Assignment 4
Information Retrieval

In this assignment you will be improving upon a rather poorly-made information retrieval system. You will build an inverted index to quickly retrieve documents that match queries and then make it even better by using term-frequency inverse-document-frequency weighting and cosine similarity to compare queries to your data set. Your IR system will be evaluated for accuracy on the correct documents retrieved for different queries and the correctly computed tf-idf values and cosine similarities.

You will be using your IR system to find relevant documents among a collection of 60 short stories by Rider Haggard. The training data is located in the data/ directory under the subdirectory RiderHaggard/. Within this directory you will see yet another directory raw/. This contains the raw text files of 60 different short stories written by Rider Haggard. You will also see in the data/ directory the files queries.txt and solutions.txt

Your goal is to improve upon the IR system given. This involves implementing:

- **Inverted Index:** Implement an inverted index - a mapping from words to the documents in which they occur.
- **Boolean Retrieval:** Implement a Boolean retrieval system, in which you return the list of documents that contain all words in a query using the strategy from lectures.
- **TF-IDF:** Compute and store the term-frequency inverse-document-frequency value for every word-document co-occurrence:
  $$w_{t,d} = (1+\log_{10}\mathrm{tf}_{t,d})\times\log_{10}(N/\mathrm{df}_t)$$
- **Cosine Similarity:** Implement cosine similarity in order to improve upon the ranked retrieval system, which currently retrieves documents based upon the Jaccard coefficient between the query and each document. Note that when computing $w_{t,q}$ (i.e., the weight for the word w in the query) you should *not* include the idf term. That is, $w_{t,q}=1+\log_{10}\mathrm{tf}_{t,q}$. **Also note that the reference solution uses *ltc.lnn* weighting for computing cosine scores**.

To improve upon the information retrieval system, you must implement and/or improve upon the following functions:

- index(self): This is where you will build the inverted index. The documents will have already been read in for you at this point, so you will want to look at some of the instance variables in the class:
  - self.titles
  - self.docs
  - self.vocab
- boolean_retrieve(self, query): This function performs Boolean retrieval, returning a list of document IDs corresponding to the documents in which all the words in query occur.
- compute_tfidf(self): This function computes and stores the tf-idf values for words and documents. For this you will probably want to be aware of the class variables vocab and docs which hold, respectively, the list of all unique words and the list of documents, where each document is a list of words.
- rank_retrieve(self, query): This function returns a priority queue of the top ranked documents for a given query. Right now, it ranks documents according to their Jaccard similarity with the query, but you will replace this method of ranking with a ranking using the cosine similarity between the documents and query.

It is better to work on them in that order, because some of them build on each other. It also gets a bit more complex as you work down this list.

Your IR system will be evaluated on a set of five queries (encoded in the file **queries.txt**):

- separation of church and state
- priestess ritual sacrifice
- demon versus man
- african marriage queen
- zulu king

Your system will be tested on the four parts mentioned above: the inverted index, boolean retrieval, computing the correct tf-idf values, and implementing cosine similarity using the tf-idf values.

In order to run your code, execute

```
python assignment4.py
```

This will run you IR system and test it against the development set of queries. If you want to run your IR system on other queries, you can do so by replacing the last line above with

```
python assignment4.py "My very own query"
```

Note that the first time you run this, it will create a directory named stemmed/ in ./data/RiderHaggard/. This is meant to be a simple cache for the raw text documents. Later runs will be much faster after the first run. *However*, this means that if something happens during this first run and it does not get through processing all the documents, you may be left with an incomplete set of documents in ./data/RiderHaggard/stemmed/. If this happens, simply remove the stemmed/ directory and re-run!