Homework 5 (due by 4 pm on Dec 3)

Objectives:

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

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 and 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, cosine similarity.

It is a measure of similarity between two vectors by measuring the cosine of the angle between them and ***it has been used in search, text mining, and data mining***. The cosine of the angle between two vectors thus 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

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.

{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.

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

Good news is that you can convert all of the words in a file to be lowercase.

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_

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 = 1.57

To find out whether your program meets the requirements or not, your client, **TERRY LEE CONSULTING**, provided three test files along with the expected results for you. Your code must produce the expected output to pass the ***minimal*** acceptance tests.

Looking at the test files, you should be able to understand what methods need to be implemented in your class. Here is the first test file for your reference.

```
/**********************************************************************
 * 95-772 Data Structures for Application Programmers
 * Acceptance Test (case 1)
 *
 **********************************************************************/
public class Test1 {
    public static void main(String[] args) {
        Similalrity map1 = new Similalrity("hello there nice to meet you you look nice");
        System.out.println(map1.numberOfLines() + " lines.");
        System.out.println(map1.numOfWords() + " words.");
        System.out.println(map1.numOfWordsNoDups() + " distinct words");
        System.out.println(map1.euclideanNorm() + " Euclidean norm.\n");

        Similalrity map2 = new Similalrity("time to say hello nice seeing you should meet
again");
        System.out.println(map2.numberOfLines() + " lines.");
        System.out.println(map2.numOfWords() + " words.");
        System.out.println(map2.numOfWordsNoDups() + " distinct words.");
        System.out.println(map2.euclideanNorm() + " Euclidean norm.\n");

        System.out.println(map1.dotProduct(map2.getMap()) + " dot product.");
        System.out.println(map1.distance(map2.getMap()) + " distance.");
    }
```

```
}
/* EXPECTED OUTPUT
0 lines.
9 words.
7 distinct words
3.605551275463989 Euclidean norm.

0 lines.
10 words.
10 distinct words.
3.1622776601683795 Euclidean norm.

7.0 dot product.
0.9097531579442097 distance.
*/
```

Be careful!

- ***When implementing the dotProduct() method, you need to make sure it does not fall into quadratic running time. Think smart!***
- **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*.**

Deliverables:

- A few sheets of paper that have your initial code as well as your comments (Submit this in class that is on the due date.)
  - *Be sure to mention what data structures from the Java Collections Framework you chose and for what purpose!*
- Your source code. (Submit your Similarity.java file on Blackboard by the due date.)

Grading:

Your homework will be graded first by compiling and testing it. After that, we will read your code to determine appropriate methods/classes are used. In addition, we will judge the overall style and modularity of your code. Points will be deducted for poor design decisions, uncommented and unreadable code. Your comments on your paper should be able to demonstrate your proper understanding of the code.

We will not accept any late submission and please make sure to submit the correct version of source code file. We will only grade the one that is submitted before the due.

- Working code: 60 points
- Coding style (refer to the guideline): 20 points
- Your paper code's comments: 20 points