CSCI 585 – Homework 4

1. Write a gremlin command that creates the given graph

First, we need to create the graph and use its traversal: -

```
gremlin> graph=TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> g=graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
```

Then we create the graph by using following SINGLE command: -

```
gremlin> g.addV().property(T.id,
"CS101").as("V1").addV().property(T.id,
"CS201").as("V2").addV().property(T.id,
"CS220").as("V3").addV().property(T.id,
"CS334").as("V4").addV().property(T.id,
"CS240").as("V5").addV().property(T.id,
"CS681").as("V6").addV().property(T.id,
"CS400").as("V7").addV().property(T.id,
"CS526").as("V8").addE("prereq").from("V2").to("V1").addE("prereq").fro
m("V3").to("V2").addE("prereq").from("V4").to("V2").addE("prereq").from
("V5").to("V3").addE("prereq").from("V6").to("V4").addE("prereq").from(
"V7").to("V4").addE("prereq").from("V8").to("V7").addE("coreq").from("V
5").to("V3").addE("coreq").from("V8").to("V7")
```

We confirm the result by echoing the traversal 'g' on the screen: -

```
gremlin> g
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]
```

First, Tinkergraph open() function is used to open (create) a new TinkerGraph instance. Then we use an iterable object (g) which is obtained by calling traversal() function. We use this 'g' to add nodes, edges and traverse the graph.

Now we use addV() and addE() funcitons to add vertices and edges to the graph. Note that each vertex has an id (Course number) and an alias (Vx). While every edge has a property (Prereq or Coreq). Dot (.) operator is used to chain all the function calls and hence the graph is constructed using only 1 statement as described above.

2. Write a query that will output JUST the doubly connected nodes:

Doubly connected nodes are the onces which are connected by the ususal "Prereq" edge as well as "Coreq" edge. Hence, we should look for node pairs A, B such that A and B are connected by "Coreq" edge. This is done in following command: -

```
gremlin> g.V().as("a").outE("coreq").inV().as("b").select("a","b")
==>[a:v[CS240],b:v[CS220]]
==>[a:v[CS526],b:v[CS400]]
```

Here we find all those nodes (A) such that their out edge has “Coreq” property and the other end of the edge (IN node) is called (B). Finally, we select nodes A and B which are printed.

3. **Write a query that will output all the ancestors (Prereqs) of a node**.

```
gremlin> g.V("CS526").repeat(__.out("prereq")).emit()
==>v[CS400]
==>v[CS334]
==>v[CS201]
==>v[CS101]
```

We have taken node “CS526” as example here, but we can use any node in the graph. Repeat acts like a while loop here. But note that there is no stopping condition specified, as there is no such condition. We want to get all the ancestors, which means we plan on running the loop till the end of the graph. Here “__” is used for anonymous reference. So here, __.out(“prereq”) matches any vertex with outward edge having property “prereq”. Emit function emits (prints) output of command after every repeat instance. Hence in all, the statement starts with CS526 node and repeats until it can find more nodes connected with “prereq” edge.

4. **Write a query that will output the max depth starting from a given node (provides a count (including itself) of all the connected nodes till the deepest leaf)**.

Here we take node CS101 as example. Looking at the graph, we can see that path 101->201->334->400->526 is the longest chain (with length 5). We need to apply following approach:

a. Starting from CS 101, we need to see all the nodes which can be reached by inward edges having property “prereq”. We consider inward edges because direction of edges is opposite to our traversal.

```
gremlin> g.V("CS101").repeat(__.in("prereq")).emit()
==>v[CS201]
==>v[CS220]
==>v[CS334]
==>v[CS240]
==>v[CS681]
==>v[CS400]
==>v[CS526]
```

Here we take que from previous command in Q3 and traverse the graph finding prereq nodes repeatedly, starting with CS101 node.

b. But we need to count length of the path to these nodes. For that we need paths: -

```
gremlin> g.V("CS101").repeat(__.in("prereq")).emit().path()
==>[v[CS101],v[CS201]]
==>[v[CS101],v[CS201],v[CS220]]
==>[v[CS101],v[CS201],v[CS334]]
==>[v[CS101],v[CS201],v[CS220],v[CS240]]
==>[v[CS101],v[CS201],v[CS334],v[CS681]]
==>[v[CS101],v[CS201],v[CS334],v[CS400]]
==>[v[CS101],v[CS201],v[CS334],v[CS400],v[CS526]]
```

c. Count their lengths. For this, we use count() function, but it returns total count of outputs in the entire statement. But we need count for each path listed here. So we use count(local).

```
gremlin>
g.V("CS101").repeat(__.in("prereq")).emit().path().count(local)
==>2
==>3
==>3
==>4
==>4
==>4
==>5
```

d. Now we just find maximum of all these values. This gives us maximum length of a path starting with CS101 and going to the leaf.

LAST ANSWER: -

```
gremlin>
g.V("CS101").repeat(__.in("prereq")).emit().path().count(local).m
ax()
==>5
```