COP 3503 - Programming Assignment #3
A Simple Reservation System for an Airline
Total: *10 points.*
Assigned: Feb 21, 2012 (Tuesday)
Due: March 13, 2012 (Tuesday) at 11:55 PM WebCourses time

Objective

Create a Graph Data Structure, implement traversal techniques for finding connected components, shortest path in weighted graph, etc.. Implementation will require use of disjoint sets, hashing and priority queue.

Problem: A Simple Flight Reservation System for a Hypothetical Airline: A hypothetical low cost airline (HypAir), wants us to develop a reservation system. HypAir has scattered flight coverage in US. It has flights connecting many cities in South East region, North West region and West region, but does not have flight connecting one region to another. It seems that the airline does not care about how one will commute between cities that are in two different regions.
HypAir has a fixed number of flights with fixed time.

HypAir requires us to support the following query in their reservation system.

Given two cities find if they have any direct or indirect connection.
If the connection exists between the specified cities then find and print the flight details that would fly someone from the first city to the second city (a) in cheapest price and (b) in least number of hops.
In each case you will show the intermediate cities if any.

HypAir insists that the system implementation must strictly follow the following instructions.

Instructions:

- Underlying Data Structure: You are required to solve the problem using Graph data structures and its associated algorithm. Cities will be vertices, and flights will be directed edges. Each flight will have an originating city and a terminating city, a flight number, start time, duration, cost. A flight will be a directed edge, will appear in the adjacency list of the originative city, and will store the id of the terminating city, and the other details. The resulting graph will be directed weighted graph. The cities will be identified by a name.
- City Name to Vertex Index Mapping: You will use a hash function for mapping a city to a vertex index. In the beginning you may use Java HashMap for the purpose. However, you are required to replace it with your own implementation of HashMap class. (Hashing will be taught before the submission deadline.) The class should cover at least the following two methods:
    - put(<city name>, <id>): to insert the mapping between city name and vertex id number. ex: map.put("Orlando", idSoFar)

    - get(<city name>): to return a valid vertex id associated with the city name if the city exists in the Map, otherwise returns an invalid id (0 if the ids start with 1, -1 if the ids start with 0).
      ex: if (map.get("Orlando")==0){ idSoFar = idSoFar +1; map.put("Orlando", idSoFar)}.
- Connection between cities: Cities may belong to different regions, and hence may not have connection. So the first part of the query is to find out whether cities are connected. To handle this part of the query, you will run a traversal algorithm (BF or DF) to identify the connected components, and you will store the vertices of the connected components in a disjoint sets data structure. You will use "merge tree by height" operation for set union. This traversal and disjoint sets computation may be done at a preprocessing step or may be done during the very first query. Once the connected components are found, for every query, you will simply find out the disjoint set (findSet operation) to which each of the two cities belongs. If they belong to the same set then they have a connection between them, otherwise, they do not have any connection. The preprocess (or the first query) will incur the cost of the graph traversal and disjoint set building. The next query onwards, the cost should simply be O(logN).
  NOTE: You must not carry out connected component finding and disjoint building for every query. It must be done only once.
- Find the Minimum Hop Flight Path: You will apply shortest path finding in unweighted graph (Breadth First Traversal variation) algorithm to find the minimum hop connecting the origin city to the target city. Once the target city is found you will terminate the finding algorithm, print the path, the cost of traversal and any other detail that the requirement demands.
- Find the Cheapest Connecting Flight Path: You will apply shortest path finding in weighted graph (Dijkstra's algorithm: will be taught soon) algorithm to find the cheapest path connecting the origin city to the target city. Once the target city is found you will terminate the finding algorithm, print the path, the cost of traversal and any other detail that the requirement demands. You may use the indirect priority queue from Assignment 3 or use Java Priority queue for finding the vertex with minimum path length so far.

Input

The program must accept input from file **Assignment04.txt** with following information:

- 1st line: <N: the number of cities>
- 2nd line: <M: Number of flights>
- Followed by M lines of flight information: <originating city name> <terminating city name> <flight #> <originating time> <duration> <cost>
- Followed by a list of queries (one line per query)
- <start city name>, <end city name>
- …

Output

For every query:
"No path" if no path exists between the cities
or
"Minimum Hop flight:" followed by path and cost, and then " Minimum Cost flight:" followed by path and cost.

Deliverables

You must submit the source code (The .java files, not the .class files) for your programs, and report over WebCourses by 11:55 PM on Thursday, March 13, 2012. You must upload your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will not be accepted.
The package must contain:
- Assignment04.java, the main java files and any additional for your assignment.
- Assignment04.pdf: the writing component of your assignment. This document must contain three sections.

(a) Status of the project

(b) Describe your implementation of HashMap class with pseudocodes for the two required methods put() and get().

(c) Pseudocode of the two find path algorithms between the cities. Note the path finding algorithm must terminate after it has found the shortest path to the target.

Restrictions

- Follow posted assignment submission guideline.
- Your program must accept the input file (**Assignment04.txt**) from the same directory as where the Java source file is placed. NOTE: Do not code any absolute directory path.

Grading Policy: Total 10 points

You program must compile and run. (0 credit for code not compiling or not running.)
This submission will be graded from a total of 10 points using the following criteria:

- Create your own HashMap implementation for mapping between cities and vertex id. (2 points)
- Read the input file and create the graph. (1 point)
- Preprocess for Connected Component Finding and storing in disjoint sets and Finding if connection exists between two cities by findSets operation (1+1+1= 3 points)
- Shortest path in unweighted graph (1 point)
- Shortest path in weighted graph (2 points)
- Report: (1points)