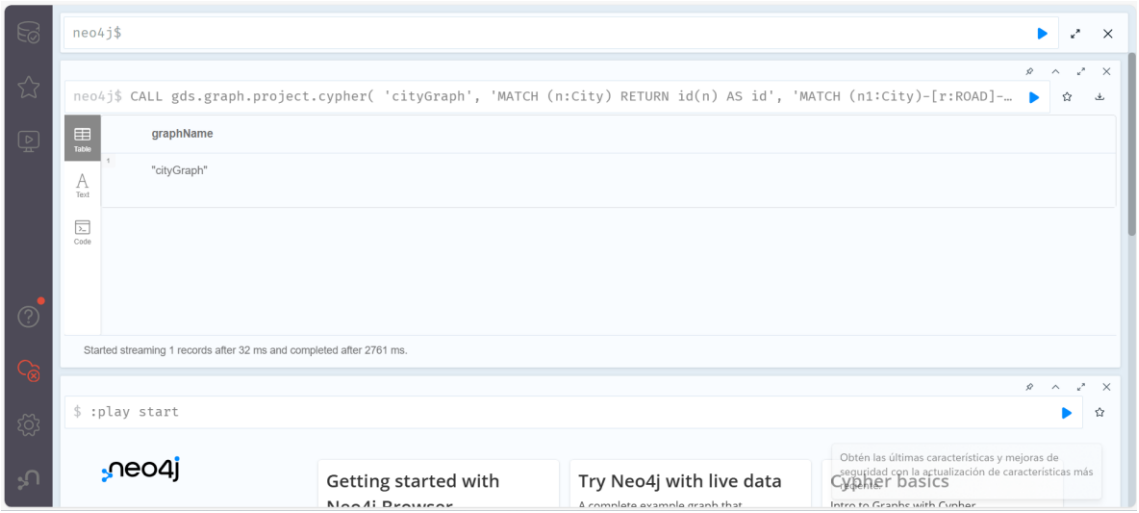
RECORRIDOS Y CAMINOS MÍNIMOS EN NEO4J

Neo4j

RECORRIDOS Y CAMINOS MÍNIMOS EN NEO4J

1- Primero Proyectamos el grafo para GDS. Esta proyección crea una representación optimizada del grafo en la memoria de la librería GDS para ejecutar los algoritmos rápidamente. Comando de proyección:

```
CALL gds.graph.project.cypher(
  'cityGraph',
  'MATCH (n:City) RETURN id(n) AS id',
  'MATCH (n1:City)-[r:ROAD]-(n2:City) RETURN id(n1) AS source, id(n2) AS target, r.distance AS weight'
) YIELD graphName
```

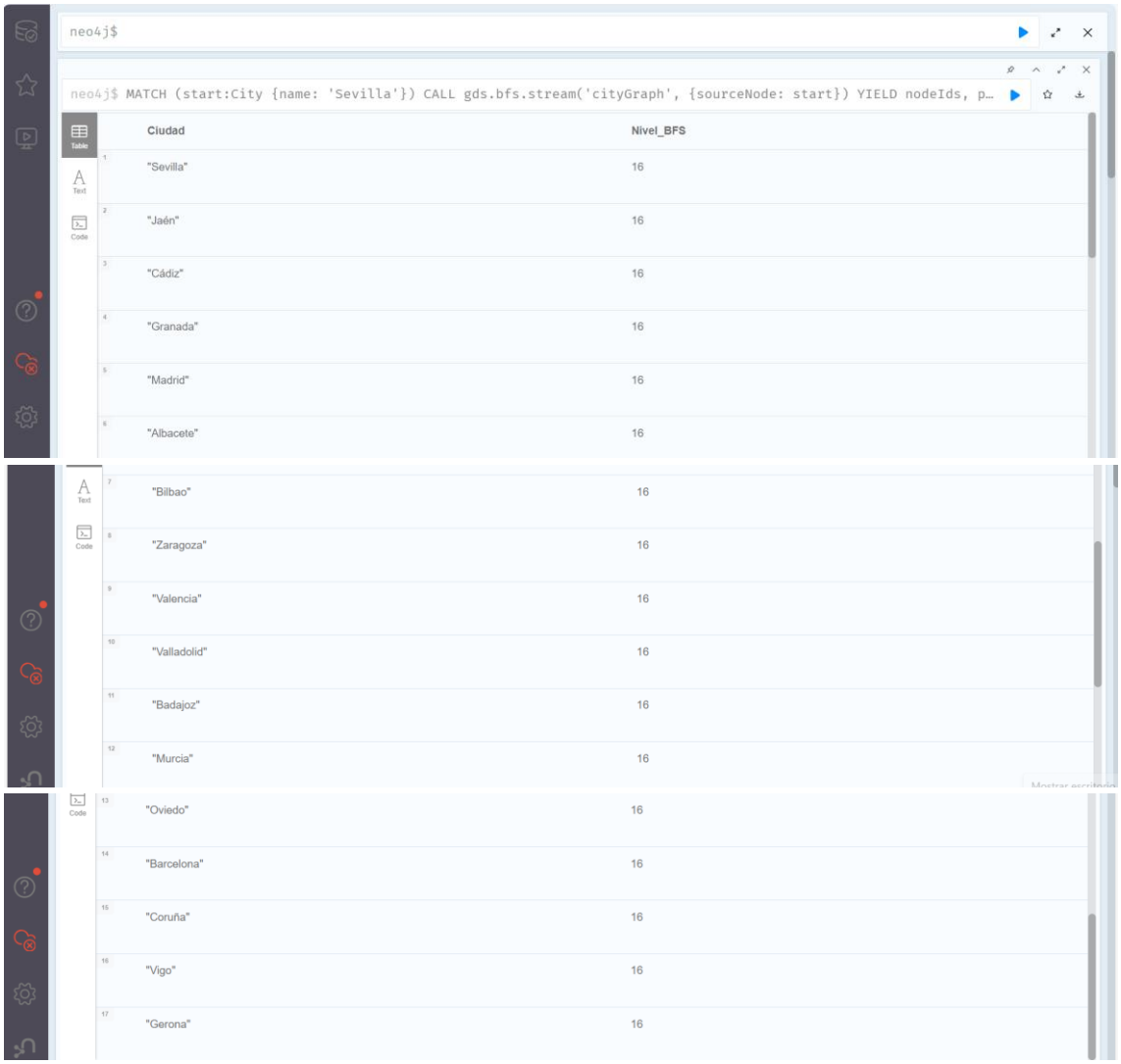


2- RECORRIDOS DE BÚSQUEDA (BFS Y DFS)

a. RECORRIDO BFS DESDE SEVILLA

El recorrido *Breadth-First Search (BFS)* encuentra nodos por "niveles" de distancia, explorando primero a los vecinos más cercanos. Se utiliza la función length(path) para obtener el nivel o número de saltos.

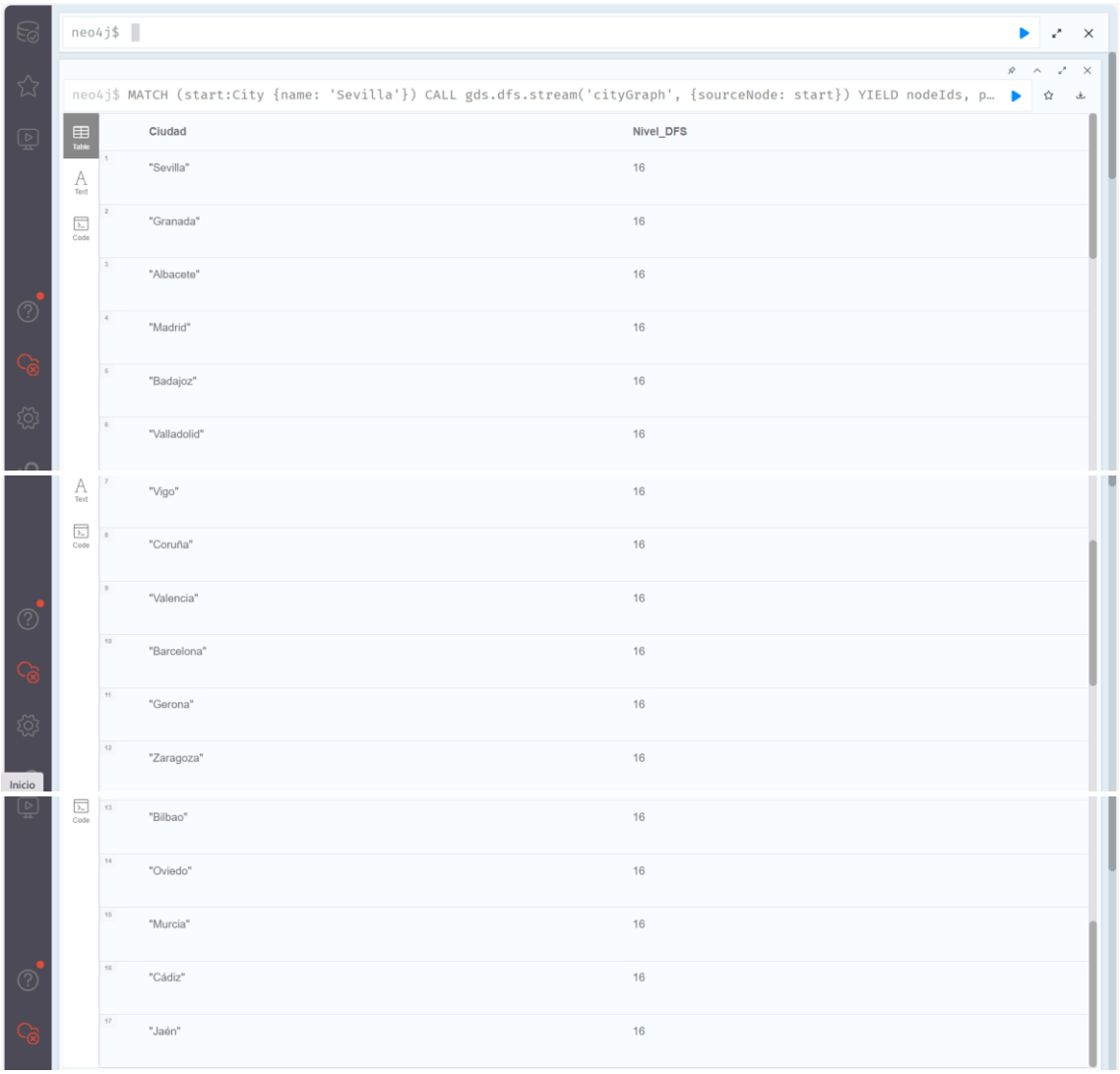
```
MATCH (start:City {name: 'Sevilla'})
CALL gds.bfs.stream('cityGraph', {sourceNode: start})
YIELD nodeIds, path // Usamos path para obtener la ruta
UNWIND nodeIds AS nodeId
RETURN gds.util.asNode(nodeId).name AS Ciudad,
length(path) AS Nivel_BFS
```



b. RECORRIDO DFS DESDE SEVILLA

El recorrido *Depth-First Search (DFS)* explora lo más profundo posible a lo largo de cada rama antes de retroceder.

MATCH (start:City {name: 'Sevilla'})
CALL gds.dfs.stream('cityGraph', {sourceNode: start})
YIELD nodeIds, path // Usamos la sintaxis probada
UNWIND nodeIds AS nodeId
RETURN gds.util.asNode(nodeId).name AS Ciudad, length(path) AS Nivel_DFS



c. RECORRIDO BFS ESPECIFICANDO NODOS DESTINO CORUÑA

Este recorrido se detiene tan pronto como encuentra el nodo destino (Coruña).

```
MATCH (start:City {name: 'Sevilla'}), (target:City {name: 'Coruña'})
CALL gds.bfs.stream('cityGraph', {sourceNode: start, targetNodes: [target]})
YIELD path
RETURN [n IN nodes(path) | n.name] AS Ruta_BFS_Coruña,
    length(path) AS Paradas_Minimas
```

d. RECORRIDO DFS ESPECIFICANDO NODOS DESTINO CORUÑA

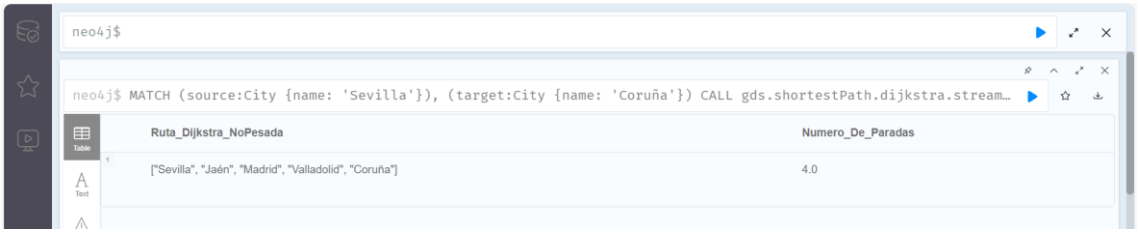
Este recorrido usa el enfoque DFS y se detiene al encontrar Coruña.

MATCH (start:City {name: 'Sevilla'}), (target:City {name: 'Coruña'})
CALL gds.dfs.stream('cityGraph', {sourceNode: start,
targetNodes: [target]})
YIELD path
RETURN [n IN nodes(path) \| n.name] AS Ruta_DFS_Coruña,
length(path) AS Paradas

e. CAMINO MÍNIMO (DIJKSTRA) ENTRE SEVILLA Y CORUÑA (NO PESADO)

Busca el camino con el menor número de paradas.

MATCH (source:City {name: 'Sevilla'}), (target:City {name: 'Coruña'})
CALL gds.shortestPath.dijkstra.stream('cityGraph', {
sourceNode: source,
targetNode: target
// relationshipWeightProperty se omite, resultando en coste unitario.
})
YIELD path, totalCost
RETURN [n IN nodes(path) \| n.name] AS Ruta_Dijkstra_NoPesada,
totalCost AS Numero_De_Paradas



f. CAMINO MÍNIMO (DIJKSTRA) ENTRE SEVILLA Y CORUÑA (PESADO)

El algoritmo A* es una extensión de Dijkstra que utiliza una función heurística (una estimación de distancia en línea recta) para priorizar la búsqueda y encontrar la ruta óptima de manera más rápida.

Comentario y Verificación: Fallo de Software

Inicialmente, el algoritmo A* debe ejecutarse sin coordenadas reales, utilizando una heurística constante (null), lo que debería hacer que se comporte exactamente como Dijkstra (ejercicio 6). Sin embargo, la versión de software GDS 2.5.4 instalada falla al procesar las propiedades NULL proyectadas, lo que impide la ejecución (The property 'lat' has not been loaded).
Para resolver este fallo y obtener el resultado final, fue necesario modificar el grafo temporalmente añadiendo coordenadas a los nodos clave y reproyectando el grafo.

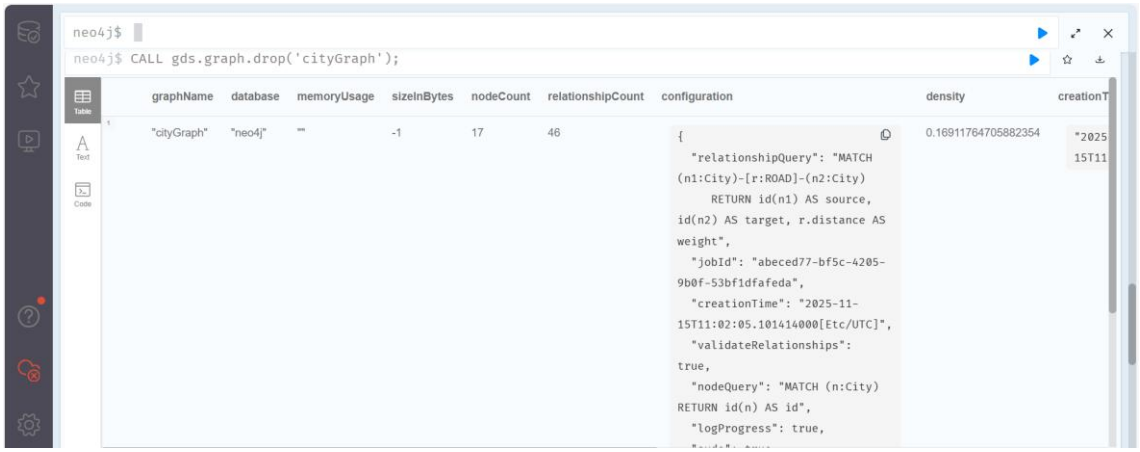
Pasos Ejecutados para Obtener el Resultado (Verificación)

Se ejecutan los siguientes pasos para que el algoritmo A* se ejecute:

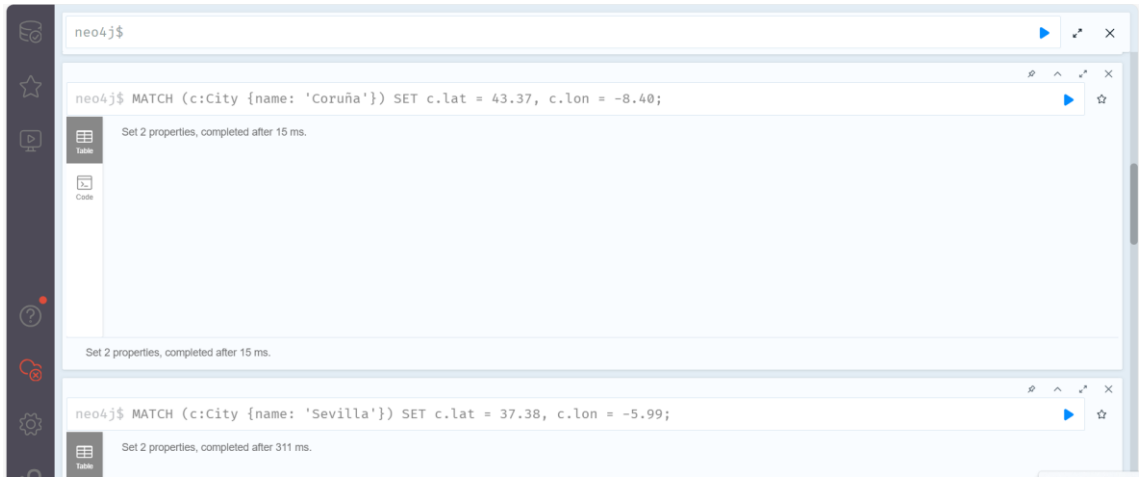
Paso 1: Modificar el Grafo y Reproyectar (Para evitar el Error de Software)

Se añaden coordenadas a los nodos Sevilla y Coruña para que A* pueda cargar las propiedades lat y lon, y luego se reproyecta el grafo para usar esos valores. Para ello realizaremos:

- **Eliminar la proyección anterior:** ***CALL gds.graph.drop('cityGraph')***



- **Añadir coordenadas a Sevilla y a Coruña:**
  MATCH (c:City {name: 'Sevilla'}) SET c.lat = 37.38, c.lon = -5.99;
  MATCH (c:City {name: 'Coruña'}) SET c.lat = 43.37, c.lon = -8.40;



- **Crear la nueva proyección con las coordenadas reales**
  CALL gds.graph.project.cypher('cityGraph',
  'MATCH (n:City) RETURN id(n) AS id, n.lat AS lat, n.lon AS lon',
  'MATCH (n1:City)-[r:ROAD]-(n2:City) RETURN id(n1) AS source,
  id(n2) AS target, r.distance AS weight') YIELD graphName

Paso 2: Ejecución del Algoritmo A*

Se ejecuta el algoritmo A* utilizando las propiedades lat y lon que ahora existen en la proyección.

```
MATCH (source:City {name: 'Sevilla'}), (target:City {name: 'Coruña'})
CALL gds.shortestPath.astar.stream('cityGraph', {
  sourceNode: source,
  targetNode: target,
  relationshipWeightProperty: 'weight',
  latitudeProperty: 'lat',
  longitudeProperty: 'lon'
})
YIELD path, totalCost
RETURN [n IN nodes(path) | n.name] AS Ruta_Astar,
    totalCost AS Distancia_Total_Km
```

Conclusión

El resultado del algoritmo A* es idéntico al de Dijkstra pesado (1225.0 Km). ¿Qué habría que modificar en el grafo actual para obtener un resultado diferente?

La modificación necesaria es añadir coordenadas geográficas precisas (lat, lon) a todos los nodos que componen el grafo y sus rutas. Si la heurística se calcula correctamente, A* podría encontrar una ruta más eficiente si el camino más corto por distancia no fuera también el más directo geográficamente, pero en el grafo actual, la ruta más corta por distancia es también la óptima.