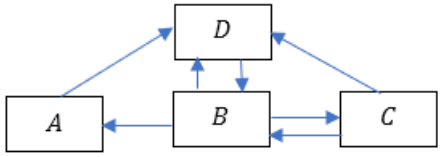
Exercice 1 : Vrai ou Faux

1) Un graphe ne peut pas avoir plus de sommet 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 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 sommet sont reliées par un chemin : un graphe complet est connexe mais pas réciproquement
7) Vrai car un cycle est par définition un chemin 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



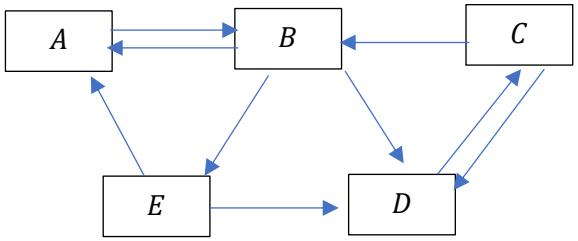
On a ici un graphe orienté dont la matrice d'adjacence est :

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

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

Exercice 3 :

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



$A : B$

$B : E, D, A$

$C : B, D$

$D : C$

$E : A, D$

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

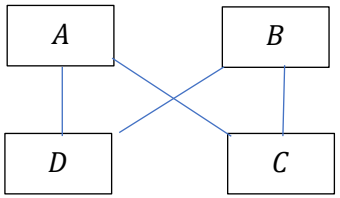
Exercice 4 :

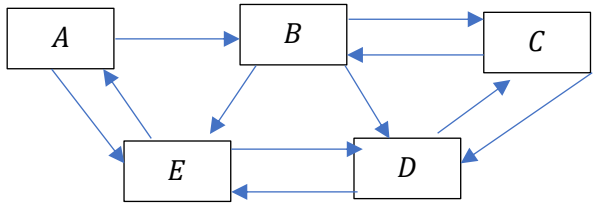
Tracer les graphes non pondérés associés aux matrices d'adjacence suivantes :

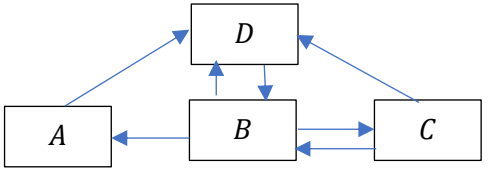
$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

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

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







Exercice 5 :

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

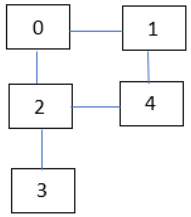
```
dico={"A":["B","E"],
      "B":["A","C","E"],
      "C":["B"],
      "D":[],
      "E":["A","B"]}
```

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])
```

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 d'adjacence de la manière suivante :



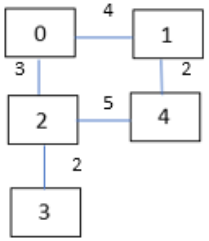

```
"""on déclare les sommets"""

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

"""on déclare les arêtes"""

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

1) Ecrire la matrice d'adjacence.

2) Proposer un programme permettant d'obtenir la matrice d'adjacence pour une liste d'adjacence *A* quelconque écrite suivant le modèle proposé.

Le graphe précédent en maintenant pondéré

La liste *A* s'écrit alors :



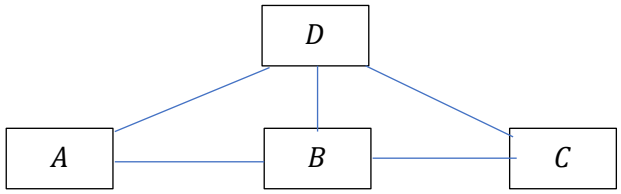

```
A= [[0,1,4],[0,2,3],[1,4,2],[2,4,5],[2,3,2]]
```

3) 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.

```
import numpy as np

"""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]]
"""matrice avec tableau numpy"""
def matrice1(A):
    tab=np.zeros((len(A),len(A)))
    for i in range(len(A)):
        [i,j]=A[i]
        tab[A[i][0],A[i][1]]=tab[A[i][1],A[i][0]]=1
    return tab
print(matrice1(A))

def matrice1_bis(A):
    tab=np.zeros((len(A),len(A)))
    for a in A:
        [i,j]=a
        tab[i,j]=tab[j,i]=1
    return tab
print(matrice1_bis(A))

"""matrice avec liste"""
def matrice2(A):
    tab=[[0 for i in range(len(A))] for i in range(len(A))]
    for i in range(len(A)):
            tab[A[i][0]][A[i][1]]=tab[A[i][1]][A[i][0]]=1
    return tab
print(matrice2(A))

"""avec graphe pondéré"""
A= [[0,1,4],[0,2,3],[1,4,2],[2,4,5],[2,3,2]]
def matrice3(A):
    maxi=max([A[i][2] for i in range(len(A))])
    maxi=maxi+1
    tab=[[maxi for i in range(len(A))] for i in
range(len(A))]
    for i in range(len(A)):

tab[A[i][0]][A[i][1]]=tab[A[i][1]][A[i][0]]=A[i][2]
    return tab
print(matrice3(A))
```

Exercice 7 :

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



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

On a une matrice symétrique

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

```
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"""
    s_degre=0
    for i in range(len(m)):
        for j in range(len(m[i])):
            s_degre=s_degre+m[i][j]
    return s_degre//2

def aretes2(m):
    """on peut compter le nombre d'arrête
    en comptant les éélments 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]==1:
                compteur+=1
    return compteur

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


def sommets(dico,s):
    if s in dico:
        return dico[s]
    else :
        return None
```

Exercice 8 :

On considère dans cet exercice des graphes non orientés et non pondérés. Une variable g est définie pour représenter un graphe. Un sommet est représenté par un caractère comme "A"

1) Ecrire une fonction f_1 qui prend en paramètres un graphe g représenté par un dictionnaire des listes d'adjacence et un sommet s et qui ajoute ce sommet au graphe en tant que sommet isolé.
2) Ecrire une fonction f_2 qui prend en arguments un graphe g représenté par une liste des listes d'adjacence et un point s et ajoute le point au graphe en tant que sommet isolé.
3) Ecrire une fonction f_3 qui prend en paramètre un graphe g représenté par une matrice d'adjacence (liste de listes) et complète la liste pour ajouter un sommet isolé.

```
def f1(dic,s):
    """fonction qui ajoute le sommet s
    isolé au dictionnaire"""
    dic[s]=[]

def f2(liste,s):
    """fonction qui ajoute le sommet s
    isolé à liste"""
    liste.append([s,[]])

def f3(matrice):
    """fonction qui rajoute le sommet s
    isolé dans la matrice d'adjacence"""
    for i in matrice:
        i.append(0)#il faut rajouter un 0 à chaque ligne
    n=len(matrice)
    liste=[0]*(n+1)
    matrice+=[liste]
```

Exercice 9 :

Les hypothèses de travail restent celles de l'exercice précédent.

1) Ecrire une fonction f_4 qui prend en arguments un graphe g représenté par un dictionnaire des listes d'adjacence et une arête a et ajoute l'arête au graphe. Une arête est représentée par une liste de deux sommets $[s_1, s_2]$. Les sommets s_1 et s_2 appartiennent déjà au graphe.
2) Ecrire une fonction f_5 qui prend en paramètres un graphe g représenté par une liste des listes d'adjacence et une arête a et ajoute l'arête a au graphe (a n'était pas encore présente). Une arête est définie comme dans la question précédente.
3) Ecrire une fonction f_6 qui prend en paramètre un graphe g représenté par une matrice d'adjacence et une arête a et complète la matrice pour ajouter l'arête au graphe (pas d'ajout de sommets). Une arête est représentée par une liste contenant les deux indices des extrémités de l'arête.

```python
def f4(dic,a):
    dic[a[0]].append(a[1])
    dic[a[1]].append(a[0])


def f4bis(dic,a):
    s1,s2=a[0],a[1]
    liste1,liste2=dic[s1],dic[s2]
    liste1.append(s2)
    liste2.append(s1)
    dic[s1],dic[s2]=liste1,liste2


def f5(liste,a):
    for i in range(len(liste)):
        if liste[i][0]==a[0]:
            liste[i][1].append(a[1])
        if liste[i][0]==a[1]:
            liste[i][1].append(a[0])


def f6(matrice,a):
    i,j=a[0],a[1]
    matrice[i][j]=matrice[j][i]=1
```

Exercice 10 :

Les conditions sont celles des deux exercices précédents.

1) Ecrire une fonction f_7 qui prend en paramètre un graphe g représenté par un dictionnaire des listes d'adjacence et renvoie la liste des sommets.
2) Ecrire une fonction f_8 qui prend en paramètre un graphe g représenté par une liste des listes d'adjacence et renvoie une liste des sommets.

```
def f7(dico):
    liste_sommet=[]
    for cle in dico:
        liste_sommet.append(cle)
    return liste_sommet


def f8(liste):
    liste_sommet=[]
    for i in range(len(liste)):
        liste_sommet.append(liste[i][0])
    return liste_sommet

def f8bis(liste):
    liste_sommet=[]
    for i in liste:
        liste_sommet.append(i[0])
    return liste_sommet
```

Exercice 11 :

Les conditions sont celle des trois derniers exercices.

Ecrire une fonction f_9 qui prend en paramètres une matrice d'adjacence et un dictionnaire d (tout 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 :

```
dic={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]]
```

```
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 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 12 :

On dispose d'un graphe non orienté et non pondéré dont les listes d'adjacences s'écrivent :

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

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

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

Compléter le programme ci-dessous (python3.6 !) :

```
def conversion1(g):
    sommets={}
    n=0
    for s in g:
        sommets[s]=n
        n=n+1
    mat = [[0 for i in range(n)] for j in range(n)]
    ......................
        ..................
        ................
            ................
            ..............
    return mat
```

2) La fonction *conversion*2 prend en paramètre une matrice d'adjacence et renvoie le graphe correspondant. On doit donc trouver le graphe *g* à partir de *m*

```
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
```