Word Ladders

Thanks to Prof. Knuth from Stanford for the original idea

This is a quite challenging program using ideas we haven't discussed. I encourage you to take a stab at it – it's fun! One of the better ways to approach this problem is using a data structure called the queue. You can read more about queues at http://java.sun.com/javase/6/docs/api/java/util/Queue.html Notice that in Java, Queue is an interface. That means you can't actually create objects of type Queue but you can use classes that implement it. Check out http://java.sun.com/javase/6/docs/api/java/util/LinkedList.html

Also, reading words from a file multiple times can get very computationally expensive. You may want to use an array (http://java.sun.com/docs/books/jls/second_edition/html/arrays.doc.html)

Word ladders are connections from one word to another formed by changing one letter at a time with each resulting string being an English word. You will write a program to read in all roughly 5,800 five-letter English words and construct word ladders between any two typed in by a user. For example, to turn stone into money, one possible ladder is (replace 't' by 'h', replace 'o' by 'i', etc.): stone shone shine chine chins coins corns cores cones coney money

You'll probably want to start with a sample data file with, say, a dozen words including a pair you know creates a word ladder. 5,800 is a lot of records and may crash your program as you're developing!

Here is a sample run of your program:

```
Enter two 5-letter words: smart brain
smart
start
stark
stack
slack
black
blank
bland
brand
braid
brain
Enter two 5-letter words: angel devil
There is no path from angel to devil
Enter two 5-letter words: quit
```

Algorithm ideas

One way to solve this is as follows: put a starting word in a queue, then add all words 1 letter away from it. You'll also need to keep track of which word caused one to get on the queue by, for example, using an array. So in the previous example, when adding start to the queue, you would need to remember that it came from smart.   And when adding stark, you'll remember that it came from start which came from smart. You can work like this enqueueing words 2 letters away from the start, 3 letters away, etc… until you find the ending word.