```python
import numpy as np
```

I Notion de graphe

I A Définition

■ Définition 1. *[Graphe à n sommets, sommets, arêtes] Soit $n \in \mathbf{N}^\star$.*

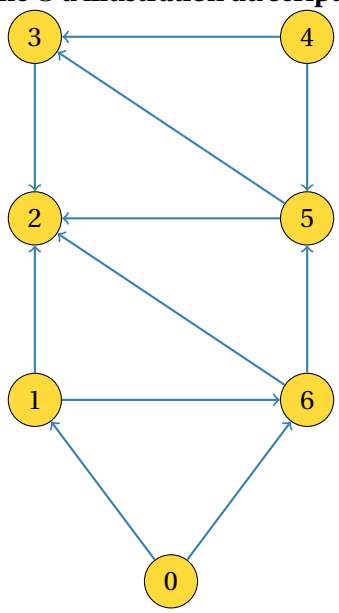
- *Graphe orienté $G = (V, A)$: c'est la donnée :*
  - *d'un ensemble fini d'entiers $V = \{0, 1, \ldots, n-1\}$, appelés **sommets** du graphe,*
  - *et d'un ensemble A de **couples** d'éléments de V. Les éléments de A sont appelés **arêtes** du graphe.*
- *Graphe non orienté : même définition que celle de graphe orienté, à ceci près que l'ensemble A est un ensemble de* paires *d'éléments de V.*

I B Représentation graphique d'un graphe

En général :
- les sommets d'un graphe G sont représentés par des cellules (des pastilles, quoi),
- et les arêtes :
  - dans le cas de graphes orientés : par des flèches allant de i vers j si $(i, j) \in A$.
  - dans le cas de graphes non-orientés : simplement par un segment liant i à j si $\{i, j\} \in A$.

■ Exemple 1. (qui sert dans tout le document).

Graphe G d'illustration du script du II



■ Exercice 1. Donner ci-après les ensembles A et V de ce graphe G après avoir dit si il est orienté ou non.

M. Patel
2024-2025
Classes de B_2, Rennes

■ **Définition 2.** *[Voisin d'un sommet i d'un graphe orienté] Soit $G = (A, V)$ un graphe orienté et $i \in V$. On dit que le sommet $j \in V$ est un voisin de i si (i, j) est une arête de G, c'est-à-dire : $(i, j) \in A$.*

■ **Exercice 2.** Écrire mathématiquement la définition de voisin pour un graphe non orienté :

I C Représentation matricielle d'un graphe

■ **Définition 3.** *[matrice d'adjacence] Soit $G = (A, V)$ un graphe, et notons $V = \{1, \dots, n\}$ ses sommets. On appelle matrice d'adjacence de G la matrice $M \in \mathscr{M}_n(\mathbf{R})$ de coefficient $m_{i,j}$ égal à :*

- *1 si les sommets i et j sont reliés par une arête.*
- *0 sinon.*

■ **Exercice 3.**

1. Quelle propriété remarquable possède la matrice d'adjacence d'un graphe non orienté ?
2. Écrire la matrice d'adjacence du graphe G de l'exemple 1, et la matrice d'adjacence de la version non orientée de ce même graphe.

■ **Exercice 4.** Soit $G = (A, V)$ un graphe orienté de matrice d'adjacence de G la matrice M. En termes de voisins :

1. Que représente la somme des éléments de la ligne i dans M ?
2. Que représente la somme des éléments de la colonne j dans M ?

I D Implémentations possibles sous Python

Comme toujours en informatique, l'implémentation efficace d'un objet repose sur un choix adapté de la structure de données pour le représenter et résoudre le problème souhaité. Sous **Python**, on peut représenter un graphe :

1. par sa martrice d'adjacence.
2. par une simple liste `L`, appelée **Liste d'adjacence** du graphe. Dans ce cas pour $j \in V$, `L[j]` contient la liste (éventuellement vide) des voisins du sommet `j` dans le graphe. La liste `L` est donc de longueur Card V.

■ **Exercice 5.** Définir en Python la liste d'adjacence `L_adj` du graphe de l'exemple 1.

■ **Exercice 6.**

1. Écrire une fonction `L2M(M)` qui prend en entrée une matrice d'adjacence d'un graphe et qui renvoie en sortie sa liste d'adjacence.
2. Écrire une fonction `M2L(L)` qui prend en entrée la liste d'adjacence d'un graphe et qui renvoie en sortie sa matrice d'adjacence.

I E Chemin dans un graphe

■ Définition 4. *[Chemin dans un graphe, cycle d'un graphe, longueur d'un chemin]*

1. *Un chemin dans un graphe G entre deux sommets i et j de ce graphe est une suite finie $(s_0, \dots, s_\ell)$ de sommets du graphe telle que :*
   — $s_0 = i$,
   — $s_\ell = j$,
   — *Pour tout entier k dans $\{1, \dots, \ell\}$, s_k est un voisin de s_{k-1}.*
2. *Avec les notations précédentes, un chemin pour lequel $s_0 = s_\ell$ et $s_k \neq s_0$ sinon s'appelle un cycle du graphe.*
3. *La longueur du chemin $(s_0, \dots, s_\ell)$ est par définition égale à ℓ.*

■ Exemple 2.
— Une arête est un chemin de longueur 1.
— Dans le graphe de l'exemple 1 :
  — $(0, 1, 6, 2)$ est un chemin de longueur 3 allant de 0 à 2.
  — $(2, 5, 3)$ n'est pas un chemin de 2 à 3.

■ Exercice 7. Soit $G = (A, V)$ un graphe de matrice d'adjacence la matrice $M \in \mathscr{M}_n(\mathbf{R})$.

1. Écrire la formule du produit matriciel pour le coefficient $d_{i,j}$ de la matrice M^2 et soit $m_{i,k} \times m_{k,j}$ un terme donné de la somme définissant $d_{i,j}$
2. Soit k un entier fixé Quelles sont les valeurs possibles du produit $m_{i,k} \times m_{k,j}$, et de quoi mesure-t-il l'existence en termes de chemins dans G ?
3. En déduire ce que représente le coefficient $d_{i,j}$ dans M^2 en termes de chemins dans G.
4. De façon générale (pas de preuve attendue) :
   a) Que représente le coefficient en position (i, j) dans M^k ?
   b) Que représente la somme des coefficients diagonaux de M^k ?

■ Exercice 8.

1. Écrire une fonction `nbcycles(M,l)` prenant en entrée une matrice d'adjacence `M` d'un graphe G, un entier `l` et renvoyant en sortie le nombre de cycles de longueur `l` dans G
2. Calculer le nombre de cycles de ℓ ($\ell \in \{2, \dots, 5\}$) dans le graphe G de l'exemple 1.

II Parcours en largeur (algorithme BFS)

— Dans la plupart des situations qui intéressent les programmeurs, les graphes étudiés sont gigantesques, il est hors de question, voire impossible, d'entrer la matrice ou la liste d'adjacence dans la machine.
— Néanmoins, pour ces mêmes situations, le graphe complet ne nous intéresse pas, c'est plutôt l'existence d'un chemin reliant un sommet donné s à un autre qui importe, précisément le parcours effectué pour aller de s à un sommet x d'intérêt.
— L'idée est alors de découvrir le graphe par une exploration systématique (mais pas forcément exhaustive !) à partir de s. L'algorithme de *parcours en largeur*[1] est un type de parcours systématique. Son implémentation est un incontournable de l'informatique, car il donne de façon inattendue une solution optimale à des problèmes courants, c'est pourquoi il est présenté ici.

1. En anglais, l'algorithme de parcours en largeur se dit : Breadth-first-seach algorithm (abrégé BFS).

II A Principe et implémentation

Le principe du parcours en largeur d'un graphe en partant d'un sommet s donné du graphe est le suivant.

Principe de l'algorithme Parcours en largeur depuis un sommet s.

1. On part de s.
2. On découvre tous les voisins de s (déf. 2) : ces voisins sont marqués comme découverts lors du parcours à partir de s, et insérés dans une «file d'attente» `File` l'un après l'autre. Cette file est la file des voisins qu'il reste à explorer. Pour systématiser l'exploration (et donc n'oublier personne !), on traite la file d'attente comme dans la vie : le premier arrivé dans la file est le premier à être examiné.
3. Ensuite, pour chaque voisin v de s qui vient d'être découvert lors du parcours (c'est-à-dire inséré dans la file), on répète les étapes précédentes **1.** et **2.** en prenant à cette étape v au lieu de s.
4. Si on ne découvre plus de voisins à v dans l'étape **2.**, cela signifie que la file est vide : le parcours est terminé. Il a révélé tous les sommets accessibles en largeur depuis s. Ce qui répond à la question de savoir si un sommet est accessible (par ce parcours) depuis s, et surtout, cela donne le chemin le plus court pour y arriver.

Implémentation en Python

```
1  def parcours(start):
2      """
3      Parcours en largeur depuis  start.
4      """
5      File = [(start,[start])] #(depart,
6                               # chemin)
7      vus = []
8      while File: # idem que
9                  # while len(File)>0:
10         s,path = File.pop(0)
11         if s not in vus:
12             vus.append(s)
13             V = Lvoisins(s)
14             for v in V:
15                 if not v in vus:
16                     File.append((v,path+[v]))
17             if is_gagnant(s):
18                 return path
19     return []
```

■ **Remarque 1.** Pour un graphe dont la liste d'adjacence n'est pas connue, le parcours en largeur reste possible : il suffit seulement de disposer d'une fonction capable de calculer les voisins d'un sommet donné du graphe pour chaque sommet découvert.

II B Application

Afin de comprendre cet algorithme, on reprend le graphe de l'exemple 1, et on cherche à voir si le sommet 3 est accessible depuis 0 par ce parcours.

TP 10 : graphes

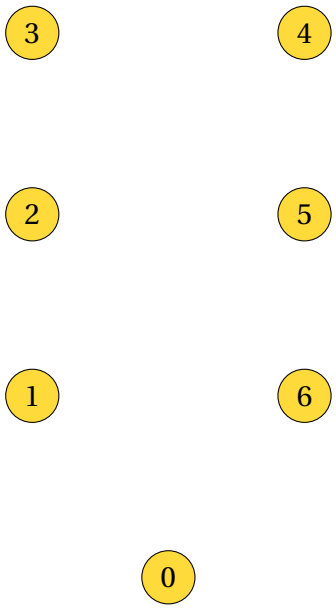
■ **Exercice 9.** Programmer une fonction `Lvoisins(x,L)` qui prend en entrée un entier `x` représentant un sommet d'un graphe de liste d'adjacence `L`, et renvoyant en sortie la liste des voisins de `x` dans ce graphe.

■ **Exercice 10.** Programmer une fonction `is_gagnant(x)` qui prend en entrée un entier `x` représentant un sommet d'un graphe et renvoyant en sortie le booléen `True` si et seulement si le sommet `x` vaut `3`.

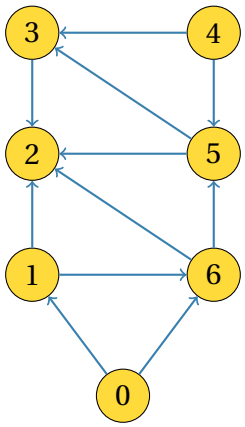
■ **Exercice 11.** Reprendre le graphe de l'exemple 1 ci-contre, avec sa matrice d'adjacence `L_adj` et tester le parcours en largeur depuis le sommet 0 en exécutant le script `exemple-BFS.py`, puis en tapant dans la console :

```
console
>>> parcours(0,L)
```

Remplir le tableau ci-dessous au fur et à mesure du parcours, et matérialiser aussi au fur et à mesure les arêtes découvertes pour comprendre le fonctionnement de cet algorithme (il est bon d'avoir le graphe sous les yeux pour cela, il est reproduit ci-contre):



Sommet v en cours	**État de la file** (↑ : sommet en traitement)	**Voisins découverts de v**
0	[(0, [0])] ↑	



■ **Exercice 12.** Retester la fonction parcours, en considérant cette fois un parcours depuis le sommet `0` et en examinant si le sommet `4` est accessible par un parcours en largeur.

M. Patel
2024-2025
Classes de B_2, Rennes