IF4a – Graphes : Vocabulaires et représentations

Le mathématicien suisse Leonhard Euler montre, en 1735, l'impossibilité de passer une fois et une seule par chacun des sept ponts de la ville de Königsberg en revenant à son point de départ. Ce problème est souvent considéré comme la naissance de la théorie des graphes. Le physicien Gustav Kirchhoff utilise les graphes en 1847 pour résoudre un problème d'électricité qu'Arthur Cayley élargira. On trouve des applications de cette théorie dans d'innombrables domaines comme les réseaux informatiques…

Objectifs :
- Connaître le vocabulaire des graphes
- Comprendre la structure relationnelle d'un graphe.
- Être capable d'implémenter un graphe.

I – Vocabulaire

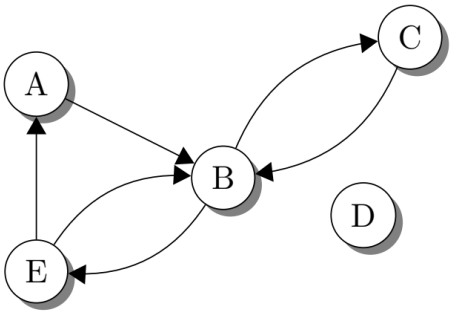
I-1) Vocabulaire général

- Graphe

Un graphe est un ensemble de sommets reliés entre eux par des arcs. Par exemple sur une page « internet » chaque page forme un sommet et les liens permettant de naviguer d'une page à l'autre constituent les arcs.

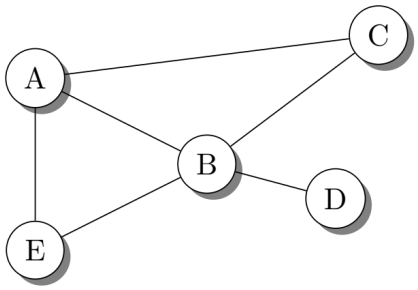
- Graphe orienté

Voici un exemple de graphe possédant cinq sommets et six **arcs**. Sur cet exemple il y un arc qui va de a vers b, mais il n'y en a pas qui va de b vers a. On parle de graphe orienté lorsqu'on distingue ainsi un sens pour les arcs.



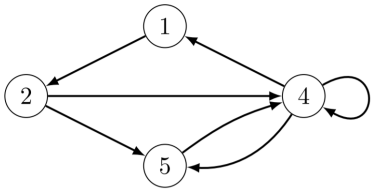
- Graphe non orienté

Lorsqu'en revanche le sens des arcs n'est pas significatif, c'est-à-dire que l'on s'intéresse uniquement à la présence d'un arc entre deux sommets, on parle de graphe non orienté. On préfère alors parler d'**arêtes** au lieu d'arcs dans ce cas. Il y a donc 5 sommets et 6 arêtes dans notre cas.



- Boucle

Une boucle est une arête qui a pour extrémités les mêmes sommets.



- Ordre

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

- Degré

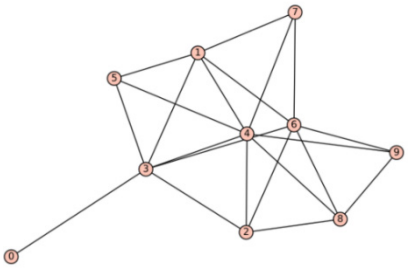
Le nombre de voisins d'un sommet est son degré, il est noté $d(sommet)$. C'est le nombre d'arêtes liées à ce sommet. Si le sommet à un degré « 0 » on dit qu'il est isolé.

Dans le cas d'un graphe orienté on parle de :

- Degré entrant : nombre d'arcs arrivant au sommet. Le degré entrant d'un sommet A se note $d_+(A)$.
- Degré sortant : nombre d'arcs sortant du sommet. Le degré sortant d'un sommet A se note $d_-(A)$.

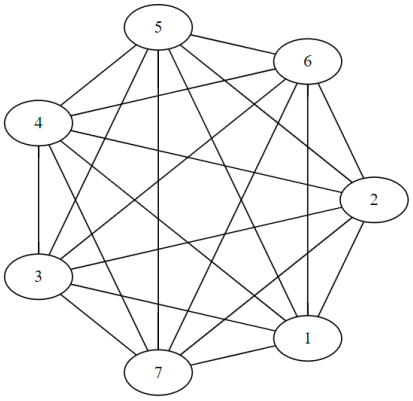
- Graphe simple

Un graphe est simple si deux sommets quelconques sont reliés au plus par une arête



- Graphe complet

Si chacun des sommets est relié directement à tous les autres, on parle de graphe complet.

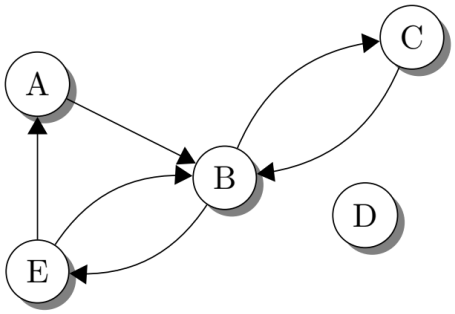


I-2) Voisinage et chemin

- Chemin

Lorsqu'il y a un arc d'un sommet s vers un sommet t, on dit que t est adjacent à s. Les sommets adjacents à s sont également appelés les voisins de s.

Dans un graphe donné, un chemin reliant un sommet s à un sommet t est une séquence finie de sommets reliés deux à deux par des arcs menant de s à t.



Sur le schéma pour aller de E à C on peut prendre le chemin :

$$E \to B \to C \; mais \; aussi \; E \to A \to B \to C$$

- Chemin simple ou élémentaire

Un chemin est dit simple s'il n'emprunte pas deux fois le même arc et élémentaire s'il ne passe pas deux fois par le même sommet. (On parle aussi de chaîne pour les graphes orientés)

- Cycle

Un chemin simple reliant un sommet à lui-même est appelé un cycle : c'est le cas par exemple de $A \to B \to E \to A$

- Longueur et Distance

La longueur d'un chemin est le nombre d'arcs qui constituent ce chemin.
La distance entre deux sommets est la longueur du plus court arc reliant ces deux sommets.

- Connexité

Un graphe est connexe si pour tout couple de sommets il existe un chemin reliant ces deux sommets.

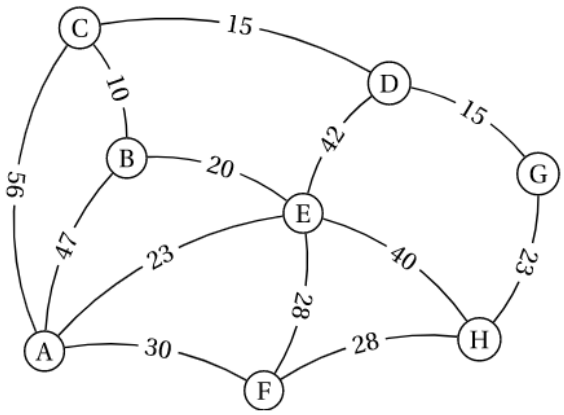
I-3) Poids et étiquette

On peut ajouter des informations sur les graphes comme le poids et les étiquettes.

- Un graphe est pondéré si un nombre, un poids, est associé à chaque arête ou à chaque arc.
- Un graphe est étiqueté si un texte, une étiquette, est associé à chaque arête ou à chaque arc.

Dans un réseau routier par exemple, un poids peut être le nombre de kilomètres d'une route liant deux lieux, une étiquette peut être le nom de cette route.

Exemple de graphes pondérés :

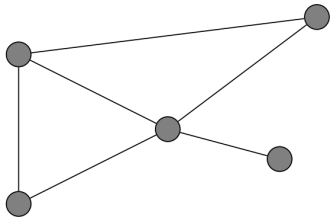


II – Représentation schématique

II-1) Principe

La manière la plus simple de représenter un graphe est de faire un dessin. Les sommets sont représentés par des points, les arêtes par des lignes, chacune reliant deux points.

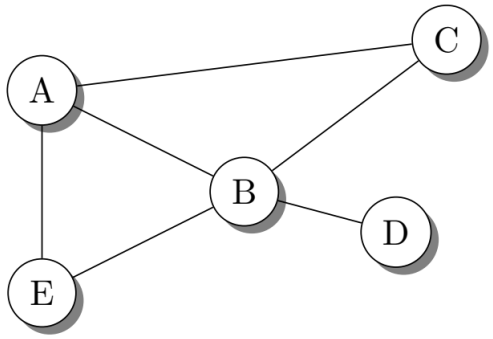
Le graphe suivant est non orienté et a cinq sommets. Il est connexe et d'ordre 5.



Les sommets sont représentés par des points mais attention, un sommet peut représenter une entité complexe : une personne dans un réseau social, un carrefour dans un réseau routier, un routeur dans un réseau internet…

II-2) Exemples

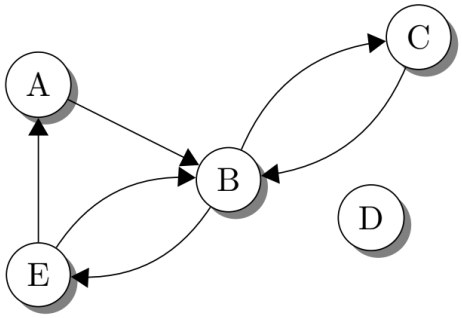
a) Graphe non orienté



Il est plus pratique de nommer les sommets : A B C D E.

- Le degré du sommet A est 3.
- Ses voisins sont B, C et E.
- L'arête [AB] a pour extrémités A et B.
- [AC] est un chemin de longueur 1, [AEBD] est un chemin de longueur 3, [ABD] est un chemin de longueur 2. [ACBEA] est un cycle.
- L'ensemble des sommets est S = {A, B, C, D, E}.
- L'ensemble des arêtes est A = {{A, B}, {A, C}, {A, E}, {B, C}, {B,D}, {B,E}}.

b) Graphe orienté

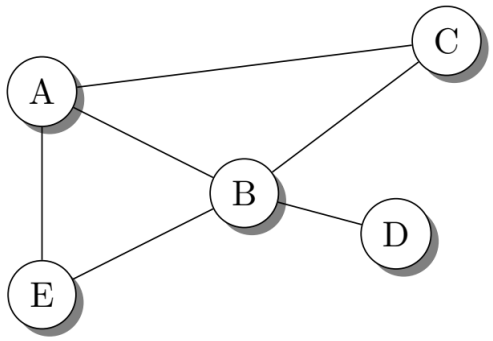


- Ce graphe est orienté, non connexe.
- Le sommet D est isolé.
- Le degré entrant de B est 3, le degré sortant est 2…

III – Listes et matrices d'adjacence

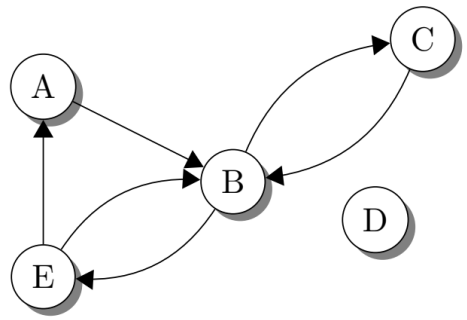
III-1) Listes

On peut représenter un graphe non orienté en précisant pour chacun de ses sommets la liste de ses voisins. Ces listes s'appellent des listes d'adjacence. L'ordre d'écriture n'a pas d'importance. Dans le cas de graphes orientés, on peut présenter des listes de successeurs ou des listes de prédécesseurs ou les deux. On obtient pour l'exemple suivant les listes d'adjacences suivantes :



- A: B, C, E
- B: A, C, D, E
- C: A, B
- D: B
- E: A, B

On obtient pour l'exemple de graphe orienté listes de successeurs suivantes :



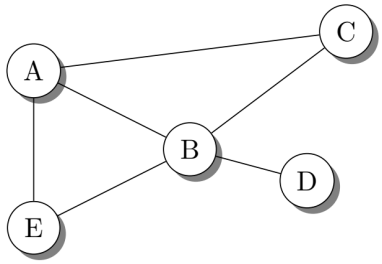
- A: B
- B: C, E
- C: B
- D:
- E: A, B

III-2) Matrices

En mathématiques, on peut associer à un graphe une matrice carrée (n,n), ou un tableau, où n est le nombre de sommets. Les sommets sont numérotés de 1 à n. Il s'agit d'un tableau à n lignes et n colonnes.

A l'intersection d'une ligne i et d'une colonne j le nombre représente la présence ou l'absence d'une arête entre les sommets i et j : 1 pour la présence, 0 pour l'absence.

- Graphe non orienté



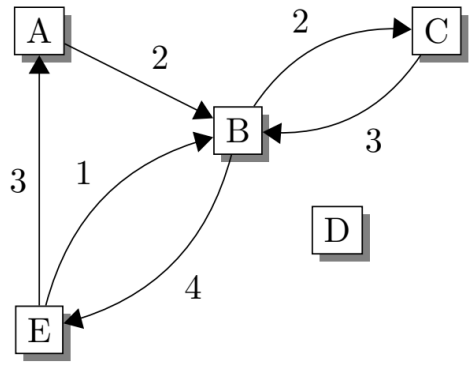
	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	1
C	1	1	0	0	0
D	0	1	0	0	0
E	1	1	0	0	0

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

La diagonale ne contient que des 0 et est un axe de symétrie du tableau. (Il faudrait des boucles pour avoir des 1). On dit que la matrice est symétrique.

Le même graphe peut être représenté par des matrices différentes. Elles dépendent de l'ordre des sommets qui est pris en compte. La matrice représentative du graphe est appelée matrice d'adjacence.

- Graphe orienté



$$\begin{pmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 4 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Pour tenir compte de la pondération des arcs, on remplace les « 1 » par la valeur de la pondération dans la matrice. Par exemple le point B vérifie pour ces successeurs :

- B : (C,2), (E,4) et 0 pour le reste.

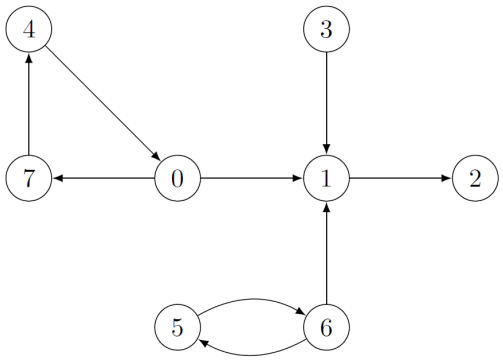
IV) Représentation d'un graphe en python

IV-1) Implémentation par matrice d'adjacence

a) Principe

En Python, on représente généralement une matrice (n, n) par une liste contenant n listes de longueur n. Chacune de ces n listes représente une ligne de la matrice. Les éléments de chacune de ces n listes sont les n coefficients d'une ligne. Si on numérote les lignes de la matrice de 1 à n, il faut faire attention aux indices qui en Python commencent à 0.

Pour simplifier le problème, numérotons les sommets par des chiffres et prenons comme exemple le graphe orienté suivant G1 :



Celui-ci est représenté par huit sommets tel que $S = \{0, \dots, 7\}$
Et si on prend les successeurs on a 9 arêtes tel que :

$$A = \{(0,1), (0,7), (1,2), (3,1), (4,0), (5,6), (6,1), (6,5), (7,4)\}$$

On définit donc un graphe en écrivant par exemple la matrice 8×8 tel que :

$$G1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Comme le graphe n'a pas de boucles on a une diagonale remplie de zéro. La première ligne se lit : « Le sommet « 0 » a pour successeurs les sommets « 1 » et « 7 ».

Dans le cas d'un graphe pondéré, le principe est le même. Pour un graphe pondéré, les coefficients 1 sont remplacés par les poids.

b) Algorithme

L'implémentation consiste à utiliser une matrice de booléens de taille $(n \times n)$, dont la case d'indice (i, j) vaut « True » si et seulement si l'arête (i, j) appartient à A. Notons que le graphe est non orienté si et seulement si la matrice qui lui est associé est symétrique.

Le principal intérêt de cette représentation est de pouvoir vérifier en temps constant si une arête existe. De la même manière, on a maintenant la possibilité d'ajouter ou de retirer une arête en temps constant, que le graphe soit orienté ou non.

Pour construire un certain graphe, on peut commencer par construire une matrice où tous les booléens sont « False » :

```
def nouveau(n):
    return([[False]*n for i in range(n)])
```

Ensuite on peut ajouter des arêtes au graphe, en donnant la valeur « True » à certains éléments de cette matrice. Ainsi on peut ajouter une arête entre les sommets 4 et 0 par la fonction :

```
def ajoute_arete(G,i,j):
    G[i][j]=True
```

De même on pourrait en supprimer par :

```
def supprime_arete(G,i,j):
    G[i][j]=False
```

On peut aussi avoir besoin de rajouter ou supprimer des sommets. On peut supposer que le sommet ajouté se place en dernière position mais si ce n'est pas le cas, il faudra en tenir compte.

```
import random as rd

def nouveau(n):
    return([[False]*n for i in range(n)])

def ajoute_arete(G,i,j):
    G[i][j]=True

def supprime_arete(G,i,j):
    G[i][j]=False

def ajoute_sommet(G):
    n=len(G)
    for i in range(n):
        G[i].append(False) #On ajoute "False" à la fin de chaque ligne
    G.append([False]*(n+1)) #On ajoute une ligne de False pour le sommet supplémentaire 'n+1'
```

```
def supprime_sommet(G,i):
    n = len(G)
    for k in range(n):
        G[k]=G[k][:i]+G[k][(i+1):] #On supprime la colonne i
    del G[i] #On supprime la ligne i
```

```
n=3
G = nouveau(n)
for i in range(n):
    for j in range(n):
        if rd.random()>0.5:
            G[i][j]=True
print(G)
supprime_sommet(G,2)
print(G)
```

```
[[True, False, False], [True, False, True], [True, True, False]]
[[True, False], [True, False]]
7
```

IV-2) Implémentation par liste d'adjacence

a) Principe

Cette implémentation consiste à représenter le graphe par une liste de n listes, où la case d'indice i contient la liste des sommets j tels que (i ; j) soit une arête du graphe (appelée liste d'adjacence du sommet i). On peut se contenter de cette liste puisque la fonction « len » permet de récupérer la valeur de n. On implémente donc le graphe G1 de la manière suivante :

```
G1 = [[1 ,7] ,[2] ,[] ,[1] ,[0] ,[6] ,[1 ,5] ,[4]]
```

Par exemple [1,7] représente les successeurs de « 0 »…

Dans l'exemple ci-dessus, les listes d'adjacences sont triées par ordre croissant. Ce n'est pas du tout une nécessité absolue mais c'est pratique.

b) Algorithme

```
import random as rd

def nouveau(n):
    return([[] for i in range(n)]) #Création d'un graphe à n sommets
                                   #initialement sans arête
def ajoute_arete(G,i,j):
    G[i].append(j)

def supprime_arete(G,i,j):
    G[i].remove(j)

#Attention, si le graphe est non orienté il faudra supprimer l'arête (j,i)
#mais aussi (i,j)
def ajoute_arete_no(G,i,j):
    G[i].append(j)
    G[j].append(i)
def supprime_arete_no(G,i,j):
    G[i].remove(j)
    G[j].remove(i)

#Pour ajouter un sommet, on va rajouter une liste vide au bout de G
def ajoute_sommet(G):
    G.append([])
```

Pour supprimer un sommet i du graphe, c'est un peu plus pénible. En effet, à partir du moment où S contient des entiers consécutifs, un renommage des sommets {i+1,…,n+} est nécessaire. On définit à cet effet une fonction de réindexation. On met à jour ensuite toutes les listes et on supprime la i-ème liste.

```
#Pour ajouter un sommet, on va rajouter une liste vide au bout de G
def ajoute_sommet(G):
    G.append([])

#Pour supprimer un sommet on utilise une fonction reindexe.
def reindexe(i,j):
    if j>i:
      return j-1
    else :
      return j

def supprime_sommet(G,i):
    n = len(G)
    for k in range(n):
        G[k]=[reindexe(i,x) for x in G[k]] #On met la valeur x ou x-1 dans G[k]
    G.pop(i)                               #On enlève le noeud tout en affichant G

def supprime_sommet2(G,i):
    n = len(G)
    for k in range(n):
        if i in G[k] : G[k].remove(i) # On efface les arcs reliés au noeud i
        G[k]=[reindexe(i,x) for x in G[k]] #On met la valeur x ou x-1 dans G[k]
    G.pop(i)                               #On enlève le noeud tout en affichant G

n=4
G = nouveau(n)
print(G)
ajoute_arete(G,0,1)
ajoute_arete(G,0,2)
ajoute_arete(G,1,2)
ajoute_arete(G,2,3)
ajoute_arete(G,3,0)
print(G)
#supprime_arete(G,0,1)
#supprime_arete(G,1,2)
#supprime_sommet(G,1) #A utiliser après les suppressions d'arete
supprime_sommet2(G,1)
print(G)
```

IV-3) Implémentation par dictionnaire

En informatique, un dictionnaire est une structure de donnée permettant des stocker de couples de la forme (clé, valeur), les clés étant nécessairement deux à deux distinctes. L'utilité principale du dictionnaire est de représenter des graphes dont les sommets sont quelconques (pas des entiers comme précédemment) par listes d'adjacence.

Ainsi l'ensemble des sommets du graphe est l'ensemble des clés du dictionnaire.

Les dictionnaires dans le cas de manipulation de graphes sont pratiques si les sommets sont différents des entiers. Voici un exemple simple avec de petites fonctions.

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

def deg(g,s): #g est un graphe, s est un sommet , calcule le voisinage de s
  if s in g:
    return len(g[s])

def sommets(g,s):  #Donne le voisinage de s
  if s in g:
    return g[s]

def ajoute_sommet_i(g,s): #ajoute un sommet isolé
  g[s]=[]

print(deg(Z,"A"))
print(sommets(Z,"A"))
ajoute_sommet_i(Z,"G")
print(Z)
```

```
2
['B', 'E']
{'A': ['B', 'E'], 'B': ['A', 'C', 'E'], 'C': ['B'], 'D': [], 'E': ['A', 'B'], 'G': []}
```