TP n 15 - Graphes : représentation et parcours

Notions abordées

- Représentation sous forme de matrice d'adjacence
- Représentation sous forme de table de listes d'adjacence
- Révision des tableaux, tableaux de tableaux, listes chaînées et pointeurs en C
- Détection de graphe biparti par un algorithme de parcours
- Tri topologique et détection de circuits dans un graphe orienté

Exercice 1 Représentation par matrice d'adjacence

Une matrice de booléens symétrique de dimensions $n \times n$ M représente le graphe non orienté $G = (V, E)$ défini par $V = [1..n]$ et $E = \{\{i, j\} \mid M_{i,j} = \text{vrai}\}$. Dans tout cet exercice on dit "un graphe" pour signifier "un graphe non orienté représenté par une matrice d'adjacence".

Question 1
Dessiner ci-contre le graphe représenté par la matrice d'adjacence ci-dessous.

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Question 2
Quelles conditions doit vérifier une matrice de booléens pour représenter un graphe non orienté sans boucles ?

A Structure de graphe par matrice d'adjacence en C

Afin de représenter des graphes par leur matrice d'adjacence en C, on définit la structure suivante, qui est disponible dans le fichier `graphe_mat_adj_etu.c` sur cahier de prépa

```c
struct s_graphe_mat {
  int n; //nombre de sommets
  bool** mat; // tableau de tableau de booléens de dimensions nxn
};

typedef struct s_graphe_mat gra_m;
```

Dans la suite on supposera que de telles structures sont toujours convenablement remplies, c'est-à-dire que le champ `mat` est bien un tableau de dimension $n \times n$ où n est la valeur enregistrée dans le champ `n` de la structure. On veillera donc à préserver cet invariant.

Question 3
Définir une fonction `cree_graphe_vide` qui prend en argument un nombre n de sommet et crée une nouvelle structure de graphe par matrice d'adjacence contenant un graphe à n sommet mais aucune arrête. La fonction renvoie le pointeur vers cette nouvelle structure.

Question 4
Définir une fonction `supprime_graphe` qui prend en argument l'adresse d'une structure de graphe codé par matrice d'adjacence et qui libère l'espace mémoire qu'elle occupe.

Question 5
Grâce à la fonction `affiche_graphe` fournie dans le fichier, tester la la fonction `cree_graphe_vide`.

Question 6
Définir une fonction `ajoute_arete` qui prend en argument l'adresse d'une structure de graphe codé par matrice d'adjacence et deux entiers représentant des sommets de ce graphe, et qui ajoute, si besoin est, l'arête reliant ces deux sommets au graphe.

Question 7
Grâce à la fonction `ajoute_arete_from_tab` fournie dans le fichier, créer plusieurs graphes ayant de nombreuses arêtes en vue des tests des prochaines fonctions. Vérifier si besoin le résultat grâce à la fonction d'affichage.

B Fonctions usuelles de manipulation de graphe

Question 8
Définir une fonction `nb_sommets` qui donne le nombre de sommets d'un graphe.
Quelle est sa complexité ?

Question 9
Définir une fonction `sont_voisins` qui teste si deux sommets sont voisins dans un graphe.
Quelle est sa complexité ?

Question 10
Définir une fonction `nb_aretes` qui donne le nombre d'arêtes d'un graphe.
Quelle est sa complexité ?

Question 11
Définir une fonction `nb_voisins` qui donne le degré d'un sommet dans un graphe.
Quelle est sa complexité ?

Question 12
Donner une relation entre le nombre d'arêtes et les degrés des sommets d'un graphe non orienté. Utiliser cette relation pour tester les fonctions précédentes.

Question 13
Définir une fonction `graphe_induit` qui étant donné un graphe $G = (V, E)$ et un ensemble $W \subseteq V$ calcule le **graphe induit** par G sur W défini comme $(W, \{\{u, v\} \in E \mid u \in W \text{ et } v \in W\})$.

C Variations autour de cette représentation

Question 14
Dans un contexte où l'espace mémoire serait une ressource critique, pourrait-on réduire l'espace mémoire utilisé pour stocker un graphe sans perte d'information, et sans changer la complexité pire cas des opérations susmentionnées ?

Question 15
Comment adapter cette façon de représenter les graphes aux graphes orientés ? Qu'est ce qu'il faudrait changer ? Quelles complexités d'opérations usuelles seraient remises en cause ?

Exercice 2 Représentation par listes d'adjacences

Dans cet exercice on s'intéresse à la représentation de graphes non orientés par listes d'adjacence, c'est-à-dire par un tableau de listes de voisins (listes sans doublons représentant des ensembles). En effet, un tableau T de n listes d'entiers de $[1..n]$ représente le graphe $G=(V,E)$ défini par $V=[1..n]$ et $E=\{\{i,j\}\,|\,i \text{ est présent dans } T_j\}$.[1] Dans tout cet exercice on dit "un graphe" pour signifier "un graphe non orienté représenté par une table de listes d'adjacence". Afin de ne pas créer de conflit avec les fonctions de l'exercice précédent, coder les fonctions de cet exercice dans un nouveau fichier.

Question 1
Donner la table de listes d'adjacence du graphe non orienté de la question 1 de l'exercice 1.

Question 2
Quelles conditions doit vérifier un tableau de listes d'entiers pour représenter un graphe non orienté.

A Structure de graphe par listes d'adjacence en C

Afin de représenter des graphes par leurs listes d'adjacence en C,on définit la structure suivante, qui est disponible dans le fichier `graphe_listes_adj_etu.c` sur cahier de prépa

```c
struct s_graphe_listes {
  int n; //nombre de sommets
  liste_c* tab; // tableau de tableau de n listes chaînées d'entiers
};

typedef struct s_graphe_listes gra_l;
```

Dans la suite on supposera que de telles structures sont toujours convenablement remplies, c'est-à-dire que le champ `tab` est bien un tableau de n listes chaînées contenant des entiers entre 0 et $n-1$, où n est la valeur enregistrée dans le champ `n` de la structure. On veillera donc à préserver cet invariant.

Afin de ne pas avoir à recoder toutes les fonctions permettant la manipulation de listes chaînées, on réutilisera le code réalisé lors du TP sur les listes chaînées, ou bien directement les fichiers `liste_ch.c` et `liste_ch.h` fournis sur cahier de prépa. Avant de continuer, il faut se remémorer le fonctionnement des listes chaînées, prendre connaissances des fonctions disponibles et des hypothèses de ces fonctions (*nombre d'entre elles ne s'appliquent qu'à une liste chaînée non vide, i.e. différente de* `NULL`).

[1] cette définition est correcte sous réserve des conditions sur T que l'on demande d'expliciter à la question 2.

Question 3
Définir une fonction `cree_graphe_vide` qui prend en argument un nombre n de sommet et crée une nouvelle structure de graphe par liste d'adjacence contenant un graphe à n sommet mais aucune arrête. La fonction renvoie le pointeur vers cette nouvelle structure.

Question 4
Définir une fonction `supprime_graphe` qui prend en argument l'adresse d'une structure de graphe codé par matrice d'adjacence et qui libère l'espace mémoire qu'elle occupe.

Question 5
Grâce à la fonction `affiche_graphe` fournie dans le fichier, tester la la fonction `cree_graphe_vide`.

Question 6
Définir une fonction `ajoute_arete` qui prend en argument l'adresse d'une structure de graphe codé par matrice d'adjacence et deux entiers représentant des sommets de ce graphe, et qui ajoute, si besoin est, l'arête reliant ces deux sommets au graphe.

Question 7
Grâce à la fonction `ajoute_arete_from_tab` fournie dans le fichier, créer plusieurs graphes ayant de nombreuses arêtes en vue des tests des prochaines fonctions. Vérifier si besoin le résultat grâce à la fonction d'affichage.

B Fonctions usuelles de manipulation de graphe

Question 8
Définir une fonction `nb_sommets` qui donne le nombre de sommets d'un graphe.
Quelle est sa complexité ?

Question 9
Définir une fonction `sont_voisins` qui teste si deux sommets sont voisins dans un graphe.
Quelle est sa complexité ? Peut-on réduire cette complexité en remarquant que le fait d'être voisin définit une relation symétrique ?

Question 10
Définir une fonction `nb_arretes` qui donne le nombre d'arêtes d'un graphe.
Quelle est sa complexité ?

Question 11
Définir une fonction `nb_voisins` qui donne le degré d'un sommet dans un graphe.
Quelle est sa complexité ?

Question 12
Définir une fonction `graphe_induit` qui étant donné un graphe $G = (V, E)$ et un ensemble $W \subseteq E$ calcule le graphe induit par G sur W.

C Variations autour de cette représentation

Question 13
Comment adapter cette façon de représenter les graphes aux graphes orientés ? Qu'est ce qu'il faudrait changer et/ou dupliquer? Que deviendraient les complexités des opérations usuelles.

Exercice 3 Détection de graphe biparti

On rappelle qu'un graphe non orienté $G=(V,E)$ est dit **biparti** s'il existe $\{V_1, V_2\}$ une partition de V telle que les arêtes de G ont toutes une extrémité dans V_1 et l'autre dans V_2.

Question 1

Écrire le pseudo code d'un algorithme de parcours qui teste si un graphe est biparti en essayant de le bi-colorier de manière gloutonne.

Question 2

Dans le pseudo-code proposé, identifier quelles sont les opérations coûteuses, et en déduire :

- → quelle représentation de graphe est la plus adaptée pour cet algorithme (c'est-à-dire celle pour laquelle l'algorithme sera de plus faible complexité) ;
- → quelle structure de données est utilisée pour représenter les sommets visités/ouverts/fermés.

Question 3

Définir en C une fonction `est_biparti` qui teste si un graphe est biparti.