Universidad Autónoma de Madrid, Escuela Politécnica Superior

Grado en Ingeniería Informática

Búsqueda y Minería de Información 2013-2014

Práctica 3: Implementación de un motor de búsqueda

Fechas

- Comienzo: 21 de febrero
- Entrega: 14 de marzo (hora límite 12:00)

Objetivos

El objetivo de esta práctica es aprender aspectos fundamentales sobre la implementación de un motor de búsqueda, en concreto:

- El procesado de texto
- La creación de índices de búsqueda
- La implementación de modelos de recuperación de información
- La evaluación de resultados de búsqueda

Para ello se van a desarrollar varios **índices** (realizando o no realizando determinadas tareas de procesado de texto) y varios **modelos de recuperación de información**: el modelo Booleano, el modelo vectorial y el modelo de búsqueda literal.

Colección de documentos

Los índices y buscadores de esta práctica se van a construir y evaluar sobre las tres colecciones de documentos HTML de la práctica 2: **clueweb-1K.tgz** (10 MB), **clueweb-10K.tgz** (82 MB) y **clueweb-100K.tgz** (865 MB).

Se recuerda que cada colección está formada por:

- Un fichero comprimido **docs.zip**, con los documentos HTML.
- Un fichero **queries.txt** con las consultas de búsqueda, en el formato `consulta_id:consulta`.
- Un fichero **relevance.txt** con aquellos documentos relevantes a cada consulta. Por cada consulta, se da la lista de documentos siguiendo el formato `consulta_id \t documento_1 \t documento_2 \t ... documento_N`.

Ejercicios

Ejercicio 1: Creación de índices [3 puntos]

En esta práctica se van implementar en Java varios indexadores y dos modelos de búsqueda. Los diseños de clases y arquitectura son libres. Sin embargo, para poder evaluarlos de forma común, se exige que soporten las clases e interfaces de la práctica 2[1]:

- Clase `es.uam.eps.bmi.search.Term`
- Clase `es.uam.eps.bmi.search.TextDocument`
- Clase `es.uam.eps.bmi.search.indexing.Posting`
- Interfaz `es.uam.eps.bmi.search.indexing.Index`
- Clase `es.uam.eps.bmi.search.searching.ScoredTextDocument` (*implements* `java.lang.Comparable`)
- Interfaz `es.uam.eps.bmi.search.searching.Searcher`
- Interfaz `es.uam.eps.bmi.search.parsing.TextParser`

[1] La función getDocumentPostings no es necesario de implementar

En este ejercicio se pide diseñar e implementar en Java varios indexadores que cumplan la interfaz **`es.uam.eps.bmi.search.indexing.Index`**. Los índices se distinguirán por las funcionalidades de procesado de texto que proporcionen.

En concreto, la solución de este ejercicio constará al menos de:

- **Clase `es.uam.eps.bmi.search.indexing.BasicIndex`** [2 puntos], asociada a un índice que no hace ni filtrado de *stopwords* ni *stemming* de términos. Se deja a libre elección aspectos de procesado adicionales (además de los asociados al procesado de código HTML), p.e. filtrado de signos de puntuación, distinción o no de mayúsculas y minúsculas, etc.
- **Clase `es.uam.eps.bmi.search.indexing.StopwordIndex`** [0,5 puntos], asociada a un índice que sí hace filtrado de *stopwords*, pero no hace *stemming* de términos. Esta clase se puede implementar como una extensión de `BasicIndexing`. Podrá hacer uso de la lista de *stopwords* suministrada en el fichero **stopwords.txt** u otra diferente.
- **Clase `es.uam.eps.bmi.search.indexing.StemIndex`** [0,5 puntos], asociada a un índice que no hace filtrado de *stopwords*, pero sí hace *stemming* de términos. Esta clase se puede implementar como una extensión de `BasicIndexing`. Podrá hacer uso del **Snowball stemmer** proporcionada en esta práctica u otro diferente.
- **Clase `es.uam.eps.bmi.search.indexing.AdvancedIndex`**, asociada a un índice que sí hace filtrado de *stopwords* y *stemming* de términos. Se puede implementar como una extensión de `StopwordIndex` y `StemIndexing`.
- **Clase `es.uam.eps.bmi.search.indexing.IndexBuilder`** con un único método *main* que a partir de la colección de documentos proporcionada creará sus índices de los cuatro tipos anteriores. El *main* recibirá dos argumentos de entrada: la ruta de la carpeta que contiene la colección de documentos con los que crear los cuatro índices, y la ruta de la carpeta en la que almacenar los índices creados. En concreto, en esa carpeta se crearán cuatro subcarpetas – `basic`, `stopword`, `stem`, `advanced` – en las que almacenar los índices correspondientes.

Ejercicio 2: Implementación de modelos de recuperación de información [6 puntos]

En este ejercicio se pide diseñar e implementar en Java varios buscadores que cumplan la interfaz **`es.uam.eps.bmi.search.searching.Searcher`**. Los índices se distinguirán por las funcionalidades de procesado de texto que proporcionen. En concreto, la solución de este ejercicio constará al menos de:

- **Clase `es.uam.eps.bmi.search.searching.BooleanSearcher`** [1 puntos], asociada a un buscador que implemente el modelo de recuperación de información Booleano. Este buscador deberá permitir establecer si los términos de una consulta están relacionados con operadores OR o AND. De este modo, si el buscador está en modo OR, la consulta "Brad Pitt romance films" será equivalente a "Brad OR Pitt OR romance OR films"; Y si está en modo AND, será equivalente a "Brad AND Pitt AND romance AND films".
- **Clase `es.uam.eps.bmi.search.searching.TFIDFSearcher`** [3 puntos], asociada a un buscador que implemente el modelo de recuperación de información vectorial con ponderación de términos TF-IDF.
- **Clase `es.uam.eps.bmi.search.searching.LiteralMatchingSearcher`** [2 puntos], asociada a un buscador que implemente una búsqueda literal. Esto es, una búsqueda en la que las palabras de la consulta van entre comillas, y los documentos devueltos deben contener estos términos de forma consecutiva y en el mismo orden en que se dan en la consulta. Para simplificar, no será necesario soportar consultas mixtas con partes literales y partes sin entrecomillar. Los resultados deberán ordenarse según los principios del modelo vectorial, como si las palabras entrecomilladas fuesen un único término.

Ejercicio 3: Evaluación de resultados de búsqueda [1 punto]

En este ejercicio se pide implementar una clase Java **`es.uam.eps.bmi.search.searching.SearcherTest`** que permita ejecutar consultas con los 4 buscadores implementados (**`BooleanSearcher`** en modo OR, **`BooleanSearcher`** en modo AND, **`TFIDFSearcher`** y **`LiteralMatchingSearcher`**) sobre los 4 índices creados (**`BasicIndex`**, **`StopwordIndex`**, **`StemIndex`** y **`AdvancedIndex`**). De este modo, en total se considerarán 16 configuraciones de motor de búsqueda (índice + buscador).

Estas 16 configuraciones se ejecutarán y se evaluarán con las consultas y juicios de relevancia dados en los ficheros de cada una de las 3 colecciones ClueWeb. Como en la práctica 2, en ésta se pide calcular y reportar los valores promedios de P@5 y P@10 obtenidos para las consultas de cada colección.

La solución al ejercicio será un documento PDF de nombre **bmi1314_XXXX_p3_YY_resultados.pdf** con tablas mostrando los valores de P@5 y P@10 promedios, y un análisis de los resultados obtenidos. Este análisis no tendrá como objetivo determinar qué motor de búsqueda es mejor, sino detectar razones por las que los motores recuperan

documentos no relevantes, y argumentar posibles modificaciones/extensiones de índices y buscadores para la mejora de estos.

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 calificación de la práctica en la calificación final de prácticas es del **25%**.

NOTA: La selección del método de indización y búsqueda repercutirá en la calificación de la práctica. Por ejemplo, una práctica en la que el índice se implemente en RAM puntuará a lo sumo 8 sobre 10, mientras que una implementación con índice en disco aspira a la máxima nota.

Entrega

La entrega de esta práctica consistirá en un fichero ZIP con el nombre **bmi1314_XXXX_p3_YY.zip**, donde XXXX debe sustituirse por el grupo 2461 ó 2462 según corresponda, e YY debe sustituirse por el número de pareja (01, 02, …, 10, …). Este fichero contendrá:

- Una carpeta **src** con todos los paquetes y ficheros fuente .java desarrollados.
- Una carpeta **doc** con los documentos HTML generados mediante la herramienta javadoc. Nótese que las clases y métodos implementados deberán ir debidamente documentados con sus correspondientes cabeceras.
- Un documento **bmi1314_XXXX_p3_YY_memoria.pdf** en el que se expliquen aquellos aspectos de diseño e implementación que se consideren oportunos sobre los ejercicios 1 y 2, p.e. estructuras de datos, contenido del índice (en RAM y/o en disco), cálculos desarrollados en los buscadores, ejemplos de resultados de búsqueda para algunas consultas, etc.
- El documento **bmi1314_XXXX_p3_YY_resultados.pdf** con los resultados del ejercicio 3.

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