que será quien reciba las consultas del usuario y las ejecute sobre el índice creado anteriormente.

Ambos módulos vienen controlados por un modulo **auxiliar** cuya función será dirigir al usuario entre las distintas funcionalidades que permite realizar este proyecto.

Por último, también debemos destacar el pequeño script que fue desarrollado para destacar la colección de libros en formato HTML de la web *"Project Gutenberg"*.

2.1 Indexador

En primer lugar, para crear el indexador en sí, lo más importante que hemos tenido en cuenta ha sido diseñar un analizador que cuente con un filtro de palabras vacías.

```
public Indexer(String indexDirectoryPath) throws IOException {
    // Add stop words
    List<String> words = Execute.readWords();
    CharArraySet stopSet = StopFilter.makeStopSet(words);
    // Builds an analyzer with stop words
    StandardAnalyzer analyzer = new StandardAnalyzer(stopSet);
    // A Directory provides an abstraction layer for storing a list of files. A directory contains only files (no sub-folder hierarchy)
    Directory indexDirectory =  FSDirectory.open(Paths.get(indexDirectoryPath));
    // Holds all the configuration that is used to create an IndexWriter
    IndexWriterConfig iwriter = new IndexWriterConfig(analyzer);
    // An IndexWriter creates and maintains an index
    writer = new IndexWriter(indexDirectory, iwriter);
}
```

Ilustración 1 - Creación del índice principal

Como dato para tener en cuenta, es que existen dos conjuntos de palabras vacías utilizables, una para español y otra para inglés. Este parámetro es configurable por el usuario (después será explicado más en detalle en el módulo de configuración).

Una vez creado el indexador (que a su vez contiene el analizador) simplemente hemos usado el *"writer"* generado para ir añadiendo cada documento de la colección a nuestro índice.

En lo que queda de código que no ha sido mostrado *("Indexer.java")* se realiza lo que se comenta el en párrafo anterior, el análisis y adición en el índice en un bucle de cada elemento de la colección.

Por otro lado, se utiliza una clase auxiliar para ejecutar el indexador *("ExecuteIndex.java")*, en la que se configura como parámetros el directorio tanto del índice como de la colección. También se realiza un cálculo del tiempo empleado en la realización del indexado.

2.2 Buscador

La construcción de nuestro buscador también ha sido muy simple y cuenta con 4 pasos principales.

Lo primero que se debe hacer es construir el *"IndexReader"* que hará uso del *"index"* que ha sido creado en el punto anterior.

Después, hemos construido la consulta en sí, es decir la *"query"*. Para ello, hemos tenido en cuenta también en este caso la aplicación del análisis y de las palabras vacías en el input dado por el usuario (también se ha tenido en cuenta que si no existe ningún índice en ese momento será autogenerado utilizando la colección por defecto).

Una vez disponibles tanto el lector del índice como la consulta del usuario se realiza la consulta en sí, en la que, como curiosidad se ha intentado desarrollar un pequeño filtro umbral utilizando la puntuación de la búsqueda resultante.

2.2.1 Umbral

El objetivo de esta configuración es intentar eliminar situaciones en las que se tiene una consulta muy definida y nuestro buscador siempre devuelve como mínimo el número de resultados establecidos por defecto en su configuración, hacer una primera aproximación de cómo podría solucionarse el problema, ya que el parámetro *"score"* es muy variable de una consulta a otra y no posee un valor estándar.

En resumen, lo que hemos hecho es que, una vez obtenidos los resultados de la búsqueda, ajustar el parámetro de puntuación de cada una de ellas utilizando el valor máximo y mínimo obtenido en este conjunto.

Además, se ha utilizado un parámetro compensatorio en el cálculo del umbral para tener en cuenta las palabras utilizadas en la búsqueda del usuario.

```
public static Boolean calculateIndividualScore(Float score, String searchQuery, Float maxScore, Float minScore) {
    Boolean res = false;
    Integer words = Arrays.stream(searchQuery.split( regex: " ")).filter(x -> !x.equals(" ")).collect(Collectors.toList()).size();

    Float scoreNormalize = ((score + words * 0.47f) * 10) / maxScore;

    if ((1 - normalize(scoreNormalize, minScore)) < Constants.umbral) {
        res = true;
    }

    return res;
}
```

Ilustración 2 - Filtro por umbral

Cuando se hace el recuento de resultados, en el caso de que la puntuación normalizada y ajustada no supere el umbral establecido por el usuario, esta búsqueda será eliminada de la solución final.

2.3 Configurador

En primer lugar, en el archivo *"Constants.java"* se establece el conjunto de variables modificables por el usuario. De ellas, 6 son fácilmente modificables a través de las instrucciones interactivas disponibles en consola que serán explicadas en la sección de manual de usuario.

- ***MAX_SEARCH***. Establece el número de resultados mostrados tras la búsqueda realizada por el usuario.
- ***indexDir***. Es el directorio donde será almacenado el índice generado (y consultado cuando se realizan las búsquedas).
- ***dataDir***. Es el directorio donde se encuentran la colección de los ficheros de consulta.
- ***language***. Configura el lenguaje de las palabras vacías que será utilizado que podrá ser ingles (EN) o español (ES).
- ***umbral***. Valor utilizado para realizar el filtro basándose en la puntuación obtenida en la búsqueda.
- ***useUmbral***. Configura si se desea hacer filtrado por umbral o no.

```
En primer lugar, que desea hacer:
   (1)  Realizar búsqueda
   (2)  Volver a indexar
   (3)  Configurar el sistema
   (0)  Salir
Elija entre [0, 1, 2, 3]
3
¿Qué desea hacer?
   (1) Modificar el directorio del índice. Actual: ./src/index/indexDir/
   (2) Modificar el directorio de los documentos. Actual: ./src/index/docs1/
   (3) Editar el idioma de palabras vacías. Actual: EN
   (4) Editar máximo número de resultados. Actual: 10
   (5) ¿Usar umbral?. Actual: true
   (6) Editar el valor del umbral. Actual: 0.12
   (0) Salir
Elija entre [0, 1, 2, 3, 4, 5, 6]
2
Introduzca el nuevo directorio para los documentos
./src/index/docs2/
Directorio anterior: ./src/index/docs1/
Nuevo directorio: ./src/index/docs2/
¿Desea realizar otra operación? [s, n]
n
¡Adiós!
```

Ilustración 3 - Ejecución del configurador

En la imagen anterior se muestra el resumen de lo que podría realizarse. Una vez dentro del menú de configuración el sistema preguntará al usuario que desea modificar y únicamente tendrá que seguir las instrucciones que se le solicitan.

2.4 Scraping

Para la tercera colección que se ha utilizado (conjunto de libros en formato HTML), debido a que la única opción para obtenerla era la descarga individual de cada uno de los archivos, se ha desarrollado un pequeño script que descarga el top de libros de ese mes.

```
u = urlopen(url)
try:
    html = u.read().decode('utf-8')
finally:
    u.close()

soup = BeautifulSoup(html)
lis = soup.find_all("a", href=lambda href: href and "ebooks" in href)
lis = list(filter(lambda x: (x.get('href')), lis))
lis = list(map(lambda x: (x.get('href').replace('/ebooks/', '')), lis))
lis = list(set(lis))
lis = list(filter(lambda x: (x is not None and x.isnumeric()), lis))

print('Descargando')
print('Posibles libros a descargar: ' + str(len(lis)))

i = 1

with alive_bar(total=len(lis), title='Progress') as bar:
    z = 1
    for li in lis:
        f = open('./books/' + str(z) + '-' + li+'.html', 'w')
        f.close()
        z = z + 1
    for li in lis:
        link = 'https://www.gutenberg.org/files/' + li + '/' + li + '-h/' + li + '-h.htm'
        res = False
        try:
            urlretrieve(link, './books/' + str(i) + '-' + li + '.html')
        except Exception as e:
            print('Fallo al descargar: ' + link + ' Error: ' + str(e))
            res = True
```

Ilustración 4 – Pequeño extracto scrapping

Gracias a este, hemos obtenido una colección de 120 elementos (60 mb), que representan los libros más descargados del último mes. En este caso se han descargado en formato HTML por facilidad, pero no sería complicado modificarlos para obtenerlos en formatos ya sean *".txt"* o bien *".epub"*.

3 MANUAL DE USUARIO

Para ejecutar nuestro programa tendremos que ejecutar el archivo *"Main.java"*, disponible en el directorio *"./src/main/Main.java"*.

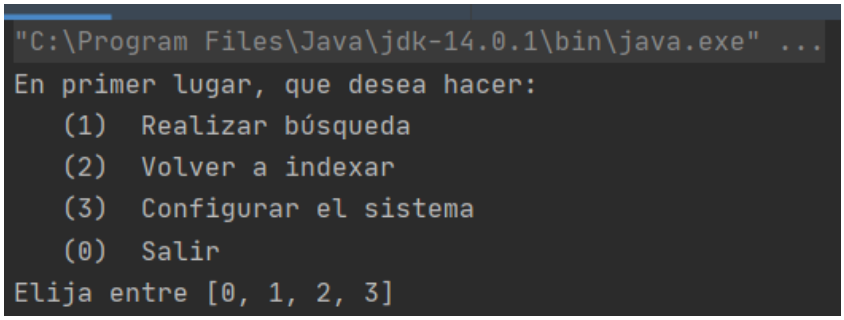

Ilustración 5 - Pantalla inicial

La imagen anterior muestra la pantalla inicial de la ejecución de nuestro programa. Como puede ver podemos elegir:

3.1 *Realizar búsqueda.*

Simplemente si introducimos 1 se iniciará el procede a iniciar el proceso de búsqueda.

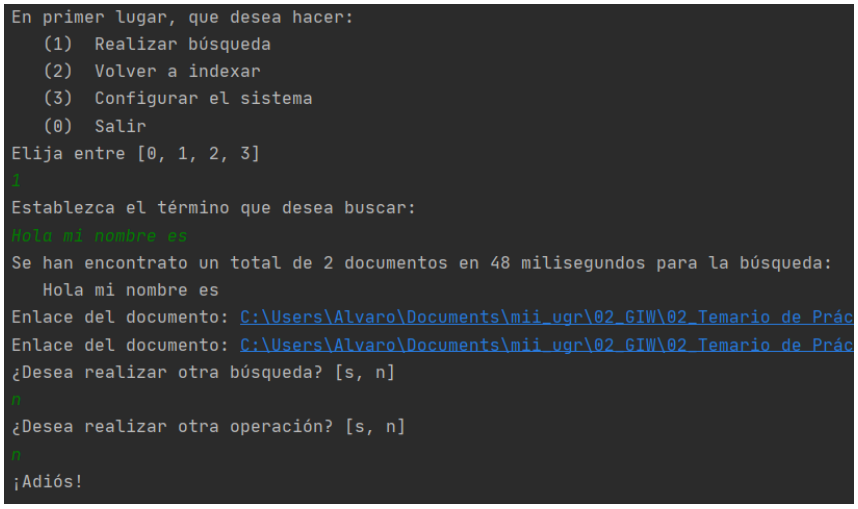

Ilustración 6 - Proceso de búsqueda

Nada más entrar en el menú búsqueda se le solicita al usuario la cadena a buscar. Una vez introducida comienza el proceso de búsqueda y muestra de resultados (hiperenlaces accesibles), además de mostrar la puntuación obtenida.

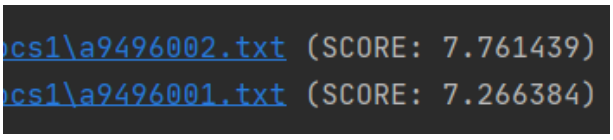

Ilustración 7 – Puntuación

En el caso de que no exista ningún índice creado y se intentará realizar una búsqueda se creará utilizando la colección establecida por defecto.

3.2 Volver a indexar

El objetivo de este menú es volver a construir el índice en los casos en los que o bien se ha modificado la colección o deseamos utilizar otra.

```
En primer lugar, que desea hacer:
   (1)  Realizar búsqueda
   (2)  Volver a indexar
   (3)  Configurar el sistema
   (0)  Salir
Elija entre [0, 1, 2, 3]
2
Borrando el índice anterior...
Indexando archivos.
Procesando---------100%
Se han indexado un total de 288 archivos en 524 ms
¿Desea realizar otra operación? [s, n]
```

Ilustración 8 - Indexación

El sistema en todo momento informa al usuario del porcentaje que queda por indexar y finalmente el total de archivos y tiempo empleado. Cada vez que se reconstruye el índice hemos tenido en cuenta borrar el anterior.

3.3 Configurar el sistema.

Si accedemos a este menú seremos capaces de configura los parámetros editables que comentamos en la sección anterior.

```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" ...
En primer lugar, que desea hacer:
   (1)  Realizar búsqueda
   (2)  Volver a indexar
   (3)  Configurar el sistema
   (0)  Salir
Elija entre [0, 1, 2, 3]
3
¿Qué desea hacer?
   (1) Modificar el directorio del índice. Actual: ./src/index/indexDir/
   (2) Modificar el directorio de los documentos. Actual: ./src/index/docs1/
   (3) Editar el idioma de palabras vacías. Actual: EN
   (4) Editar máximo número de resultados. Actual: 10
   (5) ¿Usar umbral?. Actual: true
   (6) Editar el valor del umbral. Actual: 0.12
   (0) Salir
Elija entre [0, 1, 2, 3, 4, 5, 6]
2
Introduzca el nuevo directorio para los documentos
./src/index/docs2/
Directorio anterior: ./src/index/docs1/
Nuevo directorio: ./src/index/docs2/
¿Desea realizar otra operación? [s, n]
```

Ilustración 9 - Configurar colección utilizada

En el ejemplo anterior hemos modificado la colección utilizada para indexar. Cabe destacar que si queremos reconstruir el sistema índice **no podemos parar la ejecución del programa** ya que se reestablecerán los valores por defecto. En este caso tendríamos que marcar la opción *"s"* en la última pregunta que aparece en la imagen anterior y volver a seleccionar la opción *"2"*, *"Volver a indexar"*.

4 PRUEBAS

Respecto a pruebas, en lo que más nos podemos centrar es en el uso del umbral. Por ejemplo, para este caso utilizaremos el conjunto *"doc1"* y la cadena *"Mi nombre es"*.

Si no utilizáramos el umbral, obtendríamos un resultado similar a este:

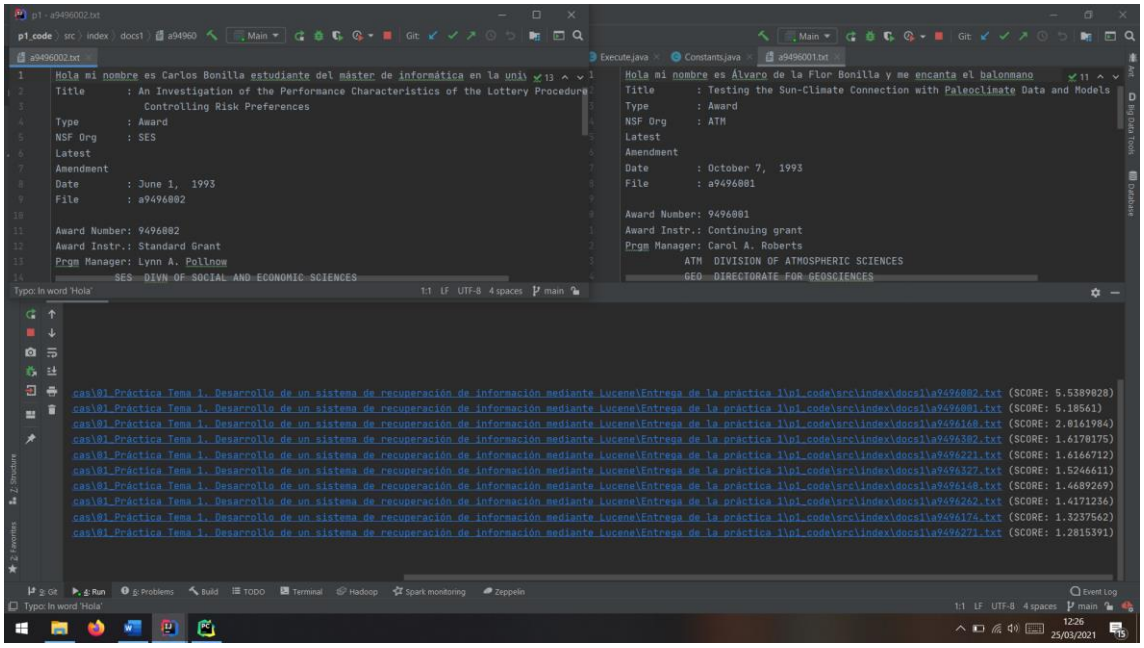

Ilustración 10 - Búsqueda sin umbral

Como puede verse, hay dos documentos que claramente son válidos en la búsqueda, sin embargo, el resto no tienen ningún sentido en este contexto, ya que de hecho no contienen la mayoría de las palabras del input del usuario.

En el caso de que activemos el filtro por umbral, el resultado será el siguiente.

```
Elija entre [0, 1, 2, 3]
1
Establezca el término que desea buscar:
Mi nombre es
Se han encontrato un total de 2 documentos en 52 milisegundos para la búsqueda:
   Mi nombre es
Enlace del documento: C:\Users\Alvaro\Documents\mii_ugr\02_GIW\02_Temario de Prácticas\0
Enlace del documento: C:\Users\Alvaro\Documents\mii_ugr\02_GIW\02_Temario de Prácticas\0
¿Desea realizar otra búsqueda? [s, n]
```

Ilustración 11 - Búsqueda con umbral

En este caso, todos los resultados que se alejan mucho de las mejores puntuaciones obtenidas son eliminadas del conjunto mostrado finalmente.

5 BIBLIOGRAFÍA

https://lucene.apache.org/

http://www.lucenetutorial.com/lucene-in-5-minutes.html

https://dzone.com/articles/apache-lucene-a-high-performance-and-full-featured