Exercice 005 *Composantes fortement connexes : méthode matricielle*

Le code fourni pour cet exercice se trouve dans le fichier fourni `cfc_matricielle.c`. *Il s'agit de types pour représenter des graphes par matrices d'adjacence, et de fonctions basiques de manipulation de graphe.*

On s'intéresse d'abord à des *matrices booléennes*. Dans notre cas, il s'agira de matrices carrées de taille $n \times n$, dont les coefficients sont des booléens, autrement dit des valeurs d'un ensemble $\{V, F\}$. Dans l'implémentation, une matrice carrée booléenne de taille $n \times n$ sera considérée comme un tableau à deux dimensions contenant des éléments de type `bool` ; on accède à la case d'indices `(i,j)` dans la matrice `m` par `m[i][j]`.

On définit la somme et le produit de matrices booléennes comme pour des matrices sur un corps K, mais où les opérateurs booléens OU et ET jouent respectivement le rôle de l'addition et de la multiplication. Ainsi, si $A = (a_{ij})_{i,j\in[\![0;n-1]\!]}$ et $B = (b_{ij})_{i,j\in[\![0;n-1]\!]}$ sont deux matrices de taille $n \times n$, on définit la somme et le produit de A et de B comme suit :

- la somme, notée $A + B$, est une matrice $(c_{ij})_{i,j\in[\![0;n-1]\!]}$ telle que, pour tous $i, j \in [\![0; n-1]\!]$, $c_{ij} = a_{ij} \vee b_{ij}$.
- le produit, noté AB, est une matrice $(c_{ij})_{i,j\in[\![0;n-1]\!]}$ telle que, pour tous $i, j \in [\![0; n-1]\!]$, $c_{ij} = \bigvee_{k\in[\![0;n-1]\!]} (a_{ik} \wedge b_{kj})$.

1. A titre d'exemple, calculer :
$$\begin{pmatrix} F & V \\ V & F \end{pmatrix} \begin{pmatrix} V & F \\ V & F \end{pmatrix} + \begin{pmatrix} F & F \\ F & V \end{pmatrix}$$
2. Donner sans justification la matrice N jouant le rôle de neutre pour l'addition, et la matrice I jouant le rôle de neutre pour le produit.
3. Ecrire une fonction `liberer_matrice(bool** m, int n)`, qui libère toute la mémoire utilisée par une matrice `m` de taille $n \times n$.
4. Ecrire une fonction `bool** nulle(unsigned int n)`, qui renvoie la matrice N de taille $n \times n$, nouvellement allouée.
5. Ecrire une fonction `bool** identite(unsigned int n)`, qui renvoie la matrice I de taille $n \times n$, nouvellement allouée.
6. Ecrire une fonction `void somme(unsigned int n, bool** m1, bool** m2, bool** m3)` qui écrit dans la matrice `m3` (qu'on suppose déjà allouée) la somme `m1 + m2`, où 3 matrices sont de taille `n` × `n`.
7. Ecrire une fonction `void produit(unsigned int n, bool** m1, bool** m2, bool** m3)` qui écrit dans la matrice `m3` (qu'on suppose déjà allouée) le produit `m1m2`, où ces 3 matrices sont de taille `n` × `n`.
8. Quelle est la complexité des fonctions précédentes ?

Dans la suite de l'exercice, nous allons maintenant utiliser les fonctions écrites précédemment sur les matrices de booléens pour proposer une méthode de calcul de composantes fortement connexes sur des graphes orientés, implémentés par matrice d'adjacence.

Si $G = (S, A)$ est un graphe orienté, on supposera sans pertes de généralités que S est un ensemble de la forme $S = [\![0; n-1]\!]$. Pour un tel graphe, sa *matrice d'adjacence* $B = (b_{ij})_{i,j\in S}$ est définie par : pour tous $i, j \in S$, b_{ij} vaut V si $(i, j) \in A$ et F sinon. C'est exactement cette matrice que contient le champ `adj` dans le type `graphe` dans l'implémentation.

On fournit un graphe G_0 représenté sur la figure 1, qu'on pourra soi-même implémenter en se basant sur les fonctions fournies dans `cfc_matricielle.` pour effectuer des tests.

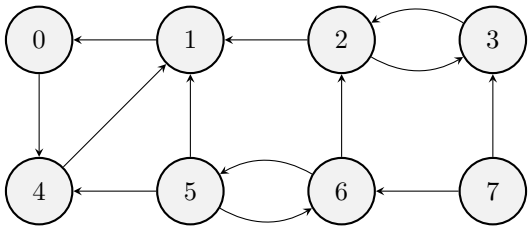

Figure 1: Un graphe orienté G_0

9. Si B est la matrice d'adjacence d'un graphe orienté $G = (S, A)$ sommets, on note $B^p = (m_{ij})_{i,j \in S}$ la matrice obtenue en calculant le produit $BB \dots B$ comportant p fois la matrice B. Montrer que :

   Pour tout i, j, $m_{ij} = V$ si et seulement s'il existe un chemin d'exactement p arêtes de i à j.

10. On pose $B_p = \sum_{k=0}^{p} B^k$. On note $B_p = (q_{ij})_{i,j \in S}$. Montrer que :

    Pour tout i, j, $q_{ij} = V$ si et seulement s'il existe un chemin d'au plus p arêtes de i à j.

11. En exhibant un variant judicieux, montrer que la suite $(B_p)_{p \in \mathbb{N}}$ est stationnaire, et atteint une limite qu'on notera B_*. Que représente B_* ? Au delà de quelle valeur de p est on certain que B_* est atteint ?

Dans la suite, si on considère un graphe G de matrice d'adjacence B, la matrice B_* sera appelée *matrice d'accessibilité de G*.

12. Ecrire une fonction `bool** chemins(const graphe g)` qui renvoie la matrice d'accessibilité du graphe `g`. *Indication : on ne calculera pas de puissance de matrice pour ne pas perdre en efficacité ; on peut donc d'abord montrer que la suite $(B_p)_{p \in \mathbb{N}}$ vérifie une relation de récurrence.*
13. Donner la complexité de la fonction précédente.

Après avoir calculé la matrice d'accessibilité, on en déduit immédiatement une méthode assez naturelle pour calculer les composantes fortement connexes du graphe.

14. Ecrire une fonction `unsigned int* cfc (const graphe g)`, qui renvoie un tableau `tab` de taille $|S|$ de sorte que, pour tout `i`, la case `tab[i]` contient le numéro de la composante connexe où se trouve `i`. On les numérotera dans l'ordre où elles sont rencontrées, en partant de `0`.

15. Quelle est la complexité de cette méthode ? Est-elle plus intéressante à utiliser que l'algorithme de Kosaraju ?