Supplemental/Alternative Program No. 4

EECS 214/395 –Data Structures and Data Management
Fall 2014
Instructor: Professor Peter Scheuermann

Assigned: 11/25/14 Due: 12/08/14 (midnight)—no extensions

In response to public demand this program can serve as bonus for those of you who want to improve your standing. In the class, it will count the same number of points (10) as the one with Dijsktra's algorithm. Alternatively, you may also choose to do this program instead of the other one

The Word Ladder Problem

Let us consider the following puzzle called a word ladder. Transform the word "FOOL" into the word "SAGE". In a word ladder puzzle you must make the change occur gradually by changing one letter at a time. At each step you must transform one word into another word, you are not allowed to transform a word into a non-word. The word ladder puzzle was invented in 1878 by Lewis Carroll, the author of *Alice in Wonderland*. The following sequence of words shows one possible solution to the problem posed above.

```
FOOL
POOL
POLL
POLE
PALE
SALE
SAGE
```

There are many variations of the word ladder puzzle. For example you might be given a particular number of steps in which to accomplish the transformation, or you might need to use a particular word.

Your program should read from a file of words of length k, store them in a graph and use an appropriate graph algorithm to answer the question: Given a source word and a target word, is it possible to transform the source to the target in at most k steps, by changing one letter at a time, and if yes, what is the sequence? The intermediate steps should also be valid words from the input file.

We can solve this by using a single-source shortest path algorithm.

- Represent the relationships between the words as a graph.
- Use a breadth first search to find an efficient path from the starting word to the ending word.

1. Building the Word Ladder Graph

Our first problem is to figure out how to turn a large collection of words into a graph. What we would like is to have an edge from one word to another if the two words are only different by a single letter. If we can create such a graph, then any path from one word to another is a solution to the word ladder puzzle. Figure 1 shows a small graph of some words that solve the FOOL to SAGE word ladder problem. Notice that the graph is an undirected graph and that the edges are un-weighted.

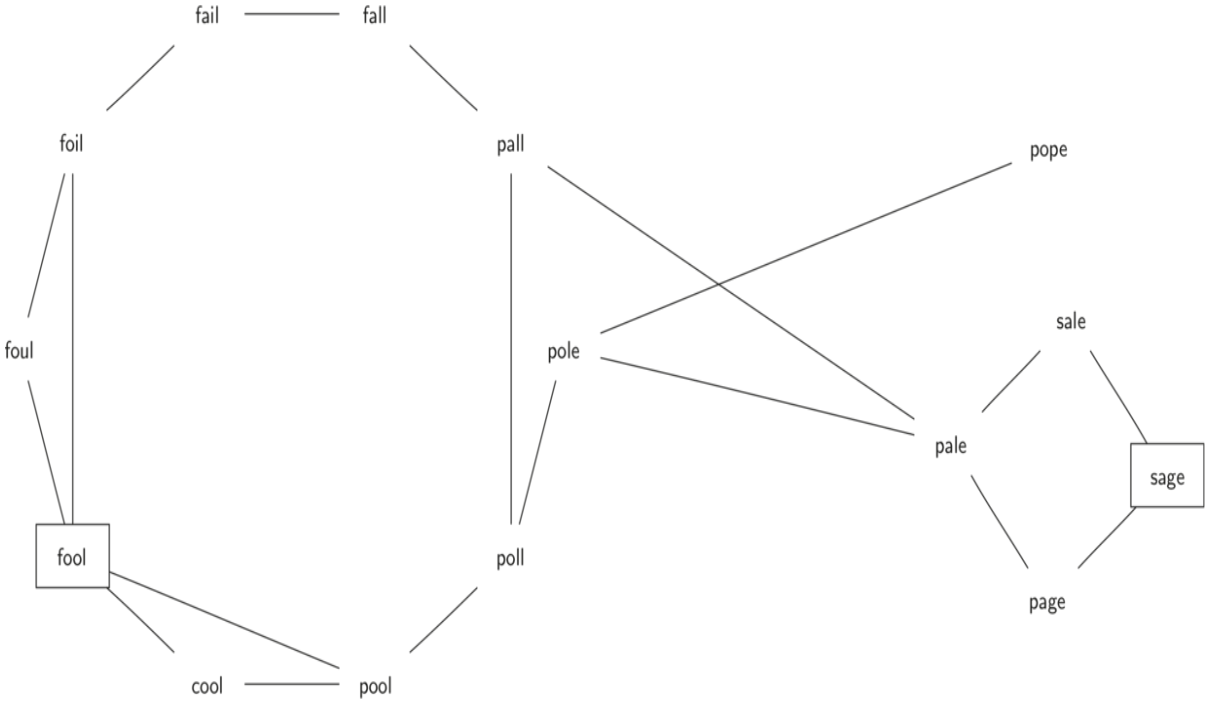

Figure1. A Small Word Ladder Graph

Your input will consist of a list of words that are all the same length *k*. You should create a graph represented as an adjacency list where every vertex in the graph corresponds to one word in the list. To figure out how to connect the words, you could compare each word in the list with every other. If the two words in question are different by only one letter, we can create an edge between them in the graph. This is an $O(n^2)$ algorithm.

(Note : We can do better then this by using a dictionary approach)

Input & Output

The program will read the list of words from a file. All words will be of the same length k.

It will then repeatedly ask the user to enter a source word and a target word. If it is possible to transform the source to the target in at most k steps, the sequence of transformations should be printed. Otherwise, it should print, "Cannot convert".

Submission & Grading

Submission is done through the Northwestern BlackBoard system. After you login to the system, you will find the link that allows you to upload your assignment.

We request all programs to be compiled on the T-lab machines using g++. You should modify the template make file for compilation/submission, which can also be downloaded from Blackboard under assignment.

The files to be submitted include the following:

1. all source code files, .h, .cpp/ .cc files
2. the make file you used to compile your program
3. A brief (less than a page) description of any important design decisions you made while implementing the project.

To submit your project:

Before submission, you must use the following command to create a zip ball that includes all the files required above. You should not submit the executable! You should submit this zip file only. To do that, cd into your working directory on the T---lab machines and then type:

```
tar czf your_name.tgz
```

where `your_name` is your name.

A make rule that will do the trick:

```
submission:
        ${TAR} ${NAME}-${PROJ}.tar $DELIVERY$ $COMPRESS$
        ${TEAM}-${PROJ}.tar
```

You would then only have to type

```
make NAME=<your_name> submission
```

Late submission policy:

Programs will be accepted for at most two days later with a penalty of 10% off for each day. After submission, if you discover a mistake and want to submit a revised copy, you can simply redo the whole procedure and overwrite your old file.

We will recompile and test your program on the T---lab machines. We will use a sample data file other than the one supplied to you to test whether your program runs correctly. We will also be checking the quality of your solution, efficiency and programming style. Please refer to the first handout for more detailed grading policy. The points are assigned as follows:

1. Implement the priority queue class and its methods: 30 pts
2. Implement correctly the directed graph. 20 pts Every memory leakage will result in 2 pts deducted.
3. Implement the distance---based shortest path algorithm: 40 pts.
4. Programming style: including OO design, proper comments etc. 10 pts

If you have problem finishing the entire program, we suggest you follow these steps to get partial credits for the work you have done.

Good luck!