COSC6340: Database Systems
Project: Linear Recursive Queries in SQL
Phase 1

Instructor: Carlos Ordonez

1 Introduction

You will develop a Java program that generates SQL code and connects to a DBMS via JDBC to evaluate recursive queries [1]. Phase 1 will require understanding the basic definitions and algorithms. Phase 2 will require incorporating and comparing query optimizations.

The DBMS will be HP Vertica running on Linux. Notice Vertica does not currently support recursive queries (which require WITH a common table extension CTE). If you want to try recursive queries on a different DBMS you can try them in MS SQL Server, which follows the WITH syntax described in the paper. The client Java program can run on Windows or Linux. The connection interface will be JDBC. All input and output of your program will be specified as input parameters, processing will be done with SQL queries.

2 Program requirements

Your program will generate SQL code based on one input table, whose basic schema corresponds to a graph $G = (V, E)$. Your program must produce an answer to some well-known graph problems.

1. You will implement the Semi-Naive algorithm described in Section III.C of [1].
2. You will solve two fundamental problems: (1) finding the shortest path between all pairs of points and (2) detecting if G has cycles.
3. You need to stop on either maximum recursion depth k or a fixpoint computation. You can convert the linear recursion to a loop since it is a tail recursion.
4. No demanding computation should happen in Java. That is, you cannot export the table or load the adjacency matrix in an array.
5. Optional: computing a measure of connectivity, which is crucial in social network analysis (e.g. Facebook, Twitter, email, phone calls). This is solved by computing the SVD of the Laplacian of G. You should display the 2nd largest eigenvalue, which measures connectivity. Optional: your program computes the Laplacian table and displays λ_2.

2.1 Programming details

1. Information to be processed is in SQL database tables.

2. You need to use JDBC to submit queries, but your program should be able to generate an SQL script file for all the queries used in your program. If any of these parameters are missing, your program should return an error message and stop.

3. You can use any library or package to solve SVD. Options include LAPACK, Matlab, R.

4. The TA will query table R. You can use this table name, but column names must come from table T.

2.2 Theory

Most algorithms in CS are recursive. The hardest problems tend to involve graphs. Recursive queries are the basic SQL computation mechanism combining recursion and graphs. The shortest path and detecting cycles require computing G^+ from G.

Optional: Spectral graph clustering requires computing the Laplacian of G and its Singular Value Decomposition (SVD) to get its eigenvalues. There are many references on graph theory, linear algebra on computing the Laplacian. Wikipedia has a brief introduction. For the connectivity part you should compute and store the Laplacian with SQL.

3 Program call and Output

Here is a specification of input/output.

- syntax for call:

  ```
  java rq E=<string>;i=<string>;j=<string>;v=<double>;k=<integer>
  ```

  where E is the input table representing a list of edges, i, j are the vertex ids, i is a source vertex, j is a destination vertex and v is an edge value (e.g. distance) and R is the result table with a similar schema to E (just adding p, the number of paths, and recursion depth d) and k is maximum recursion depth. Column p will be automatically created and therefore it will not be a program parameter, p should be initialized to 1. Notice table T should be created by your program, whereas E is the input table.

- input: Set of edges E, possibly with multiple rows per edge (i.e. duplicates, PK of E not given as parameter).

- Output:

  (1) your program must produce the table R containing the minimum distances among all potential paths as measured by v between each pair of vertices. You are not required to store the specific paths, but you can store them in auxiliary path *path tracking* tables with d columns (call them P_d, where d indicate recursion depth).

  (2) your program must display "G has cycles=Y/N". You are not required to show the specific cycle.

- Examples:

  "java rq E=flights;i=from;j=to;v=distance;k=5"

  "java rq E=texting;i=personA;j=personB;v=textlength;k=8"

  Notice E has unique rows between pair of cities for the flights table, whereas E may have repeated rows (multiple messages) for the tweeter data set.

- If the connection to the DBMS fails, you need to write an error message. Your program will be tested with scripts complying with the main call specified above. Your messages should go to the screen. Therefore, a GUI is discouraged and unnecessary.

4 Examples of input and SQL

You should re-create the example in [1] to understand definitions and the basic algorithm.

References

[1] C. Ordonez. Optimization of linear recursive queries in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(2):264–277, 2010.