Dijkstra Single-Source Shortest Path

Integrantes:
17000666
Ruth Betzabe Castro Acosta

20216390
Julian Enrique Chan Palomo

Materia:
Estructuras de Datos

Profesor:
Luis Basto Díaz

Fecha de entrega:
28/05/2023

Índice

1. Reporte técnico

a. Descripción del dataset a utilizar, explicar de qué se trata, así como sus propiedades o atributos.

El dataset que estamos usando consiste en la medida de la distancia entre ciudades. Nuestro dataset tiene por campos Ciudad origen, Ciudad destino y Distancia.

- Ciudad origen: Este atributo representa la ciudad de partida o el punto de origen de la ruta o conexión. Puede ser representado como un valor de texto o un identificador único para cada ciudad, en nuestro caso es el nombre de la ciudad.
- Ciudad destino: Este atributo representa la ciudad de destino o el punto final de la ruta o conexión. Al igual que la ciudad origen, puede ser representado como un valor de texto o un identificador único, y representamos con el nombre de la ciudad.
- Distancia: Este atributo indica la distancia entre la ciudad origen y la ciudad destino. Por lo general, se representa como un valor numérico que puede estar en unidades de medida como kilómetros, millas u otras unidades de longitud, en nuestro caso es kilometros.

Dependiendo de la distancia relacionada entre ciudades podríamos conocer algunas de estas variables:

Tiempo de viaje: Puede incluir un atributo adicional que represente el tiempo estimado o real de viaje entre la ciudad origen y la ciudad destino. Esto proporciona información sobre la duración del viaje aparte de la distancia.

Modo de transporte: Puede ser útil agregar un atributo que indique el modo de transporte asociado con la ruta, como avión, tren, automóvil, etc. Esto permite diferenciar entre diferentes formas de viaje y puede ser útil para análisis posteriores.

Costo del viaje: Otra opción es incluir un atributo que represente el costo asociado con el viaje entre las dos ciudades. Esto puede ser relevante para análisis de presupuesto, comparaciones de costos, etc.

URL del dataset:
https://drive.google.com/drive/folders/1sKFX0ncWihLG3-reDUsFwe8kxBQrizeW?usp=sharing

b. Descripción a detalle sobre su funcionalidad (pueden tomarlo del Neo4J o buscar en otras referencias).

El algoritmo de Dijkstra Single-Source encuentra la ruta más corta desde un nodo de origen a todos los demás nodos en un grafo ponderado. Comienza desde el nodo de origen y gradualmente explora y actualiza las distancias de los nodos adyacentes. Al final, obtendrás la distancia más corta y el camino para llegar a cada nodo alcanzable desde el nodo de origen. Es una herramienta útil para calcular rutas eficientes en mapas o redes de transporte.

c. Descripción de la definición y llamadas de las funciones/procedimientos de cada algoritmo
(parámetros de entrada, elementos de salida, parámetros de configuración, etc.).

Table 1. Parameters

Name	Type	Default	Optional	Description
graphName	String	`n/a`	no	The name of a graph stored in the catalog.
configuration	Map	`{}`	yes	Configuration for algorithm-specifics and/or graph filtering.

Table 2. Configuration

Name	Type	Default	Optional	Description
nodeLabels	List of String	`['*']`	yes	Filter the named graph using the given node labels.
relationshipTypes	List of String	`['*']`	yes	Filter the named graph using the given relationship types.
jobId	String	`Generated internally`	yes	An ID that can be provided to more easily track the algorithm's progress.
logProgress	Boolean	`true`	yes	If disabled the progress percentage will not be logged.
sourceNode	Integer	`n/a`	no	The Neo4j source node or node id.
relationshipWeightProperty	String	`null`	yes	Name of the relationship property to use as weights. If unspecified, the algorithm runs unweighted.

Table 3. Results

Name	Type	Description
index	Integer	0-based index of the found path.
sourceNode	Integer	Source node of the path.
targetNode	Integer	Target node of the path.
totalCost	Float	Total cost from source to target.
nodeIds	List of Integer	Node ids on the path in traversal order.
costs	List of Float	Accumulated costs for each node on the path.
path	Path	The path represented as Cypher entity.

d. Descripción de los comandos cypher para la ejecución (llamadas) de las funciones/procedimientos de cada algoritmo.

```
CALL gds.shortestPath.dijkstra.stream('nombreGrafo', {
  sourceNode: id_del_nodo_origen,
  targetNode: id_del_nodo_destino,
```

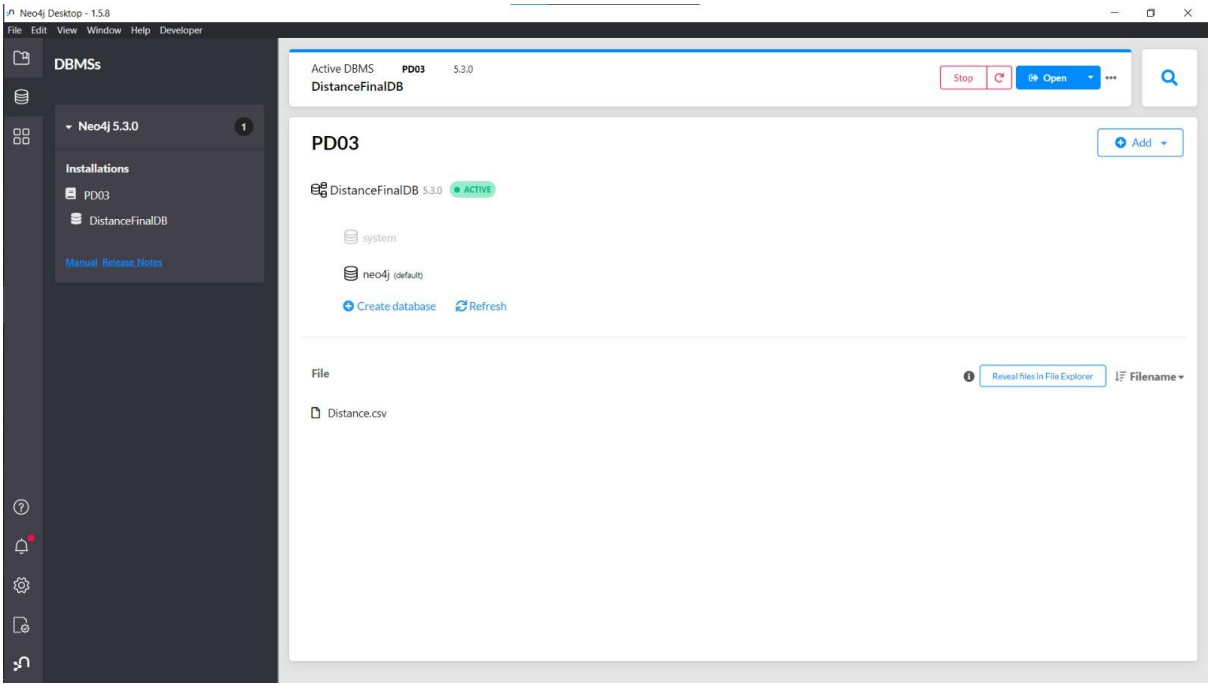
```
  relationshipWeightProperty: 'weight'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN index, sourceNode, targetNode, totalCost, nodeIds, costs, path;
```

- En la primera línea, CALL gds.shortestPath.dijkstra.stream realiza la llamada a la función gds.shortestPath.dijkstra.stream para ejecutar el algoritmo de Dijkstra en el grafo especificado.
- 'nombreGrafo' es el nombre del grafo en el que se realizará el cálculo. Asegúrate de reemplazarlo con el nombre correcto de tu grafo.
- { sourceNode: id_del_nodo_origen, targetNode: id_del_nodo_destino, relationshipWeightProperty: 'weight' } son los parámetros de configuración del algoritmo. Reemplaza id_del_nodo_origen con el identificador del nodo de origen y id_del_nodo_destino con el identificador del nodo de destino entre los que deseas encontrar la ruta más corta. 'weight' es el nombre de la propiedad de peso utilizada en las relaciones del grafo. Asegúrate de reemplazarlo si utilizaste un nombre diferente en tu grafo.
- YIELD especifica las variables que deseas recibir como resultados del algoritmo. En este caso, se obtienen index, sourceNode, targetNode, totalCost, nodeIds, costs y path.

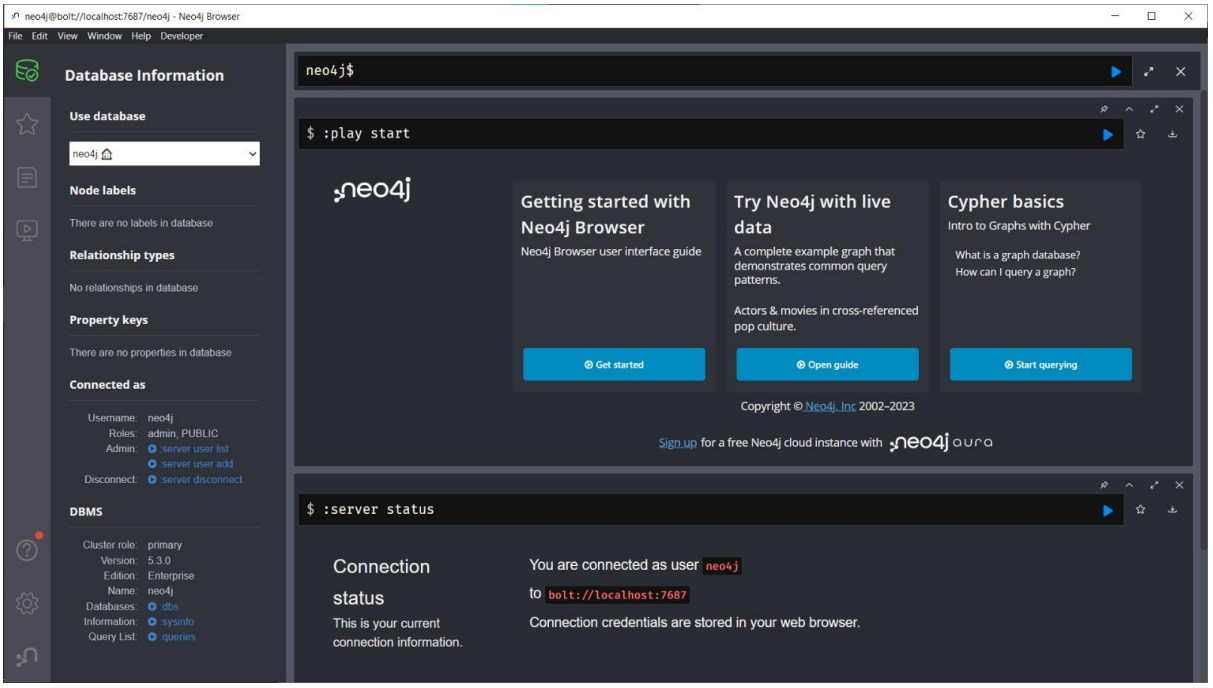
e. Mostrar imágenes (screenshots) con las salidas de los procedimientos, grafos, tablas, etc.

Procedimiento de la creación del proyecto

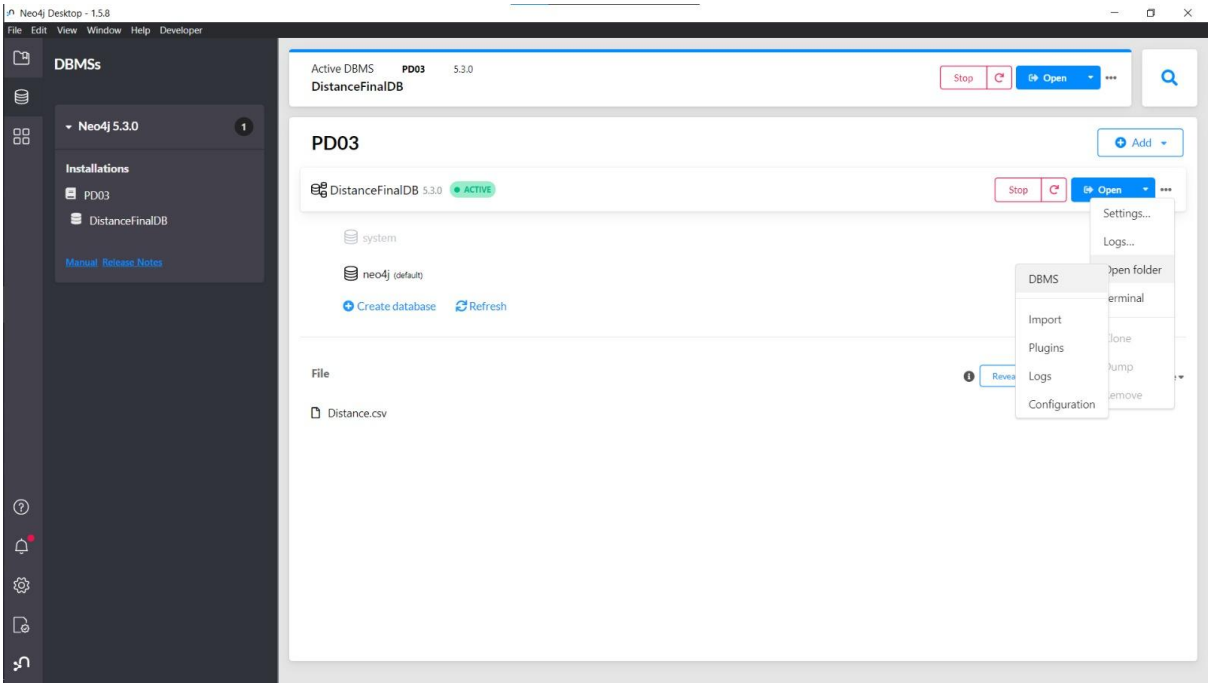
Creamos el proyecto y la base de datos



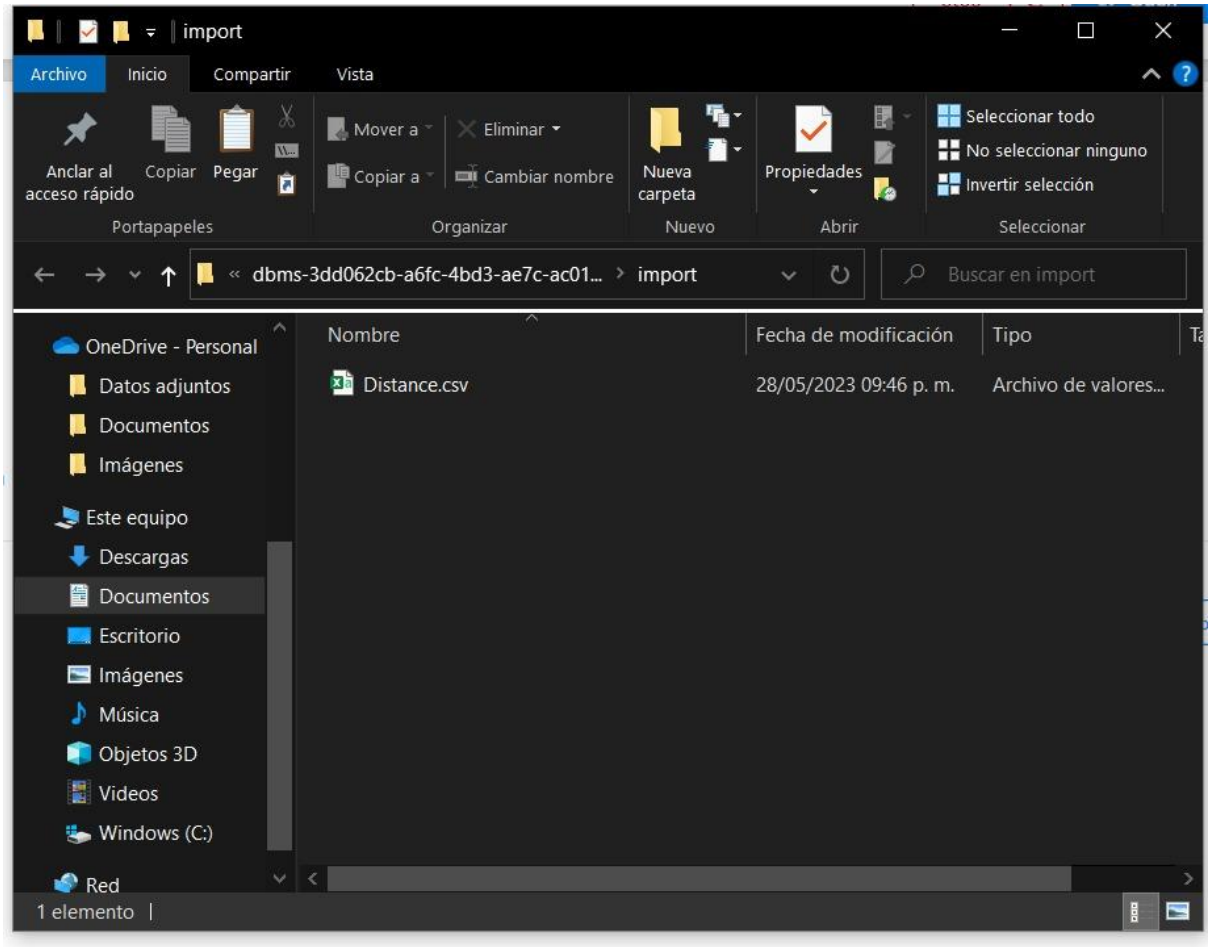
Vista de la base de datos



Para importar el dataset vamos a import



pegamos nuestro archivo con el dataset



Para importar la información a la base de datos hacemos lo siguiente:

Creamos una restricción (constraint) en la base de datos que garantiza la unicidad de los nodos con la etiqueta "Ciudad" en base a la propiedad "nombre", usamos este comando:

CREATE CONSTRAINT FOR (c:Ciudad) REQUIRE c.nombre IS UNIQUE;

Cargamos los datos del CSV con este comando:

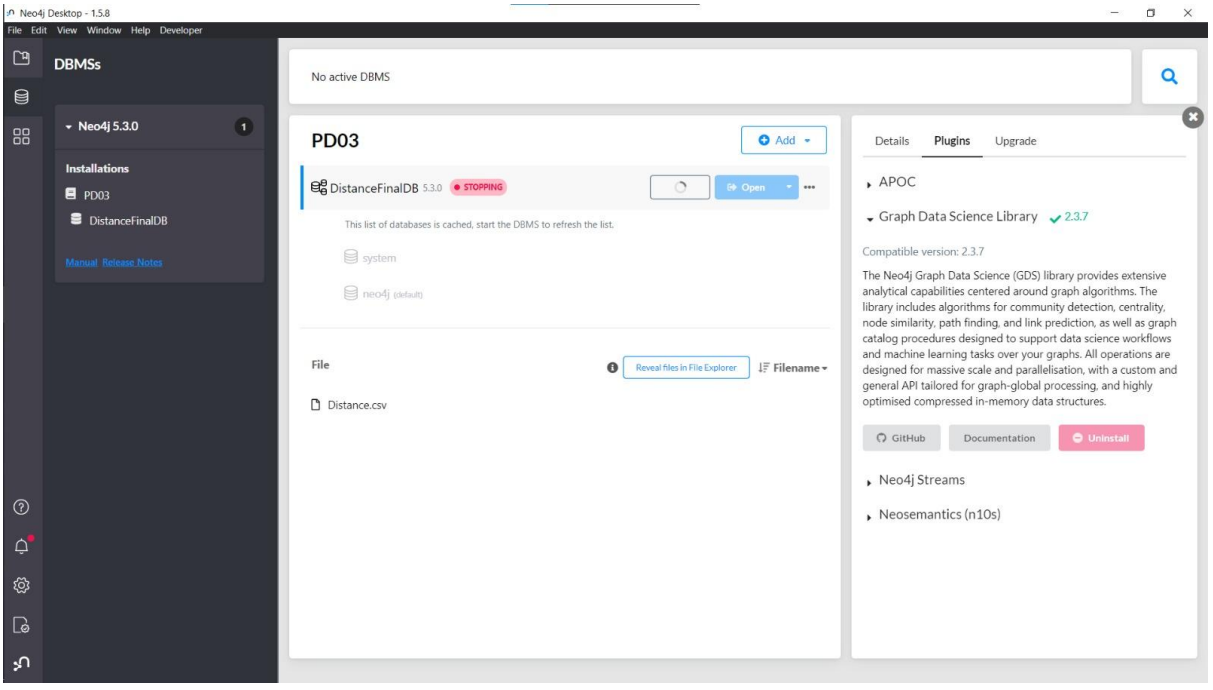
LOAD CSV WITH HEADERS FROM 'file:///tu_archivo.csv' AS row
MERGE (ciudadOrigen:Ciudad {nombre: row.CiudadOrigen})
MERGE (ciudadDestino:Ciudad {nombre: row.CiudadDestino});

Creamos las relaciones de los datos con este comando:

LOAD CSV WITH HEADERS FROM 'file:///tu_archivo.csv' AS row
MATCH (ciudadOrigen:Ciudad {nombre: row.CiudadOrigen})
MATCH (ciudadDestino:Ciudad {nombre: row.CiudadDestino})
CREATE (ciudadOrigen)-[:DISTANCIA {valor: toFloat(row.Distancia)}]->(ciudadDestino);

En este caso nuestro archivo se llama Distance.csv

Tenemos que tener instalado Graph Data Science Library:



Luego le ponemos el nombre al grafo que usaremos

```
CALL gds.graph.project.cypher(
  'distancesGraph',
  'MATCH (c:Ciudad) RETURN id(c) AS id',
  'MATCH (c1:Ciudad)-[d:DISTANCIA]->(c2:Ciudad) RETURN id(c1) AS source, id(c2) AS
target, d.valor AS weight'
)
```

```
YIELD graphName, nodeCount, relationshipCount;
```

Luego aplicamos el algoritmo:

```
CALL gds.shortestPath.dijkstra.stream('distancesGraph', {
  sourceNode: id_del_nodo_origen,
  targetNode: id_del_nodo_destino,
  relationshipWeightProperty: 'weight'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN index, sourceNode, targetNode, totalCost, nodeIds, costs, path;
```

2. Referencias

- Neo4j. (s.f.). Dijkstra Single Source Algorithm. Recuperado de https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-single-source/
- Neo4j. (s.f.). Neo4j Desktop Installation. Recuperado de https://neo4j.com/docs/graph-data-science/current/installation/neo4j-desktop/
- Neo4j. (s.f.). Graph Data Science Library Installation. Recuperado de https://neo4j.com/docs/graph-data-science/current/installation/
- Neo4j. (s.f.). Neo4j Documentation. Recuperado de https://neo4j.com/docs