Homework 5

Due: 4PM on December 5, 2017 (Tuesday)

Objectives:

- Have an experience to use Map interface to solve an interesting problem of comparing two documents.
- Try to build an application that meets your client's minimal acceptance tests and even more.

In homework 1 and homework 3, you implemented `MyArray` and `SortedLinkedList` classes through which you were able to parse words from a text file. They have some of the useful capabilities and information.

Additionally, in homework 4, you implemented `MyHashTable` class to store words and their frequencies from a file into HashTable so that you can search a word faster and see the frequency of the word.

Moving on, it is about time for you to handle two files, not just one. A main question to be answered in this homework assignment is:

How similar two documents are?

The similarities between documents are to be determined by the degree of the overlapping in contents of two documents. There are many different and complex algorithms to answer this question. In this homework, you are going to use an algorithm, called *cosine similarity*.

It is a measure of similarity between two vectors by measuring the cosine of the angle between them and ***the algorithm has been used in search, text mining, and data mining***. The cosine of the angle between two vectors basically determines whether two vectors are pointing in roughly the same direction or not. (http://en.wikipedia.org/wiki/Cosine_similarity)

$$similarity = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

where "." denotes the dot-product of the two frequency vectors A and B. And, ||A|| denotes the length of a vector.

In this homework, you are to implement the following Java class

```
public class Similarity
```

You must implement the following constructors and methods in your Similarity class.

```
public Similarity(String string)
public Similarity(File file)
public int numOfLines()
public "?" numOfWords()
```

```
public int numOfWordsNoDups()
public double euclideanNorm()
public double dotProduct(Map<String, “?”> map)
public double distance(Map<String, “?”> map)
public Map<String, “?”> getMap()
```

If a String argument is passed when creating a Similarity object, you do not need to worry about number of lines. You can assume that it is one line with no special characters for new lines in the argument. The return type of the `numOfWords()` method is something you need to figure out to meet the requirements. Likewise, the data type of mapped values (marked with question marks above) in your map data structure is something you need to figure out to meet the requirements. Think carefully about what you need to be careful when `getMap()` method returns a reference to the Map instance in your class.

There are four steps you need to take care of to find out the cosine similarity between two texts or two files (documents).

Step 1: Similar to the results of homework 4, find out word frequency distribution for a text. Your class should have two constructors, one taking a string value and the other taking a file name. And, using proper classes of the Java Collections Framework, you should store data into them. For example, if you read the string, “nice to meet you you look nice” then the first information you need is as follows. In terms of how to split text, please refer to HW3Driver.java file.

{look=1, meet=1, nice=2, to=1, you=2}

Step 2: Once you have words and their frequencies, the frequency of each word in the text should be used to find the Euclidean norm that is the length of the vector.

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$$

Step 3: Next step you need to take is to find out the dot-product of two frequency vectors X and Y as follows.

$$X = \{x_1, x_2, \ldots, x_n\};\ Y = \{y_1, y_2, \ldots, y_n\};$$

$$X \bullet Y = x_1\, y_1 + x_2\, y_2 + \ldots + x_n\, y_n$$

Step 4: Now you are ready to find out the cosine similarity or distance, dist(D1, D2), of two texts or files as follows.

$$\text{dist}(D_1, D_2) = arccos\left(\frac{\text{freq}(D_1) \bullet \text{freq}(D_2)}{\|\text{freq}(D_1)\| * \|\text{freq}(D_2)\|}\right)$$

A word is a sequence of letters [a..zA..Z] that does not include digits [0..9] and the underscore character. Here are examples of non-words: abc123, a12bc, 1234, ab_c, abc_

Gently ignore any non-words in your code. Unlike homework 4, your application is in charge of converting all the words passed to you either through a String object or a file to lowercase.

Obviously, the distance between two identical documents is 0.
And if two texts or documents do not have any common words, then the distance is **`Pi/2`**.
To find out whether your program meets the requirements or not, your client, TERRY LEE CONSULTING, Inc., provided three test files along with the expected results for you. Your code must produce the EXACTLY SAME values of the expected output to pass the minimal acceptance tests.

Be careful!

- When implementing the `dotProduct()` method, you need to make sure it does not fall into quadratic running time on average. Think carefully!
- Processing large files might lead to an incorrect distance between two documents. Think about why! Your code must be able to handle any given input file regardless of its size (extremely large and maybe with so many duplicate words, etc.). Even if it may not seem to happen with realistic random datasets, what your client is asking here is for you to be EXTREMELY defensive here and your code should reflect it.
- Make sure to use proper constants and methods in `java.lang.Math` class.

Deliverables:

- Submit your `Similarity.java` file using Autolab (https://autolab.andrew.cmu.edu). Do not zip it. Do not use package.
    - Using either single-line or multi-line comments, be sure to mention what data structures from the Java Collections Framework you chose and for what purpose.
    - Also, at the top of your `dotProduct` method, clearly mention why your implementation does not fall into the quadratic running time complexity on average.

Grading:

Autolab will grade your assignment as follows.

- Working code: 90 points
- Coding conventions: 10 points
    - We'll deduct one point for each coding convention issue detected.

In case you do not pass test case(s), please spend some time to think about edge cases you may have missed and test thoroughly before asking questions to the TAs and resubmitting.

Autolab will show you the results of its grading within approximately a few minutes of your submission. You may submit multiple times so as to correct any problems with your assignment. Autolab uses the last submission as your grade.

The TAs will take a look at your source code to check correctness and design. The most important criterion is always correctness. Buggy code is useless (even if you may think a found bug is very minor). It is important that your code be readable and well organized. This includes proper use of clear comments. Points will be deducted for poor design decisions, unreadable code.

As mentioned in the syllabus, we will be using the Moss system to detect software plagiarism. Make sure to read the cheating policy and penalty in the syllabus. *Any cheating incident will be considered very seriously. The University also places a record of the incident in the student's permanent record.*

Late submissions will not be accepted and, if you have multiple versions of your code file, make sure you do submit the correct version.