Kevin Nguyen
EID: kdn433
Project 1
CS371r

Project 1 implementation

The purpose of this project is be able to index whole phrases for document retrieval. However, the original system did not support phrases, but only individual tokens. The implementation involves additional functionality to existing code, new functions, and a new java file. It should be noted that only minor additions were made to the original code, but everything that was original still have their standard functionality. There are a total of three files that was modified with additional functionality and they're InvertedIndex.java, HashMapVector.java, and Document.java. In order to implement existing functionality, reading the original code is essential then applying small additions at a time to complete the project.

InvertedPhraseIndex.java was a new java file that was created to implement the phrase indexing. InvertedPhraseIndex.java is responsible for managing a list of possible bigrams and to sort the list of possible bigrams; as a result, we can remove low frequency bigrams from the index list later. There are two distinct functions in the file and they're called BigramProcess and filterBigrams. BigramProcess will check by conditional probability and obtain as many bigrams as possible. The bigrams are stored into a map with the count for safe keeping. However, if the phrase already exist then we just update the count value in the data structure otherwise it will be overwritten. The function, filterBigrams, will sort the bigram list first. Then, it will take the sorted bigram list and filter out the non-top 1000 bigrams from the index list by calling HashMapVector.java. The java file is also the new entry point when the program runs instead of InvertedIndex.java; as a result, InvertedIndex.java extends to InvertedPhraseIndex.java. InvertedIndex.java is the subclass whereas the InvertedPhraseIndex.java is the superclass.

The InvertedIndex.java still retains the functionality of indexing through the document and utilizing the query vector as normal. However, since the indexing occurs here in this file, a new constructor must be made for the superclass, InvertedPhraseIndex.java. When indexing through the document and tokens, new statements were inserted into the IndexDocument functions to call the bigram functions and by calling the initBigrams method which starts off the bigram operations. The bigrams are collect on the first pass through the system, but the individual tokens and bigrams are still added to the index list. It should be noted that the index list can become ridiculously large until it is filtered. After the index list is fully populated, filterBigrams function gets called after looking at all documents. FilterBigrams is also assisted by the HashMapVector.java file.

The HashMapVector.java was added more functionality in order to assist with bigram filtering. The new functionality has methods called sortThisMap and removeLowRanked. SortThisMap will sort the bigram map list by values. It is sorted descendingly and we populate a new sorted list of the bigrams after the 1000th element in the original list. It's also noted that a comparator was implemented and overridden to compare values in the map structure. As a result, we can use Collections.sort function to assist in our implementation. The new sorted list of non-top 1000 bigrams is returned and the removeLowRanked is eventually called. The index list is searched through and a comparison is made with the sorted bigram list. If there is a match found between the two structures then the index list must have that item removed because the bigram list is a list of non-top 1000 bigrams. As a result, the index list is filtered and it will get smaller.

The Document.java also had further functionality added. It had additional functionality to account for large query vectors instead of a single token. However, single token queries are still possible and the original functionality has been retained. The function determinedQuery is called to determine if the query is short or long and the appropriate helper function is called. The original functionality is called through the getQueryToken which gets single tokens to query. The function, getQueryPhrase, will put the tokens together and use that as the query vector for indexing.

To use the program, ensure that the extracted files are in the ir/vsr folder. In the command line setup the environment as needed with export CLASSPATH='.:/u/[user]' if it is not done yet. Then call javac *.java to compile all java files as needed. Lastly, java ir.vsr.InvertedPhraseIndex -html /u/mooney/ir-code/corpora/cs-faculty to run the program with the given document source in html mode. The mode or document source materials can be changed as needed. The program will iterate through the documents and it will also output the top 1000 phrases (no spaces) with their frequency. The user can then type in a query to lookup a document reference as needed. The original functionality remains the same, but with added features for phrase indexing.

It's interesting that the results are different from the expected results because the bigrams aren't the same. As a result, the output may look different since the bigrams that was stored could be different from another user's output. The program will work, but trying different high ranked bigrams could possibly yield better results. The issue is most likely that the process of collecting the bigrams was wrong. The process involves seeing if the count between two tokens in a given document are identical and if it is, then add the tokens together as a bigram phrase otherwise continue. Each token was analyzed repetitively and it should be noted that some tokens that appear to be a part of potential bigrams were far apart from each other. Otherwise, those tokens of potential bigrams would have been closer together. The implementation is not perfect; as a result, further improvements are needed to determine bigrams. For now, the program works by using different phrases instead of the expected ones.