PROJECT REPORT ON

Word Ladder Problem

SUBMITTED BY

Ji-In Moon (Francisco Moon)

V00712062

University of Victoria

Faculty of Engineering: Computer Science

CSC 106: THE PRACTICE OF COMPUTER SCIENCE

Spring 2016

Instructor

Bill Bird

Introduction:

The main purpose of this project was to create a program that takes in list of words and create a Graph data structure in which the individual words represent vertices, and edges formed if and only if these words differ by exactly single character. Then, it asks for two words to find shortest route from-to, and display computed path as list of words.

Class Hierarchy:

- class java.lang.Object
    - class WordLadder
    - class Graph
        - class undiGraph
        - class Vertex
        - class Edge

Instructions:

1) Along with this documentation, all the necessary classes can be found within the attached file named 'WordLadder.' There are five java classes - *WordLadder, Graph, undiGraph, Vertex*, and *Edg*e. Note that while these classes are not package protected, it is recommended that these are to be compiled and run within a same folder.
2) Full specification of each classes is provided above each method headers in the form of comment. Javadoc is also available.
3) List of words that the program accepts should be: (a) of all the same lengths, (b) separated by a space, and (c) one string (no multiple line inputs). Sample list is provided for you already in the 'wordladder.txt'. It is the same list of words from assignment #5.
4) The main client is the *WordLadder*. Upon running it, it will automatically read the input string of words in wordladder.txt and display adjacency list representation of the graph. Then, it will ask for input of two words to compute shortest route from-to, and display the path.

Report:

The major change to the initial implementation design of representing the ADT Graph as array of linked-lists has been made when it became apparent that having single Node object representing both the vertex and edge in the linked-list was not feasible during first testing.

Due to nature of how Node kept track of its edges by simply referencing other Nodes, it created a infinite loop when traversing the graph (i.e. Node1.next pointing at Node2, and Node2.next referencing Node1). This initial problem was addressed by splitting Node class into two separate objects, *Vertex* and *Edge*.[1] While Vertex classes retained original data-fields of Node class, the edges simply contained a simple reference pointer to target vertex that it has formed an edge with.

Another major change was the implementation of REPL in WordLadder class when it was noticed that having to re-run the program repeatedly to compute shortest route between another set of two words on same list of words was becoming tiresome. Along with this change, it was decided that data-field of Vertex class should be initialized for each new traversal with different parameters.

Other than these changes, the product is now fully complete and works as described in the initial proposal.

[1] The idea of seperating the Node object into Vertex and Edge is credited to following code: http://en.literateprograms.org/index.php?title=Special%3aDownloadCode/Dijkstra%27s_algorithm_%28Java%29&oldid=15444

References and Resources:

Burr, Bill. CSC106 - Spring 2016 Lecture Slides: “Graph Algorithms I” & “Graph Algorithms II”. UVic.

De Lillo, N. TechReport: “Implementation of Graphs Using Java.util”. <http://support.csis.pace.edu/CSISWeb/docs/techReports/techReport224.pdf> Mar 2016. Pace Univ.

Literateprograms. Dijkstra’s algorithm (Java). <http://en.literateprograms.org/index.php?title=Special%3aDownloadCode/Dijkstra%27s_algorithm_%28Java%29&oldid=15444>

Wikipedia. Adjacency List. <https://en.wikipedia.org/wiki/Adjacency_list>

Wikipedia. Breadth-first Search. <https://en.wikipedia.org/wiki/Breadth-first_search>