EE368 Spring 2015

Project 3 (Generating and Analyzing Social Networks):

Assigned Date: March 31, 2015
Due Date: 10:00am April 27, 2015

Note: Satisfactory completion of this project satisfies two ABET curriculum outcomes for this class (outcomes 3, 4). You must earn a score of 60% or higher on this project in order to receive a passing grade for this class.

NOTE: DO NOT SHARE YOUR CODE WITH ANY ONE. ANY SHARING OF CODE WILL LEAD TO AN "F" GRADE IN THE COURSE.

Project Description:

You have recently been hired by Google Inc., as a social network analyst. The company maintains profiles of a large number of users. In this project you need to develop a C program (project3.c) to perform the following two tasks.

(1) Based on the profiles of a given number of users, generate a social network
(2) Analyze this network to compute various social networking parameters

Detail of these tasks is as follow:

1. Generation of a social network:

A Social Network is a graph, $G=(V,E)$, where V is the set of vertices representing users. Each vertex contains one profile. Accordingly, the graph contains number of vertices equal to the number of users/profiles.

A user profile has a unique User ID, followed by a set of eight attributes, representing: `Age, Gender, Marital status, Race, Birth-place, Language, Occupation, and Income`. Assume, all the eight attributes are represented as integers. In essence, a profile is a vector of eight attributes, plus a User ID field.

The attribute dataset for users will be provided in the text file called input.txt. More detail about this file is given on the last page.

In graph $G=(V,E)$, E is the set of links/edges between vertices. An edge defines a friendship relationship between the vertices. The strength of friendship between any two vertices/users (for example with IDs, *a* and *b*) is computed based on the **friendship distance** (L_{ab}) between the profiles of the two users. L_{ab} is computed

based on the eight attributes of the profile vectors, say *a ={a(1), a(2), ..., a(8)}* and *b ={b(1), b(2), ..., b(8)},* as described in the following three steps:

i. First compute the pair-wise **un-normalized Euclidean Distance** UL_{ab} between nodes *a* and *b* as:

$$UL_{ab} = \sqrt{\sum_{i=1}^{8} (a(i) - b(i))^2}$$

ii. Then find $L_{\max}$ which is the maximum over ALL the pair-wise **un-normalized Euclidean** distance values, i.e. UL_{ab} , (can you guess how many such values are there?).

iii. Subsequently, compute the **Friendship Distance** L_{ab} between *a* and *b* as follows:

$$L_{ab} = 1 - (UL_{ab} / L_{\max})$$

Note, 0 ≤ L_{ab} ≤ 1. Also, the value of L_{ab} should be truncated (not rounded up or down) to two decimal positions. It is recommended that the truncated L_{ab} be stored as integers in memory. You only convert it back to float/double number before you print it.

Next, you create an edge between nodes *a* and *b* if L_{ab} is **greater than** a given threshold δ . For your project, at the run time, we will provide two values for threshold, δ_1 and δ_2 , with small and large values, representing a highly active social group (dense graph with many edges) and less socially active group (sparse graph with less edges), respectively.

Once, an edge is created, you must keep its L_{ab} value attached to the edge. In this way you will have a weighted graph, needed for the second part of the project.

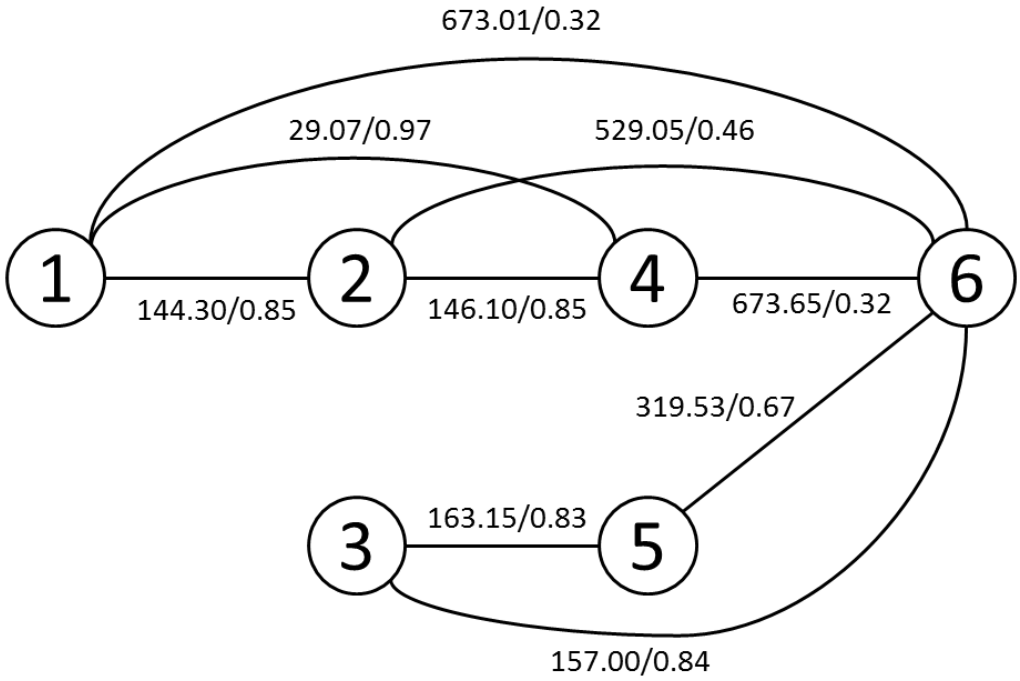
Note: Do not include any self-edges in your graph.

Example:

An example, of a weighted social network (graph G) for six users with their profile dataset is given below. **NOTE, the first value in each line is the USER ID. It is NOT an attribute used for computing the value L_{ab}.**

```
1,39,1,6,1,1,1,3,0
2,33,1,6,2,1,1,10,144
3,40,2,4,1,1,1,6,830
4,10,1,6,1,1,1,1,0
5,21,2,6,1,1,1,3,992
6,39,1,4,1,1,1,6,673
```

Note, the numbers are separated by a single comma, with NO extra space. Each line is terminated by a single '\n' character.



The above graph is created using a similarity threshold value of 'δ = 0.3' among the 6 profile vectors of attributes given above. In this example, for clarity, each edge has been labeled with two values: UL_{ab} / L_{ab}. You can verify that for this example, $L_{\max}$ = 992.16. In this example as we have selected the value of δ = 0.3, therefore, if L_{ab} > δ = 0.3, between any two nodes, we create an edge between those nodes and assign it the corresponding L_{ab} weight.

For part 2 of this project you only need to work with this weight.

(2) Analysis of the social networks:

Using the two weighted graphs (Dense and Sparse social networks) generated in Step 1, for the given value of thresholds, δ_1 and δ_2, you are required to implement the following six queries to determine various social networking parameters for both the Dense and the Sparse Network.

****

Query 1 (20 pints): For a given user ID, called the source node, (provided in the `input.txt` file), find the ID(s) of those nodes which has(have) the minimum path length among all the Shortest Source-Destination paths, computed based on L_{ab} using Dijkstra's algorithm. Also, print the value of this path.

Expected Output of Query 1:
Value of the shortest path length, Sorted Node ID(s)

Example: In the above graph, if the given user ID is 1, the shortest path lengths from 1 to 2, 3, 4, 5, 6 are found to be 0.78, 1.16, 0.64, 0.99, 0.32, respectively. The minimum path length of all the shortest source-destination paths is 0.32, which is the length of the shortest path from 1 to 6, and there is no other node that has the same shortest path length. Thus you should print:

```
0.32,6
```

****

Query 2 (20 points): For the user ID (source node) given in Query 1, and another threshold parameter alpha α, (also provided in the `input.txt` file), find the number of nodes such that their shortest distance from the source node is less than α (Do not print their IDs).

Expected Output of Query 2:
Number of nodes

Example: In the above example, if the given user ID is 1 and α is 1.5. Since all five other nodes have shortest source-destination path lengths less than 1.5, you should print:

```
5
```

****

Query 3 (20 points): For the same user ID (given in Query 1), find the total number and the ID(s) of user(s) who are directly connected to that user (that is, the immediate neighbors)

Expected Output of Query 3:
Number of immediate neighbors, Sorted ID(s) of immediate nodes

Example: In the above example, since 1 has three immediate neighbors 2, 4, and 6, you should print:

```
3,2,4,6
```

****

Query 4 (20 points): For the same user ID (given in Query 1), find the total number and the ID(s) of user(s) who are exactly 2-hops away from that user (i.e. neighbors of the immediate neighbors). These can be called second-level friends. **Note: 2-hops away neighbors should include all nodes that are reachable from the source node in exactly 2 hops, including those that are also reachable in 1 hop, but not including those that are only reachable in 1 hop but not in 2 hops.**

Expected Output of Query 4:
Number of exactly 2-hops away numbers, Sorted ID(s) of these 2-hop nodes

Example: In the above example, from source node 1 we can get to any other destination nodes in exactly 2 hops, therefore you should print:

```
5,2,3,4,5,6
```

****
Query 5 (10 points): Find the average degree of a node (average number of immediate neighboring friends per user/node). **Note: Use double precision variable for accumulation and division, but truncate the result to two decimal positions (not rounded up or down) for printing.**

Expected Output of Query 5:
Average of number of immediate nodes

Example: In the above example, the average degree of a node is 3. You should print:

```
3.00
```

****
Query 6 (10 points): Determine the average number of second-level friends (exactly 2-hops away neighbors) per user/node. **Note: Use double precision**

variable for accumulation and division, but truncate the result to two decimal positions (not rounded up or down) for printing.

Expected Output of Query 6:
Average of number of 2-hops away nodes

Example: In the above example, the average number of 2-hops away nodes is 5. You should print:

```
5.00
```

****

Input File Format (input.txt):

- Number of users, Value of δ_1, Value of δ_2, Node/user ID for Query 1, value of α
- Profile records for all the users (note, the first parameter of each record represents the user ID)

Sample Input file:

```
6,0.3,0.5,1,1.5
1,39,1,6,1,1,1,3,0
2,33,1,6,2,1,1,10,144
3,40,2,4,1,1,1,6,830
4,10,1,6,1,1,1,1,0
5,21,2,6,1,1,1,3,992
6,39,1,4,1,1,1,6,673
```

Out File Format (output.txt):

For both Dense and Sparse networks, the output of Queries 1 -6 must be directed to an output file (output.txt) with each answer printed on one line. First, print all the answers for the Dense network. Then print all the answers for the Sparse network. Insert a single empty line between the answers for the Dense network and the Sparse network. Separate each number by a single comma with NO extra space and end each line with a single '\n' character.

Note: Please follow exactly this output format. Otherwise your submission will NOT be graded.

Sample Output file:

```
0.32,6
5
3,2,4,6
5,2,3,4,5,6
3.00
5.00

0.85,2
2
2,2,4
2,2,4
2.00
2.00
```

Compiling, Execution and Grading

The scoring guidelines above only apply to program that work properly. If your program will not compile you will receive a 0 grade. Be sure that your program compiles successfully with the following command. **You can include math library**.

```
gcc –Wall project3.c -lm
```

- **The command line format for execution is:**

  ```
  executable-name input.txt > output.txt
  ```

  Note: Your program should only take exactly ONE command-line argument, which is the input filename, the output should be printed directly to the screen (stdout). We will redirect your screen dump to an output file for grading. DO NOT use output filename as a second argument to your main function.

- Submit your project using the turnin command from the directory containing your project3.c file:

  ```
  turnin -c ee368ta -p proj3_s15 project3.c
  ```