Grands Réseaux d'Interaction
TP 1 : Structure de données pour parcourir un graphe

Fabien de Montgolfier
fm@irif.fr

6 janvier 2023

1 Règles générales en TP

- Les feuilles de TP se trouveront sur Moodle, cours *IFECY070 Grands Réseaux d'Interaction*. La clef d'inscription est `GRI2023`.
- **Les TPS sont notés**. Les rendus sont à faire soit individuellement, soit en groupe de **2 étudiants maximum**. 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.
- **Inscrivez-vous sur Moodle** dans un groupe.
- Chaque groupe a TP une semaine sur deux
- Les groupes 1X (chiffre 1 plus une lettre) commencent le TP le 6 janvier et **rendent le TP le 20 janvier**, les 2X commencent le 13 janvier et **rendent le TP le 27 janvier**
- Les TPs se font en **Java** exclusivement, en Java 11 et sans API externe (seule celle par défaut est autorisée)
- Vos programmes doivent impérativement prendre leurs paramètre en ligne de commande comme indiqué dans l'énoncée. Il est exclus de les prendre interactivement ou par des constantes en dur dans le code (pas de nom de fichier dans le code !). Le non-respect des consignes peut perturber la correction automatique qui sera effectuée ensuite avec un fort impact négatif sur la note.
- Chaque dépôt doit être constitué d'une seule archive au **format zip** (pas `.tar` ni autre). 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 sans créer de répertoire (tous les fichiers sont dans la racine du ZIP : pas de répertoire `src`). **Un seul dépôt sur Moodle SVP**, pas deux !
- La compilation doit se faire avec `javac TP1.java` sans aucune autre option, et l'éxécution par `java TP1 [arguments]`. Il en sera de même pour les TPs suivants : `javac TP2.java`, etc. Ne créez pas de `package`, car ils empêchent la compilation depuis le répertoire courant sans fixer de `classpath`.
- Les critères pour l'évaluation sont, par ordre décroissant : le respect des consignes, la correction du résultat, le temps d'exécution, la mémoire utilisée (inutilise d'essayer de grapiller quelques octets, je mets simplement un seuil OK/trop gourmand) et la beauté du code.

2 Format de fichier de graphe orienté

On considérera pour ce TP et les suivant un format de fichier très simple pour stocker les graphes. Il s'agit d'un format texte. Chaque ligne

— soit commence par `#` et est un commentaire, à ignorer

— soit est deux nombres décimaux, séparés par des espaces ou des `\t` (tabulation)

Chaque ligne (hors commentaire) correspond à un arc, le premier nombre étant le numéro du sommet d'origine, et le deuxième celui de destination de l'arc. Chaque ligne même la dernière doit terminer par `\n` (retour à la ligne "*à la unix*"). De plus le fichier doit être **trié** (selon le premier sommet comme clé primaire et le deuxième comme clé secondaire) et **sans doublon**.

Si un fichier `graphe.txt` suit ce format sauf qu'il n'est pas trié ou contient des doublons, on peut le rendre conforme grâce à la commande Unix

```
sort graphe.txt -u -k 1n -k 2n
```

2.1 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 utilisables sont fournis sous forme d'un fichier en `.txt.gz` (à décompresser avec `gunzip`. Par exemple, le premier graphe nommé est `ego-Facebook` : cliquer sur le lien amène sur une page avec divers paramètres de graphe (permettant de vérifier le programme) et trois liens vers des fichiers. Celui qui nous intéresse est `facebook_combined.txt.gz`. Mais attention, c'est un graphe non-orienté, on verra en section 2.2 comment pré-traiter les graphes non-orientés.

Dans cette base on trouve des graphes de toutes tailles, certains avec des centaines de millions d'arcs ou arêtes. La base précise "Undirected" ou "Directed" selon que le graphe est orienté ou non. Parfois les graphes sont triés, parfois non et il faut les trier en pré-traitement. Des fois ils ont des *trous* : certains numéro de sommets n'apparaissent pas. On verra en section 3 comment gérer cela (en option)

2.2 Graphes non orientés

Dans ce TP et les suivants, on traitra les graphes non-orientés comme des graphes **orientés symétriques**. C'est-à-dire qu'une arête 1—2 sera transormée en **deux arcs** $1 \to 2$ et $2 \to 1$. Concrètement, dans le fichier non-orienté il y a une seule ligne, dans cet exemple

```
1 2
```

et il faut faire apparaître aussi l'autre :

```
2 1
```

La commande suivante fait le travail, en ignorant les lignes commençant par `#` et en écrivant chaque arc dans l'autre sens avant de trier :

```
awk '/^[^#]/ {print $0"\n" $2 " " $1 }' graphe_nonoriente.txt |
  sort  -u -k 1n -k 2n > graphe_oriente_symetrique.txt
```

Notez que les graphes *Undirected* de la base de Stanford **doivent** être pré-traités par la commande ci-dessus avant chargement : c'est le fihcier `graphe_oriente_symetrique.txt` qui sera chargé par votre programme.

3 Représentation d'un graphe en mémoire

Le but de ce TP est avant tout d'avoir une réprésentation en mémoire, qui resservira ensuite. Vous **devez** respecter les caractéristiques suivantes :

- Dans ce TP (et dans les autres sauf exception), on considérera uniquement des **graphes non orientés**, sans boucles ni arêtes multiples
- Chaque sommet a un numéro qui est un `int` (ce qui nous laisse travailler avec des graphes jusqu'à deux milliards de sommets environ).
- Il y a une classe nommée `Sommet`
- Les sommets du graphes sont réunis dans un `Sommet[]`, un **tableau** de sommets (pas une `ArrayList` ou autre)
- Il n'y a pas de classe ou type pour les arêtes, on n'en aura pas besoin (on n'utilisera pas de poids ou de marquage des arêtes)
- Toutes les listes d'adjacance sont représentées à la queue leu leu dans un tableau de type `int []`
- tout le graphe doit être représenté avec seulement des données de taille constante (petites et ne dépendant pas de la taille du graphe), plus le tableau des `n Sommmet`, plus le tableau des arêtes qui est un `int[m]`. Pour stocker le graphe en mémoire, **vous n'avez pas le droit à d'autres structures de taille dynamique ou non constante que ces deux tableaux**. En revanche, pour le chargement ou le parcours, d'autres structures de données temporaires peuvent être utilisées, mais seulement au sein de la méthode (ou du constructeur) faisant le traitement (et le garbage collector les détruira à la fin du chargement ou du parcours).
- Ces deux tableaux étant respectivement alloués par un `new Sommet[nS]` et `new int[m]`, il faut connaître à l'avance `nS` et `m`, par une première lecture du fichier d'entrée
- `m` est simplement le nombre de lignes du fichier texte (hors lignes de commentaires)
- pour `nS`, la taille du tableau de sommets, il y a deux façons de faire :
  - soit `nS` est le numéro du plus grand sommet qui apparaît dans le fichier, plus 1. Par exemple si le graphe a trois sommets, lesquels ont numéro 1, 3 et 5, alors `nS=6`. Les cases 0, 2 et 4 seront inutilisées
  - soit `nS` est le nombre de sommets. Mais il faut alors **renuméroter** les sommets. Par exemple si les trois sommets ont numéro 1, 3 et 5, alors `nS=3`, et on renumérote le sommet 1 en 0, le 3 en 1, et le 5 en 2. Dans ce cas, un `Sommet` doit contenir son nom, pour affichage : dans l'exemple, le nom du deuxième `Sommet`, qui sera utilisé pour les affichages, est alors `5` (on n'affiche pas le *numéro* de sommet utilisé en interne, qui est 2, pour le sommet de *nom* 5).
- La version de base du TP ne demande pas de renumérotation. En extension (cf section 4.2) vous pouvez l'implémenter, ce qui fait un programme parfois beaucoup plus efficace en mémoire.

4 Travail demandé

Il est demandé de faire un programme qui, étant donnés deux paramètres
- un nom de fichier de graphe (fichier texte au format décrit ci-dessus)
- et un numéro x de sommet du graphe

charge le graphe en mémoire (obligatoirement de la façon décrite ci-dessus), puis affiche sur la sortie standard
- soit un message d'erreur en une ligne contenant le mot "`ERREUR`" en majuscule, plus d'autres informations utiles
- soit, s'il n'y a pas eu d'erreur, les 6 nombres suivants, un par ligne (donc 5 lignes en tout, rien d'autre que 6 nombres en 6 lignes) :
  - La taille Sn du tableau de sommets qui est donc soit le numéro du plus grand sommet plus un (si on ne renuméote pas) soit le nombre n de sommets (si renumérotation)
  - le nombre exact m d'arêtes
  - le degré maximum d'un sommet (nombre de voisins du sommets qui en a le plus)
  - le nombre de sommets accessibles depuis x
  - l'eccentricité de x, c'est-à-dire la longueur du plus long des plus courts chemins partant de x
  - le nombre de composantes connexe **dans le cas où le graphe est orienté symétrique**. Si l'on suppose que l'on a fait le pré-traitement décrit en section 2.2, alors c'est le nombre de fois où il faut relancer un parcours pour visiter tous les sommets. Sinon, si le graphe en entrée n'est pas orienté symétrique, on peut afficher n'importe quel nombre, sans avoir à tester si le graphe est orienté symétrique et sans écrire "`ERREUR`" (la correction n'est pas évaluée dans ce cas)

4.1 Conseils

Plusieurs choix sont possibles pour lire le ficher : utiliser `Scanner` (plus simple) ou bien `BufferedReader` (étonnament plus rapide). En fait si vous programmez bien, le goulot d'étranglement temporel sera la conversion des chaînes de caractère du fichier d'entrée.

Pour les deux dernieres réponses, il est nécéssaire d'implémenter un **parcours en largeur** (BFS). Pour l'implémenter le parcours, il faut une file (rappel : vous avez le droit d'utiliser des structures de taille non constante du moment qu'elles seront déréférencées à la fin du parcours). En Java elles implémentent `Deque` (abbréviation de *Double Ended QUEue*) pour stocker les sommets à visiter, par exemple une `ArrayDeque` ou une `LinkedList`.

4.2 Première extension possible : renuméroter les sommets

Comme on a vu en section 3, la taille `Sn` du tableau de sommets est soit le plus grand numéro de sommet plus 1, soit le nombre de sommets. Mais dans ce deuxième cas il faut renuméroter les sommets. Dans les affichages on affichera l'ancien nom, qu'il faut donc conserver.

4.3 Deuxième extension possible : gérer les fichiers d'entrée non triés

En section 2 on a vu que les fichiers d'entrés devait être triés, et que la commande `sort` peut être utilisée en pré-traitement. Mais vous pouvez choisir de lire aussi des fichiers non triés.

Alors que la représentation en mémoire impose que tous les arcs ayant origine du sommet x soit consécutifs dans le fichier des arcs.

4.4 `readMe.txt`

Si vous faites des extensions, un fichier intitulé obligatoirement `readMe.txt` devra préciser lesquelles. Ce fichier doit aussi contenir un lien vers les sources si vous avez copié des bouts de code en ligne, sous peine que sinon je détecte cela comme un pompage et vous mette 0. Ce fichier peut contenir toute autre information utile, mais je ne demande pas un rapport : il n'est pas obligé d'écrire un `readMe.txt` s'il n'y a rien de particulier à dire.

5 Exemples

Si `exemple.txt` contient :

```
# Exemple
1 2
1 4
1 6
2 4
4 8
8 10
11 12
12 13
```

alors la commande `java TP1 exemple.txt 1` avec l'option de renumérotation doit afficher :

```
9
8
3
6
3
XXX
```

car il y a 9 sommets : 1,2,4,6,8,10,11,12,13 (sans renumérotation le premier nombre affiché serait 14). On affiche ensuite 8 car il y a 8 arcs, puis 3 car le sommet de plus fort degré (le sommet 1) a 3 voisins (2, 4 et 6). Ensuite on affiche 6 car les sommets 1,2,4,6,8 et 10 sont accessibles depuis 1. Ensuite on affiche 3 car le plus long chemin partant de 1 est 1,2,8,10 et sa longueur (en nombre d'arcs) est 3, donc le sommet 1 a eccentricité 3. Enfin la dernière ligne mon programme affiche un nombre mais comme le graphe n'est pas orienté symétrique il n'a aucun sens, je l'ai remplacé par XXX.

La commande `/usr/bin/time -f"%e\n%M"` donne sur deux lignes le temps d'éxécution (en seconde) et la mémoire **maximale** (en Ko) allouée à un moment donnée par le processus (toute la JVM). Les deux dernières valeurs données sont les valeurs du corrigé sur ma propre machine. Vous pouvez faire mieux, soit que vous ayez plus de CPU, soit que vous programmiez plus efficacement que moi ! Par contre mon script tue les programmes au bout de 10 minutes, donc un temps supérieur à 10 minutes vous sera compté comme un échec.

— Le graphe `as20000102.txt` est celui de `as-733` sur Stanford, les autres ont des noms explicites.
— Certains fichiers, dont `wiki-Vote`, `ca-AstroPh.txt`, `roadNet-CA` et `web-BerkStan.txt`, contiennent des `\r` (retour à la ligne à la DOS) qui doivent être enlevés, d'où le `sed` initial parfois.
— Seul `as20000102.txt` est trié comme il faut.
— `ca-AstroPh.txt` est un graphe orienté symétrique mais non trié, et avec beaucoup de doublons (arcs répétés)
— On a toujours choisi un sommet de départ dans la plus grande composante connexe, de sorte que le nombre de sommets accessibles (quatrième ligne de sortie) des version `_sym.txt` (orienté symétrique) soit la taille de la plus grande composante connexe, et il est égal au paramètre *Nodes in largest WCC* dans la base de Stanford

```
$ /usr/bin/time -f"%e\n%M" java TP1  as20000102.txt 42
6473
13895
1458
28
3
XXX (nombre effacé a posteriori)
0.14
50672
$ awk '/^[^#]/ {print $0"\n" $2 " " $1 }' as20000102.txt
  | sort -u -k 1n -k 2n > as20000102_sym.txt
$/usr/bin/time -f"%e\n%M" java TP1 as20000102_sym.txt 42
6474
26467
1459
6474
6
1
0.20
55192
$ sed -e 's/\r//' wiki-Vote.txt | sort -u -k 1n -k 2n > wiki-Vote_sorted.txt
$ /usr/bin/time -f"%e\n%M" java TP1 wiki-Vote_sorted.txt 42
7115
103689
893
2317
4
XXX (nombre effacé a posteriori)
0.25
66312
```

```
$ sed -e 's/\r//' wiki-Vote.txt | awk '/^[^#]/ {print $0 "\n" $2 " "  $1 }'
    |  sort -u -k 1n -k 2n > wiki-Vote_sym.txt
$ /usr/bin/time -f"%e\n%M" java TP1 wiki-Vote_sym.txt 42
7115
201524
1065
7066
5
24
0.30
71656
$ sed -e 's/\r//' ca-AstroPh.txt |  sort -u -k 1n -k 2n > ca-AstroPh_sorted.txt
$ /usr/bin/time -f"%e\n%M" java TP1 ca-AstroPh_sorted.txt 289
18772
396160
504
17903
9
290
0.43
99692
$ sed -e 's/\r//' roadNet-CA.txt | awk '/^[^#]/ {print $0 "\n" $2 " "  $1 }'
    |  sort -u -k 1n -k 2n > roadNet-CA_sym.txt
$ /usr/bin/time -f"%e\n%M" java TP1 roadNet-CA_sym.txt 0
1965206
5533214
12
1957027
554
2638
2.16
647924
$ sed -e 's/\r//' web-BerkStan.txt | awk '/^[^#]/ {print $0 "\n" $2 " "  $1 }'
    |  sort -u -k 1n -k 2n > web-BerkStan_sym.txt
$ /usr/bin/time -f"%e\n%M" java TP1 web-BerkStan_sym.txt 42
685230
13298940
84230
654782
123
676
3.30
853728
```

```
$ awk '/^[^#]/ {print $0 "\n" $2 " "  $1 }' soc-pokec-relationships.txt
    |  sort -u -k 1n -k 2n > soc-pokec-relationships_sym.txt
$ /usr/bin/time -f"%e\n%M" java TP1 soc-pokec-relationships_sym.txt 1
1632803
44603928
14854
1632803
9
1
34.27
924128
```