### **Overview:**

|  |  |
| --- | --- |
| stone  **A**tone  a**L**one  **C**lone  clon**S**  c**O**ons  co**N**ns  con**E**s  cone**Y**  **M**oney | Word ladders were invented by Lewis Carroll in 1878, the author of *Alice in Wonderland*. A ladder is a sequence of words that starts at the starting word, ends at the ending word, In a word ladder puzzle you have to change one word into another by altering a single letter at each step. Each word in the ladder must be a valid English word and must have the same length. For example, to turn stone into money, one possible ladder is given on the left.  Many ladder puzzles have more than one possible solution. Your program must determine the shortest word ladder. Another path from stone to money is  stone store shore chore choke choky cooky cooey coney money **Objectives**  * Practice implementing and using a Queue data structure. * Gain an understanding of the algorithms used for efficient implementation. * Practice working in small teams to solve problems, design algorithms and write code  **Instructions** Your program will accept starting and ending words from the input file called ["infile.txt"](https://drive.google.com/open?id=13RVL6PNfGP8V69_yIX27bNcLoLKFxId6). Then, you read the dictionary file called [“dictionary.txt ''](https://drive.google.com/open?id=13WF_uoOVuOdJ79gGuNpZToRUCbZwHdgx) store all words in a [LinkedList](https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html). Finally, you build a word ladder between starting and ending words  There are several ways to solve this problem. One simple method involves using stacks and queues. The algorithm (that you must implement) works as it follows  Get the starting word and search through the dictionary to find all words that are one letter different. Create stacks for each of these words, containing the starting word (pushed first) and the word that is one letter different. Enqueue each of these stacks into a queue. This will create a queue of stacks! Then dequeue the first item (which is a stack) from the queue, look at its top word and compare it with the ending word. If they equal, you are done - this stack contains the ladder. Otherwise, you find all the words one letter different from it. For each of these new words create a deep copy of the stack and push each word onto the stack. Then enqueue those stacks to the queue. And so on. You terminate the process when you reach the ending word or the queue is empty.  You have to keep track of used words! **Otherwise, an infinite loop occurs.** **Example** The starting word is *smart*. Find all the words one letter different from smart, push them into different stacks and store stacks in the queue. This table represents a queue of stacks.  ---------------------------------------------------- | scart | start | swart | smalt | smarm | | smart | smart | smart | smart | smart | ----------------------------------------------------  Now dequeue the front and find all words one letter different from the top word *scart*. This will spawn seven stacks:  --------------------------------------------------------------------- | scant | scatt | scare | scarf | scarp | scars | scary | | scart | scart | scart | scart | scart | scart | scart | | smart | smart | smart | smart | smart | smart | smart | ----------------------------------------------------------------------  which we enqueue to the queue. The queue size now is 11. Again dequeue the front and find all words one letter different from the top word *start*. This will spawn four stacks:  ---------------------------------------- | sturt | stare | stark | stars | | start | start | start | start | | smart | smart | smart | smart | ----------------------------------------  Add them to the queue. The queue size now is 14. Repeat the procedure until either you find the ending word or such a word ladder does not exist. Make sure you do not run into an infinite loop! **Queue** You are to implement a Queue data structure based on Java’s [Queue<E>](https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html) class using [LinkedList](https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html). **Stack** You are to use Java's [Stack class](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Stack.html). **Dictionary** You read the dictionary file from the course web page at <https://drive.google.com/open?id=13WF_uoOVuOdJ79gGuNpZToRUCbZwHdgx>  The dictionary file contains exactly one word per line. The number of lines in the dictionary is not specified. |

### **Output**

Your program must output to the console one word ladder from the start word to the end word. Every word in the ladder must be a word that appears in the dictionary. This includes the given start and end words--if they are not in the dictionary, you should print "There is no word ladder between ..." Remember that there may be more than one ladder between the start word and the end word. Your program may output any one of these ladders. The first output word must be the start word and the last output word must be the end word. If there is no way to make a ladder from the start word to the end word, your program must output "There is no word ladder between ..."

For testing purposes, I provide you with [the input file](https://drive.google.com/open?id=13RVL6PNfGP8V69_yIX27bNcLoLKFxId6) and [the expected output](https://drive.google.com/open?id=1Y-dUihqzlrcGy37kZhndGn6uYVfiFVQhK4Z0IKGE4qI).

### **Starter Code:**

Download the following file

* [WordLadder.java](https://drive.google.com/open?id=1fdz3uOMFm-6271Y8xG1l3gvViRFcmrxp)