Structure de données abstraites (SDA)	Structures relationnelles	Les graphes – Représentations et Implémentations en python

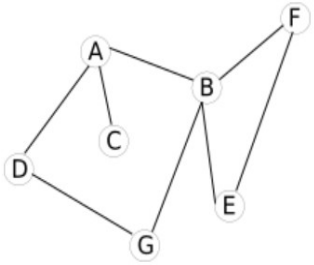
Il existe deux méthodes permettant d'implémenter un graphe : les matrices d'adjacences et les listes d'adjacences.

A) Représentation par matrice d'adjacence

Définition : Une matrice est un **tableau** à double entrée
La matrice A ci-contre est constituée de 5 lignes et 4 colonnes.
On appelle matrice **carrée** une matrice qui comporte le même nombre de lignes et de colonnes.
Les matrices d'adjacences sont des matrices carrées.

$$A = \begin{pmatrix} 2 & 3 & 2 & 10 \\ 5 & 0 & 8 & 8 \\ 9 & 2 & 7 & 12 \\ 4 & 11 & 14 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

Cas d'un graphe non orienté



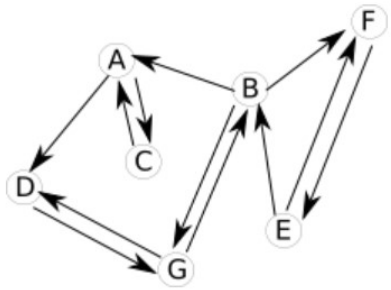
Sa matrice d'adjacence est :

$$\begin{pmatrix} 0&1&1&1&0&0&0 \\ 1&0&0&0&1&1&1 \\ 1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&1 \\ 0&1&0&0&0&1&0 \\ 0&1&0&0&1&0&0 \\ 0&1&0&1&0&0&0 \end{pmatrix}$$

A chaque ligne correspond un sommet du graphe et à chaque colonne correspond aussi un sommet du graphe. À chaque intersection ligne i-colonne j (ligne i correspond au sommet i et colonne j correspond au sommet j), on place un 1 s'il existe une arête entre le sommet i et le sommet j, et un zéro s'il n'en existe pas.

	A	B	C	D	E	F	G
A	0	1	1	1	0	0	0
B	1	0	0	0	1	1	1
C	1	0	0	0	0	0	0
D	1	0	0	0	0	0	1
E	0	1	0	0	0	1	0
F	0	1	0	0	1	0	0
G	0	1	0	1	0	0	0

Cas d'un graphe orienté

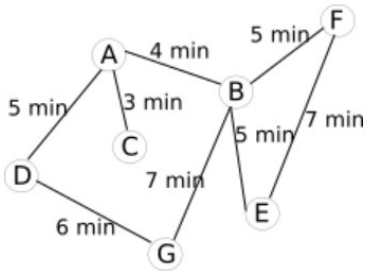


Sa matrice d'adjacence est :

$$\begin{pmatrix} 0&0&1&1&0&0&0 \\ 1&0&0&0&0&1&1 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1 \\ 0&1&0&0&0&1&0 \\ 0&0&0&0&1&0&0 \\ 0&1&0&1&0&0&0 \end{pmatrix}$$

La matrice n'est plus symétrique par rapport à la diagonale si le graphe est orienté

Cas d'un graphe pondéré :
Il est aussi possible d'utiliser une matrice d'adjacence pour implémenter un graphe pondéré : on remplace les 1 par les valeurs liées à chaque arc.



$$\begin{pmatrix} 0&4&3&5&0&0&0 \\ 4&0&0&0&5&5&7 \\ 3&0&0&0&0&0&0 \\ 5&0&0&0&0&0&6 \\ 0&5&0&0&7&0&0 \\ 0&5&0&0&7&0&0 \\ 0&7&0&6&0&0&0 \end{pmatrix}$$

Exercice 1 *soit* $M=\begin{pmatrix} 0&1&1&1 \\ 1&0&1&0 \\ 1&1&0&1 \\ 1&0&1&0 \end{pmatrix}$ *la matrice d'adjacence d'un graphe G*

(a) combien de sommet G a t-il ? | 3 | 4 | 16 | on ne peut pas savoir |

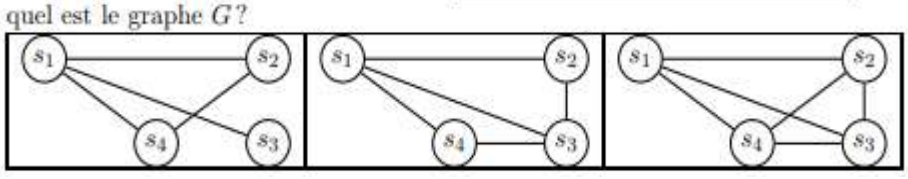
(b) combien d'arêtes G a t-il ? | 4 | 5 | 10 | on ne peut pas savoir |

(c) quel est le degré du premier sommet ? | 3 | 4 | 5 | on ne peut pas savoir |

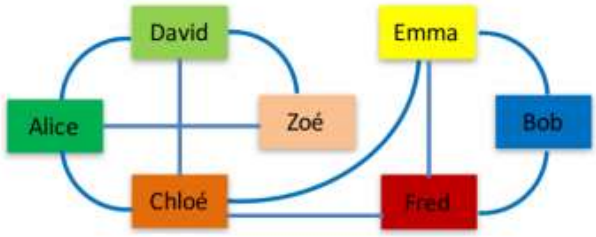
(d) les 2e et 4e sommets sont-ils adjacents ? | oui | non | on ne peut pas savoir |

(e) les 2e et 3e sommets sont-ils adjacents ? | oui | non | on ne peut pas savoir |

(f) quel est le graphe G ?



Exercice 2 : *Compléter la matrice d'adjacence et écrire son codage Python correspondant au graphe non orienté suivant traduisant la relation « est ami avec »*



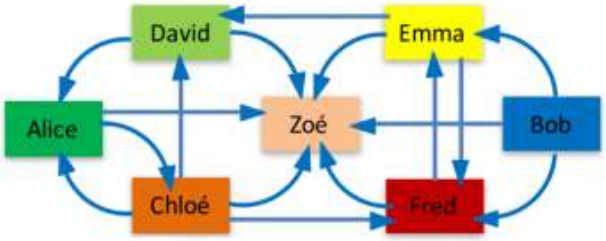
Graphe non orienté traduisant la relation : « X est ami avec Y ».

	Alice	Bob	Chloé	David	Emma	Fred	Zoé
Alice							
Bob							
Chloé							
David							
Emma							
Fred							
Zoé							

Matrice d'adjacence

Ecrire un programme Python permettant de calculer le nombre d'amis de chaque utilisateur du réseau « d'amitiés » précédent.

Exercice 3 *: Dans le modèle du mini-réseau social précédent, le graphe est non orienté : il traduit seulement le fait qu'un utilisateur est ami avec un autre (relation bijective). La notion de « followers » que l'on rencontre dans de nombreux réseaux sociaux (Twitter en est un exemple), nécessite quant à elle d'orienter les arêtes du graphe (elles deviennent alors des arcs) afin de traduire la relation « X » suit « Y ». Ainsi dans l'exemple de graphe orienté ci-dessous, Alice (origine) suit Zoé et Chloé (extrémités).*



Graphe orienté de traduisant la relation : « X suit Y »

	Alice	Bob	Chloé	David	Emma	Fred	Zoé
Alice	0	0	1	0	0	0	1
Bob	0	0	0	0	1	1	1
Chloé	1	0	0	1	0	1	1
David	1	0	0	0	0	0	1
Emma	0	0	0	1	0	1	1
Fred	0	0	0	0	1	0	1
Zoé	0	0	0	0	0	0	0

Matrice d'adjacence

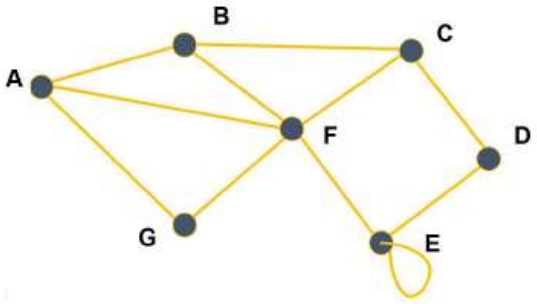
1. *Comment peut-on obtenir facilement le nombre de personnes suivies par une personne donnée ?*
2. *Comment peut-on obtenir facilement le nombre de personnes qui suivent une personne donnée ?*
3. *Compléter le programme Python de la question de l'exercice 2 afin qu'il affiche en plus du nombre d'amis pour une personne donnée, le nombre de personnes qu'elle suit et le nombre de personnes qui la suivent.*

B) Représentation par listes d'adjacences ?

Une autre façon de représenter un graphe est d'associer à chaque sommet la liste des sommets auxquels il est relié. Dans le cas d'un graphe orienté on parle de **liste de successeurs**, alors que dans un le cas d'un graphe non orienté on parle de **liste de voisins**

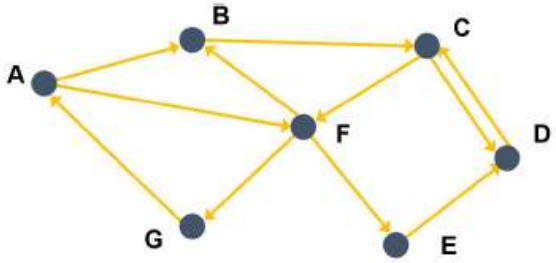
Une façon simple et efficace est d'utiliser un **dictionnaire** où chaque sommet est associé à la liste des successeurs/voisins

Graphe non orienté :




```
dico1 = {
    "A": ["B", "F", "G"],
    "B": ["A", "C", "F"],
    "C": ["B", "D", "F"],
    "D": ["C", "E"],
    "E": ["D", "E", "F"],
    "F": ["A", "B", "C", "E", "G"],
    "G": ["A", "F"]
}
```

Graphe orienté :




```
dico2 = {
    "A": ["B", "F"],
    "B": ["C"],
    "C": ["D", "F"],
    "D": ["C"],
    "E": ["D"],
    "F": ["B", "E", "G"],
    "G": ["A"]
}
```

C) Comparaisons des deux représentations :

La matrice d'adjacence est simple à mettre en œuvre mais nécessite un espace mémoire proportionnel à n× n (où n est le nombre de sommets). Ainsi un graphe de 1000 sommets nécessite une matrice d'un million de nombres même si le graphe ne contient que peu d'arêtes/arcs. Pour le même graphe, le dictionnaire ne mémoriserait pour chaque sommet que les voisins/successeurs (les 1) sans à avoir à mémoriser les autres(les 0). En revanche, pour un graphe contenant beaucoup d'arêtes, le dictionnaire occuperait plus d'espace mémoire que la matrice d'adjacence.

L'accès aux voisins est plus rapide avec le dictionnaire car il n'est pas nécessaire de parcourir toute la ligne.

De plus, l'utilisation d'un dictionnaire permet de nommer les sommets sans ambiguïté et ne les limite pas à des entiers

Enfin, au lieu d'utiliser le type liste (list de python) pour mémoriser les voisins, on peut avantageusement utiliser le type ensemble (type prédéfini set de python) qui est une structure de données permettant un accès plus efficace aux éléments.

Ouvrir le notebook graphes_implementation.ipynb pour implémenter des classes de graphes