CS/IT 200

Lab 6: Word Ladders

Due Date: One week after lab session, 11:59pm

Submission

- Submit all code in a single zip file to the Lab 6 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 `lab6_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.

Part II – Building the dictionary of valid words

You have been provided with `dictionary.txt` which contains every valid English word for this project. This file contains exactly one word per line. You should not assume a specific number of lines in the file.

In `wordladder.py`, complete the `read_dictionary()` function which should read through that text file and add each word to the `wordlist` dictionary. In wordlist, the keys should be

Acknowledgement: This assignment is adapted from S&Q: Word Ladder, created by Victor Adamchik, Carnegie Mellon University, and posted on EngageCSEdu.

word lengths, and the values should be lists of words with that length. For example, the entries might include:

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

Part III – 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. 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.

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 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	stare	stark	stars
start	start	start	start
smart	smart	smart	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 `lab6.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: `lab6.py`, `lab6_queue.py`, `stacks.py`, and `wordladder.py`.