

## Exercices sur les Graphes et leur implémentation

Un exemple simple présente ici un graphe créé à partir d'une liste de voisins : `[[1, 2], [0, 2, 3], [0, 1, 3], [1, 2]]`

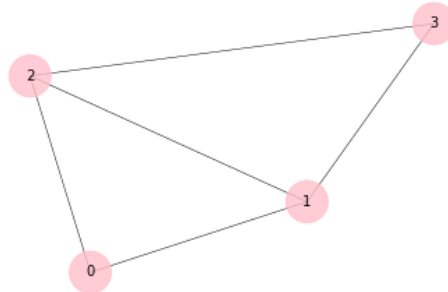


FIGURE 1 – graphe créé à partir de la liste d'adjacence

La première sous-liste correspond aux liens que forme le sommet 0. Ici, c'est donc avec les sommets 1 et 2.

### 1.1 Interface avec des fonctions

1. Rappels : que retourne chacune des expressions logiques :

```

1 # expression a
2 j = 1
3 j in [1,2]
4 # expression b
5 j = 1
6 j in [2,3]
7 # expression c
8 j = 1
9 L = [[1,2],[3,4]]
10 j in L[0]
11 # expression d
12 j = 1
13 L = [[1,2],[3,4]]
14 j in L[1]
```

2. Ecrire une fonction `est_lie` qui prend 3 paramètres, `L`, `i` et `j` et qui retourne `True` si les sommets `i` et `j` sont liés. *Exemple d'utilisation :*

```

1 >>> L = [[1,2],[3,4]]
2 >>> est_lie(L,0,1)
3 True
```

3. Soit la liste `L` suivante : `L = [[1,2],[3,4]]` On souhaite ajouter la valeur 5 à la 2e sous-liste de `L` pour obtenir :

```
1 print(L)
2 [[1,2],[3,4,5]]
```

Quelle instruction python faut-il écrire ?

4. Ecrire une fonction `ajoute_lien` qui prend en paramètres `L`, `i` et `j`, et qui ajoute `j` à la liste d'adjacence `L` à la sous-liste de rang `i`.

## 1.2 Interface en POO : liste de successeurs

La classe suivante permet d'implémenter ce graphe :

```
1 class GraphLS:
2     def __init__(self, lst):
3         self.lst = lst
4
5     def est_lie(self, i, j):
6         return j in self.lst[i]
7
8     def ajoute_sommet(self, i, j):
9         # a completer
10        ...
```

La méthode de classe `est_lie` retourne `True` si les sommets `i` et `j` sont liés, c'est à dire, si `j` est dans la liste au rang `i`.

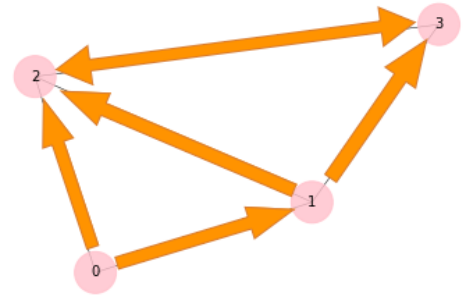
La méthode de classe `ajoute_sommet` sera décrite plus tard.

### 1.2.1 Graphe non orienté

**Questions :**

1. Ecrire les instructions de construction de l'objet `graphe` à partir de la classe `GraphLS`. L'objet `graphe` implémente le graphe du schéma ci-dessus.
2. Ecrire l'instruction qui vérifie si les sommets 1 et 2 sont liés.
3. La méthode de classe `ajoute_sommet` prend 2 paramètres, `i` et `j`. Elle a pour rôle d'ajouter au sommet `i` un nouveau lien vers le sommet `j`. Compléter la méthode `ajoute_sommet` ci-dessus.
4. Ecrire l'instruction qui ajoute le lien `0 <-> 3`.

### 1.2.2 Graphe orienté



Cette représentation est particulièrement adaptée aux *graphes orientés* :

On peut représenter un graphe avec une liste chaînée des successeurs :

*sommet => liste de sommets liés suivants :*

- 0 => 1, 2
- 1 => ...
- 2 => ...
- 3 => ...

Le sommet 0 aura alors 2 successeurs, les noeuds 1 et 2.

1. Compléter la liste de successeurs
2. Ecrire les instructions quiinstancient chaque sommet du graphe en exemple ci-dessus. Renseigner les noms des sommets (attribut `val`), ainsi que chaque successeur (attribut `suiv`).
3. Ecrire l’instruction qui instancie le graphe.
4. Ecrire la méthode de classe qui vérifie si le sommet `premier` est lié au sommet successeur de valeur `j`.

### 1.3 Matrice d’adjacence

Cette liste `L` peut aussi être mise sous forme d’une matrice `M` :

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

ce qui forme une matrice, en présentant les sous-listes l’une sous l’autre :

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

Avec la première sous-liste `[0, 1, 1, 0]` le sommet 0 est lié aux sommets 1 et 2, ce qui est signifié par le `1` aux index 1 et 2.

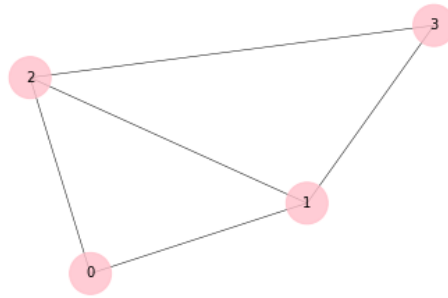


FIGURE 2 – graphe correspondant à la matrice M

Cette représentation en matrice est particulièrement adaptée aux *graphes pondérés*. On remplace alors le 1 dans la matrice par le poids de l'arête.

### Questions :

1. A partir de la matrice précédente : Dessiner un tableau avec pour lignes et colonnes les numeros des sommets du graphe, et pour valeurs
  - 0 (les sommets ne sont pas liés)
  - ou 1 (les sommets sont liés).
2. Ecrire une fonction qui vérifie si les sommets  $i$  et  $j$  sont directement liés.
3. Ecrire une fonction qui ajoute au sommet  $i$  un nouveau lien vers le sommet  $j$ . Puis l'instruction qui ajoute le lien  $0 \leftrightarrow 3$ .
4. **Graphe pondéré** : Les valeurs de la matrice indiquent le poids de chaque arête, lorsqu'elle existe. Ces poids peuvent représenter par exemple la distance en suivant le chemin repéré par l'arête. (*cartographie*). Représenter le graphe pondéré dont la matrice d'adjacence est donnée ci-dessous :

$$\begin{pmatrix} 0 & 10 & 10 & 9 \\ 10 & 0 & 5 & 10 \\ 10 & 5 & 0 & 10 \\ 9 & 10 & 10 & 0 \end{pmatrix}$$

5. Représenter le graphe pondéré et orienté pour la matrice suivante :

$$\begin{pmatrix} 0 & 0 & 0 & 9 \\ 0 & 0 & 5 & 10 \\ 0 & 5 & 0 & 10 \\ 9 & 10 & 10 & 0 \end{pmatrix}$$

## 1.4 Graphe avec étiquette, dictionnaire

### 1.4.1 Rappels

Soit le dictionnaire D suivant :  $D = \{ 'a' : [ 'b', 'c' ] \}$

1. Quelle instruction donne la liste des clés de D ?
2. Que retourne l'instruction `D['a']` ?
3. Que retourne l'instruction `'b' in D` ?
4. Que retourne l'instruction `'b' in D['a']` ?
5. Comment est modifié le dictionnaire D lorsque l'on exécute : `D['a'].append('d')` ?
6. Comment est modifié le dictionnaire D lorsque l'on exécute : `D['b'] = ['a', 'c']` ?

### 1.4.2 Implémentation

Pour implémenter le graphe, on utilisera un *dictionnaire* comme structure de données. Les clés étant les étiquettes des sommets, et les valeurs, la liste des sommets adjacents :

`D = {'a' : ['b', 'c'], 'b' : ['a', 'c', 'd'], 'c' : ['a', 'b', 'd'], 'd' : ['b', 'c']}`

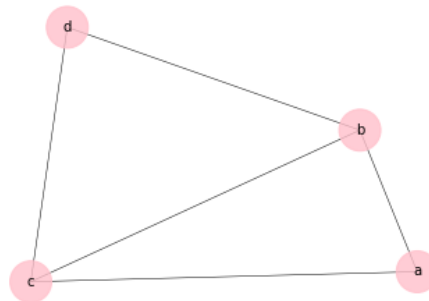


FIGURE 3 – graphe correspondant au dictionnaire D

#### Questions :

1. Représenter la matrice d'adjacence correspondante.
2. Ecrire une fonction qui vérifie si les sommets i et j sont liés. i et j sont des lettres parmi les clés et valeurs du graphe.
3. Ecrire une fonction qui ajoute au sommet i un nouveau lien vers le sommet j.