TD3

Sujet 3

- Requirements: **Python3**

Exercice 1

Télécharger le dataset: `wget https://snap.stanford.edu/data/higgs-social_network.edgelist.gz`

extraire le dataset: `gunzip -d higgs-social_network.edgelist.gz`

Ouvrir sqlite: `sqlite3`

Ouvrir ou creer la database: `.open database.db`

Changer le separator: `.separator " "`

Creer la table: `CREATE TABLE social_network(to_node TEXT, from_node TEXT);`

Importer le dataset dans la table table: `.import higgs-social_network.edgelist social_network`

Lancer exo1.sql script: `.read exo1.sql`

Exercice 2 : *SPARQL*

Chaque exercice a son propre fichier n.sparql situé dans le dossier td3

1.(a) Expliquez le résultat de la requete suivante.

```
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ( COUNT (*) as ? c ) WHERE
{
    ?x rdf:type ?t
}
```

Cette requete compte le nombre de triplet ayant un type.

2.Expliquez le résultat de la requete suivante.

```
PREFIX humans:<http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT * WHERE {
    ? x humans : hasSpouse ? y
}
```

Cette requete retourne les couples de personnes ayant un conjoint.

- For questions 3 - 9 voir le fichier correspondant 3-9.sparql dans td3 dir
- For question 8.b (cloture transitive) il faut rédiriger la sortie de la requete 8.a dans un fichier puis lancer la requete 8.b avec ce fichier comme input.

  `./sparql --data G --query 8-a.sparql > output.txt`

Run with

```
./sparql --data output.txt --query Q
```

Sujet 4

Placer le dossier 'td4' folder dans hadoop-2.9.1 directory

Exercice 1 : *Word count*

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.1.jar -input sujet-mapreduce-fichiers/input-word-count/ -output out -mapper td4/word-count-map.py -reducer td4/word-count-reduce.py
```

On peut vérifier que le résultat soit bon avec la commande:

```
diff out/part-00000 sujet-mapreduce-fichiers/expected-outputs/word-count.txt
```

Exercice 2 : *Produit matriciel*

One round:

Map: Pour M, on produit un nombre de couples égal au nombre de colonnes de N. Chaque couple est de la forme ((i, k), j, Mij) où i est le numéro de la ligne de M, j le numéro de la colonne de M, k est le un nombre entre 0 et le nombre de colonnes de N et Mij la valeur de l'élément.

Pour N, on produit un nombre de couples égal au nombre de lignes de M. Chaque couple est de la forme ((i, k), j, N) où j est le numéro de la ligne de N, k le numéro de la colonne de N, i est le un nombre entre 0 et le nombre de lignes de M et Nkj la valeur de l'élément.

Reduce: Chaque clé (i, k) est associée à une liste contenant toutes les valeurs (j, mij) et (j, njk), pour toutes les valeurs possibles de j. Le but est de calculer le produit mij x njk pour chaque j. Pour cela, on relie les deux valeurs de la liste qui ont la même valeur de j, pour chaque j. Ensuite, ces produits sont additionnés par rapport à la clé (i, k).

On peut le tester avec la commande:

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.1.jar -input sujet-
mapreduce-fichiers/input-matmul/ -output out -mapper td4/oneround-map.py -reducer
td4/oneround-reduce.py
```

Two rounds:

Première étape Map: Pour chaque élément de la matrice M on produit un couple (j, (M, i, Mij)) où j est le numéro de la colonne de M, i le numéro de la ligne et Mij la valeur de l'élément et M le nom de la matrice 'M'.

Concernant la matrice N, on produit un couple (j, (N, k, Nij)) où j est le numéro de la ligne de N, k le numéro de la colonne et Nij la valeur de l'élément et N le nom de la matrice 'N'.

Premier étape Reduce: Le but est de faire une jointure sur les clés (j) et de calculer le produit des éléments de la matrice M et N ayant le même numéro de colonne et de ligne respectivement.

Pour chaque clé j, on récupère les couples (M, i, Mij) et (N, k, Nij) et on produit Mij x Nij .On produit un couple(i, k, Mij* Nij) qui est le résultat de la multiplication de la ligne i de M et de la colonne k de N.

Deuxième étape Map: La deuxième étape map est juste l'identité.

Deuxième étape Reduce: Pour chaque clé (i, k) on somme les valeurs associées. Le résultat est ((i, k), v)) où v est la somme des valeurs associées à la clé (i, k).

Pour tester avec hadoop, voir le script 'scpt'. Il faut placer le script td4/scpt dans le repertoire hadoop-2.9.1. Executer le script avec la commande:

```
./scpt
```

scpt

```
mkdir tworound 2> /dev/null
rm tworound/out1.txt 2> /dev/null
rm -r out/
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.1.jar -input sujet-
mapreduce-fichiers/input-matmul/ -output out -mapper td4/tworound-map1.py -reducer
td4/tworound-reduce1.py
cat out/* > tworound/out1.txt
rm -r out/
cat tworound/*
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.1.jar -input tworound/
-output out -mapper td4/tworound-map2.py -reducer td4/tworound-reduce2.py
cat out/*
```

On peut aussi tester avec la commande sort sous linux:

```
cat sujet-mapreduce-fichiers/input-matmul/* | td4/tworound-map1.py | sort |
td4/tworound-reduce1.py  | td4/tworound-map2.py  | sort | td4/tworound-reduce2.py
```

Output:

```
1,1     2
1,2     2
1,3     15
1,4     11
2,1     5
2,2     0
2,3     30
2,4     15
3,1     2
3,2     6
3,3     21
3,4     21
```

Exercice 3 : *Agrégats sur des données Twitter*

Question 1:

Pour tester, on peut utiliser le script script-twitter1.sh

mapper:

Le mapper s'appelle twitter1-mapper.py Le mapper fonctionne comme celui de wordcount. Pour chaque ligne, on affiche le nom de la personne suivi et 1.

reducer:

Le reducer s'appelle twitter1-reduce.py. Comme pour le wordcount, tant qu'on est sur le même utilisateur, on compte le nombre de follower. Si l'utilisateur de la ligne est différent de celui d'avant, on affiche l'utilisateur d'avant et son nombre de follower.

Question 2:

Pour tester, on peut utiliser le script script-twitter2.sh

mapper:

Le mapper s'appelle twitter2-mapper.py Ce mapper est simplement l'identité, on affiche l'utilisateur et son nombre de followers.

reducer:

Le reducer s'appelle twitter2-reduce.py. Pour chaque ligne, on effectue 4 étapes:

- étape 1: On incrémente de 1 un compteur qui compte le nombre de personne suivi.

- étape 2: On incrémente par le nombre de follower de cet utilisateur un compteur qui compte le nombre de relation.
- étape 3: On cherche à savoir si l'utilisateur a moins de followers que celui qui en a le moins actuellement. Si c'est le cas, on remplace min_user par celui-ci et min_followers par son nombre de followers.
- étape 4: On cherche à savoir si l'utilisateur a plus de followers que celui qui en a le plus actuellement. Si c'est le cas, on remplace max_user par celui-ci et max_followers par son nombre de followers.