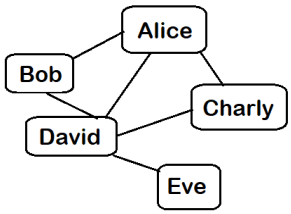
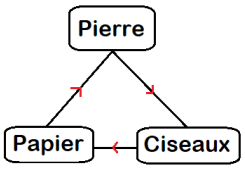
Les graphes sont des objets mathématiques constitués de différents objets (les sommets) mis en relation les uns avec les autres via des arêtes.

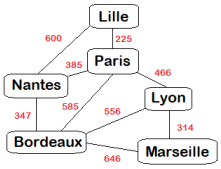
Ils sont au coeur de nombreux domaines de l'informatique et des mathématiques tels que les automates, les réseau ou la théorie des jeux.

1 Définitions

Différentes situations illustrées par des graphes :







1. Graphe des amis (graphe non-orienté) ;
2. Règles du pierre-papier-ciseaux (graphe orienté) ;
3. Routes et distances entre des villes (graphe pondéré).

Graphe orienté : Un graphe orienté G est la donnée de deux ensembles :

1. V un ensemble fini de *sommets* (Vertices) ;
2. $E \subseteq V \times V$ un ensemble d'*arêtes* (Edges). $E \subseteq V \times V$ signifie que E consiste en un ensemble de couples (v_1, v_2) avec v_1, $v_2 \in V$.

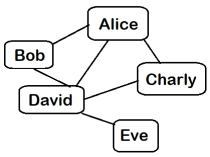
On interprète ces ensembles de la manière suivante : les sommets sont des objets reliés entre eux par les arêtes, arêtes pouvant n'aller que dans un sens. Plus précisément, deux sommets v_1 et v_2 sont reliés ssi le couple (v_1, v_2) est présent dans E.

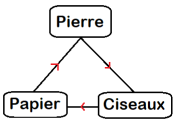
Graphe non-orienté : Un graphe non orienté est un graphe orienté dans lequel chaque arête est présente dans les deux sens :

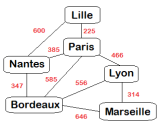
$$(v_1, v_2) \in E \Rightarrow (v_2, v_1) \in E$$

Graphe pondéré : Un graphe pondéré est un graphe orienté muni d'une *fonction de poids* $w : E \longrightarrow \mathbb{R}$ (weight) associant à chaque arête un certain poids.

Reprenons nos exemples :







1. — $V = \{\text{Alice}, \text{Bob}, \text{Charly}, \text{David}, \text{Eve}\}$
   — $E = \{$(Alice,Bob) ,(Bob,Alice) ,(Alice,David) ,(David,Elice) ,(Alice,Charly) ,(Charly,Alice) ,(Bob,David) ,(David,Bob) ,(Charly,David) ,(David,Charly) ,(David,Eve) ,(Eve,David)$\}$
2. — $V = \{\text{Pierre}, \text{Papier}, \text{Ciseaux}\}$
   — $E = \{$(Pierre, Ciseaux), (Ciseaux, Papier), (Papier, Pierre)$\}$
3. — $V = \{$ Lille, Paris, Nantes, Bordeaux, Lyon, Marseille $\}$
   — $E =$ long...
   — Table de w :

(v_1, v_2)	$w(v_1, v_2) = w(v_2, v_1)$
(Lille, Paris)	225
(Lille, Nantes)	600
(Paris, Nantes)	385
(Paris, Lyon)	466
(Paris, Bordeaux)	585
(Nantes, Bordeaux)	347
(Lyon, Bordeaux)	556
(Lyon, Marseille)	314
(Marseille, Bordeaux)	646

Pour chacune des situations suivantes, déterminer le type de type de graphe le plus adapté à la modélisation et proposer un exemple (avec au moins 4 sommets) sous la forme d'un schéma puis détailler la formalisation mathématique :

1. lieux d'un jeu vidéo ;
2. "followers" sur un réseau social
3. liens internes d'un site web ;
4. trafic routier.

2 Représentation par listes d'adjacence

Une première possibilité de représentation d'un graphe consiste à donner, pour chaque sommet, la liste des sommets étant reliés (de manière orientée ou non) à celui-ci.

Par exemple, pour l'exemple 1, une implémentation peut être :

```
noms = ["Alice", "Bob", "Charly", "David", "Eve"]
#ici "Alice" sera le sommet 0, "Bob" le sommet 1, etc.

amis = 5*[[]]
amis[0] = [1,2,3] #alice a pour amis Bob et Charly
amis[1] = [0,3] #etc.
amis[2] = [0,3]
amis[3] = [0,1,2,4]
amis[4] = [3]
```

Ici on définit une correspondance entre des indices et les noms des personnes représentées dans le tableau `noms` puis, pour chaque indice, on donne la liste des indices des amis correspondants dans le tableau `amis`.

Ou bien, en programmation orientée objet :

```
class Personne :
    def __init__(self, nom_personne):
        self.nom = nom_personne
        self.amis = []

alice = Personne("Alice")
bob = Personne("Bob")
charly = Personne("Charly")
david = Personne("David")
eve = Personne("Eve")

alice.amis += [bob, charly, david]
bob.amis += [alice, david]
charly.amis += [alice, david]
david.amis += [alice, bob, charly, eve]
eve.amis += [david]
```

Ici, chaque personne est définie par son nom et la liste de ses amis (initialement vide).

1. Dans les deux implémentations précédentes, écrire :
   (a) une fonction / une méthode affichant les noms des amis d'une personne donnée.
   (b) une fonction permettant de savoir si deux personnes sont amies ou non.
2. Représenter avec l'implémentation de votre choix l'exemple 2.

Si on veut représenter un graphe pondéré, il faut en plus de la donnée des sommets et des arêtes, renseigner le poids de chaque arête. Une possibilité consiste à modifier les listes d'adjacences de sorte que celles-ci contiennent des tuples (voisin, poids de l'arête associée).

Exercice : Écrire une représentation, par des tableaux ou en POO, de l'exemple 3 (distance entre les villes).

3 Représentation par matrice d'adjacence

Soit $V = \{0, 1, ..., n-1\}$ un ensemble de sommets numérotés et $G = (V, E)$ un graphe non-pondéré. Une autre possibilité de représentation est la matrice d'adjacence M de G. Celle-ci est de taille $n \times n$ et définie par :

$$M_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

Par exemple, dans l'exemple 1, en remplaçant Alice, Bob, etc. par 0, 1, 2, 3, 4, on peut donner la matrice suivante :

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

L'implémentation est assez proche de la version tableau des listes d'adjacence, en reprenant l'exemple 1 :

```
noms = ["Alice", "Bob", "Charly", "David", "Eve"]
#ici "Alice" sera le sommet 0, "Bob" le sommet 1, etc.

amis = 5*[[]]#
amis[0] = [0,1,1,1,0] #alice a pour amis Bob et Charly
amis[1] = [1,0,0,1,0] #etc.
amis[2] = [1,0,0,1,0]
amis[3] = [1,1,1,0,1]
amis[4] = [0,0,0,1,0]
```

Exercice :

1. Implémenter l'exemple 2 (pierre, papier, ciseaux) dans cette représentation.
2. Écrire une fonction affichant "gagné", "perdu" ou "égalité" étant donné deux choix au pierre, papier, ciseaux et utilisant la matrice d'adjacence.

On choisira une représentation ou l'autre en fonction de la situation à représenter. Quelques pistes :

— La représentation matricielle est plus gourmande en espace mémoire car pour un graphe à n sommets, la matrice est toujours de taille n^2 alors qu'avec les listes d'adjacence, on n'a que $|E| \leq n^2$ voisins à retenir.

— En revanche, savoir si une arête est présente entre deux sommets est plus rapide dans la représentation matricielle, car elle ne nécessite qu'un appel de $M[i][j]$ alors que dans la représentation par listes d'adjacence une recherche dans une liste (de complexité linéaire en sa taille) est nécessaire.

— On privilégiera donc une représentation matricielle dans les cas où le nombre de voisins par sommet est important (matrice dite dense, beaucoup de 1) et la représentation par liste d'adjacence si le nombre de voisin par sommet est petit (matrice dite vide).

— Enfin, on verra plus tard que selon la représentation utilisée les algorithmes de graphes associés ne s'effectuent pas de la même manière, ce qui peut également orienter le choix de l'implémentation.