Execution Instructions

This project is written and Java and built with the Apache Maven build system. To compile the project, run the following command in the directory that contains the pom.xml build file:

```
mvn compile
```

To then run the program, execute the following command, where ¡directory¿ is replaced with the directory that contains the Wikipedia article files:

```
mvn exec:java -Dexec.args="<directory>"
```

As an alternative, you can create a jar file and supply the arguments to the jar file on the command line. To create and execute the jar file, run the following command.

```
mvn package
java -jar <jar-file> <directories>
```

Motivations

The ideas for this project arise from studying the Watson machine that was created by IBM's DeepQA project. At its core, Watson is a question and answer machine that uses techniques from machine learning and natural language processing to attempt to come up with answers to questions. In 2011, Watson competed on *Jeopardy!*, defeating the best human players. The goal of this project was to create a comparatively simple emulation of Watson and have it answer questions based upon a corpus of Wikipedia articles.

This program uses a moderately sized collection of Wikipedia articles as a base for answering questions that are formatted in a fashion similar to how questions are posed in *Jeopardy!*. The reasoning behind using Wikipedia articles as an answer base rests upon the idea that the articles related to the answer for a question will have some version of that question in the text of the article. In theory, a *Jeopardy!* style question could be answered by such a system by finding the Wikipedia article that is most related to the terms in the question.

Methodology

To achieve the speed and precision necessary for this project, the state-of-the-art library Lucene was used to handle text processing and indexing. In addition to basic indexing capabilities, Lucene also provides us with the ability to efficiently search through this large corpus of text without having to scan the entire collection each time. Lastly, Lucene also has various text processing utilities, such as stemming and stop word removal for English, which help to improve the retrieval statistics for articles in the collection. The success of these additional utilities varied and will be discussed in the results section below.

In an effort to improve the retrieval statistics of the system, approaches which sought to leverage aspects of the Wikipedia corpus as well as the format of the questiosn being asked were made with varying degrees of success. For example, each question in *Jeopardy!* falls under a specified category. One of the initial optimization efforts attempted to use this category to narrow down the articles to those which are related to the category in question. This was unsuccessful because the category supplied alongside *Jeopardy!* questions is often a pun or play on words which was hard for the system to detect. Efforts could be made through the use of machine learning models to train the system on how to properly weight and respond to the categories of the *Jeopardy!* questions.

Efforts were also made to remove potential clutter from the user's queries. In a *Jeopardy!* style question, there were query elements that provded detrimental to the performance of the system. Some examples of these were punctuation elements. However, some of the elements found in a query were actually beneficial to performance. During performance testing, it was discovered that leaving quotations in a query actually benefits performants. The reason for this has to do with the fact that quotations in questions usually refer either to specific quotes by a person or excerpts from something like a song or book, which is may be referenced in an author or artists Wikipedia page. In the case that it is explicitly referenced, it does a very good job of narrowing down the set of answers. This operation also helped to prevent errors when Lucene attempted to parse queries and finding syntax that resembled Lucene query operators.

In the end, the overall query path involved some pre-processing of the query to remove unnecessary punctuation and then a lookup against the contents of the documents in the collection.

Implementation Challenges

There are a variety of aspects that are specific to this problem space that contribute to making this a difficult problem to solve. An example would include the sheer amount of content in the Wikipedia corpus. Not only just the sheer volume of content, but the structure of the corpus is also challenging to work with. This combines with the abstract nature of the queries, formated as *Jeopardy!* questions, to make accurate retrieval a difficult problem to solve.

One of the first challenges that I was presented with was the structure of the articles in the corpus. There are many specifiers in the document structure that I did not take advantage of, but likely could have been leveraged to greatly improve the retrieval statistics for the system. These include things like redirects, which have the effect of supplying aliases for documents in the corpus. Because these are not handled in the system, questions which call specifically for those results will always fail to accurate retrieve results. Additionally, articles in the corpus are also related through soft links (the "see also" sections) and categories present in some of the articles. These soft links and categories could be used to create a hierarchical structure which could be combined with link analysis or a tiered index system to improve the overall performance.

The other major challenge that I faced was effectively tuning the system. When dealing with the BM25 similarity algorithm, tuning the hyperparameters b and k_1 to perform optimally was a challenge. My results prior to tuning experiments with BM25 were much worse than my results with a standard TF-IDF model, but once the parameters were properly tuned, BM25 resulted in much better performance compared to TF-IDF. This process was a combination of trial and error combined with thinking about the kinds of assumptions I could make about the documents in the collection.

Results

The system's generate performance is described in the table below. The performance statistics are reported as Mean Reciprocal Ranks as this provides the most straight forward information at a glance at how well the system is performing.

	TF-IDF Scoring	Okapi BM25 Scoring
Stemming	.22	.27
Lemmatization	N/A	N/A
No processing	.17	.27

As is shown in the table above, the best performance in the system was achieved by using a tuned Okapi BM25 scoring system when stemming was taking place. In that environment, the system was able to correctly answer 27 of the 100 test questions. This is compared to the lowest configuration, TF-iDF without any stemming. Results for lemmatization are not reported because lemmatization tests were not run due to the required indexing time.

Error Analysis

This system exhibited several different kinds of errors in its retrieval throughout testing and tuning. One of the most common errors that the system would exhibit would be a tendency towards more broad and general topics pages for answers. For example, if you posed a question which referenced a specific treaty in history, a common result would be a page on international treaties in general as opposed to the specific treaty that the question actually referred to. This seems to be because general pages are more likely to have the terms in their articles just due to length. Attempts to correct for this by tuning the BM25 algorithm proved unsuccessful.

The other large error in my retrieval system had to do with the way that I was parsing articles. I did not make any efforts to establish any relationships between articles in the system which were linked in the corpus, either through aliases or direct links. This limits my system because it means that certain terms will never be retrieved since the relationships that might be explicit in a query are not translated into the index representation. This could be remedied by the addition of fields to my document representation that allowed the system to encode and understand links.

Effects of Stemming

Stemming had a very large effect on the recall and precision of articles in this system. In the non-stemmed experiments, the system was able to retrieve a moderate amount of documents but there was sometimes a lot of distance between a retrieved result and the desired answer. Stemming helped solved this somewhat, although more work would need to be done to leverage distinct aspects of this corpus to increase recall even more.