TD : Les graphes

Représenter une situation réelle par un graphe

Exercice 1

Les situations suivantes peuvent être décrits par des graphes :

- ✗ un réseau d'ordinateurs,
- ✗ des maisons avec des boîtes aux lettres,
- ✗ le métro parisien,
- ✗ une ville, dans laquelle il n'y a plus ni radio ni télé, et dans laquelle se propage par SMS l'information de l'arrivée d'un tsunami,
- ✗ une population au sein de laquelle se propage une épidémie.

1. Pour chaque situation, **indiquer** à quoi correspondent les sommets et les arrêtes.
2. **Décrire** à quoi correspond le plus court chemin entre deux sommets.

Exercice 2

1. **Décrire** un réseau social comme un graphe en indiquant ce que représentent les sommets et les arêtes. **Justifier** que ce graphe n'est pas forcément orienté.
2. **Indiquer** les liens sociaux correspondant à deux sommets adjacents

Analyser un graphe

Exercice 3

Imaginer un réseau social où chaque personne a dix amis,

1. **Déterminer** combien de personnes au maximum un message diffusé aux amis des amis des amis, etc. peut-il atteindre en une heure, si le délai entre la réception et le rediffusion d'un message est de cinq minutes.
2. **Interpréter** ce résultat par rapport au risque de propagation d'une rumeur sur un réseau social.

Exercice 4

Sur le plus grand réseau social du monde, il y a environ un milliard de personnes connectées chacune à cent autres personnes environ. Un chemin de deux arêtes – ami d'ami – connecte donc une personne à environ 100×100 = 10 000 autres personnes si on néglige les amis communs et, disons, à environ 100×50 = 5 000 si l'on tient compte des amis communs.
Un chemin de trois arêtes – ami d'ami d'ami – connecte une personne à environ 100 × 50×50 = 250 000 autres personnes.

1. En supposant grossièrement qu'on ne gagne ainsi que la moitié de nouveaux contacts à chaque arête, **déterminer** à combien de personnes environ est-on connecté avec un chemin de six arêtes.
2. Dans un tel réseau social, **déterminer** la distance maximale entre deux personnes.

Il se trouve que ce calcul approximatif donne un résultat très proche de la réalité.

Représenter un graphe par une matrice d'adjacence

Considérons un réseau social dans lequel les relations entre individus sont des relations symétriques : si *A* est ami avec *B*, alors *B* est aussi ami avec *A*.
On peut alors construire un graphe dans lequel les nœuds sont les individus et les arêtes représentent les relations d'amitié.
On considère aussi que ce réseau est un mini réseau qui contient 8 personnes : Alban, Béatrice, Charles, Déborah, Éric, Fatima, Gérald et Hélène dont les relations qui lient d'amitié ces membre sont :

- Alban est ami avec Béatrice, Déborah, Éric et Fatima.
- Béatrice est amie avec Alban, Charles, Déborah, Éric et Gérald.
- Charles est ami avec Béatrice, Déborah et Hélène.
- Déborah est amie avec Alban, Béatrice, Charles et Gérald.
- Éric est ami avec Alban et Béatrice.
- Fatima est amie avec Alban, Gérald et Hélène.
- Gérald est ami avec Béatrice, Déborah, Fatima et Hélène.
- Hélène est amie avec Charles, Fatima et Gérald.

1. **Illustrer** par un graphe ce réseau social.
2. **Déterminer** la matrice d'adjacence de ce réseau

La classe GrapheM suivante permet de définir un graphe grâce à sa table d'adjacence.

```
class GrapheM :
    '''Graphe represente par sa matrice d'adjacence,
    où les sommets sont les entiers o, 1, ..., n-1'''
    def __init__(self, n) :
          self.__n = n
          self.__adj = [[False] * n for _ in range(n)]

    def ajouter_arete(self, s1, s2) :
          '''Ajoute un arete entre les sommets s1 et s2'''
          self.__adj[s1][s2] = True
          self.__adj[s2][s1] = True

    def arete(self, s1, s2) :
          '''Indique la presence d'une arete ou non entre s1 et s2'''
          return self.__adj[s1][s2]

    def voisins(self, s) :
          '''Renvoie la liste des voisins de s'''
          v =[]
          for i in range(self.__n) :
                if self.__adj[s][i] :
                v.append(i)
          return v
```

3. **Implémenter** ce graphe avec la classe GrapheM.
4. **Modifier** la classe GrapheM afin d'ajouter les méthode suivantes :

➔ Méthode **`Afficher()`** pour afficher le graphe sous la forme suivante :

0 → 1 3
1 → 0 3 4

➔ Méthode **`supprimer_arete(s1,s2)`** pour supprimer l'arête entre les sommets s1 et s2

➔ Méthode **`degre(s)`** qui donne le nombre d'arêtes issus du sommet s

➔ Méthode **`nb_aretes()`** qui donne le nombre d'arêtes du graphe

➔ Méthode **matrix_to_list()** qui renvoie la liste des voisins à partir de la matrice d'adjacence définie en attribut.
Le graphe renvoyé sera codé sous la forme d'un dictionnaire, les clés étant les sommets, numérotés de 0 à n-1 et les valeurs associées sont les listes des successeurs du sommet.

Représentation d'un graphe par une liste des successeurs

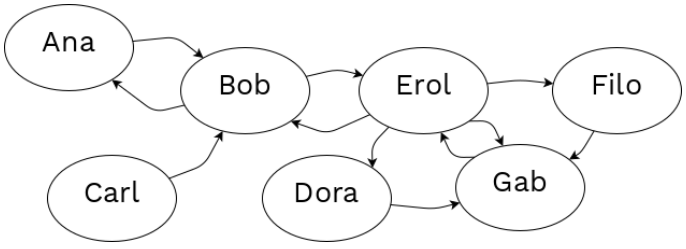
La classe Graphe_oriente_ls suivante permet de définir un graphe grâce à une liste des successeurs.

```
class Graphe_oriente_ls :
'''Graphe decrit par un dictionnaire d'adjacence'''
    def __init__(self) :
        self.__adj = {}

    def ajouter_sommet(self,s) :
        if s not in self.__adj : self.__adj[s]=[]

    def ajouter_arc(self, s1, s2) :
        self.ajouter_sommet(s1)
        self.ajouter_sommet(s2)
        self.__adj[s1].append(s2)

    def sommets(self) :
        return list(self.__adj.keys())

    def voisins(self, s) :
        return self.__adj[s]
```

Programme disponible sur .\donnee\NSI\graphes\graphe_oriente_ls.py

1. **Ajouter** à la classe, une méthode **predecesseurs()** qui renvoie un dictionnaire dont les clés sont les sommets et les valeurs associées sont les listes des prédécesseurs du sommet.
2. **Ajouter** à la classe, une méthode **arc_existe(s1, s2)** qui renvoie vrai si un arc est défini entre s1 et s2

On considère, dans cet exercice, le graphe orienté figure 1, représentant les personnes suivies par d'autres personnes sur un réseau social.



3. **Définir** le dictionnaire correspondant au graphe proposé ci-dessus.

Les arêtes de ce graphe peuvent être représentées par un tuple de deux chaînes de caractères. La première chaîne de caractère représentant le sommet de départ et la

deuxième le sommet d'arrivée de l'arc. Pour exemple le tuple suivant représente l'arc repéré sur la figure 1 liant Ana à Bob :

('Ana', 'Bob')

La liste de tous les arcs de ce graphe est la suivante :

```
arcs = [('Ana', 'Bob'), ('Bob', 'Ana'), ('Carl', 'Bob'), ('Erol', 'Bob'), ('Bob', 'Erol'), ('Erol', 'Filo'), ('Erol', 'Gab'), ('Filo', 'Gab'), ('Gab', 'Erol'), ('Erol', 'Dora'), ('Dora', 'Gab')]
```

4. **Créer** une fonction **`charge_arcs(graphe, arcs)`** qui prend pour argument une instance de la classe graphe_oriente_ls et la liste des arcs. Cette fonction doit insérer dans l'objet de la classe graphe_oriente_ls, les sommets et les arcs définis dans la liste arcs. **Tester** et **valider** cette fonction

5. **Ecrire** une fonction **`amis_d_amis(graphe, pers)`** qui prend en argument une instance de la classe Graphe_oriente_la et une chaîne de caractère au nom de la personne et qui renvoie la liste des amis des amis de pers, à l'exclusion d'elle même et sans doublon.