Travaux Pratiques n°6

Cours d'Informatique de Deuxième Année

—Licence MI L2.2—

Parcours de graphes

Ce TP constitue une introduction aux graphes. Nous nous proposons de réaliser un parcours en profondeur et en largeur d'un graphe non-orienté.

Rappels sur les graphes

Le graphe est un structure de données très utile (et utilisée) permettant de représenter des relations entre éléments. Un graphe est composé d'un ensemble S de sommets (*vertex* en anglais) mis en relation par un ensemble A d'arêtes ou arcs (*edge* en anglais), couples d'éléments de S.

Il existe des types particuliers de graphes. Parmi eux :

- Les graphes non-orientés où la directionnalité de l'arête n'intervient pas : ainsi si $(i, j) \in A$ alors $(j, i) \in A$.
- Les arbres qui sont des graphes particuliers avec définition d'un sommet racine : tout sommet de l'arbre ne peut alors être atteint que par un chemin unique depuis la racine.
- Les graphes acycliques (dont font partie les arbres) qui ne comportent aucun cycle (un graphe comporte un cycle lorsqu'il est possible de trouver un chemin arbitrairement long entre deux sommets de celui-ci).
- Les automates qui sont des graphes dont les arêtes sont étiquetées avec définition d'une liste d'états initiaux et d'états terminaux.
- Les graphes d'arêtes valuées où l'on associe à chaque arête une valeur.

Nous nous intéressons ici à la structure générale de graphe (non-orienté et non-acyclique).

Structures C utilisées Afin de représenter un graphe, nous utilisons la structure C suivante :

```
typedef char boolean;

typedef struct _Graphe
{
    unsigned int nbSommets;
        /* nombre de sommet du graphe */
    boolean * aretes;
        /* aretes[i*nbSommets + j] vaut 1 s'il y a une êarte de i vers j, 0 Sinon. */
} Graphe;
```

Nous définissons donc une structure *Graphe* contenant le nombre n de sommets du graphe (numérotés de 0 à $n-1$) ainsi qu'une matrice d'adjacence qui nous donne les les arêtes du graphe.

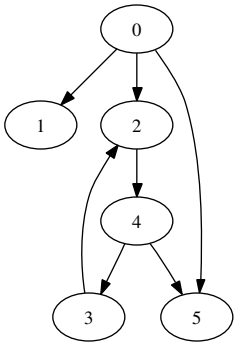

FIG. 1 – Exemple de graphe comportant un cycle (2, 4, 3)

▸ **Exercice 1.** ***Création du graphe par matrice d'adjacence*** *Écrire une fonction* Graphe_creation *prenant en paramètre le nombre de sommets du graphe et retournant un pointeur vers une structure* Graphe.
Écrire ensuite une fonction de signature `Graphe * Graphe_lecture(FILE *)` *chargeant en mémoire un graphe exprimé par un fichier au format ASCII suivant :*

$$\boxed{\begin{array}{l} n \\ a(0,0)\ a(0,1)\ \cdots\ a(0,n) \\ a(1,0)\ a(1,1)\ \cdots\ a(1,n) \\ \cdots \\ a(n,0)\ a(n,1)\ \cdots\ a(n,n) \end{array}}$$

où n est le nombre de sommets du graphe et $a(i,j) = 1$ ssi le graphe contient une arête (i,j) (sinon $a(i,j) = 0$).

▸ **Exercice 2.** ***Destruction du graphe par listes de successeurs*** *Écrire une fonction de signature* `void Graphe_destruction(Graphe *)` *libérant la mémoire occupée par le graphe passé en paramètre.*

Parcours en profondeur

On souhaite maintenant parcourir en profondeur un graphe en affichant à chaque sommet atteint son identificateur. Tout comme pour le parcours d'arbre (qui n'est d'autre qu'un graphe avec certaines particularités), il est nécessaire de maintenir une pile. Il est possible de s'affranchir de la gestion manuelle de la pile en utilisant des appels récursifs de fonction : il s'agit de l'approche que nous allons envisager.

Il est nécessaire pour mener le parcours en profondeur d'un graphe de considérer deux points essentiels :

- Contrairement à un arbre, un graphe, dans le cas général, peut posséder plusieurs chemins de la racine utilisée pour le parcours jusqu'à un autre sommet.
- Un graphe peut posséder des cycles, i.e. un chemin (de longueur non-nulle) d'un sommet vers lui-même. L'algorithme de parcours doit se terminer : un éventuel cycle ne doit être parcouru qu'une unique fois.

Nous avons donc besoin de savoir, lors du parcours, si un sommet n'a pas déjà été atteint par un autre chemin. Deux approches peuvent être envisagées :

- Visiter chaque sommet une unique fois lors de l'exploration, même si celui-ci est accessible par plusieurs chemins.
- Visiter le sommet pour chaque chemin différent (sans cycle) depuis la racine.

Nous présentons ici un exemple de parcours en profondeur du graphe en figure 1 à partir du sommet 0 (une seule visite par sommet) :

```
0
  1
  2  4
        3
        5
```

Dans le cadre de l'exemple précédent, à partir du sommet 0 utilisé pour démarrer l'exploration, il existe des chemins vers tous les sommets du graphes. Ce n'est pas toujours le cas : si l'on souhaite parcourir tous les sommets du graphe, il est alors nécessaire de recommencer le parcours en profondeur en choisissant pour sommet de démarrage un sommet non encore exploré. On peut ainsi répéter l'opération d'exploration jusqu'à ce que tous les sommets soient visités.

▸ **Exercice 3.** ***Parcours en profondeur*** *Rédiger une fonction récursive réalisant le parcours en profondeur du graphe à partir d'un sommet de démarrage en affichant l'identificateur de chaque sommet. On utilisera un tableau de booléens pour savoir si un sommet n'a pas déjà été atteint. Faire ensuite une fonction réalisant un parcours en profondeur du graphe en débutant du sommet* 0*, puis renouvellant l'exploration si nécessaire sur les sommets non visités.*

Parcours en largeur

Le parcours en largeur d'un graphe consiste à partir d'un sommet de démarrage puis à parcourir ses sommets adjacents, puis les sommets adjacents de ses sommets adjacents, ... On parcourt donc dans un premier temps tous les sommets accessibles à une distance 1 de la racine, puis à une distance 2, ... Ce parcours nécessite l'utilisation d'une file de type FIFO (*first in, first out*) : on y ajoute les sommets adjacents non-visités du sommet que l'on vient d'y retirer pour exploration. Là encore, il est nécessaire de connaître si un sommet a déjà été atteint lors du parcours pour éviter le parcours de multiples fois du même sommet ainsi que le bouclage dans des cycles.

Voici un exemple de parcours en largeur à partir du sommet racine 0 du graphe présenté en figure 1 :

```
0
1 2 5
4
3
```

La même remarque que pour l'exploration en profondeur s'applique : il peut être nécessaire de ré-appliquer l'exploration depuis les sommets non explorés.

▸ **Exercice 4.** ***Parcours en largeur*** *Écrire une fonction réalisant le parcours en largeur d'un graphe à partir d'un sommet spécifié puis une autre fonction permettant de parcourir l'ensemble du graphe en largeur.*

▸ **Exercice 5.** ***Puissance d'une matrice représentatrice de graphe (facultatif)*** *Soit M la matrice représentatrice du graphe G. $M_{ij} = 1$ ssi il existe une arête (i, j) (sinon $M_{ij} = 0$). Que représente M^n (pour $0 < n \leq |S| - 1$) ? Et $M^{|S|-1}$? On définit ainsi la multiplication matricielle :*

$$(M \cdot N)_{ij} = \min\left(1, \sum_{k=1}^{|S|} M_{ik} M_{kj}\right)$$

. Écrire une fonction permettant de calculer M^n en complexité temporelle $O(\log(n)|S|^3)$.

Petite citation à méditer...

```
Fait divers
Son lave glace devient fou. Il meurt noyé dans sa voiture.
Les nuls.
```