School of Computing Sciences

Module: CMP-5014Y Data Structures and Algorithms
Assignment: Coursework Assignment 1

Set by: Jason Lines (j.lines@uea.ac.uk)
Checked by: Anthony Bagnall (ajb@uea.ac.uk)
Date set: Tuesday 22nd October 2019
Value: 15%

Date due: Wednesday 4th December 2019 3pm **(Week 11)**
Returned by: Wednesday 15th January 2019
Submission: PDF prepared with PASS submitted electronically on e:vision

Learning outcomes

To be able to: take informal descriptions of algorithms and describe them formally in pseudocode; assess the run-time complexity of algorithms; implement formal descriptions of algorithms using the Java programming language; run and report the findings of practical timing experiments.

Specification

Overview

This coursework assignment requires you to design, analyse, implement and test algorithms for the task of comparing text documents. This will be done by comparing all words in a document to a given dictionary of words to create a feature vector, and a distance measure will be implemented to compare documents by their resulting feature vectors.

The task is described in detail with informal descriptions of the functionality that is required; it is your task to first design and **formally describe** algorithms to solve this problem. You will then **analyse** your algorithms and **implement them in Java**. Following this, you will run a series of **timing experiments** to visualise the run-time of your algorithms, presenting and briefly discussing your results.

Please note: **do not use advanced data structures in this assignment** (such as hash maps, trees, tries, etc.). You will cover these later in the module. The purpose of this assignment is to reinforce your knowledge of describing, analysing, implementing and timing algorithms. You may use arrays when you implement your algorithms to represent lists but please do not use any Java data structure classes, such as those that implement the `Collection` interface (for example, do not use `ArrayList`).

Description

Documents

A plain text document A containing w words can be thought of as a *list* data structure, where:

$$\mathbf{A} =< a_1, a_2, ..., a_w >,$$

and a_i is the i^{th} word in the document. Note that $\mathbf{A}$ may include duplicate words, and for simplicity, all punctuation has been removed and assume that words can only be made up of lower-case letters. An example would be:

$$\mathbf{A} =<\texttt{this is an example document that is made up of words there are no capital letters but duplicates are allowed}>,$$

where in this example $w = 20$ and $a_4 =$`example`.

Dictionaries

A dictionary $\mathbf{Q}$ is also a list of s words:

$$\mathbf{Q} =< q_1, q_2, ..., q_s >,$$

However, unlike documents, a dictionary **may not include duplicate words**.

Feature Vectors

A feature vector $\mathbf{F}$ can be computed for a given document $\mathbf{A}$ and dictionary $\mathbf{Q}$ by counting the number of occurrences of each dictionary word within the document. For example, given a document:

$$\mathbf{A} =< \texttt{hello to you and hello to the world} >$$

and a dictionary:

$$\mathbf{Q} =< \texttt{cat hello world the apple} >,$$

the resulting feature vector would be:

$$\mathbf{F} =< 0, 2, 1, 1, 0 >,$$

where the length of $\mathbf{F}$ will be the same length as the dictionary $\mathbf{Q}$. For example, $q_2 = \text{hello}$ appears twice within $\mathbf{A}$, hence $f_2 = 2$, whereas $f_5 = 0$ as the word apple does not appear at all in $\mathbf{A}$.

Document Similarity Distance (DSD)

The Document Similarity Distance (**DSD**) can be computed to compare two documents using feature vectors derived from a common dictionary. More formally, given two documents, $\mathbf{A}$ and $\mathbf{B}$, and a dictionary $\mathbf{Q}$, feature vectors $\mathbf{F}_A$ and $\mathbf{F}_B$ can be calculated for documents $\mathbf{A}$ and $\mathbf{B}$ using the steps discussed above. These feature vectors can then be used to calculate the Document Similarity Distance between $\mathbf{A}$ and $\mathbf{B}$ as:

$$\mathbf{DSD}(\mathbf{F}_A, \mathbf{F}_B) = \sum_{i=1}^{s} |f_{A,i} - f_{B,i}|.$$

Exercises

Your task is to complete the following exercises to formally describe, analyse, implement (and test) the functionality that has been discussed in the problem definition and then carry out timing experiments.

Reminder: all implementation must be done in **Java** and please **do not use in-built Java data structure classes** such as those that extend the `Collection` interface. Do not use more advanced data structures such as hash tables and do not use classes such as `ArrayList` class. You may use arrays wherever necessary, however.

Part A: Calculating Feature Vectors

1. Design and write a formal description of an algorithm to calculate a feature vector when given a document and a dictionary as inputs. **(10 marks)**

2. Analyse the run-time complexity of your algorithm from question 1 to calculate document feature vectors. **(10 marks)**

3. Implement your algorithm from question 1 in Java and call it `calculateFeatureVector`. The input to your algorithm should be two `String` arrays (one for a document and one for a dictionary) and the output should be a single array of integers. **(10 marks)**

4. Design and conduct timing experiments using your implementation from question 3. How you design your experiments is your decision but you should perform multiple resamples using various document and dictionary lengths. Present your results appropriately, and briefly comment on your results and how they compare to your expectations following your run-time complexity analysis. **(10 marks)**

   For the purpose of testing your algorithm some example code has been provided in `CourseworkUtilities.java` on **Blackboard**. The `generateDictionary` method will randomly generate a dictionary and `generateDocument` will generate a document when passed a dictionary. Please note that, for simplicity, the *words* that are generated will be random characters, but this is sufficient for testing and timing experiments.

   Note: all code to generate your experimental results should be included in your submission (with informative comments) to demonstrate how you designed and performed your experiments. You do not need to submit `CourseworkUtilities.java` as this is given to you.

Part B: Comparing Documents

5. Design and write a formal description of an algorithm to compute the **Document Similarity Distance (DSD)** of two feature vectors (you can assume the input feature vectors were generated using the same dictionary and do not need to check for this). Specifically, the input of this algorithm should be two integer arrays (for the two feature vectors) and the output should be the DSD between them. **(12 marks)**

6. Design and write a formal description of an algorithm that finds the closest match for all documents within an input list.

   Specifically, the inputs of your algorithm should be:

   - a list of documents **D** of length n, and
   - a dictionary, **Q**.

   Your algorithm should go through each document in $\mathbf{D}$ and compute the DSD to each other document (you may reference your algorithm from question 5 here - there is no need to describe this logic again). The algorithm should record the index of document that was the closest match to each document (excluding itself) within $\mathbf{D}$ and return a list of these indices.

   For example, given $\mathbf{D} =< d_1, d_2, d_3 >$, suppose the closest match to d_1 was d_2, the closest match to d_2 was d_3 and the closest match to d_3 was d_2. The output would therefore be $< 2, 3, 2 >$ (hint: remember that we index from 1 in formal descriptions but index from 0 in code). **(12 marks)**

7. Analyse the run-time complexity of your algorithm in question 6 to find the closest documents in a list of documents. **(12 marks)**

8. Implement your algorithm from question 6 in Java to find the indices of the best matches to each document in an input list of documents. Your method should be called `findNearestDocuments` and the inputs to your method should should be:

   - a `String[][]` to store a list of documents, and
   - a `String[]` to store a dictionary.

   Your method should return a single integer array. **(12 marks)**

9. Design and conduct timing experiments using your implementation from question 8. Present your results on a graph and briefly comment on your results and how they compare to your expectations following your run-time complexity analysis. **(12 marks)**

   Note: again, all code to generate your experimental results should be included in your submission (with informative comments) to demonstrate how you designed and performed your experiments. You are welcome to reuse the utility methods provided for question 4 to assist with your experiments.

Relationship to formative work

Please see the following lectures and lab exercises for background on each of the specified tasks:

- **Describing algorithms**: the lectures in week 2 introduced informal and formal descriptions of algorithms. The lab in week 2 included a tutorial and exercises on writing formal descriptions of algorithms, and this has been reinforced with further examples throughout the lectures and labs.

- **Analysing algorithms**: the lectures in week 3 and 4 introduced the concept and structure of analysing algorithms. Further examples were included in the lab in week 4, and an additional optional seminar sheet (with solutions) has been uploaded to blackboard for further practice.

- **Timing experiments**: some informal timing experiments have been shown in the lectures, but see the lab sheets in weeks 3 and 4 for specific exercises to practice creating timing experiments for running code in Java.
- **Algorithm design**: this has been taught throughout the prerequisite modules for this module. It was further reinforced in the second part of the lecture in week 4 with a case study of designing an algorithm for a specific problem, starting with an inefficient solution and iterating through more efficient solutions.

Deliverables

You must submit a single .pdf electronically on e:vision that contains all of your work. This .pdf should be prepared using the PASS application that is installed on all UEA CMP lab machines. Please ensure that you include answers to all questions in the assignment that you have attempted **and also include all source code used to implement and run your experiments**. Your code should be clear and include comments to aid the marker where necessary (a full javadoc is not necessary, however).

It is acceptable to do your analyses on paper by hand if that is easier for you, but make sure to scan them, save as a .pdf, and include them in your final .pdf. Please do not use spaces in any file names that you include in your PASS submission. `aFileName.pdf` would be fine, but `a File Name.pdf` will not work correctly with PASS.

An example of using PASS to format this coursework with instructions will be provided on Blackboard nearer the submission date to help anyone who is unfamiliar with the system.

Finally, **please make sure that everything in your submission is clear and legible**. If anything is not clear and visible in your final .pdf then we cannot mark it. It is your responsibility to make sure that the .pdf that you submit includes all of your work and that it is a fair representation of the effort that you have put into this assignment.

Resources

- **Previous exercises**: If you get stuck when completing the coursework please revisit the lab exercises that are listed in the *Relationship to formative work* section during your allocated weekly lab sessions. The teaching assistants in the labs will not be able to help you with your coursework directly, but they will be more than happy to help you understand how to answer the (very) related exercises in the lab sheets. You will then be able to apply this knowledge and understanding to the new problems in this coursework assignment.
- **Discussion board**: if you have clarification questions about what is required then please ask these questions on the Blackboard discussion board. This will enable other students to also benefit from the question/answer. Please check that your question has not been asked previously before starting a new thread.
- **Course text:** Goodrich, M. T., Tamassia, R. (2005) *Data Structures and Algorithms in Java*, 4th edition. As mentioned in lectures, this text book is very useful and this older edition is freely available online if you do a quick search for it. Chapter 4 of this edition in particular will be helpful for analysis of algorithms, but any edition of the book will include helpful information.

Marking Scheme

Itemised marks are provided throughout the assignment description. To summarise:

1. **Part 1: Calculating Feature Vectors (40 marks)**
    1.1. Design and description (10 marks)
    1.2. Analysis (10 marks)
    1.3. Java implementation and testing (10 marks)
    1.4. Timing experiments and discussion (10 marks)
2. **Part 2: Comparing Documents (60 marks)**
    2.1. Design and description of DSD (12 marks)
    2.2. Design and description of finding closest documents (12 marks)
    2.3. Analysis of finding closest documents (12 marks)
    2.4. Java implementation and testing (12 marks)
    2.5. Timing experiments and discussion (12 marks)

Total: 100 Marks

Further information:

- **Formal descriptions**: marks will be awarded for the correctness of your algorithms and your ability to describe them accurately in pseudocode. Make sure that you clearly describe all elements of your algorithm and use comments where appropriate.
- **Algorithm analyses**: marks will be awarded for correctness of your evaluation and for following the correct procedures when analysing algorithms. Using LaTeX/Overleaf for writing up your analysis is encouraged, but it is acceptable to do them by hand/other means instead if you prefer (such as scanning analyses done by hand and including them in your final submission). If you do this however then it is your responsibility to make sure your writing/presentation is clear and legible - we cannot give you marks if we cannot understand it!
- **Implementation**: you should submit your Java code for the required implementation questions and marks will be awarded for correctness and comprehensibility of your code. This includes using standard programming conventions (such as correct indentation, sensible variable names, etc.) as well as informative and reasonable comments to explain complex code. There is no need to define a full javadoc but there should be helpful comments where necessary to make it easy for a reader to understand your code.
- **Timing experiments**: marks will be awarded for experimental design, presentation of results, and discussion of results. Make sure that you perform many resamples using various input sizes when conducting your experiments, and make sure to present results using clear, well-formatted graphs. As mentioned in the question, you should also briefly discuss your results in terms of whether they agree with your runtime complexity analysis. This does not have to be a long discussion but it should demonstrate your understanding of how your practical results and algorithm analyses are related.