CSE2ALG / CSE5ALG: Algorithms and Data Structures

Assignment Part 1, 2021

Department of Computer Science and IT, La Trobe University

Assessment

This part of the assignment is worth **15%** of the final mark.

Deadline

13 April 2021, Tuesday, at 10:00 AM.

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. You must plan to allow for both scheduled and unscheduled downtime. **Late submission policy does NOT apply to this assessment**.

As this part of the assignment is worth **15% or more** of the final mark for this subject, please apply for **Special Consideration** if you need request an extension of time for submission due to any unforeseen circumstances that substantially affect your ability to complete it on time. More details can be found via the following link (https://www.latrobe.edu.au/students/admin/forms/special-consideration). **Once your application is approved by the University, the subject coordinator (or the lecturer) will grant you an extension of time for submission**.

Plagiarism

Plagiarism refers to submitting somebody else's work in a manner that gives the impression that the work is your own, which is an **academic misconduct**. The CSIT Department treats academic misconduct seriously. When it is detected, penalties will be strictly imposed.

Submission

You are supposed to submit all the files that are required for the tasks described in this instruction, from your **latcs8** account. Please make sure you have navigated to in the same directory where the files being submitted are. You must submit each file separately using the submit command **`submit ALG`**. For example, if the file is named *LexiconTester.java*, please use the following command for submission:

```
submit ALG LexiconTester.java
```

After submitting the files, you may run the following command that lists the files submitted from your account:

```
verify
```

You can submit the same filename as many times as you like before the deadline. The previously submitted file will be replaced by the latest one. If you encounter any problem regarding submission, please email the lecturer for assistance.

Platform

While you are free to develop the code on any operating system, **your solution must compile and execute using `javac` and `java` commands on the latcs8 server.**

Return of Result

The lecturer will mark your submission with a marking sheet during the next face-to-face / online lab classes after the deadline (i.e., 13 and 16 April 2021). You will be notified with your mark soon after marking. If you have any doubt, please immediately raise it to the lecturer before the lab class ends. **Any post-lab inquiry will NOT be accepted**. If you cannot attend the lab classes, please email the lecturer for an alternative arrangement.

Restriction

You are **NOT** allowed to use **ANY** of the classes in the **Java Collections Frameworks** (JCF); the only exception is the *ArrayList* class.

More details about the JCF can be found via the following link (https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html).

Violating this restriction will lead to a mark of ZERO for this part of assignment.

Marking Criteria

You mark for this part of the assignment relates to the program's capability (i.e., whether being compiling and executing), the code's quality (i.e., whether being reasonable and in good style), and the submission's accomplishment (i.e., whether satisfying the task requirements).

Details can be found in the **example marking sheet**.

Description of Tasks

In many situations, we may need to find a **word** based on one or more letters in it. For example, when making or completing a crossword you may want to find a word that has 4 letters, starts with *J* and ends with *A*. In this assignment, the program you will write will create a lexicon of words from various text files.

A **word** is a sequence of letters. Words will be treated in a case-insensitive manner. A simple option is to store all the words in lower case. Initially, a word is obtained as a sequence of characters separated by whitespace characters. Then numbers and punctuation characters, if any, must be removed. More specifically,

- If the word “and” occurs twice in the file, it is only stored once in the lexicon.
- If the word “you” also occurs as “YOU”, only one version is stored.
- Punctuation, such as commas, quotes, hyphens, full stops and question marks, are removed.
- If the word “don’t” or the word "second-hand" appears in the input texts, they will be stored without punctuation (i.e., “dont” and “secondhand” respectively).
- The ‘word’ “1”, for example, will be ignored as it contains numeric characters.

For each word, we also keep the following information:

- The frequency: How many times the word appears in the input files.
- The list of neighbors: A **neighbor** of a word *w* is a word that is of the same length and differs from *w* by only one letter. For example, if the word *w* refers to “cat”, then:
    - “cathy” is not *w*’s neighbor, for they are not of the same length.
    - “man” is not w’s neighbor, for they differ by more than one letter.
    - “can” is a neighbor of *w*.
    - “fat” is a neighbor of *w*.
    - “cot” is a neighbor of *w*.

Task 1 Design and implement the program *LexiconTester.java*. This program will

1. Read two text files, *in1.txt* and *in2.txt* (i.e., filenames are hard-coded), and construct a lexicon that contains words from the two files.
2. Write the words from the lexicon to the text file *out.txt*. Specifically, these words must be in sorted in alphabetical order.

The information on each word, which is also written to the text file, includes the frequency and the list of its neighbors. The list of neighbors must also be in alphabetical order.
To output each word, first the spelling of the word is displayed, followed by the frequency, and then followed by the word's neighbors (in alphabetical order, inside a pair of square brackets). The format must be as shown in the example below, which is for some words from *in1.txt* and *in2.txt* given as test files.

```
a 11 [i]
about 1 []
acknowledged 1 []
all 1 []
also 1 []
and 6 []
answer 1 []
at 2 [it]
austen 1 []
be 2 [by, he, me]
been 1 []
bennet 3 []
but 1 []
by 1 [be, my]
can 1 [cat, man]
cat 1 [can, fat, hat, mat]
… (not finished)
```

3. Include **two** sorting algorithms (i.e., a default algorithm and an alternative algorithm). It is entirely up to you which two sorting algorithms will be included, as long as they are in the subject and different regarding **time complexity**.
   Note that the program can toggle between the two sorting algorithms using the first command-line argument. Assume "Y" for the alternative algorithm, and otherwise for the default one.

Task 2 Write a report *LexiconReport.pdf* in the PDF format. This report will

1. Describe which two sorting algorithms are selected, and the reason for your choice.
2. Describe the difference of the two algorithms regarding time complexity (i.e., the Big-Oh) in the best, worst and average cases.
3. Describe the difference of the two algorithms regarding time usage (i.e., how many seconds) for generating the lexicon in Task 1.

Note Submitting additional Java files as auxiliaries is allowed, as long as these files are created originally by you and can compile on the latcs8 server.

CSE2ALG / CSE5ALG Assignment Part 1: Marking Sheet

Semester 1, 2021

Student Name		Student Number	

	Task 0: LexiconTester.java		
		Mark	Max
1	Use any of the classes in the JCF, other than the ArrayList class.		-100
Task 1: LexiconTester.java (Max Mark: 80)			
Section 1: Blackbox Testing (Max Mark: 50)			
2	**javac LexiconTester.java** No error occurs when the program compiles. If other java files are submitted, then all files must compile.		10
3	**java LexiconTester** No error occurs when the program runs, and *out.txt* is created.		8
4	**java LexiconTester Y** No error occurs when the program runs, and *out.txt* is created.		7
5	In *out.txt*, words and neighbours are all available and ascendingly sorted. Each exception results in deduction of 1 mark.		10
6	In *out.txt*, word frequencies are all available and calculated correctly. Each exception results in deduction of 1 mark.		7
7	In *out.txt*, neighbours are all available and selected correctly. Each exception results in deduction of 1 mark.		8
Section 2: Code Checking (Max Mark: 30)			
8	Two sorting algorithms, different regarding time complexity, are included.		16
9	Code is reasonably written (i.e., it contains necessary classes and functions, indicating the author made efforts to accomplish).		7
10	Code is in a good programming style (i.e., it helps the lecturer to read and understand code and to avoid introducing errors).		7
	Task 2: LexiconReport.pdf (Max Mark: 20)		
11	The report describes the selected algorithms and the reason.		7
12	The report describes the time complexities of the two algorithms.		7
13	The report reports the time usages of the two algorithms.		6
Total			100