Lab 6 – Apache Lucene

Deadline: +1 week

1 Exercises

Your task is to create a simple Apache Lucene application. This exercise is divided into two parts:

- Firstly, you will use Apache Lucene to index a collection of HTML documents.
- Then, you will make several queries and search the collection (index) for the most relevant documents.

Before going further:

1. Download Indexer.java, Searcher.java, Constants.java, and pages.zip from http://www.cs.put.poznan.pl/mtomczyk/ir/lab6/. Create a java project using any IDE and add these files to the project (unzip pages.zip!).
2. Download lucene-7.2.1.zip (www.cs.put.poznan.pl/mtomczyk/ir/lab6/) and unzip it to the project's directory. Add the following jars to your project: lucene-core-7.2.1.jar, lucene-queryparser-7.2.1.jar and lucene-backward-codecs-7.2.1.jar.
3. Download tika-app-1.17.jar (http://www.cs.put.poznan.pl/mtomczyk/ir/lab2/) and add it to your project.

1.1 Exercise 1

Firstly, use Apache Lucene to index the collection of HTML files:

1. Go to **indexer.java**. It generates a Lucene index.
2. Seek for a **indexDocuments** method. There are several **TODOs** that must be completed.
3. Firstly, take a look at a **getHTMLDocuments** method. It loads the files from the collection, iterates over the files stored in "pages" folder, and constructs Document (Apache Lucene class) objects. This method is complete. Go to a **getHTML-Document** method, which processes a single file.

4. Read the comments carefully. Focus on the Document $\leftrightarrow$ Field relation. What is the Field? What do STORED and INDEXED flags mean?
5. Finish this method (you may notice that Apache Tika is used for content extraction).
6. Go back to the **indexDocuments** method. Now, having a list of Documents, you should create an index using **IndexWriter**. Follow the comments.
7. If you had completed all **TODOs**, run Indexer.java. You should see that the index was generated and stored in "index" folder.

1.2 Exercise 2

Let's search for some documents.

1. Go to **Searcher.java**. It is suggested to start with a **printResultsForQuery** method.
2. This method is invoked after a Query object is constructed. As you may notice, an **IndexSearcher** object is passed as an argument. This object provides a method for searching for the first N documents which are the most relevant to the query. The method should notify (log) about the found documents. The log, for each document, should be in the following form:
   SCORE: FILENAME (Id=...ID) (Content=CONTENT) (Size=SIZE)
3. Now, go to the **main** method. Follow the comments and construct several queries:
   (a) **TermQuery**: for seeking for documents that contain some term.
   (b) **BooleanQuery**: boolean query, e.g., "medicine AND drug".
   (c) **RangeQuery**: for seeking for documents whose attribute values are between a range of some numbers, e.g., size of the document $\in [50B, 2000B]$.
   (d) **PrefixQuery**: like TermQuery, but the prefix is provided instead of a whole string (i.e., "antelope" and "ant" match the "ant" prefix).
   (e) **WildcardQuery**: you can use ? to indicate any single letter (e.g., "cats" matches "cat?") or use $*$ to indicate multiple letters (e.g., "antelope" matches "ant*").
   (f) **FuzzyQuery**: for seeking for terms that are similar to the provided term, e.g., "mammal" matches "mamml".
   (g) Use **QueryParser** to parse human-entered query strings.