**CS/COE 1501 Project 4**

**Goal:**

To gain a better understanding of graphs and graph algorithms through practical implementation.

**High-level description:**

You will be designing and implementing a simple information program for a fictional airline. Your program should be designed to be accessed by employees of the company. Through your program, they will be able to issue queries about the prices and distances of all of the routes offered by the company. Your program should operate via a menu-driven console interface (no GUI).

**Specifications:**

1. At the start of your program, the user should be prompted for a filename containing all of the routes offered by the airline. Two example files are provided for you: airline\_data1.txt and airline\_data2.txt. Route files should contain: the number of cities serviced by your 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.
2. You must represent the graph as an adjacency list. The cities 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 of the listed routes are bi-directional (i.e., the that airline offers flights in both directions for the same price).
3. With the route list loaded, your program should be able to answer the following queries:
   1. Show the entire list of direct routes, distances and prices. This amounts to outputting the entire graph (again, no graphics) in a well-formatted way. Note that this operation does not require any sophisticated algorithms to implement.
   2. Display (again, no graphics -- points and edges are ok) a minimum spanning tree for the service routes based on distances. This could be useful for maintenance routes and/or shipping supplies from a central supply location to all of the airports. If the route graph is not connected, this query should identify and show a minimum spanning tree for each connected component of the graph.
   3. 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, in the real world, prices are not necessarily additive for hops on a multi-city flight. However, to keep the algorithm fairly simple, for this project, you should 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).
   4. 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.
   5. 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.
   6. Remove a route from the schedule. The user enters the vertices defining the route. Again, note that this may also affect the searches / algorithms indicated above.
   7. 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).
4. You must use the algorithms and implementations discussed in class for your queries. For example, to obtain the MST you must use either Prim’s or Kruskal’s algorithm and for the shortest distance and shortest price paths you must use Dijkstra’s algorithm. To obtain the shortest hops path you must use breadth-first search.

**Submission Guidelines:**

* **DO NOT SUBMIT** any IDE package files.
* You must name the primary driver for your program Airline.java.
* You must be able to compile your program by running "javac Airline.java".
* You must be able to run your program with "java Airline.java".
* You must fill out info\_sheet.txt.
* Be sure to remember to push the latest copy of your code back to your GitHub repository before the the assignment is due. At the deadline, the repositories will automatically be copied for grading. Whatever is present in your GitHub repository at that time will be considered your submission for this assignment.

**Additional Notes/Hints:**

* A sample output file is provided for you in sample\_output.txt. This file provides an example of two runs of the program, one for each input file.
* Though code for the algorithms used in the assignment has been provided by the authors of your text book, note that use of this code will require extensive adaptations to account for the two weights of each edge in the graph.

**Grading Rubric**

* Adjacency list representation is used: 10
* Data file I/O works as specified: 5
* Menu interface is user-friendly: 10
* Queries:
  + List of routes/distances/prices: 5
  + MST: 14
  + Shortest path by distance: 7
  + Shortest path by price: 7
  + Shortest path by number of hops: 7
  + All trips less than some dollar amount: 20
  + Add/remove routes: 10
* Assignment info sheet/submission: 5