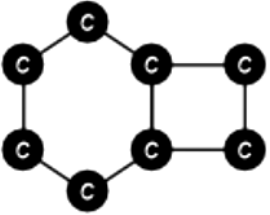
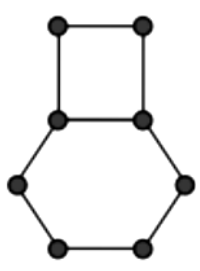
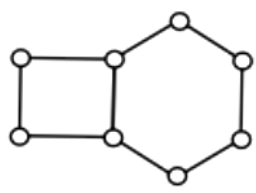



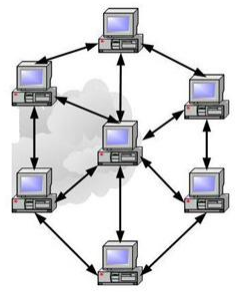
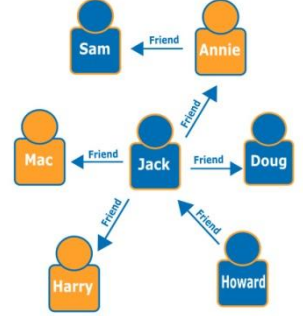

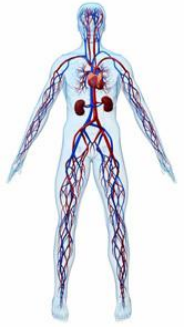
## Les graphes (Types abstraits)

CHEMISTRY	SOCIAL NETWORKS	BIOLOGY	MATH
 <p>BENZOCYCLOBUTADIENE</p> <p>● CARBON ATOMS — <math>\sigma</math>-ELECTRON BONDS</p>	 <p>● INDIVIDUALS — FRIENDSHIPS</p>	 <p>PPI (SUB)NETWORK OF A SIMPLE ORGANISM</p> <p>○ PROTEINS — INTERACTIONS</p>	<p>THEY LOOK THE SAME TO ME.</p> <p>LET'S CALL IT A GRAPH.</p> 

"MATHEMATICS IS THE ART OF GIVING THE SAME NAME TO DIFFERENT THINGS."  
JULES HENRI POINCARÉ (1854-1912)

Source : <http://spikedmath.com/382.html>

Si on va au plus général, un « graphe » est une représentation d'un jeu d'informations qui permet de caractériser des relations entre différents « sujets »... Plus concrètement :

			
Réseau internet : on relie des machines	Réseau social : relations de type « like »	Réseau routier : on relie des villes	Réseau sanguin : organes reliés par des vaisseaux

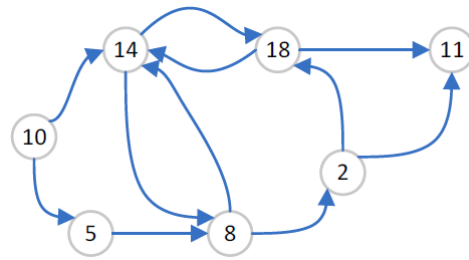
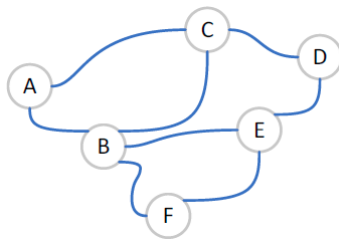
*Nota : Les graphes sont en fait une généralisation des arbres, dont on parlera plus en détail plus tard dans l'année...*

## I) Définitions

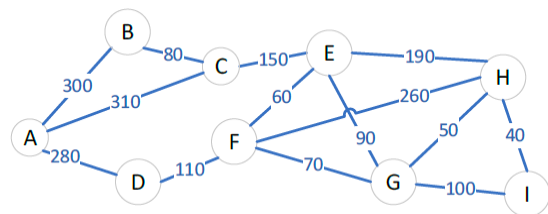
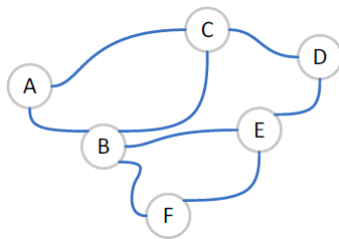
### I.1) Types de graphes

Ils sont nombreux... On retrouve cependant des grands classiques, selon ce que le graphe doit contenir/représenter comme informations.

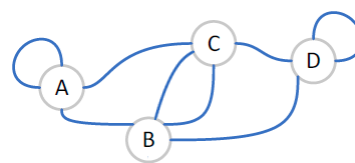
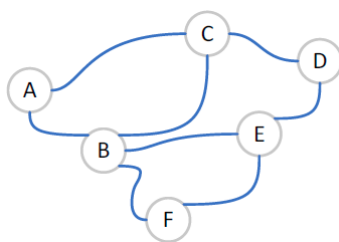
➤ Selon que les « liens » sont directionnels ou non :



➤ Selon ce que les liens signifient :



➤ Selon le nombre de liens directs entre deux sommets :



Et on peut évidemment avoir des combinaisons de ces caractéristiques (ainsi que d'autres, comme la connexité, dont on reparlera plus tard dans l'année...).

## **I.2) Constitution d'un graphe / définitions**

- **Sommets / nœuds :** (en anglais : vertex/vertices ou nodes)

- **Liens entre deux nœuds ou plus :**

Beaucoup de termes changent légèrement selon qu'on parle de graphes orientés ou non :

<i>Notion</i>	<i>Graphe...</i>	
	<i>non orienté</i>	<i>orienté</i>
Lien direct entre deux sommets		
Suites de sommets liés directement		
Suites de sommets dont les deux extrémités sont identiques		

- **Étiquetage des nœuds ou des arêtes/arcs :**

Il est fréquent qu'on associe des données aux sommets (nom, valeur, ...) ou aux arêtes/arcs. On parle alors d'un *graphe étiqueté* (Nota : si les étiquettes sur les arêtes sont des valeurs numériques, on a alors un graphe pondéré).

- **Notions de voisinage, adjacence :**

Pour un *graphe orienté*, il peut y avoir deux types de voisins, donc :

- 
- 
- 

Pour un graphe *non orienté* :

- **Ordre d'un graphe :**

## II) Implantations

### II.1) Idée générale

Quelle que soit l'implantation utilisée, elle doit permettre :

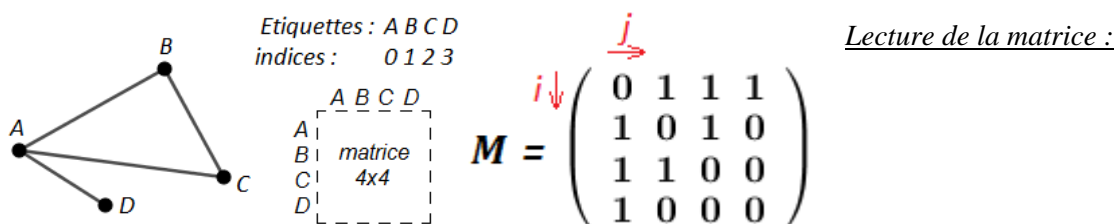
- D'identifier chaque sommet (soit par un index, soit par son étiquette)
- De dire si tel sommet est lié ou non à tel sommet
- Si le graphe est pondéré, de donner le poids de l'arête/l'arc
- Si le graphe est orienté, on doit pouvoir différencier les deux directions possibles

### II.2) Matrices d'adjacences

➤ **Principe :** C'est la version « mathématicienne » du graphe.

Un sommet peut à priori être lié à n'importe quel autre sommet du graphe, donc pour  $N$  sommets, on doit utiliser une matrice carrée  $M$  de taille  $N \times N$ .

Pour savoir quelle ligne/colonne correspond à quel sommet, on doit définir un ordre pour les sommets (à définir par l'utilisateur, si les sommets ne sont pas numérotés).



➤ **Implantation :** Un « tableau 2D » donc, en python, on utilise en général une liste de listes (de nombres). De cette façon :

```
[[0, 1, 1, 1],
 [1, 0, 1, 0],
 [1, 1, 0, 0],
 [1, 0, 0, 0]]
```

➤ **Intérêts / désavantages :**

- Facile d'identifier si le graphe est orienté ou non :

Dans un graphe non orienté, toutes les arêtes sont identiques quand on les parcourt dans les deux sens, donc  $M_{i,j} = M_{j,i}$ , donc la matrice doit être symétrique selon la diagonale principale.

- Graphes simples ou multigraphes :

On identifie facilement les graphes contenant des boucles : on cherche des valeurs sur la diagonale principale.

- *Graphes pondérés ou non :*

S'il n'y a que des 0 et des 1, le graphe est non pondéré. S'il y a d'autres valeurs, c'est soit un graphe pondéré, soit un multigraphe (la valeur donnant le nombre d'arêtes liant les deux sommets, par exemple).

- *Advantages :*

- Simple de mise en œuvre
- Facile d'avoir accès aux voisins d'un sommet :
  - Les successeurs sont dans la ligne du sommet
  - Les prédécesseurs sont dans la colonne du sommet.

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

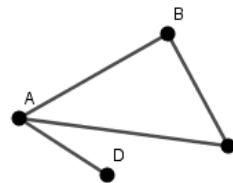
- *Inconvénients :*

- Beaucoup de place perdue en mémoire si pas beaucoup d'arêtes/arcs (car il faut tout de même stocker les 0 !)
- Peu lisible...

### II.3) Listes d'adjacence

Si le graphe est peu dense, les listes d'adjacences sont plus intéressantes que les matrices car elles vont alors prendre moins de place en mémoire.

Selon qu'on travaille avec les indices ou les étiquettes, on peut les implanter de différentes manières. En reprenant l'exemple de la section précédente :



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

Etiquettes : A B C D  
indices : 0 1 2 3

**Principe :**

Sommet	Voisins
A	[B,C,D]
B	[A,C]
C	[A,B]
D	[A]

**Implantations possibles :**

<i>Liste de listes d'indices</i>	<i>Listes de listes d'étiquettes, avec une liste d'étiquettes associées (pour retrouver les indices/etiquettes)</i>
<pre> graphe = [     [1,2,3],     [0,2],     [0,1],     [0], ] </pre>	<pre> etiquettes = ['A','B','C','D'] graphe = [ ['B','C','D'],             ['A','C'],             ['A','B'],             ['A'], ] </pre>

**Advantages :**

- Plus légères en mémoire que les matrices (s'il n'y a pas beaucoup d'arêtes)
- Souvent plus lisible, mais pas totalement claire non plus...

***Inconvénients :***

- Il faut savoir quel sommet correspond à quelle liste

- Pour les graphes pondérés, il faut stocker des tuples : (*étiquette*, *poids*)  $\Rightarrow$  lourd
- Pour les graphes orientés, souvent on ne donne que la liste des successeurs et il faut alors reconstruire la liste de prédécesseurs...
- C'est LENT pour certaines opérations, si les sous-listes sont longues (ex :  $v$  in  $lst$  qui est  $O(n)$ )
- Pénible à définir pour les graphes non orientés (tout « en double »)

#### **II.4) Dictionnaires d'adjacence**

Pour pouvoir utiliser directement les étiquettes des sommets dans le code, on peut partir sur un dictionnaire. Cela permet aussi d'éviter d'avoir à passer par des indices.

Les valeurs stockées peuvent prendre différentes formes, selon le graphe :

- Graphe non pondéré :
- Graphe pondéré :

*L'exemple précédent :*

```
graphe = {
    'A': set(['B', 'C', 'D']),
    'B': set(['A', 'C']),
    'C': set(['A', 'B']),
    'D': set(['A']),
}
```

*Un graphe orienté pondéré (successeurs) :*

```
graphe = {
    'A': {'B':12, 'C':3},
    'B': {},
    'C': {'A':5, 'F':10},
    'D': {'A':2, 'B':4, 'C':18},
    'F': {} }
```

Le dictionnaire d'adjacence ci-dessus à droite décrit le graphe pondéré et orienté ci-contre :

- *Avantages :*
  - Lisible
  - Efficace sur tous types d'opérations
- *Inconvénients :*
  - Plus gourmand en mémoire que la liste d'adjacence équivalente
  - Pénible à définir pour les graphes non orientés (tout « en double »)
  - On n'a pas directement l'information concernant les prédécesseurs...

#### **Remarque sur les sets :**

En gros, les *sets* sont des dictionnaires, mais avec uniquement des clefs, et aucune valeur associée aux clefs. Le principal intérêt est de pouvoir y faire des ajouts, suppression, recherches d'éléments en temps constant,  $O(1)$ , alors que les mêmes opérations sur les listes sont très souvent en  $O(n)$ .