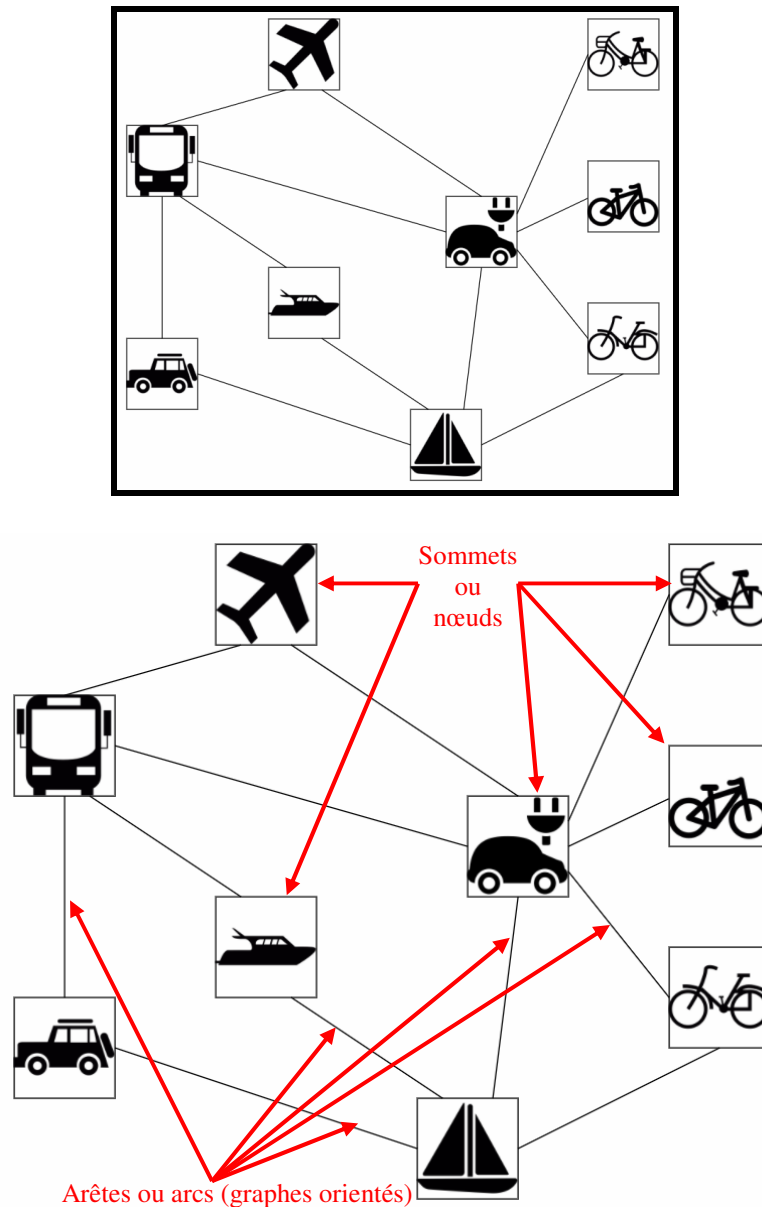


LES GRAPHES

1. LES GRAPHES : DEFINITIONS ET VOCABULAIRE

Un graphe est un objet mathématique (très utilisé, notamment en informatique) constitué de **sommets** ou **nœuds** reliés entre eux par des **arêtes**, **liens** ou **lignes** (pour les graphes non orientés) ou **arcs** ou **flèches** (pour les graphes orientés) :



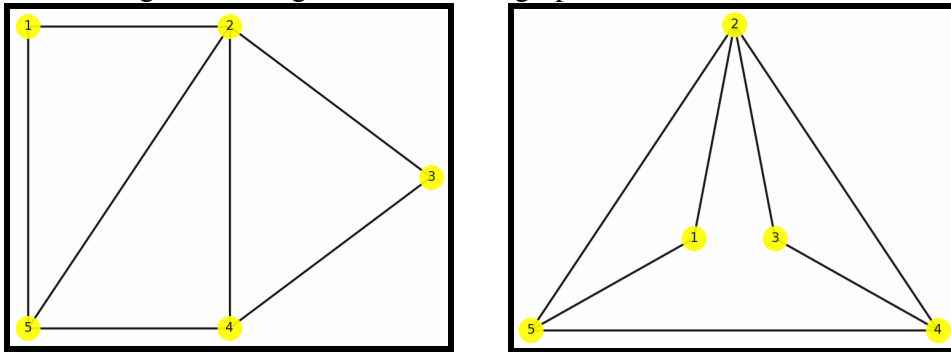
Visualisez la vidéo de présentation des graphes (si besoin, grâce aux paramètres, vous pouvez ralentir la vitesse de lecture) : <https://www.youtube.com/watch?v=YYv2R1cCTa0>

1.1. LES GRAPHES NON ORIENTES

Un **graphe non orienté** est un couple (X, E) où X est un ensemble fini de sommets et E un ensemble de **paires** $\{x, y\}$ de deux éléments de X appelées **arêtes**, **liens** ou **lignes**.

Il est possible de représenter les graphes par leur **diagramme sagittal**. Attention, il ne faut pas confondre un graphe et sa représentation sagittale qui n'est pas unique.

Voici 2 diagrammes sagittaux du même graphe non orienté :



Pour ce graphe nous notons : $G_1 = \{X=\{1,2,3,4,5\}, E=\{\{1,2\}, \{1,5\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{4,5\}\}$

Le nombre de sommets d'un graphe s'appelle l'**ordre du graphe**.

Pour le graphe ci-dessus, donnez l'ordre du graphe :

Le **degré d'un sommet** est le nombre d'arêtes issues de ce sommet.

Pour le graphe ci-dessus, donnez le degré du sommet 1 : **2**

Pour le graphe ci-dessus, donnez le degré du sommet 2 : **5**

Pour le graphe ci-dessus, donnez le degré du sommet 3 : **2**

Pour le graphe ci-dessus, donnez le degré du sommet 4 : **3**

Pour le graphe ci-dessus, donnez le degré du sommet 5 : **4**

Les deux sommets sont **adjacents** ou **voisins** s'il existe une arête qui les relie.

Pour le graphe ci-dessus, dans le tableau suivant mettre une croix pour les sommets adjacents :

Sommets	1	2	3	4	5
1		X			X
2	X		X	X	X
3		X		X	
4		X	X		X
5	X	X		X	

Le **voisinage d'un sommet** est l'ensemble de ses sommets adjacents.

Pour le graphe ci-dessus, indiquez le voisinage du sommet 1 : **2, 5**

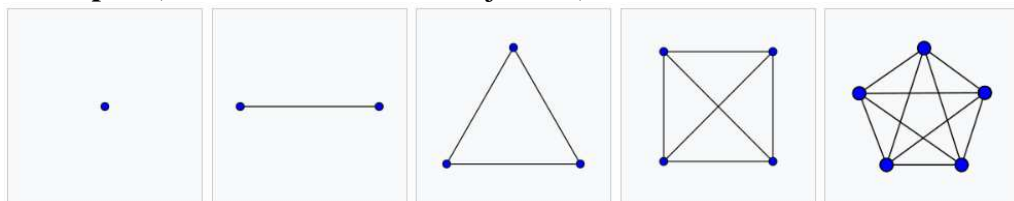
Pour le graphe ci-dessus, indiquez le voisinage du sommet 2 : **1, 3, 4, 5**

Pour le graphe ci-dessus, indiquez le voisinage du sommet 3 : **2, 4**

Pour le graphe ci-dessus, indiquez le voisinage du sommet 4 : **2, 3, 5**

Pour le graphe ci-dessus, indiquez le voisinage du sommet 5 : **1, 2, 4**

Un graphe dans lequel deux sommets quelconques sont reliés par au moins une arête est un **graphe complet** (tous les sommets sont adjacents) :



Indiquez si le graphe ci-dessus (haut de la page 2) est complet, sinon indiquez pourquoi : **non car tout les sommet ne sont pas reliev entre eux**

Le **chemin** est une **chaîne**, qui correspond à une suite d'arêtes consécutives dans un graphe, on la désigne par la liste des sommets qu'elle comporte. La **longueur d'une chaîne** est égale au nombre d'arêtes qui la composent. La **distance entre deux sommets** d'un graphe est la longueur du chemin le plus court. Une chaîne dont l'extrémité et l'arrivée sont les mêmes est une **chaîne fermée**. Une chaîne dont toutes les arêtes sont distinctes est une **chaîne simple**. Une chaîne dont tous les sommets (sauf les extrémités pour les chaînes fermées) sont distincts est une **chaîne élémentaire**.

Pour le graphe ci-dessus (haut de la page 2), indiquez toutes les chaînes simples possibles permettant relier le sommet 1 au sommet 4 (de 1 vers 4) et précisez leurs longueurs :

- 1 - 2 - 4
- 1 - 2 - 5 - 4
- 1 - 2 - 3 - 4
- 1 - 5 - 2 - 4
- 1 - 5 - 2 - 3 - 4
- 1 - 5 - 4

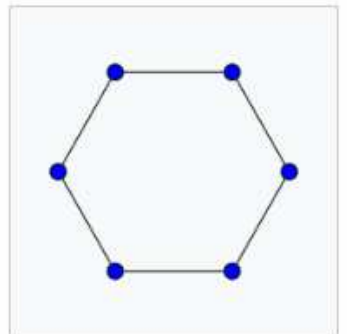
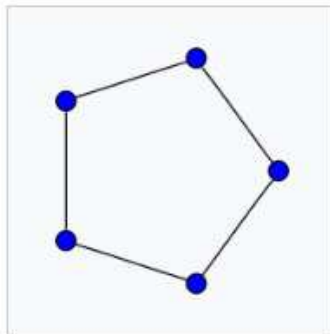
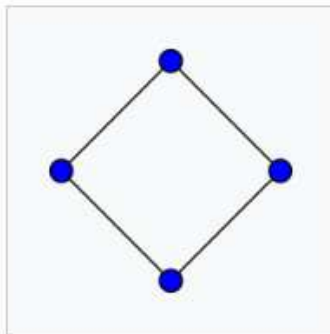
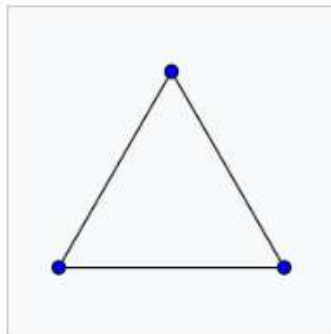
La distance entre les sommets 1 et 4 est : **3**

Un **cycle** est une chaîne simple qui commence et se termine au même sommet. C'est donc une chaîne fermée avec des arêtes distinctes.

Pour le graphe ci-dessus (haut de la page 2), indiquez tous les cycles possibles en partant du sommet 1 et précisez leurs longueurs:

- 1 - 2 - 3 - 4 - 5 - 1
- 1 - 2 - 5 - 1
- 1 - 2 - 4 - 5 - 1
- 1 - 5 - 2 - 1
- 1 - 5 - 4 - 2 - 1
- 1 - 5 - 4 - 3 - 2 - 1

Un **graphe** est **cyclique** si un sommet quelconque est uniquement voisin avec son sommet précédent et son sommet suivant :

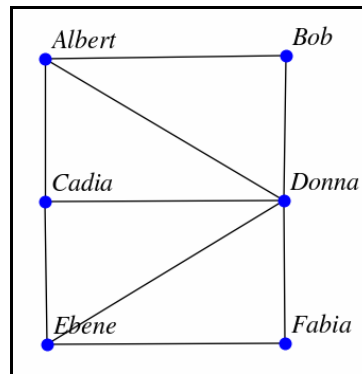


Exercice 1 :

Soit le réseau social suivant :

- Albert est ami avec Bob, Cadia et Donna.
- Bob est ami avec Albert et Donna.
- Cadia est amie avec Albert, Ebène et Donna.
- Donna est amie avec tous les autres abonnés.
- Ebène est amie avec Cadia, Donna et Fabia.
- Fabia est amie avec Ebène et Donna.

La description de ce réseau social, malgré son faible nombre d'abonnés, est déjà quelque peu rébarbative, nous le représentons donc par le graphe suivant :



A l'aide de ce graphe, complétez à l'aide de croix, le tableau des liens suivant :

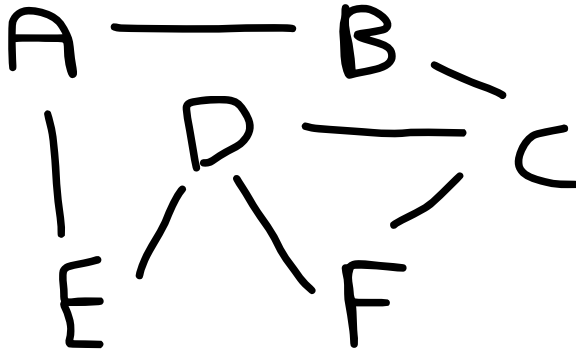
Liens	Albert	Bob	Cadia	Donna	Ebène	Fabia
Albert		x	x	x		
Bob	x			x		
Cadia	x			x	x	
Donna	x	x	x		x	x
Ebène			x	x		x
Fabia				x	x	

Exercice 2 :

Complétez le tableau des liens et construisez un graphe de réseau social à partir des informations suivantes:

- Ariel est amie avec Britanie et Evahn
- Britanie est amie avec Ariel et Cacilie
- Cacilie est amie avec Britanie, Fyriel et Dzenan
- Dzenan est ami avec Cacilie, Fyriel et Evahn
- Evahn est ami avec Ariel, Dzenan et Fyriel
- Fyriel est amie avec Cacilie, Dzenan et Evahn

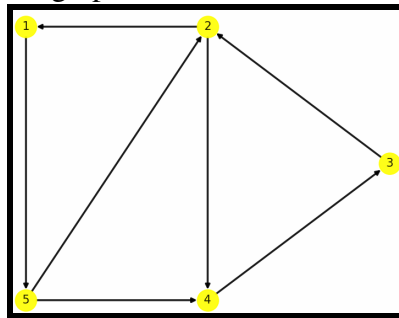
Liens	Ariel	Britanie	Cacilie	Dzenan	Evahn	Fyriel
Ariel		x			x	
Britanie	x		x			
Cacilie		x		x		x
Dzenan			x		x	x
Evahn	x			x		x
Fyriel			x	x	x	



1.2. LES GRAPHES ORIENTES

Un **graphe orienté** est un couple (X, E) où X est un ensemble fini de sommets et E un ensemble de **couples** (x, y) de deux éléments de X appelées **arcs** ou **flèches**.

Voici le diagramme sagittal d'un graphe orienté :



Pour ce graphe nous notons : $G_2 = \{X = \{1, 2, 3, 4, 5\}, E = \{(1, 5), (2, 1), (2, 4), (3, 2), (4, 3), (5, 2), (5, 4)\}\}$

Les deux sommets x et y sont **adjacents** s'il existe un arc (x, y) qui les relie. x est l'origine de l'arc et y son extrémité. x est un **prédécesseur** de y et y un **successeur** de x .

Pour le graphe ci-dessus, dans le tableau suivant mettre une croix pour les sommets adjacents :

Sommets successeurs		1	2	3	4	5
Sommets prédécesseurs	1					
	2					
	3					
	4					
	5					

Le **voisinage sortant d'un sommet** est l'ensemble de ses sommets successeurs.

Pour le graphe ci-dessus, indiquez le voisinage sortant du sommet 1 : 5
 Pour le graphe ci-dessus, indiquez le voisinage sortant du sommet 2 : 1-4
 Pour le graphe ci-dessus, indiquez le voisinage sortant du sommet 3 : 2
 Pour le graphe ci-dessus, indiquez le voisinage sortant du sommet 4 : 3
 Pour le graphe ci-dessus, indiquez le voisinage sortant du sommet 5 : 2-4

Le **degré sortant d'un sommet** est le nombre de successeurs.

Pour le graphe ci-dessus, indiquez le degré sortant du sommet 1 : 1

Pour le graphe ci-dessus, indiquez le degré sortant du sommet 2 : 2

Pour le graphe ci-dessus, indiquez le degré sortant du sommet 3 : 1

Pour le graphe ci-dessus, indiquez le degré sortant du sommet 4 : 1

Pour le graphe ci-dessus, indiquez le degré sortant du sommet 5 : 2

Le **voisinage entrant d'un sommet** est l'ensemble de ses sommets prédécesseurs.

Pour le graphe ci-dessus, indiquez le voisinage entrant du sommet 1 : 2

Pour le graphe ci-dessus, indiquez le voisinage entrant du sommet 2 : 3 - 5

Pour le graphe ci-dessus, indiquez le voisinage entrant du sommet 3 : 4

Pour le graphe ci-dessus, indiquez le voisinage entrant du sommet 4 : 2 - 5

Pour le graphe ci-dessus, indiquez le voisinage entrant du sommet 5 : 1

Le **degré entrant d'un sommet** est le nombre de prédécesseurs.

Pour le graphe ci-dessus, indiquez le degré entrant du sommet 1 : 1

Pour le graphe ci-dessus, indiquez le degré entrant du sommet 2 : 2

Pour le graphe ci-dessus, indiquez le degré entrant du sommet 3 : 1

Pour le graphe ci-dessus, indiquez le degré entrant du sommet 4 : 2

Pour le graphe ci-dessus, indiquez le degré entrant du sommet 5 : 1

Un **chemin** est une suite d'arcs consécutifs dans un graphe orienté, on le désigne par la liste des sommets qu'il comporte. La **longueur d'un chemin** est égale au nombre d'arcs qui le composent. La **distance entre deux sommets** d'un graphe orientés est la longueur du chemin le plus court. Un chemin dont l'extrémité et l'arrivée sont les mêmes est un **chemin fermée**. Un chemin fermé dont tous les arcs sont distincts est un **circuit**.

Pour le graphe ci-dessus (milieu de la page 6), indiquez 3 chemins possibles permettant relier le sommet 1 au sommet 4 (de 1 vers 4) et précisez leurs longueurs :

-
-
-

La distance entre les sommets 1 et 4 (de 1 vers 4) est :

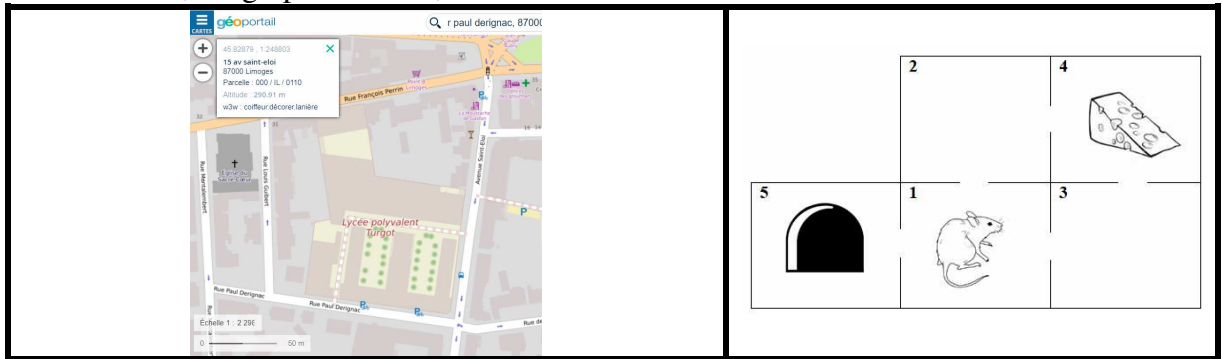
Pour le graphe ci-dessus (milieu de la page 6), indiquez 7 chemins fermées en partant du sommet 2 et précisez leurs longueurs:

-
-
-
-
-
-
-

Pour le graphe ci-dessus (milieu de la page 6), indiquez tous les circuits possibles en partant du sommet 1 et précisez leurs longueurs:

-
-
-

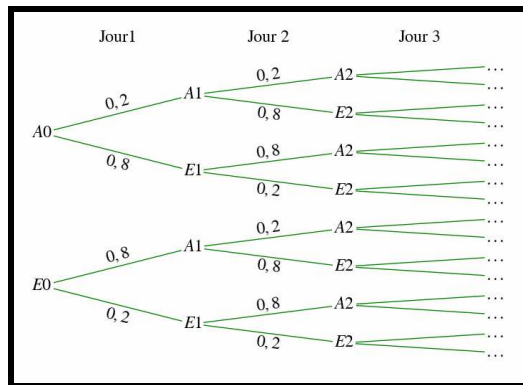
Les graphes orientés sont, entre autres, utilisés pour représenter les réseaux routiers avec sens de circulation, les graphes d'états,...



Exercice 1 :

Chaque matin, l'allumeur de lampadaire change l'état d'un lampadaire avec une probabilité de 0,8. Le lampadaire a deux états possibles : allumé (noté A) ou éteint (noté E).

Cette situation peut être modélisé par l'arbre pondéré suivant :

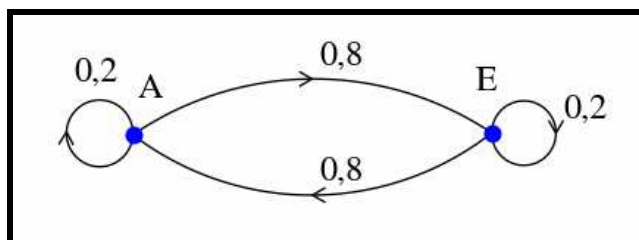


Un graphe probabiliste est un graphe orienté pondéré dans lequel la somme des poids des arêtes (lignes entre deux points) issues de chaque sommet (points) est égale à 1.

Les graphes probabilistes sont utilisés pour modéliser l'évolution d'un système pouvant changer aléatoirement d'état :

- les sommets du graphe sont les états possibles du système ;
- le poids d'une arête orientée issue du sommet i et d'extrémité j est la probabilité conditionnelle de la réalisation de l'évènement j à l'étape $n+1$ sachant que l'évènement i est réalisé à l'étape n .

L'exercice de l'allumeur de lampadaire peut également être modélisé par le graphe probabiliste à 2 sommets suivant. On peut remarquer que la somme des pondérations des arêtes partant de chaque sommet est égale à 1.



En vous aidant du graphe ci-dessus, indiquez les probabilités conditionnelles dans les cases du tableau suivant :

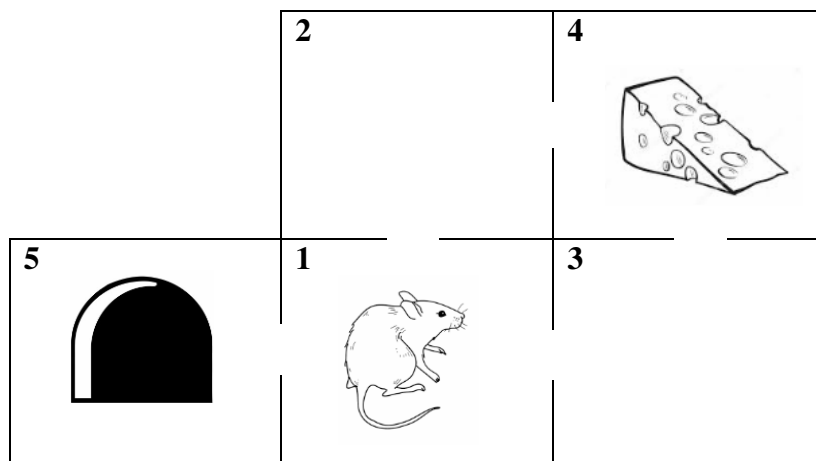
Etats actuels \ Etats futurs	Allumé	Eteint
Allumé		
Eteint		

Exercice 2 :

Une chaîne de Markov (Andreï Andreïevitch Markov, 14 juin 1856 - 20 juillet 1922 est un mathématicien russe.)

Une chaîne de Markov est un processus aléatoire portant sur un nombre fini d'états, avec des probabilités de transition sans mémoire. L'état change au cours du temps avec une probabilité définie au préalable et le nouvel état ne dépend que de l'état présent.

La souris dans le labyrinthe : Une souris se déplace dans le labyrinthe de la figure ci-dessous. Initialement, elle se trouve dans la case 1. A chaque minute, elle change de case en choisissant, de manière équiprobable, l'une des cases adjacentes. Dès qu'elle atteint soit la nourriture (case 4), soit sa tanière (case 5), elle y reste.



- Tracez l'arbre pondéré correspondant à cette situation.
- Au bout de combien de temps minimum la souris atteint-elle sa tanière ? Avec quel probabilité ?
- Au bout de combien de temps minimum la souris atteint-elle sa nourriture ? Avec quel probabilité ?
- Tracez le graphe probabiliste correspondant à cette situation.

2. REPRESENTATION DES GRAPHS

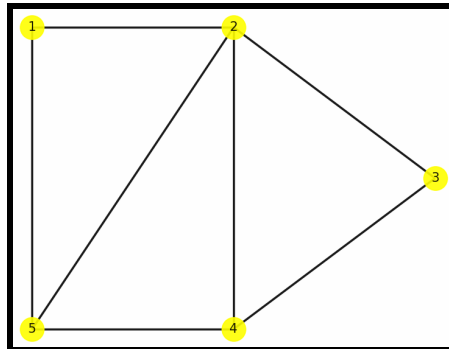
Il existe plusieurs façons de représenter un graphe en informatique :

- Les listes d'adjacences ou listes de successeurs / de prédécesseurs,
- Les matrices d'adjacences

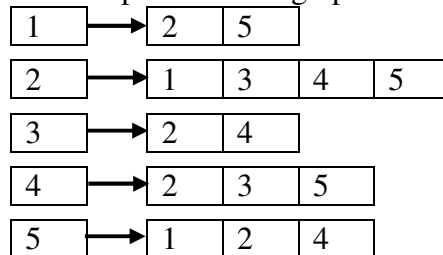
2.1. REPRESENTATION DES GRAPHS PAR LISTES D'ADJACENCES, LISTES DE SUCESSEURS / DE PREDECESSEURS

Pour les graphes non orientés, on définit la liste des sommets du graphe et à chaque élément de cette liste, on associe une autre liste qui contient les sommets liés à cet élément.

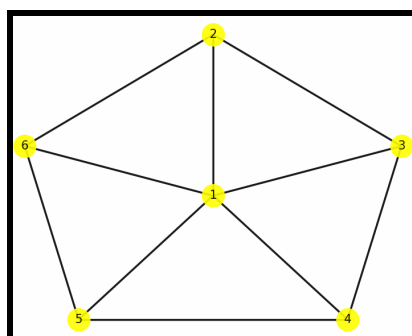
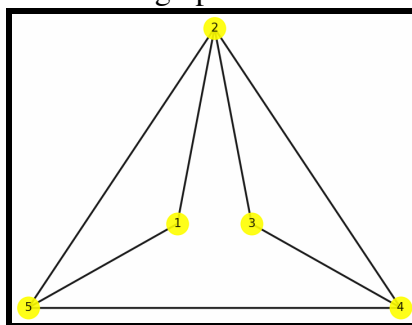
Soit le graphe non orienté suivant :



La **liste d'adjacence** correspondant à ce graphe non orienté est :



Donnez les listes d'adjacences des graphes non orientés suivants :

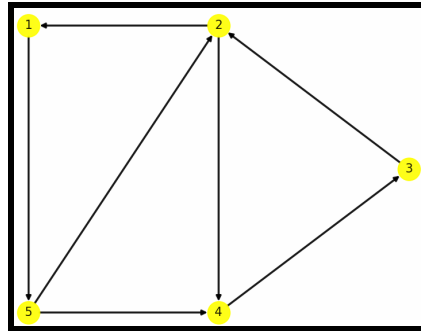


Pour les graphes orientés, il est nécessaire de définir 2 listes :

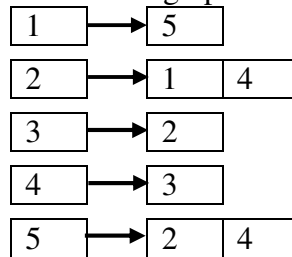
- la liste des successeurs et
- la liste des prédécesseurs.

Si un arc va du sommet A vers le sommet B (flèche de A vers B). On dit que B est un successeur de A et que A est un prédécesseur de B.

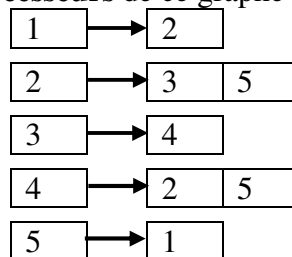
Soit le graphe orienté suivant :



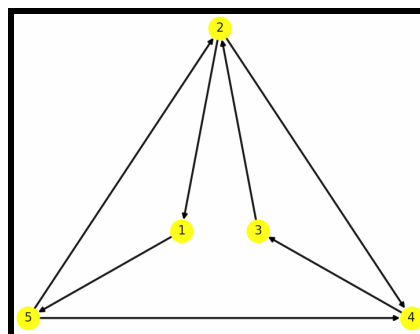
La liste des successeurs de ce graphe orienté est :

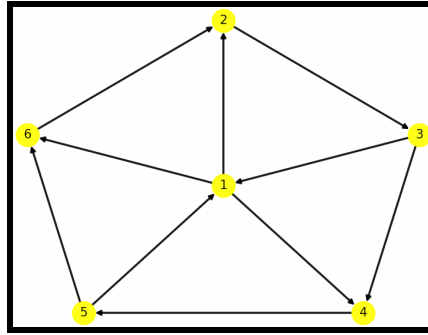


La liste des prédécesseurs de ce graphe orienté est :



Donnez les listes des successeurs et les listes des prédécesseurs des graphes orientés suivants :





2.2. REPRESENTATION DES GRAPHES PAR MATRICES D'ADJACENCES

Une matrice est un tableau à double entrée.

On appelle **matrice** A de **dimension** $n \times p$ un tableau de nombres comportant n lignes et p colonnes. Ces nombres sont appelés **coefficients** (ou éléments) de la matrice.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix}$$

Ligne 1
Ligne 2
Ligne n

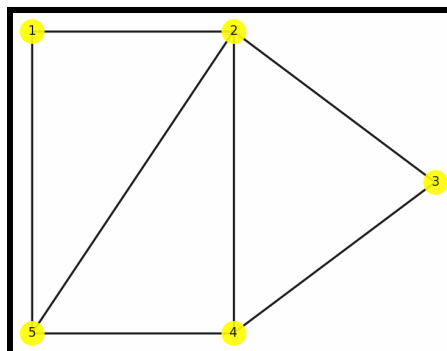
Colonne 1 Colonne 2 Colonne p

Exemple : $A = \begin{pmatrix} 1 & -2 & 5 \\ 0 & 3 & -7 \end{pmatrix}$ est une matrice de dimension 2×3 comportant 2 **lignes** et 3 **colonnes**. Les coefficients sont : $a_{11}=1$, $a_{12}=-2$, $a_{13}=5$, $a_{21}=0$, $a_{22}=3$ et $a_{23}=-7$.

On appelle **matrice carrée** une matrice qui comporte le **même nombre de lignes et de colonnes**.

Les **matrices d'adjacences** sont des **matrices carrées**.

Soit le graphe non orienté suivant :



La matrice d'adjacence de ce graphe non orientés est construite ainsi : A chaque ligne correspond un sommet du graphe et à chaque colonne correspond également un sommet du graphe. A chaque intersection ligne - colonne, on place un 1 s'il existe une arête entre les 2 sommets et un zéro s'il n'en existe pas.

$$\begin{array}{c}
 \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
 \begin{pmatrix}
 1 & 2 & 3 & 4 & 5 \\
 \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} &
 \begin{array}{c} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{array} &
 \begin{array}{c} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{array} &
 \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{array} &
 \begin{array}{c} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{array}
 \end{pmatrix}
 \end{array}$$

Il n'existe pas d'arête entre le sommet 3 et 5 donc on met un 0 à l'intersection de la ligne 3 et la colonne 5 (on met également un 0 à l'intersection de la ligne 5 et la colonne 3).

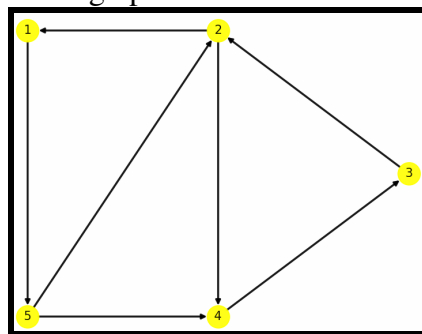
Il existe une arête entre le sommet 1 et 2 donc on met un 1 à l'intersection de la ligne 1 et la colonne 2 (on met également un 1 à l'intersection de la ligne 2 et la colonne 1).

La matrice d'adjacence à un axe de symétrie : la diagonale allant de l'intersection des sommets 1-1 à l'intersection des sommets 5-5 (droite en pointillés sur le schéma ci-dessus).

La matrice d'adjacence de ce graphe non orientés est donc :

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

Il est également possible d'établir une matrice d'adjacence pour un graphe orienté. Le principe reste le même. Soit le graphe orienté suivant :



$$\begin{array}{c}
 \text{vers} \\
 \text{de}
 \end{array}
 \begin{array}{c}
 1 \\ 2 \\ 3 \\ 4 \\ 5
 \end{array}
 \begin{pmatrix}
 1 & 2 & 3 & 4 & 5 \\
 \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} &
 \begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{array} &
 \begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} &
 \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} &
 \begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}
 \end{pmatrix}$$

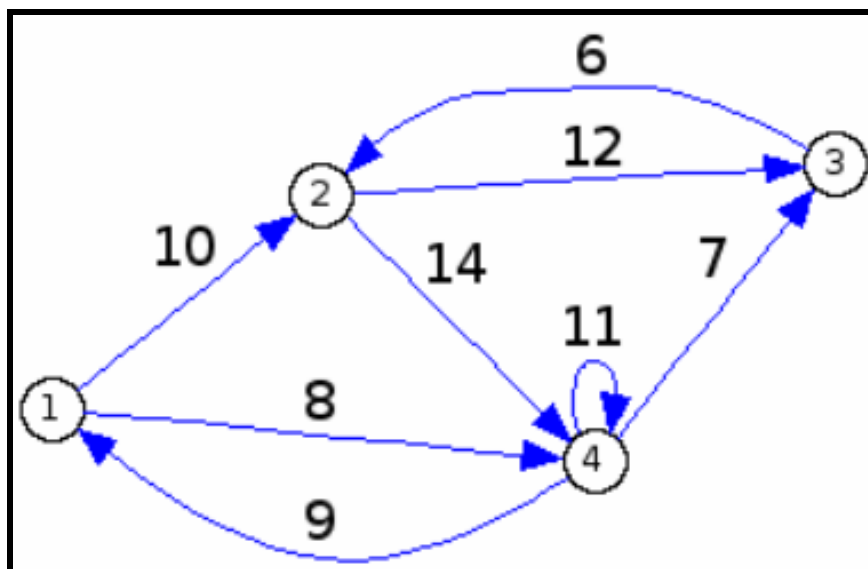
Il n'existe pas d'arc du sommet 3 vers le sommet 5 donc on met un 0 à l'intersection de la ligne 3 et la colonne 5.

Il existe un arc du sommet 1 vers le sommet 5 donc on met un 1 à l'intersection de la ligne 1 et la colonne 5.

La matrice d'adjacence de ce graphe orienté est donc :

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

Il est également possible d'établir une matrice d'adjacence pour un graphe orienté et pondéré : on remplace les 1 par les valeurs liées à chaque arc. Soit le graphe orienté et pondéré suivant :



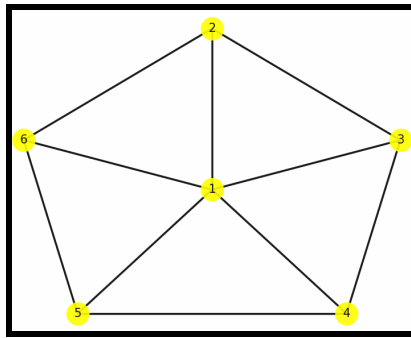
Pour tracer un graphe orienté et pondéré en ligne :

https://wims.univ-cotedazur.fr/wims/fr_tool~geometry~graphviz.fr.html

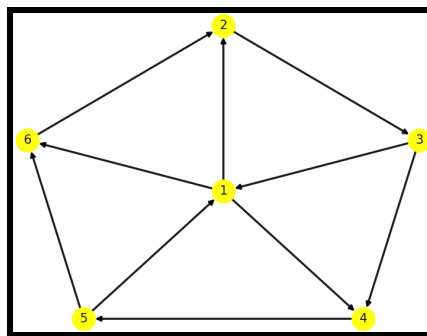
La matrice d'adjacence de ce graphe orienté et pondéré est donc :

$$A = \begin{pmatrix} 0 & 10 & 0 & 8 \\ 0 & 0 & 12 & 14 \\ 0 & 6 & 0 & 0 \\ 9 & 0 & 7 & 11 \end{pmatrix}$$

Donnez la matrice d'adjacence pour le graphe non orienté suivant :



Donnez la matrice d'adjacence pour le graphe orienté suivant :



2.3. MATRICES D'ADJACENCES OU LISTES D'ADJACENCES

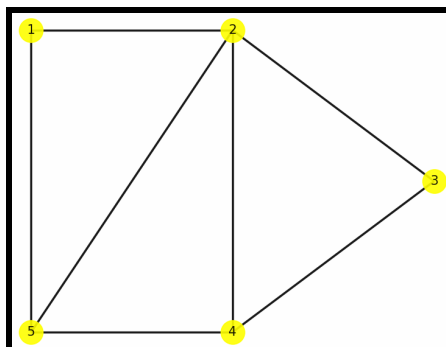
Le choix se fait en fonction de la densité du graphe (rapport entre le nombre d'arêtes / d'arcs et le nombre de sommets. Pour un graphe dense on utilisera plutôt une matrice d'adjacence. Le choix dépend aussi des algorithmes utilisés. Certains algorithmes travaillent plutôt avec les listes d'adjacences alors que d'autres travaillent plutôt avec les matrices d'adjacences.

3. IMPLEMENTATION DES GRAPHS EN PYTHON

3.1. IMPLEMENTATION EN PYTHON DES GRAPHS PAR LISTES D'ADJACENCES, LISTES DE SUCESSEURS / DE PREDECESSEURS

En Python, Il est possible de travailler avec des listes d'adjacences en utilisant les dictionnaires avec des tuples () non modifiables ou des listes [] modifiables.

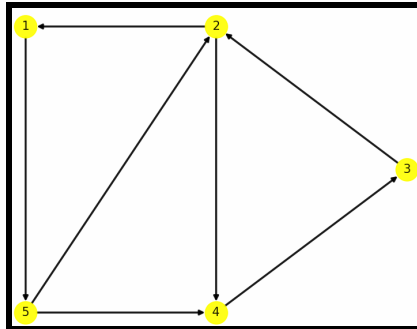
Pour le graphe non orienté suivant :



La liste d'adjacence est :

```
list_adja = {'1': ('2', '5'), '2': ('1', '3', '4', '5'), '3': ('2', '4'),
             '4': ('2', '3', '5'), '5': ('1', '2', '4')}
```

Pour le graphe orienté suivant :



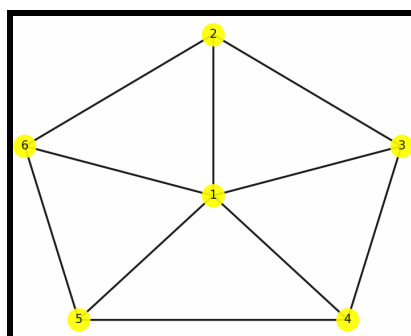
La liste des successeurs est :

```
list_succ = {'1': ('5'), '2': ('1', '4'), '3': ('2'), '4': ('3'),
             '5': ('2', '4')}
```

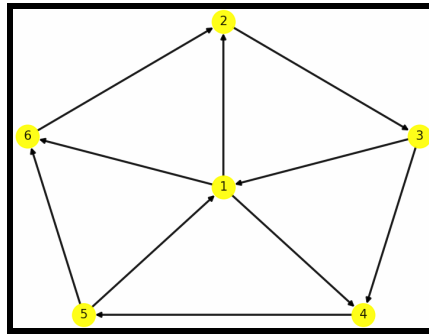
La liste des prédécesseurs est :

```
list_pred = {'1': ('2'), '2': ('3', '5'), '3': ('4'), '4': ('2', '5'),
             '5': ('1')}
```

Donnez le script d'implémentation de la liste d'adjacence pour le graphe non orienté suivant :

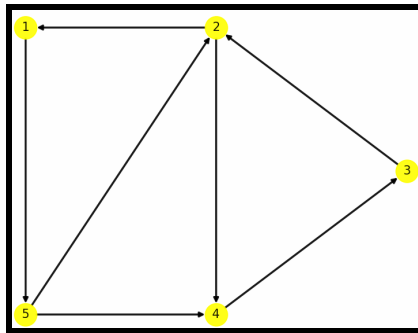


Donnez les scripts d'implémentation de la liste des successeurs et de la liste des prédécesseurs du graphe orienté suivant :



3.2. IMPLEMENTATION EN PYTHON DES GRAPHS PAR MATRICES D'ADJACENCES

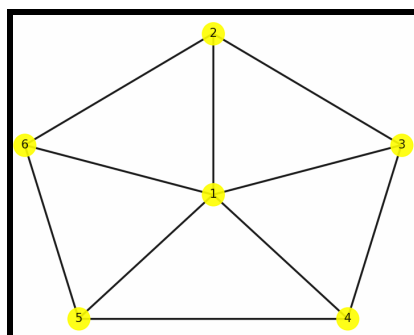
Pour le graphe orienté suivant :



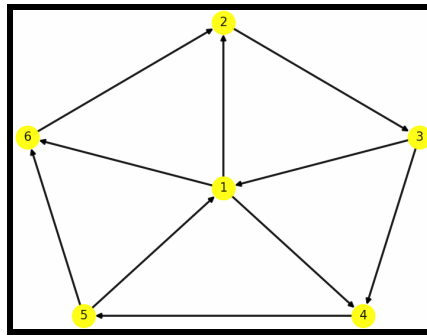
La matrice d'adjacence est :

```
mat_adja = [[0,0,0,0,1],
             [1,0,0,1,0],
             [0,1,0,0,0],
             [0,0,1,0,0],
             [0,1,0,1,0]]
```

Donnez le script d'implémentation de la matrice d'adjacence pour le graphe non orienté suivant :



Donnez les scripts d'implémentation de la matrice d'adjacence du graphe orienté suivant :



3.3. PASSER D'UNE REPRESENTATION DES GRAPHS PAR MATRICES D'ADJACENCES A UNE REPRESENTATION PAR LISTES D'ADJACENCES

Complétez le script suivant qui permet de passer d'une représentation des graphes par matrice d'adjacence à une représentation par liste d'adjacence :

```
liste1= ['1','2','3','4','5','6'] # liste des sommets
n=len(liste1) # nombre de sommets = longueur de la liste
m=[[0,1,1,1,1,1],[1,0,1,0,0,1],[1,1,0,1,0,0],[1,0,1,0,1,0],[1,0,0,1,0,1],[1,1,0,0,1,0]] # matrice d'adjacences
g=dict() # dictionnaire représentant la liste d'adjacences

. . .

. . .

print("La matrice d'adjacence est : ", m) # on affiche la matrice d'adjacences
print("La liste d'adjacence correspondante est : ", g) # on affiche de dictionnaire représentant la liste d'adjacences
```

Ce script permettra d'afficher ceci :

```
La matrice d'adjacence est : [[0, 1, 1, 1, 1, 1], [1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 0, 0], [1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1], [1, 1, 0, 0, 1, 0]]

La liste d'adjacence correspondante est : {'1': ['2', '3', '4', '5', '6'], '2': ['1', '3', '6'], '3': ['1', '2', '4'], '4': ['1', '3', '5'], '5': ['1', '4', '6'], '6': ['1', '2', '5']}
```

Dessinez le graphe correspondant

3.4. PASSER D'UNE REPRESENTATION DES GRAPHS PAR LISTES D'ADJACENCES A UNE REPRESENTATION PAR MATRICES D'ADJACENCES

Complétez le script suivant qui permet de passer d'une représentation des graphes par liste d'adjacence à une représentation par matrice d'adjacence :

```
liste= ['1','2','3','4','5','6'] # liste des sommets
g={'1': ['2', '3', '4', '5', '6'], '2': ['1', '3', '6'], '3': ['1', '2', '4'], '4': ['1', '3', '5'], '5': ['1', '4', '6'], '6': ['1', '2', '5']} # dictionnaire représentant la liste d'adjacences
n=len(liste) # longueur de la liste = nombre de sommets

. . .

. . .

print("La liste d'adjacence est : ", g) # on affiche de dictionnaire
représentant la liste d'adjacences
print("La matrice d'adjacence correspondante est : ", m) # on
affiche la matrice d'adjacences
```

Ce script permettra d'afficher ceci :

```
La liste d'adjacence est : {'1': ['2', '3', '4', '5', '6'], '2': ['1', '3', '6'], '3': ['1', '2', '4'], '4': ['1', '3', '5'], '5': ['1', '4', '6'], '6': ['1', '2', '5']}

La matrice d'adjacence correspondante est : [[0, 1, 1, 1, 1, 1],
[1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 0, 0], [1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1], [1, 1, 0, 0, 1, 0]]
```

3.5. PYTHON, AUTRES REPRESENTATION DES GRAPHS

En Python, il est également possible de travailler avec des listes d'adjacences en utilisant des listes [].

Les sommets d'un graphe sont numérotés de 1 à n, alors que les indices en Python commencent à 0. Nous pouvons donc faire le choix de représenter la liste d'adjacence par une liste de listes de 0 à n et en laissant « la liste 0 » vide.

La liste d'adjacence du script précédent sous forme de dictionnaire :

```
La liste d'adjacence est : {'1': ['2', '3', '4', '5', '6'], '2': ['1', '3', '6'], '3': ['1', '2', '4'], '4': ['1', '3', '5'], '5': ['1', '4', '6'], '6': ['1', '2', '5']}
```

Sera représentée par la liste de liste suivante :

```
La liste d'adjacence est : [[], [2,3,4,5,6], [1,3,6], [1,2,4], [1,3,5], [1,4,6], [1,2,5]]
```

Le sommet « 0 » n'a pas d'adjacence (pas d'arc ou d'arêtes avec d'autres sommets).

Le sommet « 1 » est relié par des arcs ou des arêtes avec les sommets « 2 », « 3 », « 4 », « 5 » et « 6 ».

Le sommet « 2 » est relié par des arcs ou des arêtes avec les sommets « 1 », « 3 » et « 6 ».

...

Vous trouverez sur l'ENT une classe **Tableau_Term_TP8_cours14_Graphe** (pour le tableau des degrés des sommets d'un graphe), une classe **Matrice_Term_TP8_cours15_Graphe** (pour les matrices d'adjacences) et une classe **ListeAdj_a_completer_Term_TP8_cours13_Graphe** à compléter en vous aidant des informations ci-dessous et que vous devrez renommer sous la classe **ListeAdj_Term_TP8_cours13_Graphe** avant de faire les tests.

Dans un premier temps vous devrez créer un programme principal nommé **test_list_et_mat_adj_Term_TP8_cours18_Graphe** qui contiendra le script suivant :

```
from ListeAdj_Term_TP8_cours13_Graphe import ListeAdj
from Matrice_Term_TP8_cours15_Graphe import Matrice
from Tableau_Term_TP8_cours14_Graphe import Tableau
from Conversions_Term_TP8_cours17_Graphe import *

lstAdj = ListeAdj(5)
# créer un graphe de 5 sommets = une liste de 6 listes mais
# la liste[0] reste vide
lstAdj.ajoutArete(1,2)
lstAdj.ajoutArete(1,5)
lstAdj.ajoutArete(2,3)
lstAdj.ajoutArete(2,4)
lstAdj.ajoutArete(2,5)
lstAdj.ajoutArete(3,4)
lstAdj.ajoutArete(5,4)
```

Testez ce script, que réalise-t-il ?

Il devra évoluer au fur et à mesure que vous souhaiterez tester vos nouveaux scripts et il vous permettra de tester toutes les fonctions de la classe

ListeAdj_Term_TP8_cours13_Graphe

Dans toute la suite, nous considérons que **g** est un graphe non orienté. Les plus rapides pourront essayer d'adapter ces fonctions au cas des graphes orientés.

La classe **ListeAdj_Term_TP8_cours13_Graphe** vous est proposée pour représenter les graphes par leurs listes d'adjacences via l'attribut **listA**. Remarquez que celui-ci est construit de telle sorte que **g.listA[i]** est la liste d'adjacence du sommet **i** du graphe **g**. Il n'y a par convention, pas de sommet 0, donc pour tout graphe **g**, **g.listA[0]** est donc une liste vide.

Compléter la définition de cette classe **ListeAdj_Term_TP8_cours13_Graphe** en y ajoutant :

- **nbSommets(self)** : retourne le nombre de sommets du graphe **g**

- **nbAretes(self)** : retourne le nombre d'arêtes du graphe **g**
- **ajoutArete(self, i, j)** : ajoute l'arête (**i, j**) au graphe **g**
- **enleveArete(self, i, j)** : enlève une arête (**i, j**) du graphe **g** (si elle existe !)
- **getAretesSommet(self, i)** : retourne la liste d'adjacence du sommet **i** du graphe **g**
- **degSommet(self, i)** : retourne le degré du sommet **i**
- **degre(self)** : retourne un tableau **d** tel que **d.tab[i]** est le degré du sommet **i**

3.5.1. PYTHON, CONVERSIONS ENTRE LES REPRESENTATION DES GRAPHS

Dans le programme

Conversions_a_completer_Term_TP8_cours17_Graphe et que vous devrez renommer sous le programme **Conversions_Term_TP8_cours17_Graphe**, écrivez les fonctions de conversions suivantes :

- **listeToMatrice(g)** : retourne la matrice d'adjacence représentant **g** donné par ses listes d'adjacences. Nous utiliserons la classe **Matrice_Term_TP8_cours15_Graphe**.
- **areteToListe(n, l)** : retourne les listes d'adjacence (utiliser la classe **ListeAdj_Term_TP8_cours13_Graphe**) représentant un graphe donné par son nombre de sommets **n** et sa liste **l** d'arêtes.
- **matToListe(m)** : retourne les listes d'adjacences représentant un graphe donné par sa matrice d'adjacence **m**.

Afin de tester vos nouvelles fonctions de conversions vous devez ajouter au programme principal nommé **test_list_et_mat_adj_Term_TP8_cours18_Graphe** le script suivant :

```
print()
print("La liste d'adjacence est " + str(lstAdj))
print()
print("La matrice d'adjacence faite à partir de la liste d'adjacence
est " + str(listeToMatrice(lstAdj)))
print()
print("La liste d'adjacence faite à partir de la matrice d'adjacence
est " + str(matToListe(listeToMatrice(lstAdj))))
print()
liste_Arretes = [[1, 5], [2,1], [2,4], [3,2], [4,3], [5,2], [5,4]]
print("La liste d'arrêtes est " + str(liste_Arretes))
print()
print("La liste d'adjacence faite à partir de la liste d'arrêtes est
" + str(areteToListe(lstAdj.nbSommets(), liste_Arretes)))
```

Lorsque le programme **Conversions_Term_TP8_cours17_Graphe** sera terminé vous devrez le tester avec le programme suivant

TestConversions_Term_TP8_cours16_Graphe :

Deux façons se présentent à vous :

* Sous un terminal python taper l'instruction :

python -m unittest TestConversions_Term_TP8_cours16_Graphe

(quand **Conversions_Term_TP8_cours17_Graphe** est correct vous avez l'indication dans le terminal : « Le test liste vers graphe est OK », « Le test liste vers matrice est OK », « Le test matrice vers graphe est OK » et « OK ».

* Sous edupython :

- charger le programme **Conversions_Term_TP8_cours17_Graphe**
- charger le programme **TestConversions_Term_TP8_cours16_Graphe**
- exécuter le programme **Conversions_Term_TP8_cours17_Graphe**
- sous **TestConversions_Term_TP8_cours16_Graphe**, aller dans « **Affichage** », puis dans « **Fenêtre de l'IDE** » et enfin dans « **Tests des éléments** », sélectionnez les 3 tests et cliquez sur « **play** » (quand tout est au vert cela prouve que le programme **Conversions_Term_TP8_cours17_Graphe** est correct et l'affichage dans la console est le suivant : « Le test liste vers graphe est OK », « Le test liste vers matrice est OK » et « Le test matrice vers graphe est OK ».

3.5.2. PYTHON, PARCOURS DES GRAPHS

Nous allons nous intéresser aux algorithmes de parcours d'un graphe. L'idée est de « visiter » tous les sommets d'un graphe en partant d'un sommet quelconque. Ces algorithmes de parcours d'un graphe sont à la base de nombreux algorithmes (routage des paquets de données dans un réseau, découverte du chemin le plus court pour aller d'une ville à une autre, ...). Il existe 2 méthodes pour parcourir un graphe :

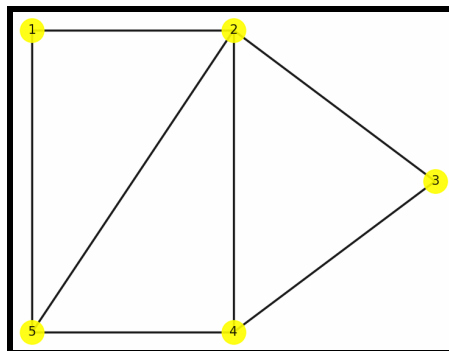
- Le parcours en largeur
- le parcours en profondeur

3.5.2.1. PYTHON, PARCOURS EN LARGEUR DES GRAPHS

Le **parcours en largeur** consiste, à partir d'un sommet donné, à visiter tous ses voisins non visités, puis les voisins non visités de ses voisins, etc. On énumère tous les voisins avant de passer au voisin.

Visualisez la vidéo suivante : <https://www.youtube.com/watch?v=8vnbENILU5E>

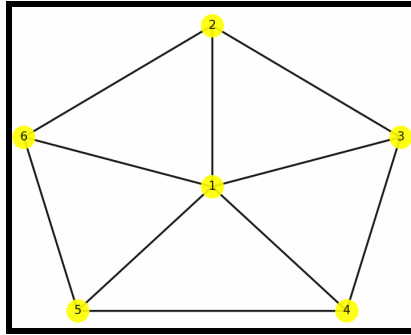
Pour le graphe non orienté suivant :



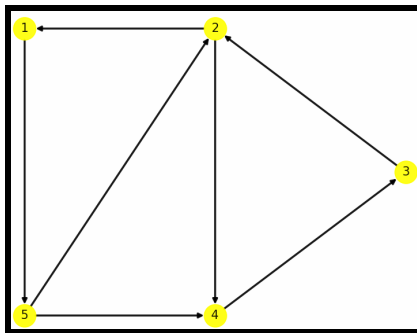
Le parcours en largeur à partir du sommet 1 est [1, 2, 5, 3, 4]. Les voisins de 1 sont 2 et 5, les voisins non visités de 2 sont 3 et 4.

L'ordre dans lequel les sommets sont traités est alors appelé **ordre de parcours en largeur**.

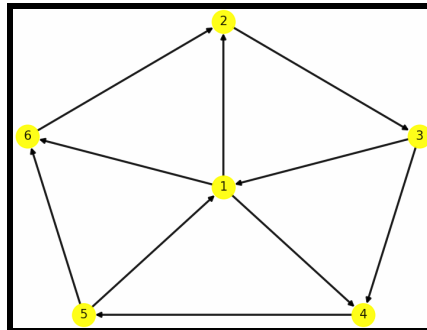
Donnez l'ordre de parcours en largeur à partir du sommet 4 du graphe non orienté suivant :



Donnez l'ordre de parcours en largeur à partir du sommet 1 du graphe orienté suivant :



Donnez l'ordre de parcours en largeur à partir du sommet 1 du graphe orienté suivant :



Dans le programme

Parcours_a_completer_Term_TP8_cours19_graphe,

- représentez les graphes G1, G2 et G3,
- écrivez une fonction **largeur(g, i)** qui effectue le parcours en largeur du graphe **g** à partir du sommet **i**. Vous sauvegarderez votre travail sous **Parcours_Term_TP8_cours19_graphe.**

Aides :

- On utilise un tableau **visite** tel que **visite[i]=0** si le sommet **i** n'a pas encore été visité et **visite[i]=1** dans le cas contraire.
- On utilise une liste **file** pour gérer la file d'attente des sommets à visiter prochainement.

- Le résultat de l'appel de la fonction sera la liste des sommets visités dans l'ordre de ce parcours représenté par une liste **ordreVisite**.
- On utilisera les fonctions **nbSommets** et **getAretesSommet** de la classe **ListeAdj_Term_TP8_cours13_Graphe**.
- Principe : pour cela, au fur et à mesure que l'on rencontre de nouveaux sommets (non encore visités), on mémorise leurs voisins dans une file d'attente **file** pour une visite prochaine. **visite** est tableau de booléens tel que **visite[i] = Vrai** si et seulement si le sommet **i** a déjà été visité. **file** est la file des sommets qu'on devra prochainement visiter.
- Algorithme :

```

largeur(g,i) : {parcours en largeur du graphe G à partir du sommet i}
  initialiser un tableau visite à Faux
  file = [i]
  visite[i-1] = Vrai
  Tant que F n'est pas vide
    considérer y la tête de file (et l'enlever de file)
    pour chaque successeur z de y
      Si visite[z] = faux
        visite[z] = vrai
        ajouter z à la fin de la file

```

3.5.2.2. PYTHON, PARCOURS EN PROFONDEUR DES GRAPHS

Le **parcours en profondeur** consiste, à partir d'un sommet donné, à suivre un chemin le plus loin possible, sans passer deux fois par le même sommet. On énumère un seul voisin avant de passer au suivant. Quand on rencontre un sommet déjà visité, on fait marche arrière pour revenir au sommet précédent et explorer les chemins ignorés précédemment.

Visualisez les vidéos suivantes :

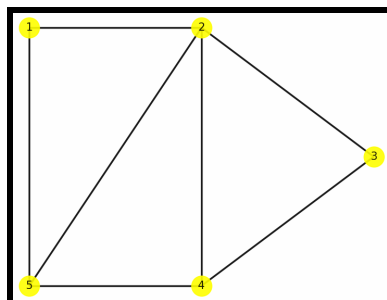
<https://www.youtube.com/watch?v=kcedjJOjDpg>

<https://www.youtube.com/watch?v=wPLAsAuenA>

Ordre de **parcours en première et en dernière visite** : on traite les sommets **i** rencontrés, soit en première visite (quand on commence le parcours en profondeur de **i**, quand on avance dans le graphe), soit en dernière visite (quand ce parcours est terminé et qu'on remonte au sommet qui a appelé récursivement **profond(g, i)**, quand on recule dans le graphe).

L'ordre dans lesquels les sommets sont traités est alors respectivement appelé **ordre de parcours en profondeur en première visite** ou **ordre de parcours en profondeur en dernière visite**. L'ordre de parcours en profondeur en dernière visite n'est pas toujours l'ordre inverse de l'ordre de parcours en profondeur en première visite.

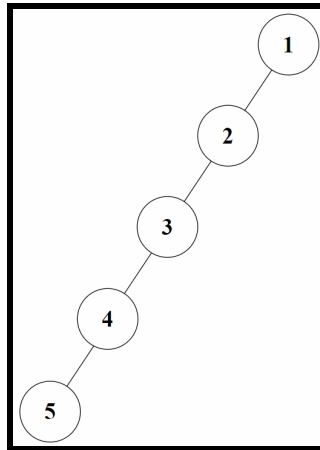
Pour le graphe non orienté suivant :



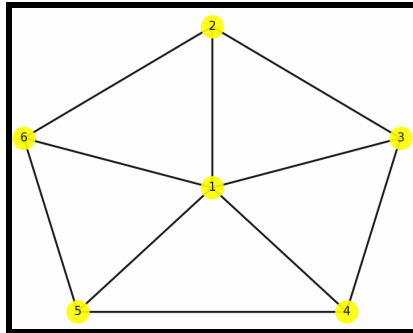
Le parcours en profondeur en première visite à partir du sommet 1 est [1, 2, 3, 4, 5]. Les voisins de 1 sont 2 et 5, on prend 2. Les voisins non visités de 2 sont 3, 4 et 5, on prend 3. Le voisin non visités de 3 est 4. Le voisin non visités de 4 est 5.

Le parcours en profondeur en dernière visite à partir du sommet 1 est [5, 4, 3, 2, 1]. Lors du parcours en profondeur, le dernier nœud visité est 5, le précédent est 4, ...

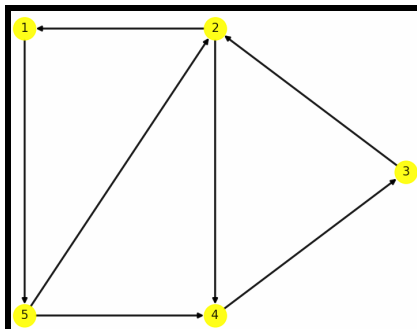
Afin de nous aider dans la détermination du parcours en profondeur en première visite et en dernière visite, on peut tracer l'arbre de parcours en profondeur à partir du sommet concerné. Pour le parcours en profondeur en première visite, on lit l'arbre à partir de la racine. Pour le parcours en profondeur en dernière visite, on lit l'arbre à partir des feuilles.



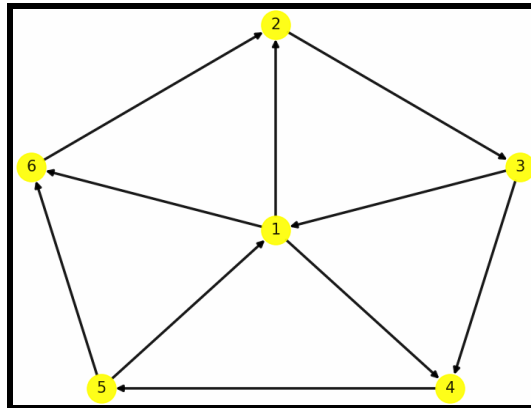
Donnez l'ordre de parcours en profondeur en première et en dernière visite à partir du sommet 4 du graphe non orienté suivant :



Donnez l'ordre de parcours en profondeur en première et en dernière visite à partir du sommet 1 du graphe orienté suivant :



Donnez l'ordre de parcours en profondeur en première et en dernière visite à partir du sommet 1 du graphe orienté suivant :



Comme souvent, lorsqu'on écrit un programme récursif, on distinguera les fonctions auxiliaires récursives (**profondeurRecPremiereVisite(g, i, visite, ordreVisite)** et **profondeurRecDerniereVisite(g, i, visite, ordreVisite)**) et les fonctions d'appels (**profondeurPremiereVisite(g, i)** et **profondeurDerniereVisite(g, i)**) dont le rôle est d'initialiser les variables et d'enclencher le premier appel récursif.

Dans le programme précédent **Parcours_Term_TP8_cours19_graphe**, écrire les fonctions suivantes :

- **profondeurRecPremiereVisite(g, i, visite, ordreVisite)** et **profondeurRecDerniereVisite(g, i, visite, ordreVisite)** : fonctions auxiliaires récursives qui provoquent un parcours en profondeur du graphe à partir du sommet **i**. Ces fonctions ne retournent aucun résultat et se contentent de mettre à jour les paramètres **visite** et **ordreVisite**.
- **profondeurPremiereVisite(g, i)** et **profondeurDerniereVisite(g, i)** : qui effectuent le parcours en profondeur du graphe **g** à partir du sommet **i**.

Aides :

- Le résultat de l'appel de la fonction sera la liste des sommets visités dans l'ordre de ce parcours représenté par une liste **ordreVisite**.
- En fonction de l'endroit où on positionne la mise à jour de la liste **ordreVisite** la fonction donnera l'ordre de première visite ou l'ordre de dernière visite.
- Principe : cet algorithme se conçoit naturellement de manière récursive. Pour savoir si un sommet a déjà été visité, nous utilisons un tableau de booléens **visite** tel que **visite[x] = Vrai** si et seulement si le sommet **x** a déjà été visité.
- Algorithme :

profond(G,i) : {parcours en profondeur du graphe G à partir du sommet i}

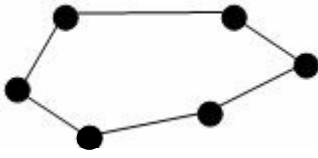

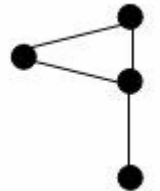
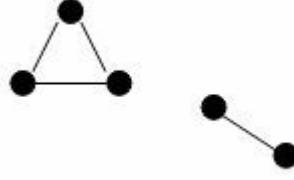
```

initialiser un tableau visite à Faux
visite[i] = Vrai {première visite de x}
Pour chaque voisin y de i faire
    Si visite[y] = faux alors
        profond(G,y)
    {Sinon on détecte une revisite de y}
{traiter x en dernière visite}

```

3.5.3. PYTHON, GRAPHE CONNEXES

Nous souhaitons tester la connexité d'un graphe à l'aide d'un parcours en largeur. Un **graphe non orienté est connexe** si et seulement si **un parcours à partir du sommet 1 visite tous les sommets**.

Graphes connexes	Graphes non connexes
	
	

Copiez et modifiez votre programme **largeur(g, i)** permettant de faire le parcours en largeur d'un graphe **g** à partir du sommet **i** en une fonction booléenne **estConnexe(g)** qui permettra de déterminer si un graphe **g** est connexe ou non.

Testez votre fonction **estConnexe(g)** sur les graphes **g1**, **g2** et **g3** déjà présents dans le programme **largeur(g, i)** et y ajoutez le graphe **glbis** suivant. Vous donnerez une représentation de ce nouveau graphe :

```

def construireGrapheGlbis():
    g = ListeAdj(5)
    g.ajoutArete(1, 5)
    g.ajoutArete(2, 4)
    g.ajoutArete(3, 2)
    g.ajoutArete(4, 3)
    return g

```