TP 19
Graphes

Les **graphes** sont des structures de données (comme les listes, tableaux, dictionnaires, …) énormément utilisées en informatique et en mathématiques car ils sont assez simples à manipuler et modélisent de très nombreuses situations.

I Notion de graphe

Un graphe est tout simplement donné par un ensemble de sommets, reliés entre eux par des arêtes :

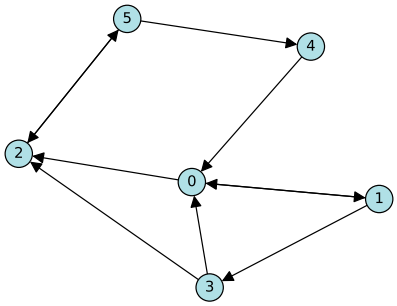


Figure 1. – Un graphe

Ce genre de dessin modélise de très nombreuses situations différentes :

- Une carte géographique, où les sommets sont des villes et les arêtes sont des routes reliant ces villes. La carte du métro parisien est assurément un graphe, de même que la carte des lignes de train en France.

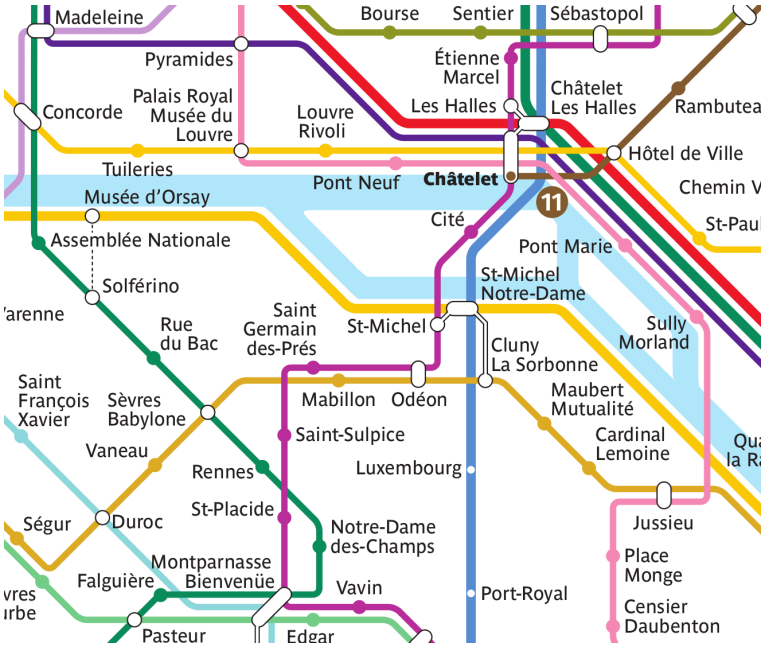


Figure 2. – Zoom sur un graphe bien connu. Source : RATP.

- Un réseau social, où les sommets sont des personnes et une arête entre deux personnes signifie que ces personnes sont amies, ou bien (suivant le sens de la flèche) que l'un est un *follower* de l'autre. Les gros réseaux sociaux ont absolument besoin d'algorithmes efficaces opérant sur des graphes avec des millions d'informations.

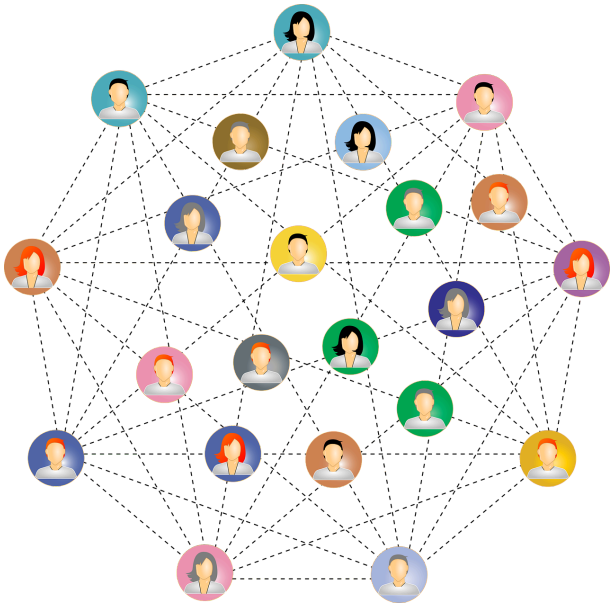


Figure 3. – Graphe d'un réseau social. Source : Pixabay (image libre).

- Un jeu de stratégie, où chaque sommet représente un état possible du jeu, et une arête représente une façon de passer d'un état à un autre. Jouer une partie revient à démarrer sur un sommet initial puis passer d'un sommet à un autre via des arêtes, jusqu'à arriver sur un sommet représentant une partie gagnée. Un jeu de labyrinthe peut se représenter par un graphe dont le but est d'arriver au sommet final, en partant d'un sommet initial donné.

La formalisation mathématique est celle-ci, prenant en compte la petite flèche dessinée sur les arêtes qui correspond à une orientation :

Définition

Un **graphe orienté** est la donné d'un ensemble fini S (ensemble des **sommets**) et d'un sous-ensemble $A \subset S \times S$ (ensemble des **arêtes**). Si x, y sont deux sommets, la condition $(x, y) \in A$ signifie qu'il y a une arête de x vers y.

Une autre variante de graphe est celle où on ne s'occupe pas du sens des arêtes, qu'on représente donc comme un simple trait entre sommets : deux sommets x et y sont ou bien ne sont pas connectés, peu importe dans quel ordre.

Définition

Un **graphe non orienté** est un graphe $G = (S, A)$ vérifiant la condition :

$$\forall (x, y) \in S^2, \quad (x, y) \in A \Leftrightarrow (y, x) \in A$$

Autrement dit on interprète la condition $(x, y) \in A$ comme signifiant qu'il existe une arête entre les sommets x et y, et ceci est équivalent à dire qu'il existe une arête entre les sommets y et x.

Notre définition générale de graphe n'exclut pas l'existence de **boucle** : une arête d'un sommet x à lui-même, $(x, x) \in A$.

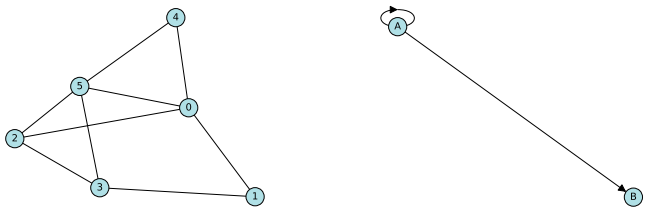


Figure 4. – À gauche, un graphe non orienté, représenté sans flèches.
À droite, une boucle sur le sommet A.

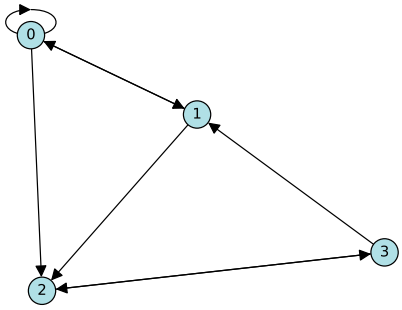
II Des graphes en Python

II.1 Représentation

Il y a plusieurs façons de représenter un graphe orienté $G = (S, A)$ en Python. On suppose qu'on a d'abord numéroté les N sommets de 0 à $N-1$, ainsi $S = \{0, 1, ..., N-1\}$.

- Par **liste d'adjacence** : on donne une liste `L` où pour chaque indice de sommet `i`, `L[i]` est la liste des sommets **vers lesquels** mène une arête issue de `i`. Mathématiquement c'est la liste des $\{j \in S \mid (i, j) \in A\}$.
- Par **matrice d'adjacence** : on donne une liste de listes `M` représentant une matrice carrée, où `M[i][j]` vaut 1 s'il y a une arête du sommet `i` vers le sommet `j` et 0 sinon.

Prenons par exemple le graphe très simple



Alors vérifiez que la liste d'adjacence est

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

et que la matrice d'adjacence est

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

- Si on ne souhaite pas numéroter les sommets, alors on peut leur donner un nom et travailler comme sur les listes d'adjacence mais avec à la place un **dictionnaire d'adjacence** : un dictionnaire où chaque clé est un sommet et la valeur correspondante est la liste des sommets vers lesquels il est relié. Voici par exemple un bout de graphe non-orienté représentant des lignes de train en France :

```
trains = {"Paris": ["Versailles", "Lyon", "Angers", "Besançon", "Clermont"],
"Versailles": ["Paris"], "Lyon": ["Paris", "Marseille", "Grenoble", "Besançon"],
"Angers": ["Paris"], "Besançon": ["Paris", "Lyon"], "Clermont": ["Paris"], "Marseille":
["Lyon", "Nice"], "Grenoble": ["Lyon"], "Nice": ["Marseille"]}
```

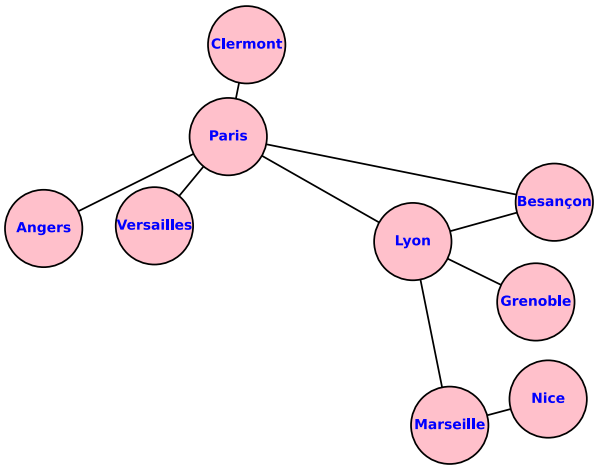


Figure 6. – Carte de France…

En fait, chaque représentation possible a ses avantages et ses inconvénients…

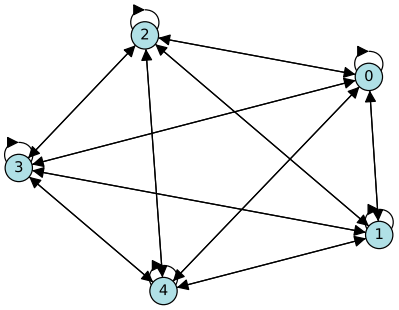
Exercice 1 *(Passer éventuellement et revenir plus tard)*
Écrire des fonctions

1. `liste_vers_matrice(L)` : convertit un graphe représenté par une liste d'adjacence, vers une matrice d'adjacence. On aura besoin de savoir créer une matrice nulle, et d'une double boucle pour itérer sur les listes d'adjacence de chaque sommet.
2. `matrice_vers_liste(M)` : convertit un graphe représenté par une matrice d'adjacence, vers une liste d'adjacence. On aura besoin de créer une liste de listes nulles, puis d'une double boucle pour parcourir la matrice d'adjacence et on utilisera `append` pour ajouter au fur et à mesure les sommets dans la liste d'adjacence.

II.2 Quelques graphes particuliers

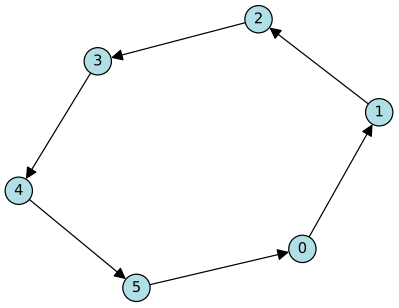
Étudions maintenant quelques exemples particuliers de graphes. On choisit pour les questions suivantes de travailler avec des matrices d'adjacence.

Exercice 2
Un graphe $G = (S, A)$ est dit **complet** s'il y a des arêtes entre tous les sommets possibles, c'est-à-dire $A = S \times S$. En particulier **le graphe complet** à n sommets est le graphe sur l'ensemble des sommets $\{0, 1, ..., n-1\}$ qui sont tous reliés entre eux.



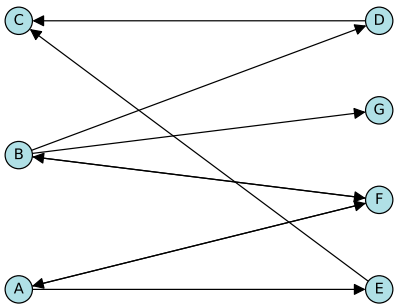
Écrire une fonction `graphe_complet(n)` qui renvoie la matrice d'adjacence du graphe complet à n sommets.

Exercice 3
Le **graphe cyclique** à n sommets est le graphe dont l'ensemble de sommets est $\{0, 1, ..., n-1\}$ et tel que le sommet i est relié au sommet $i+1$ (incluant le sommet $n-1$ relié au sommet 0). Il se représente naturellement en cercle.



Écrire une fonction `graphe_cyclique(n)` qui renvoie la matrice d'adjacence du graphe cyclique sur n sommets.

Exercice 4
Un graphe $G = (S, A)$ est dit **biparti** si on peut diviser l'ensemble des sommets en deux, $S = S_1 \cup S_2$ avec $S_1 \cap S_2 = \emptyset$, tel que les arêtes ne peuvent exister qu'entre un élément de S_1 et un élement de S_2, dans un sens ou dans l'autre. Mathématiquement $A \subset (S_1 \times S_2) \cup (S_2 \times S_1)$. Cela représente par exemple un ensemble de personnes dans une soirée divisé entre hommes et femmes, et où une arête de x vers y signifie « x a invité y à danser » ; il se représente naturellement avec les ensembles de sommets S_1 et S_2 face à face.



En particulier, **le graphe biparti complet** sur (n, p) sommets est le graphe dont l'ensemble des sommets est $S = \{0, 1, ..., n + p - 1\}$ divisé entre $S_1 = \{0, ..., n - 1\}$ et $S_2 = \{n, ..., n + p - 1\}$, et qui admet toutes les arêtes possibles entre les sommets de S_1 et de S_2.

Écrire une fonction `graphe_biparti_complet(n, p)` qui renvoie la matrice d'adjacence du graphe biparti complet sur (n, p) sommets.

II.3 Degré d'un sommet

Définition

Soit G un graphe orienté et soit x un sommet de G.

1. Le **degré sortant** de x est le nombre d'arêtes partant de x.
2. Le **degré entrant** de x est le nombre d'arêtes arrivant à x.
3. Le **degré** du sommet x est la somme du degré sortant et du degré entrant.
4. Le **degré total** d'un graphe est le maximal des degrés de tous ses sommets.

Dans un graphe représentant un réseau social, où une arête de x vers y signifie que x est un *follower* de y, le degré sortant de x est le nombre de personnes que suit x, et le degré entrant est le nombre de followers de x.

Exercice 5
Écrire les fonctions, prenant en argument un graphe orienté G représenté par une matrice d'adjacence `M` et un sommet `x` (un nombre entier) :

1. `degré_sortant(M, x)`,
2. `degré_entrant(M, x)`,
3. `degré(M, x)`,
4. `degré_total(M)`.

III Chemins et connexité

Un thème important est d'étudier les chemins dans un graphe, en passant d'un sommet au suivant via une arête.

Définition

Soit $G = (S, A)$ un graphe.

1. Soient $x, y \in S$ deux sommets. Un **chemin** dans G de x à y est la donnée d'une suite de sommets $(s_0, ..., s_n)$ tels que :
   - $s_0 = x$,
   - $s_n = y$,
   - $\forall 0 \leqslant i \leqslant n-1$, $(s_i, s_{i+1}) \in A$.
2. Le nombre n ci-dessus est appelé la **longueur** du chemin.
3. La **distance** de x à y est la longueur minimale parmi tous les chemins possibles de x à y.
4. Un **cycle** est un chemin partant d'un sommet et arrivant à lui-même : $s_0 = s_n$.

Remarquons que par convention, il existe toujours un chemin de longeur 0 d'un sommmet x à lui-même. Les arêtes de G sont exactement les chemins de longueur 1.

Un chemin dans un graphe, qu'il soit représenté en numérotant les sommets ou bien en leur donnant un nom (dictionnaire d'adjacence), est représenté en Python comme une simple liste `C` des sommets par lesquels le chemin passe, dans l'ordre. Alors `len(C)-1` est la longueur du chemin.

Exercice 6

1. On représente un chemin par une liste `C` des sommets à parcourir, dans l'ordre, dans un graphe représenté par une matrice d'adjacence `M`. Écrire une fonction `est_chemin_possible(C, M)` qui renvoie `True` si ce chemin est bien possible (c'est à dire s'il existe bien une arête de `C[i]` à `C[i+1]`, pour tout i) et `False` sinon.
2. Pouvez-vous écrire la même fonction mais en prenant cette fois-ci comme argument un dictionnaire d'adjacence ?

Exercice 7 *Mathématiques*

Soit G un graphe orienté et soit M sa matrice d'adjacence. Démontrer par récurrence que pour tout $p \in \mathbb{N}$, le coefficient (i, j) de M^p est le nombre de chemins de longueur exactement p reliant le sommet i au sommet j. On se concentrera sur le cas $p = 2$.

La notion suivante a du sens seulement si G est non-orienté.

Définition

Soit G un graphe non-orienté.

1. G est dit **connexe** si, entre tous sommets x et y, il existe au moins un chemin.
2. Les **composantes connexes** sont les sous-graphes de G maximaux (formés d'un plus grand nombre possible de sommets parmi ceux de G) qui sont connexes.

Un graphe non-connexe ressemble à ses composantes connexes posées simplement les unes à côté des autres.

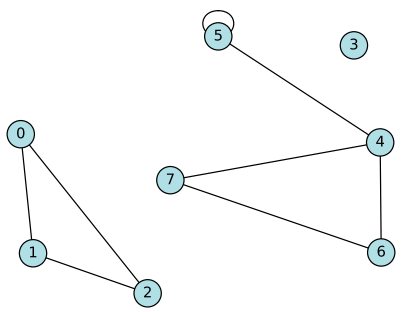


Figure 10. – Un graphe avec trois composantes connexes

Remarque. Si le graphe G a N sommets, alors pour tous sommets x, y, *si* il existe un chemin de x vers y, *alors* il en existe un de longueur inférieure ou égale à $N+1$ (*pourquoi ?*). Cela donne au moins en théorie un algorithme pour tester la connexité de G sans tester une infinité de chemins possibles : on écrit la matrice d'adjacence M, on calcule M^{N+1}, et cette matrice ne doit contenir que des coefficients strictement positifs.

Remarque. La matrice d'adjacence d'un graphe non-orienté est une matrice *symétrique*… En général, pour un graphe orienté, qu'est-ce que la transposée de la matrice d'adjacence ?

IV Parcours aléatoire

Exercice 8

Le chat passe son temps entre quatre activités : dormir sur le canapé (C), manger (M), sortir (S), et… dormir sur le lit (L). Son parcours d'une journée typique est le suivant :

- S'il dort sur le canapé, alors ensuite il peut aller manger ou sortir ou bien passer sur le lit.
- S'il mange, alors ensuite il peut sortir ou bien dormir sur le lit.
- S'il sort, alors ensuite il peut dormir sur le canapé ou le lit ou bien manger.
- S'il dort sur le lit, alors il peut rester sur le lit ou bien passer sur le canapé.

À chaque heure, il passe aléatoirement d'une activité à l'autre. On représente cette situation par un graphe orienté à quatre sommets, représentant les quatre activités possibles du chat, et des flèches indiquant qu'il peut passer d'une activité à l'autre.

1. Dessiner sur feuille le graphe correspondant.
2. Écrire en Python le dictionnaire d'adjacence `chat` correspondant.
3. Écrire une fonction `suivant(G, x)` prenant en argument un graphe `G` (quelconque) représenté par un dictionnaire d'adjacence, et un sommet `x`, et qui choisit au hasard un des sommets auxquels on peut accéder depuis `x`. On pourra pour cela utiliser la fonction `randint` du module `random` pour choisir un élément au hasard dans une liste.
4. En déduire une fonction `parcours(G, e, n)` qui part d'un sommet initial `e` dans un graphe `G`, et n fois de suite, choisit un sommet suivant au hasard.
5. Appliquer la fonction sur le dictionnaire `chat`, avec n assez grand, en affichant à chaque étape quelle est l'activité du chat.
6. Améliorer la fonction `parcours` en une fonction `parcours_compte(G, e, n)` qui renvoie un dictionnaire, constitué des même clés que `G`, qui indique combien de fois le parcours est passé sur chaque sommet ; et l'appliquer au graphe `chat`.


$\Omega = \{C, M, S, L\}$
from random import randint
$\mathbb{P}(X) = \sum_{i=1}^{n} \mathbb{P}(A_i)\mathbb{P}_{A_i}(X)$