Word Ladder Problem

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
William B.Bird

Objective:

Write a program which reads a list of words, as well as start and end points for a word ladder, then prints out the shortest possible word ladder.

Outcome:

By end of this project, the goal is to be able to:

- Create and implement ADT Graph using combination of array and linked-lists.
- Understand basics of graph theory and two fundamental Graph traversal algorithms: Depth First Search(DFS) and Breadth First Search(BFS).
- Apply better understanding of data abstraction by continuing to practice principles of encapsulation and modularity via Java interface.
- Further familiarize with implementations of various data structures(queues, etc..).

Summary:

The program takes in list of words and generate a graph with vertices as the words. Then, edges are made between two vertices if and only if the words differ by exactly single character. Then, BFS is performed on the graph to generate a tree that contains information of shortest route from W1 to W2.

Details:

The program would be written entirely in Java, and mainly comprised of two classes: WordLadder and Graph. The WordLadder class will be a main client that will read given text-file, generate and perform graph operations, and ask user for two words to find shortest route from-to. The ADT Graph will provide typical graph operations such as addVertex(), addEdge(), and so on.

Note that while there are many good Java library sources available, for the purposes of learning how to implement ADT Graph and the traversing algorithms, will try to minimize the usage of java.util package, and other outside library sources unless time constraint becomes an issue.

The graph will be generated by adding each individual vertices - 'words' - into an array. Then, edges are indicated by additional nodes connecting from these vertices in the array (ADT Graph is implemented as an array of linked-list with head referencing the vertex, and each subsequent nodes as edges). This will be a simple procedure of comparing two Strings.

Then, the shortest route from two words are computed by performing *Breadth First Search*. First, search the graph (an array) for starting Node, set its distance to 0 and enqueue it. Note that We do not require additional initialization as the vertices themselves keep track of its distance from the starting point and its parent in data-field of Node class (unless we want to perform a separate search with different parameter). Then, while the queue is not empty following algorithm[1] is looped:

Create a temp node and **references** dequeue'd node.
For every node, n, that temp node has an edge with,
If n.distance == -1 (indicates that it has not been visited yet), **do**
Set n.distance == temp.distance+1
Set n.parent == temp
Enqueue n

[1] Wikipedia, BFS. https://en.wikipedia.org/wiki/Breadth-first_search#Pseudocode

This will result in a *tree* structure which we can traverse easily via following node.parent path - tracing back to starting node. And this will be the shortest route between two nodes (in this case from W1 to W2).

The project is estimated to be completed within 10 hours, and with possible of future extension including GUI implementation for providing better interface than simple text-terminal output (but doubtful).

Resources:

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

Tutorialspoint. Data Structure - Graph Data Structure. <http://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm>

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

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