Graphes : Représentations

Quentin Fortier

April 4, 2022

Structure abstraite graphe

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

1. ajouter / supprimer une arête
2. savoir s'il existe une arête entre 2 sommets
3. connaître la liste des voisins d'un sommet
4. ...

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

Matrice d'adjacence

On peut représenter un graphe non orienté (V, E), où $V = \{0, ..., n-1\}$ 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

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

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

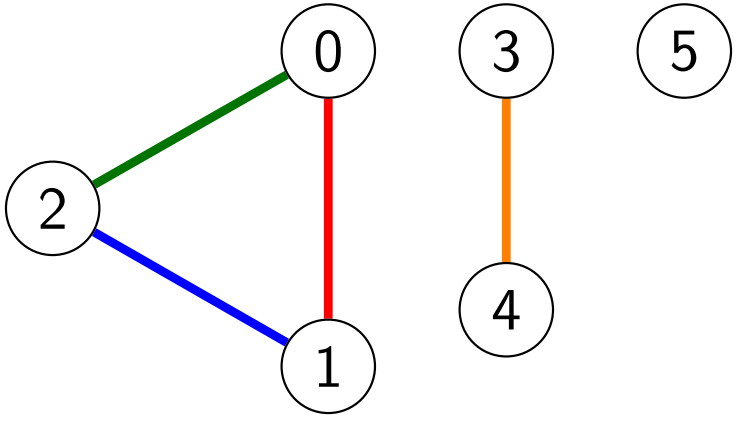
Remarque : A est symétrique.

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

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

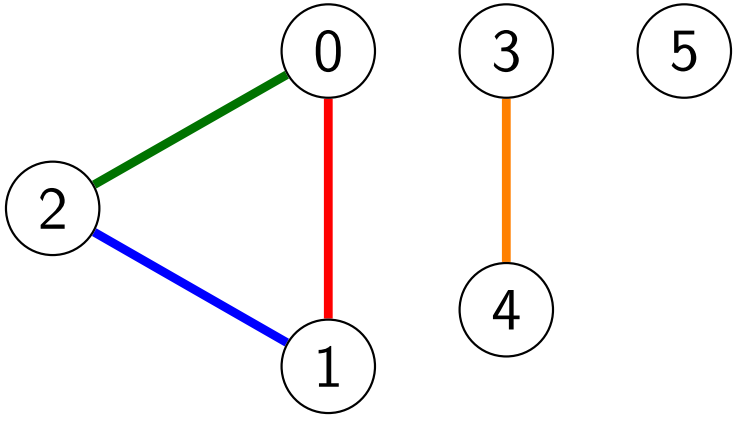
A n'est pas symétrique (a priori).

Exemple de matrice d'adjacence (non orienté)



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

Exemple de matrice d'adjacence (non orienté)



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

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

Matrice d'adjacence

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

Matrice d'adjacence

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

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	`M[i]`	`M[i][j]`	`len(M)`	`len(M[0])`
Signification	$i$ème ligne	élément ligne i, colonne j	nombre de lignes	nombre de colonnes

Matrice d'adjacence

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

```
def creer_graphe_matrice(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

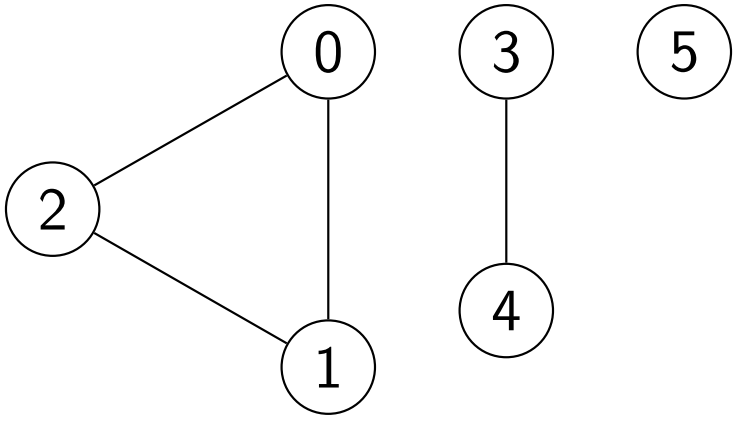
Création de matrice d'adjacence d'un graphe à n sommets et 0 arête :

```
def creer_graphe_matrice(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
```

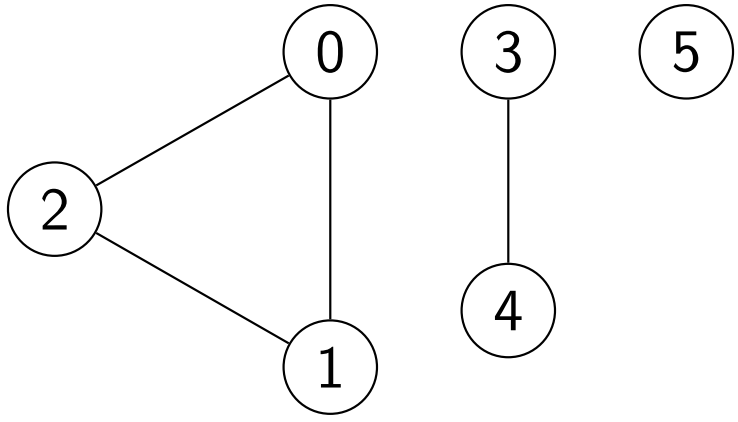
Matrice d'adjacence



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

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

Matrice d'adjacence



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

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

Exercice

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

Matrice d'adjacence

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
```