CSCI 230 -- Project 4 (Graphs)

Due Monday, 06/01/2020 (MW Section)
Due Tuesday, 06/02/2020 (TTh Section)

This project is designed to help you understand graph structure, its implementation, and usage. You must implement **AdjacencyListGraph** class, which contains node/edge information and **adjacency list using a list or vector**. Additional graph processing algorithms such as shortest paths can be added to this class or another class such as **GraphAlgorithms**. You should try to utilize existing classes in previous labs (modify as needed) and graph algorithms discussed in book/class if applicable. Download the two data files, ***P4Airports.txt*** and ***P4Flights.txt***, for this project. The first file contains a list of airport records: 3-letter airport code and city name. The second file contains flight connections between airports – source airport code, destination airport code, cost of the flight, and flight number. Read in the original data files to store the airport information and the flights between pairs of airports and their associated information (i.e., costs and flight numbers). Provide the user with following menu:

1. Display airport information
2. Find a cheapest flight from one airport to another airport
3. Find a cheapest roundtrip from one airport to another airport
4. Add a flight from one airport to another airport
5. Delete a flight from one airport to another airport
6. Find a flight with fewest stops from one airport to another airport
7. Find all flights from one airport to another airport
8. Find an order to visit all airports starting from an airport
9. Delete an existing airport

Q. Exit

Your program will continue until the user wants to exit this program. Output the revised flight connections to a new file before the program terminates. This new file has same information and format as current flight file and you should name it as P4FlightsRev1.txt. You would also need to output an updated airport file as well and you should name it as P4AirportsRev1.txt (extra credit option 9 only).

You need to include a hidden debugging feature for your project to display the graph when option 0 is entered; it should list all vertices and edges in a readable format like lab 9. If two airports are needed then you must enter source airport following by destination airport and you should validate airport code (i.e., output an error message if applicable). For option 1, you need to display airport information if the input 3-letter airport code is found. For option 2, user will be prompted for the two airport codes; if the connection is found, display the cheapest flight -- all airport codes starting with the originating airport, flight number, and total cost (like LAX -- AA1000 - - > DFW, $189.00) . Option 3 is like option 2, but you need to provide both departing flight and returning flight if a roundtrip is possible. For example, display the cheapest flights -- all

airport codes starting with the originating airport and total cost (like LAX -- AA1000 - -> DFW and DFW -- DL4321 - -> SFO -- UA2000 - - > LAX, $367.99). For option 4, user will be prompted to enter the two airport codes, flight number, and the cost (you can assume that the user can modify the existing flight number and cost if a flight already exists). For option 5, user will be prompted to enter the two airport codes and a flight will be removed if applicable. Option 6 is like option 2 but select a flight with fewest stops. For option 7, you can list all flights in any order. For option 8, enter a starting airport and provide an order to visit all airports if it is possible; otherwise, indicate not possible. As for option 9, enter airport code and remove it if applicable. Display a message "Unavailable" for options 6, 7, 8, and 9 unless you are doing extra credit.

Make sure to exercise different situations to convince yourself that your program is working correctly. Draw the original digraph and the revised digraph after its revision for the main test cases (refers to a US map or gets a US map if you need to so your diagram is easy to follow). In addition, refer to the digraph as you annotate each test case (more points are assigned to test cases). **You must run the following test cases in this order as one set of test cases and add more test cases as needed:**

1. Run option 0
2. Display airport information for SFO
3. Find the cheapest flight from LAX to JFK
4. Find the cheapest flight from JFK to LAX
5. Delete a flight between LAX and SFO
6. Add a flight UA8888 from DFW to JFK for $200.00
7. Add a flight UA7777 from JFK to MSY for $199.00
8. Find the cheapest roundtrip from LAX to JFK
9. Run option 0
10. Quit

Extra Credit: You can earn up to 5 extra points if you can implement two items of options 6, 7, 8, and 9. Please clearly indicate which extra credit features that you are implementing. Run the following test cases on original graph for implemented option as applicable.

- Option 6 – JFK to SFO
- Option 7 – JFK to MSY
- Option 8 – start from LAX
- Option 9 – delete MSY and then run option 0

Team Project: It is not required but it is highly recommended that you work in a team of two members. You must sign up by Monday, 05/18/20, for MW class or Tuesday, 05/19/20, for TTh class if it is going to be a team project. Make sure to split the responsibilities equally and work together. You do need to provide a short description on who did what and your experience as a team project. It is possible that all team members may not receive the same score. **If it is a team project, you need to implement two**

items of extra credit options as a required component and you can optionally implement two more items for extra credit.

Alternate Project: It is not required but it is highly recommended that you work in a team of two members with one comfortable with C++ and the other with Java. You will convert Graph files in Java book to C++. It should be like the files given for lab 8, but it must be a template version. The files are Edge.java, Vertex.java, Graph.java, AdjacencyMapGraph.java, GraphAlgorithms.java, and GraphExamples.java. The implementations for these files do not have to be exact since there are different features for the two languages such as interfaces and standalone functions, but should be as close as possible. A map or a list can be used to store the adjacency list. You should use existing STL such as map, list, and priority_queue as supporting data structures. Make sure to include a driver to test various algorithms. You can earn up 5 additional extra credit points by including good test cases to show that your implementations are correct.

Please provide documentation and applying good coding style because it is part of the grade. You must come up with enough test cases since the test cases are also part of the grade. Please submit the following items (items 1 – 5 as one pdf file on Canvas, item 6 as one zip file so 2 files):

1. Title page with name, class, project number, and relevant information about your program (compiler and system used, file names).
2. Notes about your program (status of your program and lessons learned at the minimum).
3. Drawing(s) of your graph and annotated test cases.
4. Printouts of any input/output.
5. A printout of the source code.
6. A copy of your source code on Canvas (.h /.cpp or .java file)

Your program will be graded as follow:

- Correctness/Efficient: 35 points
- Test Cases: 7 points
- Documentation/Coding Style: 8 points

```
P4Airports.txt
LAX    Los Angeles
SFO    San Francisco
DFW    Denver
ORD    Chicago
BOS    Boston
JFK    New York
MIA    Miami
MSY    New Orleans
SEA    Seattle
```

P4Flights.txt

```
LAX   SEA    199.99  UA1234
LAX   DFW    189.00  AA1000
SFO   LAX     79.00  UA2000
DFW   LAX    199.00  AA2000
DFW   SFO     99.99  DL4321
ORD   DFW     50.00  UA9999
ORD   BOS    179.00  UA3000
BOS   ORD    149.00  UA4000
BOS   JFK     99.00  JB2345
JFK   ORD     99.00  JB5432
JFK   MIA     49.00  UA5000
JFK   MSY    220.00  DL3555
MIA   MSY     50.00  DL6789
MSY   LAX    190.00  SW6000
MSY   DFW    109.00  SW7654
SEA   ORD    179.50  UA5430
```
