TP 19
Graphes

Les *graphes* sont des structures de données (comme les listes, tableaux, dictionnaires, ...) énormément utilisées en informatique et en mathématiques.

I Notion de graphe

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

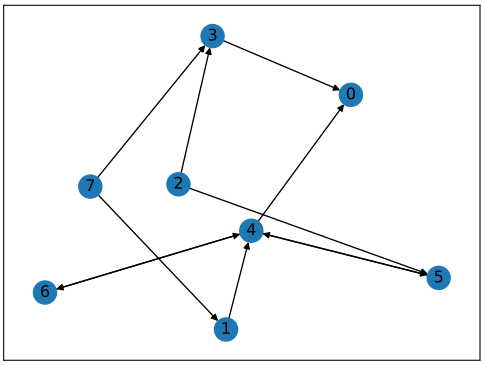

FIGURE 1 – Un graphe

Ce genre de dessin modélise de très nombreuses situations différentes :

1. Une carte géographique, où les sommets sont des villes et les arêtes sont des routes reliant ces villes. La carte du métro parisien est train certainement un graphe, de même que la carte des lignes de train en France.
2. Un réseau social, où les sommets sont des personnes et une arête entre deux personnes signifie que ces personnes sont amies. Les gros réseaux sociaux ont absolument besoin d'algorithmes efficaces opérant sur des graphes avec des millions d'informations.
3. 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.

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

Définition 1. 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 où ne sont pas connectés, peu importe dans quel ordre.

Définition 2. Un **graphe non orienté** est un graphe $G = (S, A)$ vérifiant la condition :

$$\forall (x, y) \in S^2, \quad (x, y) \in A \iff (y, x) \in A \tag{1}$$

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.

Remarque 1. Il existe en fait de nombreuses variantes de la définition de graphes :

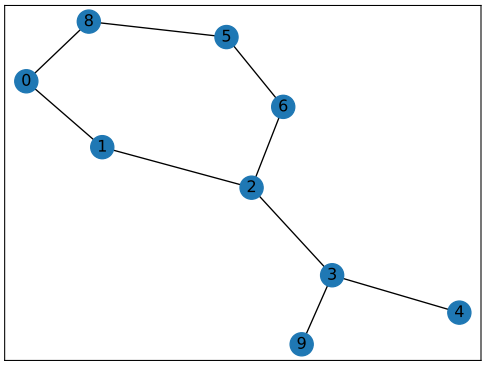

FIGURE 2 – Un graphe non orienté

1. Nous autorisons les **boucles** : une arête d'un sommet à lui-même, c'est à dire $x \in S$ tel que $(x, x) \in A$. On pourrait s'intéresser uniquement aux graphes sans boucles.
2. On pourrait vouloir plusieurs arêtes possibles entre deux mêmes sommets, correspondant à différentes façons d'aller de x à y. Cela n'arrive pas dans notre définition : il y a au plus une arête entre deux sommets.

Il est assez courant de vouloir attacher des nombres sur les sommets ou sur les arêtes des graphes. Par exemple sur un graphe représentant un réseau routier, les sommets sont les villes et les arêtes sont les routes avec une distance en kilomètre connue, et on pourra s'intéresser à des algorithmes pour trouver le chemin le plus court d'une ville à une autre...

II Représentation informatique des graphes

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

1. Par **liste d'adjacence** : on donne une liste `L` où pour chaque indice de sommet `i`, `L[i]` est la **liste** des sommets auxquels mène une arête issue de `i`. Mathématiquement c'est la liste des $\{j \in S \mid (i, j) \in A\}$.
2. 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

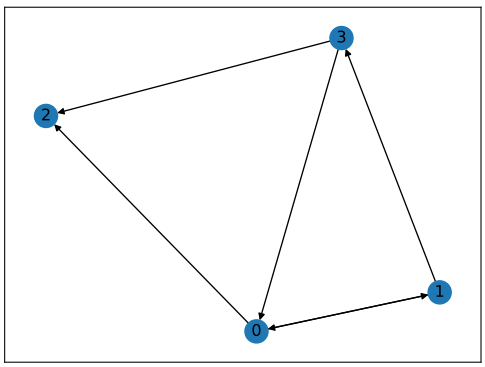

FIGURE 3 – Un exemple

Alors vérifiez que la liste d'adjacence est

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

et que la matrice d'adjacence est

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

Si on ne souhaite pas numéroter les sommets, alors on peut leur donner un nom et travailler comme en 1 mais avec à la place des **dictionnaires d'adjacence** : un dictionnaire où chaque clé est un sommet et la valeur correspondante est la liste des sommets auquel il est relié. Voici par exemple un bout de graphe représentant des lignes de train :

```
trains = {"paris": ["lyon", "marseille", "brest"], "lyon": ["paris", "marseille"],
          "marseille": ["lyon", "paris"], "brest": ["paris"]}
```

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

Exercice 1. Écrire des fonctions `liste_vers_matrice(L)` et `matrice_vers_liste(M)`. La première convertit une liste d'adjacence en matrice d'adjacence, et la deuxième fait précisément l'inverse.

Exercice 2. Écrire une fonction `est_non_oriente(M)` prenant en argument une matrice d'adjacence `M` et qui renvoie `True` si cela représente bien un graphe non orienté, et `False` sinon.

III Chemins

Définition 3. Soit $G = (S, A)$ un graphe. Soient $x, y \in S$ deux sommets. Un **chemin** dans G de x à y est la donnée d'une suite de sommets $s_0, \dots, s_n$ tels que :

1. $s_0 = x$,
2. $s_n = y$,
3. $\forall i \in [\![0, n-1]\!]$, $(s_i, s_{i+1}) \in A$.

Autrement dit il s'agit de passer de sommets en sommets en passant à chaque fois sur une arête, partant de x pour arriver à y. Ici le nombre $n-1$ est la **longueur** du chemin ($n-1$ est le nombre d'arêtes parcourues).

Définition 4. Le graphe non orienté G est **connexe** si pour tous sommets $x, y \in S$ avec $x \neq y$, il existe un chemin reliant x à y.

Un graphe non connexe ressemble à plusieurs graphes posés les uns à côté des autres, sans liens entre eux. Sur cet exemple, il n'y a pas de chemin entre 0 et 5 :

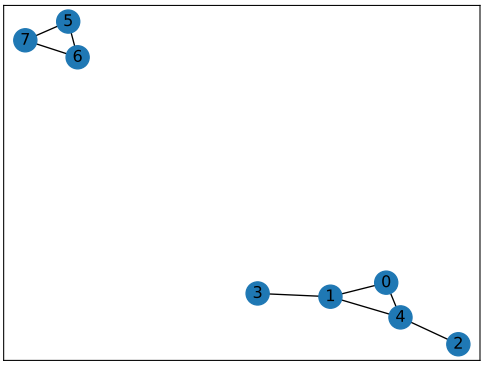

FIGURE 4 – Un graphe non connexe

Exercice 3. 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 `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.
Par exemple dans le graphe d'exemple de la partie précédente, le chemin `[0, 1, 3, 2]` est bien possible, mais pas `[3, 2, 0, 1]`. Le chemin plus long `[0, 1, 3, 0, 1, 0, 2]` est aussi possible.

Exercice 4. Soit G un graphe orienté et soit M sa matrice d'adjacence. Montrer que pour tout $p \in \mathbb{N}$ le coefficient (i, j) de M^p donne le nombre de chemins de longueur exactement p reliant le sommet i au sommet j.

Cela est en fait même vrai pour $p = 0$, où M^p est la matrice identité : on considère que tout sommet est relié à lui-même par un chemin de longeur 0. Cela n'a pas de rapport avec les boucles, qui sont des chemins d'un sommet à lui-même de longueur 1.

IV Annexe : dessiner les graphes en Python

La bibliothèque `networkx` permet de dessiner des graphes. Elle s'utilise conjointement avec `matplotlib` :

```
import matplotlib.pyplot as plt
import networkx as nx
```

On déclare alors une variable `G` de type graphe puis on ajoute des sommets et des arêtes et on demande de dessiner le graphe ; à la fin il faut l'afficher.

```
# G est un graphe orienté ; nx.Graph() pour non-orienté
G = nx.DiGraph()
# ajouter des sommets nommés 0, 1, 2, 3
G.add_nodes_from([0, 1, 2, 3])
# ajouter des arêtes, données par des tuples de sommets
G.add_edges_from([(0, 1), (0, 2), (1, 0), (1, 3), (3, 0), (3, 2)])
# tracer le graphe
nx.draw_networkx(G)
# afficher
plt.show()
```

La bibliothèque contient de nombreuses fonctions pour travailler sur les graphes.
Remarquez d'ailleurs que même la question de dessiner un graphe donné abstraitement n'est pas si évidente : si on place d'abord les sommets n'importe comment, les arêtes vont se croiser... Peut-on éviter les croisements ? C'est compliqué...
Documentation : https://networkx.org/documentation/stable/tutorial.html