CSE2/CSE5ALG– Algorithms and Data Structures – 2019 Assignment – Part 1

Assessment: This part 1 of the assignment is worth 10% of the final mark for this subject.

Due Date: Friday 3rd May 2019, at 10:00 AM

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime. Penalties are applied to late assignments (accepted up to 5 days after the due date only).

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Computer Engineering treats academic misconduct seriously. When it is detected, penalties are strictly imposed.

Submission Details: Submit all the Java files that are required for the tasks described in this handout. The code has to run under Unix on the latcs8 machine. You submit your files from your latcs8 account. Make sure you are in the same directory as the files you are submitting. Submit each file separately using the submit command. For example, for the file called (say) LexiconTester.java, use command:

```
submit ALG LexiconTester.java
```

After submitting the files, you can 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 assignment deadline; the previously submitted copy will be replaced by the latest one.

Testing Platform: While you are free to develop the code for this assignment on any operating system, your solution must run on the latcs8 system. We should be able to compile your classes with the simple command `javac *.java`, and execute your programs with a simple command, e.g. `java LexiconTester`.

Return of Assignment: Assignments will be returned within three weeks of the due date. Students will be notified by email and via the CSE2/CSE5ALG website when marking sheets are available for collection.

Assignment Objectives

- To understand various data structures and searching and sorting techniques;
- To analyse these techniques in relation to a particular problem;
- To implement these techniques within a Java program.

Problem Description

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 sources (that is, from various text files) and will allow the user to search for words that fit particular patterns.

The operational definition of what is taken to be a word will be given later in this handout. 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

  Note: The motivation for this requirement is to make the lexicon capable of supporting the following game. Two words are given – let us called them "start word" and "end word". The aim is to transform the start word into the end word by changing one letter at a time, subject to the condition that all the intermediate "words" are valid words. Obviously, this game is equivalent to finding a path from the start word to the end word along the edges connecting two neighbors.

Assignment Requirements

The assignment is divided into two parts. For part 1, you are required to write a simple version, where execution efficiency is not taken into account. For part 2, you are required to write an "optimized" version where efficiency, in addition to correctness, is the major concern.

With the exceptions of ArrayList, you are not permitted to use any of the classes in the Java Collections Framework, e.g. TreeMap, HashMap, Arrays.

Task 1

For this task, you are to design and implement the program `LexiconTester.java`. This program will

1. Read two text files, `sample1-pp.txt` and `sample2-zoo.txt`, and construct a lexicon that contains words from the two files;
2. Write the words from the lexicon to the text file `sample3-wordlist.txt`.

Further information about the task is given below.

Building the lexicon

The first text file to be read in is the file `sample1-pp.txt` shown below:

```
Pride and Prejudice
by Jane Austen
Chapter 1
It is a truth universally acknowledged that a single man in possession of a
good fortune must be in want of a wife.
However little known the feelings or views of such a man may be on his first
entering a neighbourhood, this truth is so well fixed in the minds of the
surrounding families, that he is considered the rightful property of some one
or other of their daughters.
"My dear Mr.  Bennet," said his lady to him one day, "have you heard that
Netherfield Park is let at last?"
Mr.  Bennet replied that he had not.
"But it is," returned she; "for Mrs.  Long has just been here, and she told
me all about it."
Mr.  Bennet made no answer.
"Do you not want to know who has taken it?" cried his wife impatiently.
"YOU want to tell me, and I have no objection to hearing it."
This was invitation enough.
```

The second text file to be read in (and its words added to the lexicon) is the file `sample2-zoo.txt` below:

```
You can see zebras and yaks at the zoo.
Yes, zebras and yaks and wombats too.
You will also meet a fat cat, wearing a hat, sitting on a mat, reading a map
while drinking from a cup.
```

What is a word?

A word is a sequence of letters. Words will be treated in a case-insensitive manner. A simple option is to stored 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 would be stored without punctuation, i.e. `"dont"` and `"secondhand"` respectively.
- The 'word' `"1"`, for example, will be ignored as it contains numeric characters.

Saving the lexicon

The words from the created lexicon are to be written to file `sample3-wordlist.txt`. 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.

The format must be as shown in the example below for a number of words.

```
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]
chapter 1 []
considered 1 []
cried 1 []
cup 1 []
...
```

For 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).

Task 2

For this task, you are to write the program `WordMatchTester.java`. This program will

1. Read two text files, `sample1-pp.txt` and `sample2-zoo.txt`, and construct a lexicon that contains words from the two files (as in Task 1);

2. Find the words that match a number of patterns, and display the results (both the patterns and matching words) on the screen and write them to the file `sample4-results.txt`.

   The test cases (patterns) should be carefully chosen, and the reasons for including them must be documented in the program.

Further information about the task is given below.

Finding matching words

The patterns to match the words in the lexicon are to be included (hard-coded) as strings in the test program. A pattern consists of a sequence of letters, with no spaces within the sequence. In addition, there are two wildcard characters allowed. A `'?'` symbol in the pattern can match with any one character in a word, while a `'*'` symbol can match with any number of characters in a word (zero or more).

For each pattern, the pattern and the words that match the pattern are to be displayed on the screen and write to the text file. Each matching word, together with its frequency, appears on a separate line. The words are in alphabetical order.

For example, the pattern:

```
ma?
```

may result in the words below being displayed to the screen

```
man 2
map 1
mat 1
may 1
```

The pattern:

```
?o?
```

may result in the following words being displayed. Note the words are displayed in lexicographic order.

```
for 1
not 2
too 1
you 5
zoo 1
```

The pattern:

```
Mr*
```

may result in the output:

```
mr 3
mrs 1
```

The pattern:

```
i*
```

may result in the output:

```
i 1
impatiently 1
in 3
invitation 1
is 5
it 5
```

The pattern

```
?ear*
```

may result in the output:

```
dear 1
heard 1
hearing 1
wearing 1
```

Any pattern that has no matches will result in the message

```
No words in the lexicon match the pattern
```

output to the screen and written to the file.

The patterns should be designed for a comprehensive testing of the correctness of what you have implemented. As a suggestion, you should include at least 10 test cases (i.e. patterns). They must be hardcoded into the program, and the reasons for including them must be *documented* in the program.

A suggestion

It is up to you to decide on how to organize your code. However, to make your code easier to write and to test, you may want to have a class called `Helper`, and in this class you may put methods such the one to determine if two strings are neighbors of each other, etc.

Marking Scheme Overview

- The program (classes) for Task 1 will be marked on correctness, and coding style and comments.
- The program for Task 2 will be marked on correctness, coding style and comments, and the matching patterns included for the testing.
- 10% of the total mark of part 1 of the assignment will be allocated to coding style and comments: Do the programs follow good design and programming practices? Do the indentation and code layout follow a good, consistent standard? Are the identifiers meaningful? Are comments being used effectively?

■