R2.07 TP 1

Introduction aux graphes avec `Sagemath`

28 janvier 2026 – B. Colombel

Sommaire

Dans ce TP, les graphes sont définis comme une paire (V,E), où V représente un ensemble de sommets (*vertices* en anglais) et E un ensemble d'arêtes (*edges* en anglais), c'est-à-dire de paires non ordonnées (ou ordonnée si le graphe est orienté) de sommets.

1 Construire un graphe

1.1 À partir de la matrice d'adjacence

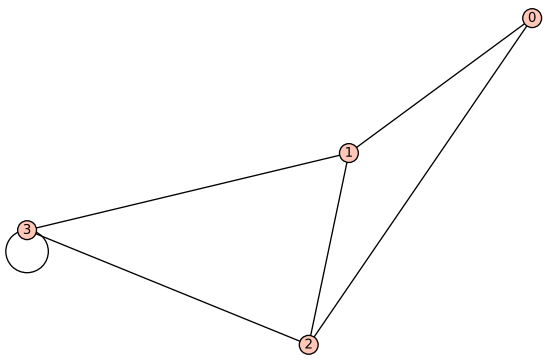
Si $\mathcal{G} = (V,E)$ est un graphe orienté, sa matrice d'adjacence $M = (m_{ij})$ est définie par :

$$m_{ij} = \begin{cases} 1 & \text{si il existe un arc allant de } i \text{ vers } j \\ 0 & \text{sinon} \end{cases}$$

Pour un graphe non orienté, la définition est similaire sans tenir compte du sens de l'arc.
La matrice d'adjacence d'un graphe non orienté est donc symétrique ; les coefficients de la diagonale égaux à 1 marquent la présence d'une boucle.
Par exemple, le graphe de matrice

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

est le graphe :



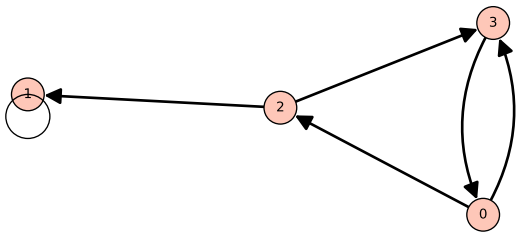
[1]:
```
M = matrix([[0, 1, 1, 0],
            [1, 0, 1, 1],
            [1, 1, 0, 1],
            [0, 1, 1, 1]])

G = Graph(M)  # Graphe non-orienté de matrice d'adjacence M
G.show() # Afficher le diagramme du graphe
```

On peut alors retrouver l'ordre et la taille du graphe :

[2]:
```
G.order(), G.size()
```

Pour les graphes **orientés**, il fuadra utiliser le mot clé `DiGraph()` à la place de `Graph()` :

[3]:
```
N = matrix([[0, 0, 1, 1],
            [0, 1, 0, 0],
            [0, 1, 0, 1],
            [1, 0, 0, 0]])

G2 = DiGraph(N)
G2.show()
```



[4]:
```
G2.order(), G2.size()
```

On peut alors retrouver la matrice d'adjacence des graphes avec la méthode `adjacency_matrix()` .

[5]:
```
G.adjacency_matrix()
```

[5]:
```
[0 1 1 0]
[1 0 1 1]
[1 1 0 1]
[0 1 1 1]
```

[6]:
```
G2.adjacency_matrix()
```

[6]:
```
[0 0 1 1]
[0 1 0 0]
[0 1 0 1]
[1 0 0 0]
```

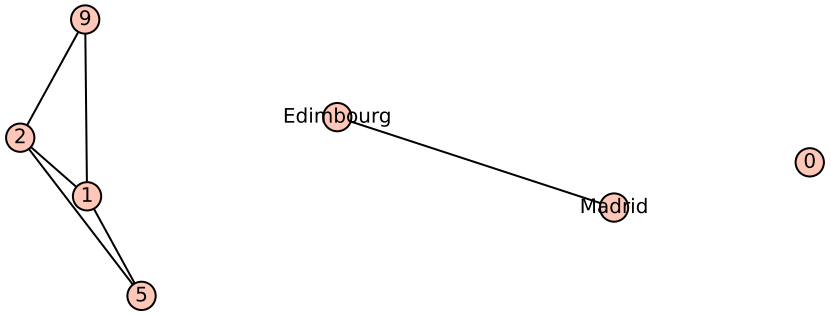


FIGURE 1 – Construction d'un graphe pas à pas

1.2 À partir de rien

Le graphe de la figure 1 page 3 est défini sur l'ensemble de sommets {0,1,2,5,9,"Madrid","Edimbourg"} et a pour arêtes {(1,2),(1,5),(1,9),(2,5),(2,9)} ainsi que ("Madrid","Edimbourg").

[7]:
```
g = Graph()
```

Par défaut, `g` est un graphe vide. L'exemple suivant illustre comment lui ajouter sommets et arêtes : lorsqu'une arête est créée, les sommets correspondant — s'ils ne sont pas déjà présents dans le graphe — sont silencieusement ajoutés. Nous contrôlons le déroulement de la procédure avec des méthodes dont les rôles sont faciles à deviner :

[8]:
```
g.order(), g.size()
```

[9]:
```
(0, 0)
```

[10]:
```
g.add_vertex(0)
g.order(), g.size()
```

[10]:
```
(1, 0)
```

[11]:
```
g.add_vertices([1, 2, 5, 9])
g.order(), g.size()
```

[11]:
```
(5, 0)
```

[12]:
```
g.add_edges([(1,5), (9,2), (2,5), (1,9)])
g.order(), g.size()
```

[12]:
```
(5,4)
```

[13]:
```
g.add_edge("Madrid", "Edimbourg")
g.order(), g.size()
```

[13]:
```
(7, 5)
```

Remarques.
1. Ajouter l'arête (1,2) est équivalent à ajouter l'arête (2,1).
2. les méthodes `add_vertex()` et `add_edge()` ont toutes les deux un « pluriel » (`add_vertices()` et `add_edges()`) qui prend une liste en argument et permet une écriture plus compacte.

Il est bien entendu possible de supprimer ensuite les éléments ajoutés à l'aide des méthodes `delete_*` et d'énumérer les sommets ou les arêtes, sur lesquels nous itérerons souvent.

[14]:
```
g.delete_vertex(0)
g.delete_edges([(1,5), (2,5)])
g.order(), g.size()
```

[14]:
```
(6, 3)
```

[15]:
```
g.vertices(sort = False) # nécessaire car le noms des sommets ne sont pas de même type
```

[15]:
```
[1, 2, 5, 9, 'Edimbourg', 'Madrid']
```

[16]:
```
g.edges(sort = False) # nécessaire car le noms des sommets ne sont pas de même type
```

[16]:
```
[(1, 9, None), (2, 9, None), ('Edimbourg', 'Madrid', None)]
```

Remarque. Les arêtes d'un graphe sont en réalité pour Sage des triplets, dont la dernière entrée est une étiquette qui sert la plupart du temps à indiquer le « poids » d'une arête dans le cas d'un graphe pondéré. Nous y reviendrons dans d'autres TP. Par défaut, l'étiquette vaut `None`.

Lorsque l'on connaît à l'avance les sommets et arêtes d'un graphe, on peut le construire de manière compacte de la manière suivante :

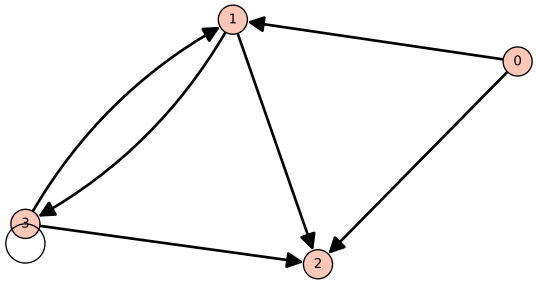
[17]:
```
g = Graph({
0 : [],
1 : [5, 9],
2 : [1, 5, 9],
'Edimbourg' : ['Madrid']})

g.show()
```

La structure de données utilisée ici s'appelle un *dictionnaire*.

2 Prédécesseurs, successeurs, voisins

2.1 Graphes orientés

Exercice 1. Construire le graphe ci-dessous (attention à la boucle du sommet 3) :



Dans un graphe orienté :
- le sommet j est un **successeur** du sommet i s'il existe une arête partant du sommet i et arrivant au sommet j
- dans ce cas, i est un **prédécesseur** de j.

Dans toute la suite, les graphes seront supposés **simples**, c'est-à-dire, *sans arêtes parallèles ni boucles.*

Exercice 2. On considère un graphe **orienté** G de matrice d'adjacence M ; x désigne un sommet de G, les sommets étant numérotés de 1 à $n-1$ si G est d'ordre n.

1. Construire une fonction `successeurs(M,x)` qui retourne une liste contenant les successeurs du sommet x dans le graphe orienté de matrice M.

2. Construire une fonction `predecesseurs(M,x)` sur le même modèle que la fonction précédente.
3. Construire les fonctions `degre_sortant(M,x)` et `degre_entrant(M,x)` qui retournent le degré entrant/sortant du sommet x.

Exemples d'utilisation.
Soit A la matrice du graphe construit à l'exercice n° 1.

[21]:
```
successeurs(A,1)
```

[21]:
```
[2, 3]
```

[22]:
```
predecesseurs(A,1)
```

[22]:
```
[0, 3]
```

[23]:
```
degre_sortant(A,3), degre_entrant(A,3)
```

[23]:
```
(3, 2)
```

2.2 graphe non-orienté

Exercice 3. Dans un graphe **simple non orienté** :
- les **voisins** du sommet x sont tous les sommets *adjacents* à x

1. Construire une fonction `voisins(M,x)` qui retourne une liste contenant les voisins du sommet x dans le graphe non orienté de matrice M.
2. Construire une fonction `degre(M,x)` qui retourne le degré du sommet x.

3 Graphes circulant

Le graphe circulant de paramètres n et d est un graphe à n sommets numérotés de 0 à $n-1$ (que l'on peut se représenter disposés le long d'un cercle), tel que deux sommets u et v sont reliés par une arête si

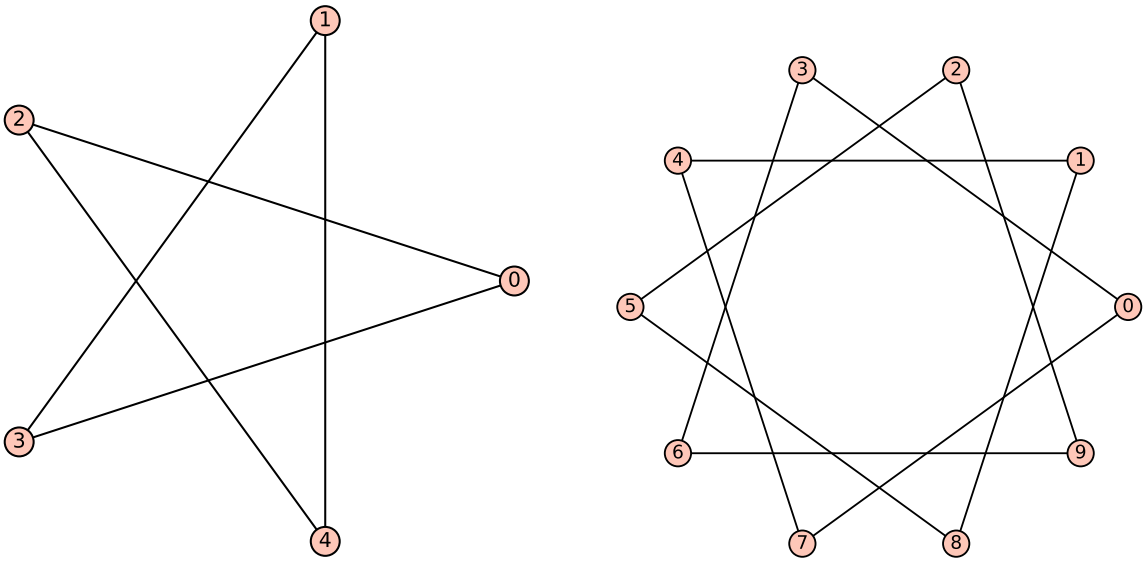
$$u \equiv v + c \mod n$$

avec $c = -d$ ou $c = d$.

Exercice 4. Écrire une fonction prenant en paramètres n et d, et renvoyant le graphe associé.

(Voir les exemples de la figure 2 page 6).

Astuce. Utiliser l'option `layout = 'circular'` de la méthode `show()`.



(a) $n = 5$ et $d = 2$ (b) $n = 10$ et $d = 7$

FIGURE 2 – Graphes circulant