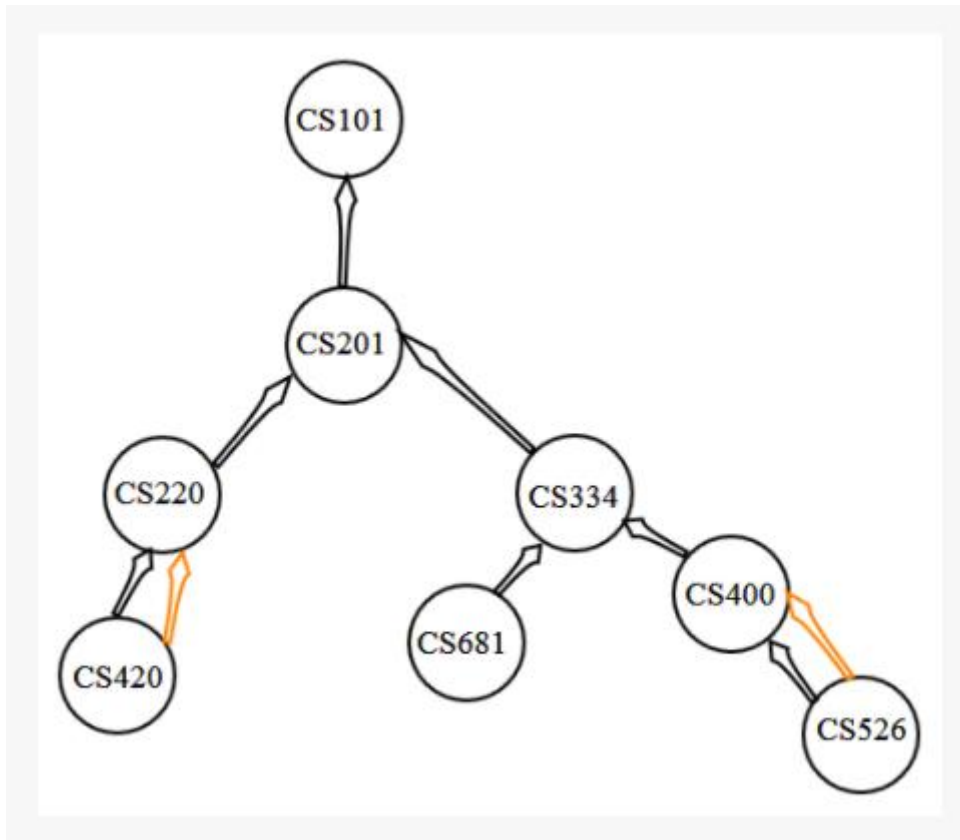


## HOMEWORK 4

### GRAPH:



**QUERY 1:** Write a Gremlin command that creates the above graph [hint - you will also need a 'traversal' for it]. The command could be a multi-statement one, or a single line one.

```
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(T.id,'CS101').next()
==>v[CS101]
gremlin> v2=g.addV("CS201").property(T.id,'CS201').next()
==>v[CS201]
gremlin> v3=g.addV("CS220").property(T.id,'CS220').next()
==>v[CS220]
gremlin> v4=g.addV("CS420").property(T.id,'CS420').next()
==>v[CS420]
gremlin> v5=g.addV("CS334").property(T.id,'CS334').next()
==>v[CS334]
gremlin> v6=g.addV("CS400").property(T.id,'CS400').next()
==>v[CS400]
gremlin> v7=g.addV("CS681").property(T.id,'CS681').next()
==>v[CS681]
gremlin> v8=g.addV("CS526").property(T.id,'CS526').next()
==>v[CS526]
gremlin> g.addE("prereq").from(v2).to(v1)
```

```

==>e[0][CS201-prereq->CS101]
gremlin> g.addE("prereq").from(v3).to(v2)
==>e[1][CS220-prereq->CS201]
gremlin> g.addE("prereq").from(v4).to(v3)
==>e[2][CS420-prereq->CS220]
gremlin> g.addE("prereq").from(v5).to(v2)
==>e[3][CS334-prereq->CS201]
gremlin> g.addE("prereq").from(v6).to(v5)
==>e[4][CS400-prereq->CS334]
gremlin> g.addE("prereq").from(v7).to(v5)
==>e[5][CS681-prereq->CS334]
gremlin> g.addE("prereq").from(v8).to(v6)
==>e[6][CS526-prereq->CS400]
gremlin> g.addE("coreq").from(v8).to(v6)
==>e[7][CS526-coreq->CS400]
gremlin> g.addE("coreq").from(v4).to(v3)
==>e[8][CS420-coreq->CS220]
gremlin> g
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]

```

Screenshot: (query + output)

```

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(T.id,'CS101').next()
==>v[CS101]
gremlin> v2 = g.addV("CS201").property(T.id,'CS201').next()
==>v[CS201]
gremlin> v3 = g.addV("CS220").property(T.id,'CS220').next()
==>v[CS220]
gremlin> v4 = g.addV("CS420").property(T.id,'CS420').next()
==>v[CS420]
gremlin> v5 = g.addV("CS334").property(T.id,'CS334').next()
==>v[CS334]
gremlin> v6 = g.addV("CS400").property(T.id,'CS400').next()
==>v[CS400]
gremlin> v7 = g.addV("CS681").property(T.id,'CS681').next()
==>v[CS681]
gremlin> v8 = g.addV("CS526").property(T.id,'CS526').next()
==>v[CS526]
gremlin> g.addE("prereq").from(v2).to(v1)
==>e[0][CS201-prereq->CS101]
gremlin> g.addE("prereq").from(v3).to(v2)
==>e[1][CS220-prereq->CS201]
gremlin> g.addE("prereq").from(v4).to(v3)
==>e[2][CS420-prereq->CS220]
gremlin> g.addE("prereq").from(v5).to(v2)
==>e[3][CS334-prereq->CS201]
gremlin> g.addE("prereq").from(v6).to(v5)
==>e[4][CS400-prereq->CS334]
gremlin> g.addE("prereq").from(v7).to(v5)
==>e[5][CS681-prereq->CS334]
gremlin> g.addE("prereq").from(v8).to(v6)
==>e[6][CS526-prereq->CS400]
gremlin> g.addE("coreq").from(v8).to(v6)
==>e[7][CS526-coreq->CS400]
gremlin> g.addE("coreq").from(v4).to(v3)
==>e[8][CS420-coreq->CS220]
gremlin> g
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]
gremlin>

```

**Explanation:**

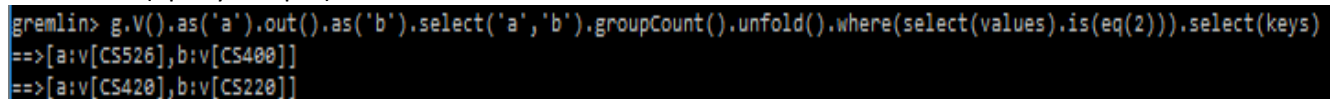
First step is to create an empty graph by using `Tinkergraph.open()` . Then add vertices and edges with the help of functions `g.addV()` and `g.addE()` along with the labels specified like "CS101" and "prereq" We can add properties such as "T.id" or "name" for the vertices and edges as well. `next()` will return the next result in the running sequence of the creation of vertices. `from(x)` and `to(y)` commands will help in the creation of an edge from x vertex to y vertex respectively.

**QUERY 2**

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

```
gremlin>
g.V().as('a').out().as('b').select('a','b').groupCount().unfold().where(select(values).is(eq(2))).select(keys)
==>[a:v[CS526],b:v[CS400]]
==>[a:v[CS420],b:v[CS220]]
```

Screenshot ( query+output)



```
gremlin> g.V().as('a').out().as('b').select('a','b').groupCount().unfold().where(select(values).is(eq(2))).select(keys)
==>[a:v[CS526],b:v[CS400]]
==>[a:v[CS420],b:v[CS220]]
```

**Explanation:**

This query uses `select()` to select labeled steps within a path (as defined by `as()` in a traversal). `groupCount()` will count the number of edges between the vertices labelled as "a" and "b". `unfold()` is used to iterate through the Map of <Vertex Pairs,Count>. `where()` clause checks the condition for the edges having a count equal to 2. Finally, `select(keys)` will only display vertex pairs and emits the "count" as it is not required in the output.

**QUERY 3**

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

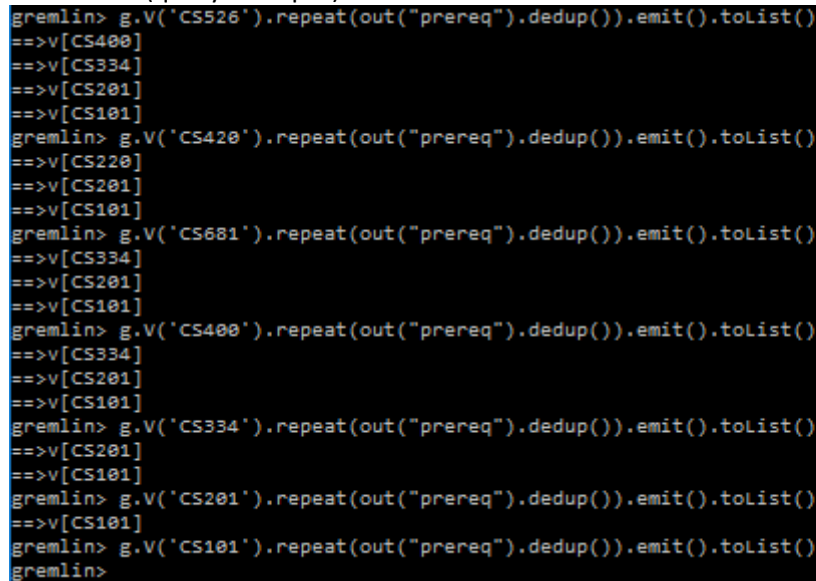
```
gremlin> g.V('CS526').repeat(out("prereq").dedup()).emit().toList()
==>v[CS400]
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS420').repeat(out("prereq").dedup()).emit().toList()
==>v[CS220]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS681').repeat(out("prereq").dedup()).emit().toList()
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS400').repeat(out("prereq").dedup()).emit().toList()
==>v[CS334]
==>v[CS201]
```

```

==>v[CS101]
gremlin> g.V('CS334').repeat(out("prereq").dedup()).emit().toList()
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS201').repeat(out("prereq").dedup()).emit().toList()
==>v[CS101]
gremlin> g.V('CS101').repeat(out("prereq").dedup()).emit().toList()
gremlin>

```

Screenshot (query + output)



```

gremlin> g.V('CS526').repeat(out("prereq").dedup()).emit().toList()
==>v[CS400]
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS420').repeat(out("prereq").dedup()).emit().toList()
==>v[CS220]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS681').repeat(out("prereq").dedup()).emit().toList()
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS400').repeat(out("prereq").dedup()).emit().toList()
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS334').repeat(out("prereq").dedup()).emit().toList()
==>v[CS201]
==>v[CS101]
gremlin> g.V('CS201').repeat(out("prereq").dedup()).emit().toList()
==>v[CS101]
gremlin> g.V('CS101').repeat(out("prereq").dedup()).emit().toList()
gremlin>

```

### Explanation:

This query uses `repeat()` function for looping in order to check the outgoing edges labelled “prereq”. `Dedup()` will omit the duplicate results and will simply output the unique ones. `Emit()` is placed after `repeat()`, so it is evaluated on the traversers leaving the repeat-traversal. `toList()` will return a list of all the objects in the pipe.

### QUERY 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.

```

gremlin> g.V('CS101').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>5
gremlin> g.V('CS201').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>4
gremlin> g.V('CS420').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>1
gremlin> g.V('CS334').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>3
gremlin> g.V('CS681').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()

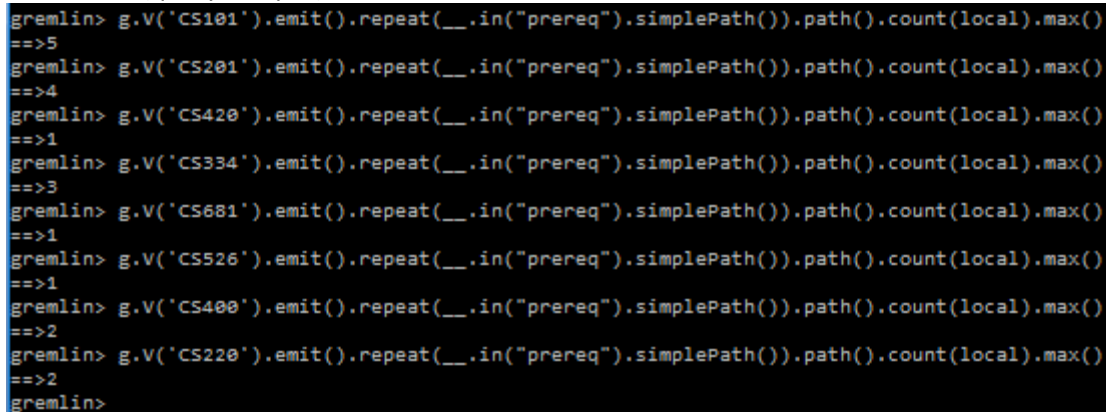
```

```

==>1
gremlin> g.V('CS526').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>1
gremlin> g.V('CS400').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>2
gremlin> g.V('CS220').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>2
gremlin>

```

Screenshot (query+output)



```

gremlin> g.V('CS101').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>5
gremlin> g.V('CS201').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>4
gremlin> g.V('CS420').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>1
gremlin> g.V('CS334').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>3
gremlin> g.V('CS681').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>1
gremlin> g.V('CS526').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>1
gremlin> g.V('CS400').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>2
gremlin> g.V('CS220').emit().repeat(__.in("prereq").simplePath()).path().count(local).max()
==>2
gremlin>

```

### Explanation:

The traversal starts from a vertex with the specified label, then traverses in on the "prereq" edges , finds the length of path and displays the longest one.

This query uses emit() before repeat(), so it is evaluated on the traversers prior to entering the repeat-traversal.

(note: "in" is a reserved keyword in Groovy, hence verbosity is added i.e, \_\_.in().)

simplePath() is used so that the traverser do not repeat its path through the graph.

Its uses count(local) and max() to count the maximum number of instances out of all the paths incoming to the specified vertex.

### BONUS:

Express Q1 as TWO commands [or ONE] (0.25 points); Q2, Q3, Q4 as a SINGLE command each.

```

gremlin> g= TinkerGraph.open().traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> g.addV("CS101").property(T.id, 'CS101').as('CS101').
.....1>      addV("CS201").property(T.id, 'CS201').as('CS201').
.....2>      addV("CS220").property(T.id, 'CS220').as('CS220').
.....3>      addV("CS420").property(T.id, 'CS420').as('CS420').
.....4>      addV("CS334").property(T.id, 'CS334').as('CS334').
.....5>      addV("CS681").property(T.id, 'CS681').as('CS681').
.....6>      addV("CS526").property(T.id, 'CS526').as('CS526').
.....7>      addV("CS400").property(T.id, 'CS400').as('CS400').
.....8>      addE("prereq").from('CS201').to('CS101').

```

```

.....9>      addE("prereq").from('CS220').to('CS201').
.....10> addE("prereq").from('CS420').to('CS220').
.....11> addE("prereq").from('CS334').to('CS201').
.....12> addE("prereq").from('CS400').to('CS334').
.....13> addE("prereq").from('CS681').to('CS334').
.....14> addE("prereq").from('CS526').to('CS400').
.....15> addE("coreq").from('CS526').to('CS400').
.....16> addE("coreq").from('CS420').to('CS220').iterate()
gremlin>
gremlin> g
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]

```

Screenshots:

1. Creating an empty graph:

```

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

```

2. Adding vertices and edges:

```

gremlin> g= TinkerGraph.open().traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> g.addV("CS101").property(T.id, 'CS101').as('CS101').
.....1>      addV("CS201").property(T.id, 'CS201').as('CS201').
.....2>      addV("CS220").property(T.id, 'CS220').as('CS220').
.....3>      addV("CS420").property(T.id, 'CS420').as('CS420').
.....4>      addV("CS334").property(T.id, 'CS334').as('CS334').
.....5>      addV("CS681").property(T.id, 'CS681').as('CS681').
.....6>      addV("CS526").property(T.id, 'CS526').as('CS526').
.....7>      addV("CS400").property(T.id, 'CS400').as('CS400').
.....8>      addE("prereq").from('CS201').to('CS101').
.....9>      addE("prereq").from('CS220').to('CS201').
.....10>     addE("prereq").from('CS420').to('CS220').
.....11>     addE("prereq").from('CS334').to('CS201').
.....12>     addE("prereq").from('CS400').to('CS334').
.....13>     addE("prereq").from('CS681').to('CS334').
.....14>     addE("prereq").from('CS526').to('CS400').
.....15>     addE("coreq").from('CS526').to('CS400').
.....16>     addE("coreq").from('CS420').to('CS220').iterate()
gremlin>
gremlin> g
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]

```

### Explanation:

Function chaining is used to write the query in a single line. Instead of adding vertices and edges line by line. They are added into the graph with the help of function chaining. iterate() function is used explicitly to iterate the traversal.

Q2, Q3, Q4 - Same as the above(already in a single command).