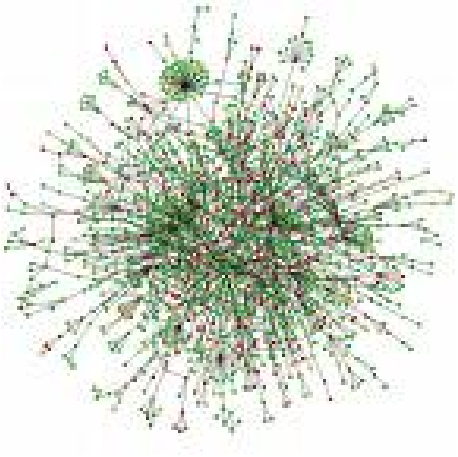
CAS CS 112 – Spring 2008, Assignment 6

due at 10:00 pm on Thursday, May 1



In this assignment we will investigate a real network using some of the techniques we discussed in class. Our network comes from biology: the vertices are Yeast proteins and the edges are physical interactions. You can read more about protein-protein interactions (PPIs) here: `http://en.wikipedia.org/wiki/Protein-protein_interaction`. Our task will be to use some simple techniques to find the "important" vertices (proteins) in the network. Typically, this is referred to as measuring the "centrality" of each vertex in the graph: `http://en.wikipedia.org/wiki/Centrality`.

Building the graph

(20 Points) You will be provided a **network.txt** file which contains the network data in edge-list format. Each line specifies a (tab-delimited) pair (a, b) which corresponds to an interaction between a and b. The **network.txt** file gives an undirected connected graph. If edge (a, b) is specified, so is (b, a), and there are no duplicate edges. The network is very sparse, so we will be using an adjacency list representation to store it. In Java, a convenient way to represent a network is with the following data structure:

```
HashMap<String,HashSet<String>>
```

where each vertex (represented by its name in the form of a String) is mapped to the set of its neighbors.

Your first task is to read in the **network.txt** file line by line and add the corresponding edges to your graph representation. You will find it convenient to use the **String.split()** method to split each line into two Strings on the tab character.

Ranking vertices by degree

(10 points) The simplest way to determine the importance of a vertex is by its degree. For example, in a social interaction network the person who knows the most people may

be important (but of course that is not always the case). Using our graph representation, it is very easy to determine the degree of each vertex (just get the size of its neighbor set). Produce a list of vertices sorted by decreasing degree in a file called **degreeRanking.txt**, where each line contains the name of the vertex and its degree. Here for simplicity it is okay to use a slower ($O(n^2)$) sort.

You should notice that most vertices have very low degree, but there is still a considerable number of vertices with high degree. These high degree vertices are typically called hubs.

Ranking vertices by clustering coefficient

(30 Points) Another (better) way to determine the importance of a vertex is by considering how dense its immediate neighborhood is. One way to measure this is by considering what fraction of its neighbors are connected. This metric is known as the **clustering coefficient**. Formally, for undirected graph $G = (V, E)$, define the neighbor set of $v_i \in V$, denoted by N_{v_i}, to be the set of vertices connected to v_i:

$$N_{v_i} = \{v_j\} : e_{i,j} \in E.$$

.
Then the clustering coefficient of v_i, denoted by $CC(v_i)$, is the fraction of vertex pairs in N_{v_i} that share an edge:

$$CC(v_i) = \frac{|\{e_{jk}\}|}{\binom{|N_{v_i}|}{2}} : v_j, v_k \in N_{v_i}, e_{jk} \in E.$$

Compute the clustering coefficient of each node, and again produce a sorted list (in the same format as before, sorted by decreasing clustering coefficient) in a file called **clusteringRanking.txt**.

Ranking vertices by betweenness

(40 Points) Finally, you will use another centrality metric to rank the proteins. Intuitively, a vertex which is central in a network is traversed on a lot of shortest paths. This is precisely what **betweenness** measures: for vertex v it gives the number of shortest paths that v occurs on (not including paths from and to v of course). You can see that the betweenness score of each node should range from 0 to $\binom{n-1}{2}$, where n is the number of vertices.

Your task is to compute the betweenness of each vertex and produce a sorted list (as before) in a file called **betweennessRanking.txt**. To calculate a betweenness score for each vertex do the following: implement a breadth-first search and each time you run it (you will need to run it from every vertex) trace every shortest path back using the predecessor information, incrementing the counters for the vertices that you encounter. You can use a HashMap to implement the counters. Of course, don't increment the counters for the last and first vertex on the path. You can use another HashMap to keep track of predecessors when you run your breadth-first search.

Helpful Hints

You can use the following code to iterate over the set of Map entries (useful for accessing each vertex and the set of its neighbors):

```
Iterator it = map.entrySet().iterator();
while(it.hasNext()){
      Map.Entry entry = (Map.Entry) it.next();
      String vertex = (String) entry.getKey();
      HashSet neighbors = (HashSet) entry.getValue();
}
```

Also, given vertex "v" you can use the enhanced for-loop to access all of its neighbors:

```
HashSet neighbors = map.get("v");
for(String n : neighbors)
      System.out.println("v is connected to " + n);
```

In each part of the assignment you need to assign every vertex a value (degree, clustering coefficient, betweenness) and then produce a sorted list based on those values. A simple way to accomplish this is to make a Map (containing <vertex, value> pairs) for each ranking and then write a function which takes a Map and prints a sorted list. It is acceptable for your sorting function to just iterate over the Map n times, where n is the initial size of the Map, and each time print and remove the <vertex,value> pair containing the largest value.

Submissions

Submit the three files **degreeRanking.txt, clusteringRanking.txt**, and **betweennessRanking.txt**, along with all your code. Put your code in two files: **GraphToolbox.java**, which contains the object that stores the graph as a private data member, and all the other functionality that you need, and **GraphClient.java**, which creates a **GraphToolbox** object and calls its methods to complete the assignment.

A note about the rankings

You may be curious about what the rankings that you produce are useful for. Ranking proteins in terms of importance based on PPI data is a well-studied problem in bioinformatics. One way to determine the quality of a computationally derived ranking is by doing an experiment such as a gene knockout. A gene (protein) is knocked out and it is observed whether the organism survives or not, if it does not it is labeled a synthetic lethal. Naturally, a ranking which lists the synthetic lethals first is preferred. It turns out that clustering coefficient is the best computational predictor of lethality, followed by betweenness and degree. Of course, other methods which rank vertices (based on the adjacency matrix or the transition probability matrix of the graph) are also applicable here, but in general for the results to be believable one needs to argue about the quality of the interaction data first.