TP 17 : Manipulation de graphes en C

Dans ce TP, on se limite à la manipulation de graphes représentés par des matrices d'adjacence en C (contrairement au DS qui manipulait des listes d'adjacence avec un indicateur du nombre de voisins). Ce TP est adapté d'un problème de concours, reformulé en C et comme un problème sur les graphes.
On se place dans le cadre des graphes orientés avec des bouclettes. Soit $\mathcal{G} = (S, A)$ un tel graphe avec $n = |S|$ sommets, dont on fixe une énumération $x_0, \dots, x_{n-1}$ (on confondra donc S et $[\![0, n-1]\!]$). De plus, **on suppose que toutes les bouclettes** (x, x) **sont dans le graphe.**
Dans la suite, on note $x \Rightarrow_k y$ s'il existe un chemin de taille k allant de x à y. Notamment, $x \Rightarrow_0 y$ ssi $x = y$. On notera $x \Rightarrow y$ s'il existe $k \in \mathbb{N}$ tel que $x \Rightarrow_k y$, c'est-à-dire si y est accessible depuis x.
Dans la suite, on définira les deux sommets suivants :

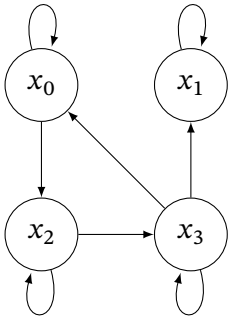

FIGURE 1 – Graphe $\mathcal{G}_1$

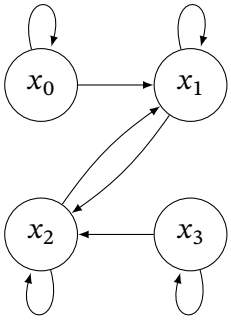

FIGURE 2 – Graphe $\mathcal{G}_2$

Exercice 1 – Cadre de travail, fonctions utilitaires

▶ **Question 1** Soient $x, y \in S$ deux sommets d'un graphe orienté $\mathcal{G}$, et $k, k' \in \mathbb{N}$ avec $k \leq k'$. Montrer que si $x \Rightarrow_k y$, alors $x \Rightarrow_{k'} y$.

▶ **Question 2** Soient $x, y \in S$. Montrer que $x \Rightarrow y$ ssi $x \Rightarrow_{n-1} y$.

On représente chaque graphe par une matrice de booléens. De plus, pour deux matrices de booléens $\mathcal{M}_n(\mathbb{B})$, on définit leur produit par la même formule que le produit de deux matrices de réels, en remplaçant la somme par $\vee$ et le produit par $\wedge$. Par exemple :

$$\begin{pmatrix} \texttt{vrai} & \texttt{vrai} \\ \texttt{faux} & \texttt{vrai} \end{pmatrix} \begin{pmatrix} \texttt{faux} & \texttt{vrai} \\ \texttt{vrai} & \texttt{faux} \end{pmatrix} = \begin{pmatrix} \texttt{vrai} & \texttt{vrai} \\ \texttt{vrai} & \texttt{faux} \end{pmatrix}$$

En C, on représentera ces matrices de booléens par des pointeurs de type `bool**` : ce pointeur pointe vers un tableau de `bool*`, qui pointe eux-mêmes vers un tableau de booléens qui correspond à une ligne de la matrice. Ainsi, la syntaxe `m[i][j]` permet d'accéder à la case (i, j).

▶ **Question 3** Écrire une fonction `bool** create_matrix(int n, int p)` qui alloue sur le tas le tableau de pointeurs vers les lignes, chacune de ces lignes et initialise toute la matrice à `false`.

▶ **Question 4** Écrire la fonction de libération correspondante `void free_matrix(bool** m, int n)`.

▶ **Question 5** Écrire une fonction `bool** identity(int n)` qui renvoie une matrice identité de taille n.

▶ **Question 6** Écrire une fonction `bool** produit(bool** a, bool** b, int n, int p, int q)` qui à deux matrices a et b de tailles (n, p) et (p, q) renvoie la matrice produit ab (au sens du produit défini plus haut). *Une complexité cubique est acceptable ici (même si on pourrait également faire mieux!). Une façon pas trop compliquée de coder cette fonction est de commencer par coder la même fonction, avec la multiplication de deux matrices de réels, puis de remplacer les* + *par des* || *et des* $*$ *par des* &&.

Exercice 2 – Clôture transitive d'un graphe

Comme d'habitude, on représente un graphe $\mathcal{G}$ par sa matrice d'adjacence M. On note que cette matrice est exactement la matrice des $x_i \Rightarrow_1 x_j$ avec $i, j \in [\![0, n-1]\!]$.

▶ **Question 1** Montrer que pour tous $i, j \in [\![0, n-1]\!]$ et pour tout $k \in \mathbb{N}$, on a $(M^k)_{i,j} =$ `vrai` ssi $x_i \Rightarrow_k x_j$.

▶ **Question 2** Montrer que la suite des (M^k) est stationnaire à partir du rang $n-1$.

On appelle M^{n-1} la *clôture transitive* de la matrice M.

▶ **Question 3** Écrire une fonction `bool** closure(bool** m, int n)` qui calcule la clôture transitive de la matrice `m`. On fera attention à ne pas laisser de fuite de mémoire (ce qui sera la principale difficulté).

▶ **Question 4** Écrire une fonction `bool* accessible(bool** m, int n, int x)` qui renvoie un tableau t de booléens de taille n où t_y est vrai ssi $x \Rightarrow y$. On exige une complexité en $\mathcal{O}(n^2)$.

▶ **Question 5** En déduire une nouvelle fonction `closure2` de même prototype et valeur de retour que `closure` mais de complexité en $\mathcal{O}(n^3)$.

Exercice 3 – Axiomes et systèmes d'axiomes

Axiomes On dit qu'un sommet x est un *axiome* dans le graphe $\mathcal{G}$ si pour tout $y \in S$, si $y \Rightarrow x$ alors $x \Rightarrow y$.

▶ **Question 1** Déterminer les axiomes des graphes $\mathcal{G}_1$ et $\mathcal{G}_2$.

▶ **Question 2** Écrire une fonction `bool is_axiome(bool** clos_m, int n, int i)` qui prend en entrée la clôture d'une matrice, sa taille n et un sommet i et renvoie si ce sommet est un axiome.

Suites unidirectionnelles de sommets On dit que la suite de sommets $\langle y_0, \dots, y_k \rangle$ est une suite unidirectionnelle de sommets de $\mathcal{G}$ si pour tout $i \in [\![0, k-1]\!]$, $y_i \Rightarrow x_{i+1}$ et $y_{i+1} \not\Rightarrow x_i$.

▶ **Question 3** Montrer que les sommets d'une suite unidirectionnelle sont distincts.

▶ **Question 4** Soit y un sommet de $\mathcal{G}$. Montrer qu'il existe un axiome x tel que $x \Rightarrow y$.

▶ **Question 5** Déterminer les composantes fortement connexes de $\mathcal{G}_2$.

▶ **Question 6** On considère une composante fortement connexe C de $\mathcal{G}$ contenant un axiome. Montrer que tous les sommets de C sont des axiomes.

Systèmes d'axiomes On dit que $X \subseteq S$ est un *système d'axiomes* si pour tout $y \in S$, il existe $x \in X$ tel que $x \Rightarrow y$. De plus, on dit qu'une composante fortement connexe du graphe $\mathcal{G}$ est une *composante source* s'il contient un axiome.

▶ **Question 7** Montrer qu'en choisissant un sommet par composante source, on obtient un système d'axiomes.

▶ **Question 8** En déduire une fonction `bool* axiom_system(bool** clos_m, int n)` qui prend en entrée une clôture transitive de `m`, sa taille `n` et renvoie un système d'axiomes représenté par un tableau de booléens alloué sur le tas, avec des `true` pour les sommets dans le système et des `false` sinon.