**Lab 2.**

Load the hash map that you created in Lab 1 in main memory. Next, examine the file that contains the queries: queries.txt. Your job is to compute the top 20 documents for each query using the TF-IDF algorithm and cosine similarity distance. Create a hash map that has as a key the query number and a value an ArrayList of the top 20 relevant documents in order. Store the hash map to a file. Please use query numbers starting with 1 and incrementing by 1 for each query. In other words, your query numbers should be between 1 and 225. **This is different than the query number in the quieries.txt file.**

Note that you will need to create a document vector for each query using the TF-IDF algorithm. Feel free to store the query number and document vector in a hash map. Please, also normalize the frequencies of the words in the documents by dividing by the highest frequency. Then you can use the cosine metric to compute the distance between the query and each document.

Here is an example code that reads the document vectors data.

try (ObjectInputStream is = new ObjectInputStream(newFileInputStream(new File("./files/docvector")))) {

documentVectors = (DocumentCollection)

is.readObject();

} catch (Exception e) {

System.out.println(e);

}

Here is the expected output for the first query:

documents for query 1: [13, 184, 12, 51, 486, 1268, 327, 435, 746, 875, 665, 686, 359, 878, 494, 14, 1144, 1186, 332, 154]

Hints:

1. Remember that it is possible that a query word does not appear in any document. In this case, set the inverse document frequency to 0.
2. There are documents that contain no text. If a document is empty, then the distance from it to any query should be equal to 0 and the distance function should not be called.

Please follow the following design.

labs.Lab2.java (about 30 lines)

Contains two variables:

* public static DocumentClasses.DocumentCollection documents; //the documents
* public static DocumentClasses.DocumentCollection queries; //the queries

main(): Load data from your binary file.

Next, initialize the queries variable, and call the normalize() method on both variables. Finally, print the 20 most relevant documents for each query.

DocumentClasses.TextVector (about 110 lines)

* make class abstract
* add three abstract methods:
  + public abstract Set<Map.Entry<String, Double>> getNormalizedVectorEntrySet() //returns the normalized frequency for each word
  + public abstract void normalize(DocumentCollection dc); //will normalize the frequency of each word using the TF-IDF formula
  + public abstract double getNormalizedFrequency(String word); //will return the normalized frequency of the word
* add getL2Norm() method that returns a double. The method calls the getNormalizedFrequency method to get the normalized frequencies. Then it returns the square root of the sum of the squares of the frequencies.
* Add findClosestDocuments(DocumentCollection documents, DocumentDistance distanceAlg) method that returns the 20 closest documents as an ArrayList<Integer>. Calls the method findDistance on the distanceAlg variable multiple times. (about 15 lines of code)

DocumentClasses.DocumentDistance is an interface. Contains a single method signature.

double findDistance(TextVector query, TextVector document, DocumentCollection documents); //will return the distance between the query and document

DocumentClasses.CosineDistance implements DocumentDistance. Contains implementation of abstract method.

DocumentClasses.QueryVector/DocumentVector extends TextVector. Contains variable:

private HashMap<String, Double> normalizedVector = new HashMap<String, Double>();

Implements the three abstract methods.

DocumentClasses.DocumentCollection (slight change).

* Constructor: takes a second parameter of type String. If equal to “document”, adds DocumentVector objects to HashMap. Otherwise, it adds QueryVector objects to HashMap.
* Contains normalize(DocumentCollection dc) method that calls the normalize(dc) method on each document in the collection.