Lab 11

Due 11:59pm, 14 May (Saturday)

Handout 14
CSCI 136: Spring, 2005
9 May

Flytrap Airlines

1 Short Answer

Bring answers to these two questions to lab:

15.4 15.7

2 Lab Program

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

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

Since the end of the semester is near, *this lab is due by Saturday at midnight*. You should be able to finish most (if not all) of it during the lab time on Wednesday.

Also, you may (but are not required to) work with a partner in lab this week. Just submit one copy of the lab with both your names at the top of the files.

2.1 The FTA Reservation System

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.

```
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
Not possible.
enter command> trip SFO ALB 1200
   FTA 212             SFO -> IAD          1245 until 2043       (298 min)
   FTA 7843            IAD -> ALB          2120 until 2243       (83 min)
enter command> trip ALB PDX 1000
   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 `airports`, `flights`, `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.

2.2 Program Structure

The starter code contains several classes for you to use.

- `Airport`: An airport object just contains an airport code (ALB,SFO, etc.) and a city name (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. Since flights cross time zones, the arrival time may be different than just a departure time plus the duration.

- `Route`: A `Route` object 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 a map from airport code to `Airport` objects. We can use this to, for example, print an alphabetical list of airports or convert a string like "SFO" to the `Airport` object stored in the `schedule` graph. This class already contains code to read in the data files, build the schedule, and process some of the commands. 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.

2.3 Programming Tasks

1. The starter code and javadoc for the files is available on the class handouts page. Look through this documentation before you start to work.
2. Implement the `airports` and `flights` commands. These commands should just extract the appropriate information from the `schedule` graph and `airports` map.
3. Next, complete the implementation `distance`. The main routine to perform this operation is `printDistance`, which uses two helper methods. The first, which I provide, is 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:

```
/**
 * An implementation of Dijkstra's algorithm to compute route
 * distances.  You should not modify this method.
 * @pre g is a schedule 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`). To finish implementing the `distance` command, you must write a method with the following signature:

```
/**
 * @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.

4. Your last major task is to implement the `trip` command. The algorithm will be structured in the same way as `distance`. You will need to write 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:

```
/**
 * 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 schedule 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 for now.

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

```
/**
 * @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`.

5. Once it is working, change your program so that passengers are always given at least 30 minutes between landing at an airport and departing from it. Be careful of how you do the math on numbers representing time.

6. Feel free to add additional features to your system. Possible extensions include the following.

   (a) Add commands to add/remove airports, add/cancel specific flights, etc.

   (b) Find the *longest* route or itinerary with the largest number of connections. Dijkstra's algorithm will not work for these— why?

   (c) Anything else you find interesting.

2.4 Deliverables

Submit your modified `FlytrapAirlines.java` file with `turnin` no later than midnight, Saturday. This is a strict deadline.