Les graphes

Table des matières

1 Les graphes **2**
- 1.1 Introduction . . . 2
- 1.2 Définition . . . 2

2 Implémentation d'un graphe **4**
- 2.1 Implémentation d'un graphe à l'aide d'une matrice d'adjacence . . . 4
  - 2.1.1 Graphe simple . . . 4
  - 2.1.2 Graphe orienté . . . 5
  - 2.1.3 Graphe pondéré . . . 5
- 2.2 Implémentation avec des listes adjacentes . . . 6
  - 2.2.1 Cas d'un graphe non orienté . . . 6
  - 2.2.2 Cas d'un graphe orienté . . . 7

3 Le type abstrait Graphe orienté **9**

4 Algorithme sur les graphes **9**
- 4.1 Le parcours en largeur d'abord . . . 9
- 4.2 le parcours en profondeur d'abord . . . 10
- 4.3 Recherche de chaînes ou de chemins . . . 11
- 4.4 Recherche de cycles . . . 11

5 Implémentation en python **11**

Ce qu'il faut savoir faire à l'issue du chapitre :

- .
- .

1 Les graphes

1.1 Introduction

Imaginons le mini réseau social suivant :

Cécilia est amie avec Mathieu et Brice

Mathieu est ami avec Cécilia, Brice, Franck et Nathan

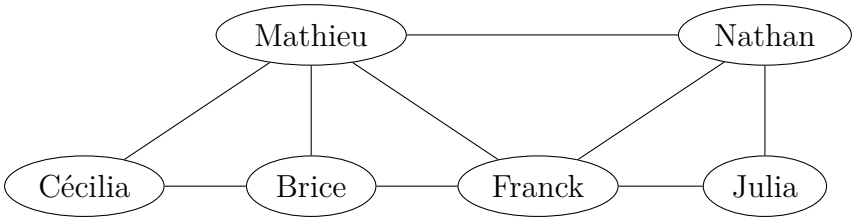
Brice est ami avec Cécilia, Mathieu, Franck

Franck est ami avec Brice, Mathieu, Nathan, Julia

Julia est amie avec Franck et Nathan.

Nathan est ami avec Mathieu, Julia et Franck

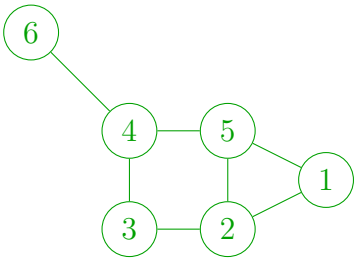
La description de ce réseau social, malgré son faible nombre d'abonnés, est déjà quelque peu rébarbative, alors imaginez cette même description avec un réseau social comportant des millions d'abonnés !
Il existe un moyen plus "visuel" pour représenter ce réseau social : les **graphes** :



1.2 Définition

Définition 10.1

Un **graphe simple** est un graphe composé de sommets et d'arêtes :

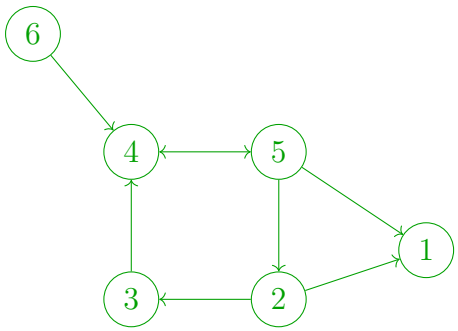


Remarque

Le terme **arête** désigne donc la relation entre deux sommets dans le cas d'un graphe non orienté.

Définition 10.2

Un **graphe orienté** est un graphe avec des flèches appelées **arcs**.

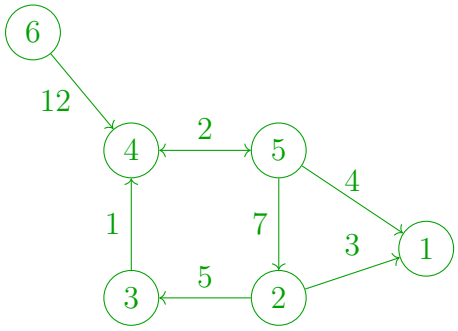


Remarque

Le terme **arc** désigne donc la relation entre deux sommets dans le cas d'un graphe orienté.

Définition 10.3

Dans certains cas, toutes les arêtes (resp les arcs) ne se valent pas, et on peut ainsi attribuer un poids à chaque arêtes (resp chaque arc). On parle de **graphe pondéré** (resp. de **graphe orienté pondéré**) :



Définition 10.4

Une **chaine** est une suite d'arêtes consécutives dans un graphe non orienté, un peu comme si on se promenait sur le graphe. On la désigne par les lettres des sommets qu'elle comporte.

Exemple

Définition 10.5

Un **chemin** est une suite d'arcs consécutifs dans un graphe orienté, un peu comme si on se promenait sur le graphe. On la désigne par les lettres des sommets qu'elle comporte.

Exemple

Définition 10.6

Un **cycle** est une chaine ou un chemin qui commence et se termine au même sommet.

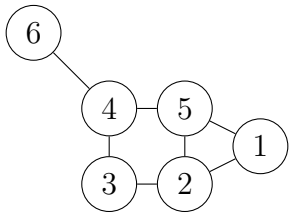
Exemple

2 Implémentation d'un graphe

2.1 Implémentation d'un graphe à l'aide d'une matrice d'adjacence

2.1.1 Graphe simple

Reprenons le graphe simple pour commencer :



La matrice associée au graphe ci-dessus est :

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

Cette matrice est obtenue en remplissant un tableau où chaque ligne correspond au sommet de départ et chaque colonne au sommet d'arrivée :

		A	B	C	D	E	F
		❶	❷	❸	❹	❺	❻
A	❶	0	1	0	0	1	0
B	❷	1	0	1	0	1	0
C	❸	0	1	0	1	0	0
D	❹	0	0	1	0	1	1
E	❺	1	1	0	1	0	0
F	❻	0	0	0	1	0	0

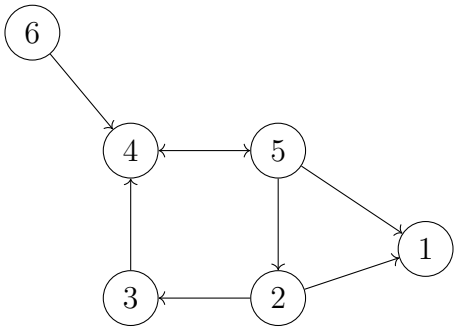
Il faut savoir qu'à chaque ligne correspond un sommet du graphe et qu'à chaque colonne correspond aussi un sommet du graphe. À chaque intersection ligne i-colonne j (ligne i correspond au sommet i et colonne j correspond au sommet j), on place un 1 s'il existe une arête entre le sommet i et le sommet j, et un zéro s'il n'existe pas d'arête entre le sommet i et le sommet j.

La case verte indique qu'il n'y a pas de relation de D vers B.

On remarque ici une symétrie par rapport à une des diagonale ; ceci s'explique par la fait d'avoir un graphe simple.

2.1.2 Graphe orienté

Reprenons le graphe orienté :

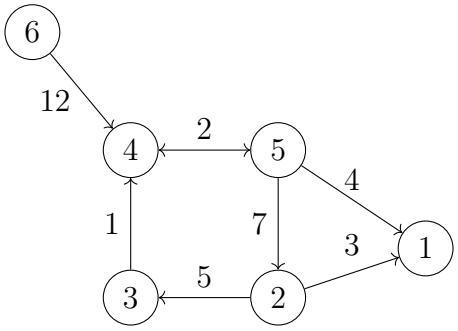


La matrice associée au graphe ci-dessus est :

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

2.1.3 Graphe pondéré

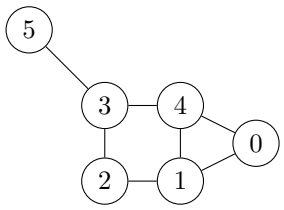
Reprenons le graphe pondéré et orienté :



Il suffit ici de remplacer les "1" par les pondérations respectives.
La matrice associée au graphe ci-dessus est :

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 4 & 7 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 12 & 0 & 0 \end{pmatrix}$$

Exercice 10.1
On considère le graphe suivant :



1. Construire la fonction **matrice_vide(n)** qui prend en argument la taille de la matrice et qui renvoie la matrice carrée n*n avec False pour chaque coefficient

```
>>> matrice_vide(6)
[[False, False, False, False, False, False], [False, False, False, False, False, False
], [False, False, False, False, False, False], [False, False, False, False, False, Fal
se], [False, False, False, False, False, False], [False, False, False, False, False, F
alse]]
```

2. Construire la fonction **ajouter_arc(mat,s1,s2)** de paramètres mat (une matrice), s1 et s2 des sommets. La fonction crée dans la matrice un lien qui matérialise l'arc orienté de s1 vers s2
3. Créer la matrice ci-dessus.
4. Créer une fonction **afficher_arc** de paramètres mat, s1 et s2 et qui renvoie True si il y un arc orienté de s1 vers s2, False sinon
5. Construire une fonction **sommet_voisinage(mat,s)** de paramètre une matrice mat et un sommet s. La fonction renvoie un tableau avec la liste des sommets voisins de s

```
>>> voisinage_sommet(mat,4)
[0, 1, 3]
>>> voisinage_sommet(mat,2)
[1, 3]
```

Exercice 10.2

Implémenter une classe Graphe avec l'interface ci-dessous :

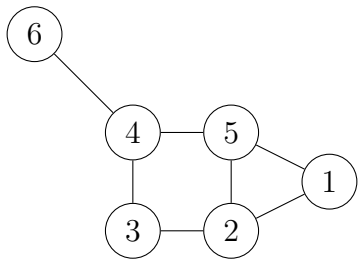
- Un constructeur **___init___(self,n)** avec n la taille de la matrice carrée. Cette classe contiendra deux attributs : n qui sera la taille de la matrice et adj qui sera la matrice de taille n*n , avec chaque coefficient égal à False
- Une méthode **creer_arc(self,s1,s2)** avec s1 et s2 deux sommets. La fonction indique True dans la matrice pour matérialiser le lien de s1 vers s2
- La méthode **afficher_arc(self,s1,s2)** qui renvoie True s'il y a un lien de s1 vers s2, False sinon.
- La méthode **sommet_voisin(self,s)** de paramètre s un sommet, et qui renvoie la liste des sommets voisins à s.

Créer l'objet t qui correspond à l'arbre ci-dessus.
**

2.2 Implémentation avec des listes adjacentes

2.2.1 Cas d'un graphe non orienté

Reprenons le graphe non orienté :

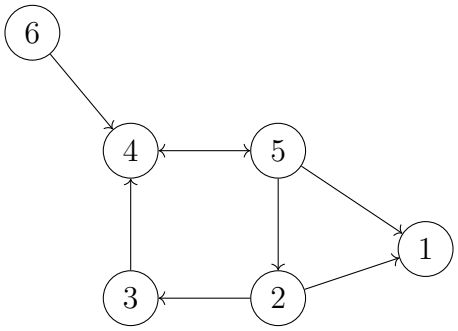


On définit une liste des sommets du graphe. À chaque élément de cette liste, on associe une autre liste qui contient les sommets lié à cet élément :

Sommet	Sommet(s) liés
1	2 ;5
2	1 ;3 ;5
3	2 ;4
4	3 ;5 ;6
5	1 ;2 ;4
6	4

2.2.2 Cas d'un graphe orienté

Il est alors nécessaire ici de faire deux tableaux : un pour les prédécesseurs et un pour les successeurs de chaque sommet :



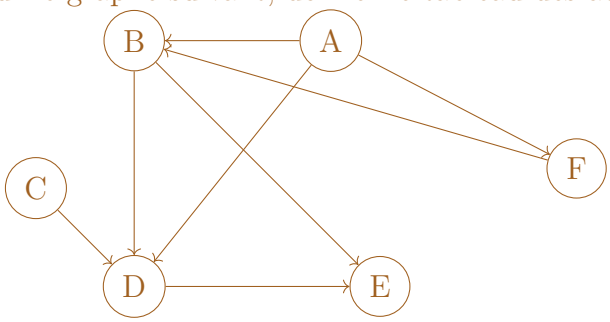
Sommet	Sommet(s) adjacent(s) prédécesseur(s)
1	2 ;5
2	5
3	2
4	3 ;5 ;6
5	4
6	

Sommet	Sommet(s) adjacent(s) successeurs(s)
1	
2	1 ;3
3	4
4	5
5	1 ;2
6	4

Savoir-Faire 10.1

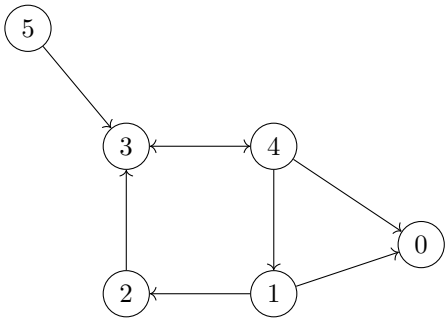
Savoir écrire la liste des adjacences

Pour le graphe suivant, donner le tableau des adjacences.



Exercice 10.3

Considérons le graphe orienté :



On décide de représenter ce graphe par une liste des successeurs.
Réaliser une classe GraphV qui prend comme argument une liste de successeurs. Cette classe comportera une méthode **est_lie(self,i,j)** qui renvoie True s'il y a un lien de i vers j, False sinon. La classe comportera aussi une méthode **graph_vers_matrice(self)** qui renvoie la matrice d'adjacence au graphe.

```
>>> g = GraphV([[],[0,2],[3],[4],[1,0],[3]])
>>> g.est_lie(5,3)
True
>>> g.est_lie(3,5)
False
>>> g.est_lie(3,3)
False
>>> g.graphe_vers_matrice()
[[0, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [1, 1,
0, 0, 0, 0], [0, 0, 0, 1, 0, 0]]
```

***

Remarque

Comment choisir l'implémentation à utiliser (matrice d'adjacence ou liste d'adjacence) ?

le choix se fait en fonction de la densité du graphe, c'est-à-dire du rapport entre le nombre d'arêtes et le nombre de sommets. Pour un graphe dense on utilisera plutôt une matrice d'adjacence.
certains algorithmes travaillent plutôt avec les listes d'adjacences alors que d'autres travaillent plutôt avec les matrices d'adjacences. Le choix doit donc aussi dépendre des algorithmes utilisés.

3 Le type abstrait Graphe orienté

Conventions :

- Pour ajouter un arc, il faut que ses extrémités existent
- Si un sommet est supprimé, alors tous les arcs incidents à ce sommet sont supprimés aussi.

4 Algorithme sur les graphes

Il existe 2 méthodes pour parcourir un graphe :

- Le parcours en largeur d'abord
- le parcours en profondeur d'abord

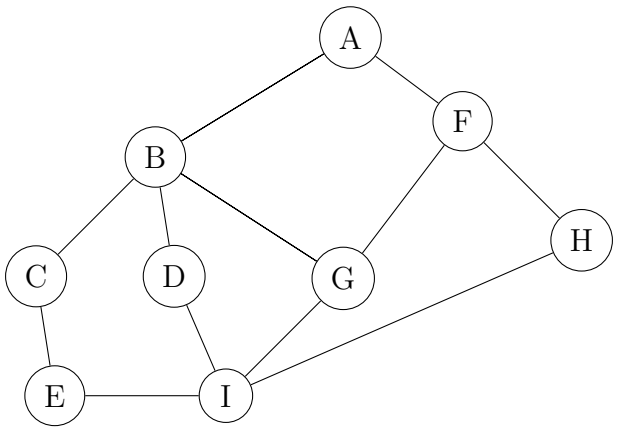
4.1 Le parcours en largeur d'abord

Nous allons travailler sur un graphe G(V,E) avec V l'ensemble des sommets de ce graphe et E l'ensemble des arêtes de ce graphe.

On adoptera un code couleur :

- Sommet de couleur verte si le sommet n'a pas encore été visité
- Sommet de couleur rouge si le sommet a été visité

```
1  VARIABLE
2  G : un graphe
3  s : noeud (origine)
4  u : noeud
5  v : noeud
6  f : file (initialement vide)
7  # On part du principe que pour tout sommet est initialement vert
8  DEBUT
9  Function PARCOURS_ LARGEUR(G,s)
10 |  s.couleur ← rouge
11 |  enfiler (s,f)
12 |  TANT QUE f non vide FAIRE
13 |  |  u ← defiler(f)
14 |  |  pour chaque sommet v adjacent au sommet u :
15 |  |  if v.couleur n'est pas rouge then
16 |  |  |  v.couleur ← rouge
17 |  |  |  enfiler(v,f)
18 |  |  end
19 end
20 FIN
```



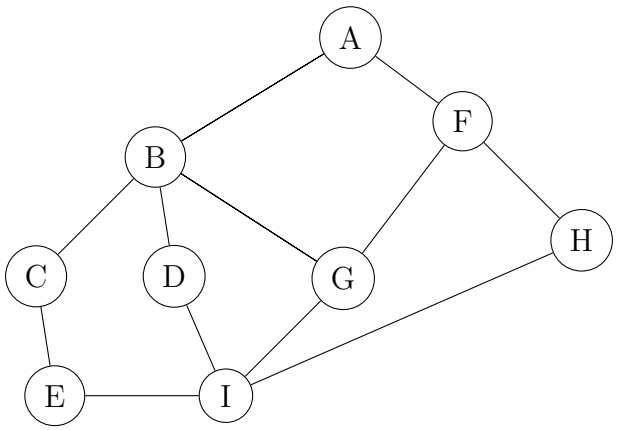
Utiliser l'algorithme du parcours en largeur d'abord pour ce graphe, et donner l'ordre des sommets visités.

Remarque

- On commence à visiter A
- On visite ensuite les sommet à une distance 1 de A : les sommets B et F
- On visite ensuite les sommet à une distance 2 de A : C,D,G et H
- Et on termine par ceux situés à une distance égale à 3 : E et I.

4.2 le parcours en profondeur d'abord

```
1  VARIABLE
2  G : un graphe
3  u : noeud
4  v : noeud
5  f : file (initialement vide)
6  # On part du principe que pour tout sommet est initialement vert
7  DEBUT
8  Function PARCOURS_ PROFONDEUR(G,u)
9  |   u.couleur ← rouge
10 |   for chaque sommet v adjacent à u do
11 |   |   if v couleur n'est pas rouge then
12 |   |   |   PARCOURS_ PROFONDEUR(G,v)
13 |   |   end
14 |   end
15 end
```



Utiliser l'algorithme du parcours en profondeur d'abord pour ce graphe, et donner l'ordre des sommets visités, en sachant qu'il n'y a pas qu'une seule solution.

4.3 Recherche de chaînes ou de chemins

4.4 Recherche de cycles

5 Implémentation en python

Il est possible de travailler avec des listes d'adjacences en Python en utilisant les dictionnaires :

```
#liste d'ajacence pour le graphe cartographie
l = {'1':('2','5), '2':('1', '3', '5'), '3':('2','4'),
 '4':('3', '5','6'), '5':('1', '2', '4'), '6':('4')}
```