Programming Task P01: Information Retrieval System

Mohammed Aftab, Evelina Dukaj, Ronald Mendez,
Mohammed Nadeem, Chigozie Kenneth Okafor

November 29, 2019

1 Introduction

This program is an implementation of lucene based information retrieval system. Firstly all the documents, has to be filtered as HTML or text files, go through the process of preprocessing, that includes tokenization and stemming. Based on the stemmed documents, indexing is done. After the creation of index, search is performed.

Our program accepts four arguments as below:

```
java -jar IR_P01.jar [path to document folder] [path to index folder] [VS/OK]
[user query]
```

The output contains a list with the relevant documents that contain the given query, and for each of them the rank, the path, the title (If it is a HTML File), last modified date and time, and the document score.

Note: Only Top 10 most relevant document will be shown on a page. For the rest user needs to enter a page number to move display the next relevant documents.

The program that we have written contains the following classes:
ParseDocs(FilterDocs, ParseHTMLDocs, ParseTextDocs), Tokenizer, TextStemmer, Index and SearchDocs
explained below in details under each section.

2 ParseDocs.java

2.1 Inputs

Path to document folder.

2.2 Code Flow

The goal of ParseDoc class is to extract some relevant information from each document that is in the collection, this information includes: document-folder path, date of its last modification and for HTML documents, in addition extracts its title. To this end, the class uses a couple of methods that reference to task specific classes:

with in **the parseDocs** Class, a listAllFiles method has been created in order to make a copy of all the documents into a new folder (/stemmed), and then uses this same folder to send the copied documents to the following method.

NOTE: This has only been done for educational purposes. The idea behind is to compare the documents before and after they have been pre-processed, and evaluate the process. It can also be done directly without creating this new folder(Stemmed).

The listAllFiles method, classifies the documents depending on the extension (.txt/.html), and then calls either of the classes: ParseTextDocs; where the folder path, date of last modification, and content of document are linked as attributes of an instance, while in the parseHTMLDocs class; Accepts same attributes as the ParseTexDocs Class plus the document title, and also uses the jsoup library to extract text and title contents from the HTML file. It is important to mention that the path stored here points to the stemmed folder, but it will be updated to the original folder during the Indexing step.

2.3 Outputs

Info-list of documents.

3 Tokenizer.java

3.1 Inputs

List of strings(Text lines)

3.2 Imported elements

The process of Tokenization consists of converting List of lines into a stream of words. For this process it is important to distinguish what a word is. Punctuation, case of letters, hyphenation and white space are some points that may cause problems in tokenization. For this, here we are using TokenStream. The TokenStream uses the analyzer which is a public class of Lucene that creates tokens, by making an analysis(pre-processing) to the components of the list using english module.

3.3 Code Flow

After the process of parsing documents by distinguishing them as text or HTML documents, tokenization is done.

First a stopWords object type CharArraySet is created to import the list of English-language stop words from the class EnglishAnalyzer, then within a loop, the stream object type TokenStream is created to apply the tokenStream analyzer on the list of characters, that will be returned as tokens, after that the tokens will be passed along with the list of stop words in the StopFilter in order to remove the predefined stop words from the tokens list, and then will be stored in the result list.

3.4 Outputs

List of tokens

4 TextStemmer.java

4.1 Inputs

List of tokens

4.2 Imported elements

In this section, the additional lucene default classes that where required in order to use some specific functions are defined. Among them:

The Analyzer (org.apache.lucene.analysis.Analyzer)[1], which is a class defined in the core of Lucence library, that contains a wide variety of parsing specialized methods that can be used during the text analysis. This class also has several subclasses that inherit some of its methods, among them; WhitespaceAnalyzer, SimpleAnalyzer, StopAnalyzer and StandardAnalyzer.

The StandardAnalyzer (org.apache.lucene.analysis.standard .StandardAnalyzer), this class, is one of the most complete analyzers of its kind, among its functions it has; handling different types of "words" like names or e-mail addresses, lower casing, punctuation elimination, and duplicates removal.

The PorterStemmer (org.tartarus.snowball.ext.PorterStemmer)[2], which is a class generated by a Snowball to Java compiler, to implement an English-language-Stemming algorithm as the Porter Stemmer is, which will fulfill the task of returning the root of the analyzed word.

4.3 Code Flow

The getStemmedValue method's goal is to return a list of stemmed words, using the list of tokens which comes from Tokenizer class.

To this end, it was necessary to create several objects: the object (tk) of the class Tokenizer, that will be used to import the list of tokens to the TextStemmer class. An analyzer object (az) type StandardAnalyzer, which can be used to define the analyzer to be applied. The stemming object (stemmer) to apply the Porter stemming algorithm. A new list (resultList) to storage the analyzer output was required, and a second list (ls) where the output of Porter algorithm will be stored.

The implementation is developed as follows; first the StandardAnalyzer is applied to the termList that comes from the Tokenizer class, here the lowercasing, duplicates and punctuation marks removal take place.

Finally a loop is used to go through the resultList, term by term, applying over them Porter stemming algorithm and storing the resulting terms in a final list (ls) which is stored in the stemmed directory(done in respective parsing classes of corresponding document types), which is the output of this class.

4.4 Outputs

List of Stemmed terms (ls) & stemmed directory .

5 Index.java

5.1 Inputs

List of stemmed files(docList)

5.2 Code Flow

The code implemented for indexing in our task is exactly the same as Indexing class of Lucene demo in association with a small subsection having the HTML files processed.

docList,(in Main Class)now contains the objects which have actual path of the file, Last modified date and Title of the file (if only the file is HTML).

Indexing beings with having the writer created, **createWriter** is an inbuilt method of Lucene, in which the directory(path) where the indexed files need to be specified is passed. After having the writer created we have just gotten the path of the stem folder fetched.

Files.isDirectory method, verifies if the path mentioned is a directory or not and **walkFileTree**(inbuilt java method) traverses through each file in the directory path specified, post which using **FileVisitResult visitFile** the Title, Content, Last Modified, actual path are obtained, just to finally use **doc.add** to create the index, which also truggers the creation of a file and folder where the index information is stored.

5.3 Outputs

Index file saved in the **Index** directory.

6 SearchDocs.java

6.1 Inputs

Indexed document Path and query to search

6.2 Flow of code

The code implemented for searching in our task is very similar to the SearchFiles part of Lucene demo with minor modification for search of multiple fields and Displaying Output.

Since the documents are indexed and a Index(user defined) directory is created. This directory path is now passed to the searchDocs.search() from Main.java.

searchDocs.search method will instantiate the **MultiFieldQueryParser (accept all the fields on which index is performed),IndexSearcher,Analyzer**. Then a scanner will accept the queries from the console perform the preprocessing step performed previously while indexing. These instantiated objects are passed as parameters to doPagingSearch method. **searchDocs.doPagingSearch** The query and hitsPerPage is passed to IndexSearcher which retrieves the Top relevant result from the lucene index.

Looping is done on the result to display the required information to the user and also continuous search option is give so that the user can search other queries.

References

[1] Lucence. "Analizer JAVA Class". In: (). URL: `https://lucene.apache.org/core/6_4_0/core/org/apache/lucene/analysis/Analyzer.html#tokenStream-java.lang.String-java.io.Reader-`.

[2] Lucence. "Porter Stemmer". In: (). URL: `https://github.com/weavejester/snowball-stemmer/blob/master/src/java/org/tartarus/snowball/ext/englishStemmer.java/`.