Algorithms – Graph Project

This programming project is to implement a few graph algorithms. You will be designing and implementing a program for an airline travel agency. Through your program, customers will be able to issue queries about the prices and distances of the routes offered by the company. Your program should **operate via a menu-driven console interface**.

Your program read one of the two files containing all the routes offered by the airline, one for domestic flights and one for international flights, http://www.cs.uakron.edu/~duan/class/435/projects/airline_domestic.txt and http://www.cs.uakron.edu/~duan/class/435/projects/airline_international.txt and store the information in a graph. The route file contains: the number of cities serviced by the airline, the names of each of those cities, and a list of routes between cities. A route between cities should be presented as the numbers of the two cities (as they appear in the file starting from 1), the distance between the two cities, and the cost of flying on that route.

You must represent the two graph as an adjacency list. The cities (vertices) should minimally have a string for a name and any other information you want to add. The data should be input from the routes file specified by the user. The edges will have multiple values (distance, price) and can be implemented as either a single list of edges with two values each, or as two separate lists of edges, one for each value. Note that this means edges in your graph will have multiple weights. You can assume that all the listed routes are bi-directional (i.e., the that airline offers flights in both directions for the same price).

Part I: Your program should be able to answer the following queries:

- Show the entire list of direct routes, distances and prices. This amounts to outputting the entire graph in a well-formatted way (you do not need to use graphics). Note that this operation does not require any sophisticated algorithms to implement.
- Display a minimum spanning tree for the service routes based on distances (not necessary to use graphics, points and edges are okay). This could be useful for maintenance routes and/or shipping supplies from a central supply location to all the airports. If the route graph is not connected, this query should identify and show all the minimum spanning trees of the connected components of the graph.

Part II: Your program should allow for each of the **three** "shortest path" searches below. For each search, the user should be allowed to enter the source and destination cites (names, not numbers) and the output should be the cities in the path (starting at the source and ending at the destination), the "cost" of each link in the path and the overall "cost" of the entire trip (if multiple paths "tie" for the shortest, you only need to print one out). If there is no path between the source and destination, the program should indicate that fact.

- Shortest path based on total miles (one way) from the source to the destination. Assuming distance and time are directly related, this could be useful to passengers who are in a hurry. It would also appeal to passengers who want to limit their carbon footprints.

- Shortest path based on price from the source to the destination. This option is a bit naïve, since prices are not necessarily additive for hops on a multi-city flight. However, to keep the algorithm simple, we assume the prices ARE additive. Since distance and price do NOT always correspond, this could be useful to passengers who want to save money
- Shortest path based on number of hops (individual segments) from the source to the destination. This option could be useful to passengers who prefer fewer segments for one reason or other (ex: traveling with small children).

Part III. Given a dollar amount entered by the user, print out all trips whose cost is less than or equal to that amount. In this case, a trip can contain an arbitrary number of hops (but it should not repeat any cities – i.e. it cannot contain a cycle). This feature would be useful for the airline to print out weekly "super saver" fare advertisements. Be careful to implement this option as efficiently as possible, since it has the possibility of having an exponential run-time (especially for long paths). Consider a recursive / backtracking / pruning approach.

Part IV. Add the following functionality to your program so the airline could update their routes.

- Add a new route to the schedule. Assume that both cities already exist, and the user enters the vertices, distance and price for the new route. Clearly, adding a new route to the schedule may affect the searches / algorithms indicated above.
- Remove a route from the schedule. The user enters the vertices defining the route. This may also affect the searches / algorithms indicated above.
- Quit the program. Before quitting, your routes should be saved back to the file (the same file and format that they were read in from, but containing the possibly modified route information).

Note: You should use the efficient algorithms and implementations for your queries; to obtain the MST you should use either Prim's or Kruskal's algorithm and for the shortest distance and shortest price paths the Dijkstra's algorithm. To obtain the shortest hops path you should use breadth-first search.

What to submit.

1. Grade your own project before submit.
2. Submit your source code. Make sure to test your code on more than one set of data. DO NOT submit programs that are not *reasonably correct*! To be considered *reasonably correct*, a program must be completely documented and work correctly for sample data provided with the assignment.
3. A text file named readme.txt. You should use this file to give the reader a very clear description on how to run your code, a high-level explanation of your source code and point to anything unusual or notable about your program.
4. For graduate students, a report of the analysis of the algorithms for solving the problem given in this project. You should present your asymptotic analysis on the space requirement of the algorithms. You should also run your program for many different settings and present your empirical running results in tables/figures.

Source code submission instructions. When you are ready to submit, obtain a printed copy of your program. Remember to sign and attach the required Academic Integrity Pledge cover sheet. The printed program and cover sheet must be turned in to your instructor by the start of class on the date the program is due. You must submit an electronic copy of the program using your Springboard dropbox. Follow these steps:

1. Create a folder named jsmith_1 (but use your first initial and last name).
2. Place just the source files inside the folder.
3. Right-click on the folder and choose Send To... Compressed Folder (or use some other Zip utility to archive the entire folder). The goal is to create a zip archive named jsmith_1.zip (your initial and name) that contains the folder which contains the source files for your project.
4. Drop this single zipped file to the drop box.

Please pay attention to the naming conventions for the submission files. These must be followed exactly for you to receive credit. Invalid submissions will need to be resubmitted with a penalty assessed.

Be sure to electronically submit your working solution before the due date! Do not submit non-working programs. The electronic submission time will be used to assess late penalties (if applicable).

Grading. Your code will be graded on correctness, efficiency, clarity and elegance. What you claim in your writeup (readme.txt, report) will be checked by running your code. For **graduate students**, your report must be written clearly and professionally. Use figures and tables wherever you can to justify your conclusions. Your report should follow the format and suggestions given the document at: http://www.cs.york.ac.uk/projects/howtowrt.html .