Grands Réseaux d'Interaction
TP n° 1 : chargement de grands graphes et affichage des degrés

à rendre pour le 22 octobre

I) Règles générales en TP

- Les feuilles de TP se trouveront sur Moodle, cours *Grands Réseaux d'Interaction 2018* (GRI2018). La clef d'inscription est `GRI2019`. Les rendus de TP sont à faire sur Moodle également.
- Les TPS sont notés. Les rendus sont à faire en groupe de **2 étudiants**. Vous pouvez faire le TP seul **si vous y êtes explicitement autorisé** (envoyez un mail à `fm@irif.fr`). Aucun groupe de 3 ou plus étudiants ne sera autorisé. Il est autorisé de s'entre-aider entre groupes mais **le plagiat donnera une note de 0**. Il est bien sûr possible que la même courte portion de code se retrouve dans plusieurs programmes mais en cas d'excès, c'est 0.
- Les TPs se font en **Java** exclusivement
- Vos programmes doivent prendre leurs paramètre en ligne de commande plutôt qu'interactivement ou par des constantes en dur dans le code (pas de nom de fichier dans le code !). Lancé sans paramètre, ils doivent afficher une aide (syntaxe). Quand la syntaxe n'est pas évidente il est fortement conseillé de rajouter un fichier `readme` sinon le correcteur risque de ne pas savoir tester votre programme et de vous donner une mauvaise note.
- Chaque rendu doit être constitué d'une seule archive au **format tar**, donc non-compressée (pas de zip, tar.gz, rar...) Elle ne doit pas contenir les grands graphes qui vous seront fournis, ni des `.class`, ni des `.jar`, logs ou autres choses inutiles. Cette archive doit se décompresser en créant un répertoire (et non en mettant les fichiers dans la racine du répertoire courant, ni le fichier `TP1.java` dans un sous-répertoire `src`) portant les nom et prénom du ou des deux auteurs. S'il y a deux auteurs leurs noms seront séparés par un `_` Par exemple les étudiants Emmett Brown et Marty McFly rendront une archive créée par `tar cf BrownEmmet_McFlyMarty.tar BrownEmmett_McFlyMarty/*.java`. Attention, si ce nommage n'est pas respecté l'un des membres d'un binôme peut ne pas être noté (Moodle ne donnant qu'un seul nom) !
- Si `javac TP1.java` ne suffit pas il faut inclure un `Makefile`
- Un seul membre du binôme doit faire un rendu sur Moodle.
- Les critères pour l'évaluation sont la correction du résultat, le temps d'exécution, la mémoire allouée et la pureté du code.

II) La base de données de Stanford

Le site `https://snap.stanford.edu/data/` est une vraie mine d'or pour le traitement de grands graphes réels. Les graphes sont en général fournis sous forme d'un fichier texte, où chaque ligne contient deux nombres (en décimal ASCII) : origine puis extrémité d'un arc. Si le graphe est non-orienté, alors il s'agit des deux extrémités d'une arête (la base précise le cas). On trouve toutes les tailles, du millier aux centaines de millions d'arêtes (ou arcs). Les lignes commençant par un `#` sont des commentaires à ignorer. On ne s'intéressera qu'aux graphes de la base qui suivent cette syntaxe simple.

Remarquez que parfois les graphes sont triés (les arcs ou arêtes arrivent dans l'ordre lexicographique), parfois non, et que il y a parfois des «trous» (numéros de sommets non attribués, que l'on interprête comme des sommets de degré 0). La pluspart des paramètres qui seront à calculer en TP sont déjà donnés dans la base, ce qui permet de vérifier vos calculs.

III) Représentation d'un graphe en mémoire

Le but de ce TP est double : fournir des statistiques sur les degrés, mais aussi de disposer d'une réprésentation en mémoire du graphe. Vous devez respecter les caractéristiques suivantes :

- Le chargement du graphe doit être une étape distincte des calculs : le fichier d'entrée peut être lu plusieurs fois pendant le chargement, mais avant les calculs (en particulier avant de calculer qui est sommet de degré maximum) le fichier d'entrée doit être fermé définitivement.
- Il ne faut pas garder en mémoire du texte provenant du fichier, sauf éventuellement des commentaires. Le type `String` et ses variantes ne peuvent être utilisés pour coder l'adjacence du graphe.
- Il doit y avoir une classe `Graphe` et une classe `Sommet`.
- Un `Graphe` doit contenir des informations sur le graphe (nombre de sommets, d'arêtes...) ainsi que tous les `Sommets` : on évitera les variables `static` et les passages de paramètres trop nombreux aux méthodes.
- Un `Sommet` doit contenir des informations telles que son degré, et surtout sa liste d'adjacence. Il ne faut pas de matrice d'adjacence.
- Il est obligatoire que votre implémentation de la liste d'adjacence utilise un itérateur à délai constant (la méthode `next()` se fait en $O(1)$) ou bien un accès en temps constant au $i$ème voisin d'un sommet. Il est par exemple posssible d'utiliser soit une `ArrayList`, soit une `LinkedList`, soit un `Vector`, soit un tableau...

IV) Travail demandé

Il est demandé de faire un programme qui, étant donné un nom de graphe, charge le graphe en mémoire, puis affiche

- la mémoire allouée à la fin du chargement du graphe (voir ci-dessous section V))
- S'il y a lieu, le nombre de boucles ignorées et de doublons ignorés (voir section VI))
- Le nombre n de sommets
- Le nombre m d'arêtes après suppression des doublons éventuels (voir section VI))
- Le numéro du sommet de degré maximum. S'il y a plusieurs sommets de degré maximum c'est celui de plus petit numéro
- Sa liste d'adjacence (liste pas forcément triée de ses voisins), sur une seule ligne, sous forme de nombres (numéros de sommet) séparés par des espaces
- Enfin, la distribution des degrés du graphe. Il faut afficher plusieurs lignes, chacune de la forme
  `i, nbi`
  où `nbi` est le nombre de sommets de degré `i`, pour `i` allant de 0 jusqu'au degré maximum du graphe. Le dernier `nbi` affiché ne sera donc pas 0 mais le nombre de sommets de degré maximum. La somme de tous les `nbi` doit être égale à n

La classe principale de votre programme doit s'appeler `TP1` de sorte que l'on puisse lancer votre programme par la ligne de commande suivante :

```
java TP1 graphe.txt
```

On suppose que le fichier est décompressé (format `txt` pas `txt.gz`) au format de Stanford. Votre programme ne doit rien afficher d'autre que ce qui est demandé (pas 500 lignes de debug !) et donc sur les exemples donnés (page suivante ou sur Moodle) doit avoir autant que possible le même affichage (sauf bien sûr la consommation mémoire).

V) Affichage de la mémoire allouée

Il est obligatoire de placer, à la fin du du programme, un code affichant la mémoire allouée de la JVM, que l'on obtient par :

```
System.out.println("Mémoire allouée : " +
     (Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory()) + "octets");
```

Le volume mémoire alloué et le temps de calcul seront pris en compte pour la note, qui sera meilleure chez ceux ayant les plus petites valeurs. À titre de comparaison, le corrigé, sur `nivose` le dernier exemple (`web-BerkStan.txt`), met 21s de temps utilisateur et 11s de temps réel, et alloue 449Mo. NB : on obtient le temps par `time` en ligne de commande.

VI) Gestion des boucles et doublons

Pour ce TP on considérera que **tous les graphes donnés en entrée sont non-orientés**, même si le contraire est écrit en commentaire dans le fichier ou sur le site Web. Cela a comme conséquence que entre deux sommets possibles on pourra voir une fois ou deux fois la même arête (*doublon*). Par exemple `ca-AstroPh.txt` commence par la ligne `84424 276` (arc de 84424 à 276). Il est possible qu'il y ait ensuite une ligne `276 84424` (arc dans l'autre sens). et possible qu'elle n'existe pas : dans les deux cas 276 est voisin de 84424 ; et 84424 est voisin de 276. Il y a donc une seule arête (et non pas arc) entre 84424 et 276. Votre code devra gérer cela. Si vous ne le faites pas (c'est à dire si votre programme ne donne des résultats corrects que si chaque arête est présente une seule fois en entrée) alors il faut le signaler clairement. Votre note sera alors moins bonne que si vous gérez la suppression des doublons, mais meilleure que si vous ne l'avez pas dit.

Il arrive parfois aussi que le graphe contienne des boucles, c'est à dire une arête dont l'origine est égale à l'extrémité. . Par exemple `ca-AstroPh.txt` contient la ligne `19282 19282` qui est une boucle sur 19282. Il faut ignorer de telles boucles (elle ne comptent pas pour le degré du sommet et ne seront pas stockées dans le graphe en mémoire)

VII) Exemples

Vous trouverez sur Moodle d'autres exemples. Le nom du fichier vient de celui de la base de Stanford, par exemple `roadNet-CA.out` est le résultat de l'exécution sur `roadNet-CA.txt` correspondant au fichier trouvé sur `https://snap.stanford.edu/data/roadNet-CA.html`

```
$ java TP1 roadNet-CA.txt
2766607 doublons ont ete supprimes
Nombre de sommets : 1971280
Nombre d'aretes : 2766607
Sommet de degré max (de numéro minimal) : 562818
Sa liste d'adjacence (ligne suivante) :
562769 562823 562770 562826 562771 562824 562810 562833 562817 562825 562821 562827
Ditribution des degrés :
0 6075
1 321027
2 204754
3 971276
4 454208
5 11847
6 1917
7 143
8 30
9 1
10 2
11 0
12 1
Mémoire allouée : 308890512 octets

$ java TP1 filenotfound.txt 127393 57507
Erreur entree/sortie sur filenotfound.txt

$ java TP1 badfile.txt 127393 57507
Erreur format ligne 1
```