# Chapitre 10
# Graphes

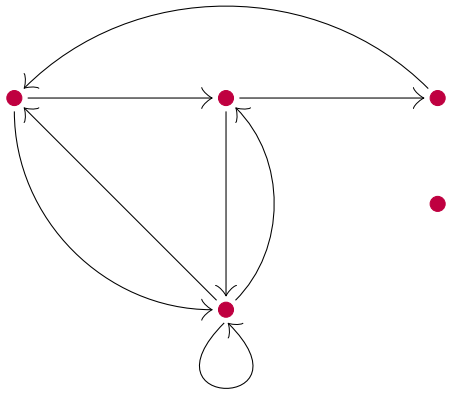


# 1 Définitions

## 1.1 Graphes

### 1.1.1 Graphes orientés

Un **graphe** est un ensemble de **sommets** (ou **nœuds**) qui peuvent être reliés entre eux par des **arêtes** (ou **arcs**).



Plus précisément, une arête est un couple de sommets : si $s$ et $t$ sont deux sommets d'un graphe, le couple $(s,t)$ représente l'arête allant du sommet $s$ vers le sommet $t$.

Un **graphe (orienté)** $\mathcal{G}$ est donc la donnée

- ⋆ d'un ensemble $S$ de **sommets** ;
- ⋆ d'une partie $A$ de $S \times S$, c'est-à-dire d'un ensemble de *couples* de sommets, qui est l'ensemble des **arêtes** ;

le graphe est alors noté $\mathcal{G} = (S, A)$.

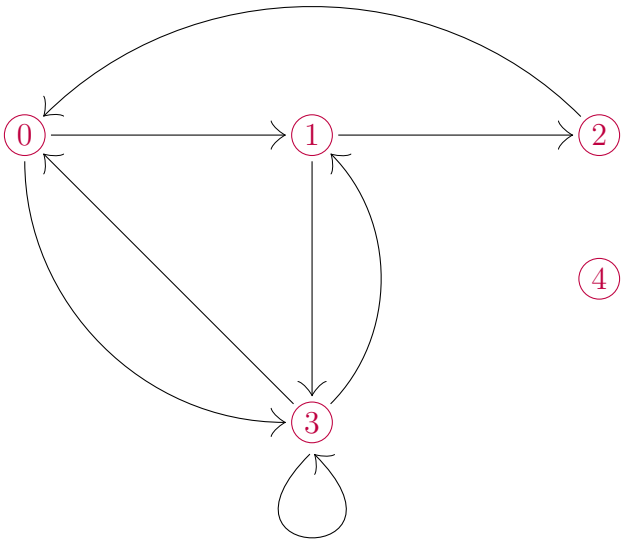



FIGURE 1 – Un exemple $\mathcal{G}_1$ de graphe orienté

Dans le graphe $\mathcal{G}_1$ de la figure 1, l'ensemble des sommets est $S = \{0;1;2;3;4\}$ et l'ensemble des arêtes est

$$A = \Big\{(0,1),(0,3),(1,2),(1,3),(2,0),(3,0),(3,1),(3,3)\Big\}.$$

### 1.1.2 Graphes non orientés

Lorsque le sens des arêtes n'a pas d'importance, on dit que le graphe est **non orienté**.

Plus précisément, un graphe $\mathcal{G} = (S, A)$ est dit **non orienté** lorsque, pour tous sommets $s$ et $t$ de $S$, si $(s,t) \in A$, alors $(t,s) \in A$ aussi.

Dans un graphe non orienté, pour représenter les arêtes $(s,t)$ et $(t,s)$ entre deux sommets $s$ et $t$, on trace un seul trait (sans pointe de flèche) reliant les sommets $s$ et $t$ plutôt qu'une flèche dans chaque sens.

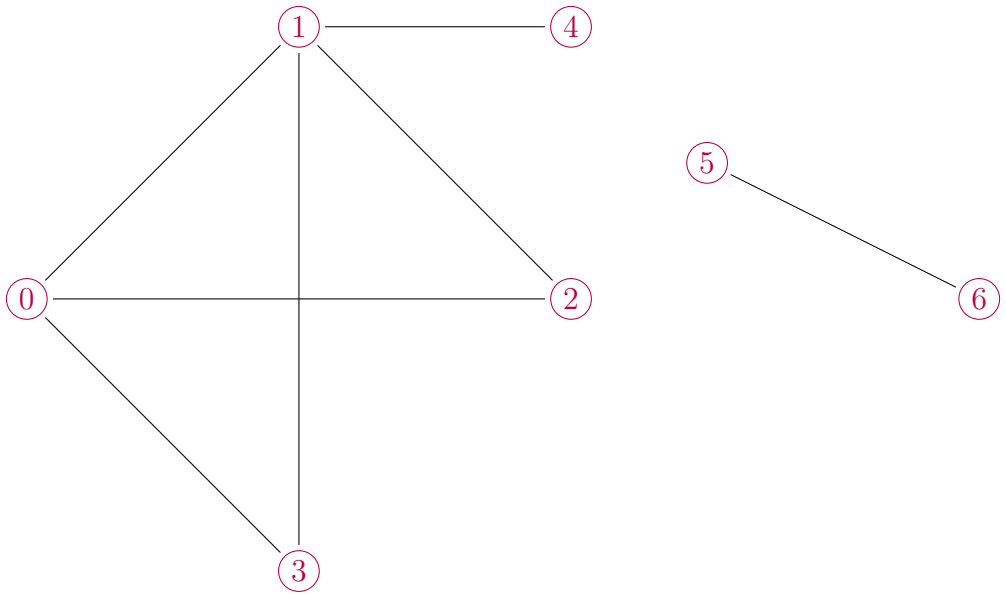



FIGURE 2 – Un exemple $\mathcal{G}_2$ de graphe non orienté

### 1.1.3 Exemples de graphes

Les graphes apparaissent naturellement dans de très nombreux contextes. En voici quelques exemples.

- ⋆ Le graphe du web : le graphe orienté dont les sommets sont les pages web et les arêtes sont les liens entre pages web.
- ⋆ Le réseau ferroviaire : le graphe (non orienté) dont les sommets sont les gares et les arêtes sont les lignes de train.
- ⋆ Le réseau routier : le graphe dont les arêtes sont les routes et les sommets sont les carrefours entre les routes.
- ⋆ Un réseau social : un graphe dont les sommets sont les gens et les arêtes sont les liens d'amitié (virtuelle ou non) entre les gens.
- ⋆ Le graphe dont les sommets sont les différentes configurations d'un jeu (échecs, dames chinoises, jeu de la tablette de chocolat, etc.) et les arêtes sont les coups permettant de passer d'une configuration à une autre.
- ⋆ Le graphe dont les sommets sont les pays, deux pays étant reliés par une arête s'ils sont frontaliers [1].

1. Lorsqu'on colorie une carte du monde, on voudra donc ne pas colorier de la même couleur deux pays voisins dans ce graphe, c'est-à-dire deux pays reliés par une arête. Le très célèbre théorème des quatre couleurs dit qu'il est toujours possible, avec cette contrainte, de colorier une carte en seulement quatre couleurs.

## 1.2 Chemins et connexité

Soit $\mathcal{G} = (S, A)$ un graphe. Soient $s$ et $t$ deux sommets du graphe $\mathcal{G}$.
Dans le graphe $\mathcal{G}$, un **chemin** du sommet $s$ au sommet $t$ est une suite finie $(s_0, s_1, ..., s_p)$ de sommets de $S$ telle que :

- $\star$ $s_0 = s$ et $s_p = t$ ;
- $\star$ pour tout $i \in [\![0; p-1]\!]$, on ait $(s_i, s_{i+1}) \in A$.

La **longueur** d'un chemin est le nombre d'arêtes qui le constituent : avec les notations précédentes, la longueur du chemin $(s_0, s_1, ..., s_p)$ est $p$.

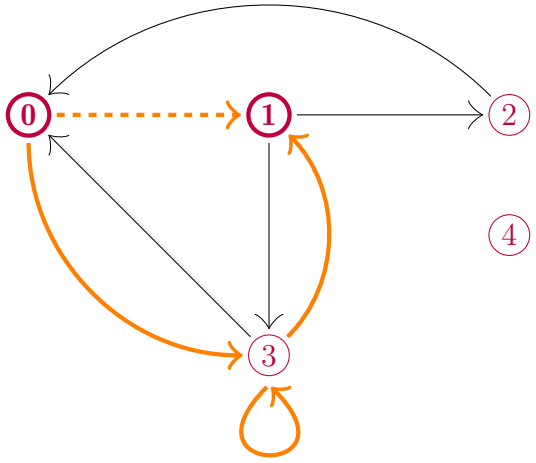



FIGURE 3 – Deux chemins du sommet 0 au sommet 1 dans le graphe $\mathcal{G}_1$ (de la figure 1). Le chemin en trait plein et gras est de longueur 3 et le chemin en pointillés de longueur 1.

Un graphe non orienté est dit **connexe** si pour tout couple $(s, t)$ de sommets du graphe, il existe un chemin de $s$ à $t$ dans le graphe.

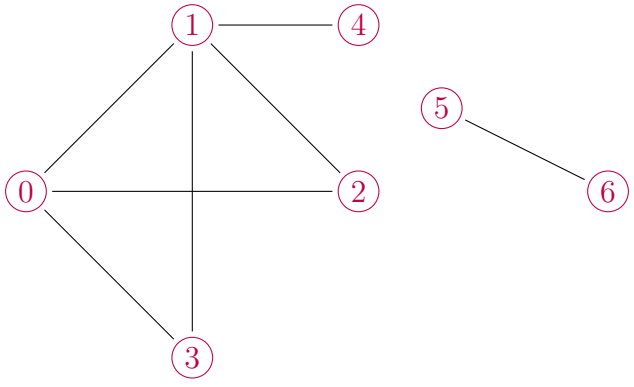



FIGURE 4 – Le graphe $\mathcal{G}_2$ n'est pas connexe : aucun chemin ne relie les sommets 4 et 5.

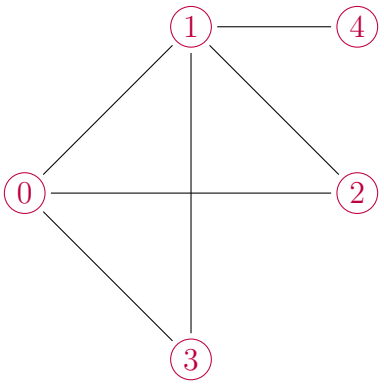



FIGURE 5 – Ce graphe est connexe.

## 1.3 Graphes pondérés

Pour représenter par un graphe un réseau de transport, il est pertinent d'ajouter au graphe l'information du temps de trajet entre les stations/nœuds du réseau, c'est-à-dire du temps de parcours de chaque arête.

Plus généralement, un **graphe pondéré** est un graphe à chaque arête duquel est associé un nombre, appelé le **poids** ou l'**étiquette** de l'arête.

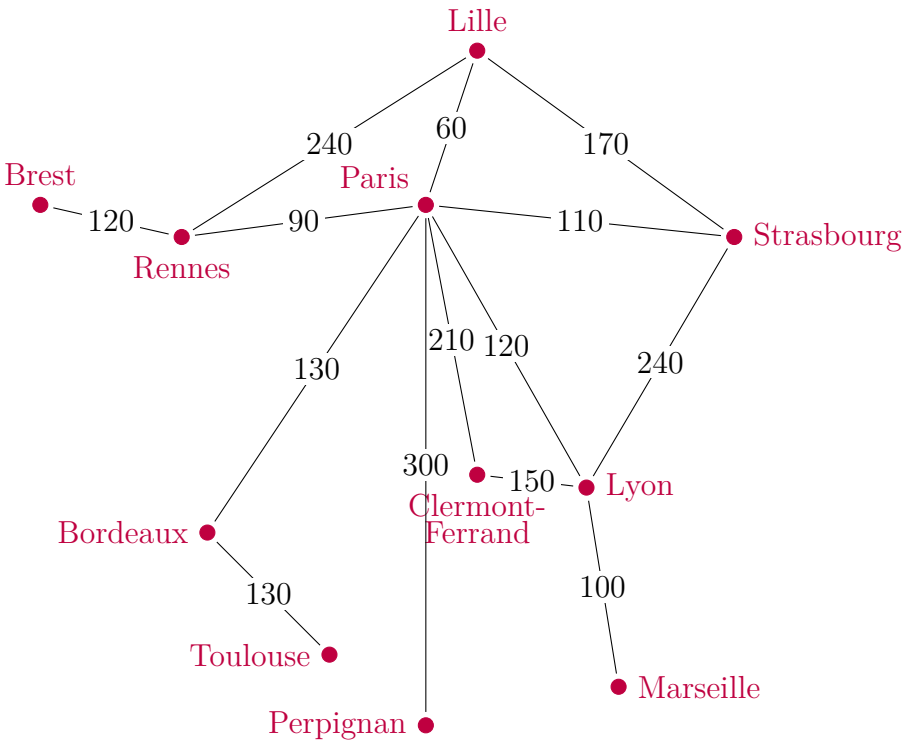



FIGURE 6 – Le graphe du réseau ferroviaire, pondéré par les temps de trajet (en minutes)

On rencontre également des graphes pondérés en probabilités : un arbre de probabilités est un graphe dont les sommets sont des évènements et dont les poids des arêtes sont des probabilités conditionnelles.

Par exemple, considérons une urne contenant trois boules rouges et deux boules jaunes. On tire successivement et sans remise deux boules dans l'urne. Pour tout $i \in \{1; 2\}$, on note $R_i$ l'évènement «La $i$-ème boule tirée est rouge.» et $J_i$ l'évènement «La $i$-ème boule tirée est jaune.».

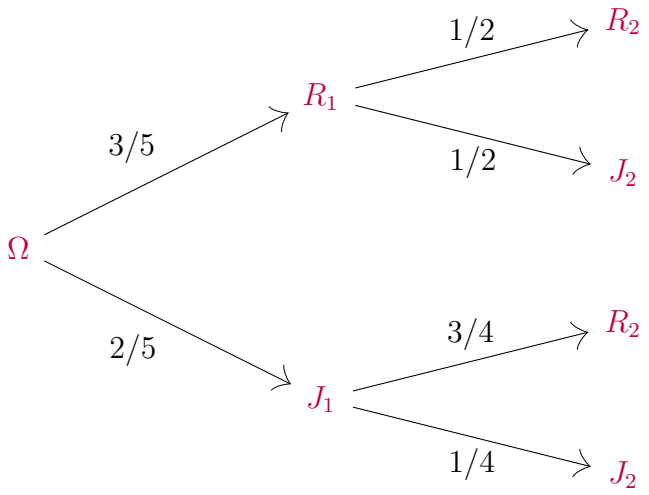



FIGURE 7 – Arbre de probabilités associé à l'expérience. La probabilité de tirer une boule rouge au second tirage sachant que la première boule tirée est jaune est de $\mathbb{P}_{J_1}(R_2) = \dfrac{3}{4}$.

Autre exemple : une chaine de Markov peut se décrire par un graphe orienté pondéré dont les sommets sont les états d'une variable aléatoire et dont les poids des arêtes sont les probabilités de passer d'un état à un autre.

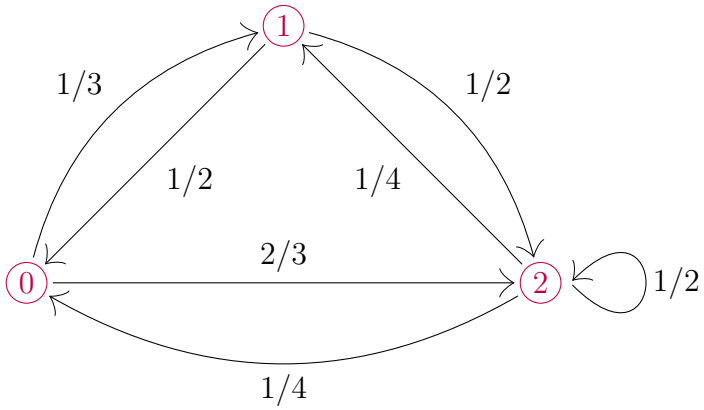



FIGURE 8 – Un exemple de graphe de transition de chaine de Markov

# 2 Représentations informatiques

Présentons à présent deux manières de représenter un graphe en machine.

## 2.1 Listes d'adjacence

Une première manière de représenter un graphe consiste à donner la *liste d'adjacence* de chaque sommet.

Étant donné un sommet $s$ du graphe, la **liste d'adjacence** de $s$ est la liste de tous ses **voisins (sortants)** (ou **successeurs**), c'est-à-dire la liste des tous les sommets $t$ tels qu'il existe une arête du sommet $s$ vers le sommet $t$ dans le graphe, c'est-à-dire la liste de tous les sommets du graphe vers lesquels $s$ pointe.

On regroupe alors les listes d'adjacence soit dans une liste (si les sommets du graphe sont numérotés en partant de zéro), soit dans un dictionnaire, dont les clés sont les noms des sommets et les valeurs les listes d'adjacence associées.

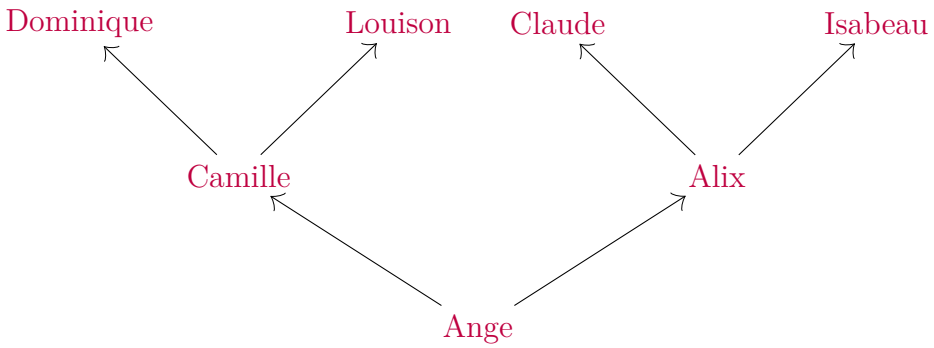



FIGURE 9 – L'arbre généalogique $\mathcal{A}$ d'Ange : les arêtes vont d'une personne vers ses parents.

Le dictionnaire d'adjacence du graphe $\mathcal{A}$ est le suivant :

```
D = {
    "Ange" : ["Camille", "Alix"],
    "Camille" : ["Dominique", "Louison"],
    "Alix" : ["Claude", "Isabeau"],
    "Dominique" : [],
    "Louison" : [],
    "Claude" : [],
    "Isabeau" : []
    }
```

L'instruction

```
D["Camille"]
```

donne alors la liste d'adjacence du sommet `"Camille"`, c'est-à-dire la liste des parents de Camille.

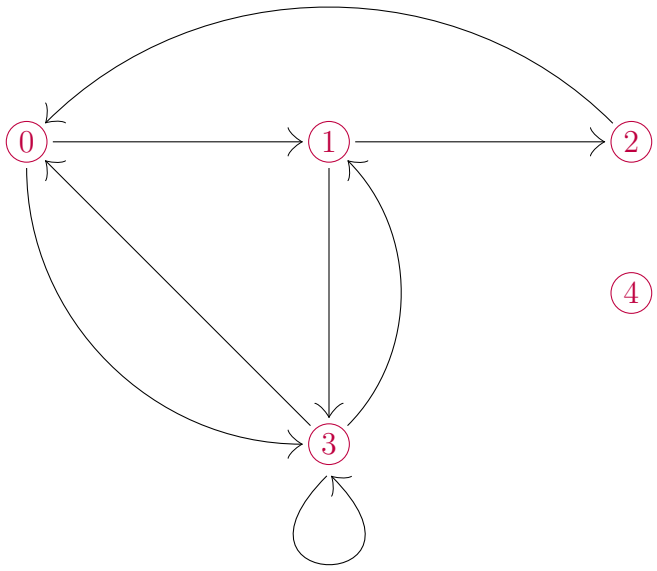



FIGURE 10 – Le graphe $\mathcal{G}_1$ (de la figure 1)

La liste des listes d'adjacence du graphe $\mathcal{G}_1$ est la suivante :

```
[ [1, 3], [2, 3], [0], [0, 1, 3], [] ]
```

et son dictionnaire d'adjacence est le suivant :

```
{0 : [1, 3], 1 : [2, 3], 2 : [0], 3 : [0, 1, 3], 4 : []}
```

## 2.2 Matrice d'adjacence

Une seconde manière de représenter un graphe consiste à donner sa *matrice d'adjacence*.

Soit $\mathcal{G} = (S, A)$ un graphe à $n$ sommets, dont les sommets sont numérotés de 0 à $n-1$.
La **matrice d'adjacence** du graphe $\mathcal{G}$ est la matrice $M$ de $M_n(\mathbb{R})$ telle que pour tout couple $(i, j) \in [\![0; n-1]\!]^2$ de sommets, on ait :

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in A, \text{ c'est-à-dire s'il existe une arête de } i \text{ vers } j \\ 0 & \text{sinon.} \end{cases}$$

La matrice d'adjacence du graphe $\mathcal{G}_1$ (des figures 1 et 10) est la suivante :

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

Remarquons que la matrice d'adjacence d'un graphe non orienté est symétrique.

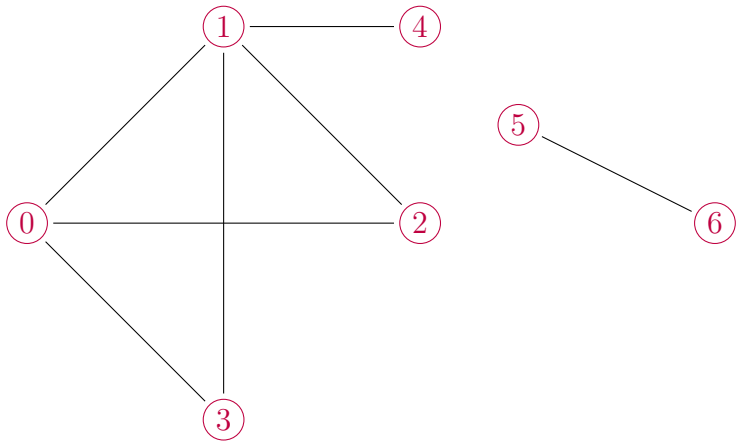



FIGURE 11 – Le graphe $\mathcal{G}_2$ (de la figure 2)

La matrice d'adjacence du graphe $\mathcal{G}_2$ est la suivante :

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

Lorsque le graphe est pondéré par des poids tous non nuls, on remplace dans la matrice d'adjacence les coefficients 1 par les poids des arêtes.

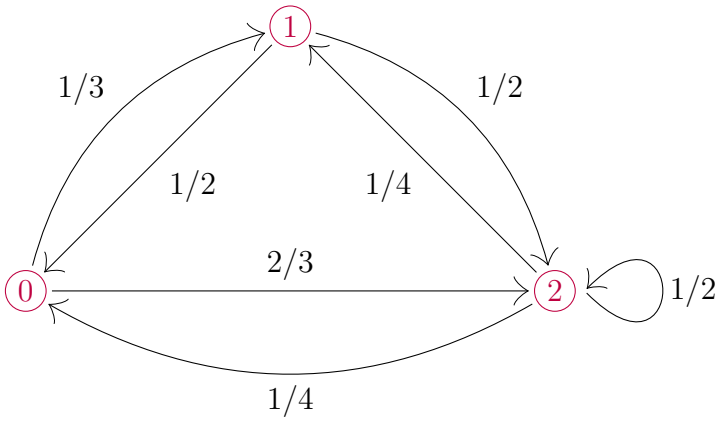



FIGURE 12 – Le graphe pondéré $\mathcal{G}_3$ de la figure 8

La matrice d'adjacence du graphe pondéré $\mathcal{G}_3$ (des figures 8 et 12) est la suivante :

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

# 3 Parcours en largeur d'un graphe

Étant donné un graphe, orienté ou non, il est très utile de savoir explorer tous les sommets du graphe qui sont accessibles depuis un sommet donné. Cela permet par exemple d'étudier la connexité du graphe, de trouver le plus court chemin (s'il en existe) entre deux sommets du graphe, et plus largement, de rechercher dans un graphe tous les sommets ayant une certaine propriété.

Il existe deux manières de parcourir ainsi les sommets d'un graphe :

- ⋆ le *parcours en profondeur*, que l'on utilise par exemple pour explorer un labyrinthe, et qui est au programme de seconde année ;
- ⋆ le *parcours en largeur*, que l'on étudie dans cette section.

Le **parcours en largeur** d'un graphe à partir d'un sommet donné consiste à commencer par explorer les voisins du sommet, puis les voisins de ses voisins, puis les voisins des voisins de ses voisins, et ainsi de suite.

Plus précisément, considérons $\mathcal{G}$ un graphe à $n$ sommets, numérotés de 0 à $n-1$, ainsi qu'un sommet $s$ du graphe $\mathcal{G}$.
On souhaite parcourir en largeur le graphe $\mathcal{G}$ à partir du sommet $s$, et obtenir ainsi la liste de tous les sommets accessibles depuis le sommet $s$, c'est-à-dire reliés à $s$ par un chemin.

## 3.1 Les structures de données utilisées

Pour cela, on inscrira dans trois listes l'avancement du parcours, de la manière suivante.

- ⋆ La liste `Visités`, initialement vide, contiendra tous les sommets que l'on a déjà visités dans le parcours.
  C'est la liste que l'on renverra à la fin du parcours : elle contiendra alors tous les sommets du graphe $\mathcal{G}$ qui sont accessibles depuis le sommet $s$.
- ⋆ La liste `À_visiter` sera la **file** d'attente des sommets que l'on doit encore visiter : les premiers sommets que l'on visitera seront les premiers de la file, et les nouveaux sommets à visiter seront ajoutés en queue de liste.
  Initialement, la liste `À_visiter` contiendra le sommet $s$.
- ⋆ Pour éviter de visiter plusieurs fois un même sommet, on crée une liste `Marqués` de $n$ booléens, telle que pour tout sommet $i \in [\![0; n-1]\!]$, l'élément `Marqués[i]` vaille `True` si le sommet $i$ a déjà été visité ou inscrit dans la file d'attente, `False` sinon.
  Initialement, le seul sommet marqué (avec `True`) est le sommet $s$.

## 3.2 Algorithme de parcours en largeur

Tant que la liste `À_visiter` n'est pas vide, c'est-à-dire tant qu'il reste dans la file d'attente des sommets à explorer, on effectue les actions suivantes :

- ⋆ on visite le premier sommet de la file d'attente `À_visiter` :
  - — on l'ajoute à la liste `Visités` des sommets visités ;
  - — on le supprime de la file d'attente `À_visiter` ;
- ⋆ on ajoute à la file d'attente `À_visiter` tous les successeurs de ce sommet qui ne sont pas déjà marqués, c'est-à-dire qui n'ont pas déjà été visités ou ajoutés à la file d'attente ;
- ⋆ on marque chacun de ces successeurs dans la liste `Marqués`.

## 3.3 Implémentation

Lorsque le graphe est donné sous forme de liste de listes d'adjacence, le parcours en largeur décrit précédemment s'implémente comme suit.

```
def Parcours_largeur(LL, s):
    """
    Entrées : - un graphe donné sous forme de
                liste LL de listes d'adjacence ;
              - un sommet s du graphe.
    Sortie : la liste des sommets accessibles depuis le sommet s,
             dans l'ordre du parcours en largeur.
    """
    n = len(LL) # Nombre de sommets du graphe.
    Marqués = n * [False]
    # Pour l'instant, aucun sommet n'a été marqué.
    À_visiter = [s] # File d'attente des sommets à visiter.
    # On commencera par le sommet s.
    Marqués[s] = True
    # Le sommet s est marqué (il est dans la file d'attente).
    Visités = []

    while len(À_visiter) != 0:
    # Tant qu'il reste des sommets en attente :
        premier_file = À_visiter.pop(0)
        # premier_file est le premier sommet de la file.
        # Il est supprimé de la file À_visiter.
        Visités.append(premier_file)
        for voisin in LL[premier_file]:
        # voisin parcourt les successeurs du sommet premier_file.
            if not Marqués[voisin]:
            # Si le sommet voisin n'a pas encore été marqué :
                À_visiter.append(voisin)
                Marqués[voisin] = True
    return Visités
```

En TP, nous adapterons le programme précédent au cas où le graphe est donné par sa matrice d'adjacence et à d'autres problèmes qui se résolvent par un parcours en largeur.

Détaillons au tableau le fonctionnement de l'algorithme de parcours en largeur sur le graphe $\mathcal{G}_4$ de la figure 13, en partant du sommet 7.

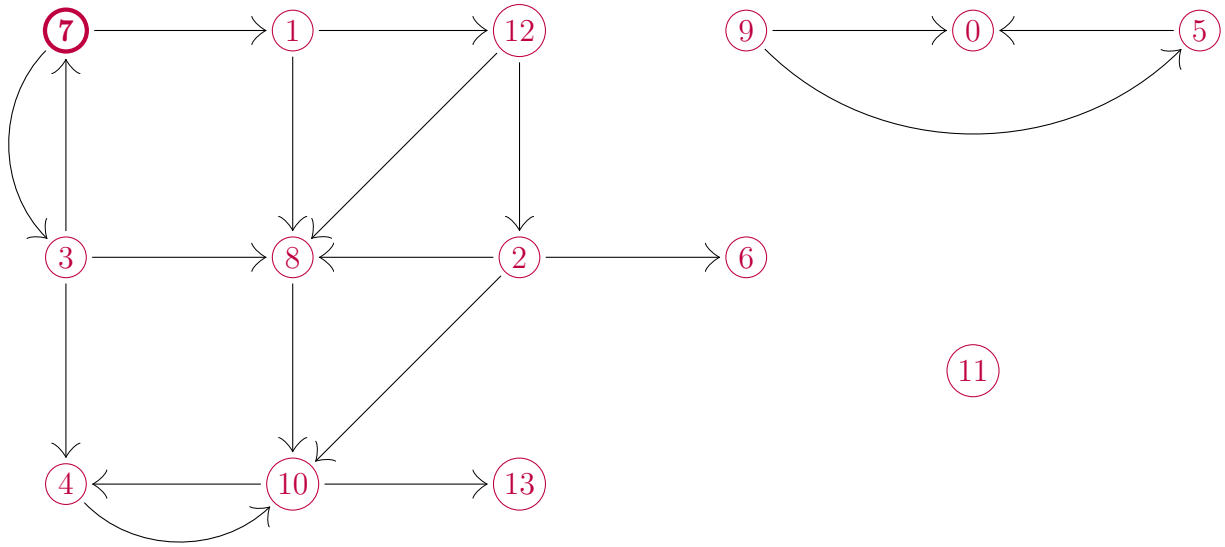



FIGURE 13 – Le graphe $\mathcal{G}_4$