CSCI203/CSCI803 ASSIGNMENT 3

(10 marks + 2 demo marks)

Step-1 demo due during your Week-10 Lab class (2 marks)
Final submit due Week-12, Fri 11:59pm 25 Oct. (10 marks)

This assignment involves an extension to the single source - single destination shortest path problem.

The Program

Your program should:

1. Open the text file "ass3.txt". (Note: "ass3.txt" should be a hardcoded as a constant.)
2. Read a graph from the file.
3. Find the shortest path between the start and goal vertices specified in the file.
4. Print out the vertices on the path, in order from start to goal.
5. Print out the length of this path and the total number of vertices visited.
6. Devise a strategy for determining the second shortest path between the vertices.
7. Find the second shortest path between the start and goal vertices specified in the file.
8. Print out the vertices on the path, in order from start to goal.
9. Print out the length of this path and the total number of vertices visited.
10. Optimize your shortest and second shortest path algorithms to improve their performance.

The data files are constructed as follows:

- Two integers: nVertices and nEdges, the number of vertices and edges in the graph.
- nVertices triples consisting of the label and the x- and y-coordinates of each vertex.
- nEdges triples consisting of the labels of the start and end vertices of each edge, along with its weight. Note: the weight associated with an edge will be greater than or equal to the Euclidean distance between its start and end vertices as determined by their coordinates.
- Two labels, the indicating the start and goal vertices for which the paths are required.

A proposed solution to the second shortest path problem is as follows:

For each edge e_i on the shortest path:
    Find the shortest path on $(V, E - \{e_i\})$. *// shortest path without edge e_i*
The shortest of these is the second shortest path.

Questions

Think about this! Is this proposed solution correct always?

- What if we require that the second shortest path be longer than the shortest path?
- What if the graph contains cycles?
- What if the graph is undirected?

Explain your answers. If necessary explain how you might modify the proposed algorithm to address any issues that you identify.

Note: you may implement either the proposed solution or any modification you develop. You are not required to implement a modified proposal if you do not wish to do so.

Step-1 *(Week-10 demo, 2 marks)*

For step 1, you should read the data file into adjacency lists, or an adjacency matrix, and print on the screen the first 5 vertices and the vertices they are connected to together with their weights. e.g.:

```
a:  c(35)  d(27)  e(48)
b:  d(35)  g(27)
c:  b(125) e(20)  f(56)  h(31)
   . . .
```

Note: The data shown above is for format purposes only.

Step-2 *(Discovering the Shortest Path) (2 marks)*

For step 2, you should implement Dijkstra's algorithm and find the shortest path between the start and goal vertices specified in the input file. Print out the vertices on the path (in order from start to goal), the length of this path and the total number of nodes visited by the algorithm to discover the shortest path.

Step-3 *(Discovering the Second Shortest Path) (4 marks)*

For step 3, you should devise a strategy for determining the second shortest path between the start and goal vertices specified in the file and implement your solution. Print out the vertices on the path (in order from start to goal), the length of this path and the total number of nodes visited by the algorithm to discover the second shortest path.

Step-4 *(Optimisation) (2 marks)*

For step 4, implement the A* algorithm and repeat Step-2 and Step-3. Print the same information to show the improved performance.

Step-5 *(Report) (2 marks)*

In a comment block at the bottom of your program (no more than 30 lines of text) list the data structures used by your program and describe your solution to the second shortest path problem. Also, provide answer to the questions. For this step, marks will be awarded based on accuracy and correctness.

Compilation:

All programs submitted must compile and run on banshee:

```
C:       gcc ass2.c
C++:     g++ ass2.cpp
Java:    javac ass2.java
Python:  python ass2.py
```

- Programs which do not compile on banshee with the above commands will receive zero marks. It is your responsibility to ensure that your program compiles and runs correctly.

Marking Guide:

- Marks will be awarded for the appropriate use of data structures and the efficiency of the program at processing the input and producing the correct output.
- Marks may be deducted for untidy or poorly designed code.
- Appropriate comments should be provided where needed.
- There will be a deduction of up to 4 marks for using STL, or equivalent libraries, rather than coding the data structures and algorithms yourself. You may use *string* or *String* type for storing words or text, if you wish.
- All coding and comments must be your own work.

Submission:

Assignments should be typed into a single text file named "ass2.*ext"* where *"ext"* is the appropriate file extension for the chosen language. You should run your program and copy and paste the output into a text file named: "output.txt"

Submit your files via the *submit* program on banshee:

submit -u *user* -c csci203 -a 3 ass3.*ext output.txt*

- where *user* is your unix userid and *ext* is the extn of your code file.

Late assignment submissions without granted extension will be marked but the points awarded will be reduced by 1 mark for each day late. Assignments will not be accepted if more than five days late. An extension of time for the assignment submission may be granted in certain circumstances. Any request for an extension of the submission deadline must be made via SOLS before the submission deadline. Supporting documentation should accompany the request for any extension.