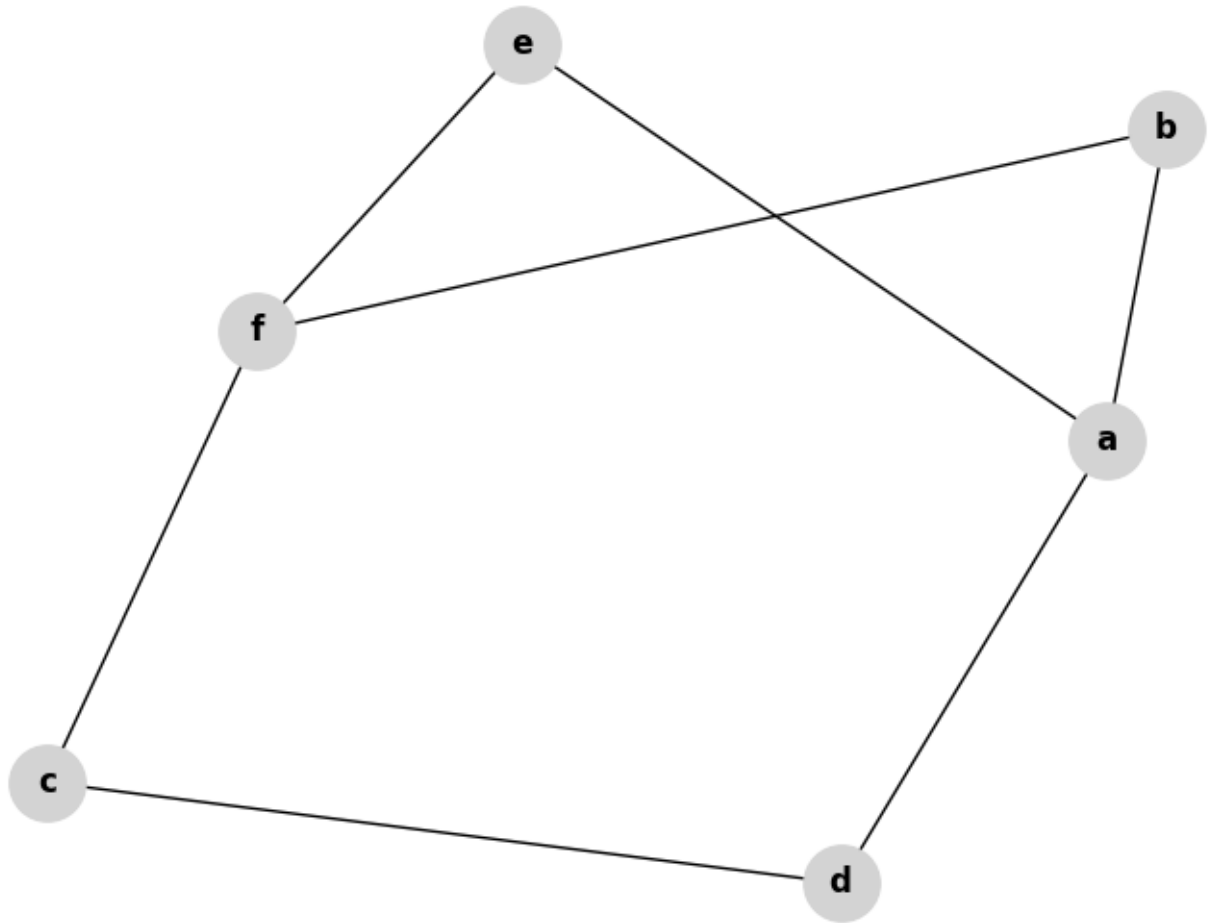


Numérique et Sciences Informatiques  
Chapitre V - Structures de données relationnelles  
Travaux Dirigés 09

## I. Graphe non orienté

1. Voici la représentation d'un graphe non orienté.



- Donner les voisins du sommet  $a$ .
- Donner le degré du sommet  $e$ .
- Donner la représentation de ce graphe par la liste des voisins.
- Donner la représentation de ce graphe par sa matrice d'adjacence.

2. Voici la matrice d'adjacence d'un graphe non orienté de sommets  $a, b, c, d$  et  $e$ .

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

- Quels sont les voisins du sommet  $a$  ?
- Quel est le degré du sommet  $c$  ?
- Donner la représentation de ce graphe par la liste des voisins.
- représenter ce graphe par sommets et arêtes.

## II. Utilisation de la bibliothèque NetworkX

Le but de cette partie est de créer un graphe non orienté avec la bibliothèque `NetworkX` mais aussi de voir certaines de ses fonctionnalités.

Le principe est le suivant :

- On importe le module.

- On crée un graphe vide.
- On lui ajoute des sommets.
- On lui ajoute des arêtes (ou arcs).

1. Exécuter le logiciel \*Thonny\*.

2. Importer le module 'networkx' sous l'alias 'nx'.

3. Intéressons-nous aux graphes non orientés.

Le but est de refaire le graphe de la partie I :

(a) Créer un graphe vide :

```
1      graphe1 = nx.Graph()
```

(b) Ajouter 6 sommets.

Pour ajouter un sommet ('node' dans ce module), on écrit l'instruction suivante :

```
1      graphe1.add_node(etiquette_du_sommet)
```

(c) Ajouter les arêtes.

Pour ajouter une arête ('edge' dans ce module), on écrit l'instruction suivante :

```
1      graphe1.add_edge(etiquette_1 , etiquette_2)
```

(d) Dessiner le graphe.

Pour dessiner le graphe, on écrit les instructions suivantes :

```
1      import matplotlib.pyplot as plt
2      nx.draw(graphe1, with_labels = True, font_weight = 'bold',
3              node_size = 800, node_color = 'lightgrey')
4      plt.show()
```

4. Testez les différentes instructions ci-dessous. Indiquez ce qu'elle font.

```
1  >>> graphe1.is_directed()
2  >>> graphe1.degree('a')
3  >>> graphe1.degree()
4  >>> print(nx.info(graphe1))
5  >>> print(nx.info(graphe1, 'a'))
6  >>> nx.is_empty(graphe1)
7  >>> graphe1.nodes()
8  >>> graphe1.number_of_nodes()
9  >>> graphe1.neighbors('a')
10 >>> list(graphe1.neighbors('a'))
11 >>> list(nx.all_neighbors(graphe1, 'a'))
12 >>> list(nx.non_neighbors(graphe1, 'a'))
13 >>> list(nx.common_neighbors(graphe1, 'a', 'c'))
14 >>> graphe1.edges()
15 >>> graphe1.number_of_edges()
16 >>> list(nx.non_edges(graphe1))
17 >>> list(nx.all_simple_paths(graphe1, 'a', 'e'))
18 >>> list(nx.simple_cycles(graphe1))
```

### III. Application : les cartes routières

Toute cette partie se fait dans le logiciel *Thonny*.

1. Dans un nouveau fichier Python, importer le fichier `TD09\data_villes.py`.

Il contient :

- un tableau `liste_des_villes`. Chaque élément de ce tableau est un tableau dont le premier élément est le nom d'une ville et les deux autres éléments sont ses coordonnées GPS.
- une matrice de distance `matrice_distance` qui donne les distances entre 2 villes.

Ainsi `matrice_distance[0][1]` renverra la distance entre la ville n°0 et la ville n°1 de la liste des villes. On peut donc dire que la distance entre *Annecy* et *Auxerre* est de 342.06521589 km.

2. Créer un graphe non orienté `villes`, d'étiquettes le nom des villes et tel que :

- *Lille* est relié à *Lens*, *Boulogne*, *Paris*
- *Boulogne* est reliée au *Havre*
- *Le Havre* est relié à *Caen*, *Paris*
- *Caen* est relié à *Paris*, *Rennes*, *Brest*
- *Rennes* est relié à *Nantes*, *Paris*
- *Paris* est relié à *Bordeaux*, *Auxerre*, *Strasbourg*
- *Strasbourg* est relié à *Nancy*
- *Nancy* est relié à *Metz*, *Sedan*
- *Lyon* est relié à *Auxerre*, *Saint-Etienne*, *Annecy*, *Grenoble*, *Marseille*
- *Bordeaux* est relié à *Toulouse*
- *Toulouse* est relié à *Marseille*
- *Marseille* est relié à *Nice*

3. Dessiner ce graphe.

4. Créer deux variables globales `NUMERO_VILLES` et `NOM_VILLES`, toutes deux des dictionnaires et qui donne respectivement la correspondance entre le nom de la ville et son numéro pour la première et le numéro de la ville et son nom pour la deuxième.

5. Créer la matrice d'adjacence de ce graphe. *On pourra utiliser une fonction.*

6. Grâce à la matrice d'adjacence, déterminer le degré du sommet *Lyon*.

7. Déterminer tous les chemins qui vont de *Lille* à *Marseille*.

8. On ne peut pas déterminer le plus court des chemins sans ajouter un poids à chaque chemin. Par défaut, le poids de chaque arête vaut 1. Pour ajouter un poids à un chemin, il suffit d'écrire les instructions suivantes :

```
1 villes[sommet1][sommet2]['weight'] = valeur
```

9. Déterminer le plus court chemin qui va de *Lille* à *Marseille*.