# Feuille de TP n°3

Dans ce TP, on utilise le logiciel de calcul formel Maple pour implémenter des fonctions de manipulation des graphes. On aura deux représentations possible pour un graphe :

- une représentation sous forme de *liste d'adjacence*. Un graphe à $n$ sommet sera représenté par une liste de longueur $n$, dont le $i$-ème élément est la liste des sommets $j$ tels qu'il existe une arête $i \to j$.
- Une représentation sous forme de *matrice d'adjacence*. Un graphe à $n$ sommets est représenté par une matrice de taille $n \times n$, où le coefficient $(i, j)$ vaut 1 s'il y a une arête $i \to j$, et 0 s'il n'y en a pas.

Remarque : on peut également représenter de manière similaire un graphe avec des arêtes pondérées. Dans la première représentation, au lieu de donner pour chaque sommet simplement la liste de ses successeurs, on donne la liste des couples $(j, p)$ où $j$ est le successeur et $p$ la pondération. Dans la deuxième représentation, le coefficient $(i, j)$ vaut $p$.

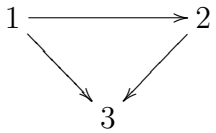
En Maple, les listes sont représentées par la syntaxe `[a,b,c]`. On peut accéder au $i$-ème élément d'une liste `L` par la syntaxe `L[i]`. Notez que la numérotation commence à 1 ! Le nombre d'éléments de la liste `L` est donné par `nops(L)`.

Pour déclarer une matrice, on utilise la syntaxe `Matrix([[a,b],[c,d]])` qui donne la matrice

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Notez que la syntaxe `Matrix(n)` où `n` est un entier produit la matrice carrée de taille `n` initialisée avec des 0. On accède au coefficient $(i, j)$ de la matrice `M` par la syntaxe `M[i,j]`. Le nombre de lignes de `M` est donné par `op(1,M)[1]` et le nombre de colonnes par `op(1,M)[2]`.

Par exemple, le graphe




est représenté sous forme de liste d'adjacence par `G := [[2,3],[3],[]]` et sous forme de matrice par `M := Matrix([[0,1,1],[0,0,1],[0,0,0]])`.

Pour déclarer une fonction en Maple, on utilise la syntaxe suivante.

```
f := proc(variables)

 contenu de la fonction

 return resultat;
end proc;
```

On peut déclarer des variables locales à l'intérieur d'une procédure en utilisant le mot-clé `local`.

Utilisez la combinaison de touches Maj-Entrée pour insérer un saut de ligne dans une fonction.

Voici également les syntaxes des tests et des boucles.

```
if condition then
 [...]
else
 [...]
fi;

for i from 1 to n do
 [...]
od;

while condition do
 [...]
od;
```

**Exercice 1.** Programmer des fonctions `list_to_matrix` et `matrix_to_list` pour passer d'une représentation d'un graphe à l'autre.

**Exercice 2.** Étant donné un graphe, programmez une fonction qui calcule le graphe opposé (dont le sens de chaque arête a été inversé). Quelle est la représentation la plus adaptée pour ce calcul ?

On pourra utiliser la bibliothèque `LinearAlgebra` (que l'on charge à l'aide de la syntaxe `with(LinearAlgebra)`).

**Exercice 3.** Programmez une fonction qui effectue un parcours en profondeur d'un graphe. Le résultat renvoyé par la fonction devra au moins contenir les informations suivantes : le temps de dernier passage en chaque sommet, et les composantes connexes de la forêt obtenue. Vous pouvez représenter ces informations de la manière que vous jugerez la plus adaptée.

**Exercice 4.** Programmer l'algorithme de calcul des composantes fortement connexes vu en cours. On pourra programmer une fonction auxiliaire pour réordonner les sommets d'un graphe de manière adéquate.

Il existe en Maple une bibliothèque pour la théorie des graphes. Il s'agit de la bibliothèque `GraphTheory` : vous pouvez la charger et lire son aide. Elle contient notamment une fonction de calcul des composantes fortement connexes dont le nom est `StronglyConnectedComponents`. Votre programme donne-t-il bien le même résultat que cette fonction ?