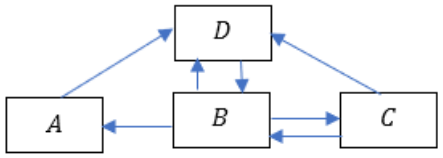
Exercice 1 : Vrai ou Faux

1) Un graphe ne peut pas avoir plus de sommets que d'arrêtes.
2) Une matrice d'adjacence d'un graphe à n sommets contient $2n$ coefficients.
3) Une matrice d'adjacence d'un graphe orienté est symétrique.
4) Le nombre d'arrêtes d'un graphe non orienté à n sommets est la moitié de la somme des degrés des sommets.
5) L'ordre d'un graphe est le nombre d'arcs ou d'arrêtes.
6) Un graphe est connexe si chaque sommet est adjacent à tous les autres
7) Un chemin ou une chaine qui passe deux fois par le même sommet contient un cycle.
8) Un graphe non orienté d'ordre 5 contient au maximum 10 arêtes reliant des sommets distincts

1) Faux, on peut avoir des sommets isolés et donc plus de sommets que d'arrêtes. On peut aussi un graphe d'ordre 2 avec une arrête.
2) Non, il en contient n^2
3) Non, c'est la matrice d'un graphe non orienté qui est symétrique
4) Oui, car en sommant les degrés de chaque sommet, on compte deux fois chaque arrête
5) Non, l'ordre d'un graphe c'est le nombre de sommets
6) Non, si chaque sommet est adjacent à tous les autres, le graphe est complet. Pour un graphe connexe, toute paire de sommets est reliée par un chemin ou chaîne: un graphe complet est connexe mais pas réciproquement
7) Vrai car un cycle est par définition un chemin ou une chaîne qui passe par le même sommet
8) Oui, 5*4/2 s'il est complet

Exercice 2 :

Ecrire la liste et la matrice d'adjacence des successeurs associées au graphe orienté non pondéré ci-dessous :



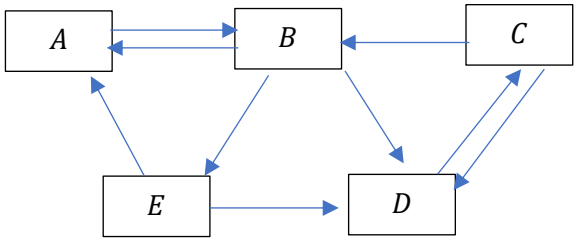
On a ici un graphe orienté dont la matrice d'adjacence est :

$A: D \quad B: A, D, C \quad C: B, D \quad D: B$

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

Exercice 3 :

Ecrire la liste et la matrice de successeurs du graphe orienté non pondéré suivant :



$A : B$

$B : E, D, A$

$C : B, D$

$D : C$

$E : A, D$

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

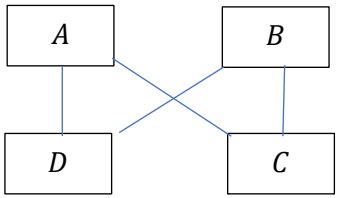
Exercice 4 :

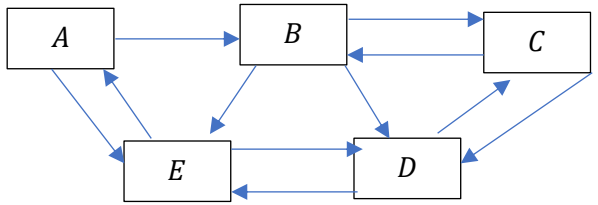
Tracer les graphes non pondérés associés aux matrices d'adjacence suivantes (matrice d'adjacence des successeurs si graphe orienté) :

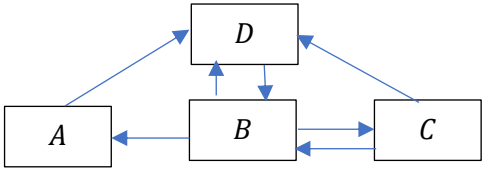
$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

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

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







Exercice 5 :

On considère un graphe non pondéré implémenté à l'aide d'un dictionnaire *dico*.

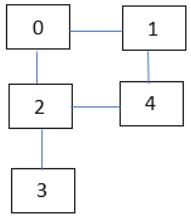
```
dico={"A":["B","E"],
      "B":["A","C","E"],
      "C":["B"],
      "D":[],
      "E":["A","B"]}
```

Ecrire une fonction *deg* prenant pour argument *dico* et une chaîne représentant un des sommets du graphe et qui renvoie son degré.

```
def deg(dic,sommet):
    L=dic[sommet]
    deg=len(L)
    return deg


def deg2(dic,sommet):
    if sommet in dic:
        return len(dic[sommet])
def deg3(dico,s):
    return len(dico[s])
```

Exercice 6 :

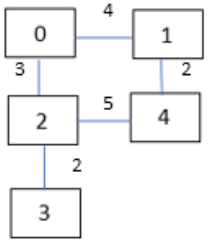
On considère le graphe non orienté et non pondéré dont les sommets sont notés 0,1,2,3,4. On déclare la liste *S* des sommets et la liste *A* des listes d'adjacence de la manière suivante :




```
"""on déclare les sommets"""

S = [k for k in range(0,5)]

"""on déclare les arêtes"""

A= [[0,1],[0,2],[1,4],[2,4],[2,3]]
```

1) Proposer un programme permettant d'obtenir la matrice d'adjacence associée à une liste des listes d'adjacence *A* quelconque écrite suivant le modèle proposé. On pourra utiliser le formalisme des tableaux numpy ou celui des listes.

Le graphe non orienté précédent en maintenant pondéré.



La liste *A* s'écrit alors :

```
A= [[0,1,4],[0,2,3],[1,4,2],[2,4,5],[2,3,2]]
```

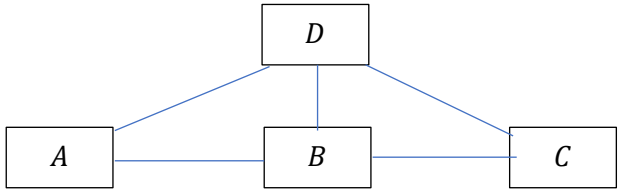
2) Proposer un programme permettant d'obtenir la matrice d'adjacence à partir de *A*. En l'absence de liaison, le poids associé est égal à la valeur maximale du poids du graphe augmenté d'une unité.

```
"""avec un graphe non pondéré et non orienté"""
"""on déclare les sommets"""
S = [k for k in range(0,5)]
"""on déclare les arêtes"""
A= [[0,1],[0,2],[1,4],[2,4],[2,3]]
import numpy as np
"""vesrion numpy"""
def matrice(A):
    n=len(A)
    M=np.zeros((n,n))
    for i in A:
        arrete=i
        p=arrete[0]#prédécesseur
        s=arrete[1]#successeur
        M[p,s]=M[s,p]=1
    return M
print(matrice(A))
"""version listes"""
def matrice2(A):
    n=len(A)
    M=[[0 for i in range(n)] for i in range(n)]#génération
par compréhension efficace pour éviter les effets de bords
    for i in A:
        arrete=i
        p=arrete[0]#prédécesseur
        s=arrete[1]#successeur
        M[p][s]=M[s][p]=1
    return M
print(matrice2(A))
"""pour un graphe pondéré et non orienté"""
A= [[0,1,4],[0,2,3],[1,4,2],[2,4,5],[2,3,2]]
"""version numpy"""
def matrice3(A):
    n=len(A)
    maxi=max([A[i][2] for i in range(n)])
    maxi=maxi+1
    M=np.ones((n,n))*maxi
    for i in A:
        p=i[0]
        s=i[1]
        poids=i[2]
        M[p,s]=M[s,p]=poids
    return M
print(matrice3(A))

def matrice4(A):
```

```
    n=len(A)
    maxi=max([A[i][2] for i in range(n)])
    maxi=maxi+1
    M=[[maxi for i in range(n)] for i in range(n)]#génération
par compréhension efficace pour éviter les effets de bords
    for i in A:
        arrete=i
        p=arrete[0]#prédécesseur
        s=arrete[1]#successeur
        poids=arrete[2]
        M[p][s]=M[s][p]=poids
    return M
print(matrice4(A))
```

Exercice 7 :

On considère le graphe non orienté et non pondéré dessiné ci-dessous :



1) Déclarer, « à la main » et sous python, la définition de la liste *m* qui représente la matrice d'adjacence de ce graphe (0 sans arrête et 1 avec arête).
2) Ecrire une fonction *aretes* qui prend en paramètre une matrice *m* représentant un graphe et renvoie le nombre d'arêtes du graphe (non pondéré et non orienté).
3) Déclarer « à la main » un dictionnaire en Python, où les clés sont les sommets de départ, les valeurs sont les sommets d'arrivées, permettant ainsi de représenter la liste d'adjacence du graphe ci-dessus.
4) Ecrire une fonction *sommets* qui prend en argument un sommet *s* et une liste des listes d'adjacence sous la forme d'un dictionnaire, et renvoie la liste des sommets liés par une arrête au sommet *s*. *None* sera renvoyé si le sommet n'est pas dans le dictionnaire.

On a une matrice symétrique

```
M=[[0,1,0,1],
   [1,0,1,1],
   [0,1,0,1],
   [1,1,1,0]]
def aretes(M):
    """compter les arrêtes c'est compter
    à une facteur 2 près
    la somme des degrés"""
    compteur=0
    n=len(M)
    for i in range(n):
        for j in range(n):
            if M[i][j]!=0:#fonctionne si pondéré
                compteur=compteur +1
    return compteur//2#fonctionne pas si orienté !
print(aretes(M))

def aretes2(m):
    """on peut compter le nombre d'arrête
    en comptant les éléments non nuls
    en dessous de la diagonale"""
    compteur=0
    for i in range (len(m)):
        for j in range(i+1,len(m)):
            if m[i][j]!=0:
                compteur+=1
    return compteur
print(aretes2(M))

dico={"A":["B","D"],
      "B":["A","C","D"],
      "C":["B","D"],
      "D":["A","B","C"]}
def sommets(dico,s):
    if s in dico:
        return dico[s]
    else :
        return None
print(sommets(dico,"A"))
```

Exercice 8 : Conversion

Dans cet exercice, on considère des graphes non pondérés et non orientés. Les sommets sont représentés par des entiers ainsi matrice d'adjacence, liste des listes d'adjacence et dictionnaire des listes d'adjacence sont tels que :

```
#matrice d'adjacence
m=[[0,1,0,1],
   [1,0,1,1],
   [0,1,0,1],
   [1,1,1,0]]
#dictionnaire associée
{0: [1, 3], 1: [0, 2, 3], 2: [1, 3], 3: [0, 1, 2]}
#liste des listes d'adjacence
[[0, [1, 3]], [1, [0, 2, 3]], [2, [1, 3]], [3, [0, 1, 2]]]
```

1) Écrire une fonction mat_to_list qui prend en argument une matrice d'adjacence et qui renvoie la liste des listes d'adjacence associée.

2) Écrire une fonction list_to_mat qui prend en argument une liste des listes d'adjacence et renvoie la matrice associée.

3) Écrire une fonction list_to_dict qui prend en argument une liste d'adjacence et qui renvoie le dictionnaire des listes d'adjacence associée.

4) Écrire une fonction dict_to_mat qui prend en argument un dictionnaire des listes d'adjacence et qui renvoie la matrice d'adjacence associée écrite sous la forme d'une liste de liste.

```
def mat_to_list(m):
    n=len(m)
    L=[[i,[]] for i in range(n)]
    for i in range(n):
        for j in range(i+1,n):
            if m[i][j]==1:
                L[i][1].append(j)
                L[j][1].append(i)
    return L
def mat_to_liste2(M):
    n=len(m)
    L=[[i,[]] for i in range(n)]
    for i in range(n):
        for j in range(n):
            if m[i][j]==1:
                L[i][1].append(j)
    return L
def list_to_mat1(liste):
    n=len(liste)
    m=[[0 for i in range (n)] for i in range(n)]
    for i in range(n):
        liste_voisin=liste[i][1]
        for j in liste_voisin :
            m[i][j]=1
    return m
import numpy as np
def list_to_mat2(liste):
    n=len(liste)
    m=np.zeros((n,n))
    for i in range(n):
        liste_voisin=liste[i][1]
        for j in liste_voisin :
            m[i][j]=1
    return m
def list_to_dic(liste):
    dico={}
    for i in range(len(liste)) :
        dico[i]=liste[i][1]
    return dico
def dict_to_mat(dico):
    n=len(dico)
    m=[[0 for i in range(n)] for i in range(n)]
    for i in dico :
        for j in dico[i]:
            m[i][j]=1
    return m
```

Exercice 9 : Conversion (suite)

En réutilisant les fonctions précédentes, écrire une fonction :

1) dict_to_list qui prend en argument une dictionnaire d'adjacence et qui renvoie le liste d'adjacence associée.
2) mat_to_dict qui prend en argument un matrice d'adjacence et qui renvoie la dictionnaire d'adjacence associée.

```
def dict_to_liste(dico):
    M=dico_to_matrice(dico)
    return mat_to_liste(M)
def mat_to_dict(M):
    L=mat_to_liste(M)
    return liste_to_dict(L)
```

Exercice 10 :

On considère dans cet exercice des graphes non orientés et non pondérés. Ecrire une fonction f qui prend en paramètres une matrice d'adjacence et un dictionnaire d (tous deux associés au même graphe) et qui renvoie la liste des arêtes. Le dictionnaire contient des clés qui sont des indices et des valeurs qui sont les noms des sommets.

Par exemple :

```
d={0: "A", 1:"B",2:"C", 3: "D"}
matrice=[[0,1,1,0],
         [1,0,0,0],
         [1,0,0,0],
         [0,0,0,0]]
```

On renvoie : [['A', 'B'], ['A', 'C']]

```
def f9(matrice,dic):
    liste=[]
    for i in range(len(matrice)):
        for j in range(i+1,len(matrice)):
            if matrice[i][j]==1:
                liste.append([dic[i],dic[j]])
    return liste
```

Rq : on peut aussi utiliser la liste des listes d'adjacence :

```
def f9_bis(matrice,dic):
    liste=[]
    for i in range(len(dic)):
        sous_liste=[dic[i]]
        voisin=[]
        for j in range(len(matrice[i])):
            if matrice[i][j]==1:
                voisin.append(dic[j])
        sous_liste.append(voisin)
        liste.append(sous_liste)
    return liste
```

Exercice 11 :

On dispose d'un graphe non orienté et non pondéré dont le dictionnaire des listes d'adjacences s'écrit :

```
g={"A":["B","D"],"B":["A","C","D"],"C":["B"],"D":["A","B"]}
```

1) La fonction *conversion*1 prend en paramètre un tel dictionnaire et renvoie la matrice d'adjacence correspondante :

```
m =[[0,1,0,1],
    [1,0,1,1],
    [0,1,0,0],
    [1,1,0,0]]
```

Ecrire *conversion*1

2) Compléter la fonction *conversion*2 afin d'obtenir la conversion inverse à la question précédente.

```
def conversion2(m):
    sommets={}
    n=len(m)
    for i in range(n):
        sommets[i]=chr(65+i)#conversion chr(65)=A,chr(66)=B..
    g={}
    ...................
        ...................
        ...................
            ......................
                ....................
    return g
```

Corrigé

```
def conversion1(g):
    sommets={}
    n=0
    for s in g:
        sommets[s]=n#{"B":0,"C":1,"A":2,"D":3} car python3.5
ne conserve pas l'ordre des clés
        n=n+1
    mat = [[0 for i in range(n)] for j in range(n)]
    for s in sommets :
        i=sommets[s]
        for adj in g[s]:
            j=sommets[adj]
            mat[i][j]=1
    return mat

print(conversion1(g))

def conversion2(m):
    sommets={}
    n=len(m)
    for i in range(n):
        sommets[i]=chr(65+i)
    g={}
    for i in range(n):
        g[sommets[i]]=[]
        for j in range(n):
            if m[i][j]==1:
                g[sommets[i]].append(sommets[j])
    return g
```