Graphes

I. Généralités sur les graphes

1) Définitions et vocabulaire

Définition 1 : *Graphe*

Un graphe $\mathcal{G}$ est constitué d'un ensemble de sommets $\mathcal{S}$ et d'un ensemble d'arcs $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$.
On note $\mathcal{G} = (\mathcal{S}, \mathcal{A})$.

Définition 2 : *Vocabulaire sur les graphes*

Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe.

- ⋆ Un sommet peut également être appelé un nœud.
- ⋆ Un arc peut également être appelé une arête (mais ce terme s'utilise plutôt pour des graphes non orientés : voir plus loin).
- ⋆ Un arc de la forme (s, s) est appelé une boucle.

Définition 3 : *Graphe orienté, graphe non orienté*

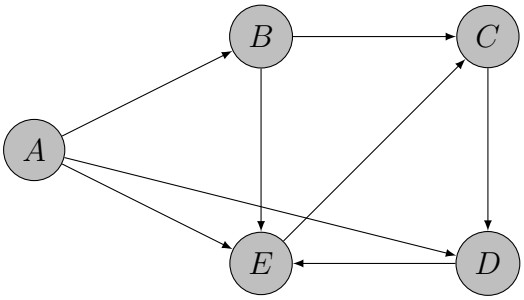
Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe.

(i) On dit que $\mathcal{G}$ est un graphe orienté lorsqu'il existe au moins deux sommets $s, s' \in \mathcal{S}$ tels que $(s, s') \in \mathcal{A}$ et $(s', s) \notin \mathcal{A}$.

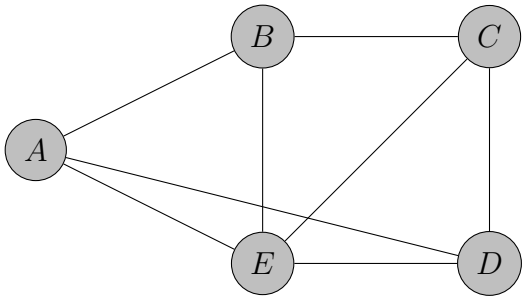
(ii) On dit que $\mathcal{G}$ est un graphe non orienté lorsque pour tout couple de sommets $(s, s') \in \mathcal{S} \times \mathcal{S}$, on a

$$(s, s') \in \mathcal{A} \quad \Leftrightarrow \quad (s', s) \in \mathcal{A}$$

Dans ce cas, on appelle arête, l'ensemble des deux arcs $\{(s, s'), (s', s)\}$.



Graphe orienté



Graphe non orienté

2) Adjacence et chemin dans un graphe

Définition 4 : *Sommets voisins*

Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe et s et s' deux sommets de $\mathcal{G}$.
On dit que s' est adjacent à s ou que s' est un voisin de s lorsque $(s, s') \in \mathcal{A}$ est un arc.
L'ensemble des sommets adjacents à un sommet s est appelé l'ensemble des voisins de s.

Remarque : Dans un graphe orienté, la relation « être voisin de » n'est pas symétrique : le sommet B peut être voisin de A sans que le sommet A soit voisin de B.

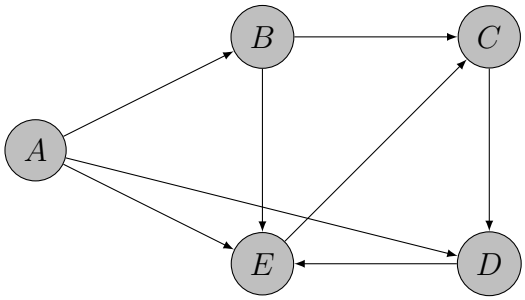
Dans un graphe non orienté, cette relation est symétrique.

Implémentation d'un graphe

Comment représenter un graphe informatiquement ? Il existe plusieurs façons de procéder.
Une première façon consiste à construire un dictionnaire d'adjacence : pour chaque sommet, on calcule l'ensemble de ses voisins et on les range dans un dictionnaire dont les clefs sont les sommets.

Exemple :

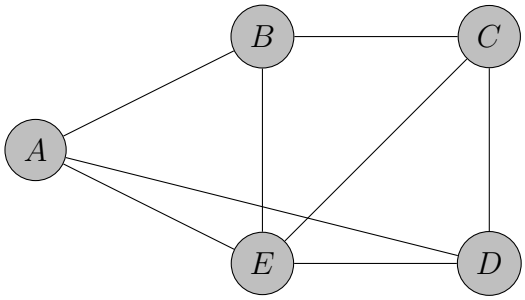
⋆ Pour le graphe orienté suivant :



le dictionnaire d'adjacence est

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

⋆ Pour le graphe non orienté suivant :

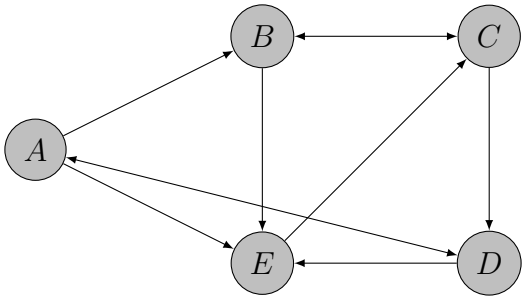


le dictionnaire d'adjacence est

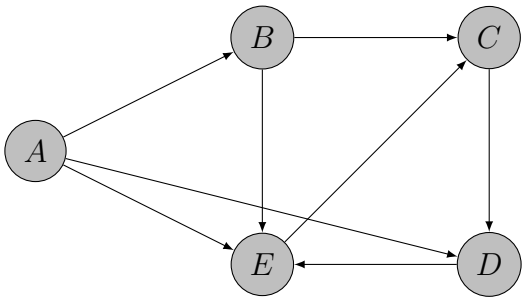
```
D = {"A": ["B", "D", "E"], "B": ["A", "C", "E"], "C": ["B", "D", "E"],
     "D": ["A", "C", "E"], "E": ["A", "B", "C", "D"]}
```

Définition 5 : *Chemin dans un graphe*

Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe.
Un chemin dans le graphe $\mathcal{G}$ est une suite de sommets dont chaque sommet est adjacent avec le précédent : $s_1 \to s_2 \to \cdots \to s_n$ avec pour tout $i \in [\![2, n]\!]$, s_i est adjacent à s_{i-1}.



Exemple : Pour le graphe orienté suivant :



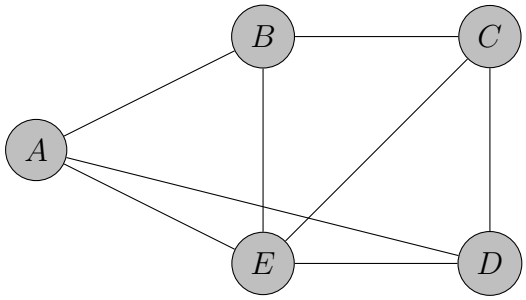
- $A \to B \to E \to C \to D \to E$ est un chemin.
- $A \to B \to C \to D \to E \to A$ n'est pas un chemin.

Définition 6 : *Connexité d'un graphe non orienté*

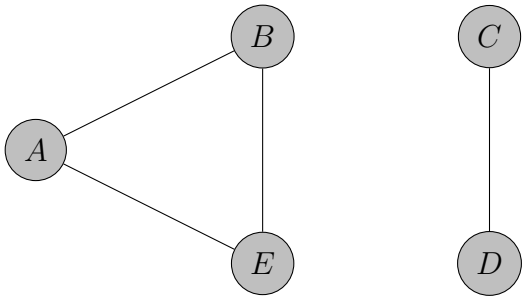
Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe non orienté.
On dit que $\mathcal{G}$ est connexe lorsque pour tout couple de sommets $(s, s') \in \mathcal{S} \times \mathcal{S}$ il existe un chemin entre s et s' :

$$s = s_0 \to s_1 \to s_2 \to \cdots \to s_n = s'$$

Dans le cas contraire, on dit que $\mathcal{G}$ n'est pas connexe.



Graphe connexe



Graphe non connexe

3) Matrice d'adjacence

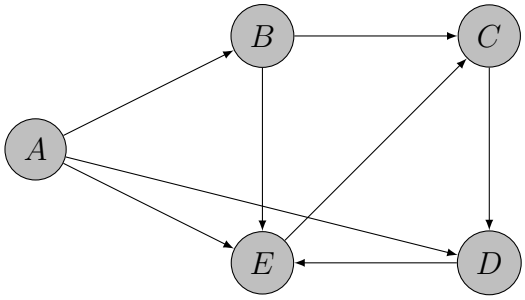
Définition 7 : *Matrice d'adjacence d'un graphe*

Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe à n sommets numérotés de 1 à n.
On appelle matrice d'adjacence de $\mathcal{G}$ la matrice $M = (m_{i,j}) \in \mathcal{M}_n(\mathbb{R})$ telle que pour tous $(i,j) \in [\![1,n]\!]^2$:

$$m_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in \mathcal{A} \text{ i.e. } j \text{ est adjacent à } i \\ 0 & \text{sinon} \end{cases}$$

Exemple :

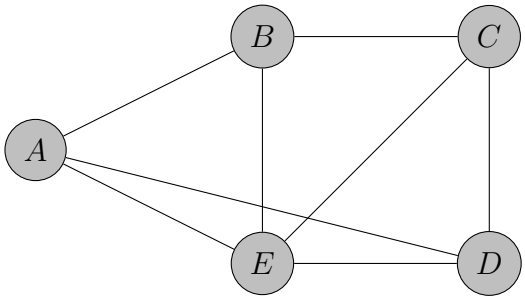
⋆ Pour le graphe orienté suivant :



la matrice d'adjacence est

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

⋆ Pour le graphe non orienté suivant :



la matrice d'adjacence est

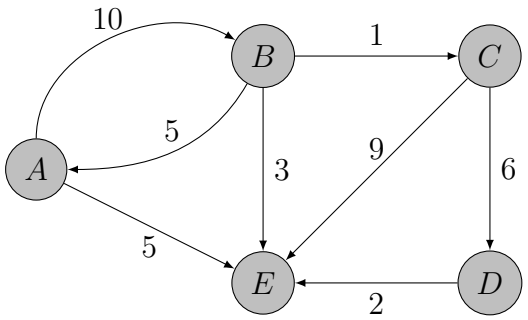
$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

La matrice d'adjacence d'un graphe non orienté est symétrique.

II. Graphes pondérés

Définition 8 : *Pondération d'un graphe*

Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe.
Pondérer le graphe $\mathcal{G}$ signifie attribuer à chaque arc (ou arête) une valeur numérique appelée étiquette ou poids de l'arc ou de l'arête.
Mathématiquement, cela revient à définir une fonction $\omega : \mathcal{A} \to \mathbb{R}$ telle que pour tout arc $(s, s') \in \mathcal{A}$, l'étiquette de cet arc est $\omega(s, s')$. On note alors $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \omega)$.



La matrice d'adjacence s'adapte parfaitement au cas des graphes pondéré contrairement à la représentation par dictionnaire d'adjacence.

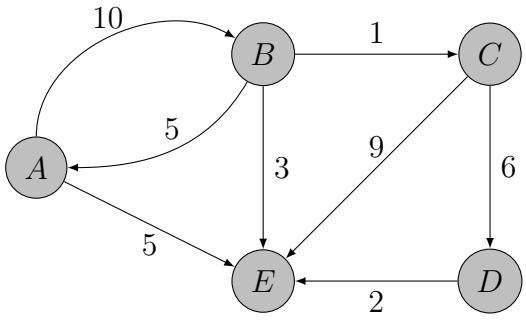
Définition 9

Matrice d'adjacence d'un graphe pondéré Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \omega)$ un graphe pondéré à n sommets numérotés de 1 à n.
On appelle matrice d'adjacence de $\mathcal{G}$ la matrice $M = (m_{i,j}) \in \mathcal{M}_n(\mathbb{R})$ telle que pour tous $(i,j) \in [\![1,n]\!]^2$:

$$m_{i,j} = \begin{cases} \omega(i,j) & \text{si } (i,j) \in \mathcal{A} \text{ i.e. } j \text{ est adjacent à } i \\ 0 & \text{sinon} \end{cases}$$

Exemple :

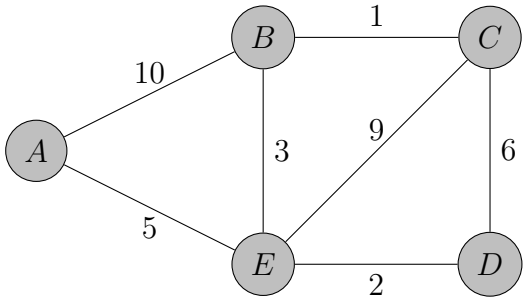
⋆ Pour le graphe orienté et pondéré suivant :



La matrice d'adjacence est

$$M = \begin{pmatrix} 0 & 10 & 0 & 0 & 5 \\ 5 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 6 & 9 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

⋆ Pour le graphe non orienté et pondéré suivant :



La matrice d'adjacence est

$$M = \begin{pmatrix} 0 & 10 & 0 & 0 & 5 \\ 10 & 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 6 & 9 \\ 0 & 0 & 6 & 0 & 2 \\ 5 & 3 & 9 & 2 & 0 \end{pmatrix}$$

III. Parcours d'un graphe

1) Principe

Parcours d'un graphe

Beaucoup de problèmes sur les graphes peuvent être résolus en parcourant les sommets du graphe un à un.

Exemples de problèmes sur les graphes :

- ⋆ Déterminer si un graphe est connexe.
- ⋆ Déterminer s'il existe un chemin entre deux sommets et trouver ce chemin (labyrinthe).
- ⋆ Déterminer le plus court chemin entre deux sommets (GPS).

Il existe plusieurs façon de parcourir un graphe en fonction de l'ordre des sommets que l'on visite :

- ⋆ Parcours en profondeur.
- ⋆ Parcours en largeur.

On suppose dans cette partie que les graphes sont représentés par des dictionnaires d'adjacence : pour chaque sommet, on connaît l'ensemble de ses voisins.

2) Parcours en largeur

Principe du parcours en largeur

Le parcours en largeur consiste à partir d'un sommet s_0 et à visiter tous les sommets voisins de s, puis tous les sommets voisins de ces sommets (les sommets à distance 2 de s_0), et ainsi de suite en s'éloignant progressivement du sommet initial s_0.

Problème du plus court chemin

Étant donné un sommet s d'un graphe $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ non pondéré, on veut calculer le nombre minimal d'arcs ou d'arêtes pour aller de s vers chaque autre sommet du graphe.

Étant donné un sommet `s` qui sera notre sommet de départ, l'algorithme de parcours en largeur du graphe va fonctionner de la façon suivante :

⋆ On initialise une liste `vus = [s]` qui contiendra les sommets découverts mais non encore visités (les sommets en bleu).

⋆ On initialise un dictionnaire `distances` telle que pour tout sommet `u`, `distances[u] =` ∞ si `u`$\neq$`s` et `distances[s] = 0`. Ce dictionnaire contiendra, à la fin de l'algorithme, toutes les distances du sommet `s` aux autres sommets

⋆ Tant que la liste `vus` n'est pas vide :

- on retire le premier élément `u` de `vus` : on considère qu'on le visite
- on ajoute à la fin de `vus` tous les voisins `v` de `u` qui
  - ▶ n'ont pas encore été visités (`distances[v] =` ∞)
  - ▶ ne sont pas déjà dans la liste `vus`

  et on met à jour `distances[v] = distances[u] + 1`.

```
def parcours_en_largeur(G, s):
    vus = [s]
    distances = {u: float("inf") for u in G}
    distances[s] = 0

    while vus:
        u = vus.pop(0)
        for v in G[u]:
            if distances[v] == float("inf") and v not in vus:
                vus.append(v)
                distances[v] = distances[u] + 1

    return distances
```

3) Parcours en profondeur

Principe du parcours en profondeur

Le parcours en profondeur consiste à partir d'un sommet s_0 et à visiter un de ses voisins s_1, puis un voisin de s_1 (disons s_2), puis un voisin de s_2 (disons s_3), et ainsi de suite jusqu'à ce qu'on arrive à un sommet qui n'a pas de voisin non encore visité. À ce moment-là, on revient en arrière pour visiter les autres voisins du sommet précédent, et ainsi de suite.

Problème du labyrinthe

Étant donné deux sommets e et s d'un graphe $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ non pondéré, on veut déterminer s'il existe un chemin entre e et un autre sommet s du graphe, et trouver ce chemin.

```
def parcours_en_profondeur(G, e, s, vus = []):
    vus.append(e)

    if e == s:
        return [s]

    for v in G[e]:
        if v not in vus:
            chemin = parcours_en_profondeur(G, v, s, vus)
            if chemin is not None:
                return [e] + chemin

    return None
```