etape 1:

importer les graphes dans neo4j

Pour l’importation dans Neo4j (exemple du graph Area_5) :
CREATE INDEX ON :Area_5(id);

USING PERIODIC COMMIT 200 à A changer par “:auto” en function de la version
LOAD CSV WITH HEADERS FROM "file:/HF_gadm36_5.csv" as l FIELDTERMINATOR '\t'
MERGE(loc:Area_5{id:l.gid}) ON CREATE SET loc.gid_0=l.gid_0, loc.name_0=l.name_0, loc.gid_1=l.gid_1, loc.name_1=l.name_1, loc.gid_2=l.gid_2, loc.name_2=l.name_2 , loc.gid_3=l.gid_3, loc.name_3=l.name_3, loc.gid_4=l.gid_4, loc.name_4=l.name_4, loc.gid_5=l.gid_5,loc.name_5=l.name_5;

USING PERIODIC COMMIT 500 LOAD CSV WITH HEADERS FROM "file:/HF_circulationGraph_5.csv" as l FIELDTERMINATOR '\t' MATCH(from:Area_5{id: l.gid_from}) MATCH(to:Area_5{id: l.gid_to}) MERGE (from) -[:trip{year:toInteger(l.year), month:toInteger(l.month), nat:l.country, age:l.age, NB:toInteger(l.NB)}]-> (to);

etape 2:

agréger les arêtes pour avoir les graphes par nationalité/année

```
MATCH (a1:Area_5) -[t:trip]-> (a2:Area_5)
where t.nat = 'United States' and t.year=2016 and t.month=12
MERGE (new_1:Area{name:a1.name_5, name_4:a1.name_4, name_3:a1.name_3, name_2:a1.name_2, name_1:a1.name_1, name_0:a1.name_0})
MERGE (new_2:Area{name:a2.name_5, name_4:a2.name_4, name_3:a2.name_3, name_2:a2.name_2, name_1:a2.name_1, name_0:a2.name_0})
MERGE (new_1) -[new_t:totalTripUSA2016]-> (new_2)
ON CREATE SET new_t.totalNb=t.NB, new_t.totalYear=t.year, new_t.totalNat = t.nat,new_t.totalMonth=t.month
ON MATCH SET new_t.totalNb = new_t.totalNb + t.NB
```

pour agréger j’ai prit l’habitude d’appeler ces graphes totalTrip+PAYS+annee

puis supprimer les arêtes de poids < 5

etape 3:

une fois qu'on a nos graphes, on cherche le noeud avec le weighted betweenness centrality le plus élevé qu'on utilisera comme racine pour calculer notre spanning tree avec Prim

l'algo du WBC ne marche pas sur Neo4j il ne prend pas en compte le poids sur les aretes, donc il faut passer par python et la librairie networkx

-> exporter les graphes en csv

```
MATCH path = ()-[t:totalTripFRA2013]-()

WITH relationships(path) AS rels

UNWIND rels AS rel

WITH DISTINCT rel AS rel

RETURN startNode(rel).name AS source,endNode(rel).name AS destination, rel.totalNb AS cost
```

etape 4:

utiliser le notebook compute_maxbetweenness.ipynb

affiche le noeud avec le plus haut WBC, c’est donc ce qu’on va utiliser comme racine dans notre prim (qui n’est pas sur networkx donc j’utilise Neo4J)

etape 5:

on lance l’algo de spanning tree sur neo4j avec les noeuds racines:

```
MATCH (n:Area{name:"Bordeaux"})
CALL algo.spanningTree.maximum('Area', 'totalTrip', 'totalNb', id(n),
  {write:true, writeProperty:"MAXSTFRA2013"})
YIELD loadMillis, computeMillis, writeMillis, effectiveNodeCount
RETURN loadMillis, computeMillis, writeMillis, effectiveNodeCount;
```

pour les spanning tree j’ai prit l’habitude d’appeler les relationships MAXST+PAYS+année

etape 6:

extraire les graphes en CSV et utiliser le notebook export_json_rif.ipynb qui génère les json

```
MATCH path = ()-[t:MAXSTFRA2018]-()
WITH relationships(path) AS rels
UNWIND rels AS rel
WITH DISTINCT rel AS rel
RETURN startNode(rel).name AS source,endNode(rel).name AS destination, rel.totalNb AS cost
```

il y’a plein d’autres cellules dans le notebook qui serviront peut être donc j’ai laissé

si vous avez une question: https://www.linkedin.com/in/hugo-prevoteau/

pour certains processus sont automatisables avec la librairie py2neo mais j'avais des problemes avec neo4j donc j'ai arrêté d'utiliser (= pas besoin de faire les aller retour à la main entre neo4j et python comme j'ai fait)

il y'a du code que j'utilisais pour ça dans le notebook "extraction_arbres.ipynb"

bon courage