**Word Ladder**

Word ladder puzzles were invented by Lewis Carroll, the author of *Alice in Wonderland*,in 1878. In a word ladder puzzle, you change one word into another by altering a single letter 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:

stone sto**r**e s**h**ore **c**hore cho**k**e chok**y** co**o**ky coo**e**y co**n**ey **m**oney

Many puzzles have multiple solutions – your goal is to write a program that will find the shortest ladder.

**Word Ladder**

Using the starting and ending words from the file called **"input.txt"**, your task is to write a program that will build a word ladder between the starting and ending words.

There are several ways to solve this problem - one method involves using stacks and queues. The algorithm works as follows:

*Get the starting word and search through the dictionary to find all words that differ by one letter. 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 - creating a queue of stacks! Next, dequeue the first item (which is a stack), look at its top word and compare it with the ending word. If they're equal, you are done - this stack contains the ladder.*

*Otherwise, find all words one letter different from it. For each of these new words create a 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 must keep the track of used words, otherwise an infinite loop occurs.

Example, using a starting word smart: First, find all 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 |

-----------------------------------------

Dequeue the front and find all words one letter different from the top word. 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 *more* 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. See the FAQ if you're thoroughly confused.

Using the **"dictionary.txt"** file, your program must output (to the console) one word ladder, from the start word to the end word (taken from the input file). Every word in the ladder must be a word that appears in the dictionary (this includes the given start and end words). Make sure to test your methods as you go – don't wait until you've coded the entire project to test that individual components work.

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 (always trying to find the shortest ladder, of course). 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...

Done correctly, your program should produce the output\* seen in the **"output.txt"** file. Note – depending on the speed of your overall algorithm, it could take some time to find the ladders for some word pairs.

*\*As stated previously, the actual order of the words in the ladder may be different depending on how you wrote your program. Your program SHOULD at least output the same number of words in the ladder, as it should always be finding the* shortest *ladder*

**(Optional) Read the dictionary from a website!**

Using a URL object, read the dictionary from the site "http://www.andrew.cmu.edu/course/15-121/dictionary.txt" and store into a HashSet. The link was working at the time of writing. You'll have to figure out how this is done on your own; the URL API may be helpful (as will a Scanner).

**(Advanced) Generate all possible word ladders**

Write a method public List<Stack<String>> buildLadder(String start, String end, int length)

in the WordLadder class that returns ALL ladders of a given length, not necessarily the shortest. For example, there are only two "shortest" ladders between sail and ruin:

sail, rail, rain, ruin

sail, sain, rain, ruin

However, there are 47 ladders of length 5. Here are some of them:

sail, mail, main, rain, ruin

sail, pail, pain, rain, ruin

sail, bail, rail, rain, ruin

sail, wail, rail, rain, ruin

sail, jail, rail, rain, ruin

If you discover that your program is running out of memory, 1) panic, 2) find a way to increase the [heap](https://en.wikipedia.org/wiki/Memory_management#HEAP) size of the Java Virtual Machine (JVM).

*Based on the project* ***Word Ladder***

[*http://www.cs.cmu.edu/~adamchik/15-121/labs/HW-4%20Word%20Ladder/lab.html*](http://www.cs.cmu.edu/~adamchik/15-121/labs/HW-4%20Word%20Ladder/lab.html)