Assignment 5

CS 311 Data Structures
Spring Quarter, 2006

Due: Thursday 06/01/06, 11:59pm

Brief description

In this assignment you will implement a graph and use it to solve a number of problems. Read the whole assignment before you start writing code. Several issues regarding the graph implementation will be handled more smoothly if you have an idea of how the problems are to be solved.

START EARLY!

Part 0

Code a graph implementation that uses an adjacency list to store the edge information.

Each Vertex should be identified by its name (a string). Each edge should store a weight and whether it is directed or not. Additional attributes may be added as needed.

The graph class should contain methods for inserting vertices and edges, performing a topological sort, applying a shortest path algorithm and traversing the graph. Keep in mind that the graph representation should be hidden from the user. In order to insert an edge, for example, one would only have to specify the names of the vertices to be linked.

Part 1

A well-known word game involves trying to transform one word into another through a short sequence of one-letter substitutions. For example, the word "pears" can be transformed to "scans" in less than 5 steps as follows: "pears" → "sears" → "seats" → "scats" → "scans".

Your program should read a dictionary 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 dictionary words.

As will be discussed in class, this can be solved by using a single-source shortest path algorithm.

Input/Output

The program will read the dictionary from a file (through a command-line argument). All words of the dictionary 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".

If a -d option has been given in the command line, the program should also print the results of a traversal of the graph right after it is built.

Part 2

Implement a method that performs a topological sort of the vertices and prints them out in that order. The input graph will be read from a file and will have to be directed. See the section titled **Input/Output** in part 3, for an example.

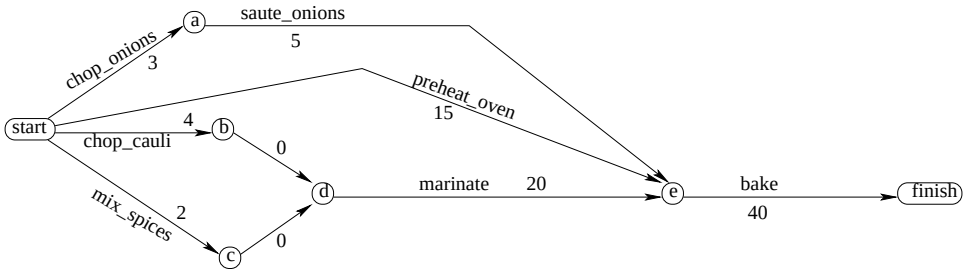
Part 3

Preparing a meal involves several steps, some of which have to be done in order. Others can be done simultaneously if there are several cooks working together. For example, one cook could mix the spices while another cuts up the meat. Each one of these tasks takes a certain amount of time and both tasks need to be complete before tossing the meat with the spices. If it takes 3 minutes to measure and mix the spices and 1 minute to cut the meat, then tossing the meat and spices will have to be done at the end of the 3 minutes. The *longest* event is what determines how long the whole process will take.

There are two issues here. How to use a graph to represent such a process and then how to find how soon the process can be complete, assuming that independent parts can be done in parallel (by assistant cooks).

Each vertex in the graph should represent a stage towards the completion of the process. There is an edge between two stages if one directly follows the other. The weight on the edge is the time it takes to complete the target stage after the source stage is complete. The edge is, of course, directed.

Here's an example of such a graph:



The longest tasks and the prerequisites relation are what determine the time it will take to complete the task. In this example, the dish will be prepared (at best) in 64 minutes. The path that determines this is "start" → "b" → "d" → "e" → "finish". This is called the *critical path* and as you can see is the longest one from "start" to "finish".

An algorithm for solving the problem would be similar to a shortest path algorithm for a directed graph (e.g. Dijkstra). However, since we need a longest path we should preprocess the weights in some way.

There is another issue we need to be careful about. In Dijkstra's algorithm, when we remove an element from the priority queue it is guaranteed that the final (shortest) distance of that element from the source has been computed. This will not necessarily be the case here. Removing the node that is currently the furthest from the source does not mean we will not find another path later on that will be even longer. For example, after processing the start node's neighbors in the graph shown above, we'll have "e" as the furthest element from "s". If we go on to process it next, however, and remove it from the queue, we will miss the fact that b-d-e is even longer. A priority queue, then, is not the data structure we need. The vertices need to be processed in a different order than the one imposed by a priority queue.

Input/Output

Your program will read the graph one edge at a time. For example, the graph shown above would be given as:

```
s a 3
s b 4
s c 2
s e 15
a e 5
b d 0
c d 0
d e 20
e f 40
```

It is always assumed that "s" is the start node and "f" the finish node. The program should print the critical path and its total length.

If a -d option has been given in the command line, the program should also print the results of a traversal of the graph right after it is built.

Sample executable

A sample executable will be provided so that you can see the expected behavior of the program.

Running your program

Command-line options are:

```
-1 to test problem 1 (can be combined with -d for debugging) followed by the file name.
-2 to test problem 2 (can be combined with -d for debugging) followed by the file name.
-3 to test problem 3 (can be combined with -d for debugging) followed by the file name.
```

It is assumed that the user will always provide a correctly formatted data file.

Submitting your code

As usual, submit a tarball of your sources (no executables please) to `cs311@cs.northwestern.edu` and don't forget to cc yourself.