CS102 – Algorithms and Programming II
Lab Programming Assignment 2
Fall 2022

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn2_YourSurname_YourName.zip
- Replace the variables "YourSurname" and "YourName" with your actual surname and name.
- You may ask questions on Moodle.
- Upload the above zip file to Moodle by the deadline (if not significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TA during the lab. You will resubmit your code once you demo your work to the TA.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Document Similarity Using Verb Frequencies

In this assignment, you are going to implement a Java program to estimate document similarity based on verb frequencies. The frequency of the words can give information about the content of a document. The more similar the word frequencies the more similar two documents would be. For the sake of simplicity, we will only count the occurrences of specific verbs included in the given text file ("*verbs-dictionaries.txt"*). Each line of this text file contains an English verb together with its different forms, each form separated by a tab. First word in each line corresponds to the base form of the verb of interest. While counting the verbs in a document, you are going to consider the different forms of a verb being the same. For example, occurrences of "have", "has", "had", and "having" will be considered as the same verb in the base form: "have".

In this assignment, you should ignore case sensitivity for string comparisons as verbs may appear differently based on their context.

Implement a **Verb** class that keeps the base form of the verb as a string. This class should also contain a string array to hold the different forms of the verb. Constructor of this class should get five forms of the verb as parameters and set the class variables accordingly. Implement a *boolean check(string word)* method that returns true if the given word is equal to the base form or any alternative form of the **Verb** object. We will use this method for finding out the corresponding Verb object, given a word of the document.

Implement a **VerbDictionary** class to hold an array of **Verb** objects (you may also use ArrayList for this task). This class should include *void processDictionary(string fileName)* to create Verb objects processing the given "*verbs-dictionaries.txt".* In this file there are a total of 6710 verbs, each with 5 forms. You may create your arrays based on these numbers. This method should go through each line of this file and create a Verb object using the five different forms of the verb. You may use StringTokenizer for getting the words of a line in this file, using the delimiter "\t" for the tab character.

In **VerbDictionary,** you should implement *int findVerbIndex(string word)* method, which returns the index of a verb if the given word exists in the array of Verb objects, and -1 if it is not included in the verb array in any form. For this task, you need to iterate over all the **Verb** objects in the array and use their *check* method to see if the given word matches any existing verb. When you find a matching one, you can return the index of the **Verb** object in the array. If you iterate over all the verbs and cannot find a matching one, simply return -1.

Implement an **InputDocument** class to process the verb counts of a given document text file. This class should keep an int array to track the number of occurrences of each verb in the **VerbDictionary**. Since **VerbDictionary** contains 6710 verbs, you need an integer array of size 6710. This integer array would count the occurrences of the corresponding verbs in **VerbDictionary**. For example, if index 0 is the verb "abandon" in **VerbDictionary**, then index 0 of the integer array in **InputDocument** will be the count of the verb "abandon" in the document. We provide 10 text files that contain public domain books for you to test your implementation in the attached "Documents.zip" file.

The constructor of **InputDocument** class should receive the filename of the input document to read the text file. For ease of access, you should place the input files into your class folder. You should process the text file word by word (you may use spaces and punctuation as delimiters) and use *findVerbIndex* method of **VerbDictionary** to find the indexes of the verbs included in your dictionary. You can use StringTokenizer to get the words of each line of your text file. Note that you may provide a set of different delimiters as a String for the tokenizer. For example, in the sample output, we used the delimiters as " .,;?!()[]{}'\"“”". You may include more symbols for better results. For each word of the document, try to find its verb index using *findVerbIndex* method of **VerbDictionary**, if it exists, increment the corresponding count in **InputDocument**. If the word does not exist in **VerbDictionary**, then you should continue with the next word of the document. After going through all the document, the integer array in **InputDocument** should have the count of each verb in our dictionary. Include a totalVerbCount integer in this class that is equal to the sum of all verb counts (sum of all elements of the integer array in **InputDocument)**.

Include a method *void printTop5()* in **InputDocument** to print out the top 5 most common verbs appearing in this document. For this task, you need to find the max, second max, third max, fourth max, and fifth max element in the verb count array. Iterate over all the elements while keeping the indexes and counts of the top 5. You should not sort the array for this task. Print the base form of the verbs as well as their count in this method.

We also want to find out the frequencies of the verbs. **InputDocument** should contain a *double getFrequency(int verbIndex)* method to calculate the frequency of a verb using the following formula:

$$frequency(verb) = \frac{count(verb)}{\sum count(verb)}$$

Basically, to find the frequency, divide the count of the verb to totalVerbCount. Note that although the count of a specific verb and count of all the verbs are integers, you should make a floating number division in *getFrequency* method. Keep a double array of size 6710 for the frequencies of the verbs. Include a *void calculateAllFrequencies()* method to calculate the frequency of each verb of the document.

InputDocument class should also include *a double calculateDissimilarity(InputDocument other)* method for calculating the similarity between two documents. This method should calculate the similarity of the two documents based on the verb frequencies using the following formula:

$$dissimilarity(D1, D2) = \sum |frequency(verb, D1) - frequency(verb, D2)|$$

If D1 and D2 are the same document, their dissimilarity would be 0; the higher the dissimilarity the more different the two documents are. Note that, here you need to iterate over all the verbs in **VerbDictionary**, and calculate the frequency differences of the same verb in documents D1 and D2 while summing the absolute value of the differences.

In your main method, first create a **VerbDictionary** object and process the given dictionary ("verbs-dictionaries.txt"). Test whether your *findVerbIndex* method works by trying to find the indexes of different verbs. Then, create 10 **InputDocument** objects using the text files in "Documents.zip". The constructor of **InputDocument** should calculate the verb counts. Display the top 5 verbs in each document. Calculate all verb frequencies for each document. Then, display a dissimilarity matrix by calculating pairwise dissimilarities between each document. This matrix should have 10 rows and 10 columns, each cell corresponds to the pairwise dissimilarity using *calculateDissimilarity* method. Use a suitable formatting for the numbers so that the output is easy to read, for example, you may display 4-5 digits after the comma.

Sample output is provided at the end of this document.

Preliminary Submission: You will submit an early version of your solution before the lab. This version should at least include the following:

- **Verb** and **VerbDictionary** classes should be completed based on the given description.
- Also include a main method to test the implemented functionality, i.e., *processDictionary* and *findVerbIndex* methods.

You will have time to complete your solution in the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.

Sample Output:

```
Index of "abandon" is 0.
Index of "abandoned" is 0.
Index of "enjoying" is 2040.
Index of "notaverb" is -1.

Top 5 verbs in each document:
0: "be": 1707, "have": 824, "say": 449, "peter": 380, "do": 310
1: "be": 686, "say": 523, "have": 280, "do": 172, "trick": 155
2: "be": 4957, "have": 2481, "do": 873, "come": 793, "see": 746
3: "be": 645, "say": 641, "pooh": 381, "have": 291, "do": 182
4: "be": 721, "have": 478, "do": 149, "room": 132, "trick": 119
5: "be": 825, "have": 435, "say": 255, "trick": 167, "do": 129
6: "be": 1061, "have": 490, "say": 333, "do": 240, "come": 184
7: "be": 2979, "have": 2186, "do": 771, "will": 744, "say": 645
8: "be": 1783, "have": 1082, "say": 499, "see": 268, "trick": 264
9: "be": 2895, "say": 1025, "get": 1004, "up": 834, "out": 794

Dissimilarity matrix:
N 0      1      2      3      4      5      6      7      8      9
0 0.0000 0.7225 0.5663 0.8131 0.7281 0.6316 0.6764 0.7189 0.6258 0.7845
1 0.7225 0.0000 0.7524 0.6108 0.7642 0.7042 0.7189 0.8428 0.7358 0.7802
2 0.5663 0.7524 0.0000 0.8443 0.6885 0.6358 0.6688 0.6143 0.6062 0.7100
3 0.8131 0.6108 0.8443 0.0000 0.8457 0.8106 0.7737 0.9038 0.8623 0.8239
4 0.7281 0.7642 0.6885 0.8457 0.0000 0.7242 0.7680 0.7999 0.7679 0.8724
5 0.6316 0.7042 0.6358 0.8106 0.7242 0.0000 0.6884 0.7482 0.6389 0.8125
6 0.6764 0.7189 0.6688 0.7737 0.7680 0.6884 0.0000 0.7085 0.6903 0.7657
7 0.7189 0.8428 0.6143 0.9038 0.7999 0.7482 0.7085 0.0000 0.6617 0.7705
8 0.6258 0.7358 0.6062 0.8623 0.7679 0.6389 0.6903 0.6617 0.0000 0.7388
9 0.7845 0.7802 0.7100 0.8239 0.8724 0.8125 0.7657 0.7705 0.7388 0.0000
```

Note that your execution may yield different numbers based on how you parse the documents and the delimiters you use for the tokenizer.