Les graphes

Capacités attendues

- ✓ Modéliser des situations sous forme de graphes.
- ✓ Vocabulaire : sommets, arcs, arêtes, graphes orientés ou non orientés.
- ✓ Implémentation d'un graphe : matrice d'adjacence, liste de successeurs/de prédécesseurs.
- ✓ Passer d'une représentation à une autre.
- ✓ 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.

1 Généralités

Définitions

En informatique, un graphe est une structure de données relationnelle composée

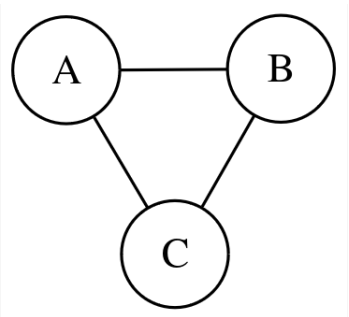
-
-

Les **voisins** d'un sommet sont
Deux sommets sont dit s'ils sont reliés par une arête.

Graphes non-orientés

Un graphe non-orienté est un graphe dont les arêtes n'ont **pas de sens**.
Les sommets en lien direct avec un même sommet sont appelés ses .

Exemple :



B et C sont les de A.

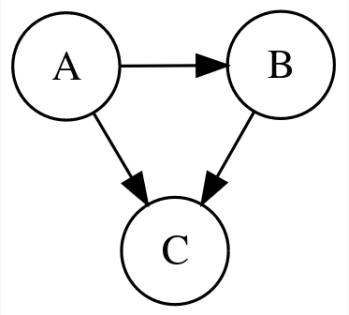
Graphe orienté

Un graphe orienté est un graphe dont les arêtes sont appellées ont un **sens** et sont représentées par des flèches.

Les sommets en lien direct avec un sommet sont appelés ses ou ses selon le sens de la flèche.

Exemple :

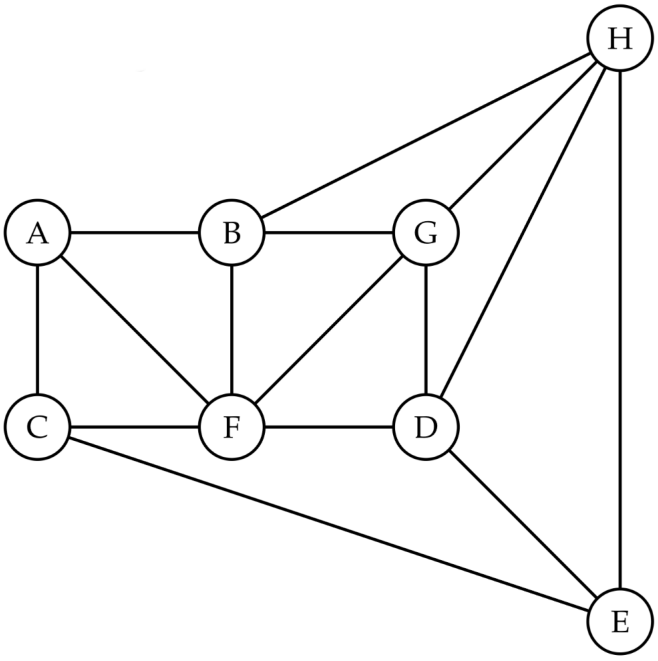
Imaginons un site web composé de 3 pages nommées A, B et C et représenté ci-dessous :



- sur A sont présents des liens vers B et C ;
- B et C sont les de A ;
- sur B est présent un lien vers C ;
- C est de A et A est de B.

Exercice 1

On considère le graphe représenté ci-dessous.



1. Quels sont les voisins du sommet A ? du sommet B ?
2. Quel type de variable permettrait de représenter les arêtes ?

3. On appelle **degré d'un sommet** son nombre de voisins.

   (a) Quels sont les sommets de degré minimal ? de degré maximal ?

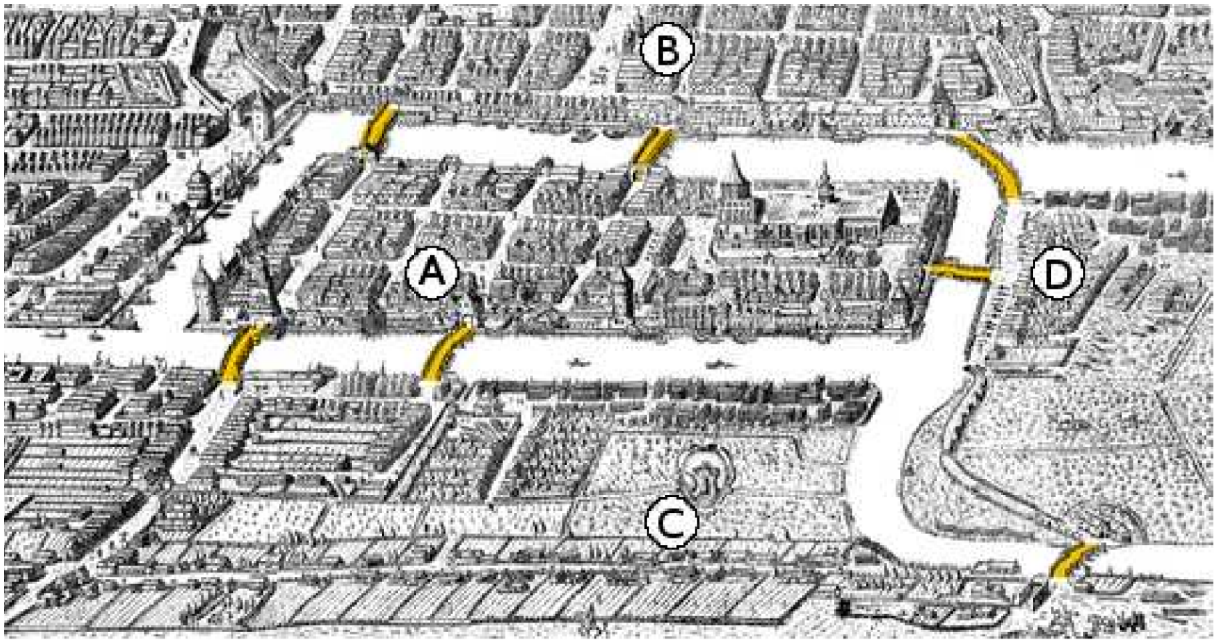
   (b) Calculer le degré moyen de ce graphe.

4. Déterminer le nombre de chemins de longueur 3 reliant le sommet A au sommet E.

Exercice 2

Dans la ville de Königsberg, on cherche à déterminer le parcours d'une promenade qui permette de passer une fois et une seule sur chacun des ponts de la ville.

On fournit ci-dessous une carte de la ville



1. Montrer que le problème peut être modélisé par un graphe qu'on dessinera.

2. Ce problème admet-il une solution ?

Exercice 3

Le graphe complet à n sommets est le graphe où chacun de ses n sommets est relié à tous les autres.

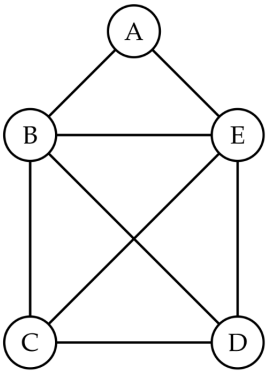
On dit qu'un graphe est planaire s'il est possible de le représenter sans que deux arêtes ne se coupent.

1. Montrer que le graphe complet à 4 sommets est planaire en le dessinant.

2. Le graphe complet à 5 sommets est-il planaire ?

Exercice 4

Sur le graphe ci-dessous, déterminer un parcours eulérien, c'est-à-dire un ordre de parcours des sommets de manière à passer sur toutes les arêtes exactement une fois.



Exercice 5

Citer quelques exemples de situation qu'on peut représenter par des graphes.

2 Liste d'adjacence

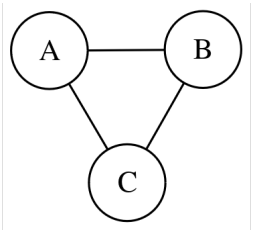
Pour manipuler les graphes, il faut leur associer une représentation mathématique permettant par la suite de les implémenter.

La liste d'adjacence est **une** façon de représenter un graphe.

> **Définition**
> On peut représenter un graphe en établissant, pour chacun de ses sommets, la **liste des sommets** auxquels il est **relié**.

Exemple :

Reprenons le graphe déjà vu précédemment.



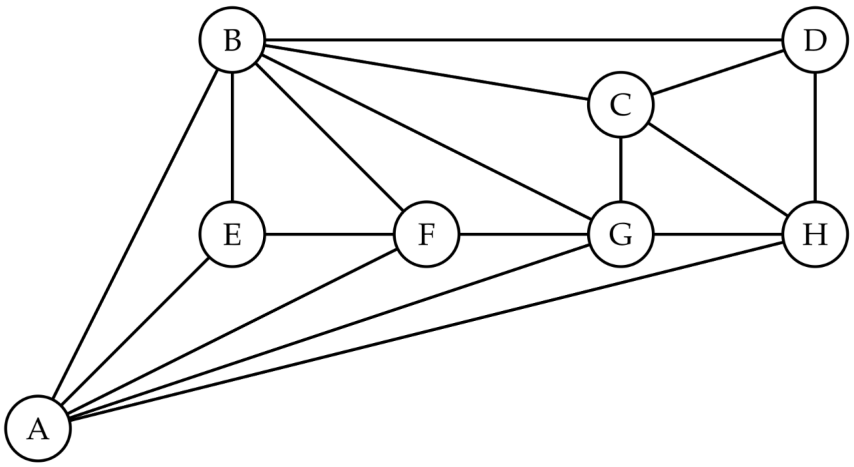
Sa liste d'adjacence est l'ensemble des voisins de chaque sommet :

- "A" : "B", "C"
- "B" : "A", "C"
- "C" : "A", "B"

Exercice 6

1. Télécharger depuis notre site et décompresser l'archive graphes.zip.
2. Le fichier `exemple.py` permet d'afficher un graphe à partir d'un dictionnaire résumant sa liste d'adjacence.

   On demande de définir le dictionnaire permettant d'obtenir le graphe ci-dessous. (Les sommets sont à déplacer à l'affichage.)



Exercice 7

Dans le fichier `graphal.py`, compléter la classe `GraphAL` qui implémente une structure de graphe basée sur sa liste d'adjacence.

On y définit les méthodes suivantes :

- le constructeur qui prend en argument un dictionnaire représentant la liste d'adjacence ;
- les méthodes `degre_min`, `degre_max`, `degre_moyen` qui prend en argument l'étiquette d'un sommet et renvoie son degré minimal, maximal et moyen respectivement.

3 Matrice d'adjacence

La matrice d'adjacence est **une** façon de représenter un graphe.

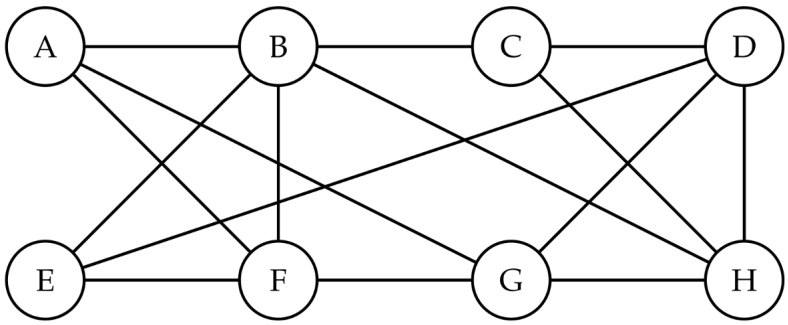
Définition
Étant donné un graphe non orienté comptant n sommets, on numérote de 1 à n les sommets (dans un ordre arbitraire, par exemple dans l'ordre lexicographique des étiquettes).
La matrice d'adjacence associée au graphe (et à la numérotation choisie) est le tableau à double entrée dont le coefficient à l'intersection de la i–ème ligne et de la j–ième colonne est égale à :

-
-

Propriété
La matrice d'adjacence d'un graphe non-orienté est forcément par rapport à sa diagonale.

Exercice 8

1. Compléter la matrice d'adjacence associé au graphe ci-dessous, sachant que les étiquettes sont numérotées dans l'ordre alphabétique.



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

2. Comment calculer le degré d'un sommet à l'aide de la matrice de l'adjacence ?

Exercice 9

Quelle différence observe-t-on entre l'utilisation d'une matrice et d'une liste d'adjacence ?

Exercice 10

On considère la matrice d'adjacence suivante.

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

Représenter le graphe associé à cette matrice en prenant pour étiquettes les premières lettres de l'alphabet.

Exercice 11

Quelle est la matrice d'adjacence d'un graphe complet à n sommets ?

Exercice 12

Dans le fichier `grapham.py`, compléter la classe `GraphAM` qui implémente une structure de graphe basée sur sa matrice d'adjacence.

On y définit les méthodes suivantes :

- le constructeur qui prend en argument un tableau de tableaux représentant la matrice d'adjacence ;
- les méthodes `degre_min`, `degre_max`, `degre_moyen` qui prend en argument le numéro d'un sommet et renvoie son degré minimal, maximal et moyen respectivement.

4 Passer d'une représentation à une autre

On choisit la façon de représenter un graphe en fonction du contexte. On peut aisément passer d'une liste à une matrice et réciproquement.

Exercice 13

Il s'agit de compléter le code fourni dans le fichier `list_matrix.py` issu de l'archive déjà téléchargée et décompressée précédemment.

Un script de test est proposé avec les affichages appropriés.

1. Compléter la fonction `get_adjacency_matrix` qui prend en argument un dictionnaire `adj_list` représentant la liste d'adjacence d'un graphe et renvoie la matrice d'adjacence associée.
2. Compléter la fonction `get_adjacency_list` qui prend en argument une matrice d'adjacence `adj_matrix` d'un graphe (un tableau de tableaux) et renvoie un dictionnaire `adj_list` représentant sa liste d'adjacence.

5 Les parcours sur les graphes

5.1 Parcours en profondeur

Définition récursive

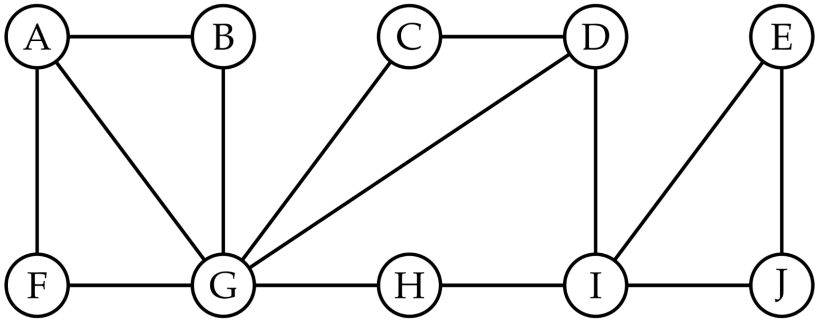
La parcours en profondeur d'abord *(Depth-First Search)* d'un graphe consiste à explorer récursivement, en partant d'un sommet quelconque, les voisins de chacun des sommets. **Cependant, afin de garantir que le parcours du graphe se termine, il est nécessaire de marquer chacun des sommets lors de leur visite.** Ainsi, seuls les voisins qui n'ont pas encore été marqués sont explorés récursivement.
On donne ci-dessous une implémentation en langage naturel du parcours en profondeur d'abord.

```
fonction parcours_profondeur(sommet):
    marquer sommet
    pour chaque voisin de sommet :
        si voisin n'a pas été marqué :
            parcours_profondeur(voisin)   # appel récursif
```

Dans d'un tel parcours, l'ordre dans lequel sont explorés les sommets dépend de la manière dont sont ordonnés les voisins de chaque sommet.

Exercice 14

On considère le graphe représenté ci-dessous.



1. On suppose que les listes d'adjacence de chacun des sommets sont ordonnés dans l'ordre alphabétique.
   (a) Si les sommets n'étaient pas marqués lors du déroulement de l'algorithme de parcours en profondeur d'abord, quels seraient les sommets visités en partant du sommet A ?
   (b) Donner l'ordre de visite des sommets lors du parcours en profondeur d'abord, en partant :
      - du sommet A :
      - du sommet E :
      - du sommet H :
2. On suppose maintenant que les listes d'adjacence sont triées dans l'ordre lexicographique décroissant.

   Donner l'ordre de visite des sommets lors du parcours en profondeur d'abord, en partant :
   - du sommet A :
   - du sommet E :
   - du sommet H :

Exercice 15

Dans la classe `GraphAL` du fichier `graphal.py`, écrire une méthode récursive

`parcours_prof(self, sommet, deja_vus=[])`

qui, en effectuant un parcours en profondeur d'abord, collecte les étiquettes des sommets d'un graphe `adj_lst` en partant du sommet `sommet`.

L'argument `deja_vus` est une liste initialement vide qui sert à marquer les sommets (en ajoutant leur étiquette à la fin de cette liste). La syntaxe « `sommet in deja_vus` » permet ainsi de tester si le sommet `sommet` a été marqué.

On donne ci-dessous l'instruction permettant de déclarer la liste d'adjacence du graphe considéré.

```
adj_list = { "A" : list("BFG"),
             "B" : list("AG"),
             "C" : list("DG"),
             "D" : list("CGI"),
             "E" : list("IJ"),
             "F" : list("AG"),
             "G" : list("ABCDFH"),
             "H" : list("GI"),
             "I" : list("DEHJ"),
             "J" : list("EI")}
```

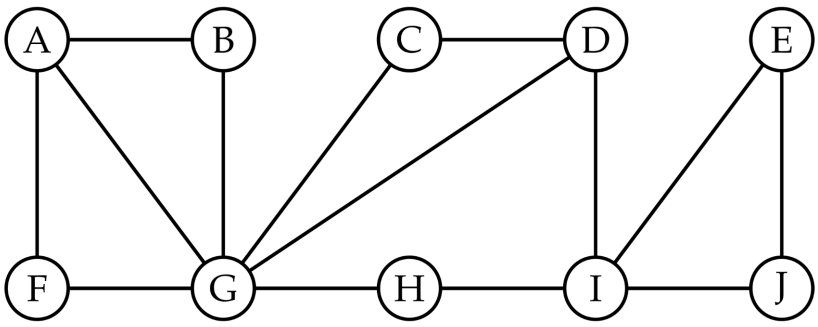
Définition itérative

Le parcours en profondeur d'abord d'un graphe peut être programmé de manière purement itérative en utilisant une pile : on remplace les appels récursifs par l'empilement des sommets voisins qui doivent être explorés. On donne ci-dessous une implémentation en langage naturel.

```
fonction parcours_profondeur_it(sommet_départ):
    marquer sommet_départ
    pile = nouvelle pile
    empiler sommet_départ
    tant que la pile n'est pas vide:
        sommet = dépiler
        pour chaque voisin de sommet :
            si voisin n'est pas marqué :
                marquer voisin
                empiler voisin
```

Exercice 16

On considère le graphe représenté ci-dessous.



1. On suppose que les listes d'adjacence de chacun des sommets sont ordonnés dans l'ordre alphabétique. Donner l'ordre de dépilage des sommets lors du parcours en profondeur d'abord itératif, en partant :
   - du sommet A :
   - du sommet E :
   - du sommet H :
2. Dans la classe `GraphAL` du fichier `graphal.py`, écrire la méthode itérative

   `parcours_prof_it(self, adj_lst, sommet)`

   qui, en effectuant un parcours en profondeur d'abord, collecte les étiquettes des sommets d'un graphe `adj_lst` en partant du sommet `sommet`. Elle devra renvoyer la liste des sommets visités.

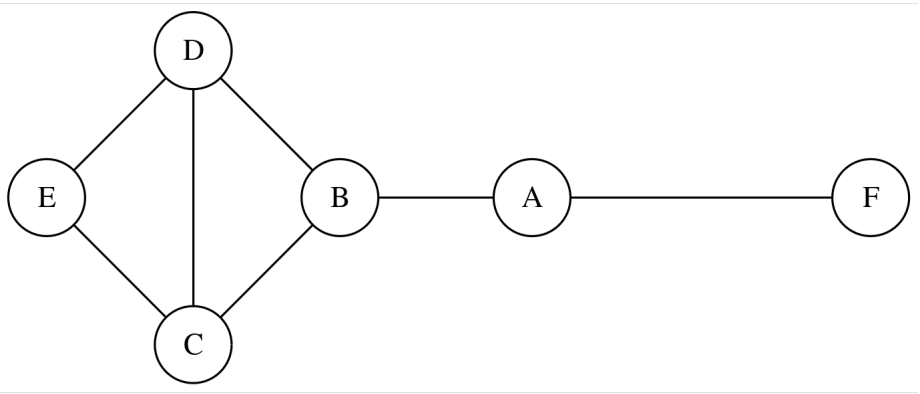
Recherche de chemin

Pour chercher un chemin entre deux sommets, on peut s'inspirer du parcours en profondeur.
$\rightarrow$ Que faut-il adapter ?

Recherche de cycle

Dans un graphe non orienté, un cycle est une suite d'arêtes consécutives **distinctes** dont les deux sommets extrémités sont identiques.
Pour obtenir tous les cycles d'un graphe, on peut s'inspirer de l'algorithme de recherche d'un chemin.

$\rightarrow$ Identifier les cycles sur le graphe suivant.



Exercice 17

En utilisant un parcours en profondeur (version récursive ou itérative), écrire une fonction `has_cycle(adj_list)` qui prend en argument la liste d'adjacence `adj_list` d'un graphe (sous la forme d'un dictionnaire) et renvoie la valeur `True` si celui possède un cycle, `False` sinon.

5.2 Parcours en largeur

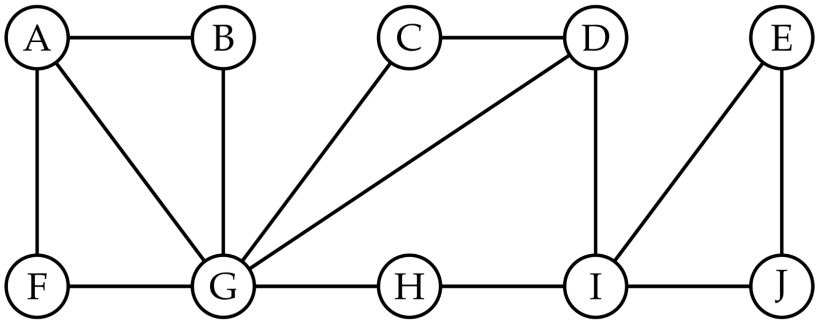
Définition

Le parcours en largeur d'abord d'un graphe s'obtient en remplaçant la pile dans l'algorithme de parcours en profondeur d'abord par une file.

```
fonction parcours_largeur(sommet_départ):
    marquer sommet_départ
    file = nouvelle file
    enfiler sommet_départ
    tant que la file n'est pas vide:
        sommet = défiler
        pour chaque voisin de sommet :
            si voisin n'est pas marqué :
                marquer voisin
                enfiler voisin
```

Exercice 18

On considère le graphe représenté ci-dessous.



1. On suppose que les listes d'adjacence de chacun des sommets sont ordonnés dans l'ordre alphabétique. Donner l'ordre de visite des sommets lors du parcours en largeur d'abord, en partant :
   - du sommet A :
   - du sommet E :
   - du sommet H :
2. Implémenter la fonction `parcours_largeur(adj_lst, sommet)` qui, en effectuant un parcours en largeur d'abord, collecte les étiquettes des sommets d'un graphe `adj_lst` en partant du sommet `sommet`.

   Le graphe `adj_lst` sera présenté sous la forme d'une liste d'adjacence (dictionnaire).
3. Vérifier la validité de la fonction sur les exemples de la question 1.