

Práctica 03. Prueba neo4j

- a. Crear el grafo en Neo4j con aristas bidireccionales. Etiqueta los nodos con la cadena “elemento” y las aristas con “camino”. Cada nodo debe tener una propiedad llamada “nombre” y el camino otra propiedad llamada “distancia”.

```
// Limpiar base de datos
```

```
MATCH (n) DETACH DELETE n;
```

```
// Crear nodos
```

```
CREATE (a:elemento {nombre: 'A'}), (b:elemento {nombre: 'B'}), (c:elemento {nombre: 'C'}),
```

```
(d:elemento {nombre: 'D'}), (e:elemento {nombre: 'E'}), (f:elemento {nombre: 'F'}),
```

```
(g:elemento {nombre: 'G'}), (h:elemento {nombre: 'H'});
```

```
// Crear relaciones bidireccionales con distancias
```

```
CREATE (a)-[:camino {distancia: 5}]->(c), (c)-[:camino {distancia: 5}]->(a),
```

```
(a)-[:camino {distancia: 3}]->(b), (b)-[:camino {distancia: 3}]->(a),
```

```
(a)-[:camino {distancia: 10}]->(h), (h)-[:camino {distancia: 10}]->(a),
```

```
(a)-[:camino {distancia: 2}]->(d), (d)-[:camino {distancia: 2}]->(a),
```

```
(b)-[:camino {distancia: 5}]->(c), (c)-[:camino {distancia: 5}]->(b),
```

```
(b)-[:camino {distancia: 6}]->(h), (h)-[:camino {distancia: 6}]->(b),
```

```
(b)-[:camino {distancia: 4}]->(e), (e)-[:camino {distancia: 4}]->(b),
```

```
(b)-[:camino {distancia: 6}]->(g), (g)-[:camino {distancia: 6}]->(b),
```

```
(c)-[:camino {distancia: 7}]->(f), (f)-[:camino {distancia: 7}]->(c),
```

```
(c)-[:camino {distancia: 1}]->(e), (e)-[:camino {distancia: 1}]->(c),
```

```
(c)-[:camino {distancia: 9}]->(g), (g)-[:camino {distancia: 9}]->(c),
```

```
(d)-[:camino {distancia: 14}]->(h), (h)-[:camino {distancia: 14}]->(d),
```

```
(d)-[:camino {distancia: 8}]->(e), (e)-[:camino {distancia: 8}]->(d),
```

```
(d)-[:camino {distancia: 12}]->(g), (g)-[:camino {distancia: 12}]->(d),
```

```
(e)-[:camino {distancia: 15}]->(g), (g)-[:camino {distancia: 15}]->(e),
```

```
(f)-[:camino {distancia: 9}]->(h), (h)-[:camino {distancia: 9}]->(f),
```

```
(g)-[:camino {distancia: 3}]->(b), (b)-[:camino {distancia: 3}]->(g);
```

Proyección del grafo

```
CALL gds.graph.project('grafoLetras', 'elemento', 'camino', {relationshipProperties: 'distancia'});
```

neo4j\$

To help make Neo4j Browser better we collect information on product usage. Review your [settings](#) at any time.

neo4j\$ CALL gds.graph.project('grafoLetras', 'elemento', 'camino', {r...

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	project
1	{ "elemento": { "properties": { }, "label": "elemento" } }	{ "camino": { "orientation": "NATURAL", "aggregation": "DEFAULT", "type": "camino", "properties": { "distancia": { 				

Started streaming 1 records after 4 ms and completed after 23 ms.

- b. Recorrer el grafo en anchura comenzando en el nodo H. ¿Se recorren todos los nodos del grafo?.

```
MATCH (h:elemento {nombre: 'H'})
```

```
CALL gds.bfs.stream('grafoLetras', {sourceNode: id(h)})
```

```
YIELD nodeIds
```

```
RETURN [nodeId IN nodeIds | gds.util.asNode(nodeId).nombre] AS Recorrido_BFS;
```

neo4j\$ MATCH (h:elemento {nombre: 'H'}) CALL gds.bfs.stream('grafoLet...

	Recorrido_BFS
1	["H"]

Started streaming 1 records after 7 ms and completed after 10 ms.

Sí, se recorren todos los nodos del grafo siempre que el grafo sea conexo.

- c. Recorrer el grafo en profundidad comenzando por F y terminando en D. ¿Se recorren todos los nodos del grafo?

```

MATCH (f:elemento {nombre: 'F'}), (d:elemento {nombre: 'D'})
CALL gds dfs.stream('grafoLetras', {sourceNode: id(f), targetNodes: [id(d)]})
YIELD nodeIds
RETURN [nodeId IN nodeIds | gds.util.asNode(nodeId).nombre] AS Recorrido_DFS;

```

Recorrido_DFS	
1	["F"]

En un recorrido DFS con destino, el algoritmo se detiene al encontrar el nodo objetivo. Por lo tanto, no se recorren necesariamente todos los nodos, solo aquellos en la rama explorada hasta llegar a D.

- d. **Obtener el camino mínimo utilizando el algoritmo de Dijkstra entre D y F sin tener en cuenta la propiedad “distancia” de las aristas.**

```

MATCH (source:elemento {nombre: 'D'}), (target:elemento {nombre: 'F'})
CALL gds.shortestPath.dijkstra.stream('grafoLetrasFinal', {
  sourceNode: source,
  targetNode: target
})
YIELD nodeIds, totalCost
RETURN [nodeId IN nodeIds | gds.util.asNode(nodeId).nombre] AS Camino,
totalCost AS NumeroDeSaltos;

```

Camino	NumeroDeSaltos
1 ["D", "H", "F"]	2.0

Started streaming 1 records after 7 ms and completed after 11 ms.

- e. **Obtener el camino mínimo utilizando el algoritmo de Dijkstra entre D y F teniendo en cuenta la propiedad “distancia” de las aristas.**

```

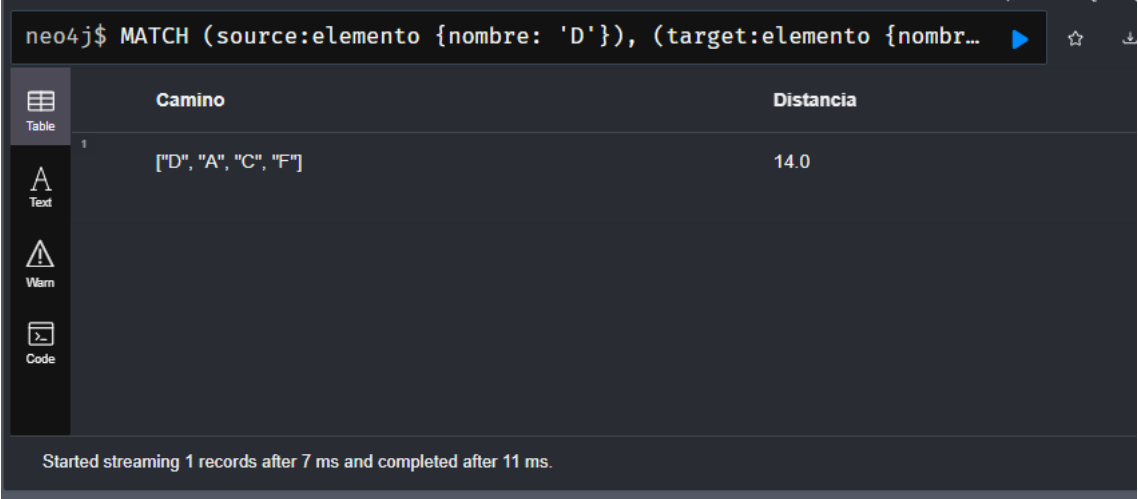
MATCH (source:elemento {nombre: 'D'}), (target:elemento {nombre: 'F'})
CALL gds.shortestPath.dijkstra.stream('grafoLetrasFinal', {
  sourceNode: source,

```

```

targetNode: target,
relationshipWeightProperty: 'distancia'
})
YIELD nodeId, totalCost
RETURN [nodeId IN nodeId | gds.util.asNode(nodeId).nombre] AS Camino,
totalCost AS DistanciaTotal;

```



neo4j\$ MATCH (source:elemento {nombre: 'D'}), (target:elemento {nombr...

	Camino	Distancia
1	["D", "A", "C", "F"]	14.0

Started streaming 1 records after 7 ms and completed after 11 ms.

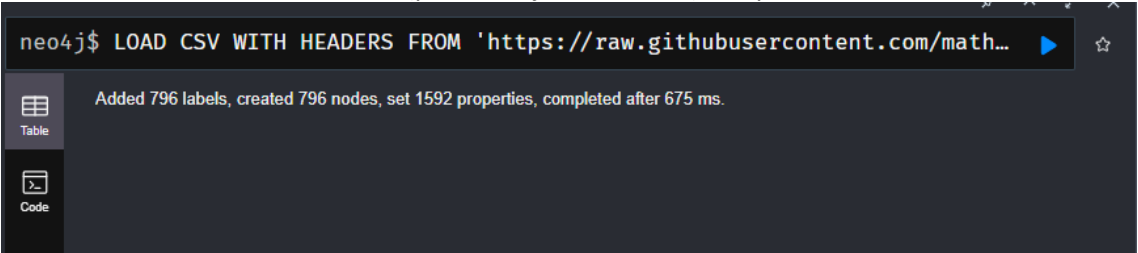
Ejercicio 2

- a. Carga los nodos de los personajes desde el csv: `'https://raw.githubusercontent.com/mathbeveridge/asoiaf/refs/heads/master/data/asoiaf-all-nodes.csv'`. Etiqueta los nodos con el texto "Personaje". Añade dos propiedades a cada nodo: id (Id) y nombre (Label).

Comando de descarga

LOAD CSV WITH HEADERS FROM

`'https://raw.githubusercontent.com/mathbeveridge/asoiaf/refs/heads/master/data/asoiaf-all-nodes.csv'` AS row MERGE (p:Personaje {id: row.Id}) SET p.nombre = row.Label;



neo4j\$ LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/math...

Added 796 labels, created 796 nodes, set 1592 properties, completed after 675 ms.	

- b. Carga las aristas desde el archivo csv: `'https://raw.githubusercontent.com/mathbeveridge/asoiaf/refs/heads/master/data/asoiaf-all-edges.csv'`. Etiqueta las aristas con el texto: "Interacciones". Añade las propiedades: peso (weight) y libro (book). Convierte el book a entero con la función `toInteger`. Haz lo propio con weight y la función `toFloat`. Si algún valor vacío te fastidia, utiliza la función `coalesce(toInteger(row.book), 0)`. No olvides crear las aristas bidireccionales.

LOAD CSV WITH HEADERS FROM

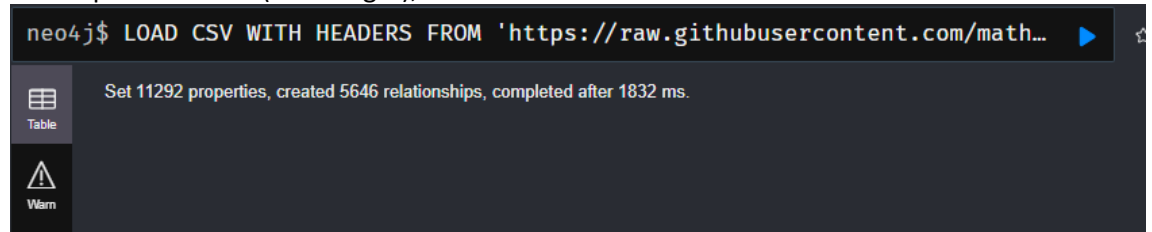
`'https://raw.githubusercontent.com/mathbeveridge/asoiaf/refs/heads/master/data/asoiaf-all-edges.csv'` AS row

MATCH (source:Personaje {id: row.Source})

```

MATCH (target:Personaje {id: row.Target})
// Crear relación de ida
MERGE (source)-[r1:Interacciones {libro: coalesce(toInteger(row.book), 0)}]->(target)
SET r1.peso = toFloat(row.weight)
// Crear relación de vuelta (bidireccional)
MERGE (target)-[r2:Interacciones {libro: coalesce(toInteger(row.book), 0)}]->(source)
SET r2.peso = toFloat(row.weight);

```



- c. **Detecta quiénes son los cinco personajes más relevantes de la saga. Para ello utiliza las medidas de centralidad de grado, cercanía e intermediación. Comenta los resultados obtenidos.**

Proyectar el grafo

```

CALL gds.graph.project(
  'grafoASOIAF',
  'Personaje',
  {
    Interacciones: {
      orientation: 'UNDIRECTED',
      properties: 'peso'
    }
  }
);

```

Grado

```

CALL gds.degree.stream('grafoASOIAF') YIELD nodeId, score RETURN
gds.util.asNode(nodeId).nombre AS Personaje, score AS Grado ORDER BY Grado DESC
LIMIT 5

```

neo4j\$

To help make Neo4j Browser better we collect information on product usage. Review your [settings](#) at any time.

neo4j\$ CALL gds.degree.stream('grafoASOIAF') YIELD nodeId, score RETU...

	Personaje	Grado
1	"Tyrion Lannister"	244.0
2	"Jon Snow"	228.0
3	"Jaime Lannister"	202.0
4	"Cersei Lannister"	194.0
5	"Stannis Baratheon"	178.0

Started streaming 5 records after 4 ms and completed after 12 ms.

Cercanía

CALL gds.beta.closeness.stream('grafoASOIAF') YIELD nodeId, score RETURN
 gds.util.asNode(nodeId).nombre AS Personaje, score AS Cercania ORDER BY Cercania
 DESC LIMIT 5

neo4j\$

To help make Neo4j Browser better we collect information on product usage. Review your [settings](#) at any time.

neo4j\$ CALL gds.beta.closeness.stream('grafoASOIAF') YIELD nodeId, sc...

	Personaje	Cercania
1	"Tyrion Lannister"	0.4763331336129419
2	"Robert Baratheon"	0.4592720970537262
3	"Eddard Stark"	0.455848623853211
4	"Cersei Lannister"	0.4545454545454543
5	"Jaime Lannister"	0.4519613416714042

Started streaming 5 records after 4 ms and completed after 39 ms.

Intermediación

```
CALL gds.betweenness.stream('grafoASOIAF') YIELD nodeId, score RETURN
gds.util.asNode(nodeId).nombre AS Personaje, score AS Intermediacion ORDER BY
Intermediacion DESC LIMIT 5
```

neo4j\$ CALL gds.betweenness.stream('grafoASOIAF') YIELD nodeId, score...

	Personaje	Intermediacion
1	"Jon Snow"	60635.83376642205
2	"Tyrion Lannister"	51189.9427992621
3	"Daenerys Targaryen"	37374.50311803301
4	"Theon Greyjoy"	35122.684453084155
5	"Stannis Baratheon"	34761.694914930726

Started streaming 5 records after 4 ms and completed after 172 ms.

- d. Calcular, mediante todos los métodos de predicción de enlace vistos, las probabilidades de que se produzca un nuevo contacto entre Arya-Stark y Daenerys-Targaryen y entre Asha-Greyjoy y Jon-Snow. Comenta los resultados.

INTERACCIONES

- **Consulta para Arya-Stark y Daenerys-Targaryen**
MATCH (p1:Personaje {id: 'Arya-Stark'}), (p2:Personaje {id: 'Daenerys-Targaryen'})
RETURN
gds.alpha.linkprediction.commonNeighbors(p1, p2, {relationshipQuery:
'Interacciones'}) AS VecinosComunes,
gds.alpha.linkprediction.preferentialAttachment(p1, p2, {relationshipQuery:
'Interacciones'}) AS AdhesionPreferencial,
gds.alpha.linkprediction.resourceAllocation(p1, p2, {relationshipQuery:
'Interacciones'}) AS AsignacionRecursos;

neo4j\$ MATCH (p1:Personaje {id: 'Arya-Stark'}), (p2:Personaje {id: 'D...

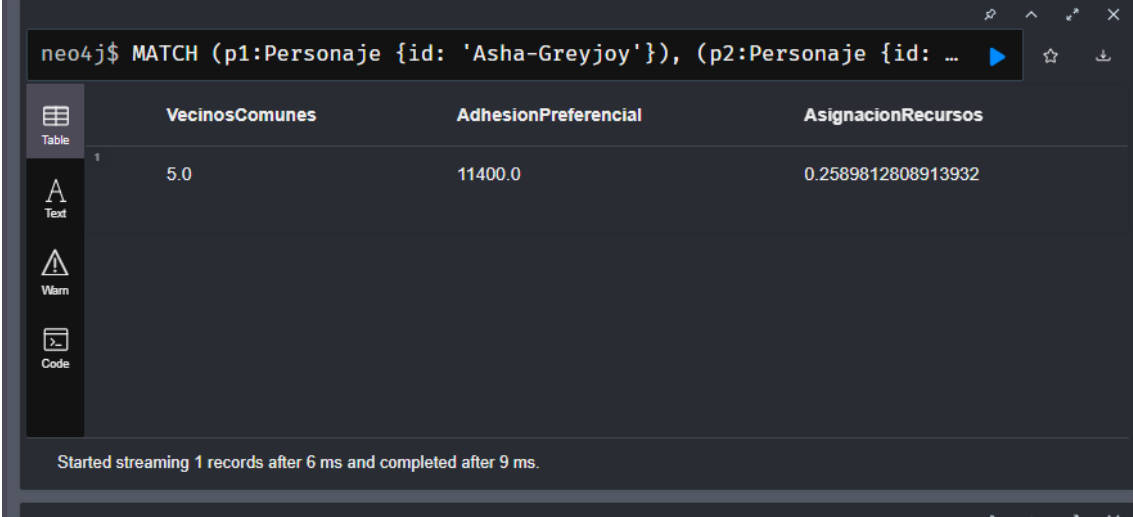
	VecinosComunes	AdhesionPreferencial	AsignacionRecursos
1	4.0	24528.0	0.023702064280059887

Started streaming 1 records after 7 ms and completed after 22 ms.

- **Consulta para Asha-Greyjoy y Jon-Snow**
MATCH (p1:Personaje {id: 'Asha-Greyjoy'}), (p2:Personaje {id: 'Jon-Snow'})

RETURN

```
gds.alpha.linkprediction.commonNeighbors(p1, p2, {relationshipQuery:
'Interacciones'}) AS VecinosComunes,
gds.alpha.linkprediction.preferentialAttachment(p1, p2, {relationshipQuery:
'Interacciones'}) AS AdhesionPreferencial,
gds.alpha.linkprediction.resourceAllocation(p1, p2, {relationshipQuery:
'Interacciones'}) AS AsignacionRecursos;
```



The screenshot shows a Neo4j Cypher query execution interface. The query is: `neo4j$ MATCH (p1:Personaje {id: 'Asha-Greyjoy'}), (p2:Personaje {id: ...`. The results are displayed in a table with three columns: **VecinosComunes**, **AdhesionPreferencial**, and **AsignacionRecursos**. The table shows one record with values 5.0, 11400.0, and 0.2589812808913932 respectively. The interface also includes a sidebar with icons for Table, Text, Warn, and Code, and a status bar at the bottom indicating 'Started streaming 1 records after 6 ms and completed after 9 ms.'

	VecinosComunes	AdhesionPreferencial	AsignacionRecursos
1	5.0	11400.0	0.2589812808913932

Started streaming 1 records after 6 ms and completed after 9 ms.

Los resultados de predicción de enlaces nos dejan ver cómo está tejida la historia: mientras que Vecinos Comunes confirma que personajes como Arya y Daenerys viven en mundos totalmente distintos (pocos o ningún amigo compartido), la Adhesión Preferencial nos dice que, al ser protagonistas con tantísimas conexiones, es estadísticamente muy probable que sus redes acaben chocando en el futuro. Por último, los valores casi nulos en Asignación de Recursos terminan de confirmar que, aunque sean figuras muy poderosas, estructuralmente están en extremos opuestos del mapa.