



ITESO

Universidad Jesuita
de Guadalajara

Ingeniería en Sistemas Computacionales

Minería de Grafos

Mini Proyecto 3

Equipo 1

IS727272 - Marco Ricardo Cordero Hernández

Porcentaje de créditos aprobados: 91%

Tlaquepaque, Jal., 27 de noviembre de 2023

Índice

| | |
|---|----|
| Introducción..... | 1 |
| Desarrollo..... | 3 |
| Carga y transformación de datos..... | 3 |
| Análisis básico | 5 |
| Aplicación de los algoritmos de búsqueda..... | 10 |
| Estructura básica de uso de GDS..... | 11 |
| Modificación estructural previa..... | 13 |
| Aplicación de los algoritmos..... | 13 |
| Presentación de los resultados | 22 |
| Tablero de análisis básico | 22 |
| Tablero de caminos..... | 22 |
| Conclusiones y recomendaciones..... | 23 |
| Bibliografía | 25 |

Introducción

Casi como si se tratase de un portento ominoso, el trabajo final del curso es también uno de los trabajos finales de la carrera, en donde se ha tenido la oportunidad de explorar una infinidad de metodologías y herramientas útiles para dar soporte a empresas o innovar a través de la implantación de la tecnologías en procesos previamente manuales. Pero esto no es lo que le da el misticismo al trabajo que se presenta actualmente, sino que lo hace su tópico de desarrollo y consecuente discusión.

Es bien sabido que el campo matemático relativo al estudio, disección, análisis e implementación de los grafos encuentra su origen en el famoso problema de los puentes de Königsberg (Paoletti, 2006), un tópico que irremediablemente funciona como camino introductorio a temas como los vistos en el curso actual. Por si no se le ha investigado ya lo suficiente, el problema indica la existencia de siete puentes dentro de la ciudad de Königsberg (de ahí el nombre que se le atribuye), el objetivo es recorrer cada uno de ellos pero con la restricción de hacerlo pasando una sola vez por los mismos. En una publicación científica que data en el año 1735, la leyenda matemática Leonhard Euler (pronunciado como *Oiler*) daría una solución más bien decepcionante pero correcta a la problemática presente (la cual por cierto encontró trivial), en donde indicaba que eran necesarios como mínimo ocho puentes para lograr el cometido deseado, haciendo entonces, al menos para la topología de ese entonces, imposible una solución al problema de los puentes. A raíz de esto, dos cosas, una cierta y otra mero rumor, emergieron: por parte de Euler, este personaje estaría introduciendo la teoría de grafos a las matemáticas como un campo totalmente nuevo y de sumo interés por sus posibles aplicaciones incluso en ese tiempo; por parte de Königsberg, en 1875, sus habitantes construyeron un nuevo puente. La parte del rumor viene en que se dice que esta acción fue propiciada por el deseo de dar solución al histórico problema que le daría cierta fama a la ciudad. Lo cierto es que el puente sí fue construido, y destruido por bombarderos británicos 70 años más tarde, casi al final de la segunda guerra mundial. Hoy en día, Kaliningrado es como se le conoce a esta pintoresca ciudad rusa, y su legado es inefable.

Sin duda alguna el destino no favorece a todos, sin embargo, siempre se puede especular a pesar de todo y acerca de cualquier cosa. Remontando un poco más atrás del problema mencionado hacia hechos históricos, el carácter portuario de Königsberg propiciaba un entorno económico favorable para sus residentes, lo cual permitió a la ciudad la construcción del famoso septeto (después octeto) de puentes, por los cuales se paseaban los habitantes. La suposición que se presenta viene en formato de duda: si un comerciante hubiese deseado recorrer las cuatro regiones de la ciudad que conectaban los puentes ¿Cuál habría sido la ruta más corta? Claro que este problema puede resultar incluso aún más trivial que el principal por la escasa cantidad de componentes que contempla la traducción del grafo representativo, pero cuando la esencia de esta suposición se traslada a temas de optimización, ya no resulta tan burdo.

Al rededor de 1920, el matemático Karl Menger, propone el “problema del mensajero”, hoy en día mejor conocido como el “problema del vendedor viajero” (travelling salesman problem), el cual

propone la tarea de encontrar la *ruta más corta* entre un número finito de puntos geográficos conociendo la distancia entre ellos; el giro interesante de esto resulta en que visitar los puntos más cercanos desde un origen no siempre otorga un *camino* óptimo (Cummings, 2000). De manera más formal, la solución a este problema reside en la búsqueda y obtención de un ciclo hamiltoniano, es decir, un ciclo que recorre todos los vértices de un grafo, optimizado por el peso mínimo total.

Muchas propuestas han emergido para dar solución a este nuevo problema de grafos, varias de ellas aún válidas en la actualidad, sin embargo, en 1956, el científico computacional Edsger Dijkstra, en tan solo 20 minutos y sin hacer uso de ningún recurso más que el de la voluntad y el poder de la mente, creó un algoritmo para encontrar el camino más corto entre Róterdam y Groninga, todo mientras degustaba un café en compañía de su prometida (Frana & Misa, 2010). Este algoritmo, además de causar muchas instancias de cefaleas para aquellos que no tienen la destreza de su inventor para implementarlo en tiempo récord, ha probado ser el más veloz de los algoritmos para caminos más cortos desde un nodo origen hacia un nodo destino dentro de grafos arbitrarios con **pesos no negativos** dentro de sus relaciones.

Muchos de los conceptos mencionados en esta introducción pueden resultar confusos y ambiguos, y para atender a esto, el documento presente propone la exploración de un conjunto de datos completamente nuevo en donde se detalla un tipo alterno de interacción social, de forma que se intentará demostrar la utilidad de algoritmos de *caminos* y *búsqueda* en contextos distintos a los tradicionales.

Desarrollo

Después de una extensiva senda del conocimiento, el desarrollo final presentado para el curso involucra la ya clásica exploración inicial de un set de datos desconocido para su posterior manipulación con el propósito de aplicar los algoritmos mencionados.

Como ya es de costumbre, las herramientas predilectas para llevar a cabo este desarrollo son la base de datos basada en grafo *Neo4j* en su versión de escritorio 1.5.9 (s.f.) en conjunto de su extensión *Graph Data Science* (GDS) en su versión 2.5.4 (s.f.). También, cabe mencionar que la versión del proyecto sobre el cual se realizan las implementaciones cuenta con la versión 5.13.0, la más reciente al momento de la producción del presente.

Carga y transformación de datos

En esta ocasión, a diferencia de los proyectos desarrollados anteriormente, la fuente de datos ha sido seleccionada personalmente, es decir, no ha sido proporcionada por ninguna entidad más que la propia investigación. Estos datos corresponden a la interacción dentro de la plataforma radial y de recomendación musical *Last.fm* (s.f.) entre usuarios y artistas para el año 2011. Los archivos originales utilizados se encuentran adjuntos a este reporte o como descarga desde su [fuente directa](#).

El formato en que se presentan los archivos no es el común *.csv*, sino que ahora se cuenta con archivos *.dat*. Esto no resulta un impedimento mayor, ya que tanto los archivos de valores separados por comas, como los que se encuentran presentes, sirven al mismo propósito: almacenar datos arbitrarios, la gran diferencia es que los del primer tipo cuentan con un formato preestablecido *pero* que no están obligados a implementarlo. Aprovechando esta flexibilidad, y al analizar los contenidos del set presente, se puede encontrar que tienen un esquema muy similar a los alternos, solo que los valores actuales están separados por *tabuladores* (*t*, posición 9 en el estándar ASCII), algo que no supone una limitante para Neo4j. El código de importe es el siguiente:

```
// Artistas (≈ 14 segundos)
LOAD CSV WITH HEADERS FROM 'file:///dat_files/artists.dat' AS row
FIELDTERMINATOR '\t'
MERGE (a:Artist {name: row.name})
ON CREATE SET a.id = toInteger(row.id);

// Índice para artistas
CREATE INDEX index_artist_name FOR (a:Artist) ON (a.name);

// Usuarios a través de user_artists (≈ 6.3 minutos)
LOAD CSV WITH HEADERS FROM 'file:///dat_files/user_artists.dat' AS row
```

MDG

```
FIELDTERMINATOR '\t'

MERGE (u:User {id: toInteger(row.userID)})

WITH u, row
MATCH (a:Artist)
WHERE a.id = toInteger(row.artistID)
MERGE (u)-[:LISTENS_TO {weight: toInteger(row.weight)}]->(a);

// Índice para usuarios
CREATE INDEX index_user_id FOR (u:User) ON (u.id);

// Relación de amistad entre usuarios (≈ 1.3 segundos)
LOAD CSV WITH HEADERS FROM 'file:///dat_files/user_friends.dat' AS row
FIELDTERMINATOR '\t'

MATCH (u1:User), (u2:User)
WHERE u1.id = toInteger(row.userID)
AND u2.id = toInteger(row.friendID)
MERGE (u1)-[:FRIEND_OF]->(u2);

// Etiquetas (≈ 30 segundos)
LOAD CSV WITH HEADERS FROM 'file:///dat_files/tags.dat' AS row
FIELDTERMINATOR '\t'

MERGE (t:Tag {id: toInteger(row.tagID)})

ON CREATE SET t.tagValue = row.tagValue;

// Índice para etiquetas
CREATE INDEX index_tag_value FOR (t:Tag) ON (t.tagValue);

// Relación de etiquetado entre usuarios y artistas (≈ 31 minutos)
LOAD CSV WITH HEADERS FROM 'file:///dat_files/user_taggedartists.dat' AS row
FIELDTERMINATOR '\t'

MATCH (u:User), (t:Tag), (a:Artist)
WHERE u.id = toInteger(row.userID)
AND t.id = toInteger(row.tagID)
AND a.id = toInteger(row.artistID)
MERGE (u)-[:USES {id: toInteger(row.tagID),
    date: date(row.year + '-' + row.month + '-' + row.day)}]->(t)
MERGE (t)-[:TAGS]->(a);
```

```
// Complemento de relación de etiquetado con timestamps (≈ 6 segundos)
LOAD CSV WITH HEADERS FROM 'file:///dat_files/user_taggedartists-timestamps.dat' AS row
FIELDTERMINATOR '\t'
MATCH (u:User)-[l:LISTENS_TO-->(a:Artist)
WHERE u.id = toInteger(row.userID)
AND a.id = toInteger(row.artistID)
WITH u, l, a, datetime({epochMillis: toInteger(row.timestamp)}) AS tstamp
SET l.tagged_at = COALESCE(l.tagged_at, []) + tstamp;
```

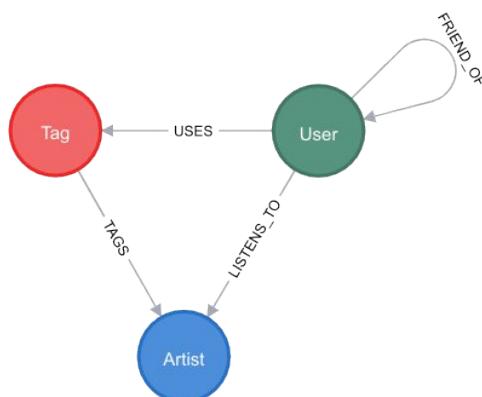
Nota: Un problema conocido de este set de datos es la repetición de entradas con las que cuenta, por ello, las sentencias para su importe han sido escritas de forma que se erradique este problema, sin embargo, el costo de estas operaciones resulta en un tiempo de ejecución total de 40 minutos, esto debido a los productos cartesianos que Neo4j crea internamente con cada sentencia.

Análisis básico

A continuación se presenta nuevamente un análisis básico del grafo cargado, porque como ya se ha aprendido y mencionado, esta parte determina gran parte del éxito de las implementaciones posteriores.

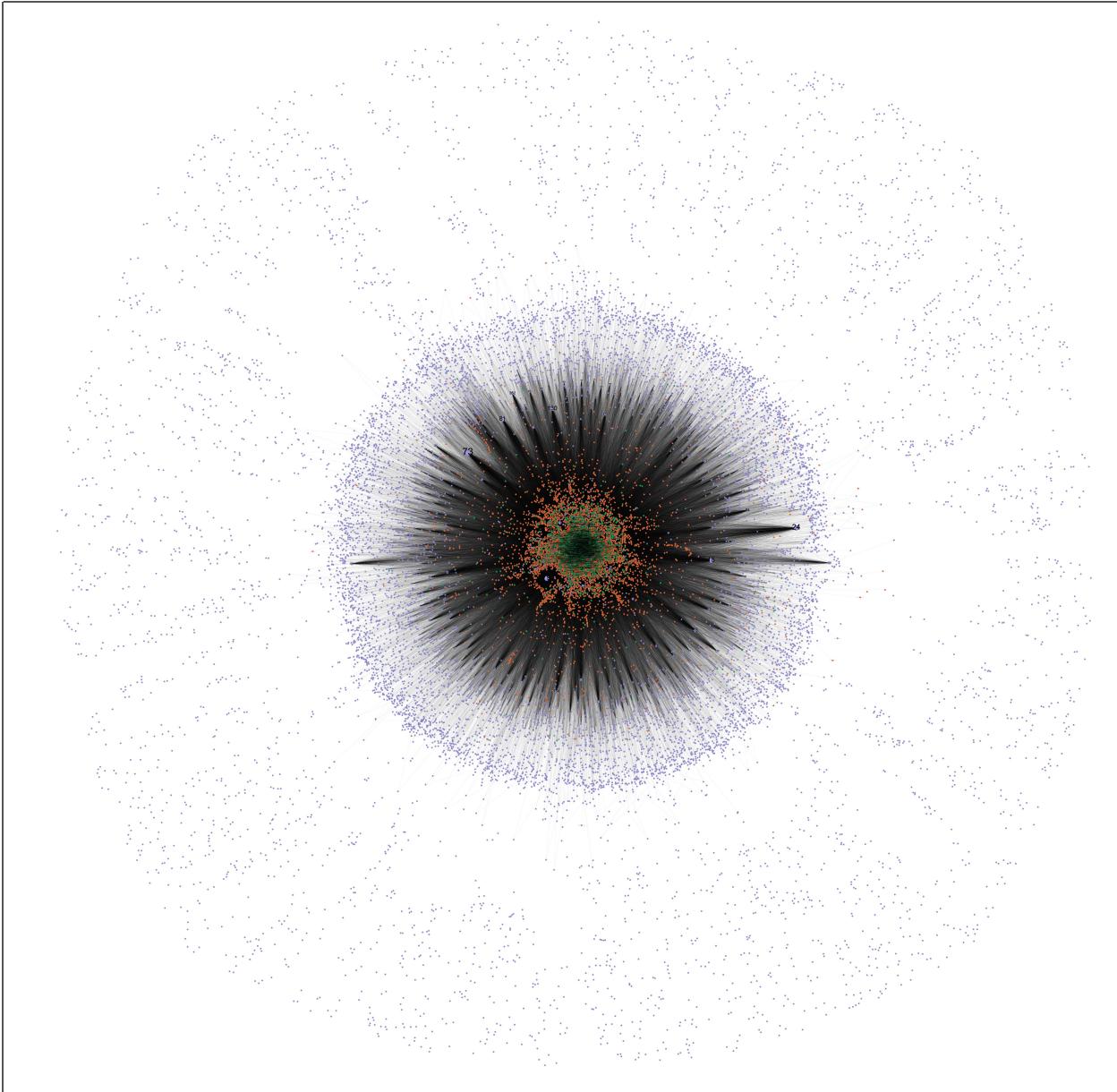
1. Esquema del grafo

```
CALL db.schema.visualization;
```



2. Grafo entero

```
// Hecho en Gephi
```



3. ¿Cuáles nodos hay?

```
CALL db.labels() YIELD label RETURN COLLECT(label) AS NODOS;
```

NODOS

["Artist", "User", "Tag"]

4. ¿Cuántos nodos hay?

```
MATCH (n) RETURN LABELS(n) AS tipoNodo, COUNT(DISTINCT n) AS totalNodos
UNION
MATCH (n) RETURN 'Total' AS tipoNodo, COUNT(DISTINCT n) AS totalNodos;
```

| tipoNodo | totalNodos |
|------------|------------|
| ["Artist"] | 8848 |
| ["User"] | 1892 |
| ["Tag"] | 11946 |
| "Total" | 22686 |

5. ¿Cuáles atributos tienen los nodos?

```
CALL db.schema.nodeTypeProperties()
YIELD nodeType AS Nodo, propertyName AS Propiedad, propertyTypes AS Tipo;
```

| Nodo | Propiedad | Tipo |
|-------------|------------|------------|
| ".`Artist`" | "id" | ["Long"] |
| ".`Artist`" | "name" | ["String"] |
| ".`User`" | "id" | ["Long"] |
| ".`Tag`" | "id" | ["Long"] |
| ".`Tag`" | "tagValue" | ["String"] |

MDG

6. ¿Cuáles son las relaciones?

```
MATCH ()-[r]->() RETURN DISTINCT TYPE(r) AS RELACIONES;
```

| RELACIONES |
|--------------|
| "LISTENS_TO" |
| "FRIEND_OF" |
| "USES" |
| "TAGS" |

7. ¿Cuántas son las relaciones?

```
MATCH ()-[r]->() RETURN TYPE(r) as tipoRelacion, COUNT(r) as totalRelaciones
UNION
MATCH ()-[r]->() RETURN "Total" as tipoRelacion, COUNT(r) as totalRelaciones;
```

| tipoRelacion | totalRelaciones |
|--------------|-----------------|
| "LISTENS_TO" | 81331 |
| "FRIEND_OF" | 25434 |
| "USES" | 66881 |
| "TAGS" | 81883 |
| "Total" | 255529 |

8. ¿Cuáles atributos tienen las relaciones?

```
CALL db.schema.relTypeProperties()
YIELD relType AS Relacion, propertyName AS Propiedad, propertyTypes AS Tipo;
```

| Relacion | Propiedad | Tipo |
|-----------------|-------------|-------------------|
| ".`LISTENS_TO`" | "weight" | ["Long"] |
| ".`LISTENS_TO`" | "tagged_at" | ["DateTimeArray"] |
| ".`FRIEND_OF`" | <i>null</i> | <i>null</i> |
| ".`TAGS`" | <i>null</i> | <i>null</i> |
| ".`USES`" | "id" | ["Long"] |
| ".`USES`" | "date" | ["Date"] |

9. Diámetro del grafo (dirigido)

// Hecho en Gephi

Network Diameter

10

El grafo actual, aún cuando cuenta con múltiples relaciones, no resulta tan conexo como otros analizados, porque haría falta recorrer 10 nodos desde un origen aleatorio para alcanzar todos los demás. Esto podría demostrar la baja calidad de los datos o un conjunto de personas poco sociables.

10. Densidad del grafo

// Hecho en Gephi

Graph Density

0

La densidad obtenida da soporte a la respuesta anterior, de manera que se comprueba el bajo nivel de conexión del grafo al obtener un resultado de 0. Esto más que desalentador resulta interesante para los algoritmos que se aplicarán al mismo grafo más adelante, puesto que la búsqueda de caminos será un tanto intrigante.

11. ¿El grafo es homogéneo o heterogéneo?

```

MATCH (n)
WITH COLLECT(DISTINCT LABELS(n)) AS NodeTypes
RETURN NodeTypes, COUNT(NodeTypes) AS TOTAL;

MATCH ()-[r]->()
WITH COLLECT(DISTINCT TYPE(r)) AS RelantionshipTypes
RETURN RelantionshipTypes, COUNT(RelantionshipTypes) AS TOTAL;

```

| NodeTypes | TOTAL | RelantionshipTypes | TOTAL |
|---------------------------------|-------|---|-------|
| [{"Artist"}, {"User"}, {"Tag"}] | 1 | ["LISTENS_TO", "FRIEND_OF", "USES", "TAGS"] | 1 |

Sí es homogéneo, ya que todos los nodos y relaciones tienen las mismas propiedades

Aplicación de los algoritmos de búsqueda



Mejor visto en [GitHub](#)

En ocasiones pasadas, se han realizado demostraciones de algoritmos aplicables a grafos de categorías denominadas como de centralidad y de comunidad. Ambas cuentan con diversos propósitos útiles que obtendrían métricas interesantes sobre el contexto en donde se aplican.

El mapa mental anterior presenta algunos de los algoritmos de caminos y búsqueda disponibles como parte de su propia denominación, no obstante, existen aún más por explorar. El alcance del proyecto actual contempla la implementación de los siguientes algoritmos:

- Delta-Stepping Single-Source Shortest Path: Este algoritmo es básico para el aprendizaje del conjunto de caminos y búsqueda; combina Dijkstra con Bellman-Ford, permitiendo pesos negativos sin sacrificar eficacia.
- Dijkstra Source-Target Shortest Path: El ya discutido celebre algoritmo puede ser implementado a través de Neo4j y visualizado casi de manera inmediata, lo cual sirve como herramienta pedagógica.
- Random Walk: Como su nombre indica, el algoritmo emprende una ruta aleatoria entre dos nodos de origen (o más) y destino respectivamente; Este puede servir como herramienta auxiliar en un sistema de recomendaciones, recolectando los nodos por los que pasa en su camino para aplicar análisis posteriores sobre los mismos.
- Minimum Weight Spanning Tree: Este algoritmo puede usarse para resolver el problema del vendedor viajero, ya que toma un conjunto de nodos que tienen que ser visitados sin excepción. La ventaja del actual es que no importa la dirección de las relaciones, extendiendo las posibilidades de su implementación en entornos con mayor diversidad.
- Depth First Search: El último algoritmo a implementar es una variación del anterior. La diferencia radica en que el actual visita todos los nodos con mayor nivel de profundidad en una estructura de árbol. La ventaja agregada con la que cuenta es que su ejecución resulta eficiente en términos de tiempo y memoria.

Para la fácil presentación de los resultados se hará uso de *Bloom* (s.f.), una herramienta de visualización en tiempo real del grafo analizado.

Estructura básica de uso de GDS

Como ya es costumbre, se presenta el esquema básico de la implementación de los algoritmos a través de GDS:

```
// Creación de subgrafo/proyecto
CALL gds.graph.project(
    'nombre_del_proyecto',
    ['TIPOS_DE_NODOS'],
    {
        NOMBRE_DE_RELACION: {
```

```

        orientation: 'ORIENTACION_DE_LAS_RELACIONES [NATURAL | REVERSE | UNDIRECTED]' ,
        properties: 'PROPIEDAD(ES)_DE_LAS_RELACIONES'
    }
}

);

// Cálculo de memoria requerida (Opcional y aplicable únicamente hacia algoritmos con
// calidad de producción)
CALL gds.ALGORITMO_A_APPLICAR.write.estimate(
    'nombre_del_proyecto',
    {writeProperty: 'PROPIEDAD_A_CALCULAR'}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// Ejecución del algoritmo
CALL gds.[CALIDAD_DEL_ALGORITMO.]ALGORITMO_A_APPLICAR.stream('nombre_del_proyecto')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).PROPIEDAD_ORIGINAL_DEL_NODO As Name, score
ORDER BY score DESC, Name ASC;

// Creación de atributo con resultado individual
CALL gds.[CALIDAD_DEL_ALGORITMO.]ALGORITMO_A_APPLICAR.write(
    'nombre_del_proyecto',
    {writeProperty: 'PROPIEDAD_A_ESCRIBIR'}
)
YIELD centralityDistribution, nodePropertiesWritten
RETURN
    centralityDistribution.min AS MinScore,
    centralityDistribution.max AS MaxScore,
    centralityDistribution.mean AS MeanScore,
    nodePropertiesWritten;

```

Nota: Muchos de los algoritmos a implementar no usan la plantilla estructural para obtener sus resultados, sin embargo, como regla general, es buena práctica seguir estos lineamientos como punto de partida para cualquier otra invocación de GDS. En este caso, la parte más crítica en donde vale la pena invertir tiempo de investigación para una subsecuente implementación es la estimación de memoria, ya que los algoritmos presentes pueden ser muy agresivos en cuestión de consumo de recursos computacionales.

Modificación estructural previa

Sin lugar a duda, la búsqueda de un conjunto de datos adecuado para el proyecto ha sido uno de los pasos más tediosos y difíciles, incluso impidiendo el inicio de su desarrollo preliminar. Los datos seleccionados ponen a prueba la sanidad mental del autor de estas palabras y en duda no solo la capacidad de llevar a cabo lo solicitado de manera integra, sino que también la habilidad del raciocinio común.

Un paso adicional previo a la aplicación de los algoritmos es una ligera modificación estructural al esquema general del grafo, esto motivado por el hecho de que las únicas relaciones con peso dentro del mismo se encuentran entre usuarios y artistas. El siguiente código atiende esta cuestión:

```

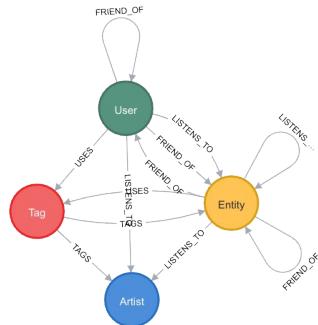
MATCH (n:Artist|User)
SET n:Entity;                                         Added 10740 labels, completed after 386 ms.

MATCH (n:Entity)-[r:FRIEND_OF]->(m:Entity)
SET r.weight = 1;                                     Set 25434 properties, completed after 109 ms.

MATCH (t:Tag)-[r:TAGS]->(a:Artist)
SET r.weight = 1;                                     Set 81883 properties, completed after 365 ms.

```

Con seis simples líneas de código se resuelve el problema anterior, y afortunadamente, se consigue en tiempo mínimo. Aunque esto resulta conveniente, las relaciones existentes también se ven afectadas con este cambio adicional, algo para tomar en cuenta en la siguiente sección.



Aplicación de los algoritmos

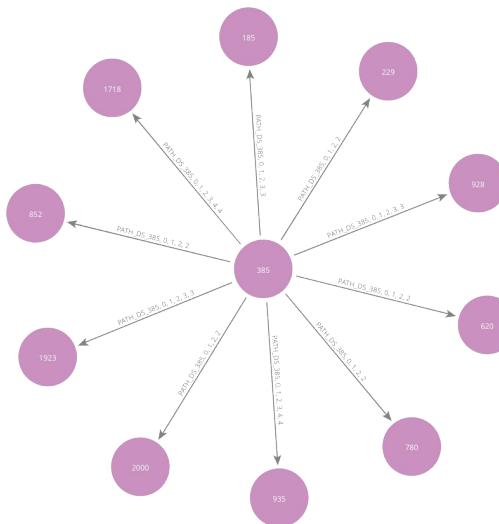
| 1 | |
|---|--|
| ¿Cuál es el camino más corto desde un nodo hacia todos los demás? (Delta-Stepping) | ¿Cuáles son las interacciones más rápidas para llegar desde un usuario hacia todos los contactos de sus amistades? |
| <pre>// Frase en Bloom: Caminos cortos desde usuario \$ID hacia \$n usuarios // Implementación dirigida</pre> | |

```

MATCH (u:User)-[r:PATH_DS_385]->(v:User)
WHERE u.id = $ID
RETURN u, r, v
LIMIT $n;

```

Parámetro de GDS → user.id = 385
 Parámetros de Bloom → ID = 385, n = 10



2

¿Cuál es el camino más costoso de todos los caminos cortos generados? (Delta-Stepping)

```

// Frase en Bloom: Camino más costoso desde usuario $ID
MATCH (u:User)-[r:PATH_DS_385]->(v:User)
WHERE u.id = $ID
RETURN u.id AS Origen, v.id AS Destino, r.totalCost AS Costo
ORDER BY Costo DESC, Destino DESC
LIMIT 1;

```

// Frase en Bloom: Camino más costoso desde usuario \$ID

```

MATCH (u:User)-[r:PATH_DS_385]->(v:User)
WHERE u.id = $ID
RETURN u, v, r
ORDER BY r.totalCost DESC, v.id DESC
LIMIT 1;

```

¿Con cuál usuario sería más difícil entablar una amistad?

Parámetro de GDS → user.id = 385

Parámetro de consulta → u.id = 385; Parámetro de Bloom → ID = 385

| Origen | Destino | Costo |
|--------|---------|-------|
| 385 | 1962 | 6.0 |



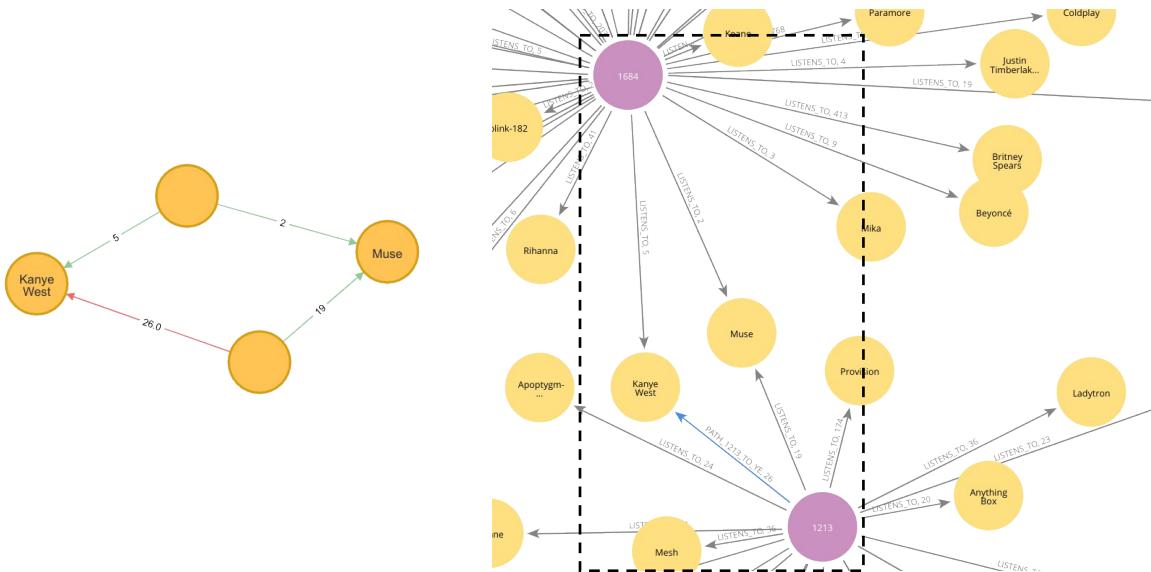
3

Suponiendo relaciones bidireccionales entre nodos ¿Cuál es una forma alterna de obtener el camino más corto desde un origen hacia otro nodo específico? (Dijkstra)

¿Cuál es el camino más corto que existe entre un usuario arbitrario y el artista Kanye West?

```
// Frase en Bloom: Dijkstra desde usuario $ID hacia artista $name
MATCH (u:User)-[r:PATH_1213_TO_YE]-(a:Artist), (n:Entity)
WHERE ID(n) IN r.nodeIds
WITH u, r, a, n
WHERE u.id = $ID
AND a.name = $name
RETURN u, r, a, n;
```

Parámetros de GDS y Bloom → user.id = 1213, artist.name = Kanye West



La suma de los pesos encontrados en las relaciones utilizadas es de 26, lo cual se refleja en la ejecución del algoritmo. Esto se interpreta como el gusto compartido por el usuario seleccionado

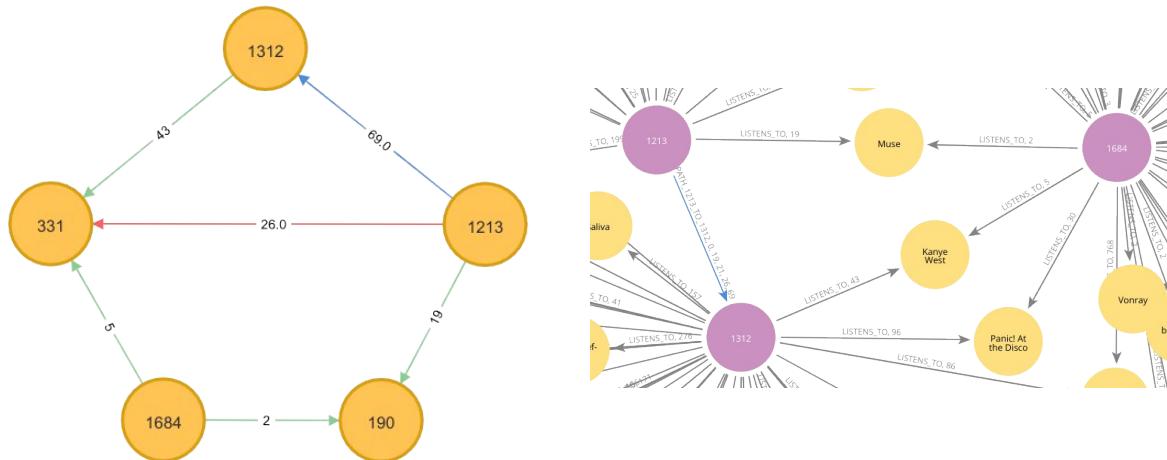
con otro a través de la banda *Muse*; con base en esa coincidencia, se puede llegar al artista originalmente establecido, recordando que la ejecución realizada no contó con dirección.

4

¿Cuáles nodos se encuentran en común desde un origen hacia un destino del mismo tipo?
(Dijkstra)

```
// Frase en Bloom: Dijkstra desde usuario $ID1 hacia usuario $ID2
MATCH (u1:User)-[r:PATH_1213_TO_1312]-(u2:User), (n:Entity)
WHERE ID(n) IN r.nodeIds
WITH u1, r, u2, n
WHERE u1.id = $ID1
AND u2.id = $ID2
RETURN u1, r, u2, n;
```

Parámetros de GDS y Bloom → u1.id = 1213, u2.id = 1312



Inadvertidamente, se han seleccionado dos nodos que posiblemente pertenezcan al mismo círculo social, por ende, se han obtenido los mismos resultados, en donde los artistas recolectados son KanYe West y Muse.

5

¿Qué nodos de un tipo específico se recolectan al realizar una caminata aleatoria partiendo desde un par de nodos definidos? Usando una distancia de 5 pasos (Random Walk)

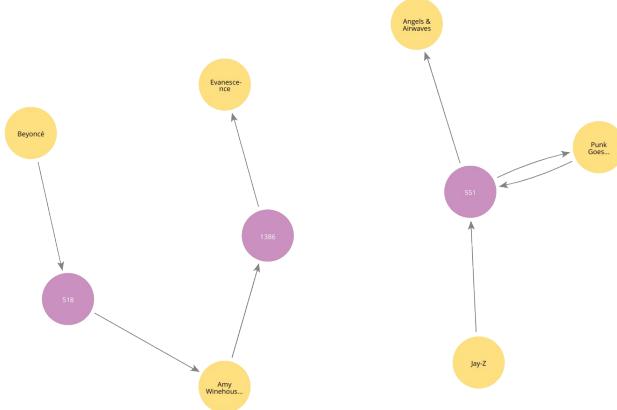
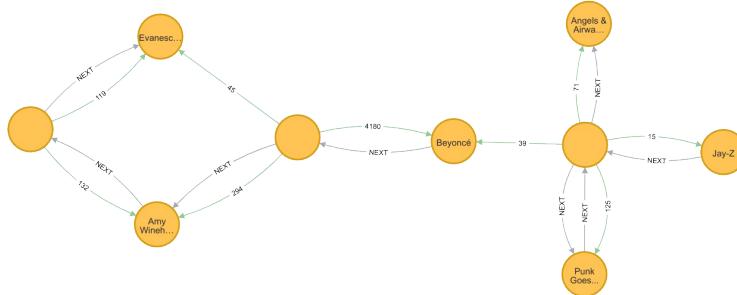
```
// Frase en Bloom: Caminata aleatoria de $n pasos usando a $a1 y $a2
MATCH (a:Artist)
```

¿Que artistas surgen a partir de un recorrido al azar utilizando a Beyoncé y Jay-Z como puntos de partida? Máximo 5 resultados

```

WHERE a.name IN [$a1, $a2]
WITH COLLECT(a) AS sourceNodes
CALL gds.randomWalk.stream(
  'RandomWalk-U',
  {
    sourceNodes: sourceNodes,
    walkLength: $n,
    walksPerNode: 1,
    randomSeed: 385,
    concurrency: 1
  }
)
YIELD nodeId, path
RETURN nodeId, path;
  
```

Parámetros de GDS y Bloom → a1 = Jay-Z, a2 = Beyoncé



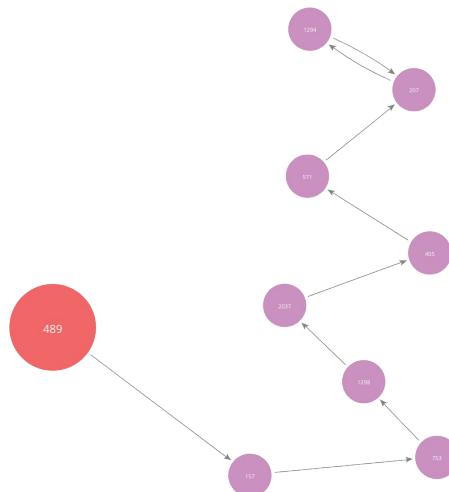
Las ejecuciones muestran resultados distintos ya que, como el algoritmo indica, cada caminata resulta en un descubrimiento distinto en la mayoría de las ocasiones. Estos resultados pueden utilizarse para encontrar similitudes a través de orígenes diversos sin la necesidad de modificar la estructura del grafo.

¿Qué otro tipo de caminatas aleatorias se pueden implementar? (Random Walk)

¿Cuáles usuarios son alcanzables cuando se hace una búsqueda aleatoria en la plataforma con al menos 10 usuarios?

```
// Frase de Bloom: Caminata aleatoria desde usuario $usr con $n pasos
MATCH (u:User)
WHERE u.id = $usr
WITH COLLECT(u) AS sourceNodes
CALL gds.randomWalk.stream(
    'RandomWalk-U-1',
    {
        sourceNodes: sourceNodes,
        walkLength: $n,
        walksPerNode: 1,
        randomSeed: 727272,
        concurrency: 1
    }
)
YIELD nodeIds, path
RETURN nodeIds, path;
```

Parámetros de GDS y Bloom → usr = 489, n = 10



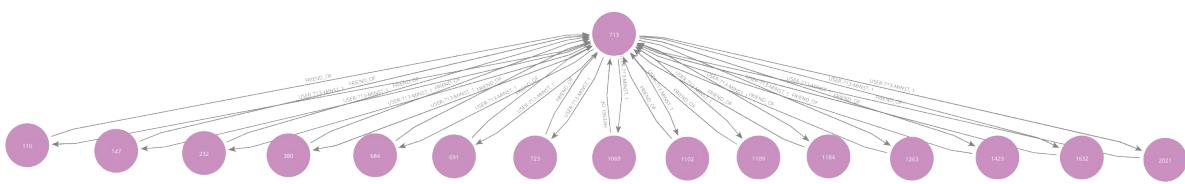
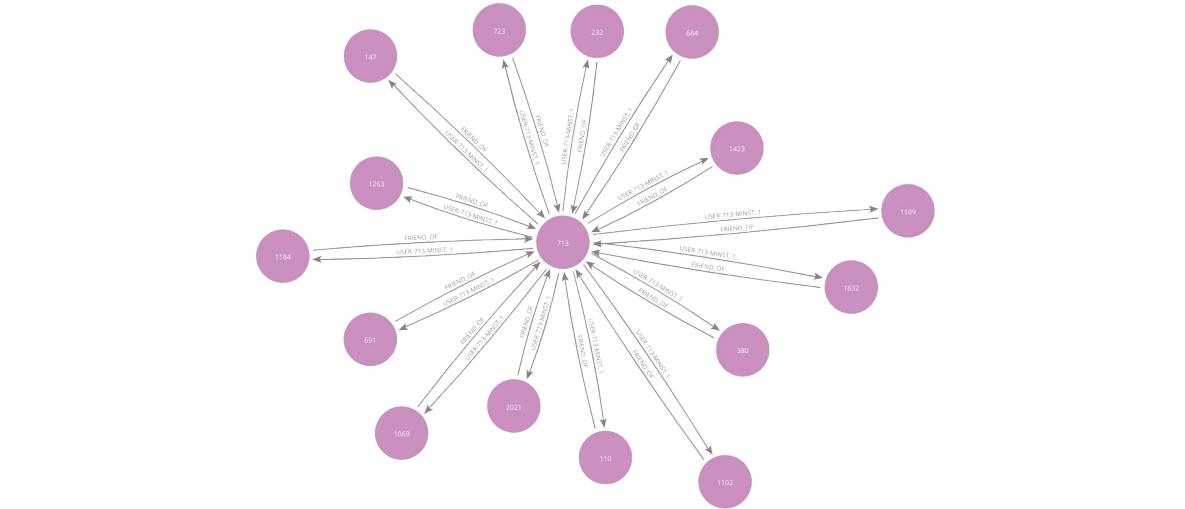
De nuevo como proceso intermedio para un proceso de recomendación, los nodos obtenidos en esta ocasión pueden regresar a los artistas que escuchan y aplicar un procesamiento posterior para sugerir artistas destacados que escuchan ciertos miembros de la aplicación.

¿Cuál es el árbol mínimo para un nodo?
(Spanning Tree)

¿Cuál sería una manera eficaz de compartir el nuevo lanzamiento de un álbum dentro de la plataforma desde un usuario inicial?

```
// Frase de Bloom: MWST de usuario $id
MATCH (source:User)-[r:USER-713-MINST]-(leaf:User)
WHERE source.id = $id
RETURN source, r, leaf;
```

Parámetro → source.id = 713

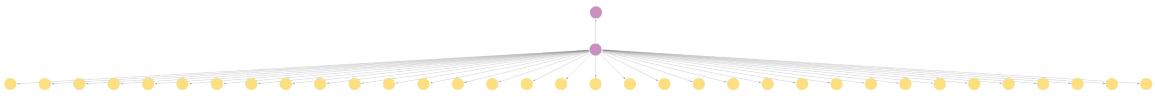


En este caso, como se ha de recordar, los pesos de las relaciones entre usuarios siempre es de 1, por ende, el resultado es bastante plano, lo cual puede corroborarse desde la vista jerárquica. La solución más burda a esto sería asignar pesos aleatorios a las relaciones para simular diferentes niveles de amistad o relaciones duraderas, aún así, el propósito demostrativo se cumple, y se sabe que utilizando este método se puede obtener el círculo social con mejor nivel de difusión.

¿Cuál es el árbol mínimo de un nodo que cuente con al menos 3 niveles de profundidad?
(Spanning Tree)

¿Cuáles son los fanáticos y colegas inmediatos partiendo desde un usuario dentro de la plataforma?

```
// Frase de Bloom: MWST extendido de entidad $id
MATCH (source:User)-[r:MINSTONES]-(leaf)
WHERE source.id = $id
RETURN source, r, leaf;
```



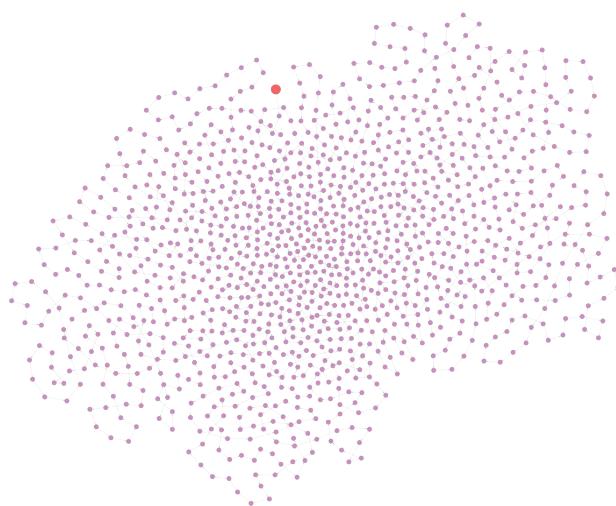
9

¿Qué nodos resultan de una búsqueda a profundidad de 5 niveles? (Depth First Search)

¿Cuántos nuevos amigos pueden encontrarse al indagar a través de 5 usuarios partiendo desde otro usuario?

```
// Frase de Bloom: DFS a $n niveles desde $id
MATCH (source:User {id: $id})
CALL gds.dfs.stream(
    'DFS-U',
    {
        sourceNode: source,
        maxDepth: $n
    }
)
YIELD path RETURN path;
```

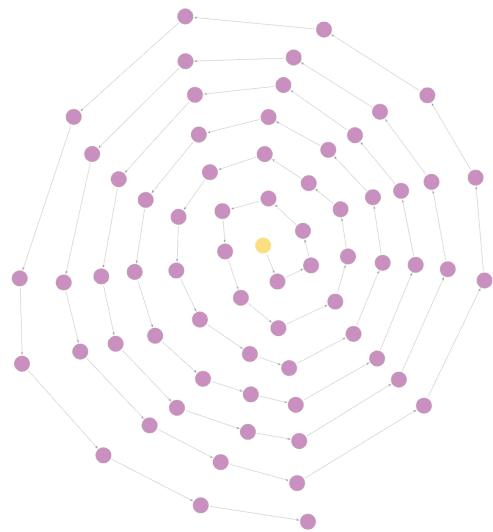
Parámetros de GDS y Bloom → id = 24, n = 5



Aunque no es visible en el resultado, contando el origen, el resultado de esta ejecución fue de

1258 nodos de un total de 1892, es decir que si a este resultado se le tratara como una estructura de árbol, no haría falta demasiada profundidad en sus niveles para alcanzar a todos los usuarios posibles dentro de la plataforma. Si Kevin Bacon estuviera registrado como artista o usuario, seguramente saldría muy rápido dentro de la búsqueda.

| | |
|--|--|
| 10 | |
| <p>¿Qué nodos resultan de la exploración superficial de otro tipo de nodo? (Depth First Search)</p> <pre>// Frase de Bloom: Exploración inicial con DFS para \$name MATCH (source:Artist {name: \$name}) CALL gds.dfs.stream('DFS-D', { sourceNode: source, maxDepth: 1 }) YIELD path RETURN path;</pre> | <p>¿Qué usuarios resultan prominentes al hacer una búsqueda preliminar desde un artista?</p> |
| Parámetros de GDS y Bloom → name = Deftones | |



Por cuestiones de rendimiento, la vista jerárquica de estos resultados no es posible, sin embargo, se puede ver que la banda seleccionada está fuertemente conectada ya desde el inicio, arrojando 66 posibles conexiones de interés con usuarios, una métrica relevante para cuestiones de marketing o promoción dentro de la plataforma.

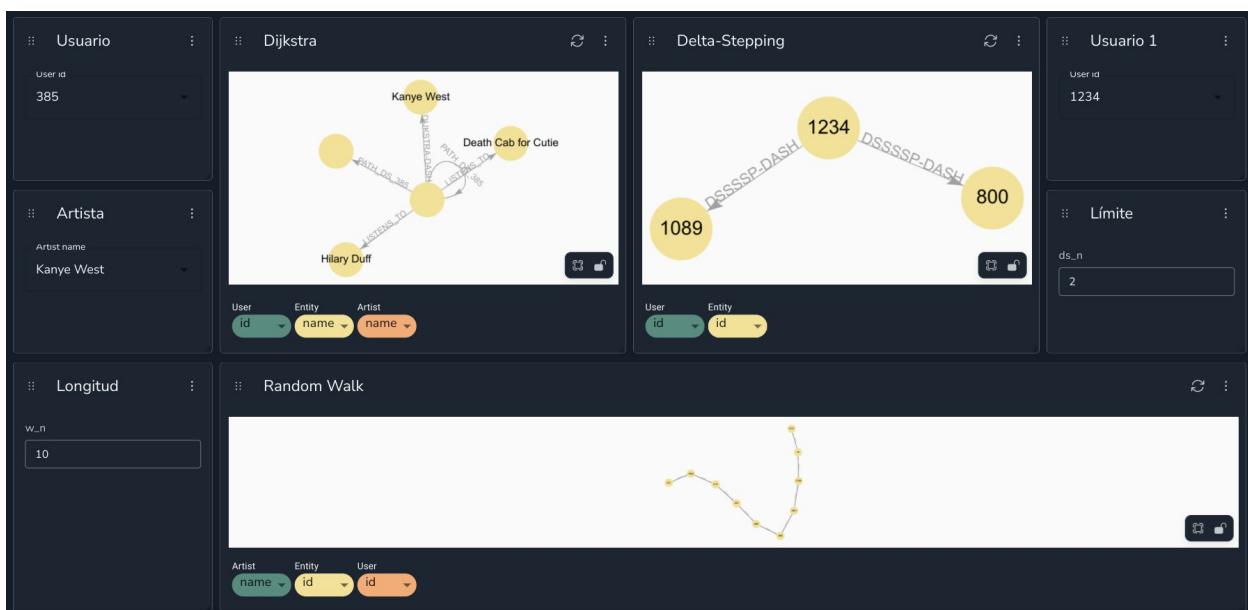
Presentación de los resultados

Los resultados obtenidos en esta ocasión pretenden ser demostrados a través de materiales de apoyo visual en un momento únicamente dedicado a ese propósito, en donde se habrá de mostrar unos tableros realizados a través de NeoDash (de Jong et al, s.f.) en su versión 2.4.0, una herramienta de Business Intelligence capaz de trasladar código en lenguaje Cypher directamente hacia componentes visuales, lo cual es idóneo para la presente.

Tablero de análisis básico



Tablero de caminos



Conclusiones y recomendaciones

En la introducción ya se hablaba del algoritmo de Dijkstra; esta invención, casi como si se tratase de una serendipidad, trae consigo una reflexión muy bella de la mano de su propio autor:

“Sin lápiz y papel, estás casi forzado a evitar todas las complejidades evitables”

Esto por supuesto hace referencia a la magnífica historia de origen del algoritmo mencionado, el cual fue diseñado a través de un fascinante y exclusivo proceso mental. Esta reflexión es aplicable a tantos aspectos de la vida, que independientemente de si cuentan con un carácter académico, lograrían las soluciones más eficientes para los problemas más intrínsecos. Tal es el ejemplo del conjunto de datos utilizado en el desarrollo demostrado, el cual trajo consigo una reminiscencia de proyectos pasados, en donde se vio el ejercicio sumamente interesante de sistemas primitivos de recomendaciones objetivas dentro de plataformas digitales, puesto que algunos de los algoritmos implementados en esta ocasión incluso podrían utilizarse como soporte o reemplazo total de aquellos algoritmos de similaridad y comunidad.

Lamentablemente, el tiempo nunca ha sido un aliado, y en esta ocasión, la variable independiente por defecto ha propiciado una implementación un tanto rudimentaria debido a la premura de la selección del conjunto de datos explorado, ya que, como se pudo ver, lamentablemente no todos los componentes del grafo general fueron utilizados. Esto sin duda alguna tiene toda la pinta de excusa y por encima de todo puede parecer que demerite el progreso logrado hasta este punto.

A pesar de las adversidades enfrentadas por un tratamiento quizás superficial de un set de datos de dudosa calidad, los resultados obtenidos han aportado a dos propósitos estrechamente relacionados: la aplicación de algoritmos revisados en clase relacionados a caminos, y el aprendizaje general de lo que conllevó esto. Como en todos los desarrollos pasados, los retos encontrados en esta ocasión han sido frustrantes e incluso han puesto en tela de juicio la capacidad personal para realizar las operaciones más básicas, no obstante, enfrentarse y resolver este tipo de retos al final propiciará un conocimiento enfocado y más especializado, el cual se puede decir con toda seguridad que transgredirá más allá de lo revisado en el ámbito presente.

Pasando a las recomendaciones para la plataforma, suponiendo una colaboración activa dentro de la plataforma, la principal sugerencia que con latencia ferviente se extiende es la inversión en pesos en todas y cada una de las relaciones presentes de forma nativa, esto es posible de forma sencilla. Por ejemplo, para las relaciones de amistad, un análisis mínimamente exhaustivo entre un par de usuarios conectados puede conducirse con cada interacción entre ellos, de esta forma, las conexiones existentes de amistad se aprovecharían para realizar investigaciones más exhaustivas. De parte de la implementación, existe una ambigüedad en cuanto a la calidad de la fuente de datos y la implementación demostrada, ya que los nodos de tipo “Tag” no lograron ser usados, sin embargo, llevar a cabo esto resultó ser una tarea bastante compleja debido al poco valor agregado que aparentemente demostraba, al menos desde la forma en que se ingresaron al grafo manipulado.

El portento ominoso del cual se hablaba en la introducción hace referencia a nada más y nada menos que a los algoritmos, más específicamente a su categoría, los caminos. Este espacio de reflexión final tiene es particularmente especial, ya que resulta ser uno de los últimos de la carrera en donde se ha tenido la oportunidad de hacer grandes investigaciones y adquirir conocimientos invaluables de todo tipo de áreas del saber. Si la vida fuera fácil, este sería el final absoluto, sin embargo, como ya se ha mencionado, el tiempo no está de nuestro lado, y es momento de emprender un camino adicional, solo que está vez sin Neo4j... Tal vez.

La ingeniería ha estado plagada por doquier de retos que hace algunos años hubiera parecido imposibles, pero a medida que uno de tantos caminos se fueron explorando, nuevos horizontes fueron descubiertos sin siquiera buscarlos. En los días más tempranos de este trayecto profesional, ver un grafo con tantos nodos como los que se han visto en la clase seguramente hubieran inducido una locura y una incomodidad con un porqué incierto, pensando en cómo es posible lograr eso, llegando incluso al extremo filosófico de cuestionar si nosotros programamos a las computadoras, o si en realidad es lo inverso. Con el progreso y la aplicación del conocimiento, se ha llegado a las soluciones más burdas posibles, pero al final funcionales ¿Cómo es esto posible? Evitando todas las complejidades evitables. En el caso actual, el papel y lápiz de Dijkstra han sido las propias barreras impuestas del cuestionamiento de la posibilidad de la consolidación del objetivo, es decir, preguntarse si es posible hacerlo, y la holgura final que ha hecho posible llegar hasta este punto es la ignorancia de la posibilidad para ser reemplazada por el cómo se haría.

Con esta última reflexión, se hace una agridulce despedida, recordando tantos momentos de angustia, sobrellevados por la recompensa que viene después. El mundo que está a punto de abrirse es incierto y desconocido, así como muchos grafos vistos en estos proyectos, pero se sabe perfectamente que las herramientas para desarmarlo con la esperanza de llegar a un nivel aceptable de su entendimiento están disponibles, es cuestión de saber cómo implementarlas, para así poder decidir qué camino tomar. A partir de ahora ya no existe la pregunta de si algo es posible, sino que ahora hay que preguntar cómo se haría.

Gracias.

Bibliografía

Paoletti, T. (2006). *Leonard Euler's Solution to the Konigsberg Bridge Problem*. Recuperado de <https://maa.org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem>.

Frana, P. L., & Misa, T. J. (2010). An interview with Edsger W. Dijkstra. Communications of the ACM, 53(8), 41–47. <https://doi.org/10.1145/1787234.1787249>.

Cummings, N. (2000). *A brief History of the Travelling Salesman Problem*. Recuperado de <https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/>.

Neo4j. (s.f.). Neo4j Graph Database. Recuperado de <https://neo4j.com/product/neo4j-graph-database>.

Neo4j. (s.f.). The Neo4j Graph Data Science Library Manual v2.5. Recuperado de <https://neo4j.com/docs/graph-data-science/current/>.

Last.fm. (s.f.). Acerca de Last.fm. Recuperado de <https://www.last.fm/es/about>.

Neo4j. (s.f.). Neo4j Bloom. Recuperado de <https://neo4j.com/product/bloom/>.

de Jong, N., Conjeaud, M., Agudelo Ramirez, H., et al. (s.f.). NeoDash - Dashboard Builder for Neo4j. Recuperado de <https://neo4j.com/labs/neodash/>.