TP3 AA
Polytech Grenoble, RICM3

Jean-François Méhaut
UGA-CEA/LIG

27 Mars 2018

Ce TP comprend deux parties : Une première partie sur des fonctions élémentaires sur les graphes. Une seconde partie sur l'examen d'Algo qui a été proposé en 2017.

1 Graphes

Un graphe est composé d'un ensemble de nœuds/sommets reliés par des arcs/arêtes. Les déclarations de type sont dans le fichier `graphe.h`. Le sujet de ce TP porte sur des graphes **pondérés** (poids sur les arcs/arêtes) et **orientés** (flèches sur les arcs/arêtes). Le fichier `graphe.c` contient les fonctions de lecture/affchage d'un graphe passé en paramètre au programme.

L'archive contient les fichiers `graphe.h`, `graphe.c`, `Makefile` ainsi que différents fichiers contenant différents graphes. Les fichiers de données contiennent plusieurs lignes : La première indique le nombre de nœuds du graphe. Les lignes suivantes les labels des nœuds du graphe. Les lignes suivantes les arcs avec les poids. Les fonctions `lire_graphe` et `ecrire_graphe` sont fournies. Le fichier `graphe.c` contient les fichiers à compléter. Le type `chemin_t` est à compléter dans le fichier `graphe.h`.

1.1 Fonctions Élémentaires sur les graphes

Des fonctions élémentaires sont à implémenter :
- Calcul du nombre d'arcs d'un graphe `g`.
- Calcul du nombre de sommets d'un graphe `g`.
- Calcul des degrés entrant et sortant d'un noeud n du graphe `g`.
- Calcul des degrés minimaux et maximaux d'un graphe `g`.
- Dire si un graphe `g` est indépendant ou pas. Les arêtes du graphe n'ont pas de sommet en commun.
- Dire si un graphe `g` est complet ou pas. le graphe est complet si toutes les paires de noeud/sommet sont jointes par un arc.
- Dire si un graphe `g` est régulier ou pas. le graphe est régulier si tous les nœuds/sommet ont le même degré.
- Afficher le graphe `g` avec un parcours en profondeur.
- Fonction de coloriage d'un graphe `g`.
- Afficher le graphe `g` avec un parcours en largeur.
- Calcul du plus court chemin entre deux nœuds d'un graphe `g`.

1.2 Examen 2017

Vous compléterez les fichiers `graphe.h` et `graphe.c` avec les fonctions de l'examen 2017. Quelques définitions caractérisant les chemins et les graphes :

1. Un chemin est une suite consécutive d'arcs dans un graphe orienté.
2. Un chemin **élémentaire** est un chemin ne passant pas deux fois par un même nœud, c'est à dire un chemin dont tous les nœuds sont distincts.
3. Un chemin **simple** est un chemin ne passant pas deux fois par une même arête, c'est à dire un chemin dont toutes les arêtes sont distinctes.
4. Un chemin est dit **Eulérien** si toutes les arêtes du graphe sont utilisées dans le chemin.
5. Un graphe est dit **Eulérien** si il existe au moins un chemin qui soit **Eulérien**.
6. Un chemin est dit **Hamiltonien** si tous les nœuds du graphe sont utilisés dans le chemin.
7. Un graphe est dit **Hamiltonien** si il existe au moins un chemin qui soit **Hamiltonien**.
8. La **longueur** d'un chemin est la somme des poids des arêtes.
9. La **distance** entre deux nœuds x et y est la longueur du plus court chemin entre x et y.
10. L'**excentricité** d'un nœud est sa distance maximale avec les autres nœuds du graphe.
11. Le **diamètre** d'un graphe est l'excentricité maximale de ses nœuds.

1.3 Questions

1. (2 points) Définissez le type `chemin_t` mémorisant les informations nécessaires pour un chemin. Cette déclaration de type s'appuiera, soit sur la représentation par matrice d'adjacence des graphes, soit sur la représensation par liste chaînee des graphes. Le choix est important pour les questions suivantes. Ce type `chemin_t` va être utilisé dans les questions suivantes.
2. (1 point) Décrivez en C l'implémentation de la fonction `chemin_elementaire` qui vérifie si un chemin est **élémentaire** ou pas. La fonction `chemin_elementaire` renvoie 1 si le chemin `c` est **élémentaire**, 0 sinon.

```
int chemin_elementaire (pgraphe_t g, chemin_t c)
{
  ...
}
```

3. (1 point) Décrivez en C l'implémentation de la fonction `chemin_simple` qui vérifie si un chemin est **simple** ou pas. La fonction `chemin_simple` renvoie 1 si le chemin `c` est **simple**, 0 sinon.

```
int chemin_simple (pgraphe_t g, chemin_t c)
{
  ...
}
```

4. (2 points) Décrivez en C l'implémentation de la fonction `chemin_eulerien` qui vérifie si un chemin est **Eulérien** ou pas. La fonction `chemin_eulerien` renvoie 1 si le chemin `c` est **Eulérien**, 0 sinon.

```
int chemin_eulerien (pgraphe_t g, chemin_t c)
{
  ...
}
```

5. (2 points) Décrivez en C l'implémentation de la fonction `chemin_hamiltonien` qui vérifie si un chemin est **Hamiltonien** ou pas. La fonction `chemin_hamiltonien` renvoie 1 si le chemin `c` est **Hamiltonien**, 0 sinon.

```
int chemin_hamiltonien (pgraphe_t g, chemin_t c)
{
  ...
}
```

6. (3 points) Décrivez en C l'implémentation de la fonction `graphe_eulerien` qui vérifie si un graphe est **Eulérien** ou pas. La fonction `graphe_eulerien` renvoie 1 si le graphe `g` est **Eulérien**, 0 sinon.

```
int graphe_eulerien (pgraphe_t g)
{
  ...
}
```

7. (3 points) Décrivez en C l'implémentation de la fonction `graphe_hamiltonien` qui vérifie si un graphe est **Hamiltonien** ou pas. La fonction `graphe_hamiltonien` renvoie 1 si le graphe `g` est **Hamiltonien**, 0 sinon.

```
int graphe_hamiltonien (pgraphe_t g)
{
  ...
}
```

8. (2 points) Décrivez en C l'implémentation de la fonction `distance` qui calcule la **distance** entre deux nœuds x et y du graphe `g`.

```
  int distance (pgraphe_t g, pnoeud_t x, pnoeud_t y)
{
  ...
}
```

9. (2 points) Décrivez en C l'implémentation de la fonction `excentricite` qui calcule pour un nœud `n` son **excentricité** dans le graphe `g`.

```
int excentricite (pgraphe_t g, pnoeud_t n)
{
  ...
}
```

10. (3 points) Décrivez en C l'implémentation de la fonction `diametre` qui calcule le **diamètre** du graphe `g`.

```
int diametre (pgraphe_t g)
{
  ...
}
```