I. Introduction aux graphes

1 - Vocabulaire

Définition :

Un **graphe** G est un schéma contenant des points appelés **sommets** ou **noeuds** reliés ou non par des **arêtes** (ou **segments** ou **lignes**).

On utilise la notation **G = (S, A)**, un couple d'ensemble finis, tels que :

- S est l'ensemble des **sommets** de G.
- A est l'ensemble des **arêtes** de G.

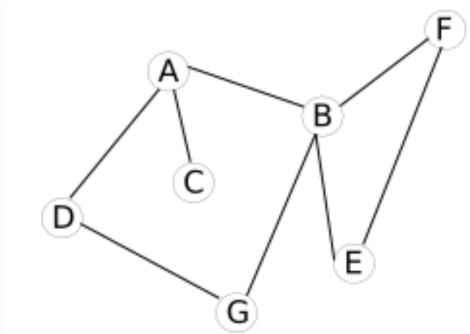
Si une arête relie deux sommets notés S et S', on dit que **les sommets S et S' sont voisins ou adjacents**.

Propriété :

L'ordre d'un graphe est le nombre total de sommets.

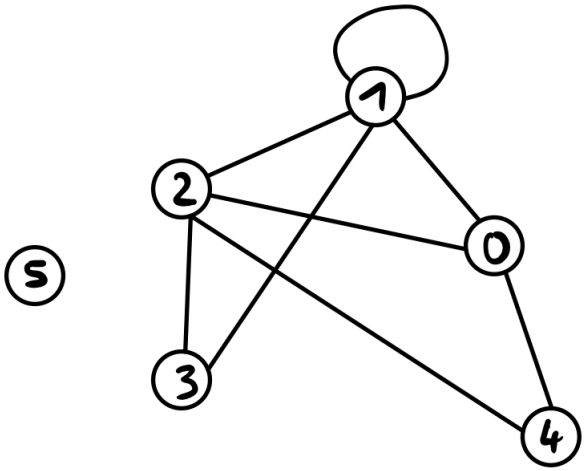
Exemple 1 :

Les sommets représentent les lieux et les arêtes représentent les routes qui les relient. Voici un exemple de représentation avec un graphe non orienté et non pondéré.



2 - Vocabulaire

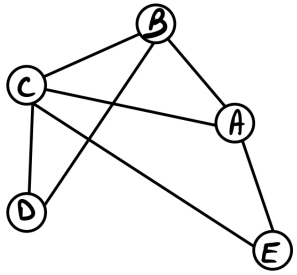
Pour les définitions, on se base sur le graphe suivant :



- La **taille** d'un graphe non orienté est le nombre total d'arêtes. *Dans notre exemple, la taille est de 8.*
- Une **boucle** est une arête reliant un sommet à lui-même.
- Le **degré** d'un sommet S, noté $d(S)$, est égal au nombre d'arêtes dont ce sommet est une extrémité. **Attention : une boucle sera comptée deux fois dans le degré d'un sommet**.
  Dans notre exemple, on a $d(0) = 3$, $d(1) = 5$, $d(2) = 4$, $d(3) = 2$, $d(4) = 2$ *et* $d(5) = 0$.
- Une **chaîne** reliant un sommet S à un sommet S' est une suite d'arêtes consécutives permettant de se rendre de S à S'.
- La **longueur d'une chaîne** est le nombre d'arêtes de la chaîne, ou la somme des poids des arêtes qui le constituent.
- La **distance** `dist(S,S')` entre deux sommets S et S' est la longueur de **la plus courte chaîne** reliant S à S'. S'il n'existe pas de chaîne entre S et S', alors `dist(S, S') =` ∞, notée inf.
- Une chaîne est **simple** si toutes les arêtes de la chaîne sont différentes.
- Une chaîne est **élémentaire** si tous les sommets sont différents sauf pour le sommet d'arrivée qui peut être confondu avec le sommet de départ (cas des cycles)
- Un **cycle** simple est une chaîne simple telle que le sommet d'arrivée est le même que le sommet de départ.

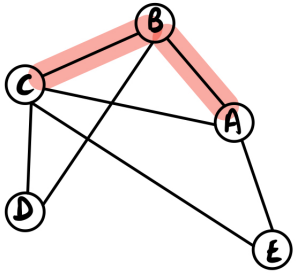
- Un sommet S' est **accessible** à partir d'un sommet S s'il existe une chaîne reliant S à S'.

- Un graphe **non orienté** est **connexe** (ou simplement connexe) si tous les sommets de G sont accessibles entre eux, c'est-à-dire que, pour toute paire de sommets S et S', il existe une chaîne reliant S à S'. Il n'y a pas de sommet isolé.

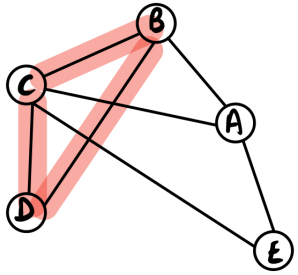


Le graphe est connexe.

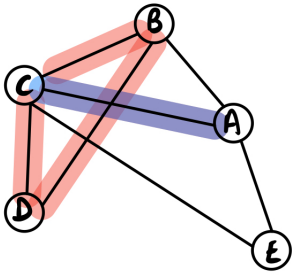
Exemples :



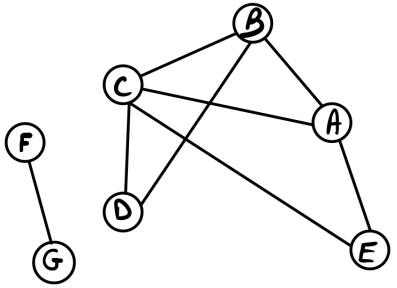
La chaîne ABC est une chaîne simple de distance 2.



La chaîne BCDB est une chaîne simple et élémentaire de longueur 3 correspondant également à un cycle simple.



La chaîne ACBDCA n'est pas une chaîne simple car on utilise deux fois l'arête AC. Elle est de longueur 5.

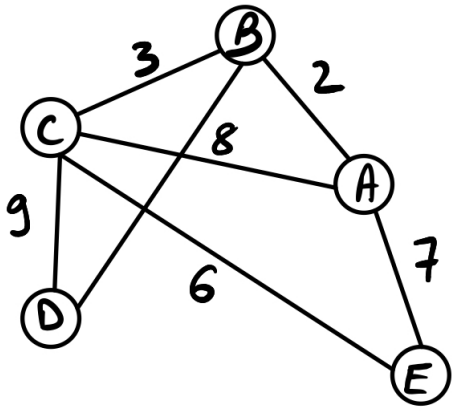


Le graphe n'est pas connexe.

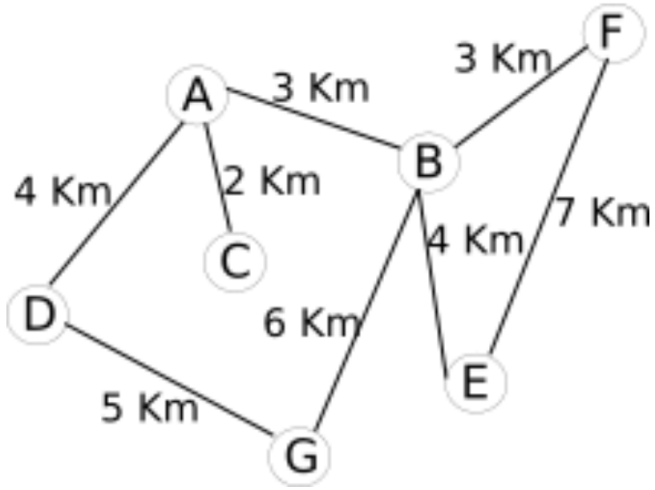
3 - Pondération

Propriété :
On appelle **graphe pondéré** un graphe dont les arêtes sont affectées d'un nombre appelé **poids**.

Le poids d'une arête peut représenter, par exemple, la distance entre deux sommets voisins pour un réseau routier. Dans certains graphes, le poids des arcs peut être négatif.



Il est également possible de représenter un tel problème avec un graphe pondéré. Le poids de chaque arête pouvant correspondre à la distance entre chaque lieu ou au temps de parcours entre chaque lieu.

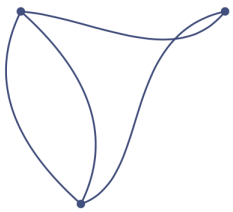


graphe pondéré (Km) cartographie

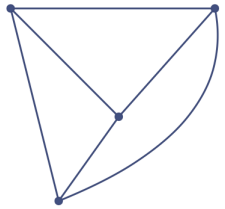
4 - Orientation

Propriété :
Un graphe est **orienté** si les arêtes sont **orientées**, c'est-à-dire dire si on ne peut les parcourir **que dans un sens**. Dans ce cas, une arête est appelée un **arc**.

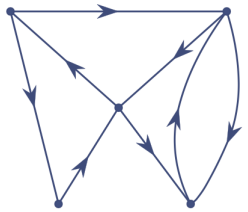
Premiers exemples :



Graphe G_1



Graphe G_2



Graphe G_3

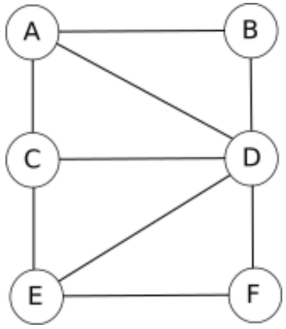
- Le graphe G_1 est un graphe non orienté d'ordre 3.
- Le graphe G_2 est un graphe non orienté d'ordre 4.
- Le graphe G_3 est un graphe orienté d'ordre 5.
- Le graphe G_4 est un graphe non orienté d'ordre 7.

Rappel : pour un graphe orienté, les arêtes sont orientées et sont appelées des **arcs**.

- Si on note S l'origine de l'arc et S' son extrémité, on dit que S' est le **successeur** de S et S le **prédécesseur** de S.
- La **taille** d'un graphe orienté est le nombre total d'arcs.
- Le **degré sortant** d'un sommet S, noté $d_+(S)$ est le nombre d'arcs dont S est le point de départ.
- Le **degré entrant** d'un sommet S, noté $d_-(S)$ est le nombre d'arcs dont S est le point d'arrivée.
- Le degré du sommet S est : $d(S) = d_+(S) + d_-(S)$

Exemple 2 :

On peut représenter un réseau social par un graphe. Les sommets sont des personnes. Deux personnes sont adjacentes lorsqu'elles sont amies par exemple.

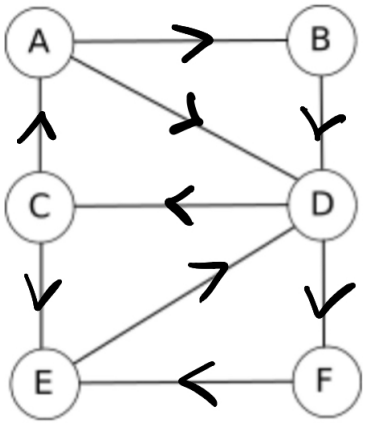


graphe réseau social

Ici, on voit par exemple que A est ami avec B, C et D et que F n'est ami qu'avec E et D.

Exemple 3 :

On peut imaginer le représenter par un graphe orienté si l'amitié n'est pas réciproque entre deux personnes.

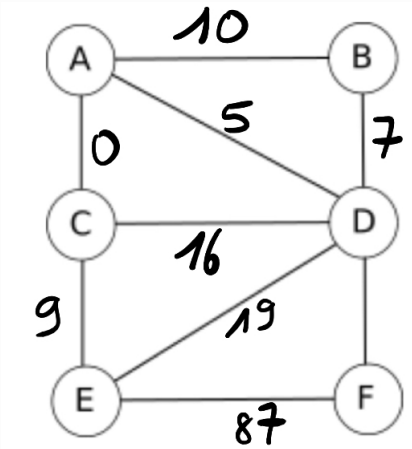


graphe réseau social

Ici, on voit par exemple que A apprécie B et D mais que seul C apprécie A.

Exemple 4 :

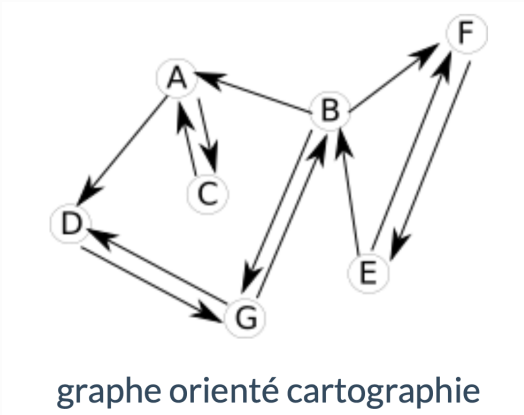
Enfin, on peut imaginer représenter le réseau par un graphe pondéré ou le poids de chaque arête correspond au nombre de photos où les deux personnes reliées apparaissent ensemble.



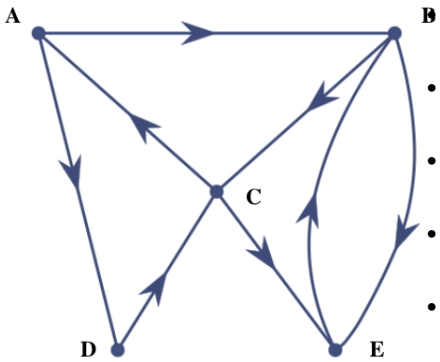
graphe réseau social

Exemple 5 :

On peut imaginer en donner une représentation par un graphe orientée si on ajoute des routes en sens unique et des routes en double sens.



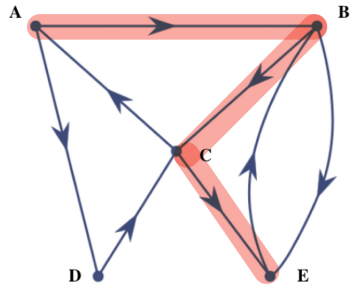
Exemple :



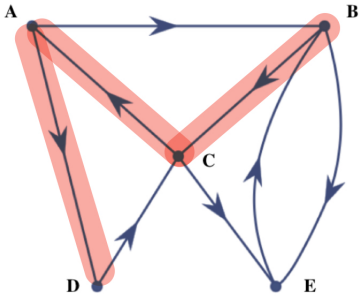
- $d_+(A) = 2$ et $d_-(A) = 1$ donc $d(A) = 3$
- $d_+(B) = 2$ et $d_-(B) = 2$ donc $d(B) = 4$
- $d_+(C) = 2$ et $d_-(C) = 2$ donc $d(C) = 4$
- $d_+(D) = 1$ et $d_-(D) = 1$ donc $d(D) = 2$
- $d_+(E) = 1$ et $d_-(E) = 2$ donc $d(E) = 3$

- Un **chemin** reliant un sommet S à un sommet S' est une suite d'arcs consécutifs permettant de se rendre de S à S'.
- La **longueur d'un chemin** est le nombre d'arcs du chemin, ou la somme des poids des arcs qui le constituent.
- La **distance** `dist(S,S')` entre deux sommets S et S' est la longueur du **plus court chemin** reliant S à S'. S'il n'existe pas de chemin entre S et S', alors `dist(S, S') =` ∞, notée inf.
- Un chemin est **simple** si tous les arcs du chemin sont différents.
- Un chemin est **élémentaire** si tous les sommets sont différents sauf pour le sommet d'arrivée, qui peut être confondu avec le sommet de départ (dans le cas des circuits).
- Un **circuit** est un chemin simple tel que le sommet d'arrivée est le même que le sommet de départ.
- Un sommet S' est **accessible** à partir d'un sommet S s'il existe un chemin reliant S à S'.
- Un **graphe orienté** est **fortement connexe** si tous les sommets de G sont accessibles entre eux, c'est-à-dire que, pour toute paire de sommets S et S', il existe un chemin reliant de S à S' et un chemin reliant de S' à S. Il n'y a pas de sommet isolé.

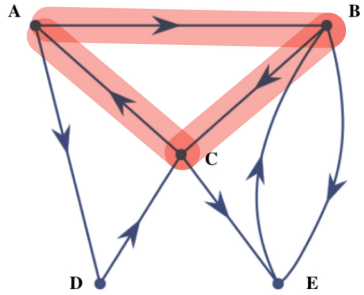
Exemples :



Le chemin ABCE est un chemin simple et élémentaire de longueur 3. Ceci correspond à la distance entre les sommets A et E qui vaut 3 : `dist(A, E) = 3`. On ne peut pas faire ACE contrairement à un graphe non orienté.

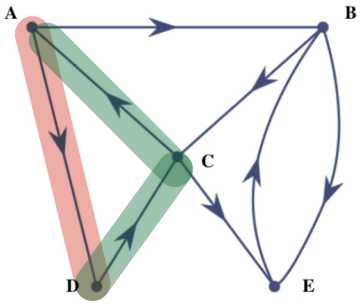


Le chemin BCAD est un chemin simple et élémentaire de longueur 3. Ceci correspond à la distance entre les sommets B et D qui vaut 3 : `dist(B, D) = 3`. On ne peut pas faire BCD contrairement à un graphe non orienté.

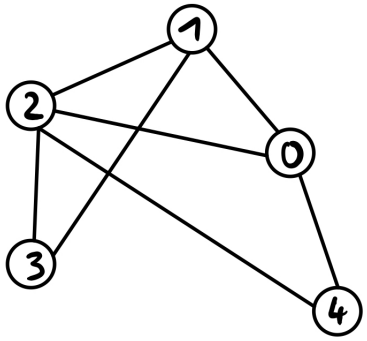


Représentation du circuit ABCA.

AM, JB, NG, SR



Attention, pour les graphes orientés à distinguer `dist(A, B)` et `dist(B, A)`. Ici, on a `dist(A, D) = 1` et `dist(D, A) = 2`



La matrice d'adjacence pour ce graphe est :

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

II. Représentation d'un graphe

Après cette introduction, il est temps d'apprendre comment représenter un graphe en Python. Il existe différentes manières de le faire.

1 - Matrice d'adjacence et graphe non-orienté

L'idée de la matrice d'adjacence est de représenter quels sont les sommets voisins et ceux qui ne le sont pas. Pour un graphe d'ordre n (n sommets), on va donc représenter le graphe par une matrice de taille `n x n`. Le contenu de la matrice pourra être différent selon les cas.

a) Graphe non pondéré

Règles de construction de la matrice d'adjacence :

- Si deux sommets différents i et j sont reliés par une arête, alors $M[i,j] = 1$
- Si les deux sommets i et j ne sont pas reliés, alors $M[i,j] = 0$

Propriété :

Pour un graphe sans boucle, tous les éléments de la diagonale sont nuls : $M[i,i] = 0$. Pour un graphe non orienté, la matrice est symétrique puisque $M[i,j] = M[j,i]$

Exemple :

On peut déduire très simplement de la matrice d'adjacence quels sommets sont reliés. Dans notre exemple, on voit bien que le sommet 2 est relié à tous les autres sommets.

On peut alors la représenter en Python à l'aide d'une liste de listes :

```
M = [[0,1,1,0,1],[1,0,1,1,0], [1,1,0,1,1], [0,1,1,0,0], [1,0,1,0,0]]
```

Exemple 6 :

Proposer une fonction `G(N)` décrivant un graphe où chaque sommet i est relié avec ses voisins directs $i+1$ et $i-1$.

```
def G(N):
    M = [ [0 for j in range(N)] for i in range(N) ]
    for i in range(1,N-1):
        M[i][i+1] = 1
        M[i][i-1] = 1
    M[0][1] = 1
    M[N-1][N-2] = 1
    return M
```

b) Graphe pondéré non orienté

Définition :

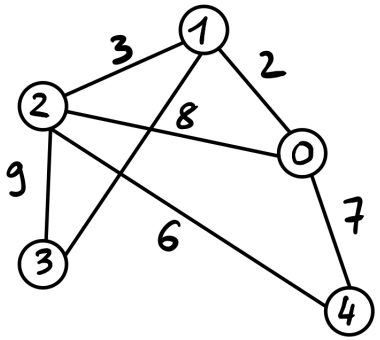
Règles de construction de la matrice d'adjacence :

Si deux sommets différents i et j sont reliés par une arête, alors $M[i,j] = d$ où d est le poids de l'arête.

Si les deux sommets i et j ne sont pas reliés, alors $M[i,j] = \infty$ (on affecte la valeur ∞ pour différencier d'une arête de poids nul)

Pour un graphe sans boucle, les éléments de la diagonale sont nuls : $M[i,i] = 0$

Exemple :



La matrice d'adjacence pour ce graphe est :

$$\begin{pmatrix} 0 & 2 & 8 & \infty & 7 \\ 2 & 0 & 3 & 8 & \infty \\ 8 & 3 & 0 & 9 & 6 \\ \infty & 8 & 9 & 0 & \infty \\ 7 & \infty & 6 & \infty & 0 \end{pmatrix}$$

On peut alors la représenter en Python à l'aide d'une liste de listes :

```
M = [[0,2,8,'inf',7],[2,0,3,8,'inf'],[8,3,0,9,6],\
['inf',8,9,0,'inf'],[7,'inf',6,'inf',0]]
```

c) Propriétés de la matrice d'adjacence

Exemple 7 :

On dispose d'un graphe non orienté, pondéré ou non, représenté par sa matrice d'adjacence. On note M la matrice d'adjacence et N le nombre de sommets.

☐ 1 - Proposer une fonction donnant le nombre de sommets du graphe et évaluer sa complexité

☐ 2 - Proposer une fonction donnant le nombre d'arêtes du graphe et évaluer sa complexité

☐ 3 - Pour un graphe non orienté, on veut écrire une fonction `voisins(M, i)` retournant la liste des sommets voisins du sommet i. M est la matrice d'adjacence. On propose une fonction dans le cadre d'un graphe non pondéré.

Nombre de sommets du graphe :

On a simplement `N = len(M)`. Complexité à coût constant.

Nombre d'arêtes du graphe :

Pour un graphe non orienté, le nombre d'arêtes est égal à la demi-somme de tous les coefficients non nuls de la matrice d'adjacence. (dans le cas d'un graphe pondéré, ce serait tous les coefficients non nuls et non égaux à `'inf'`).

```
def nombre_arete(M):
    n = len(M)
    cpt = 0
    for i in range(n):
        for j in range(n):
            if M[i][j] != 0:
                cpt += 1
    return cpt / 2
```

La complexité est quadratique en n.

Liste des voisins d'un sommet :

```
def voisins(M, i):
    n = len(M[i])
    L = []
    for j in range(n):
        if M[i][j] == 1:
            L.append(j)
    return L
```

Degré d'un sommet :

Pour un graphe non orienté, on veut écrire une fonction `degre(M, i)` retournant le degré du sommet i. M est la matrice d'adjacence. On propose une fonction dans le cadre d'un graphe non pondéré. On rappelle que s'il y a une boucle, on doit compter deux fois la boucle.

```
1  def degre(M, i):
2      n = len(M[i])
3      cpt = 0
4      for j in range(n):
5          if M[i][j] == 1 and i == j:
6              cpt += 2 #cas de la boucle
7          elif M[i][j] == 1 and i != j:
8              cpt += 1 #cas d'une arete
9      return cpt
```

2 - Matrice d'adjacence et graphe orienté

a) Graphe non-pondéré

Règles de construction de la matrice d'adjacence :

Définition :

S'il existe un arc allant de i vers j, alors $M[i,j] = 1$

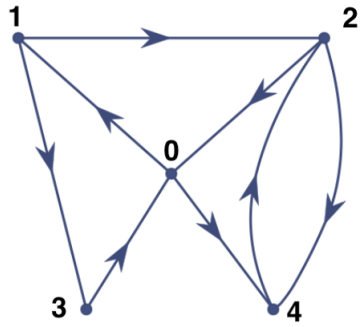
S'il n'existe pas d'arc allant de i vers j, alors $M[i,j] = 0$

Propriété :

Pour un graphe sans boucle, tous les éléments de la diagonale sont nuls : $M[i,i] = 0$

Pour un graphe non orienté, la matrice n'est pas nécessairement symétrique puisqu'on peut avoir $M[i,j] \neq M[j,i]$

Exemple :



La matrice d'adjacence pour ce graphe est :

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

On peut alors la représenter en Python à l'aide d'une liste de listes :

```
1  M = [[0,1,0,0,1],[0,0,1,1,0], [1,0,0,0,1], [1,0,0,0,0], [0,0,1,0,0]]
```

Remarque 1 :

Contrairement à un graphe non-orienté, la matrice d'adjacence d'un graphe orienté n'est pas symétrique.

b) Graphe pondéré

Définition :

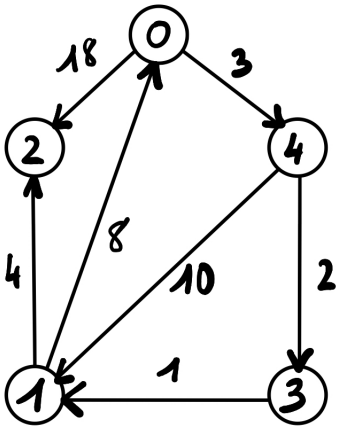
Règles de construction de la matrice d'adjacence :

S'il existe un arc allant de i vers j alors $M[i,j] = d$ où d est le poids de l'arc.

S'il n'existe pas d'arc allant de i vers j alors $M[i,j] = \infty$ (on affecte la valeur ∞ pour différencier d'un arc de poids nul)

Pour un graphe sans boucle, les éléments de la diagonale sont nuls : $M[i,i] = 0$

Exemple :



La matrice d'adjacence pour ce graphe est :

$$\begin{pmatrix} 0 & \infty & 18 & \infty & 3 \\ 8 & 0 & 4 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 1 & \infty & 0 & \infty \\ \infty & 10 & \infty & 2 & 0 \end{pmatrix}$$

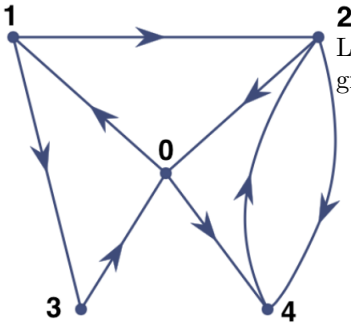
On voit par exemple très bien à l'aide de cette matrice qu'aucun arc ne part du sommet numéro 2. Le graphe n'est pas fortement connexe.

On peut alors la représenter en Python à l'aide d'une liste de listes :

```
M = [[0,'inf',18,'inf',3],[8,0,4,'inf','inf'],['inf','inf',0,'inf','inf'],
['inf',1,'inf',0,'inf'],['inf',10,'inf',2,0]]
```

Exemple 8 :

On dispose d'un graphe orienté, pondéré ou non, représenté par sa matrice d'adjacence. On note M la matrice d'adjacence et N le nombre de sommets. On veut écrire quelques fonctions de base. On reprend l'exemple d'un graphe non pondéré :



La matrice d'adjacence pour ce graphe est :

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

- ☐ 1 - Déterminer le nombre de somment
- ☐ 2 - Déterminer le nombres d'arcs
- ☐ 3 - Déterminer les successeurs d'un sommet `i`
- ☐ 4 - Déterminer les prédécesseurs d'un sommet `i`
- ☐ 5 - En déduire le degré d'un sommet.

Nombre de sommets du graphe :

On a simplement `N = len(M)`. Complexité à coût constant.

Nombre d'arcs du graphe :

Pour un graphe orienté, le nombre d'arcs est égal à la somme de tous les coefficients non nuls de la matrice d'adjacence. (dans le cas d'un graphe pondéré, ce serait tous les coefficients non nuls et non égaux à `'inf'`).

```
def nombre_arcs(M):
    n = len(M)
    cpt = 0
    for i in range(n):
        for j in range(n):
            if M[i][j] != 0:
                cpt += 1
    return cpt
```

La complexité est quadratique en n.

Liste des successeurs d'un sommet :

Pour un graphe orienté, on veut écrire une fonction `successeurs(M, i)` retournant la liste des successeurs du sommet i. Il s'agit des éléments de coefficient non nul sur la **ligne** i de la matrice d'adjacence. On note M cette matrice.

```
def successeurs(M, i):
    n = len(M[i])
    L = []
    for j in range(n):
        if M[i][j] == 1:
            L.append(j)
    return L
```

Liste des prédécesseurs d'un sommet :

Pour un graphe orienté, on veut écrire une fonction `predecesseurs(M, i)` retournant la liste des prédécesseurs du sommet i. Il s'agit de des coefficients non nuls de la **colonne** i de la matrice d'adjacence. On note M cette matrice.

```
def predecesseurs(M, i):
    n = len(M)
    L = []
    for j in range(n):
        if M[j][i] == 1:
            L.append(j)
    return L
```

Degré d'un sommet :

Pour un graphe orienté, on a $d(S) = d_+(S) + d_-(S)$. On peut alors proposer, pour un graphe sans boucle :

```
def degre(M, i):
    L1 = successeurs(M, i)
    L2 = predecesseurs(M, i)
    return len(L1) + len(L2)
```

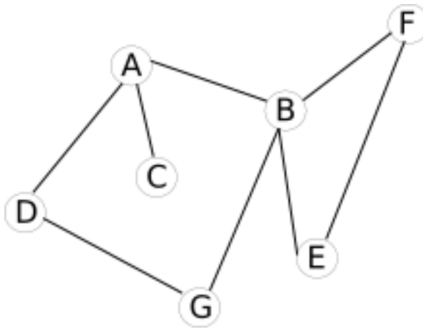
3 - Liste d'adjacence

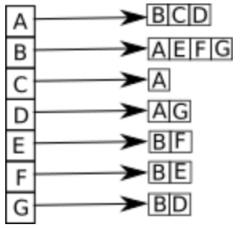
a) Graphe non orienté

Il est également courant de décrire un graphe à l'aide d'une liste d'adjacence. Pour un graphe non orienté, on procède de la manière suivante :

- On commence par définir une liste contenant les sommets du graphe.
- A chaque élément de cette liste, on associe une autre liste qui contient les sommets liés à cet élément.

Reprenons l'exemple de la cartographie et associons une liste d'adjacence au graphe :





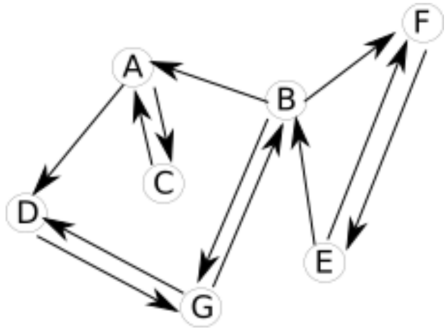
liste d'adjacence du graphe cartographie

b) Graphe orienté

Pour un graphe orienté, on procède de la manière suivante :

- On commence par définir une liste contenant les sommets du graphe.
- A chaque élément de cette liste, on associe deux autres listes : la liste des successeurs de cet élément et la liste des prédécesseurs de cet élément.

Reprenons l'exemple de la cartographie orientée et associons une liste d'adjacence au graphe :

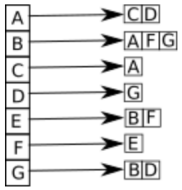


graphe orienté cartographie

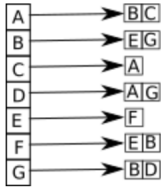
Il est en fait beaucoup plus pratique de représenter en Python la liste d'adjacence par des dictionnaires.

c) Rappels sur les dictionnaires

Dans un dictionnaire, on associe une valeur à une **clé** et non pas à un **indice**.



liste d'adjacence successeurs du graphe orienté cartographie



liste d'adjacence prédécesseurs du graphe orienté cartographie

— Le type de la clé peut être un entier, un nombre flottant, une chaîne de caractères mais pas une liste ou un tableau.

— Le type de la valeur associée à la clé peut être quelconque.

Les éléments d'un dictionnaire ne sont pas ordonnés. On ne peut pas utiliser un indice comme pour les listes pour accéder à un élément. Chaque élément du dictionnaire est identifié par une clé unique. On utilise alors la clé pour rechercher la valeur correspondante du couple (clé, valeur).

En Python, les éléments sont séparés par des accolades. On définit alors un élément du dictionnaire dans Python en précisant la clé, suivie de " :" et de la valeur associée.

Création d'un dictionnaire vide

```
dico = {} #creation dictionnaire vide
dico = dict() #autre maniere de creer un dictionnaire vide
```

Création d'un dictionnaire non vide

```
dico = {"MPSI1":48, "MPSI2":48}
```

Ajout d'un couple (clé, valeur)

On ajoute un élément en précisant la nouvelle clé et la valeur associée : Syntaxe : `dico[cle]=valeur`

```
dico = {"MPSI1":48, "MPSI2":48}
print(dico)
dico["MP2I"] = 47
print(dico)
```

Accès à un élément

On accède aux éléments à l'aide de la clé. Syntaxe : `dico[cle]`

```
dico = {"MPSI1":48, "MPSI2":48}
print(dico["MPSI1"])
```

Changement d'une valeur

Les dictionnaires sont des objets mutables. On peut changer la valeur associée à une clé. Syntaxe : `dico[cle]=valeur`

```
dico = {"MPSI1":48, "MPSI2":48}
print(dico)
dico["MPSI1"]=46
print(dico)
```

Taille d'un dictionnaire

La syntaxe est la même que pour toutes les collections : `len(dico)`

```
dico = {"MPSI1":48, "MPSI2":48}
print(len(dico))
```

Présence d'une clé dans un dictionnaire

La syntaxe est la même que pour la présence d'un élément dans un collection : `cle in dico`

```
dico = {"MPSI1":48, "MPSI2":48}
print("MPSI1" in dico)
print("MP2I" in dico)
```

Copie d'un dictionnaire

La syntaxe est la même que pour toutes les collections : `dico.copy()`

```
dico = {"MPSI1":48, "MPSI2":48}
d = dico.copy()
d["MP2I"]=47
print(d)
print(dico)
```

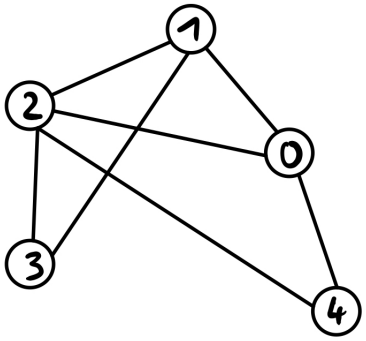
Parcours d'un dictionnaire

On fait une itération sur les clés du dictionnaire et on peut alors accéder aux clés ou aux valeurs associées :

```python
dico = {"MPSI1":48, "MPSI2":48}
for cle in dico:
    print(cle)
    print(dico[cle])
```

d) Cas d'un graphe non orienté et non pondéré

Les clés du dictionnaire représentent les sommets. La valeur de chaque clé correspond à la liste des sommets adjacents. Exemple :

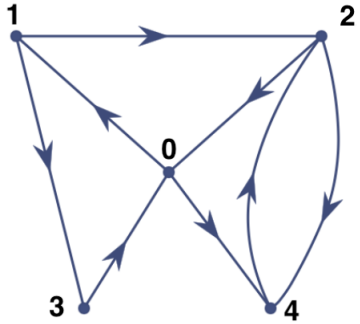


Le dictionnaire permettant de décrire ce graphe est :

```python
dico = {0:[1, 2, 4], 1:[0, 2, 3], \
2:[0, 1, 3, 4], 3:[1, 2],
4:[0,2]}
```

e) Cas d'un graphe orienté et non pondéré

Les clés du dictionnaire représentent les sommets. La valeur de chaque clé correspond par exemple à la liste des successeurs. Exemple :

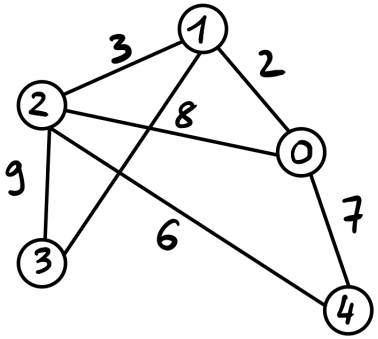


Le dictionnaire permettant de décrire ce graphe par cette méthode est :

```python
dico = {0:[1, 4], 1:[2,3], \
2:[0, 4], 3:[0],
4:[2]}
```

f) Cas d'un graphe non orienté et pondéré

Les clés du dictionnaire représentent les sommets. La valeur de chaque clé correspond à un dictionnaire dont les clés sont les sommets adjacents et la valeur de la clé est le poids de l'arête entre le premier sommet (première clé) et le deuxième sommet (deuxième clé). Exemple :



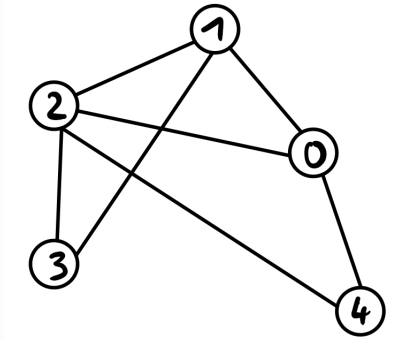
Le dictionnaire permettant de décrire ce graphe est :

```python
dico = {0:{1:2, 2:8, 4:7}, \
1:{0:2, 2:3, 3:8},\
2:{0:8, 1:3, 3:9, 4:6} \
3:{1:8, 2:9}, 4:{0:7,2:6}}
```

On peut accéder aux différents poids des arêtes avec la syntaxe `dico[i][j]`. Exemple : `dico[0][4] = 7`

Exemple 9 :

On ne s'intéresse qu'au graphe non orienté dans cette partie. On notera `dico` le dictionnaire décrivant le graphe. On repart de l'exemple :



Le dictionnaire permettant de décrire ce graphe est :

```python
dico = {0:[1, 2, 4], 1:[0, 2, 3], \
2:[0, 1, 3, 4], 3:[1, 2],  4:[0,2]}
```

- Déterminer le nombre de sommets du graphe
- 2 - Déterminer la liste des voisins d'un sommet `i`
- 3 - Déterminer le degré d'un sommet `i`.

Nombre de sommets du graphe :

On a simplement `N = len(dico)`. Complexité à coût constant.

Nombre d'arêtes du graphe :

Pour un graphe non orienté, le nombre d'arêtes est égal à la demi-somme de tous les éléments présents dans les listes valeurs des clés du dictionnaire.

```
def nombre_arete(dico):
    n = len(dico)
    cpt = 0
    for cle in dico:
        cpt += len(dico[cle])
    return int(cpt / 2)
```

Liste des voisins d'un sommet :

Pour un graphe non orienté, on veut écrire une fonction `voisins(dico, i)` retournant la liste des sommets voisins du sommet i.

```
def voisins(dico, i):
    L = []
    for elt in dico[i]:
        L.append(elt)
    return L
```

Degré d'un sommet :

Pour un graphe non orienté, on veut écrire une fonction `degre(dico, i)` retournant le degré du sommet i. On propose une fonction dans le cadre d'un graphe non pondéré sans boucle.

```
def degre(dico, i):
    return len(dico[i])
```