TP3 Algo Avancée
Polytech Grenoble, INFO3

Jean-François Méhaut

18 Mai 2020

Ce TP comprend deux parties : Une première partie sur les algorithmes de base sur les graphes (coloriage, parcours en profondeur, parcours en largeur, coloriage, Dijkstra). Une seconde partie sur des fonctions vérifiant certaines propriétés sur les graphes. Vous travaillerez sur des graphes orientés et pondérés.

1 Structure de données

Avant de commencer le TP, je vous conseille de bien analyser les structures de données utilisées pour représenter les graphes. Le graphe est constitué d'une liste chaînée de sommets et pour chaque sommet une liste chaînée d'arcs sortants.

Un graphe est représenté par une liste chaînée de sommets. Un sommet se définit par une structure `sommet_t` comprenant un label (nom du sommet) et la liste d'arcs sortants du sommet. La structure contient également un pointeur vers le sommet suivant dans le graphe. Le champ couleur a été ajouté à cette structure pour réaliser l'algorithme de coloriage de graphe. Cette structure est définie dans le fichiet `graphe.h`.

```
typedef struct s
{
  int         label ;              // label du sommet
  parc_t      liste_arcs ;         // arcs sortants du sommet
  struct s    *sommet_suivant ;    // sommet suivant dans le graphe
  int         couleur ;           // couleur du sommet
} sommet_t, *psommet_t ;
```

Un arc est représenté par une struture `arc_t`. `poids` indique le poids (ou longueur) de l'arc. `sommet` indique vers quel sommet l'arc est destinataire. Cette structure peut être complétée pour certains algorithmes, par exemple pour indiquer que l'arc a déjà été utilisé ou exploré.

```
typedef struct a
    {
      int poids ;                  // poids de l arc
      psommet_t   dest ;           // pointeur sommet  destinataire
      struct a *  arc_suivant ;    // arc suivant
    } arc_t, *parc_t ;
```

L'algorithme de coloriage de graphe est fourni dans le fichier `graphe.c` et utilise ces deux structures de données. Il vous permet de voir comment les structures `sommet_t` et `arc_t` sont utilisées.

Le fichier `io_graphe.c` propose une fonction de lecture et une fonction d'écriture de graphe.

2 Graphes : Algorithmes de base

L'algorithme du coloriage est fourni dans le fichier `graphe.c`. Je vous invite à regarder cet algorithme et à tester avec différents graphes. Plusieurs fichiers contenant des graphes se situent dans le répertoire `data`. Vous pouvez exécuter le programme `test_graphe` avec le fichier `gr_planning` et le graphe présenté dans la vidéo `youtube`. Un autre graphe à 10 sommets est fourni dans les fichiers `gr_sched1` et `gr_sched2`. L'exécution donne deux résultats différents. Pouvez-vous dire pourquoi ?

1. Implémentez l'algorithme du parcours en largeur d'un graphe `g` à partir d'un sommet `r`. Vous afficherez les labels des sommets du graphe dans l'ordre du parcours.

```
void afficher_graphe_largeur (pgraphe_t g, int r)
{
 /*
   afficher les sommets du graphe avec un parcours en largeur
   a partir du sommet dont le label est r
 */

  return ;
}
```

2. Implémentez l'algorithme du parcours en profondeur d'un graphe `g` à partir d'un sommet dont le label est `r`. Vous afficherez les labels des sommets du graphe dans l'ordre du parcours.

```
void afficher_graphe_profondeur (pgraphe_t g, int r)
{
  /*
    afficher les sommets du graphe avec un parcours en profondeur
    a partir du somment dont le label est r
  */

  return ;
}
```

3. Implémentez l'algorithme de Dijkstra d'un graphe `g` à partir d'un sommet dont le label est `r`. Vous définirez aussi une nouvelle fonction qui affiche le résultat de l'algorithme de Dijkstra.

```
int dijkstra (pgraphe_t g, int r)
{
  /*
     Algorithme de Dijkstra sur le graphe g
     sommet de depart est le sommet dont le label est r
  */
  return 0 ;
}
```

3 Propriétés sur les graphes

Nous supposerons que les fonctions présentées en cours ou développées pendant les séances de TP sont opérationnelles. Vous pouvez les réutiliser (appeler), si vous en avez besoin, pour répondre à certaines des questions. Un graphe est composé d'un ensemble de nœuds/sommets reliés par des arcs/arêtes. Le sujet de cet examen porte sur des graphes **pondérés** (poids sur les arcs) et **orientés** (flèches sur les arcs).

Quelques définitions caractérisant les chemins et les graphes :

1. Un chemin est une suite consécutive d'arcs dans un graphe.
2. Un chemin **élémentaire** est un chemin ne passant pas deux fois par un même sommet, c'est à dire un chemin dont tous les sommets sont distincts.
3. Un chemin **simple** est un chemin ne passant pas deux fois par le même arc, c'est à dire un chemin dont tous les arcs sont distincts.
4. Un chemin est dit **Eulérien** si tous les arcs du graphe sont utilisés 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 sommets 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 arcs.
9. La **distance** entre deux sommets x et y est la longueur du plus court chemin entre x et y.
10. L'**excentricité** d'un sommet est sa distance maximale avec les autres sommets du graphe.
11. Le **diamètre** d'un graphe est l'excentricité maximale de ses sommets.

3.1 Questions

Vous devez modifier et compléter les fichiers. Le fichier `graphe.h` contiendra les prototypes (signatures) des nouvelles fonctions. Le fichier `graphe.c` contiendra les implémentations de ces fonctions. Vous compléterez le fichier `test_graphe.c` pour tester les différentes fonctions.

4. Définissez le type `chemin_t` mémorisant les informations nécessaires pour un chemin. Le choix est important pour certaines des questions suivantes.

5. Décrivez en C l'implémentation de la fonction `elementaire` qui vérifie si un chemin est **élémentaire** ou pas. La fonction `elementaire` renvoie 1 si le chemin `c` est **élémentaire**, 0 sinon.

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

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

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

7. Décrivez en C l'implémentation de la fonction `eulerien` qui vérifie si un chemin est **Eulérien** ou pas. La fonction `eulerien` renvoie 1 si le chemin `c` est **Eulérien**, 0 sinon.

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

8. Décrivez en C l'implémentation de la fonction `hamiltonien` qui vérifie si un chemin est **Hamiltonien** ou pas. La fonction `hamiltonien` renvoie 1 si le chemin `c` est **Hamiltonien**, 0 sinon.

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

9. 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)
  {
    ...
  }
```

10. 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)
  {
    ...
  }
```

11. Décrivez en C l'implémentation de la fonction `distance` qui calcule la **distance** entre deux sommets avec les labels x et y dans le graphe `g`.

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

12. Décrivez en C l'implémentation de la fonction `excentricite` qui calcule pour un sommet de label `n` son **excentricité** dans le graphe `g`.

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

13. Décrivez en C l'implémentation de la fonction `diametre` qui calcule le **diamètre** du graphe `g`.

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