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 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 (source {Name: 'A'}), (destination {Name: 'P'}),
     path = allShortestPaths((source)-[:TO*]->(destination))
RETURN [node IN NODES(path) | node.Name] AS Paths

//All Shortest Paths with Path Conditions:
MATCH (source {Name: 'A'}), (destination {Name: 'P'}),
     p = allShortestPaths((source)-[:TO*]->(destination))
WHERE length(p) > 5
RETURN [n IN nodes(p) | 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 [n IN nodes(p) | n.Name] AS Nodes, length(p) AS PathLength,
REDUCE(s = 0, e IN relationships(p) | s + toInteger(e.dist)) AS PathDist
ORDER BY PathDist asc 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) | dist + toInt(rel.dist)) AS distance, path
RETURN path, distance

//Dijkstra's algorithm SSSP:
MATCH (from: MyNode {Name:'A'}), (to: MyNode)
WHERE from <> to
MATCH path = shortestPath((from)-[:TO*]->(to))
WITH from, to, path, REDUCE(dist = 0, rel in relationships(path) | dist + toInteger(rel.dist)) AS
distance
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 [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
```

```
//Graph not containing the immediate neighborhood of a specified node (include leaf nodes)
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'))
MATCH (d {Name:'D'})-[:TO]-(b)-[:TO]->(leaf)
WHERE NOT((leaf)-->())
RETURN (leaf), n,r,m

//Graph not containing the immediate neighborhood of a specified node (include root nodes)
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'))
MATCH (d {Name:'D'})-[:TO]-(b)<-[:TO]-(root)
WHERE NOT((root)<--())
RETURN (root), n,r,m


//Graph not containing the immediate neighborhood of a specified node (full version):
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
```