Assignment 8 – Path Reconstruction

Description:

This will be a continuation of assignment 7. The final problem we will be looking to solve is path reconstruction. Our code is capable of determining what the shortest path is between two stations, but it can't tell us what routes we need to take. We will write some code to do this for us.

We can implement the following pseudo code to achieve this goal:

```
let dist be a |V| × |V| array of minimum distances initialized to ∞
(infinity)
let next be a |V| × |V| array of vertex indices initialized to null

procedure FloydWarshallWithPathReconstruction ()
   for each edge (u,v)
      dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
      next[u][v] ← v
   for k from 1 to |V| // standard Floyd-Warshall implementation
      for i from 1 to |V|
         for j from 1 to |V|
            if dist[i][k] + dist[k][j] < dist[i][j] then
               dist[i][j] ← dist[i][k] + dist[k][j]
               next[i][j] ← next[i][k]

procedure Path(u, v)
   if next[u][v] = null then
       return []
   path = [u]
   while u ≠ v
       u ← next[u][v]
       path.append(u)
   return path
```

This code is the exact same as the Floyd-Warshall we've been using. However, the differeneces have been highlighted in red.

At a minimum, the following public functions need to be implemented:

```
Graph ( int routes, int stations )
```

Initialize total number of routes and stations

```
void addRoute( int from, int to, int weight )
```

Add route to adjacency matrix with appropriate weight. Set next matric appropriately

```
bool isRoute( int from, int to )
```

Return bool value indicating existence of route.

```
void calcShortestRoutes( void )
```

Run Floyd on the adjacency matrix to solve all-pairs shortest path

```
int shortestRoute( string src, string dst )
```

Return the shortest route between src and dst stations

```
void printPath( string src, string dest )
```

Print the stations that make up the shortest path between two stations

You may need to add more private/public functions in order to accomplish your task. This assignment should build on your work from assignment 7 so there is no need to completely rewrite that code.

Remember; always know what type of graph you are using. Your program should be built to handle a weighted, directed graph.

Your program will still be expected to handle station names instead of relying on integer identifiers. Therefore, you will need to read data from two separate files and keep track of the information appropriately.

The format of the first file will remain the same as assignment 7 so you shouldn't have to rewrite any of that code.

It will still be formatted as follows:

```
5 6
0 1 7
3 1 10
4 3 12
1 4 12
4 2 32
1 2 3
```

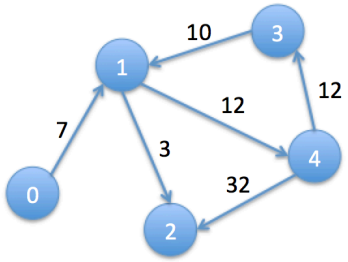
The first line of the file will contain two integers indicating the number of stations and routes. Following this first line will be n (n being equal to the total number of routes) lines representing different routes and their associated weights. For example, line 2 of the above example indicates that there is a route from station 0 to station 1 with a weight of 10. The data from this file should be used to initialize and populate your graph.

The second file will contain a list of station ids to station name mappings. It will be formatted as follows:

```
0 Red
1 Green
2 Blue
3 Purple
4 Yellow
```

It will contain n lines (n corresponding to the number of stations read from the first file) where each line contains a number and a name. The number maps a name to the identifiers used in the first file.

For example, the previous two example files would represent the following graph:



0	Red
1	Green
2	Blue
3	Purple
4	Yellow

You will also need to implement some sort of menu that will allow users to interact with your program.

The following is an example program execution using the previous two example files:

```
$ ./a.out
-- Menu --
     0 => Exit
     1 => Print Menu
     2 => Check if path exists
2
From what station are you leaving? Red
To what station are you traveling? Yellow
There is a route between Red and Yellow with a minimum distance
of 19 miles.
```

```
Your path will be Red->Green->Yellow
2
From what station are you leaving? Yellow
To what station are you traveling? Red
There is no route between Yellow and Red
0
```

You can be creative as you want with your menu, but at a minimum it must allow the user to exit, print the menu, and determine if a path exists between two stations. If a path does exist, your program needs to respond with the shortest route and the path.

Due:

December 16th, 2016 11:59 PM