COMP 543 Assignment #2

Due February 17th at 11:55PM.

1 Description

For this assignment, you will be writing a few stored procedures in SQL to analyze a graph data set. The data set to analyze contains a subset of gene-gene interactions from BioGRID, the Biological Repository for Interaction Datasets. The data set has around 7,000 genes and 39,500 relations between those genes. The data set is comprised of two database tables:

```
nodes (id, symbol);
edges (id, refId);
```

The first table gives a unique gene identifier, as well as the symbol for the gene. The second table indicates relationships between the genes (note that references have a direction).

Your task is to write two stored procedures that analyze this data.

1.1 Connected Components

You will first write a stored procedure that treats the graph as being undirected (that is, do not worry about the direction of reference) and finds all connected components in the graph that have between five and eight genes (inclusive), printing out the associated lists of symbols. My implementation found five such connected components in the data set. To refresh your memory, a connected component is a subgraph such that there exists a path between each pair of nodes in the subgraph. Such a subgraph must be maximal in the sense that it is not possible to add any additional nodes that are connected to any node in the subgraph.

The standard method for computing a connected component is a simple breadth-first search. Pick a random starting node, and then search for all nodes reachable from the starting node, then search for all nodes reachable from all of *those* nodes, and then search for all of the nodes reachable from *those* nodes, and so on, until no new nodes are found. The entire set of discovered nodes is a connected component. If there are any nodes that are not part of any connected component analyzed so far, then pick one of those nodes, and restart the computation. You are done when all of the nodes are part of exactly one connected component.

Your program should first compute all of the connected components, and then print out all of the connected components that are have at least five members and no more than eight. When you print out the components, print each gene ID as well as the symbol. Within each component, order the components by symbol name.

1.2 PageRank

PageRank is a standard graph metric that is well-known as the basis for Google's original search engine. The idea behind PageRank is simple: we want a metric that rewards web pages (or in our case, gene interactions) that are often pointed to by other pages. The more popular the page, the greater the PageRank.

To accomplish this, PageRank models a web surfer, starting at a random page, and randomly clicking links. The surfer simply goes to a page, sees the links, and picks one to follow. After each link clicked, there is a probability $1 - d$ that the surfer will jump to a random page; d is called the *damping factor*. A standard value for d is 0.85 (everyone should use this so we are all doing the same thing). Given this setup, the so-called "PageRank" of a web page (or a gene) is the probability that when the user stops clicking (or following citations), s/he will land on the page. Since so-called "sinks" (those pages that don't link

anywhere else) would accumulate all of this probability under the simplest model, it is assumed that those pages with no out-links instead link with equal probability to everyone else.

There are many ways to compute the PageRank of every page (or gene!) in a data set. The simplest is an iterative computation. Let $PR_i(\text{gene}_j)$ denote the estimated PageRank of the gene paper$_j$ at iteration i; assume that there are n gene in all. We start out with $PR_0(\text{gene}_j) = \frac{1}{n}$ forall j. Then, at iteration i, we simply set:

$$PR_i(\text{gene}_j) = \frac{1-d}{n} + d\left(\sum_{k \in \{\text{genes referencing gene}_j\}} \frac{PR_{i-1}(\text{gene}_k)}{\text{num genes referenced by gene}_k}\right)$$

This iterative process is continued until there is only a small movement in probability across iterations. In our case, we'll continue as long as:

$$0.01 < \sum_j |PR_i(\text{gene}_j) - PR_{i-1}(\text{gene}_j)|$$

Your goal for this problem is to write one ore more stored procedures that together compute the PageRank of each of the genes in the graph. You will run your code, and use it to print out the 10 genes with the greatest PageRank, as well as the PageRank for those genes. Again, when you print out a gene, print out both the gene ID and the gene symbol. In this case, order the output in descending order by page rank.

2 Getting Started

First, in your database, create two tables:

```
CREATE TABLE nodes (
  id INTEGER,
  symbol VARCHAR (100));
CREATE TABLE edges (
  id INTEGER,
  refId INTEGER);
```

Once you've done this, unzip the archive provided, and use these two files to load the data into the database.

3 A Note on Speed

It is very important that you try to do as much as possible declaratively. Looping through the contents of a table using a cursor is necessarily going to be slow. You should try to do as much as is possible using declarative SQL queries. Use loops and conditionals to guide the overall control flow, and when there's clearly no way to do what you want using declarative SQL. On this assignment, there's often a 100× or more difference in performance between a well-written code that is mostly using declarative queries, and one written with a lot of loops. Speed does not matter, but it's easy to write a code that is so slow it will not complete in a reasonable time. Not to mention that declarative queries are easier to code and debug!

4 Turnin

Create a document that contains your SQL code, as well as the results from running your code. By 11:55P on the due date, submit this document electronically to Canvas. You can either submit a PDF file, a text file, or copy and paste your results and submit that way. Other formats (such as Microsoft Word) are not acceptable.

5 Academic Honesty

With a bit of searching, you can probably find SQL codes that implement one or both of these algorithms. Since the goal here is figuring out how to do such computations in SQL, and finding an SQL code on the web that does this sort of defeats this goal, We're going to specify that it is not acceptable to examine or otherwise use any SQL implementations of either algorithm—whether an implementation by a classmate, an SQL code in a textbook, or something on the web. However, discussions with classmates are fine, as is examining other SQL codes on the web (that don't implement these two algorithms, or any part thereof). If you are unsure what is allowed, just ask!

6 Grading

Each problem is worth 50% of the overall grade. If you get the right answer and your code is correct, you get all of the points. If you don't get the right answer or your code is not correct, you won't get all of the points; partial credit may be given at the discretion of the grader.