Les graphes : présentation des deux modèles de représentation

On présente ici les graphes qui peuvent avoir de nombreuses applications : ils illustrent parfaitement la notion de réseau de sommets et on essaiera de comprendre quelles sont les modélisations possibles de ces graphes dans le langage Python. D'ailleurs, cela nous permettra de mieux appréhender, plus tard, le principe de coloriage, indispensable pour la bonne gestion des parcours dans un graphe.

1 Introduction à la notion de graphe

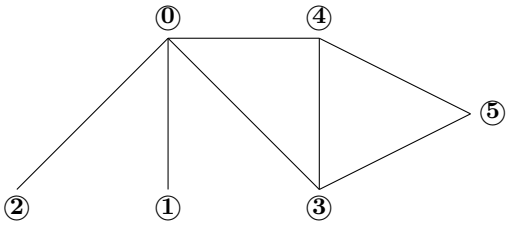
Les **graphes** sont très utiles, car leur compréhension permet de modéliser différentes situtations dans lesquelles il y a différents états possibles et pour lesquelles on peut passer d'un état à un autre.

Bien entendu, ils vous ont permis de décrire vos premières situations probabilistes, mais ils nous permettront aussi de modéliser d'autres problèmes d'évolution que ce soit pour le transport d'informations, la répartition de tâches ou d'actions à effectuer, pour le déplacement physique d'individus, ou plus généralement tous les problèmes nécessitant de représenter des réseaux d'objets reliés les uns aux autres.

Concrètement, on peut définir un **graphe non orienté** comme un couple (S, A) avec :

$$\begin{cases} S \text{ l'ensemble des sommets du graphe} \\ A \text{ une partie de } S \times S, \text{ décrivant l'ensemble des arêtes } (i,j) \in S^2 \text{ reliant deux sommets} \end{cases}$$

Par exemple, on peut considérer le graphe suivant :



Dans un graphe non orienté de sommets $S = [\![0, n]\!]$, on appelle **degré d'un sommet** d_i le nombre d'arêtes sortantes de ce sommet i, et on peut montrer que le nombre d'arêtes A vérifie toujours : $A = \frac{1}{2}\sum_{i=0}^{n} d_i$.

D'ailleurs,

- un graphe peut être **orienté** si on distingue le sens de parcours d'un sommet à un autre. Autrement dit, en notant $S = [\![0, n]\!]$ les sommets d'un graphe, on distingue l'arête (i, j) de l'arête (j, i) pour $i \neq j, (i, j) \in S^2$.
  Dans ce cas, on parle plutôt d'**arcs entre deux sommets** que d'arêtes.
- un graphe peut être **valué** si pour toute arête $(i, j) \in S^2$, on lui associe un poids $\omega_{ij} \in \mathbb{R}$. Ce poids peut alors représenter une probabilité, une distance, un temps ou toute autre mesure associée au problème donné.

Remarque Ici, on travaillera avec des **graphes sans boucle**, c'est à dire qu'on suppose qu'il n'y a pas d'arête permettant de relier le sommet i à lui-même, mais il faut savoir que cela ne décrit pas toutes les situations.

2 Deux modèles de représentation

Pour représenter un graphe dans le langage Python, on pourra procéder de deux façons :

1. à l'aide d'une **liste d'adjacence**, c'est à dire qu'on fait le choix de décrire les relations entre les sommets à l'aide d'une liste ou d'un dictionnaire, pour lesquelles l'indice du sommet renvoie le nom de ses **voisins** : les sommets qui lui sont adjacents. Par exemple, si on reprend le graphe précédent :

```
In :  G=[[1,2,3,4],[0],[0],[0,4,5],[0,3,5],[4,3]]
```
python

ou encore :

```
In :  G={0:[1,2,3,4],1:[0],2:[0],3:[0,4,5],4:[0,3,5],5:[4,3]}
```
python

En fait, le réel avantage du dictionnaire est de pouvoir nommer comme on veut les sommets du graphe, et de faire des appels directs pour aller chercher les sommets voisins.

2. à l'aide d'une **matrice d'adjacence**, c'est à dire que pour un graphe à n sommets, on fait le choix de décrire les relations entre les sommets à l'aide d'une matrice carrée d'ordre $M \in \mathcal{M}_n(\mathbb{R})$ et pour laquelle on convient que :

$$\forall (i,j) \in [\![0,n]\!]^2,\ m_{ij} = \begin{cases} 1, \text{ s'il existe une arête entre les sommets } i \text{ et } j \\ 0, \text{ sinon} \end{cases}$$

Par exemple, si on reprend le graphe précédent :

```
In :  M=array([[0,1,1,1,1,0],[1,0,0,0,0,0],[1,0,0,0,0,0],[1,0,0,0,1,1],[1,0,0,1,0,1],[0,0,0,1,1,0]])
```
python

Evidemment, quand le graphe est non orienté, on aura une matrice symétrique. Par contre, pour les graphes plus complexes, qu'ils soient orientés, valués avec ou sans boucle, **c'est cette représentation qu'on préfèrera car elle pemet d'ajouter le poids des arcs entre deux sommets** en posant :

$$\forall (i,j) \in [\![0,n]\!]^2,\ m_{ij} = \begin{cases} \omega_{ij}, \text{ s'il existe un arc reliant } i \text{ à } j \text{ et de poids } \omega_{ij} \\ 0, \text{ sinon} \end{cases}$$

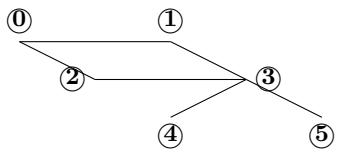
Remarques

1. Il y a quand même une limite à cette représentation, c'est qu'elle nécessite de stocker n^2 informations et peut donc être gourmande en place mémoire.
2. Pour les graphes valués, on peut choisir de mettre le poids à 0 pour exprimer qu'il n'y a pas d'arc entre deux sommets. Pourtant, on choisit souvent une autre convention :
dans le langage Python, on peut utiliser la borne `float('inf')` car elle donne un poids infini, ce qui permet de faire des comparaisons avec les autres poids. En effet,

```
In : a=float('inf'); a>2
                    True
```
python

Exercice 1 (première utilisation de la matrice d'adjacence). [ ]
On considère un graphe $G = (S, A)$ à n sommets, qu'on suppose non orienté, non valué et sans boucle, et on note M_G sa matrice d'adjacence.

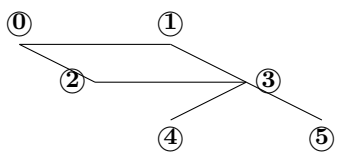
1. Donner, pour le graphe suivant, sa matrice d'adjacence :



2. Dans le langage Python, construire la fonction `voisins(M:array,i:int)->list` qui pour tout graphe de matrice d'adjacence M renvoie la liste des voisins du sommet i. On pourra ajouter une pré-condition pour vérifier si i est bien un sommet possible.
3. Construire les fonctions `aretes(M:array)->int` et `degretotal(M:array)->int` qui pour tout graphe de matrice d'adjacence M renvoie le nombre d'arêtes, et la somme des degrés du graphe.

Exercice 2 (matrice d'adjacence et liste d'adjacence). [ ]
On considère un graphe $G = (S, A)$ à n sommets, qu'on suppose non orienté, non valué et sans boucle, et on note L_G sa liste d'adjacence.

1. Donner, pour le graphe suivant, sa liste d'adjacence :

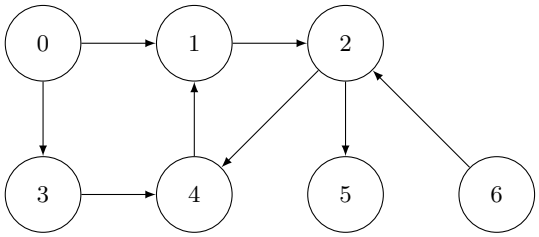


2. Dans le langage Python, construire la fonction de conversion `matricetoliste(M:array)->list` qui pour tout graphe de matrice d'adjacence M renvoie la liste d'adjacence de ce graphe.
3. De la même façon, construire la fonction de conversion `listetomatrice(L:list)->array` qui pour tout graphe de liste d'adjacence L renvoie la matrice d'adjacence de ce graphe.

Exercice 3 (utilisation d'un dictionnaire pour modéliser un graphe orienté). [ ]

Dans cet exercice, les graphes ont leurs sommets numérotés à partir de 0 et ils sont orientés. On les représente par un **dictionnaire d'adjacence**.

Par exemple, le graphe :



est représenté par le dictionnaire qui associe chaque sommet (clef) à ses voisins (valeurs) :

`d` = {0: [1, 3], 1: [2], 2: [4, 5], 3: [4], 4: [1], 5: [], 6: [2]}

Enfin, on appelle **degré d'un sommet** le nombre d'arêtes qui partent depuis ce sommet vers des sommets voisins.

1. Ecrire en langage Python une fonction $degreMax(d : dict) \longrightarrow int$ qui reçoit en entrée un dictionnaire d'adjacence représentant un graphe orienté et renvoie le degré sortant maximal parmi tous les degrés sortants des sommets du graphe.

Si G est un graphe orienté, on appelle **graphe inverse** de G le graphe possédant les mêmes sommets ainsi que les mêmes arêtes mais en sens inverse par rapport à celles de G.

2. Représenter le graphe inverse du graphe orienté donné en introduction. Ecrire alors en langage Python une fonction $grapheInv(d : dict) \longrightarrow dict$ qui renvoie un dictionnaire d'adjacence du graphe inverse du graphe représenté par `d`.

On souhaite colorier notre graphe orienté. Les couleurs sont représentées par des entiers naturels. La coloration du graphe est modélisée par une liste `L` telle que `L[s]` est égale à la couleur attribuée au sommet `s`. Deux sommets du graphe reliés par une arête ne doivent pas être de la même couleur (coloration du graphe valide).

3. Ecrire en langage Python une fonction $colorationValide(d : dict, L : list) \longrightarrow bool$ qui renvoie `True` si la coloration `L` du graphe représenté par `d` est valide et `False` dans le cas contraire.

4. Donner la complexité dans le pire des cas de la fonction précédente en fonction du nombre N de sommets et du nombre M d'arêtes. Justifier votre réponse.