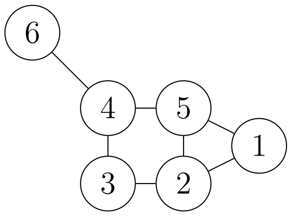
20.2 Implémentation d'un graphe

NSI terminale - JB Duthoit

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

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

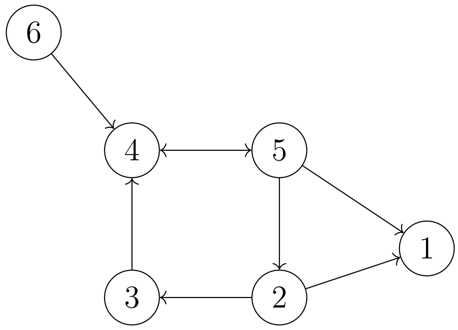
Il faut savoir qu'à chaque ligne correspond un sommet du graphe et qu'à chaque colonne correspond aussi un sommet du graphe. À chaque intersection ligne i-colonne j (ligne i correspond au sommet i et colonne j correspond au sommet j), on place un 1 s'il existe une arête entre le sommet i et le sommet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

Graphe orienté

Reprenons le graphe orienté :

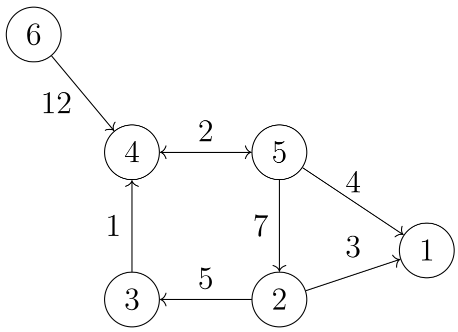


La matrice associée au graphe ci-dessus est :

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

Graphe pondéré

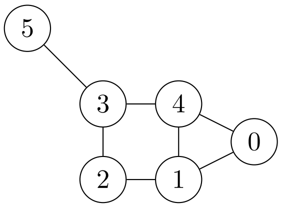
Reprenons le graphe pondéré et orienté :



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 20.190
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 20.191
Implémenter une classe Graphe avec l'interface ci-dessous :

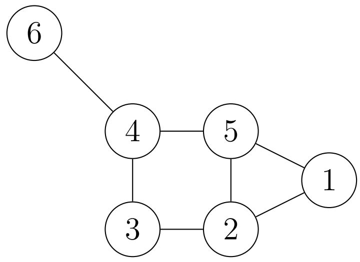
- Un constructeur **___init___(self,n)** avec n la taille de la matrice carrée. Cette classe contiendra deux attributs : n qui sera la taille de la matrice et adj qui sera la matrice de taille n*n , avec chaque coefficient égal à False
- Une méthode **creer_arc(self,s1,s2)** avec s1 et s2 deux sommets. 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.
**

20.2.2 Implémentation avec des listes adjacentes

Cas d'un graphe non orienté

Reprenons le graphe non orienté :

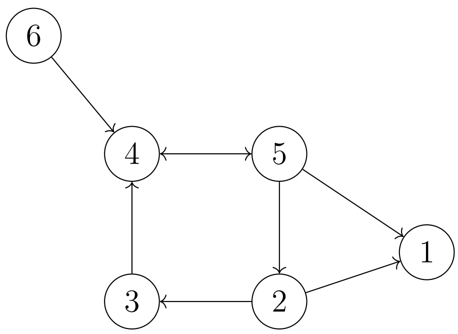


On définit une liste des sommets du graphe. À chaque élément de cette liste, on associe une autre liste qui contient les sommets lié à cet élément :

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

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	

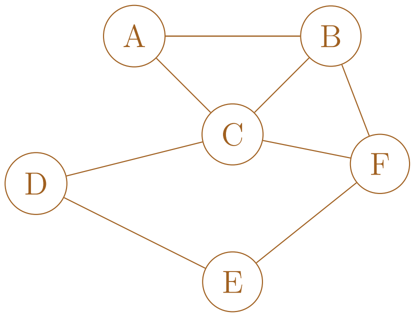
Sommet	Sommet(s) adjacent(s) successeurs(s)
1	
2	1 ;3
3	4
4	5
5	1 ;2
6	4

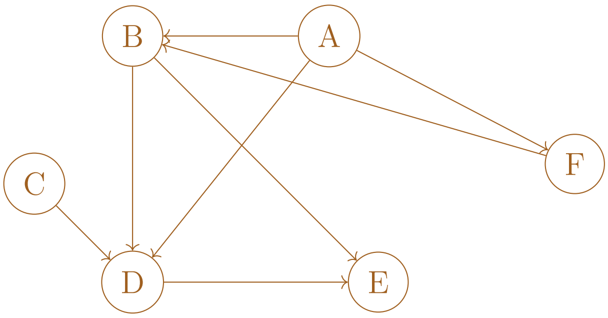
Savoir-Faire 20.13

Savoir écrire la liste des adjacences

Pour le graphe suivant, donner le tableau des adjacences.

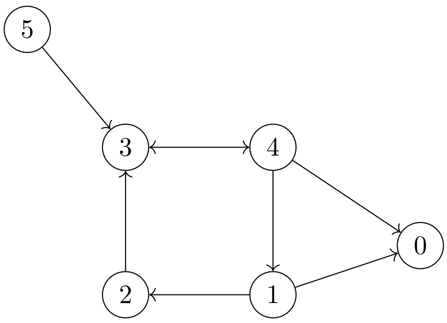
Graphe 1





Exercice 20.192

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.

Exercice 20.193

Créer une classe **GrapheNO(self,n)** où n est le nombre de sommets du graphe. On utiliser une matrice d'adjacence. On implémentera les méthodes **ajouter_arete(self,s1,s2)**, **arete(self,s1,s2)**, **voisins(self,s)**, **sommet(self)** et **retirer_arete(self,s1,s2)**.

Exercice 20.194

Dessiner tous les graphes non orientés ayant 3 sommets

Exercice 20.195

Noé, Isa, Karim, Lola, Véro et Fred doivent suivrent des cours de Maths(M), PC,SVT, NSI et Français (F).
Tous ne doivent pas suivrent tous les cours, car certains ont déjà validé des matières.
Le tableau suivant donne les matières que chaque élèves doit suivre :

Nom	Noé	Isa	Karim	Lola	Véro	Fred
Matières	M-PC-SVT	PC	M-SVT	PC-SVT-NSI	NSI-F	SVT-F

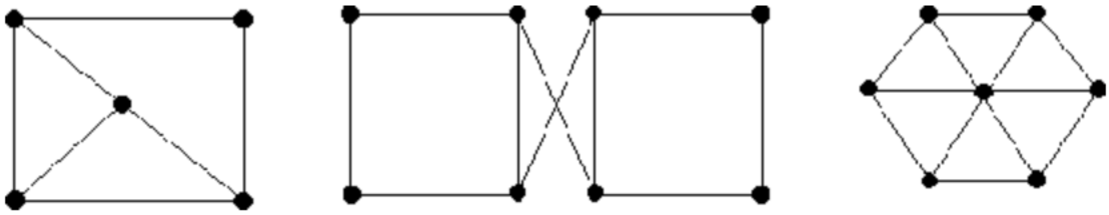
L'objectif est de prévoir un planning optimal pour que chacun puisse suivre les cours. Est-il possible de mettre Maths et PC en même temps par exemple ? (non, car Noé doit suivre M et PC)

1. Construire un graphe non orienté à 5 sommets (M-PP-SVT-NSI-F) en considérant qu'une arête entre deux sommets signifie une incompatibilité . Par exemple, on reliera le sommet M et le sommet PC, car il y a une incompatibilité entre M et PC (pour Noé par exemple, qui doit suivre les deux).
2. On coloriera chaque sommet avec un minimum de couleur et en respectant :
   - Chaque sommet doit avoir une couleur
   - Chaque sommet doit avoir une couleur différente de son voisin.
3. En déduire un planning optimal possible.
   ***

☛ Le problème de la coloration d'un graphe avec un minimum de couleur est un exercice difficile . Il n'existe pas, pour l'instant, d'algorithme efficace pour répondre à cette question.

Exercice 20.196

Cet exercice ne rentre pas directement dans le programme de NSI ; il permet de se familiariser avec la coloration de graphe dans 3 exemples très simples.
Coloration de graphe : chaque sommet doit être dans une couleur différente de ses voisins.
Nombre chromatique : nombre minimal de couleur pour colorier un graphe.
On considère les 3 graphes suivants :



Déterminer dans chacun des cas le nombre chromatique.

***

Exercice 20.197

Comprendre et analyser ces fonctions, qui permettent de colorier un graphe.

```
def coloriage(g):
    """ coloriage d'un graphe avec un algorithme glouton"""
    couleur = {} # dico avec les couleurs numérotées avec des
    ent à partir de zéro
                    # couleur= {2:3,3:4, 4:4} signifie que
  les sommet 1 est colorié en coul 3 et abs
                    # sommet 3 et sommet 4 en coul 4
    n = 0
    for s in g.sommets():
        c = mex(g.voisins(s),couleur)  # c est la couleur
  pour le sommet s en cours
        couleur[s] = c
        n = max(c+1,n) #explications plus tard
    return couleur, n

def mex(voisins,couleur):
    '''voisins est une liste de voisins, couleur un dico
    renvoie la plus petite couleur non utilisée'''
    n = len(voisins)
    dispo = [True] * (n + 1)
    for v in voisins :
        if v in couleur and couleur[v] <= n :
            dispo[couleur[v]] = False
    for c in range(n+1):
        if dispo[c]:
            return c
```