Lab 07 Specification – Graphs and Graph Algorithms
Due Friday, 27th April 2018 1PM
Total - 50 points.

This is a **team** based lab, so you are allowed to work together with your team mates.
Please refer to the syllabus for information about **late** submission policy.
This lab is related to the **textbook** Chapter 4 pg: 518 - 542, 604 - 629 reading and the **lecture slides** 21, 22, and 24 review.

- 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 06 required you to develop a lot of your own data structures and code, we'll *mostly* use the textbook's code for Lab 7. We have spent the last few classes discussing graphs and various algorithms we can perform on them – Prim's and Kruskal's Algorithms for minimum spanning trees, DFS and BFS for traversing graphs, etc. Now, we'll put these all algorithms into a single information system for a fictional airline.

Part One: An Airline Information System (35 points)

To begin, you will need to generate a text file called **airline-route-data** that contains 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, the **airline-route-data** file contains a representation of a graph in which the cities are vertices and the destinations can be reached by edges.

The dataset should include atleast 30 vertices and 45 edges to build the graph. Although one can manually throw in some data to meet the dataset requirements, you are strongly encouraged to think of ways to include more vertices and edges and automate the dataset generation process in order to make it real.

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.

1. You can implement this by modifying the `Edge` class code shown in textbook page no 610 provided to contain `weight1` and `weight2` properties
2. You can maintain two separate `EdgeWeightedGraph` objects from the class file provided in textbook page no 611, one with distance weights and one with cost weights.
3. You can develop your own implementation of the Graph data structure.

Note: Edge class, EdgeWeightedGraph class, and all other class file dependencies of the java code files are provided in the textbook are available for you along with this lab specification folder inside the source-code/src folder. Simply follow the comments in the code file to compile and execute the program.

A short sample file is shown below. 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. (The book code already handles this for you.)
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 discussed in class and in the textbook for your queries. For example, to obtain the MST you can use either use Prims or Kruskal algorithm. 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?

Submission Details

1. You are required to submit this lab by sending an email with a zipped version of your cmpsc250-lab07-YOURFirstInitialLastName folder.
2. Subject of your email should say ”CMPSC250: Team X Lab 07 Submission”. Here X needs to be replaced with your group number. Refer the group sheet in the course webpage.
3. One email should be sent for your teams lab submission.
4. Send the email to amohan@allegheny.edu // CC the email to all your team members.
5. You should add the following statement in the body of your email
   By doing this submission, I understand that I and my team members are subject to the Honor Code policy.
   Lab submitted by: X1, X2, and X3 (Here X1, X2, and X3 are the name of your team members)
6. Shifting team members is not ideal and not allowed, unless there is an extreme situation which is discussed with the instructor prior to your submission.
7. Provide README file inside your code, that will serve as your documentation and it should give details on how to run your program, give a short description of your algorithms and their worst case complexity and any additional sources you have used. Only one submission from one of the group members is needed, but make sure all of the group members' names are on the submitted documents.

Points Distribution

The breakdown for this assignments points is :

- Part one: 35 points
- Part two: 10 points
- Documentation (README file) : 5 points

Note

- Although this is not due next week, I expect you to be in the lab next week to continue working on your lab and to ask any questions or clarification needs you may have.