UNIVERSIDAD
DE GRANADA

Gestión de Información en la Web

Desarrollo de un Sistema de Recuperación de Información con la biblioteca *Lucene*

Práctica 3

Víctor Vázquez Rodríguez
victorvazrod@correo.ugr.es
76664636R

Máster Universitario en Ingeniería Informática

Curso 2019/20

Índice

1. Introducción

El objetivo de esta práctica era construir un indexador y un buscador de documentos usando la librería **Apache Lucene**. Como colección de documentos, he decidido usar las guías docentes de todas las asignaturas tanto del grado como del máster en Ingeniería Informática de la ETSIIT (en total, 112 documentos), con la idea de poder buscar en estas guías por su contenido. Para la implementación, se ha usado Java. Además, se han usado otras librerías para crear las aplicaciones, como es **picocli** para la aplicación de línea de comandos del indexador y **JavaFX** para la interfaz gráfica del buscador.

Al no poder extraer campos concretos de los documentos PDF de las guías docentes, indexamos nuestra colección por su contenido completo.

2. Indexador

Como su propio nombre indica, el indexador es la aplicación que construye el índice a partir de una colección de documentos dada. Se trata de una aplicación de línea de comandos que, como ya hemos indicado, hemos construido con **picocli**, una librería orientada a este tipo de aplicaciones. Esta librería nos permite definir *flags* y otros parámetros de entrada para los comandos, facilitando su manejo. A continuación, se muestra la definición del comando creado para la indexación.

```java
@Command(name = "index", description = "Create or update an index from a
    set of documents.", mixinStandardHelpOptions = true, version =
    "index 1.0")
public class Index implements Callable<Integer> {

    @Parameters(paramLabel = "DIR", description = "Documents' directory
        to be indexed.", arity = "1..1")
    private File docsDir;

    @Option(names = { "-i", "--index" }, paramLabel = "INDEX",
        description = "Path to index. Defaults to /tmp/index.")
    private File indexFile = new File("/tmp/index");

    @Option(names = { "-s", "--stopwords" }, paramLabel = "STOPWORDS",
        description = "Stopwords file.", required = true)
    private File stopwordsFile;

    @Option(names = {"-u", "--update"}, description = "Update index
        instead of creating a new one.")
    private boolean update;

    public static void main(String... args) {
        int exitCode = new CommandLine(new Index()).execute(args);
        System.exit(exitCode);
    }

    @Override
    public Integer call() throws Exception {
        // Index documents
    }
}
```

Como se puede ver, nuestra aplicación requiere, por lo menos, de dos entradas: las rutas al fichero de palabras vacías (*stopwords*) y a la colección de documentos. También se puede indicar una ruta para almacenar el índice aunque, si no se especifica, se usa `/tmp/index` por defecto. Destacamos también el *flag* `-u`, el cuál permite indicar que se está actualizando un índice ya existente, con lo que no tenemos que realizar todo el proceso de nuevo si no es necesario.

En cuanto a la lógica que se encarga de realizar la indexación, es la siguiente:

```
try {
    // Read stopwords from file
    var stopwords = readStopwords(stopwordsPath);

    // Set up index writer
    Directory dir = FSDirectory.open(indexPath);
    Analyzer analyzer = new SpanishAnalyzer(new CharArraySet(stopwords,
        true));
    IndexWriterConfig config = new IndexWriterConfig(analyzer);

    // Update index or create a new one
    if (update) {
        config.setOpenMode(OpenMode.CREATE_OR_APPEND);
    } else {
        config.setOpenMode(OpenMode.CREATE);
    }

    // Create index writer and process documents
    IndexWriter writer = new IndexWriter(dir, config);
    indexDocuments(writer, docsPath);

    // Close index writer
    writer.close();
} catch (Exception e) {
    System.out.println("Can't create index - Error: "+ e.getMessage());
    return 1;
}
```

El método `indexDocuments` es el encargado de recorrer todos los archivos de la colección dada y llamar a `indexDocument` para cada uno de ellos. Este último, extrae el texto del archivo PDF y añade el nuevo documento al índice, como se puede ver a continuación:

```
public void indexDocument(IndexWriter writer, Path file) throws
    IOException {
    try (PDDocument pdf = PDDocument.load(file.toFile())) {
        // Extract text from PDF
        PDFTextStripper stripper = new PDFTextStripper();
        stripper.setLineSeparator("\n");
        stripper.setStartPage(1);
        String contents = stripper.getText(pdf);
        pdf.close();

        // Create Lucene document
        Document doc = new Document();

        // Add the path of the file
```

```
        doc.add(new StringField("path", file.toString(),
            Field.Store.YES));

        // Add the contents of the file
        doc.add(new TextField("contents", contents, Field.Store.NO));

        // Add the document to the index
        writer.addDocument(doc);
    }
}
```

Como se puede apreciar, indexamos tanto el contenido del documento como la ruta hacia el mismo.

2.1. Manual de uso

Usando el fichero `jar` proporcionado, se puede ejecutar la aplicación de la siguiente manera:

```
java -jar indexer.jar --stopwords=[Ruta a fichero] --index=[Ruta a
    indice] [Ruta a coleccion]
```

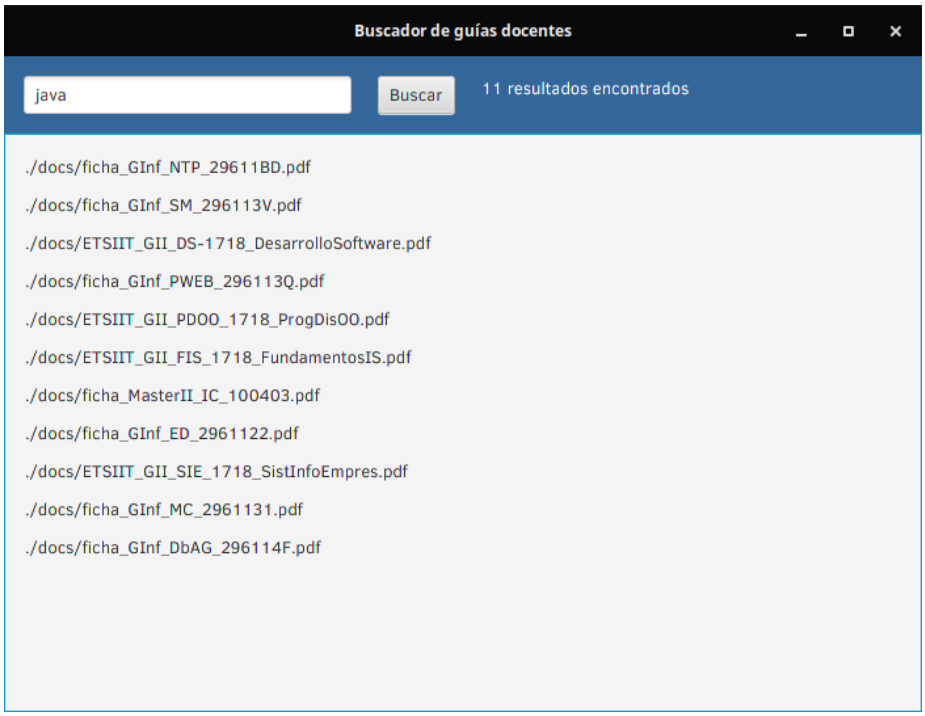
Alternativamente, si se va a actualizar un índice ya creado, se puede usar:

```
java -jar indexer.jar -u --stopwords=[Ruta a fichero] --index=[Ruta
    a indice] [Ruta a coleccion]
```

Nota: Como ya se ha indicado anteriormente, no es completamente necesario indicar una ruta para el índice, usándose `/tmp/index` por defecto.

3. Buscador

A diferencia del indexador, la aplicación del buscador sí que dispone de interfaz gráfica, permitiendo al usuario realizar múltiples consultas sobre el índice de manera sencilla en una misma sesión. Para construir esta interfaz, se ha usado **JavaFX**, obteniendo el resultado siguiente:



Para realizar las búsquedas, se necesitan tanto un `QueryParser` para interpretar las consultas (con el mismo procedimiento usado para procesar los documentos) como un `IndexSearcher` para realizar las búsquedas como tal sobre el índice. Estos objetos se construyen así:

```
private IndexSearcher createIndexSearcher(Path indexPath) {
    IndexSearcher searcher = null;

    try {
        Directory indexDir = FSDirectory.open(indexPath);
        IndexReader reader = DirectoryReader.open(indexDir);
        searcher = new IndexSearcher(reader);
    } catch (IOException e) {
        System.out.println("Can't create index searcher - Error: " +
            e.getMessage());
        System.exit(1);
    }
```

```
    return searcher;
}

private QueryParser createQueryParser(Path stopwordsPath) {
    QueryParser parser = null;

    try {
        // Read stopwords from file
        var stopwords = readStopwords(stopwordsPath);

        // Create parser
        Analyzer analyzer = new SpanishAnalyzer(new
            CharArraySet(stopwords, true));
        parser = new QueryParser("contents", analyzer);
    } catch (IOException e) {
        System.out.println("Can't create query parser - Error: " +
            e.getMessage());
        System.exit(1);
    }

    return parser;
}
```

Una vez hecho esto, creamos la interfaz, dotando al botón de la funcionalidad de búsqueda, la cuál se puede ver a continuación:

```
// Search button
Button searchBtn = new Button("Buscar");
searchBtn.setOnAction(new EventHandler<ActionEvent>() {

    @Override
    public void handle(ActionEvent e) {
        msgLabel.setText("");

        String inputString = queryTF.getText();
        if (inputString.isEmpty()) {
            msgLabel.setText("Por favor, introduce un termino para la
                busqueda");
            return;
        }

        try {
            Query query = parser.parse(inputString);
            var results = indexSearcher.search(query, 50).scoreDocs;
            resultsPane.refresh(results);
            msgLabel.setText(results.length +" resultados encontrados");
        } catch (ParseException pe) {
            msgLabel.setText("No se ha podido procesar la consulta");
```

```
        } catch (IOException ioe) {
            msgLabel.setText("Error en la consulta del indice");
        }
    }
});
```

Como su propio nombre indica, `resultsPane` es el panel de la interfaz en el que se muestran los resultados. Llamando a su método `refresh` con los nuevos resultados, lo que hacemos es refrescar el panel para mostrarlos. La construcción de los resultados en la interfaz se puede ver a continuación:

```
public void refresh(ScoreDoc[] results) {
    this.getChildren().clear();

    for (int i = 0; i < results.length; i++) {
        try {
            Document doc = indexSearcher.doc(results[i].doc);
            this.getChildren().add(buildResult(doc));
        } catch (IOException e) {
            System.out.println("Can't retrieve result - Error: " +
                e.getMessage());
        }
    }
}

private HBox buildResult(Document doc) {
    HBox box = new HBox();
    box.setSpacing(15.0);

    // Add label with doc path
    Label pathLabel = new Label(doc.get("path"));
    box.getChildren().add(pathLabel);

    return box;
}
```

3.1. Manual de uso

Para lanzar la aplicación del buscador, debemos ejecutar el archivo `jar` desde la línea de comandos, indicando las rutas al índice y al archivo de palabras vacías, de esta manera:

```
java -jar searcher.jar --stopwords=[Ruta a fichero] --index=[Ruta a
    indice]
```

Una vez en la interfaz, para realizar una búsqueda sólo debe introducir los términos de la misma en el campo de texto y pulsar sobre el botón "Buscar".

4. Referencias

Para la realización de esta práctica se han usado algunas fuentes complementarias al guión proporcionado, las cuáles se indican a continuación:

- Documentación de Apache Lucene 8.4.1, incluye ejemplos de uso que se han observado para la implementación.
- Tutorial de Oracle sobre JavaFX, usado como introducción al uso de las librerías.
- Documentación de picocli, usada para la construcción de la aplicación de línea de comandos del indexador.
- Consulta de StackOverflow sobre indexación de PDFs con Lucene.