Lab 13 — Airline Routes

In this lab, we will develop a program to display an airline route map as a graph, using real-world data obtained from the website http://openflights.org. Unfortunately this data is several years out of date, but it will have to do for now. To get started, download **lab13_files.zip** from the class web page, which contains our Graph implementation from class, along with several text files of airline and flight data.

1. The file **airlines.txt** contains a list of 2-letter airline codes for 80 airlines (many of which are no longer around). The file **airports.txt** contains information for 775 U.S. airports, organized into four fields: the official 3-letter airport code, the latitude and longitude of the airport (in that order), and the name of the city or town the airport is located in. Finally, **routes.txt** contains over 7800 U.S. airline routes, each one specified by its origin and destination airport codes, the distance of the route in miles, and the code of the airline serving the route. Open these files and examine their contents.

2. Open **Vertex.java** and examine the code. This is the code we developed in class, with a few new instance variables and methods added at the top. The `x` and `y` instance variables will be used to store the longitude and latitude information for an airport. We will use the getPredecessor() method later. Also examine **Graph.java**, which contains a new version of the addVertex method that accepts *x* and *y* values in addition to a label string. The **Airport.java** class definition will be used to hold all of the information for an individual airport (its 3-letter code, longitude, latitude, and city). **Flights.java** will be the main program.

3. Before we create any graphs, it will be helpful to first build a dictionary of airport information from the file **airports.txt**. Add the following code to the main program in Flights, which will read in the information from the airports.txt file and build a HashMap (i.e., a dictionary) called *info* containing 3-letter airport codes as the keys, and Airport objects as their associated values:

```
Scanner in = new Scanner(new File("airports.txt"));
Map<String,Airport> info = new HashMap<String,Airport>();
while (in.hasNextLine()) {
    String airportCode = in.next();
    double latitude = Double.parseDouble(in.next());
    double longitude = Double.parseDouble(in.next());
    String city = in.nextLine().trim(); // removes extra whitespace
    Airport airport = new Airport(airportCode, longitude, latitude, city);
    info.put(airportCode, airport);
}
in.close();
System.out.printf("Read information for %d airports\n", info.size());
```

   Your resulting dictionary should contain information for 775 airports. To retrieve information about a specific airport from the dictionary, you can use the `get` method as shown in the examples below:

`info.get("JFK").getLongitude()`	should return longitude -73.7789 for JFK airport
`info.get("JFK").getLatitude()`	should return latitude 40.6398 for JFK airport
`info.get("JFK").getCity()`	should return the string "New York"

4. Next, add similar code to build a dictionary called *airlines* from the file **airlines.txt** that stores 2-letter airline codes (such as "AA") as the dictionary keys, and airline names (such as "American Airlines") as their associated values. Your dictionary should be of type HashMap<String,String>. For example, calling `airlines.get("AA")` should return the string "American Airlines".

5. Now we are ready to build a Graph to represent the flight network of an airline chosen by the user. Your program should ask the user for a 2-letter airline code, and then open the file **routes.txt** and read in each line of the file, skipping those lines with an airline code that doesn't match the one specified by the user. For the lines that do match, the origin and destination airport codes should be added to the Graph as new vertices, and a new directed edge should be added between them, using the distance as the edge cost. However, a new vertex should be added ONLY if it does not yet exist! You can use the Graph method **containsVertex** to check if the origin or destination airport is already in the Graph, as in the example code below:

   ```
   if (!graph.containsVertex(originCode)) {
       double longitude = info.get(originCode).getLongitude();
       double latitude = info.get(originCode).getLatitude();
       graph.addVertex(originCode, longitude, latitude);
   }
   ```

6. Test your program by building a flight graph for the very small (and now defunct) airline Compass Airlines, which has airline code CP. This airline serves only two routes: from Aberdeen (ABR) to Pierre (PIR), South Dakota, and back again, 117.4 miles each way. Printing out your Graph object using System.out.println should produce the output below:

   ```
   Vertex: ABR
      Edge: ABR--117.4-->PIR
   Vertex: PIR
      Edge: PIR--117.4-->ABR
   ```

   As another test, try USA3000 Airlines, with airline code U5. This airline serves 8 airports with 14 routes, between Baltimore (BWI), Chicago (ORD), Cleveland (CLE), Detroit (DTW), Pittsburgh (PIT), Philadelphia (PHL), St. Louis (STL), and Ft. Myers, Florida (RSW). The line below will report the total number of vertices and edges in the graph:

   ```
   System.out.printf("%d airports and %d routes in route graph\n",
                     graph.getVertices().size(), graph.getEdges().size());
   ```

   Also try Sun Country Airlines, with airline code SY. This airline serves 14 airports and 27 routes, with Minneapolis-St. Paul (MSP) as the main hub.

7. You can visualize your flight graphs more clearly by using the pre-compiled class GraphWindow, which creates a pop-up map of the USA showing the geographical locations of each vertex in the graph, along with the edges between vertices. Add the code below to your program, which creates a window of size 990 × 550 pixels and sets the window coordinates to the corresponding longitude (x) and latitude (y) ranges for the continental U.S. It then draws the edges of the graph in yellow and the vertices in white.

   ```
   GraphWindow win = new GraphWindow(graph, "usa.gif", 990, 550);
   win.setCoords(-124.9, -66.55, 25.5, 50.5); // west, east, south, north
   win.drawEdges(Color.YELLOW);
   win.drawVertices(Color.WHITE);
   ```

   Now try some larger airlines, such as Delta (DL), American (AA), United (UA), JetBlue (B6), Southwest (WN), Frontier (F9), Spirit (NK), Midwest Airlines (YX), Virgin America (VX), or even Great Lakes Airlines (ZK) – which seems to be based in Denver of all places!