

# **Universidad Autónoma de Yucatán**

## **Facultad de Matemáticas**



**LIS - 4° Semestre**  
**Estructura de Datos**

**Reporte Técnico**

**Docente**

Basto Díaz Luis Ramiro

**Alumnos - Equipo 1 - Matrícula**

Aguilar Ramírez Ian - A21216391

Arreola Hernández Diego Francisco - A21216376

**Ciclo escolar 2022-2023**

**Lunes 29 de mayo del 2023**

## Índice

<b>Descripción del Dataset</b>	<b>2</b>
<b>Funcionalidad del Algoritmo</b>	<b>3</b>
<b>Definiciones y Llamadas de las Funciones</b>	<b>4</b>
<b>Comandos Cypher</b>	<b>6</b>
<b>Referencias Bibliográficas</b>	<b>10</b>

## Descripción del Dataset

El dataset a utilizar durante el desarrollo del algoritmo Dijkstra Source-Target Shortest Path en la plataforma Neo4j.com, consiste en una Red firmada ponderada de confianza de Bitcoin Alpha; es decir, es el dataset de una red de personas que confían en quién que comercia con Bitcoin en una plataforma llamada Bitcoin Alpha. Dado que los usuarios de Bitcoin son anónimos, es necesario mantener un registro de la reputación de los usuarios para evitar transacciones con usuarios fraudulentos y riesgosos. Los miembros de Bitcoin Alpha califican a otros miembros en una escala de: -10 (desconfianza total) a 10 (confianza total) en pasos de 1. Esta es la primera red dirigida con signo ponderada explícita disponible para la investigación.

Estadísticas del Dataset	
<b>Nodos</b>	3, 783
<b>Relaciones</b>	24, 186
<b>Rango de peso por relación</b>	-10 a +10
<b>Porcentaje de relaciones positivas</b>	93%

El formato del dataset es el siguiente: SOURCE, TARGET, RATING.  
Dónde:

- SOURCE: ID de nodo origen
- TARGET: ID de nodo destino
- RATING: La clasificación del origen para el destino, que va de -10 a +10 en pasos de 1

*\*Para el correcto funcionamiento del algoritmo se hizo un cambio: Se cambiaron todos los números negativos a 0, ya que el algoritmo no acepta valores negativos. También se utilizó una versión sin peso del dataset.*

## Funcionalidad del Algoritmo

El algoritmo Dijkstra Source-Target Shortest Path o Ruta más corta de origen-destino de Dijkstra, es un algoritmo clásico utilizado para encontrar la ruta más corta desde un nodo de origen (source) hacia un nodo de destino (target) en un grafo ponderado dirigido o no dirigido. Utiliza una estrategia de exploración de los nodos adyacentes para encontrar la ruta más corta de forma eficiente. Este algoritmo se utiliza en dispositivos GPS para encontrar la ruta más corta entre la ubicación actual y el destino. Tiene amplias aplicaciones en la industria, especialmente en dominios que requieren modelado de redes.

Rasgos del algoritmo:

- *Rasgo dirigido.* El algoritmo está bien definido en un gráfico dirigido.
- *Rasgo no dirigido.* El algoritmo está bien definido en un gráfico no dirigido.
- *Rasgo homogéneo.* El algoritmo tratará todos los nodos y relaciones en sus gráficos de entrada de manera similar, como si fueran todos del mismo tipo.
- *Rasgo ponderado.* El algoritmo admite la configuración para establecer propiedades de relación y/o nodo para usar como pesos. El algoritmo considerará por defecto cada nodo y/o relación como igualmente importante.

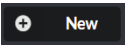
El algoritmo solo puede funcionar con gráficos que tienen pesos positivos. Esto se debe a que, durante el proceso, se deben sumar los pesos de los bordes para encontrar el camino más corto. Si hay un peso negativo en el gráfico, entonces el algoritmo no funcionará correctamente. Una vez que un nodo se ha marcado como "visitado", la ruta actual a ese nodo se marca como la ruta más corta para llegar a ese nodo. Y los pesos negativos pueden alterar esto si el peso total se puede disminuir después de que se haya producido este paso.




Descripción de la funcionalidad del algoritmo:

1. *Inicialización:* El algoritmo comienza asignando una distancia inicial infinita a todos los nodos del grafo, excepto al nodo origen que se inicializa con una distancia de 0. Además, se crea un conjunto vacío de nodos visitados.
2. *Bucle principal:* Luego, el algoritmo realiza un bucle mientras haya nodos sin visitar, para hacer un seguimiento de la distancia más corta actualmente conocida de cada nodo hasta el nodo origen y actualizar estos valores si encuentra una ruta más corta. En cada iteración, se selecciona el nodo "no visitado" con la distancia más pequeña; es decir, el nodo más cercano al origen, entonces en consecuencia dicho nodo se marca como "visitado" y se agrega a la ruta.

3. *Actualización de distancias*: Para cada nodo adyacente al nodo seleccionado, se calcula la distancia acumulada sumando la distancia del nodo seleccionado al nodo adyacente. Si esta distancia acumulada es menor que la distancia actualmente asignada al nodo adyacente, se actualiza la distancia. Esto se hace para garantizar que siempre se esté siguiendo la ruta más corta.
4. *Repetición del bucle*: Después, se repite el bucle principal hasta que se hayan visitado todos los nodos del grafo o hasta que se llegue al nodo de destino. Durante el proceso, se van actualizando las distancias de los nodos adyacentes y se va construyendo la ruta más corta desde el origen hacia el destino.
5. *Reconstrucción de la ruta más corta*: Por último, una vez que se ha alcanzado el nodo de destino o se han visitado todos los nodos, se reconstruye la ruta más corta desde el origen hasta el destino utilizando la información de las distancias y los nodos precedentes. Obteniendo una ruta que conecta el nodo origen con todos los demás nodos siguiendo el camino más corto posible para llegar a cada nodo. Esta ruta representa el camino óptimo desde el nodo de origen hasta el nodo de destino.

## Definiciones y Llamadas de las Funciones

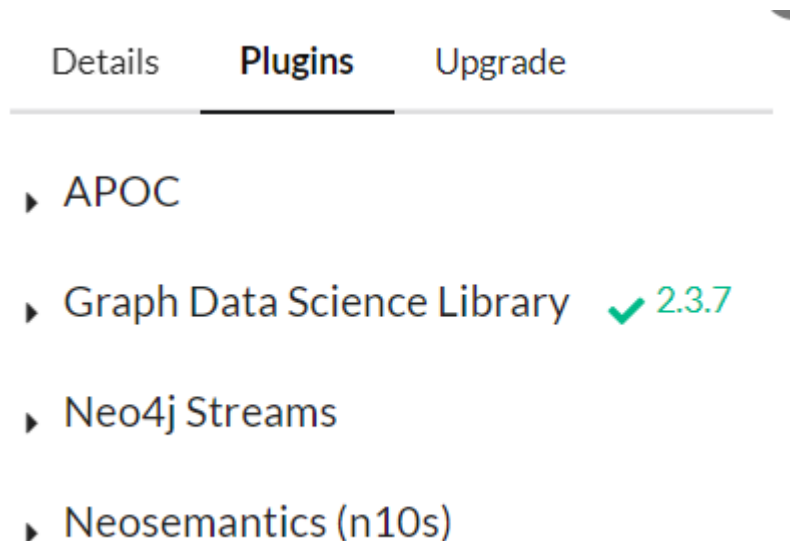
El primer paso que se debe hacer antes de ingresar los comandos para las funciones del algoritmo Dijkstra Source-Target Shortest Path es crear un nuevo proyecto en el programa de software que se utilizará para ejecutar el algoritmo, llamada Neo4j Desktop el cual es una base de datos para grafos. Para esto, abriremos el Neo4j Desktop, que antes debió ser descargado, haremos clic en el botón , después en “create project” y listo se creará un nuevo proyecto; también se puede cambiar el nombre del proyecto, nosotros cambiamos el nombre del proyecto a “Dijkstra Source-Target Shortest Path”.

El segundo paso es agregar una base de datos al proyecto, para crear una agregar la base de datos hay que hacer clic en el botón , luego en “Local DBMS”, después se le asigna un nombre y una contraseña a la base de datos y por último se oprime el botón . Ahora, hay que iniciar la base de datos oprimiendo el botón , una vez iniciada la base de datos debe aparecer lo siguiente:

 ACTIVE

   ...

El tercer paso es descargar el librería “Graph Data Science Library” y cargar el dataset, para descargar el plugin hay que hacer clic en el nombre de la base de datos lo que abrirá una pestaña, en esa pestaña haremos clic en el apartado de “Plugins”, luego en “Graph Data Science Library” y oprimiremos el boton “Install” para instalar la librería:



Para descargar el dataset simplemente hay que hacer clic en en los tres puntos que se encuentran en la esquina superior derecha del apartado de la base de datos creada, a continuación, haremos clic en “Open folder” y luego en “Import”, esto abrirá la ubicación de la carpeta Import, lo que hay que hacer es guardar el dataset en esa ubicación y listo.

El cuarto y último paso es abrir el Neo4j Browser, para esto sencillamente hay que hacer clic en el botón “Open” lo que abrirá la ventana del Browser.

Ahora ya en el Browser, para hacer la llamada del algoritmo hay que usar el comando:

***CALL gds.shortestPath.dijkstra.stream()***

Al cual se le da como parámetro de entrada:

- El nombre de la proyección del grafo
- El nodo fuente
- El nodo destino

*\*En el caso de que el grafo sea ponderado se le deberá pasar un parámetro adicional adicional: el nombre de la propiedad de las relaciones (Ej: distancia, costo, etc).*

El algoritmo puede retornar un número de resultados:

- El nodo fuente
- El nodo destino
- El costo total
- La lista de nodos en el camino y sus costos

## Comandos Cypher

Una vez que se haya cargado correctamente el dataset al proyecto crearemos el grafo con los datos del archivo CSV usando los siguientes comandos y los ingresamos en este apartado:

```
neo4j$
```

```
LOAD CSV FROM "file:///soc-sign-bitcoinalpha.csv" AS line  
MERGE (a:user {id: line[0]})  
MERGE (b: user { id: line[1]})  
MERGE ((a)-[:rate{cost: toInteger(line[2])}]->(b))
```

Donde:

- **LOAD:** Carga los datos en la base de datos de Neo4j desde una fuente externa. Importa los datos desde el archivo CSV. También, permite mapear las columnas de los datos de origen a propiedades de nodos o relaciones en la base de datos.
- **MERGE:** Combina o crea elementos en la base de datos según las condiciones que se le han dado. Busca un patrón en la base de datos y avisa si no existe, para así poder crearlo; o si existe, actualizarlo.

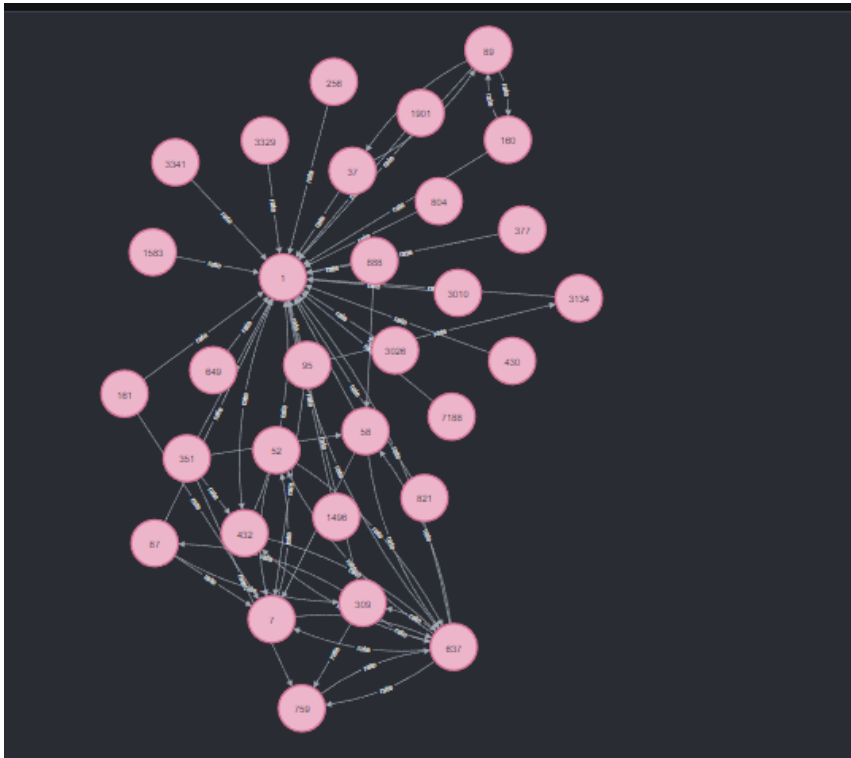
Salida:

```
neo4j$ LOAD CSV FROM "file:///soc-sign-bitcoinalpha.csv" AS line MERGE (a:user {id: line[0]}) MERGE (b: user { id: line[1]}) MERGE ((a)-[:rate{cost: toInteger(lin...
```



Added 3783 labels, created 3783 nodes, set 27969 properties, created 24186 relationships, completed after 27924 ms.

Added 3783 labels, created 3783 nodes, set 27969 properties, created 24186 relationships, completed after 27924 ms.



Una vez creado el grafo, con los siguientes comandos crearemos una proyección o copia del grafo creado anteriormente el cual, dicha proyección es necesaria para ejecutar el algoritmo:

```
CALL gds.graph.project( 'mygraph','user', 'rate',{ relationshipProperties:
toInteger('cost') }
) YIELD
    graphName, nodeProjection, nodeCount, relationshipProjection,
    relationshipCount, projectMillis
```

Donde:

- **CALL:** Invoca y ejecuta procedimientos almacenados en la base de datos de Neo4j. Los procedimientos almacenados son fragmentos de código reutilizable que realizan operaciones específicas y pueden recibir argumentos y devolver resultados.
- **YIELD:** Devuelve los resultados de la consulta hecha en Cypher. Lo utilizamos para especificar los datos que queríamos obtener como resultado de la consulta. También, se puede usar para devolver propiedades de nodos o relaciones, contar elementos, realizar cálculos y más. Los resultados devueltos por YIELD pueden ser utilizados por otros comandos o consultas posteriores.



Salida:

```
neo4j$ CALL gds.graph.project('mygraph','user','rate',{ relationshipProperties: toInteger('cost') }) YIELD graphName, nodeProjecti...
```

	graphName	nodeProjection	nodeCount	relationshipProjection	relationshipCount	projectMillis
1	"mygraph"	{ "user": { "label": "user", "properties": { } } }	3783	{ "rate": { "orientation": "NATURAL", "indexInverse": false, "aggregation": "DEFAULT", "type": "rate", "properties": { } } }	48372	4908

Started streaming 1 records after 7 ms and completed after 4919 ms.

Con la proyección podemos llamar y ejecutar el algoritmo, usando estos comandos:

```
MATCH (source:user {id: '10'}), (target:user {id: '20'})  
CALL gds.shortestPath.dijkstra.stream('mygraph', {  
  sourceNode: id(source),  
  targetNode: id(target)  
})  
YIELD nodeIds, costs  
RETURN [id IN nodeIds | gds.util.asNode(id).id] AS path, costs;
```

Donde:

- **MATCH:** Busca patrones en la base de datos, para recuperar ciertos datos en la base de datos sin modificarlos. Es uno de los comandos más importantes de Cypher y lo utilizamos para encontrar los nodos y relaciones que cumplan las condiciones que le dimos para la consulta.
- **RETURN:** Lo usamos para especificar los datos que queríamos recibir como resultado de la consulta en Cypher. Devuelve los nodos, relaciones, propiedades de nodos o relaciones, resultados de cálculos, en este caso la suma. Es el último comando en una consulta Cypher y se encarga de devolver los resultados al usuario.

Salida:

```
neo4j$ MATCH (source:user {id: '10'}), (target:user {id: '20'}) CALL gds.shortestPath.dijkstra.stream('mygraph', { sourceNode: id(sou...
```

	path	costs
1	["10", "2", "20"]	[0.0, 1.0, 2.0]

Started streaming 1 records after 13 ms and completed after 26 ms.

## Referencias Bibliográficas

- Neo4j Docs. (2023). Dijkstra Source-Target Shortest Path. Recuperado de: <https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-source-target/#algorithms-dijkstra-source-target-examples-memory-estimation>
- Cassingena Navone, E. (2020). Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction. Recuperado de: <https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/>
- Kumar, S., Spezzano, F., Subrahmanian, V.S., Faloutsos, C. (2016). IEEE International Conference on Data Mining (ICDM). Recuperado de: [Edge Weight Prediction in Weighted Signed Networks](#)
- Kumar, S., Hooi, B., Makhija, D., Kumar, M., Subrahmanian, V.S., Faloutsos, C. (2018). 11th ACM International Conference on Web Search and Data Mining (WSDM). Recuperado de: [REV2: Fraudulent User Prediction in Rating Platforms](#)
- Leskovec, J. (2023). Bitcoin Alpha trust weighted signed network. Recuperado de: <http://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html>