Teoría de Grafos para Big Data

Neo4j

Instrucciones

Esta guía tiene consultas para aprender lo básico de Cypher y Neo4J. Lea y ejecute cada una de las consultas, preocupándose de entender qué hace cada parte de ella.
Para esta actividad, trabajaremos con el sandbox de Neo4j, ubicado en

`https://neo4j.com/sandbox-v2/`

Tras registrarnos y crear una cuenta, podemos acceder a una instancia de Neo4J con un grafo cargado haciendo click en "Launch Sandbox". Inicialmente usaremos el sandbox "Recommendations", que consiste en una base de datos de películas, actores y directores, junto con evaluaciones de usuarios.

Para ver el grafo, en cualquier momento podemos ejecutar la siguiente consulta:

```
MATCH (n) RETURN n LIMIT 100
```

y comprobar que estamos en el grafo correcto.

IMPORTANTE: Es recomendable siempre agregar `LIMIT` al final, ya que puede ocurrir que la pestaña del navegador se quede pegada al tratar de dibujar grafos muy grandes.
MÁS IMPORTANTE: En caso de accidentes o problemas, se puede reiniciar el *sandbox* a su estado original. Para esto basta con volver al panel de los sandboxes[1] y en el cuadro de "Recommendations" hacer click en Advanced → Shutdown Sandbox. Luego es cosa de iniciarlo una vez más y estará en su estado original.

El modelo

Neo4J trabaja con el modelo de *Property Graphs*. En este modelo tendremos nodos conectados entre si por aristas. Los **nodos** tienen un tipo y pueden tener varias propiedades. Las propiedades las podemos pensar como pares key-value. Por ejemplo, si tenemos un nodo de tipo **Persona**, este puede contener los siguientes pares key-value:

```
{'nombre': 'Kevin Bacon', 'edad': 60}
```

[1] `https://neo4j.com/sandbox-v2/`

Esto indica que aquel nodo posee las propiedades `'nombre'` y `'edad'`, con los valores "Kevin Bacon" y 60, respectivamente.
Además, existen **aristas** que conectan los nodos entre si. Estas aristas tienen una etiqueta. Por ejemplo, dos nodos persona `a` y `b` pueden estar conectados por la arista con etiqueta `sigue` que va desde `a` hacia `b` si la persona representada por el nodo `a` sigue a la persona representada por el nodo `b` (en una red social, por ejemplo). Todas las aristas son dirigidas. Además, las aristas pueden tener propiedades.

Creando nodos y aristas

Para crear un nodo en Neo4J, podemos realizar la siguiente consulta:

```
CREATE (n:Person { name: 'Fernando Pieressa', role: 'Ayudante proyecto' })
```

Antes de crear una arista, vamos a crear un segundo nodo:

```
CREATE (n:Person { name: 'Stephanie Chau', role: 'Ayudante proyecto' })
```

Observación: En ambos casos, `n` actúa como una variable.

Para crear una arista entre ellos:

```
MATCH (a:Person),(b:Person)
WHERE a.name = 'Stephanie Chau' AND b.name = 'Fernando Pieressa'
CREATE (a)-[r:FOLLOWS]->(b)
RETURN type(r)
```

En la consulta anterior estamos capturando todos los nodos `a`, `b` (`a` y `b` son variables) que sean de tipo persona en que el nombre del primer nodo sea "Stephanie Chau" y del segundo sea "Fernando Pieressa". Luego estamos creando una arista que sale desde ese primer nodo y va al segundo, que es de tipo `FOLLOWS`.

El lenguaje de consultas Cypher

El lenguaje de consultas se basa en encontrar patrones dentro del grafo. Por ejemplo la siguiente consulta:

```
MATCH (a:Person {name: 'Stephanie Chau'}) RETURN a
```

Busca todos los nodos de tipo persona cuyo nombre sea "Stephanie Chau". Esos nodos son mapeados a la variable `a`. Luego retorno todos los nodos `a` que cumplen con aquella condición. Ahora, ¿Puedes suponer lo que hace la consulta a continuación?

```
MATCH (a:Person {name: 'Stephanie Chau'})-[r]->(b) RETURN a, r, b
```

Si te fijas bien, estamos seleccionando en la variable `a` todos los nodos con el nombre "Stephanie Chau". Luego para esos nodos, queremos obtener todas sus aristas salientes (fijate

en la dirección de la "flecha" en la consulta) y los nodos hacia donde llegan esas aristas. Finalmente para cada posible mapeo en el grafo, retornamos las variables `a`, `r` y `b`.

En general, los patrones básicos de Cypher son:

- `(p) --> (q)` y `(p) <-- (q)`, que representan una arista que va de `p` a `q` o de `q` a `p`, respectivamente, y `(p) --> ()`, que representa una arista de `p` a *algún* nodo.
- `(p) -[n:ETIQUETA]-> (q)`, que representa una arista `n` que va de de `p` a `q`, y que además la arista está etiquetada con `ETIQUETA`.
- Estos caminos se pueden combinar, por ejemplo `(p) <-- (q) --> (r)` o incorporando una etiqueta: `(p) <-[n:ETIQUETA]- (q) <-- (r)`
- Para retornar usamos `RETURN`. Podemos retornar todo o parte de todo lo que hemos nombrado en nuestra consulta `(p)`, `(q)`, etc. Es posible usar `*` en return para que lo retorne todo.
- Intenta distintas combinaciones en el grafo de películas. También pruebe agregando restricciones a los atributos de los nodos y las aristas (nombres de actores, nombres de películas). Además, puedes quitar la dirección de las aristas, para que no importe hacia donde van.
- Pruebe con `MATCH c= (p) -[n]-> (q) RETURN *`. ¿Que es `c` en este caso?

Puedes ver más en la documentación oficial de Neo4J:

`http://neo4j.com/docs/developer-manual/current/cypher/`

1. Número de Bacon

Esta parte de la guía usa el sandbox "Recommendations" de Neo4J.

1.1. Explorando el grafo

Escribe las siguientes consultas y observa el resultado:

1. Todas las películas en las que ha actuado Kevin Bacon.

   ```
   MATCH (:Actor {name: "Kevin Bacon"})-[:ACTED_IN]->(m:Movie)
   RETURN *
   ```

2. Todas las películas en las que ha actuado Kevin Bacon, pero ahora usando `WHERE`.

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)
WHERE a.name = "Kevin Bacon"
RETURN *
```

3. Los actores que han actuado en una película con Kevin Bacon.

```
MATCH (b:Actor {name: "Kevin Bacon"})-->(m:Movie)<--(actor:Actor)
RETURN *
```

4. Los actores que han actuado en al menos dos películas con Kevin Bacon.

```
MATCH (b:Actor {name: "Kevin Bacon"})-->(m1:Movie)<--(a:Actor)
WITH b, m1, a
MATCH (b)-->(m2:Movie)<--(a)
WHERE m1<>m2
RETURN *
```

   ¿Por qué esta consulta no entrega resultados? Reemplaza a Kevin Bacon por Tom Hanks y ve qué ocurre.

5. Los pares de actores que han actuado juntos y con Kevin Bacon.

```
MATCH (b:Actor {name: "Kevin Bacon"})-->(m1:Movie)<--(a1:Actor)
WITH b, a1
MATCH (b)-->(m2:Movie)<--(a2:Actor)-->(m3:Movie)<--(a1)
RETURN *
```

1.2. Agregación

Las siguientes consultas tienen que ver con las funciones de agregación de Neo4J:

1. La cantidad de nodos en el grafo.

```
MATCH (n) RETURN COUNT(n)
```

2. La cantidad de películas en las que actúa Kevin Bacon.

```
MATCH (:Actor {name: "Kevin Bacon"})-->(m:Movie)
RETURN COUNT(m)
```

3. La cantidad promedio de películas por actor.

```
MATCH (a:Actor)-->(m:Movie)
WITH a, COUNT(m) as c
RETURN AVG(c)
```

4. La lista de los actores, ordenada por quien más ha actuado.

   ```
   MATCH (a:Actor)-->(m:Movie)
   RETURN a, COUNT(m) as c ORDER BY c DESC
   ```

1.3. Caminos y número de Bacon

Definimos el número de Bacon como sigue:

- Kevin Bacon tiene número de Bacon 0.
- Si un actor aparece en una película con alguien que tiene número de Bacon n, entonces esa persona tiene número de Bacon $n + 1$, a menos que ya tenga un número de Bacon menor.
- Todos los actores que no son alcanzables de esta manera desde Kevin Bacon tienen número de Bacon infinito.

Ejecute las siguientes consultas:

1. Todos los nodos conectados entre sí

   ```
   MATCH (a:Actor)-[:ACTED_IN]-(m:Movie)-[:ACTED_IN]-(b:Actor)
   WHERE a<>b
   RETURN * LIMIT 100
   ```

   Como dijimos anteriormente, el `LIMIT` lo utilizamos para evitar que el navegador colapse. En esta consulta el `WHERE` señala que los nodos deben ser distintos. ¿Por qué es necesario esto?

2. Todos los nodos que estan contectados a Kevin Bacon por un camino de cualquier largo

   Un primer acercamiento es la siguiente consulta:

   ```
   MATCH (:Actor {name: "Kevin Bacon"})-[:ACTED_IN*]-(a:Actor)
   RETURN a
   ```

   Aquí, estamos buscando a todos los nodos que estén conectado por cualquier número de aristas del tipo `ACTED_IN`. Sin embargo, rápidamente vemos que el sistema colapsa. Podemos ahora intentar limitar el largo del camino:

   ```
   MATCH (:Actor {name: "Kevin Bacon"})-[:ACTED_IN*..5]-(a:Actor)
   RETURN a
   ```

   Con esto conseguimos algunos nodos sin que el sistema colapse, pero con algo de ojo podemos notar que empiezan a aparecer varios caminos entre pares de nodos.

   Esta consulta busca *todos* los caminos entre Kevin Bacon y todos los nodosa los que está conectado, lo que obviamente puede generar caminos arbitrariamente largos. Como nos interesa solamente saber si existe un camino entre Kevin Bacon y el otro nodo, podemos preguntar solamente por el camino más corto:

```
MATCH p=shortestPath((b:Actor {name: "Kevin Bacon"})-[:ACTED_IN*]-(a))
RETURN p LIMIT 1000
```

3. Para un actor determinado (por ejemplo Tom Hanks), verifique si existe un camino entre Kevin Bacon y él, y entre él y Kevin Bacon.

   ```
   MATCH p=shortestPath((a:Actor)-[:ACTED_IN*]-(b:Actor))
   WHERE a.name = "Kevin Bacon" and b.name="Tom Hanks"
   RETURN p
   ```

4. El actor con más caminos a Kevin Bacon.

   La siguiente consulta representa la respuesta que buscamos (**¡no ejecutes esta consulta!**):

   ```
   MATCH p=((a:Actor)-[:ACTED_IN*]-(:Actor {name: "Kevin Bacon"}))
   RETURN a, COUNT(*) as c order by c desc limit 1
   ```

   Sin embargo, la consulta se quedará pegada sin importar de cuánta memoria y tiempo dispongamos. ¿Qué está pasando? ¿En qué tipos de grafos esta consulta se puede responder?

1.4. Modificando el grafo

Ahora vamos a agregar aristas al grafo. En caso de accidentes o problemas, recuerda que puedes reiniciar el *sandbox*.

1. Para todo par de actores que han aparecido en la misma película, agregue en ambas direcciones un arco con etiqueta `ACTUA_CON`.

   ```
   MATCH (b:Actor)-->(m:Movie)<--(a:Actor)
   CREATE (a)-[:ACTUA_CON]->(b)
   ```

2. Para el nodo "Kevin Bacon", fije el atributo `numero_bacon` en 0. Para esto la sintaxis es igual a `CREATE`, pero usando `SET` en vez de `CREATE`.

   ```
   MATCH (b:Actor {name: "Kevin Bacon"}) SET b.numero_bacon=0
   ```

3. Escribe una consulta que al ejecutarla, encuentre los nodos sin número de Bacon (`WHERE NOT EXISTS(n.numero_bacon)`) que están conectados a un nodo con número de Bacon finito, y les cree su número de Bacon.

   ```
   MATCH (a:Actor)-[:ACTUA_CON]-(b:Actor)
   WHERE EXISTS(a.numero_bacon) and NOT EXISTS(b.numero_bacon)
   WITH b, min(a.numero_bacon) as m
   SET b.numero_bacon=m+1
   RETURN *
   ```

Ojo con la tercera línea. Esa línea hace que siempre se fije el número de Bacon de un nodo a partir del vecino con el número de bacon más bajo. Si no usamos esa línea y solamente fijamos el número de bacon del nodo a partir del número de Bacon de un vecino cualquiera, podemos estar eligiendo un camino más largo y fijando el número en un valor mayor al real.

4. Ejecute la consulta anterior 5 veces. ¿Cuáles son los nodos con número de Bacon menor o igual a 5? ¿Qué pasa con los otros nodos?

   ¿A qué algoritmo o tipo de consulta vista en clases se parece este procedimiento?

5. Crea ahora una consulta que automáticamente llene el número de Bacon de todos los nodos que son alcanzables desde Kevin Bacon, usando el largo del camino más corto desde Kevin Bacon. Para esto usaremos la función `length`.

   ```
   MATCH
   p=shortestPath((b:Actor {name: "Kevin Bacon"})-[:ACTUA_CON*]-(a:Actor))
   SET a.numero_bacon = length(p)
   RETURN a, length(p)
   ```

6. Encuentra el número de actores que NO son alcanzables desde Kevin Bacon.

   Como fijamos el número de bacon de todos los actores alcanzables desde Kevin Bacon en el paso anterior, los actores no alcanzables son aquellos que no tienen número de bacon:

   ```
   MATCH (n:Actor) WHERE NOT EXISTS(n.numero_bacon)
   RETURN n
   ```