

CS/IT 200

Lab 4: Word Ladders

Recommended Due Date: Thursday, October 8, 11:59pm

Submission

- Submit all code in a single zip file to the Lab 4 assignment folder on Kodiak.

Goals: Practice implementing and using the Queue data structure. Gain an understanding of algorithms used for efficient implementation.

Overview

Word ladders were invented by Lewis Carroll, the author of *Alice in Wonderland*, in 1878. In a word ladder puzzle, you try to go from the starting word to the ending word, by changing a word into another word 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 shown here:

stone atone alone clone clons coons conns cones coney money

Many ladder puzzles have more than one possible solution. Your program must determine a shortest word ladder. Another path from **stone** to **money** is:

stone store shore chore choke choky cooky cooey coney money

Part I – A new Queue class

Your first step is to implement a new Queue data structure. Instead of being built around a list or Nodes, this queue will work by manipulating two stacks.

You have been given `lab4_queue.py`, which creates the class and provides the header for each method you are expected to write. The constructor is already complete, though you may add other simple attributes if you desire. For each other method, replace the line containing `pass` with your own code.

Note: It may not be possible to get each method to run in $O(1)$ time.

Part II – Building word ladders

There are several ways to solve the problem of building a word ladder between given starting and ending words. One simple method involves stacks and queues. To help you, you've been given `wordlist`, a dictionary of words organized by their word lengths. For example:

```
wordlist[3] → ['cat', 'dog', 'the', 'cow', ...]      # All 3 letter words
wordlist[4] → ['that', 'this', 'rich', 'poor', ...]  # All 4 letter words
```

The algorithm (that you must implement) works as follows:

Get the starting word and search through the wordlist 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 are equal, you are done – this stack contains the ladder. (Return it!) Otherwise, you find all of the words one letter different from it. For each of these new words, create a deep copy of the stack (use `copy.deepcopy()`) 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.

As you go, you have to keep track of used words, otherwise an infinite loop occurs. Also, make sure to check that the start word and end word aren't the same!

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 smart	start smart	swart smart	smalt smart	smarm smart
----------------	----------------	----------------	----------------	----------------

Now dequeue the front and find all words one letter different from the top word **scart**. This will spawn seven stacks:

scant scart smart	scatt scart smart	scare scart smart	scarf scart smart	scarp scart smart	scars scart smart	scary scart smart
-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------

which we enqueue to the queue. The queue size is now 11. Again, dequeue the front and find all words one letter different from the top word **start**. This will spawn four stacks:

sturt start smart	stare start smart	stark start smart	stars start smart
-------------------------	-------------------------	-------------------------	-------------------------

Add them to the queue. The queue size is now 14. Repeat the procedure until you find the ending word or such a word ladder does not exist. Make sure you do not run into an infinite loop!

In `wordladder.py`, you will implement this algorithm in the `build_ladder()` function, whose header we have provided along with a few lines of code to get you started.

We have already provided you with `lab4.py`, which contains a `main()` function. This function reads our input examples and calls your `build_ladder()` function, outputting the returned ladder or saying that no ladder exists.

We have provided you with `input.txt` (the input file) and `output.txt` (the expected output). Note that it is okay if your output differs, so long as each word in the ladder is valid and the ladder is the same length as the expected ladder.

What to Submit

- Submit a zip file containing all code: `lab4.py`, `lab4_queue.py`, `stacks.py`, and `wordladder.py`.

Rubric

Grade	Overall	Part I	Part II
4	At least a 3 in Part I, and a 4 in Part II	All five methods behave as expected.	Passes all test cases
3	At least a 2 in Part I, and at least a 3 in Part II.	Minor bug in one of: enqueue, dequeue, first; or major bug in <code>is_empty</code> or <code>__len__</code>	Issues stem from 1-2 of these: initial queue is incorrect; loop termination condition is incorrect; uses <code>deepcopy</code> incorrectly; or tracks used words incorrectly
2	At least a 1 in Part I, and at least a 2 in Part II.	Minor bug in 2 or more functions; or major bug in 1 of enqueue, dequeue, or first.	Issues stem from all of above; OR from finding words 1 letter different; OR from main loop logic issue
1	At least a 1 in both parts, or a 3 or better in Part II.	Major bug in 2 of enqueue, dequeue, or first.	Multiple issues from those above
0	Does not meet standards for a 1	Uses some major structure other than the 2 stacks given (e.g. list, Nodes).	Many major issues, does not reflect understanding of how to use stacks and queues