DataMgmt - Laboratoire n°3
NoSQL : Base de données orientées graphe avec Neo4J

18.11.2021

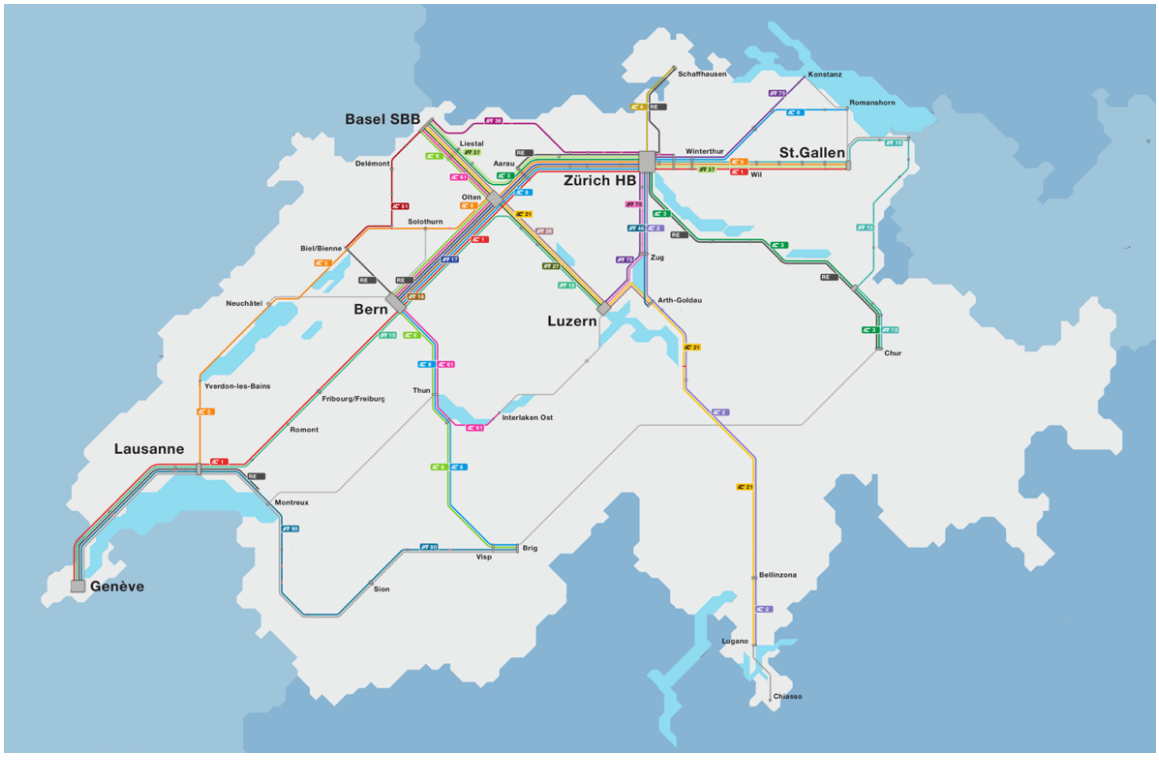
1. Introduction

Ce laboratoire a pour but d'explorer un type de base de données NoSQL, les base de données orientées graphe avec Neo4J (https://neo4j.com/) :

1. Nous allons utiliser Neo4J afin de modéliser les grandes lignes du réseau ferroviaire de la Suisse avec :
    - Les sommets du graphe correspondant aux villes
    - Les arêtes correspondent aux lignes de chemin de fer les reliant

2. Les villes se trouvent dans le fichier *cities.csv*, chaque ville possède un nom (*name*), ses coordonnées GPS (*latitude* et *longitude*) ainsi que le nombre d'habitants (*population*),

3. Les lignes du réseau ferroviaire sont décrites dans le fichier *lines.csv*. Chaque ligne décrit les deux villes qu'elle relie (*city1* et *city2*) et possède également trois poids : la distance en kilomètres séparant les deux villes (*km*), la durée moyenne du trajet en minutes (*time*), ainsi que le nombre de voies sur la ligne (*nbTracks*).



1.1 Organisation

Ce laboratoire doit être réalisé par groupe de 2 étudiants au maximum.

1.2 Rendu

Rendre un zip contenant :

- Le code source de votre implémentation.
- Les fichiers html correspondants aux résultats des différentes tâches à effectuer.

1.3 Date de rendu

Voir sur moodle.

1.4 Environnement de travail

- Vous devez avoir Python 3.7 ou plus récent d'installé.
- Il peut vous être utile de créer un environnement virtuel.

1.5 Fichiers fournis

- *docker-compose.yml* : permet de faciliter l'installation et la configuration d'une instance Neo4J contenant les plugins nécessaires pour l'application d'algorithmes sur les graphes.

- *requirement.txt* : contient la liste des dépendances python. Pour les installer :
  ```
  > pip install -r "requirement.txt"
  ```

- *index.py* : fichier **à compléter** qui contiendra le code permettant de générer les sommets et les arêtes modélisant notre réseau ferroviaire avec Neo4J, ainsi que la génération de données pour nos algorithmes sur les graphes.
  Actuellement, contient uniquement le code permettant de générer les sommets (les villes) à partir du fichier *cities.csv*.

- *display.py* : fichier **à compléter** permettant de travailler sur notre base de données graphe, d'afficher notre réseau et le résultat des algorithmes sur une carte.
  Actuellement, contient uniquement le code permettant d'afficher les sommets (les villes) sur la carte.

1.6 Requêtes utiles

- `MATCH(n) DETACH DELETE n`
  Supprime tous les nœuds et arrêtes de notre base de données

- `CALL gds.graph.drop('myGraph')`
  Supprime le graphe 'myGraph' (du plugin Graph Data Science) de notre base de données

2. Travail demandé

2.1 Création des lignes de chemin de fer reliant les villes

On souhaite à présent ajouter les lignes de chemin de fer à notre base de données graphe. Dans votre fichier *index.py*, vous allez devoir créer des relations entre deux nœuds (nos villes) pour chaque ligne décrite dans le fichier *lines.csv*. Ces relations auront les propriétés *km*, *time* et *nbTracks*.

Après avoir créé ces relations, il vous faudra les récupérer et les afficher sur la carte en modifiant le fichier *display.py*.

Rendu :

- Un fichier *2.1.html* contenant la carte où les liens entre les villes sont affichés.

Ressources :

- https://neo4j.com/docs/cypher-manual/current/clauses/create/#create-create-a-relationship-and-set-properties

2.2 Requête sur les villes

A présent, on vous demande d'écrire une requête permettant de mettre en évidence sur la carte les villes ayant plus de 100'000 habitants et qui se situent à 4 arrêts ou moins de Lucerne.

Rendu :

- Un fichier *2.2.html* mettant en évidence les villes retournées sur la carte.

Ressources :

- Clause WHERE : https://neo4j.com/docs/cypher-manual/current/clauses/where/
- Relation de longueur variable : https://neo4j.com/docs/cypher-manual/current/clauses/match/#varlength-rels

2.3 Algorithme de plus court chemin (Shortest Path)

Pour les deux derniers points, on souhaite utiliser les algorithmes sur les graphes. On va pour cela utiliser le plugin Graph Data Science de Neo4J (directement installé si le docker fourni est utilisé).

Ici on cherche à trouver le plus court chemin entre 2 villes selon un critère précis (le nombre de kilomètres ou le nombre de minutes). Nous allons donc utiliser l'algorithme de Dijkstra Source-Target du plugin Graph Data Science (voir ressources).

Rendu :

- Un fichier *2.3.1.html* mettant en évidence le chemin le plus court (en utilisant les *kilomètres*) entre Genève et Coire.
- Un fichier *2.3.2.html* mettant en évidence le chemin le plus court (en utilisant les *minutes*) entre Genève et Coire.

Ressources :

- https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-source-target/#algorithms-dijkstra-source-target-examples
- https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-source-target/#algorithms-dijkstra-source-target-examples-stream

2.4 Arbre couvrant de poids minimum (Minimum Spanning Tree) :

Un arbre couvrant de poids minimum est un sous graphe qui connecte tous les sommets et dont le coût total des arêtes est le plus bas possible.

Nous voulons effectuer des travaux d'entretien sur le réseau ferroviaire, mais dans un souci d'économie nous souhaitons réduire le coût des travaux au maximum. Toutes les villes devront être accessibles par une ligne rénovée. Chaque ligne possède un nombre de voies allant de 1 à 4. Le coût pour rénover 1 km de ligne de chemin de fer varie selon le nombre de voies, il est de 1 million par voie par kilomètre, soit :

- 4M CHF par km pour les lignes ayant 4 voies
- 3M CHF par km pour les lignes ayant 3 voies
- 2M CHF par km pour les lignes ayant 2 voies
- 1M CHF par km pour les lignes ayant 1 voie

On souhaite donc tout d'abord ajouter une propriété de coût sur nos relations (nos lignes de chemin de fer) qui indique le prix de rénovation d'une ligne (voir ressources). C'est cette nouvelle propriété de coût qui sera utilisée pour trouver notre arbre couvrant de point minimum.

Ensuite, nous allons utiliser l'algorithme du Minimum Weight Spanning Tree (voir ressources) offert par le plugin Graph Data Science afin de construire les arêtes qui représentent notre arbre couvrant de poids minimal.

Rendu :

- Un fichier *2.4.html* mettant en évidence les lignes à rénover sur la carte.

Ressources :

- Ajouter une propriété à une relation : https://neo4j.com/docs/cypher-manual/current/clauses/set/#set-set-a-property
- Arbre couvrant de poids minimum : https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/minimum-weight-spanning-tree/#algorithms-minimum-weight-spanning-tree-sample