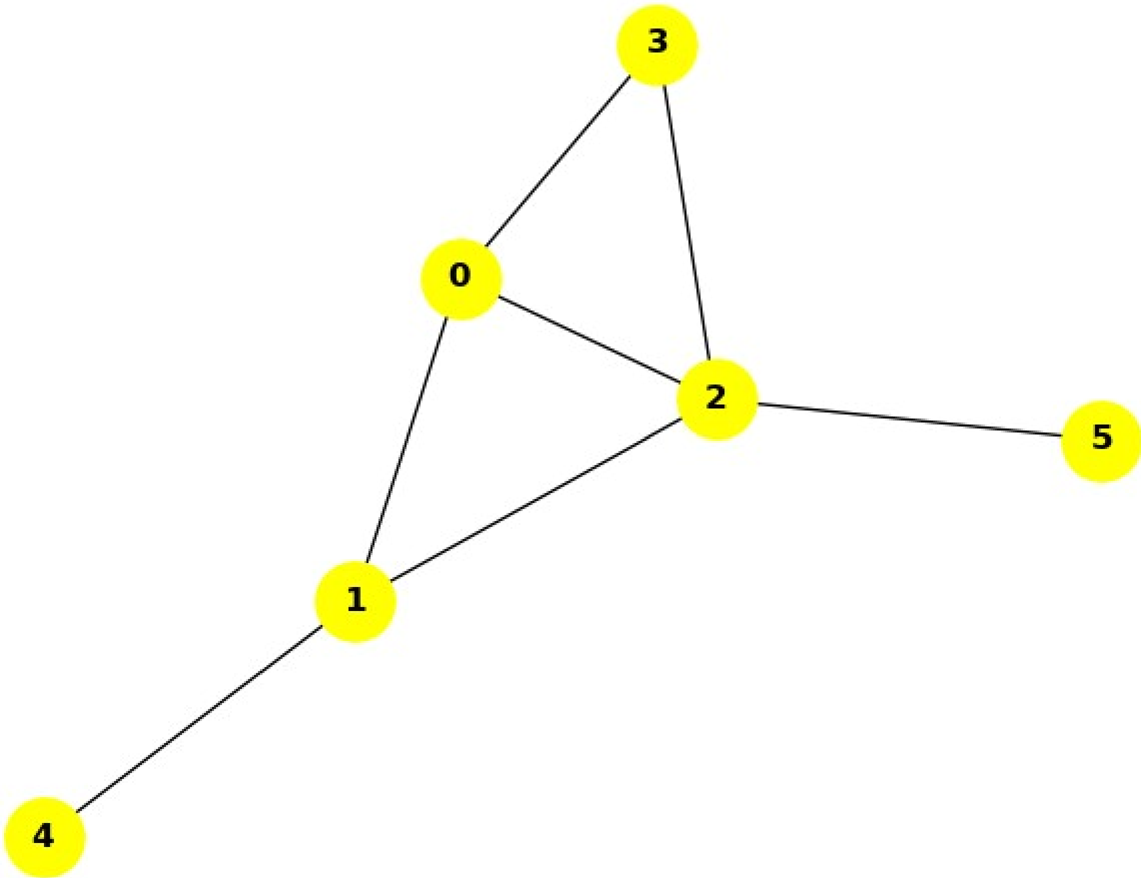
Les graphes

Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)

Sommaire

Les **graphes** sont des **structures de données relationnelles** très utiles pour la modélisation de nombreuses problématiques, dans des domaines variés.

Exemple :



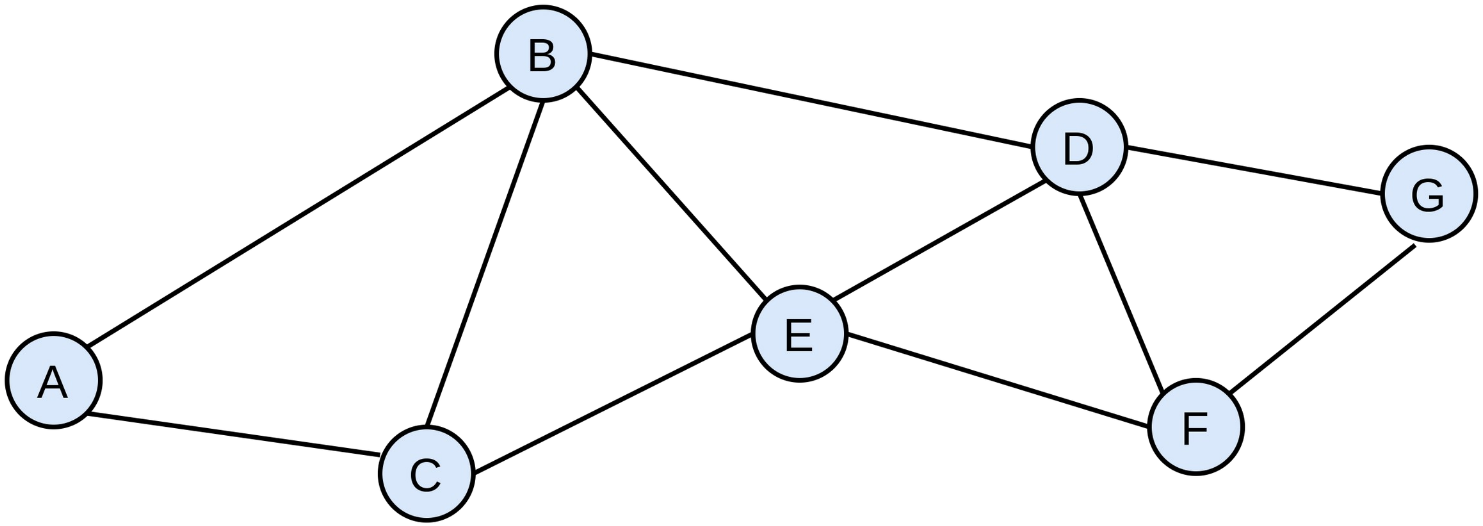
1.1 Définitions sur les graphes

Définitions :

- Un **graphe** est constitué de **sommets** dont certains sont reliés par des **arêtes**.
- Deux sommets reliés par une arête sont dits **adjacents**.
- Un graphe est **complet** lorsque tous ses sommets sont adjacents.
- L'**ordre** d'un graphe est le nombre de sommets de ce graphe.
- Une **boucle** est une arête qui relie un sommet à lui même.
- Le **degré** d'un sommet est le nombre d'arêtes dont ce sommet est une extrémité (une boucle compte double).
- Un sommet est dit **isolé** s'il n'est relié à aucun autre sommet du graphe.
- Un graphe est dit **simple** s'il ne possède aucune boucle et si deux arêtes ne relient jamais les mêmes paires de sommets.

EXERCICE 1

On considère le graphe simple non orienté G1 suivant.



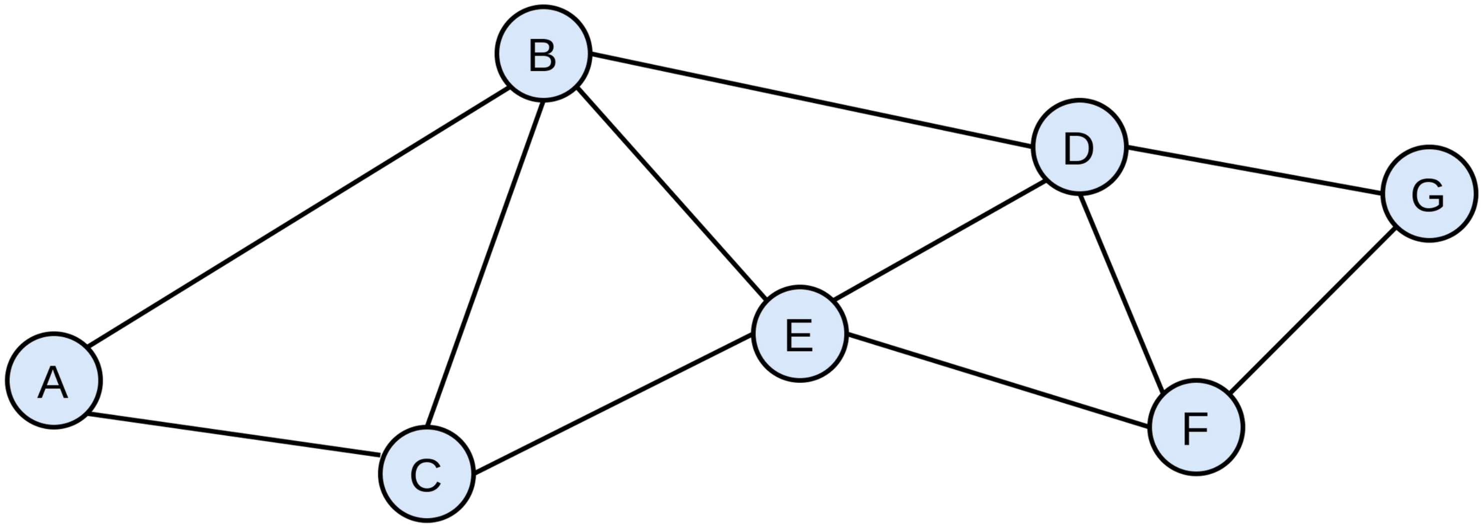
Compléter le tableau suivant :

Sommets								
Degrés								

Ce graphe est-il complet ?

CORRECTION

On considère le graphe simple non orienté G1 suivant.



Compléter le tableau suivant :

Sommets	A	B	C	D	E	F	G	**Total**
Degrés	2	4	3	4	4	3	2	22

22/2 = 11 : il y a 11 arêtes dans ce graphe.

Ce graphe n'est pas complet car les sommets A et E ne sont pas adjacents.

1.2 Matrice d'adjacence

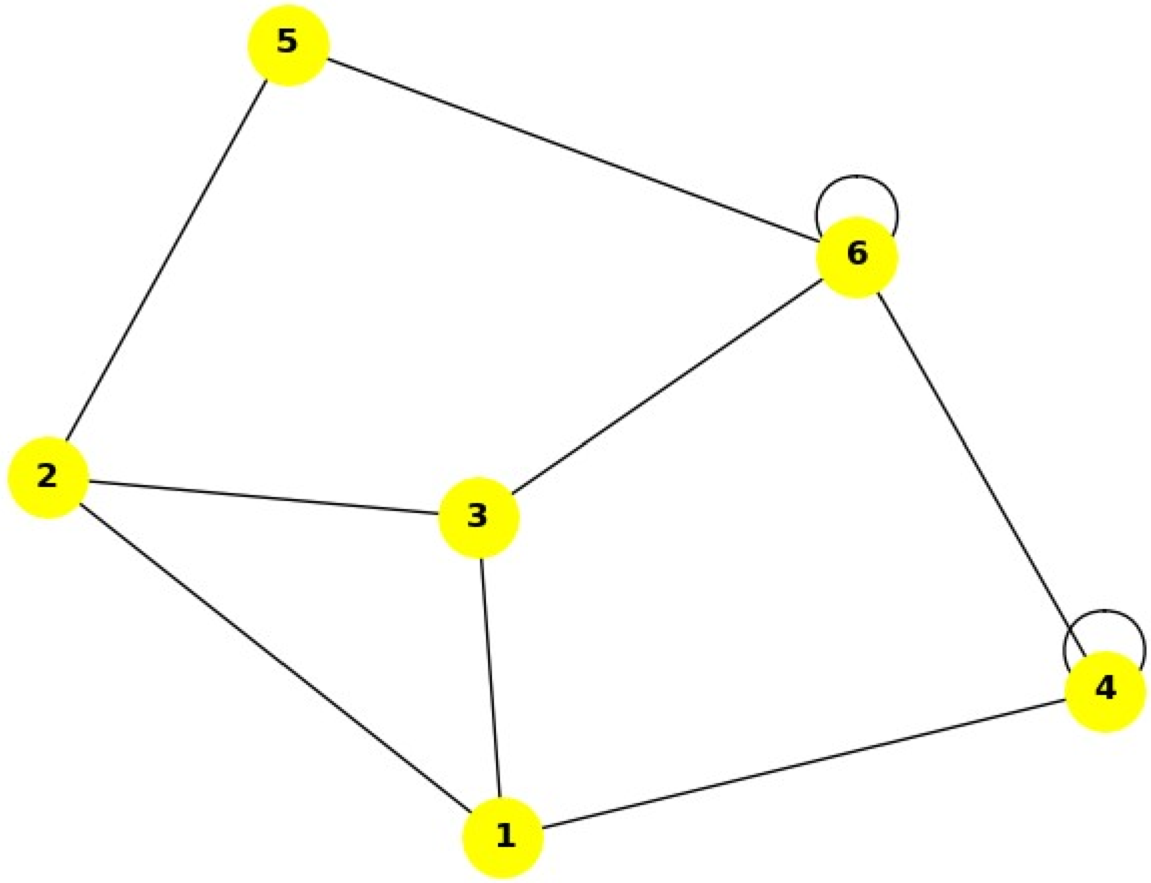
Définition (matrice d'adjacence) : La **matrice d'adjacence** associée à un graphe non orienté d'ordre n dont les sommets sont numérotés de 1 à n est la **matrice carrée symétrique d'ordre** n où le terme figurant en ligne i et colonne j est égal au nombre d'arêtes reliant les sommets i et j.

Exemple : Déterminer la matrice d'adjacence du graphe G1 de l'exercice 1.

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

EXERCICE 2

On considère le graphe non orienté G2 suivant.



Écrire sa matrice d'adjacence

CORRECTION

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

EXERCICE 3

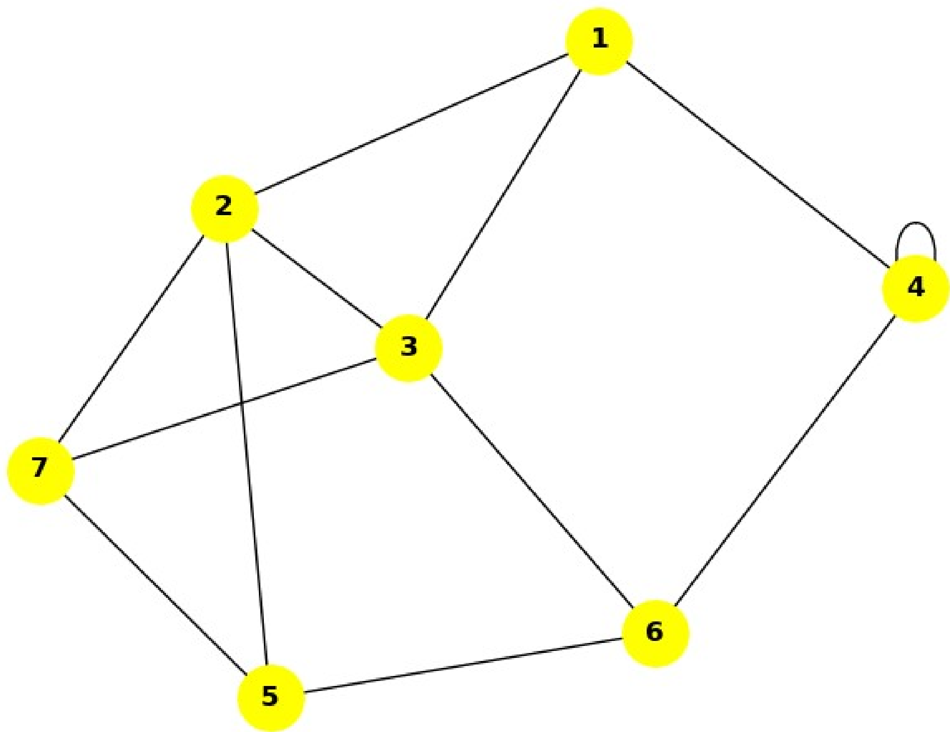
On considère la matrice d'adjacence suivante :

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

Dessiner le graphe non orienté correspondant.

CORRECTION

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



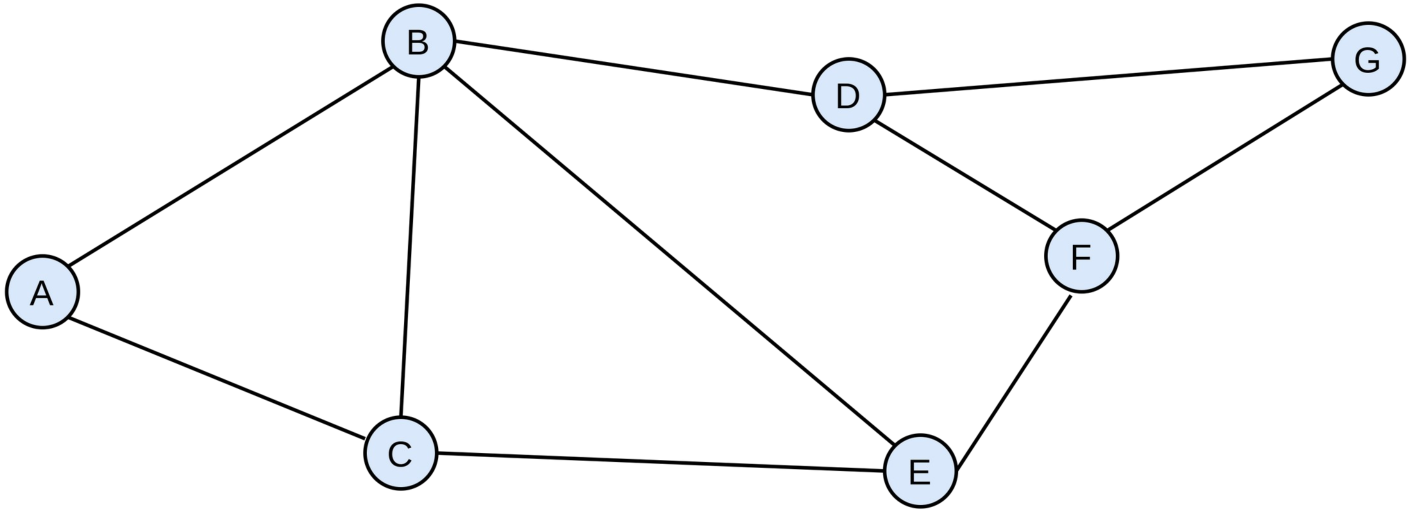
1.3 Chaînes et cycles

Définitions : On considère un graphe G .

- Une **chaîne** de G est une liste ordonnée de sommets de G reliés deux à deux par une arête.
- Une **chaîne fermée** est une chaîne dont l'origine et l'extrémité sont identiques.
- Un **cycle** est une chaîne fermée dont toutes les arêtes sont distinctes.
- Un graphe est dit **connexe** s'il existe une chaîne qui passe par tous les sommets de ce graphe.

EXERCICE 4

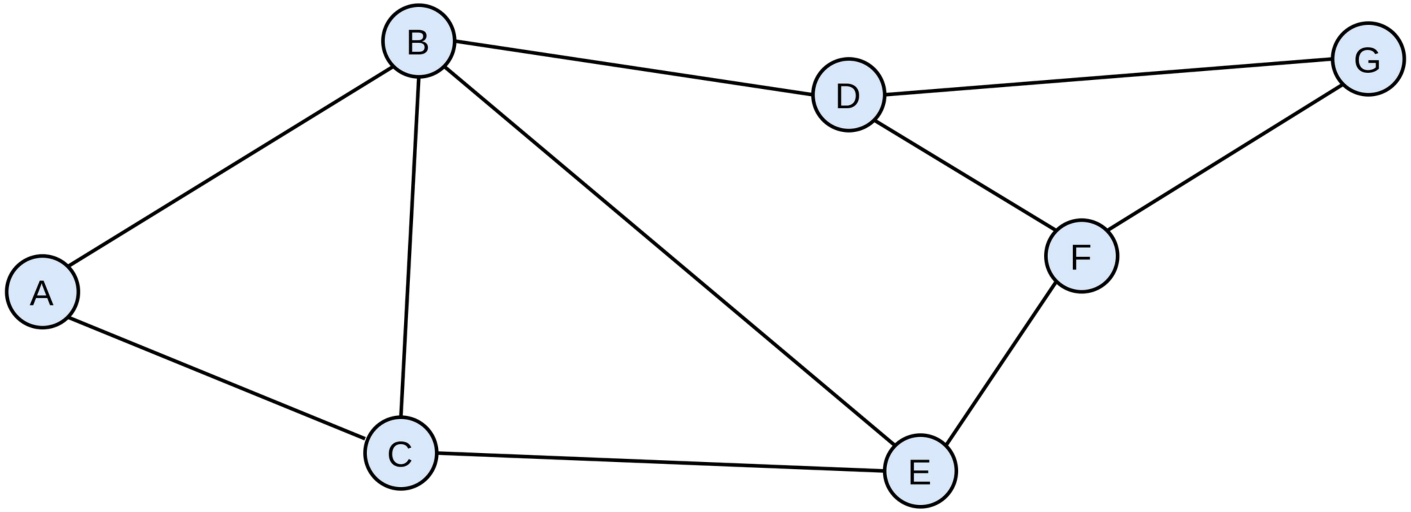
On considère le graphe simple non orienté suivant.



a. Justifier que ce graphe est connexe.
b. Donner un cycle de ce graphe.

CORRECTION

On considère le graphe simple non orienté suivant.



a. La chaîne A – B – C – E – F – D – G passe par tous les sommets du graphe, donc ce graphe est connexe.

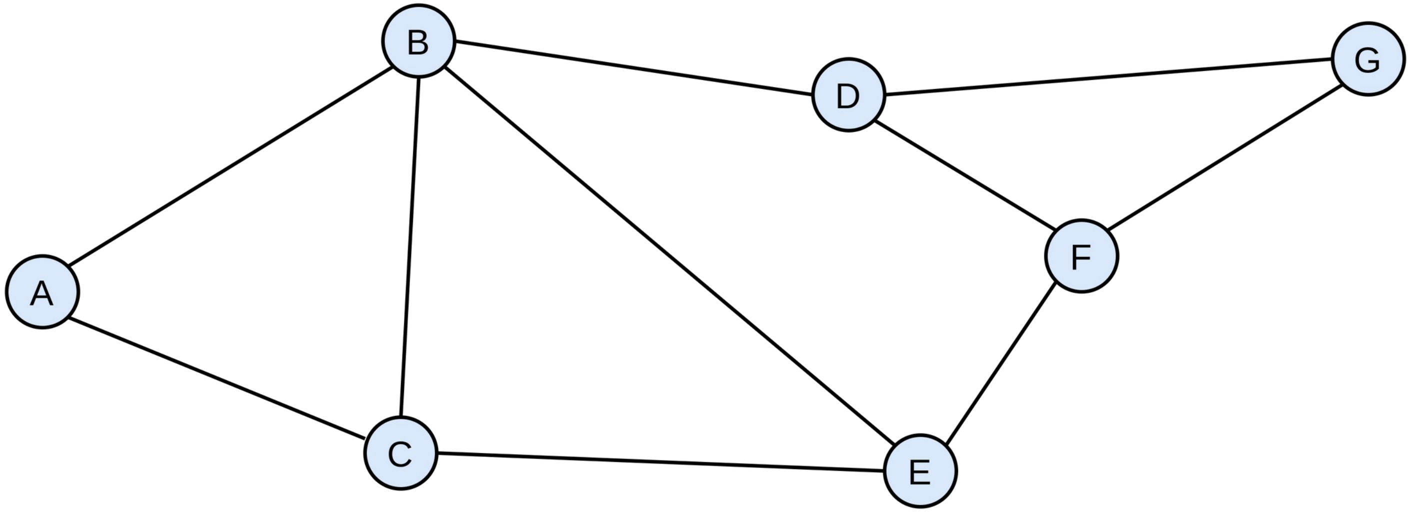
b. A – B – C – A est un cycle de ce graphe.

Définitions : On considère un graphe G .

- La **longueur d'une chaîne** est le nombre d'arêtes qui la composent.
- La **distance entre deux sommets** du graphe est la longueur de la plus petite chaîne reliant ces deux sommets.
- Le **diamètre** du graphe est la plus grande distance entre deux sommets.
- L'**excentricité d'un sommet** est la distance maximale existant entre ce sommet et les autres sommets.
- On appelle **centre** d'un graphe, tout sommet d'excentricité minimale.
- On appelle **rayon** d'un graphe G, l'excentricité d'un centre de G.

EXERCICE 5

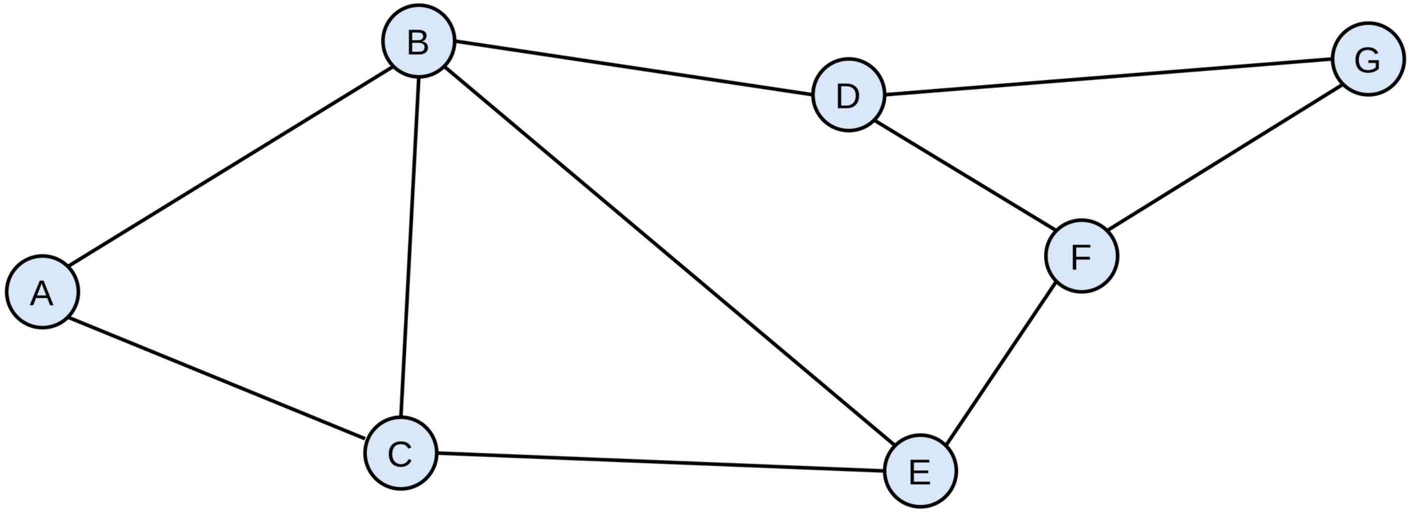
On considère le graphe simple non orienté suivant.



a. Donner la distance entre les sommets A et G .
b. Donner le diamètre de ce graphe.
c. Donner l'excentricité de tous les sommets de ce graphe.
d. En déduire le rayon et les centres de ce graphe.

CORRECTION

On considère le graphe simple non orienté suivant.



a. La distance entre les sommets A et G est 3 .
b. Le diamètre de ce graphe est 3 .
c. Excentricité de tous les sommets.

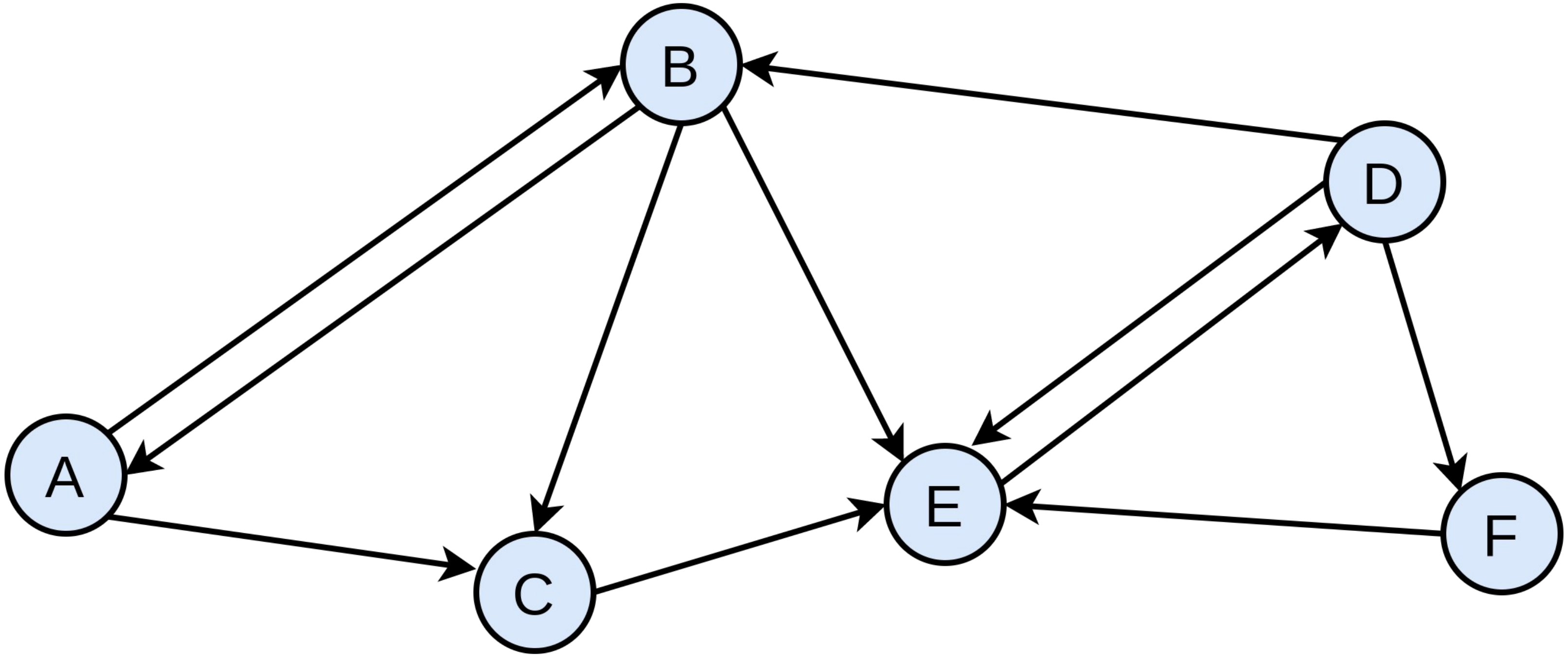
Sommet	A	B	C	D	E	F	G
Excentricité	3	2	3	2	2	3	3

d. On en déduit que le rayon de ce graphe est 2 et les centres sont : B , D et E .

1.4 Graphe orienté

Définition : Un **graphe orienté** est un graphe pour lequel chaque arêtes a un sens.

Exemple : Le graphe G3 ci-dessous est un graphe orienté.



EXERCICE 6

On considère le graphe orienté G3 précédent.
Écrire sa matrice d'adjacence.

 Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)

CORRECTION

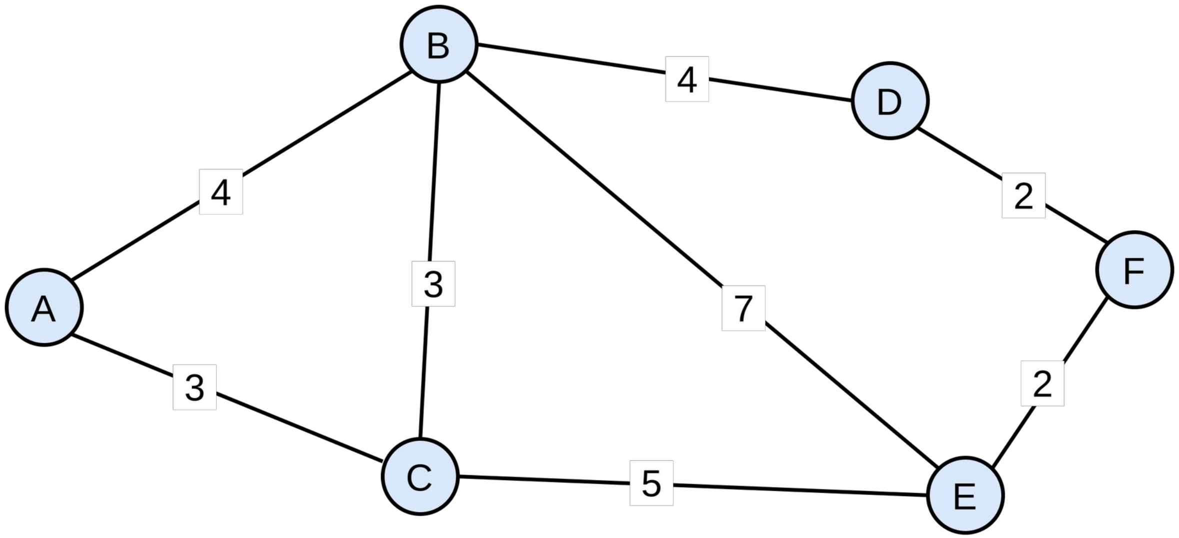
$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

1.5 Graphe pondéré

Définition 1 : Un **graphe pondéré** est un graphe dont les arêtes sont étiquetées avec des nombres positifs appelés **poids de l'arête**.

Définition 2 : Dans un graphe pondéré, le **poids d'une chaîne** est égal à la somme des poids de chaque arêtes qui la composent.

Exemple : Le graphe ci-dessous est pondéré.



Matrice des poids : Donner la matrice des poids du graphe précédent.

$$P = \begin{pmatrix} 0 & 4 & 3 & 0 & 0 & 0 \\ 4 & 0 & 3 & 4 & 7 & 0 \\ 3 & 3 & 0 & 0 & 5 & 0 \\ 0 & 4 & 0 & 0 & 0 & 2 \\ 0 & 7 & 5 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 & 2 & 0 \end{pmatrix}$$

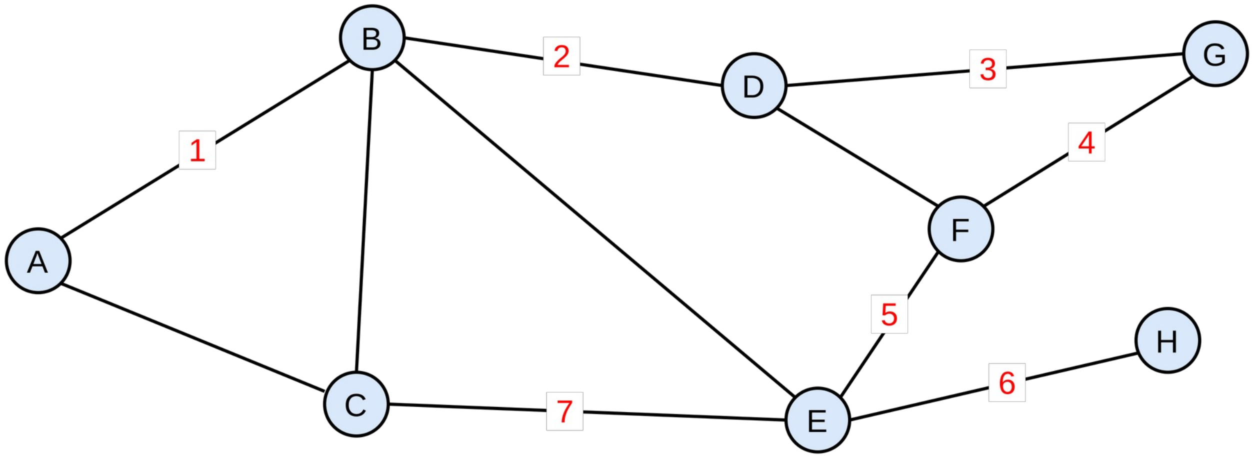
Propriété : Un graphe simple non orienté et pondéré, peut être caractérisé par sa matrice des poids.

2 . Parcours d'un graphe

2.1 Parcours en profondeur

Principe : L'exploration d'un **parcours en profondeur** (DFS : *Depth-First Search*) depuis un sommet S fonctionne comme suit. On suit un chemin dans le graphe jusqu'à un cul-de-sac ou jusqu'à atteindre un sommet déjà visité. On revient alors sur le dernier sommet où on pouvait suivre un autre chemin puis on explore un autre chemin. L'exploration s'arrête quand tous les sommets depuis S ont été visités.

Exemple :



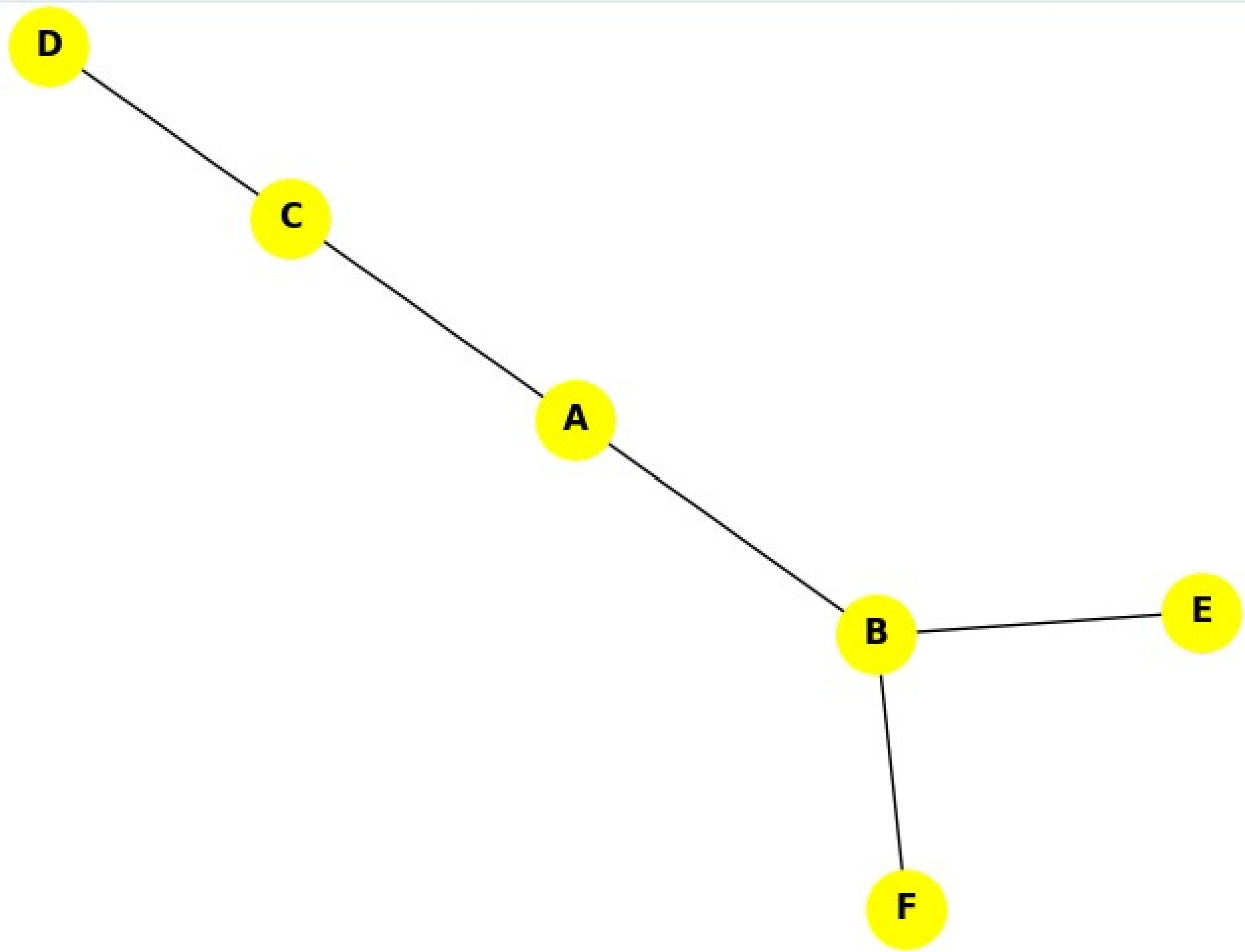
[A, B, D, G, F, E, H, C]

Algorithme récursif : La fonction récursive suivante retourne sous forme de **liste** le parcours en profondeur d'un graphe simple non orienté G en partant du sommet s.

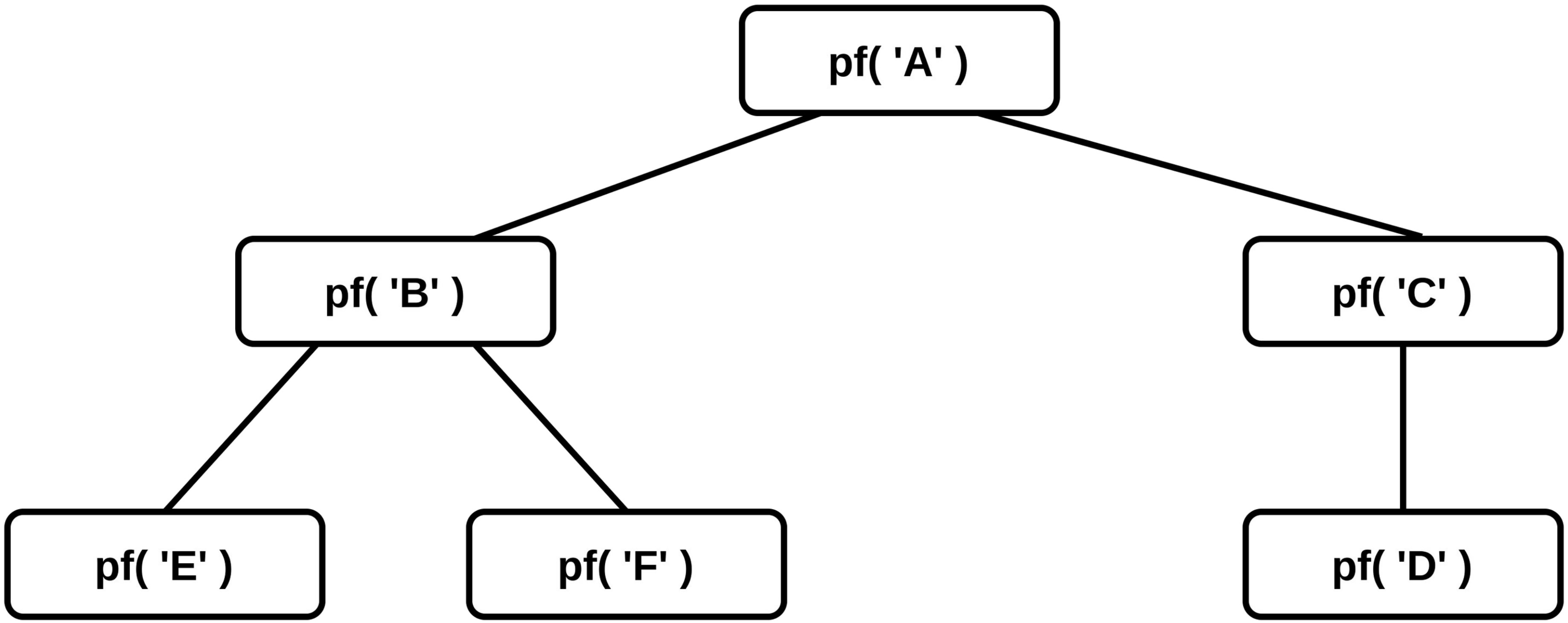
```
parcours_profondeur1(G, s, liste) :
    ajouter le sommet s dans la liste
    Pour tous les sommets a adjacents à s :
        Si a n'est pas dans la liste :
            parcours_profondeur(G, a, liste)
    retourner la liste
```


EXERCICE 7

On considère le graphe simple non orienté G ci-dessous.
Dessiner sous forme d'un arbre, les appels récursifs successifs de la fonction **parcours_profondeur1( )** que l'on notera **pf (s)** en abrégée, si l'appel initial est **pf(G, 'A', liste)** , puis donner la liste retournée.



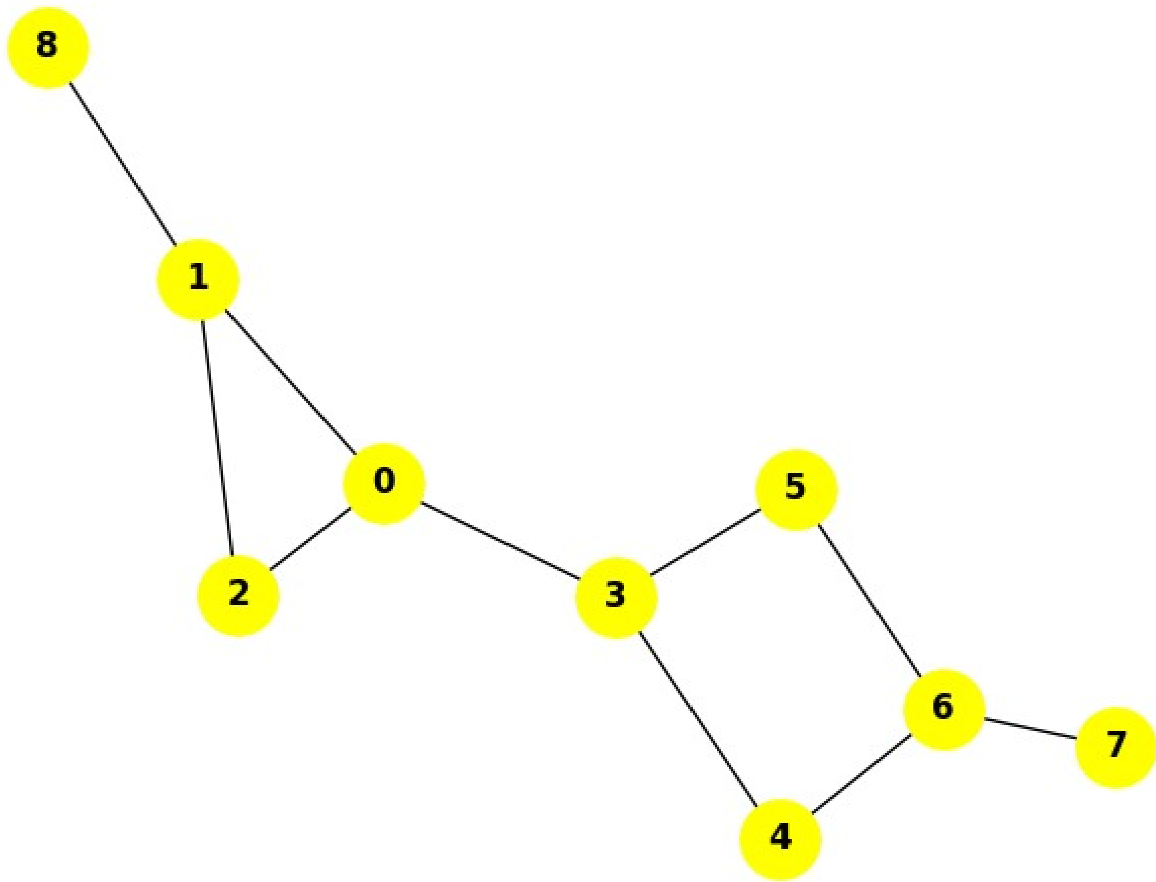
CORRECTION



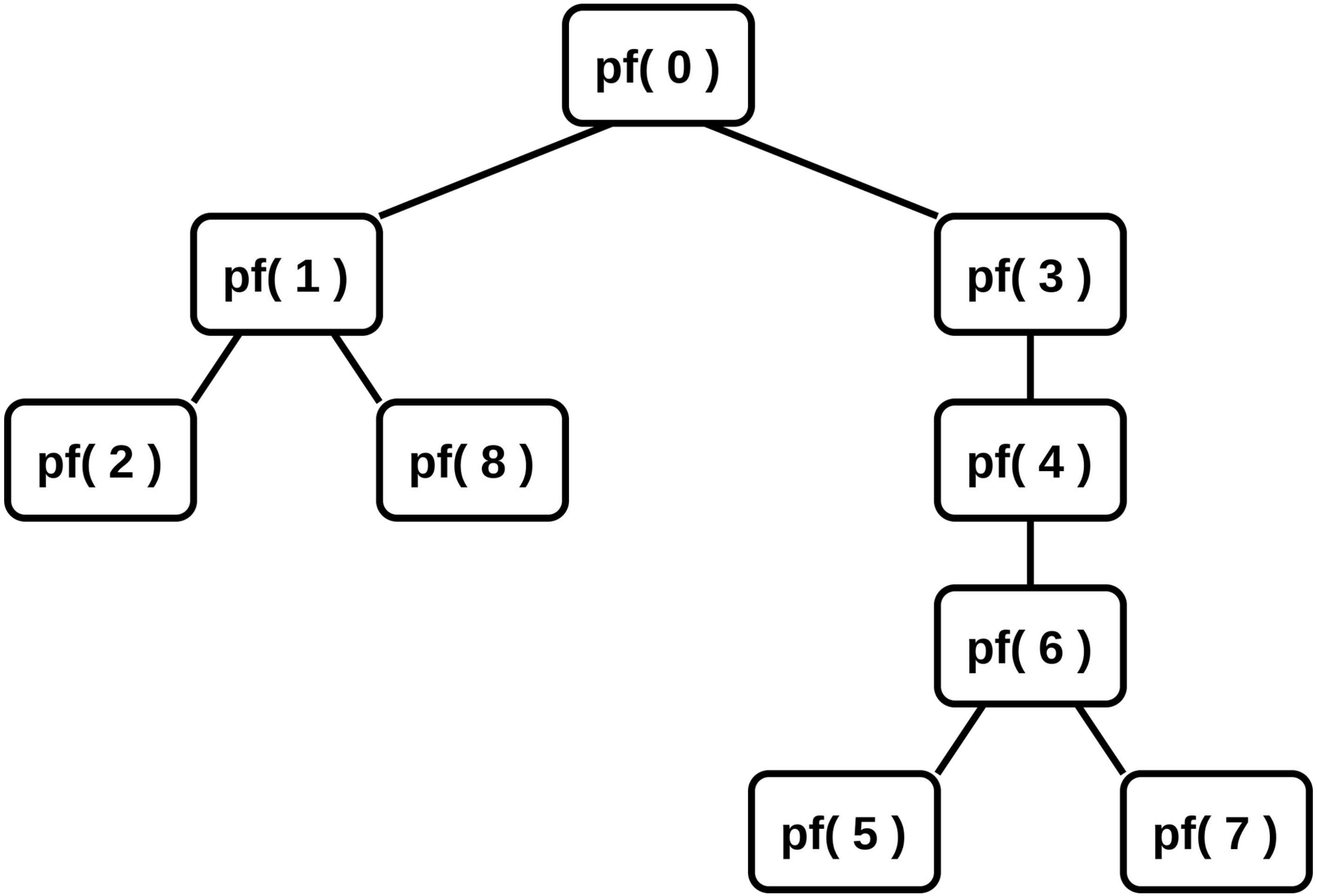

```
>>> parcours_profondeur1(G, 'A', [])
['A', 'B', 'E', 'F', 'C', 'D']
```

EXERCICE 8

On considère le graphe simple non orienté G ci-dessous.
Dessiner sous forme d'un arbre, les appels récursifs successifs de la fonction **parcours_profondeur1( ) que l'on notera pf ( )** si l'appel initial est **pf(G, 0, liste)** puis donner la liste retournée.



CORRECTION



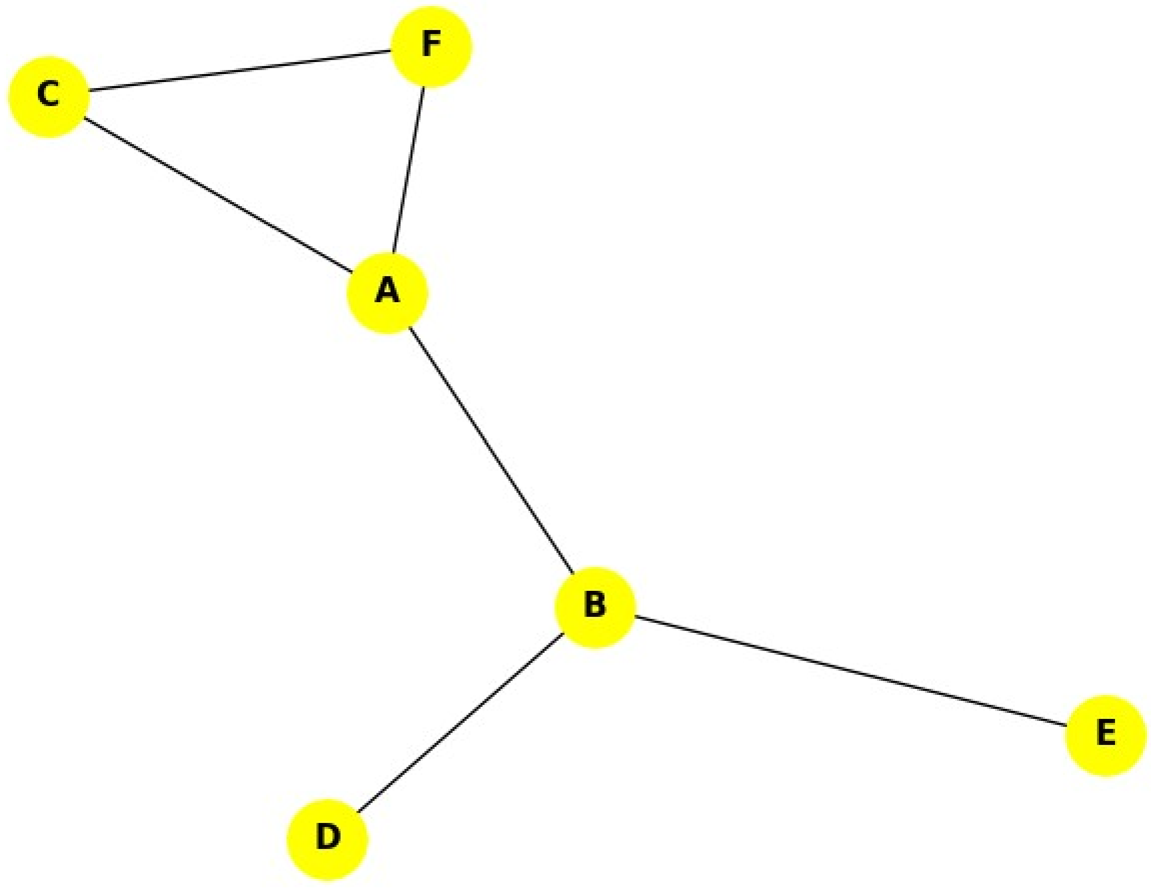

```
>>> parcours_profondeur1(G, 0, [])
[0, 1, 2, 8, 3, 4, 6, 5, 7]
```


Algorithme itératif : La fonction itérative suivante retourne sous forme de **liste** le parcours en profondeur d'un graphe simple non orienté G en partant du sommet s et en utilisant la structure de pile.

```
parcours_profondeur2(G, s) :
    créer une liste vide
    créer une pile vide
    empiler le sommet s
    Tant que la pile n'est pas vide :
        on dépile s
        si s n'est pas dans la liste :
            on ajoute s à la liste
        Pour tous sommets a adjacents à s :
            Si a n'est pas dans la liste :
                on empile le sommet a
    retourner la liste
```


EXERCICE 9

On considère le graphe simple non orienté G ci-dessous.
Détailler l'évolution de la pile et de la liste lors de l'appel de la fonction itérative **parcours_profondeur2(G, 'A')** .



CORRECTION

1. liste = [ ]

pile = 'A'

2. liste = ['A']

pile = 'F' 'C' 'B'

3. liste = ['A', 'F']

pile = 'C' 'B'

4. liste = ['A', 'F', 'C']

pile = 'B'

5. liste = ['A', 'F', 'C', 'B']

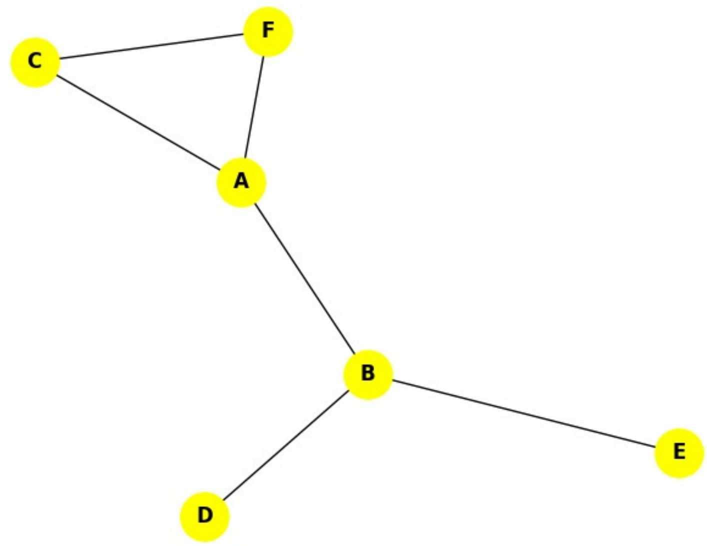
pile = 'E' 'D'

6. liste = ['A', 'F', 'C', 'B', 'E']

pile = 'D'

7. liste = ['A', 'F', 'C', 'B', 'E', 'D']

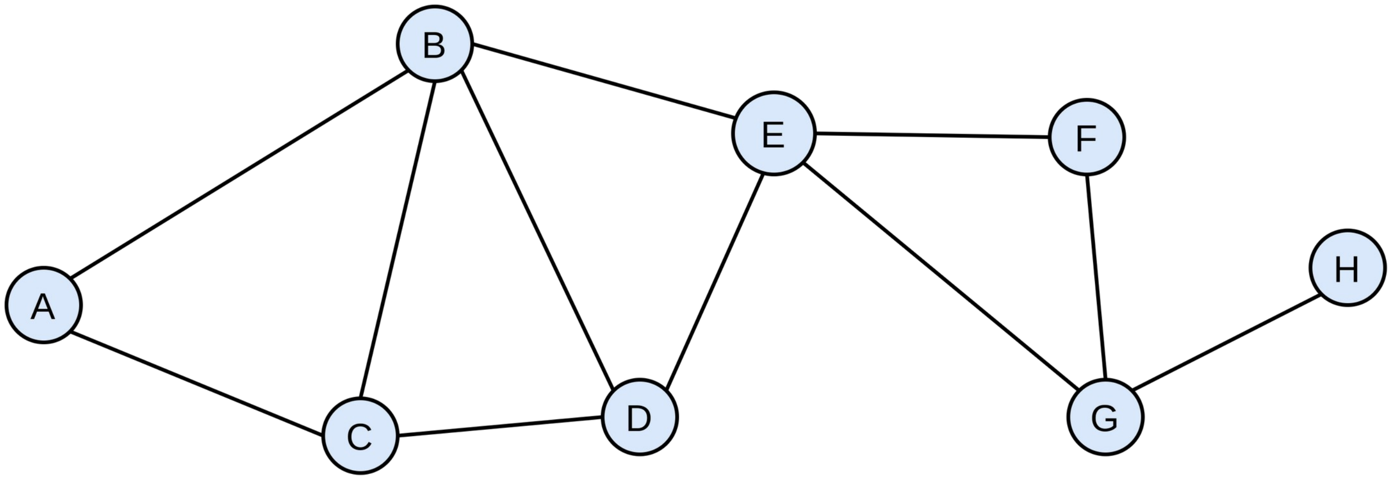
pile =



2.2 Parcours en largeur

Principe : L'exploration d'un **parcours en largeur** (BFS : *Breadth-First Search*) commence à partir d'un nœud source. Puis on liste tous les voisins de la source, pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une **file** dans laquelle on prend le premier sommet et on place en dernier ses voisins non encore explorés. Les nœuds déjà visités sont marqués afin d'éviter qu'un même nœud soit exploré plusieurs fois.

Exemple :



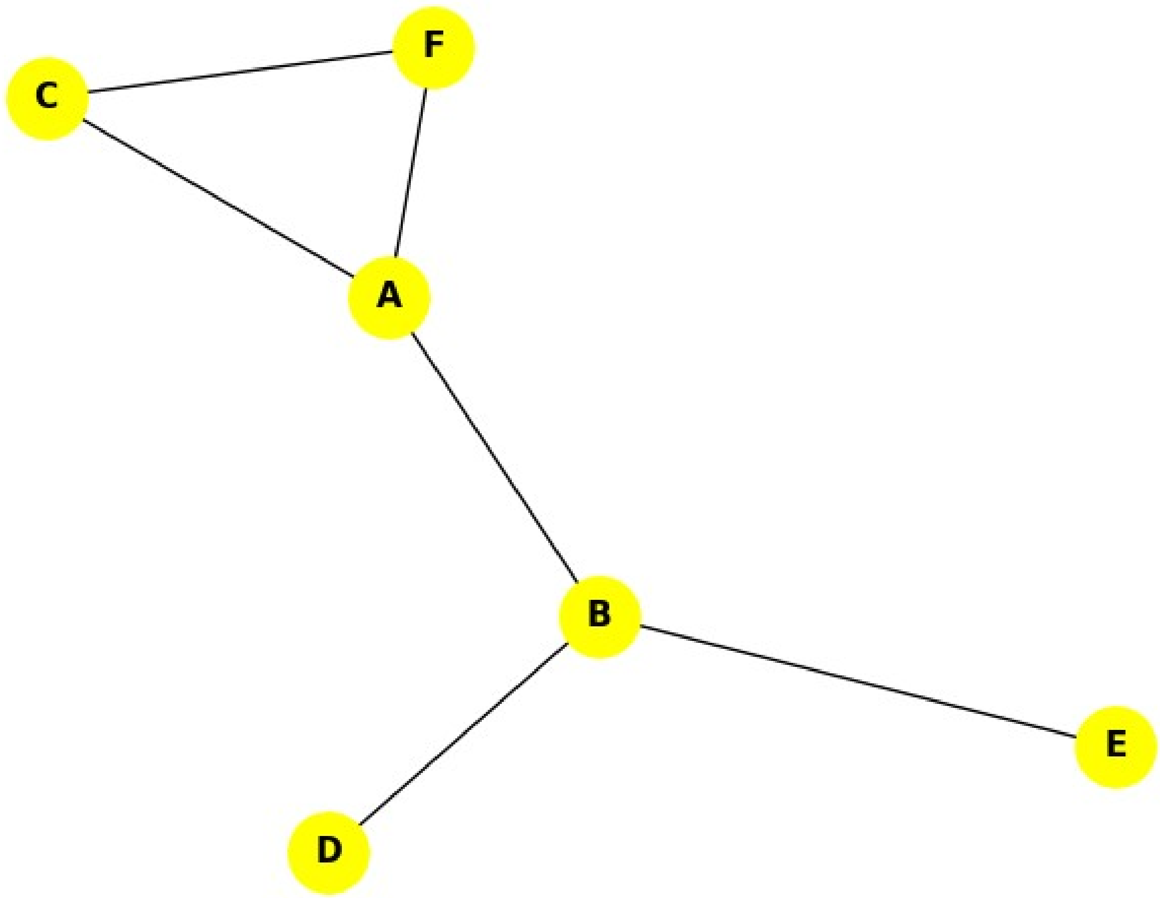
[A, B, C, D, E, F, G, H]

Algorithme itératif : La fonction itérative suivante retourne sous forme de **liste** le parcours en largeur d'un graphe simple non orienté G en partant du sommet s et en utilisant la structure de file.

```
parcours_largeur(G, s) :
    créer une liste vide
    créer une file vide
    enfiler le sommet s
    Tant que la file n'est pas vide :
        on défile s
        Si s n'est pas dans la liste
            on ajoute s à la liste
        Pour tous sommets a adjacents à s :
            Si a n'est pas dans la liste :
                on enfile le sommet a
    retourner la liste
```


EXERCICE 10

On considère le graphe simple non orienté G ci-dessous.
Détailler l'évolution de la file et de la liste lors de l'appel de la fonction itérative **parcours_largeur(G, 'A')** .



CORRECTION

1. liste = [ ]

file = 'A'

2. liste = ['A']

file = 'F' , 'C' , 'B'

3. liste = ['A', 'B']

file = 'E' , 'D' , 'F' , 'C'

4. liste = ['A', 'B', 'C']

file = 'E' , 'D' , 'F'

5. liste = ['A', 'B', 'C', 'F']

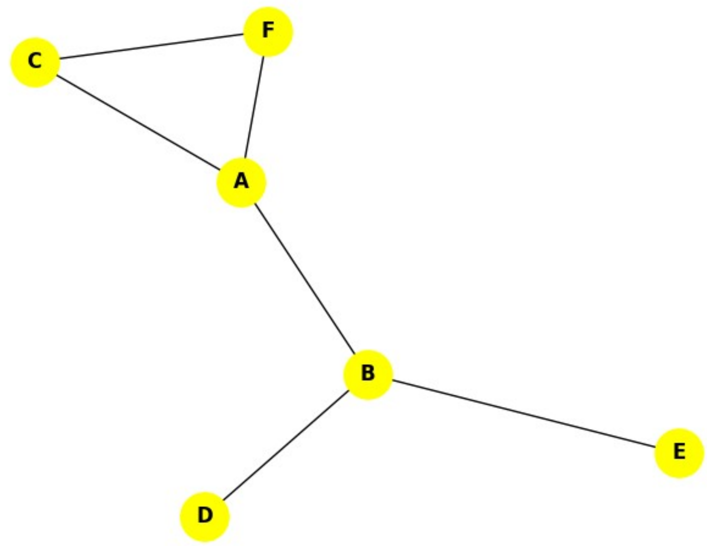
file = 'E' , 'D'

6. liste = ['A', 'B', 'C', 'F', 'D']

file = 'E'

7. liste = ['A', 'B', 'C', 'F', 'D', 'E']

file =



3 . Implémentations d'un graphe

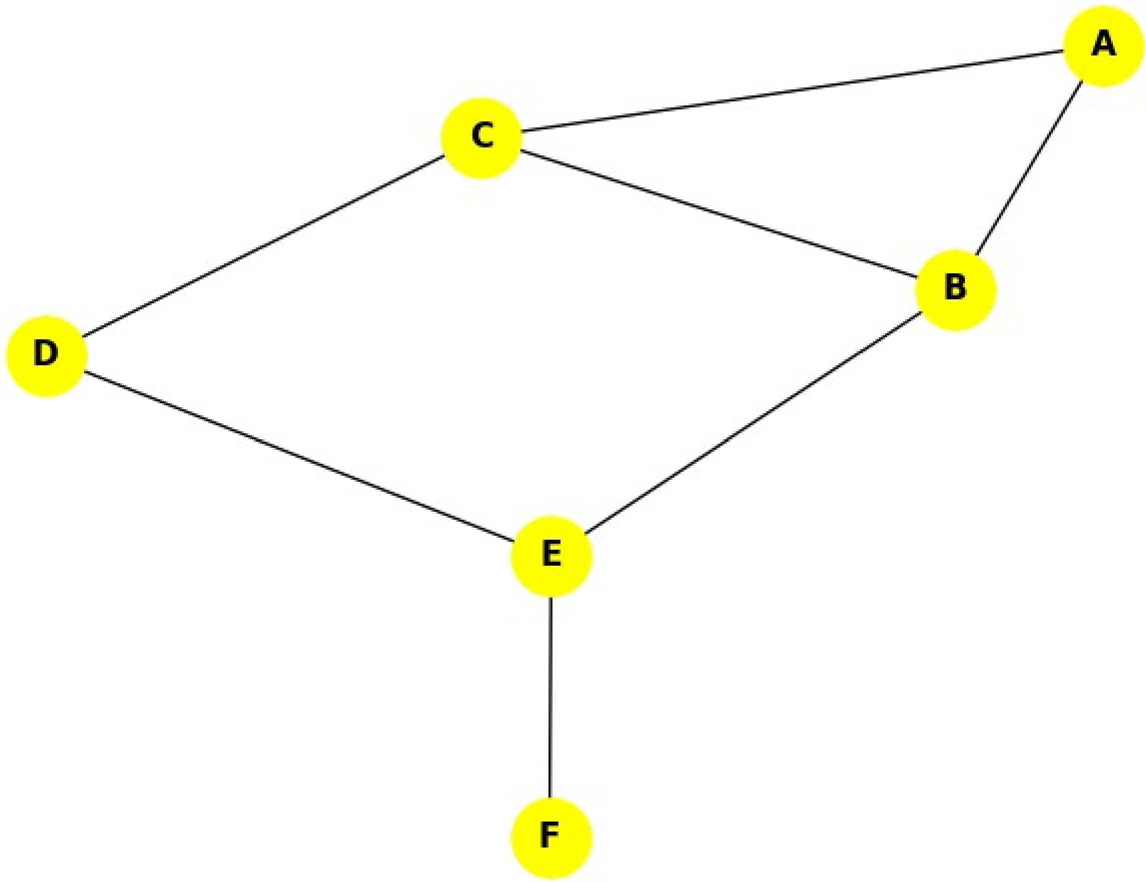
3.1 Implémentation d'un graphe non orienté avec un dictionnaire

Principe : On peut implémenter un graphe en utilisant la structure de dictionnaire. Les clés sont les sommets du graphe et pour chaque clé, la valeur est la liste des sommets adjacents.

Exemple :

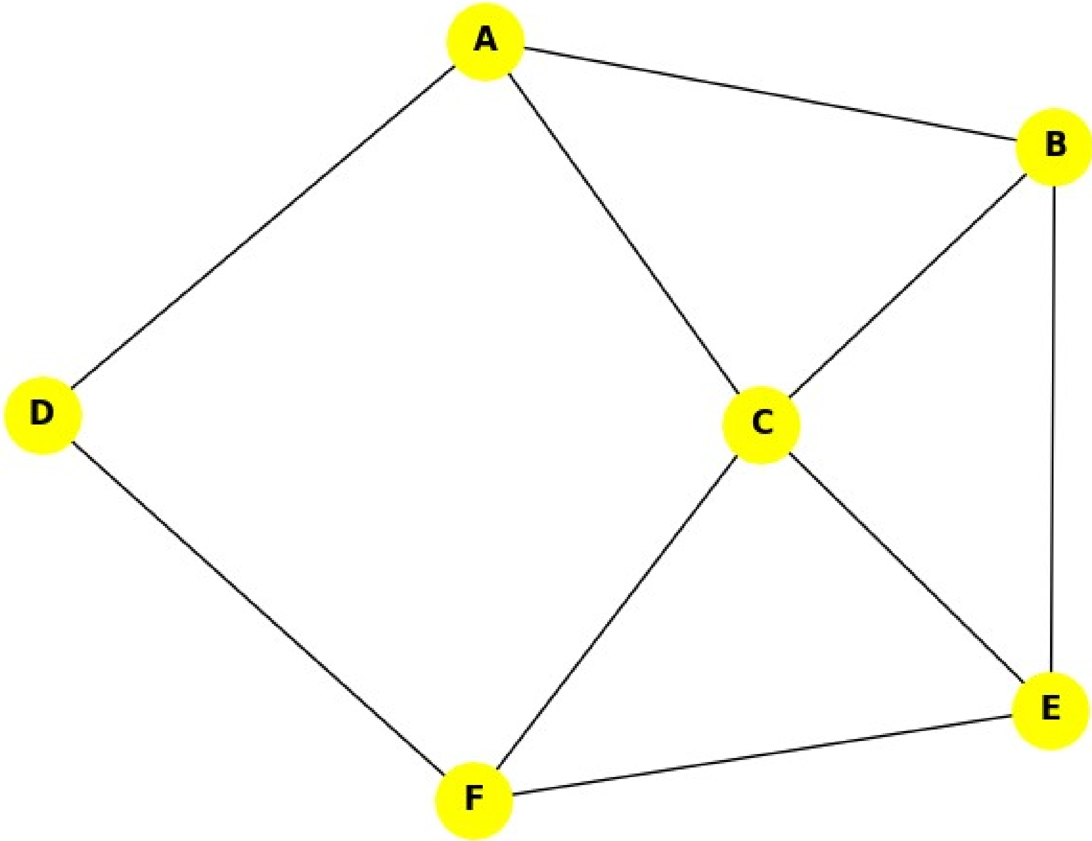
```python
G = dict()
G['A'] = ['B', 'C']
G['B'] = ['A', 'C', 'E']
G['C'] = ['A', 'B', 'D']
G['D'] = ['C','E']
G['E'] = ['B','D','F']
G['F'] = ['E']
```

Dessiner le graphe correspondant.


A
C
B
D
E
F

EXERCICE 11

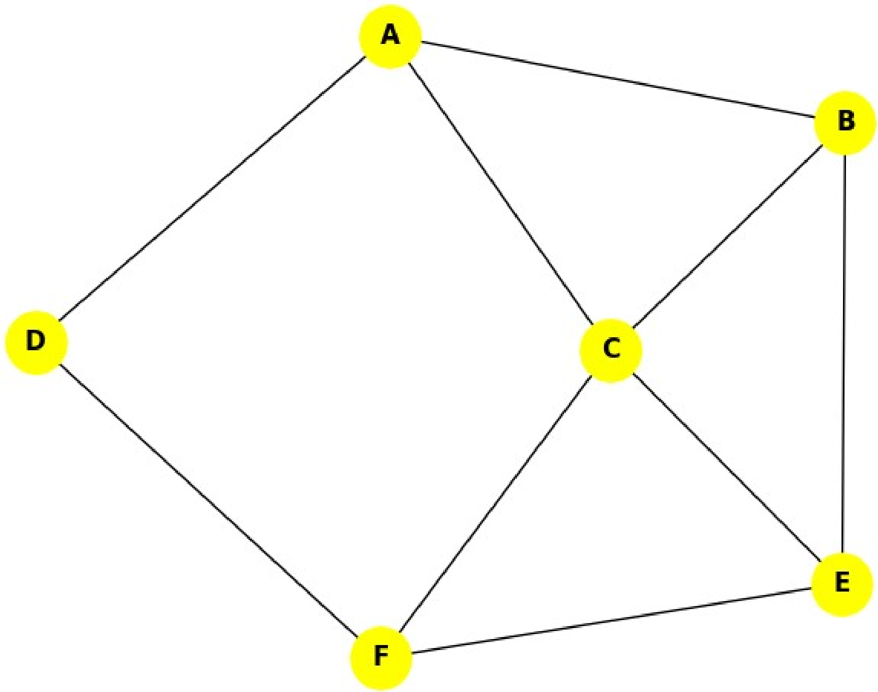
On considère le graphe simple non orienté G ci-dessous.



Écrire le code Python pour l'implémentation de ce graphe avec un dictionnaire.

Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)

CORRECTION




```python
G = dict()
G['A'] = ['B', 'C', 'D']
G['B'] = ['A', 'C', 'E']
G['C'] = ['A', 'B', 'E', 'F']
G['D'] = ['A','F']
G['E'] = ['B','C','F']
G['F'] = ['C', 'D', 'E']
```

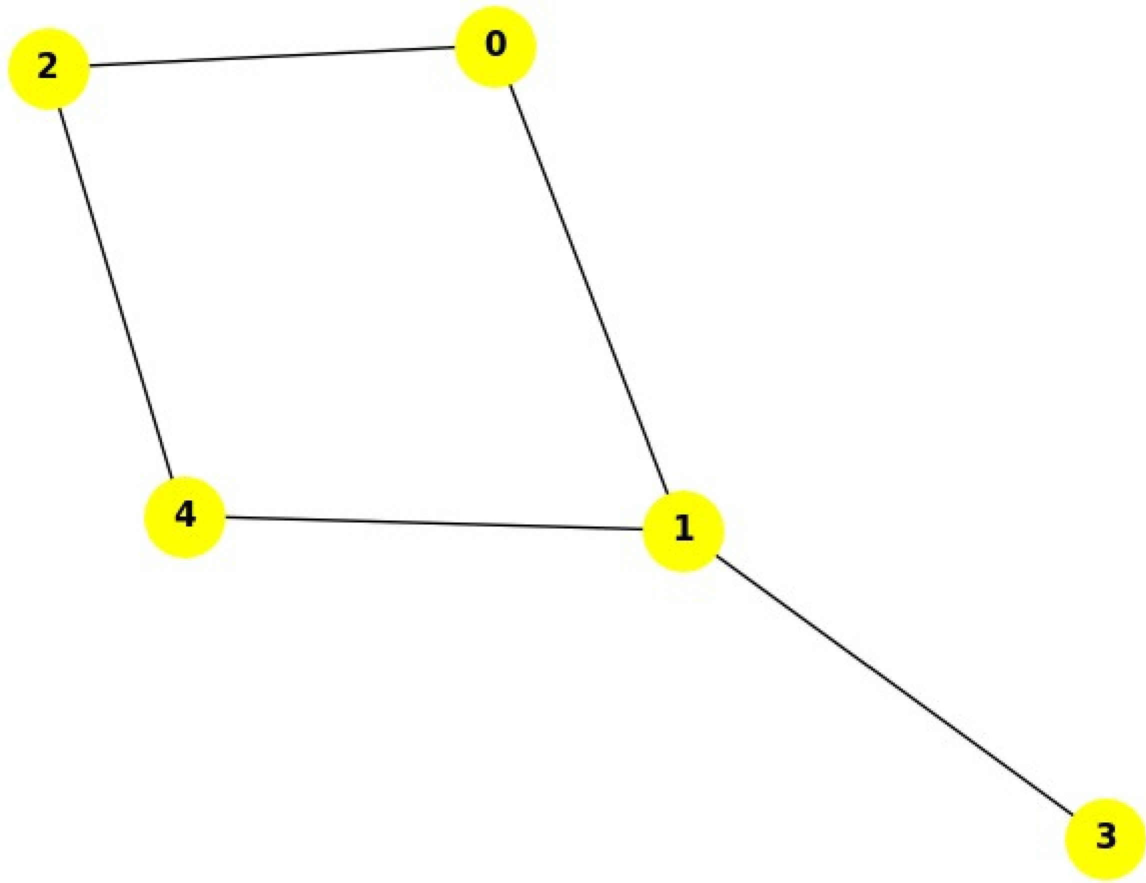
3.2 Implémentation d'un graphe non orienté avec une matrice

Principe : On peut implémenter un graphe en utilisant une structure de liste de listes qui correspond à la matrice d'adjacence. Dans cette implémentation, on numérotera les sommets du graphe : 0 ; 1 ; 2 ; 3 ; etc...

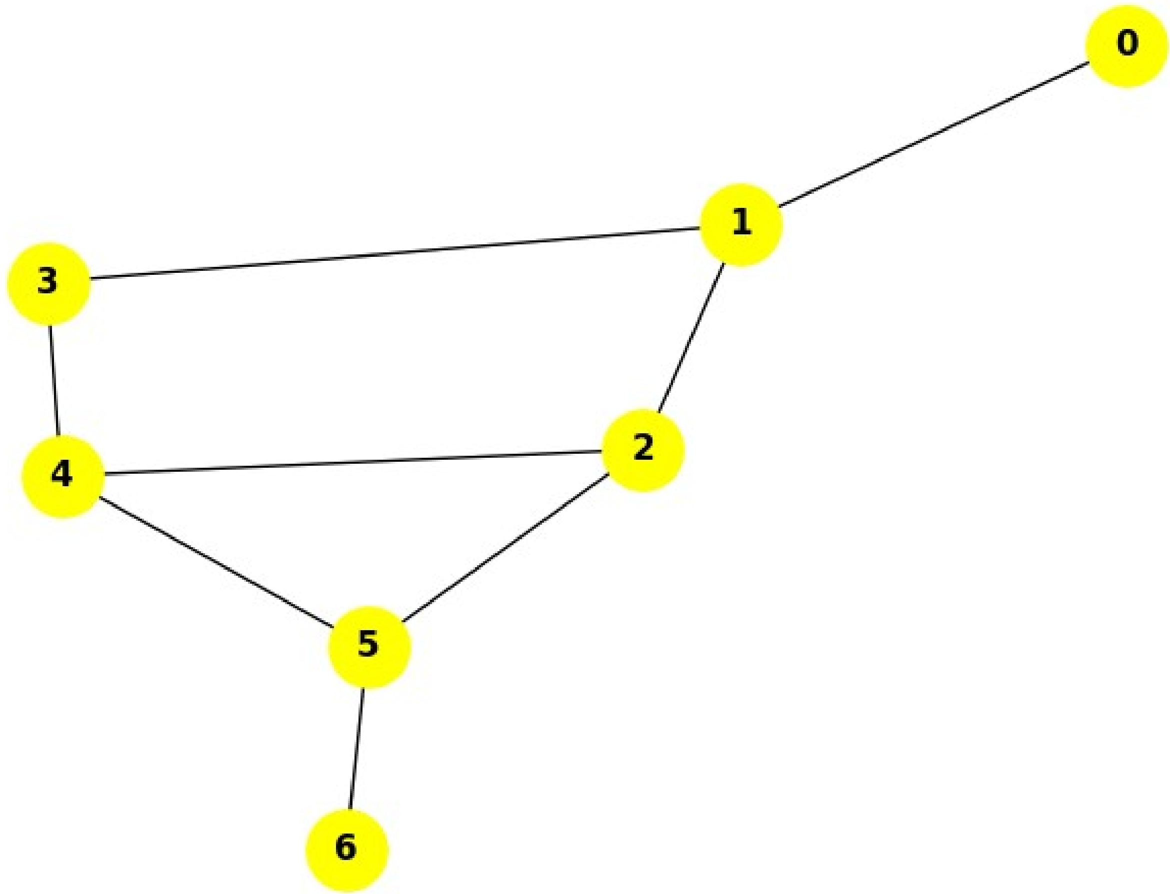
Exemple :

```
G = [
    [0, 1, 1, 0, 0],
    [1, 0, 0, 1, 1],
    [1, 0, 0, 0, 1],
    [0, 1, 0, 0, 0],
    [0, 1, 1, 0, 0]
    ]
```

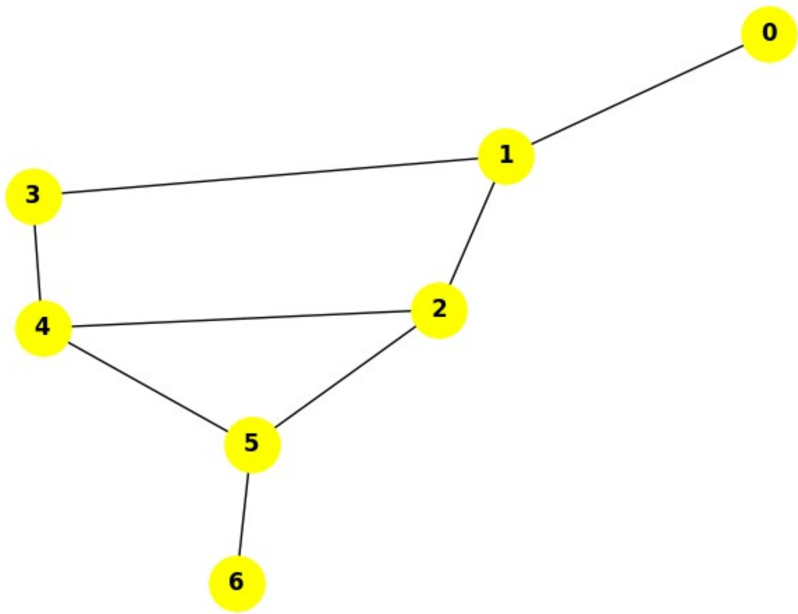
Dessiner le graphe correspondant.


2
0
4
1
3

On considère le graphe simple non orienté G ci-dessous.



Écrire le code Python pour l'implémentation de ce graphe avec une matrice.




```
G = [
    [0, 1, 0, 0, 0, 0, 0]
    [1, 0, 1, 1, 0, 0, 0]
    [0, 1, 0, 0, 1, 1, 0]
    [0, 1, 0, 0, 1, 0, 0]
    [0, 0, 1, 1, 0, 1, 0]
    [0, 0, 1, 0, 1, 0, 1]
    [0, 0, 0, 0, 0, 1, 0]
    ]
```

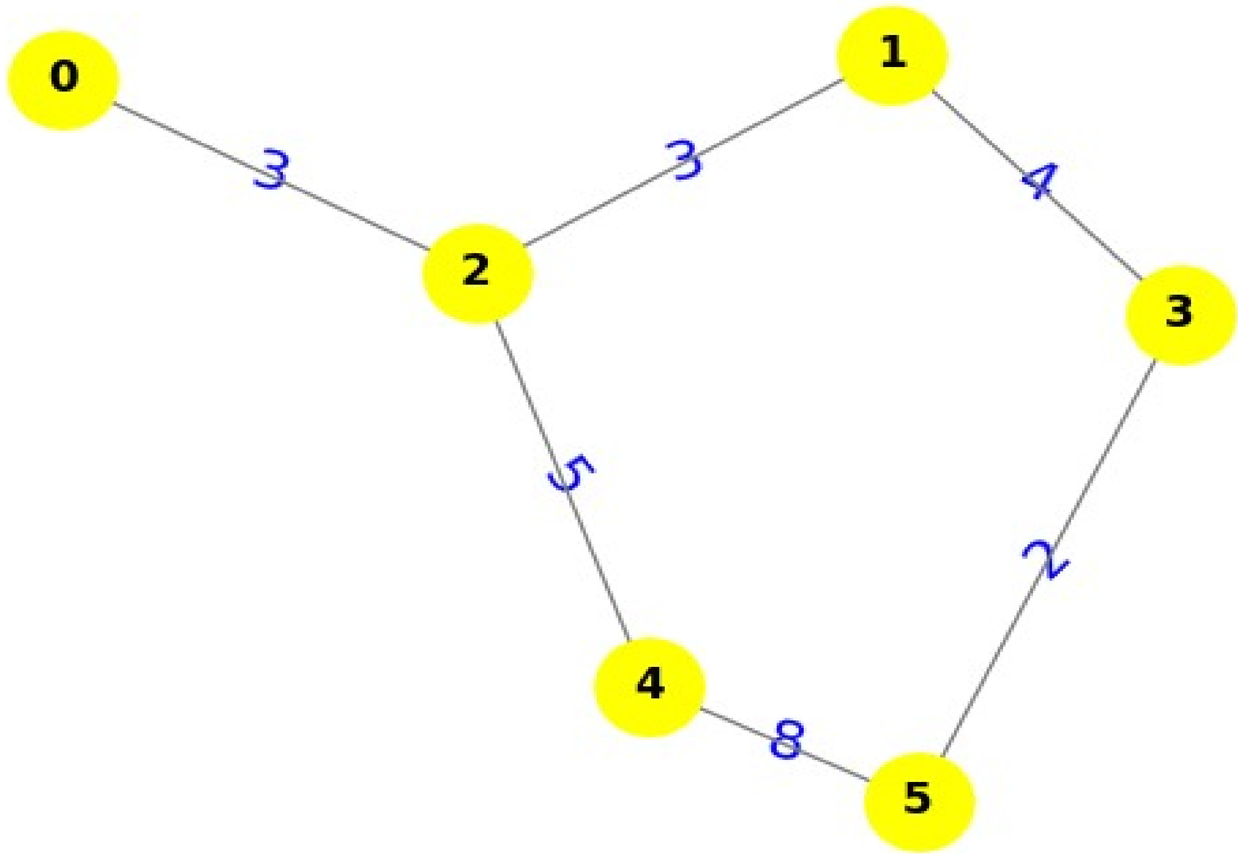
3.3 Implémentation d'un graphe pondéré avec une matrice

Principe : On peut implémenter un graphe en utilisant une structure de liste de listes qui correspond à la matrice des poids. Dans cette implémentation, on numérotera les sommets du graphe : 0 ; 1 ; 2 ; 3 ; etc...

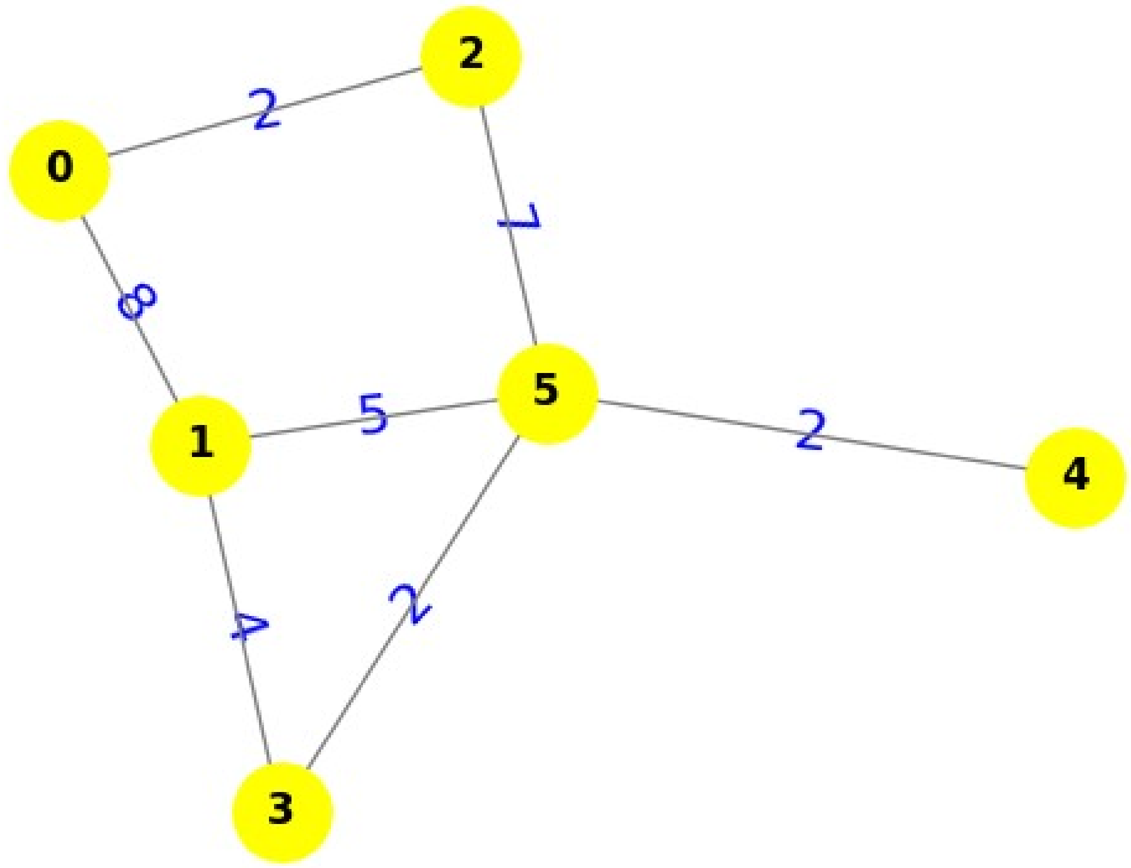
Exemple :

```
G = [
    [0, 0, 3, 0, 0, 0],
    [0, 0, 3, 4, 0, 0],
    [3, 3, 0, 0, 5, 0],
    [0, 4, 0, 1, 0, 2],
    [0, 0, 5, 0, 0, 8],
    [0, 0, 0, 2, 8, 0]
    ]
```

Dessiner le graphe pondéré correspondant.

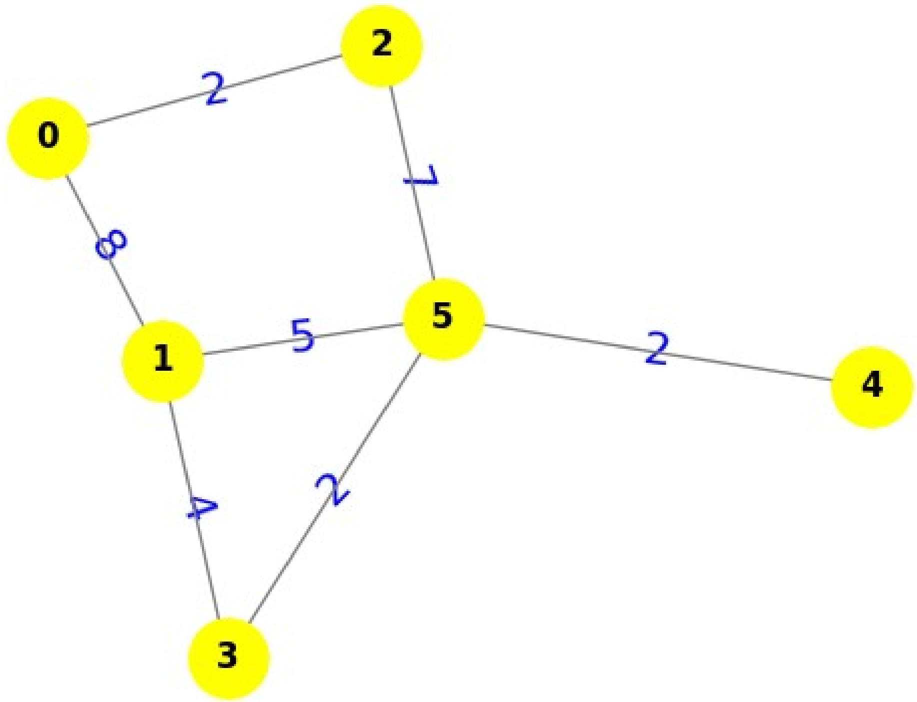

0
1
2
3
4
5
3
3
4
5
2
8

On considère le graphe pondéré G ci-dessous.



Écrire le code Python pour l'implémentation de ce graphe avec une matrice.

CORRECTION




```
G = [
    [0, 8, 2, 0, 0, 0],
    [8, 0, 1, 4, 0, 5],
    [2, 1, 0, 0, 1, 7],
    [0, 4, 0, 0, 0, 2],
    [0, 0, 1, 0, 0, 2],
    [0, 5, 7, 2, 2, 0]
    ]
```

