Informatique

TP 6

Graphes « simples »

Résumé

Ce TP introduit la notion de graphe simple.

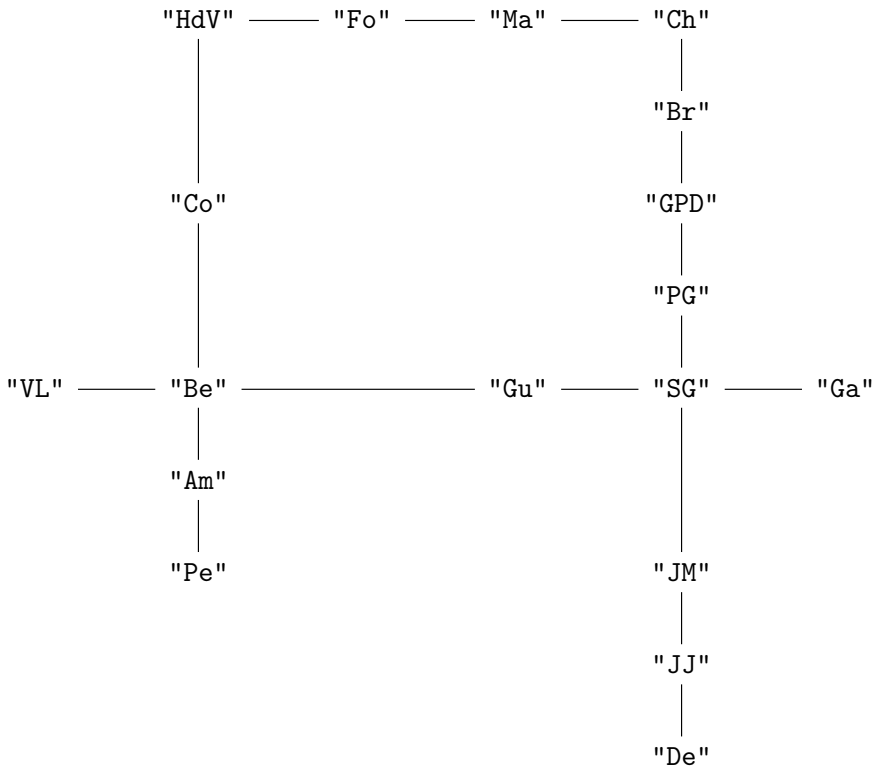
1 Introduction

Un graphe est donné par un ensemble (fini) de **sommets**, deux sommets pouvant être reliés entre eux par une **arête**. [1]

Ce TP concerne uniquement des graphes dits **simples**, dans lesquels les arêtes n'ont pas d'orientation (elles sont symétriques), et ne relient jamais un sommet directement avec lui même.

Exemple 1.1. *Certains plans de réseaux de transports collectifs peuvent être vus comme des graphes. Les sommets de ces graphes sont les stations. Deux sommets sont reliés par une arête s'ils sont des arêts consécutifs d'une même ligne.*

Par exemple, le graphe suivant représente un fragment du réseau de métro d'une grande ville française.



1. On supposera toujours que les sommets sont en nombre fini. Mais il existe aussi des graphes infinis ...

Il existe une énorme quantité de choses qui peuvent être représentées par des graphes. On peut penser par exemple aux réseaux sociaux (les sommets sont les membres et les arêtes les contacts), à des réseaux de transport d'électricité, au réseau Internet etc. [2]

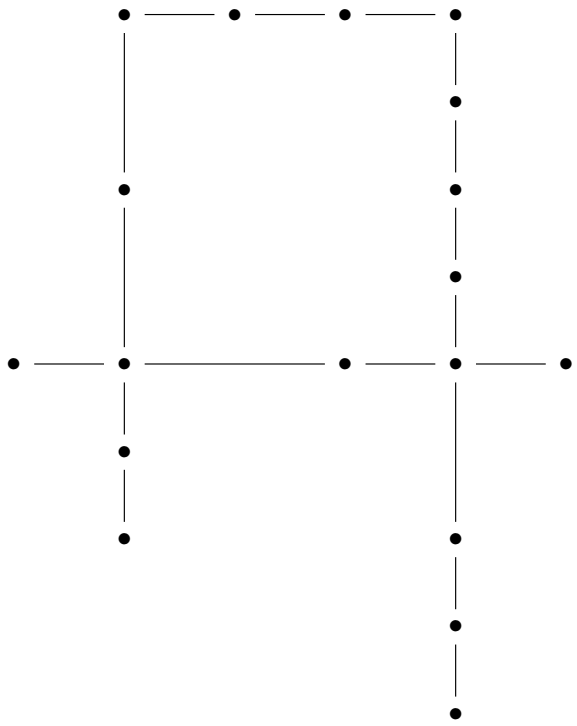
Définition. Un **graphe simple** est une paire $G = (V, E)$, où V est un ensemble fini de **sommets**. E est l'ensemble des **arêtes** de G. Dans un graphe simple, une arête est un ensemble constitué d'exactement deux sommets (distincts). Deux sommets $u, v \in V$ sont reliés par une arête lorsque $\{u, v\} \in E$.

Exemple 1.2. *Le graphe représenté à l'Exemple 1.1 est un graphe simple. L'ensemble de ses sommets est*

$$\{\texttt{"HdV"}, \texttt{"Fo"}, \texttt{"Ma"}, \texttt{"Ch"}, \texttt{"Br"}, \texttt{"Co"}, \texttt{"GPD"}, \texttt{"PG"}, \texttt{"VL"}, \texttt{"Be"}, \texttt{"Gu"}, \texttt{"SG"}, \texttt{"Ga"}, \texttt{"Am"}, \texttt{"Pe"}, \texttt{"JM"}, \texttt{"JJ"}, \texttt{"De"}\}$$

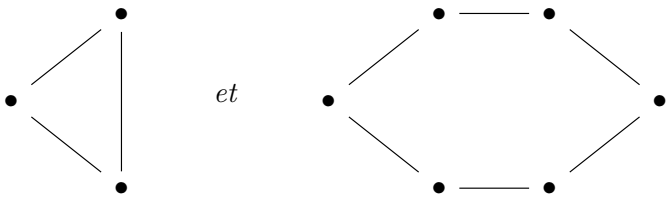
Les arêtes de ce graphe sont dessinées par des traits pleins entre deux sommets. Par exemple, $\{\texttt{"HdV"}, \texttt{"Fo"}\}$ *et* $\{\texttt{"Fo"}, \texttt{"Ma"}\}$ *sont des arêtes de ce graphe.* **Attention :** $\{\texttt{"HdV"}, \texttt{"Ma"}\}$ *n'est pas une arête de ce graphe.*

Exemple 1.3. *Quand on dessine un graphe à la main, on ne donne pas toujours de noms aux sommets. Par exemple, le graphe simple de l'Exemple 1.1 peut être dessiné*



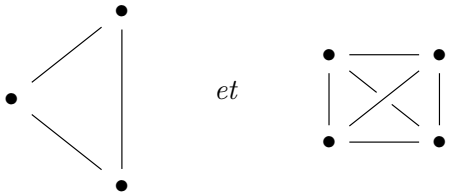
Exemple 1.4. *Voici quelques autres exemples de graphes simples.*

(1) Deux **cycles**

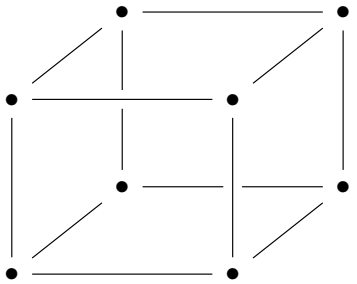


2. Dans un autre registre, les schémas représentant usuellement les constellations sont des graphes.

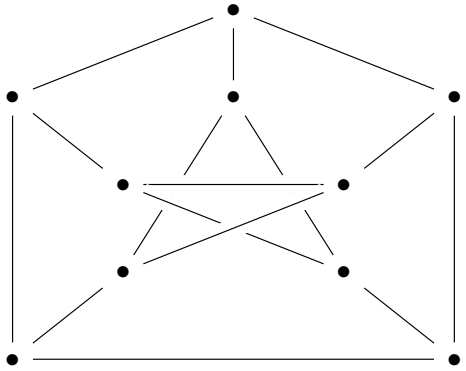
(2) Deux graphes **complets**



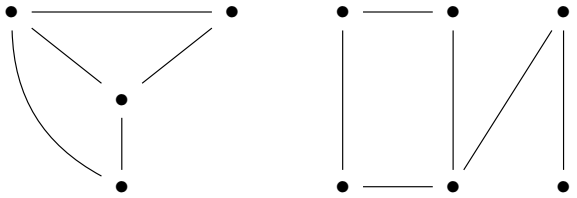
(3) Un cube



(4) Le graphe de **Petersen**



(5) Un graphe **non connexe**



Remarque 1.5. *Soit $G = (V, E)$ un graphe simple.*

(1) Étant donnés $u, v \in V$, on a toujours $\{u, v\} = \{v, u\}$. En particulier, $\{u, v\} \in E$ si, et seulement si, $\{v, u\} \in E$. En d'autres termes, les arêtes des graphes simples sont symétriques. En particulier, les graphes simples sont **non-orientés**.

*(2) Étant donné $u \in V$, l'ensemble $\{u, u\} = \{u\}$ a exactement un élément. Donc $\{u, u\}$ n'est jamais une arête de G. Ainsi, les graphes simples n'ont pas d'***auto-arêtes**.

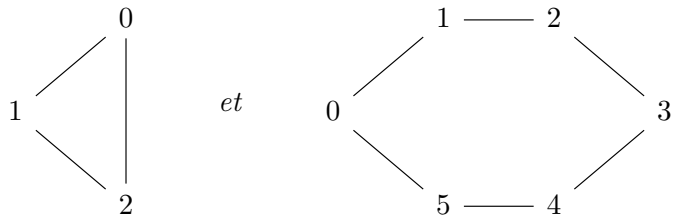
Définition. Soit $G = (V, E)$ un graphe simple. Deux sommets $u, v \in V$ de G sont **adjacents** s'il existe une arête entre u et v, c'est-à-dire si $\{u, v\} \in E$. Si u et v sont adjacents, alors u est un **voisin** de v (et donc v est un voisin u).

2 Représentation des graphes

Nous allons considérer deux représentations en machine des graphes : la représentation par **listes d'adjacence** d'une part, et la représentation par **matrices d'adjacence** d'autre part.

Dans les deux cas, les sommets d'un graphe sont représentés par des entiers. Plus précisément, si $G = (V, E)$ est un graphe (simple) avec $n = |V|$ sommets, alors pour représenter G en machine on supposera que V est de la forme $\{0, \dots, n-1\}$. Cela revient à numéroter les éléments de V avec les entiers $0, \dots, n-1$.

Exemple 2.1. *Voici une numérotation possible des sommets de chacuns des deux graphes de l'Exemple 1.4(1) :*



Question 1. *Pour chacuns des deux graphes de l'Exemple 1.4(1), pouvez-vous donner des numérotations des sommets qui diffèrent de celles de l'Exemple 2.1 ?* Q.1

Question 2. *Pour chacuns des autres graphes $G = (V, E)$ de l'Exemple 1.4, proposer une numérotation de V avec les entiers $0, \dots, |V| - 1$. Dessiner les graphes ainsi obtenus.* Q.2

2.1 Listes d'adjacence

Soit $G = (V, E)$ un graphe simple avec $V = \{0, \dots, n-1\}$. Une **liste d'adjacence** de G est une liste `lst` de longueur n telle que pour tout $i = 0, \dots, n-1$, `lst`$[i]$ liste les sommets de G qui sont adjacents à i.

Exemple 2.2. *Voici des listes d'adjacence pour les deux graphes de l'Exemple 2.1 :*

$$[[1,2]\,,\,[0,2]\,,\,[1,0]] \qquad \textit{et} \qquad [[1,5]\,,\,[0,2]\,,\,[1,3]\,,\,[2,4]\,,\,[3,5]\,,\,[4,0]]$$

Question 3. *Proposer une liste d'adjacence pour chacuns des graphes de la Question 2.* Q.3

2.1.1 Dessins de graphes en Python

La bibliothèque NetworkX permet de dessiner des graphes représentés par listes d'adjacence. [3] Le fichier `tp06utils.py` fournit une fonction `draw_from_adj_lst` qui encapsule les primitives NetworkX permettant de dessiner un graphe simple. [4] Une fois correctement téléchargé, on pourra utiliser les fonctions de `tp06utils.py` dans des scripts avec la ligne suivante.

```
from tp06utils import *
```

Exemple 2.3. *Par exemple, dans une console* Python,

```
>>> triangle = [[1,2], [0,2], [1,0]]
>>> draw_from_adj_lst(triangle)
```

Question 4. *Utiliser la fonction* `draw_from_adj_lst` *pour dessiner chacuns des graphes de la Question 2.* Q.4

3. Voir https://networkx.org/.
4. Le fichier `tp06utils.py` est disponible à http://perso.ens-lyon.fr/colin.riba/teaching/cpes/tp/tp06utils.py.

Remarque. Il serait tout à fait possible de considérer une représentation similaire aux listes d'adjacence, mais utilisant des *dictionnaires* Python.[5] Une telle approche permettrait de manipuler des graphes dont les sommets ont des *noms*, resprésentés sous forme de chaînes de caractères. Nous ne suivrons pas cette approche, mais rien ne vous empêche d'implémenter par vous même, **en plus de ce qui est demandé**, des opérations et algorithmes sur des graphes représentés en utilisant les dictionnaires Python.

2.2 Matrices d'adjacence

Nous utiliserons des matrices de la bibliothèque NumPy, importée par la commande

```
import numpy as np
```

Soit $G = (V, E)$ un graphe simple avec $V = \{0, \dots, n-1\}$. La **matrice d'adjacence** de G est la matrice `M` de dimension $n \times n$ telle que pour tout $i, j \in \{0, \dots, n-1\}$,

$$\texttt{M}[i,j] \quad = \quad \begin{cases} 1 & \text{si } \{i,j\} \in E \\ 0 & \text{sinon} \end{cases}$$

Exemple 2.4. *Voici les matrices d'adjacence pour les deux graphes de l'Exemple 2.1 :*

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \qquad \textit{et} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Remarque 2.5. *Les matrices d'adjacence de graphes simples sont toujours symétriques. De plus, la diagonale ne contient que des* 0.

Question 5. *Donner une matrice d'adjacence pour le graphe de l'Exemple 1.4(3).* Q.5

Question 6. *Écrire une fonction* Python *qui prend en argument une liste d'adjacence et qui renvoie une matrice d'adjacence (pour le même graphe).* Q.6

Question 7. *Donner des matrices d'adjacence pour les graphes (4) et (5) de l'Exemple 1.4.* Q.7

Question 8. *Écrire une fonction* Python *qui prend en argument une matrice d'adjacence et qui renvoie une liste d'adjacence (pour le même graphe).* Q.8

2.3 Quelques opérations élémentaires sur les graphes

Voici quelques opérations élémentaires importantes sur les graphes :
— obtenir la liste des voisins d'un sommet.
— tester si deux sommets sont adjacents.
— ajouter/retirer une arête d'un graphe.

Nous allons comparer les coûts de ces opérations pour les représentations par liste d'adjacence et par matrice d'adjacence.

Question 9. *Pour chacune des deux représentations de graphes, écrire une fonction* Python *qui prend en arguments un graphe G et un sommet v de G, et qui renvoie la liste des voisins de v. Donner le nombre d'opérations en fonction du nombre de sommets et du nombre d'arêtes de G.* Q.9

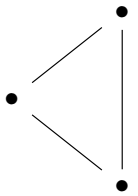
Question 10. *Pour chacune des deux représentations de graphes, écrire une fonction* Python *qui prend en arguments un graphe G et deux sommets u, v de G. La fonction doit renvoyer* `True` *si $\{u, v\}$ est une arête de G, et renvoyer* `False` *sinon. Donner le nombre d'opérations en fonction du nombre de sommets et du nombre d'arêtes de G.* Q.10

Question 11. *Pour chacune des deux représentations de graphes, écrire une fonction* Python *qui prend en arguments un graphe G, deux sommets u, v de G, et qui* **ajoute** *l'arête $\{u, v\}$ à G. Donner le nombre d'opérations en fonction du nombre de sommets et du nombre d'arêtes de G.* Q.11

Question 12. *Pour chacune des deux représentations de graphes, écrire une fonction* Python *qui prend en arguments un graphe G, deux sommets u, v de G, et qui* **retire** *l'arête $\{u, v\}$ de G. Donner le nombre d'opérations en fonction du nombre de sommets et du nombre d'arêtes de G.* Q.12

Définition. La **taille** d'un graphe est la somme de son nombre de sommets et de son nombre d'arêtes (ainsi, la taille de $G = (V, E)$ est $|V| + |E|$).

Exemple 2.6. *La taille du graphe suivant est* 6 *:*



Question 13. *Donner la taille de chacun des deux graphes suivants.* Q.13

Discussion. D'un point de vue purement algorithmique, les listes d'adjacence sont souvent préférées aux matrices d'adjacence. Les raisons principales sont les suivantes.

- Les listes d'adjacence ont une taille linéaire en la taille des graphes, alors que les matrices d'adjacence ont une taille quadratique en le nombre de sommets.
  Ainsi, pour des graphes peu denses, la représentation par listes d'adjacence est plus consise que la représentation par matrices d'adjacence.
- Pour un certain nombre d'algorithmes, l'opération cruciale est d'obtenir la liste des voisins d'un sommet donné. Cette opération est beaucoup plus efficace avec des listes d'adjacence qu'avec des matrices d'adjacence.

Cependant, il y a de bonnes raisons pour savoir utiliser les deux représentations.

- D'une part, les matrices d'adjacence sont parfois plus faciles à manipuler. De plus, la taille des matrices d'adjacence n'est plus un handicap lorsqu'on considère des graphes suffisamment denses.
- D'autre part, dans certains cas concrets, les graphes ne sont pas *explicitement* représentés sous forme de listes ou de matrices d'adjacence, mais sont plutôt obtenus à partir de requêtes sur diverses données (on parle de représentation *implicite*). Les coûts de ces requêtes peuvent imposer un modèle similaire soit aux listes d'adjacence, soit aux matrices d'adjacence.

5. Voir https://docs.python.org/3/tutorial/datastructures.html#dictionaries.

3 Exercices

Question 14. *Écrire une fonction* Python *qui prend en argument une matrice. Cette fonction doit renvoyer* `True` *si son argument est la matrice d'adjacence d'un graphe simple, et renvoyer* `False` *sinon.* Q.14

Question 15. *Écrire une fonction* Python *qui prend en argument une liste de listes d'entiers. Cette fonction doit renvoyer* `True` *si son argument est la liste d'adjacence d'un graphe simple, et renvoyer* `False` *sinon.* Q.15

Graphes complets. Un graphe simple $G = (V, E)$ est **complet** si pour tous sommets distincts $u, v \in V$, il y a une arête entre u et v (c'est-à-dire $\{u, v\} \in E$).

Pour chaque $n \geq 1$, on note K_n le graphe complet de sommets $0, \ldots, n-1$.

Question 16. *Dessiner les graphes* K_3 *et* K_4. Q.16

Question 17. *Donner la taille du graphe* K_n *(pour* $n \geq 1$*).* Q.17

Question 18. *Donner la matrice d'adjacence et la liste d'adjacence de* K_5*. Le dessiner en* Python *(vous pouvez utiliser les outils du §2.1.1).* Q.18

Question 19. *Pour chacune des deux représentations de graphes, écrire une fonction* Python *qui prend argument un entier* `n` ≥ 1 *et qui renvoie le graphe* $\mathsf{K}_{\texttt{n}}$. Q.19

Question 20. *Pour chacune des deux représentations de graphes, écrire une fonction* Python *qui prend argument un graphe. La fonction doit renvoyer* `True` *si ce graphe est complet, et renvoyer* `False` *sinon.* Q.20