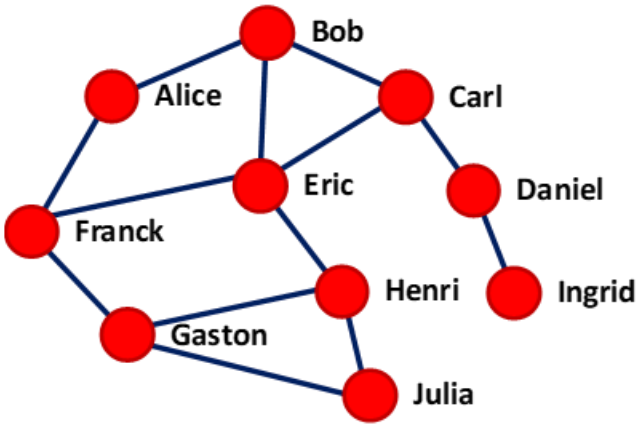
Structures de données relationnelles : graphes

I./ Généralités sur les graphes :

Un *graphe* est une *structure de données relationnelle* constituée de *nœuds* reliés éventuellement par des *arêtes*. Les arêtes traduisent les *relations* qui existent entre deux nœuds du graphe. Ces relations peuvent être *symétriques*, ou *asymétriques*. Dans une relation symétrique, si le nœud A est en relation avec le nœud B, alors le nœud B est en relation avec le nœud A. On représente les arêtes correspondant à une relation symétrique par des segments. On parle alors de *graphe non orienté*. Dans une relation asymétrique, si le nœud A est en relation avec le nœud B, alors le nœud B n'est pas forcément en relation avec le nœud A. On représente les arêtes correspondant à une relation asymétrique par des flèches (ou *arêtes orientées*). On parle alors de *graphe orienté.*

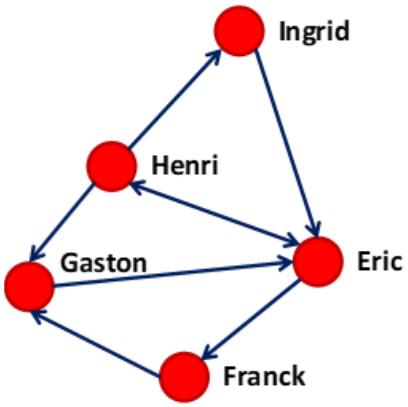
Exemple : on peut représenter un réseau social par un graphe. Les nœuds correspondent aux utilisateurs du réseau social, et les arêtes correspondent aux relations d'amitié.

Dans certains réseaux sociaux, comme Facebook par exemple, les relations d'amitié sont symétriques : si A est ami avec B, alors B est ami avec A :



Graphe non orienté (modèle « Facebook »)

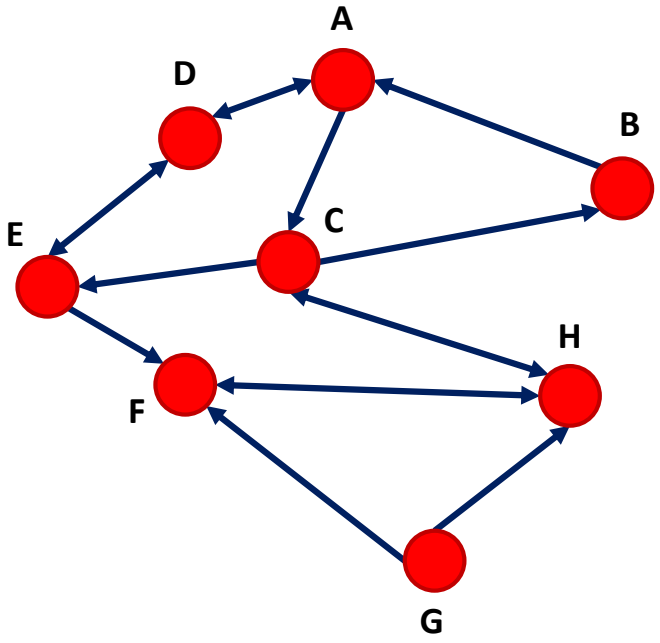
Dans d'autres réseaux sociaux, comme Instagram par exemple, les relations entre utilisateurs sont asymétriques : si A suit B, alors B ne suit pas forcément A :



Graphe orienté (modèle « Instagram »)

Dans cet exemple, Ingrid suit Eric, mais Eric ne suit pas Ingrid. Henri suit Eric, et Eric suit aussi Henri. Le sens des flèches indique le sens des relations.

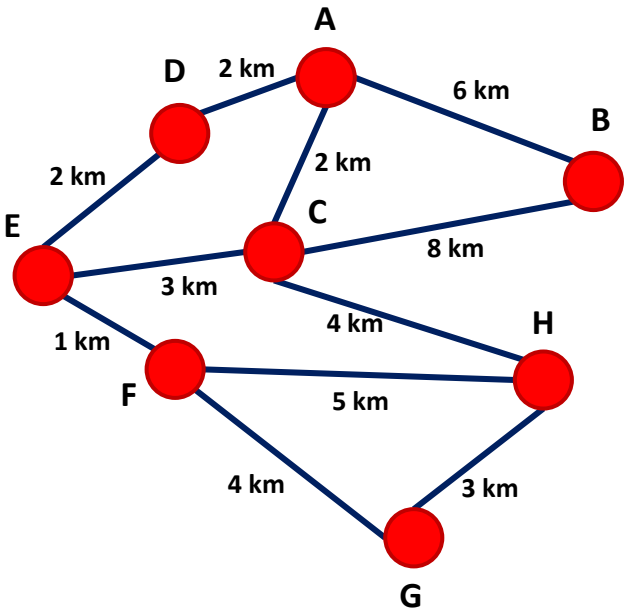
Autre exemple : on peut aussi utiliser les graphes pour représenter un réseau routier. Les nœuds correspondent alors à des villes, et les arêtes aux routes les reliant. Une arête non orientée, correspondant à une relation symétrique, permettra de représenter une route à double sens. Une arête orientée, correspondant à une relation asymétrique, permettra de représenter une route à sens unique.



Représentation d'un réseau routier par un graphe orienté

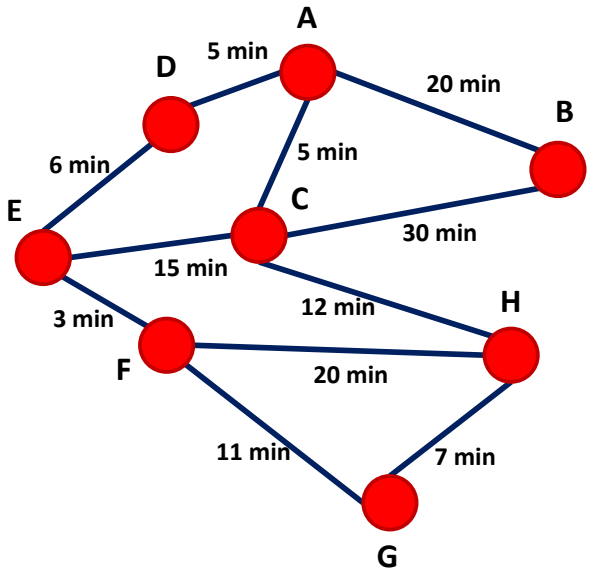
Dans cet exemple, on peut aller de A à D, et de D à A. Par contre, on peut aller de A à C, mais pas de C à A.

On peut associer une valeur, appelée *poids*, à chaque arête du graphe : on parle alors de *graphe pondéré*. Par exemple, dans le cas de notre réseau routier, on peut associer à chaque arête la distance en kilomètres qui sépare les deux villes. (Pour simplifier, on considère un graphe non orienté).



Représentation du réseau routier par un graphe non orienté pondéré par la distance en km

On pourrait aussi associer à chaque arête le temps en minutes nécessaire pour aller d'une ville à l'autre.



Représentation du réseau routier par un graphe non orienté pondéré par le temps de parcours en minutes

Ces pondérations peuvent être très utiles pour trouver le meilleur chemin entre deux points à l'aide d'un GPS. Si le conducteur veut utiliser l'itinéraire le plus court en kilomètres, le GPS se servira du graphe pondéré par les distances en kilomètres. Si le conducteur veut utiliser l'itinéraire le plus court en temps, le GPS se servira du graphe pondéré par les temps de parcours en minute. Dans ce dernier cas, il est intéressant de noter que ces temps de parcours peuvent changer au cours du temps, à cause de l'évolution du traffic, ou d'un accident. La valeur des poids peut donc changer, et certaines applications, comme Waze, peuvent mettre à jour la valeur de ces poids.

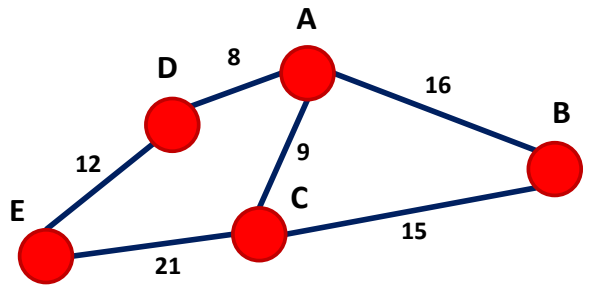
Quelques points de vocabulaire :

- **Chemin :** c'est une suite d'arêtes consécutives reliant deux nœuds distincts d'un graphe. On le désigne par les lettres des noeuds qu'il comporte. Il peut exister plusieurs chemins qui relient deux mêmes nœuds d'un réseau. Par exemple, dans le cas du réseau routier précédent, un chemin possible pour aller de A à H est [A, C, H]. Un autre chemin possible est [A, B, C, H].
- **Cycle :** c'est un chemin qui commence et se termine par le même nœud. Par exemple, [E, C, H, F, E] est un cycle. [A, B, C, A] en est un autre.
- **Distance :** la distance entre deux noeuds d'un graphe est la somme des poids des arêtes que l'on rencontre lorsqu'on parcourt le plus court chemin entre les deux nœuds. Par exemple, dans notre réseau routier, le plus court chemin entre A et E est [A, D, E], que l'on considère la distance en kilomètres ou le temps en minutes. (Attention : dans d'autres graphes, le plus court chemin entre deux points peut être différent selon la pondération utilisée). La distance entre A et E est donc de 4 kilomètres, ou de 11 minutes, selon la métrique utilisée. Dans le cas d'un graphe non pondéré, la distance correspond tout simplement au nombre minimal d'arêtes qu'il faut parcourir pour aller d'un nœud à l'autre.
- **Voisins :** deux nœuds d'un graphe sont voisins si et seulement si ils sont directement reliés par une arête. Les points F et H sont voisins dans notre exemple.

II./ Comment représenter un graphe :

1./ Représentation d'un graphe par une matrice d'adjacence :

Une matrice est un tableau de nombres. Pour chaque nœud du graphe, la *matrice d'adjacence* recense le poids des arêtes qui sépare le nœud de ses voisins. Soit l'exemple suivant :



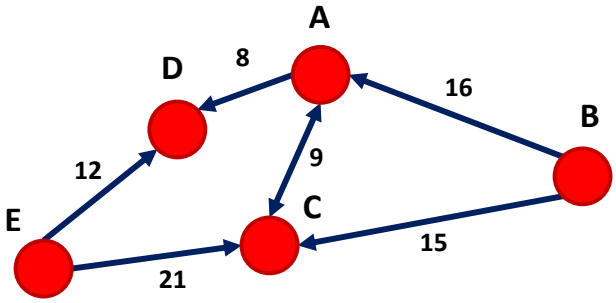
La matrice d'adjacence de ce graphe est :

	A	B	C	D	E
A	**0**	**16**	**9**	**8**	**0**
B	**16**	**0**	**15**	**0**	**0**
C	**9**	**15**	**0**	**0**	**21**
D	**8**	**0**	**0**	**0**	**12**
E	**0**	**0**	**21**	**12**	**0**

La première ligne correspond au nœud A. Les voisins du nœud A sont les nœuds B, C et D. La distance entre A et lui-même est égale à 0. La distance entre A et B est de 16. La distance entre A et C est de 9. La distance entre A et D est de 8. Le nœud A n'est pas voisin avec le nœud E, donc on place 0 dans la case correspondante.

Les autres lignes se remplissent de la même façon. On remarque que la matrice d'adjacence est symétrique, ce qui est caractéristique d'un graphe non orienté.

Modifions le graphe précédent pour le transformer en graphe orienté :



La matrice d'adjacence de ce graphe est :

	A	B	C	D	E
A	**0**	**0**	**9**	**8**	**0**
B	**16**	**0**	**15**	**0**	**0**
C	**9**	**0**	**0**	**0**	**0**
D	**0**	**0**	**0**	**0**	**0**
E	**0**	**0**	**21**	**12**	**0**

On remarque que la matrice d'adjacence n'est pas symétrique, ce qui est caractéristique d'un graphe orienté.

L'avantage de la représentation d'un graphe par sa matrice d'adjacence est sa grande simplicité : en effet on peut stocker cette matrice sous la forme d'un tableau, donc d'une liste de listes en Python. Le problème de cette représentation est qu'elle prend beaucoup de place en mémoire. En effet, si le graphe comporte N nœud, la matrice d'adjacence contient N^2 nombres. De plus, cette matrice d'adjacence comportera un très grand nombre de valeurs nulles dans le cas général (on parle de *matrices creuses*). Pour éviter ces problèmes, on peut utiliser une autre représentation, les *listes d'adjacences*.

2./ Représentation d'un graphe par une liste d'adjacence :

Pour établir la liste d'adjacence d'un graphe, on commence par recenser tous les nœuds composants ce graphe. Ensuite, pour chacun de ces nœuds, on fait la liste de tous les autres nœuds qui sont en relation, avec les poids correspondants. Si on reprend les deux exemples précédents, on a :

- *Pour le graphe non orienté :*
  - A : [(B, 16), (C , 9), (D, 8)]
  - B : [(A, 16), (C, 15)]
  - C : [(A, 9), (B, 15), (E, 21)]
  - D : [(A, 8), (E, 12)]
  - E : [(C, 21), (D, 12)]

- *Pour le graphe orienté :*
  - A : [(C , 9), (D, 8)]
  - B : [(A, 16), (C, 15)]
  - C : [(A, 9)]
  - D : [ ] (Le point D n'est en relation avec aucun autre point)
  - E : [(C, 21), (D, 12)]

Cette représentation prend moins de place en mémoire, par contre sa manipulation sera plus complexe. C'est cette représentation que l'on utilisera dans l'implémentation en Python suivante.

III./ Première implémentation des graphes en Python :

Pour représenter la liste d'adjacence, on va utiliser un dictionnaire dont les clefs seront les différents nœuds du graphe, et les valeurs seront des listes comportant des tuples avec les nœuds en relation et le poids de l'arête (voir le paragraphe précédent).

```
# les deux modules suivants sont utilisés pour l'affichage des graphes
import matplotlib.pyplot as plt
import math

class Graphe():
    """Graphe implémenté à l'aide d'un dictionnaire
    var = Graphe()       : crée un graphe vide
    var = Graphe(dict)   : crée un graphe à partir du dictionnaire 'dict'
    Les clefs du dictionnaires correspondent aux différents noeuds du graphe
    Les valeurs correspondent aux arêtes (noeud d'arrivée et poids)
    Exemple :
    {'A': [('M', 65), ('R', 90), ('T', 80)],
     'G': [('L', 70)],
     'L': [('G', 70), ('P', 230), ('T', 260)],
     'M': [('A', 65), ('P', 95), ('R', 90), ('T', 55)],
     'P': [('L', 230), ('M', 95), ('T', 130)],
     'R': [('A', 90), ('M', 90)],
     'T': [('A', 80), ('L', 260), ('M', 55), ('P', 130)]}
    """

    def __init__(self, noeuds = None):
        """Initialisation avec un graphe vide
        __noeuds est un dictionnaire
        - clé = Valeur du noeud (chaine, entier)
        - valeur = liste d'aretes (clé, poids)"""
        if noeuds is None:
            self.__noeuds = dict()
        else:
            self.__noeuds = noeuds


    def ajouter_noeud(self, noeud):
        """Ajoute un nouveau noeud au graphe"""
        if not noeud in self.__noeuds:
            self.__noeuds[noeud] = []


    def ajouter_arete(self, noeud1, noeud2, poids=1):
        """Ajoute une arête au graphe
        si poids n'est pas renseigné, il prendra la valeur 1
        si les noeuds n'existent pas encore, ils sont créés"""
        # On s'assure que les arêtes existent
        self.ajouter_noeud(noeud1)
        self.ajouter_noeud(noeud2)
        # On crée la connexion noeud1 -> noeud2
        self.__noeuds[noeud1].append((noeud2, poids))


    def liste_noeuds(self):
        """Renvoie la liste des noeuds"""
        noeuds = list(self.__noeuds.keys())
        noeuds.sort()
        return noeuds


    def voisins(self, noeud):
        """Renvoie la liste des noeuds voisins de 'noeud'"""
        if noeud in self.liste_noeuds():
            return [i[0] for i in self.__noeuds[noeud]]
        else:
            return []


    def arete(self,noeud1, noeud2):
        """Renvoie le poids de l'arete noeud1->noeud2 ou 0 si pas d'arête"""
        if noeud2 not in self.voisins(noeud1):
            return 0
        for i in self.__noeuds[noeud1]:
            if i[0] == noeud2:
                return i[1]
```

```
    def affiche(self):
        """Affiche une représentation du graphe avec les valeurs des poids"""
        x_noeuds = []
        y_noeuds = []
        angle = 0
        noeuds = self.liste_noeuds()
        N = len(noeuds)
        # on place d'abord tous les noeuds du graphe à égale distance sur un cercle unitaire
        for noeud in range(N):
            x_noeuds.append(math.cos(angle))
            y_noeuds.append(math.sin(angle))
            angle += 2 * math.pi / N
        plt.scatter(x_noeuds, y_noeuds, marker = "o")
        # on écrit ensuite les étiquettes des différents noeuds
        for noeud in range(N):
            plt.annotate(noeuds[noeud], xy = (x_noeuds[noeud], y_noeuds[noeud]), size = 'xx-large')
        # pour chaque noeud du graphe...
        for i in range(N):
            x0 = x_noeuds[i]
            y0 = y_noeuds[i]
            noeud = noeuds[i]
            # ... on regarde quels sont les noeuds voisins
            voisins = self.voisins(noeud)
            # pour chaque voisin...
            for j in voisins:
                x1 = 0
                y1 = 0
                for k in range(N):
                    if j == noeuds[k]:
                        x1 = x_noeuds[k]
                        y1 = y_noeuds[k]
                        break
                # on trace un vecteur entre le noeud de départ et le voisin considéré
                plt.arrow(x0,y0,x1-x0,y1-y0,length_includes_head=True,head_width=0.05,head_length=0.1)
                # et on affiche le poids au milieu du vecteur
                poids = self.arete(noeud, noeuds[k])
                plt.annotate(str(poids), xy = ((x0 + x1) / 2, (y0 + y1) / 2), size = 'large', color = 'red')
        # on affiche la représentation graphique obtenue
        plt.show()
```

Après avoir tapé le script de cette classe « **`Graphe`** », utilisez la commande **`help(Graphe)`** et testez les différentes méthodes pour comprendre leur utilisation. Une fois cela fait, utilisez ces méthodes pour recréer et afficher les différents réseaux vus dans le cours : les deux réseaux sociaux et les trois réseaux routiers.

Remarques :

- Pour un réseau non orienté, si on veut relier deux nœuds A et B, il faut penser à créer la relation de A vers B ET la relation de B vers A.
- Pour un réseau non pondéré, il suffit de mettre le même poids (différent de zéro) à chaque relation. L'habitude est de prendre la valeur du poids égale à 1.
- La méthode « **`affiche`** » place les nœuds dans l'ordre alphabétique sur un cercle en les espaçant régulièrement : la représentation graphique obtenue n'est donc pas forcément celle du cours : il faudra regarder si les nœuds sont reliés de la même façon afin de vérifier que les deux représentations correspondent au même graphe.

IV./ Deuxième implémentation des graphes en Python :

Créer une deuxième classe pour implémenter les graphes en utilisant les matrices d'adjacence. On représentera ces matrices d'adjacence par des listes de listes (voir la fiche « Rappels Python » sur les listes).

2025
Bruno PEYROT