COMP 330/543 Assignment #2

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 citation information for about 5000 papers from the Arxiv high-energy physics theory paper archive. The data set has around 14,400 citations between those papers. The data set is comprised of two database tables:

```
nodes (paperID, paperTitle);
edges (paperID, citedPaperID);
```

The first table gives a unique paper identifier, as well as the paper title. The second table indicates citations between the papers (note that citations 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 citation) and finds all connected components in the graph that have more than four and at most ten papers, printing out the associated lists of paper titles. My implementation found eight 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 larger than size four, and no larger than size ten. When you print out the components, print each paper ID as well as the title.

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, physics papers) 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 physics paper) 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 physics paper!) in a data set. The simplest is an iterative computation. Let $PR_i(\text{paper}_j)$ denote the estimated PageRank of the paper paper_j

at iteration i; assume that there are n papers in all. We start out with $PR_0(\text{paper}_j) = \frac{1}{n}$ forall j. Then, at iteration i, we simply set:

$$PR_i(\text{paper}_j) = \frac{1-d}{n} + d \left(\sum_{k \in \{\text{papers citing paper}_j\}} \frac{PR_{i-1}(\text{paper}_k)}{\text{num citations in paper}_k} \right)$$

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:

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

Your goal for this problem is to write one or more stored procedures that together compute the PageRank of each of the papers in the graph. You will run your code, and use it to print out the 10 papers with the greatest PageRank, as well as the PageRank for those papers. Again, when you print out a paper, print out both the paper ID and the paper title.

2 Getting Started

First, log onto SQLServer, go to your database, and create two tables:

```
CREATE TABLE nodes (
  paperID INTEGER,
  paperTitle VARCHAR (100));
CREATE TABLE edges (
  paperID INTEGER,
  citedPaperID INTEGER));
```

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

One important thing: don't keep anything important on the database server! There's a reasonable chance that people are going to leave many gigabytes of "trash" (junky, leftover tables) on the server as the result of this assignment, which can cause it to become unstable. If this happens, I might have to go in and start removing trash-clogged databases so that people can continue working. Thus, it's important that you save all of your code somewhere else.

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$\times$ 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, I'm 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.