CSC 225 - SUMMER 2018
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 3
UNIVERSITY OF VICTORIA

Due: Sunday, August 5th, 2018 before 11:55pm. **Late assignments will not be accepted.**

1 Assignment Overview

Consider the following list L of English words.

$L =$ `bash, bath, cans, cape, caps, case, cash, cats, dame,`
`date, dice, dime, fail, fate, hats, haze, hazy, late,`
`lazy, mace, mail, male, mall, maps, mare, mars, math,`
`mats, maze, mice, pans, pass`

A **word ladder** is a sequence of words such that each word in the sequence differs from the previous word in exactly one position. For example,

`mice` → `mace` → `mare` → `mars` → `mats` → `cats`

is a word ladder from `mice` to `cats`. For given start and end words, there may be multiple word ladders in a particular word list. For example, both of the below sequences are word ladders from `math` to `cash`.

`math` → `mats` → `cats` → `caps` → `cape` → `case` → `cash`

`math` → `bath` → `bash` → `cash`

Your task for this assignment is to write a program which reads a list of words from a text file, then finds a **shortest** word ladder between two words provided as command line arguments. As noted above, there may be multiple word ladders (even multiple word ladders of the minimum length) between the provided pair of words. Your program may generate any of the various possibilities in such cases (as long as the ladder is valid and has minimum length).

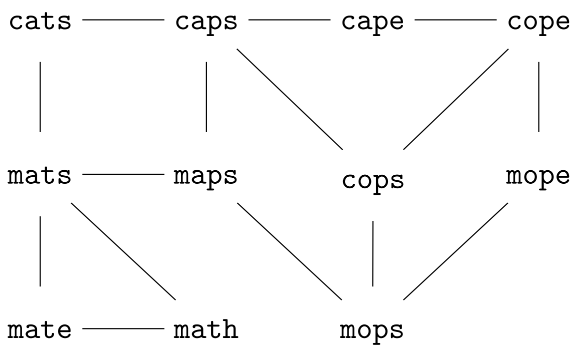
1.1 Graph Interpretation

One way to identify word ladders in a list of words is to model the list of words as a graph, with each word corresponding to a vertex and edges connecting pairs of words that differ in exactly one letter.

For example, consider the list of 11 words below.

$L =$ `cape, caps, cats, cope, cops, mate, math, maps, maps,`
`mope, mops`

The following 11 vertex and 15 edge graph models the relationships between the words in L.



A word ladder between two words is equivalent to a path between the two corresponding vertices in the graph, and the fundamental traversal algorithms can therefore be used to find word ladders.

1.2 `Ladder` program interface

Your `Ladder` program will accept command line arguments in the form

```
$ java Ladder <word list file> <start word> <end word>
```

where `<word list file>` is the path to a text file containing words and the `<start word>` and `<end word>` arguments are words. Note that the two provided words are not required to be present in the text file (although if they are not present, no word ladder will exist). The specification for the input word list file is given in Section 1.3.

If a valid word ladder is found between the two words, the program will print the word ladder, with one word per line, including both the start word and end word. For example, if the program is invoked with the command line

```
$ java Ladder some_words.txt cats mops
```

where `some_words.txt` contains the word list given in Section 1.1, the program could produce any one of the following three word ladders (all of which are of minimum length). There is no requirement for the program to find or output all of the different possibilities.

Ladder 1:

```
cats
caps
cops
mops
```

Ladder 2:

```
cats
caps
maps
mops
```

Ladder 3:

```
cats
mats
maps
mops
```

If the program does not find a valid word ladder (of any length) between the two provided words, the output will be the message "`No word ladder found.`". In both cases (whether or not a word ladder is found), the program must produce **no other output** besides what is specified in this section.

1.3 Input Format and Constraints

The word list file provided to the `Ladder` program will contain a list of words with one word per line and any blank lines (as well as any other incidental whitespace) completely ignored. As a result, it is impossible for the empty string `""` to be a valid element of any word list. However, the word list file may contain blank lines (or spaces at the beginning and end of lines). Any blank lines must be ignored, and all leading or trailing whitespace in each line must be removed before handling the word (you may want to use the `trim()` method of a Java String to achieve this).

The length of the word list is not bounded (so you may not assume that it will have any particular maximum length). However, the following restrictions apply to each word in the list. Note that you may **assume** that all of these constraints hold, and you are not required to verify any of them at run-time (any input which does not meet the criteria below is considered invalid and will not be used as a test case for your program).

- A particular word will never appear multiple times in the same file.
- Words will contain **only** the lowercase letters `a` through `z`. No uppercase letters or non-letter characters (including whitespace, hyphens or other punctuation) are permitted.
- The length of any word inside the word list is guaranteed to be at most 100 characters. Therefore, you may assume that any operation which is applied to a single word (or a pair of words) runs in constant time. In particular, you may assume that comparing two words is a constant time operation.

Some starter code, which parses the command line arguments and reads the word list into a Java `LinkedList` structure, has been posted to conneX. You are not required to use this starter code, but since it includes some input validation (such as checking for non-letter characters), you may at least want to consult it when writing your solution.

2 Running Time Requirements

As in previous assignments, the primary goal of any solution to this assignment must be a valid and correct implementation, regardless of running time. You will not receive any marks for running time unless your solution is correct (since it is trivial to produce a fast but non-working program).

If your solution uses the graph model described in Section 1.1, it will likely follow the steps below.

- Read the word list into an array or linked list.
- Build a graph by using the word list as a vertex set and connecting two words which differ in one position by edges.
- Run a traversal algorithm (probably BFS) to find a word ladder.

If the graph has n vertices and m edges, it should be possible to achieve a running time of $O(n+m)$ for the traversal algorithm (although care must be taken to ensure that the data structures used by the traversal do not have any hidden complexity, as in previous assignments). Although the number of edges m can be $O(n^2)$ for general graphs, it is unlikely to be so large in the graphs used to find word ladders, so for most inputs it is reasonable to expect that $O(n + m)$ is significantly less than $O(n^2)$.

However, the most straightforward method to construct the graph (which simply compared every word to every other word to identify words which are adjacent) requires $O(n^2)$ time. You will likely find that the most difficult part of the assignment is reducing the complexity of the graph construction rather than speeding up the traversal algorithm.

To receive full marks, the **entire program** must be correct and achieve a worst-case running time of $O(n+m)$, where n is the number of words in the word list and m is the number of edges produced by the construction described in Section 1.1 (which may not necessarily be the number of edges in the graph you construct, since you are free to use any construction you want). If this running time cannot be achieved, your submission will still receive a very high mark if the running time of the entire program is $O((n + m)\log_2(n))$ in the worst case. Finally, if you are only successful in implementing the basic construction algorithm, but your program is still correct, you may receive up to 10/14 if your entire program has a worst-case running time of $O(n^2)$.

3 Using Outside Code

You are encouraged to use the features of the Java Standard Library (including any of the data structures it provides) in your code. If you use a standard library data structure, make sure you are aware of the running times of the operations you use, since that will be important for determining the running time of your program.

There should be no need to use large volumes of code from other sources (such as outside libraries or the internet) in this assignment. If you believe that your implementation requires an outside library, talk to your instructor.

If you find a small snippet of code on the internet that you want to use (for example, in a StackOverflow thread), put a comment in your code indicating the source (including a complete URL). Remember that using code from an outside source without citation is plagiarism.

You are encouraged to discuss the assignment with your peers, and even to share implementation advice or algorithm ideas. However, you are not permitted to use any code from another CSC 225 student under any circumstances, nor are you permitted to share your code in any way with any other student (or the internet) until after the marking is complete. Sharing your code with others before marking is completed, or using another student's code for assistance (even if you do not directly copy it) is plagiarism.

4 Evaluation

Submit all `.java` files needed to compile your assignment electronically via conneX. Your code must compile and run correctly in the Linux environment in ECS 242. If your code does not compile as submitted, you will receive a mark of zero.

This assignment is worth 7% of your final grade and will be marked out of 14 with a combination of automated testing and human inspection. This assignment will not be evaluated by interactive demos.

The marks are distributed among the components of the assignment as follows.

Marks	**Component**
8	The requirements in this document are met and the output of the program is correct on a variety of tested inputs.
2	On an input containing n words, where the resulting graph has m edges, the asymptotic worst case running time of the entire program is $O(n^2)$.
2	The asymptotic worst case running time of the entire program is $O((n+m)\log_2(n))$.
2	The asymptotic worst case running time of the entire program is $O(n+m)$.

You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions or resubmissions will be accepted after the due date has passed. You will receive a mark of zero if you have not officially submitted your assignment (and received a confirmation email) before the due date.

Your main program must be contained in a file called `Ladder.java`. You may use additional files if needed by your solution (as long as the program can be invoked from the command line using the syntax in Section 1.2). Ensure that each submitted file contains a comment with your name and student number.

Ensure that all code files needed to compile and run your code in the Linux environment in ECS 242 are submitted. Only the files that you submit through conneX will be marked. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. **If you do not receive such an email, you did not submit the assignment.** If you have problems with the submission process, send an email to the instructor **before** the due date.