Polytech'Montpellier – IG3 **2009-2010**

PROJET C

Réseaux sociaux : recherche de composantes fortement connexes

Contexte du projet

Les réseaux sociaux tels que facebook sont de plus en plus présents dans notre quotidien et leur analyse est très importante, en particulier pour comprendre la structuration et les relations entre individus, et l'influence des réseaux. Les relations sont alors analysées par différentes méthodes issues notamment de la théorie des graphes. En particulier, la recherche de composantes fortement connexes permet non seulement de retrouver les sous-structures du réseau, mais aussi de pointer les « points d'articulation ».

Travail à réaliser

On considère des données issues d'un réseau social : quels sont les membres (leur id, leur nom et la fréquence avec laquelle ils vérifient leur compte) et quelles sont les relations de confiance entre eux (qui fait confiance à qui).
Dans le cadre de ce projet, il s'agira :

1) *de trouver les composantes fortement connexes* à l'aide des algorithmes connus dans ce cadre (voir annexe), c'est-à-dire les groupes de personnes qui se font confiance réciproquement (directement ou indirectement : si X fait confiance à Y et Y à Z alors on dit que X fait confiance indirectement à Z).
   Une personne **importante** est une personne qui, si elle est supprimée de son groupe, implique que le groupe ne reste plus fortement connexe (au moins deux personnes ne peuvent plus se faire confiance réciproquement). (voir section bonus)
2) *de permettre la diffusion efficace de messages*. On considèrera alors un réseau dans lequel chaque personne doit se connecter et vérifier son compte, en prenant pour hypothèse qu'une personne vérifie son compte régulièrement (toutes les X heures avec X = 1, 2, 24, 72, etc.). Le problème est d'envoyer un message d'une personne à une autre dans le temps le plus court possible. Les personnes n'étant pas forcément amies directement, il faut utiliser des intermédiaires pour transmettre le message. Pour cela, une personne peut renvoyer un message vers une autre personne lorsqu'elle vérifie son compte. On considère alors comme temps de transmission le cas le moins favorable.

Les données en entrée devront être lues à partir d'un fichier puis stockées en mémoire à l'aide de structures adéquates. Les résultats devront être écrits dans un fichier de sortie.

Bonus :

- Analyse de fuite mémoire
- Tester le programme avec de données quasi-réelles de facebook
- Trouver les personnes importantes pour chaque groupe fortement connexe. C'est à vous de décider comment les afficher. La façon recommandée est de mettre une étoile devant les personnes importantes (lorsqu'on affiche les groupes fortement connexes) :
  Ex (voir ci-dessous) : `*12331, *564, 76549`
- Qualité du code (structure, indentation, commentaires), efficacité de la solution (on va tester aussi sur des graphes avec un grande nombre de personnes, 2000 par exemple)

Votre programme doit accepter comme paramètres le nom du fichier d'entrée et le nom du fichier de sortie. Exemple :

./projetc reseau.in resultat.out

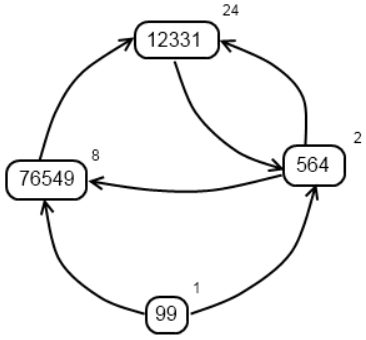
Le temps d'exécution dans tous les cas doit être inferieur à 5s.

Format fichier d'entrée

	Explication
1 ligne	n *(le nombre de personnes dans le réseau social)*
n lignes	id, nom personne, fréquence
1 ligne	m *(le nombre de relations entre les personnes)*
m lignes	id1, id2 *(personne id1 a confiance en id2)*
1 ligne	q *(le nombre de questions que l'on se pose)*
q lignes	id1 -> id2

Exemple **:**

```
4
Pierre Bollampally, 12331, 24
Gouman Rin, 564, 2
Jnkeea van der Zwaan, 76549, 8
Popescu George, 99, 1
6
12331, 564
564, 76549
76549, 12331
564, 12331
99, 564
99, 76549
3
99 -> 564
99 -> 76549
99 -> 12331
```



Format fichier de sortie

	Explication
1 ligne	nc *(le nombre de composantes fortement connexes)*
nc lignes	id1, id2, …
q lignes	t_min : id1, id2, … , idk

Exemple (pour le fichier d'entrée ci-dessus):

```
2
12331, 564, 76549
99
2 : 99, 564
8 : 99, 76549
26 : 99, 564, 12331
```

Interface avec facebook

Un code vous sera fourni qui vous permettra de récupérer directement des données de facebook. Le code met à votre disposition une fonction de signature :

```
int get_friends_list(char* id, fb_account** friends)
```

Attention ! Le code utilise l'outil *curl* (http://curl.haxx.se/) pour faire les requêtes HTTP vers les serveurs facebook et la librairie TRE (http://laurikari.net/tre/about/) pour utiliser des expressions régulières. Un autre document vous sera fourni qui explique le code et comment l'utiliser.

Le code est aussi un bon exemple de comment interagir avec un site web en utilisant le C et aussi comment utiliser les expressions régulières (comme vues en cours).

Vous pouvez utiliser cette fonction pour générer vos tests de deux façons :

- A partir de l'id facebook de quelqu'un (que vous trouvez dans des url comme http://www.facebook.com/home.php?#!/profile.php?id=565375466) vous pouvez recupérer la liste des amis (id et nom). Après, vous pouvez générer de façon aléatoire les relations entre les amis et la fréquence avec laquelle chacun d'entre eux vérifie son compte.
- A partir de l'id facebook de quelqu'un récupérez la liste d'amis. Après, vous continueé avec chaque ami en récupérant sa liste d'amis, etc. Attention ! Il faut s'arrêter à un moment donné, sinon le graphe deviendra vite très très grand. Deux ou trois niveaux maximum doivent suffire. Comme pour le point précédent il faut générer de façon aléatoire si un ami fait confiance à un autre et les fréquences.

Modalités

Le projet ne sera considéré comme fini qu'après validation sur un très grand ensemble de tests comprenant des types et tailles de réseaux différents. Une **soutenance** aura lieu pendant laquelle vous devrez présenter tout ce que vous avez fait.

Le projet est à rendre par mail à dinu@lirmm.fr et laurent@lirmm.fr à une date qui vous sera indiquée ultérieurement. Le sujet du mail devra être de la forme < **[Projet C] noms** > et l'archive jointe au mail devra être nommée <**noms.zip**> ou <**noms.tar**> où noms correspond à la *liste des noms de famille des participants par ordre alphabétique séparés par un _* .

L'archive devra comporter :

- un *readme*
- un répertoire des fichiers sources (/src) avec
  - un *makefile*
  - un sous-répertoire des fichiers .c (/src/source)
  - un sous-répertoire des fichiers .h (/src/headers)
- un répertoire des tests et leurs résultats (/test)
- un répertoire de documentation comportant (/doc)
  - la charte de programmation adoptée
  - un rapport de synthèse (maximum 20 pages)
  - un rapport de tests
  - un schéma (PDF) de description des modules et de leurs interactions.

Le projet peut être fait en C ou C++

Annexe

Composante fortement connexe d'un sommet

Soit $G = (X, U)$ un graphe orienté. On définit la *composante fortement connexe* d'un sommet x, notée $CFC_G(x)$ par :

$CFC_G(x) = \{y \in X$ tq il existe dans G un chemin de x à y et un chemin de y à $x\}$

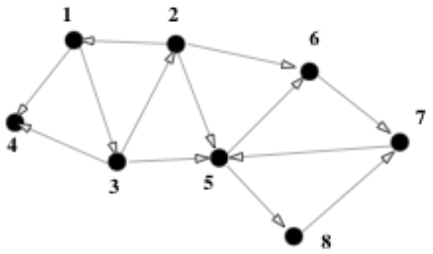


FIG. 1 – Sur ce graphe, $CFC_G(1) = \{1, 2, 3\}, CFC_G(4) = \{4\}, CFC_G(6) = \{5, 6, 7, 8\}$

L'ensemble des composantes fortement connexes de G forme une partition des sommets de G.

Le calcul des composantes fortement connexes de G repose sur deux parcours en profondeur (algorithme PP, voir annexe), l'un du graphe G, l'autre de son dual G^D (c'est-à-dire le graphe obtenu en inversant le sens des arcs de G)

Algorithme de calcul des composantes fortement connexes de G

Données : $G = (X, U)$ un graphe orienté

Résultat : Les composantes fortement connexes de G

1– Exécuter $PP(G)$ pour calculer $\{f(u)/u \in X\}$;

2– Caculer G^D le graphe dual de G (obtenu en inversant le sens des arcs de G);

3– Exécuter $PP(G^D)$ mais dans la boucle principale, on considérera les sommets u par ordre décroissant des $f(u)$;

4– Chaque arborescence de la forêt obtenue en 3 est une composante fortement connexe;

Sur l'exemple, le premier parcours en profondeur de G donne ce qui suit :

Le deuxième parcours en profondeur , parcours de G^D en utilisant l'ordre décroissant des $f(u)$ donne :

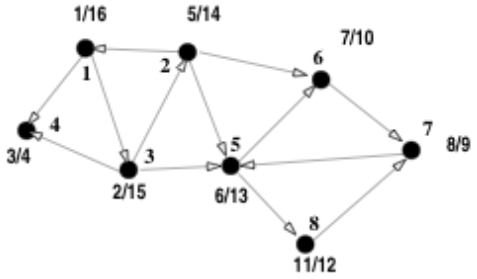


FIG. 2 – Les sommets ordonnés par ordre décroissant des $f(u)$ sont : $1, 3, 2, 5, 8, 6, 7, 4$

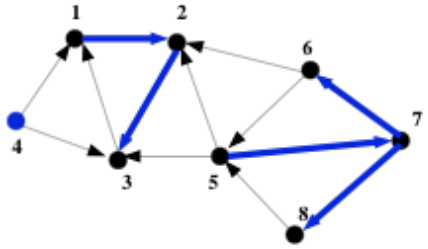


FIG. 3 – Les composantes connexes de la forêt de liaison sont bien $\{1, 2, 3\}, \{4\}, \{5, 6, 7, 8\}$

Annexe : algo de parcours en profondeur

PP(G)
Données : $G = (X, U)$ un graphe orienté
Résultat : les valeurs $d(u)$ et $f(u)$ pour tout u de X
pour chaque $u \in X$ **faire**
  etat(u) <- blanc;
fin
t <- 0;
pour chaque $u \in X$ **faire**
  si *etat(u)=blanc* **alors**
    PProf(u);
  fin
fin

PProf(u)

Données : $G = (X, U)$ un graphe orienté, u un sommet de G

etat(u) = atteint;
t <- t + 1;
d(u) <- t ;
pour chaque $v \in Successeurs(u)$ **faire**
    si *etat(v)=blanc* **alors**
        PProf(v);
    fin
fin
etat(x)=explore ;
t <- t + 1;
f(u) <-t;