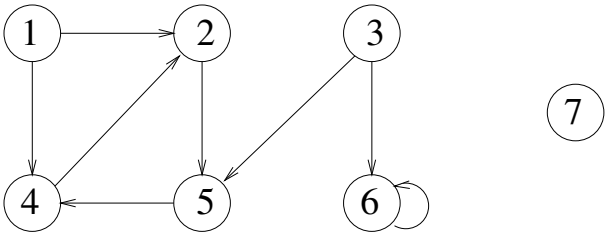
Nous allons nous intéresser à des algorithmes travaillant sur la représentation des graphes par matrice d'adjacence. Rappelons que la *matrice d'adjacence* d'un graphe G à n sommets est la matrice carrée d'ordre n telle que $M_{ij} = 1$ (ou VRAI) si (i, j) est un arc de G et $M_{ij} = 0$ (ou FAUX) autrement.



	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0
5	0	0	0	1	0	0	0
6	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0

1 Fermeture transitive de graphe

Le but du calcul de la fermeture transitive d'un graphe est de déterminer pour tout couple de sommets s'il existe un chemin reliant le premier au second. Un algorithme efficace (en $\Theta(n^3)$) est le suivant :

Algorithme 1 FERMETURE-EFFICACE(G)

soit n le nombre de sommets de G
soit M la matrice d'adjacence de G
soit H un graphe à n sommets de matrice d'adjacence N
pour $i \leftarrow 1$ **à** n **faire**
    pour $j \leftarrow 1$ **à** n **faire**
        si $i = j$ **alors**
            $N_{ij} \leftarrow 1$
        sinon
            $N_{ij} \leftarrow M_{ij}$
pour $k \leftarrow 1$ **à** n **faire**
    pour $i \leftarrow 1$ **à** n **faire**
        pour $j \leftarrow 1$ **à** n **faire**
            $N_{ij} \leftarrow N_{ij}$ ou (N_{ik} et N_{kj})
retourner H

Un type `Graphe` est défini en C de la manière suivante :

```
struct graphe{
```

```
  int n;
  int **mat;
};
typedef struct graphe Graphe;
```

On rappelle que l'initialisation dynamique d'un tableau à deux dimensions s'effectue comme celle d'un tableau de tableaux à une seule dimension. Il est donc nécessaire de commencer par allouer un tableau de pointeurs puis d'allouer chaque ligne du tableau. On obtient le code C suivant :

```
void initGraphe(int nb, Graphe *G)
{
  int i,j;
  (*G).n=nb;
  (*G).mat=(int **)malloc(sizeof(int *)*nb);
  for(i=0;i<nb;i++)
    (*G).mat[i]=(int *)malloc(sizeof(int)*nb);
  for(i=0;i<(*G).n;i++)
    for(j=0;j<(*G).n;j++)
      (*G).mat[i][j]=0;
}
```

Exercice 1 *Programmer l'algorithme précédent de calcul efficace de la fermeture transitive.*

2 Recherche de chemins les plus courts dans un graphe

Considérons maintenant le problème de la recherche de chemins les plus courts dans le cas d'un graphe dont chaque arc a une *longueur*, i.e. un poids associé. Pour cela, on utilise l'algorithme de Aho, Hopcroft et Ullman rappelé ci-dessous avec comme structure algébrique $\mathcal{S}(\sqcup, \odot, \emptyset, \varepsilon) = \mathbb{R}^+ \cup \infty(\min, +, \infty, 0)$ (pour tout x, $x^* = 0$) [voir poly p.91]. La notion de matrice d'adjacence est généralisée de la manière suivante : s'il existe un arc reliant le sommet i au sommet j l'entrée correspondante de la matrice prend comme valeur le poids de cet arc. S'il n'y a pas d'arc, l'entrée de la matrice prend la valeur ∞.

Algorithme 2 AHO-HOPCROFT-ULLMAN(G)

```
1  soit n le nombre de sommets de G
2  soit M la matrice d'adjacence (généralisée) de G
3  soit H un graphe à n sommets de matrice d'adjacence N
4  pour i ← 1 à n faire
5      pour j ← 1 à n faire
6          si i = j alors
```

```
7              N_ij ← 0
8          sinon
9              N_ij ← M_ij
10 pour k ← 1 à n faire
11     pour i ← 1 à n faire
12         pour j ← 1 à n faire
13             N_ij ← min(N_ij, N_ik + N_kj)
14 retourner H
```

Exercice 2 *Programmer l'algorithme de Aho-Hopcroft-Ullman et l'appliquer au problème de la recherche des chemins les plus courts.*

Exercice 3 (Application) *Montrer qu'à partir des distances connues indiquées ci-dessous, la distance minimale permettant d'aller d'une ville à l'autre est celle indiquée dans le second tableau.*

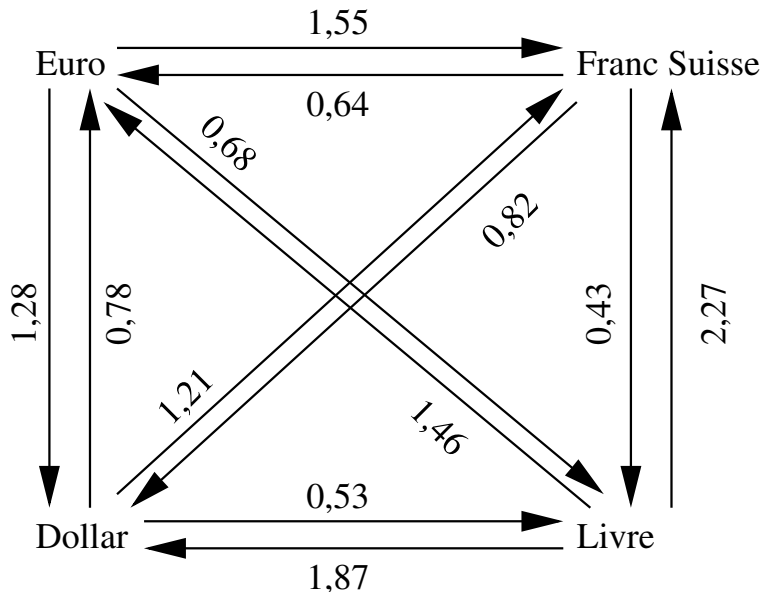
	Paris	Chartres	Beauvais	Reims	Evreux	Rouen	Le Mans	Nice	Brest	Grenoble	Lyon
Paris		82	60	140					564	576	472
Chartres					77		117				
Beauvais						80					
Reims											
Evreux						52					
Rouen											
Le Mans											
Nice									1351	340	480
Brest											890
Grenoble											104
Lyon											

	Paris	Chartres	Beauvais	Reims	Evreux	Rouen	Le Mans	Nice	Brest	Grenoble	Lyon
Paris	0	82	60	140	159	140	199	916	564	576	472
Chartres	82	0	142	222	77	129	117	998	646	658	554
Beauvais	60	142	0	200	132	80	259	976	624	636	532
Reims	140	222	200	0	299	280	339	1056	704	716	612
Evreux	159	77	132	299	0	52	194	1075	723	735	631
Rouen	140	129	80	280	52	0	246	1056	704	716	612
Le Mans	199	117	259	339	194	246	0	1115	763	775	671
Nice	916	998	976	1056	1075	1056	1115	0	1334	340	444
Brest	564	646	624	704	723	704	763	1334	0	994	890
Grenoble	576	658	636	716	735	716	775	340	994	0	104
Lyon	472	554	532	612	631	612	671	444	890	104	0

3 Spéculation monétaire

Considérons une application bien plus attrayante de l'algorithme précédent. On forme un graphe dont chaque sommet représente une devise (Euro, Dollar, Livre, Franc Suisse...). Ce graphe est complet, i.e. un arc relie chaque paire de sommet. Chaque arc est de plus pondéré par le taux de change entre les deux monnaies correspondantes.

Exercice 4 *Écrire un programme recherchant une chaîne de conversion entre monnaies permettant de s'enrichir. Appliquer le programme au graphe ci-dessous. Remarquer qu'un petite erreur de conversion permet de trouver un moyen de s'enrichir infiniment !*



4 Pour aller plus loin...

Une amélioration intéressante des algorithmes précédents consiste à les modifier de manière à indiquer non seulement la longueur des chemins les plus courts (dans le cas traité en section 2) mais également de fournir ces chemins explicitement.