Programming Assignment 1: Apache Lucene Programming

Due: Oct 15th at 11:59pm

This assignment is to be done individually

Important Note:

The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the TAs if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use Google to find solutions for assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment

Getting Started

This section will introduce you to working with the Lucene library. We will help you to walk through a common codebase we have built in order to help you get familiar with Lucene library as much as possible.

Codebase

- The codebase is already in our GitLab at https://csil-git1.cs.surrey.sfu.ca/nguyenc/lucene-solr.
- You should be able to clone it to your own workspace in order to do the programming tasks described in the next sections.
- For this assignment, we use the latest version of Lucene, branch 6.6 (6.6.7), with Java 8. You can check its detail API here: https://lucene.apache.org/core/6_6_6/index.html
- The codebase is a fork from Lucene/Solr open source code (https://github.com/apache/lucene-solr) with some customizations in order to allow you to run it inside a Docker environment (https://www.docker.com/). So you need to have Docker installed on your machine. The CSIL computers have been equiped with Docker already, so feel free to use them.
- The purpose of having Lucene/Solr running inside a Docker container is to help you work on this assignment using mostly any OS you prefer, Linux, Mac or Windows. If you are curious about how the Docker container is built, look at the **Dockerfile** in the source code.

Project Data

- We are going to use **Wiki Small** data (6043 documents) from our textbook http://www.search-engines-book.com/. Take a look at the tar.zip file to know how it look like.
- We have included the data for you, within the codebase at location **lucene/demo/data**. In the subsequent sections, you will use it in to demonstrate indexing and querying process.

Compiling

- Checkout the codebase to local machine with git command:
  git clone https://csil-git1.cs.surrey.sfu.ca/nguyenc/lucene-solr
  cd lucene-solr

- Build Docker image from the source code (make sure that we have **.** (i.e current location) at the end of the command):
  docker build -t cmpt456-lucene-solr:6.6.7 .

- Run the Docker image we just built in order to activate the Docker container:
  docker run -it cmpt456-lucene-solr:6.6.7

Demo

In this section, we help you to get familiar with Lucene basic components by running 2 simple programs:

- **Index Files**: this program uses standard analyzers to create tokens from input text files, convert them to lowercase then filer out predefined list of stop-words.

  The source code is stored in this file within the codebase:
  lucene/demo/src/java/org/apache/lucene/demo/IndexFiles.java

  Index demo data with the following command inside the Docker container:
  **ant -f lucene/demo/build.xml **
  -Ddocs=lucene/demo/data/wiki-small/en/articles/ run-indexing-demo

- **Search Files**: this program uses a query parser to parse the input query text, then pass to the index searcher to look for matching results.

  The source code is stored in this file within the codebase:
  lucene/demo/src/java/org/apache/lucene/demo/SearchFiles.java

  Search demo data with the following command inside the Docker container:
  ant -f lucene/demo/build.xml run-search-index-demo

You are expected to run these examples, understand Lucene components used in the indexing and querying process in order to make further extensions in the below programming tasks.

Text Parsing (30 pts)

In the first part of the assignment, you will learn how to use Lucene to build search capabilities for documents in various formats, such as HTML, XML, PDF, Word. In fact, Lucene does not care about the parsing of these and other document formats, and it is the responsibility of the application using Lucene to use an appropriate **parser** to convert the original format into plain text before passing that plain text to Lucene.

In the class **IndexFiles.java** within the Demo section, you can see that it indexes the content of html files, including all html tags (<body>, <head>, <table>, etc.) which is nonsense. In this section, we want you to create a new class called **HtmlIndexFiles.java** to:

- Use a HTML parser to parse input files to extract **the title** and **text content only** of the HTML files. Text content should not contain any HTML tags.
- Use standard analyzers to create tokens from the result of parser, convert them to lowercase then filter out based on a predefined list of stop-words (similar to the way **IndexFiles.java** works)

Hint: there is an already implemented HTML parser in this class

`org.apache.lucene.benchmark.byTask.feeds.DemoHTMLParser`

Tokenization (30 pts)

In the second part of the assignment, you will experience how plain text passed to Lucene for indexing goes through a process generally called tokenization. Tokenization is the process of breaking input text into small indexing elements – tokens. The way input text is broken into tokens heavily influences how people will then be able to search for that text.

As you have seen in the **IndexFiles.java**, we have used class **StandardAnalyzer** in order to control the tokenization process. Look at its source code, you can see this class extends the **createComponents** method to build a standard tokenization process to convert tokens to lowercase then filer out based on a predefined list of stop-words.

In this section, we want you to create a class called **CMPT456Analyzer.java** to control the tokenization process as follows:

- Create a **stopwords.txt** to keep all our custom stop words. You can use the stopwords list from the textbook: http://www.search-engines-book.com/data/stopwords
- Convert tokens to lowercase then filter out based on our custom stopwords list
- Use a Porter stemmer for stemming

Hint: Porter stemmer is already implemented in Lucene. Make use of it.

Similarity Metrics (40 pts)

In the last part of the assignment, you will have chance to touch one of the core modules of querying process which is the ranking module. When user issues a query, Lucene will use index created during the indexing process to look for matching documents. More importantly, these matching documents will be sorted by a **customizable ranking function** before returning the final results to the user.

Before asking you to implement a ranking function, we want you to make use of Lucene to compute some basic metrics:

- **Document Frequency**: Returns the number of documents containing the term.
- **Term Frequency**: Returns the total number of occurrences of the term across all documents (the sum of the freq() for each doc that has this term).

You need to create a **SimpleMetrics.java** program to demonstrate how you can find the 2 above metric score for a given term. Hint: Make use of IndexReader (http://lucene.apache.org/core/6_6_6/core/org/apache/lucene/index/IndexReader.html#totalTermFreq%28org.apache.lucene.index.Term%29)

Next, we want you to implement a custom ranking/similarity function base on TFIDFSimilarity (https://lucene.apache.org/core/6_6_6/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html) provided by Lucene. In particular, you need to create a class called **CMPT456Similarity.java** to support custom **tf()** and **idf()** as follows:

$$tf(t\ in\ d) \quad = \quad (1 + frequency)^{1/2}$$

$$idf(t) \quad = \quad 1 \ + \ log\ (\frac{docCount+2}{docFreq+2})$$

Hint: Extend class **ClassicSimilarity** (https://lucene.apache.org/core/6_6_6/core/org/apache/lucene/search/similarities/ClassicSimilarity.html) instead of directly implementing TFIDFSimilarity

The next thing we want you to do is to alter the way Lucene scoring. You will need to create **TFIDFHtmlIndexFiles.java** and **TFIDFSearchFiles.java** in which you want to use **CMPT456Similarity** for your indexing & querying process.

Hint: take a look at this to learn how to change the similarity scoring: https://lucene.apache.org/core/6_6_6/core/org/apache/lucene/search/similarities/package-summary.html#changingSimilarity

Submit Your Assignment

The assignment must be submitted online at https://coursys.sfu.ca. You need to submit the following files:

1. HtmlIndexFiles.java
2. CMPT456Analyzer.java
3. SimpleMetrics.java
4. CMPT456Similarity.java
5. TFIDFHtmlIndexFiles.java
6. TFIDFSearchFiles.java