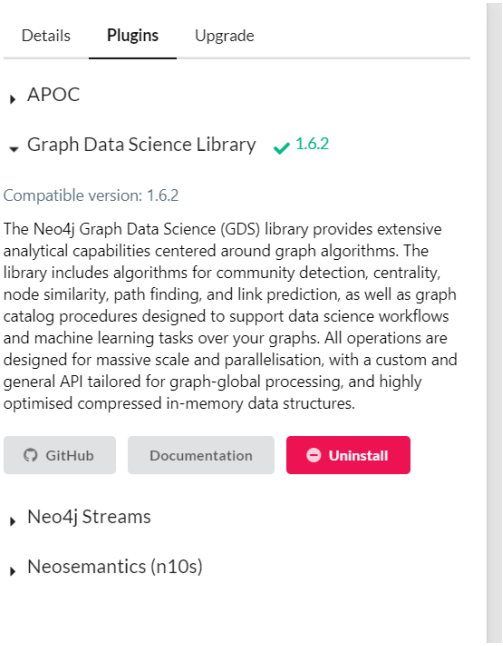
Manual Técnico de aplicación web “Algoritmos de búsqueda con Neo4j”.

Requisitos.

1.-Para ejecutar la aplicación web en una computadora es necesario cumplir los siguientes requisitos.

- Tener instalado NodeJs. (*https://nodejs.org/es/*)
- Tener instalado Neo4j Desktop. (*https://neo4j.com/download/?ref=get-started-dropdown-cta*)
- Tener instalado el plugin Graph Data Science 1.6.2 en Neo4j (necesario para ejecutar los algoritmos de búsqueda).



- Un editor de texto (por ejemplo: Visual Studio Code).
- Un navegador web (por ejemplo: Chrome) .

2. Obtener el código fuente del proyecto desde github (https://github.com/scarapia/PracticasCaminos).

Índice.

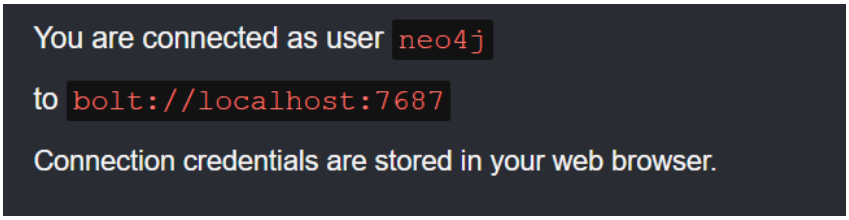
Iniciando la aplicación.

1.- Dirigirse a la carpeta raíz del proyecto

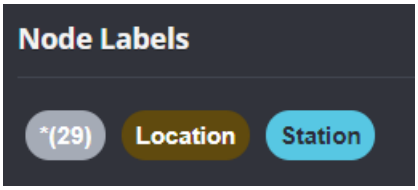
```
C:\Users\scfac\Documents\Practicas-Profesionales\Neo4j-Node.Js>
```

2. En la carpeta raíz del proyecto, instalar los módulos node necesarios con el comando **npm install** .

3. Crear una base de datos en Neo4j.



4.- En nuestra base de datos tenemos dos etiquetas: **"Location**" y **"Station",** 29 nodos y 7 tipos de relaciones.



5.- Iniciar el servidor con el comando: **npm run start** (esto inicia nuestro archivo principal, **app.js** , donde se ejecutan todas las acciones seleccionadas por el usuario). Podemos verificar que el servidor se inició correctamente por el mensaje : “Server Started on Port 3000”.

```
PS C:\Users\scfac\Documents\Practicas-Profesionales\Neo4j-Node.Js> npm run start

> neo4j-nodejs@1.0.0 start C:\Users\scfac\Documents\Practicas-Profesionales\Neo4j-Node.Js
> node app

Server Started on Port 3000
```

6.- Abriendo el navegador web en la dirección **localhost:3000** podemos visualizar la aplicación (ver en siguiente pagina).

Agregar nodo a Station

Nombre Nodo

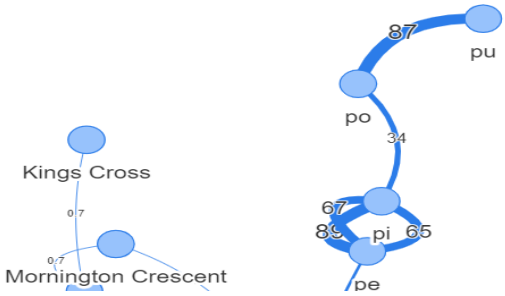
Latitud

Longitud

Crear Nodo

Nombre	Kings Cross	Euston	Camden Town	Mornington Crescent	Kentish Town	pe	pu	po	pi	pa	ya
Latitud	51.5308	51.5282	51.5392	51.5342	51.5507	76	12	546	56	123	745
Longitud	-0.1238	-0.1337	-0.1426	-0.1387	-0.1402	76	12	546	56	123	12

Nodos conectados en Label Station



Conectar nodos en Station

Nombre nodo inicio

Nombre nodo destino

Distancia

Conectar Nodos

Borrar nodo en Station

Nombre nodo a borrar

Borrar Nodo

Algoritmo A-Star* en nodos en Station

Nombre nodo fuente (Source)

Nombre nodo objetivo (Target)

Ejecutar y crear camino A*STAR

Borrar Camino A*Star en Station

Borrar Camino A*STAR

Borrar Camino A*STAR

En la aplicación podemos:

- Crear nodos con propiedades: nombre, latitud y longitud.
- Borrar nodos.
- Conectar nodos con alguna distancia (relationships).
- Ejecutar algoritmos de búsquedas de caminos (para esto, es necesario que tengamos nodos conectados, ya que necesitamos nodos inicio y destino).

Carpeta views.

En la carpeta **views**, se encuentran los archivos con extensión **.ejs**. Estos archivos son los que el usuario ve al entrar a la aplicación (npm run start).

El primer archivo .ejs**, index2.ejs**, es el que se renderiza al iniciar la aplicación. Este se inicia en el archivo **app.js.**

```
58            res.render('index2', {
59              station: stationGraphsArr,
60              location: locationGraphsArr
61            }); // index2.ejs
```

Dentro del archivo index2.ejs, se llama al archivo "**relconnection.ejs**", mediante la instrucción: **include('relconnection')**

```
35    </table>
36    <%- include('relconnection')%>;
```

Este archivo dibuja el grafo de relaciones existentes que se indique en el código del documento, en nuestro caso, utilizamos el comando inicial para dibujar las relaciones de tipo "CONNECTION".

```
62              initial_cypher:  " MATCH p=()-[r:CONNECTION]->() RETURN p LIMIT 25 "
63
```

Nota*: Los archivos con extensión **.html** pueden ser cambiados a .**ejs** para su visualización en la aplicación, pero deben estar dentro de la carpeta **views.**

Por el momento, la aplicación solo muestra el primer archivo que se llame en el archivo (por ejemplo, si se crea otra llamada para incluir el segundo archivo .ejs, **nodosstation.ejs**, esta llamada será ignorada y solo se mostrara el grafo del archivo **relconnection.ejs**.

Archivo app.js.

Este archivo es el corazón de la aplicación.

Aquí se definen:

- La conexión a la base de datos.
- La obtención de nodos existentes en la base datos.
- La creación de nodos en la base de datos.
- La eliminación de nodos.
- La creación de relaciones entre dos nodos.
- La ejecución de algoritmos de búsqueda de caminos.

Conexión a la base datos.

Conectar la base de datos con Express usando las credenciales utilizadas al crear la base de datos (user, password, puerto bolt), mediante la variable **driver** en el archivo **app.js**. Nuestra base de datos tiene el usuario "*neo4j*" y contraseña "*password*". *Nota: la base de datos debe estar activa al momento de realizar la conexión; también durante la ejecución de la aplicación.

```
20  var driver = neo4j.driver('bolt://localhost', neo4j.auth.basic('neo4j', 'password'));
```

Obtención de nodos existentes en la base datos.

En el archivo app.js, se utiliza un arreglo stationGraphsArr , el cual obtiene las propiedades de los nodos existentes en la base de datos y se utiliza para mostrarlos en la tabla que se observa al iniciar la aplicación (ver paso # de iniciando la aplicación).

```
28  app.get('/', function(req, res){
29    session // Station Label
30      .run('MATCH (n:Station) RETURN n ')
31      .then(function(result) {
32
33        var stationGraphsArr = []; // Array de Station Grafos
34
35        result.records.forEach(function(record) {
36          stationGraphsArr.push({
37            id: record._fields[0].identity.low, // Datos de Grafos
38            name: record._fields[0].properties.name, // Station
39            latitude: record._fields[0].properties.latitude,
40            longitude: record._fields[0].properties.longitude
41            //distance: record._fields[0].properties.distance
42
43          });
44
45        });
```

Creación de nodos en la base de datos.

En el archivo **app.js**, se abre una sesión (**session**) en la cual se ejecuta la instrucción para la creación de un nodo en la base datos en la etiqueta “Station”, el cual será creado a partir de los datos con que se llenen los campos de texto en la aplicación.

El código “session.close()” se encuentra comentado para el correcto funcionamiento de la aplicación.

***Nota:** se utiliza el comando “MERGE” para que se evite la creación de nodos duplicados (con el mismo nombre), si se utiliza el comando “CREATE”, pueden crearse nodos repetidos.

```
77  app.post('/station/add',function(req,res) { //agrega a Station
78    var name = req.body.station_name;
79    var latitude = Number.parseInt(req.body.station_latitude); //Number.parseInt convierte texto de usuario a numero en texto
80    var longitude = Number.parseInt(req.body.station_longitude); //Number.parseInt convierte texto de usuario a numero en texto
81
82    session // Codigo Crea nodo en Label Station
83      //.run('CREATE (n:Station {name: $nameParam, latitude: $latitudeParam, longitude: $longitudeParam }) RETURN n.name', {nameParam:name, latitudePa
84      //MERGE SOLO permite nodos con Nombres DIFERENTES
85      .run(' MERGE (n:Station {name: $nameParam, latitude: $latitudeParam, longitude: $longitudeParam}) RETURN n.name', {nameParam:name, latitudeParam
86      .then(function(result){
87        res.redirect('/');
88
89        session.close();
90      })
91      .catch(function(err){
92        console.log(err);
93      });
94
95
96    res.redirect('/');
97    //session.close(); // cerrar session despues de cada run
98
99  });
100
```

Eliminación de nodos en la base de datos.

En el archivo **app.js**, se abre una sesión (sesión) donde se ejecuta la instrucción de eliminar el nodo con el nombre que se haya puesto en el campo de texto: “Nombre de nodo a borrar” , seguido de oprimir el botón “borrar Nodo”. Todas las relaciones que tenga el nodo también se borran.

El código “session.close()” se encuentra comentado para el correcto funcionamiento de la aplicación.

```
126 app.post('/station/delete',function(req,res) { //Borra Nodo  en  Station
127   var name = req.body.stationDelete_name;
128   //var latitude = Number.parseInt(req.body.station_latitude); //Number.parseInt convierte texto de usuario a numero en texto
129   //var longitude = Number.parseInt(req.body.station_longitude); //Number.parseInt convierte texto de usuario a numero en texto
130
131   session // Codigo Borra nodo en Label Station
132     //.run('CREATE (n:Station {name: $nameParam, latitude: $latitudeParam, longitude: $longitudeParam }) RETURN n.name', {nameParam:name, latitudePa
133     //MERGE SOLO permite nodos con Nombres DIFERENTES
134     .run('  MATCH (n {name: $nameParam})DETACH DELETE n', {nameParam:name})
135     .then(function(result){
136       res.redirect('/');
137
138       session.close();
139     })
140     .catch(function(err){
141       console.log(err);
142     });
143
144
145   res.redirect('/');
146   //session.close(); // cerrar session despues de cada run
147
148 });
149
```

Conectar nodos en Station

Nombre nodo inicio

Nombre nodo destino

Distancia

Conectar Nodos

Borrar nodo en Station

Nombre nodo a borrar

Borrar Nodo

Algoritmo A-Star* en nodos en Station

Nombre nodo fuente (Source)

Nombre nodo objetivo (Target)

Ejecutar y crear camino A*STAR

Borrar Camino A*Star en Station

Borrar Camino A*STAR

Borrar Camino A*STAR

Conectar nodos en la base datos.

En el archivo **app.js**, se abre una sesión (session) para realizar la creación de una relación de tipo "CONNECTION" entre dos nodos en la etiqueta **"Station"**. Los nodos serán conectados mediante los datos introducidos en los campos de texto : "Nombre nodo inicio " , "Nombre nodo destino", y "Distancia", seguido de oprimir el botón: "Conectar Nodos" .

```
103  app.post('/station/startfinish/add',function(req,res) { //Conecta Nodos en Station
104    var name1 = req.body.stationStart_name;
105    var name2= req.body.stationFinish_name;
106    var distance = Number.parseInt(req.body.station_distance);
107
108    session // Codigo Crea relacion de nodos en Label Station
109      //.run('MATCH(a:Station {name1:$nameParam1}), (b:Station{name2:$nameParam2}) MERGE (a)-[r:CONNECTION]-(b) RETURN a,b', {nameParam1:name1,name
110      .run(' MATCH (a:Station {name: $nameParam1}), (b:Station {name: $nameParam2}) MERGE( (a)-[r:CONNECTION{distance:$distanceParam}]->(b)) RETURN
111      .then(function(result){
112        res.redirect('/');
113
114        session.close();
115      })
116      .catch(function(err){
117        console.log(err);
118      });
119
120
121    res.redirect('/');
122    //session.close(); // cerrar session despues de cada run
123
124  });
125
```

Conectar nodos en Station

Nombre nodo inicio

Nombre nodo destino

Distancia

Conectar Nodos

Borrar nodo en Station

Nombre nodo a borrar

Borrar Nodo

Algoritmo A-Star* en nodos en Station

Nombre nodo fuente (Source)

Nombre nodo objetivo (Target)

Ejecutar y crear camino A*STAR

Borrar Camino A*Star en Station

Borrar Camino A*STAR

Borrar Camino A*STAR

Ejecución de algoritmos de búsqueda de caminos.

Para ejecutar un algoritmo de búsqueda, es necesario crear un grafo y “escribirlo” en la base datos.

Como ejemplo utilizaremos el algoritmo A* (A estrella).

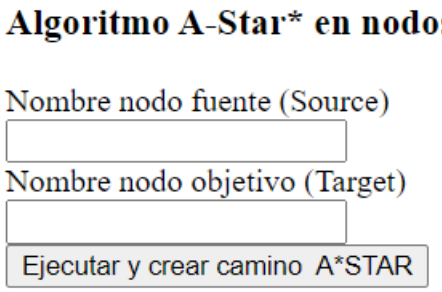
Primero necesitamos crear el grafo, entonces damos en click en el botón que se llama: “crear grafo A*(myGraphStar)”.

Este botón ejecuta la instrucción de crear un grafo llamado “myGraphStar2”, el cual existirá en la etiqueta Station en Neo4j.

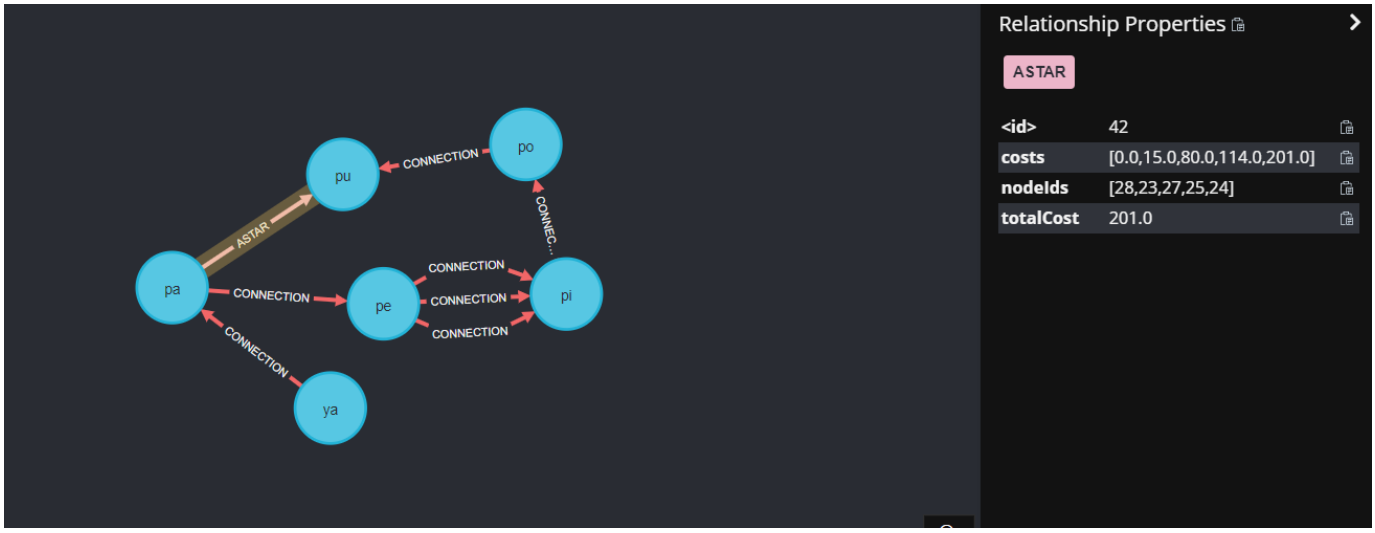
```
196
197 app.post('/myGraphStar',function(req,res) { //myGraphStar
198
199
200   session // Codigo Crear myGraphStar
201     //.run('CREATE (n:Station {name: $nameParam, latitude: $latitudeParam, longitude: $longitudeParam }) RETURN n.name', {nameParam:name, latitudePa
202     .run("CALL gds.graph.create('myGraphStar2','Station','CONNECTION',{nodeProperties: ['latitude', 'longitude'],relationshipProperties: 'distance'}
203     .then(function(result){
204       res.redirect('/');
205
206       session.close();//faltaba cerrar session
207     })
208     .catch(function(err){
209       console.log(err);
210     });
211
212
213
214
215   res.redirect('/');
216   //session.close(); //cerrar sesion despues de cada run
217
218 });
```

Una vez creado el grafo, podemos "escribirlo" en Neo4j. Esto nos permitirá dibujar y guardar la ruta de nodos que se obtienen al utilizar este algoritmo de búsqueda.

El paso siguiente consiste en elegir nuestros nodos fuente y objetivo, esto consiste en escribir los nombres de los nodos a utilizar y oprimir el botón "Ejecutar y crear camino A*STAR ".



Habiendo hecho lo anterior, podemos dirigirnos a Neo4j y podremos observar que el camino "ASTAR" se ha creado y guardado en la base datos. De igual manera podemos ver que nos muestra la ruta de nodos visitados y el costo total.



Nota*: es posible mostrar el grafo en la aplicación mediante los archivos .ejs.

De manera análoga, se pueden ejecutar los otros algoritmos de búsqueda siguiendo los mismos pasos.

Referencias

Para más información respecto a la implementación y funcionamiento de los algoritmos se recomienda visitar la liga:

https://neo4j.com/docs/graph-data-science/current/algorithms/pathfinding/#algorithms-path-finding

Para mas información respecto al uso de JavaScript con Neo4j:

https://neo4j.com/developer/javascript/