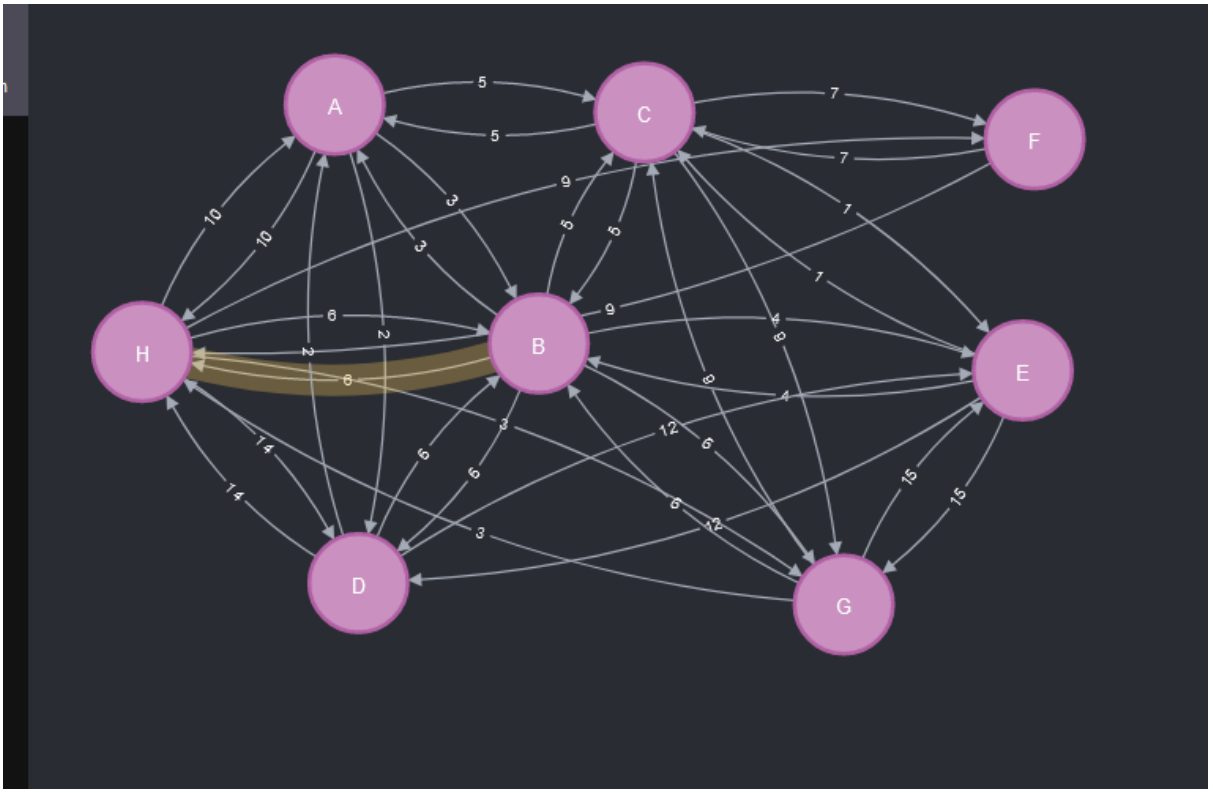
Actividad 4.
Prueba Neo4j

Nombre: Alvaro Lucio-Villegas de Cea

Ejercicio 1. Dado el grafo de la figura:
a) Crear el grafo en Neo4j (Aristas bidireccionales)

```
CREATE
   (A:Letras  {Nombre:'A'}),
   (C:Letras  {Nombre:'C'}),
   (F:Letras  {Nombre:'F'}),
   (H:Letras  {Nombre:'H'}),
   (B:Letras  {Nombre:'B'}),
   (E:Letras  {Nombre:'E'}),
   (D:Letras  {Nombre:'D'}),
   (G:Letras  {Nombre:'G'}),


   (A)-[:distancia{km:5}]->(C),
         (C)-[:distancia{km:5}]->(A),
   (A)-[:distancia{km:10}]->(H),
         (H)-[:distancia{km:10}]->(A),
   (A)-[:distancia{km:2}]->(D),
   (D)-[:distancia{km:2}]->(A),
   (A)-[:distancia{km:3}]->(B),
         (B)-[:distancia{km:3}]->(A),
   (C)-[:distancia{km:5}]->(B),
         (B)-[:distancia{km:5}]->(C),
   (C)-[:distancia{km:7}]->(F),
         (F)-[:distancia{km:7}]->(C),
   (C)-[:distancia{km:1}]->(E),
         (E)-[:distancia{km:1}]->(C),
   (F)-[:distancia{km:9}]->(H),
         (H)-[:distancia{km:9}]->(F),
   (H)-[:distancia{km:6}]->(B),
         (B)-[:distancia{km:6}]->(H),
   (H)-[:distancia{km:3}]->(G),
         (G)-[:distancia{km:3}]->(H),
   (H)-[:distancia{km:14}]->(D),
         (D)-[:distancia{km:14}]->(H),
   (B)-[:distancia{km:4}]->(E),
         (E)-[:distancia{km:4}]->(B),
   (B)-[:distancia{km:6}]->(D),
         (D)-[:distancia{km:6}]->(B),
   (B)-[:distancia{km:6}]->(G),
         (G)-[:distancia{km:6}]->(B),
   (E)-[:distancia{km:12}]->(D),
         (D)-[:distancia{km:12}]->(E),
   (E)-[:distancia{km:15}]->(G),
         (G)-[:distancia{km:15}]->(E),
   (C)-[:distancia{km:9}]->(G),
         (G)-[:distancia{km:9}]->(C)
```



b) Recorrer el grafo en anchura y en profundidad, comenzando en el nodo H

-Anchura:

```
CALL gds.graph.project('AnchuraH','Letras','distancia',{relationshipProperties:'km'})

MATCH (H:Letras{Nombre:'H'})
WITH id(H) AS inicio
CALL gds.bfs.stream('AnchuraH',{sourceNode:inicio})
YIELD path
UNWIND [n in nodes(path) | n.Nombre] AS tags
RETURN tags
```

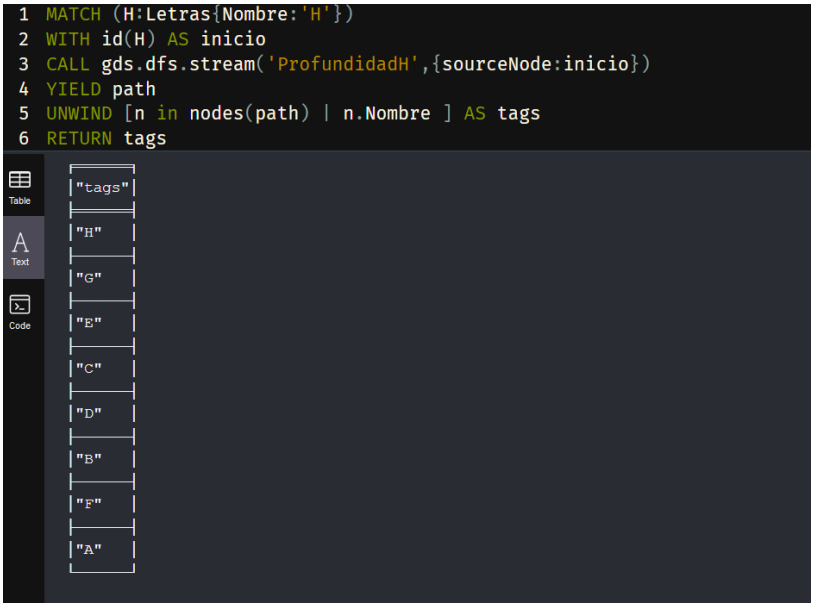
```
1 MATCH (H:Letras{Nombre:'H'})
2 WITH id(H) AS inicio
3 CALL gds.bfs.stream('AnchuraH',{sourceNode:inicio})
4 YIELD path
5 UNWIND [n in nodes(path) | n.Nombre ] AS tags
6 RETURN tags
```

Table / Text / Code

"tags"
"H"
"A"
"F"
"B"
"D"
"G"
"C"
"E"

-Profundidad:

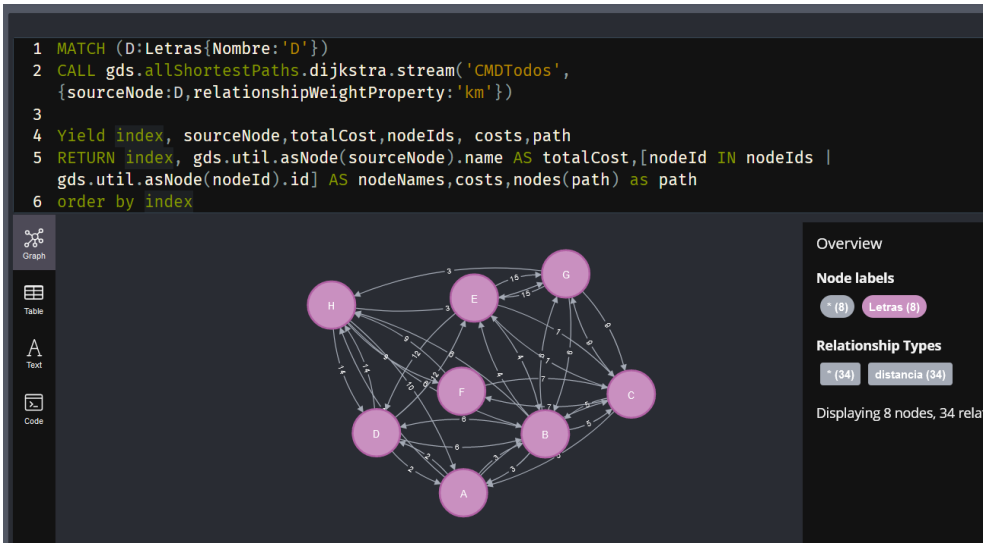
```
CALL gds.graph.project('ProfundidadH','Letras','distancia',{relationshipProperties:'km'})

MATCH (H:Letras{Nombre:'H'})
WITH id(H) AS inicio
CALL gds.dfs.stream('ProfundidadH',{sourceNode:inicio})
YIELD path
UNWIND [n in nodes(path) | n.Nombre ] AS tags
RETURN tags
```



c) Obtener el camino mínimo, utilizando el algoritmo de Dijkstra, entre D y el resto de los nodos del grafo

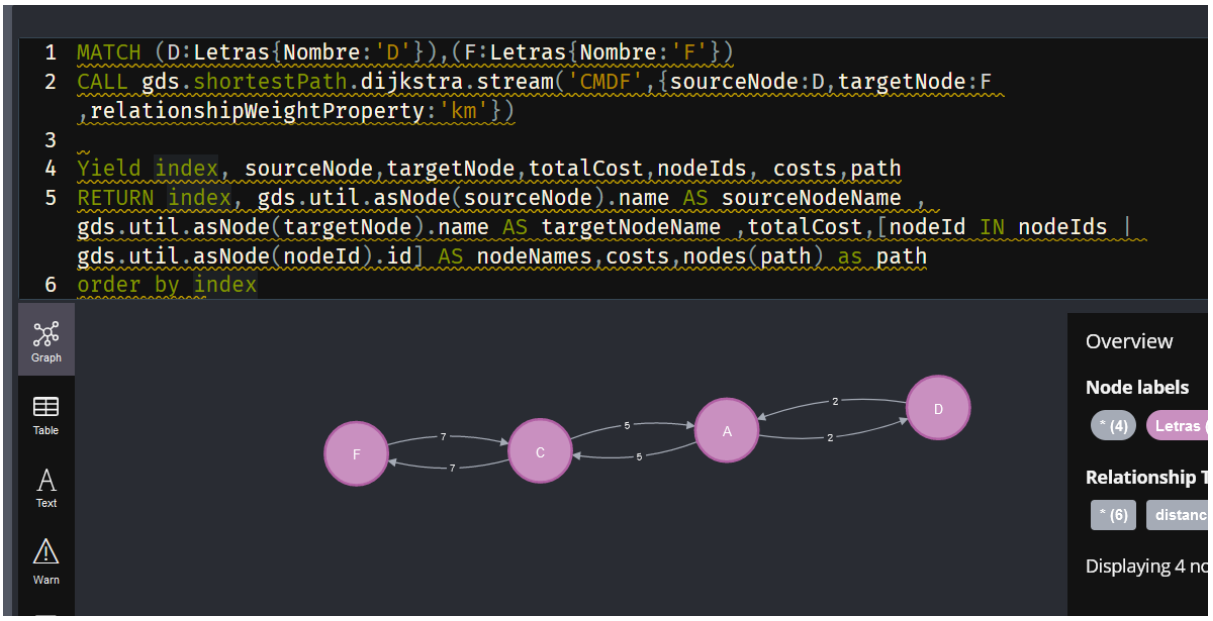
```
CALL gds.graph.project('CMDTodos','Letras','distancia',{relationshipProperties:'km'}    )

MATCH (D:Letras{Nombre:'D'})
CALL
gds.allShortestPaths.dijkstra.stream('CMDTodos',{sourceNode:D,relationshipWeightPrope
rty:'km'})

Yield index, sourceNode,totalCost,nodeIds, costs,path
RETURN index, gds.util.asNode(sourceNode).name AS totalCost,[nodeId IN nodeIds |
gds.util.asNode(nodeId).id] AS nodeNames,costs,nodes(path) as path
order by index
```

```
MATCH (D:Letras{Nombre:'D'})
CALL gds.allShortestPaths.dijkstra.stream('CMDTodos',
{sourceNode:D,relationshipWeightProperty:'km'})

Yield index, sourceNode,totalCost,nodeIds, costs,path
RETURN index, gds.util.asNode(sourceNode).name AS totalCost,[nodeId IN nodeIds |
gds.util.asNode(nodeId).id] AS nodeNames,costs,nodes(path) as path
order by index
```



d) Obtener el camino mínimo, utilizando el algoritmo de Dijkstra, entre D y F

```
CALL gds.graph.project('CMDF','Letras','distancia',{relationshipProperties:'km'}    )

MATCH (D:Letras{Nombre:'D'}),(F:Letras{Nombre:'F'})
CALL gds.shortestPath.dijkstra.stream('CMDF',{sourceNode:D,targetNode:F
,relationshipWeightProperty:'km'})

Yield index, sourceNode,targetNode,totalCost,nodeIds, costs,path
RETURN index, gds.util.asNode(sourceNode).name AS sourceNodeName ,
gds.util.asNode(targetNode).name AS targetNodeName ,totalCost,[nodeId IN nodeIds |
gds.util.asNode(nodeId).id] AS nodeNames,costs,nodes(path) as path
order by index
```



Ejercicio 2. Medidas de centralidad en los siete reinos
Vamos a estudiar diversas relaciones entre los personajes de Juego de Tronos. Las siguientes relaciones se han obtenido evaluando cuando dos personajes coinciden de diversas maneras:
● Coinciden en una conversación
● Coinciden en un lugar
● Dos personajes hablan sobre un tercero
Para más información sobre la extracción de estos datos, en la línea de comandos de neo4j sandbox escribir :play got y pulsar sobre “Exploratory Data Analysis”

Si prefieres cargar directamente la información, ejecuta estos cinco scripts, cada uno obtenido de un libro diferente.

Cuidado porque al copiar del PDF las líneas se separan con un retorno de carro y eso rompe las URLs. Iván dice: “También podéis importar la información de golpe y no por libros” Solo tenéis que buscar un poco y aplicar la lógica a la hora de construir las instrucciones.

```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/neo4j-examples/graphgists/master/bro
wser-guides/data/asoiaf-book1-edges.csv' AS row
MERGE (src:Character {name: row.Source})
MERGE (tgt:Character {name: row.Target})
// relationship for the book
MERGE (src)-[r:INTERACTS1]->(tgt)
ON CREATE SET r.weight = toInteger(row.weight), r.book=1
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/neo4j-examples/graphgists/master/bro
wser-guides/data/asoiaf-book2-edges.csv' AS row
MERGE (src:Character {name: row.Source})
MERGE (tgt:Character {name: row.Target})
// relationship for the book
MERGE (src)-[r:INTERACTS2]->(tgt)
ON CREATE SET r.weight = toInteger(row.weight), r.book=2
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/neo4j-examples/graphgists/master/bro
wser-guides/data/asoiaf-book3-edges.csv' AS row
MERGE (src:Character {name: row.Source})
MERGE (tgt:Character {name: row.Target})
// relationship for the book
MERGE (src)-[r:INTERACTS3]->(tgt)
ON CREATE SET r.weight = toInteger(row.weight), r.book=3
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/neo4j-examples/graphgists/master/bro
wser-guides/data/asoiaf-book45-edges.csv' AS row
MERGE (src:Character {name: row.Source})
MERGE (tgt:Character {name: row.Target})
// relationship for the book
```

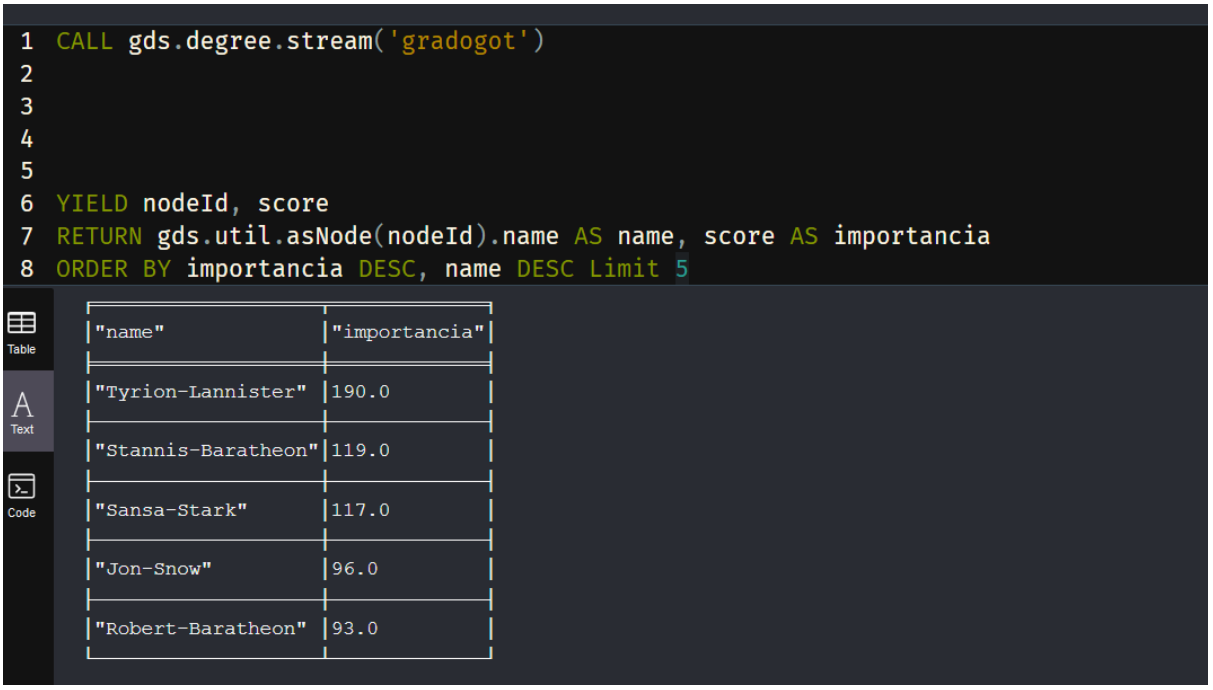
MERGE (src)-[r:INTERACTS45]->(tgt)
ON CREATE SET r.weight = toInteger(row.weight), r.book=45

a) Detectar quiénes son los cinco personajes más relevantes de la saga. Para ello utiliza las medidas de centralidad de grado, cercanía e intermediación

-Grado

```
CALL
gds.graph.project('gradogot','Character',{INTERACTS1:{orientation:'REVERSE'},INTERACT
S2:{orientation:'REVERSE'},INTERACTS3:{orientation:'REVERSE'},INTERACTS45:{orientat
ion:'REVERSE'}})
```

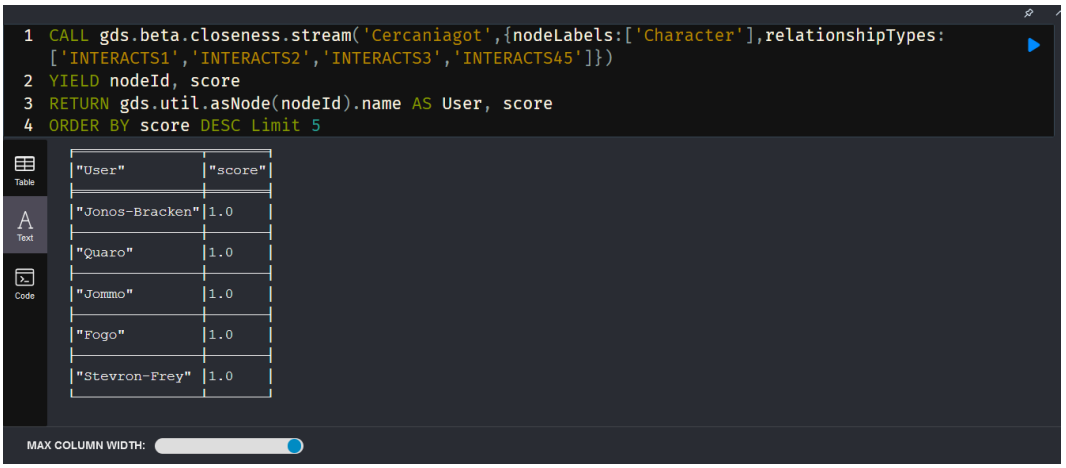
```
CALL gds.degree.stream('gradogot')




YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS importancia
ORDER BY importancia DESC, name DESC Limit 5
```



-Cercanía

```
CALL
gds.graph.project('Cercaniagot','Character',{INTERACTS1:{orientation:'REVERSE'},INTERACTS2:{orientation:'REVERSE'},INTERACTS3:{orientation:'REVERSE'},INTERACTS45:{orientation:'REVERSE'}})
```

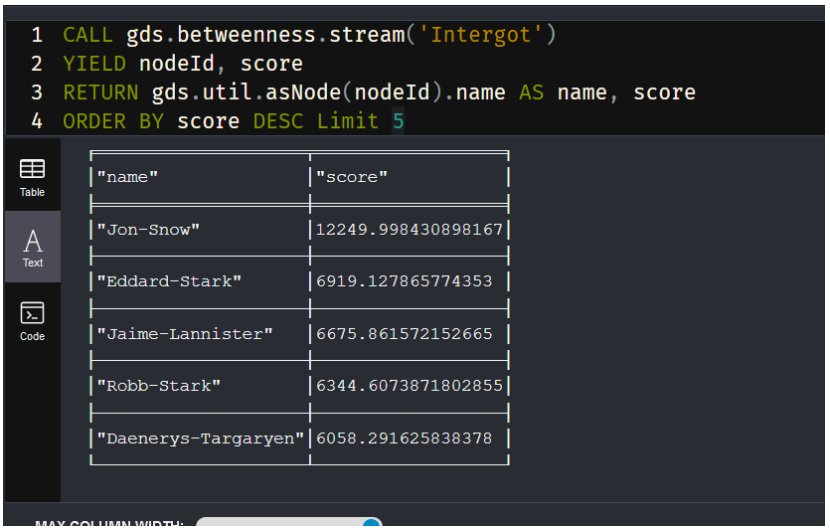
```
CALL
gds.beta.closeness.stream('Cercaniagot',{nodeLabels:['Character'],relationshipTypes:['INTERACTS1','INTERACTS2','INTERACTS3','INTERACTS45']})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS User, score
ORDER BY score DESC Limit 5
```



-Intermediación

```
CALL
gds.graph.project('Intergot','Character',{INTERACTS1:{orientation:'REVERSE'},INTERACTS
2:{orientation:'REVERSE'},INTERACTS3:{orientation:'REVERSE'},INTERACTS45:{orientatio
n:'REVERSE'}})
```

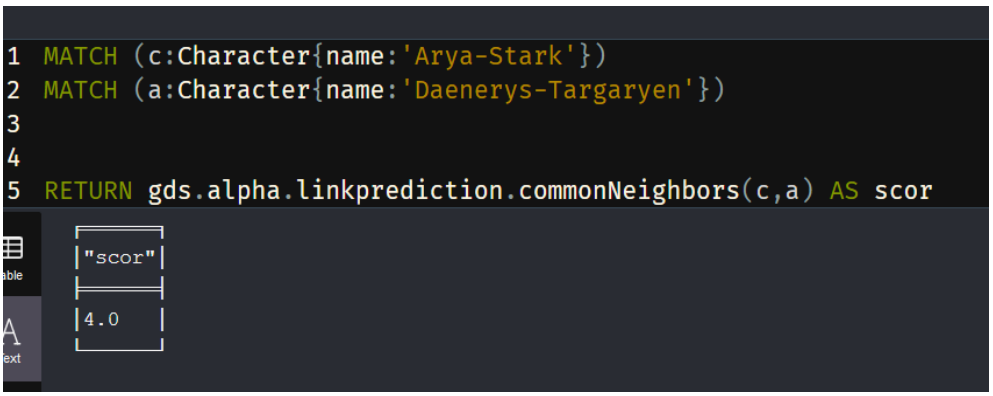
```
CALL gds.betweenness.stream('Intergot')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC Limit 5
```



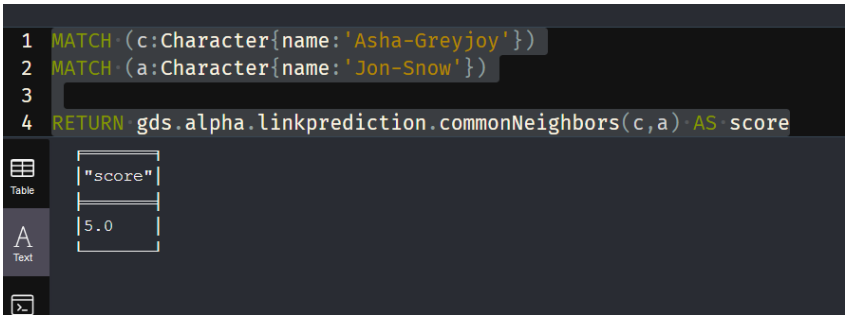
b) Calcular, mediante todos los métodos de predicción de enlace vistos, los valores de posibilidad de que se produzca un nuevo contacto entre Arya-Stark y Daenerys-Targaryen y entre Asha-Greyjoy y Jon-Snow.

- Vecinos comunes

```
MATCH (c:Character{name:'Arya-Stark'})
MATCH (a:Character{name:'Daenerys-Targaryen'})

RETURN gds.alpha.linkprediction.commonNeighbors(c,a) AS score
```

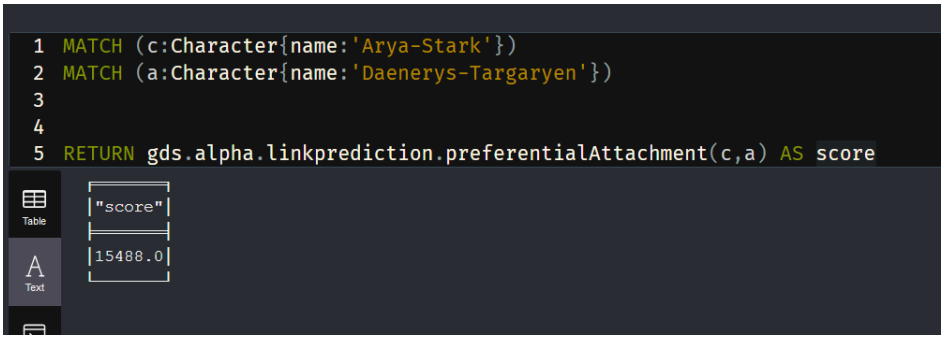



```
MATCH (c:Character{name:'Asha-Greyjoy'})
MATCH (a:Character{name:'Jon-Snow'})

RETURN gds.alpha.linkprediction.commonNeighbors(c,a) AS score
```

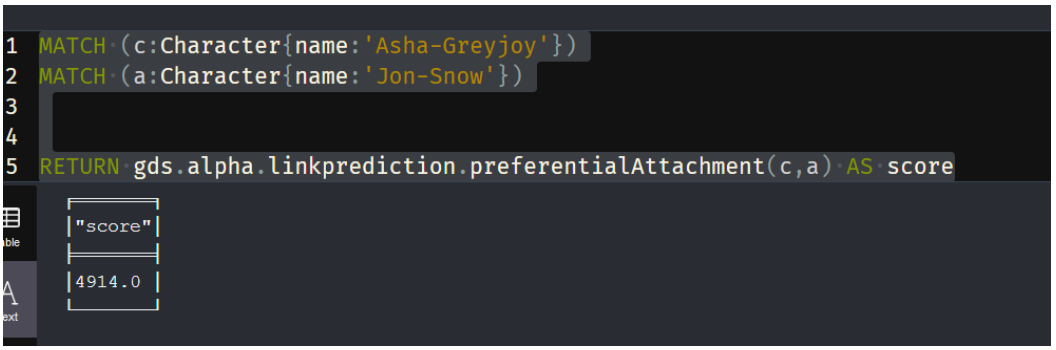


-Adhesión preferencial

```
MATCH (c:Character{name:'Arya-Stark'})
MATCH (a:Character{name:'Daenerys-Targaryen'})


RETURN gds.alpha.linkprediction.preferentialAttachment(c,a) AS score
```

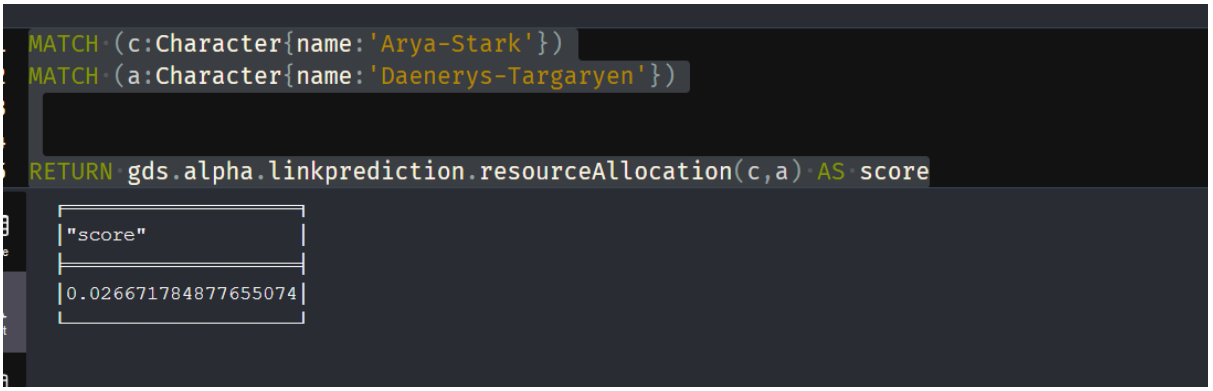



```
MATCH (c:Character{name:'Asha-Greyjoy'})
MATCH (a:Character{name:'Jon-Snow'})


RETURN gds.alpha.linkprediction.preferentialAttachment(c,a) AS score
```

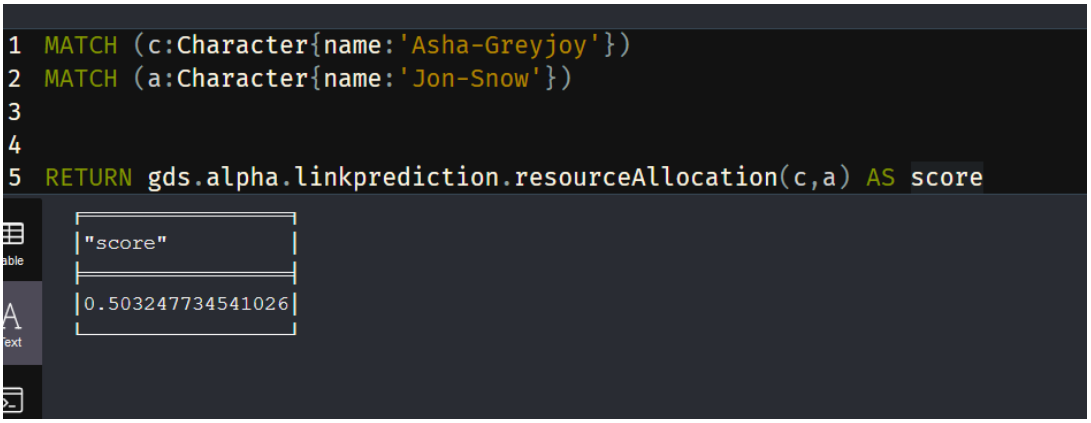


Asignación de recursos

```
MATCH (c:Character{name:'Arya-Stark'})
MATCH (a:Character{name:'Daenerys-Targaryen'})


RETURN gds.alpha.linkprediction.resourceAllocation(c,a) AS score
```




```
MATCH (c:Character{name:'Asha-Greyjoy'})
MATCH (a:Character{name:'Jon-Snow'})


RETURN gds.alpha.linkprediction.resourceAllocation(c,a) AS score
```



Nota: Para buscar un nombre a través de una subcadena usar: match(c:Character) where c.name contains "Snow" return c