Analyse de Chemins dans un graphe avec Cypher

//Visualisation du graphe

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

//Nombre de noeuds

```
match (n:MyNode)
return count(n)
```

//Nombre d'arcs

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

//Recherche des feuilles (nœuds qui ne sont origine d'aucun arc) :

```
match (n:MyNode)-[r:TO]->(m)
where not ((m)-->())
return m
```

// Recherche des racines (noeuds qui ne sont destination d'aucun arc) :

```
match (m)-[r:TO]->(n:MyNode)
where not (()-->(m))
return m
```

//rechercher les nœuds reliés en triangles :

```
match (a)-[:TO]->(b)-[:TO]->(c)-[:TO]->(a)
return distinct a, b, c
```

//Rechercher les voisins d'ordre 2 (2ème niveau) d'un nœud de nom D :

```
match (a)-[:TO*..2]-(b)
where a.Name='D'
return distinct a, b
```

//Trouver les boucles (liens réflexifs) dans un graphe :

```
match (n)-[r]->(n)
return n, r limit 10
```

//Trouver les liens multi-graphes (2 liens différents entre les 2 mêmes nœuds) :

```
match (n)-[r1]->(m), (n)-[r2]-(m)
where r1 <> r2
return n, r1, r2, m limit 10
```

//Trouvez le graphe induit par un ensemble de nœuds donnés :

```
match (n)-[r:TO]-(m)
where n.Name in ['A', 'B', 'C', 'D', 'E'] and m.Name in ['A', 'B', 'C', 'D', 'E']
return n, r, m
```

//Recherche d'un chemin entre deux nœuds spécifiques (quelque soit l'orientation) :

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

//Recherche du plus court chemin entre deux nœuds spécifiques (solution naïve) :

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

//Retour de la longueur entre deux nœuds :

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

// Recherche du plus court chemin entre deux nœuds spécifiques (solution optimisée) :

```
match p=shortestPath((a)-[:TO*]-(c))
where a.Name='A' and c.Name='P'
return p, length(p) limit 1
```

// Recherche de tous les plus courts chemins entre deux nœuds :

```
MATCH p = allShortestPaths((source)-[r:TO*]-(destination))
WHERE source.Name='A' AND destination.Name = 'P'
RETURN p
```

// Recherche les plus courts chemins sous condition :

```
MATCH p = allShortestPaths((source)-[r:TO*]->(destination))
WHERE source.Name='A' AND destination.Name = 'P' AND LENGTH(NODES(p)) > 5
RETURN p, length(p)
```

//Clacul du diamètre d'un graphe :

```
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
```

//Algorithme de Dijkstra de calcul d'un plus court Chemin entre A et P :

```
MATCH (from: MyNode {Name:'A'}), (to: MyNode {Name:'P'}),
path = shortestPath((from)-[:TO*]->(to))
WITH REDUCE(dist = 0, rel in rels(path) | dist + toInteger(rel.dist)) AS distance, path
RETURN path, distance
```

//Algorithme de Dijkstra de calcul des plus courts Chemins partant du nœud A :

```
MATCH (from: MyNode {Name:'A'}), (to: MyNode),
path = shortestPath((from)-[:TO*]->(to))
WITH REDUCE(dist = 0, rel in rels(path) | dist + toInteger(rel.dist)) AS distance, path, from, to
RETURN from, to, path, distance order by distance desc
```

//Graphe sans le nœud D :

```
match (n)-[r:TO]->(m)
where n.Name <> 'D' and m.Name <> 'D'
return n, r, m
```

//Chemins les plus courts ne passant pas par un nœud donné (D) :

```
match p=shortestPath((a {Name: 'A'})-[:TO*]-(b {Name: 'P'}))
where not('D' in [n in nodes(p)| n.Name])
return p, length(p)
```

// Analyse de connectivité du graphe

// Degrés sortants des noeuds

```
match (n:MyNode)-[r]->()
return n.Name as Node, count(r) as Outdegree
order by Outdegree
union
match (a:MyNode)-[r]->(leaf)
where not((leaf)-->())
return leaf.Name as Node, 0 as Outdegree
```

// Degrés entrants des noeuds

```
match (n:MyNode)<-[r]-()
```

```
return n.Name as Node, count(r) as Indegree
order by Indegree
union
match (a:MyNode)<-[r]-(root)
where not((root)<--())
return root.Name as Node, 0 as Indegree

// Degrés des nœuds sans distinction du sens
match (n:MyNode)-[r]-()
return n.Name, count(distinct r) as degree
order by degree

//Degré rajouté comme propriété du nœud
match (n:MyNode)-[r]-()
with n, count(distinct r) as degree
set n.deg = degree
return n.Name, n.deg

// Histogramme des degrés du graphe
match (n:MyNode)-[r]-()
with n as nodes, count(distinct r) as degree
return degree, count(nodes) order by degree asc

// Construction de la matrice d'adjacence du graphe
match (n:MyNode), (m:MyNode)
return n.Name, m.Name,
case
when (n)-->(m) then 1
else 0
end as value
```