UNIVERSITY OF BIRMINGHAM

College of Engineering and Physical Sciences | School of Computer Science

Full Stack Application Development [06 34252]

YiCS – Software Workshop 2 [06 34169]

Software Workshop 2 [06 34157]

Full Stack Application Development (Dubai) [06 34236]

Software Workshop 2 (Dubai) [06 34188]

Week 9 Lab Exercises

Topic: GraphDB – Link Prediction in Neo4j

Some Data

- Copy and paste the following into an empty Neo4J sandbox.

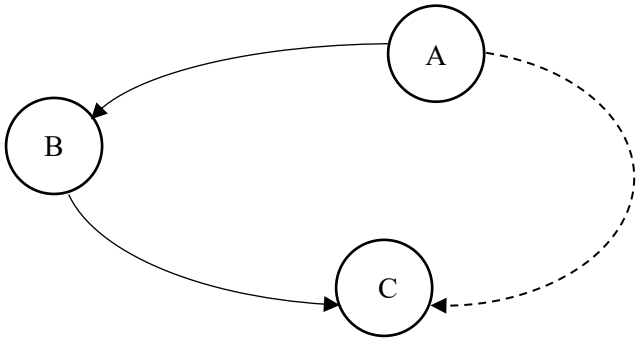
```
CREATE (a:Person {Name: 'Anna'})
CREATE (b:Person {Name: 'Brent'})
CREATE (c:Person {Name: 'Colette'})
CREATE (d:Person {Name: 'Damon'})
CREATE (e:Person {Name: 'Elsa'})
CREATE (f:Person {Name: 'Freddie'})
CREATE (g:Person {Name: 'Gretchen'})
CREATE (h:Person {Name: 'Harry'})
CREATE (i:Person {Name: 'Ingrid'})
CREATE (j:Person {Name: 'Jerome'})
CREATE (k:Person {Name: 'Kate'})
CREATE (l:Person {Name: 'Leon'})
CREATE (m:Person {Name: 'Maddy'})
CREATE (n:Person {Name: 'Niles'})
CREATE (o:Person {Name: 'Orla'})
CREATE (p:Person {Name: 'Padraig'})
CREATE (q:Person {Name: 'Quinlan'})
CREATE (r:Person {Name: 'Rupert'})
CREATE (s:Person {Name: 'Sadie'})
CREATE (t:Person {Name: 'Tyler'})
CREATE (u:Person {Name: 'Uma'})
CREATE (v:Person {Name: 'Victor'})
CREATE (w:Person {Name: 'Wilma'})
CREATE (x:Person {Name: 'Xander'})
CREATE (y:Person {Name: 'Yvette'})
CREATE (z:Person {Name: 'Zane'})

CREATE (a)-[:FRIEND_OF]->(b)
CREATE (b)-[:FRIEND_OF]->(c)
CREATE (a)-[:FRIEND_OF]->(d)
CREATE (d)-[:FRIEND_OF]->(c)
CREATE (b)-[:FRIEND_OF]->(e)
CREATE (e)-[:FRIEND_OF]->(f)
CREATE (f)-[:FRIEND_OF]->(e)
CREATE (e)-[:FRIEND_OF]->(b)
CREATE (b)-[:FRIEND_OF]->(g)
CREATE (g)-[:FRIEND_OF]->(h)
CREATE (h)-[:FRIEND_OF]->(f)
CREATE (g)-[:FRIEND_OF]->(b)
CREATE (h)-[:FRIEND_OF]->(g)
CREATE (f)-[:FRIEND_OF]->(h)
CREATE (e)-[:FRIEND_OF]->(i)
CREATE (i)-[:FRIEND_OF]->(d)
CREATE (j)-[:FRIEND_OF]->(k)
CREATE (k)-[:FRIEND_OF]->(l)
CREATE (l)-[:FRIEND_OF]->(m)
CREATE (m)-[:FRIEND_OF]->(n)
CREATE (n)-[:FRIEND_OF]->(m)
CREATE (m)-[:FRIEND_OF]->(b)
CREATE (b)-[:FRIEND_OF]->(m)
CREATE (l)-[:FRIEND_OF]->(f)
```

- Some nodes are unconnected, add your own FRIEND_OF relationships to the dataset if you wish.

Elements to explore

- Triadic Closure – Simple prediction based on completing the triangle.



If A has a connection to B, and B has a connection to C, then we can PREDICT a connection from A to C

- Use this query to identify the pattern of three nodes connected linearly by two FRIEND_OF relationships. Use the Text output to view all combinations individually

```
MATCH (a)-[:FRIEND_OF]->(b)-[:FRIEND_OF]->(c)
RETURN [a.Name,c.Name]
```

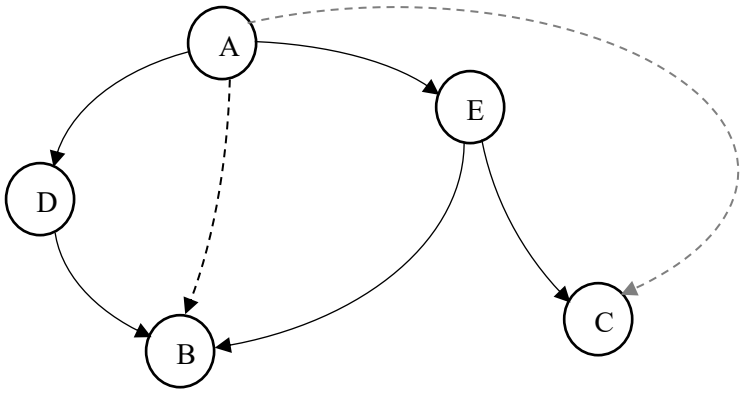
Which unwanted pattern will this also detect?

- We can create the Triadic Closure using the following:

```
MATCH (a)-[:FRIEND_OF]->(b)-[:FRIEND_OF]->(c)
WHERE NOT (a)-[:FRIEND_OF]->(c)
      AND a.Name <> c.Name
CREATE (a)-[:FRIEND_OF]->(c)
```

What two things does the WHERE clause ensure does not happen?

- Extended to Neighbour Count



In this case we are looking for multiple occurrences of paths from one node to another to further reinforce the missing relationship.

A shares 2 Common Neighbours (**D** and **E**) with **B**, **A** only shares 1 (**E**) with **C**. Therefore, a link between **A** and **B** is more likely than **A** and **C**

First, reset your data (see below)

In the following query is a short form of multiple relationship traversals between two nodes.

```
(b)-[:FRIEND_OF*2]->(e)
```

represents the same as:

```
(e)-[:FRIEND_OF]->()-[:FRIEND_OF]->(e)
```

Run:

```
MATCH Path=(b)-[:FRIEND_OF*2]->(e)
RETURN length(Path),nodes(Path)[0].Name,nodes(Path)[2].Name
```

This will list all FRIEND_OF paths of length 2 between your nodes, showing who the intervening friend is.
Again, note there are some paths which we do not want.

```
MATCH Path=(b)-[:FRIEND_OF*2]->(e)
WHERE b.Name<>e.Name
RETURN length(Path),nodes(Path)[0].Name,nodes(Path)[2].Name
```

Rids us of any two step loops that return to the original node.

Run:

```
MATCH Path=(b)-[:FRIEND_OF*2]->(e)
WITH b,e,COUNT(Path) AS Output
WHERE Output > 1
  AND NOT EXISTS ((b)-[:FRIEND_OF]->(e))
  AND b.Name <> e.Name
RETURN b,e,Output
```

Compare the outputs from the last two queries and interpret.

What does the COUNT do?

Inbuilt Link-Prediction Algorithms in neo4j

Adamic Adar

The Adamic Adar algorithm was introduced in 2003 by Lada Adamic and Eytan Adar to predict links in a social network.

```
MATCH (a:Person)
MATCH (b:Person)
WITH a,b, gds.alpha.linkprediction.adamicAdar(a, b) AS score
WHERE score > 0.0
  AND a.Name <> b.Name
RETURN a, b, score
ORDER BY score DESC, a, b
```

Preferential Attachment

Preferential attachment means that the more connected a node is, the more likely it is to receive new links. This algorithm was popularised by Albert-László Barabási and Réka Albert through their work on scale-free networks.

```
MATCH (a:Person)
MATCH (b:Person)
WITH a, b,
     gds.alpha.linkprediction.preferentialAttachment(a, b) AS
score
WHERE score > 0.0
  AND a.Name <> b.Name
RETURN a, b, score
ORDER BY score DESC, a, b
```

Total Neighbours

Total Neighbours computes the closeness of nodes, based on the number of unique neighbours that they have. It is based on the idea that the more connected a node is, the more likely it is to receive new links.

```
MATCH (a:Person)
MATCH (b:Person)
WITH a,b,
     gds.alpha.linkprediction.totalNeighbors(a, b) AS score
```

```
WHERE score > 0.0
  AND a.Name <> b.Name
RETURN a, b, score
ORDER BY score DESC, a, b
```

Others are available, see link on Canvas

Resetting the data

Use the following two commands to remove all relationships, then all nodes, then re-enter the data if you make a mistake or wish to do something different

```
MATCH ()-[r:FRIEND_OF]->() DELETE r
```

```
MATCH (n:Person) DELETE n
```

If you still have the Person Node connected from sheet 1 remove that also

```
MATCH p=()-[r:Drives]->() DELETE p
```