

# Graphes : Représentations

Quentin Fortier

April 11, 2022

# Représentation de graphe

On souhaite stocker un graphe en Python, avec les opérations suivantes :

- ➊ ajouter / supprimer une arête
- ➋ savoir s'il existe une arête entre 2 sommets
- ➌ connaître la liste des voisins d'un sommet
- ➍ ...

Avec, si possible, une faible complexité en temps et espace.

# Représentation de graphe

On souhaite stocker un graphe en Python, avec les opérations suivantes :

- ➊ ajouter / supprimer une arête
- ➋ savoir s'il existe une arête entre 2 sommets
- ➌ connaître la liste des voisins d'un sommet
- ➍ ...

Avec, si possible, une faible complexité en temps et espace.

Il existe principalement 2 représentations possibles de graphes :

- ➊ Par matrice d'adjacence
- ➋ Par liste d'adjacence

# Matrice d'adjacence : Définition

On peut représenter un graphe non orienté  $(V, E)$ , où  $V = \{0, \dots, n-1\}$  par une **matrice d'adjacence**  $A$  de taille  $n \times n$  définie par :

- $A_{i,j} = 1 \iff \{i, j\} \in E$
- $A_{i,j} = 0 \iff \{i, j\} \notin E$

Remarque :  $A$  est symétrique.

# Matrice d'adjacence : Définition

On peut représenter un graphe non orienté  $(V, E)$ , où  $V = \{0, \dots, n-1\}$  par une **matrice d'adjacence**  $A$  de taille  $n \times n$  définie par :

- $A_{i,j} = 1 \iff \{i, j\} \in E$
- $A_{i,j} = 0 \iff \{i, j\} \notin E$

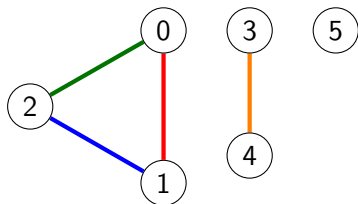
Remarque :  $A$  est symétrique.

Pour un graphe orienté  $(V, \vec{E})$  :

- $A_{i,j} = 1 \iff (i, j) \in \vec{E}$
- $A_{i,j} = 0 \iff (i, j) \notin \vec{E}$

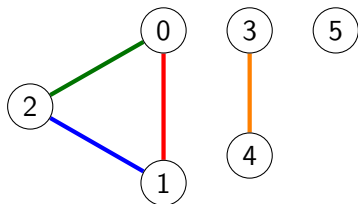
$A$  n'est pas symétrique (a priori).

## Matrice d'adjacence : Exemple



$$\begin{pmatrix} 0 & \textcolor{red}{1} & \textcolor{green}{1} & 0 & 0 & 0 \\ \textcolor{red}{1} & 0 & \textcolor{blue}{1} & 0 & 0 & 0 \\ \textcolor{green}{1} & \textcolor{blue}{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \textcolor{orange}{1} & 0 \\ 0 & 0 & 0 & \textcolor{orange}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

## Matrice d'adjacence : Exemple



$$\begin{pmatrix} 0 & \textcolor{red}{1} & \textcolor{green}{1} & 0 & 0 & 0 \\ \textcolor{red}{1} & 0 & \textcolor{blue}{1} & 0 & 0 & 0 \\ \textcolor{green}{1} & \textcolor{blue}{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \textcolor{orange}{1} & 0 \\ 0 & 0 & 0 & \textcolor{orange}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Quitte à permuter lignes et colonnes (i.e renuméroter les sommets), les composantes connexes apparaissent par bloc.

## Matrice d'adjacence : Exemple

En Python, **une matrice est une liste de listes** (ou un tableau de tableaux, avec numpy).



## Matrice d'adjacence : Exemple

En Python, **une matrice est une liste de listes** (ou un tableau de tableaux, avec numpy).

Par exemple, voici une matrice avec sa représentation en Python :

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

$$M = [[1, 0, 3], [2, 2, 1]]$$

## Matrice d'adjacence : Exemple

En Python, **une matrice est une liste de listes** (ou un tableau de tableaux, avec numpy).

Par exemple, voici une matrice avec sa représentation en Python :

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

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

Code	<code>M[i]</code>	<code>M[i][j]</code>	<code>len(M)</code>	<code>len(M[0])</code>
Signification	<i>i</i> ème ligne	élément ligne <i>i</i> , colonne <i>j</i>	nombre de lignes	nombre de colonnes

# Matrice d'adjacence : Exemple

Création de matrice d'adjacence d'un graphe à  $n$  sommets et 0 arête :

---

```
def make_matrix(n):  
    G = []  
    for i in range(n): # nombre de colonnes  
        L = [] # création de la ième ligne  
        for j in range(n):  
            L.append(0)  
        G.append(L)  
    return G
```

---

# Matrice d'adjacence : Exemple

Création de matrice d'adjacence d'un graphe à  $n$  sommets et 0 arête :

---

```
def make_matrix(n):  
    G = []  
    for i in range(n): # nombre de colonnes  
        L = [] # création de la ième ligne  
        for j in range(n):  
            L.append(0)  
        G.append(L)  
    return G
```

---

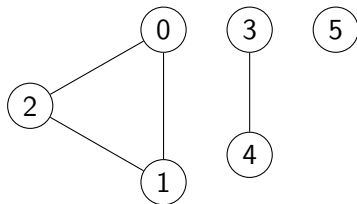
Ajout d'une arête :

---

```
def add_edge(G, u, v):  
    G[u][v] = 1  
    G[v][u] = 1
```

---

## Matrice d'adjacence : Exemple



Création de ce graphe en Python, par matrice d'adjacence :

---

```
G = make_matrix(6)
add_edge(G, 0, 1)
add_edge(G, 0, 2)
add_edge(G, 1, 2)
add_edge(G, 3, 4)
```

---

## Matrice d'adjacence : Degré

### Exercice

Écrire une fonction  $\text{deg}(G, v)$  pour calculer le degré d'un sommet  $v$  dans un graphe représenté par matrice d'adjacence  $G$ .

# Matrice d'adjacence : Degré

## Exercice

Écrire une fonction `deg(G, v)` pour calculer le degré d'un sommet  $v$  dans un graphe représenté par matrice d'adjacence  $G$ .

On regarde le nombre de 1 sur la ligne  $v$  de la matrice :

---

```
def deg(G, v):  
    res = 0  
    for j in range(len(G[v])):  
        if G[v][j] == 1:  
            res += 1  
    return res
```

---

# Matrice d'adjacence : Liste des voisins

## Exercice

Écrire une fonction `voisins(G, v)` pour renvoyer la liste des voisins de  $v$  dans  $G$ .



# Matrice d'adjacence : Liste des voisins

## Exercice

Écrire une fonction `voisins(G, v)` pour renvoyer la liste des voisins de `v` dans `G`.

---

```
def voisins(G, v):  
    res = []  
    for j in range(len(G[v])):  
        if G[v][j] == 1:  
            res.append(j)  
    return res
```

---

# Matrice d'adjacence : Avec POO

Exemple de **programmation orientée objet** (hors programme) :

---

```
class Graph:
    def __init__(self, n):
        self.G = [[0]*n for _ in range(n)]
    def add_edge(self, u, v):
        self.G[u][v] = 1
        self.G[v][u] = 1
    def del_edge(self, u, v):
        self.G[u][v] = 0
        self.G[v][u] = 0
    def is_edge(self, u, v):
        return self.G[u][v] == 1
    def deg(self, u):
        d = 0
        for v in range(len(self.G[u])):
            if self.G[u][v] == 1:
                d += 1
        return d
```

---

# Liste d'adjacence

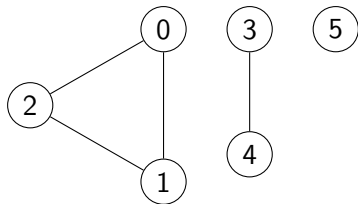
La représentation par **liste d'adjacence** consiste à stocker, pour chaque sommet, la liste de ses voisins.

# Liste d'adjacence

La représentation par **liste d'adjacence** consiste à stocker, pour chaque sommet, la liste de ses voisins.

On utilise pour cela une liste  $G$  de listes, telle que  $G[u]$  soit la liste des voisins de  $u$ .

## Liste d'adjacence : Exemple



### Matrice d'adjacence

```
[[0, 1, 1, 0, 0, 0],  
 [1, 0, 1, 0, 0, 0],  
 [1, 1, 0, 0, 0, 0],  
 [0, 0, 0, 0, 1, 0],  
 [0, 0, 0, 1, 0, 0],  
 [0, 0, 0, 0, 0, 0]]
```

### Liste d'adjacence

```
[[1, 2],  
 [0, 2],  
 [0, 1],  
 [4],  
 [3],  
 []]
```

# Liste d'adjacence : Fonctions

Pour créer une liste d'adjacence d'un graphe à  $n$  sommets et 0 arête :

---

```
def make_list(n):  
    G = []  
    for i in range(n):  
        G.append([])  
    return G
```

---

# Liste d'adjacence : Fonctions

Pour créer une liste d'adjacence d'un graphe à  $n$  sommets et 0 arête :

---

```
def make_list(n):  
    G = []  
    for i in range(n):  
        G.append([])  
    return G
```

---

Ou, avec une création de liste par compréhension :

---

```
def make_list(n):  
    return [[] for i in range(n)]
```

---

## Liste d'adjacence : Fonctions

Pour ajouter une arête à une liste d'adjacence d'un graphe orienté :



## Liste d'adjacence : Fonctions

Pour ajouter une arête à une liste d'adjacence d'un graphe orienté :

---

```
def add_edge(G, u, v):  
    G[u].append(v)
```

---

### Exercice

Écrire une fonction pour calculer le nombre d'arêtes d'un graphe orienté représenté par liste d'adjacence.

## Liste d'adjacence : Fonctions

### Exercice

Écrire une fonction pour calculer le nombre d'arêtes d'un graphe orienté représenté par liste d'adjacence.

### Exercice

Écrire deux fonctions pour convertir une matrice d'adjacence en liste d'adjacence et vice-versa.

# Liste d'adjacence : Fonctions

## Exercice

Écrire une fonction pour calculer le nombre d'arêtes d'un graphe orienté représenté par liste d'adjacence.

## Exercice

Écrire deux fonctions pour convertir une matrice d'adjacence en liste d'adjacence et vice-versa.

## Exercice

Écrire une fonction pour renvoyer la transposée d'un graphe orienté représenté sous forme de liste d'adjacence, c'est-à-dire en inversant le sens de toutes les flèches.

# Comparaison matrice d'adjacence / liste d'adjacence

Pour un graphe orienté à  $n$  sommets et  $m$  arêtes :

Opération	Matrice d'adjacence	Liste d'adjacence
espace	$\Theta(n^2)$	$\Theta(n + m)$
ajouter (u, v)	$O(1)$	$O(1)$
supprimer (u, v)	$O(1)$	$O(\deg^+(u))$
existence (u, v)	$O(1)$	$O(\deg^+(u))$
voisins de u	$\Theta(n)$	$\Theta(\deg^+(u))$
ajouter sommet	X	X
supprimer sommet	X	X

# Comparaison matrice d'adjacence / liste d'adjacence

Pour un graphe orienté à  $n$  sommets et  $m$  arêtes :

Opération	Matrice d'adjacence	Liste d'adjacence
espace	$\Theta(n^2)$	$\Theta(n + m)$
ajouter (u, v)	$O(1)$	$O(1)$
supprimer (u, v)	$O(1)$	$O(\deg^+(u))$
existence (u, v)	$O(1)$	$O(\deg^+(u))$
voisins de u	$\Theta(n)$	$\Theta(\deg^+(u))$
ajouter sommet	X	X
supprimer sommet	X	X

Si  $m = \Theta(n^2)$  (graphe dense) : matrice d'adjacence conseillée.

Si  $m = O(n)$  (graphe creux, ex : arbre) : liste d'adjacence conseillée.