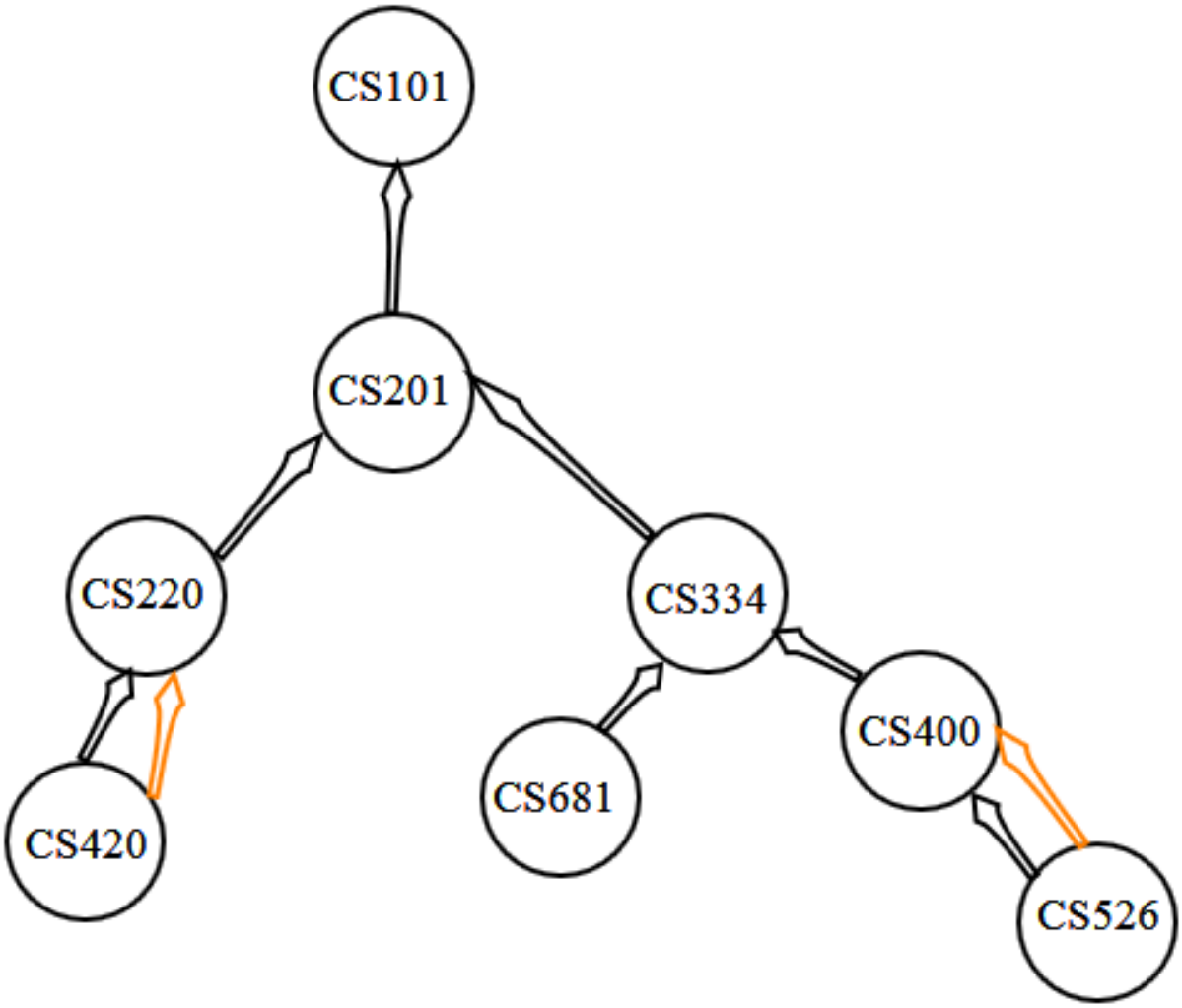
CSCI 585: HW4
USC ID:6386461387
Name: Prakhar Sethi
Language: Gremlin



Given Graph

Q.1: Write a Gremlin command that creates the above graph.

Query:

```
graph=TinkerGraph.open()
g=graph.traversal()

v1= g.addV("CS101").property(id,'CS101').property("prereq of","CS201”).property(‘name’,’CS101’).next()
v2= g.addV("CS201").property(id,'CS201').property("prereq of","CS220,CS334").property(‘name’,’CS201’).next()
g.addE("requires pre-req").from(v2).to(v1).property(id,1)
v3= g.addV("CS220").property(id,'CS220').property("prereq of","CS420").property("coreqs","CS420").property(‘name’,’CS220’).next()
g.addE("requires pre-req").from(v3).to(v2).property(id,2)
v4=g.addV("CS420").property(id,'CS420').property(‘name’,’CS420’).next()
g.addE("requires pre-req").from(v4).to(v3).property(id,3)
g.addE("is a co-req of").from(v4).to(v3).property(id,4)
v5=g.addV("CS334").property(id,'CS334').property("prereq of","CS681,CS400").property(‘name’,’CS334’).next()
g.addE("requires pre-req").from(v5).to(v2).property(id,5)
v6=g.addV("CS681").property(id,'CS681').property(‘name’,’CS681’).next()
g.addE("requires pre-req").from(v6).to(v5).property(id,6)
v7=g.addV("CS400").property(id,'CS400').property("prereq of","CS526").property("coreqs of", "CS526").property(‘name’,’CS400’).next()
g.addE("requires pre-req").from(v7).to(v5).property(id,7)
v8=g.addV("CS526").property(id,'CS526').property(‘name’,’CS526’).next()
g.addE("requires pre-req").from(v8).to(v7).property(id,8)
g.addE("is a co-req of").from(v8).to(v7).property(id,9)

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

Explanation:

TinkerGraph.open(): This will create an open graph with no edges and vertices.
Graph.Traversal(): It is used for graph traversal and here g is assigned as a traverser.
addV(): This will add a vertex to the graph with the name which is written in the parentheses.
Property(): This will add properties to the vertex like its id, name, etc.
addE(): This will add an edge in the graph in its parentheses the label is given
from(): It will define from where edge will start
to():It will define where edge will end.
Next(): It will take out the current node from iterator and go the next vertex command.

Logic:

First of all, I made an open graph and assign g as the traverser. I started adding vertices in top to bottom manner vertices consists of various properties like name, id and prereq or coreq. Then I added edges between the nodes. When all edges and nodes are added then I used ‘g’ to traverse the whole and display the number of node and edges in the graph.

The above query will create a graph:

```
guest-wireless-upc-1603-10-120-055-095:bin prakharsethi$ sh gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph=TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> g=graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> v1=g.addV("CS101").property(id,'CS101').property("prereq of","CS201").property('name','CS101').next()
==>v[CS101]
gremlin> v2=g.addV("CS201").property(id,'CS201').property("prereq of","CS220,CS334").property('name','CS201').next()
==>v[CS201]
gremlin> g.addE("requires pre-req").from(v2).to(v1).property(id,1)
==>e[1][CS201-requires pre-req->CS101]
gremlin> v3=g.addV("CS220").property(id,'CS220').property("prereq of","CS420").property("coreqs","CS420").property('name','CS220').next()
==>v[CS220]
gremlin> g.addE("requires pre-req").from(v3).to(v2).property(id,2)
==>e[2][CS220-requires pre-req->CS201]
gremlin> v4=g.addV("CS420").property(id,'CS420').property('name','CS420').next()
==>v[CS420]
gremlin> g.addE("requires pre-req").from(v4).to(v3).property(id,3)
==>e[3][CS420-requires pre-req->CS220]
gremlin> g.addE("is a co-req of").from(v4).to(v3).property(id,4)
==>e[4][CS420-is a co-req of->CS220]
gremlin> v5=g.addV("CS334").property(id,'CS334').property("prereq of","CS681,CS400").property('name','CS334').next()
==>v[CS334]
gremlin> g.addE("requires pre-req").from(v5).to(v2).property(id,5)
==>e[5][CS334-requires pre-req->CS201]
gremlin> v6=g.addV("CS681").property(id,'CS681').property('name','CS681').next()
==>v[CS681]
gremlin> g.addE("requires pre-req").from(v6).to(v5).property(id,6)
==>e[6][CS681-requires pre-req->CS334]
gremlin> v7=g.addV("CS400").property(id,'CS400').property("prereq of","CS526").property("coreqs of", "CS526").property('name','CS400').next()
==>v[CS400]
gremlin> g.addE("requires pre-req").from(v7).to(v5).property(id,7)
==>e[7][CS400-requires pre-req->CS334]
gremlin> v8=g.addV("CS526").property(id,'CS526').property('name','CS526').next()
==>v[CS526]
gremlin>  g.addE("requires pre-req").from(v8).to(v7).property(id,8)
==>e[8][CS526-requires pre-req->CS400]
gremlin> g.addE("is a co-req of").from(v8).to(v7).property(id,9)
==>e[9][CS526-is a co-req of->CS400]
gremlin> g
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]
```

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

Query:

g.V().as('a').out('requires pre-req').as('b').in('is a co-req of').as('c').select('a','b').where('a',eq('c'))

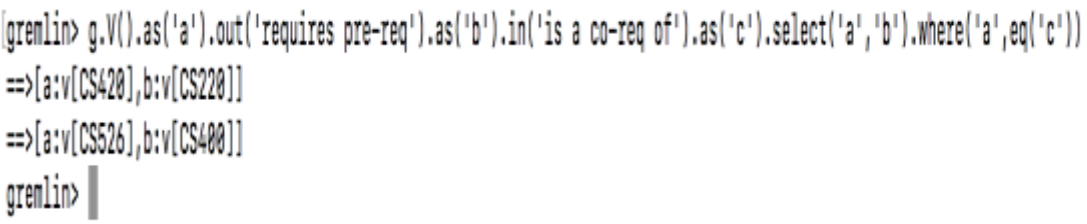
Explanation:

g represents the whole graph
V() function lists all the vertices of the traverser
As() function gives alias name to the command
Out() function will traverse to the adjacent vertices who have outgoing edges with label 'has-prereq' and the vertices are given a name 'b'
In() function will traverse to the adjacent vertices who have incoming edges with label 'has–coreq' and the vertices are given name 'c'
Select() function is similar to sql select it will select only vertices with name a and b
Where() function is a condition which is applied on whole command

Logic:

All vertices are named 'a', vertices that are connected with out edge 'requires pre-req' are named as 'b'. Now on all the 'b' vertices that have in edge of label 'is a co-req of' are named as 'c'. Only 'a' and 'b' are selected but with condition that 'a' equals 'c'. This will display doubly connected edges from a to b.

Output Screenshot:



Q.3: **Write a query that will output all the ancestors (for us, these would be prereqs) of a given vertex**

Query:

To output all the ancestors of a given vertex:

g.V().has(id,'CS526').repeat(out('requires pre-req')).emit()

OR

g.V('CS526').repeat(out('requires pre-req')).emit()

Explanation:

g is graph traverser,
V() is particular it can contain a vertex in parentheses or separately using has() command as shown above,
Repeat() is used for looping over traversal with a break condition like emit in our case
Out() will simply output the vertex connected to the edge with label
Emit() it is operated on the traverser which is g in my case and repeat the process again now using new vertex.

Logic:

Here, I am selecting 'CS526' vertex, then I will go to vertex connected with edge having label 'requires pre-req' and emits its name and this will repeat because of repeat till no out edges are found.

To output all ancestors of a given vertex including that vertex:

g.V('CS526').emit().repeat(out('requires pre-req'))

Using emit before repeat will output 'CS526' then it goes to out which will give another vertex which in turn will be emitted and whole process will continue.

Ouput Screenshot (Second is the answer):

```
gremlin> g.V().has(id,'CS526').emit().repeat(out('requires pre-req'))
==>v[CS526]
==>v[CS400]
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin> g.V().has(id,'CS526').repeat(out('requires pre-req')).emit()
==>v[CS400]
==>v[CS334]
==>v[CS201]
==>v[CS101]
```

Q.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). This would give us a total count of the longest sequence of courses that can be taken, after having completed a prereq course.**

Query:

g.V().has(id,'CS101').emit().repeat(__.in('requires pre-req')).tail(1).path().unfold().count()

Explanation: g represents the whole graph
V() function lists all the vertices of the traverser
Has(id,'CS101') will select vertex with id CS101 that means the first vertex of our graph.
Emit() function will perform emission of the vertices first it will give CS101 then it will enter in repeat() function.
Repeat() function is straightforward and simply traverse through all vertices whose incoming edges have label of 'requires pre-req'.
Tail(1) function in gremlin similar to limit in sql. As limit function gives the first objects tail() function gives last objects. Tail(1) will only give the last object.
Path() function will give the path from first vertex to last vertex in a particular path.
Unfold() will display the vertices of the path line by line.
Count() function will count the total number of lines.
__.in(): It is used to give path from top to bottom

Logic:

I have selected 'CS101' vertex and emits its name because it is at level 1 then I have to move downwards so I will go the vertices with edges into 'CS101' having label 'requires pre-req' and put this on repeat. I will get many edges from this query. Then I choose the last path which will outward me to the last vertex and find the path of all the vertices. This all will come in a line to display in multiple lines I will use unfold function and count the lines which will display the depth of the graph.

```
gremlin> g.V().has(id,'CS101').emit().repeat(__.in('requires pre-req')).tail(1).path().unfold().count()
==>5
```

BONUS QUESTION

Q.1: Query:

g.addV('CS101').property(id,'CS101').property('prereq of','CS201').property('name','CS101').as('101').addV('CS201').property(id,'CS201').property('prereq of','CS220,CS334').property('name','CS201').as('201').addV('CS220').property(id,'CS220').property('prereq of','CS420').property('coreqs','CS420').property('name','CS220').as('220').addV('CS420').property(id,'CS420').property('name','CS420').as('420').addV('CS334').property(id,'CS334').property('prereq of','CS681,CS400').property('name','CS334').as('334').addV('CS681').property(id,'CS681').property('name','CS681').as('681').addV('CS400').property(id,'CS400').property('prereq of','CS526').property('coreqs of','CS526').property('name','CS400').as('400').addV('CS526').property(id,'CS526').property('name','CS526').as('526').addE('requires prereq').from('201').to('101').property(id,1).addE('requires prereq').from('220').to('201').property(id,2).addE('requires prereq').from('420').to('220').property(id,3).addE('is a co-req of').from('420').to('220').property(id,4).addE('requires prereq').from('334').to('201').property(id,5).addE('requires prereq').from('681').to('334').property(id,6).addE('requires prereq').from('400').to('334').property(id,7).addE('requires prereq').from('526').to('400').property(id,8).addE('is a co-req of').from('526').to('400').property(id,9)

In the above query:
addV() will create a vertex
addE() will create an edge
property() will hold property of either edge or vertex
from() will hold the vertex that will be the tail in the direction of edge
to() will hold vertex that will be the head in the direction of edge
as() will give a label to the command

Q.2: Query:

g.V().as('a').out('has-prereq').as('b').in('has-coreq').as('c').select('a','b').where('a',eq('c'))

Q.3 Query:

g.V('CS526').repeat(out('requires pre-req')).emit()

Q.4 Query:

g.V().has(id,'CS101').emit().repeat(__.in('requires pre-req')).tail(1).path().unfold().count()