CS160 - Assignment 3
Due: Friday Oct. 9, 6pm

For the next step in our IR system we're going to be adding functionality to do ranked retieval using vector space models (i.e. TF-IDF). Given a query, the system will return a ranked set of documents using the cosine similarity between the query term vector and the document term vectors. For the query vector representation, we will always just use term frequeny counts. For the document representation, we will support a number of normalization/weighting techniques that we have discussed, specifically: term log frequency, term boolean frequency, inverse document frequency weighting and cosine normalization. Your index should be able to be built with a combination of these, depending on user specifications.

Although we will be modifying the data stored in the index, our new system will be able to support BOTH boolean and ranked retrieval at the same time. Any query that contains "AND", "OR" or "!", will be treated like a boolean query and return the boolean query results. Maintaining this is not challenging, but I want to make it explicit so that you keep it in the back of your mind as you are modifying the code. For this assignment I am giving you a fair amount of freedom with how you implement this functionality. The main requirements are that you follow the minimal specification below, that your code work :) and that it works efficiently.

We will be building on top of the code that we used last time. I have provided a working version of the code from last time at "/common/cs/cs160/assign3", however, you are welcome to extend your solution to assignment 2. If you decide to use your own code, you will be held accountable for any mistakes/issues from assignment 2 that may cause interactions with the current assignment.

You may work with a partner if you'd like. You **MUST** both be there when you are working on the assignment (either coding or writeup) and you may

only use one computer, i.e. I want you to do pair programming. When you submit your write-up, make sure both people's names are in there.

Before starting this assignment read through the **entire** document. Although I've given you more freedom this time, certain interfaces are made explicit to both assist you in developing your code and to make my life easier for testing. At the end I've included some helpful hints and tools that may be useful. If there is any ambiguity or question about what you are being asked to do, ask the me to clarify.

1. What to implement/changes

   Below are descriptions of the classes you must implement. In each case, I have included a skeleton of the class in the code I provided you and have defined which public methods you must define. Since you may be reusing your own code, for pre-existing classes (e.g. PostingsList) I've explicity listed the additional functions that you must add.

   (a) Handling different query results: QueryResult.java

   You don't have to do anything here, but take a look at this new interface. This allows us to support multiple different types of query results from the system (in our case, the ranked result and the result from a boolean query). Two of our classes will implement this interface.

   (b) Posting list representation: PostingsList.java

   I have provided a postings list implementation for a boolean index. You will need to change this to support functionality for doing ranked retrieval. The following MUST be changed, but you may also need to add other functions not listed here:

   - Rather than just occuring or not occuring, you must modify your postings list representation to support weights associated with each entry. For example with no normalization, this would represent the term frequency of a term in a document.
   - Your other functions (not, andMerge and orMerge) should still work appropriately for boolean queries.
   - PostingsList now implements QueryResult. The only major change to our existing code is to add the

     `public double[] getScores()`

method. This method should just return 1.0 for the score of each document.

(c) Ranking query results: VectorResult

For a boolean query, the result is a PostingsList. For a ranked query, the result is a VectorResult. This class stores the results of our ranked query, which is a ranked list of the document ids and the score associated with each document. As with PostingsList, VectorResult implements the QueryResult interface.

(d) Index generation and query processing: Index.java

As with boolean queries, our Index class will generate and store the index as well as handle queries to the index.

- Modify the index construction to keep track of term frequencies.
- I've added enums in the class that define our different normalization techniques. These are used to specify how we should normalize our document counts in our index when it is being constructed.
- To support these normalization techniques, the constructor has been modified.
- You'll need to modify your index construction to support the different normalization techniques. This will require some cooperation with the PostingsList class so you may need to add functionality in multiple places.
- Implement the rankedQuery method, which issues a ranked query to our index:

  `public VectorResult rankedQuery(String textQuery)`

  . This is should be roughly equivalent to the functionality in Figure 6.14, except we will return a full sorted list of results. Note that the resulting VectorResult must be sorted in decreasing order by score (ties should be broken by docID, with lower docIDs occuring before/earlier than higher docIDs). You can enforce this sorted order either in this method or in the VectorResult class, but either way, it must be efficient, that is only sort once.

(e) Querying the index: Search.java

The class BooleanSearch has been changed to Search. There is now added functionality for ranked queries (which are the default)

to detect boolean queries (as described above) and issue those queries to the index. You'll also notice the index construction now passes in the normalization parameters. You don't need to make any changes to this class, but it is provided as an interface into the index and for you to see how the index is created and used.

2. Hints/Comments

   - If you're unsure about the normalization techniques, look at table 6.7 in the book.
   - Your code should be efficient! It will take a little longer to create the index and to answer ranked queries than before, but it should all still happen relatively quickly.
   - When applying the normalization techniques, the easiest way to do it is to first create the index with just term frequencies. Then, for each of the normalization steps, make a pass over the index and modify the weights.
   - The length normalization does NOT normalize the postings list length; it normalizes the document lengths. This is a little tricky to do, but look at the book/notes and think about how you might do this.
   - We are NOT doing any normalization on the query. For the query, we're just using term frequencies.
   - Note that there is a natural ordering to apply the normalization techniques (and make sure that the last thing you do is the length normalization!).
   - Use natural logs for all of your logs (this is the default for java)
   - Make sure you don't change any methods of the existing code. We want to be able to issue both boolean and ranked queries using our predefined interfaces.
   - It can be useful to make a sample test set to check your results. Try as best as possible to do incremental changes and then test to make sure that functionality is working before adding additional functionality.
   - The amount of code actually written for this assignment won't be a lot. The hardest part will be figuring out what to change. I would encourage you to spend an hour mapping out how you plan

to modify the existing implementation before actually starting coding. This will make your life much simpler in the long run and save you time.

3. What to turn in and how to turn it in

   - What to turn in:
     - A "jar" file of your code, which should contain all classes required to get your code working, including the original files I provided. See the assignment 1 writeup for details on creating a jar file. Make sure that you check the box to include the source in your jar.
     - A text file with the following information:
       (a) Name(s)
       (b) What was the most challenging part of this assignment?
       (c) How long did it take you?
       (d) When did you start?
   - How to turn it in

     See the course web side for details (it's the same procedure as last time).