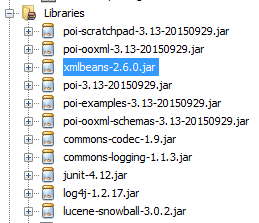
Ryan Kane 12/3/19

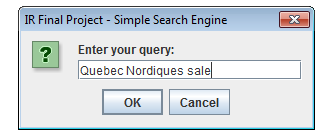
Final Project Documentation

README:

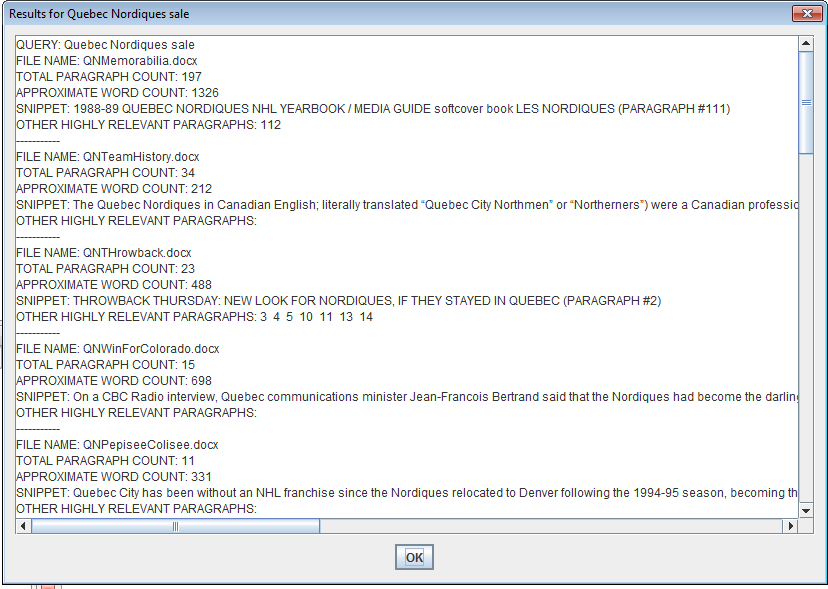
1. Extract all files and directories from the zip file into a new folder (this was already done on the flash drive submission but not the e-mail submission).
2. Create a new blank java project and import the two JAVA files found in the SRC folder to the default package generated by the project (InvertedIndex.JAVA and SearchEngine.JAVA).
3. Navigate to the “Stemmer” folder in the extracted directory and add the lucene-snowball-3.0.2 JAR file into the new project’s library. This JAR file is required in order for Porter’s Stemming Algorithm to work in the program.
4. Navigate to the “poi-3.13” folder. This contains all of the files of the Apache POI 3.13 library, which is required to read the docx files that comprise the corpus. Move all available JAR files in the “poi-3.13” directory, the lone JAR file in the “ooxml-lib” directory, and all JAR files in the “lib” directory into the project library. There should be ten JAR files from the Apache library added to the project. The project library should now look as follows:



1. Save the project. The program should now be able to be ran from the command line by compiling both source files and running SearchEngine.JAVA. The main function expects one command line input argument, which is the absolute path to the directory labeled “Corpus” from the extracted files (i.e. project was tested using “H:\Information Retrieval and Org\Project\Corpus”). It is important that the directory structured from the provided zip remains the same and that the path of this directory is specifically passed as the first command line argument (args[0]) in order for the program to function correctly.
2. Upon running, the Inverted Index will be constructed from the contents of the Project folder. A few seconds after, a new window will appear prompting the user for input. Type in a query and click “OK”.



1. Upon clicking “OK”, a new window displaying the results to the query will appear. The precision and recall for a particular query can be found by scrolling down to the bottom of the window.



1. Clicking “OK” or exiting out of the window will open the query window seen in step 6, where another query can be entered and its results can be viewed. You will be redirected to the results window from step 7 with the results for this new query.
2. When you are finished entering all necessary queries, click “Cancel” or simply exit out of the query window on the screen. The result window will pop up one last time, showing a list of all of the results from the queries that were entered during the session. The results for the entire session is also saved in a text file “results.txt”, which is written to the same directory containing the Java project.

Notes:

1. Many key components of the program are reliant on the directories and file locations remaining the same as they were assembled in the zip. For instance, locating the file to extract stop words and classifying corpus documents to the query they belong are driven by their directory location.
2. The Apache POI library for reading docx files is extremely sensitive to spacing. The library provides no means for extracting line by line, and only by paragraph. Most of the data extracted for counting term frequencies and determining which snippet appears in the results is based off of a word count relative to the paragraph. Each document has been checked to make sure that no blank paragraphs exist, as this alters paragraph counts, making it difficult to locate snippets. Please be careful when viewing corpus documents as small changes can throw off snippet results.
3. The snippet displayed in the results is meant to a “best guess” of the paragraph most relevant to the key terms provided in the user queries. For any given document in the results, the paragraph returned should be the one that contains the most instances of any key term from the query. The list that appears beneath the snippet labeled “other highly relevant paragraphs,” is simply the the paragraphs that had the same number of key terms as the one selected for the snippet.
4. Any major decisions made in the code, such as how to balance the weight of term and document frequencies, data structure choices, or even how many results to include for each query, have their reasons explained in the comments where they are implemented.
5. The results are returned in order of most to least relevant, based on their generated score. The number of results returns varies by query, but is always twenty or greater. After the twentieth result is found, all subsequent results are measured to see how distant their scores are from that twentieth score in order to prevent highly similar documents from being excluded.
6. Every query and corpus that was tested/retrieved contains content regarding sports (namely hockey and football). This caused queries relating to general topics about a sport to often have less than optimal precision and recall, which was expected. The FBS is a college football division in which many teams featured in other queries are a part of. The NHL is a professional hockey league that the Kansas City Scouts and Quebec Nordiques formerly belonged to.
7. Calculation of precision and recall values are contingent on successfully extracting the corpus sub-directory related to the keywords of a query. If none of the keywords in a query match the label assigned to a sub-directory, the query is still processed and results are obtained without a precision and recall calculation (see InvertedIndex.calculatePrecisionRecall function).

Queries Used to Collect Corpus Documents and Test Program:

1. How many Norris did Nik Lidstrom win?
2. Why did Georgia Tech leave the SEC?
3. Quebec Nordiques sale
4. Kansas City Scouts history
5. Iron Bowl results
6. FBS rankings explained
7. NHL top players
8. NFL sack leaders
9. Heisman race updates
10. Stanford Cardinal football

Results:

QUERY: How many Norris did Nik Lidstrom win?

PRECISION: 85.71%

RECALL: 90.00%

QUERY: Why did Georgia Tech leave the SEC?

PRECISION: 95.24%

RECALL: 100.00%

QUERY: Quebec Nordiques sale

PRECISION: 95.24%

RECALL: 100.00%

QUERY: Kansas City Scouts history

PRECISION: 86.36%

RECALL: 95.00%

QUERY: Iron Bowl results

PRECISION: 76.19%

RECALL: 80.00%

QUERY: FBS rankings explained

PRECISION: 73.91%

RECALL: 85.00%

QUERY: NHL top players

PRECISION: 26.92%

RECALL: 35.00%

QUERY: NFL sack leaders

PRECISION: 95.24%

RECALL: 100.00%

QUERY: Heisman race updates

PRECISION: 90.48%

RECALL: 95.00%

QUERY: Stanford Cardinal football

PRECISION: 95.24%

RECALL: 100.00%