



ITESO, Universidad
Jesuita de Guadalajara

Minería de Grafos

Proyecto análisis de GoT s2

Estudiantes:

- Velásquez Borda, Mateo Josué.
- Medina Munguia, Iñaki.

Profesor:

- Ortega Guzmán, Victor Hugo

Fecha de entrega:

- 31 de Marzo de 2025

Índice

Introducción	3
Comprendión del Proceso Analítico	4
Identificar objetivos del Negocio	4
Identificar requisitos de información	4
Identificar métricas exitosas	5
Identificar el plan del proyecto	6
Alcance del proyecto	6
Duración del proyecto	6
Entregables acordados	6
Definir preguntas para la toma de decisiones	6
Construcción del Grafo	7
Selección de formato de almacenamiento de datos	7
Extraer, transformar y cargar datos	7
Análisis básico exploratorio	8
Preguntas exploratorias	9
Respuestas a las preguntas exploratorias	9
Minería de Grafos	15
Seleccionar algoritmos de minería de Grafos	16
Aplicación de algoritmos de centralidad	16
Preguntas de Centralidad	17
Aplicación de los algoritmos de Centralidad	22
Respuesta a las preguntas de Centralidad	29
Aplicación de los algoritmos de comunidad	44
Preguntas de detección de Comunidades	44
Aplicación de los algoritmos de detección de comunidades	49
Respuesta a las preguntas de detección de comunidades	67
Conclusiones y recomendaciones:	84
Conclusiones en referencia a los hallazgos:	84
Recomendaciones:	87
Referencias bibliográficas:	88

Introducción

En la actualidad, el crecimiento exponencial de los datos ha impulsado el desarrollo de nuevas formas de almacenamiento y análisis que permitan descubrir patrones y relaciones complejas. En este contexto, las bases de datos basadas en grafos han emergido como una poderosa herramienta para modelar datos altamente interconectados, proporcionando una representación más intuitiva y eficiente que las bases de datos relacionales tradicionales. Su capacidad para gestionar y analizar estructuras de datos complejas hace que sean especialmente valiosas en diversos dominios de aplicación, como los sistemas de información empresarial, los sistemas de detección de delitos financieros, los sistemas de información de transporte y los sistemas de recomendación [1].

Este proyecto tiene como objetivo principal la construcción y análisis de una base de datos de grafos a partir de información extraída de la segunda temporada de la serie Game of Thrones, utilizando la metodología propuesta por el artículo “A Methodology for Knowledge Discovery in Labeled and Heterogeneous Graphs (KDG)” [1]. Además se utilizaron dos artículos adicionales: en primer lugar el trabajo recepcional de la Ing. Sarahi Partida Ochoa [2], y el artículo de “Analítica de grafos para identificar entidades relevantes y comunidades en Mercado Libre: un estudio de caso” [3] cómo guía en algunas cuestiones prácticas.

A partir de dos archivos CSV que contienen nodos y relaciones, se transformarán los datos en un modelo gráfico que permitirá aplicar algoritmos de análisis de redes. Con esto, se busca identificar patrones clave en la interacción entre los personajes, determinando su nivel de influencia y agrupando comunidades en función de su conectividad dentro del grafo. Además buscaremos, recolectaremos y cargaremos dos archivos CSV extra que contendrán los nodos Casa y las respectivas Relaciones de los personajes con sus casas

Objetivos secundarios:

- ❖ Construir una base de datos basada en grafos a partir de los archivos CSV extraídos del repositorio: <https://github.com/mathbeveridge/gameofthrones/tree/master/data>.
- ❖ Elaborar archivos CSV de las Casas de Juego de tronos y de las Relaciones de los Personajes con sus respectivas casas.
- ❖ Aplicar algoritmos de centralidad para identificar los personajes más influyentes dentro de la red de relaciones de la segunda temporada.
- ❖ Implementar técnicas de detección de comunidades para agrupar personajes y analizar sus interacciones dentro del grafo.
- ❖ Visualizar la estructura y las relaciones de los datos mediante herramientas especializadas en análisis de grafos.

El dominio de los datos utilizados en este estudio provienen del repositorio antes mencionado y corresponden exclusivamente a la segunda temporada de Game of Thrones. En estos archivos se encuentran las interacciones entre personajes, representadas en forma de nodos (personajes) y relaciones (interacciones entre ellos). Estos datos permiten construir una red de relaciones que refleje la dinámica social dentro de la serie, proporcionando una base sólida para el análisis posterior.

Comprensión del Proceso Analítico

Identificar objetivos del Negocio

El objetivo principal del proyecto será el de convertir los archivos CSV encontrados en un repositorio de github (<https://github.com/mathbeveridge/gameofthrones/tree/master/data>) exclusivamente los que corresponden a la segunda temporada de la serie de juego de tronos, para la carga de datos, transformación de los mismos en información mediante la minería de grafos y la visualización de la misma mediante ciertas herramientas.

Se establecen los siguientes objetivos específicos:

- ❖ Construir el modelo del grafo.
- ❖ Generar la base de datos basada en grafos para permitir su posterior manejo de algoritmos de grafos.
- ❖ Realizar el análisis básico exploratorio para comprender la estructura y las propiedades de la base de datos.
- ❖ Realizar el análisis de centralidad para identificar nodos clave.
- ❖ Realizar el análisis de comunidades para implementar técnicas de detección de comunidades para agrupar personajes y analizar sus interacciones dentro del grafo.
- ❖ Documentar los resultados apropiadamente de tal forma que sea lo más conciso posible.
- ❖ Proponer ciertas sugerencias y recomendaciones en la documentación.
- ❖ Presentar los resultados mediante una presentación que permita a cualquier persona que no haya visto la serie entender la dinámica de la misma y los personajes más relevantes de la misma.

Identificar requisitos de información

Para el desarrollo de este proyecto se necesitó averiguar las casas a la que pertenecían cada uno de los miembros del archivo csv “got-s2-nodes.csv”. Para esta misión se utilizaron las páginas web referenciadas en la bibliografía [19] y [20].

Se emplearán las siguientes herramientas:

- Neo4j: Base de datos orientada a grafos que permitirá almacenar y gestionar las conexiones entre los nodos de la red.
- Neo4j Browser: Interfaz gráfica para la carga, consulta y manipulación de los datos dentro de la base de datos.
- Neo4j Bloom: Herramienta de visualización avanzada para explorar la estructura del grafo de manera interactiva.
- Neo4j NeoDash: Plataforma para la creación de dashboards que facilitarán la visualización y el análisis de los datos.
- Gephi: Software de análisis de redes que complementará la visualización y exploración de la estructura del grafo.

Para realizar el análisis del grafo, se implementarán distintos algoritmos en dos áreas clave: centralidad y detección de comunidades.

➤ Algoritmos de centralidad:

- Degree Centrality: Evalúa la cantidad de conexiones directas de un nodo dentro del grafo.

- Closeness Centrality: Determina la proximidad de un nodo con respecto a los demás, considerando su accesibilidad dentro de la red.
- Betweenness Centrality: Identifica los nodos que actúan como puentes entre diferentes partes del grafo.
- PageRank: Calcula la importancia de cada nodo en función de la cantidad y calidad de sus conexiones.
- Articulation Points: Este algoritmo se usa para identificar vulnerabilidades en redes, donde la eliminación de estos puntos podría desconectar partes importantes de la red.
- Bridges: Este algoritmo identifica estas aristas críticas, porque funcionan como “puentes” y en caso de eliminarse podrían desconectar al grafo y dividirlo en subgrafos.

➤ Algoritmos de detección de comunidades:

- Triangle Count: Este algoritmo cuenta la cantidad de triángulos en un grafo, es decir, conjuntos de tres nodos que están todos conectados entre sí.
- Clustering Coefficient: Analiza la densidad de conexiones entre nodos vecinos.
- Strongly Connected Components: Identifica grupos de nodos fuertemente conectados dentro del grafo.
- Weakly Connected Components: Identifica grupos de nodos débilmente conectados dentro del grafo
- Label Propagation: Agrupa los nodos en comunidades basándose en la propagación de etiquetas a través de la red.
- K-1 Coloring: El algoritmo de coloreo k-1 asigna colores a los nodos de un grafo de tal forma que ningún par de nodos adyacentes tengan el mismo color.
- Modularity Metric: La modularidad es una métrica utilizada para evaluar la calidad de una partición de un grafo en comunidades. Un valor alto de modularidad indica que hay una gran densidad de conexiones dentro de las comunidades y pocas conexiones entre diferentes comunidades.
- Louvain Modularity: Detecta comunidades optimizando la modularidad de la estructura del grafo.

La implementación de estos algoritmos permitirá extraer información valiosa sobre la interacción y relevancia de los personajes en la trama, brindando una perspectiva única sobre la estructura social de la serie. A través de este análisis, se podrán revelar patrones ocultos en la narrativa, proporcionando una herramienta útil para el estudio de la base de datos.

Identificar métricas exitosas

En este contexto no se identificaron “métricas exitosas” puesto que dentro de los objetivos no está planteado hacer un análisis para elaborar un plan correctivo, sino que la investigación tiene un enfoque más exploratorio. Por consiguiente, es obsoleto e innecesario plantear métricas exitosas.

Identificar el plan del proyecto

Este será el documento del proyecto. Por consiguiente, definimos los siguientes puntos:

Alcance del proyecto

Alcance del proyecto: este proyecto se centrará exclusivamente en la segunda temporada de Game of Thrones, utilizando los datos disponibles en el repositorio de GitHub: <https://github.com/mathbeveridge/gameofthrones/tree/master/data>. La base de datos incluirá personajes, casas y sus relaciones, representando cada personaje y cada casa como un nodo y cada interacción entre los personajes y pertenencia a las casas como una relación en el grafo.

Duración del proyecto

La duración estimada de este proyecto abarca desde las 13:00 del jueves 20 de marzo hasta las 13:00 del martes 31 de marzo. Lo que nos deja un aproximado de 11 días.

Entregables acordados

Para esta actividad se acordaron los siguientes entregables:

- Archivo DUMP de la base de datos.
- Archivo PDF que sirva como un informe del proyecto (este documento).
- Archivo JSON del tablero creado en NeoDash
- Archivo CQL utilizado para escribir el código necesario para cada una de las consultas a la base de datos.
- Archivos CSV utilizados para crear la base de datos.
- Presentación de power point para difundir y compartir los descubrimientos, resultados y conclusiones obtenidos en el proyecto.

Definir preguntas para la toma de decisiones

El objetivo principal de la elaboración de estas preguntas es facilitar la recopilación de información relevante, evaluar las opciones disponibles y considerar los factores que influyen en la toma de decisiones bien informada y efectiva. [1]

En este apartado, se presentan una serie de preguntas de interés general que serán abordadas a lo largo de todo el proyecto de investigación:

- ¿Cuáles son los cinco personajes más importantes de la serie en la temporada 2?
- ¿Cuáles son los cinco personajes menos importantes de la serie en la temporada 2?
- Dado un personaje específico, ¿con cuáles otros le conviene formar una alianza?
- ¿Existen bandos armados? Si los hay, ¿son realmente fuertes o presentan debilidades?
- ¿Cómo afectarían las alianzas si se eliminara a cierto personaje?
- ¿Hay alguna estrategia óptima para ganar en Juego de Tronos?
- ¿Qué casas son las más poderosas en Juego de Tronos?
- ¿Qué habría que hacer para que la posición de las alianzas cambie?

Estas preguntas no tienen como propósito ser respondidas mediante consultas simples, sino que servirán como eje para guiar el desarrollo del proyecto a lo largo de toda la

investigación. Lo ideal es que sean resueltas a partir de la combinación de respuestas a interrogantes más específicas.

Por ejemplo, la pregunta "¿Cuáles son los cinco personajes más importantes?" es, en cierto sentido, subjetiva, ya que requiere definir el criterio de importancia en función de distintos factores. A lo largo de este documento, se abordará esta y otras cuestiones con el fin de construir respuestas fundamentadas.

Construcción del Grafo

Ahora basándonos en la metodología previamente establecida deberíamos seguir con los siguientes puntos: (2.1: Análisis exploratorio de datos) que implica realizar un análisis exploratorio de la información, y (2.2: Diseño de modelo de grafos) en dónde formulamos el modelo de grafos [1]. Sin embargo, la información del repositorio ya nos brinda todos los datos necesarios en dos archivos CSV. Por consiguiente es innecesario realizar los puntos 2.1 y 2.2 de la metodología. Así que continuamos directamente con el punto (2.3).

Selección de formato de almacenamiento de datos

En el paso (2.3), se selecciona el formato de almacenamiento más adecuado. Para este proyecto, es fundamental aprovechar las capacidades de una base de datos basada en grafos que almacene la información de forma nativa, ya que este enfoque resulta más eficiente para el análisis de los archivos CSV de la serie de Juegos de Tronos. En este caso, se ha optado por utilizar Neo4j, un sistema de gestión de bases de datos orientado a grafos de código abierto, implementado en Java.

Extraer, transformar y cargar datos

En esta parte del proceso extraer la información implicó descargar el repositorio de github: <https://github.com/mathbeveridge/gameofthrones/tree/master/data>. La trasformación de los datos simplemente fue la selección de los datos específicos de la temporada 2 de Juego de tronos. Y por último la carga de datos consistió en mover los archivos CSV a la carpeta Import de la base de datos "GOT-S2" en el Software de NEO4J; en dónde se aplicaron las siguientes consultas en el Neo4j Browser:

```
//cargar los nodos
LOAD CSV WITH HEADERS FROM 'file:///got-s2-nodes.csv' AS row
CREATE (:Personaje {
    id: row.Id,
    name: row.Label
}) ;

//comprobar
MATCH (p:Personaje)
RETURN p.id, p.name;

//cargar los relaciones
```

```
LOAD CSV WITH HEADERS FROM 'file:///got-s2-edges.csv' AS row
MATCH (source:Personaje {id: row.Source}),
      (target:Personaje {id: row.Target})
CREATE (source)-[:HABLA_CON {conversaciones: toInteger(row.Weight),
                           season: toInteger(row.Season)}]->(target);

//comprobar
MATCH (p:Personaje)-[r:HABLA_CON]->(p2:Personaje)
RETURN p, r, p2;
```

Por otro lado posteriormente al procesamiento de estos archivos CSV que se nos brindaron fue necesario agregar dos archivos CSV extras que poseían la información de las casas de cada uno de los Personajes (si es que la tienen), para la temporada 2 de Juego de Tronos. A continuación adjuntamos el código de Cypher para procesar los archivos:

```
//Cargar las casas:
LOAD CSV WITH HEADERS FROM 'file:///got-s2-casas.csv' AS row
CREATE (:Casa {
    id: toInteger(row.Id),
    name: row.Label
}) ;

//comprobar
MATCH (c:Casa)
RETURN c.id, c.name;

//cargar los relaciones
LOAD CSV WITH HEADERS FROM 'file:///got-s2-Personajes-casa.csv' AS row
MATCH (source:Personaje {id: row.IdPersonaje}),
      (target:Casa {name: row.House})
CREATE (source)-[:ES_LEAL_A]->(target);

//comprobar
MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
RETURN p,r,c;
```

Análisis básico exploratorio

Dado que el proyecto consiste en analizar archivos CSV previamente cargados en un repositorio de GitHub, es fundamental formular una serie de preguntas para comprender la estructura y el funcionamiento de la base de datos basada en grafos con la que trabajaremos. Estas preguntas nos permitirán generar una mejor noción sobre la estructura de la base de datos, documentar nuestros hallazgos y registrar las respuestas obtenidas durante el proceso de exploración y análisis.

Preguntas exploratorias

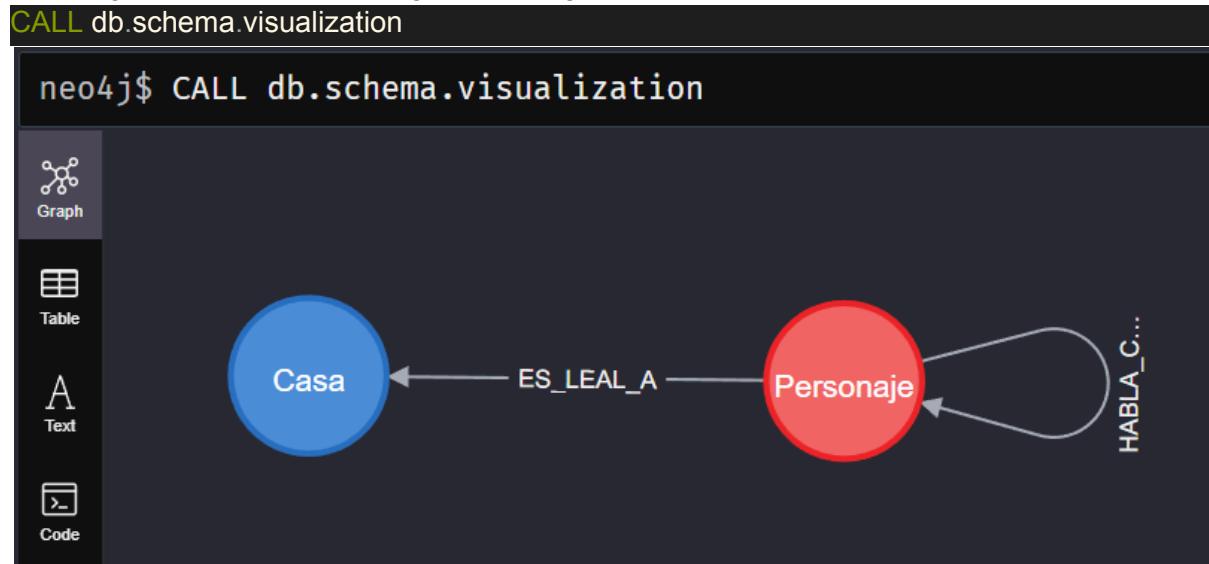
Las preguntas exploratorias planteadas son las siguientes:

1. ¿Cómo es el esquema general del grafo?
2. ¿Cuáles tipos de nodos existen y que atributos tienen?
3. ¿Cuáles tipos de relaciones existen y que atributos tienen?
4. ¿Cuántos Personajes existen?
5. ¿Cuántas Casas existen?
6. ¿Cuántas relaciones "HABLA_CON" existen?
7. ¿Cuántas relaciones "ES_LEAL_A" existen?
8. ¿Cuál es el promedio de conversaciones?
9. ¿Cuál es la casa con menos gente leal?
10. ¿Quién es el personaje con más la mayor cantidad de conversaciones como atributo de HABLA_CON?
11. ¿Cuál es la casa a la que más gente le es Leal?
12. ¿Cuál es el diámetro del grafo?
13. ¿Cuál es la densidad del grafo?
14. ¿Cuál es la ruta más corta entre Jon y Cersei?
15. ¿Cuántas personas han hablado con un determinado personaje?
16. ¿Cuál es el promedio de conversaciones en una persona?
17. ¿Cuántas conversaciones tiene 'DAENERYS'?
18. ¿Cuál es el nodo con mayor número de relaciones?
19. ¿Cuántos miembros tiene cada casa?
20. ¿Cuántas personas no son leales a una casa en Juego de Tronos?

Respuestas a las preguntas exploratorias

Las respuestas a las preguntas exploratorias planteadas son las siguientes:

1. ¿Cómo es el esquema general del grafo?



2. ¿Cuáles tipos de nodos existen y que atributos tienen?

CALL db.schema.nodeTypeProperties

nodeType	nodeLabels	propertyName	propertyTypes	mandatory
1 "":Casa"	["Casa"]	"name"	["String"]	true
2 "":Casa"	["Casa"]	"id"	["Long"]	true
3 "":Personaje"	["Personaje"]	"name"	["String"]	true
4 "":Personaje"	["Personaje"]	"Id"	["String"]	true

3. ¿Cuáles tipos de relaciones existen y que atributos tienen?

`CALL db.schema.relTypeProperties`

relType	propertyName	propertyTypes	mandatory
1 "":HABLA_CON"	"conversaciones"	["Long"]	true
2 "":HABLA_CON"	"season"	["Long"]	true
3 "":ES_LEAL_A"	null	null	false

4. ¿Cuántos Personajes existen?

`MATCH (p:Personaje)`

`RETURN COUNT(p.id) AS Total_Personajes;`

Total_Personajes
129

5. ¿Cuántas Casas existen?

`MATCH(c:Casa)`

`RETURN COUNT(c.id) AS Total_Casas;`

Total_Casas
29

6. ¿Cuántas relaciones "HABLA_CON" existen?

`MATCH (p)-[r:HABLA_CON]->(k)`

`RETURN COUNT(r) AS Total_HABLA_CON;`

Total_HABLA_CON
486

7. ¿Cuántas relaciones "ES_LEAL_A" existen?

```
MATCH (p)-[r:ES_LEAL_A]->(k)
RETURN COUNT(r) AS Total_ES_LEAL_A;
```

```
1 MATCH (p)-[r:ES_LEAL_A]->(k)
2 RETURN COUNT(r) AS Total_ES_LEAL_A;
```



Table

Total_ES_LEAL_A



1

82

8. ¿Cuál es la suma total de conversaciones?

```
MATCH (p)-[r:HABLA_CON]->(k)
```

```
RETURN SUM(r.conversaciones) AS Total_conversaciones;
```

```
1 MATCH (p)-[r:HABLA_CON]->(k)
2 RETURN SUM(r.conversaciones) AS Total_conversaciones;
```



Table

Total_conversaciones



1

6166

9. ¿Cuál es el promedio de conversaciones?

```
MATCH (p)-[r:HABLA_CON]->(k)
```

```
RETURN AVG(r.conversaciones) AS Promedio_Conversaciones;
```

```
1 MATCH (p)-[r:HABLA_CON]->(k)
2 RETURN AVG(r.conversaciones) AS Promedio_Conversaciones;
```



Table

Promedio_Conversaciones



1

12.687242798353912

10. ¿Quiénes son los 5 personajes con más la mayor cantidad de conversaciones como atributo de HABLA_CON?

```
MATCH (p:Personaje)-[r:HABLA_CON]->(k:Personaje)
```

```
WITH p.id AS ID, COUNT(r) AS Cantidad_conversaciones
```

```
RETURN ID, Cantidad_conversaciones
```

```
ORDER BY Cantidad_conversaciones DESC
```

```
LIMIT 5;
```

```

1 MATCH (p:Personaje)-[r:HABLA_CON]→(k:Personaje)
2 WITH p.id AS ID, COUNT(r) AS Cantidad_conversaciones
3 RETURN ID, Cantidad_conversaciones
4 ORDER BY Cantidad_conversaciones DESC
5 LIMIT 5;

```

	ID	Cantidad_conversaciones
1	"CERSEI"	28
2	"ARYA"	25
3	"JOFFREY"	23
4	"CATELYN"	19
5	"BRONN"	15

11. ¿Cuáles son las casas que más gente leal tiene?

```

MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
WITH c.id AS ID, c.name AS nombre_casa, COUNT(r) AS Cantidad_Leales
RETURN ID, nombre_casa, Cantidad_Leales
ORDER BY Cantidad_Leales DESC
LIMIT 5;

```

	ID	nombre_casa	Cantidad_Leales
1	22	"Stark"	13
2	14	"Lannister"	11
3	3	"Baratheon"	8
4	24	"Targaryen"	6
5	10	"Greyjoy"	5

12. ¿Cuántas Relaciones hay en Total?

```

MATCH (p)-[r]->(k)
RETURN COUNT(r) AS Total_relaciones;

```

```
1 MATCH (p)-[r]-(k)
2 RETURN COUNT(r) AS Total_relaciones;
```

Table

Total_relaciones

	1
568	

A
Text

13. ¿Cuál es el diámetro del grafo?

Diámetro de la red

6 Ejecutar

Esto señala que necesito como máximo 5 saltos para llegar desde cualquier nodo de la red a otro nodo de la red (fue realizado)

14. ¿Cuál es la densidad del grafo?

Densidad de grafo

0,023 Ejecutar

Esto señala que el grafo es muy poco denso. Lo cuál tiene sentido, porque existen 158 nodos en la red y sólo 568 aristas (Resultado de sumar el total de cada cantidad de nodos y cada cantidad de relaciones). Entonces cada nodo en promedio tiene entre 4 y 5 relaciones, lo cuál es muy bajo.

15. ¿Cuántas personas han hablado con un determinado personaje?

```
MATCH
(p:Personaje{id:$neodash_personaje_id})-[r:HABLA_CON]-(p2:Personaje)
RETURN p.id AS ID,
       p.name AS Nombre,
       COLLECT(p2.name) AS Lista_Personas_Habladas;
```

Selección ID

Personaje id

DAVOS

Relaciones por ID

ID	Nombre	Lista
DAVOS	Davos	Father Seaworth, Joffrey, Loras, Myra, Matthos, Melisandre, Renly, Robb, Robert, Salladhor, Stannis

Rows per page: 5 1-1 of 1 < >

16. ¿Cuál es el promedio de conversaciones en una persona?

```
MATCH
(p:Personaje{id:$neodash_personaje_id_2})-[r:HABLA_CON]-(p2:Personaje)
RETURN p.id AS ID,
       p.name AS Nombre,
       AVG(r.conversaciones) AS Promedio_Conversaciones;
```

ID	Promedio_Conversaciones
JON	40.167

Rows per page: 5 1–2 of 2

17. ¿Cuántas conversaciones tiene 'DAENERYS'?

```

MATCH (p:Personaje{id:'DAENERYS'})-[r:HABLA_CON]->(p2:Personaje)
RETURN p.id AS ID,
       p.name AS Personaje,
       SUM(r.conversaciones) AS Total_conversaciones;
    
```

```

1 MATCH (p:Personaje{id:'DAENERYS'})-[r:HABLA_CON]→(p2:Personaje)
2 RETURN p.id AS ID,
3        p.name AS Personaje,
4        SUM(r.conversaciones) AS Total_conversaciones;
    
```

	ID	Personaje	Total_conversaciones
1	"DAENERYS"	"Daenerys"	383

18. ¿Cuál es el nodo con mayor número de relaciones?

```

MATCH (n)-[r]-(m)
RETURN n.id AS ID,
       COUNT(r) AS Total_relaciones
ORDER BY Total_relaciones DESC
LIMIT 1;
    
```

```

1 MATCH (n)-[r]-(m)
2 RETURN n.id AS ID,
3        COUNT(r) AS Total_relaciones
4 ORDER BY Total_relaciones DESC
5 LIMIT 1;
    
```

	ID	Total_relaciones
1	"JOFFREY"	37

19. ¿Cuántos miembros tiene cada casa?

```
MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
WITH c.id AS ID_Casa,
    c.name AS Casa,
    COUNT(p) AS Total_Miembros
RETURN ID_Casa, Casa, Total_Miembros
ORDER BY Total_Miembros DESC;
```

```
1 MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
2 WITH c.id AS ID_Casa,
3     c.name AS Casa,
4     COUNT(p) AS Total_Miembros
5 RETURN ID_Casa, Casa, Total_Miembros
6 ORDER BY Total_Miembros DESC;
```

	ID_Casa	Casa	Total_Miembros
1	22	"Stark"	13
2	14	"Lannister"	11
3	3	"Baratheon"	8
4	24	"Targaryen"	6
5	10	"Greyjoy"	5
6	7	"Clegane"	4
7			

20. ¿Cuántas personas no son leales a una casa en Juego de Tronos?

```
MATCH (p:Personaje)
WHERE NOT EXISTS { (p)-[:ES_LEAL_A]->(:Casa) }
RETURN COUNT(p) AS Personajes_sin_lealtad;
```

```
1 MATCH (p:Personaje)
2 WHERE NOT EXISTS { (p)-[:ES_LEAL_A]->(:Casa) }
3 RETURN COUNT(p) AS Personajes_sin_lealtad;
```

	Personajes_sin_lealtad
1	47

Minería de Grafos

La minería de grafos es una técnica potente que permite analizar diversas propiedades de los grafos, predecir sus estructuras y relaciones, y modelar patrones presentes en grafos reales [1].

En este apartado presentaremos un procedimiento para realizar un exhaustivo análisis de centralidad y de comunidades.

Seleccionar algoritmos de minería de Grafos

Para realizar el análisis del grafo, se implementarán distintos algoritmos en dos áreas clave: centralidad y detección de comunidades.

- Algoritmos de centralidad:
 - Degree Centrality: Evalúa la cantidad de conexiones directas de un nodo dentro del grafo.
 - Closeness Centrality: Determina la proximidad de un nodo con respecto a los demás, considerando su accesibilidad dentro de la red.
 - Betweenness Centrality: Identifica los nodos que actúan como puentes entre diferentes partes del grafo.
 - PageRank: Calcula la importancia de cada nodo en función de la cantidad y calidad de sus conexiones.
 - Articulation Points: Este algoritmo se usa para identificar vulnerabilidades en redes, donde la eliminación de estos puntos podría desconectar partes importantes de la red.
 - Bridges: Este algoritmo identifica estas aristas críticas, porque funcionan como “puentes” y en caso de eliminarse podrían desconectar al grafo y dividirlo en subgrafos.
- Algoritmos de detección de comunidades:
 - Triangle Count: Este algoritmo cuenta la cantidad de triángulos en un grafo, es decir, conjuntos de tres nodos que están todos conectados entre sí.
 - Clustering Coefficient: Analiza la densidad de conexiones entre nodos vecinos.
 - Strongly Connected Components: Identifica grupos de nodos fuertemente conectados dentro del grafo.
 - Weakly Connected Components: Identifica grupos de nodos débilmente conectados dentro del grafo
 - Label Propagation: Agrupa los nodos en comunidades basándose en la propagación de etiquetas a través de la red.
 - K-1 Coloring: El algoritmo de coloreo k-1 asigna colores a los nodos de un grafo de tal forma que ningún par de nodos adyacentes tengan el mismo color.
 - Modularity Metric: La modularidad es una métrica utilizada para evaluar la calidad de una partición de un grafo en comunidades. Un valor alto de modularidad indica que hay una gran densidad de conexiones dentro de las comunidades y pocas conexiones entre diferentes comunidades.
 - Louvain Modularity: Detecta comunidades optimizando la modularidad de la estructura del grafo.

Aplicación de algoritmos de centralidad

En este apartado, establecemos todos los elementos necesarios para aplicar los algoritmos de centralidad de manera eficiente y efectiva. Antes de determinar qué algoritmos serán útiles, es fundamental formular preguntas cuya resolución dependa de estos métodos. Por ello, esta sección comienza con la definición de dichas preguntas.

Preguntas de Centralidad

Pregunta Técnica	Pregunta de Estrategia
¿Cuáles son los personajes con mayor Degree en el grafo?	<p>¿Cuáles son los personajes con mayor cantidad de conversaciones de la temporada 2?</p> <p>Esta pregunta es clave para la dinámica de Juego de Tronos porque es importante conocer la importancia que tiene cierto personaje en la trama de juego de tronos. Las personas con más conversaciones probablemente sean las que más relaciones tengan o que más fuertes las conservan, lo que los hace Personajes extremadamente relevantes de Juego de tronos.</p>
¿Cuáles son los Personajes con menor Degree en el grafo?	<p>¿Cuáles son los personajes con menos conversaciones de la temporada 2?</p> <p>Esto es interesante de ver, porque podría darnos a las personas más incomunicadas de Juego de tronos, y eso podría ser útil de saber porqué probablemente sean posibles futuras conquistas de alguno de los bandos, dado que son los Personajes más incomunicados y por consiguiente los más fáciles de matar sin que pidan ayuda a sus pocos aliados.</p>
¿Cuál es la distribución del Degree de los Personajes en la red?	<p>¿Existen personajes que tengan una elevada cantidad de conversaciones con respecto al resto? ¿Todos tienen cantidades parecidas de conversaciones?</p> <p>Esta pregunta es de vital importancia para la trama, puesto que nos indica que tan marcada está la distribución de los poderes en Juego de tronos. Dado que si hay poca gente con muchas conversaciones, entonces es muy probable que el poder esté muy centralizado en la temporada 2, y sería interesante analizar cómo esto cambió en temporadas posteriores</p>
¿Cuáles son aquellos personajes con un mayor Closeness Centrality?	<p>¿Quiénes son las 5 personas con más facilidad de transmitir información? ¿Qué personaje está mejor posicionado para actuar como consejero real, considerando su acceso rápido a la mayor cantidad de otros personajes importantes? ¿Qué</p>

	<p>personajes resultan útiles para hacer que comparten información falsa en caso de querer hacer una emboscada o trampa?</p> <p>Es de vital importancia conocer aquellos personajes en los que la información viaja más rápido, puesto que el grafo en sí está hecho acerca de las comunicaciones que hay en Juego de tronos. Y conocer quienes son los que más rápido comunican la información es clave en las guerras.</p>
¿Cuáles son aquellos personajes con un menor Closeness Centrality?	<p>¿Qué personajes están más alejados del resto, lo que podría hacerlos menos influyentes? ¿Cuáles son los personajes que son más óptimos para sacrificar puesto que no representan un gran sacrificio en la red?</p> <p>Esta pregunta sería muy estratégica para los líderes de las alianzas, puesto que resulta útil saber a quienes es “menos arriesgado” perder en caso de tener que sacrificar a alguien</p>
¿Cuáles son aquellos personajes con un alto Betweenness Centrality?	<p>¿Qué personajes actúan como intermediarios clave entre distintos grupos de la red? ¿Qué personajes podrían ser eliminados para fragmentar la red en grupos desconectados? Si se quisiera planear una traición estratégica, ¿qué personaje sería el mejor objetivo para aislar y cortar la comunicación entre diferentes facciones?</p> <p>Estas tres preguntas son extremadamente relevantes para la estrategia de Juego de tronos, por lo tanto responder esta pregunta es clave y pilar fundamental de esta investigación, dado que conocer qué personajes son los intermediarios clave es de vital importancia para saber cómo quedarían los bandos en caso de que alguno de estos muriera</p>
¿Cuáles son aquellos personajes con un alto Degree pero que no tienen un alto Betweenness Centrality?	<p>¿Qué personajes tienen una gran cantidad de Conexiones pero no son considerados intermediarios clave entre los distintos grupos? ¿Cuáles son los posibles líderes y/o referentes de cada grupo que están en contra de estar en Paz?</p> <p>Si bien puede ser que haya personajes que tengan muchas relaciones y además sean</p>

	<p>intermediarios clave entre los distintos grupos, resulta relevante saber aquellos que no son intermediarios entre los otros grupos y además tienen muchas relaciones, porque serían personajes que son poderosos pero no están interesados en comunicarse con otros bandos. Por consiguiente, son las personas más propensas a querer una guerra.</p>
¿Cómo es la distribución del Betweenness centrality?	<p>¿Existe una gran cantidad de personajes intermediarios clave?</p> <p>Esto es interesante de responder porque significa que en caso de haber muchos personajes intermediarios clave el grafo estaría muy conectado, lo cual significa que la temporada 2 dentro de todo sería algo pacífica, pero en caso de haber pocos, significa que las comunicaciones están muy divididas y dependen de pocos intermediarios.</p>
¿Cuáles son aquellos personajes con mayor PageRank?	<p>¿Qué personajes tienen la mayor influencia dentro de la red, considerando la importancia de sus conexiones?</p> <p>Si un personaje fuera coronado rey basado en su influencia en la red, ¿quién tendría la mayor legitimidad y apoyo?</p> <p>Ambas preguntas se responden con la misma pregunta técnica, pero en resumidas cuentas la respuesta nos da una comprensión de quienes son muy políticamente influyentes en la temporada 2 de Juego de tronos.</p>
¿Cuáles son aquellos personajes con un bajo Degree pero con un alto PageRank?	<p>¿Existe algún personaje con relativamente pocas conexiones, pero que aún así sea muy influyente debido a la importancia de sus vínculos? ¿Existe algún personaje con alta influencia pero que aún no tenga un rol de liderazgo explícito debido a que no tiene tantas conexiones, lo que sugiere que podría ser un candidato inesperado al poder?</p> <p>Estas preguntas se responden con la misma pregunta técnica, porque aunque significan cosas distintas en la saga, a nivel técnico se responden con el mismo algoritmo. Responder estas preguntas sería clave para tratar de predecir si en las temporadas posteriores dichos personajes</p>

	si se volvieron relevantes
¿Cuáles son aquellos personajes que tienen un alto Degree pero un bajo Page Rank?	<p>¿Existen personajes que, si bien tienen muchas conexiones, tienen menor influencia en comparación con otros con menos conexiones?</p> <p>Esta pregunta es interesante, por que si un personaje tiene contacto con muchos otros personajes pero que no son tan relevantes, dado que tiene un bajo Pagerank, significa que tiene potencial para ser un buen aliado uniéndose a alguien que ya tenga un buen PageRank, generando un ejercito poderoso</p>
¿Cuáles son los personajes con menor PageRank?	<p>¿Cuáles son los personajes con contactos menos poderosos?</p> <p>Esta información podría ser útil para identificar a los que no son aliados óptimos, puesto que sus conexiones son poco relevantes, entonces en una guerra o batalla estarían prácticamente sólos</p>
Utilizando Articulation Point ¿Cuáles son aquellos personajes cuyo Articulation Point es True?	<p>¿Qué personajes habría que matar para que cuya eliminación fragmentara la red de Personajes pudiendo romper alianzas y generar guerras?</p> <p>Esta pregunta es clave para la dinámica, puesto que permite identificar Personajes importantes, tal que si los eliminamos podríamos debilitar alianzas enteras o terminar con la paz generando una guerra.</p>
Utilizando Bridge ¿Cuáles son las relaciones frágiles en la red que, si se eliminan, fragmentarían la comunidad?	<p>¿Cuáles son las relaciones clave entre personajes que, si se eliminan, podrían debilitar o dividir la estructura de la red?</p> <p>Esta pregunta es clave para la dinámica, puesto que es parecida a la que planteamos anteriormente sobre “eliminar a una persona”, pero esta no necesitaría eliminar a nadie, sino que está más enfocada en la pregunta “¿Qué relaciones diplomáticas habría que romper para debilitar a una alianza?”</p>
¿Cuál es la información de determinado personaje dado su ID?	<p>¿Qué tan importante es un personaje en base a sus métricas? ¿Qué tan caótico sería si lo mataran o desterraran y perdiera el contacto con todos?</p> <p>Esta pregunta es clave para la dinámica de</p>

	Juego de Tronos porque es importante conocer la importancia que tiene cierto personaje en la trama de juego de tronos, y cómo la importancia se puede medir con las métricas de los algoritmos de centralidad, entonces es importante tener información de cada personaje individual.
¿A qué Casa pertenecen los 10 Personajes con mayor Degree?	¿A qué casas pertenecen los Personajes con más conversaciones? Esta pregunta aunque sea muy básica es crucial para conocer cuales son aquellas casas que mejor comunicación tienen en la temporada, y por lo tanto, son casas con fuertes vínculos políticos
¿A qué casa pertenecen los 10 personajes con mayor Closeness Centrality?	¿A qué casas pertenecen los personajes centrales de la temporada? Responder esta pregunta puede darnos una idea de quienes son las casas que mejor están posicionadas para compartir y difundir información en caso de que haya una estrategia de falsa información.
¿A qué casa pertenecen los 10 personajes con mayor PageRank?	¿A qué casas pertenecen los personajes con mejores conexiones? Esta pregunta es interesante porque si vemos que hay varias casas repetidas eso nos puede indicar que más que haber personajes con buenas conexiones las casas en sí tienen buenas conexiones, lo que nos podría indicar que casas son estratégicamente poderosas en Juego de Tronos.
¿Qué personaje es el que tiene un alto Betweenness centrality entre distintas casas?	¿Qué personaje es el mejor intermediario entre diferentes casas y facciones, y qué impacto tendría si cambiara de lealtad? Esta pregunta es vital, porque en caso de querer iniciar una guerra podría bastar con eliminar a un personaje que sea el que conecte diversas casas para romper alianzas de distancia.
¿Cuáles son los Personajes con mayor PageRank por casa?	¿Cuáles son los personajes políticamente más fuertes de cada casa? Si quisiéramos debilitar una casa ¿A quien tendríamos que matar primero? Esta pregunta en el contexto de Juego de

	tronos puede ser útil para buscar estrategias, y saber a que personas es conveniente eliminar, más si son casas que no son tan fuertes, entonces con eliminar a sus líderes es suficiente para que se arrodillen ante alguna otra casa
--	--

Aplicación de los algoritmos de Centralidad

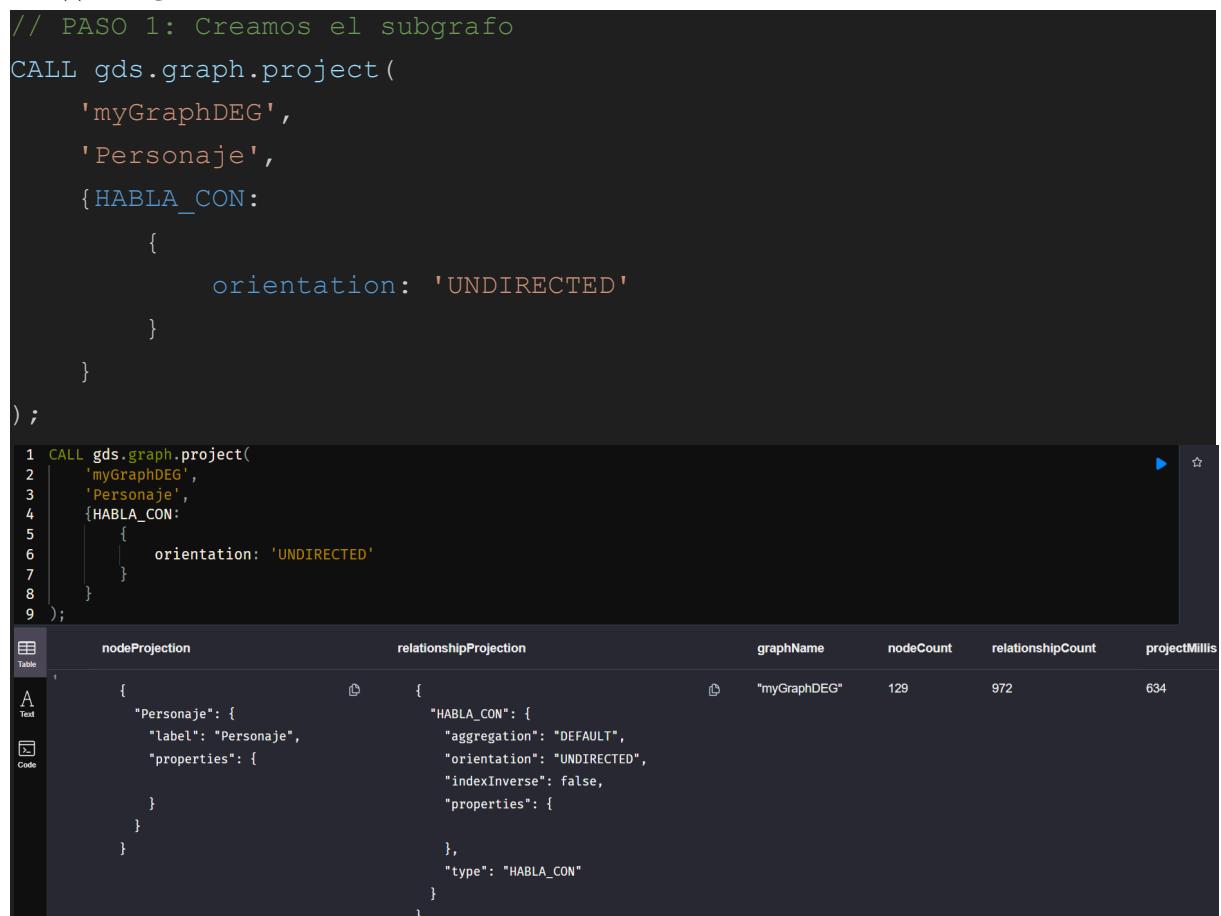
Si bien la metodología que vamos a seguir incluye la etapa (4.2) "Incluir atributos estructurados", en este caso es más conveniente agregar dichos atributos en una etapa anterior, antes de responder las preguntas de centralidad. Por lo tanto dedicamos toda esta sección a incluir los atributos a los nodos:

★ DEGREE CENTRALITY:

```
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphDEG',
    'Personaje',
    {HABLA_CON:
        {
            orientation: 'UNDIRECTED'
        }
    }
);

// PASO 2: Calculo la memoria necesaria:
CALL gds.degree.write.estimate(
    'myGraphDEG', {writeProperty: 'degree'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.degree.stream(
```



nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
<pre> { "Personaje": { "label": "Personaje", "properties": { ... } } } </pre>	<pre> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { ... }, "type": "HABLA_CON" } } </pre>	"myGraphDEG"	129	972	634

```
'myGraphDEG'
)

YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, gds.util.asNode(nodeId).name
AS ciudad_Aeropuerto, score
ORDER BY score DESC, Aeropuerto ASC;

// Paso 4: Implementamos
CALL gds.degree.write('myGraphDEG', { writeProperty: 'degree' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN      centralityDistribution.min      AS minimumScore,
centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;

1 CALL gds.degree.write(['myGraphDEG', { writeProperty: 'degree' }])
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
4 centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

	minimumScore	meanScore	maxScore	nodePropertiesWritten
A	1.0	7.534905071406401	36.00024414062499	129

★ CLOSENESS CENTRALITY:

```
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Personaje',
    'HABLA_CON'
);

1 CALL gds.graph.project(
2   'myGraphCC',
3   'Personaje',
4   'HABLA_CON'
5 );
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A	<code>{ "Personaje": { "label": "Personaje", "properties": {} } }</code>	<code>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": {} }, "type": "HABLA_CON" }</code>	"myGraphCC"	129	486	17

```
// PASO 2: calcular la memoria
CALL gds.degree.write.estimate(
    'myGraphCC', { writeProperty: 'closenessCentrality' }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
```

```

CALL gds.pageRank.stream(
    'myGraphCC'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessCentrality'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
       centralityDistribution.max AS maxScore,
       centralityDistribution.mean AS meanScore,
       nodePropertiesWritten;
1 CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessCentrality'})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimumScore,
4       centralityDistribution.max AS maxScore,
5       centralityDistribution.mean AS meanScore,
6       nodePropertiesWritten;

```

	minimumScore	maxScore	meanScore	nodePropertiesWritten
A Text	0.0	1.000007629394531	0.46118374018706093	129

★ BETWEENNESS CENTRALITY:

```

// PASO 1:
CALL gds.graph.project(
    'myGraphBC',
    'Personaje',
    'HABLA_CON'
);
1 CALL gds.graph.project(
2   'myGraphBC',
3   'Personaje',
4   'HABLA_CON'
5 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre>{ "Personaje": { "label": "Personaje", "properties": {} } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": {} }, "type": "HABLA_CON" }</pre>	"myGraphBC"	129	486	21

```

// PASO 2:
CALL gds.betweenness.write.estimate(
    'myGraphBC', {writeProperty: 'betweennessCentrality'})

```

```

YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3:
CALL gds.betweenness.stream('myGraphBC')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS name, score
ORDER BY score DESC, name ASC;

// PASO 4: Escribimos el resultado como un atributo extra en cada nodo
CALL      gds.betweenness.write('myGraphBC',          {writeProperty:
"betweennessCentrality"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
       centralityDistribution.mean AS meanScore,
       centralityDistribution.max AS maxScore,
       nodePropertiesWritten;

1 CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessCentrality"})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimunScore,
4       centralityDistribution.mean AS meanScore,
5       centralityDistribution.max AS maxScore,
6       nodePropertiesWritten;

```

	minimunScore	meanScore	maxScore	nodePropertiesWritten
A Text	0.0	18.116291164427764	306.81054687499994	129

★ PAGERANK:

```

//PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphPR',
    'Personaje',
    'HABLA_CON'
);

1 CALL gds.graph.project(
2   'myGraphPR',
3   'Personaje',
4   'HABLA_CON'
5 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre>{ "personaje": { "label": "Personaje", "properties": { ... } } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { ... }, "type": "HABLA_CON" } }</pre>	"myGraphPR"	129	486	40

```

// PASO 2: calcular la memoria
// Aca estamos calculando el pageRank

```

```

CALL gds.pageRank.write.estimate(
    'myGraphPR', {
        writeProperty: 'pagerank',
        maxIterations: 20,
        dampingFactor: 0.85
    }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.pageRank.stream(
    'myGraphPR'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerank'
    }
)
YIELD nodePropertiesWritten, ranIterations;
1 CALL gds.pageRank.write(
2     'myGraphPR',
3     {
4         maxIterations: 20,
5         dampingFactor: 0.85,
6         writeProperty: 'pagerank'
7     }
8 )
9 YIELD nodePropertiesWritten, ranIterations;

```

	nodePropertiesWritten	ranIterations
A Text	129	13

★ ARTICULATION POINTS:

```

// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphAP',
    'Personaje',
    {HABLA_CON:
        {

```

```

        orientation: 'UNDIRECTED'
    }
}
);

1 CALL gds.graph.project(
2   'myGraphAP',
3   'Personaje',
4   {HABLA_CON:
5     {
6       orientation: 'UNDIRECTED'
7     }
8   }
9 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre>{ "Personaje": { "label": "Personaje", "properties": { ... } } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { ... }, "type": "HABLA_CON" } }</pre>	"myGraphAP"	129	972	360
Code						

```

// PASO 2: calcular la memoria
CALL gds.articulationPoints.stream.estimate(
  'myGraphAP', {writeProperty: 'articulationPoint'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.articulationPoints.stream('myGraphAP')
YEILD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS name, score
ORDER BY score DESC, name ASC;

// PASO 4: añadir el atributo (0 ó 1)
CALL gds.articulationPoints.write(
  'myGraphAP',
  {
    writeProperty: 'articulationPoint'
  }
)
YIELD articulationPointCount;

```

```

1 CALL gds.articulationPoints.write(
2   'myGraphAP',
3   {
4     |   writeProperty: 'articulationPoint'
5   }
6 )
7 YIELD articulationPointCount;

```

	articulationPointCount
Table	1
A Text	16

★ BRIDGES:

```

// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
  'myGraphBridge',
  'Personaje',
  {HABLA_CON: {
    orientation: 'UNDIRECTED'
  }
})
;
```

```

1 CALL gds.graph.project(
2   'myGraphBridge',
3   'Personaje',
4   {HABLA_CON: {
5     |   orientation: 'UNDIRECTED' // Indica que las relaciones no tienen dirección
6   }
7 })
8 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<code> { "Personaje": { "label": "Personaje", "properties": { ... } } } </code>	<code> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { ... }, "type": "HABLA_CON" } } </code>	"myGraphBridge"	129	972	24

```

// PASO 2: calcular la memoria
CALL gds.bridges.stream.estimate(
  'myGraphBridge', {}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplico el algoritmo
CALL gds.bridges.stream('myGraphBridge')
YIELD from, to

```

```

RETURN gds.util.asNode(from).name AS fromName,
       gds.util.asNode(to).name AS toName
ORDER BY fromName ASC, toName ASC;

1 CALL gds.bridges.stream('myGraphBridge')
2 YIELD from, to
3 RETURN gds.util.asNode(from).name AS fromName,
4 |     gds.util.asNode(to).name AS toName
5 ORDER BY fromName ASC, toName ASC;

```

	fromName	toName
A Text	"Sam"	"Randall"
Code	"Theon"	"Captain's Daughter"
	"Theon"	"Drowned Priest"
	"Tyrion"	"Lysa"
	"Tyrion"	"Protester"
	"Tywin"	"Harren"

```

// PASO 4: añadir el atributo
// En este caso el algoritmo Bridges nos señala relaciones que de-
romperse desconectarían partes de grafo. Por lo tanto no es algo que se
guarda en un atributo (capaz sí en el de la relación)

```

ACLARACIÓN: Si bien adjuntamos los pasos para evaluar la cantidad de recursos que consumiría aplicar estos algoritmos (forma de ver que no tronaría), al estar trabajando con un grafo con menos de 200 nodos y menos de 1000 relaciones, evitamos realizar estas partes del procedimiento, porque sabemos que tenemos la capacidad computacional suficiente para que la computadora lo haga sin problema

Respuesta a las preguntas de Centralidad

1. ¿Cuáles son los personajes con mayor Degree en el grafo?

```

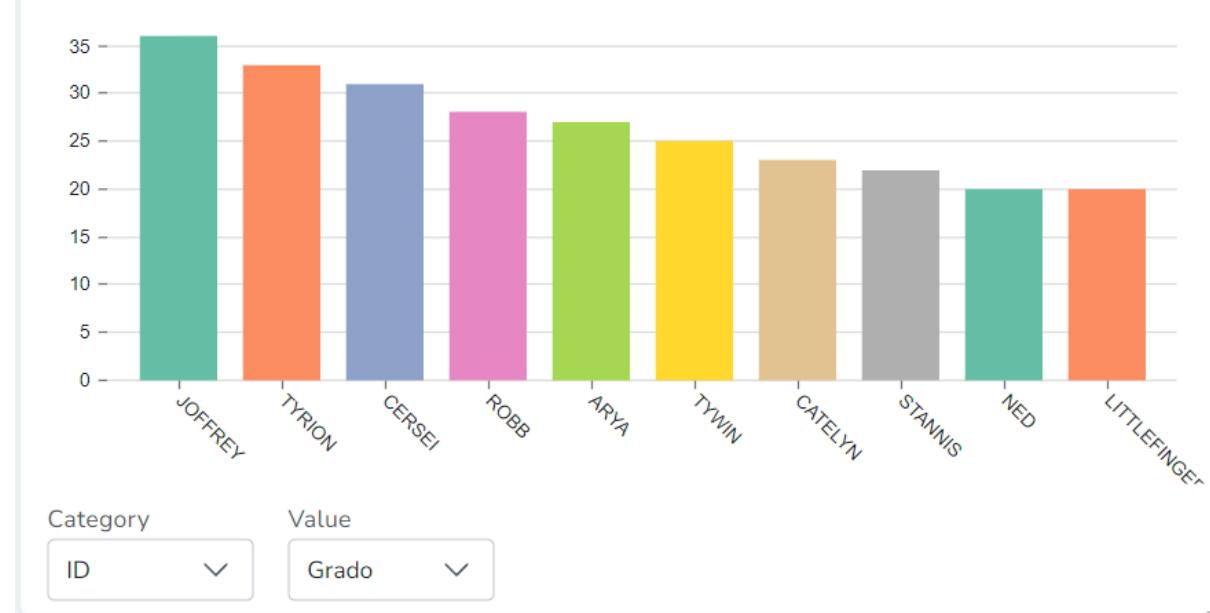
MATCH (p:Personaje)
WITH p.id AS ID,
     p.name AS Nombre,
     p.degree AS Grado

```

Estudiantes: Mateo Josué Velásquez Borda e Iñaki Medina Munguia

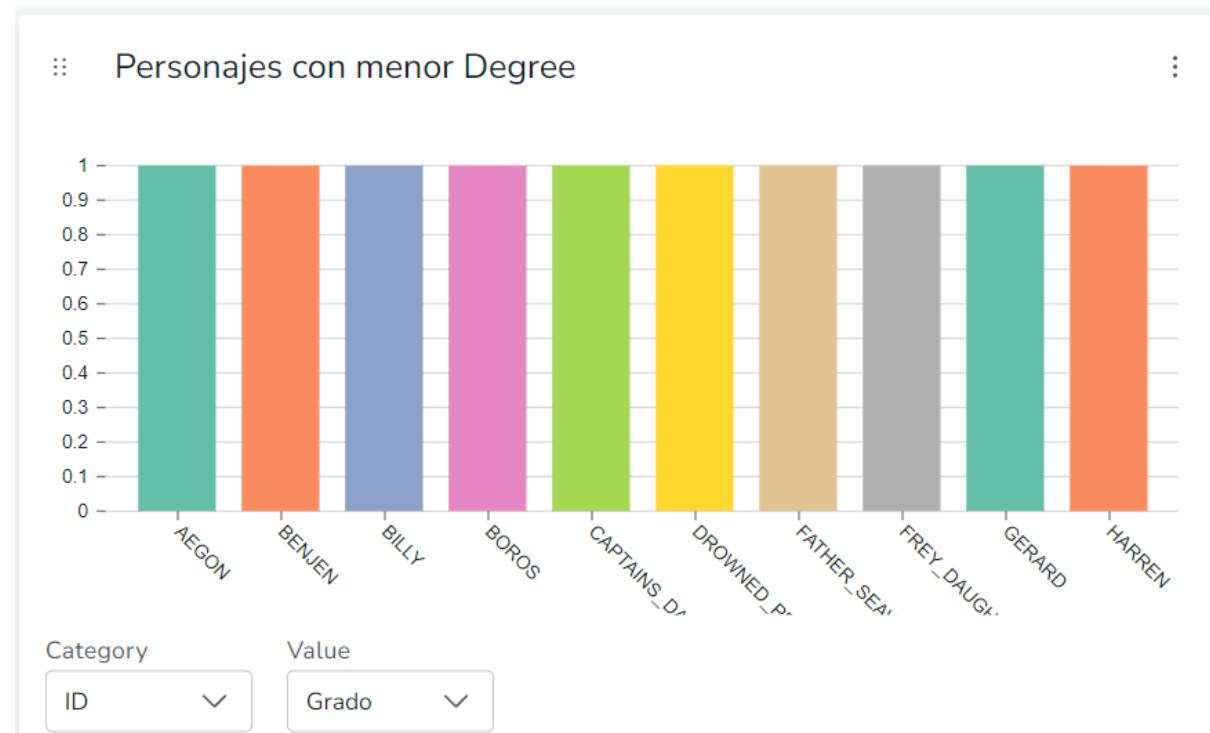
```
ORDER BY Grado DESC, Nombre ASC  
LIMIT 10  
RETURN ID, Nombre, Grado;
```

:: Personajes con mayor Degree ::



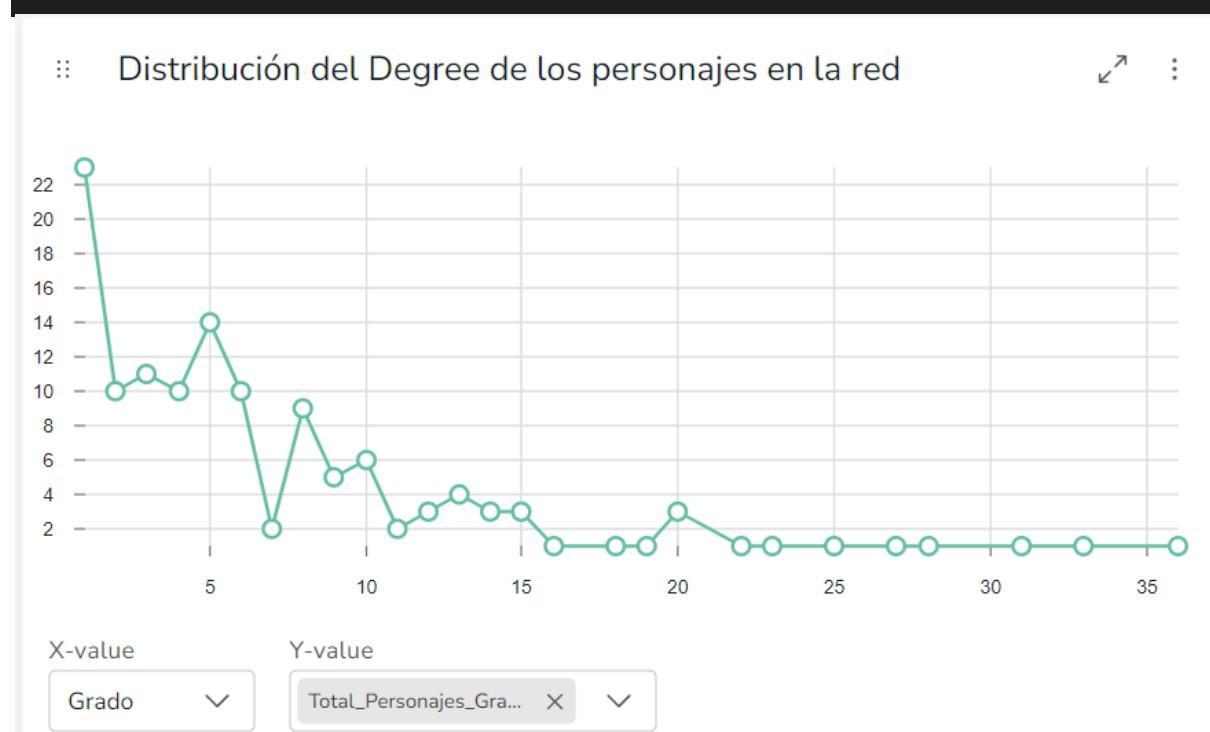
2. ¿Cuáles son los Personajes con menor Degree en el grafo?

```
MATCH (p:Personaje)  
WITH p.id AS ID,  
     p.name AS Nombre,  
     p.degree AS Grado  
ORDER BY Grado ASC, Nombre ASC  
LIMIT 10  
RETURN ID, Nombre, Grado;
```



3. ¿Cuál es la distribución del Degree de los Personajes en la red?

```
MATCH (p:Personaje)
WITH p.degree AS Grado, COUNT(p.degree) AS Total_Personajes_Grado
RETURN Grado, Total_Personajes_Grado
ORDER BY Grado DESC;
```

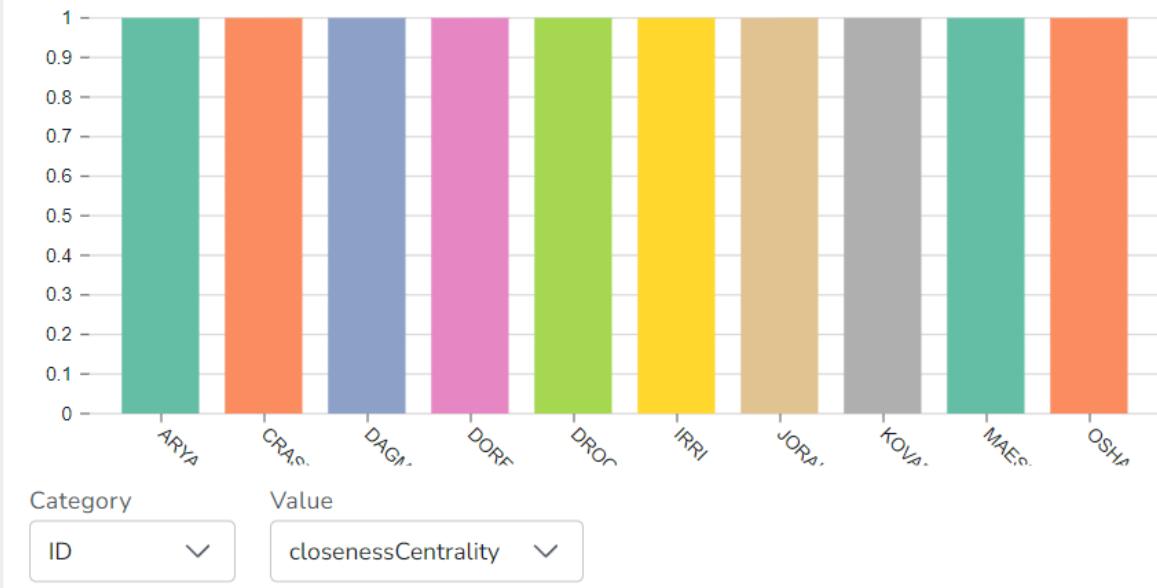


4. ¿Cuáles son aquellos Personajes con un mayor Closeness Centrality?

```
MATCH (p:Personaje)
WITH p.id AS ID,
```

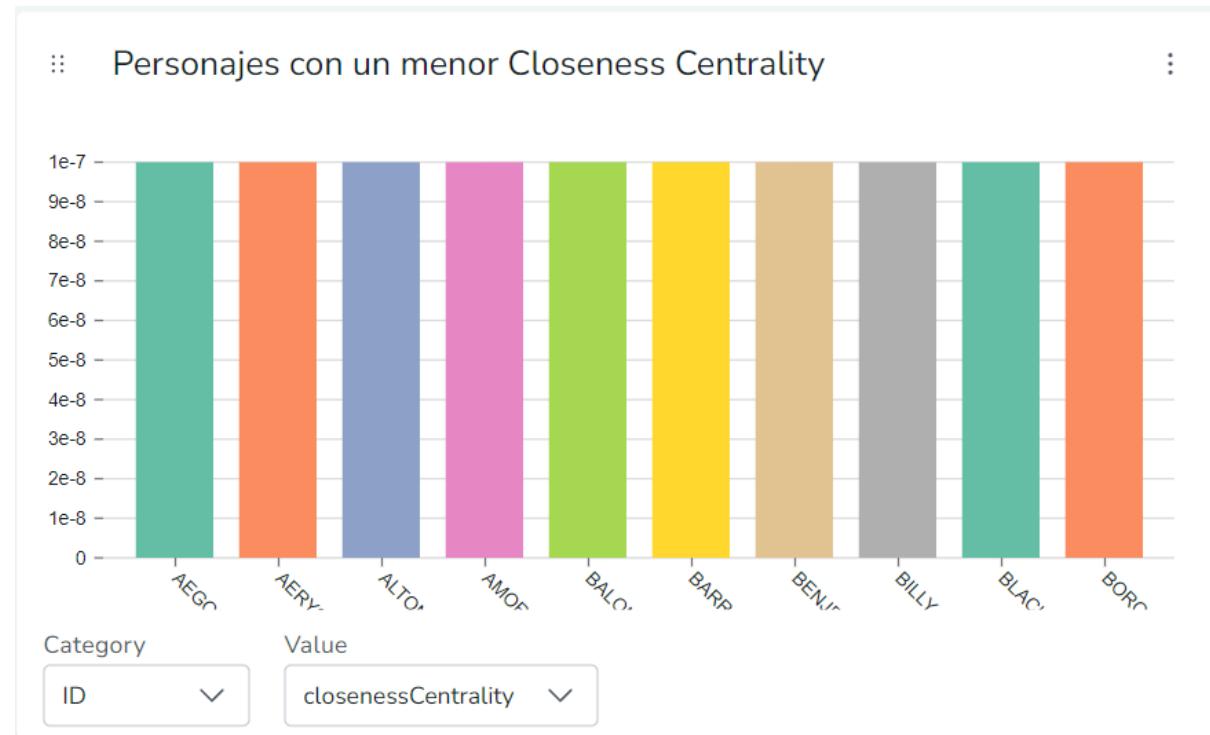
```
p.name AS Nombre,  
p.closenessCentrality AS closenessCentrality  
ORDER BY closenessCentrality DESC, Nombre ASC  
LIMIT 10  
RETURN ID, Nombre, closenessCentrality;
```

:: Personajes con un mayor Closeness Centrality ::



5. ¿Cuáles son aquellos personajes con un menor Closeness Centrality?

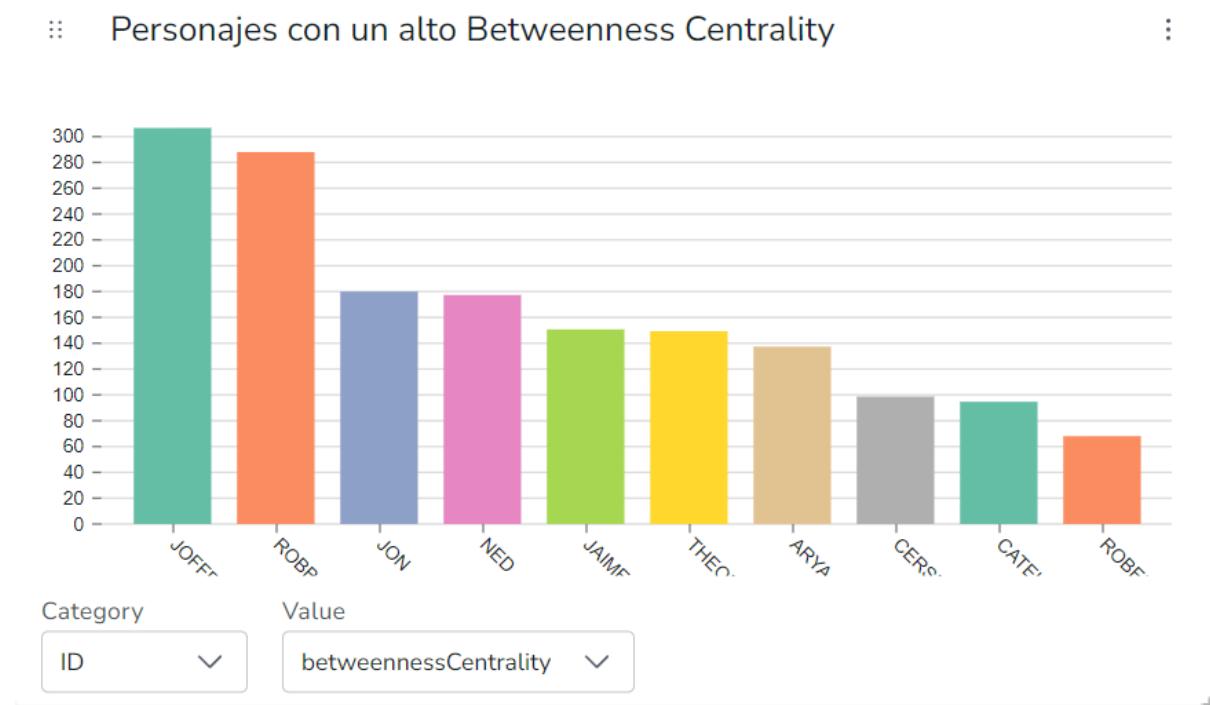
```
MATCH (p:Personaje)  
WITH p.id AS ID,  
     p.name AS Nombre,  
     p.closenessCentrality AS closenessCentrality  
ORDER BY closenessCentrality ASC, Nombre ASC  
LIMIT 10  
RETURN ID, Nombre, closenessCentrality;
```



6. ¿Cuáles son aquellos personajes con un alto Betweenness Centrality?

```

MATCH (p:Personaje)
WITH p.id AS ID,
     p.name AS Nombre,
     p.betweennessCentrality AS betweennessCentrality
ORDER BY betweennessCentrality DESC, Nombre ASC
LIMIT 10
RETURN ID, Nombre, betweennessCentrality;
    
```



7. ¿Cuáles son aquellos personajes con un alto Degree pero que no tienen un alto Betweenness Centrality?

```
MATCH (p:Personaje)
WITH p.id AS ID,
    p.name AS Nombre,
    p.degree AS Grado,
    p.betweennessCentrality AS betweennessCentrality
ORDER BY Grado DESC, Nombre ASC
LIMIT 20
ORDER BY betweennessCentrality ASC, Nombre ASC
LIMIT 10
RETURN ID, Nombre, Grado, betweennessCentrality;
```

:: Personajes con alto Degree pero bajo Betweenness Centrality ::

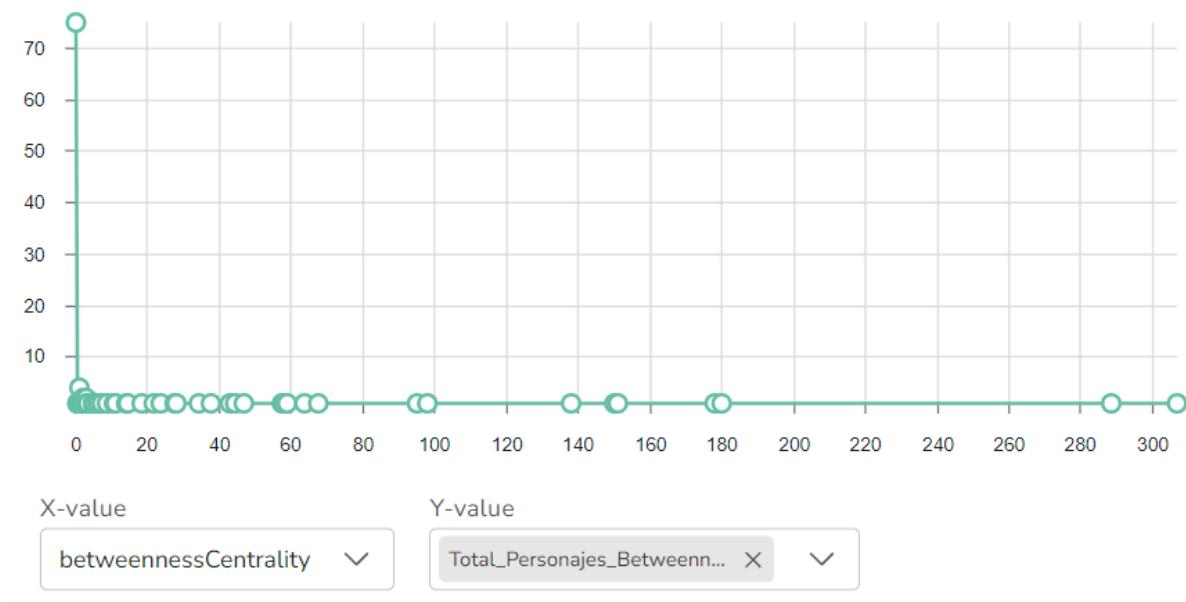
ID	Nombre	Grado	betweennessCentrality
BRONN	Bronn	15	0
VARYS	Varys	14	0
HOUND	Sandor	14	10.751
TYWIN	Tywin	25	14.202
STANNIS	Stannis	22	23.762

Rows per page: 5 ▾ 1–5 of 10 < >

8. ¿Cómo es la distribución del Betweenness centrality?

```
MATCH (p:Personaje)
WITH      p.betweennessCentrality      AS      betweennessCentrality,
COUNT(p.betweennessCentrality)  AS  Total_Personajes_Betweennes
RETURN betweennessCentrality, Total_Personajes_Betweennes
ORDER BY betweennessCentrality DESC;
```

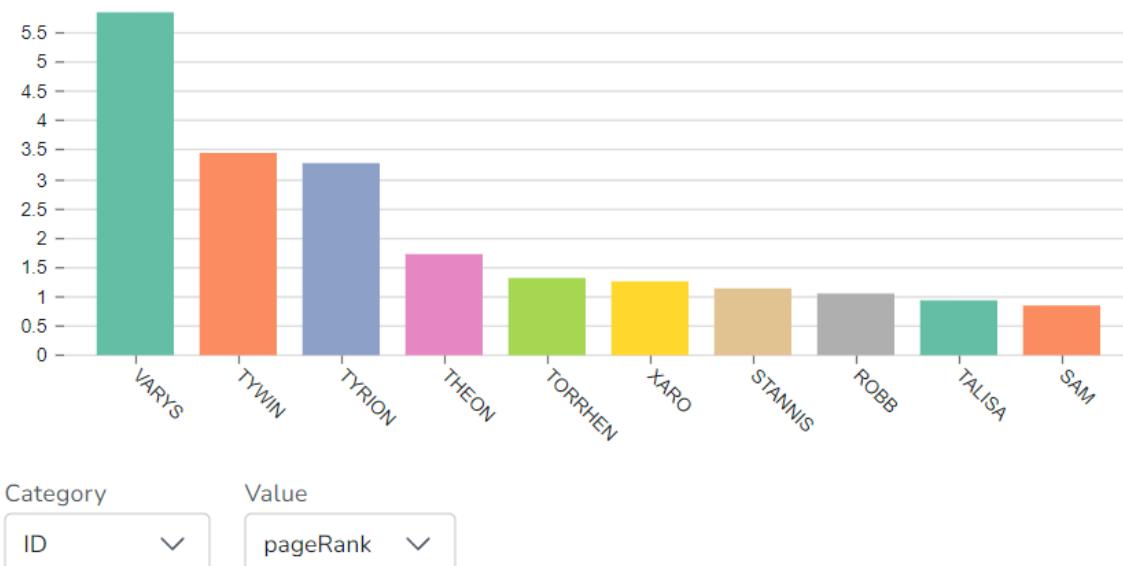
:: Distribución del Betweenness Centrality de los personajes en la red ::



9. ¿Cuáles son aquellos personajes con mayor PageRank?

```
MATCH (p:Personaje)
WITH p.id AS ID,
    p.name AS Nombre,
    p.pagerank AS pageRank
ORDER BY pageRank DESC, Nombre ASC
LIMIT 10
RETURN ID, Nombre, pageRank;
```

:: Personajes con mayor PageRank ::



10. ¿Cuáles son aquellos personajes con un bajo Degree pero con un alto PageRank?

```
MATCH (p:Personaje)
WITH p.id AS ID,
    p.name AS Nombre,
    p.degree AS Grado,
    p.pageRank AS pageRank
ORDER BY pageRank DESC, Nombre ASC
LIMIT 20
ORDER BY Grado ASC, pageRank ASC
LIMIT 10
RETURN ID, Nombre, Grado, pageRank;
```

:: Personajes con bajo Degree pero alto PageRank ::

ID	Nombre	Grado	pageRank
YGRITTE	Ygritte	4	0.707
YARA	Yara	5	0.623
TORRHEN	Torrhen	5	1.319
YOREN	Yoren	6	0.574
RODRIK	Rodrik	8	0.604

Rows per page: 5 ▾ 1–5 of 10 < >

11. ¿Cuáles son aquellos personajes que tienen un alto Degree pero un bajo Page Rank?

```
MATCH (p:Personaje)
WITH p.id AS ID,
    p.name AS Nombre,
    p.degree AS Grado,
    p.pageRank AS pageRank
ORDER BY Grado DESC, Nombre ASC
LIMIT 20
ORDER BY pageRank ASC, Grado DESC
LIMIT 10
RETURN ID, Nombre, Grado, pageRank;
```

Personajes con alto Degree pero bajo PageRank			
ID	Nombre	Grado	pageRank
BRONN	Bronn	15	0.15
CERSEI	Cersei	31	0.187
CATELYN	Catelyn	23	0.205
DAVOS	Davos	15	0.207
HOUND	Sandor	14	0.209

Rows per page: 5 ▾ 1–5 of 10 < >

12. ¿Cuáles son los personajes con menor PageRank?

```

MATCH (p:Personaje)
WITH p.id AS ID,
     p.name AS Nombre,
     p.pageRank AS pageRank
ORDER BY pageRank ASC, Nombre ASC
LIMIT 10
RETURN ID, Nombre, pageRank;
    
```



13. Utilizando Articulation Point ¿Cuáles son aquellos personajes cuyo Articulation Point es True?

```
MATCH (p:Personaje)-[r:HABLA_CON]-(:Personaje)
WHERE p.articulationPoint = 1
WITH p.id AS ID,
    p.name AS Nombre,
    COUNT(r) AS Total_Relaciones
ORDER BY Total_Relaciones DESC
RETURN ID, Nombre, Total_Relaciones;
```

:: Personajes cuyo Articulation Point es True ::

ID	Nombre	Total_Relaciones
JOFFREY	Joffrey	36
TYRION	Tyrion	33
CERSEI	Cersei	31
ROBB	Robb	28
ARYA	Arya	27

Rows per page: 5 ▾ 1–5 of 16 < >

14. Utilizando Bridge ¿Cuáles son las relaciones frágiles en la red que, si se eliminan, fragmentarían la comunidad?

```
CALL gds.graph.project(
    'myGraphBridge',
    'Personaje',
    {HABLA_CON: {
        orientation: 'UNDIRECTED' // Indica que las relaciones no tienen dirección
    }
}
);

CALL gds.bridges.stream('myGraphBridge')
YIELD from, to
RETURN gds.util.asNode(from).name AS fromName,
       gds.util.asNode(to).name AS toName
ORDER BY fromName ASC, toName ASC;
```

fromName	toName
"Aegon"	"Arya"
"Arya"	"Syrio"
"Cersei"	"Joanna"
"Craster"	"Benjen"
"Daenerys"	"Rakharo"
"Davos"	"Father Seaworth"
"Davos"	"Marya"
"Joffrey"	"Boros"
"Luwin"	"Portan"
"Myrcella"	"High Septon"
"Renly"	"Gerard"
"Rickon"	"Billy"
"Robb"	"Frey Daughter"
"Robb"	"Quent"
"Robb"	"Rennick"
"Roose"	"Ramsay"
"Sam"	"Melessa"
"Sam"	"Randyll"
"Theon"	"Captain's Daughter"
"Theon"	"Drowned Priest"
"Tyrion"	"Lysa"
"Tyrion"	"Protester"
"Tywin"	"Harren"

15. ¿Cuál es la información de determinado personaje dado su ID?

```
MATCH (p:Personaje{id:$neodash_personaje_id_3})
RETURN p.id AS ID,
       p.name AS Nombre,
       p.closenessCentrality AS ClosenessCentrality,
       p.betweennessCentrality AS BetweennessCentrality,
       p.pageRank AS pagerank,
       p.degree AS Degree,
       p.articulationPoint AS articulationPoint;
```

The screenshot shows the Neo4j Browser interface. On the left, a sidebar titled "Insertar ID reporte persona" contains a dropdown menu labeled "Personaje id" with the value "JON" selected. On the right, a main panel titled "Reporte Personaje" displays a table with the following data:

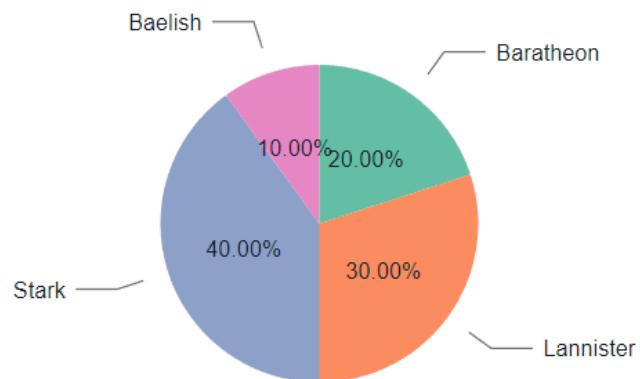
ID	JON
Nombre	Jon
ClosenessCentrality	0.475
BetweennessCentral...	180.167
pagerank	0.462
Degree	12

Below the table, there are pagination controls: "Rows per page: 5", "1–5 of 6", and navigation arrows.

16. ¿A qué Casa pertenecen los 10 Personajes con mayor Degree?

```
MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
WITH p.id AS ID,
      p.name AS Nombre,
      p.degree AS Grado,
      c.name AS Casa
ORDER BY Grado DESC, Nombre ASC
LIMIT 10
RETURN COUNT(Casa) AS Total_Casa, Casa;
```

:: Casa pertenecen los 10 Personajes con mayor Degree ::



Category

Casa

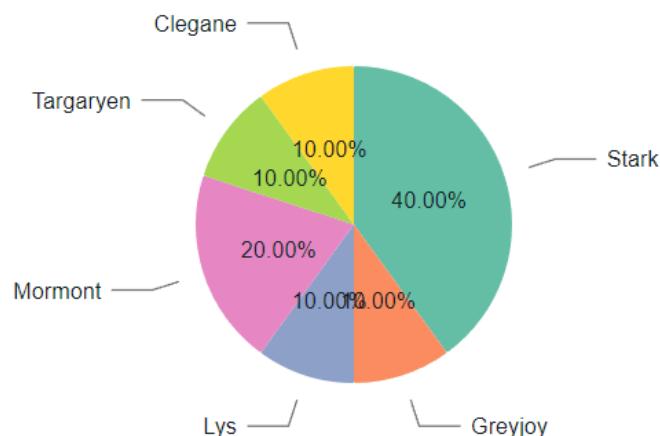
Value

Total_Casa

17. ¿A qué casa pertenecen los 10 personajes con mayor Closeness Centrality?

```
MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
WITH p.id AS ID,
    p.name AS Nombre,
    p.closenessCentrality AS ClosenessCentrality,
    c.name AS Casa
ORDER BY ClosenessCentrality DESC, Nombre ASC
LIMIT 10
RETURN COUNT(Casa) AS Total_Casa, Casa;
```

:: Casas a las que pertenecen los 10 Personajes con mayor Closeness ::



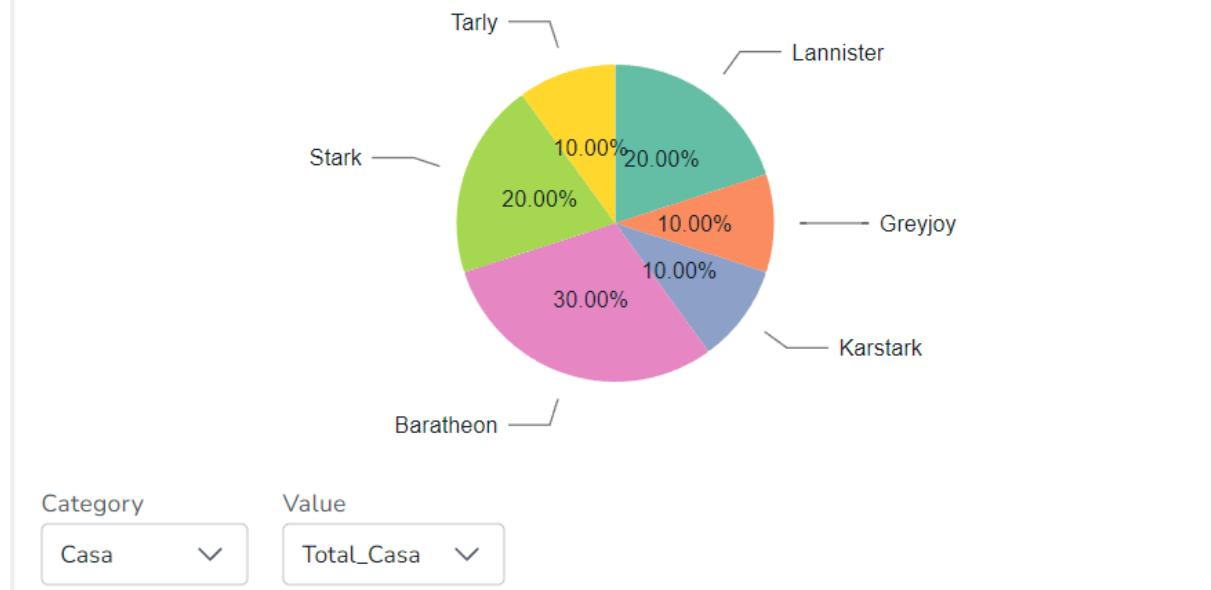
Category

Value

18. ¿A qué casa pertenecen los 10 personajes con mayor PageRank?

```
MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
WITH p.id AS ID,
      p.name AS Nombre,
      p.pagerank AS Pagerank,
      c.name AS Casa
ORDER BY Pagerank DESC, Nombre ASC
LIMIT 10
RETURN COUNT(Casa) AS Total_Casa, Casa;
```

:: Casas a las que pertenecen los 10 personajes con mayor PageRank ::



19. ¿Qué personaje es el que tiene un alto Betweenness centrality entre distintas casas?

```

MATCH (c:Casa)<--[:ES_LEAL_A]-(p:Personaje)-[:HABLA_CON]-(p2:Personaje)-[:ES_LEAL_A]->(c2:Casa)
WHERE c.name <> c2.name
WITH p.id AS ID,
      p.betweennessCentrality AS betweennessCentrality,
      c.name AS Casa1,
      c2.name AS Casa2
ORDER BY betweennessCentrality DESC
RETURN ID, betweennessCentrality, Casa1, COLLECT(DISTINCT Casa2) AS Casas_Conectadas;
    
```

:: Personaje con un alto Betweenness centrality entre distintas casas ::

ID	betweennessCentrality	Casa1	Casas_Conectadas
JOFFREY	306.81	Baratheon	Lannister, Baelish, Tyrell, Seaworth, Trant, Stark, Payne, Blount, Stokeworth, Hollard, Clegane, Slynt
ROBB	288.658	Stark	Cassel, Bolton, Baratheon, Greyjoy, Lannister, Tarth, Seaworth, Frey, Baelish, Tyrell, Karstark
JON	180.167	Stark	Tarly, Tollett, Lannister, Mormont
NED	177.833	Stark	Baratheon, Cassel, Greyjoy, Lannister, Clegane, Arryn
JAIME	150.883	Lannister	Baratheon, Stark, Baelish, Seaworth, Karstark, Targaryen, Tarth

Rows per page: 5 ▾ 1-5 of 10 ⏪ ⏩

20. ¿Cuáles son los Personajes con mayor PageRank por casa?

```

MATCH (c:Casa)<--[:ES_LEAL_A]-(p:Personaje)
WITH c.name AS Casa, p.name AS Personaje, p.pageRank AS PageRank
ORDER BY PageRank DESC
    
```

```
WITH Casa, COLLECT({Personaje: Personaje, PageRank: PageRank}) [0] AS Mejor
RETURN Casa, Mejor.Personaje AS Personaje, Mejor.PageRank AS PageRank
ORDER BY PageRank DESC;
```

:: Personajes con menor PageRank por casa ::

Casa	Personaje	PageRank
Lannister	Tywin	3.463
Greyjoy	Theon	1.732
Karstark	Torrhen	1.319
Baratheon	Stannis	1.149
Stark	Robb	1.061

Rows per page: 5 ▾ 1–5 of 29 < >

Aplicación de los algoritmos de comunidad

En este apartado, establecemos todos los elementos necesarios para aplicar los algoritmos de detección de comunidades de manera eficiente y efectiva. Antes de determinar qué algoritmos serán útiles, es fundamental formular preguntas cuya resolución dependa de estos métodos. Por ello, esta sección comienza con la definición de dichas preguntas.

Preguntas de detección de Comunidades

¿Qué personajes tienen la mayor cantidad de triángulos?

¿Cuáles son los personajes que pertenecen a una mayor cantidad de tríadas densamente conectadas, lo que indicaría una relación fuerte entre ellos?

Responder esta pregunta es clave para la dinámica de juego de tronos, porque nos daría a los personajes pertenecientes a grupos más fuertemente conectados, al menos en la variedad y cantidad de equipos de a tres personas que podría

	hacer
¿Cuáles son las casas con más cantidad de triángulos en promedio?	<p>¿Cuáles son las casas más fuertes y más comunicadas?</p> <p>Esta pregunta es interesante por que ya no se interesa en responder a cuáles son los personajes más fuertes que existen, sino en contar cuales son aquellas casas que tienen la mayor cantidad de triángulos, lo cual es interesante de ver, porque es un análisis más general de que casas son fuertes en sus conexiones.</p>
¿Qué personajes no tienen triángulos?	<p>¿Qué personajes no pertenecen a ninguna triada?</p> <p>En esta pregunta no nos interesa ver las triadas de los personajes ni de las casas, sino que nos interesa conocer que personajes no pertenecen a ninguna triada en Juego de Tronos, para tener un panorama general de los personajes con relaciones menos interconectadas.</p>
¿Cuáles son los personajes con mayor Clustering Coefficient?	<p>¿Cuáles son los personajes que tienen relaciones más interconectadas entre sí?</p> <p>Esta pregunta es interesante porque ya no le importa solamente contar la cantidad de triadas que existen, sino que es importante conocer cómo son todas las relaciones en general y ver que tan bien están conectados el resto de nodos entre ellos.</p>
¿Qué Personajes tienen un alto Degree pero un bajo Clustering Coefficient?	<p>¿Cuáles son los personajes que, a pesar de estar en varias conexiones, tienen un bajo coeficiente de agrupamiento, lo que indicaría que sus conexiones no están interconectadas entre sí?</p> <p>Responder a esta pregunta puede ser interesante, porque puede suceder que un personaje tenga muchas triadas, pero que no necesariamente esas triadas están conectadas entre ellas, lo que podría significar que ese personaje en algún momento si llegara a existir alguna guerra, se viera en la situación de tener que elegir un bando sólo que no estaría muy claro cual.</p>
Dar las comunidades SCC	¿Cuántos componentes fuertemente conectados existen en la red y qué casas o

	<p>facciones representan? ¿Podrían representar alianzas?</p> <p>Esta Pregunta es clave para la dinámica de Juego de Tronos porque nos podría dar las alianzas más factibles de Juego de Tronos en la temporada 2. Información muy útil para aquellas personas que todavía no hayan visto la serie, y en caso de no existir alguna alianza entre los personajes de los grupos fuertes, esto podría indicarnos que en un futuro podrían aliarse para cumplir algún objetivo en común, dado que tendrían una muy buena comunicación entre los miembros de la misma.</p>
¿Qué comunidades existen al aplicar WCC?	<p>¿Cuántos componentes débilmente conectados existen en la red? ¿Podrían indicar futuras alianzas?</p> <p>Esta pregunta simplemente tiene la intención de responder “¿Cuántos continentes hay en la serie?”. Es de público conocimiento que en la serie de Juego de Tronos existen hay cuatro continentes: Poniente, Essos, Sothoryos y Ulthos. Por lo tanto el WCC nos podría indicar cuantos continentes aparecen en esta temporada</p>
¿Cómo son las comunidades del Label Propagation sin semillas específicas?	<p>Si asumimos que los personajes adoptan la lealtad predominante en su entorno inmediato, ¿qué casas o facciones podrían absorber a personajes inicialmente neutrales? ¿Qué alianzas se forman de manera natural en el grafo sin intervención explícita, según la propagación de etiquetas?</p> <p>Esta pregunta es de propósito exploratorio, porque tenemos presente que las alianzas de Juego de Tronos no necesariamente son con las mejores comunidades, sin embargo, si es relevante conocer cómo hubiera sido una distribución “ideal” de las comunidades presentes en la serie</p>
¿Cómo queda el grafo si CERSEI es semilla en Label Propagation?	<p>¿Cómo quedarían las alianzas si CERSEI predominara sobre sus Conversaciones?</p> <p>Esta pregunta simplemente tiene el objetivo de conocer el alcance que tiene el poder de CERSEI sobre las relaciones de Poniente y conocer si ella tiene la capacidad de dar una buena pelea con sus aliados siendo</p>

	una de las líderes en la guerra con el norte.
¿Cómo queda el grafo si ROBB es semilla en Label Propagation?	<p>¿Cómo quedarían las alianzas si ROBB predomina sobre sus conversaciones?</p> <p>Esta pregunta tiene el objetivo de conocer el alcance que tiene el poder de ROBB sobre las relaciones de Poniente y saber si él tiene la capacidad de dar buena pelea siendo el líder en la guerra del norte contra King's Landing.</p>
¿Cómo queda el grafo si TYWIN es semilla en Label Propagation?	<p>¿Cómo quedarían las alianzas si TYWIN predominara sobre sus conversaciones?</p> <p>Esta pregunta simplemente tiene el objetivo de conocer el alcance que tiene el poder de TYWIN sobre las relaciones de Poniente y conocer si él tiene la capacidad de dar una buena pelea con sus aliados siendo uno de los líderes en la guerra con el norte.</p>
¿Cuáles son las comunidades que existen con Louvain y quienes son los que las componen?	<p>¿Qué grupos fuertemente existen en Juego de Tronos y quienes lo componen?</p> <p>Esta pregunta simplemente es para ver que comunidades existen bajo un algoritmo que nos asigna comunidades que es el Louvain Modularity y ver quienes componen casas comunidad</p>
Dar las comunidades de Louvain que no tienen relación con ningún Stark y ningún Lannister	<p>¿Cuáles casas no apoyan fuertemente ni a los Stark ni a los Lannister?</p> <p>Dado que en la serie las dos casas más importantes son los Stark y los Lannister y no todos los personajes tienen casas, sería interesante ver si existe alguna comunidad “aislada” de estos dos poderosos bandos, porque significaría que posteriormente podría existir algún giro en la trama en dónde alguno de estos grupos se una a alguno de los bandos, cambiando las reglas de Juegos de Tronos (Probablemente salgan los Dothraki).</p>
¿Cuáles son los personajes con más Closeness Centrality de cada comunidad de Louvain?	<p>En base a las comunidades detectadas anteriormente ¿Cuáles son los mejores distribuidores de información por posibles alianzas?</p> <p>Responder esta pregunta es interesante por que comenzamos a analizar la centralidad de la comunidad, en este caso</p>

	estamos analizando quienes serían los mejores distribuidores de información, que podrían servir como referentes políticos o líderes.
Aplicando Metric Modularity ¿Qué comunidades tienen el menor Metric Modularity? y ¿Qué Personajes pertenecen a esas comunidades?	<p>¿Existen personajes cuya lealtad no está clara porque están dentro de comunidades con baja modularidad, lo que sugiere conexiones ambiguas?</p> <p>Esta pregunta es para conocer si existen algunas comunidades poco relevantes o fuertes, lo que nos indicaría que en realidad en un futuro si no fortalecen sus relaciones probablemente sean absorbidas por otras alianzas más fuertes.</p>
¿Qué Personajes pertenecen a los últimos grupo de K1-Coloring?	<p>¿Qué personajes tienen la mayor cantidad de conexiones con múltiples grupos, actuando como posibles mediadores o traidores de cada posible alianza?</p> <p>Esta pregunta es clave, por con algoritmo de coloreo podemos averiguar sobre aquellos personajes que tienen más conexiones con múltiples grupos, lo cual resulta muy útil para identificar posibles traidores de algún bando o simplemente posibles mediadores de la paz.</p>
¿Existen Personajes dentro de alguna comunidad creada con Louvain tal que tengan un alto Betweenness Centrality?	<p>¿Existen personajes críticos en las comunidades, tales que al eliminarlos generarían que se vieran obligadas a desaparecer o dividirse?</p> <p>Esta pregunta trata de identificar comunidades que estén conectadas principalmente mediante algún personaje que haga de puente, de tal forma que si lo eliminamos la comunidad podría romperse.</p>
Aplicando Louvain ¿Qué personajes con alto PageRank HABLA_CON otros personajes de distintas comunidades a la propia?	<p>¿Qué personajes son potencialmente aptos para unir dos comunidades distintas para una posible alianza?</p> <p>Queremos nuevamente identificar posibles líderes políticos de alianzas entre distintas comunidades, para nuevamente prevenir que si dichos personajes mueren muchas situaciones de “paz” entre distintas comunidades/bandos podrían romperse o acabarse</p>
¿Cuáles son los grupos formados con	¿Quiénes son los personajes más

Louvain que tienen los Personajes con menores Degrees?	incomunicados de cada comunidad? ¿Por qué personajes habría que empezar a eliminar si quisieramos estratégicamente dejar debilitada dicha alianza poco a poco? La idea de esta pregunta es contar los Personajes con menores DEGREES que tienen cada comunidad, para ver cuales son aquellas alianzas que tienen los personajes menos comunicados, y por lo tanto podríamos ver que tan aisladas están, y en caso de guerra eso les podría jugar en contra.
¿Qué personajes agrupados con Louvain no pertenecen a una Casa específica?	¿Qué grupos de personajes que no tienen casa pueden ser potenciales aliados de otro grupo? La idea de esta pregunta es ver cuántas comunidades no tienen casas específicas, sino cómo que están aisladas y ver que personajes las componen. Si es que llegarán a existir.

Aplicación de los algoritmos de detección de comunidades

Si bien la metodología que vamos a seguir incluye la etapa (4.2) "Incluir atributos estructurados", en este caso es más conveniente agregar dichos atributos en una etapa anterior, antes de responder las preguntas de detección de comunidades.

★ TRIANGLES:

```
// PASO 1: Creamos el subgrafo:  
CALL gds.graph.project(  
    'myGraphTr',  
    'Personaje',  
    {  
        HABLA_CON:  
        {  
            orientation: 'UNDIRECTED'  
        }  
    }  
) ;
```

```

> CALL gds.graph.project( 'myGraphTr', 'Personaje', {HABLA_CON: {orientation: 'UNDIRECTED'}} );

```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
<pre>{ "Personaje": { "label": "Personaje", "properties": { ... } } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { ... }, "type": "HABLA_CON" } }</pre>	"myGraphTr"	129	972	300

```

// PASO 2: calcular la memoria
CALL gds.triangleCount.write.estimate(
    'myGraphTr', {writeProperty: 'triangles'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.triangleCount.stream('myGraphTr')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC, name ASC

// PASO 4: veamos que nodos pertenecen a un triangulo
CALL gds.triangleCount.write('myGraphTr', {
    writeProperty: 'triangles'
})
YIELD globalTriangleCount, nodeCount;

j$ CALL gds.triangleCount.write('myGraphTr', { writeProperty: 'triangles' }) YIELD globalTriangleCount, nodeCount

```

globalTriangleCount	nodeCount
900	129

★ LOCAL CLUSTERING COEFFICIENT

```

// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Personaje',
    {HABLA_CON:
        {
            orientation: 'UNDIRECTED'
        }
    }
);

```

```

1 CALL gds.graph.project(
2   'myGraphCC',
3   'Personaje',
4   {HABLA_CON:
5     {
6       orientation: 'UNDIRECTED'
7     }
8   }
9 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre>{ "Personaje": { "label": "Personaje", "properties": { ... } } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { ... }, "type": "HABLA_CON" } }</pre>	"myGraphCC"	129	972	29

```

// PASO 2: calcular la memoria
CALL gds.localClusteringCoefficient.write.estimate('myGraphCC', {
  writeProperty: 'clusteringCoefficient'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.localClusteringCoefficient.stream('myGraphCC')
YIELD nodeId, clusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, clusteringCoefficient
ORDER BY clusteringCoefficient DESC, name;

// PASO 4: escribir atributo
CALL gds.localClusteringCoefficient.write(
  'myGraphCC',
  {
    writeProperty: 'clusteringCoefficient'
  }
)
YIELD nodeCount, nodePropertiesWritten;

```

```

1 CALL gds.localClusteringCoefficient.write(
2   'myGraphCC',
3   {
4     writeProperty: 'clusteringCoefficient'
5   }
6 )
7 YIELD nodeCount, nodePropertiesWritten;

```

	nodeCount	nodePropertiesWritten
1	129	129

★ STRONGLY CONNECTED COMPONENTS

```

// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphSCC',
  'Personaje',
  {HABLA_CON:
    {
      Properties: 'conversaciones'
    }
  }
);

```

```

CALL gds.graph.project()
  'myGraphSCC',
  'Personaje',
  {HABLA_CON:
    {
      Properties: 'conversaciones'
    }
  }
);

```

graphName	nodeCount	relationshipCount	projectMillis
"myGraphSCC"	129	486	84
"Personaje": {			
"label": "Personaje",			
"properties": {			
"HABLA_CON": {			
"aggregation": "DEFAULT",			
"orientation": "NATURAL",			
"indexInverse": false,			
"properties": {			
"conversaciones": {			
"aggregation": "DEFAULT",			
"property": "conversaciones",			
"defaultValue": null			
}			
},			
"type": "HABLA_CON"			
}			

```

// PASO 2: calcular la memoria
CALL      gds.scc.write.estimate('myGraphSCC',           {           writeProperty:
'communitySCC' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory

```

```
// PASO 3: Aplicamos el algoritmo
CALL gds.scc.stream('myGraphSCC', {})
YIELD nodeId, communitySCC
RETURN gds.util.asNode(nodeId).name AS Name, communitySCC AS Component
ORDER BY Component, Name DESC

// PASO 4: escribir atributo
CALL gds.scc.write('myGraphSCC', {writeProperty: 'communitySCC'})
YIELD componentCount, componentDistribution;

1 CALL gds.scc.write('myGraphSCC', {writeProperty: 'communitySCC'})
2 YIELD componentCount, componentDistribution;
```

	componentCount	componentDistribution
Table	129	{ "min": 1, "p5": 1, "max": 1, "p999": 1, "p99": 1, "p1": 1, "p10": 1, "p90": 1, "p50": 1, "p25": 1, "p75": 1, "p95": 1, "mean": 1.0 }
A Text		
Code		

★ WEAKLY CONNECTED COMPONENTS

```
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphWCC',
    'Personaje',
    {HABLA_CON:
        {
            orientation: 'UNDIRECTED',
            Properties: 'conversaciones'
        }
    }
);
```

```

1 CALL gds.graph.project(
2   'myGraphWCC',
3   'Personaje',
4   {HABLA_CON:
5     {
6       orientation: 'UNDIRECTED',
7       Properties: 'conversaciones'
8     }
9   }
10 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A	<pre> "Personaje": { "label": "Personaje", "properties": { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "conversaciones": { "aggregation": "DEFAULT", "property": "conversaciones", "defaultValue": null } }, "type": "HABLA_CON" } } } </pre>		"myGraphWCC"	129	972	374

```

// PASO 2: calcular la memoria
CALL      gds.wcc.write.estimate('myGraphWCC',           {      writeProperty:
'communityWCC' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.wcc.stream('myGraphWCC')
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS name, componentId
ORDER BY componentId, name;

// PASO 4: escribir atributo
CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCC'})
YIELD componentCount, componentDistribution;

```

```
1 CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCC'})
2 YIELD componentCount, componentDistribution;
```

	componentCount	componentDistribution
1	2	{ "min": 14, "p5": 14, "max": 115, "p999": 115, "p99": 115, "p1": 14, "p10": 14, "p90": 115, "p50": 14, "p25": 14, "p75": 115, "p95": 115, "mean": 64.5 }

```
// PASO 5: comprobar el atributo
MATCH (p:Personaje)
RETURN p.communityWCC as numero_de_WCC,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_WCC DESC;
```

★ LABEL PROPAGATION (General)

```
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLP',
    'Personaje',
    {HABLA_CON:
        {
            Properties: 'conversaciones'
        }
    }
);
```

```

1 CALL gds.graph.project(
2   'myGraphLP',
3   'Personaje',
4   {HABLA_CON:
5     {
6       Properties: 'conversaciones'
7     }
8   }
9 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre> { "Personaje": { "label": "Personaje", "properties": { ... } } } </pre>	<pre> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { "conversaciones": { "aggregation": "DEFAULT", "property": "conversaciones", "defaultValue": null } }, "type": "HABLA_CON" } } </pre>	"myGraphLP"	129	486	44

```

// PASO 2: calcular la memoria
CALL gds.labelPropagation.write.estimate('myGraphLP', { writeProperty:
'labelPropagation' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphLP')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name

// PASO 4: escribir atributo
CALL      gds.labelPropagation.write('myGraphLP',           {writeProperty:
'labelPropagation'})
YIELD communityCount, ranIterations, didConverge;
1 CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagation'})
2 YIELD communityCount, ranIterations, didConverge;

```

	communityCount	ranIterations	didConverge
A Text	18	8	true

```

// PASO 5: comprobar el atributo
MATCH (p:Personaje)
RETURN p.labelPropagation as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC;

```

★ LABEL PROPAGATION (CERSEI)

```
MATCH (p:Personaje)
```

```

WHERE p.id = 'CERSEI'
SET p.comunidadInicialCERSEI = p.labelPropagation;

1 MATCH (p:Personaje)
2 WHERE p.id = 'CERSEI'
3 SET p.comunidadInicialCERSEI = p.labelPropagation;

```

Set 1 property, completed after 47 ms.

Table

```

// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphCERSEI',
    'Personaje',
    {HABLA_CON:
        {orientation:'UNDIRECTED'}
    },
    {
        nodeProperties: ['comunidadInicialCERSEI']
    }
);

```

```

1 CALL gds.graph.project(
2     'myGraphCERSEI',
3     'Personaje',
4     {HABLA_CON:
5         {orientation:'UNDIRECTED'}
6     },
7     {
8         nodeProperties:['comunidadInicialCERSEI']
9     }
10 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
Text	<pre> { "Personaje": { "label": "Personaje", "properties": { "comunidadInicialCERSEI": { "property": "comunidadInicialCERSEI", "defaultValue": null } } } } </pre>	<pre> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "HABLA_CON" } } </pre>	"myGraphCERSEI"	129	972	122

```

// PASO 2: calcular la memoria
CALL      gds.labelPropagation.write.estimate('myGraphCERSEI',           {
writeProperty: 'labelPropagationCERSEI' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphCERSEI')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community

```

```
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphCERSEI', {
    seedProperty: 'comunidadInicialCERSEI',
    writeProperty: 'labelPropagationCERSEI'
})
YIELD communityCount, ranIterations, didConverge;
```

	communityCount	ranIterations	didConverge
A Text	5	5	true

```
// PASO 5: comprobar el atributo
MATCH (p:Personaje)
RETURN p.labelPropagationCERSEI as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

★ LABEL PROPAGATION (ROBB)

```
// PASO 0: Añadimos una restricción iniciando con la semilla de CERSEI
MATCH (p:Personaje)
WHERE p.id = 'ROBB'
SET p.comunidadInicialROBB = p.labelPropagation;
```

- 1 MATCH (p:Personaje)
- 2 WHERE p.id = 'ROBB'
- 3 SET p.comunidadInicialROBB = p.labelPropagation;



Set 1 property, completed after 39 ms.

```
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphROBB',
    'Personaje',
    {HABLA_CON:
        {orientation:'UNDIRECTED'}
    },
    {
        nodeProperties:['comunidadInicialROBB']
```

```

        }
    );
}

1 CALL gds.graph.project(
2   'myGraphROBB',
3   'Personaje',
4   {HABLA_CON:
5     {orientation:'UNDIRECTED'}
6   },
7   {
8     nodeProperties:['comunidadInicialROBB']
9   }
10 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A	<pre> { "Personaje": { "label": "Personaje", "properties": { "comunidadInicialROBB": { "property": "comunidadInicialROBB", "defaultValue": null } } } } </pre>	<pre> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "type": "HABLA_CON" } } } </pre>	"myGraphROBB"	129	972	64

```

// PASO 2: calcular la memoria
CALL      gds.labelPropagation.write.estimate('myGraphROBB',
writeProperty: 'labelPropagationROBB' )
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphROBB')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphROBB', {
  seedProperty: 'comunidadInicialROBB',
  writeProperty: 'labelPropagationROBB'
})
YIELD communityCount, ranIterations, didConverge;

1 CALL gds.labelPropagation.write('myGraphROBB', {
2   seedProperty: 'comunidadInicialROBB',
3   writeProperty: 'labelPropagationROBB'
4 })
5 YIELD communityCount, ranIterations, didConverge;

```

	communityCount	ranIterations	didConverge
A	4	3	true

```

// PASO 5: comprobar el atributo
MATCH (p:Personaje)
RETURN p.labelPropagationROBB as numero_de_comunidad,
       COLLECT(p.name) AS Miembros

```

```
ORDER BY numero_de_comunidad DESC;
```

★ LABEL PROPAGATION (TYWIN)

```
// PASO 0: Añadimos una restricción iniciando con la semilla de CERSEI
MATCH (p:Personaje)
WHERE p.id = 'TYWIN'
SET p.comunidadInicialTYWIN = p.labelPropagation;

1 MATCH (p:Personaje)
2 WHERE p.id = 'TYWIN'
3 SET p.comunidadInicialTYWIN = p.labelPropagation;
```



Set 1 property, completed after 30 ms.

Table

```
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphTYWIN',
    'Personaje',
    {HABLA_CON:
        {orientation:'UNDIRECTED'}
    },
    {
        nodeProperties: ['comunidadInicialTYWIN']
    }
);
```

- 1 CALL gds.graph.project(
- 2 'myGraphTYWIN',
- 3 'Personaje',
- 4 {HABLA_CON:
- 5 {orientation:'UNDIRECTED'}
- 6 },
- 7 {
- 8 nodeProperties: ['comunidadInicialTYWIN']
- 9 }
- 10);

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre>{ "Personaje": { "label": "Personaje", "properties": { "comunidadInicialTYWIN": { "property": "comunidadInicialTYWIN", "defaultValue": null } } } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "type": "HABLA_CON" } } }</pre>	"myGraphTYWIN"	129	972	63

```
// PASO 2: calcular la memoria
CALL      gds.labelPropagation.write.estimate('myGraphTYWIN',           {
writeProperty: 'labelPropagationTYWIN' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;
```

```
// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphTYWIN')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphTYWIN', {
    seedProperty: 'comunidadInicialTYWIN',
    writeProperty: 'labelPropagationTYWIN'
})
YIELD communityCount, ranIterations, didConverge;

1 CALL gds.labelPropagation.write('myGraphTYWIN', {
2     seedProperty: 'comunidadInicialTYWIN',
3     writeProperty: 'labelPropagationTYWIN'
4 })
5 YIELD communityCount, ranIterations, didConverge;
```

	communityCount	ranIterations	didConverge
Table	4	4	true
A Text			

```
// PASO 5: comprobar el atributo
MATCH (p:Personaje)
RETURN p.labelPropagationTYWIN as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

★ K-1 COLORING

```
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphK1',
    'Personaje',
    {HABLA_CON:
        {orientation:'UNDIRECTED'}
    }
)
```

```

1 CALL gds.graph.project([
2   'myGraphK1',
3   'Personaje',
4   {HABLA_CON:
5     {orientation:'UNDIRECTED'}
6   }
7 ])
  
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre> { "Personaje": { "label": "Personaje", "properties": { "label": "Personaje" } } } </pre>	<pre> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "label": "HABLA_CON" }, "type": "HABLA_CON" } } </pre>	"myGraphK1"	129	972	86

```

// PASO 2: calcular la memoria
CALL gds.k1coloring.write.estimate('myGraphK1', { writeProperty: 'k1color' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.k1coloring.stream('myGraphK1')
YIELD nodeId, color
RETURN gds.util.asNode(nodeId).name AS name, color
ORDER BY name

// PASO 4: escribir atributo
CALL gds.k1coloring.write('myGraphK1', {writeProperty:'k1color'})
YIELD nodeCount, colorCount, ranIterations, didConverge;
  
```

	nodeCount	colorCount	ranIterations	didConverge
A Text	129	12	1	true

```

// PASO 5: Sumarle +1 al color
MATCH (p:Personaje)
SET p.k1color = p.k1color + 1
RETURN p.k1color as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC;
  
```

```
1 MATCH (p:Personaje)
2 SET p.k1color = p.k1color + 1
3 RETURN p.k1color as numero_de_comunidad,
4 |   COLLECT(p.name) AS Miembros
5 ORDER BY numero_de_comunidad DESC;
```

	numero_de_comunidad	Miembros
1	12	["Varys"]
2	11	["Tyrion"]
3	10	["Stannis"]
4	9	["Robert", "Tywin"]
5		

```
// PASO 6: comprobar el atributo
MATCH (p:Personaje)
RETURN p.k1color as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

★ LOUVAIN MODULARITY

```
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLM',
    'Personaje',
    {HABLA_CON:
        {
            orientation: 'UNDIRECTED',
            Properties: 'conversaciones'
        }
    }
);
```

```

1 CALL gds.graph.project(
2   'myGraphLM',
3   'Personaje',
4   {HABLA_CON:
5     {
6       orientation: 'UNDIRECTED',
7       Properties: 'conversaciones'
8     }
9   }
10 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A Text	<pre>{ "Personaje": { "label": "Personaje", "properties": { "conversaciones": { "aggregation": "DEFAULT", "property": "conversaciones", "defaultValue": null } } } }</pre>	<pre>{ "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "conversaciones": { "aggregation": "DEFAULT", "property": "conversaciones", "defaultValue": null } }, "type": "HABLA_CON" } }</pre>	"myGraphLM"	129	972	101

```

// PASO 2: calcular la memoria
CALL gds.louvain.write.estimate('myGraphLM', { writeProperty: 'louvain' })
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.louvain.stream('myGraphLM')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY name ASC;

// PASO 4: escribir atributo
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvain'})
YIELD communityCount, modularity, modularities;

```

	communityCount	modularity	modularities
A Text	1 6	0.5675519483818524	[0.5254026317126455, 0.5675519483818524]

```

// PASO 5: comprobar el atributo
MATCH (p:Personaje)
RETURN p.louvain as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC;

```

★ MODULARITY METRIC

// PASO 1: subgrafo

```

CALL gds.graph.project(
    'myGraphMMetric',
    {
        Personaje: {
            properties: ['louvain']
        }
    },
    {
        HABLA_CON: {
            orientation: 'UNDIRECTED',
            properties: ['conversaciones']
        }
    }
);

1 CALL gds.graph.project(
2     'myGraphMMetric',{
3         Personaje: {
4             properties: ['louvain']
5         }
6     },[HABLA_CON: {
7         orientation: 'UNDIRECTED',
8         properties: ['conversaciones']
9     }
10 ]
11 );

```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
A	<pre> { "Personaje": { "label": "Personaje", "properties": { "Louvain": { "property": "louvain", "defaultValue": null } } } } </pre>	<pre> { "HABLA_CON": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "conversaciones": { "aggregation": "DEFAULT", "property": "conversaciones", "defaultValue": null } }, "type": "HABLA_CON" } } </pre>	"myGraphMMetric"	129	972	26

```

// PASO 2: calcular la memoria
CALL gds.modularity.stats.estimate('myGraphMMetric', {
    communityProperty: 'labelPropagation',
    relationshipWeightProperty: 'conversaciones'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.modularity.stream('myGraphMMetric', {
    communityProperty: 'louvain',
    relationshipWeightProperty: 'conversaciones'
})
YIELD communityId, modularity

```

```
RETURN communityId, modularity
ORDER BY modularity DESC;
```

```
1 CALL gds.modularity.stream('myGraphMMetric', {
2   communityProperty: 'louvain',
3   relationshipWeightProperty: 'conversaciones'
4 })
5 YIELD communityId, modularity
6 RETURN communityId, modularity
7 ORDER BY modularity DESC;
```

Table	communityId	modularity
A	57	0.17676223651849066
Code	97	0.1528077747672803
	76	0.10450027349083194
	123	0.08757466815235822
	89	0.08679819932668335
	93	0.0791037117845353

```
// PASO 4: escribir atributo
CALL gds.modularity.stream('myGraphMMetric', {
  communityProperty: 'louvain',
  relationshipWeightProperty: 'conversaciones'
})
YIELD communityId, modularity

MATCH (p:Personaje)
WHERE p.louvain = communityId
SET p.modularityMetric = modularity;
```

```
1 CALL gds.modularity.stream('myGraphMMetric', {
2   communityProperty: 'louvain',
3   relationshipWeightProperty: 'conversaciones'
4 })
5 YIELD communityId, modularity
6
7 MATCH (p:Personaje)
8 WHERE p.louvain = communityId
9 SET p.modularityMetric = modularity;
```



Set 129 properties, completed after 212 ms.



```
// PASO 5: comprobamos  
MATCH (p:Personaje)  
RETURN p.modularityMetric, COUNT(*) ;
```

Table	p.modularityMetric	COUNT(*)
Text	1 0.08757466815235822	20
Code	2 0.1528077747672803	31
	3 0.10450027349083194	19
	4 0.17676223651849066	31
	5 0.08679819932668335	14
	6 0.0791037117845353	14

Respuesta a las preguntas de detección de comunidades

1. ¿Qué personajes tienen la mayor cantidad de triángulos?

```
MATCH (p:Personaje)  
RETURN p.id AS ID,  
       p.triangles as numero_de_triangulos  
ORDER BY numero_de_triangulos DESC  
LIMIT 10;
```

:: Personajes que pertenecen a más triángulos ::

ID	numero_de_triangulos
JOFFREY	176
CERSEI	151
TYRION	141
STANNIS	100
ROBB	92

Rows per page: 5 ▾ 1–5 of 10 < >

2. ¿Cuáles son las casas con más cantidad de triángulos en promedio?

```
MATCH (p:Personaje)-[r:ES_LEAL_A]->(c:Casa)
WITH c.name AS Casa,
    AVG(p.triangles) AS Promedio_Triangulos,
    SUM(p.triangles) AS Cantidad_Triangulos,
    COUNT(p.name) AS Cantidad_Miembros
RETURN Casa, Promedio_Triangulos, Cantidad_Triangulos,
Cantidad_Miembros
ORDER BY Promedio_Triangulos DESC
LIMIT 10;
```

⋮ Análisis de Triángulos por Casa.

Casa	Promedio_Triangulos	Cantidad_Triangulos	Cantidad_Miembros
Baelish	88	88	1
Stokeworth	61	61	1
Baratheon	56.125	449	8
Lannister	48.636	535	11
Tyrell	47	94	2

Rows per page: 5 ▾ 1–5 of 10 < >

3. ¿Qué personajes no tienen triángulos?

```
MATCH (p:Personaje)
WHERE p.triangles = 0
RETURN p.id AS ID,
       p.triangles as numero_de_triangulos
ORDER BY numero_de_triangulos DESC;
```

Personajes sin Triángulos	
ID	numero_de_triangulos
AEGON	0
BENJEN	0
BILLY	0
BOROS	0
CAPTAINS_DAUGHTER	0

Rows per page: 5 ▾ 1–5 of 23 < >

4. ¿Cuáles son los personajes con mayor Clustering Coefficient?

```

MATCH (p:Personaje)
WHERE p.clusteringCoefficient = 1
RETURN p.id AS ID,
       p.degree AS Grado,
       p.clusteringCoefficient AS clusteringCoefficient
ORDER BY Grado DESC, ID ASC
  
```

Personajes con mayor Clustering Coefficient		
ID	Grado	clusteringCoefficient
HAYLENE	6	1
HODOR	6	1
COLEN	5	1
DOREAH	5	1
ILYN_PAYNE	5	1

Rows per page: 5 ▾ 1–5 of 33 < >

5. ¿Qué Personajes tienen un alto Degree pero un bajo Clustering Coefficient?

```
MATCH (p:Personaje)
WITH p.id AS ID,
    p.degree AS Grado,
    p.clusteringCoefficient AS clusteringCoefficient
ORDER BY Grado DESC, ID ASC
LIMIT 20
ORDER BY clusteringCoefficient ASC, Grado DESC
LIMIT 10
RETURN ID, Grado, clusteringCoefficient;
```

:: Personajes con alto Degree pero bajo Clustering Coefficient ::

ID	Grado	clusteringCoefficient
ARYA	27	0.199
THEON	20	0.232
ROBB	28	0.243
TYRION	33	0.267
JOFFREY	36	0.279

Rows per page: 5 ▾ 1–5 of 10 < >

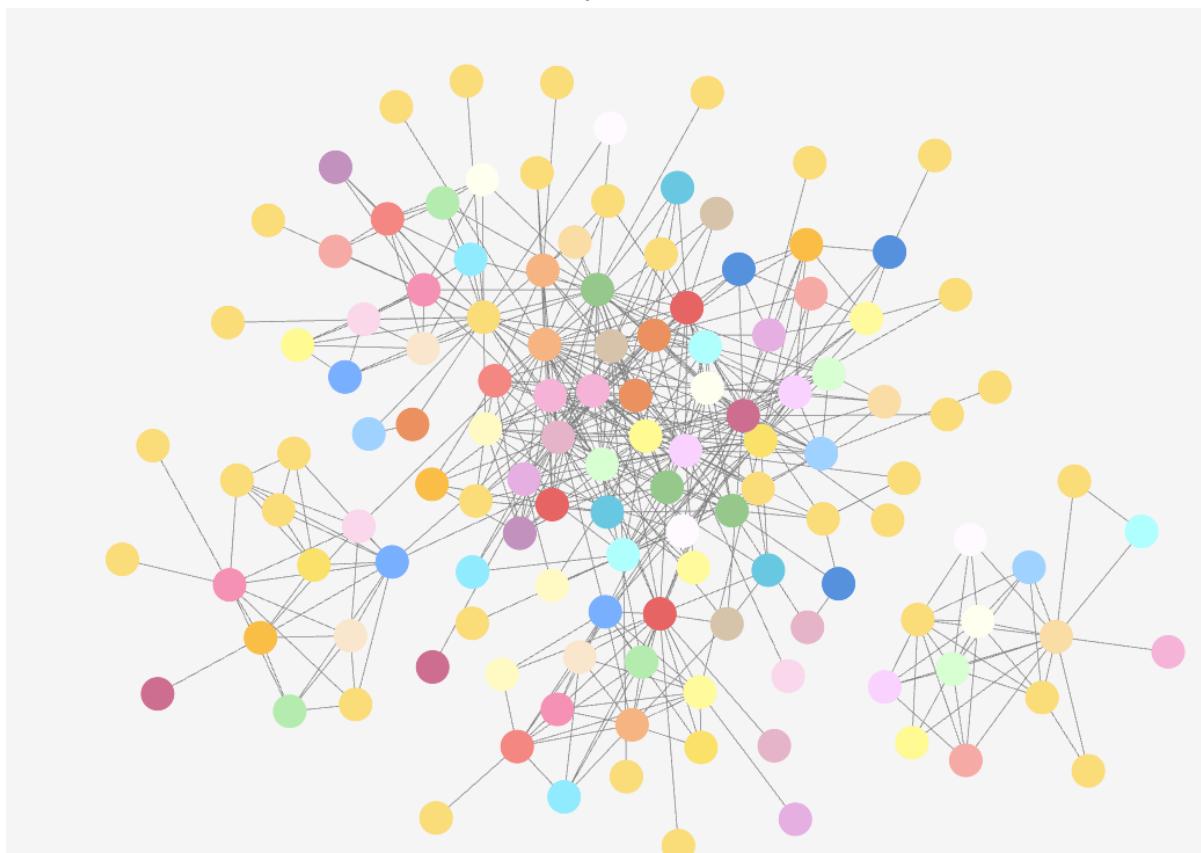
6. Dar las comunidades SCC

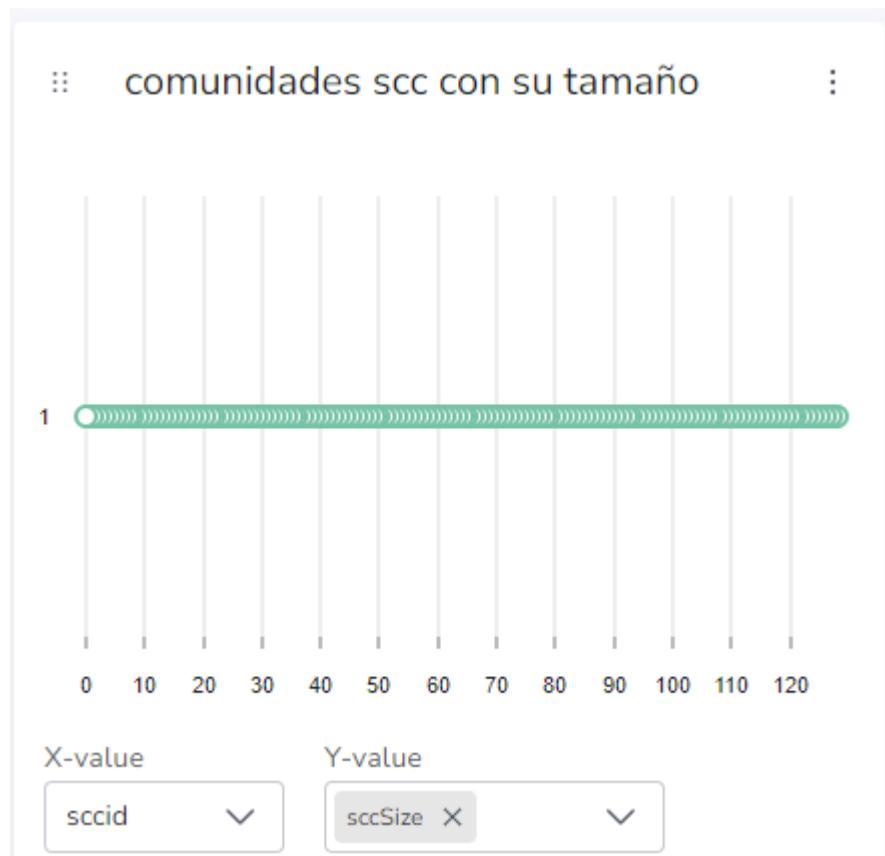
```
MATCH (p:Personaje)
RETURN p.communitySCC, COUNT(*) AS sccSize
ORDER BY sccSize DESC;
```

```
1 MATCH (p:Personaje)
2 RETURN p.communitySCC, COUNT(*) AS sccSize
3 ORDER BY sccSize DESC;
```

	p.communitySCC	sccSize
1	0	1
2	1	1
3	2	1
4	3	1
5	4	1
6	5	1

(nota: todas las comunidades son de tamaño uno ya que si consideras la dirección y que esta dirección va de un Personaje con un nombre más alto alfabéticamente al menor alfabéticamente. No se puede dar un scc mayor a 1)



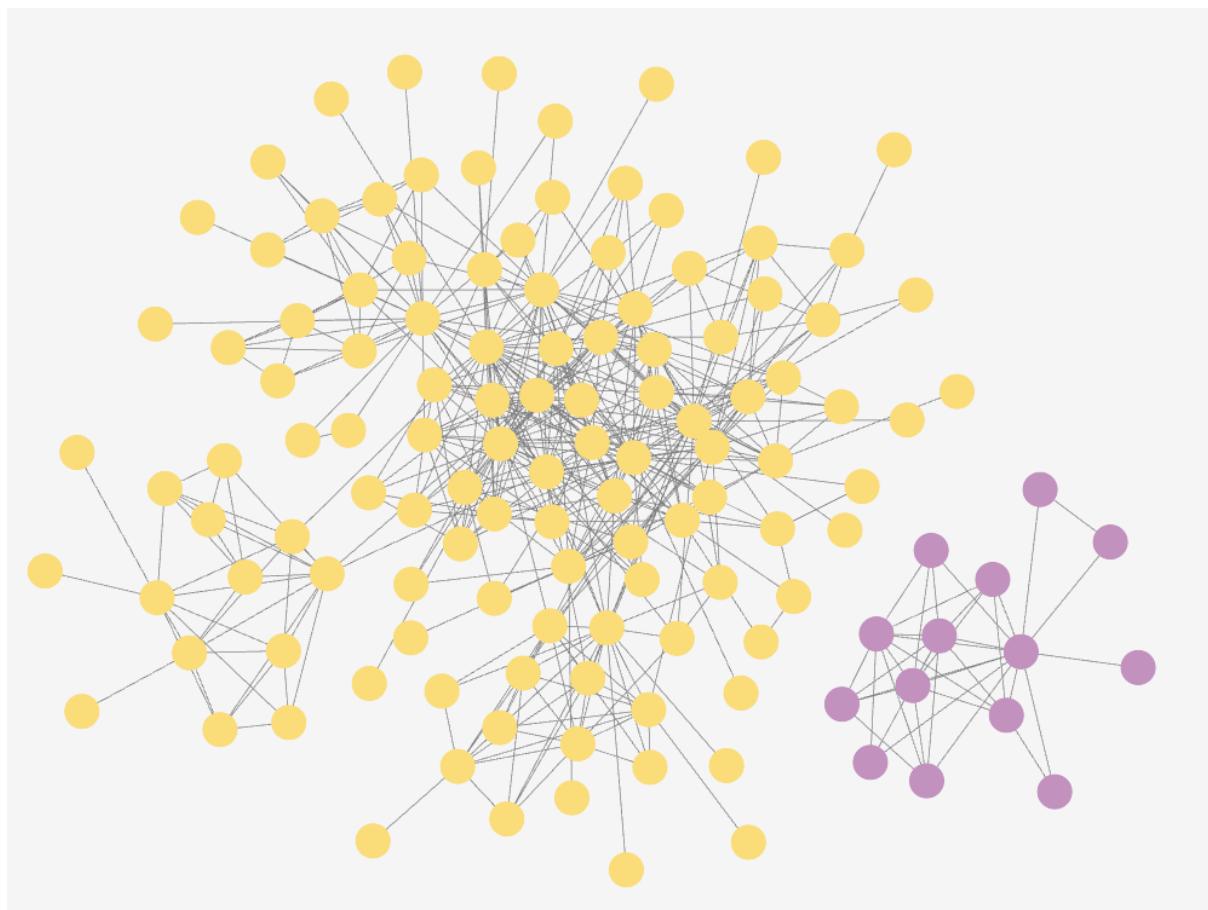


7. ¿Qué comunidades existen al aplicar WCC?

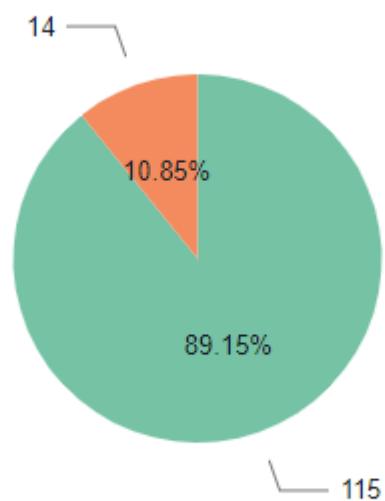
```
MATCH (p:Personaje)
RETURN p.communityWCC, COUNT(*) AS wccSize
ORDER BY wccSize DESC;
```

```
neo4j$ MATCH (p:Personaje) RETURN p.communityWCC, COUNT(*) AS wccSize ORDER BY wccSize DESC;
```

	p.communityWCC	wccSize
1	0	115
2	21	14



:: comunidades wwc con su tamaño ::



Category

wccSize

Value

wccSize

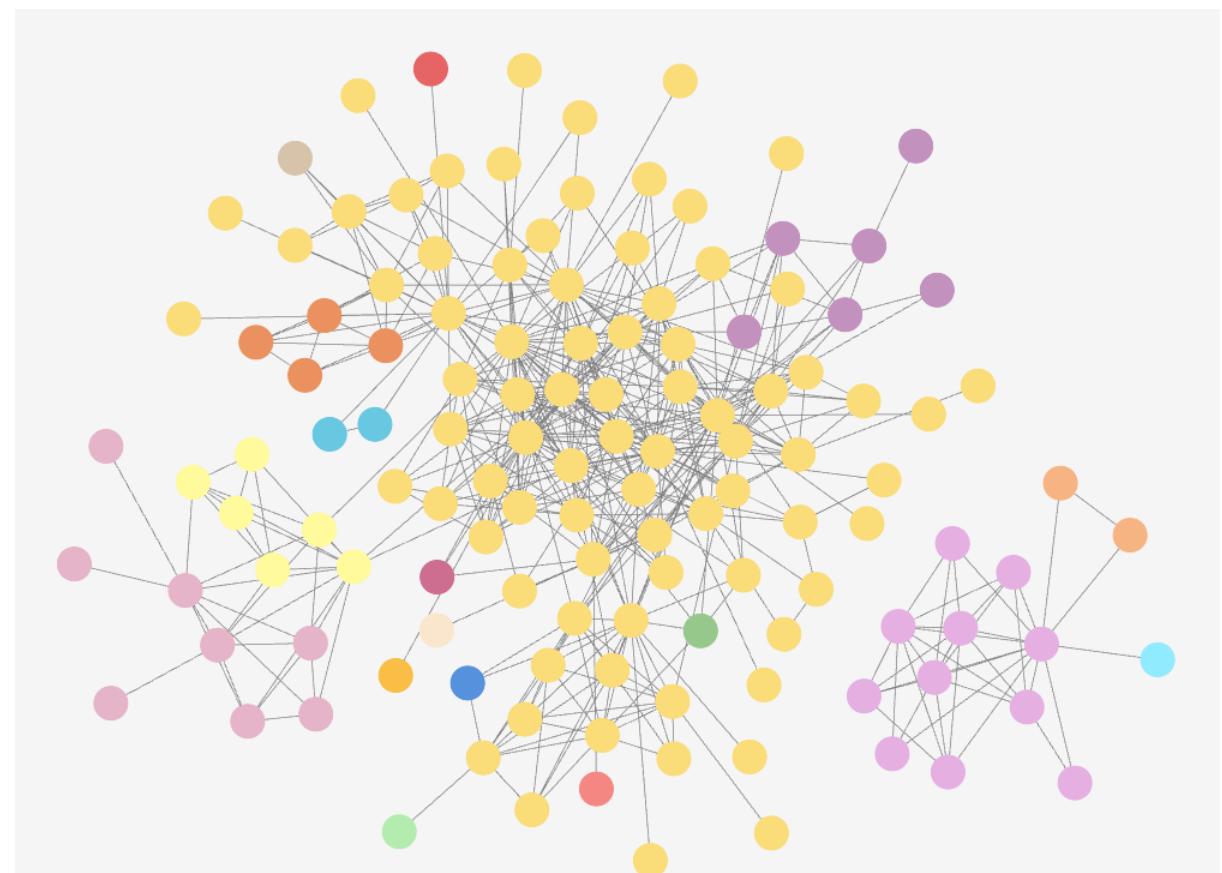
8. ¿Cómo son las comunidades del Label Propagation sin semillas específicas?

```
MATCH (p:Personaje)
RETURN p.labelPropagation, COUNT(*) AS lpSize, COLLECT(p.name)
ORDER BY lpSize DESC;
```

1 MATCH (p:Personaje)
2 RETURN p.labelPropagation AS Comunidad,
3 COUNT(p) AS Cantidad_Miembros,
4 COLLECT(p.name) AS Miembros
5 ORDER BY Cantidad_Miembros DESC;

Comunidad Cantidad_Miembros Miembros

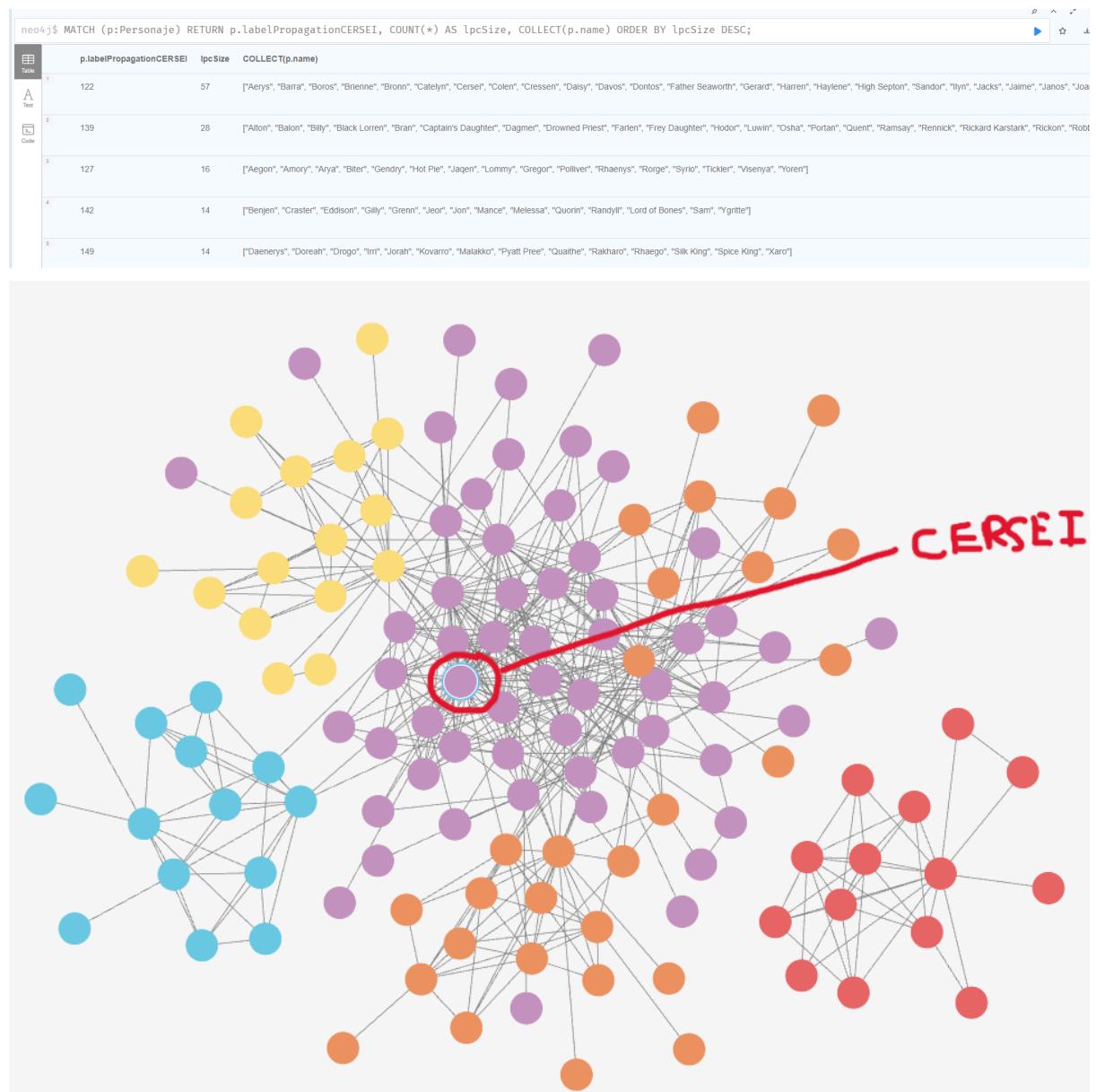
1	122	80	["Aegon", "Aerys", "Alton", "Armory", "Arya", "Balon", "Barra", "Billy", "Black Lorren", "Boros", "Bran", "Brienne", "Bronn", "Captain's Daughter", "Catelyn", "Cersei", "Colen", "Davos", "Dion", "Eddard", "Euron", "Frey", "Gendry", "Hodor", "Jaqen", "Jorah", "Kevan", "Lancel", "Margaery", "Myrcella", "Ned", "Osha", "Petyr", "Ramsay", "Rickard Karstark", "Roose", "Septon", "Talisa", "Torren"]
2	125	11	["Daenerys", "Doreah", "Irra", "Jorah", "Kovarro", "Maiakko", "Pyatt Free", "Qualith", "Silk King", "Spice King", "Xaro"]
3	105	8	["Benjen", "Craster", "Eddison", "Gilly", "Grenn", "Melessa", "Randyll", "Sam"]
4	127	6	["Jeor", "Jon", "Mance", "Quorin", "Lord of Bones", "Ygritte"]
5	118	6	["Ramsay", "Rickard Karstark", "Roose", "Septon", "Talisa", "Torren"]
6	128	4	["Bilbo", "Jaen", "Rorge", "Yoren"]



9. ¿Cómo queda el grafo si CERSEI es semilla en Label Propagation?

```
MATCH (p:Personaje)
RETURN p.labelPropagationCERSEI, COUNT(*) AS lpcSize, COLLECT(p.name)
ORDER BY lpcSize DESC;
```

Estudiantes: Mateo Josué Velásquez Borda e Iñaki Medina Munguia

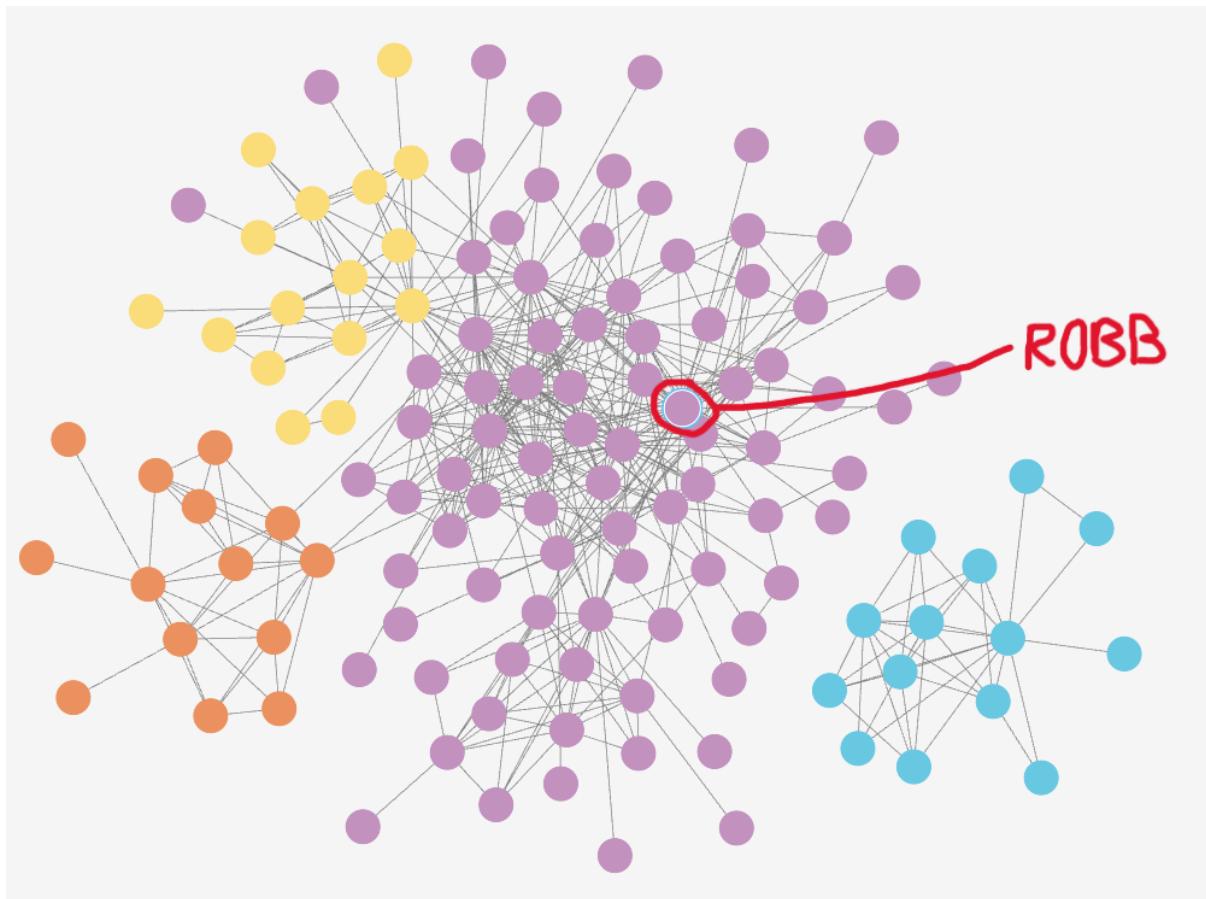


10. ¿Cómo queda el grafo si ROBB es semilla en Label Propagation?

```
MATCH (p:Personaje)
RETURN p.labelPropagationROBB, COUNT(*) AS lprSize, COLLECT(p.name)
ORDER BY lprSize DESC;
```

neo4j\$ MATCH (p:Personaje) RETURN p.labelPropagationROBB, COUNT(*) AS lprSize, COLLECT(p.name) ORDER BY lprSize DESC;

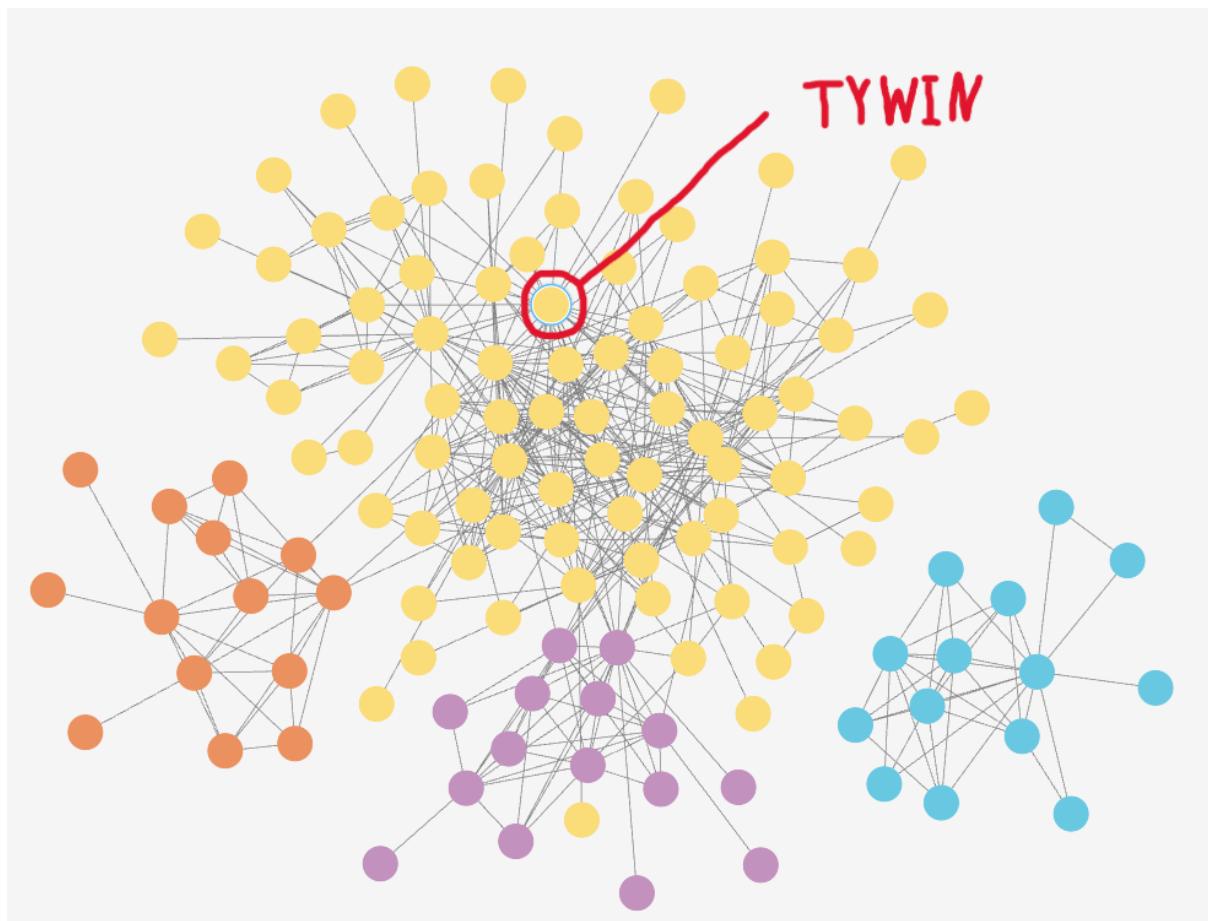
	p.labelPropagationROBB	lprSize	COLLECT(p.name)
1	122	85	["Aerys", "Alton", "Balon", "Barra", "Billy", "Black Lorren", "Boros", "Brienne", "Bronn", "Captain's Daughter", "Catelyn", "Cersei", "Colen", "Cressen", "Dagmer", "Davy", "Davos", "Dontos", "Father Seaworth", "Gerard", "Harron", "Haylene", "High Septon", "Sandor", "Ilyn", "Jacks", "Jaime", "Janos", "Joa
2	127	16	["Aegon", "Amory", "Arya", "Biter", "Gendry", "Hot Pie", "Jaqen", "Lommy", "Gregor", "Polliver", "Rhaenys", "Rorge", "Syrio", "Tickler", "Visenya", "Yoren"]
3	142	14	["Benjen", "Craster", "Eddison", "Gilly", "Greenn", "Jeor", "Jon", "Mance", "Melessa", "Quorin", "Randyll", "Lord of Bones", "Sam", "Ygritte"]
4	149	14	["Daenerys", "Doreah", "Drogo", "Imi", "Jorah", "Kovarro", "Malakko", "Pyatt Pree", "Quaithe", "Rakharo", "Rhaego", "Silk King", "Spice King", "Xaro"]



11. ¿Cómo queda el grafo si TYWIN es semilla en Label Propagation?

```
MATCH (p:Personaje)
RETURN p.labelPropagationTYWIN, COUNT(*) AS lptSize, COLLECT(p.name)
ORDER BY lptSize DESC;
```

	p.labelPropagationTYWIN	lptSize	COLLECT(p.name)
1	122	86	["Aegon", "Aerys", "Alton", "Amory", "Arya", "Balon", "Barra", "Bitter", "Boros", "Brienne", "Bronn", "Catelyn", "Cersei", "Colen", "Cressen", "Daisy", "Davos", "Dontos", "Father Seaworth", "Frey Daughter", "Gendry", "Gerard", "Haren", "Jaeha..."]
2	139	15	["Billy", "Black Lorren", "Bran", "Captain's Daughter", "Dagmer", "Drowned Priest", "Farlen", "Hodor", "Luwin", "Osha", "Portan", "Rickon", "Rodrik", "Theon", "Shepherd"]
3	142	14	["Benjen", "Craster", "Eddison", "Gilly", "Grenn", "Jeor", "Jon", "Mance", "Melessa", "Quorin", "Randy", "Lord of Bones", "Sam", "Ygritte"]
4	149	14	["Daenerys", "Doreah", "Drogo", "Iri", "Jorah", "Kovarro", "Pyatt Pree", "Qualthe", "Rakharo", "Rhaego", "Silk King", "Spice King", "Xaro"]

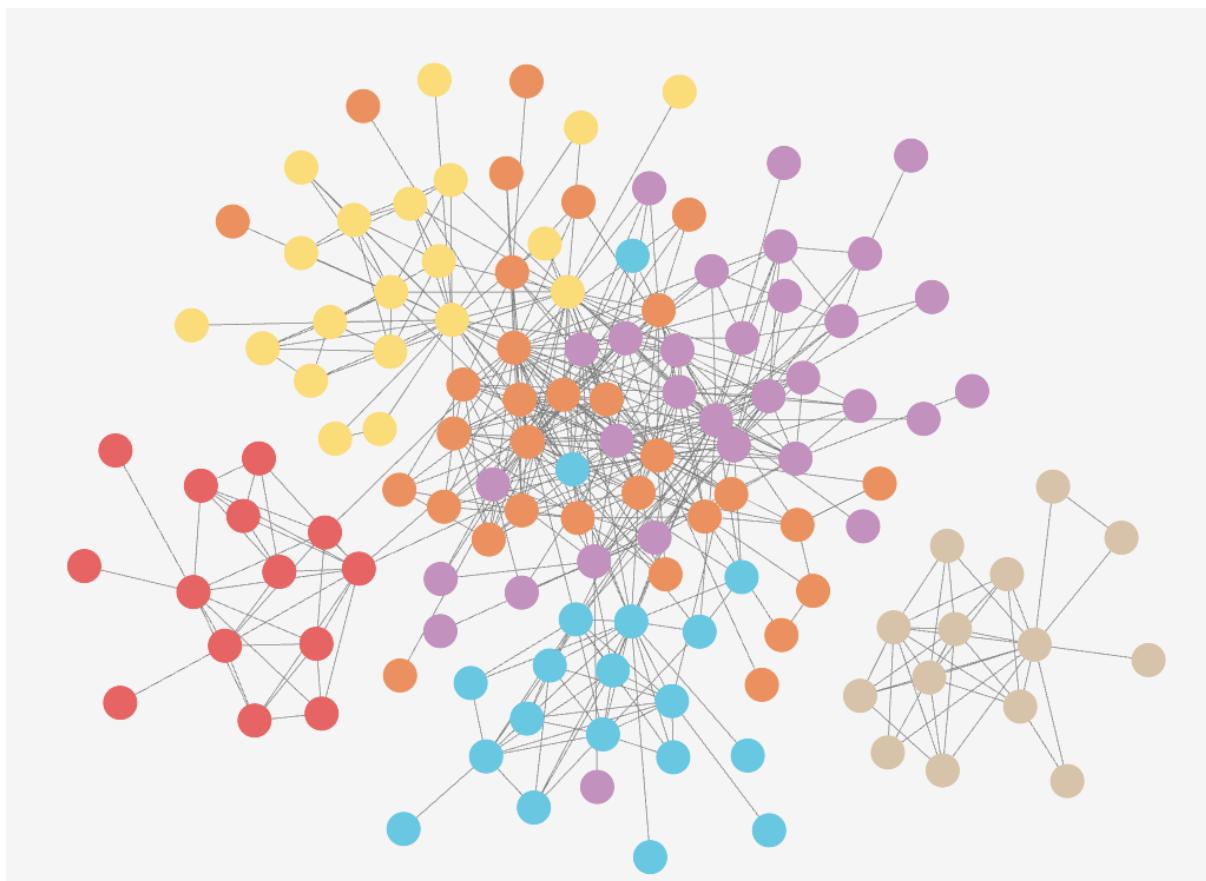


12. ¿Cuáles son las comunidades que existen con Louvain y quienes son los que las componen?

```

MATCH (p:Personaje)
RETURN p.louvain, COUNT(*) AS lSize, COLLECT(p.name)
ORDER BY lSize DESC;
    
```

	p.louvain	lSize	COLLECT(p.name)
1	97	31	["Aerys", "Alton", "Brienne", "Catelyn", "Colen", "Cressen", "Davos", "Father Seaworth", "Frey Daughter", "Gerard", "Jacks", "Jaime", "Petyr", "Loras", "Margaery", "Marya", "Matthos", "Melisandre", "Quent", "Ramsay", "Renly", "Rennick", "Rickard Karstar"]
2	57	31	["Barra", "Boros", "Bronn", "Cersei", "Daisy", "Dontos", "Haylene", "High Septon", "Sandor", "Ilyn", "Janos", "Joanna", "Joffrey", "Lancel", "Lysa", "Mandon", "Meyn", "Mhaegon", "Myrcella", "Podrick", "Protester", "Pycelle", "Robin", "Ros", "Sansa", "Shae"]
3	123	20	["Aegon", "Amory", "Arya", "Bilbo", "Gendry", "Harren", "Hot Pie", "Jaen", "Kevan", "Lommy", "Gregor", "Polliver", "Reginald", "Rhaenys", "Rorge", "Syrio", "Tickler", "Tywin", "Visenya", "Yoren"]
4	76	19	["Balon", "Billy", "Black Lorn", "Bran", "Captain's Daughter", "Dagmer", "Drowned Priest", "Farlen", "Hodor", "Jon Arryn", "Lowin", "Ned", "Osha", "Portan", "Rickon", "Rodrik", "Theon", "Shepherd", "Yara"]
5	89	14	["Benjen", "Craster", "Eddison", "Gilly", "Grenn", "Jeor", "Jon", "Mance", "Melessa", "Quorin", "Randyl", "Lord of Bones", "Sam", "Ygritte"]
6	93	14	["Daenerys", "Doreah", "Drogo", "Im", "Jorah", "Kovarro", "Malakkko", "Pyatt Pree", "Qualthe", "Rakharo", "Rhaego", "Silk King", "Spice King", "Xaro"]



13. Dar las comunidades de Louvain que no tienen relación con ningún Stark y ningún Lannister

```

MATCH (o:Personaje)-[:HABLA_CON]-(p:Personaje)-[:ES_LEAL_A]->(c:Casa)
WITH o.louvain as louvain_Comunidad,
      COLLECT(DISTINCT o.name) AS personajes,
      COLLECT(DISTINCT c.name) AS casas
WHERE NONE(casa IN casas WHERE casa = "Stark" OR casa = "Lannister")
RETURN louvain_Comunidad, personajes;
    
```

- 1 MATCH (o:Personaje)-[:HABLA_CON]-(p:Personaje)-[:ES_LEAL_A]->(c:Casa)
- 2 WITH o.louvain as louvain_Comunidad,
- 3 | COLLECT(DISTINCT o.name) AS personajes,
- 4 | COLLECT(DISTINCT c.name) AS casas
- 5 WHERE NONE(casa IN casas WHERE casa = "Stark" OR casa = "Lannister")
- 6 RETURN louvain_Comunidad, personajes;

	louvain_Comunidad	personajes
A Text	93	["Iri", "Jorah", "Kovarro", "Xaro", "Daenerys", "Malakko", "Pyatt Pree", "Quaiithe", "Spice King", "Doreah", "Drogo", "Rakharo", "Rhaego", "Silk King"]

14. ¿Cuáles son los personajes con más Closeness Centrality de cada comunidad de Louvain?

```

MATCH (p:Personaje)
WITH p.louvain AS louvain,
      p.closenessCentrality AS closeCen,
      p.degree AS grado,
      p.name AS name
    
```

```
ORDER BY closeCen DESC, grado DESC
WITH louvain, collect(closeCen) as centralityList, collect(name) as
nameList
RETURN louvain, nameList[0] as name, centralityList[0] as
centralityAttr
ORDER BY louvain ASC;
```

:: Personajes con más Closeness Centrality por comunidad ::

louvain	name	centralityAttr
57	Cersei	0.714
76	Dagmer	1
89	Craster	1
93	Jorah	1
97	Catelyn	0.75

Rows per page: 5 ▾ 1–5 of 6 < >

15. Aplicando Metric Modularity ¿Qué comunidades tienen el menor Metric Modularity? y ¿Qué Personajes pertenecen a esas comunidades?

```
MATCH (p:Personaje)
WITH MIN(p.modularityMetric) AS MIN_MMetric
MATCH (p:Personaje)
WITH MIN_MMetric,
    p.id AS ID,
    p.louvain AS ID_Comunidad,
    p.modularityMetric AS MMetric
WHERE MMetric = MIN_MMetric
RETURN ID_Comunidad, MMetric, COLLECT(ID) AS Lista_Personajes;
```

Personajes de la comunidad con menor Modularity Metric		
ID_Comunidad	MMetric	Lista_Personajes
93	0.079	DAENERYS, DOREAH, DROGO, IRRI, JORAH, KOVARRO, MALAKKO, PYATT_PREE, QUAITHE, RAKHARO, RHAEGO, SILK_KING, SPICE_KING, XARO

16. ¿Qué Personajes pertenecen a los últimos grupo de K1-Coloring?

```
MATCH (p:Personaje)
RETURN p.k1color as numero_de_comunidad,
       COLLECT(p.name) AS Miembros
ORDER BY numero_de_comunidad DESC
LIMIT 5;
```

Posibles Traidores	
numero_de_comunidad	Miembros
12	Varys
11	Tyrion
10	Stannis
9	Robert, Tywin
8	Renly, Sansa, Theon

Rows per page: 5 ▾ 1-5 of 5 < >

17. ¿Existen Personajes dentro de alguna comunidad creada con Louvain tal que tengan un alto Betweenness Centrality?

```
MATCH (p:Personaje)
RETURN p.id AS ID,
       p.betweennessCentrality AS betweennessCentrality,
       p.louvain AS louvain
ORDER BY betweennessCentrality DESC
LIMIT 10;
```

1	MATCH (p:Personaje)	
2	RETURN p.id AS ID,	
3	p.betweennessCentrality AS betweennessCentrality,	
4	p.louvain AS louvain	
5	ORDER BY betweennessCentrality DESC	
6	LIMIT 10;	
		Table
	ID	betweennessCentrality
1	"JOFFREY"	306.8095238095238
2	"ROBB"	288.6583333333333
3	"JON"	180.1666666666666
4	"NED"	177.8333333333334
5	"JAIME"	150.8833333333333
6	"THEON"	150.0333333333333
		louvain

:: Personajes con mayor Betweenness por Comunidad ::

ID	betweennessCentrality	louvain
JOFFREY	306.81	57
ROBB	288.658	97
JON	180.167	89
NED	177.833	76
JAIME	150.883	97

Rows per page: 5 ▾ 1–5 of 10 < >

18. Aplicando Louvain ¿Qué personajes con alto PageRank HABLA_CON otros personajes de distintas comunidades a la propia?

```

MATCH (p:Personaje)-[r:HABLA_CON]-(p2:Personaje)
WITH p.louvain AS louvain,
     COLLECT(DISTINCT p2.louvain) AS Comunidades_Habladas,
     MAX(ppagerank) AS MaxPageRank
MATCH (p:Personaje)
WHERE ppagerank = MaxPageRank AND p.louvain = louvain
RETURN pid AS ID, MaxPageRank AS PageRank, louvain,
       Comunidades_Habladas
ORDER BY PageRank DESC;
    
```

Estudiantes: Mateo Josué Velásquez Borda e Iñaki Medina Munguia

```

1 MATCH (p:Personaje)-[r:HABLA_CON]-(p2:Personaje)
2 WITH p.louvain AS louvain,
3 |   COLLECT(DISTINCT p2.louvain) AS Comunidades_Habladas,
4 |   MAX(p.pageRank) AS MaxPageRank
5 MATCH [p:Personaje]
6 WHERE p.pageRank = MaxPageRank AND p.louvain = louvain
7 RETURN p.id AS ID, MaxPageRank AS PageRank, louvain, Comunidades_Habladas
8 ORDER BY PageRank DESC;

```

	ID	PageRank	louvain	Comunidades_Habladas
1	"VARYS"	5.855898617698743	57	[57, 123, 97, 76, 89]
2	"TYWIN"	3.463376200277805	123	[123, 97, 57, 76]
3	"THEON"	1.7324973337253624	76	[97, 76, 57, 123, 89]
4	"TORRHEN"	1.318784925342377	97	[97, 57, 123, 76, 89]
5	"XARO"	1.2603614508428018	93	[93]
6	"SAM"	0.85060822017241	89	[89, 57, 76, 97]

III Personajes con alto PageRank por comunidad

ID	PageRank	louvain	Comunidades_Habladas
VARYS	5.856	57	57, 123, 97, 76, 89
TYWIN	3.463	123	123, 97, 57, 76
THEON	1.732	76	97, 76, 57, 123, 89
TORRHEN	1.319	97	97, 57, 123, 76, 89
XARO	1.26	93	93

Rows per page: 5 ▾ 1–5 of 6 < >

19. ¿Cuáles son los grupos formados con Louvain que tienen los menores Degrees?

```

MATCH (p:Personaje)
WITH p.degree as d, p.louvain as louvain, p.id AS ID
WHERE d = 1
RETURN louvain, COUNT(d) AS numeroDePersonajesConDegree1, COLLECT(ID)
AS Personajes
ORDER BY numeroDePersonajesConDegree1 DESC;

```

louvain	numeroDePersonajesConDegree1	Personajes
1	97	["FATHER_SEAWORTH", "FREY_DAUGHTER", "GERARD", "MARYA", "QUENT", "RAMSAY", "RENNICK"]
2	57	["BOROS", "HIGH_SEPTON", "JOANNA", "LYSA", "PROTESTER"]
3	76	["BILLY", "CAPTAINS_DAUGHTER", "DROWNED_PRIEST", "PORTAN"]
4	123	["AEGON", "HARREN", "SYRIO_FOREL"]
5	89	["BENJEN", "MELESSA", "RANDYLL"]
6	93	["RAKHARO"]

:: Cantidad de Personajes con Degree=1 por comunidad ::

louvain	numeroDePe...	Personajes
97	7	FATHER_SEAWORTH, FREY_DAUGHTER, GERARD, MARYA, QUENT, RA...
57	5	BOROS, HIGH_SEPTON, JOANNA, LYSY, PROTESTER
76	4	BILLY, CAPTAINS_DAUGHTER, DROWNED_PRIEST, PORTAN
123	3	AEGON, HARREN, SYRIO_FOREL
89	3	BENJEN, MELESSA, RANDYLL

Rows per page: 5 ▾ 1–5 of 6 < >

20. ¿Qué personajes agrupados con Louvain no pertenecen a una Casa específica?

```

MATCH (p:Personaje)
MATCH (follower:Personaje) -[r:ES_LEAL_A*..1]-> (c:Casa)
WITH p, collect(follower) as following
WHERE NONE(f in following WHERE f=p)
RETURN p.louvain as louvain, COLLECT(p.name) AS personajes
ORDER BY louvain DESC;

```

neo4j\$ MATCH (p:Personaje) MATCH (follower:Personaje) -[r:ES_LEAL_A*..1]→ (c:Casa) WITH p, collect(follower) as following WHERE c.name = "Baratheon" RETURN p.name AS louvain, following AS personajes		
	louvain	personajes
1	123	["Hot Pie", "Jaen", "Lommy", "Rorge", "Syrio", "Yoren"]
2	97	["Gerard", "Jacks", "Marya", "Melisandre", "Salladhor", "Septon"]
3	93	["Drogo", "Irri", "Kovarro", "Malakko", "Pyatt Pree", "Quaithe", "Rakharo", "Silk King", "Spice King", "Xaro"]
4	89	["Craster", "Gilly", "Grenn", "Mance", "Quorin", "Lord of Bones", "Ygritte"]
5	76	["Billy", "Captain's Daughter", "Drowned Priest", "Farlen", "Osha", "Portan", "Shepherd"]
6	57	["Barra", "Daisy", "Haylene", "High Septon", "Mandon", "Mhaegen", "Protester", "Ros", "Shae", "Timett", "Varys"]


```
:: Miembros sin casa por Comunidad ::

louvain    personajes
123    Hot Pie, Jaen, Lommy, Rorge, Syrio, Yoren
97     Gerard, Jacks, Marya, Melisandre, Salladhor, Septon
93     Drogo, Irri, Kovarro, Malakko, Pyatt Pree, Quaithe, Rakharo, Silk King, Spice King, Xaro
89     Craster, Gilly, Grenn, Mance, Quorin, Lord of Bones, Ygritte
76     Billy, Captain's Daughter, Drowned Priest, Farlen, Osha, Portan, Shepherd
```

Rows per page: 5 ▾ 1–5 of 6 < >

Conclusiones y recomendaciones:

Se recolectaron datos públicos de la segunda temporada de Juego de Tronos respecto al peso que tenían las interacciones de los personajes. Sin embargo, a dicha información se le agregaron dos archivos CSV que contenían información sobre las casas existentes en esa temporada y a qué casa pertenecen cada uno de los personajes previamente cargados en los archivos CSV anteriores. Dicha información se obtuvo de las páginas [19] y [20]. Páginas ampliamente conocidas por el fandom de Juego de Tronos.

Gracias a la notación intuitiva de nodos y relaciones, el lenguaje Cypher es mucho más sencillo que sql; más aún si se aumenta el número de joins o un patrón de búsqueda. Además, el lenguaje SQL resulta ser menos eficiente, al tener que recorrer todas las tablas mediante joins [3].

Conclusiones en referencia a los hallazgos:

En base a todo el análisis realizado y documentado en este escrito, es que podemos establecer y obtener las siguientes conclusiones:

- ❖ El grafo está compuesto por 158 nodos, de los cuales: 129 son Personajes y los otros 29 son casas. Hay un total de 486 relaciones “HABLA_CON” mientras que hay 82 relaciones “ES_LEAL_A”. Ahora en nuestro grafo la densidad es de 0,023, lo cuál es algo baja, lo que quiere decir que en realidad las comunicaciones en la Temporada 2 no son muchas, y que puede haber muchas partes incomunicadas. Lo cual se respalda con la pregunta 7 del análisis de detección de comunidades. En esa

respuesta se nos muestra claramente cómo los continentes de Poniente y Essos todavía están incomunicados.

- ❖ Las casas más relevantes en términos de cantidad de Personas son:
 - Stark.
 - Lannister.
 - Baratheon.
 - Targaryen.
 - Greyjoy.
- ❖ Los personajes con más conversaciones relaciones son: Joffrey (36), Tyrion (33), Cersei (31), Robb (28), Arya (27), Tywin (25), Catelyn (23), Stannis (22), Ned (20) y Littlefinger (20). En base a esto concluimos:
 - El personaje con más relaciones es el actual rey, lo cual tiene sentido por ser "la persona más poderosa".
 - Los líderes de la casa Lannister le ganan a los líderes de la casa Stark en cantidad de relaciones, pero ellos les ganan en cantidad de miembros con alto grado
 - Apoyándonos en la pregunta 3 del Análisis de centralidad podemos concluir que:
 - Existen muchos personajes que tienen una sola relación o dos.
 - Los poderes están centralizados en líderes o referentes de las casas más poderosas.
- ❖ En base al análisis de Closeness Centrality, podemos concluir que varios personajes de distintas casas serían muy buenos para transmitir información. Por consiguiente, ninguna casa tiene una ventaja en ese aspecto.
- ❖ En base al análisis de Betweenness Centrality podemos ver que los personajes con más Betweenness son: Joffrey (306.80952), Robb (288.65833), Jon (180.16666), Ned (177.83334) y Jaime (150.88333). Por lo tanto llegamos a las siguientes conclusiones:
 - Existen dos líderes extremadamente fuertes para alianzas: Joffrey que es el actual rey de Poniente, y Robb que quiere proclamarse como el rey del Norte.
 - Luego de ellos siguen Jon Snow, Ned Stark y Jaime Lannister, como potenciales líderes. Sin embargo, la diferencia de Betweenness Centrality es bastante notoria (Gráfico de la pregunta 8).
 - Por lo tanto, si quisiéramos debilitar alguna de las dos alianzas más fuertes, deberíamos eliminar a Joffrey o a Robb. Eventos que suceden en la "Boda Roja" y en la "Boda Púrpura".
- ❖ Con la pregunta 7 de centralidad podemos concluir que Varys y Bronn son dos personajes que aparecen bastante en la serie y les dan buenas relaciones, pero que por cómo está estructurado el grafo, jamás llegarán a ser Reyes si siguen con esa dinámica.
- ❖ Los personajes con mejores contactos (mayor PageRank) son los siguientes: Varys (5.85589), Tywin (3.46337), Tyrion (3.26619), Theon (1.73249), Torrhen (1.31878).
 - Este valor de PageRank lo podríamos visualizar cómo que personajes de Juegos de tronos tienen los mejores contactos o relaciones para hacer estrategias, es cómo ver qué Personajes se guían más por planes que por la fuerza bruta, y a veces la mejor forma de ganar es pensando y no con fuerza bruta.

- Varys si bien cómo mencionamos anteriormente no tiene el potencial para ser un Rey por que no es indispensable para unificar personas, sí es un personaje clave, dado que tiene las mejores relaciones y eso lo vuelve en uno de los aliados más importantes que cualquier bando pudiera tener.
- Tywin Lannister es el segundo miembro con más conexiones, lo que nos indica que podría ser de gran ayuda al rey, o que incluso sea el estratega de los Lannister. Al igual que Tyrion
- ❖ En base al Articulation Point, la conexión del grafo depende principalmente de 16 nodos que si sacaramos al menos 1 de ellos ya generaría que se formaran subgrafos.
 - Los personajes más relevantes son: Joffrey, Tyrion, Cersei, Robb, Arya, Tywin, Theon, Davos, Daenerys.
 - En cambio el Bridges nos señala que existen 23 relaciones críticas.
- ❖ En análisis de Algoritmos de centralidad aplicados por casas podemos concluir en rasgos generales que no hay una casa muy predominante. Sin embargo, los Stark si tienen Personajes con mucho Closeness Centrality en esta temporada.
- ❖ Los Personajes que pertenecen a grupos fuertemente conectados (tienen más tríadas) son: Joffrey (176), Cersei (151), Tyrion (141), Stannis (100) y Robb (92).
 - Por lo tanto podemos ver cómo los Lannister y los Baratheon son de las casas más fuertemente conectadas, o sea que están mejor consolidados en GoT.
 - La información del postulado 2 se nos confirma al responder la pregunta 2 del análisis de detección de comunidades.
- ❖ En base al WCC podemos concluir que existen 2 grandes comunidades, por un lado la comunidad de Poniente que están el 89,15% de los Personajes y por otro lado Essos que están el 10,85% de los Personajes Restantes de Juego de Tronos temporada 2 (gráfico pregunta 7).
- ❖ Juntando las preguntas 8, 9, 10 y 11 pudimos llegar a las siguientes conclusiones utilizando el algoritmo de Label Propagation:
 - CERSEI en esta temporada todavía es muy débil en términos de relaciones, por lo que de momento no debería representar una gran amenaza.
 - ROBB y TYWIN si representan una amenaza en caso de que quisieran ir a la guerra, debido a que si tienen una muy buena capacidad de conquista en sus relaciones. Sin embargo, ninguno de los dos predomina en todo Poniente.
 - Daenerys predomina en todo Essos.
- ❖ En cuanto al análisis de Louvain Modularity podemos concluir que se generaron 6 comunidades. 5 en Poniente y 1 de todo Essos. De las cuales podemos concluir que:
 - La comunidad 57, que es dónde están Cersei, Tyrion, Joffrey, Sansa, entre otros y la comunidad 97 que es dónde están Catelyn, Robb, Stannis, Robert, entre otros. Son las comunidades con menor personaje con mayor Closeness Centrality: Por un lado está Cersei (0,71428) y Catelyn (0,75), respectivamente. Por lo tanto, podemos concluir que aunque sean de las comunidades más fuertes y ricas en la historia, no son las más óptimas para transmitir información.
 - Con la pregunta 16 podemos ver que aquellos personajes que tienen más conexiones con múltiples grupos, lo cual resulta muy útil para identificar

posibles traidores de algún bando, mejores consejeros o informantes o simplemente posibles mediadores de la paz son:

- Varys.
- Tyron.
- Stannis.
- Robert.
- Tywin.

- Cómo señalamos en el análisis de centralidad los personajes con mejor Betweenness Centrality eran Joffrey y Robb, esto lo seguimos manteniendo y complementamos con la pregunta 17 al concluir que es uno de los factores por los cuales las comunidades 57 y 97 son las más importantes en nuestro grafo. En este momento de la serie.
- A pesar de que la comunidad 97 es de las más importantes en la trama, está en cuarto lugar en personaje con mejor PageRank, mientras que la comunidad 57 está primera con una amplia diferencia gracias a Varys. Así que esto puede jugar muy en contra a esas comunidades y la recomendación sería eliminar a Varys o simplemente hacer que traicione a los Lannister/Baratheon.

Por lo tanto podemos concluir que existen ciertos personajes muy fuertes a nivel político/militar que son: Joffrey y Robb, pero que no son los únicos ya que en Juego de Tronos la estrategia es clave, y en este asunto podemos mencionar a los que tienen las relaciones más poderosas como: Varys, Tyrion y Stannis, que serían excelentes aliados en una guerra. Además de esto la comunidad de Daenerys es la que tiene menor MMetric, por lo tanto concluimos que le falta consolidar mejor sus fuerzas para ser una mayor amenaza. Por otro lado hay una clara diferencia de comunidades dejando dos muy marcadas que son lideradas por los Baratheon/Lannister y la otra por los Stark.

Por último recalcar que se realizó un análisis con Louvain Modularity porque no era factible aplicar Strongly Connected Components, dado cómo estaba elaborado el archivo CSV que contenía las relaciones ordenadas alfabéticamente. Esto generó que nunca se generaran triángulos o subgrafos cerrados.

Recomendaciones:

- ❖ Si fuéramos espectadores que estamos viendo la segunda temporada recomendaremos dar principal atención a Robb, Jon, Catelyn, Cersei, Daenerys, Joffrey, Varys, Tywin y Tyron. Dado que en esta temporada son los personajes que se podrían considerar más importantes tanto en algoritmos de centralidad como en comunidades.
- ❖ Estar atento a posibles traiciones de Varys, Tyron y Stannis, dado que son personajes con alto PageRank pero también son los últimos de los grupos de k1-color, por lo tanto son posibles traidores.
- ❖ Estar muy atento a posibles muertes de Robb y Joffrey, porque en caso de que estos mueran las comunidades quedarían bastante debilitadas, dado que son líderes claves de la serie en esta temporada.

Referencias bibliográficas:

1. Ortega Guzmán, V. Gutiérrez Preciado, L. Cervantes, F. Alcaraz Mejía, M. (2024). *A Methodology for Knowledge Discovery in Labeled and Heterogeneous Graphs*. Electronic, Systems and Informatics Department, ITESO - The Jesuit University of Guadalajara.
2. Partida Ochoa, S. (2021). *Sistema de recomendación basado en grafos: Escenario de recomendación de productos* [Tesis de maestría, Instituto Tecnológico y de Estudios Superiores de Occidente]. REI ITESO.
3. Gutiérrez, J., & Flórez, C. (2021). *Analítica de grafos para identificar entidades relevantes y comunidades en Mercado Libre: un estudio de caso*. Revista Mutis, 11(1), 77-95.
4. Needham, M., & Hodler, A. E. (2019). *Graph algorithms: Practical examples in Apache Spark and Neo4j*. O'Reilly Media.
5. Neo4j. (s.f.). Degree centrality. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/degree-centrality/>
6. Neo4j. (s.f.). Closeness centrality. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/closeness-centrality/>
7. Neo4j. (s.f.). Betweenness centrality. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/betweenness-centrality/>
8. Neo4j. (s.f.). Page Rank. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/page-rank/>
9. Neo4j. (s.f.). Articulation Points. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/articulation-points/>
10. Neo4j. (s.f.). Bridges. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/bridges/>
11. Neo4j. (s.f.). Triangle count. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/triangle-count/>
12. Neo4j. (s.f.). Local clustering coefficient. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/local-clustering-coefficient/>
13. Neo4j. (s.f.). Strongly connected components. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/strongly-connected-components/>
14. Neo4j. (s.f.). Weakly connected components. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/wcc/>
15. Neo4j. (s.f.). Label propagation. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/label-propagation/>
16. Neo4j. (s.f.). K-1 Coloring. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/k1coloring/>
17. Neo4j. (s.f.). Modularity Metric. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/modularity/>
18. Neo4j. (s.f.). Louvain. Neo4j Graph Data Science Documentation. <https://neo4j.com/docs/graph-data-science/2.12/algorithms/louvain/>
19. <https://hieloyfuego.fandom.com/wiki/Wiki>
20. <https://www.fandom.com/universe/game-of-thrones>