CSE 373 – Data Structures and Algorithms
Autumn 2003
Wet assignment #3

Due dates.

Team formation: by 5:00pm on Thursday, Dec 04, 2003
Full assignment: by 12:00pm on Wednesday, Dec 10, 2003 (electronic)
in class on Wednesday, Dec 10, 2003 (on paper; *see instructions*)

General.

Be sure to read this *entire* document as soon as possible, since this may affect how you approach the problem! Note that some of the directions have changed in subtle ways from what they were in previous assignments, so do not assume that they are the same; it is your responsibility to read it all!

Logistics.

For this wet (programming) assignment, you need to work in a team with one other person in the class. It is up to you to decide who that person will be. It does not have to be the person whom you worked with on either of the previous wet assignments, although you are free to pair up with them if you like.

We require that one of the two students in each team email both TAs (and CC their teammate) *by 5pm on Thursday, Dec 4th* letting us know the following:

(a) who the team members will be – names and email addresses for both; and
(b) what implementation language (Java or C++) both of you are going to be using.

Note: If you fail to notify us on time, we will deduct 10% of your score on this assignment.

Problem description.

In this assignment you will implement Dijsktra's algorithms for the single-source shortest path problem in a directed graph. The number of vertices, *n*, is known in advance. One vertex, denoted '0', is the source vertex.

What you need to do.

You need to implement the following functions:

- `Read_Graph(graphFile)` – reads in the graph from a given file and stores it in a data structure of your choice. The graph is given as a list of triplets of the form (*u*, *v*, *len*). Each triplet (*u*, *v*, *len*) corresponds to a single edge from vertex *u* to vertex *v*, whose length is *len*. You can assume that *len* is a non-negative integer between 0 and 20. Also assume that the vertices are numbered 0, 1, …, *n*-1. The required time complexity of this operation is $O(n+m)$.

- `Compute_Shortest_Paths()` – applies Dijkstra's algorithm to compute the lengths of the shortest paths from the source vertex '0' to all other vertices.
  The required time complexity of this operation is $O(n^2)$.

- `Report_Shortest_Path(target)` – outputs the shortest path and its length (not only the length!) from the source '0' to `target`, or outputs `"not connected"` if `target` is unreachable from '0'.
  You can assume that `Compute_Shortest_Paths()` has been called prior to this.

The required time complexity of this operation is $O(k)$, where k is the number of edges in the shortest path from '0' to `target`.
The output of this function should be a linked list whose first element is the source vertex and whose last element is the target vertex.

- `Sort_Vertices_by_Distance()` – outputs the vertices in a non-decreasing order of their distance from the source '0'. If there are vertices that are unreachable from '0', they should appear at the end of the output in no special order.
  You can assume that `Compute_Shortest_Paths()` has been called prior to this.
  The required time complexity of this operation is $O(n)$.
  Hint: This is the only place where you will need to take advantage of the fact that the length of each edge is between 0 and 20.

The total space complexity should be $O(n+m)$, where m is the number of edges in the graph.

Development environment.
As an implementation language you are allowed to use either Java or C++. Be sure to avoid using features that are specific to a particular version of a compiler, since if you do so we likely will be unable to compile your code and you will lose points.

Submission.
The submission is electronic and part of it is also paper-based, as specified below.
We need *only one* electronic submission and *only one* printed submission per team.

What to submit. All of the following (1–5) need to be submitted *electronically* by class time on the due date:

1) Your code – the class with the code for all of the implemented operations. Also include any supporting headers, packages, and test module(s) you wrote. Finally, be sure to include the input test file(s) you used (from which your program reads in the graph(s) and initializes the data structures), so that we can successfully reproduce your results when running and testing your code.
2) A description of all modules your code consists of (including module names and a one-paragraph description of what each module does), as well as *clear* instructions on how to compile, run, and test your code. Put this in a plain text file, called `description.txt`.
3) A description of the data structures you chose to use and a *clear* explanation of the algorithms for each of the four required operations (above). One paragraph per operation should suffice. Also include (in the same paragraphs as the corresponding algorithms) your brief arguments why the time and space complexities of your algorithms are what the specification asked for (and not worse). Put this in a plain text file, called `algorithmsANDcomplexities.txt`.
4) A brief description of the tests you performed. Put this in a plain text file, called `testing.txt`.
5) A brief description of who (of the two partners in the team) did what part of the assignment: the different parts of the code, the complexity arguments, the analysis questions, and anything else you did. Be sure to include your names and emails here. Put this description in a plain text file, called `contributions.txt`.
   You are encouraged to work on all parts together, and if so, indicate that this is indeed the case. If, however, you decided to split the work, we also need to know that, as well as how you did it.

(Note that both partners are responsible for all parts of the assignments regardless of whether and how you may decide to split the work among yourselves.)

In addition, *print* and bring to class on the due date:

1) The source code of the class that implements the required four operations *only*. (Please, *do not* print header files, helper files, packages, or test modules/classes!)
2) The description text files 2–5 (described above). To save paper and trees, please merge them together into one file and then print.

Do not forget to write the names of both partners on the printed sheet you submit.

Printing guidelines. Double-sided printing and condensed code printing (i.e., double-sided at 2 pages of code per side) is highly encouraged! (This shows that you care to present yourselves as professionals.) Modern printers, including many of those available at UW facilities, can do that. Ask the lab person for assistance if you are unsure how to do it.

Where to submit. Instructions on how to electronically submit your solutions will be announced on the mailing list.

Quality criteria.

For full credit, you need to meet the following quality criteria:

1) Your submission needs to contain all parts described in the previous section.
2) Your code needs to:
   - Compile and run without errors or warnings. (We will follow the instructions you will have provided us with, including on how to test it, so those need to be accurate.)
   - Contain comments (this is really important!), making it easier for a human to understand what you have done.
3) Your argumentation about the time and space complexities needs to be sound. Brevity is encouraged but not at the expense of clarity or soundness.
4) Your printed materials need to look professional (i.e., be printed and stapled together rather than hand-written and loose).

Our advice.

By now you know that programming assignments take time, so be sure to *start early*!

Good luck!