Programming Assignment #4: Properties of Social Networks

Social networks [1] are graphs formed by relationships among people. They have become important for modern sociology, as well as for economics, anthropology, and other social sciences. As these networks become more complex and extensive, interest has increased in studying them with mathematical and computational methods. Facebook is a recent example of a concrete expansive social network.

One of the more common studies of social graphs is to examine the *centrality* of nodes, as a measure of the *influence* of people. Centrality can be defined in various ways, but high degree of centrality generally indicates that the person has relatively short paths to other nodes. The purpose of this assignment is to compute several different definitions of centrality, and compare them on actual graphs.

We consider only connected unweighted undirected graphs. By *distance* between vertices *u* and *v*, we mean the fewest number of edges on a path from *u* to *v*.

We consider the following definitions of centrality:

1) Degree

The *degree* *d(v)* of a vertex is its number of neighbors in the graph (or, equivalently, the number of incident edges). A vertex is *popular* if it is of maximum degree in the graph.

2) Eccentricity

The *eccentricity* ecc(v) of a node *v* is the length of the shortest path from that vertex to the vertex furthest away from from *v*. A *center* is a vertex of smallest eccentricity.

3) Effective eccentricity

There may be very few vertices that are really far away from a given node. Those could be viewed as outliers. The effective eccentricity eff(v) asks for the 90th percentile of distances from v. I.e., it is the value *k* such that 90 percent of the other nodes in the graph have distance at most *k* from v. More precisely, $\lceil 0.9(n-1) \rceil$ vertices in the graph (other than *v*) are of distance at most *k* from *v*. A vertex is an *effective center* if it is of minimum effective eccentricity.

4) Closeness

The *Hollywood number* H(v) of a vertex is its average distance to the other vertices. We define the *closeness* C(v) of a vertex as the reciprocal of its Hollywood number (i.e. C(v) = 1/H(v)). A *closest* node is one of maximum closeness.

Note: This is related to the concept of *farness* of a vertex, which is its distance sum, of the sum of distances to other vertices. In a graph with *n* vertices, the farness of a vertex equals its Hollywood number times *n*-1. In [1], closeness is defined to be the reciprocal of farness, which differs from our definition.

The Assignment

This assignment should be completed in groups consisting of one or two students.

Part I: Measures of centrality

Implement the following API:

```
public class Centrality {
    public Centrality(Graph G)          // Constructor

    public int degree(int v)            // Degree of v
    public int ecc(int v)               // Eccentricity of v
    public int effEcc(int v)            // 90th percentile distance from v
    public double closeness(int v)      // The closeness of node v

    public int popularVertex()     // Index of a popular vertex
    public int center()            // Index of a center (vertex)
    public int effCenter()         // Index of an effective center
    public int closest()           // Index of a closest vertex
}
```

The first four methods compute the centrality of a given node *v,* for each of the centrality definitions (except betweenness). The last four methods return the index of the vertex with optimal centrality value, for each of the four centrality definitions. Ties should be broken in favor of vertices of smallest index.

Here are the values of these parameters on a small instance:

Node	Deg	Ecc	Eff	Clo
0	2	4	4	0.462
1	2	4	4	0.462
2	4	3	3	0.667
3	3	2	2	0.667
4	2	3	3	0.545
5	2	3	3	0.500
6	1	4	4	0.353

Centrality values of nodes

```
7 8
2 4
2 3
1 2
0 1
3 4
3 5
0 2
5 6
```

tiny.txt

The total complexity of the constructor and the last three methods should not exceed O(V (V+E)). Time-consuming computation should not be repeated. The memory requirements should be linear in V, the number of vertices (besides space for storing the graph G in adjacency list representation).

Add a main routine that reads a graph from the standard input (StdIn) and outputs to the standard output (StdOut) statistics about the centrality parameters of the nodes. Specifically, it should output four lines, one for each centrality definition σ, the centrality values of the vertex that optimizes σ.

Example output output `tiny.out` produces on the sample input `tiny.txt`.

```
                Node Deg   Ecc   Eff  Clo
Popular:   2    4     3     3  0.667
Center:    3    3     2     2  0.667
Eff.ctr:   3    3     2     2  0.667
Closest:   2    4     3     3  0.667
```

The format should match this sample output. (Specifically, use `printf() with formatting fields` "`%5.3f`" and "`%3d`").

Part II: Well-connected actors and movies

The largest connected component in the movie graph (movies.txt) can be found in http://www.ru.is/~mmh/movies2.txt. In the resulting graph, compute the four measures of centrality for Kevin Bacon, Clint Eastwood, and three more actors of your choice.

Part III: Shortest paths

Extend Breadth First Search computation to actually compute the *number* of shortest paths from the starting vertex *s* to all other vertices. If P(s,v) is the number of shortest *s*-*v* paths, and *d(s,v)* is the distance between *s* and *v* (i.e., the length of the shortest *s*-*v* path), then it should satisfy

$$P(s,v) = \sum_{\substack{w \\ d(s,v)=d(s,w)+1}} P(s,w)\,.$$

This can also be used as a rule for computing the number; the important thing then is to ensure that we complete computing P(s,w) before computing P(s,v), whenever *w* is closer to *s* than *v* is.

Use this to compute the maximum number of shortest paths from node 0 to any other node in the graph (i.e., $\max_v P(0,v)$). Include this in the output of the main function in the Centrality class.

Example output on tiny.txt:

```
Node 0 has 1 shortest paths to node 0
```

Input and output examples

Example input/output pairs are given: `tiny.txt/tiny.out` and `mediumG.txt/medium.out`.

Submission

Submit a report - as a PDF file - that describes briefly and clearly your methods for solving the tasks detailed above, how you implemented the methods in the programs you developed and the results of running your programs. It should not exceed 1 page. Run your program for Parts I and III on `tiny.txt`, `mediumG.txt`, and your program for Part II on `movies2.txt`. We will additionally test your programs on two more instances.

Include with the report your source files and input/output files. This assignment will not be submitted via Mooshak.

Note: Your solutions should be professionally developed, readable and clear. This will show the world that you do care about the quality of your work and will help the people evaluating your solutions understand them efficiently. Happy evaluators give better and more accurate marks!

Remember that the solutions you deliver must be your own work. See the university's rules on plagiarism here. Acknowledge all sources, including others in the class from whom you obtained ideas.

Deadline for submission: Wednesday, 21 November 2012, 23:00

Grading scheme

Part I : 40% (correctness 25%, efficiency 10%, AP + code clarity 5%)

Part II: 15%

Part III: 25% (correctness +efficiency 20%, organization + clarity 5%)

Report quality: 20% (spelling and grammar, clarity, organization, succinctness)

References

[1] Centrality. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Centrality. Accessed 11 November 2012.

Bonus problem (worth up to 15%): Betweenness

The *betweenness* of a vertex tries to capture that a node is influential if many shortest paths (between other nodes) go through the vertex. Specifically, the betweenness of vertex *v* is the sum over all node pairs *u*, *w* (where neither is *v*), of the number of shortest u-w paths that go through *v*, divided by the total number of shortest *u-w* paths. Formally, let *P(u,w)* denote the number of shortest paths from u to w, and let $P_v(u,w)$ denote the number of shortest *u-w* paths that go through *v*. Then, the betweenness of v is defined by

$$B(v) = \sum_{\substack{u,v \in V \\ u \neq v \neq w}} \frac{P_v(u,w)}{P(u,w)} .$$

A *most between* vertex is one of maximum betweenness value.

Extend the class Centrality to support the method

```
public double betweenness(int v)    // The betweenness of v
```

Develop an efficient method for computing maximum betweenness. In graphs where the average number of vertices on shortest s-t paths is X, the complexity of the method should be $O(X \cdot V^2)$.

Run your method on the example inputs, and possibly on the movie file.