TP Graphes 1

Licence Informatique - 2nde année

Année 2025-2026

L'objectif de ce TP est de mettre en place quelques routines permettant de charger une matrice d'adjacence depuis un fichier, puis d'effectuer quelques tests sur celle-ci.

Exercice 1

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, dans votre espace personnel, un dossier `Graphes` et d'y créer un sous dossier TP1 pour ce TP). Elle contient le fichier `tp1.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 par la structure suivante (voir cours) :

```
struct MatriceAdjacence {
  int *coeff; // tableau contenant les (ordre x ordre) coefficients de la matrice
  int ordre; // nombre de sommets du graphe
}
```

On rappelle qu'une matrice d'adjacence est carrée. Le champ `ordre` de la structure corrrespond au nombre de sommets du graphe, et donc au nombre de colonnes et de lignes de la matrice. Les coefficients de la matrice sont stockés dans le tabelau `coef`, une ligne après l'autre (voir la figure du cours).

1. Ecrire la fonction :
   `void creerMatrice(MatriceAdjacence *m, int size)`
   qui permet de créer une matrice d'adjacence de taille `size` et de l'initialiser avec des 0. En particulier, la fonction devra allouer l'espace mémoire nécessaire à la représentation des `size x size` coefficients dans le champs `coef`.
2. Ecrire 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.
3. Tester votre application, qui devra porter le nom `tp1`, avec différentes valeurs de taille.

Exercice 2

Dans la suite de cet énoncé, les matrices seront relues dans des fichiers, dont quelques exemples vous sont fournis dans le dossier `Data`. Afin de pouvoir choisir quel fichier utiliser, il sera nécessaire de passer le nom du fichier sur la ligne de commandes, à la suite du nom de l'application.

Exemple : `./tp1 Data/matrice01.txt`

Modifiez la fonction `main` de telle sorte qu'elle vérifie qu'il y a bien un nom de paramètre passé sur la ligne de commandes. Si ce n'est pas le cas, ou si plusieurs paramètres sont tapés par l'utilisateur, l'application affichera un message d'erreur et s'arrêtera.

Exercice 3

Vous allez à présent ajouter la fonction de chargement du fichier à votre application. Cette fonction aura le prototype suivant :

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

avec :

— `nomFichier` : le nom du fichier à charger ;
— `mat` : un pointeur vers la matrice d'adjacence à créer et à remplir avec les coefficients contenus dans le fichier.

Pour développer cette fonction, vous allez procéder en suivant les différentes étapes énumérées ci-après, en testant votre application à l'issue de chacune d'entre-elles.

1. Ouvrir le fichier `nomFichier` et vérifier qu'il a bien été ouvert. Si ce n'est pas le cas, la fonction affichera un message d'erreur et renverra la valeur `false`. Sinon elle refermera le fichier et elle renverra la valeur true. Complétez la fonction `main` pour qu'elle appelle cette fonction de chargement et s'arrête si cette dernière retourne `false`. Vous testerez cette première version avec des noms de fichiers existants (ceux se trouvant dans le dossier `Data`) et avec des noms de fichiers inexistants.
2. Compléter la fonction de telle manière qu'elle relise la première valeur entière qui y est stockée si le fichier a bien été ouvert. Cette valeur servira à initialiser la dimension de la matrice.
3. Ajoutez le code nécessaire pour créer la zone mémoire permettant de stocker tous les coefficients, puis lire les différents coefficients qui se trouvent dans le fichier pour les ranger à leur place dans le tableau de coefficients de la matrice. On précise que comme pour la matrice, les coefficients contenus dans le fichier sont des entiers, rangés les uns derrière les autres, une ligne après l'autre.
4. Complétez la fonction `main` pour qu'elle affiche le contenu de la matrice si celle-ci a bien été chargée (vous pourrez supprimer ou commenter la partie de code correspondant à la question 1 de la fonction `main`).

Exemples :

```
./tp1 Data/toto.txt
Erreur d'ouverture de Data/toto.txt
```

```
./tp1 Data/matrice01.txt
0 1 1 0 1
1 0 1 1 0
1 1 0 1 0
0 1 1 0 0
1 0 0 0 0
```

Exercice 4

Ajouter 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`. La figure 1 vous donne quelques exemples de graphes complets ou incomplets et leur matrice d'adjacence.

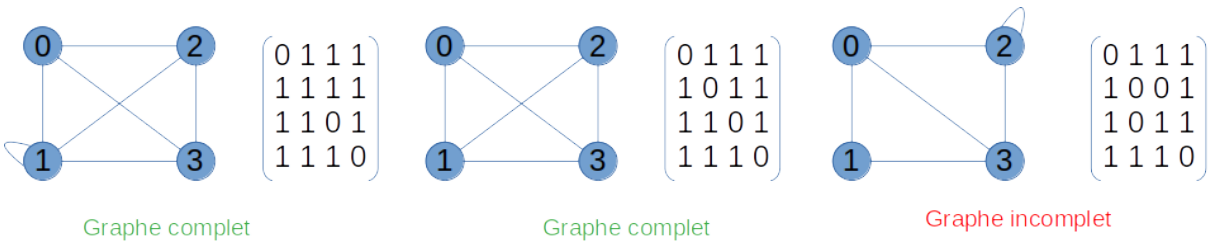

FIGURE 1 – Exemples de graphes complets et incomplets, avec leur matrice d'adjacence

Résultats à obtenir :

— `matrice01.txt` : incomplète
— `matrice02.txt` : incomplète
— `matrice03.txt` : complète

— `matrice04.txt` : incomplète

Exercice 5

Ajouter 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`.

Résultats à obtenir :

— `matrice01.txt` : symétrique
— `matrice02.txt` : non symétrique
— `matrice03.txt` : symétrique
— `matrice04.txt` : symétrique

Exercice 6

Ajouter la fonction suivante à votre application :

`bool estIsole(MatriceAdjacence mat, int i)`

qui déterminera si le sommet i ($i \in [0, ordre - 1]$) est isolé ou pas. Vous utiliserez cette fonction pour afficher tous les noeuds isolés d'une matrice d'adjacence pour les fichiers se trouvant dans le dossier `Data`. La figure 2 vous donne quelques exemples de graphes possédant ou pas des sommets isolés, accompagnés de leur matrice d'adjacence. On précise que le résultat renvoyé par votre fonction doit être valide, que le graphe soit orienté ou pas.

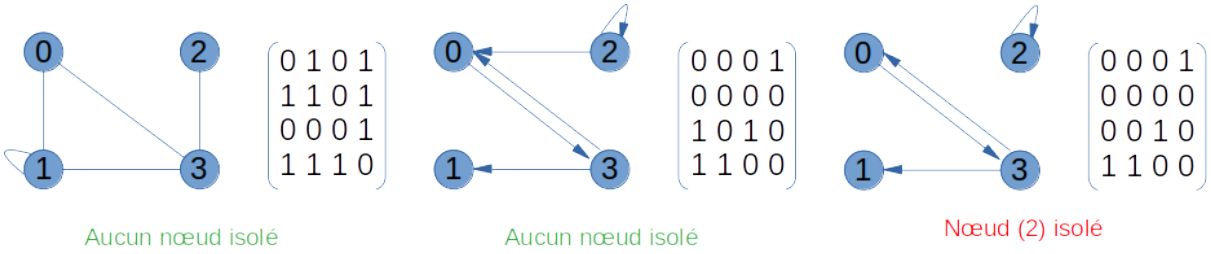

FIGURE 2 – Exemples de graphes possédant ou pas des sommets isolés, avec leur matrice d'adjacence.

— `matrice01.txt` : pas de sommets isolés
— `matrice02.txt` : pas de sommets isolés
— `matrice03.txt` : pas de sommets isolés
— `matrice04.txt` : sommets 2 et 6 isolés

Exercice 7

Ajouter la fonction suivante à votre application :

`void afficherDegresNonOriente(MatriceAdjacence mat)`

qui affichera le degré de chacun des sommets de la matrice d'adjacence passée en paramètre, en supposant que celle-ci n'est pas orientée. Vous testerez son fonctionnement sur les fichiers se trouvant dans le dossier `Data/NonOrientes`.

Résultats à obtenir :

```
./tp1 Data/NonOrientés/matrice05.txt
degré du sommet 0 : 3
degré du sommet 1 : 4
degré du sommet 2 : 1
degré du sommet 3 : 3
degré du sommet 4 : 3
degré du sommet 5 : 3
degré du sommet 6 : 1
```

```
./tp1 Data/NonOrientés/matrice06.txt
degré du sommet 0 : 4
degré du sommet 1 : 1
degré du sommet 2 : 1
degré du sommet 3 : 1
degré du sommet 4 : 1
```

Exercice 8

Ajouter la fonction suivante à votre application :

`void afficherDegresOriente(MatriceAdjacence mat)`

qui affichera le degré de chacun des sommets de la matrice d'adjacence passée en paramètre, en supposant que celle-ci est orientée. Vous testerez son fonctionnement sur les fichiers se trouvant dans le dossier `Data/Orientes`.

Résultats à obtenir :

```
./tp1 Data/Orientés/matrice07.txt
degré du sommet 0 : 4
degré du sommet 1 : 4
degré du sommet 2 : 2
degré du sommet 3 : 2
degré du sommet 4 : 2
```

```
./tp1 Data/Orientés/matrice08.txt
degré du sommet 0 : 5
degré du sommet 1 : 4
degré du sommet 2 : 3
```