TICS575 - Técnicas para Big Data

GDS

Instrucciones

En esta guía vamos a ver cómo ejecutar procedimientos de **Graph Analytics** con Neo4J. Esto gracias a un módulo estrenado el año 2020, llamado *Graph Data Science*, que permite ejecutar algunos de los procedimientos más populares. Para esta actividad, trabajaremos con el sandbox de Neo4j, ubicado en

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

Tras registrarnos y crear una cuenta, podemos acceder a una instancia de Neo4Jcon un grafo cargado haciendo click en "Launch Sandbox". Inicialmente usaremos el sandbox "Graph Data Science", en donde el grafo se basa en las interacciones de la saga de "Game of Thrones".

Esta guía se basa en el tutorial oficial del módulo GDS de Neo4J.

1. Creando grafos

Para partir, podemos hacer un resumen de todo el grafo con el siguiente comando:

```
CALL db.schema.visualization()
```

El resultado muestra las interacciones entre los distintos tipos de nodos. Los tipos `:Dead`, `:King` y `:Knight` son también de tipo persona, por lo que si los ocultas en la visualización, podrás entender mejor los datos. Veamos que en los tipos de datos que más destacan son `:Person`, `:Book`, `:Battle`, `:House` y `:Region`. Es importante ver además que los nodos de tipo `:Person` tiene varios tipoos de aristas que se relacionan con otros nodos de tipo `:Person`.

Ahora para seguir, debemos crear grafos para el catálogo del módulo GDS. Esto es, extraer los nodos y relaciones que nos interesa para ejecutar un procedimiento, y guardarlo como un grafo **en memoria**. Esto es importante, ya que para lograr obtener respuestas rápido, los algoritmos van a correr en memoria principal. Ahora vamos a crear el siguiente grafo:

```
CALL gds.graph.create('got-interactions', 'Person', {
  INTERACTS: {
    orientation: 'UNDIRECTED'
  }
})
```

Esto genera un subgrafo de todos los nodos de tipo `:Person` conectados mediante aristas de tipo `:INTERACTS`. Como la relación es simétrica, vamos a ignorar la dirección de la arista. Puedes ver los detalles de un grafo creado con el comando:

```
CALL gds.graph.list('got-interactions')
```

O bien listar el nombre de todos los grafos con:

```
CALL gds.graph.list()
```

También puedes eliminar un grafo creado con el comando:

```
CALL gds.graph.drop('got-interactions')
```

Ojo: no elimines el grafo pues lo vamos a utilizar.

2. Algoritmos de Graph Analytics

En general, todos los algoritmos se van a ver de la forma:

```
CALL gds.<algo-name>.<mode>(
  graphName: String,
  configuration: Map
)
```

Vamos a partir por el algoritmo de PageRank:

2.1. PageRank

Para computar PageRank, usamos el siguiente procedimiento:

```
CALL gds.pageRank.stream('got-interactions') YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC LIMIT 10
```

Donde estamos tomando el grafo creado anteriormente, ejecutamos el algoritmo de PageRank, y retornamos el nombre de cada nodo, junto a su valor de PageRank.

2.2. Weakly Connected Components

Para correr el algoritmo de **WCC** usamos el mismo grafo creado. El algoritmo se corre con el siguiente procedimiento:

```
CALL gds.wcc.stream('got-interactions')
YIELD nodeId, componentId
```

```
RETURN componentId AS component, count(nodeId) AS size
ORDER BY size DESC
```

Que nos entrega las distintas componentes junto a su tamaño. En este caso tenemos una componente que acumula 795 nodos, y varios nodos aislados. Si queremos descubrir a qué componente pertenece cada nodo, corremos:

```
CALL gds.pageRank.stream('got-interactions') YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC LIMIT 10
```

2.3. Triangle Count

Para continuar, vamos a agregar un nuevo grafo al catálogo:

```
CALL gds.graph.create(
  'got-interactions-1',
  'Person',
  {
    INTERACTS_1: {
      orientation: 'UNDIRECTED'
    }
  }
);
```

Que son las interacciones del primer libro de la saga (esto se ve con la propiedad `:INTERACTS_1`). Luego con el siguiente procedimiento podemos ver el número de triángulos en los que participa cada personaje:

```
CALL gds.triangleCount.stream('got-interactions-1')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC
LIMIT 10
```

2.4. Local Clustering Coefficient

Finalmente, para calcular LCC, podemos correr el siguiente procedimiento sobre el grafo agregado en el punto anterior:

```
CALL gds.localClusteringCoefficient.stream('got-interactions-1')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC
LIMIT 10
```