TP de Python numéro 9 : Graphes (premier volet)

*Semaine du **jeudi 7 mars**.*

Dans ce petit TP, on étudie les deux représentations classiques des graphes en informatique (par matrice d'adjacence ou par liste d'adjacence) et on programme les fonctions de conversion d'une représentation vers l'autre. Ensuite, on utilise les résultats du programme de mathématiques liés aux chaînes, aux chemins et à la connexité pour écrire des algorithmes classiques sur ces notions.

Dans tout ce TP, les graphes seront orientés ou non, et auront pour ensemble de sommets un ensemble de la forme $[\![0, n-1]\!]$ (où $n \in \mathbb{N}^*$ est l'ordre du graphe) de sorte que ces sommets sont naturellement numérotés, numérotation qu'on utilisera pour parler de *la* matrice d'adjacence d'un graphe considéré.

Cette numérotation à partir de 0 est plus simplement compatible avec la numérotation de `Python`.

Remarque : tout ce qui est expliqué dans ce TP est autant valide pour les graphes orientés que pour les graphes non orientés, mais pour utiliser ces résultats, il faut tout de même préciser le cadre (orienté ou non) dans lequel on se place.

I. Représentations informatiques des graphes

Il existe deux moyens classiques de représenter un graphe en informatique.

1. Représentation par matrice d'adjacence

On peut tout d'abord représenter un graphe en donnant sa matrice d'adjacence. Par exemple, Considérons les graphes $\mathcal{G}$ et $\mathcal{H}$ représentés ci-dessous :

Notons $M_{\mathcal{G}}$ et $M_{\mathcal{H}}$ leurs matrices d'adjacence. Alors :

$$M_{\mathcal{G}} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \text{ et } M_{\mathcal{H}} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

et on peut reconstruire ces graphes à partir de ces matrices. On peut donc représenter $\mathcal{G}$ et $\mathcal{H}$ en Python en donnant leurs matrices d'adjacences `MG` et `MH` données par :

```
MG=[[0,1,1,0],[1,0,0,1],[1,0,0,1],[0,1,1,0]]
MG=np.array(MG)
MH=[[0,1,1,0],[0,0,0,0],[0,0,0,1],[0,0,0,0]]
MH=np.array(MH)
```

Exercice 1

1. Représenter le graphe (non orienté) complet K_5 d'ordre 5. Donner sa matrice d'adjacence. Définir cette matrice `K5` en Python, sans rentrer ses coefficients à la main.
2. Écrire le code d'une fonction python d'entête `def MatriceK(n):` prenant en entrée un entier naturel non nul `n` et renvoyant en sortie la matrice d'adjacence du graphe complet d'ordre `n`.

2. Représentation par liste d'adjacence

Plutôt que d'utiliser leurs matrices d'adjacence, on peut représenter des graphes en donnant leur *liste d'adjacences*.

Soit s un sommet d'un graphe $\mathcal{G}$. La liste des adjacences de s est la liste, qu'on notera $V(s)$, formée par les sommets

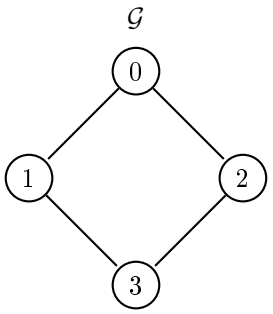
- adjacents à s si $\mathcal{G}$ est non orienté,
- qui sont le but d'une arête d'origine s si $\mathcal{G}$ est orienté.

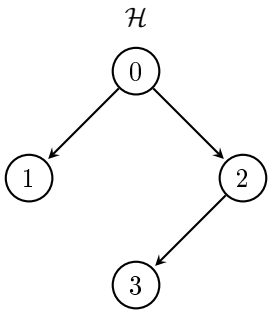
Pour représenter un graphe $\mathcal{G}$ d'ordre n, on donne alors la liste

$$V = [V(0), V(1), \ldots, V(n-1)]$$

de ces liste des adjacences. Cette liste V est appelée *la liste d'adjacence* du graphe $\mathcal{G}$. Il est clair qu'on peut retrouver un graphe $\mathcal{G}$ à partir de sa liste d'adjacence (on peut reconstituer son ordre et toutes ses arêtes).

Par exemple, reprenons les exemples précédents.





- **Dans le graphe** $\mathcal{G}$ non orienté, les listes des adjacences `V0,V1` des sommets `0` et `1` sont données par

  ```
  V0=[1,2]
  V1=[0,3]
  ```

  et la liste d'adjacence de $\mathcal{G}$ est la liste `V` donnée par :
  `V=[[1,2],[0,3],[0,3],[1,2]].`

- **Dans le graphe** $\mathcal{H}$ orienté, les listes des adjacences `V0,V1` des sommets `0` et `1` sont données par

  ```
  V0=[1,2]
  V1=[]
  ```

  et la liste d'adjacence de $\mathcal{G}$ est la liste `V` donnée par :
  `V=[[1,2],[],[3],[]].`

Exercice 2

1. Déterminer la liste d'adjacence du graphe complet (non orienté) K_4 d'ordre 4.
2. A l'aide d'une boucle, définir en python la liste des adjacences du sommet 0.
3. Sans la renter à la main, définir en python la liste d'adjacence de K_4.
4. Écrire le code d'une fonction python d'entête `def ListeAdjacenceK(n):` prenant en entrée un entier naturel non nul `n` et renvoyant en sortie la liste d'adjacence du graphe complet d'ordre `n`.

Exercice 3

1. Écrire une fonction Python d'entête `def estArete(V,i,j):` prenant en entrée la liste d'adjacence d'un graphe (orienté ou non), ainsi que deux sommets `i` et `j` de ce graphe, et renvoyant en sortie `True` si ce graphe comporte une arête de `i` vers `j`, et `False` sinon.
2. Écrire une fonction Python d'entête `def AfficheDescriptionNO(V):` prenant en entrée la liste d'adjacence d'un graphe `G` non orienté, et affichant (sans sortie) de manière lisible l'ordre de `G` et la liste des degrés des sommets de `G`.
3. Écrire une fonction Python d'entête `def AfficheDescriptionO(V):` prenant en entrée la liste d'adjacence d'un graphe `G` orienté, et affichant (sans sortie) de manière lisible l'ordre de `G`, la liste des degrés sortants des sommets de `G`, et la liste des degrés entrants des sommets de `G`.

3. Fonctions de conversion

On dispose de deux manières de représenter un graphe : par sa matrice d'adjacence, ou par sa liste d'adjacence. Dans tous les cas, l'information retenue permet de reconstruire le graphe. Il est donc naturel de demander la possibilité de passer d'une représentation à l'autre.

Exercice 4

1. Écrire la matrice d'adjacence du graphe orienté ayant la liste d'adjacence : `[[1,2,3],[0,2],[0,4],[2,4],[0,1,2,3]]`
2. Écrire une fonction python d'entête `def conversion_Liste_vers_Matrice(V):` prenant en entrée une liste `V` qui est la liste d'adjacence d'un graphe (orienté ou non), et renvoyant en sortie la matrice d'adjacence de ce graphe.
3. Tester votre fonction à l'aide des fonctions `ListeAdjacenceK` et `MatriceK` des exercices précédents.

Pour la 2., on pourra partir du code incomplet suivant :

```
def conversion_Liste_vers_Matrice(V):
        ordre =... #ordre du graphe considéré
        M=np.zeros((ordre,ordre)) #matrice d'adjacence à remplir
        for i in range(ordre): #pour chaque sommet i :
                #remplissage de la i ième ligne de M,
                #en utilisant la liste V[i]
                (...)
        return(M)
```

Exercice 5

1. Écrire la liste d'adjacence du graphe non orienté ayant la matrice d'adjacence :

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

2. Écrire une fonction python d'entête `def conversion_Matrice_vers_Liste(M):` prenant en entrée une matrice `M` qui est la matrice d'adjacence d'un graphe (orienté ou non), et renvoyant en sortie la liste d'adjacence de ce graphe.
3. Tester votre fonction à l'aide des fonctions `ListeAdjacenceK` et `MatriceK` des exercices précédents.

II. Matrice d'adjacence, chaînes et chemins, connexité

Pour cette partie, on représentera un graphe à l'aide de sa matrice d'adjacence. On utilisera les commandes Python au programme sur les matrices, vues lors du TP précédent.

Exercice 6

Dans cet exercice, `M` doit pouvoir représenter la matrice d'adjacence d'un graphe $\mathcal{G}$ orienté.

1. Écrire une fonction python d'entête `def DegSortants(M):` renvoyant en sortie la liste (dans l'ordre de ces sommets) des degrés sortants des sommets de $\mathcal{G}$.
2. Coder de même une fonction d'entête `def DegEntrant(M):` renvoyant la liste des degrés entrants des sommets de $\mathcal{G}$.

Exercice 7

Les fonctions demandées dans cet exercices doivent fonctionner indifféremment pour un graphe orienté ou non orienté.

1. Écrire le code d'une fonction Python d'entête `def NombreChemins(M,i,j):` prenant en entrée la matrice d'adjacence `M` d'un graphe $\mathcal{G}$ et les numéros `i` et `j` de deux de ses sommets, et renvoyant en sortie le nombre de chaînes (ou chemins si $\mathcal{G}$ est orienté) du sommet `i` vers le sommet `j`.

Un problème classique, sur lequel nous reviendrons, est la détermination du plus court chemin (ou de la plus courte chaîne) entre deux sommets d'un graphe.

2. Écrire le code d'une fonction Python d'entête `def distance(M,i,j):` prenant en entrée la matrice d'adjacence `M` d'un graphe $\mathcal{G}$ et les numéros `i` et `j` de deux de ses sommets, et renvoyant en sortie :

   - la longueur du plus court chemin (ou de la plus courte chaîne dans le cas non orienté) du sommet `i` vers le sommet `j` si un tel chemin (ou une telle chaîne) existe,
   - `False` s'il n'existe pas de chemins (ou de chaîne) de `i` vers `j`

   On pensera au lemme utilisé dans la démonstration du théorème liant la connexité d'un graphe à sa matrice d'adjacence : dans un graphe d'ordre n, s'il existe un chemin (ou une chaîne) d'un sommet i vers un sommet j, alors il existe un chemin de i vers j de longueur au plus $n-1$.

Remarque : le code précédent ne donne pas le plus court chemin (s'il existe) entre deux sommets, juste sa longueur. Ce problème sera abordé à nouveau dans un TP ultérieur.

Exercice 8

Écrire le code d'une fonction Python d'entête `def estConnexe(M):` prenant en entrée la matrice d'adjacence `M` d'un graphe $\mathcal{G}$ et renvoyant en sortie `True` si $\mathcal{G}$ est connexe ("fortement" dans le cas orienté), et `False` sinon.