ICT1008: DATA STRUCTURES AND ALGORITHMS

Lab 1: Word Ladder

1. Objectives

1. Revise Python programming and practice OO programming with Python.
2. Gain an understanding of algorithms used for efficient implementation.
3. Understand and apply stack & queue to solve the Word Ladder problem.

2. Overview

Word ladder is a word puzzle game invented by Lewis Carroll in 1878 [1, 2]. "A word ladder puzzle begins with two words, and to solve the puzzle one must find a chain of other words to link the two, in which two adjacent words (that is, words in successive steps) differ by one letter" [1]. Each word in the ladder must be a valid English word and must have the same length.

```
stone
Atone
aLone
Clone
clonS
cOons
coNns
conEs
coneY
Money
```

For example [2], to turn stone into money, one possible ladder is given on the right. Note that many ladder puzzles have more than one possible solution. In this lab, you must write a program to determine a shortest word ladder. Even for shortest ladder, there may be more than one. For example, another ladder from stone to money is:

stone store shore chore choke choky cooky cooey coney money

3. Instructions

You are provided with a project template called Lab_1 which can be downloaded from xSite. The template consists of 3 template source code files (MyStack.py, MyQueue.py and WordLadder.py) and a dictionary.txt file.

Important constraints:

1. You are not allowed to remove any source code from the template.
2. You have to use MyStack and MyQueue classes in your implementation.
3. You can add more functions to the template.
4. The dictionary.txt file is an incomplete English dictionary. As such, there may be cases where no ladder could be found in this simple dictionary although a solution does exist if a more complete dictionary was used. However, in this lab, you must use this dictionary.

There are several ways to solve this problem. One simple method (might not lead to good performance) involves using stacks and queues [2]. The algorithm (that you must implement) works as follows (the algorithm description and examples are extracted from [2]):

<u>Step 1:</u>
Get the starting word. Search through the dictionary to find all words that are one letter different. Create stacks for these words, containing the starting word and the word that is one letter different. Enqueue each of these stacks into a queue. This will create a queue of stacks.

Example:
Suppose 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. The figure below represents a queue of stacks.

```
---------------------------------------
| scart | start | swart | smalt | smarm |
| smart | smart | smart | smart | smart |
---------------------------------------
```

Figure 1

Step 2:
Then dequeue the first item (which is a stack) from the queue, look at its top word and compare it with the ending word of the puzzle. If they are equal, you are done – this stack contains the ladder. Otherwise, you find all the words one letter different from the top word. For each of these new words, create a copy of the stack and push each word into a stack. Then enqueue those stacks to the queue. And so on. Repeat step 2 until you reach the ending word or the queue is empty.

Example:

Now dequeue the front of the queue in Figure 1, we get the stack:

```
---------
| scart |
| smart |
---------
```

We need to find all words one letter different from the top word scart. This will create seven other stacks:

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

Figure 2

Enqueue these stacks to the queue, the queue size is now 11.

```
                                -------------------------------------------------------
------------------------------- | scant | scatt | scare | scarf | scarp | scars | scary |
| start | swart | smalt | smarm | scart | scart | scart | scart | scart | scart | scart |
| smart | smart | smart | smart | smart | smart | smart | smart | smart | smart | smart |
------------------------------- -------------------------------------------------------
```

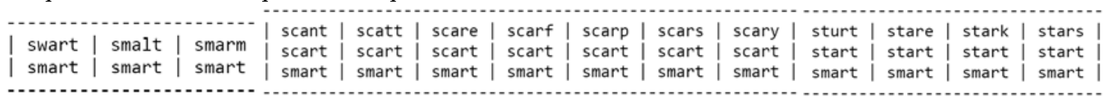
Figure 3

Again, dequeue the front and find all words one letter different from the top word start. This will create four stacks:

```
-------------------------------
| sturt | stare | stark | stars |
| start | start | start | start |
| smart | smart | smart | smart |
-------------------------------
```

Figure 4

Enqueue them to the queue, the queue size is now 14.



```
-----------------------  ------------------------------------------------------------- ---------------------------------
| swart | smalt | smarm  | scant | scatt | scare | scarf | scarp | scars | scary |  sturt | stare | stark | stars |
| smart | smart | smart  | scart | scart | scart | scart | scart | scart | scart |  start | start | start | start |
                         | smart | smart | smart | smart | smart | smart | smart |  smart | smart | smart | smart |
-----------------------  ------------------------------------------------------------- ---------------------------------
```

Figure 5

Note that you have to keep track of used words, otherwise, an infinite loop occurs.

4. Lab Assignment 1 – Stack

You are to complete the implementation of the MyStack class. Note that you should remove the keyword pass at the end of the function template after adding your source code.

More specifically, you are to complete the implementation of the following functions in the class:

- peek()
- peekAt(i)
- size()
- copyFrom(aStack)
- toString()

The requirements for each function can be found in the code template. You are free to use the Python library. As for the toString() function, let us take the following stack as an example:

|stone|
|atone|
|alone|
|clone|

Calling function toString() on the stack should return the following string:

stone atone alone clone

5. Lab Assignment 2 – Queue

You are to complete the implementation of the MyQueue class. Note that you should remove the keyword pass at the end of the function template after adding your source code.

More specifically, you are to complete the implementation of the following functions in the class:

- isEmpty()
- size()

The requirements for each function can be found in the code template. You are free to use Python libraries.

6. Lab Assignment 3 – Word Ladder

You are to complete the implementation of the WordLadder.py for the algorithm described in the “Instruction” section to solve the Word Ladder problem. Again, you are not allowed to remove or modify any of the existing code. You are, however, encouraged to add more functions to the code template to support your implementation.

The function readFromDictionary(fileName) has been implemented which reads from the dictionary.txt file and store all the words in the global dictionary array. findWords(firstWord, lastWord) is the main function which should implement the algorithm for solving the word ladder puzzle. Your program when being invoked should output one word ladder from the start word to the end word for each test case. Each word in the ladder must be a word that appears in the dictionary. This may or may not include the given start and end words. There may be more than one word ladder between the start word and the end word. Your implementation only need to output any one of these ladders.

Many test cases have been added to the code template to help you in testing your implementation. Below is a sample output screen for the test cases in the code template:

```
[stone atone alone clone clons coons conns cones coney money]
[babies gabies gables gabled sabled sailed stiled stiles steles stells steels steeps
sleeps sleepy]
[devil devel level lever leger luger auger anger angel]
[monk conk cork pork perk  perl]
[blue blae blad bead beak peak penk pink]
[heart]
[slow slot slit flit fat fast]
[atlas aulas auras aures lures lares laris labis labia labra zabra zebra ]
[babes bales balls calls cells ceils ceili chili child]
[train brain brawn brown brows brogs biogs bings bines bikes]
[brewing crewing chewing chowing choking cooking booking boobing bobbing bobbins
bobbies bobbles babbles babbled wabbled warbled warsled warsted waisted whisted
whisked whiskey]
[men man can]
[money boney bones bonks boaks boars soars scars scart smart]
[mumbo bumbo bombo bombs boobs blobs globs gloss glost ghost]
No Ladder for snow -> stop
```

Figure 6

7. Lab Optional Assignment

Add a new function in the WordLadder.py file that returns ALL ladders of a given length ladderLength (note: not necessarily shortest). For example, there are only two ladders of length 4 between sail and ruin:

sail, rail, rain, ruin
sail, sain, rain, ruin

However, there are 12 ladders of length 5 as follows:

[sail bail rail rain ruin]
[sail jail rail rain ruin]
[sail mail main rain ruin]
[sail mail rail rain ruin]
[sail pail pain rain ruin]
[sail pail rail rain ruin]

[sail rail rain rein ruin]
[sail sain main rain ruin]
[sail sain pain rain ruin]
[sail sain rain rein ruin]
[sail tail rail rain ruin]
[sail wail rail rain ruin]

Using the new function, find all the ladders up to length 8 for sail -> ruin.

8. Submission & Assessment

All source codes and answers (if any) for lab assignments must be submitted in xSite by **0800hr, 10 Feb 2017**. There will be a lab quiz on the lab assignment conducted during lecture at **0900hr, 27 Feb 2017**. You are allowed to print and use a hardcopy of your source code and answers during the quiz.

9. Note on Plagiarism

The University's policy does not allow you to copy software as well as your assessment solutions from another person. Copying of another person's work is unacceptable. It is the responsibility of all students that their assessment solutions are their own work. You must also ensure that others do not obtain access to your solutions for the purpose of copying a part of them. Where such plagiarism is detected, **both persons** involved will receive **ZERO** mark.

10. References

[1] http://en.wikipedia.org/wiki/Word_ladder
[2] CMU Computer Science: https://www.cs.cmu.edu/~adamchik/15-121/