Búsqueda y Minería de Información 2016-2017

Universidad Autónoma de Madrid, Escuela Politécnica Superior

Grado en Ingeniería Informática 4º curso

Práctica 1 – Implementación de un motor de búsqueda con Apache Lucene

Fechas

- Comienzo: lunes 6 de febrero / miércoles 8 de febrero
- Entrega: lunes 20 de febrero / miércoles 22 de febrero (hora límite 12:00)

Objetivos

Los objetivos de esta práctica son:

- La iniciación a la implementación de un motor de búsqueda.
- Una primera comprensión de los elementos básicos necesarios para implementar un motor completo.
- La iniciación al uso de la librería Lucene de Apache: creación y utilización de índices, funcionalidades de búsqueda en texto.
- La iniciación a la implementación de una función de ránking sencilla.

Lucene es una librería Java muy popular y utilizada en el desarrollo de funcionalidades de búsqueda para aplicaciones comerciales. Lucene proporciona implementaciones de algoritmos de indexación de texto, así como modelos de Recuperación de Información (IR) con los que se pueden responder consultas devolviendo ránkings de documentos según diferentes modelos de IR tales como el modelo vectorial tf-idf estudiado en la asignatura.

Los documentos que se indexarán en esta práctica, y sobre los que se realizarán consultas de búsqueda serán documentos HTML, que deberán ser tratados para extraer y procesar el texto contenido en ellos.

La práctica plantea como punto de partida un diseño de API general, que pueda implementarse de diferentes maneras, como así se hará en esta práctica y las siguientes. A modo de toma de contacto y arranque de la asignatura, en esta primera práctica se completará una implementación de la API utilizando Lucene, con lo que resultará bastante trivial la solución (en cuanto a la cantidad de código a escribir). En la siguiente práctica el estudiante desarrollará sus propias implementaciones, sustituyendo el uso de Lucene que vamos a hacer en esta primera práctica.

La finalidad del diseño de clases proporcionado es poder comparar con Lucene las implementaciones propias que el estudiante desarrollará en la siguiente práctica. Facilitará además la realización gradual del trabajo, sustituyendo primero la capa de buscador, y después la de índice.

Material proporcionado

Se proporcionan:

- Varias clases e interfaces Java con las que el estudiante integrará las suyas propias. En particular, la clase `es.uam.eps.bmi.search.test.TestEngine` incluye un progama main que deberá funcionar con las clases a implementar por el estudiante.
- Una pequeña colección docs.zip con 1.000 documentos HTML, y un pequeño fichero urls.txt. Ambas representan colecciones de prueba para depurar las implementaciones y comprobar su corrección.
- Un documento de texto “test-output.txt” con la salida estándar que deberá producir la ejecución del programa “java TestEngine”. Nota: en lugar de “C:\Users\pablo\Desktop\BMI” aparecerá obviamente la ruta donde el estudiante ejecute el proyecto Netbeans o Eclipse.

Además, el estudiante utilizará los .jar necesarios de la librería Lucene. Como mínimo, se precisarán lucene-core-6.4.1.jar y lucene-queryparser-6.4.1.jar.

Por último para el parsing de HTML se sugiere por ejemplo la librería JSoup.

Ejercicios

1. Implementación basada en Lucene

Implementar las clases e interfaces necesarias para que el programa funcione (las interfaces se muestran en cursiva). Se deja al estudiante deducir alguna de las relaciones jerárquicas entre clases.

1.1 Indexación (3pt)

Definir:

- La interfaz `es.uam.eps.bmi.search.index.`*`Index`*.
- La clase `es.uam.eps.bmi.search.index.lucene.LuceneIndex` como subclase de `es.uam.eps.bmi.search.index.AbstractIndex`.
- La clase `es.uam.eps.bmi.search.index.lucene.LuceneIndexBuilder` como implementación de `es.uam.eps.bmi.search.index.IndexBuilder`.

Se sugiere utilizar dos Field's: ruta del documento (dirección Web o ruta en disco), y contenido del documento.

La entrada para el constructor del índice podrá ser a) un fichero de texto con direcciones Web (una por línea); b) una carpeta del disco (se indexarán todos los ficheros de la carpeta, sin entrar en subcarpetas); o c) un archivo zip que contiene archivos comprimidos a indexar. Supondremos que el contenido a indexar es siempre HTML.

1.2 Búsqueda (1pt)

Implementar las clases:

- `es.uam.eps.bmi.search.lucene.LuceneEngine`.
- `es.uam.eps.bmi.search.ranking.lucene.LuceneRanking`.
- `es.uam.eps.bmi.search.ranking.SearchRankingDoc` como implementación de `Comparable<SearchRankingDoc>`.
- `es.uam.eps.bmi.search.ui.TextResultDocRenderer`, como implementación de `es.uam.eps.bmi.search.ui.`*`ResultsRenderer`*.

La figura 1 ilustra informalmente la relación entre las principales piezas del diseño de clases e interfaces a implementar y las que se proporcionan.

2. Modelo vectorial

Implementar un modelo de ránking propio, basado en el modelo vectorial.

2.1 Producto escalar (1.5pt)

Implementar un modelo vectorial propio que utilice el producto escalar (sin dividir por las normas de los vectores) como función de ránking, por medio de las clases:

- `es.uam.eps.bmi.search.vsm.VSMEngine`, como subclase de `es.uam.eps.bmi.search.AbstractEngine`.
- Una implementación propia de las interfaces del paquete `es.uam.eps.bmi.search.ranking`, que se necesitará para recibir y procesar los resultados (ránking) del motor de modelo vectorial. Ubicar estas clases en el paquete `es.uam.eps.bmi.search.ranking.impl`.

Este modelo hará uso de la interfaz `Index`, y se podrá probar con la implementación `LuceneIndex`.

En la figura 1, la clase `VSMEngine` será intercambiable con `LuceneEngine`, y las clases para almacenar los resultados harán el mismo papel que `LuceneRanking`, pero sin utilizar las clases internas de Lucene como `ScoreDocs`.

2.1 Coseno (1.5pt)

Refinar la implementación del modelo para que calcule el coseno. Para ello se necesitará extender la implementación del índice de Lucene `LuceneIndexBuilder` con el cálculo de los módulos de los vectores, que deberán almacenarse en un fichero, en la carpeta de índice junto a los ficheros que genera Lucene.

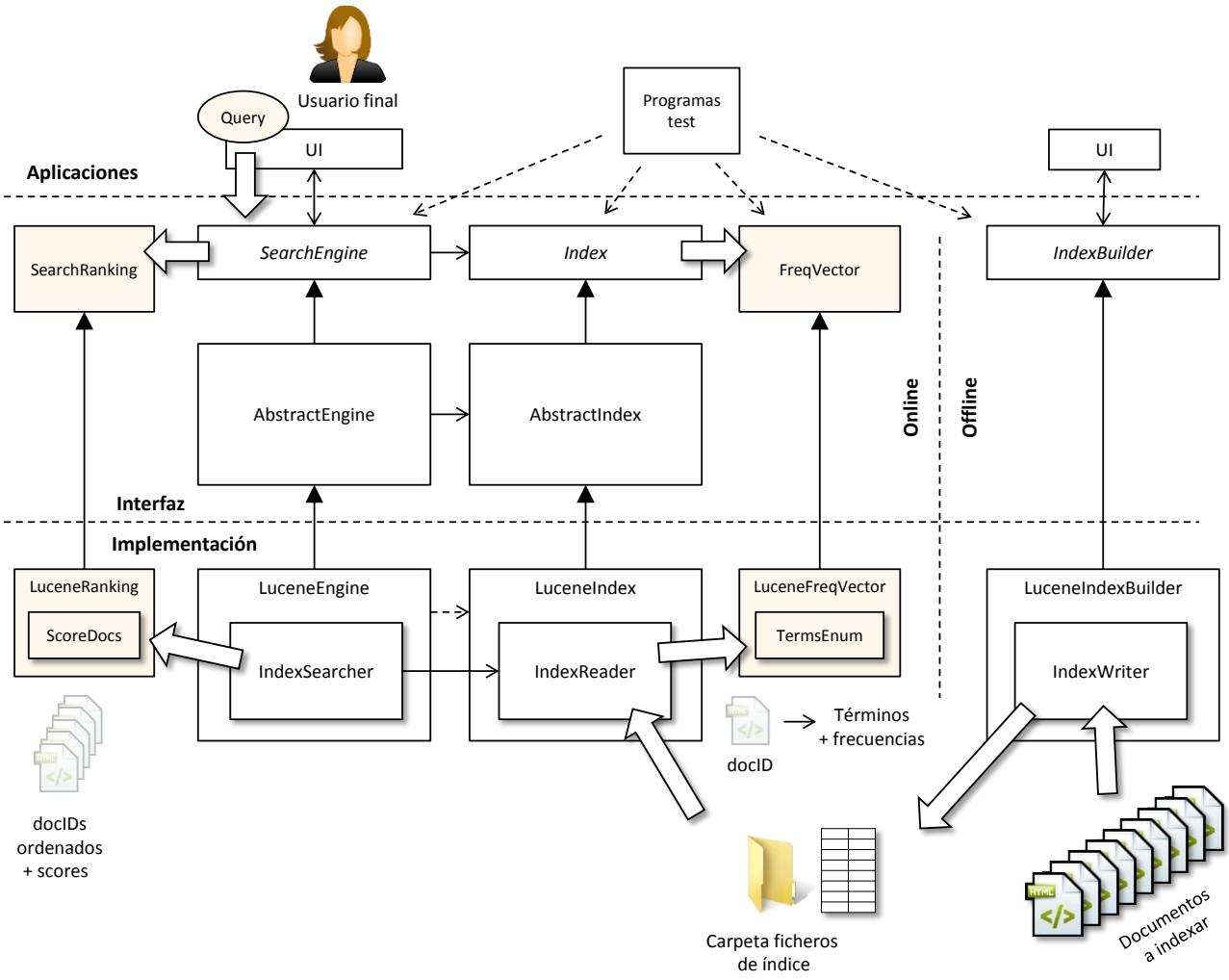

Figura 1. Arquitectura y diseño de clases.

3. Extensiones

3.1 Estadísticas de frecuencias (1pt)

Utilizando las funcionalidades de la interfaz `Index`, se implementará una clase `es.uam.eps.bmi.search.test.TermStats` con un método main que vuelque a dos ficheros a) las frecuencias totales en la colección de los términos, ordenadas de mayor a menor (guardar en termfreq.txt), y b) el número de documentos que contiene cada término, igualmente de mayor a menor (guardar en termdocfreq.txt). El formato será de una línea "término \t frec" por cada término contenido en el índice.

Las estadísticas obtenidas se visualizarán en dos gráficas en escala log-log, que se incluirán en la memoria.

3.2 Variaciones de Lucene (1pt)

Experimentar diferentes opciones de Lucene, tales como:

- El uso de stopwords y/o stemming.
- Diferentes modelos de IR: BM25, Query Likelihood, etc.
- El tratamiento de más formatos de documento, tales como pdf o Word, pudiendo utilizar (en su caso) librerías externas que los procesen.

3.3 Interfaz de usuario (2pt)

Implementar una interfaz de usuario sencilla en la que un usuario pueda introducir consultas y visualizar los resultados en una ventana, mostrando el score y la ruta de cada documento. Opcionalmente, la interfaz mostrará (por el medio que el estudiante juzgue oportuno) el contenido de un documento cuando el usuario clique sobre él.

Entrega

La entrega consistirá en un único fichero zip con el nombre **bmi-p1-XX.zip**, donde XX debe sustituirse por el número de pareja (01, 02, …, 10, …). Este fichero contendrá:

- Una carpeta **scr/** con todos (**y sólo**) los ficheros .java con las implementaciones de los ejercicios 1, 2 y 3. Los .java se ubicarán en la ruta apropiada de subcarpetas según sus packages.
- En su caso, una carpeta **lib/** con los .jar adicionales que fuesen necesarios (no se incluirán los de Lucene ni JSoup). Se recomienda no obstante consultar con el profesor antes de hacer uso de librerías adicionales.
- Una **memoria en pdf** donde se documentará:
  - Qué versión del modelo vectorial se ha implementado en el ejercicio 2.
  - El trabajo realizado en el ejercicio 3. Sugerencias: 3.1 figuras de las frecuencias; 3.2 descripción de las diferentes opciones probadas, y los cambios observados en los resultados; 3.3 explicación y captura de pantalla de la interfaz de usuario, e indicaciones para ejecutarla.
  - Y cualquier otro aspecto que el estudiante considere oportuno destacar.

  La calidad de la memoria representará orientativamente el 10% de la puntuación de los ejercicios que se documentan en la misma.

No se deberán incluir en el .zip ninguno de los materiales proporcionados por el profesor (ni tan siquiera las clases Java) ni los .jar de Lucene ni JSoup, ni los archivos de proyecto Netbeans o Eclipse –ni por supuesto ningún .class!

El fichero de entrega se enviará por el enlace habilitado al efecto en el curso **Moodle** de la asignatura.

Calificación

Esta práctica se calificará con una puntuación de 0 a 10 atendiendo a las puntuaciones individuales de ejercicios y apartados dadas en el enunciado. El peso de la nota de esta práctica en la calificación final de prácticas es del **20%**.

La calificación se basará en a) el número de ejercicios realizados y b) la calidad de los mismos.

Para dar por válida la realización un ejercicio, el código deberá funcionar (a la primera) integrado con las clases que se facilitan, tal cual se facilitan (sin ninguna modificación). El profesor comprobará este aspecto añadiendo las clases entregadas por el estudiante a las clases facilitadas en la práctica, ejecutando el programa `TestMain` así como otros main de prueba adicionales.