COL106 - Data Structures and Algorithms
Assignment 6

Courtesy: Univ of Washington https://courses.cs.washington.edu/courses/cse373/13wi/homework/8/spec.pdf

Due: 19 November 2014, 11:55 pm

Total Marks: 30

This assignment focuses on implementation of graph path searching algorithms and using those algorithms to perform searches on interesting data. You will have to add certain features to a Mini-Facebook application. Friendship data can be represented as a graph of persons, with edges connecting pairs of persons who are known to each other. The edges are weighted according to the strength of the relationship between two people. For the purpose of this assignment, we consider three types of relationships – *Acquaintance*, *Friend* and *CloseFriend*, in decreasing order of weight.

1 Features

1.1 Erdos Number

Paul Erdos was a renowned mathematician, who had collaborated with a large number of people. This has led to a metric known as the Erdos Number among mathematicians. It describes the colloborative distance between Dr. Erdos and a given mathematician. You are expected to add a similar feature to Mini-Facebook.

The Erdos number is defined as follows :

- The Erdos Number of "Erdos" is zero.
- The Erdos Number of any other 'X' is the minimum of the Erdos Numbers of X's friends plus 1.

The Breadth-First Search algorithm (BFS) is to be used to compute the Erdos Number. Complete the function **shortestPath(v1, v2)** – you must employ BFS to compute the shortest path (in terms of number of edges) from *v1* to *v2*. The length of the path gives the Erdos Number. The function must return a vector of pointers to the vertices in the path from *v1* to *v2* (both included). If no path exists, return NULL.

1.2 Reachability

Given two users *v1* and *v2*, it is of interest to know if there a chain of relationships we can follow to reach *v2* from *v1*. Complete the function **isReachable(v1, v2)** – you must employ the Depth First Search algorithm to see if *v2* can be reached from *v1*. The function must return true if *v2* is reachable, false otherwise.

1.3 Friend Suggestions

Two users who have a mutual close friend are more likely to become friends with each other than two users who have a mutual acquaintance. The likelihood that two users will become friends depends upon the minimum weight path from one to the other. Complete the function **minimumWeightPath(v1, v2)** – you must employ the Dijkstra's algorithm to compute the single source minimum weight path from *v1* to all vertices. The function must return a vector of pointers to vertices that make up the path from *v1* to *v2* (both included). If no path exists, return NULL.

2 Submission

Please download the required files from the link given on the course webpage. You are required to complete the **TODO** sections in the file `Graph.cpp`.

To run, first run "make". Then, "./a.out <absolute-path-to-database-file>".

A sample database file is provided – `facebook-graph.txt`. A line can be in one of two formats – one, "<user-1><tab><user-2><tab><relationship>", where relationship can be one of *Acquaintance*, *Friend* or *CloseFriend*, and two, "<user-1>" to specify a single vertex. (While adding Erdos to your database, spell it exactly as "Erdos"). You are required to make your database files to test your implementations.

Submit only `Graph.cpp` and all the database files that you create in a zipped format. Name your zipped file as ⟨EntryNumber⟩_⟨FirstName⟩_⟨LastName⟩.zip.

3 Guidelines

- All basic functionalities of a graph have been implemented. You will most likely require the use of :
  - **neighbors(v)** : returns a vector of pointers to all vertices x, such that an edge from v to x exists.
  - **vertices()** : returns a vector of pointers to all vertices
  - **edgeWeight(v1, v2)** : returns the weight of the edge from *v1* to *v2*.
  - **containsVertex(v)** : returns whether the graph contains vertex *v*.
  - **containsEdge(v1, v2)** : returns whether the graph contains an edge from *v1* to *v2*.
- You have to handle all corner cases appropriately.
- You may use any of the containers from the C++ Standard Template Library such as vectors, queues etc.
- If you require to print information for debugging purpose,

```
Vertex* v;
.
.
.
cout << ((FacebookVertex*)v)->toString();        //to print the vertex v
Graph* g;
.
.
.
cout << g->toString();                           //to print the graph
```

- Use the macro INT_MAX (Graph.cpp:line 10) to denote infinity.

4 Further Practice (No Marks)

Friendship datasets such as Facebook also has to maintain data such as photos, videos, comments, likes, and other such things. You can use data structures of your choice to support as much Facebook functionality as you can. Then these information can be used to enhance the relationship two people have, for example, you can add a relationship like "tagged together", etc.