Path Analytics in Neo4j With Cypher - Supplementary Resources | Coursera

Path Analytics with CYPHER//Viewing the graph

```
match (n:MyNode)-[r]->(m)

return n, r, m
```

//Finding paths between specific nodes*:

```
match p=(a)-[:TO*]-(c)

where a.Name='H' and c.Name='P'

return p limit 1
```

*Your results might not be the same as the video hands-on demo. If not, try the following query and it should return the shortest path between nodes H and P:

```
match p=(a)-[:TO*]-(c) where a.Name='H' and c.Name='P' return p order by length(p) asc limit 1
```

//Finding the length between specific nodes:

```
match p=(a)-[:TO*]-(c)

where a.Name='H' and c.Name='P'

return length(p) limit 1
```

//Finding a shortest path between specific nodes:

```
match p=shortestPath((a)-[:TO*]-(c))

where a.Name='A' and c.Name='P'

return p, length(p) limit 1
```

//All Shortest Paths:

```
MATCH p = allShortestPaths(source-[r:TO*]-destination)

WHERE source.Name='A' AND destination.Name = 'P'

RETURN EXTRACT(n IN NODES(p)| n.Name) AS Paths
```

//All Shortest Paths with Path Conditions:

```
MATCH p = allShortestPaths(source-[r:TO*]->destination)

WHERE source.Name='A' AND destination.Name = 'P' AND LENGTH(NODES(p)) > 5

RETURN EXTRACT(n IN NODES(p)l n.Name) AS Paths,length(p)
```

//Diameter of the graph:

```
match (n:MyNode), (m:MyNode)

where n <> m

with n, m

match p=shortestPath((n)-[*]->(m))

return n.Name, m.Name, length(p)

order by length(p) desc limit 1
```

//Extracting and computing with node and properties:

```
match p=(a)-[:TO*]-(c)

where a.Name='H' and c.Name='P'

return extract(n in nodes(p)ln.Name) as Nodes, length(p) as pathLength,

reduce(s=0, e in relationships(p)l s + toInt(e.dist)) as pathDist limit 1
```

//Dijkstra's algorithm for a specific target node:

```
MATCH (from: MyNode {Name:'A'}), (to: MyNode {Name:'P'}),

path = shortestPath((from)-[:TO*]->(to))

WITH REDUCE(dist = 0, rel in rels(path) l dist + toInt(rel.dist)) AS distance, path

RETURN path, distance
```

//Dijkstra's algorithm SSSP:

```
MATCH (from: MyNode {Name:'A'}), (to: MyNode),

path = shortestPath((from)-[:TO*]->(to))

WITH REDUCE(dist = 0, rel in rels(path) l dist + toInt(rel.dist)) AS distance, path, from, to

RETURN from, to, path, distance order by distance desc
```

//Graph not containing a selected node:

```
match (n)-[r:TO]->(m)

where n.Name <> 'D' and m.Name <> 'D'

return n, r, m
```

//Shortest path over a Graph not containing a selected node:

```
match p=shortestPath((a {Name: 'A'})-[:TO*]-(b {Name: 'P'}))

where not('D' in (extract(n in nodes(p)|n.Name)))

return p, length(p)
```

//Graph not containing the immediate neighborhood of a specified node:

```
match (d {Name:'D'})-[:TO]-(b)

with collect(distinct b.Name) as neighbors

match (n)-[r:TO]->(m)

where

not (n.Name in (neighbors+'D'))

and

not (m.Name in (neighbors+'D'))

return n, r, m

;

match (d {Name:'D'})-[:TO]-(b)-[:TO]->(leaf)

where not((leaf)-->())

return (leaf)

;

match (d {Name:'D'})-[:TO]-(b)<-[:TO]-(root)

where not((root)<--())

return (root)
```

//Graph not containing a selected neighborhood:

```
match (a {Name: 'F'})-[:TO*..2]-(b)

with collect(distinct b.Name) as MyList

match (n)-[r:TO]->(m)

where not(n.Name in MyList) and not (m.Name in MyList)

return distinct n, r, m
```