Gremelin Queries

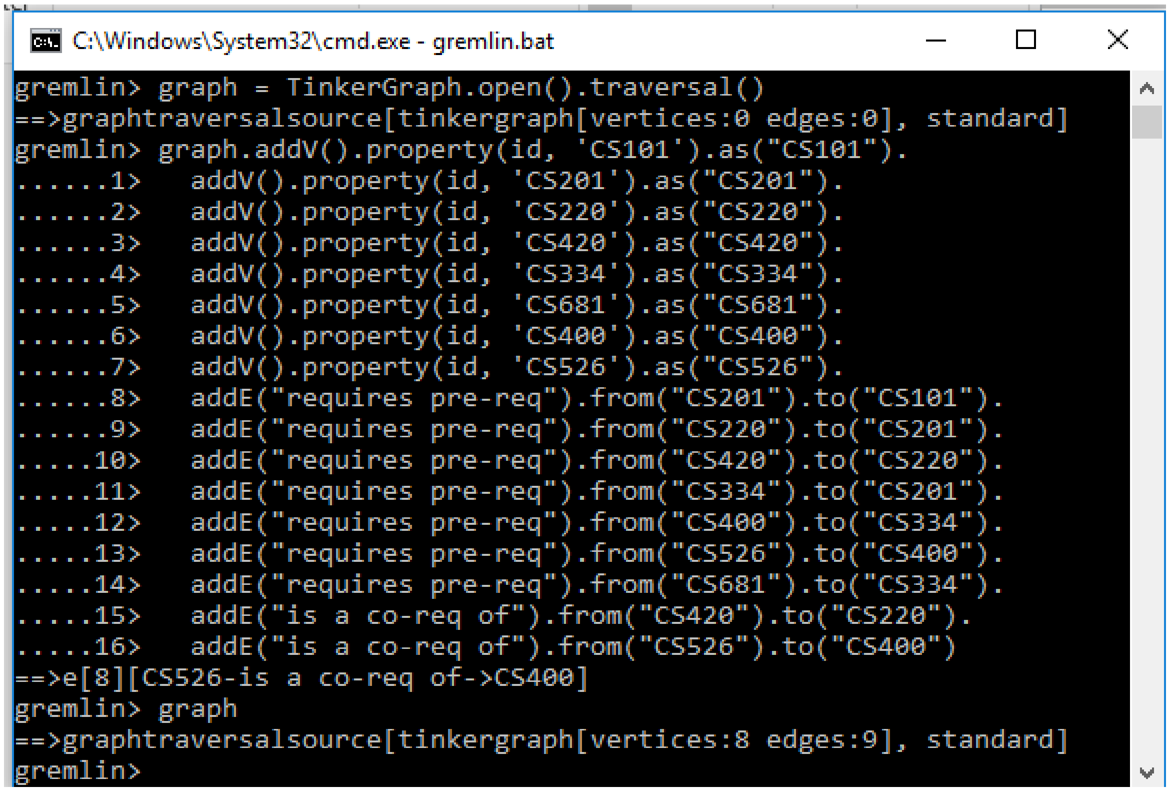
Q1) Create the graph command

```
graph = TinkerGraph.open().traversal()
graph.addV().property(id, 'CS101').as("CS101").
  addV().property(id, 'CS201').as("CS201").
  addV().property(id, 'CS220').as("CS220").
  addV().property(id, 'CS420').as("CS420").
  addV().property(id, 'CS334').as("CS334").
  addV().property(id, 'CS681').as("CS681").
  addV().property(id, 'CS400').as("CS400").
  addV().property(id, 'CS526').as("CS526").
  addE("requires pre-req").from("CS201").to("CS101").
  addE("requires pre-req").from("CS220").to("CS201").
  addE("requires pre-req").from("CS420").to("CS220").
  addE("requires pre-req").from("CS334").to("CS201").
  addE("requires pre-req").from("CS400").to("CS334").
  addE("requires pre-req").from("CS526").to("CS400").
  addE("requires pre-req").from("CS681").to("CS334").
  addE("is a co-req of").from("CS420").to("CS220").
  addE("is a co-req of").from("CS526").to("CS400")
```

EXPLANATION:

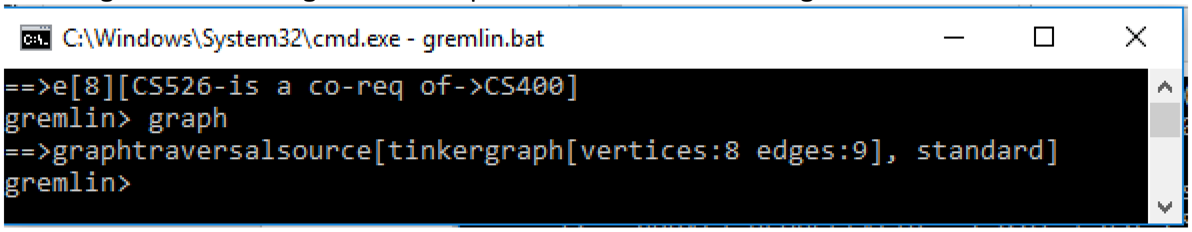
- The first line of the query returns the traversal object after creating the graph.
- Then added Vertices and Edges to the graph with functions addV() and addE().

SCREENSHOTS:
Successful running of the above query:



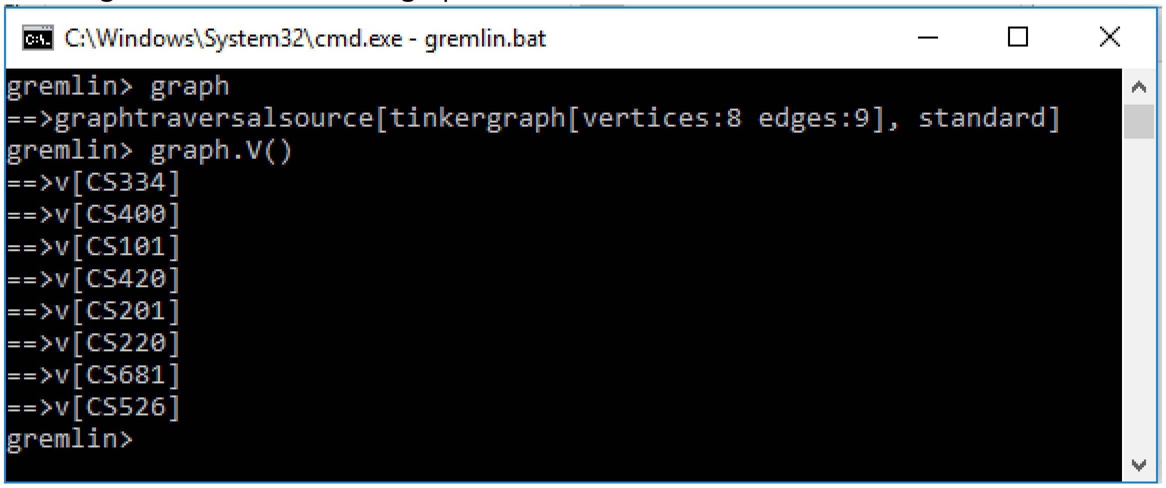
```
gremlin> graph = TinkerGraph.open().traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> graph.addV().property(id, 'CS101').as("CS101").
......1>   addV().property(id, 'CS201').as("CS201").
......2>   addV().property(id, 'CS220').as("CS220").
......3>   addV().property(id, 'CS420').as("CS420").
......4>   addV().property(id, 'CS334').as("CS334").
......5>   addV().property(id, 'CS681').as("CS681").
......6>   addV().property(id, 'CS400').as("CS400").
......7>   addV().property(id, 'CS526').as("CS526").
......8>   addE("requires pre-req").from("CS201").to("CS101").
......9>   addE("requires pre-req").from("CS220").to("CS201").
.....10>   addE("requires pre-req").from("CS420").to("CS220").
.....11>   addE("requires pre-req").from("CS334").to("CS201").
.....12>   addE("requires pre-req").from("CS400").to("CS334").
.....13>   addE("requires pre-req").from("CS526").to("CS400").
.....14>   addE("requires pre-req").from("CS681").to("CS334").
.....15>   addE("is a co-req of").from("CS420").to("CS220").
.....16>   addE("is a co-req of").from("CS526").to("CS400")
==>e[8][CS526-is a co-req of->CS400]
gremlin> graph
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]
gremlin>
```

Showing the traversal: gives the output of 8 vertices and 9 edges



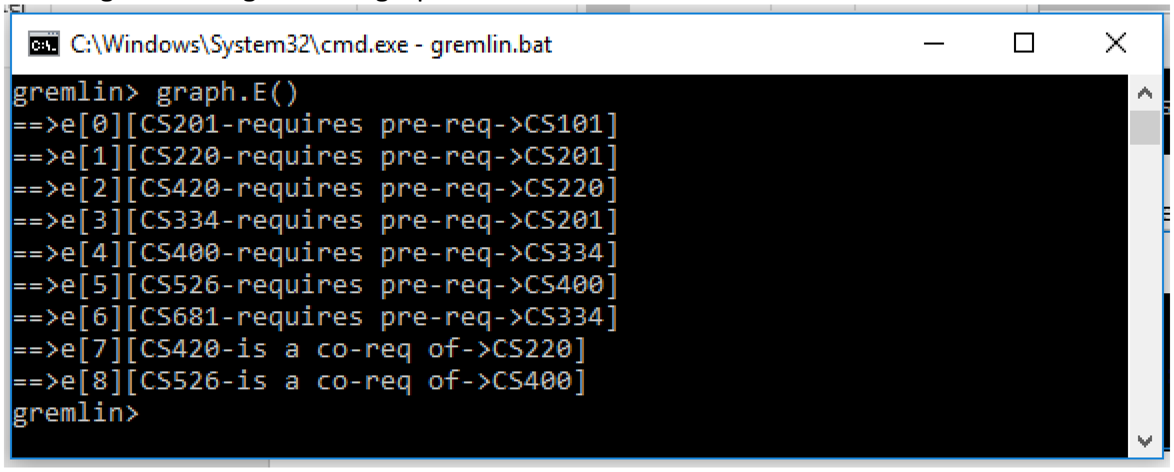
```
==>e[8][CS526-is a co-req of->CS400]
gremlin> graph
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]
gremlin>
```

Showing all the vertices in the graph:



```
gremlin> graph
==>graphtraversalsource[tinkergraph[vertices:8 edges:9], standard]
gremlin> graph.V()
==>v[CS334]
==>v[CS400]
==>v[CS101]
==>v[CS420]
==>v[CS201]
==>v[CS220]
==>v[CS681]
==>v[CS526]
gremlin>
```

Showing all the Edges in the graph:



C:\Windows\System32\cmd.exe - gremlin.bat

```
gremlin> graph.E()
==>e[0][CS201-requires pre-req->CS101]
==>e[1][CS220-requires pre-req->CS201]
==>e[2][CS420-requires pre-req->CS220]
==>e[3][CS334-requires pre-req->CS201]
==>e[4][CS400-requires pre-req->CS334]
==>e[5][CS526-requires pre-req->CS400]
==>e[6][CS681-requires pre-req->CS334]
==>e[7][CS420-is a co-req of->CS220]
==>e[8][CS526-is a co-req of->CS400]
gremlin>
```

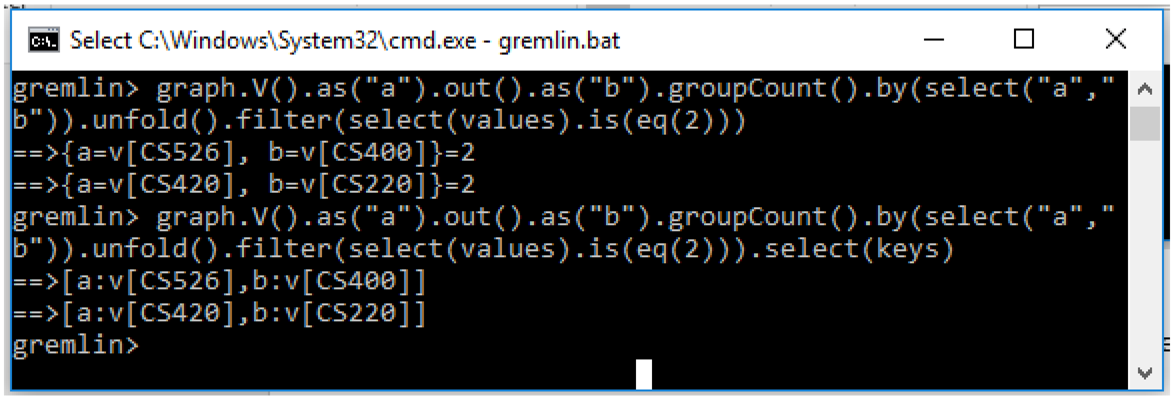
Q2) Query that will output JUST the doubly-connected nodes

graph.V().as("a").out().as("b").groupCount().by(select("a","b")).unfold().filter(select(values).is(eq(2))).select(keys)

EXPLANATION:

- Used two aliases a, b to get all the vertices and vertices having out edges.
- groupCount() is used to count the number of times elements are repeated.
- by() is used to specify a condition. Here we are getting a, b as key value pair with counts.
- unfold() is used to iterate over a list and print individually
- eq() is used to specify the count condition.
- filter() is used to filter out the vertices which has 2 edges between them.
- Select(keys) displays only the keys without giving the count.

SCREENSHOT:



Select C:\Windows\System32\cmd.exe - gremlin.bat

```
gremlin> graph.V().as("a").out().as("b").groupCount().by(select("a","
b")).unfold().filter(select(values).is(eq(2)))
==>{a=v[CS526], b=v[CS400]}=2
==>{a=v[CS420], b=v[CS220]}=2
gremlin> graph.V().as("a").out().as("b").groupCount().by(select("a","
b")).unfold().filter(select(values).is(eq(2))).select(keys)
==>[a:v[CS526],b:v[CS400]]
==>[a:v[CS420],b:v[CS220]]
gremlin>
```

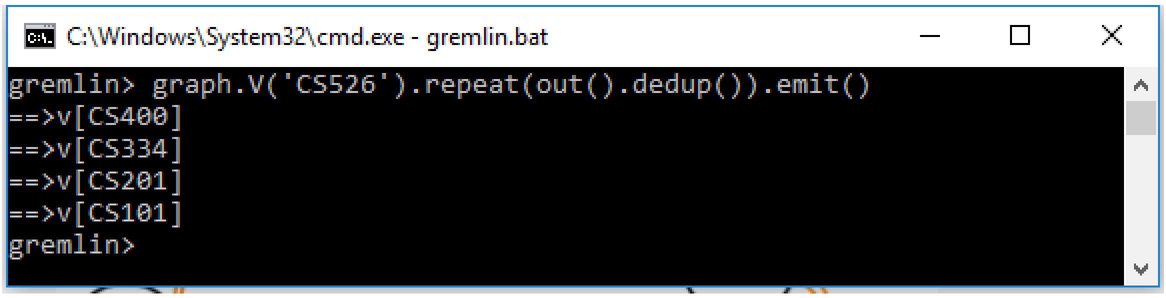
Q3) Query that will output all the ancestors

graph.V('CS526').repeat(out().dedup()).emit()

EXPLANATION:

- Out() gives all the vertices which have outward edge from the given node.
- Dedup() removes the nodes which are already visited.
- Runs the functions repeatedly until there is any node left.
- Emit() displays the nodes.
- The combination of the above functions gives a query which will output all the ancestors.

SCREENSHOT:



C:\Windows\System32\cmd.exe - gremlin.bat

```
gremlin> graph.V('CS526').repeat(out().dedup()).emit()
==>v[CS400]
==>v[CS334]
==>v[CS201]
==>v[CS101]
gremlin>
```

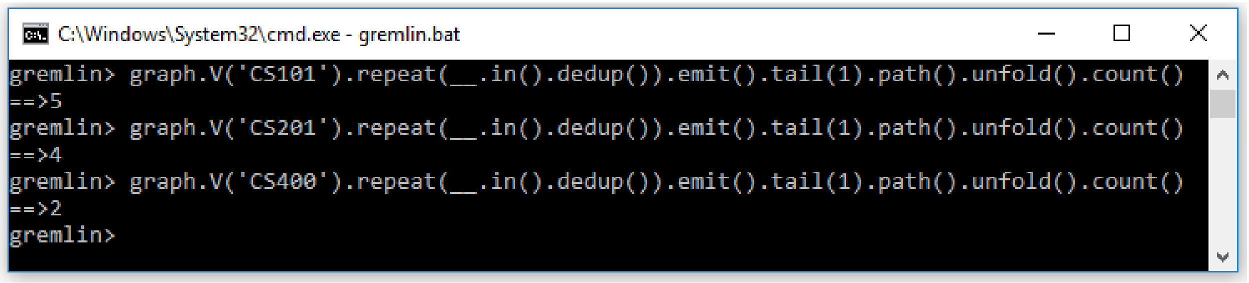
Q4) Query that will output the max depth starting from a given node

graph.V('CS101').emit().repeat(__.in().dedup()).tail(1).path().unfold().count()

EXPLANATION:

- The repeat is done on inward edges and their paths are taken.
- Then selecting the tail() to unfold and count the number of nodes present in the path.

SCREENSHOT:



C:\Windows\System32\cmd.exe - gremlin.bat

```
gremlin> graph.V('CS101').repeat(__.in().dedup()).emit().tail(1).path().unfold().count()
==>5
gremlin> graph.V('CS201').repeat(__.in().dedup()).emit().tail(1).path().unfold().count()
==>4
gremlin> graph.V('CS400').repeat(__.in().dedup()).emit().tail(1).path().unfold().count()
==>2
gremlin>
```

NOTE: All the queries are in a single statement.