ESCUELA DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería Informática

DESARROLLO DE APLICACIONES WEB II

CURSO 2021/2022

MEMORIA DE LA PRÁCTICA

Análisis de Información

AUTOR: Kiliam David Martín García

FECHA: 2/03/2022

ÍNDICE

1. Introducción 1

2. Desarrollo 1

2.1. Fichero *'set-dataset.cypher'* 1

2.2. Fichero '*create-graph.cypher*' 2

2.3. Fichero '*run-dijkstra-algorithm.cypher*' 3

3. Conclusiones 3

4. Bibliografía 4

1. Introducción

Este trabajo práctico consistirá en el análisis de una base de datos basada en grafos (creada en *Neo4j*) haciendo uso de uno de los algoritmos que podemos encontrar en el manual de la librería *Neo4j Graph Data Science* [1]. De forma concreta los de los grupos: Centralidad, Detección de Comunidades, Similaridad o Búsqueda de Rutas.

2. Desarrollo

En este caso, aplicaremos algoritmo Origen-Destino de *Dijkstra [2]* en un grafo el cual representa carreteras entre distintas ciudades españolas, concretamente el que podemos observar en la Figura 1

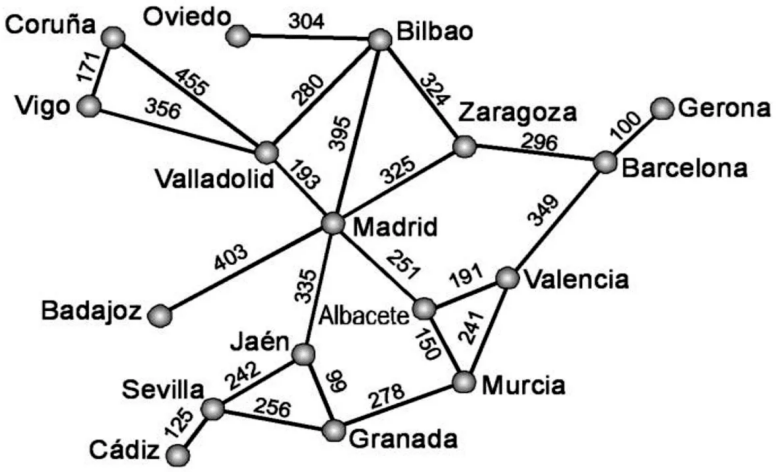

Figura 1. Grafo de carreteras entre ciudades de España

Para llevar a cabo el análisis, necesitaremos definir la base de datos y posteriormente para poder ejecutar el algoritmo de la librería (*GDS – Graph Data Science)* necesitaremos crear un grafo. Usaremos el contenido de los ficheros: '*set-dataset.cypher', 'create-graph.cypher', 'run-dijkstra-algorithm.cypher',* en ese orden para realizar las tareas descritas anteriormente

2.1. Fichero *'set-dataset.cypher'*

Este fichero servirá para crear la base de datos. Para ello, se cargarán los datos de dos ficheros: '*cities.csv'* y *'distances.csv'*. El primero, servirá para definir las ciudades y el segundo para definir la distancia que hay entre ambas. Podemos ver el contenido del fichero '*set-dataset.cypher*', '*cities.csv'* y *'distances.csv'*, en las Figuras 2 , 3 y 4 respectivamente:

```
LOAD CSV FROM "file:///cities.csv"
AS line
CREATE (:City { id: toInteger(line[0]), name: line[1] })

LOAD CSV WITH HEADERS FROM "file:///distances.csv"
AS line
MATCH (c1:City { id: toInteger(line.FromCity) }),
  (c2:City { id: toInteger(line.ToCity) })
CREATE path = (c1) -[:ROAD {name: 'ROAD-' + line.Id, distance: toInteger(line.DistanceKM) }]-> (c2)
```

Figura 2. Contenido del fichero '*set-dataset.cypher'*

```
1,Madrid
2,Zaragoza
3,Badajoz
4,Valladolid
5,Vigo
6,Coruña
7,Oviedo
8,Bilbao
9,Gerona
10,Barcelona
11,Valencia
12,Murcia
13,Albacete
14,Granada
15,Jaén
16,Sevilla
17,Cádiz
```

Figura 3. Contenido del fichero '*cities.csv'*

```
Id,FromCity,ToCity,DistanceKM
1,1,2,325
2,1,3,403
3,1,8,395
4,1,4,193
5,1,15,335
6,1,13,251
7,4,5,356
8,4,6,455
9,4,8,280
10,5,6,171
11,8,7,304
12,8,2,324
13,2,10,296
14,10,9,100
15,10,11,349
16,11,13,191
17,11,12,241
18,13,12,150
19,12,14,278
20,14,15,99
21,14,16,256
22,16,15,242
23,16,17,125
```

Figura 4. Contenido del fichero '*distances.csv'*

2.2. Fichero '*create-graph.cypher'*

El contenido de este fichero nos permitirá crear el grafo sobre el que posteriormente aplicaremos el algoritmo. Como se puede observar en la Figura 5, se ha añadido en los parámetros de creación del grafo la especificación *'UNDIRECTED'* o no dirigido. Esto es porque al tratarse de carreteras, por ejemplo, si podemos ir de Valencia a Barcelona, también podremos ir de Barcelona a Valencia.

```
CALL gds.graph.create(
    'citiesGraph',
    'City',
    {ROAD:{ orientation: 'UNDIRECTED' } },
    {relationshipProperties: 'distance' }
);
```

Figura 5. Contenido del fichero '*create-graph.csv'*

2.3. Fichero '*run-dijkstra-algorithm.cypher*'

En este último fichero, se aplicará el algoritmo de *Dijkstra* en el grafo que creamos anteriormente. A modo de ejemplo, tal y como podemos ver en la Figura 6, buscaremos el camino más corto entre Madrid y Barcelona.

```
MATCH (source:City {name: 'Madrid'}), (target:City {name: 'Barcelona'})
CALL gds.shortestPath.dijkstra.stream('citiesGraph', {
    sourceNode: source,
    targetNode: target,
    relationshipWeightProperty: 'distance'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
    index,
    gds.util.asNode(sourceNode).name AS SourceCity,
    gds.util.asNode(targetNode).name AS TargetCity,
    totalCost as TotalDistance,
    [nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS  CityNames,
    costs as Distances,
    nodes(path) as Path
ORDER BY index
```

Figura 6. Contenido del fichero '*run-dijkstra-algorithm.csv'*

Como se ve en la Figura 7, la forma más corta de llegar a Barcelona saliendo de Madrid, es pasando por Zaragoza

index	SourceCity	TargetCity	TotalDistance	CityNames	Distances
0	"Madrid"	"Barcelona"	621.0	["Madrid", "Zaragoza", "Barcelona"]	[0.0, 325.0, 621.0]

Figura 7. Resultado de la ejecución del algoritmo

3. Conclusiones

La realización de esta actividad me ha servido para descubrir una vez más el potencial que tienen este tipo de base de datos, en el que partiendo de un *dataset* cualquiera, se pueden llevar a cabo numerosos análisis aplicando el algoritmo que mejor convenga. Por lo que tal y como comenté en la práctica anterior, considero que me gusta más trabajar con bases de datos basadas en grafos que con otras bases de datos tradicionales como son las relacionales.

4. Bibliografía

[1] “The Neo4j Graph Data Science Library Manual v1.8” [En línea] Disponible en: https://neo4j.com/docs/graph-data-science/current/ [Accedido 1-mar-2022]

[2] “Algorithms” [En línea] Disponible en: https://neo4j.com/docs/graph-data-science/current/algorithms/ [Accedido 1-mar-2022]