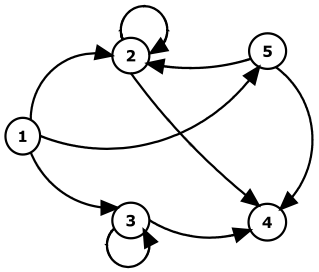
# COMMENT REPRESENTER L'INFORMATION CONTENUE DANS UN GRAPHE ?

Derrière cette notion d'information, nous avons bien sûr en tête de manipuler un graphe en informatique. Le dessin d'un graphe, sa représentation graphique est en général parlant, mais ce n'est pas une information structurée, manipulable simplement par un programme.

Bien sûr, il n'y a pas une seule de façon de représenter l'information contenue dans un graphe… Selon le contexte, certaines représentations peuvent être plus intéressantes que d'autres.

$G = (S, A)$

- $S = \{1,2,3,4,5\}$
- $A = \{(1,2), (1,3), (15); (2,2), (2,4), (3,3), (3,4), (5,2), (5,4)\}$




## PAR UN ENSEMBLE D'ARCS, EN SE REFERANT A LA DEFINITION

Mathématiquement, un graphe est défini par deux ensembles $S$ et $A$
La liste des arcs $A$ peut-être suffisante s'il n'y a pas de sommet isolé.

**Exemple en Python :** $\boldsymbol{A = [[1,2],[1,3],[1,5],[2,2],[2,4],[3,3],[3,4],[5,2],[5,4]]}$

La **place occupée** par l'information stockée est $2m$.
**Inconvénient** : peu structuré, accès aux successeurs ou prédécesseurs d'un sommet peut aisé.

## MATRICE D'ADJACENCE

Nous l'avons abordé en S1 (R1.07), un graphe peut être représenté par une matrice.

On suppose au préalable que les sommets ont été numérotés : $S = \{s_1, s_2, \dots, s_n\}$

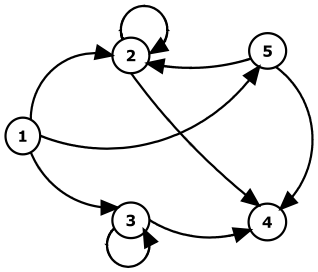
La matrice d'adjacence est une matrice d'ordre $n$ ($n$ lignes, $n$ colonnes) de terme général $m_{ij}$ (terme de la i[ème] ligne et j[ème] colonne) défini par :

$$m_{ij} = \begin{cases} 1 \; si \; (s_i, s_j) \in S \\ 0 \; si \; (s_i, s_j) \notin S \end{cases}$$

Si le graphe n'est pas orienté, on obtiendra une matrice symétrique ($m_{ij} = m_{ji}$)

$$m_{ij} = \begin{cases} 1 \; si \; \{\boldsymbol{s_i}, \boldsymbol{s_j}\} \in S \\ 0 \; si \; \{\boldsymbol{s_i}, \boldsymbol{s_j}\} \notin S \end{cases}$$

*EXEMPLE*




Matrice d'adjacence

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

**Exemple en Python :**
En Python, nous avons appris à manipuler les matrices en utilisant des **listes de listes**. Si nous avons beaucoup de calculs à faire sur ces matrices nous pouvons aussi utiliser les **Tableaux (array)** de la bibliothèque **Numpy**

```
# Représentation de la matrice d'adjacence par une liste de liste
M=[[0,1,1,0,1],[0,1,0,1,0],[0,0,1,1,0],[0,0,0,0,0],[0,1,0,1,0]]

# Représentation par un tableau numpy
import numpy as np
A=np.array(M)
```

**Place occupée** par l'information stockée : $n^2$
**Inconvénient** : peut occuper beaucoup de place inutile si le graphe présente beaucoup de sommets et relativement peu d'arcs (graphe creux).

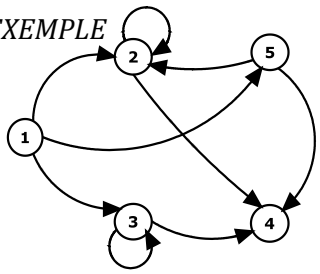
## LISTE D'ADJACENCE

Avec la matrice d'adjacence, pour chaque sommet nous avons toujours $n$ valeurs (binaires).
S'il y a peu d'arcs, il vaut mieux associer à chaque sommet la liste de ses successeurs

L'objectif de la représentation par liste d'adjacence est donc :

- d'éviter d'utiliser des matrices surdimensionnées (matrices creuses) dans le cas où le nombre d'arcs est relativement faible,
- de proposer une représentation plus structurée que la simple liste d'arcs, qui permet un accès aux successeurs et prédécesseurs plus aisée.

*EXEMPLE*




| | |
|---|---|
| 1 | 2,3,5 |
| 2 | 2,4 |
| 3 | 3,4 |
| 4 | |
| 5 | 2,4 |

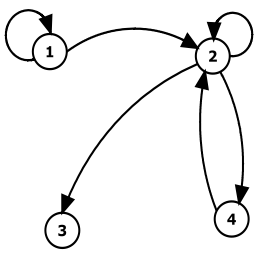
**Exemple en Python :**
En Python, nous pourrions utiliser une liste de listes, chaque liste correspondant aux successeurs du sommet correspondant. Nous avons cependant la structure plus adaptée, la structure de **`dictionnaire`** :

```
# Représentation d'une liste d'adjacence par un dictionnaire
G={1:[2,3,5],2:[2,4],3 :[3,4],4 :[],5 :[2,4]}
```

**Place occupée** par l'information stockée $n + m$

*Remarque : sous Python, il existe bien sûr des bibliothèques spécifiques permettant de manipuler des graphes, igraph, Networkx… Ces bibliothèques sont particulièrement intéressantes pour visualiser les graphes ou pour utiliser des algorithmes classiques… Mais ce n'est pas l'objet de ce cours de les utiliser.*

## AUTRE EXEMPLE




$G = (S, A)$

- $S = \{1,2,3,4\}$
- $A = \{(1,1), (1,2), (2,2), (2,3), (2,4), (4,2)\}$

Matrice d'adjacence

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

**Représentation Python**

```
# Matrice d'adjacence représentée par une liste de liste
M=[[1,1,0,0],[0,1,1,1],[0,0,0,0],[0,1,0,0]]
# Tableau array
import numpy as np
A=np.array(M)
# représentation de la liste d'adjacence par un dictionnaire
G={1:[1,2],2:[2,3,4],3 :[],4 :[2]}
```

# EXERCICES

## EXERCICE 1 : GENERALITE SUR LES DICTIONNAIRES & QCM

Les dictionnaires permettent d'associer des valeurs à des clés. L'accès aux valeurs se fait par les clés, mais contrairement aux listes, il ne peut pas se faire par des indices.

- Création d'un dictionnaire (à trois clés) :
  `Dic1={'nom': 'Dupont', 'prenom': 'Pierre', 'age': 18}`
- Longueur : `len(Dic1)` → `3`
- Accès à une valeur à partir d'une clé : `Dic1['prenom']` → `'Pierre'`
- Définition d'un dictionnaire en compréhention : `Dic={i:2*i for i in range(3)}`
  Contenu de Dic → `{0: 0, 1: 2, 2: 4}`
- Dictionnaire vide : `Dic2={}`
  `len(Dic2)` → `0`
- Ajout d'un couple (clé, valeur) : `Dic2['prenom']='Pierre'`
  Si `Dic2` était le dictionnaire vide, `Dic2` contient alors `{'prenom': 'Pierre'}`
- Suppression d'un couple (clé, valeur) : `del Dic1['prenom']`
- Itération sur un dictionnaire

```
for cle in Dic1.keys():
    print(cle)              →  nom
                               prenom
                               age

for val in Dic1.values():
    print(val)              →  Dupont
                               Pierre
                               18

for (cle,val) in Dic1.items():
    print(cle,'-->',val)    →  nom --> Dupont
                               prenom --> Pierre
                               age --> 18
```

```
for cle in Dic1:        ←
    print(cle)
```

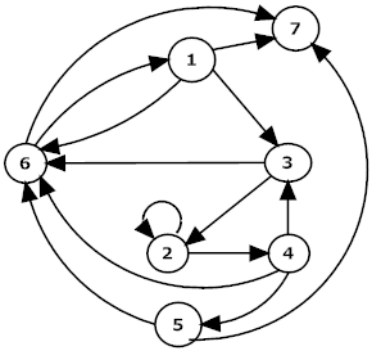
```
list(Dic1.keys()) ou list(Dic1)  →  ['nom','prenom','age']
list(Dic1.values())  →  ['Dupont','Pierre',18]
list(Dic1.items())  →
[('nom','Dupont'),('prenom','Pierre'),('age',18)]
```

Après avoir testé et compris les manipulations précédentes sur Python.

***Test d'entrainement – Python – Dictionnaires :***

## EXERCICE 2 : REPRESENTATION D'UN GRAPHE

On considère le graphe $G = (S, A)$ ci-contre :




Déterminer :

1. $S, A$, l'ordre et la taille du graphe,
2. La matrice d'adjacence de $G$
   *Donner une implémentation sous Python avec une liste de listes*
3. La liste d'adjacences de $G$ sous forme de tableau
   *Donner une implémentation sous Python avec un dictionnaire*

## EXERCICE 3

Soit $G = (S, A)$ un graphe représenté par la matrice d'adjacence ci-dessous :

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

$S = \{1,2,3,4,5,6,7\}$

1. Donner la liste $A$ des arcs du graphe.
2. Représenter graphiquement $G$
3. Donner une liste d'adjacence de $G$ sous forme de tableau et une implémentation sous Python à l'aide d'un dictionnaire

## EXERCICE 4

Mêmes questions que pour l'exercice 1 avec l'exemple suivant.

$G = (S, A)$ est un graphe défini par :

$$S = \{A, B, C, D, E, F\} \quad \text{et} \quad A = \{(B, A), (B, C), (C, A), (C, D), (D, E), (D, F), (E, F)\}$$

## EXERCICE 4 : TEST D'ENTRAINEMENT – PYTHON – REPRESENTATION D'UN GRAPHE

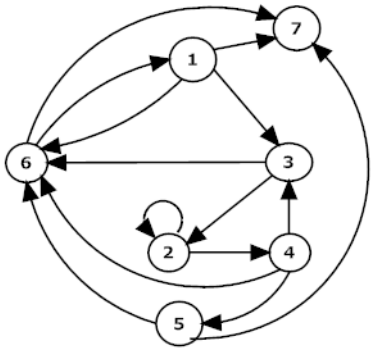
EXERCICE 5 : PYTHON

A. On travaille sous Python en implémentant des graphes par des listes de listes représentant leur matrice d'adjacence. Construire les fonctions suivantes :
   a. $\boldsymbol{succ}(\boldsymbol{M},\boldsymbol{s})$ : retourne la liste des successeurs du sommet $\boldsymbol{s}$ dans le graphe défini par la matrice d'adjacence $\boldsymbol{M}$ (exprimée sous forme de liste de liste)
   b. $\boldsymbol{nb\_succ}(\boldsymbol{M},\boldsymbol{s})$ : retourne le nombre de successeurs du sommet $\boldsymbol{s}$ dans le graphe défini par la matrice d'adjacence $\boldsymbol{M}$ (exprimée sous forme de liste de liste)
   **c.** Mêmes questions avec les prédécesseurs, définir les fonctions $\boldsymbol{pred}(\boldsymbol{M},\boldsymbol{s})$ et $\boldsymbol{nb\_pred}(\boldsymbol{M},\boldsymbol{s})$
B. Mêmes questions dans le cas où le graphe est implémenté par un dictionnaire $\boldsymbol{D}$. Définir les fonctions $\boldsymbol{succ\_dic}(\boldsymbol{D},\boldsymbol{s}), \boldsymbol{nb\_succ\_dic}(\boldsymbol{D},\boldsymbol{s}), \boldsymbol{pred\_dic}(\boldsymbol{D},\boldsymbol{s}), \boldsymbol{nb\_pred\_dic}(\boldsymbol{D},\boldsymbol{s})$
C. Construire une fonction $\boldsymbol{mat\_adj}(\boldsymbol{D})$ qui prend en paramètre $D$ un dictionnaire définissant un graphe et qui renvoie la matrice d'adjacence (sous forme de liste de listes) du graphe correspondant.
D. Réciproque : construire une fonction $\boldsymbol{dico}(\boldsymbol{M})$ qui prend en paramètre une matrice d'adjacence d'un graphe et renvoie le dictionnaire du graphe correspondant.

# CORRIGES

EXERCICE 2 : REPRESENTATION D'UN GRAPHE

On considère le graphe $G = (S, A)$ ci-contre :




Déterminer :

1. $S, A$, l'ordre et la taille du graphe :
   $S = \{1,2,3,4,5,6,7\}$
   $A = \{(1,3),(1,6),(1,7),(2,2),(2,4),(3,2),(3,6),(4,3),(4,5),(4,6),(5,6),(5,7),(6,1),(6,7)\}$
   Ordre du graphe : $n = 7$
   Taille du graphe : $m = 14$
2. La matrice d'adjacence de $G$

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

*Implémentation sous Python avec une liste de listes*

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

3. La liste d'adjacences de $G$ sous forme de tableau

| Sommet | Successeur(s) |
|---|---|
| 1 | 3,6,7 |
| 2 | 2,4 |
| 3 | 2,6 |
| 4 | 3,5,6 |
| 5 | 6,7 |
| 6 | 1,7 |
| 7 | / |

*Implémentation sous Python avec un dictionnaire*

```
Dic={1:[3,6,7]
     ,2:[2,4]
     ,3:[2,6]
     ,4:[3,5,6]
     ,5:[6,7]
      ,6:[1,7]
     ,7:[]}
```

## EXERCICE 3

Soit $G = (S, A)$ un graphe représenté par la matrice d'adjacence ci-dessous :

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

$S = \{1,2,3,4,5,6,7\}$

4. Donner la liste $A$ des arcs du graphe.

$S = \{1,2,3,4,5,6,7\}$

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

1. Donner une liste d'adjacence de $G$ sous forme de tableau et une implémentation sous Python à l'aide d'un dictionnaire

| Sommet | Successeur(s) |
|---|---|
| 1 | 4,5,7 |
| 2 | 3,6 |
| 3 | 2,6 |
| 4 | 7 |
| 5 | 4,7 |
| 6 | / |
| 7 | 5 |

*Implémentation sous Python avec un dictionnaire*

```
Dic={1:[4,5,7]
     ,2:[3,6]
     ,3:[2,6]
     ,4:[7]
     ,5:[4,7]
     ,6:[]
     ,7:[5]}
```

## EXERCICE 3

Mêmes questions que pour l'exercice 1 avec l'exemple suivant.

$G = (S, A)$ est un graphe défini par :

$S = \{A, B, C, D, E, F\}$ et $A = \{(B, A), (B, C), (C, A), (C, D), (D, E), (D, F), (E, F)\}$

1. $S, A$, l'ordre et la taille du graphe :

   $S = \{A, B, C, D, E, F\}$ $A = \{(B, A), (B, C), (C, A), (C, D), (D, E), (D, F), (E, F)\}$

   Ordre du graphe : $n = 6$

   Taille du graphe : $m = 7$

2. La matrice d'adjacence de $G$

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

*Implémentation sous Python avec une liste de listes*

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

3. La liste d'adjacences de $G$ sous forme de tableau

| Sommet | Successeur(s) |
|---|---|
| A | / |
| B | A,C |
| C | A,D |
| D | E,F |
| E | F |
| F | / |

*Implémentation sous Python avec un dictionnaire*

```
Dic={'A':[],
     'B':['A','C'],
     'C':['A','D'],
     'D':['E','F'],
     'E':['F'],
     'F':[]}
```

EXERCICE 5 : PYTHON

A. On travaille sous Python en implémentant des graphes par des listes de listes représentant leur matrice d'adjacence. Construire les fonctions suivantes :

a. $\boldsymbol{succ}(\boldsymbol{M},\boldsymbol{s})$ : retourne la liste des successeurs du sommet $\boldsymbol{s}$ dans le graphe défini par la matrice d'adjacence $\boldsymbol{M}$ (exprimée sous forme de liste de liste)

```
def succ(M,s):
    liste_succ=[]
    for j in range(len(M)):
        if M[s][j]==1:
            liste_succ.append(j)
    return liste_succ
```

b. $\boldsymbol{nb\_succ}(\boldsymbol{M},\boldsymbol{s})$ : retourne le nombre de successeurs du sommet $\boldsymbol{s}$ dans le graphe défini par la matrice d'adjacence $\boldsymbol{M}$ (exprimée sous forme de liste de liste)

```
def nb_succ(M,s):
    nb=0
    for j in range(len(M)):
        if M[s][j]==1:
            nb=nb+1
    return nb
```

Autre méthode :

```
def nb_succ(M,s):
    nb=0
    for val in M[s]:
        if val==1:
            nb=nb+1
    return nb
```

**c.** Mêmes questions avec les prédécesseurs, définir les fonctions $\boldsymbol{pred}(\boldsymbol{M},\boldsymbol{s})$ et $\boldsymbol{nb\_pred}(\boldsymbol{M},\boldsymbol{s})$

```
def pred(M,s):
    liste_pred=[]
    for i in range(len(M)):
        if M[i][s]==1:
            liste_pred.append(i)
    return liste_pred
```

```
def nb_pred(M,s):
    nb=0
    for i in range(len(M)):
        if M[i][s]==1:
            nb=nb+1
    return nb
```

Autre méthode :

```
def nb_pred(M,s):
    nb=0
    for ligne in M:
        if ligne[s]==1:
            nb=nb+1
    return nb
```

**B.** Mêmes questions dans le cas où le graphe est implémenté par un dictionnaire $\boldsymbol{D}$. Définir les fonctions $\boldsymbol{succ\_dic}(\boldsymbol{D},\boldsymbol{s}),\boldsymbol{nb\_succ\_dic}(\boldsymbol{D},\boldsymbol{s}),\boldsymbol{pred\_dic}(\boldsymbol{D},\boldsymbol{s}),\boldsymbol{nb\_pred\_dic}(\boldsymbol{D},\boldsymbol{s})$

```
def succ_dic(D,s):
    return D[s]
```

```
def nb_succ_dic(D,s):
    return len(D[s])
```

```
def pred_dic(dico,s):
    liste_pred=[]
    for cle in dico:
        if s in dico[cle]:
            liste_pred.append(cle)
    return liste_pred
```

```
def nb_pred_dic(dico,s):
    nb=0
    for cle in dico:
        if s in dico[cle]:
            nb=nb+1
    return nb
```

C. Construire une fonction $\boldsymbol{mat\_adj(D)}$ qui prend en paramètre $D$ un dictionnaire définissant un graphe et qui renvoie la matrice d'adjacence (sous forme de liste de listes) du graphe correspondant.

```
def mat(D):
    m=[[0 for i in range(len(D))]for i in range(len(D))]
    for cle in D:
        for val in D[cle] :
            m[cle][val]=1
    return m
```

D. Réciproque : construire une fonction $\boldsymbol{dico(M)}$ qui prend en paramètre une matrice d'adjacence d'un graphe et renvoie le dictionnaire du graphe correspondant.

```
def dico(M):
    n=len(M)
    d={}
    for i in range(n):
        d[i]=[]
        for j in range(n):
            if M [i][j]==1:
                d[i].append(j)
    return d
```