Homework 1

In this homework you will build and search inverted indices by hand, using your own code, and using Lucene.

This homework is due at 11:59pm anywhere in the world on October 16th.

Part 1

Task: Using slide #16 from the lecture on September 27th as a guide, draw an inverted index for the following document collection:

Doc-ID	Text
1	it's friday friday
2	gotta get down on friday
3	friday friday
4	getting down on friday

We'll get you started:

term	doc.freq.		postings list
get	1	=>	2

Submit: Turn in the full inverted index for this tiny document set.

Part 2

Copy into your own `/l2/users` directory the file `/l2/users/cs506-tir/smalltweets.txt`.

This file contains a list of tweets (i.e., messages posted on Twitter). In this part of the homework, you will be building an inverted index for this set of tweets. Assume that each line in the file (i.e., each tweet) is a single document. The number starting at the beginning of the line and ending at the first whitespace is the document ID. The remainder of the line is the text of the document itself.

Task: Build an inverted index of this set of tweets. You can use the approach outlined in the Manning et al. book, or you can use your own approach (e.g., while reading through the tweets, create a hash whose keys are terms and whose values are lists of postings).

A few notes:

- You do not need to do any text normalization, unless you want to. Simply splitting on whitespace is

fine.
- You can use any programming language you like.

Submit: Turn in your commented code and a print-out of the postings for all of the terms beginning with the letter "a".

Part 3

Task: For this part of the homework, you will implement the merge algorithm for intersecting the postings of two terms, which was described in slides 13-14 of Lecture 2 from September 27th. When there are more than two search terms, be sure to perform the most restrictive intersection first, as described on slide 16.

Submit: Turn in your commented code and a list of document IDs matching the following boolean queries:

- egg AND cheese
- chocolate AND strawberry
- eggs AND cheese AND bacon

Part 4

Task: In this part of the homework, you will fill in gaping holes in a Java program that uses Lucene to index documents, review the tokenization, inspect the index, search for a query, and print out the matching documents and their scores. The program is found here:

```
/l2/users/cs506-tir/IndexHW1.java
```

Copy the file to your own directory. Open it in a text editor, and you'll find several places that say:

```
// ===> YOUR CODE GOES HERE <====== //
```

This will be followed by a description of what code you need to write and some suggestions for how you should go about writing it.

Here are some general helpful hints:

- Run your code just on the smalltweets.txt file to start, especially when you're doing all the printing involved in inspecting the index and tokenization. When you've got something running, email Emily and ask for a link to the big corpus and a reminder about how to use Condor.
- Don't forget to set your classpath! You can look in HW0's runTest.sh to see how to set the classpath.
- We are using Lucene 3.6.1. The API for Lucene changes frequently, so you should always make sure you are loooking at the Javadoc for the right version.
- HW0's IndexExperiment.java also contains many helpful examples.

Submit: Submit your commented code. We will be running it on the cluster to make sure it works, so please verify that it compiles and runs correctly on one of the bigbird machines.

Bonus

Task: Write your own Analyzer that extends StandardAnalyzer (source here). This new Analyzer should do at least two new things:

- Use the Lucene tokenizer for email and web addresses.
- Use a larger set of stop words of your own choosing.

Replace the analyzer in the IndexHW1.java code with your new analyzer. Think of some queries that you think might retrieve different results with your new analyzer.

Submit: Submit your commented code. We will be running it on the cluster to make sure it works, so please verify that it compiles and runs correctly on one of the bigbird machines. In addition, give a few examples where the results have changed (if you can find any), and explain why you think this might be.