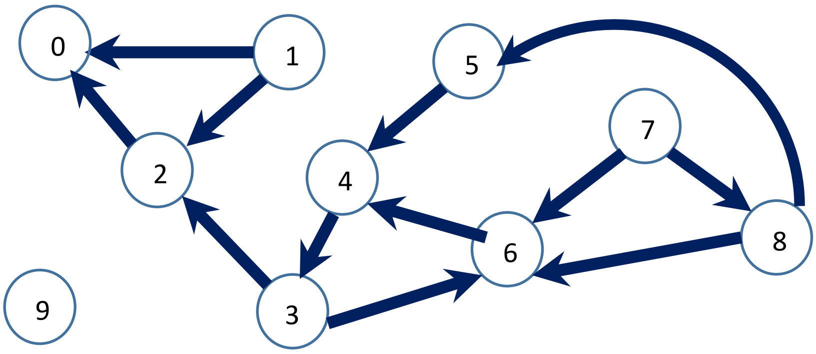
Activity 2

Graphs in Neo4J: Simple queries

Exercise 1: We will express queries over graph databases using Cypher, the high-level query language for Neo4J. In order to check the correctness of the result sets, we will start with a small graph representing a subset of the web structure, as shown below:



The **minigraphweb database** contains this information and will be used in this activity. Check your answers.

Use Case 1: Vertex Degree

For each vertex compute its in-degree and out-degree. The result set is a list of vertices and both values. Those vertices that do not have outgoing edges and/or incoming edges, must not appear in the answer. (Note that the result set is not a graph)

Answer:

MATCH (src)-->(n)-->(target)

RETURN n.name, COUNT(distinct src) as indegree, COUNT (distinct target) as outdegree

ORDER BY n.name

Alternative solution:

MATCH (src)-->(n)-->(target)

MATCH ()-[r1]->(n)-[r2]->()

RETURN n.name, COUNT(distinct r1) as indegree, COUNT (distinct r2) as outdegree

ORDER BY n.name

Use Case 2: Vertex Degree variation

For each vertex calculate its in-degree and out-degree. The result set is a list **of all vertices in the graph** and both values. Those vertices that do not have outgoing edges and/or incoming edges, **must appear** in the answer with value "0".

Answer

MATCH (n)

OPTIONAL MATCH ()-[r1]->(n)

OPTIONAL MATCH (n)-[r2]->()

RETURN n.name, COUNT(distinct r1) as indegree, COUNT (distinct r2) as outdegree

ORDER BY n.name

Use Case 3: Calculating a maximum value

Find the maximum vertex in-degree. (the resultset is not a graph)

Answer

MATCH (n)

OPTIONAL MATCH (src)-->(n)

WITH n, COUNT(distinct src) as indegree

RETURN MAX(indegree)

Use Case 4: Find influential nodes

Find the subgraph which contains nodes whose in-degree is maximal in the graph (you should obtain **only one node**). Try it also with the out-degree (you should obtain **four nodes**).

Answer

For indegree:

Traversing the graph twice

MATCH (n)

OPTIONAL MATCH (n)<--(src)

WITH n.name, COUNT(distinct src) as indegree

WITH MAX(indegree) AS max

MATCH (nn)

OPTIONAL MATCH (nn)<--(src)

```
WITH nn, COUNT(distinct src) as valor, max

WHERE valor = max

RETURN nn.name, valor
```

Alternative, traversing the graph only once.

```
MATCH (n)

OPTIONAL MATCH  (src)-->(n)

WITH n, COUNT(distinct src) as indegree

WITH COLLECT ( [n.name,  indegree]) as tuples, MAX(indegree) as max

RETURN  [e  in tuples WHERE e[1]= max |e]
```

Use Case 5: Distance between nodes

For each pair of vertices, calculate the distance, i.e. the shortest simple path between them (without repeated edges in the path). Do not show the distance between two disconnected nodes (infinite distance). Exclude paths when source and target are the same node.

Answer

```
MATCH path= (n)-[*]->(p)

WHERE n <> p

RETURN n.name, p.name, min( length(path) ) as distance

ORDER BY n.name, p.name
```

Use Case 6: Distance between nodes using Cypher function

Solve the query in Use Case 5, but using the **shortestPath** built-in Cypher function**.** It has one parameter that represents a pattern path and returns the shortest path that matches this pattern. If there exists more than one shortest path, it returns any of them. For example, shortestPath((n1) –[*1..2] -> (n2)) returns a shortest path that connects n1 and n2, with distance 1 or 2.

Answer

```
MATCH path= shortestPath((n)-[*]->(p) )

WHERE n <> p

WITH nodes(path) as listanodos

RETURN [n in listanodos|n.name]
```

Use Case 7: Diameter

Compute the diameter of the graph, i.e. the longest distance between two nodes in the graph (excluding disconnected pairs of nodes).

Answer

```
MATCH p=(n1)-[*1..]->(n2)

WHERE n1 <> n2

WITH n1, n2, MIN(length(p)) as distance

RETURN MAX(distance)
```

Alternative

```
MATCH p=(n1)-[*1..]->(n2)

WHERE n1 <> n2

WITH n1.name, n2.name, MIN(length(p)) as distance

RETURN distance

ORDER BY distance DESC

LIMIT 1
```

Use Case 8: webgraph3.db.

Repeat use cases 1 to 7 using the webgraph3.db, which represents the same information as the relational databases in Activity 1.

Use Case 9: Paths

Compute all the n-hops in a graph, and compare against the results obtained using PostgreSQL in Activity 1. The result must have the form: start, length, path, end.

```
MATCH path = (end)<-[*1..]-(start)

WHERE ALL (n in nodes(path) where

        1 = size([m in nodes(path) where m=n]))

RETURN start.name, LENGTH(path) AS length, [p in NODES(path) | p.name], end.name

ORDER BY length
```

```
MATCH path = (end)<-[*1..]-(start:URL{name:745315})
WHERE ALL (n in nodes(path) where
        1 = size([m in nodes(path) where m=n]))
RETURN LENGTH(path) AS length, [p in NODES(path) | p.name]
ORDER BY length desc
```