Grands Réseaux d'Interaction
TP 2 : deux mesures d'importance d'un sommet

Fabien de Montgolfier
fm@irif.fr

février 2023

Les règles pour le rendu et l'évaluation restent les mêmes que le TP1 (archive sans sous-répertoire, pas de package..). En particulier votre programme doit pouvoir être lancé par :

```
unzip *.zip
javac TP2.java
java TP2 -b exemple.txt 1
```

La prochaine scéance de TP, où l'on pourra discuter du code du TP2 et où le TP3 sera présenté, aura lieu :

— Pour le groupe 1, le 24 février (rendu du TP2 sur Moodle le 24 février à 23h59)
— Pour le groupe 1, le 3 mars (rendu du TP2 sur Moodle le 3 mars à 23h59)

La **structure de données du TP1 reste d'usage obligatoire** : en particulier vous n'avez droit qu'à des variables locales aux méthodes (de chargement ou de calcul), et pour stocker le graphe à **deux tableaux** (un `Sommet[]` et un `int []` pour les arêtes) et à d'autres attributs de taille constante en mémoire dans les classes `Sommet` ou `Graphe`.

Pour l'évaluation, on tiendra davantage compte de la **mémoire allouée** que pour le TP1.

1 Travail demandé

Faire un programme Java qui prend en paramètre de la ligne de commande

1. une commande, qui peut être `"-b"` ou `"-c"`
2. un nom de fichier (graphe orienté, symétrique ou non, au format du TP1)
3. et un numéro de sommet x (attention, si vous renumérotez, le numéro est l'ancien nom)

Votre programme affiche comme résultat, soit "ERREUR" et un message, soit un seul nombre, rien d'autre ! Le nombre affiché est

— Si la commande est `"-c"` alors la cardialité entrante du sommet x
— Si la commande est `"-b"` alors la betweeness centrality du sommet x

2 Cardialité entrante

Étant donné un graphe orienté (symétrique ou pas) G et un entier k, le *k-cœur entrant* de G est le sous graphe (unique) de G consistant à enlever les sommets de **degré entrant** inférieur (strictement) à k de G, itérativement, jusqu'à stabilisation. On dit que la cardialité entrante du sommet x est k si x appartient au k-cœur entrant mais pas au $(k+1)-$cœur entrant.

La cardialité sortante, vue en cours, utilise le degré sortant. Mais elle est plus difficile à calculer. En effet, elle demande à travailler sur le graphe transposé, alors que la cardialité entrante travaille sur le graphe "normal". L'algorithme de calcul de la cardialité entrante du sommet v est en effet :

1. Calculer le degré entrant de chaque sommet
2. Puis, pour k de 0 à n :
   — tant qu'il existe un sommet x de degré entrant $< k$,
     — l'enlever
     — décrémenter le degré entrant de tous ses voisins sortants
3. Si on enlève v on s'arrête et on retourne $k-1$ (v n'est **pas** dans le k-cœur entrant mais était dans le $(k-1)-$cœur entrant).

Bien sûr "enlever" un sommet consiste juste à mettre un flag à `false`, notre structure de donnée n'est pas décrémentale. On touche juste au flag présent/enlevé du sommet, et au degré entrant de ses voisins sortants, qui compte le **nombre de voisins entrants présents**, pas les enlevés.

Pour améliorer la complexité et repérer les sommets de degré $< k$, il peut être intéressant de maintenir des listes de sommets par degré entrant, ou une file de priorité des sommets triés par degré entrant décroissant.

3 Betweenness centrality

Parmi les très nombreuses[1] mesures de centralité existantes, nous nous intéresseront dans le TD à la **betweenness centrality**[2]. On note $bet(v)$ la *betweenness centrality* du nœud v. C'est un nombre compris entre 0 et 1. Informellement, $bet(v)$ est la proportion de plus courts chemins du graphe passant par v. Un peu plus formellement, c'est la moyenne, pour chaque paire de sommets, de la proportion de plus courts chemins joignant cette paire passant par v.

Étant donnés deux sommets s et t du graphe, il peut y avoir de nombreux plus courts chemins entre s et t (la quantité peut être exponentielle en le nombre de sommets du graphe). On note $npcc(s,t)$ le nombre de plus courts chemins entre s et t. Pour un sommet v **différent** de s et de t ($s \neq t \neq v \neq s$) on note $npcc(s,v,t)$ le nombre de plus courts chemins entre s et t passant par v.

La centralité de v sur le trajet de s à t est définie par

1. `http://schochastics.net/sna/periodic.html`
2. on peut traduire par «centralité d'intermédiarité» mais ce n'est pas très beau. Ou traduire par «intermédiaritude» mais ce n'est pas très français.

$$bet(s,v,t) = \begin{cases} 0 \text{ si } npcc(s,t) = 0 \\ \dfrac{npcc(s,v,t)}{npcc(s,t)} \text{ sinon} \end{cases}$$

C'est donc un nombre qui vaut entre 0 (si v n'est situé sur aucun plus court chemin entre s et t, ce qui arrive par exemple si ces trois sommets ne sont pas dans la même composante connexe) à 1 (si tous les plus courts chemins entre s et t passent par v, ce qui veut dire qu'enlever v augmenterait la distance entre s et t voire déconnecterait le graphe).

Enfin, la centralité tout court de v est

$$bet(v) = \frac{1}{(n-1)(n-2)} \sum_{s \neq t \neq v \neq s} bet(s,v,t)$$

où n est le nombre de sommets. Comme la somme porte sur $(n-1)(n-2)$ termes (le nombre de couples de sommets ne comprenant pas v) il s'agit donc bien d'une moyenne de nombre entre 0 et 1, ce qui veut dire que $bet(v)$ est compris entre 0 et 1.

nombre de plus courts chemins

Comment calculer $npcc(s,t)$ et $npcc(s,v,t)$? Pour des graphes non valués (ce qui est le cas ici) un simple parcours en largeur partant de s donne la distance entre un sommet s et tous les autres. Ou bien, l'algorithme de Floyd-Warshall calcule la matrice des distances en $O(n^3)$. Donc on suppose que la fonction $dist(s,t)$ donnant la distance (longueur d'un plus court chemin) entre deux sommets est disponible.

Notons $Pred(s,t)$ l'ensemble des **prédécesseurs** de t sur les plus courts chemins de s à t. Cela signifie qu'il existe un plus court chemin de s à t dont l'avant-dernier sommet est p (et le dernier est donc t). $p \in Pred(s,t)$ si et seulement si pt est une arête du graphe et $dist(s,p) + 1 = dist(s,t)$. Le nombre de plus courts chemins entre s et t se calcule par

$$npcc(s,t) = \begin{cases} 1 \text{ si } dist(s,t) = 1 \\ \sum_{p \in Pred(s,t)} npcc(s,p) \text{ sinon} \end{cases}$$

Un sommet v se trouve sur (au moins) un plus court chemin entre s et t si et seulement si $dist(s,t) = dist(s,v) + dist(v,t)$. Et dans ce cas on a

$$npcc(s,v,t) = npcc(s,v) * npcc(v,t)$$

Notez que l'on peut modifier le parcours en largeur, ou bien Floyd-Warshall, pour que $npcc(s,t)$ soit calculé en même temps que $dist(s,t)$. On obtient donc respectivement un algorithme en $O(nm)$ et en $O(n^3)$, ce qui est mauvais... Il peut être intéressant de faire des calculs en parrallèle.

4 Exemples

Le graphe `ex2.txt` est fait de deux cliques de 5 sommets avec le sommet 6 faisant un pont entre les deux. Le sommet 1 appartient à une des cliques. Sa betweeness est de 0 : il n'est sur aucun plus court chemin. Le sommet 6 est sur tous les chemins reliant la première clique à la deuxième, soit 25 couples sur les 45 possibles ne le contenant pas, d'où une betweeness de 25/45.

```
> java TP2 -c ex2.txt 2
4
> java TP2 -c ex2.txt 6
2
> java TP2 -b ex2.txt 2
0.0
> java TP2 -b ex2.txt 6
0.5555555555555556
```

En regardant maintenant les graphes de Stanford, on constate que beaucoup sont des graphes non-orienté où l'on écrit l'arête $i \to j$ avec toujours $i < j$. Cela implique que considéré comme des graphes orientés, ils ont cardialité entrante 0 (le sommet 0 a toujours degré entrant 0. On l'enlève. Mais le sommet 1 a nécéssairement cardialité entrante 0 maintenant : on l'enlève, etc.). Dans ce cas il faut le symétriser comme au TP1. La cardialité entrante devient alors égale à la cardialité du graphe non-orienté correspondant. Cela peut aussi avoir une grande influence sur la betweeness. Notez que pour `as20000102.txt` la cardialité de 1 provient de l'existence de boucles comme $1 \to 1$.
`/usr/bin/time -f "%e\n%M"` affiche le temps d'exécution (en s) puis la mémoire utilisée (ko).

```
> /usr/bin/time -f "%e\n%M" java TP2 -b as20000102.txt 1
2.6711601252217057E-5
0.37
489904
> /usr/bin/time -f "%e\n%M" java TP2 -c as20000102.txt 1
1
0.09
47336
> 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 TP2 -c as20000102_sym.txt 1
12
0.13
51568
>/usr/bin/time -f "%e\n%M" java TP2 -b as20000102_sym.txt 1
0.3663158541423469
2.89
740644
```

```
> /usr/bin/time -f "%e\n%M" java TP2 -c facebook_combined.txt 107
0
0.16
61356
> /usr/bin/time -f "%e\n%M" java TP2 -b facebook_combined.txt 107
2.4316335805821553E-4
0.71
213508
> awk '/^[^#]/ {print $0"\n" $2 " " $1 }' facebook_combined.txt |
   sort  -u -k 1n -k 2n > facebook_combined_sym.txt
> /usr/bin/time -f "%e\n%M" java TP2 -c facebook_combined_sym.txt 107
70
0.22
72180
> /usr/bin/time -f "%e\n%M" java TP2 -b facebook_combined_sym.txt 107
0.48051807855146267
5.16
480328
```

Certains graphes comme `email-Eu-core.txt` ne vérifient pas le format :

```
> java TP2 -c email-Eu-core.txt 160
ERREUR graphe mal trié : liste d'adjacence du sommet 12 non consecutive !
> sort  -u -k 1n -k 2n email-Eu-core.txt >  email-Eu-core-sorted.txt
> /usr/bin/time -f "%e\n%M" java TP2 -c  email-Eu-core-sorted.txt 160
27
0.09
48660
>  /usr/bin/time -f "%e\n%M" java TP2 -b  email-Eu-core-sorted.txt 160
0.07212078608026577
0.31
69784
> /usr/bin/time -f "%e\n%M" java TP2 -c wiki-Vote.txt 2565
19
0.18
64480
> /usr/bin/time -f "%e\n%M" java TP2 -b wiki-Vote.txt 2565
0.017654409558127946
3.48
773268
```

On voit ci-dessous que l'algorithme de betweeness programmé par votre enseignant n'est pas très efficace : il utilise beaucoup trop de mémoire ce qui le fait échouer sur des moyennement gros graphes. Et que un algorithme surlinéaire devient vite *trop* lent, mais dans mon implémentation c'est la mémoire le goulot d'étranglement : il a besoin de 16Go de RAM. À vous de faire mieux !

```
> /usr/bin/time -f "%e\n%M" java TP2 -c email-Enron.txt 5038
12
0.30
94672
> java TP2 -b email-Enron.txt 5038
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
> /usr/bin/time -f "%e\n%M" java -Xmx16g TP2 -b email-Enron.txt 5038
0.06485117608227937
156.10
15099784
```

En continuant avec seulement la cardialité pour de plus gros graphes :

```
> /usr/bin/time -f "%e\n%M" java TP2 -c ca-AstroPh.txt 84424
26
0.30
102916
> /usr/bin/time -f "%e\n%M" java TP2 -c web-NotreDame.txt 0
25
0.57
174804
> /usr/bin/time -f "%e\n%M" java TP2 -c roadNet-CA.txt 0
2
2.00
881604
> /usr/bin/time -f "%e\n%M" java TP2 -c web-BerkStan.txt 254913
34
2.17
788092
> /usr/bin/time -f "%e\n%M" java TP2 -c soc-pokec-relationships.txt 2
21
22.33
2265276
```