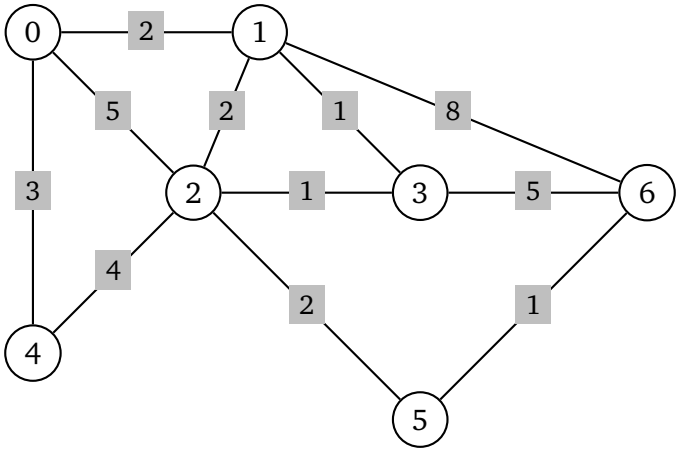
# Algorithme de Dijkstra

*« Tester un programme peut démontrer la présence de bugs, jamais leur absence. »*
Edsger Dijkstra[1]

Les graphes permettent de représenter une vaste gamme de problème : réseau routier, problème d'ordonnancement, automates cellulaires, etc. Un des problèmes les plus importants consiste à trouver le plus « court » chemin entre deux sommets d'un graphe pondéré.

## 1 — Graphe pondéré

Un *graphe pondéré* est constitué d'un ensemble de *sommets* que nous numérotons $0, 1, 2, \ldots$ Ces derniers sont reliés entre eux par des *arêtes*, et à chaque arête est associé un *poids*.



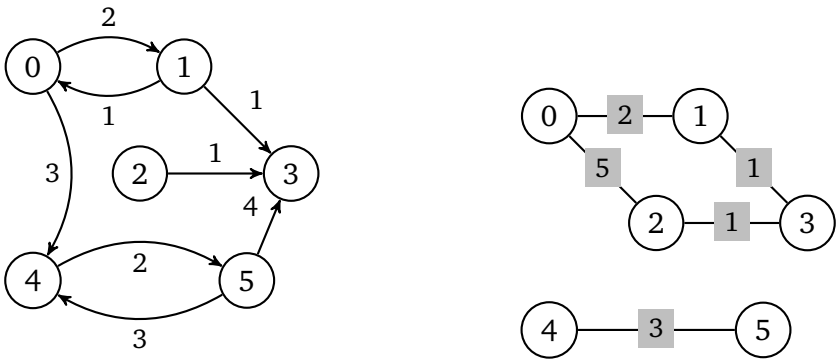

Lorsque deux sommets sont reliés par une arête, on peut se déplacer de l'un à l'autre, mais cela à un « coût » : le poids de l'arête. En plusieurs pas, on peut se rendre ainsi d'un sommet à un autre, en additionnant les coûts successifs des différentes arêtes. Dans le graphe ci-dessus, on peut envisager d'aller du sommet (0) au sommet (5) par le chemin (0) → (1) → (2) → (5), pour un coût total égal à $2 + 2 + 2 = 6$.

Un tel graphe peut s'interpréter comme un réseau routier : chaque sommet représente une ville et sur chaque arête est inscrit le coût (qui peut être la distance, le temps, le prix, etc.) du trajet entre deux villes.

L'algorithme de Dijkstra permet de déterminer le plus court chemin entre deux sommets donnés.

À titre informatif, notez que dans notre étude un chemin peut-être parcouru dans les deux sens (le graphe est « non orienté ») et que son poids est strictement positif. De plus il existe toujours au moins un chemin reliant deux sommets quelconques (le graphe est « connexe »). L'algorithme de Dijkstra s'appliquerait aussi aux graphes orientés, mais pas aux graphes de poids négatifs ou non connexes.

1. Edsger Dijkstra (1930 – 2002), mathématicien et informaticien néerlandais, prix Turing 1972.




Quelques exemples de graphes.

## 2 — Représentation informatique d'un graphe

Pour représenter le graphe, nous allons utiliser sa *matrice d'adjacence*. Il s'agit d'une matrice carrée d'ordre $n$ ($n = 7$ dans notre exemple). Pour être en accord avec l'indexation du langage Python, les lignes et les colonnes seront numérotées de 0 à 6.

Le coefficient $(i, j)$ de cette matrice est égal au poids de l'arête entre les sommets $i$ et $j$ si une telle arête existe. Il est égal conventionnellement à `False` si l'arête n'existe pas. L'arête entre un sommet et lui-même a un poids nul. Avec notre exemple, la matrice d'adjacence est

$$A = \begin{pmatrix} 0 & 2 & 5 & \mathsf{F} & 3 & \mathsf{F} & \mathsf{F} \\ 2 & 0 & 2 & 1 & \mathsf{F} & \mathsf{F} & 8 \\ 5 & 2 & 0 & 1 & 4 & 2 & \mathsf{F} \\ \mathsf{F} & 1 & 1 & 0 & \mathsf{F} & \mathsf{F} & 5 \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{pmatrix}$$

(on a abrégé `False` en F).

**Question 1** Compléter la matrice d'adjacence du graphe précédent.

**Question 2** Quelle est la particularité de cette matrice ? À quoi est-ce dû ?

## 3 — Présentation de l'algorithme

L'algorithme présenté ci-dessous va en fait permettre d'obtenir le plus court chemin entre un point de départ et tous les autres sommets du graphe (et non un point d'arrivée en particulier). C'est donc plus fort que ce qui était souhaité.

Voyons son fonctionnement à travers un exemple : on considère le sommet (0) comme étant le point de départ. On part du tableau suivant :

| | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
|---|---|---|---|---|---|---|---|
| Étape 1 | | | | | | | |

**Étape 1** Pour la première étape, on remplit la première ligne à l'aide des chemins reliant (0) à chacun de ses voisins. On indique dans le tableau la longueur de l'arête reliant (0) au sommet considéré, et on indique `False` s'il n'y a pas (encore) de chemin trouvé. On indique également de quel sommet on vient (pour cette première étape, on vient toujours de (0)). Cela donne

| | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
|---|---|---|---|---|---|---|---|
| Étape 1 | 0, (0) | 2, (0) | 5, (0) | F | 3, (0) | F | F |

Parmi tous les chemins calculés, celui reliant (0) à (1) est intéressant : c'est le plus court chemin allant de (0) à n'importe quel autre sommet en un seul pas.

> **Point crucial :** On sait désormais que pour aller de (0) à (1), le plus court chemin est d'y aller directement. En effet, tout autre chemin de (0) à (1) fera un pas de plus, ce qui augmentera nécessairement la longueur du chemin.

On marque donc ce chemin, et on passe à l'étape suivante. Remarquez qu'on avait déjà marqué le sommet (0) à l'étape initale.

| | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
|---|---|---|---|---|---|---|---|
| Étape 1 | 0, (0) | 2, (0) | 5, (0) | F | 3, (0) | F | F |

**Étape 2** On se déplace alors sur le sommet précédemment marqué : le sommet (1). On passe alors en revue tous les voisins de (1). Ils sont connectés à (0) par au plus deux chemins : soit un chemin déjà noté dans le tableau, soit un nouveau chemin constitué du chemin déjà noté entre (0) et (1) auquel on ajoute une arête.

Prenons l'exemple des voisins de (1) :

— le sommet (0) est marqué : on ne l'examine pas.

— on examine (2) : on compare 5 (longueur du chemin déjà existant) à $2+2=4$, longueur du nouveau chemin possible (0) → (1) → (2). Ce dernier est plus court : on inscrit donc 4,(1) dans la deuxième ligne.

— le sommet (3) est maintenant relié à (0) par un chemin de longueur 3 : on inscrit donc 3,(1) sur la deuxième ligne.

— pour les sommets (4) et (5), rien ne change pour le moment puisqu'il ne sont pas reliés à (1).

— Enfin (6) est maintenant relié à (0) par le chemin de longueur 10 (0) → (1) → (6) : on inscrit donc 10,(1) sur la deuxième ligne.

Le tableau devient :

| | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
|---|---|---|---|---|---|---|---|
| Étape 1 | 0, (0) | 2, (0) | 5, (0) | F | 3, (0) | F | F |
| Étape 2 | 0, (0) | 2, (0) | 4, (1) | 3, (1) | 3, (0) | F | 10, (1) |

Tout comme précédemment, on vient de trouver deux chemins de longueur minimale : l'un reliant (0) à (3), l'autre reliant (0) à (4). On sait qu'on ne pourra pas trouver de chemins plus court entre (0) et ces deux sommets.

On choisit arbitrairement un des deux (ce qui illustre qu'il n'y a pas unicité du chemin final), on le marque, et on passe à l'étape suivante.

| | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
|---|---|---|---|---|---|---|---|
| Étape 1 | 0, **0** | 2, **0** | 5, (0) | F | 3, (0) | F | F |
| Étape 2 | 0, **0** | 2, **0** | 4, (1) | 3, **1** | 3, (0) | F | 10, (1) |

**Étapes suivantes** On continue ce travail jusqu'à ce que toutes les colonnes du tableau aient été marquées. Notez qu'une fois une colonne marquée, on ne la considère plus dans les comparaisons.

**Question 3** On conclut en 6 étapes. Question : pourquoi 6 ?

La dernière est indiquée dans le tableau ci-dessous, à vous de trouver les étapes manquantes.

| | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
|---|---|---|---|---|---|---|---|
| Étape 1 | 0, **0** | 2, **0** | 5, (0) | F | 3, (0) | F | F |
| Étape 2 | 0, **0** | 2, **0** | 4, (1) | 3, **1** | 3, (0) | F | 10, (1) |
| Étape 3 | | | | | | | |
| Étape 4 | | | | | | | |
| Étape 5 | | | | | | | |
| Étape 6 | 0, **0** | 2, **0** | 4, **1** | 3, **1** | 3, **0** | 6, **2** | 7, **5** |

## 4 — Détermination du plus court chemin

Une fois le tableau calculé, il est facile de trouver le chemin optimal entre (0) et (6). Dans la colonne (6) est indiqué la longueur de ce chemin ainsi que l'étape juste avant d'arriver à (6), à savoir (5).

**Question 4** Quel est le sommet précédent (5) ? Pourquoi ? Déterminer complètement le chemin le plus court entre (0) et (6).

**Question 5** On fournit une fonction Python d'en-tête `calcule_tableau(Graphe, depart)` calculant le tableau précédent ; le graphe est passé sous forme d'une liste Python.

Proposer une fonction Python d'en-tête `plus_court_chemin(Graphe, depart, arrivee)` utilisant la fonction fournie et renvoyant le plus court chemin entre les sommet `depart` et `arrivee`.

**Dijkstra en quelques mots**

L'Algorithme de Dijkstra permet de calculer le plus court chemin entre deux sommets d'un graphe connexe non orientés.

On construit tous les chemins optimaux progressivement : on part du point de départ et on regarde les chemins de taille 1 (une arête). On conserve le chemin de longueur minimal car celui-ci ne pourra être « amélioré » par la suite. On construit ensuite des débuts de chemin et on conserve à chaque fois le plus court d'entre eux (car ceux là ne pourront être améliorés). À chaque étape, on traite définitivement un nouveau sommet, qui est marqué. Une fois tous les sommets marqués, on a accès à la longueur des plus courts chemins issus du point de départ, ainsi qu'aux chemins eux-mêmes.

## 5 — Réponses aux questions

**Question 1**

$$A = \begin{pmatrix} 0 & 2 & 5 & F & 3 & F & F \\ 2 & 0 & 2 & 1 & F & F & 8 \\ 5 & 2 & 0 & 1 & 4 & 2 & F \\ F & 1 & 1 & 0 & F & F & 5 \\ 3 & F & 4 & F & 0 & F & F \\ F & F & 2 & F & F & 0 & 1 \\ F & 8 & F & 5 & F & 1 & F \end{pmatrix}$$

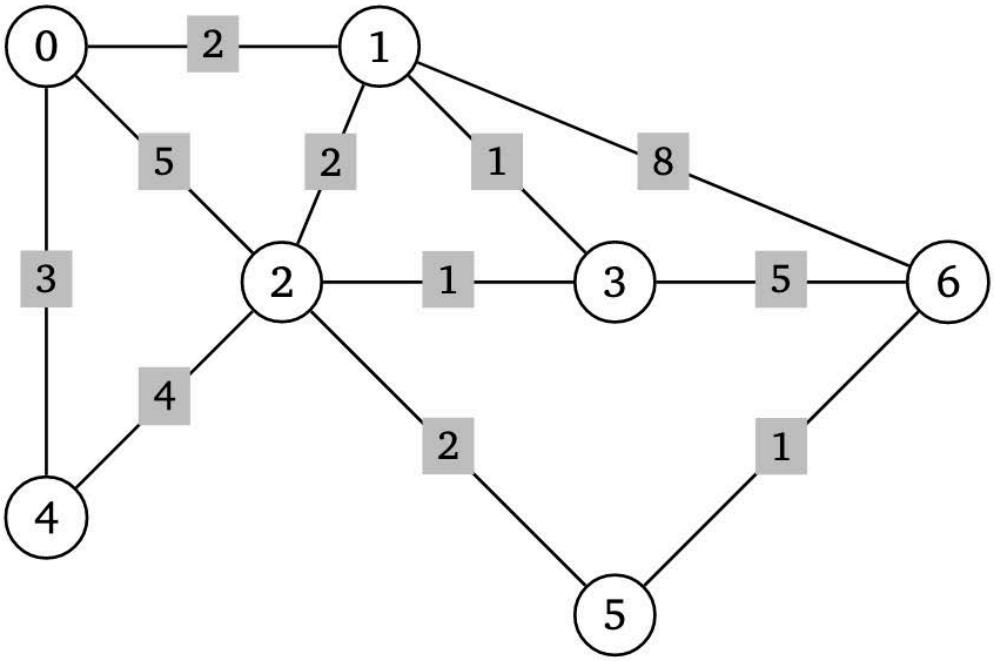
**Question 2** La matrice précédente est symétrique. Comme le graphe n'est pas orienté, l'arête reliant un sommet $i$ à un sommet $j$ et l'arête retour (reliant $j$ à $i$) ont le même poids.

**Question 3**

| | ⓪ | ① | ② | ③ | ④ | ⑤ | ⑥ |
|---|---|---|---|---|---|---|---|
| Étape 1 | 0, ⓪ | 2, ⓪ | 5, ⓪ | F | 3, ⓪ | F | F |
| Étape 2 | 0, ⓪ | 2, ⓪ | 4, ① | 3, ① | 3, ⓪ | F | 10, ① |
| Étape 3 | 0, ⓪ | 2, ⓪ | 4, ① | 3, ① | 3, ⓪ | F | 8, ③ |
| Étape 4 | 0, ⓪ | 2, ⓪ | 4, ① | 3, ① | 3, ⓪ | F | 8, ③ |
| Étape 5 | 0, ⓪ | 2, ⓪ | 4, ① | 3, ① | 3, ⓪ | 6, ② | 8, ③ |
| Étape 6 | 0, ⓪ | 2, ⓪ | 4, ① | 3, ① | 3, ⓪ | 6, ② | 7, ⑤ |

**Question 4** Le sommet précédent ⑤ est le ②. En effet, le plus court chemin entre ⓪ et ⑥ passe par ⑤ : nécessairement le sous-chemin de ⓪ à ⑤ est aussi le plus court chemin entre ces deux sommets. D'après le tableau, juste avant d'arriver à ⑤ il passe par ②.

| ⓪ | ① | ② | ③ | ④ | ⑤ | ⑥ | Fixés | Retenues | Provenances | Étapes |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2, ⓪ | 5, ⓪ | F | 3, ⓪ | F | F | ⓪ | 0 | ⓪ | 1 |
| | | 4, ① | 3, ① | 3, ⓪ | F | 10, ① | ① | 2 | ⓪ | 2 |
| | | 4, ① | | 3, ⓪ | F | 8, ③ | ③ | 3 | ① | 3 |
| | | 4, ① | | | F | 8, ③ | ④ | 3 | ⓪ | 4 |
| | | | | | 6, ② | 8, ③ | ② | 4 | ① | 5 |
| | | | | | | 7, ⑤ | ⑤ | 6 | ② | 6 |
| | | | | | | | ⑥ | 7 | ⑤ | |



**Question 5**

```
# Implémentation de l'algorithme de Dijkstra

Graphe = [[ 0,  2,  5,  False,  3,  False,  False ],
    [ 2,  0,  2,  1,  False,  False,  8 ],
    [ 5,  2,  0,  1,  4,  2,  False ],
    [ False,  1,  1,  0,  False,  False,  5 ],
    [ 3,  False,  4,  False,  0,  False,  False ],
    [ False,  False,  2,  False,  False,  0,  1 ],
    [ False,  8,  False,  5,  False,  1,  False ]]

def ligneInit(Graphe,depart) :
    """ Renvoie la première ligne du tableau """
    L = []

    # nombre de lignes de Graphe donc nombre de sommets
    n = len(Graphe)

    for j in range(n) :
        poids = Graphe[depart][j]
        if poids :
            # si l'arête est présente
            L.append([ poids, depart ])
        else :
            L.append(False)
    return [L]


def SommetSuivant(T, S_marques) :
    """ En considérant un tableau et un ensemble de sommets marqués,
    détermine le prochain sommet marqué. """

    L = T[-1]
    n = len(L)
    # minimum des longueurs, initialisation
    min = False

    for i in range(n) :
        if not(i in S_marques) :
            # si le sommet d'indice i n'est pas marqué
            if L[i]:
                if not(min) or L[i][0] < min :
                    # on trouve un nouveau minimum
                    # ou si le minimum n'est pas défini
                    min = L[i][0]
                    marque = i
```

```
    return(marque)



def ajout_ligne(T,S_marques,Graphe) :
    """ Ajoute une ligne supplémentaire au tableau """

    L = T[-1]
    n = len(L)

    # La prochaine ligne est une copie de la précédente,
    # dont on va modifier quelques valeurs.
    Lnew = L.copy()

    # sommet dont on va étudier les voisins
    S = S_marques[-1]
    # la longueur du (plus court) chemin associé
    long = L[S][0]

    for j in range(n) :
        if j not in S_marques:
            poids = Graphe[S][j]
            if poids :
                # si l'arète (S,j) est présente
                if not(L[j]) :  # L[j] = False
                    Lnew[j] = [ long + poids, S ]
                else :
                    if long + poids < L[j][0] :
                        Lnew[j] = [ long + poids, S ]

    T.append(Lnew)

    # Calcul du prochain sommet marqué
    S_marques.append(SommetSuivant(T, S_marques))
    return T, S_marques


def calcule_tableau(Graphe, depart) :
    """ Calcule le tableau de l'algorithme de Dijkstra """
    n = len(Graphe)

    # Initialisation de la première ligne du tableau
    # Avec ces valeurs, le premier appel à ajout_ligne
    # fera le vrai travail d'initialisation
    T=[[False] *n]
    T[0][depart] = [depart, 0]
```

```
    # liste de sommets marques
    S_marques = [ depart ]

    while len(S_marques) < n :
        T, S_marques = ajout_ligne(T, S_marques, Graphe)

    return T

def plus_court_chemin(Graphe, depart, arrivee) :
    """ Détermine le plus court chemin entre depart et arrivee dans
    le Graphe"""

    n = len(Graphe)
    # calcul du tableau de Dijkstra
    T = calcule_tableau (Graphe,depart)

    # liste qui contiendra le chemin le plus court, on place l'arrivée
    C = [ arrivee ]

    while C[-1] != depart :
        C.append( T[-1][ C[-1] ][1] )

    # Renverse C, pour qu'elle soit plus lisible
    C.reverse()

    return C
```