TP Graphes 2

Licence Informatique - 2nde année

Année 2025-2026

L'objectif de ce TP est de reprendre les fonctionnalités développées au TP précédent et de les implanter pour une représentation **compacte** des matrices d'adjacences.

Préliminaire

Récupérez l'archive qui est fournie avec cet énoncé et installez les fichiers qui s'y trouvent dans le dossier où vous effectuerez ce TP (il est conseillé de créer un sous dossier `TP2` dans votre dossier `Graphes` pour ce TP). Elle contient le fichier `tp2.cpp` qui correspond au squelette de l'application que vous allez développer durant ce TP, ainsi qu'un sous dossier `Data`, dans lequel figurent quelques fichiers représentant des matrices d'adjacence. Ces dernières seront représentées de manière compacte par les structures suivantes (voir cours) :

```
struct Maillon {
  int col; // numéro de la colonne à laquelle correspond le coefficient
  int coef;// coefficient de la matrice
  Maillon *suiv;// élément suivant non nul sur la ligne
} ;

struct MatriceAdjacence {
  int ordre; // nombre de sommets du graphe
  Maillon* *lignes;// tableau à allouer de taille "ordre", représentant les lignes de la matrice
};
```

Exercice 1

Compléter la fonction :

`void creerMatrice(MatriceAdjacence *mat, int size)`

qui permet de créer une matrice d'adjacence de taille `size` et de l'initialiser comme étant vide. En particulier, la fonction devra allouer l'espace mémoire nécessaire à la représentation des `size` têtes de lignes, chacune d'entre-elles étant initialisée à `nullptr`. La figure 1 illustre l'état de la variable `mat` et de la mémoire occupée à l'issue de la création d'une matrice de taille 4.

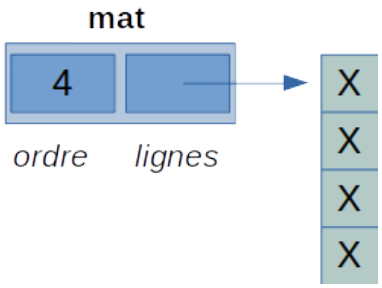

FIGURE 1 – Représentation de la variable `mat` et de la mémoire occupée à l'issue d'un appel de `creerMatrice(&mat, 4)`.

Exercice 2

Complétez la fonction de chargement du fichier, de prototype :

```
bool chargerMatrice(char *nomFichier, MatriceAdjacence *mat)
```

avec :

— `nomFichier` : le nom du fichier à charger ;

— `mat` : un pointeur vers la matrice d'adjacence à créer, qui devra contenir, en sortie de fonction, les coefficients contenus dans le fichier **s'ils sont différents de** 0 (voir figure 2).

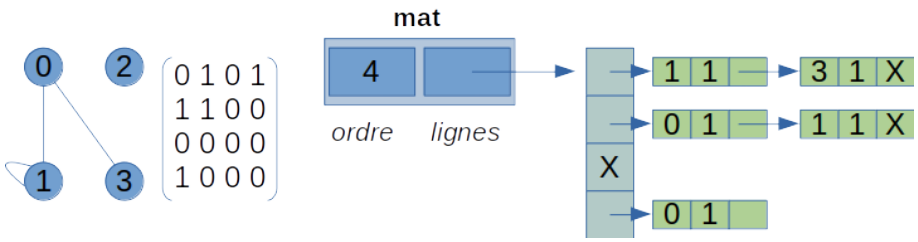

FIGURE 2 – Exemple de représentation de la variable `mat` en mémoire (partie droite) après chargement de la matrice apparaissant en partie gauche.

Exercice 3

Compléter la fonction :

```
void afficherMatrice(MatriceAdjacence mat)
```

qui permet d'afficher le contenu de la matrice. Les différentes lignes de la matrice devront apparaître les unes au dessous des autres au moment de l'affichage. L'affichage devra afficher 0 si la colonne n'est pas présente, la valeur du coefficient sinon (un exemple vous est donné sur la figure 3). Testez votre application avec les différentes matrices fournies.

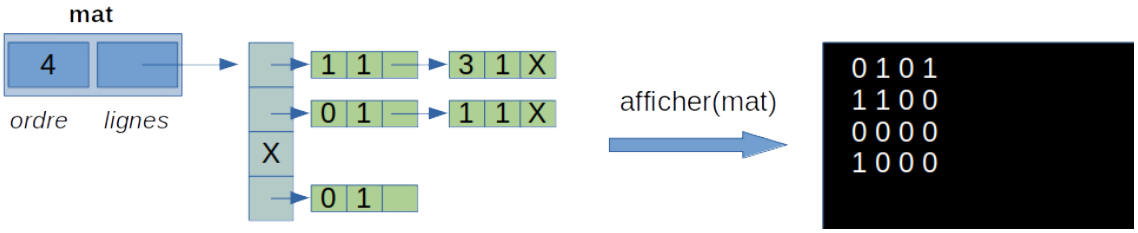

FIGURE 3 – Exemple d'affichage à obtenir en fonction des coefficients présents dans la représentation mémoire de la matrice.

Exercice 4

Compléter la fonction :

```
effacerMatrice(MatriceAdjacence *mat)
```

qui permet d'effacer toute la mémoire utilisée par une matrice, à savoir les listes chaînées représentant chaque ligne et le tableau représentant les têtes de listes chaînées.

Exercice 5

Ajouter la fonction suivante à votre application :

```
int getCoeff(MatriceAdjacence mat, int l, int c)
```

qui permet de renvoyer la valeur du coefficient de la matrice stockée aux indices (l, c), avec l l'indice de ligne ($\in [0, ordre - 1]$) et c l'indice de colonne ($\in [0, ordre - 1]$). On rappelle que, comme tous les coefficients ne sont pas présents sur chaque ligne, il faudra rechercher si l'indice de colonne c est présent dans le liste chaînée de la ligne l. Si c'est le cas, son coefficient est renvoyé, sinon la fonction renverra 0.

On supposera que les indices de ligne et de colonne sont valides, sans qu'il soit nécessaire de les tester.

Cette fonction devra être utilisée pour résoudre les questions suivantes.

Exercice 6

Complétez la fonction suivante à votre application :

```
bool estComplet(MatriceAdjacence mat)
```

dont le rôle est de vérifier si la matrice d'adjacence passée en paramètre représente ou pas un graphe complet. Vous testerez son fonctionnement sur les fichiers se trouvant dans le dossier `Data`.

Exercice 7

Complétez la fonction suivante à votre application :

```
bool estSymetrique(MatriceAdjacence mat)
```

dont le rôle est de vérifier si la matrice passée en paramètre est symétrique ou pas. Vous testerez son fonctionnement sur les fichiers se trouvant dans le dossier `Data`.

Exercice 8

Ajoutez la fonction suivante à votre application :

```
bool egales(Maillon *liste1, Maillon *liste2)
```

dont le rôle est de déterminer si les deux listes chaînées passées en paramètres sont égales (valeur de retour `true`) ou pas (valeur de retour `false`).

Exercice 9

En utilisant la fonction de la question précédente, compléter la fonction suivante de votre application :

```
bool egales(MatriceAdjacence m1, MatriceAdjacence m2)
```

dont le rôle est de déterminer si les deux matrices d'adjacence passées en paramètres sont égales (valeur de retour `true`) ou pas (valeur de retour `false`).

Exercice 10

Ajoutez et testez la fonction suivante au code déjà développé :

```
void copie(Maillon *liste1, Maillon *liste2)
```

qui recopie le contenu de la liste `l1` dans la liste l2 `m2`. Si cette dernière dispose déjà de données, celles-ci-devront être supprimées avant recopie.

Exercice 11

En utilisant la fonction de la question précédente, compléter la fonction suivante de votre application :

```
void copie(MatriceAdjacence m1, MatriceAdjacence *m2)
```

qui recopie le contenu de la matrice `m1` dans la matrice `m2`. Si cette dernière dispose déjà de données, celles-ci-devront être supprimées avant recopie.

Exercices optionnels

Reprenez les exercices 6 à 9 du TP précédent.