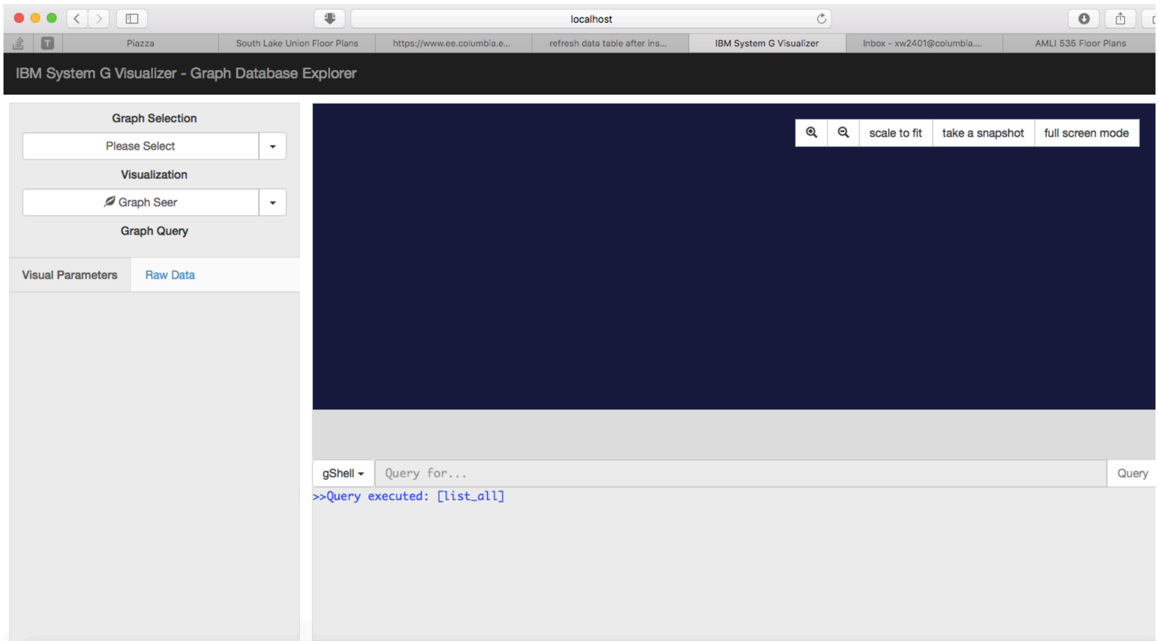
Big Data Analytics Assignment 3
Name: Xucan Wang Uni: xw2401

Part 1. Graph database

1）Download IBM System G Graph Tools



2）Create a knowledge graph and try graph queries to find relevant items

A. dataset preparation part:

https://dumps.wikimedia.org/enwiki/20161020/

enwiki-20161020-page.sql.gz 1.4 GB

2016-10-21 11:53:03 **done** Category information.
enwiki-20161020-category.sql.gz 19.4 MB

2016-10-21 11:52:57 **done** User group assignments.
enwiki-20161020-user_groups.sql.gz 193 KB

2016-10-21 11:52:53 **done** Wiki interlanguage link records.
enwiki-20161020-langlinks.sql.gz 267.4 MB

I used the page.sql and the langlinks.sql from the dataset above.

B. Use the SQL file to create tables in MySQL database

```
mysql> source /Users/wangxucan/Downloads/enwiki-20161020-langlinks.sql;
Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

```
mysql> show tables;
+-----------------+
| Tables_in_test |
+-----------------+
| langlinks      |
+-----------------+
1 row in set (0.00 sec)
```

After the langlinks table is created, we only get a subset of the langlinks data and store the data into the langlink_mini table (which has 10000 entries)

```
mysql> SELECT * INTO langlink_mini FROM langlinks Limit 10000;
ERROR 1327 (42000): Undeclared variable: langlink_mini
mysql> CREATE TABLE `langlinks_mini` (
    ->   `ll_from` int(8) unsigned NOT NULL DEFAULT '0',
    ->   `ll_lang` varbinary(20) NOT NULL DEFAULT '',
    ->   `ll_title` varbinary(255) NOT NULL DEFAULT '',
    ->   UNIQUE KEY `ll_from` (`ll_from`,`ll_lang`),
    ->   KEY `ll_lang` (`ll_lang`,`ll_title`)
    -> ) ENGINE=InnoDB DEFAULT CHARSET=binary;
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> INSERT INTO langlinks_mini select * FROM langlinks Limit 10000;
Query OK, 10000 rows affected (0.32 sec)
Records: 10000  Duplicates: 0  Warnings: 0
```

```
mysql> select count(*) from langlinks_mini;
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> source /Users/wangxucan/Downloads/enwiki-20161020-page.sql;
```

```
mysql> select count(*) from page;
+----------+
| count(*) |
+----------+
|  3416829 |
+----------+
1 row in set (2.61 sec)
```

As the number of data in the page table is too oversized to do the join job, we still take a subset of the table and store it in the page_mini table (we got 100000 records of page)

```
mysql> CREATE TABLE `page_mini` (
    ->   `page_id` int(8) unsigned NOT NULL AUTO_INCREMENT,
    ->   `page_namespace` int(11) NOT NULL DEFAULT '0',
    ->   `page_title` varbinary(255) NOT NULL DEFAULT '',
    ->   `page_restrictions` tinyblob NOT NULL,
    ->   `page_counter` bigint(20) unsigned NOT NULL DEFAULT '0',
    ->   `page_is_redirect` tinyint(1) unsigned NOT NULL DEFAULT '0',
    ->   `page_is_new` tinyint(1) unsigned NOT NULL DEFAULT '0',
    ->   `page_random` double unsigned NOT NULL DEFAULT '0',
    ->   `page_touched` varbinary(14) NOT NULL DEFAULT '',
    ->   `page_links_updated` varbinary(14) DEFAULT NULL,
    ->   `page_latest` int(8) unsigned NOT NULL DEFAULT '0',
    ->   `page_len` int(8) unsigned NOT NULL DEFAULT '0',
    ->   `page_content_model` varbinary(32) DEFAULT NULL,
    ->   PRIMARY KEY (`page_id`),
    ->   UNIQUE KEY `name_title` (`page_namespace`,`page_title`),
    ->   KEY `page_random` (`page_random`),
    ->   KEY `page_len` (`page_len`),
    ->   KEY `page_redirect_namespace_len` (`page_is_redirect`,`page_namespace`,
`page_len`)
    -> ) ENGINE=InnoDB AUTO_INCREMENT=52057688 DEFAULT CHARSET=binary;
Query OK, 0 rows affected (0.34 sec)
```

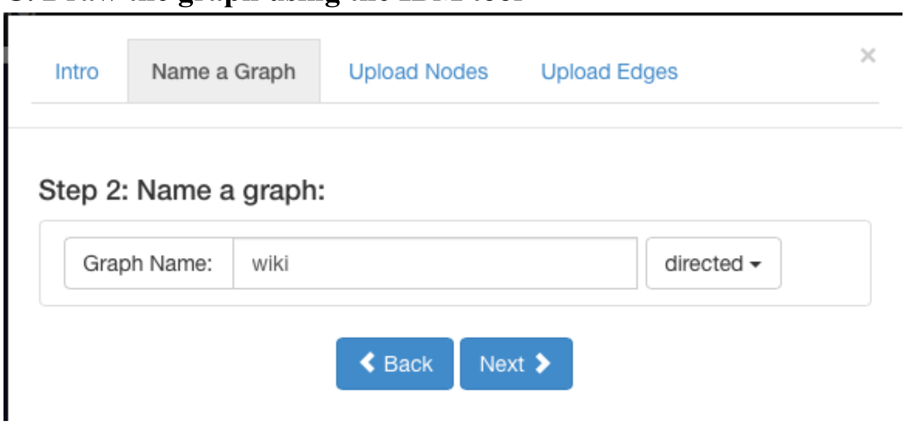
```
mysql> INSERT INTO page_mini select * FROM page Limit 100000;
```

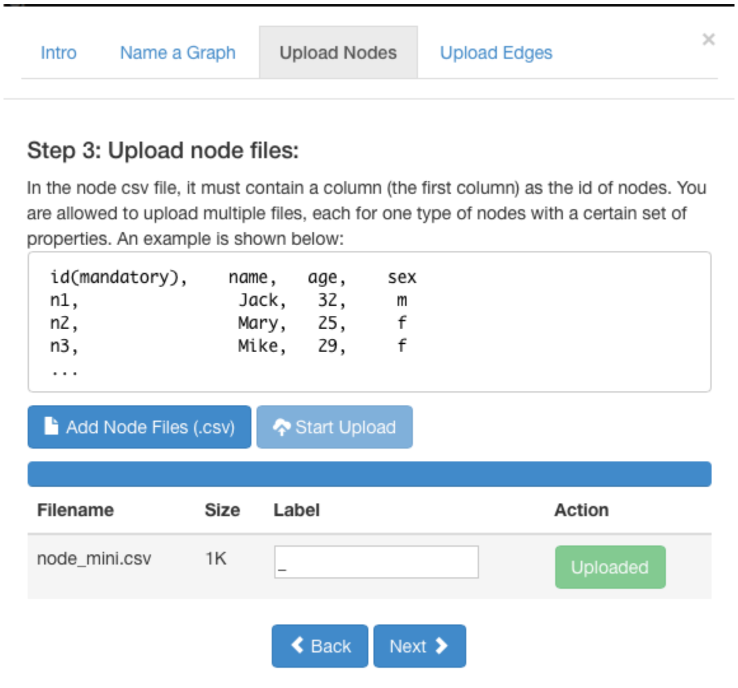
For now, we have prepared the datasets ready for join. According to the schema of the page table and langlinks table. We can get the edge.csv file with a JOIN action.

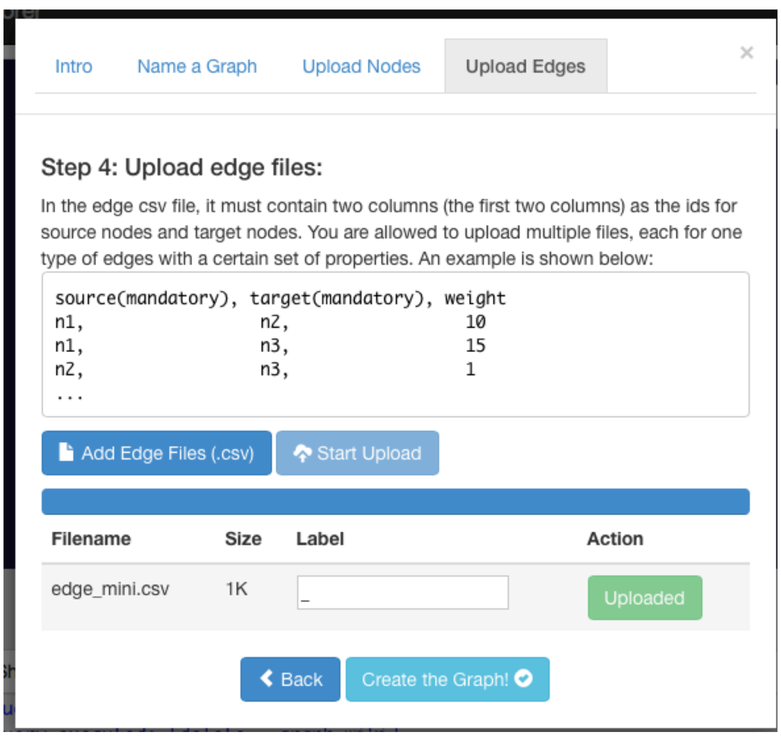
```
mysql> SELECT langlinks_mini.ll_from,page_mini.page_id
    -> FROM langlinks_mini JOIN page_mini
    -> ON langlinks_mini.ll_title=page_mini.page_title AND
    -> langlinks_mini.ll_from != page_mini.page_id
    -> INTO OUTFILE '/Users/wangxucan/Desktop/edge_mini.csv'
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n';
```

After this, we got the edge_mini and the node_mini csv file. Since we already got the edge_mini.csv we can copy all the page_id that exists in edge.csv to the node_mini csv to include all the nodes existed in

C. Draw the graph using the IBM tool

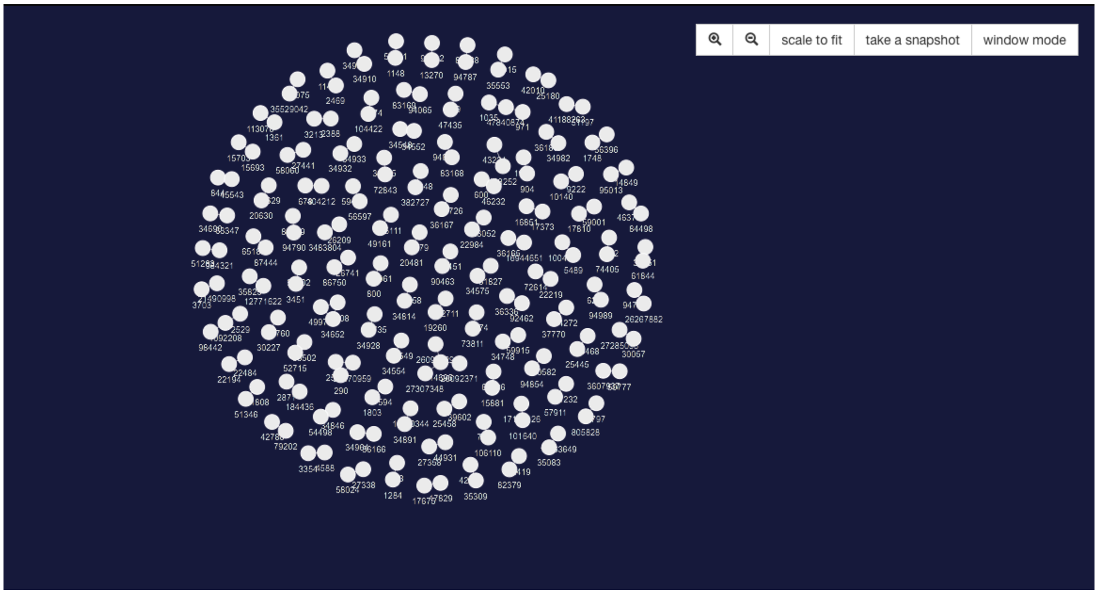




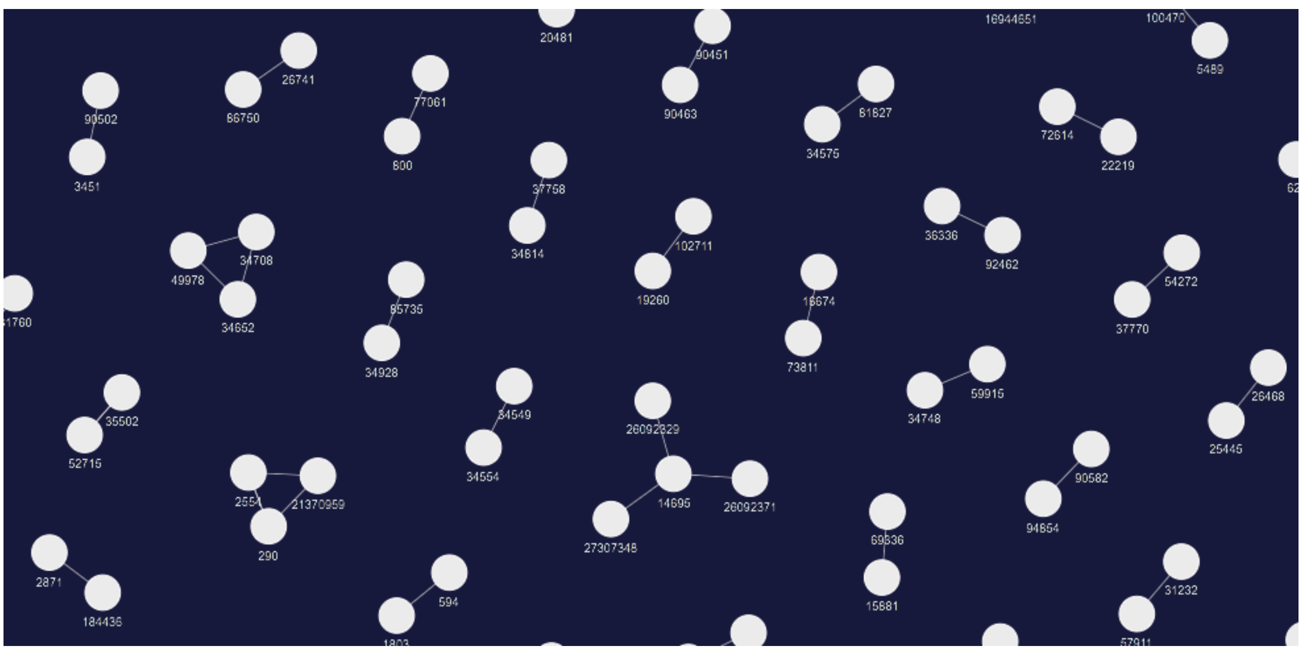


After we draw the graph, it looks like this:

```
>>Result received:
>>number of nodes: 204
>>number of edges: 114
```



Part of the detailed graph looks like this:



The raw data on the side looks like this:

Visual Parameters | Raw Data

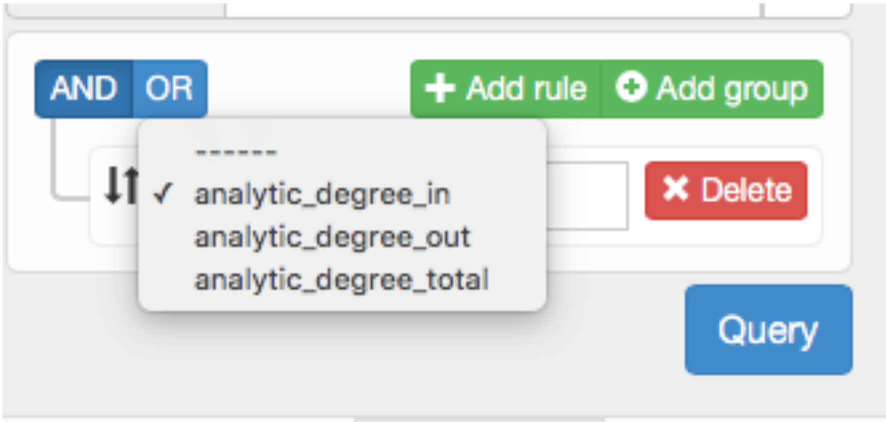

```
{
  - edges[114]: [
      - {
            source: "11370344",
            target: "34891",
            eid: "1",
            label: "_"
        },
      - {
            source: "34652",
            target: "34708",
            eid: "2",
            label: "_"
        },
      - {
            source: "34652",
            target: "49978",
            eid: "39",
            label: "_"
        },
```

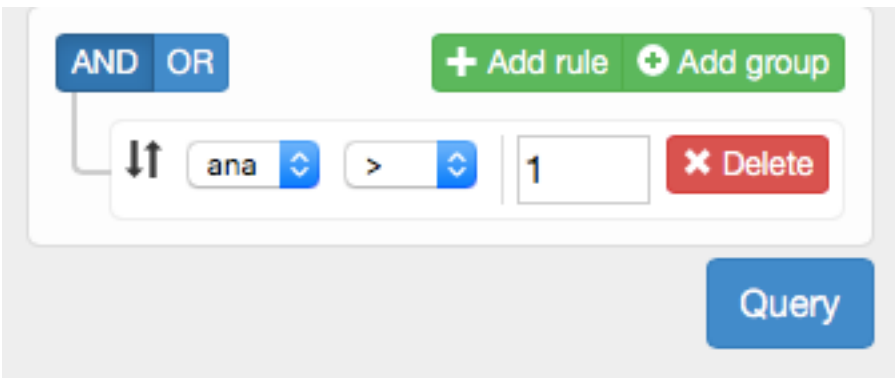

Visual Parameters | Raw Data


```
        },
      - {
            source: "404212",
            target: "678",
            eid: "8",
            label: "_"
        },
      - {
            source: "805828",
            target: "1797",
            eid: "9",
            label: "_"
        },
      - {
            source: "382727",
            target: "66248",
            eid: "10",
            label: "_"
        },
      - {
```


D. Try graph queries on the nodes

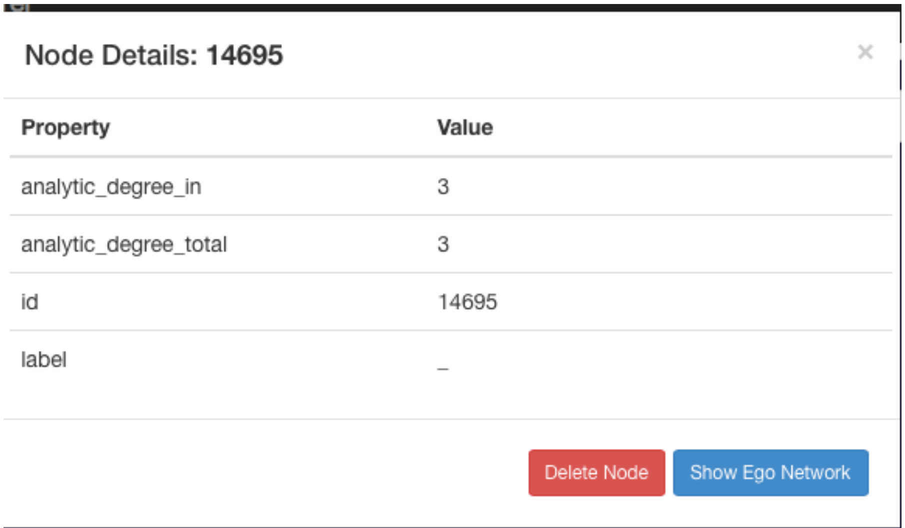
We can use the graph query to get some of the sub graph, for example:

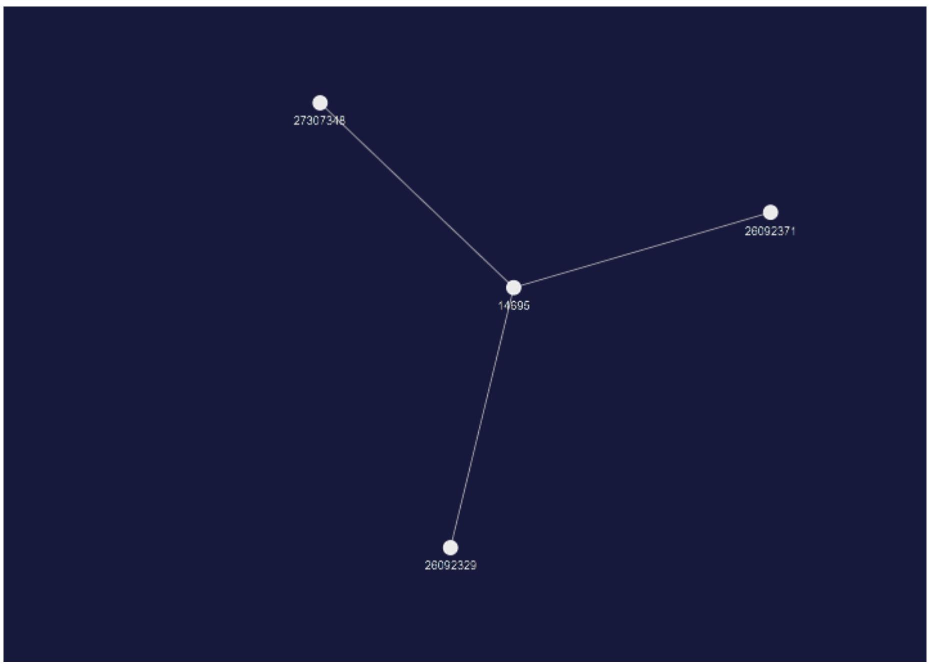




In this query, we let the graph to show the nodes which has more than 1 in_degree. The result looks like this, every point showing is the node that has in_degree>1

When we click into one of the specific node 14695, we can find that, its in_degree is 3.

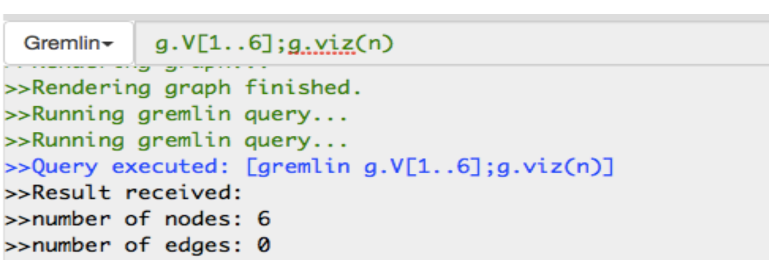




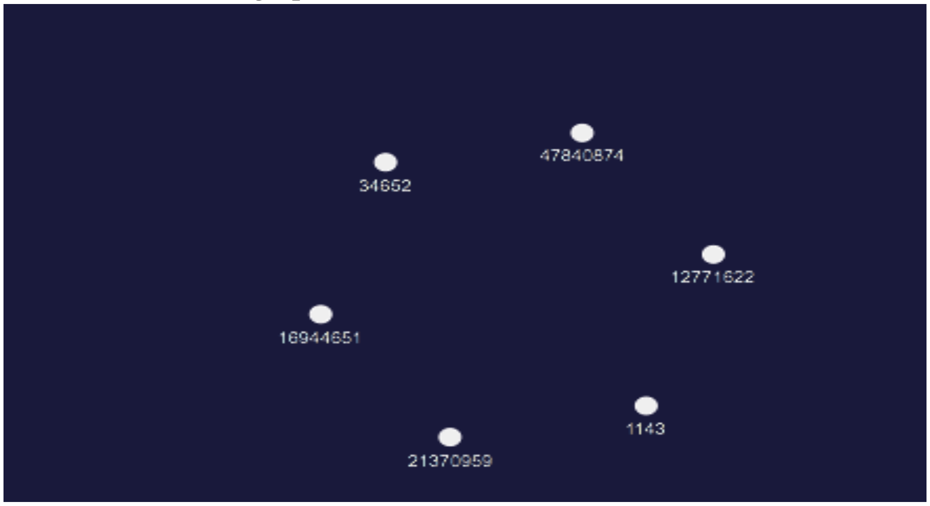
Gremlin Query:

(1) Get all the vertices with ID range:

To get the vertex object itself you can use below query g.V[1..n], which get the vertices from range 1~n

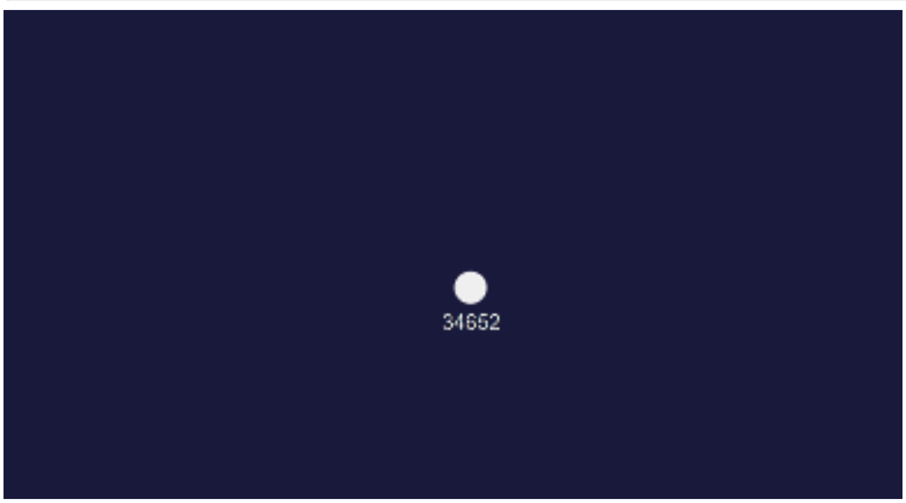


And the visualized graph looks like this:



In this case, we get only 6 vertices in the result.

(2)



(3) Get the count of all outgoing edges of a vertex.

```
Gremlin▾  g.v(1).outE().count();
>>number of nodes: 204
>>number of edges: 114
>>Rendering graph...
>>Rendering graph finished.
>>Running gremlin query...
>>Running gremlin query...
>>Query executed: [gremlin g.v(1).outE().count();]
>>===>2
```

As we can see in the result, the number of outgoing edges of the vertex with Id 1 has 2 outgoing edges.

(4) Get The Label of All Out Going Edges of A Vertex

```
>>Query executed: [gremlin g.v(1).outE().label;]
>>===>_
>>===>_
```

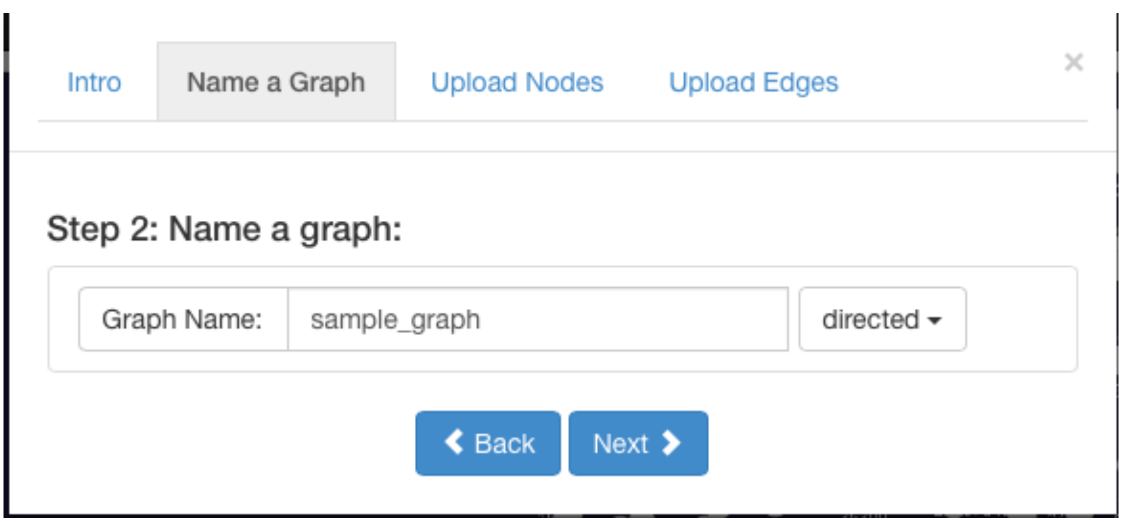
From part 3 we know that there are 2 outgoing edges of vertex with Id 1. Now we are trying to get the label of all out going edges of a vertex. As we can see in the query result, both the labels are '_' because when we create the graph and upload the node and edge csv, we didn't specify the label. So the default value is '_'

Part 2. Graph Topology Analysis

1) Graph Preparation

Choose a subgraph of the graph in part 1 or another graph you want to do

In this case, **I used another graph which I created myself. The node is from A to M.**

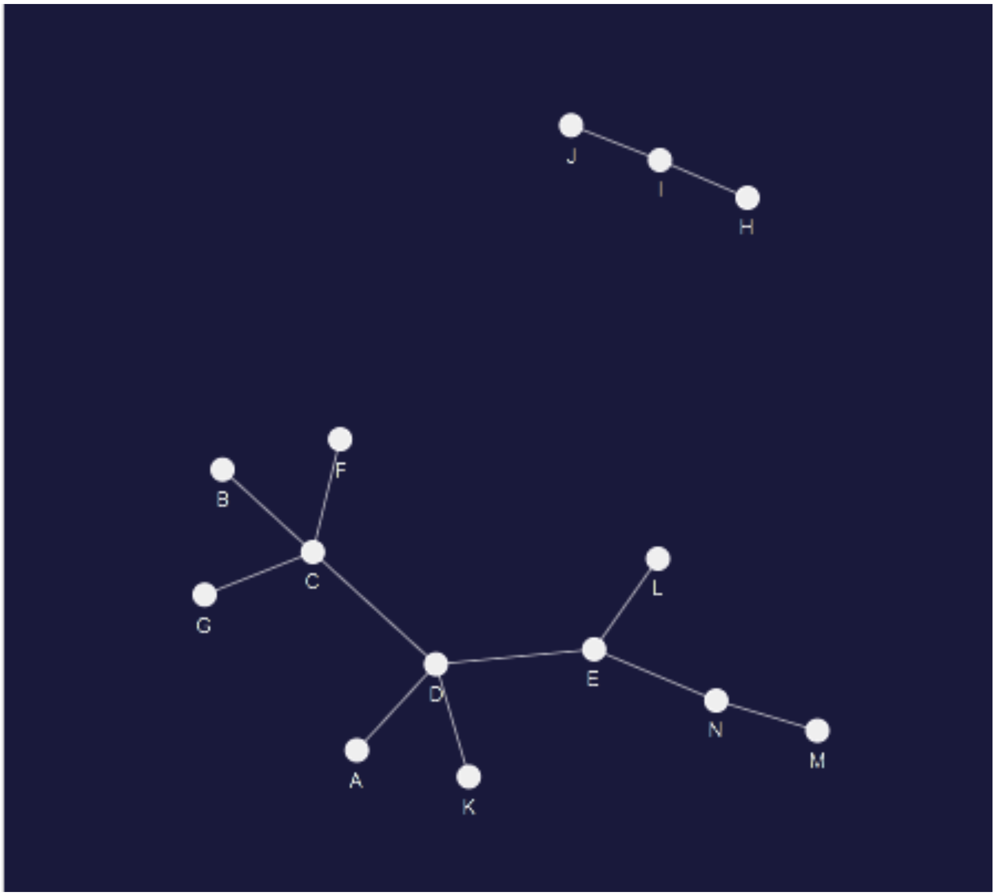


```
>>Query executed: [list_all]
>>Query executed: [gremlin g=CreateGraph.openGraph("sgtrans","sample_graph")]
>>Retrieving graph sample_graph...
>>Query executed: [get_num_vertices --graph sample_graph]
>>Query executed: [print_all --graph sample_graph]
>>Result received:
>>number of nodes: 14
>>number of edges: 12
```

Edge file looks like the following

	A	B
1	source	target
2	C	B
3	C	G
4	F	C
5	D	C
6	D	A
7	D	E
8	D	K
9	E	L
10	N	E
11	N	M
12	H	I
13	J	I

The sample_graph (directed graph) looks like this:



2) Calculate the centralities:
a. In-degree, out-degree

Degree centrality: `analytic_degree_centrality`

```
analytic_degree_centrality --graph <graph_name> --in <input_cache_name>
```

gShell ▾

```
analytic_degree_centrality --graph sample_graph
```

Query

result:

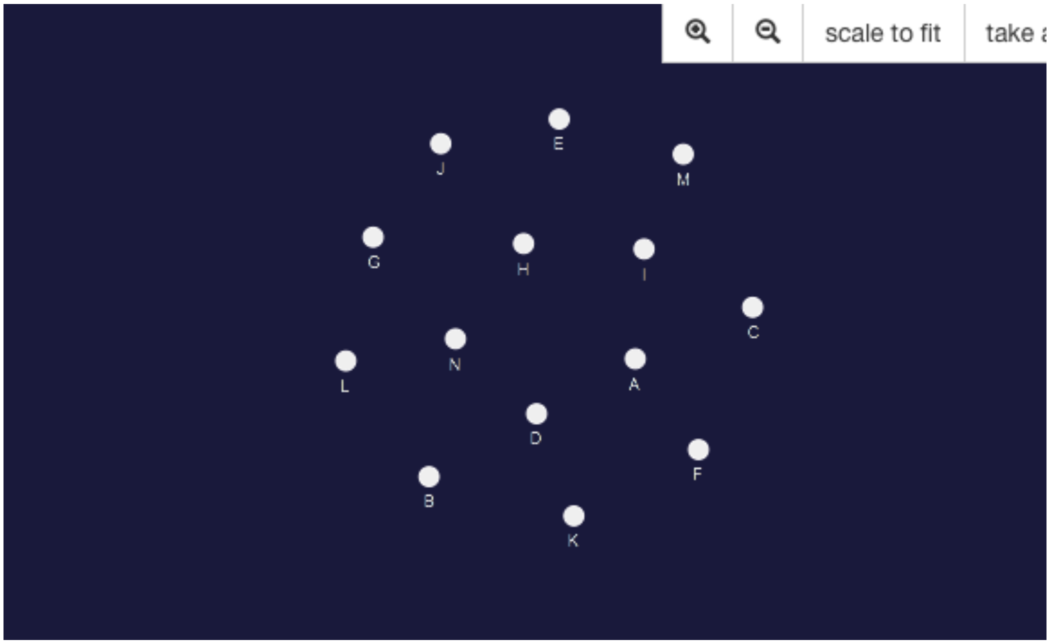
```
>>stat: avg
>>metric: analytic_degree_total
>>value: 1.714286
>>stat: avg
>>metric: analytic_degree_in
>>value: 0.857143
>>stat: avg
>>metric: analytic_degree_out
>>value: 0.857143
```

```
>>stat: max
>>metric: analytic_degree_total
>>value: 4
>>stat: max
>>metric: analytic_degree_in
>>value: 2
>>stat: max
>>metric: analytic_degree_out
>>value: 4
```

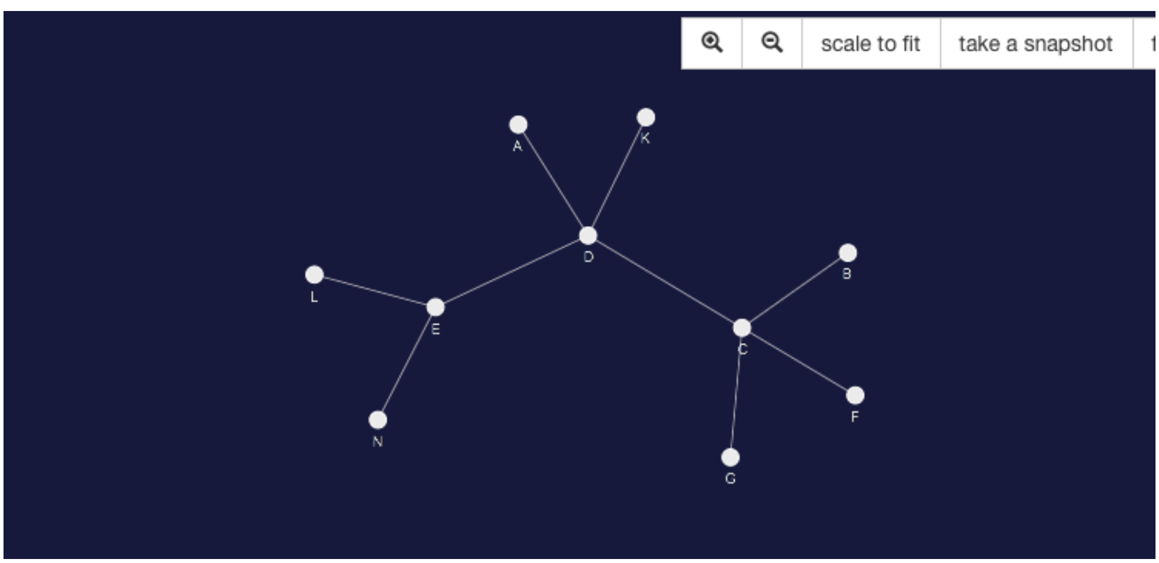
```
>>stat: min
>>metric: analytic_degree_total
>>value: 1
>>stat: min
>>metric: analytic_degree_in
>>value: 0
>>stat: min
>>metric: analytic_degree_out
>>value: 0
```

From the result we can see the whole in_degree and out_degree statistics of the sample_graph. The avg_degree_total = 1.714286, avg_degree_out= 0.857143, avg_degree_in=0.857143
max degree_total = 4, max_degree_in = 2(The node with most degree in is like node I which node J and H both points to it)

max_degree_out = 4, min_degree_total = 1, min_degree_in = 0 (vertex like F has no degree in), min_degree_out = 0 (vertex like I has no degree out)



When you click in one of the node and show the ego network, it looks like this:



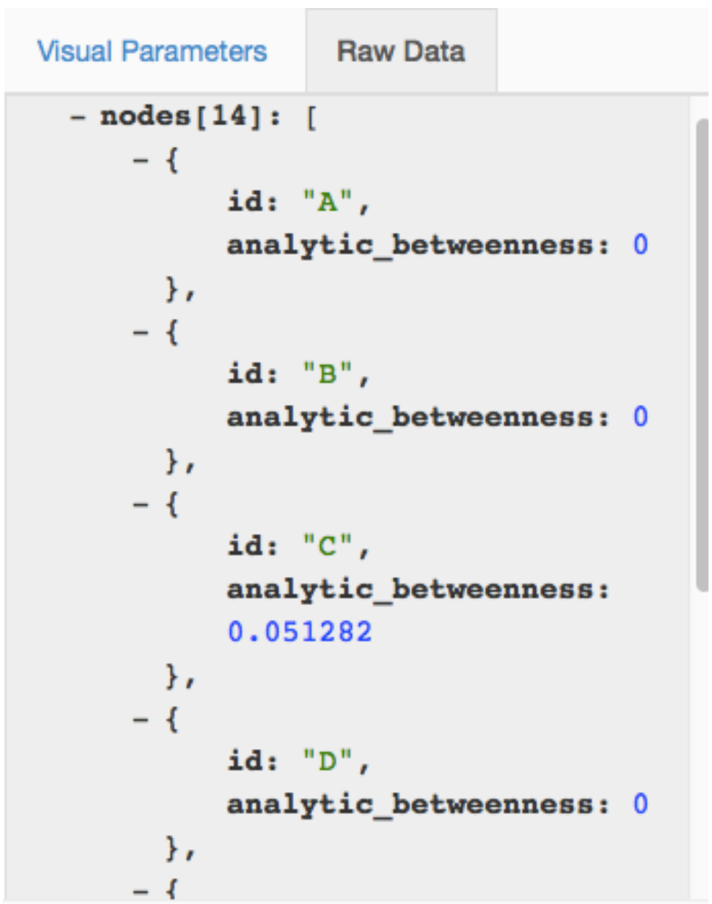
b. Betweenness

Betweenness centrality: `analytic_betweenness_centrality`

```
analytic_betweenness_centrality --graph <graph_name> --in <input_cache_name> --ignoreedgeweight
```

This analytic computes the betweenness centrality of every vertex in the graph.

Result:



This analytic computes the betweenness centrality of every vertex in the graph. By default, the betweenness centrality of each vertex is written to the graph store as a vertex property "analytic_betweenness", unless --redirect is used to redirect output to an external file.

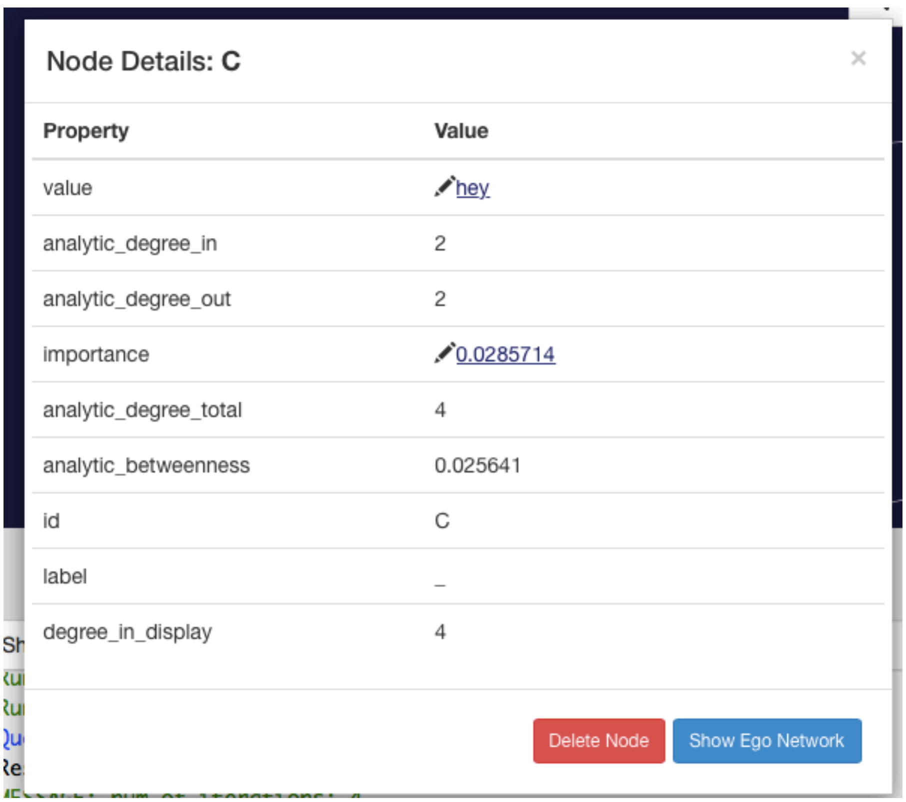
c. PageRank

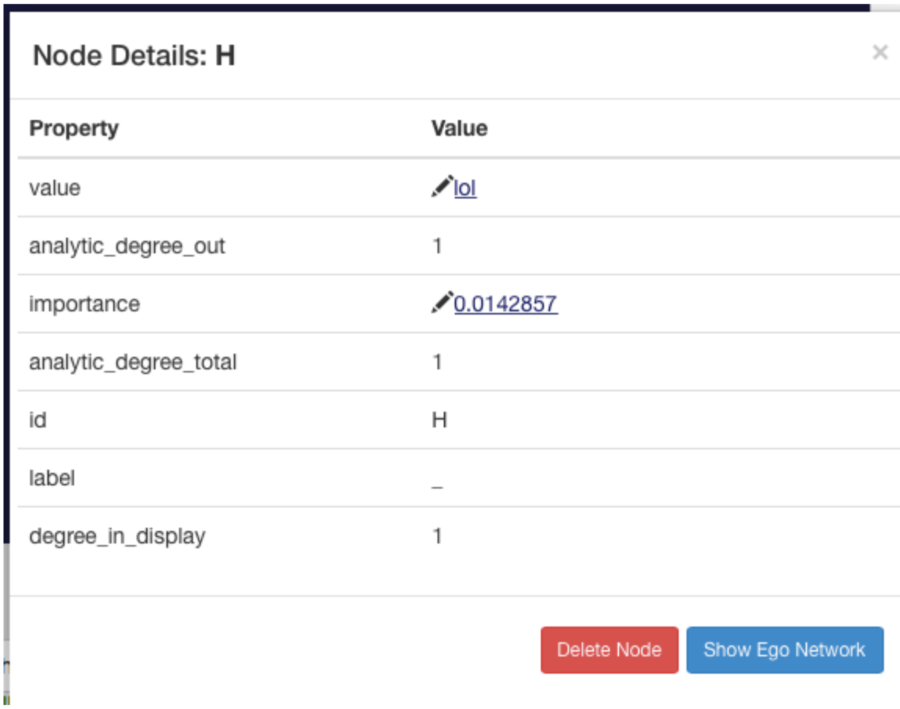
PageRank: `analytic_pagerank`

```
analytic_pagerank --graph <graph_name> --damp <damping_factor> --quad <quadratic_error> --num <max_num_iterations>
--prop <vertex_prop_name_to_store_result> --restart
```

This analytic performs persistent PageRank in a directed graph. By persistent PageRank, we mean that the importance value of each vertex in each iteration is stored as a vertex property (specified by –prop. In this case, we use importance to store this property). Thus, we can incrementally perform PageRank at any time, or after any changes to the graph.

```
>>Query executed: [analytic_pagerank --graph sample_graph --num 100 --prop importance --restart]
>>Result received:
```

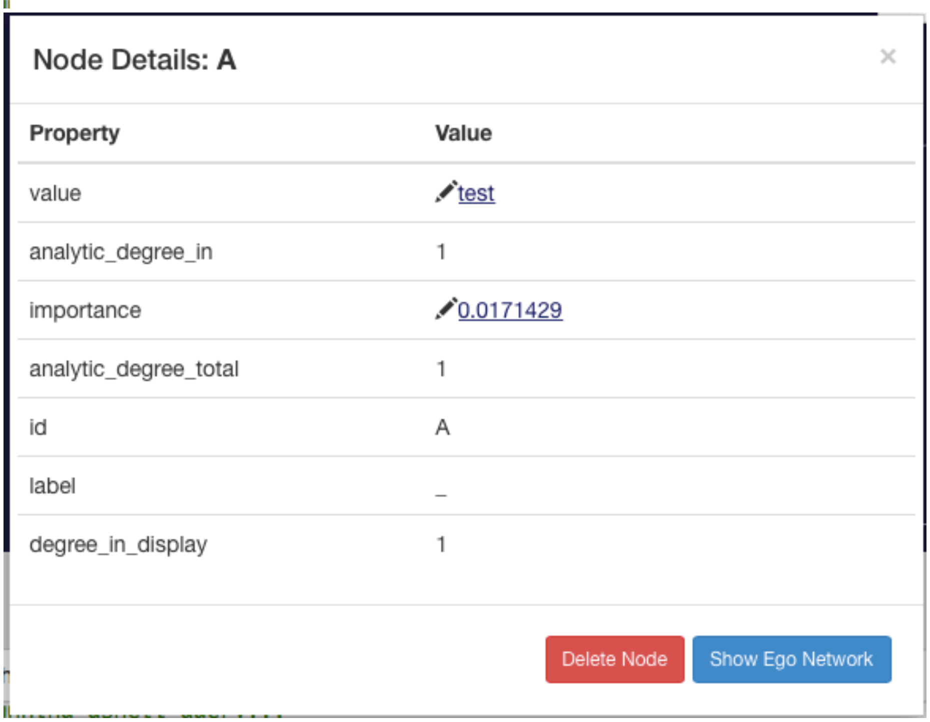




Node Details: H

Property	Value
value	lol
analytic_degree_out	1
importance	0.0142857
analytic_degree_total	1
id	H
label	_
degree_in_display	1

Delete Node Show Ego Network



Node Details: A

Property	Value
value	test
analytic_degree_in	1
importance	0.0171429
analytic_degree_total	1
id	A
label	_
degree_in_display	1

Delete Node Show Ego Network

In my opinion, pageRank shows the importance of a singe vertex in the graph. For example, if the deletion of a vertex affects more vertexes nearby, then the value of the page rank is high. Otherwise, if the vertex has no in_degree or out_degree. Then the value of the page rank is very low because the modification of the vertex has barely no effect to the whole graph instead of itself.

3) Choose some points, calculate the shortest path between them.

Shortest paths: `analytic_shortest_paths`

```
analytic_shortest_paths --graph <graph_name> --in <input_cache_name> --src <src_vid> --sink <sink_vid> --ignoreedgeweight --hidepath
```

```
gShell ▾ | analytic_shortest_paths --graph sample_graph  --src C --sink L --ignoreedgeweight | Query
```

```
>>Query executed: [analytic_shortest_paths --graph sample_graph --src C --sink L --ignoreedgeweight ]
>>Result received:
>>src: C
>>sink: L
>>distance: -1
>>path:
```

As we can see, the sample_graph is a directed graph and there is no path from C to L, so the distance = -1 and the path is null

```
gShell ▾ | analytic_shortest_paths --graph sample_graph  --src N --sink L --ignoreedgeweight | Query
```

```
>>Query executed: [analytic_shortest_paths --graph sample_graph --src N --sink L --ignoreedgeweight ]
>>Result received:
>>src: N
>>sink: L
>>distance: 2
>>path: N-->E-->L
```

In this query, we can see N is 2 hops away from L, and the path is N->E->L

Part 3: Bayesian Network

3.1: Define the Bayesian Network structure

g_shell.txt

```
create --graph Esp1 --type directed
add_vertex --graph Esp1 --id 0 --label Attack --prop obs:0
add_vertex --graph Esp1 --id 1 --label Communication --prop obs:0
add_vertex --graph Esp1 --id 2 --label GatherInfo --prop obs:0
add_vertex --graph Esp1 --id 3 --label Eml --prop obs:1 col:x1 thresh:[0.85]
add_vertex --graph Esp1 --id 4 --label LOF --prop obs:1 col:x2 thresh:[0.005]
add_vertex --graph Esp1 --id 5 --label HTTP --prop obs:1 col:x3 thresh:[0.3]
add_vertex --graph Esp1 --id 6 --label queryTerm --prop obs:1 col:x4 thresh:[0.3]
add_edge --graph Esp1 --src 1 --targ 0
add_edge --graph Esp1 --src 2 --targ 0
add_edge --graph Esp1 --src 5 --targ 2
add_edge --graph Esp1 --src 6 --targ 2
add_edge --graph Esp1 --src 3 --targ 1
add_edge --graph Esp1 --src 4 --targ 1
```

print_all graph Esp1 –redirect ~/BN_pipeline/testdir/input.json

After creating the graph and use the command to print_all graph, we get the input.json file like this. This file shows the structure of the whole graph input.json

```json
{
"edges":[
{"source":"1","target":"0","eid":"0","label":"_"},
{"source":"1","target":"0","eid":"6","label":"_"},
{"source":"2","target":"0","eid":"1","label":"_"},
{"source":"2","target":"0","eid":"7","label":"_"},
{"source":"3","target":"1","eid":"4","label":"_"},
{"source":"3","target":"1","eid":"10","label":"_"},
{"source":"4","target":"1","eid":"5","label":"_"},
{"source":"4","target":"1;","eid":"11","label":"_"},
{"source":"5","target":"2","eid":"2","label":"_"},
{"source":"5","target":"2","eid":"8","label":"_"},
{"source":"6","target":"2","eid":"3","label":"_"},
{"source":"6","target":"2","eid":"9","label":"_"}
],
"nodes":[
{"id":"1;","label":"_"},
{"id":"0","label":"Attack","obs":"0"},
{"id":"1","label":"Communication","obs":"0"},
{"id":"2","label":"GatherInfo","obs":"0"},
{"id":"3","label":"Eml","obs":"1","col":"x1","thresh":"[0.85]"},
{"id":"4","label":"LOF","obs":"1","col":"x2","thresh":"[0.005]"},
{"id":"5","label":"HTTP","obs":"1","col":"x3","thresh":"[0.3]"},
{"id":"6","label":"queryTerm","obs":"1","col":"x4","thresh":"[0.3]"}
],
"properties":[
{"type":"node","name":"thresh","value":"string"},
{"type":"node","name":"col","value":"string"},
{"type":"node","name":"obs","value":"string"}
],
"summary":[{"number of nodes":8,"number of edges":12}],
"time":[{"TIME":"0.00031805"}]
}
```

Line 34, Column 1 Tab Size: 4 JSON

As we see in the file, we created a graph with 8 nodes and 12 edges.

Graph rule structure:

The BNRules.xml file specifies the rules of the graph.

BNRules.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
 (C) Copyright IBM Corp. 2016
     All Rights Reserved
-->
<BNRules>
    <category name="spatial" rulenum="3">
        <ruleItem index="1" relation="and" target="0">
            <rule val="0">1</rule>
            <rule val="0">2</rule>
        </ruleItem>
        <ruleItem index="2" relation="and" target="1">
            <rule val="1">3</rule>
            <rule val="0">4</rule>
        </ruleItem>
        <ruleItem index="3" relation="and" target="2">
            <rule val="0">5</rule>
            <rule val="0">6</rule>
        </ruleItem>
    </category>
    <category name="temporal" rulenum="1">
        <ruleItem index="1" target="5">
            <rule>4</rule>
        </ruleItem>
    </category>
</BNRules>
```

Within “spatial” category, each rule is defined within the "ruleItem" element, and attribute “index” denotes the "ruleItem" index. Attribute “relation” denotes the rule combination relationship which could either be “and”, “or”, or “sum”. Attribute “target” denotes which node id the rule will effect on.

For the

<rule val="0">1</rule> <rule val="0">2</rule>means if node_1>0 && node_2>0, then the target node_0=1

<rule val="1">3</rule> <rule val="0">4</rule> means if node_3 > 1 && node_4 > 0, then the target node_1 = 1, otherwise node_1 = 0.

<rule val="0">5</rule> <rule val="0">6</rule> means if node_5>0&& node_6>0, then target node_2=1

Within the “temporal” category, each rule is still defined within the “ruleItem” element, and attribute “index” denotes the “ruleItem” index.

<ruleItem index=”1” target=”5”>

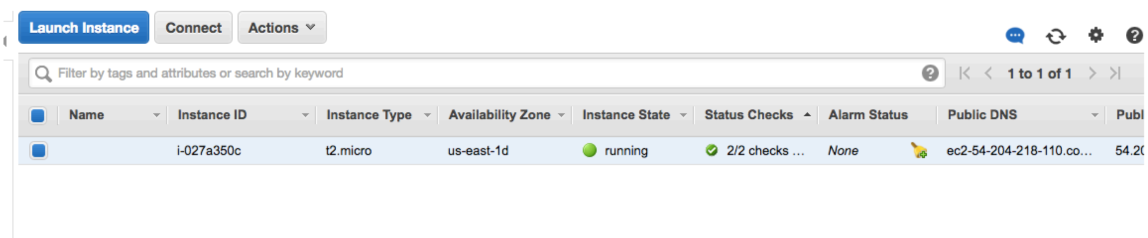
<rule>4</rule>

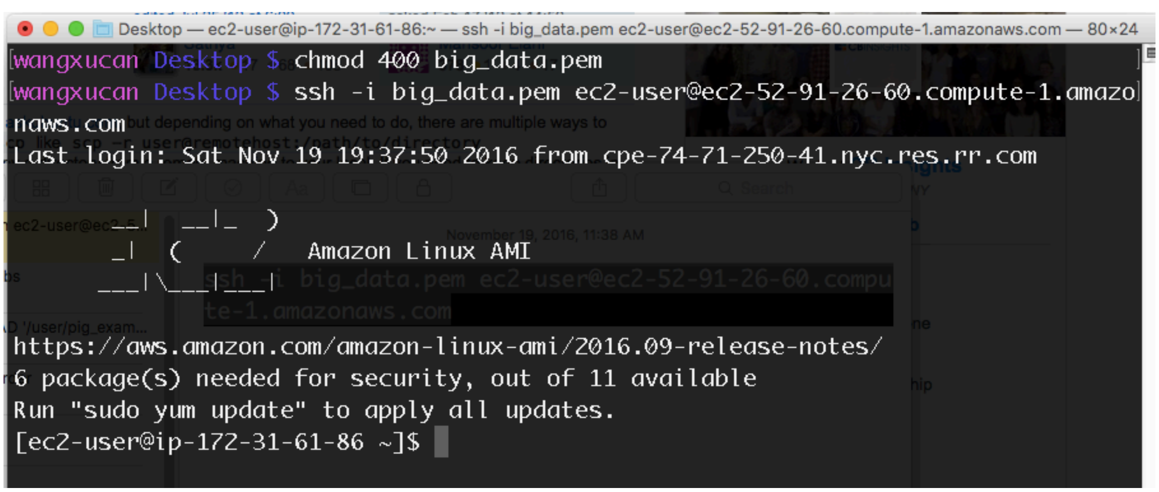
<ruleItem> means that with contain Markovian transition probabilities the value of the target node_5 can be replaced by the trigger node_4.

The feature of each node is defined in the BN_feature.json file.

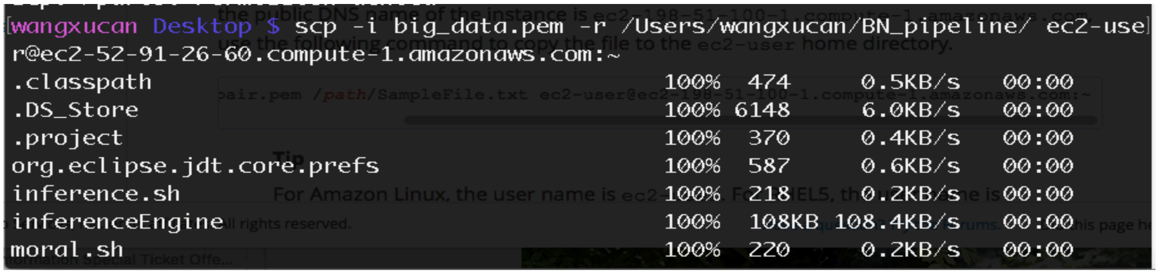
Since running on MAC it will cause the problem of can’t find file YYY, PPP

I create a linux instance on AWS and use it to run the script





First, we need to copy the entire BN_pipeline file to the AWS instance:



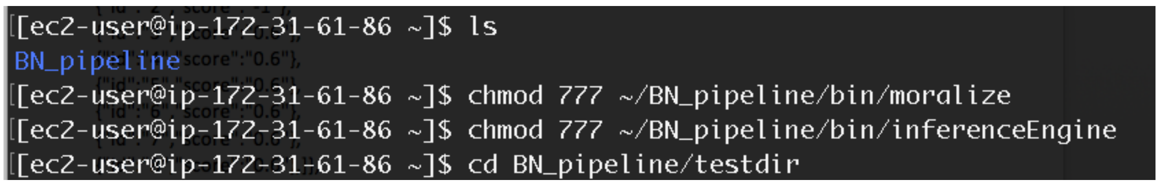
```
wangxucan Desktop $ scp -i big_data.pem -r /Users/wangxucan/BN_pipeline/ ec2-user@ec2-52-91-26-60.compute-1.amazonaws.com:~
.classpath                                   100%  474      0.5KB/s   00:00
.DS_Store                                    100% 6148      6.0KB/s   00:00
.project                                     100%  370      0.4KB/s   00:00
org.eclipse.jdt.core.prefs                   100%  587      0.6KB/s   00:00
inference.sh                                 100%  218      0.2KB/s   00:00
inferenceEngine                              100%  108KB 108.4KB/s    00:00
moral.sh                                     100%  220      0.2KB/s   00:00
```

Then after we execute the following command:

chmod 777 ~/BN_pipeline/bin/moralize

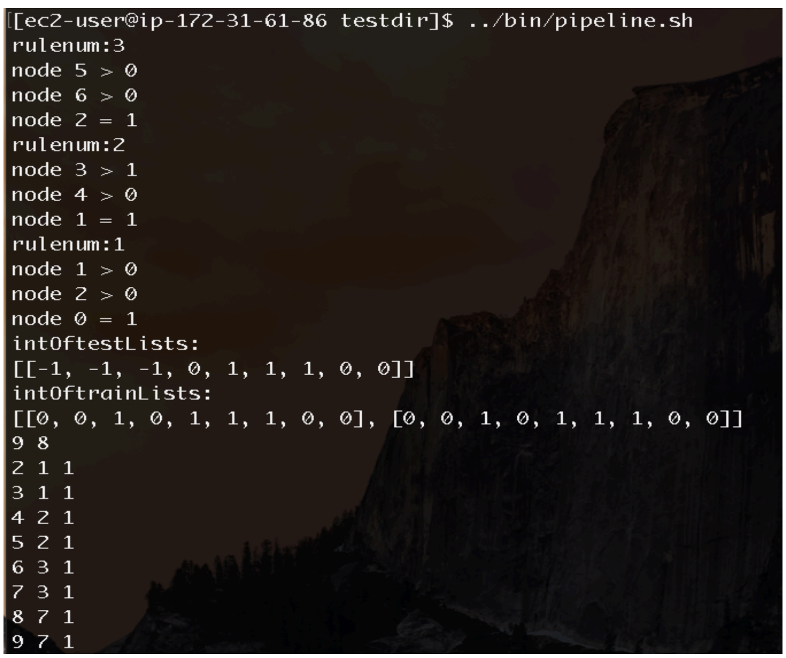
chmod 777 ~/BN_pipeline/bin/inferenceEngine

we made the moralize and inferenceEngine runnable on the aws linux instance.



```
[ec2-user@ip-172-31-61-86 ~]$ ls
BN_pipeline
[ec2-user@ip-172-31-61-86 ~]$ chmod 777 ~/BN_pipeline/bin/moralize
[ec2-user@ip-172-31-61-86 ~]$ chmod 777 ~/BN_pipeline/bin/inferenceEngine
[ec2-user@ip-172-31-61-86 ~]$ cd BN_pipeline/testdir
```

we run the BN Tool command



```
[ec2-user@ip-172-31-61-86 testdir]$ ../bin/pipeline.sh
rulenum:3
node 5 > 0
node 6 > 0
node 2 = 1
rulenum:2
node 3 > 1
node 4 > 0
node 1 = 1
rulenum:1
node 1 > 0
node 2 > 0
node 0 = 1
intOftestLists:
[[-1, -1, -1, 0, 1, 1, 1, 0, 0]]
intOftrainLists:
[[0, 0, 1, 0, 1, 1, 1, 0, 0], [0, 0, 1, 0, 1, 1, 1, 0, 0]]
9 8
2 1 1
3 1 1
4 2 1
5 2 1
6 3 1
7 3 1
8 7 1
9 7 1
```

The inference result is in output_inf_std.txt and output_inf_visual.txt， then I compared the generated results with the expected results. Which is the same as in the expected results.

```
[ec2-user@ip-172-31-61-86 testdir]$ ls
Bayesian_network.dag        expected_results    node_mapping.txt
BN_feature.json             input.json          output_inf_std.txt
BNRules.xml                 internalFile        output_inf_visual.txt
CliqueInfoPipe.property     junction_tree.txt   stdout.log
configFile                  moral.log           temporal_rules.txt
evidence.txt                moral.stderr.log
```

```
[ec2-user@ip-172-31-61-86 testdir]$ vim output_inf_std.txt
```

Desktop — ec2-user@ip-172-31-61-86:~/BN_pipeline/testdir — ssh -i big_data.pem ec2-user@ec2-54-204-218-110.compute-1.a...

```
Exec time for training [sec]:   1.368523e-04
NodeID: 0       query output:
0       0.999999999000000
1       0.000000000000000
NodeID: 1       query output:
0       0.999999999000000
1       0.000000000000000
NodeID: 2       query output:
0       0.000000000000000
1       0.999999999000000
NodeID: 3       query output:
0       0.999999999000000
1       0.000000000000000
NodeID: 4       query output:
0       0.000000000000000
1       0.999999999000000
NodeID: 5       query output:
0       0.000000000000000
1       0.999999999000000
NodeID: 6       query output:
0       0.000000000000000
1       0.999999999000000
NodeID: 7       query output:
0       0.999999999000000
1       0.000000000000000
NodeID: 8       query output:
0       0.999999999000000
1       0.000000000000000
Query Node: query output:
0       0.999999999000000
                                                1,1           Top
```

In the output file, in line "query output NodeID 0: 1.000000000000000 0.000000000000000", since we have two column, that means we have two states for node 0. The first column indicates the probability of node 0 being 0, the second column indicates the probability of node 0 being 1.

```
Evidence filename:      evidence.txt
UserID:         A0001
observation string 1: -1 -1 -1 0 1 1 1 0 0
query output NodeID 1:  1.000000000000000       0.000000000000000
                                                            35,1          Bot
```

```
[ec2-user@ip-172-31-61-86 testdir]$ vim output_inf_visual.txt
```

Desktop — ec2-user@ip-172-31-61-86:~/BN_pipeline/testdir — ssh -i big_data.pem ec2-user@ec2-54-204-218-110.compute-1.a...

```
Post-training distribution:
NodeID: 0           0.999999999000000       0.000000000000000
NodeID: 1           0.999999999000000       0.000000000000000
NodeID: 2           0.000000000000000       0.999999999000000
NodeID: 3           0.999999999000000       0.000000000000000
NodeID: 4           0.000000000000000       0.999999999000000
NodeID: 5           0.000000000000000       0.999999999000000
NodeID: 6           0.000000000000000       0.999999999000000
NodeID: 7           0.999999999000000       0.000000000000000
NodeID: 8           0.999999999000000       0.000000000000000
Query Node:     0.999999999000000       0.000000000000000
observed evidence 1: -1 -1 -1 0 1 1 1 0 0
query output NodeID 0:  1.000000000000000       0.000000000000000
query output NodeID 1:  1.000000000000000       0.000000000000000
query output NodeID 2:  0.000000000000000       1.000000000000000
query output NodeID 3:  1.000000000000000       0.000000000000000
query output NodeID 4:  0.000000000000000       1.000000000000000
query output NodeID 5:  0.000000000000000       1.000000000000000
query output NodeID 6:  0.000000000000000       1.000000000000000
query output NodeID 7:  1.000000000000000       0.000000000000000
query output NodeID 8:  1.000000000000000       0.000000000000000
```

```
[ec2-user@ip-172-31-61-86 testdir]$ vim junction_tree.txt
```

Desktop — ec2-user@ip-172-31-61-86:~/BN_pipeline/testdir — ssh -i big_data.pem ec2-user@ec2-54-204-218-110.compute-1.a...

```
UserID = A0001
NumClq = 4
ClqID = 0
ClqWidth = 3
ClqRange = 2 2 2
POT : size = 8
0       0       0       0.0
0       0       1       0.0
0       1       0       1.0
0       1       1       0.0
1       0       0       0.0
1       0       1       0.0
1       1       0       0.0
1       1       1       0.0
ParentID = -1
SepWidthPa = 0
SepMappingPa =
NumChild = 2
ChildId = 1
SepWidthCh = 1
SepMappingCh = 0
ChildId = 2
SepWidthCh = 1
SepMappingCh = 1
###########
ClqID = 1
ClqWidth = 3
ClqRange = 2 2 2
POT : size = 8
0       0       0       0.0
"junction_tree.txt" 81L, 1013C                          16,1          Top
```

```
###########
ClqID = 2
ClqWidth = 3
ClqRange = 2 2 2
POT : size = 8
0       0       0       0.0
0       0       1       0.0
0       1       0       0.0
0       1       1       0.0
1       0       0       0.0
1       0       1       0.0
1       1       0       0.0
1       1       1       1.0
ParentID = 0
SepWidthPa = 1
SepMappingPa = 2
NumChild = 1
ChildId = 3
```

```
###########
ClqID = 3
ClqWidth = 3
ClqRange = 2 2 2
POT : size = 8
0       0       0       0.0
0       0       1       1.0
0       1       0       0.0
0       1       1       0.0
1       0       0       0.0
1       0       1       0.0
1       1       0       0.0
1       1       1       0.0
ParentID = 2
SepWidthPa = 1
SepMappingPa = 2
NumChild = 0
QueryClq = 0
QueryIdx = 0
###########
```

POT indicates the joint probability of the variables. It means that each variable has two states which are "0" and "1", and the fourth column indicates the different joint probability.