CMPSC 250 – Analysis of Algorithms (Spring 2015)
Prof. John Wenskovitch
`http://cs.allegheny.edu/~jwenskovitch/teaching/CMPSC250`

Lab 9 - Graphs and Graph Algorithms
Due (via Bitbucket and hard copy) Wednesday, 22 April 2015
50 points

Lab Goals

- Collect together algorithms for minimum spanning tree, shortest path, and graph traversal into a information program.
- Write a few custom modifications to the book's source for graph storage.

Assignment Details

Since Lab 8 was entirely based around writing your own data structures and code, we'll turn back to the textbook's code for Lab 9. We have spent the last two weeks discussing graphs and various algorithms we can perform on them – Prim's and Kruskal's Algorithms for minimum spanning trees, Dijkstra's and Bellman-Ford's Algorithms for shortest path, DFS and BFS for traversing graphs, etc. Now, we'll put these separate graph classes together into a single information system for a fictional airline.

Part One: An Airline Information System (40 points)

To begin, you will load from a file (provided by the user) a list of all of the flight routes that the airline runs. These routes include the cities served and the possible destinations that can be reached non-stop from each city. Thus, this file contains a representation of a graph in which the cities are vertices and the destinations can be reached by edges.

For simplicity, you can assume that all routes are bidirectional, so that you can use an undirected graph. Alternatively, you can use a directed graph, with an edge in each direction for each route. You can think of these as the current active routes, which would be updated if a new route is added or a route is canceled due to weather or underuse. The routes (edges) should have 2 different weights: one weight based on the distance between the cities, and the other based on the price of a ticket between the cities.

A short sample file begins at the top of the next page. In this file, the first line gives the number of cities (vertices) handled by the airline, which should be interpreted as vertices $0..N-1$ for the purposes of the internal workings of the program (Pittsburgh=0, Meadville=1, Erie=2, Hong Kong=3). After the set of cities are a set of routes (edges) represented as {endpoint1, endpoint2, weight_distance, weight_cost}.

```
4
Pittsburgh
Meadville
Erie
Hong Kong
0 3 8050 1000
0 2 125 100
0 1 90 90
1 2 30 50
2 3 7920 800
```

After loading the data file, your program should present a menu to the user which will support the following queries:

1. Show the entire list of direct routes, distances, and prices. (Basically, output the entire graph in a well-formatted way.)
2. Display a minimum spanning tree for the service routes based on distance. If the route graph is not connected, this query should identify and show each of the connected subtrees of the graph.
3. Allow for each of the three shorest path searches detailed in the subparts to this item. For each search, the user should be allowed to enter the source and target cities (names, not numbers), and the output should be the cities in the path (starting at the source and ending at the target), the "cost" of each link in the path, and the "cost" of the overall trip. If multiple paths tie for the shortest, you only need to print one out. If these is no path between the source and target, the program should indicate that fact.
   (a) Shortest path based on total miles (one way) from the source to the target.
   (b) Shortest path based on price from the source to the target. To keep the algorithm fairly simple, you can assume that the prices are additive (no weird airline pricing that results in a fare from A to B that happens to be the same price as a fare from A to C that goes through B).
   (c) Shortest path based on number of flights (individual edges) from the source to the target.
4. Given a dollar amount entered by the user, print out all trips whose cost is less than or equal to that amount.
5. Add or remove a route from the schedule. The user will enter the vertices defining the route (again by city name rather than vertex number).
6. Quit the program.

Note that some of these queries sound complicated, but most are already supported by the algorithms and code provided by the textbook – we just didn't have time to talk about every feature in the book's code.

Additionally, you do not need to use graphics to display the output of these queries. Rather, a well-formatted text output is fine. For example, you could output the shortest distance from Hong Kong to Meadville like so:

```
SHORTEST DISTANCE PATH from Hong Kong to Meadville
--------------------------------------------------
Shortest distance from Hong Kong to Meadville is 7950
Path with edges (in reverse order):
Meadville 30 Erie 7920 Hong Kong
```

Again, you are permitted to use the algorithms and implementations discessed in class and in the textbook for your queries. For example, to obtain the MST you can use either Lazy Prim, Eager Prim, or Kruskal. Since these algorithms are encapsulated in classes in the author's examples, you will need to extract/modify these in order to incorporate them as methods within your application.

Part Two: While You Have Some Downtime (10 points)

Answer the following questions thoroughly:

1. Suppose you use a stack instead of a queue when running BFS. Does it still compute paths? Does it still compute shortest paths?
2. How many minimum spanning trees are possible in a generic DAG? Why?
3. If we implemented Eager Prim without a priority queue, instead scanning through all entries in the `distTo[]` array to find the next vertex to add to the tree, what is the new order of growth for a graph with V vertices and E edges? Would this ever be an appropriate algorithm for Prim?

Submission Details

For this lab, please submit a paper copy of everything listed below. Additionally, please upload all of your files to a folder in your BitBucket repository clearly labeled as "Lab 9."

1. Your source code for your airline information system.
2. Your answers to the questions in Part Two.
3. An Assignment Information Sheet filled out for your source files.

Please remember that all files that you submit should be your own work, though you are welcome to discuss high-level topics and algorithms with classmates.