# Práctica 4

Implementación de un Sistema de Recuperación de Información utilizando Lucene

Lothar Soto Palma Daniel López García Iván Calle Gil
José Carlos Entrena Jiménez

## Índice

# 1. Introducción

En esta práctica hemos desarrollado una Aplicación de Recuperación de Información, utilizando la biblioteca Lucene en su versión *6.2.1* para realizar búsquedas sobre un conjunto de *papers* de carácter matemático. Estos *papers* han sido obtenidos de la página web *www.scopus.es* en formato CSV.

Hemos recopilado un total aproximado de 20000 documentos de diversas ramas de las matemáticas, los cuales han sido almacenados en un único archivo CSV usado para la indexación, en el cual cada línea representa un único documento.

Nuestro objetivo es crear una aplicación que sea capaz de realizar distintos tipos de búsquedas sobre determinados campos relevantes a los archivos que tenemos, mediante una interfaz gráfica sencilla, obteniendo los documentos que satisfacen las consultas y siendo posible acceder a los mismos en la web de *Scopus* mediante un enlace. Esta aplicación consta de dos partes: una encargada de indexar los documentos y otra encargada de realizar las búsquedas sobre el índice creado anteriormente.

Para la implementación, hemos utilizado el lenguaje de programación Java, con la librería de Lucene proporcionada, *Luke*. Además, hemos utilizado otros lenguajes de programación para hacer tareas básicas sobre el tratamiento de los archivos con los que hemos trabajado. Veremos esto con detalle en la parte de diseño de la solución.

Debido a las restricciones de tamaño que tiene la entrega de esta práctica, la versión que subiremos a *Decsai* será una versión reducida en el tamaño del índice. No obstante, en esta memoria se expondrán las operaciones realizadas con los 20000 documentos.

# 2. Análisis de Requisitos

Lo primero que debemos tener en cuenta son los campos que vamos a considerar para la indexación y la búsqueda. Dentro de lo ofrecido por *Scopus*, hemos tomado campos lo suficientemente diversos como para poder realizar distintos tipos de búsquedas y que la aplicación tenga una utilidad real, sin llegar a tomar tantos como para que el desarrollo de la aplicación llegase a ser repetitivo.

Después de ver las posibilidades que teníamos, hemos considerado los siguientes campos para indexar y realizar búsquedas:

- Autores
- Título
- Año
- Fuente (revista de publicación)
- Página de inicio (en la revista)
- Página de fin (en la revista)
- Enlace
- Abstract
- Palabras clave del autor
- Palabras clave para indexación
- Referencias
- Idioma
- Tipo de documento

Con todos estos campos, podemos cubrir los requerimientos de la práctica: como texto no tokenizado tenemos, por ejemplo, los autores, la fuente o los enlaces. Como texto tokenizado, el título y el *abstract*. Como búsqueda numérica, las páginas de inicio y fin, y como fecha, el año de publicación. Finalmente, como facetas hemos tomado el idioma y el tipo de documento, para lo que hemos tenido que duplicar el contenido de estos campos.

Al tratar los tipos de búsqueda que deberíamos incluir en el sistema, hemos optado por tres distintos: la búsqueda estándar, que nos permite hacer consultas por términos, booleanas o numéricas, la búsqueda por proximidad, en la que se especifica la distancia máxima entre distintos términos, y la búsqueda exacta, que es un caso particular de la anterior, con distancia cero. Que la búsqueda estándar se encargue de diferenciar entre booleana, por términos o numérica se realizará dentro del código del programa.

Nos planteamos también cómo mostrar los resultados de la búsqueda, llegando a la decisión de que lo más conveniente sería mostrar el título del documento y dar un enlace a *Scopus* para poder acceder a todos los datos. Mostrar muchos datos para cada documento sería engorroso y poco útil, además de no ser el objetivo de esta práctica, por lo que optamos por una solución escueta.

Quisimos añadir algún tipo de filtro a los resultados obtenidos, para lo que tenemos tres opciones posibles. Sobre los resultados de una búsqueda, podemos seleccionar por dos facetas distintas: el idioma y el tipo de artículo. Además, se incluye una búsqueda por rango para los valores numéricos, que son el año de publicación, la página de inicio y la página de fin. Finalmente, pensamos que el usuario debería tener un mecanismo para realizar una nueva búsqueda sobre los resultados de la primera, por lo que ofrecemos la posibilidad de que se realicen nuevas búsquedas sobre cualquier otro campo y estas se combinen con la primera.

# 3. Creación de la aplicación

## 3.1. Obtención y tratamiento de datos

Para obtener los datos de Scopus, hemos aprovechado las opciones que nos da la página para descargar datos relacionados con *papers*. Hemos descargado archivos CSV de distintas búsquedas, hasta un máximo de 2000 por búsqueda (límite fijado por la web), tomando siempre los mismos campos para obtener archivos que los compartan y poder combinarlos sin problema.

Una vez descargados todos los datos, para un mejor tratamiento de los mismos, hemos combinado los archivos CSV obtenidos en uno, de forma que tenemos todo lo necesario para la indexación en un único documento. Para esto, hemos usado un *script* en bash que une los archivos sin duplicar las cabeceras. Este script, de nombre *mergeCSV.sh* toma todos los archivos CSV del directorio en el que se encuentra y los combina en uno llamado *Data.csv*.

Después de unir todos los CSV, y previo al uso del archivo para la indexación, hemos de seleccionar las columnas del archivo que nos interesan, pues no todos los campos serán considerados para la aplicación. Este proceso se hace mediante un pequeño *script* en *Ruby*, que permite de forma sencilla leer el archivo, elegir las columnas que nos interesan y escribirlas en el archivo *Final.csv*, que será el que utilicemos en el proceso de indexación.

Cabe añadir que el orden puede ser invertido. No nos importa unir primero archivos y después eliminar las columnas que no queremos, o primero eliminar columnas en todos y después unirlo en un archivo final, siempre que unamos archivos con la misma cabecera CSV.

En la versión para entregar, hemos tomado un único archivo con 710 documentos, extraídos de *papers* de estadística, para que el tamaño no excediera el límite. Este archivo ha sido tratado para dejar solo las columnas que el programa necesita.

## 3.2. Implementación

A la hora de implementar nuestro programa, hemos usado el IDE NetBeans, con el que ya éramos familiares y que además nos ofrece muchas ventajas, sobre todo a la hora de crear la interfaz gráfica. Vemos ahora en detalle la implementación de las dos partes de esta aplicación.

### 3.2.1. Indexación

Para esta parte, hemos creado un módulo *MathIndexer*, con un único archivo de código, encargado de hacer todas las operaciones necesarias para crear el índice a partir de un archivo de entrada. Se encuentran contenidas en un método, *readAndIndex*, que crea un documento Lucene y va añadiendo todos los datos que lee del archivo CSV hasta que devuelve el índice obtenido.

Este método lee el archivo CSV que recibe como parámetro línea a línea, y obtiene los distintos campos de cada línea en forma de array, separando mediante una expresión regular que distingue distintos campos en un archivo CSV. Esto nos permite añadir cada campo secuencialmente a un documento Lucene y después añadir este documento al índice, una vez hemos insertado todos los valores. Este proceso continua mientras haya líneas en el archivo que se lee.

Es importante destacar el orden de las columnas está fijado en el código; sabemos que primero van los autores, después el título, y asi sucesivamente, por lo que un archivo con distinto número de columnas o un orden distinto no nos valdría. Se ve aquí la importancia de descargar los datos de *Scopus* con unos campos específicos y tratarlos todos con el *script* en *Ruby*, para que queden todos de la misma forma.

Algunos de los campos han sido tratados para eliminar el exceso de caracteres o evitar valores vacíos, haciendo las sustituciones pertinentes. Asimismo, aquellos campos que también son facetas has sido añadidos dos veces, una en forma de campo y otra en forma de faceta, con el método *FacetField*.

Hemos tenido problemas con algunas líneas de los archivos, obteniendo un número de campos distintoe nos interesa tratar es la primera, pues en la segunda no se requiere de nada por parte del usuario. al que dbería salir. Supongemos que esto es problema de la separación de las columnas del CSV, y hemos optado por no tener en cuenta estos documentos. No obstante, en el conjunto final de 20000 documentos, hemos omitido alrededor 15, por lo que es poco significativo.

#### 3.2.2. Búsqueda

En este caso hemos creado dos módulos: *mathsearcher*, que contiene el código para las búsquedas en Lucene, y *mathInterface*, que contiene la interfaz gráfica, con todos los métodos relacionados.

En el módulo *mathsearcher*, tenemos un único archivo de código con los métodos necesarios para construir las distintas *querys*, ya sean booleanas, exactas o por proximidad. Además, tenemos el método principal que realiza búsquedas, que será llamado desde el otro módulo. En él, distinguimos el tipo de búsqueda que queremos realizar y seleccionamos el etodo adecuado para esta, construyendo la consulta correspondiente.

En el módulo *mathInterface* creamos una interfaz gráfica que contiene un objeto de la clase *MathSearcher*, encargado de hacer las búsquedas, y los distintos elementos de nuestro programa: la barra de búsquedas, los filtros, los selectores, un panel de información de la búsqueda y el cuadro de resultados. Desde aquí llamaremos al método de búsqueda del módulo anterior aplicando los filtros necesarios, y mostraremos los resultados al usuario.

# 4. Manual de usuario

## 4.1. Indexación

Para crear el índice, hemos de ejecutar el archivo principal del módulo *MathIndexer*, teniendo siempre en cuenta que debe ser aplicado sobre el archivo que queramos indexar, que será *"/Data/Estadistica.csv"* (pasado como primer argumento) en el caso de la entrega. Para hacerlo sobre el conjunto completo de documentos, lo haremos sobre *"/Data/Final.csv"*, que es el que contiene todos los datos que hemos descargado y modificado. Esto basta para crear el índice que después se usará en el buscador.

Este proceso puede tomar bastante tiempo en el conjunto completo de datos, ya que el volumen de datos con el que trabajamos es grande. Los tiempos experimentales que hemos obtenido han rondado los 20 minutos. Para archivos pequeños el tiempo de ejecución es pequeño, alrededor de un minuto, dependiendo del número de documentos del archivo.

## 4.2. GUI

Para lanzar el buscador, debemos ejecutar el archivo *mainFrame.java* del módulo *mathInterface*.

En la interfaz gráfica encontramos dos partes: a la izquierda tenemos el menú de búsqueda, con la barra de entrada, los campos y las opciones para la búsqueda, mientras que a la derecha tenemos los resultados. Vemos a continuación una imagen de la interfaz:

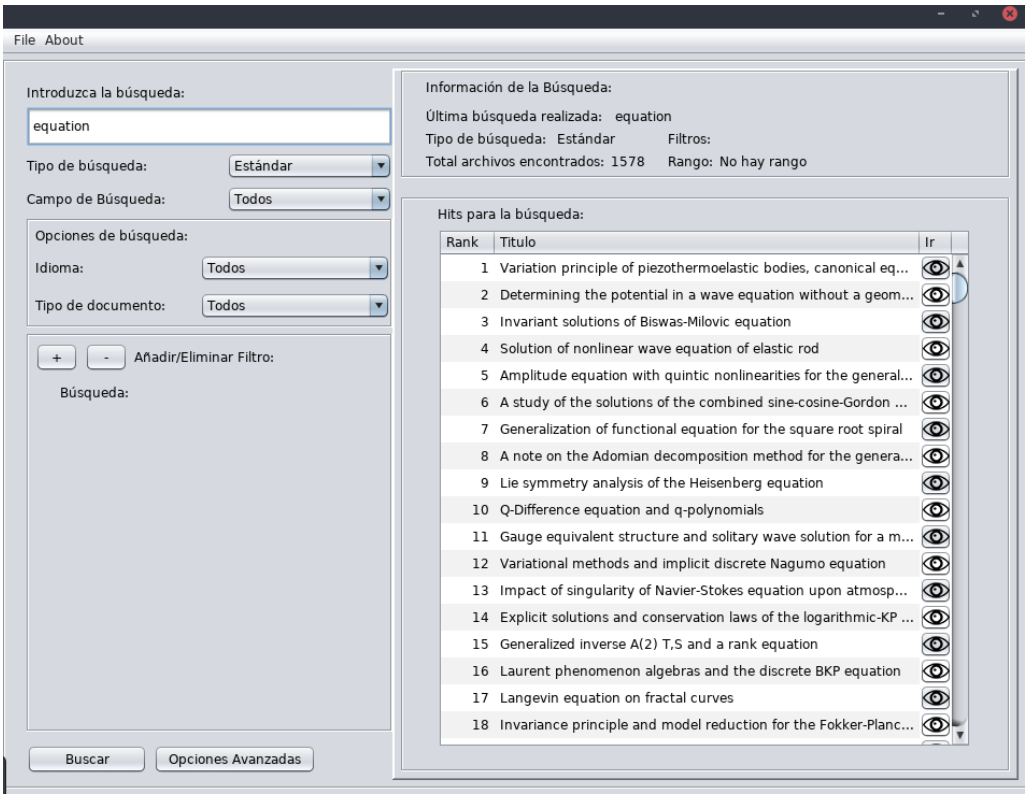


Figura 1: Imagen del programa

La primero que encontramos es una barra de entrada, donde escribiremos nuestra consulta. En los menús inferiores podemos elegir el campo sobre el que estamos buscando y el tipo de búsqueda que queremos realizar, que desplegará un menú adicional si seleccionamos la búsqueda por proximidad, el cual nos permitirá especificar la distancia.

Una vez realizada la búsqueda, disponemos de tres menús para filtrar los resultados. El primero que encontramos es el que nos permite seleccionar por idioma o por tipo de documento los *papers* obtenidos. El segundo lo encontramos al pulsar el botón inferior de “Opciones Avanzadas”, que muestra un nuevo cuadro que nos permite filtrar un rango numérico, que puede ser el año, la página de inicio o la página de fin del documento. El tercer cuadro nos permite realizar hasta cuatro búsquedas más sobre los resultados ya obtenidos, sobre el campo que queramos.

En el cuadro de resultados, es posible ordenarlos por orden alfabético si pulsamos en la columna título, alternando entre orden y orden inverso. Para volver al orden por relevancia, haremos lo mismo en la columna del ránking.