COURS DE NSI (Numérique et Sciences Informatiques)

Classe : terminale	LYCÉE INTERNATIONAL COURS LUMIÈRE
séquence 2	**Structures de données : les graphes & Algorithmes sur les graphes**
Mr BANANKO K.	

Objectifs :

Contenus	Capacités attendues	Commentaires
Graphes : structures relationnelles. Sommets, arcs, arêtes, graphes orientés ou non orientés.	Modéliser des situations sous forme de graphes. Écrire les implémentations correspondantes d'un graphe : matrice d'adjacence, liste de successeurs/de prédécesseurs. Passer d'une représentation à une autre.	On s'appuie sur des exemples comme le réseau routier, le réseau électrique, Internet, les réseaux sociaux. Le choix de la représentation dépend du traitement qu'on veut mettre en place : on fait le lien avec la rubrique « algorithmique ».
Algorithmes sur les graphes.	Parcourir un graphe en profondeur d'abord, en largeur d'abord. Repérer la présence d'un cycle dans un graphe. Chercher un chemin dans un graphe.	Le parcours d'un labyrinthe et le routage dans Internet sont des exemples d'algorithme sur les graphes. L'exemple des graphes permet d'illustrer l'utilisation des classes en programmation.

Référence Manuels hachette Education : Page 158

I- STRUCTURES DE DONNEES : LES GRAPHES

A- Introduction

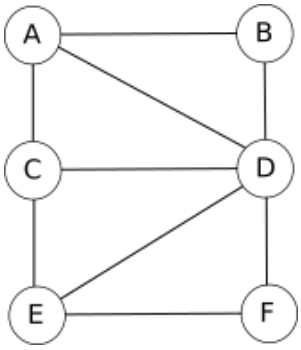
Imaginez un réseau social ayant 6 abonnés (A, B, C, D, E et F) où :

- A est ami avec B, C et D
- B est ami avec A et D
- C est ami avec A, E et D
- D est ami avec tous les autres abonnés
- E est ami avec C, D et F
- F est ami avec E et D

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 : on peut représenter chaque abonné par un cercle (avec le nom de l'abonné situé dans le cercle) et chaque relation "X est ami avec Y" par un segment de droite reliant X et Y ("X est ami avec Y" et "Y est ami avec X" étant représenté par le même segment de droite).

Voici ce que cela donne avec le réseau social décrit ci-dessus :



Graphe réseau social

B- NOTION DE GRAPHE

1- Définition d'un graphe

Un graphe est un ensemble d'objets appelés sommets dont certains reliés deux à deux par des liaisons appelées arcs. En général on associe une *étiquette* ou *nom* à chaque sommet. Il existe deux grandes familles de graphes :

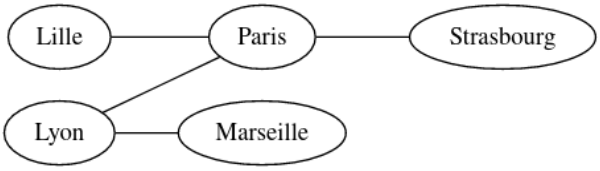
- si les arcs peuvent être orientés on parle de **graphe orienté**
- sinon de **graphe non orienté**.

On peut aussi associer une *étiquette*, en général une valeur numérique appelée poids, à chaque arc : dans ce cas on parle de **graphe pondéré** (qui peut être orienté ou non).

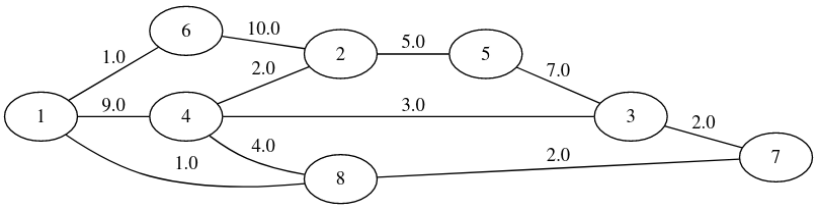
Conventions de représentation graphique :

Type de graphe	Sommet	Arc
Non orienté	Disque avec étiquette	**i – j** représenté par un segment
Orienté	Disque avec étiquette	**i -> j** représenté par une flèche orientée

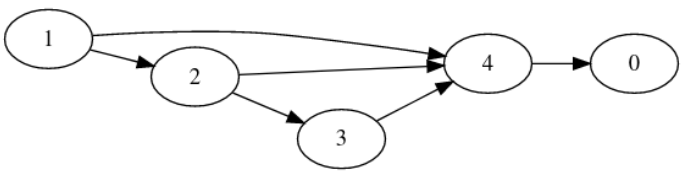
Une représentation de graphe non orienté.



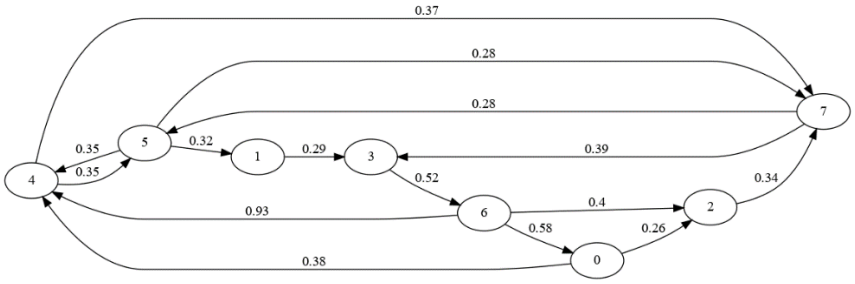
Une représentation de graphe non orienté pondéré.



Une représentation de graphe orienté



Une représentation de graphe orienté pondéré.



Les graphes sont des objets mathématiques très utilisés, notamment en informatique. Les cercles sont appelés des sommets et les segments de droites qui relient 2 sommets des arêtes.

Un graphe est un ensemble de sommets et d'arcs

Un graphe est défini par un ensemble V de **sommets** (*vertices* en anglais) et un ensemble E d'**arcs** (*edges* en anglais) qui sont des couples de sommets.

Plus formellement on dira qu'un graphe G est un couple **G = (V,E)** avec V un ensemble de sommets et E un ensemble d'arêtes

Les **arcs** peuvent être **orientés** ou **non orientés**.

Un arc est une relation d'adjacence entre deux sommets

Si un arc a pour origine le sommet x et pour extrémité le sommet y, on dit que :

- y est **adjacent** à x ou que y est un **voisin** de x .
- y est un **successeur** de x et que x est un **prédécesseur** de y

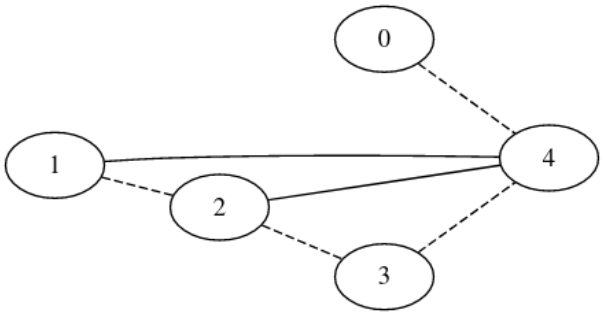
Pour un arc non orienté, on ne distingue pas prédécesseur et successeur et la relation d'adjacence est symétrique : si y est voisin de x alors x est voisin de y.

> On note x -> y un arc orienté et x – y un arc non orienté. Pour un arc orienté, on distingue l'arc x -> y de l'arc y -> x .

Exemple

Adjacence dans un graphe non orienté

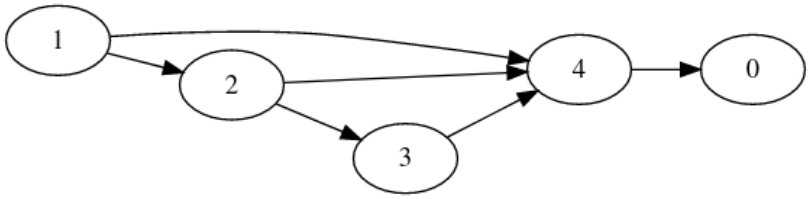
Le sommet 2 est adjacent aux sommets 1, 3 et 4 : il a trois voisins.



Adjacence dans un graphe orienté

Le sommet 2 a deux voisins 3 et 4.

Le sommet 2 est un voisin du sommet 1 mais la réciproque est fausse : l'arc orienté 1 -> 2 définit 1 comme prédécesseur de 2 et 2 comme successeur de 1.



Un chemin est une séquence d'arcs consécutifs

Un **chemin** est une séquence d'arcs consécutifs :

Ainsi le chemin x_0 ->x_1 -> x_2 ... -> x_n part de l'origine x_0 puis par le sommet x_1, puis le sommet x_2 et conduit jusqu'à l'extrémité x_n en suivant des arcs consécutifs.

La **longueur d'un chemin** est le nombre d'arcs qui le constitue.

Un **chemin simple** est un chemin sans répétition d'arcs.

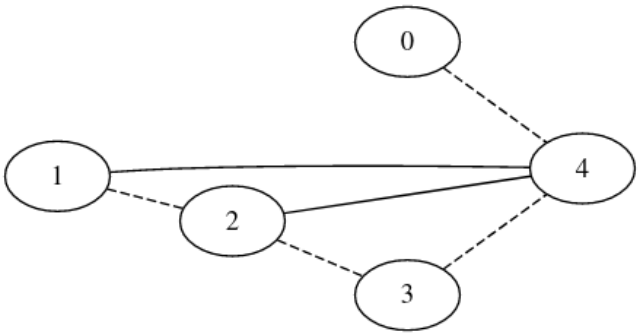
Un **cycle** est un chemin dont l'extrémité coïncide avec l'origine (une chaine qui commence et se termine au même sommet).

Exemple :

Chemin dans un graphe non orienté

0 - 4 - 3 - 2 - 1 est un chemin de longueur 4 dans le graphe non orienté ci-dessous.

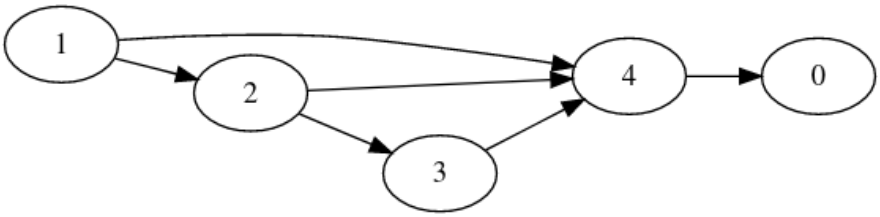
4 - 3 - 2 - 4 est un cycle de longueur 3 dans ce même graphe



Chemin dans un graphe orienté

1 -> 2 -> 3 -> 4 - > 0 est un chemin de longueur 4 dans le graphe orienté ci-dessous.

Ce graphe orienté ne contient pas de cycles.



Degré d'un arc

Dans un *graphe orienté* :

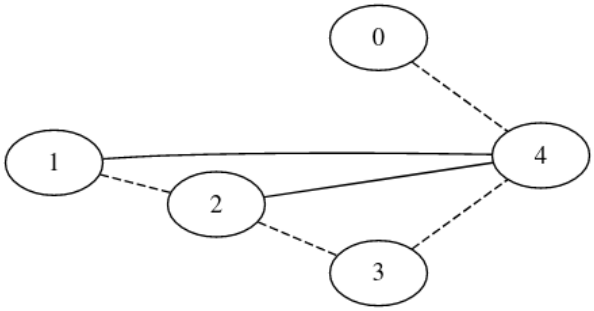
- le **degré sortant** d'un sommet est le nombre d'arcs dont ce sommet est l'origine : c'est le nombre de successeurs de ce sommet
- le **degré entrant** d'un sommet est le nombre d'arcs dont ce sommet est l'extrémité : c'est le nombre de prédécesseurs de ce sommet

Dans un *graphe non orienté*, on ne distingue pas degré sortant et degré entrant : le **degré d'un sommet** est le nombre d'arcs dont il est une extrémité, c'est le nombre de *voisins* du sommet.

Exemple :

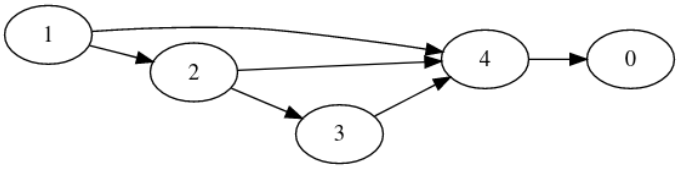
Degrés dans un graphe non orienté

Dans le graphe non orienté ci-dessous le degré du sommet 2 est de trois.



Degrés dans un graphe orienté

Dans le graphe orienté ci-dessous, le degré entrant du sommet 2 est de un et son degré sortant est de deux.



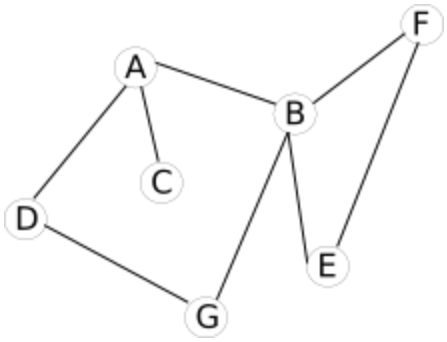
Autre utilisation possible des graphes : les logiciels de cartographie (ces logiciels sont souvent utilisés couplés à des récepteurs GPS). Ces logiciels de cartographie permettant, connaissant votre position grâce à un récepteur GPS, d'indiquer la route à suivre pour se rendre à endroit B. Comment modéliser l'ensemble des lieux et des routes ? Simplement à l'aide d'un graphe ! Chaque lieu est un sommet et les routes qui relient les lieux entre eux sont des arêtes.

Soit les lieux suivants : A, B, C, D, E, F et G.

Les différents lieux sont reliés par les routes suivantes :

- il existe une route entre A et C
- il existe une route entre A et B
- il existe une route entre A et D
- il existe une route entre B et F
- il existe une route entre B et E
- il existe une route entre B et G
- il existe une route entre D et G
- il existe une route entre E et F

Ici aussi, la représentation sous forme de graphe s'impose :



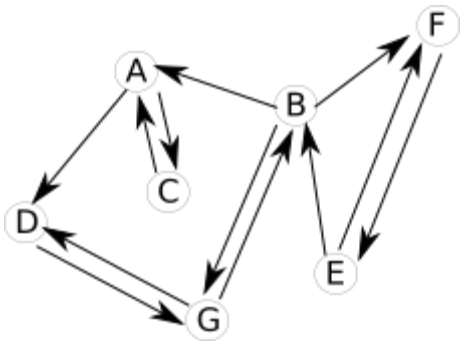
Graphe cartographie

Problème : avec cette représentation du réseau routier sous forme de graphe, il est impossible de tenir compte des routes en sens unique (par exemple il est possible d'aller de A vers D mais pas de D vers A)

Voici de nouvelles contraintes :

- il existe une route entre A et C (double sens)
- il existe une route entre A et B (sens unique B->A)
- il existe une route entre A et D (sens unique A->D)
- il existe une route entre B et F (sens unique B->F)
- il existe une route entre B et E (sens unique E->B)
- il existe une route entre B et G (double sens)
- il existe une route entre D et G (double sens)
- il existe une route entre E et F (double)

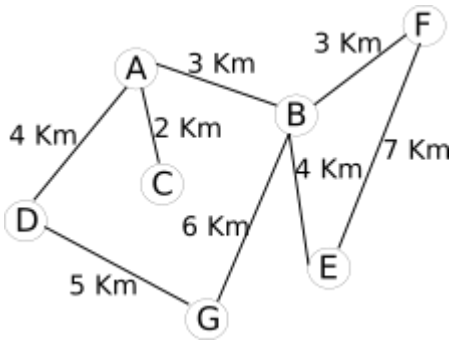
Pour tenir compte de ces nouvelles contraintes, on utilisera un graphe orienté :



Graphe orienté cartographie

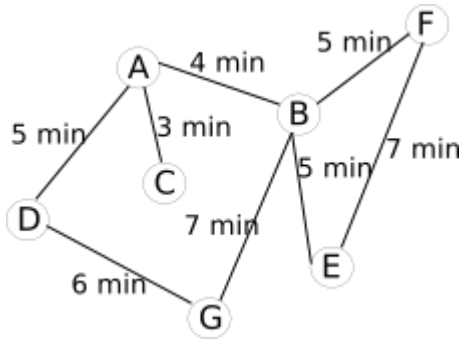
Dans un graphe orienté, les arêtes possèdent une orientation. Ces "arêtes orientées" sont souvent appelées "arcs". On dira qu'un graphe orienté G est un couple G = (V,A) avec V un ensemble de sommets et A un ensemble d'arcs.

Parfois il est intéressant d'associer aux arrêtes ou aux arcs des valeurs, on parle alors de graphes pondérés. Si nous revenons à notre "graphe cartographie", il est possible d'associer à chaque arête la distance en Km entre les 2 lieux:



Graphe pondéré (Km) cartographie

Il est aussi possible d'associer à chaque arête la durée du trajet entre 2 points :



Graphe pondéré (minutes) cartographie

En fonction du choix fait par le conducteur (trajet le plus court "**en distance**" ou trajet le plus court "**en temps**"), l'algorithme permettant de déterminer le "***chemin le plus court entre 2 points***" travaillera sur le graphe "***graphe pondéré (Km) cartographie***" ou sur le graphe "***graphe pondéré (minutes) cartographie***". À noter que le "***graphe pondéré (minutes) cartographie***" peut évoluer au cours du temps en fonction du trafic routier : une application comme Waze utilise les données en en provenance des utilisateurs de l'application afin de mettre à jour en temps réel leur "***graphe pondéré (minutes) cartographie***".

Enfin, avant de s'intéresser à l'implémentation des graphes, voici 2 définitions qui nous seront utiles par la suite :

- **Une chaine** est une suite d'arêtes consécutives dans un graphe, un peu comme si on se promenait sur le graphe. On la désigne par les lettres des sommets qu'elle comporte.
- **Un cycle** est une chaine qui commence et se termine au même sommet.

Exercice 1

i.

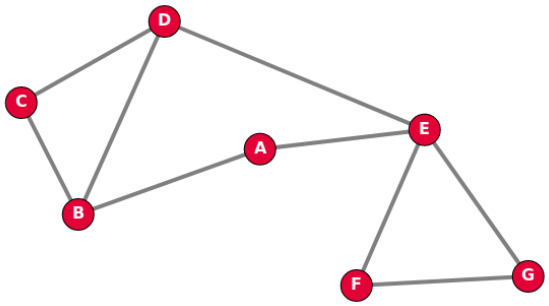
Le World Wide Web peut être modélisé par un graphe :

- Quels objets du Web seront les sommets ? et les arcs ?
- S'agira-t-il d'un graphe orienté ou non orienté ?

ii.

Dans le graphe non orienté ci-dessous, déterminez :

- le sommet de degré maximal
- le nombre de cycles
- le nombre de chemins simples entre le sommet C et le sommet G

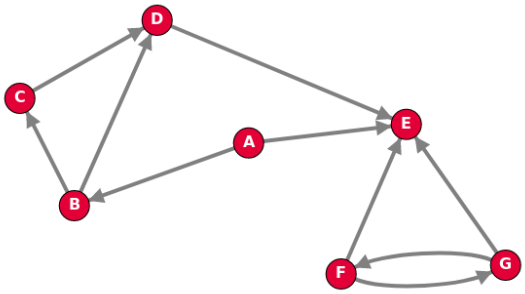


iii.

Dans le graphe orienté ci-dessous, déterminez :

- le degré entrant du sommet B et le degré sortant du sommet F

- le nombre de cycles
- le nombre de chemins simples d'origine le sommet F et d'extrémité le sommet E



Exercice 2

Nous aurons besoin d'exprimer la performance de nos algorithmes sur les graphes en fonction de paramètres de taille du graphe.

Soit un graphe constitué d'un ensemble V de sommets et E d'arcs, deux paramètres de taille sont importants :

- le nombre de sommets noté |V|
- le nombre d'arcs noté |E|

On considère dans cet exercice un graphe non orienté, dont tous les sommets peuvent être reliés deux à deux par un chemin. Un tel graphe est dit *connexe*. De plus on fixe comme contrainte qu'un seul arc peut relier deux sommets fixés (pas d'arcs parallèles).

Question

Dessinez un graphe non orienté connexe avec quatre sommets :

- qui comporte un nombre minimal d'arcs
- qui comporte un nombre maximal d'arcs

C- IMPLEMENTATION DES GRAPHES

1- Implémentation d'un graphe à l'aide d'une matrice d'adjacence

Représentation par matrice d'adjacence

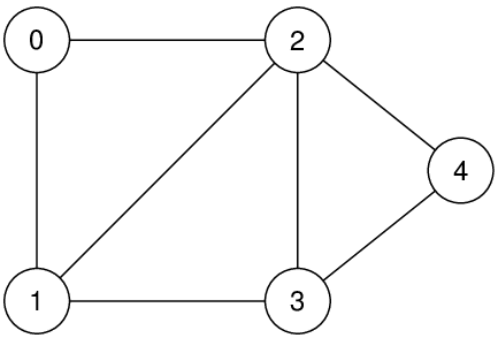
- On étiquette les sommets de 0 à $n-1$
- On représente chaque **arc** dans une **matrice d'adjacence,** c'est-à-dire un tableau à deux dimensions où on inscrit un 1 en ligne i et colonne j si l'arc i -> j est dans le graphe.

> **Matrice d'adjacence en Python**
>
> En Python, on peut représenter une matrice d'adjacences d'un graphe à n sommets par une liste de n listes de taille n. Si cette matrice est référencée par une variable mat et si 0 <= i < n et 0 <= j < n alors :
> - mat[i][j] vaut 1 si l'arc i -> j est dans le graphe
> - mat[i][j] vaut 0 si l'arc i -> j n'est pas dans le graphe

💡 *Pour un graphe non orienté, on ne distingue pas les arcs i -> j et j -> i donc s'il y a un 1 en ligne i et colonne j alors il y a un 1 en ligne j et colonne i. On dit que la matrice d'adjacence est symétrique.*

💡 *Pour un graphe pondéré on peut remplacer le 1 marquant la présence d'un arc par le poids de l'arc.*

Exemple avec un graphe non orienté



Le graphe non orienté ci-dessus peut être représenté par la matrice d'adjacence ci-dessous. La matrice est symétrique.

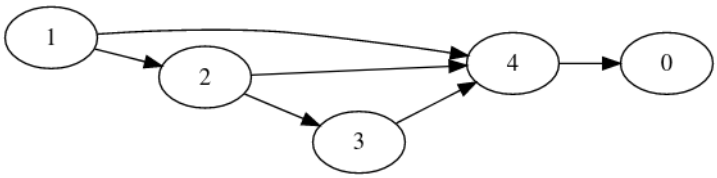
Matrice d'adjacence **arc entre deux sommets** **représentation en Python**

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

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

```
[[0, 1, 1, 0, 0],
 [1, 0, 1, 1, 0],
 [1, 1, 0, 1, 1],
 [0, 1, 1, 0, 1],
 [0, 0, 1, 1, 0]
 ]
```

Exemple avec un graphe orienté

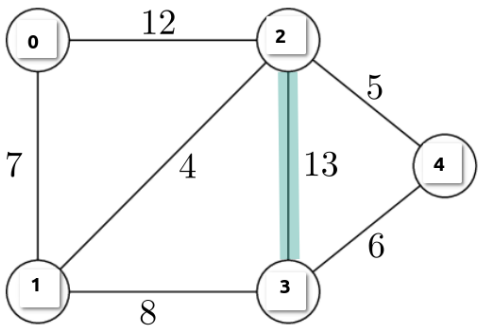


Le graphe orienté ci-dessus peut être représenté par la matrice d'adjacence ci-dessous représentée en Python. La matrice n'est pas symétrique

```
[[0, 0, 0, 0, 0],
 [0, 0, 1, 0, 1],
 [0, 0, 0, 1, 1],
 [0, 0, 0, 0, 1],
 [1, 0, 0, 0, 0]]
```

Exemple avec un graphe pondéré

Un exemple de graphe pondéré représenté par une matrice d'adjacence.



$$\begin{pmatrix} 0 & 7 & 12 & 0 & 0 \\ 7 & 0 & 4 & 8 & 0 \\ 12 & 4 & 0 & 13 & 5 \\ 0 & 8 & 13 & 0 & 6 \\ 0 & 0 & 5 & 6 & 0 \end{pmatrix}$$

Représentation en Python :

Script Python

```
[[0, 7, 12, 0, 0],
 [7, 0, 4, 8, 0],
 [12, 4, 0, 13, 5],
 [0, 8, 13, 0, 6],
 [0, 0, 5, 6, 0]]
```

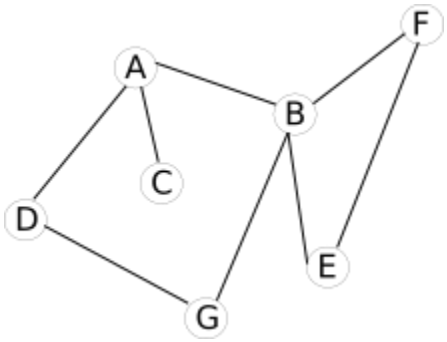
Une matrice est un tableau à double entrée :

$$A = \begin{pmatrix} 2 & 3 & 2 & 10 \\ 5 & 0 & 8 & 8 \\ 9 & 2 & 7 & 12 \\ 4 & 11 & 14 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

Exemple de matrice

La matrice A ci-dessus est constitué de 5 lignes et 4 colonnes. On appelle matrice carrée une matrice qui comporte le même nombre de lignes et de colonnes. Les matrices d'adjacences sont des matrices carrées.

Reprenons l'exemple du "graphe cartographie" :



Graphe cartographie

Voici la matrice d'adjacence de ce graphe :

0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	0	0	0	0	0
1	0	0	0	0	0	1
0	1	0	0	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0

Matrice d'adjacence du graphe cartographie

Comment construire une matrice d'adjacence ?

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.

	A	B	C	D	E	F	G
A	0	1	1	1	0	0	0
B	1	0	0	0	1	1	1
C	1	0	0	0	0	0	0
D	1	0	0	0	0	0	1
E	0	1	0	0	0	1	0
F	0	1	0	0	1	0	0
G	0	1	0	1	0	0	0

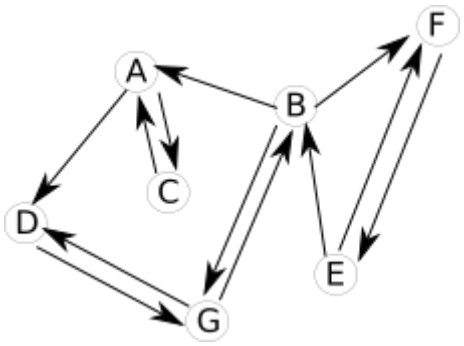
Matrice d'adjacence du graphe cartographie

- Il existe une arête entre le sommet E et le sommet F, nous avons donc placé un 1 à l'intersection de la ligne E et de la colonne F (il en est de même à l'intersection de la ligne F et de la colonne E)
- Il n'existe pas d'arête entre le sommet D et le sommet C, nous avons donc placé un 0 à l'intersection de la ligne D et de la colonne C (il en est de même à l'intersection de la ligne C et de la colonne D)

À faire vous-même 2

Vérifiez que la matrice d'adjacence proposée ci-dessus correspond bien au graphe "cartographie".

Il est aussi possible d'établir une matrice d'adjacence pour un graphe orienté. Le principe reste le même : si le sommet i (ligne) est lié au sommet j (colonne), nous avons un 1 à l'intersection (0 dans le cas contraire).



Graphe orienté cartographie

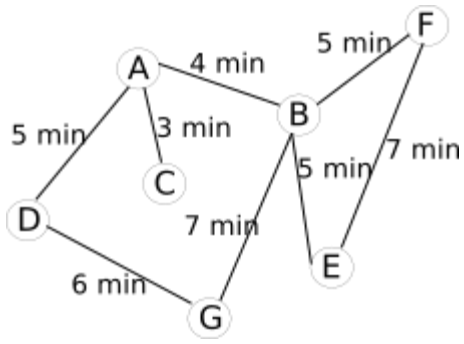
0	0	1	1	0	0	0
1	0	0	0	0	1	1
1	0	0	0	0	0	0
0	0	0	0	0	0	1
0	1	0	0	0	1	0
0	0	0	0	1	0	0
0	1	0	1	0	0	0

Matrice d'adjacence du graphe orienté cartographie

À faire vous-même 3

Vérifiez que la matrice d'adjacence proposée ci-dessus correspond bien au graphe orienté "cartographie".

Il est aussi possible d'utiliser une matrice d'adjacence pour implémenter un graphe pondéré : on remplace les 1 par les valeurs liées à chaque arc.



Graphe pondéré (minutes) cartographie

0	4	3	5	0	0	0
4	0	0	0	5	5	7
3	0	0	0	0	0	0
5	0	0	0	0	0	6
0	5	0	0	0	7	0
0	5	0	0	7	0	0
0	7	0	6	0	0	0

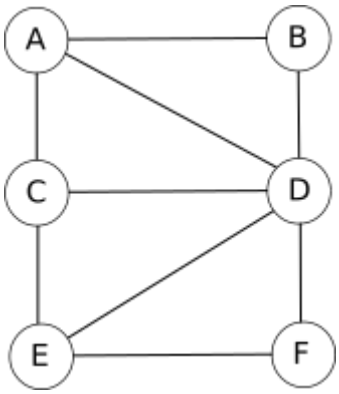
Matrice d'adjacence du graphe pondéré (minutes) cartographie

À faire vous-même 4

Vérifiez que la matrice d'adjacence proposée ci-dessus correspond bien au graphe pondéré (minutes) "cartographie".

À faire vous-même 5

Établissez la matrice d'adjacence du graphe ci-dessous.



Graphe réseau social

Il est assez simple d'utiliser les matrices d'adjacence en Python grâce aux tableaux de tableaux vus l'année dernière

```
#matrice d'ajacence pour le graphe cartographie
m = [[0, 1, 1, 1, 0, 0, 0],
     [1, 0, 0, 0, 1, 1, 1],
     [1, 0, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0, 1],
     [0, 1, 0, 0, 0, 1, 0],
     [0, 1, 0, 0, 1, 0, 0],
     [0, 1, 0, 1, 0, 0, 0]]
```

Exercice

Construire les graphes correspondants aux matrices d'adjacence suivantes:

Question 1

Matrice d'adjacence d'un graphe non orienté :

$$M_1 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Question 2

Matrice d'adjacence d'un graphe orienté :

$$M_2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Question 3

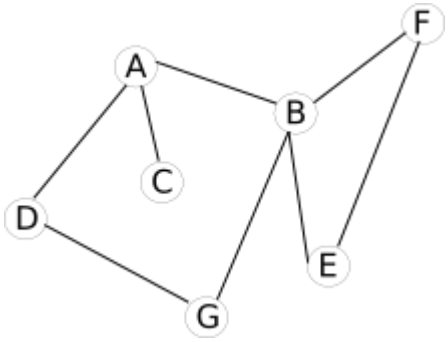
Matrice d'adjacence d'un graphe pondéré non orienté :

$$M_3 = \begin{pmatrix} 0 & 5 & 10 & 50 & 12 \\ 5 & 0 & 10 & 0 & 0 \\ 10 & 10 & 0 & 8 & 0 \\ 50 & 0 & 8 & 0 & 100 \\ 12 & 0 & 0 & 100 & 0 \end{pmatrix}$$

2- Implémentation d'un graphe à l'aide de listes d'adjacence

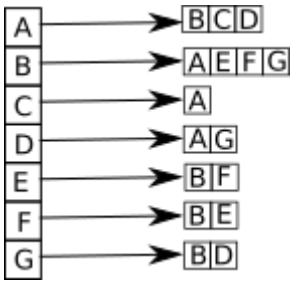
Pour commencer, on définit une liste des sommets du graphe. À chaque élément de cette liste, on associe une autre liste qui contient les sommets lié à cet élément :

Reprenons l'exemple du "graphe cartographie" :



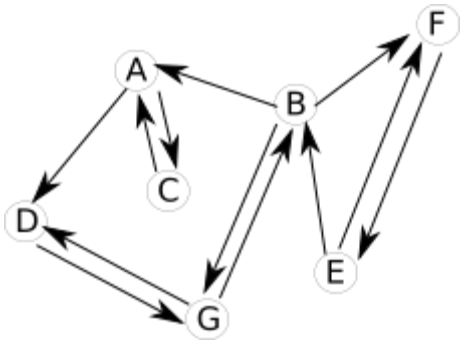
Graphe cartographie

Voici la liste d'adjacence de ce graphe :

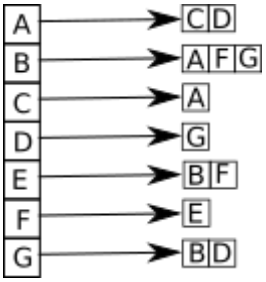


Liste d'adjacence du graphe cartographie

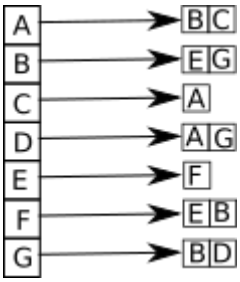
Pour les graphes orientés, il est nécessaire de définir 2 listes : la liste des successeurs et la liste des prédécesseurs. Soit un arc allant d'un sommet A vers un sommet B (flèche de A vers B). On dira que B est un successeur de A et que A est un prédécesseur de B.



Graphe orienté cartographie



Liste d'adjacence successeurs du graphe orienté cartographie



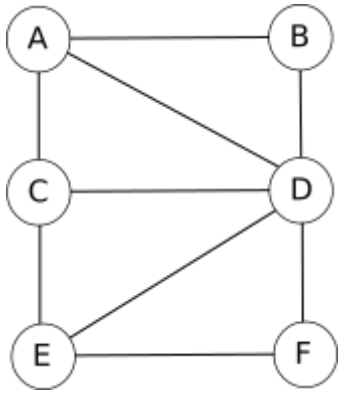
Liste d'adjacence prédécesseurs du graphe orienté cartographie

À faire vous-même 6

Vérifiez que les listes "successeurs-prédécesseurs" proposées ci-dessus correspondent bien au graphe orienté "cartographie".

À faire vous-même 7

Établissez la liste d'adjacence du graphe ci-dessous.



Graphe réseau social

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

```
#liste d'ajacence pour le graphe cartographie

l = {'A':('B','C','D'), 'B':('A', 'E', 'F', 'G'), 'C':('A'), 'D':('A', 'G'), 'E':('B', 'F'), 'F':('B', 'E'), 'G':('B'
, 'D')}
```

3- 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 (nous aurons très prochainement l'occasion d'étudier plusieurs de ces algorithmes).

II- Algorithmes sur les graphes

A- Algorithmes de parcours d'un graphe

Il est conseillé de relire au moins une fois le cours consacré aux graphes.

Nous allons commencer par nous intéresser aux algorithmes de parcours d'un graphe. L'idée du "parcours" 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 très utilisés : 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 d'abord

- le parcours en profondeur d'abord

B- 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. Un sommet u sera adjacent avec un sommet v si u et v sont reliés par une arête (on pourra aussi dire que u et v sont voisins) À chaque sommet u de ce graphe nous allons associer une couleur : blanc ou noir. Autrement dit, chaque sommet u possède un attribut couleur que l'on notera u.couleur, nous aurons u.couleur = blanc ou u.couleur = noir. Quelle est la signification de ces couleurs ?

- si u.couleur = blanc => u n'a pas encore été "découvert"
- si u.couleur = noir => u a été "découvert"

À faire vous-même 1

Étudiez cet algorithme :

```
VARIABLE
```

```
G : un graphe
```

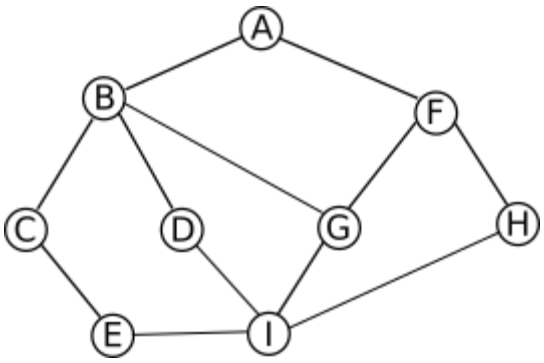
```
s : noeud (origine)
u : noeud
v : noeud
f : file (initialement vide)

//On part du principe que pour tout sommet u du graphe G, u.couleur = blanc à l'origine
DEBUT
s.couleur ← noir
enfiler (s,f)
tant que f non vide :
  u ← defiler(f)
  pour chaque sommet v adjacent au sommet u :
    si v.couleur n'est pas noir :
      v.couleur ← noir
      enfiler(v,f)
    fin si
  fin pour
fin tant que
FIN
```

À faire vous-même 2

Appliquez l'algorithme du parcours en largeur d'abord au graphe ci-dessous. Le 'point de départ' de notre parcours (le sommet s dans l'algorithme), sera le sommet A. Vous noterez les sommets atteints à chaque étape ainsi que les sommets présents dans la file f. Vous pourrez aussi, à chaque étape, donner les changements de couleur des sommets.

Si vous trouvez l'exercice ci-dessus trop difficile, vous pouvez aussi vérifier que la "découverte" des sommets peut se faire dans l'ordre suivant : A, B, F, C, D, G, H, E et I (ce n'est pas la seule solution possible)



Vous avez sans doute remarqué que dans le cas d'un parcours en largeur d'abord, on "découvre" d'abord tous les sommets situés à une distance k du sommet "origine" (sommet s) avant de commencer la découverte des sommets situés à une distance k+1 (on définit la distance comme étant le nombre d'arêtes à parcourir depuis A pour arriver à destination):

En effet, pour l'exemple du "À faire vous-même 2", nous avons bien :

sommets	A	B	F	C	D	G	H	E	I
distances depuis A	0	1	1	2	2	2	2	3	3

C- Le parcours en profondeur d'abord

On va ici retrouver le même système de couleur (blanc : sommet non visité, noir : sommet déjà visité)

À faire vous-même 3

Étudiez cet algorithme :

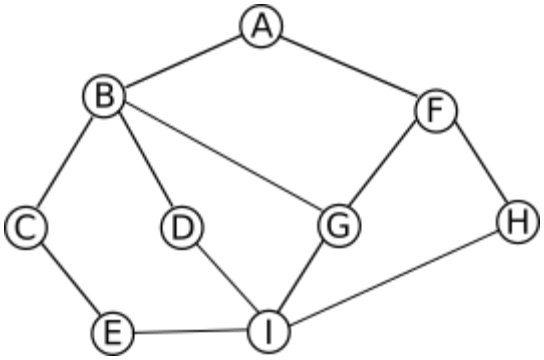
```
VARIABLE
G : un graphe
u : noeud
v : noeud
//On part du principe que pour tout sommet u du graphe G, u.couleur = blanc à l'origine
DEBUT
```

```
PARCOURS-PROFONDEUR(G,u) :
  u.couleur ← noir
  pour chaque sommet v adjacent au sommet u :
    si v.couleur n'est pas noir :
      PARCOURS-PROFONDEUR(G,v)
    fin si
  fin pour
FIN
```

Vous avez dû remarquer que le parcours en profondeur utilise une fonction récursive. J'attire votre attention sur l'extrême simplicité de cet algorithme (au niveau de sa conception), c'est souvent le cas avec les algorithmes récursifs.

À faire vous-même 4

Appliquez l'algorithme du parcours en profondeur d'abord au graphe ci-dessous. Vous pourrez vérifier que la "découverte" des sommets peut se faire dans l'ordre suivant : A, B, C, E, I, D, G, F et H (ce n'est pas la seule solution possible)



À faire vous-même 5

Comparez le résultat obtenu avec le parcours en largeur (A, B, F, C, D, G, H, E et I) et le résultat obtenu avec le parcours en profondeur (A, B, C, E, I, D, G, F et H)

Dans le cas du parcours en largeur on "découvrait" tous les sommets situés à une distance k de l'origine avant de s'intéresser aux sommets situés à une distance k+1 de l'origine. Dans le cas du parcours en profondeur, on va chercher à aller "le plus loin possible" dans le graphe : A -> B -> C -> E -> I -> D, quand on tombe sur "un cul-de-sac" (dans notre exemple, D est un "cul-de-sac", car une fois en D, on peut uniquement aller en B, or, B a déjà été découvert...), on revient "en arrière" (dans notre exemple, on repart de B pour aller explorer une autre branche : G -> F -> H)

À noter que l'utilisation d'un algorithme récursif n'est pas une obligation pour le parcours en profondeur :

À faire vous-même 6

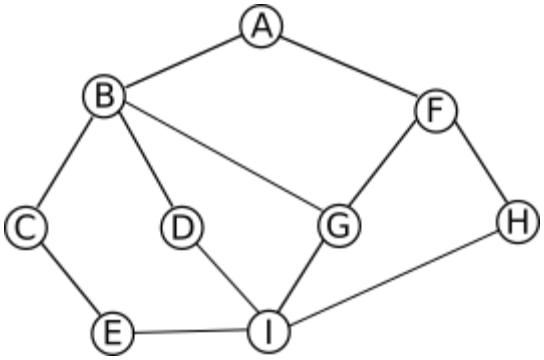
Étudiez cet algorithme :

```
VARIABLE
s : noeud (origine)
G : un graphe
u : noeud
v : noeud
p : pile (pile vide au départ)
//On part du principe que pour tout sommet u du graphe G, u.couleur = blanc à l'origine
DEBUT
s.couleur ← noir
piler(s,p)
tant que p n'est pas vide :
  u ← depiler(p)
  pour chaque sommet v adjacent au sommet u :
    si v.couleur n'est pas noir :
      v.couleur ← noir
      piler(v,p)
    fin si
  fin pour
fin tant que
FIN
```

Vous avez sans doute remarqué que la version "non récursive" (on dit "itérative") de l'algorithme du parcours en profondeur ressemble beaucoup à l'algorithme du parcours en largeur, on a juste remplacé la file par une pile.

À faire vous-même 7

Appliquez l'algorithme du parcours en profondeur d'abord au graphe ci-dessous. Vérifiez que l'on obtient bien un parcours en profondeur.



2- Cycle dans les graphes

Voici un rappel de 2 définitions vues précédemment :

- Une chaine est une suite d'arêtes consécutives dans un graphe, un peu comme si on se promenait sur le graphe. On la désigne par les lettres des sommets qu'elle comporte. On utilise le terme de chaine pour les graphes non orientés et le terme de chemin pour les graphes orientés.
- Un cycle est une chaine qui commence et se termine au même sommet.

Pour différentes raisons, il peut être intéressant de détecter la présence d'un ou plusieurs cycles dans un graphe (par exemple pour savoir s'il est possible d'effectuer un parcours qui revient à son point de départ sans être obligé de faire demi-tour).

Nous allons étudier un algorithme qui permet de "détecter" la présence d'au moins un cycle dans un graphe :

À faire vous-même 8

Étudiez cet algorithme. Que se passe-t-il quand le graphe G contient au moins un cycle ? Que se passe-t-il quand le graphe G ne contient aucun cycle :

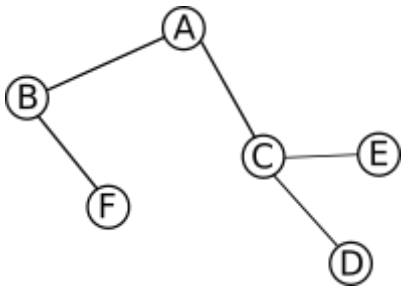
```
VARIABLE
s : noeud (noeud quelconque)
G : un graphe
u : noeud
v : noeud
p : pile (vide au départ)
//On part du principe que pour tout sommet u du graphe G, u.couleur = blanc à l'origine
DEBUT
CYCLE():
  piler(s,p)
  tant que p n'est pas vide :
    u ← depiler(p)
```

```
    pour chaque sommet v adjacent au sommet u :
```

```
      si v.couleur n'est pas noir :
        piler(v,p)
      fin si
    fin pour
    si u est noir :
      renvoie Vrai
    sinon :
      u.couleur ← noir
    fin si
  fin tant que
  renvoie Faux
FIN
```

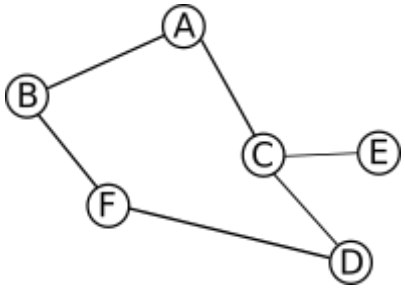
À faire vous-même 9

Appliquez l'algorithme de détection d'un cycle au graphe ci-dessous (vous partirez du sommet de votre choix).



À faire vous-même 10

Appliquez l'algorithme de détection d'un cycle au graphe ci-dessous (vous partirez du sommet de votre choix).



3- Chercher une chaine dans un graphe

Nous allons maintenant nous intéresser à un algorithme qui permet de trouver une chaine entre 2 sommets (sommet de départ et sommet d'arrivée). Les algorithmes de ce type ont une grande importance et sont très souvent utilisés (voir plus bas (après le "À faire vous-même 12") pour plus d'explications).

À faire vous-même 11

Étudiez cet algorithme

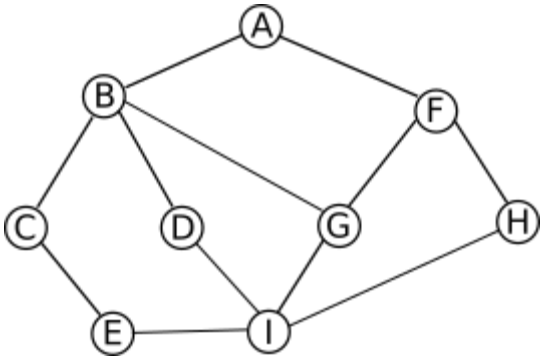
```
VARIABLE
G : un graphe
start : noeud (noeud de départ)
end : noeud (noeud d'arrivé)
u : noeud
chaine : ensemble de noeuds (initialement vide)

DEBUT
TROUVE-CHAINE(G, start, end, chaine):
  chaine = chaine U start //le symbol U signifie union, il permet d'ajouter le noeud start à l'ensemble chaine
  si start est identique à end :
    renvoie chaine
  fin si
  pour chaque sommet u adjacent au sommet start :
    si u n'appartient pas à chaine :
                nchemin = TROUVE-CHAINE(G, u, end, chaine)
      si nchemin non vide :
        renvoie nchemin
      fin si
    fin si
  fin pour
renvoie NIL
FIN
```

Vous noterez que l'algorithme ci-dessus est basé sur un parcours en profondeur d'abord.

À faire vous-même 12

Appliquez l'algorithme permettant de trouver une chaine entre un nœud de départ (Start) et un nœud d'arrivée (end) au graphe ci-dessous (vous choisirez les nœuds de départ et d'arrivée de votre choix).



Il est important de noter que dans la plupart des cas, les algorithmes de recherche de chaine (ou de chemin), travaillent sur des graphes pondérés (par exemple pour rechercher la route entre un point de départ et un point d'arrivée dans un logiciel de cartographie). Ces algorithmes recherchent aussi souvent les chemins les plus courts (logiciels de cartographie). On peut citer l'algorithme de Dijkstra ou encore l'algorithme de Bellman-Ford qui recherchent le chemin le plus court entre un nœud de départ et un nœud d'arrivée dans un graphe pondéré. Si ce sujet vous intéresse, vous pouvez visionner cette vidéo qui explique le principe de fonctionnement de l'algorithme de Dijkstra.