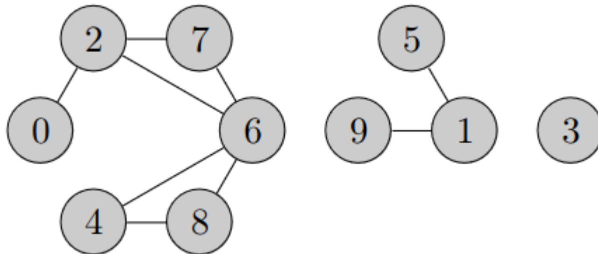
P16 - Théorie des graphes I : représenter un graphe

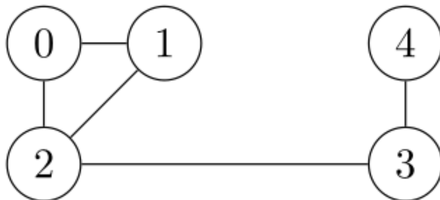
Correction

1. Matrice d'adjacence

On considère les graphes non orientés suivants (G_1 et G_2).



G_1 :



G_2 :

Exercice 1

1. Définir leurs matrices d'adjacence A_1 et A_2 dans le format np.array.

In [15]:
```
import numpy as np
A1 = np.array([[0,0,1,0,0,0,0,0,0,0], [0,0,0,0,0,1,0,0,0,1], [1,0,0,0,0,0,1,1,0,0], [0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,1,0,1,0], [0,1,0,0,0,0,0,0,0,0], [0,0,1,0,1,0,0,1,1,0], [0,0,1,0,0,0,1,0,0,0],
              [0,0,0,0,1,0,1,0,0,0], [0,1,0,0,0,0,0,0,0,0]])
A1
```

Out[15]:
```
array([[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 0, 1, 1, 0],
       [0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]])
```

In [17]:
```
A2 = np.array([[0,1,1,0,0], [1,0,1,0,0], [1,1,0,1,0], [0,0,1,0,1], [0,0,0,1,0]])
A2
```

Out[17]:
```
array([[0, 1, 1, 0, 0],
       [1, 0, 1, 0, 0],
       [1, 1, 0, 1, 0],
       [0, 0, 1, 0, 1],
       [0, 0, 0, 1, 0]])
```

2. Écrire une fonction qui renvoie la liste des degrés de chaque sommet d'un graphe donné par une matrice A.

In [19…
```
def degre(A):
    n = len(A) # nombre de sommets
    # degré du sommet i = nombre de 1 sur la ligne i = somme de la ligne i
    D = [0]*n
    for i in range(n):
        D[i] = np.sum(A[i,:])
    return D
degre(A1)
```

Out[19]: [1, 2, 3, 0, 2, 1, 4, 2, 2, 1]

In [18…
```
degre(A2)
```

Out[18]: [2, 2, 3, 2, 1]

In [6]:
```
def degre(A):
    return np.sum(A,0) # somme sur chaque ligne (paramètre 0, pour les colonnes c'est 1)
degre(A1)
```

Out[6]: array([1, 2, 3, 0, 2, 1, 4, 2, 2, 1])

In [7]:
```
def degre(A):
    return np.sum(A,1) # somme sur les colonnes, ça doit donner la même chose
degre(A1)
```

Out[7]: array([1, 2, 3, 0, 2, 1, 4, 2, 2, 1])

In [10…
```
#version sans np.sum
def degre(A):
    n = len(A) # nombre de sommets
    D = [0]*n
    for i in range(n):
        d = 0
        for j in range(n):
            d = d + A[i,j]
        D[i] = d
    return D
degre(A1)
```

Out[10]: [1, 2, 3, 0, 2, 1, 4, 2, 2, 1]

3. Écrire une fonction qui renvoie le nombre de chemin de longeur k entre les sommets i et j.

In [20…
```
import numpy.linalg as al

def nombre_chemins(A,k,i,j):
    # coefficient (i,j) de A**k
    # la puissance s'obtient avec al.matrix_power
    Ak = al.matrix_power(A,k)
    return Ak[i,j]

nombre_chemins(A1,10,0,6)
```

Out[20]: 1609

4. Écrire une fonction qui renvoie True si tous les coefficients d'une matrice carrée M sont strictement positifs et False sinon.

In [21…
```
def positive(M):
    n = len(M)
    for i in range(n):
        for j in range(n):
            if M[i,j] <= 0:
                return False
    return True

positive(A1)
```

Out[21]: False

In [13…
```
positive(np.array([[1,2,3], [4,5,6], [1,2,3]]))
```

Out[13]: True

5. En déduire, à l'aide de la propriété du cours, une fonction qui renvoie True si le graphe de matrice A est connexe, et False sinon.

In [22…
```
def connexe(A):
    n = len(A)
    S = al.matrix_power(A,0)
    for k in range(1,n):
        S = S + al.matrix_power(A,k)
    return positive(S)

connexe(A1)
```

Out[22]:
```
False
```

In [23…
```
connexe(A2)
```

Out[23]:
```
True
```

6. Écrire une fonction voisins(A,i) qui, étant donné un sommet i dans un graphe de matrice A, renvoie la liste des voisins de ce sommets (les sommets qui lui sont adjacents).

In [24…
```
def voisins(A,i):
    # on liste la position des >0 dans la ligne i de A
    n = len(A)
    L = []
    for j in range(n):
        if A[i,j] > 0:
            L.append(j)
    return L

voisins(A1,6)
```

Out[24]:
```
[2, 4, 7, 8]
```

In [25…
```
voisins(A1,3)
```

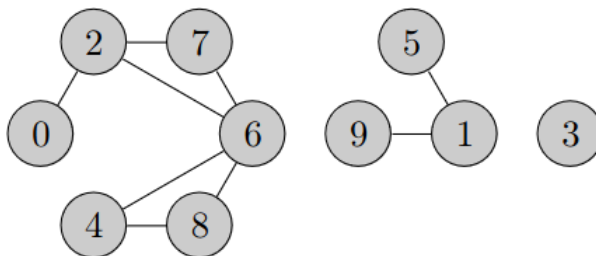
Out[25]:
```
[]
```

2. Liste d'adjacence

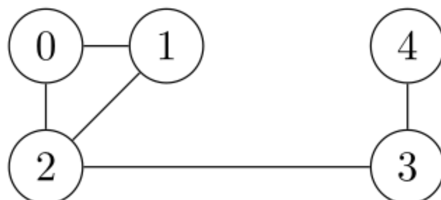
Le problème de la matrice d'adjacence est qu'elle contient beaucoup de données inutiles (on doit souvent saisir un grand nombre de 0) et devient donc rapidement peu pratique.

On va voir une autre façon de représenter les graphes : les listes d'adjacence.

On donne une liste qui contient, pour chaque sommet, la liste de ses voisins.



G_1 :



G_2 :

Exercice 2

1. Représenter les graphes G_1 et G_2 par leurs listes d'adjacence G1 et G2.

In [26…
```
G1 = [[2], [5,9],[0,6,7], [], [6,8], [1], [2,4,7,8], [2,6], [4,6], [1]]
G2 = [[1,2], [0], [0,1,3], [2,4], [3]]
```

C'est plus efficace, non ?

2. Vérifier que votre fonction voisins de l'exercice 1 donne la même chose.

In [29…
```
def liste_adjacence(A):
    n = len(A)
    G = [voisins(A, i) for  i in range(n)]
    return G

liste_adjacence(A1)
```

Out[29]:
```
[[2], [5, 9], [0, 6, 7], [], [6, 8], [1], [2, 4, 7, 8], [2, 6], [4, 6], [1]]
```

3. Réécrire une fonction renvoyant le degré des sommets, en utilisant les listes d'adjacence.

In [31…
```
def degre2(G):
    n = len(G) # nombre de sommets
    return [len(G[i]) for i in range(n)]

degre2(G1)
```

Out[31]:
```
[1, 2, 3, 0, 2, 1, 4, 2, 2, 1]
```

In [33…
```
def degre2(G):
    n = len(G) # nombre de sommets
    L = []
    for i in range(n):
        d = len(G[i]) # degré = nombre de voisins
        L.append(d)
    return L

degre2(G1)
```

Out[33]:
```
[1, 2, 3, 0, 2, 1, 4, 2, 2, 1]
```