Flytrap Airlines

It's about time to think about heading home for the holidays (or at least somewhere warmer). To facilitate your travel plans, your job this week is to implement part of a flight reservation system for Flytrap Airlines (or FTA). The goals of the lab are the following:

- To gain experience manipulating graph data structures.
- To use Dijkstra's algorithm to compute shortest paths.

Since the end of the semesters near, this lab is due by Friday at midnight. You should be able to finish it during the lab time on Wednesday without trouble. However, feel free to take until Friday if you wish to add features or experiment further with graphs. The FTA reservation system implements six commands:

- help: prints out a help message.
- quit: quits the program.
- airports: prints all airports serviced by FTA.
- flights <airport1> <airport2>: prints all direct flights from one airport to another.
- distance <airport1> <airport2>: prints the shortest path from one airport to another, using only routes flown by the airline.
- trip <airport1> <airport2> <time>: prints an itinerary for flying from one airport to another on flights flown by FTA. It should print the flight that leaves the original airport after the given time and arrives at the destination as early as possible.

Here is a sample run of the program with a small data set containing nine cities and roughly 500 flights.

```
moo:~/home/cs136/fta> java FlytrapAirlines small
enter command> airports
ALB: Albany, NY
DEN: Denver, CO
DFW: Dallas/Ft Worth, TX
IAD: Washington [Dulles], DC
JFK: New York [Kennedy], NY
ORD: Chicago [O’Hare], IL
PDX: Portland, OR
SEA: Seattle, WA
SFO: San Francisco, CA
enter command> flights ORD ALB
FTA 7600 ORD -> ALB 800 until 1056 (116 min)
FTA 7598 ORD -> ALB 1050 until 1347 (117 min)
FTA 388 ORD -> ALB 1320 until 1624 (124 min)
FTA 1448 ORD -> ALB 1750 until 2053 (123 min)
FTA 1108 ORD -> ALB 1805 until 2111 (126 min)
FTA 1572 ORD -> ALB 2045 until 2348 (123 min)
FTA 1070 ORD -> ALB 2055 until 2411 (126 min)
enter command> distance ALB PDX
```
Total Distance is 2391
   ALB -> PDX 2391
enter command> distance SFO ALB
Total Distance is 2564
   SFO -> ORD 1843
   ORD -> ALB 721
enter command> trip SFO ALB 600
    FTA 132  SFO -> ORD  630 until 1228  (238 min)
    FTA 388  ORD -> ALB 1320 until 1624  (124 min)
enter command> trip SFO ALB 1600
    FTA 212  SFO -> IAD 1245 until 2043  (298 min)
    FTA 7843 IAD -> ALB 2120 until 2243  (83 min)
enter command> trip SFO ALB 1200
    FTA 7819 ALB -> IAD 1025 until 1150  (85 min)
    FTA 353  IAD -> DEN 1155 until 1334  (219 min)
    FTA 523  DEN -> PDX 1425 until 1604  (159 min)
enter command> quit

Your job will be to implement the distance and trip commands. Most of the code has been provided for you, including a basic implementation of Dijkstra’s algorithm. There are two FTA data sets for you to use—small and large. Use the small data set while developing your code. Once it is working, you can use the large data set, which has roughly 400 airports and 7,000 flights.

Program Structure

The starter code contains several classes for you to use.

- **Airport**: This is basically an association to map airport codes (ALB, SFO, etc.) to cities (Albany, San Francisco, etc.).

- **Flight**: This represents one daily flight on FTA. A flight has a departure and arrival Airport, a flight number, a departure and arrival time, and a duration (in minutes). Departure and arrival times are encoded as integers. For example, 854 represents 8:54 a.m. and 1354 represents 1:54 p.m. Arrival time expressed in the local time of the arrival airport. For flights arriving after midnight, the arrival time will be greater than 2400. Because flights cross time zones, the arrival time may be different than just a departure time plus the duration.

- **Route**: A Route represents one flightpath serviced by FTA. It contains the distance between the arrival and departure airport, and a vector of Flights operating on that route. The flights are stored in an ordered structure, sorted by departure time.

- **FlytrapAirlines**: This class contains the schedule for the airline. The schedule instance variable is a graph whose vertices are labeled with Airport objects and whose edges are labeled by Route objects. In addition, the airports instance variable keeps an ordered list of Airports so that we may easily print an alphabetical list of airports to which FTA flies. This class already contains code to read in the data files, build the schedule, and process some of the commands. The airports command iterates over the airports structure, printing out the Airport objects. The flights command looks up the graph edge connecting the departure and arrival of airports and prints the ordered vector of Flights. The code contains more detail about what exactly you should change.

The javadoc for the files is available from the class handouts page. Look through this documentation before you start to work.
Programming Tasks

1. I have provided a basic implementation of Dijkstra’s algorithm to compute the shortest distances from a source Airport to all reachable destinations. This method has the following signature and is virtually identical to the code in the book:

```java
/**
 * An implementation of dijkstra’s algorithm to compute route
distances. You should not modify this method.
 * @pre g is a scheule graph and start is a non-null Airport
 * @post returns a map from Airport to (distance, previous-edge)
 * Associations.
 */
public Map dijkstra(Graph g, Airport start)
```

This method returns a Map, which has Airports as keys and Associations as values. The Associations store pairs of the form (distance, previous-Edge). The edge is the last edge on the shortest path from start to the Airport in question. The Association for the start is (0, null). Your first task is to finish implementing the distance command by writing a method with the following signature:

```java
/**
 * @pre distances is a map from Airport to (distance, previous-edge)
 * Associations.
 * @pre destination is the end of the route we are printing.
 * @post Prints out the route distances from the source to destination
 * by following the previous edges back to the source.
 */
protected void printShortestPath(Map distances, Airport destination)
```

This method should take the Map returned from Dijkstra’s algorithm and print out the edges on the path from the start to the destination. You will need to trace back through the edges in the associations stored in the map to do this.

2. Your second task is to implement the trip command. You will use a second version of Dijkstra’s algorithm that computes shortest paths based on arrival time rather than distance. In other words, the shortest path from one Airport to another is a series of flights where the last flight arrives at the destination earlier than the last flight on any other path from the start. Here is the signature of this method:

```java
/**
 * An implementation of dijkstra’s algorithm to compute earliest-arriving
 * itineraries. You should modify this method to enqueue new possible
 * itineraries into the priority queue.
 * @pre g is a scheule graph and start is a non-null Airport
 * @post returns a map from Airport to (arrival-time, previous-flight)
 * Associations.
 */
protected Map dijkstraEarliestArrival(Graph g, Airport start, int time)
```

The time parameter indicates the earliest time that you may leave the start airport. To support this type of search, the Map returned from dijkstraEarliestArrival should map Airports to (earliest-arrival-time, Flight) associations. You need to add code to dijkstraEarliestArrival to insert new potential paths into the priority queue according to the arrival time metric. In
order for an itinerary to be valid, each flight segment should depart after the previous flight has arrived. In reality one would want perhaps a half hour to switch planes, but ignore lay-overs on FTA.

Once you have completed the modified Dijkstra's algorithm, you should write the following method to print out an itinerary:

```java
/**
 * @pre earliestArrivals is a map from Airport to (arrivalTime, previous-Flight) Associations.
 * @pre destination is the end of the route we are printing.
 * @post Prints out the flights from the source to destination (by following the previous flights back to the source.
 */
protected void printItinerary(Map earliestArrivals, Airport destination)
```

This method should be almost identical to printShortestPath.

### Notes

1. Copy the following starter directory (don’t forget to run chmod u+w after copying):

   ```
   /private/Network/Servers/cortland.cs.williams.edu/Volumes/Courses/cs136/labs/lab11
   ```

   You can copy and change permissions on the whole directory with the following commands:

   ```
   cp -r /private/Network/.../cs136/labs/lab11 .
   chmod -R u+w lab11
   ```

2. The javadoc for the files is available from the class handouts page. Look through this documentation before you start to work.

3. Submit your modified FlytrapAirlines.java file with turnin no later than midnight on Friday, December 5. This is a strict deadline.

4. Possible extensions if you wish to explore graphs further:

   (a) Handle lay-overs so travellers will have time to make connections.

   (b) Find all reachable airports from an airport.

   (c) Find the longest route or itinerary with the largest number of connections. Dijkstra's algorithm will not work for these— why? However, you have already seen the basic algorithm for doing this.

   (d) Anything else you find interesting.