EE422C Fall 2015
Project 3: Word Ladders
Due Monday Oct 5, 2015

You are permitted to work in teams of two for this project. You may choose to work individually if you wish. If you work as a team of two, both team members should submit the project into their student directories (i.e., the project will be submitted twice into SVN) and all of the project source files MUST have the names and UTEIDs of both students inside comments at the top of the file. There will be no exceptions to this policy on team projects. Collaborating on the project and failing to follow these instructions and will be treated as a violation of academic honesty.

The aim of this assignment is to give you experience working with various collections, as well as to strengthen your algorithm and ADT design skills.

Problem Statement:

A *word ladder*[1] is a (finite) sequence of distinct words from the English language such that any two consecutive words in the sequence differ by changing one letter at a time with the constraint that each of the resulting string of letters is a legitimate word. For example, to turn "stone" into "money", one possible word ladder is:

stone
Atone
aLone
Clone
clonS
cOons
coNns
conEs
coneY
Money

Capital letters are used in the example above only to illustrate the connections. Obviously, there could be more than one word ladder between "stone" and "money". You only have to find one of them for each word pair given.

In this assignment, you are to design and implement a Java program that for any given pair of words, it generates a word ladder that connects those two words (making use of a given dictionary of legal English words). If a word ladder does not exist between the given pair, your program should output a message that says so. You are required to find a word ladder, not necessarily the shortest word ladder.

Input and Output Requirements

Your program will read commands from the standard input (i.e., from the keyboard). The basic command consists of a pair of words separated by at least one space (with no intervening punctuation

[1] http://www.learnenglish.org.uk/words/activities/revers01.html

or other words). After reading both words, your program must determine if there is a word ladder between the two words. For example, the command

smart money

instructs your program to search the dictionary (the dictionary is described below) to find a word ladder that starts with "smart" and ends with "money". If a word ladder can be found, your program must print the message

a <N>-run word ladder exists between <start> and <finish>
<start>
<first rung>
<second rung>
<…>
<finish>

Where <N> must be replaced with the number of intervening words in the word ladder between the start and finish words (i.e., don't count start or finish in your calculation of N). You must then print each of the words in the word ladder on a line by themselves, and each indented by one tab position. For the command "smart money", your program might produce the following

a 8-rung word ladder exists between smart and money
smart
start
stars
soars
socks
cocks
conks
cones
coney
money

Note that to be a valid word ladder, every word in the ladder must be a legal word that appears in the dictionary. If your program cannot find a valid word ladder between the two words, you must print

no word ladder can be found between <start> and <finish>

Finally, if either the starting word or the finish word are not in the dictionary, or if the start and finish are the same word, then your program should treat that as if no word ladder could be found. This case is intended to include situations where the start and/or finish word are not valid words. For example, the command "sm@rt m0n3y" should print

no word ladder can be found between sm@rt and m0n3y

In addition to the basic command, your program must recognize additional commands that start with the / character (i.e., the forward slash). The command /quit must result in your program terminating with no further output. Additional commands may be added at a later date. For now, only the /quit

command is legal. If you see any other command that begins with a / character, you must print the message

invalid command <txt>

where <txt> is the actual command read from the input. For example, if the command /stop were entered, your program must print

invalid command /stop

After printing the invalid command error message, your program must resume reading commands from the input stream. Note that whitespace including tab characters and newline characters is to be ignored (treated like spaces) when you are reading the input. This whitespace policy applies for any commands (including basic commands) read from the standard input.

Dictionary: You may test your project with the dictionary contained in the file named "assn3words.dat"[2], which is a text file that consists of a collection of English words with five letters each. You are to use this file to create your dictionary of five-lettered words. The format of the file is that each line starts with either the character '*' or a five letter word. You can ignore the lines that start with '*'. For each of the other lines, you are to extract the five-lettered word at the beginning of the line and add that word to your internal dictionary (while ignoring the rest of the line). Clearly, using this dictionary, it will only be possible to find word ladders when the starting word and the finishing words are both five-letters long.

Submission requirements

Create a package named ***project3*** for all of your source code. Create a public class inside package project3 called Main.java that contains ***main ().*** Remember to put all both team member names and UTEIDs on the .java files if you are working as a pair. Add all .java files to the svn repo and commit your final version of those files before the submission deadline.

Additional Considerations

- External Code – you are permitted to use any classes or interfaces within the java.lang, java.io, and java.util standard packages. You may use other 3rd-party code only with permission of the instructor. You are specifically prohibited from making use of another student's code (including students who may have taken the class in previous semesters).
- Understandability – Comment your program so that its logic would be readily apparent to any software engineer who is familiar with standard data structures and algorithms.
- Re-use – Design your code so it is suitable for future adaptations and/or expansion.
- Efficiency Risk – It is possible that additional problem/solution constraints may be needed in order to guarantee that your program runs in a reasonable amount of time and/or space. These maybe specified later.
- Extra Credit – I'd really like to include additional slash-commands that constrain (or relax constraints) on rungs in the word ladder. For example, rungs in the ladder could potentially be created by adding or removing letters from words (stop -> top by removing the 's'), or rungs

[2] This collection was compiled by Don Knuth and is a part of the Stanford Graphbase. We are using the file as-is since it may not be modified.

could be allowed or disallowed between words where the capitalization is different (chase -> Chase by capitalizing the ‘c’). I’d also really like to remove the restriction that the project only work with dictionaries consisting of 5-letter words. If I deem such additional features are reasonable, I may add them as extra credit opportunities for the project.