Introduction aux graphes avec `Python`

Lycée Richelieu
Rueil Malmaison

juin 2026

L'objectif de cette séance est de répondre aux questions en complétant le script présent dans le même répertoire que ce sujet. Si la question requiert de rédiger en langage naturel, utiliser les commentaires pour répondre.

À la fin, on déposera le script dans ce répertoire en ligne avec le mot de passe *TP_graphes_2026* et on veillera à ce que le nom du fichier permette d'identifier le candidat. Par exemple `script_graphe.py` n'est pas explicite tandis que `script_graphe_pa_fournie.py` l'est.

On rappelle que le script doit pouvoir être exécuté sans erreur de compilation par l'examinateur. Si certaines des fonctions conduisent à des erreurs, il faut commenter les lignes correspondantes.

1 Une première représentation : les listes d'adjacence

1.1 Introduction

Un graphe est un ensemble de *nœuds* (ou *sommets*) et d'*arêtes*. Une arête relie deux nœuds entre eux, ou plus exceptionnellement un nœud avec lui-même.

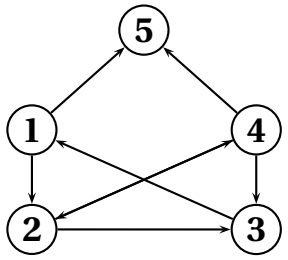
Les arêtes peuvent ou non être orientées, c'est à dire pouvoir être parcourue dans un sens uniquement ou bien dans les deux sens.

Par exemple, un réseau de métro est un graphe. Les nœuds sont les stations et les arêtes sont les lignes entre les stations.

Un graphe non orienté est un cas particulier de graphe orienté pour lequel toutes les arêtes seraient en double, dans les deux sens. Nous allons donc, en toute généralité, nous consacrer à l'étude des graphes orientés.

1.2 L'exemple

Concrètement, nous allons travailler sur le graphe suivant. *Il est conseillé de reproduire ce graphe sur une feuille car il servira dans tout ce travail.*



Ici, le graphe est composé de :

- cinq nœuds numérotés de 1 à 5 ;
- huit arêtes orientées.

1.3 Les listes d'adjacence

1.3.1 Généralités

On peut définir un graphe en spécifiant :

- la liste des sommets (S_i) ;
- pour chaque sommet S_i, les sommets que l'on peut atteindre par les arêtes partant de ce sommet.

La liste des sommets accessibles depuis un sommet fixé s'appelle *liste d'adjacence*.

En reprenant l'exemple ci-dessus, une représentation du graphe pourrait être :

Sommet	Adjacents
1	2, 5
2	3, 4
3	1
4	2, 3, 5
5	

Historiquement, dans Python on a donc codé un graphe comme un dictionnaire dont :

- les clefs sont les sommets de graphes ;
- les valeurs sont les sommets accessibles à partir de la clef.

Par exemple, pour la clef "1", les valeurs seront "2" et "5".

1.3.2 Premières manipulations

- *Algorithme n° 0:* Dans le script à restituer se trouve la déclaration des premiers sommets du graphe. Compléter cette déclaration pour obtenir le graphe présenté en début de document.
- *Algorithme n° 1:* Écrire une commande permettant d'obtenir la liste des nœuds.
- *Algorithme n° 2:* Écrire une commande permettant d'obtenir l'ensemble des nœuds accessibles en deux déplacements à partir du nœud n° 1.

1.4 Fonctions simples

Maintenant que l'on dispose d'un modèle de données pour les graphes, nous allons écrire différentes fonctions.

- *Algorithme n° 3:* Écrire une fonction `ordre` qui :
  - prend en argument un graphe ;
  - renvoie le nombre de sommets du graphe.
- *Algorithme n° 4:* Écrire une fonction `degre` qui :
  - prend en argument un graphe et un sommet ;
  - renvoie le nombre d'arêtes de ce graphe qui pointent vers ce sommet.

1.5 Fonctions de recherche de chemins

Mettons au point une fonction qui détermine tous les chemins permettant de relier un nœud de départ D et un nœud d'arrivée A.

Chaque chemin sera une liste dont :
- le premier élément doit être le départ ;
- le dernier élément doit être l'arrivée.

Lisez attentivement le paragraphe suivant. Il vous permettra de construire convenablement votre code. En particulier, il vous évitera d'entrer dans des boucles infinies !

Le chemin n'est plus valide :
- lorsqu'il comporte une boucle, c'est à dire *si on cherche à ajouter un élément déjà présent dans le chemin* ;
- lorsqu'il arrive sur une impasse, c'est à dire un nœud duquel on ne peut pas s'échapper.

Il faudra donc que la fonction soit en mesure d'éliminer, en cours de route, les chemins non valides.

La fonction validera un chemin si ce dernier aboutit au nœud d'arrivée.

- *Algorithme n° 5:* Faire une fonction `lister_chemins` qui :
  - prend en entrée un graphe, un sommet de départ et un sommet d'arrivée ;
  - renvoie une liste, éventuellement vide, de chemins permettant de relier le départ à l'arrivée, chaque chemin étant codé sous la forme d'une liste contenant les sommets parcourus.

  Voici un petit protocole de test, à partir du graphe pris en exemple :

  a) la commande `lister_chemins(G,1,5)` doit renvoyer `[[1,2,4,5],[1,5]]`.

  b) la commande `lister_chemins(G,2,1)` doit renvoyer `[[2, 3, 1], [2, 4, 3, 1]]`.

  c) la commande `lister_chemins(G,5,2)` doit renvoyer `[]`.
- *Algorithme n° 6:* Déterminer une fonction `plus_court_chemin` qui :
  - prend en entrée un graphe, un sommet de départ et un sommet d'arrivée ;
  - renvoie, s'il existe, le plus court chemin permettant de relier le départ à l'arrivée.

2 Approche matricielle

2.1 Introduction

Pour représenter les arêtes d'un graphe, on peut aussi dresser une matrice M de taille $n \times n$ où n est le nombre de sommets en définissant les coefficients de la manière suivante :

$$m_{ij} = \begin{cases} 1 \text{ s'il existe une arrête du sommet j vers le sommet i} \\ 0 \text{ sinon} \end{cases}$$

On appelle *matrice d'adjacence* cette représentation.

Le graphe pris en exemple dans ce document sera ainsi associé à la matrice d'adjacence :

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

Cette représentation n'est pas très commode pour trouver des chemins. En revanche, elle peut s'avérer très utile pour déterminer les sommets accessibles en un nombre fixé d'étapes.

C'est ce que nous allons maintenant explorer.

2.2 Calcul matriciel

On définit en Python une matrice comme étant une liste de listes. Par exemple :

- l'instruction `A = [[1,2],[3,4]]` correspond à la création de la matrice $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$;
- l'instruction $A[0][1] = -2$ remplace le coefficient en première ligne et seconde colonne par -2.

Pour fabriquer les scripts, il est interdit d'exploiter le module `numpy` ou tout autre module d'algèbre linéaire. En revanche, on pourra vérifier la cohérence des résultats grâce à ces modules.

- *Algorithme n° 7:* Fabriquer ces deux fonctions :
  - `zero` qui prend en argument deux entiers n et p et qui renvoie la matrice nulle de taille $n \times p$;
  - `identite` qui prend en argument un entier n et qui renvoie la matrice identité de taille $n \times n$.
- *Algorithme n° 8:* Fabriquer une fonction `somme` qui prend en argument deux matrices A et B et qui renvoie $A + B$ lorsque la somme est possible.
- *Algorithme n° 9:* Fabriquer une fonction `produit` qui prend en argument deux matrices A et B et qui renvoie $A \times B$ lorsque la multiplication est possible.
- *Algorithme n° 10:* Fabriquer une fonction `puissance` qui prend en argument une matrice A et un entier p et qui renvoie A^p lorsque ce calcul est possible.

2.3 Calcul sur les graphes

Dans le script, on a saisi la matrice M correspondant au graphe étudié dans ce document.

Soient les vecteurs $(E_i)_{1 \leq i \leq 5}$ de $\mathbb{R}^5$ dont toutes les coordonnées sont nulles à l'exception de la i-ème qui vaut 1. Ces vecteurs forment la *base canonique* de $\mathbb{R}^5$.

On a $ME_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$: c'est la première colonne de M.

Les coordonnées strictement positives de ce vecteur correspondent aux sommets accessibles depuis le sommet n° 1 : les sommets n° 2 et 5. En particulier, on a $ME_1 = E_2 + E_5$.

De même, ME_2 et ME_5 sont les vecteurs dont les coordonnées strictement positives correspondent aux sommets accessibles respectivement depuis E_2 et E_5.

Ainsi $M^2E_1 = MME_1 = M(E_2 + E_5)$ sera un vecteur dont les coordonnées strictement positives correspondront aux sommets accessibles *en deux étapes* depuis le sommet n° 1.

On peut généraliser ce qui vient d'être fait :
« Pour tout entier n et pour tout indice j compris entre 1 et 5, la lecture du vecteur $M^n E_j$ nous donnera les sommets accessibles en n étapes depuis le sommet n^o j. »

Algorithme n^o 11: Faire une fonction `sommets_accessibles` qui :

- prend en entrée la matrice M d'un graphe, un numéro j de sommet et un nombre n ;
- renvoie les sommets accessibles en n étapes exactement depuis le sommet n^o j.

La représentation matricielle prendra un intérêt encore plus grand pour un type particulier de graphes appelés *graphes de transition* et utilisés dans le domaine des probabilités. Dans un tel graphe, on associe la probabilité de passer d'un sommet à l'autre à chaque arête du graphe.