GESTIÓN DE INFORMACIÓN EN LA WEB

MÁSTER PROFESIONAL EN INGENIERÍA EN INFORMÁTICA

Práctica 3: Desarrollo de un Sistema de Recuperación de Información con Lucene

Autor
José Ángel Díaz García

Mail
joseadiazg02@correo.ugr.es

DNI
76139799R

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, Marzo de 2017

Índice general

Índice de figuras

Capítulo 1

Desarrollo práctico

La práctica número 3 de la asignatura Gestión de Información en la Web (GIW), se centra en el desarrollo de un sistema de recuperación de información basado en Lucene [1]. Concretamente, los objetivos de la práctica y los cuales iremos solventando o concluyendo en las siguientes secciones son los que podemos encontrar en la siguiente lista.

1. Conocer las partes principales que tiene un sistema de recuperación de información y qué funcionalidad tiene cada una.
2. Implementar un sistema de recuperación de información.
3. Emplear la biblioteca Lucene para facilitar dicha implementación.

1.1. Partes del sistema

En el sistema de recuperación de información tendremos dos partes diferenciadas en dos programas. Por un lado el programa indexador **indexer.java** que recibirá la colección de documentos y creará el indice. La segunda parte del sistema, la compone el programa de búsqueda **searcher.java** que leerá el indice creado por el anterior documento y nos ofrecerá los resultados.

1.2. Requisitos del sistema

En esta sección se realiza un pequeño análisis de requisitos funcionales y no funcionales que el sistema deberá cumplir.

1.2.1. Requisitos Funcionales

- **RF1**: El sistema permitirá la realización de búsquedas por texto de las noticias.
- **RF2**: El sistema permitirá buscar noticias por rango de fechas.
- **RF3**: Los resultados se mostraran en una caja de texto con un formato correcto para que el usuario final pueda entenderlos perfectamente.
- **RF4**: La caja de texto pasará a verde cuando la consulta ofrezca resultados.
- **RF5**: Se podrá abortar la aplicación por medio de un botón salir.

1.2.2. Requisitos No Funcionales

- **RNF1**: La aplicación de indexación deberá llevar a cabo la indexación en un tiempo aceptable..
- **RNF2**: La aplicación de indexación creara el indice en un directorio relativo dentro de su misma ubicación para permitir así a la aplicación de búsqueda obtener acceso a esta sin problemas..

1.3. Diseño de la solución

En este punto se ven los conceptos de diseño y desarrollo software tenidos en cuenta durante la práctica.

1.3.1. indexer

La aplicación de indexación esta dividida en 4 clases: La clase principal donde se lleva a cabo el proceso de indexación, una clase auxiliar para crear la estructura de datos necesaria para manejar las noticias y por ultimo las clases más importantes: **parser.java** encargada de obtener las etiquetas interesantes del fichero XML y cargarlas en las estructuras provistas para ello y por último **lucenewriter.java** que implementa todo el proceso de añadir nuevos documentos al indice.

- **noticia.java**: Tiene como atributos privado de clase los elementos de información necesarios de la noticia y como públicos una serie de métodos get-set que serán usados en el proceso de búsqueda para obtener una información que sea relevante para el usuario y de manera estructurada. Notar la sencillez que esta modularización nos ofrecería de cara a añadir nuevos elementos al indice.
- **indexer.java**: Implementa el método principal. Este comprueba que los argumentos que veremos en la seccion 1.4 sean los adecuados y tras ello obtiene cada una de las noticias, las recorre y llama a la clase lucenewriter para añadir los elementos al indice.
- **lucenewriter.java**: Esta clase implementa los métodos necesarios para el manejo del indice, por un lado abrirlo y cerrarlo y por otro añadir noticias, donde hemos tenido las siguientes consideraciones:
  - Title: Este texto lo guardamos como **StringField** para permitir búsquedas literales por el título.
  - Date: Se ha incluido el campo fecha, para poder buscar por rango de noticias, este campo será por tanto un cambo **IntField**.
  - Text: Quizá sea el elemento mas complicado dado que su tamaño nos puede complicar mucho la gestión del indice, dado que necesitamos buscar por el, pero necesitamos que el indice sea eficiente no podemos asignar StringField sino que lo guardaremos como **TextField** lo que conlleva que se lleven a cabo los procesos de tokenizado y mejora vistos en teoría.
- **parser.java**: Implementa el parseo de los documentos xml usando el DOM.

1.3.2. searcher

La aplicación de búsqueda consta de tres clases. Por un lado la anteriormente mencionada de **Noticia.java** para el manejo de noticias, por otro lado el **searcher.java**, donde se implementan las consultas en si, y por último **interfaz.java**, donde tenemos esta implementada la interfaz gráfica del sistema.

La clase **interfaz.java** ha sido creada usando la suite para creación de interfaces de java, por lo que no tiene mayor dificultad en su creación. Simplemente debemos tener cuidado de cuando se pulse el botón se llame al método en **searcher.java** con los parámetros apropiados.

Respecto a la clase **searcher.java**, consta de los siguientes métodos de búsqueda que llevan a cabo el cumplimiento de los requisitos funcionales de búsqueda vistos en 1.2.1.

- **public ArrayList<Noticia>busquedaTexto(String cadena)**:

  Este método es usado para la búsqueda de textos de la manera siguiente. Se abre la ubicación del indice y se crea un buscador sobre el indice. Se crea un parser y como el **Text** estaba tokenizado debemos tratar de igual forma la entrada de la consulta. Por último introduciremos en un ArrayList de noticias las noticias que cumplan nuestro criterio de búsqueda y las devolveremos a la clase interfaz.java para ser mostradas en la interfaz de usuario. Este método de presentación de resultados lo haremos usando los get que teníamos en la clase noticias y será usado de igual forma en todas los métodos.

- **public ArrayList<Noticia>consultaRango(Integer rangoA, Integer rangoB)**: Este método se encarga de las consultas por rango sobre **Date** para ello simplemente se fijará en este campo. Nos hemos basado en ejemplos e informacion obtenida de la documentacion de Lucene [2]. La consulta se crea de la siguiente manera:

```
Query q = NumericRangeQuery.newIntRange("Date", rangoA,
    rangoB, true, true);
```

1.4. Manual de uso

En esta sección de la memoria veremos como ejecutar y probar el sistema.

1.4.1. Indexar

El ejecutable indexar recibe tres parámetros:

1. Directorio con la colección de documentos.
2. Directorio de creación del indice.
3. Directorio de las palabras vacías.

En nuestro caso, deberemos ejecutar en terminal el siguiente comando:

```
java -jar index.jar testdata/ ../searcher/indexDir
    testdata/palabrasvacias.txt
```

1.4.2. Buscar

Una vez indexada la colección, mediante terminal accederemos al directorio searcher y ejecutaremos el comando:

```
java -jar searcher.jar
```

Tras esto podremos ver la interfaz de búsqueda que podemos ver en la imagen 1.1:

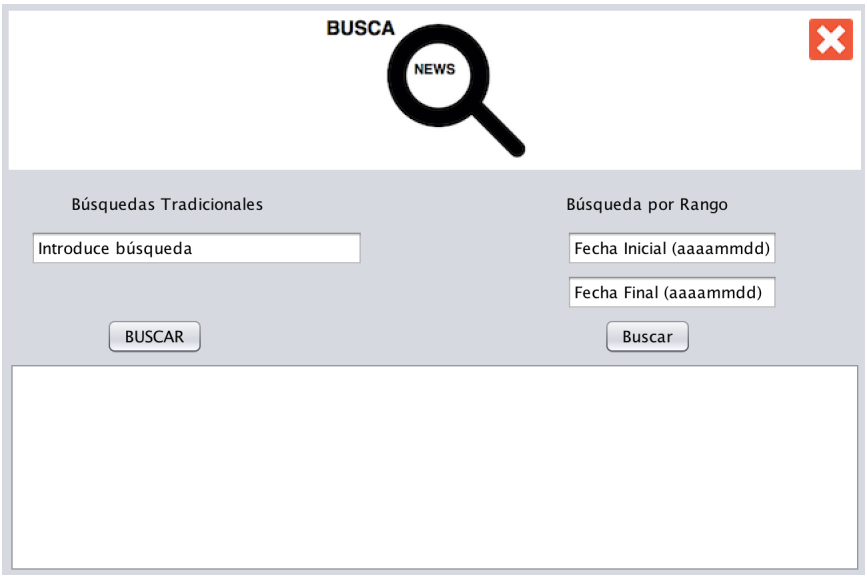

Figura 1.1: Interfaz del sistema de búsqueda.

1.5. Conclusiones

Tras la realización de la practica hemos podido constatar de la potencia de Lucene para la realización de aplicaciones de diverso tipo. También es destacable la facilidad de uso al menos de las nociones básicas y la cantidad de documentación que podemos obtener de la comunidad en internet.

Bibliografía

[1] Lucene https://lucene.apache.org

[2] API Lucene para consultas por rangohttps://lucene.apache.org/core/4_3_0/core/org/apache/lucene/search/NumericRangeQuery.html