Grands Réseaux d'Interaction
TP n° 2 : Centralité

Remarques importantes :
- Les règles générales en TP restent valides, voir feuille du TP 1.
- En particulier ce TP est à faire en binôme. Vous devez être inscrit (par votre enseignant) dans un groupe Moodle pour pouvoir le rendre. Ce groupe ne pourra comporter une seule personne que sur dérogation.
- Ce TP est à rendre sur Moodle pour le **5 novembre**

I) 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

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

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.

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.

II) 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 $O(n^3)$.

Il s'agit d'une complexité temporelle très mauvaise ! De plus la complexité en espace est très mauvaise aussi : il faut utiliser des **matrices** pour stocker les distances et les *npcc* d'où une complexité mémoire de $\Theta(n^2)$. Votre programme sera donc lent et gourmand en mémoire. Ainsi le corrigé, sur `nivose` pour la centralité de tous les 1004 sommets de `email-Eu-core.txt`, prend 19,5 secondes. Et, sur un ordinateur plus rapide, pour tous les sommets de `Wiki-Vote.txt`, 24 minutes. Notez que l'on peut avoir une complexité en mémoire linéaire, et toutes les modularités en $O(nm)$ au lieu de $O(n^3)$ comme le calcul de $bet(s,v,t)$ pour tout triplet y oblige, avec l'algorithme de Brandes, qui est nettement plus compliqué que celui présenté ici. Voir
`https://kops.uni-konstanz.de/bitstream/handle/123456789/5739/algorithm.pdf`

III) travail demandé

Faire un programme Java qui prend en paramètre de la ligne de commande un nom de fichier (graphe au format de Stanford) et, éventuellement, un certain nombre de numéros de sommets. Votre programme affiche comme résultat,

- Si des numéros de sommet sont passés en pramètre : pour chaque sommet i de la liste en paramètre, son numéro i et sa *betweenness centrality* $bet(i)$.
- Sinon : on le fait pour tous les sommets du graphe

Il faut en plus afficher au plus quatre premières lignes donnant le nombre de doublons et de boucles ignorés, le nombre de sommets, et le nombre d'arêtes, et une dernière ligne donnant la mémoire allouée, comme dans le TP1. La page suivante donne des exemples d'exécution. Le graphe gr1.txt est sur Moodle, les autres sur la base de Stanford.

```
$> java TP2 gr1.txt
Nombre de sommets : 10
Nombre d'aretes : 11
0 : 0.0
1 : 0.041666666666666664
2 : 0.4861111111111111
3 : 0.0462962962962963
4 : 0.0462962962962963
5 : 0.0462962962962963
6 : 0.4305555555555556
7 : 0.08333333333333333
8 : 0.08333333333333333
9 : 0.013888888888888888
Mémoire allouée : 650128 octets
$>  java TP2 gr1.txt 1 2 4 5 6 7 8 9 19 1
Nombre de sommets : 10
Nombre d'aretes : 11
1 : 0.041666666666666664
2 : 0.4861111111111111
4 : 0.0462962962962963
5 : 0.0462962962962963
6 : 0.4305555555555556
7 : 0.08333333333333333
8 : 0.08333333333333333
9 : 0.013888888888888888
19 : Numéro de sommet invalide
1 : 0.041666666666666664
Mémoire allouée : 650128 octets
$>java TP2 email-Eu-core.txt 1 2 3 4 5 6 160
642 boucles ont été ignorées
8865 doublons ont ete supprimes
Nombre de sommets : 1005
Nombre d'aretes : 16064
1 : 0.0011950876146088244
2 : 0.0065697199074932966
3 : 0.0016535722578744285
4 : 0.005547474395328593
5 : 0.030994686545277983
6 : 0.012386641327813848
160 : 0.08741473493634884
Mémoire allouée : 26805848 octets
$> java TP2 Wiki-Vote.txt 1 3 4 5 11 15 2565
2927 doublons ont ete supprimes
Nombre de sommets : 8298
Nombre d'aretes : 100762
1 : 0.0
3 : 1.0844835740935899E-4
```

```
4 : 1.3659863259002135E-5
5 : 8.037629232242434E-6
11 : 0.026237804900677633
15 : 0.014735614860099479
2565 : 0.045033556978395374
Mémoire allouée : 609103600 octets
```