Rapport de projet
Module : Graphe
Licence Informatique – 3ème Année

Dégénérescence des grands graphes réels

Nom : **BAH**
Prénom : **Saikou Oumar**
Groupe : **A-TP1**

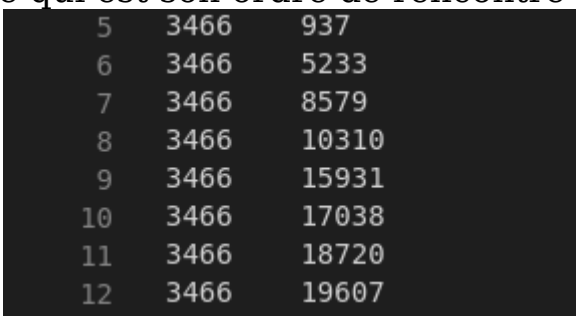
Nom : **DIALLO**
Prénom : **Mamoudou**
Groupe : **A-TP1**

Sommaire

I. Représentation du graphe

Ce projet est réalisé avec le langage c++.
Le graphe est représenté à l'aide d'un tableau de listes d'incidence. Une cellule x contient la liste de tous les voisins du x-ième sommet rencontré dans le fichier au format SNAP qui servira à la construction du graphe.
Le tableau, ainsi les listes d'incidence qu'il contient, sont en réalité un vecteur 2D (un vecteur de vecteur) d'entier appelé voisinage.
On utilisera tout au long du programme en plus du vecteur 2D voisinage, plusieurs autres variables globales :

- code : il s'agit d'une map de deux entiers, le premier val est la valeur du sommet récupéré dans le fichier SNAP, le second code est le code par lequel il est représenté dans voisinage. En effet les sommets dans les fichiers SNAP ne commencent pas forcement par 0 et ne sont pas forcement consécutifs, du coup pour éviter des milliers de vecteurs vides, chaque sommet recevra un code qui est son ordre de rencontre dans le fichier.



  Un exemple serait cet extrait de fichier SNAP dont le premier sommet commence à 3466 et la valeur la plus grande allant au-delà de 19607. Ne pouvant nous permettre de faire un vecteur de plus de 19607 cellules dont la plus part sont vides, on codera le sommet 3466 avec 0 car premier sommet trouvé. Dans la même logique 937 sera codé 1, 5233 aura pour code 2, ainsi de suite. Le tableau voisinage contiendra donc les codes et non les valeurs. L'avantage sera donc d'avoir un vecteur avec autant de cellules que de sommets mais aussi d'accéder, grâce à la map, à n'importe quel code connaissant la valeur et à n'importe quelle valeur connaissant son code.
- marqued : dans le projet, les sommets ne sont pas réellement supprimés mais marqués, marqued est un vecteur de booléen contenant autant de cellules que de sommets. Et comme les sommets sont codés à partir de 0, chaque cellule i de marqued indique si le sommet codé i est marqué (true) ou non (false).
- numCentre : un vecteur d'entier contenant les numéros de centre de tous les sommets, tout comme marqued chaque cellule i correspond au sommet codé i.

II. Construction du graphe

La fonction **construction()** permet de construire un graphe en remplissant voisinage en basant sur un fichier au format SNAP dont le chemin est donné en paramètre.
Dans un fichier SNAP, chaque ligne contient 2 valeurs séparées par un espace ou une tabulation, elles représentent les 2 sommets formant une arête. Les lignes commençant par % ou # sont des commentaires.
Si l'ouverture du fichier en lecture ne retourne aucune erreur, on récupère chaque ligne non vide qui n'est pas un commentaire. Elle est ensuite décomposé pour récupérer les 2 valeurs :

- si aucun code n'est attribué à cette valeur (il s'agit alors d'une nouvelle valeur), on lui en donne un qui est inséré dans code tout en lui créant une ligne dans voisinage, le nombre d'arêtes est incrémenté .
- On récupère deux valeurs par lignes, chacun sera ajouté dans la liste des voisins de l'autre.

Après lecture du fichier, il est fermé et les tableaux sont initialisés :
- marqued toutes les cellules à false ;
- numCentre toutes les cellules à 0 ;

III. L'algorithme de dégénérescence

Pour calculer la dégénérescence d'un graphe, nous avons utilisé l'algorithme suivant :

```
k = 0
Tant que tous les sommets ne sont pas marqués
        Pour chaque sommet du graphe
        Si il n'est pas marqué et son degré ≤ k
                On le marque
                Graphe modifié = vrai
                numCentre[sommet] = k
        Fin Si
        Fin Pour
        Si Graphe non modifié
        k++
        Fin si
        Graphe modifié = faux
Fin Tant que
Retourne k+1
```

Nous répétons le parcours intégrale du graphe tant qu'il existe des sommets non marqués. Lors de chacun de ses parcours on marque tous les sommets de degré inférieur ou égal à k. Si aucun sommet n'a été marqué pour un k donné, on l'incrémente.

NB : le degré d'un sommet est le nombre de sommet non marqué dans son voisinage.

IV. L'algorithme de dégénérescence par Matula & Beck

Nous essayerons d'implémenter l'algorithme de Matula & Beck pour le calcul optimisé de la dégénérescence avec quelques petites modifications.

Algorithme original

- Initialiser une liste de sortie L .
- Calculer un nombre d_v pour chaque sommet v dans G , le nombre de voisins de v qui ne sont pas déjà dans L . Initialement, ces nombres ne sont que les degrés des sommets.
- Initialiser un tableau D tel que $D[i]$ contienne une liste des sommets v qui ne sont pas déjà dans L pour lesquels $d_v = i$.
- Initialiser k à 0.
- Répétez n fois :
  - Parcourez les cellules du tableau $D[0]$, $D[1]$, ... jusqu'à trouver un i pour lequel $D[i]$ est non vide.
  - Fixer k à max(k , je)
  - Sélectionnez un sommet v dans $D[i]$. Ajoutez v au début de L et supprimez-le de $D[i]$.
  - Pour chaque voisin w de v pas déjà dans L , soustrayez un de d_w et déplacez w vers la cellule de D correspondant à la nouvelle valeur de d_w .

A la différence de ce dernier,

I. Nous ne parcourons pas le tableau à partir de 0, mais à partir de i-1 sauf pour i = 0, car les degrés des sommets ne diminuant que de 1 à chaque fois, D de 0 à i-2 resteront très certainement vide.
II. Le sommet n'est pas supprimé, il est uniquement marqué. Du coup, D[i] est considéré vide si il l'est réellement ou si tous ses sommets sont marqués.
III. Les voisins w de v ne sont pas déplacés mais copiés dans la cellule D précédente. Cela veut dire plusieurs cellules peuvent contenir un même sommet sans qu'il ne soit marqué. L'algorithme marquera le sommet dans la cellule i la plus petite et ne le considérera pas dans les cellules supérieurs s'il venait à le recroiser.

V. Exemples d'utilisation

Plusieurs fichiers SNAP sont disponibles dans le répertoire Files.
Pour le fichier « facebook.txt » avec 88234 lignes et une dégénérescence de 116, les temps moyens pour : la construction (0.14s), la dégénérescence (0.75s), la dégénérescence Matula & Beck (0.33s).

```
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ g++ -Wall projet.cpp -o projet
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ time ./projet
Fichier Files/facebook.txt ouvert !!
Degenerence : 116
Degenerence by Matula & Beck : 116

real    0m1,213s
user    0m1,205s
sys     0m0,008s
```

Pour le fichier test représentant l'exemple sur le sujet de projet, on obtient :

```
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ time ./projet
Fichier Files/test.txt ouvert !!
Degenerence : 4
Tableau des numéros de centre :
|3|2|3|3|3|3|2|2|2|1|

Degenerence by Matula & Beck : 4
Tableau des numéros de centre :
|3|2|3|3|3|3|2|2|2|1|
Liste de sortie L :
|0|3|2|4|5|6|7|8|1|9|

real    0m0,005s
user    0m0,002s
sys     0m0,003s
```

Le tableau des numéros de centre de la liste de sortie L de l'algorithme de Matula & Beck ne sera pas affiché pour les autres car trop imposantes.
On a également pour les fichiers info-power et info-USAir97 les résultats suivant :

```
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ g++ -Wall projet.cpp -o projet
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ time ./projet
Fichier Files/inf-power.mtx ouvert !!
Degenerence : 6
Degenerence by Matula & Beck : 6

real    0m0,162s
user    0m0,154s
sys     0m0,004s
```

```
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ g++ -Wall projet.cpp -o projet
bah@debian:~/Bureau/L3/S6/Graphes/Projet/Degenerescence$ time ./projet
Fichier Files/inf-USAir97.mtx ouvert !!
Degenerence : 27
Degenerence by Matula & Beck : 27

real    0m0,031s
user    0m0,023s
sys     0m0,004s
```