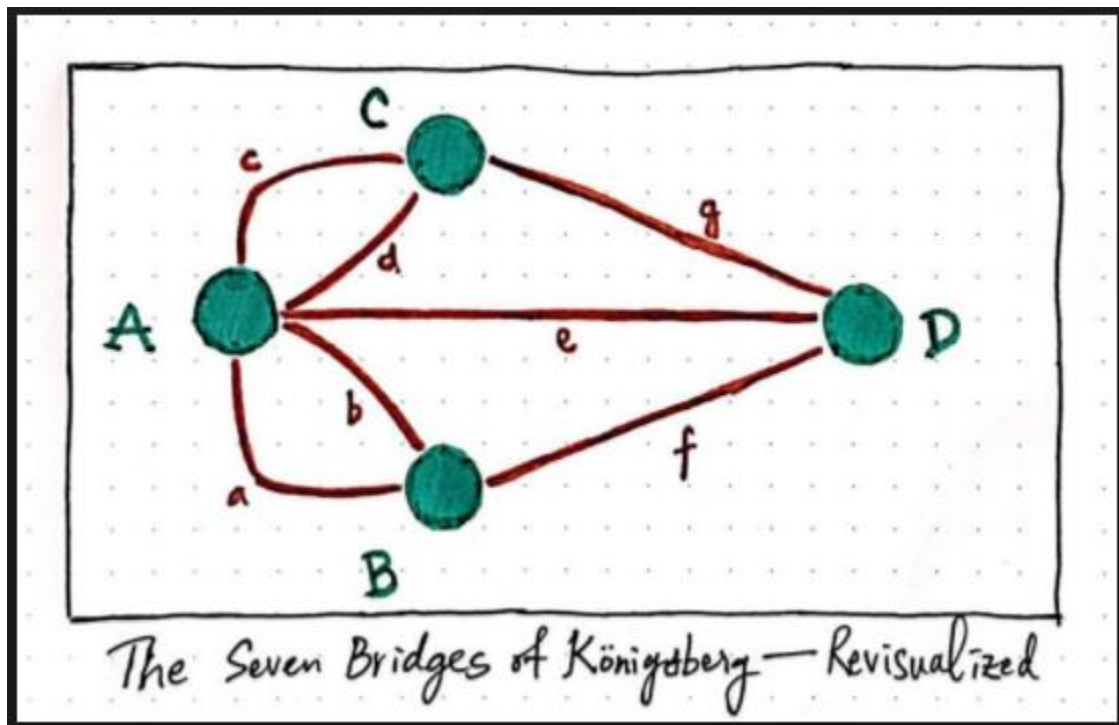


## HW4 Bonus Question

### Konigsberg Bridge Problem

A graph with 4 regions and 7 edges which shows the Königsberg bridge model.



EXPLANATION : Created the above graph with help of open(), addV(), addE() function

```
konigsberg = TinkerGraph.open().traversal()
konigsberg.addV().property(id, 'A').as("A").
addV().property(id, 'B').as("B").
addV().property(id, 'C').as("C").
addV().property(id, 'D').as("D").
addE("bridge").from("A").to("B").
addE("bridge").from("A").to("B").
addE("bridge").from("A").to("C").
addE("bridge").from("B").to("D").
```

```
addE("bridge").from("B").to("D").
```

```
addE("bridge").from("B").to("C").
```

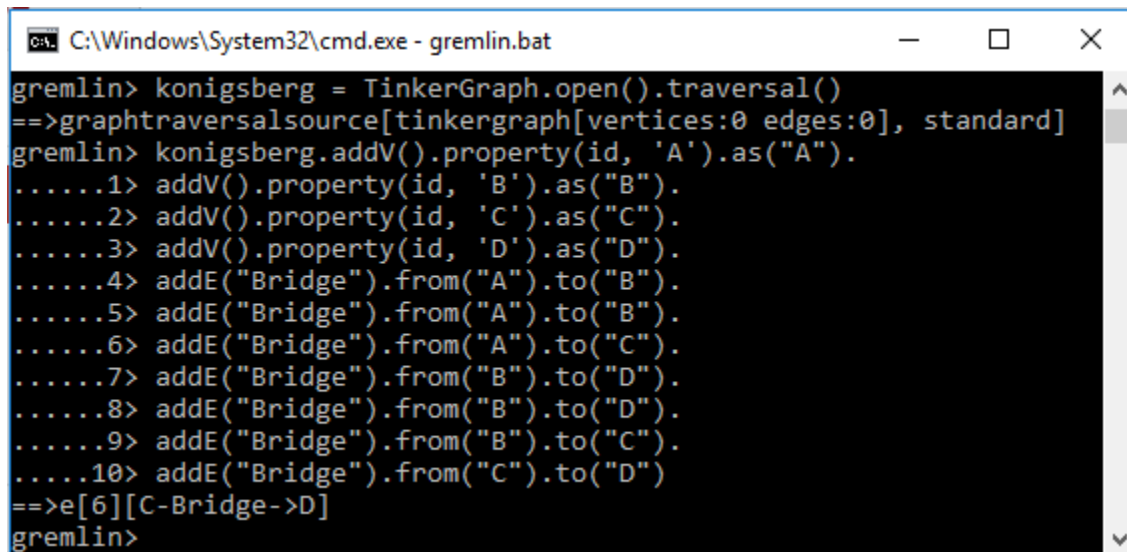
```
addE("bridge").from("C").to("D")
```

Konigsberg – to print traversal

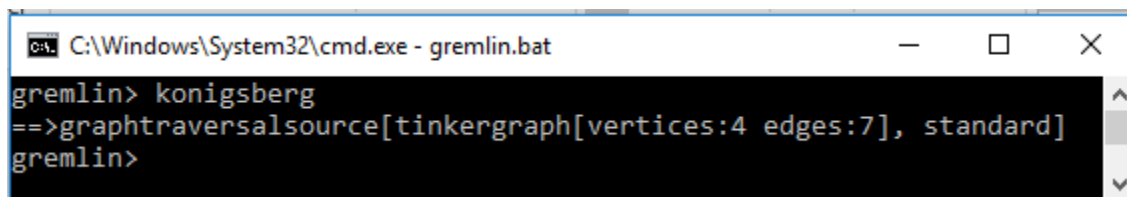
konigsberg.V() – to print vertices

konigsberg.E() – to print Edges

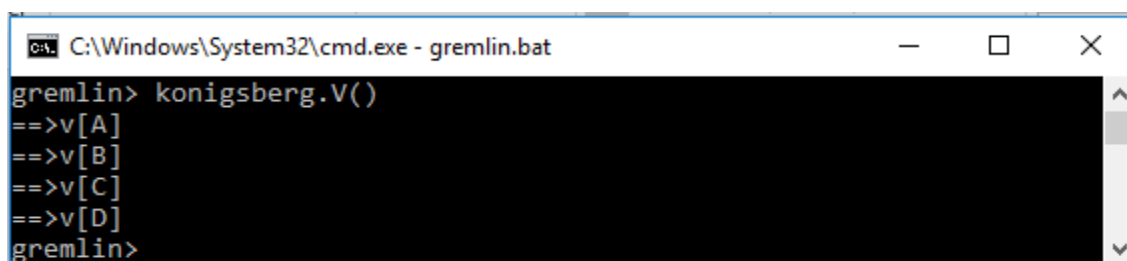
SCREENSHOTS-



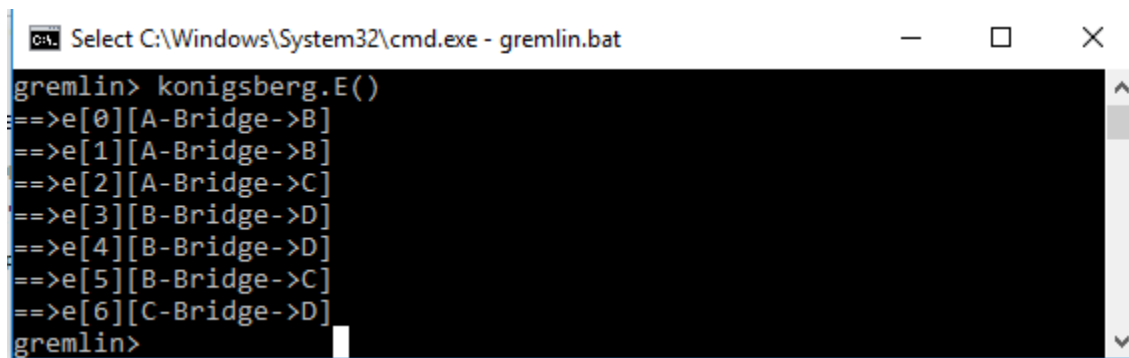
```
C:\Windows\System32\cmd.exe - gremlin.bat
gremlin> konigsberg = TinkerGraph.open().traversal()
==>graphtraversalsource[tinkergraph[vertices:0 edges:0], standard]
gremlin> konigsberg.addV().property(id, 'A').as("A").
.....1> addV().property(id, 'B').as("B").
.....2> addV().property(id, 'C').as("C").
.....3> addV().property(id, 'D').as("D").
.....4> addE("Bridge").from("A").to("B").
.....5> addE("Bridge").from("A").to("B").
.....6> addE("Bridge").from("A").to("C").
.....7> addE("Bridge").from("B").to("D").
.....8> addE("Bridge").from("B").to("D").
.....9> addE("Bridge").from("B").to("C").
.....10> addE("Bridge").from("C").to("D")
==>e[6][C-Bridge->D]
gremlin>
```



```
C:\Windows\System32\cmd.exe - gremlin.bat
gremlin> konigsberg
==>graphtraversalsource[tinkergraph[vertices:4 edges:7], standard]
gremlin>
```



```
C:\Windows\System32\cmd.exe - gremlin.bat
gremlin> konigsberg.V()
==>v[A]
==>v[B]
==>v[C]
==>v[D]
gremlin>
```



```
gremlin> konigsberg.E()
==>e[0][A-Bridge->B]
==>e[1][A-Bridge->B]
==>e[2][A-Bridge->C]
==>e[3][B-Bridge->D]
==>e[4][B-Bridge->D]
==>e[5][B-Bridge->C]
==>e[6][C-Bridge->D]
gremlin>
```

## Eulerian circuit

To show that there is NO path through the bridges and cities where, starting at any region of the city you traverse each bridge just once, and return to the starting point for Königsberg bridge.

```
konigsberg.V().sideEffect(outE("bridge").aggregate("bridges")).barrier().
```

```
  repeat(bothE().
```

```
    or(__.not(select('e')),
```

```
      __.not(filter(__.as('x').select(all, 'e').unfold().
```

```
        where(eq('x')))).as('e').
```

```
    otherV()).
```

```
  until(select(all, 'e').count(local).as("c").
```

```
    select("bridges").count(local).where(eq("c"))).hasNext()
```

EXPLANATION:

- Retrieving all the edges storing them and repeating on them.
- bothE() is used because we need to consider both inward and outward edges.
- As the edges need to be traversed only once we are taking “e” as the traversed edges.
- Repeat is done until e traversed is equal to number of edges in the graph. If this is possible then returning true else false.

SCREENSHOTS:

For 4 regions and 7 edges Königsberg bridge model.

```
C:\Windows\System32\cmd.exe - gremlin.bat
gremlin> kingsberg
==>graphtraversalsource[tinkergraph[vertices:4 edges:7], standard]
gremlin> kingsberg.V().sideEffect(outE("bridge").aggregate("bridges")).barrier().
.....1>         repeat(bothE().
.....2>             or(__.not(select('e')),
.....3>                 __.not(filter(__.as('x').select(all, 'e').unfold().
.....4>                     where(eq('x')))).as('e').
.....5>             otherV()).
.....6>         until(select(all, 'e').count(local).as("c").
.....7>             select("bridges").count(local).where(eq("c"))).hasNext()
==>false
gremlin>
```

After adding and edge from C -> A.

```
kingsberg = TinkerGraph.open().traversal()
```

```
kingsberg.addV().property(id, 'A').as('A').
```

```
    addV().property(id, 'B').as('B').
```

```
    addV().property(id, 'C').as('C').
```

```
    addV().property(id, 'D').as('D').
```

```
    addE('bridge').from('A').to('B').
```

```
    addE('bridge').from('A').to('B').
```

```
    addE('bridge').from('A').to('C').
```

```
    addE('bridge').from('B').to('D').
```

```
    addE('bridge').from('B').to('D').
```

```
    addE('bridge').from('B').to('C').
```

```
    addE('bridge').from('C').to('D').
```

```
    addE('bridge').from('C').to('A').iterate()
```

```
kingsberg
```

```
kingsberg.V().sideEffect(outE("bridge").aggregate("bridges")).barrier().
```

```
    repeat(bothE().
```

```
        or(__.not(select('e')),
```

```
            __.not(filter(__.as('x').select(all, 'e').unfold().
```

```
                where(eq('x')))).as('e').
```

```
        otherV()).
```

```
until(select(all, 'e').count(local).as("c").
      select("bridges").count(local).where(eq("c"))).hasNext()
```

Query to display the path when the above query displays true

```
g.V().sideEffect(outE("bridge").aggregate("bridges")).barrier().
```

```
repeat(bothE().or(__.not(select('e')),
    __.not(filter(__.as('x').select(all, 'e').unfold().
        where(eq('x')))).as('e').otherV()).
until(select(all, 'e').count(local).as("c").
      select("bridges").count(local).where(eq("c"))).limit(1).
path().by(id).by(constant(" -> ")).
map {String.join("", it.get().objects())}
```

```
C:\Windows\System32\cmd.exe - gremlin.bat
.....10>         addE('bridge').from('C').to('D').
.....11>         addE('bridge').from('C').to('A').iterate()
gremlin> konigsberg
=>graphtraversalsource[tinkergraph[vertices:4 edges:8], standard]
gremlin> konigsberg.V().sideEffect(outE("bridge").aggregate("bridges")).barrier().
.....1>         repeat(bothE().
.....2>             or(__.not(select('e')),
.....3>                 __.not(filter(__.as('x').select(all, 'e').unfold().
.....4>                     where(eq('x')))).as('e').
.....5>                     otherV()).
.....6>             until(select(all, 'e').count(local).as("c").
.....7>                 select("bridges").count(local).where(eq("c"))).hasNext()
=>true
gremlin> g.V().sideEffect(outE("bridge").aggregate("bridges")).barrier().
.....1>         repeat(bothE().or(__.not(select('e')),
.....2>             __.not(filter(__.as('x').select(all, 'e').unfold().
.....3>                 where(eq('x')))).as('e').otherV()).
.....4>             until(select(all, 'e').count(local).as("c").
.....5>                 select("bridges").count(local).where(eq("c"))).limit(1).
.....6>             path().by(id).by(constant(" -> ")).
.....7>             map {String.join("", it.get().objects())}
=>B -> D -> B -> C -> A -> B -> A -> C -> D
gremlin>
```