Année scolaire 2025 – 2026

Lycée Victor Hugo - Colomiers

Terminale NSI

SD
Structure de données

Thomas Ricart
ricart.lvh@laposte.net

24 mars 2025

SD01 – La Programmation Orientée Objet

1. Introduction

Définition 1: Programmation Orientée Objet

La **Programmation Orientée Objet** ou **POO** est un paradigme de programmation informatique. Il consiste en la définition et l'interaction de briques logicielles appellées `objets`.
Chaque objet est une **instance** de **classes** qui définissent le comportement et les propriétés des objets.

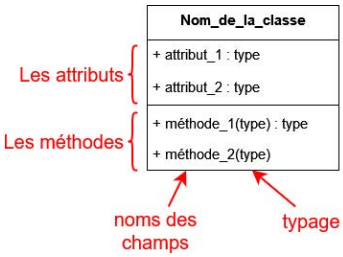
Exemple 1:

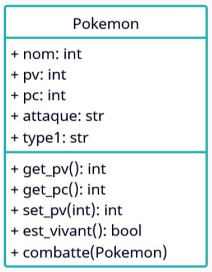
- `Pokemon` est une classe; c'est un concept. Pour le moment aucun pokemon n'existe mais juste l'idée;
- `pikachu` est un objet. C'est un pokemon particulier. On dit que c'est une instance de classe;
- une classe possède un nom et des caractéristiques (on parle d'**attributs**);
- une classe possède des fonctionnalités (on parle de **méthodes**)


```
Classe = Type d'objet
Objet = Instance de classe
```

```
Nom de la classe: Pokemon
******************************
Attributs:
    - nom (str)
    - points de vie: PV (int)
    - points de combat: PC (int)
    - attaque (str)
    - type (str)
******************************
Méthodes:
    - connaitre les PV du pokemon
    - connaitre les PC du pokemon
    - modifier les PV du pokemon
    - savoir si le pokemon est toujours vivant
```

Propriété 1: Diagramme de classe UML





- `Pokemon` est le nom de la classe
- `Pikachu` est une instance de la classe `Pokemon`
- `pikachu` est la valeur de l'attribut `nom` de l'instance `Pikachu`
- `60` est la valeur de l'attribut `pv` de l'instance `Pikachu`

Propriété 2: En python tout est objet

En python, toutes les variables sont déjà des objets. Pour vous en convaincre: `dir(list)`
Dans la liste des méthodes associées à la classe `list`, on retrouve les méthodes connues (`append`, `pop`...) mais également des méthodes spéciales (ou **méthodes magiques**) qui sont entourées de double underscores (`__init__`, `__len__`, `__add__`, etc.). Ces méthodes permettent de définir le comportement des objets de la classe.

2. Construire une classe

Définition 2: Une classe

Une classe est au minimum constituée d'un constructeur de classe.

Code Python

```
class Point:                            # Définition du nom de la classe
    def __init__(self, abs, ord):       # Constructeur de classe
        self.x = abs                    # Attribut x
        self.y = ord                    # Attribut y
```

- `class Point`: c'est la porte d'entrée de la classe;
- `def __init__(self, abs, ord)`: c'est la méthode qui est appelée au moment de la création d'une instance de la classe `Point`;
- `abs` et `ord` sont des paramètre à entrer au moment de créer l'instance;
- `self`: c'est l'objet en lui même. Pour le moment, il n'a pas de nom puisqu'on crée la classe. Ensuite `self` sera remplacé par le nom de l'instance de la classe;
- `self.x`: on crée un attribut de classe `x` qui prend l'abscisse pour valeur;
- `self.y`: on crée un attribut de classe `y` qui prend l'ordonnée pour valeur.

Définition 3: Se servir d'une classe

Pour se servir d'une classe, il suffit maintenant de créer un objet (une instance) de la classe `Point`

Code Python

```
A = Point(1, 7)     # A est une instance de la classe Point
B = Point(0, 4)     # B est également une instance de la classe Point
```

Au moment de la création de l'instance A, le constructeur est exécuté. Dans toute la suite, `self` sera donc remplacé par `A`.

Code Python

```
assert A.x == 1     # On récupère l'attribut x de l'instance A.
assert A.y == 7     # On récupère l'attribut y de l'instance A.
```

Définition 4: Accesseurs et mutateurs

En général on s'interdit d'accéder aux attributs de l'instance de la classe directement. A la place on crée des méthodes permettant d'accéder à ces attributs:

- Accesseurs (getters): permet de récupérer la valeur de l'attribut;
- Mutateurs (setters): permet de modifier la valeur de l'attribut.

Code Python

```
class Point:                                # Définition du nom de la classe
        def __init__(self, abs, ord):       # Constructeur de classe
            self.x = abs                    # Attribut x
            self.y = ord                    # Attribut y
        def get_coord(self):
            ''' Renvoie les coordonnées du point sous forme de tuple '''
            return (self.x, self.y)
        def set_coord(self, coord:tuple):
            ''' Modifie les coordonnées du point '''
            self.x, self.y = coord
```

Affichage

Python permet d'afficher de deux manières: soit via la fonctionnalité `print` soit directement dans la console. Derrière ces fonctionnalités, il se cache deux méthodes:

- Pour la fonction `print`, c'est la méthode `__str__`
- Pour l'affichage dans la console c'est la méthode `__repr__`

Par défaut ces méthodes sont crées lors de la création de la classe `Point` mais ne donnent rien d'intéressant:

```
>>> print(A)
<__main__.Point object at 0x00000233824A4D08>
```

On va donc surcharger (modifier) ces deux méthodes

Code Python

```python
def __str__(self):
    ''' Surcharge la fonction print. Affichage 'Coordonnées = (abs;ord)'''
    return f'Coordonnées = ({self.x};{self.y})'
def __repr__(self)
    ''' Surcharge __repr__. Affichage (abs;ord)'''
    return f'({self.x};{self.y})
```

Définition 5: D'autres méthodes intéressantes

Dans le même ordre d'idée, il est possible de redéfinir des méthodes comme:

- `__eq__(self, objet)` qui renvoie `True` si `self == objet`;
- `__gt__(self, objet)` qui renvoie `True` si `self > objet`;
- `__lt__(self, objet)` qui renvoie `True` si `self < objet`;
- `__add__(self, objet)` qui renvoie `self + objet`;
- `__mul__(self, objet)` qui renvoie `self * objet`;

Propriété 3: Utilisation des méthodes

Code Python

```python
>>> A = Point(1, 6)
>>> B = Point(2, 7)
>>> C = Point(1, 6)
>>> B.set_coord((4, 8))
>>> B.get_coord()
(4, 8)
>>> print(B)
Coordonnées = (4;8)
>>> B
(4;8)
>>> A == B
False
>>> A == C
True
```

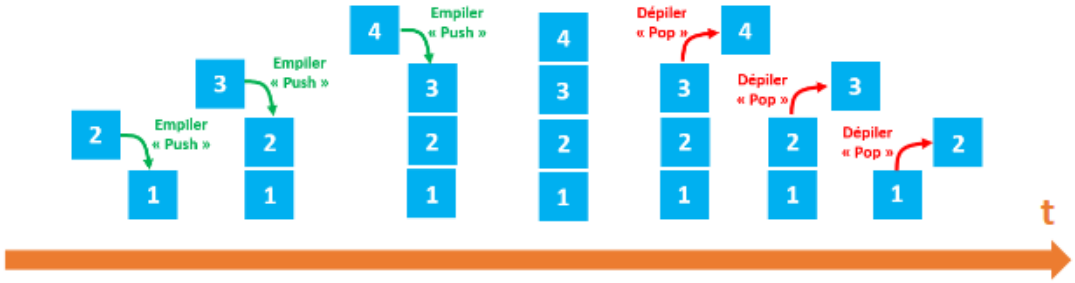
SD02 – Listes Piles Files

1. Les piles

Définition 1: Les piles – Stack

Une **pile** est un **Type Abstrait de Données (TAD)** qui fonctionne selon le principe du *dernier arrivé, premier sorti.* (Last In, First Out: LIFO). C'est le principe d'une pile d'assiettes: la dernière ajoutée au sommet de la pile sera la première à être utilisée.

Exemple 1:




```
class Pile:
*****************
pile (list)
nb_elements (int)
taille_pile (int)
*****************
creer_pile(pile, taille_pile)
empiler(pile, element)
depiler(pile) -> element
est_vide(pile) -> bool
est_pleine(pile) -> bool
```

Exercice 1: Implémenter en python la classe `Pile`

Code Python

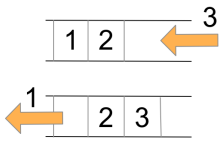
```python
class Pile:
    def __init__(self, taille = None):
        ''' Si taille est None, la pile n'a pas de taille maximum '''
        ...

    def est_vide(self):
        ''' Renvoie True si la pile est vide '''
    def est_pleine(self):
        ''' Renvoie True si la pile est pleine '''
    def empiler(self, element):
        ''' Empile element si non pleine '''
    def depiler(self):
        ''' Renvoie l'élément dépilé si non vide '''
```

2. Les files

Définition 2: Les files – Queue

Une **pile** est un **Type Abstrait de Données (TAD)** qui fonctionne selon le principe du *premier arrivé, premier sorti.* (First In, First Out: FIFO). C'est le principe d'une file d'attente. On utilise ce type de données pour par exemple la gestion des stocks dans un entrepôt (pour des raison de date de consommation) ou pour la file d'attente d'impression.



Exercice 2:

```
class File:
*****************
file (list)
nb_elements (int)
taille_file (int)
*****************
creer_file(file, taille_file)
enfiler(file, element)
defiler(file) -> element
est_vide(file) -> bool
est_pleine(pile) -> bool
```

Code Python

```python
class File:
    def __init__(self, taille = None):
        ''' Si taille est None, la file n'a pas de taille maximum '''
        ...

    def est_vide(self):
        ''' Renvoie True si la file est vide '''
    def est_pleine(self):
        ''' Renvoie True si la file est pleine '''
    def enfiler(self, element):
        ''' Enfile element si non pleine '''
    def defiler(self):
        ''' Renvoie l'élément défilé si non vide '''
```

3. Implémentation du TAD à partir d'une classe Maillon

Définition 3: Le maillon

Il est possible de définir une structure de données abstraire sans connaitre l'intégralité de la structure.

Exemple 2: La file chainée

```
class Maillon:
*************************
nom (int)
suivant (Maillon ou None)
*************************
creer_maillon(maillon)
get_suivant(maillon)
get_nom(maillon)
set_suivant(maillon, maillon_suivant)
afficher_maillon(maillon)

class File_chainee:
*************************
premier_element (Maillon ou None)
*************************
creer_file_chainee(file_chainee)
est_vide(file_chainee)
enfiler(file_chainee, maillon)
defiler(file_chainee)
afficher_file(file_chainee)
```

Code Python

```python
class Maillon:
    def __init__(self, valeur, suivant = None):
        self.valeur = valeur
        self.suivant = suivant          # de type Maillon

class File_chainee:
    def __init__(self, tete = None):
        self.tete = tete                # de type Maillon
    def est_vide(self):
        return self.tete == None
    def enfiler(self, valeur):
        maillon_a_inserer = Maillon(valeur)
        ...
    def defiler(self):
        ...
    def __str__(self):
        if not self.tete:
            return f'None'
        liste = []
        courant = self.tete
        while courant:
            liste.append(str(courant.valeur))
            courant = courant.suivant
        elements = "<-".join(liste)
        return elements
```

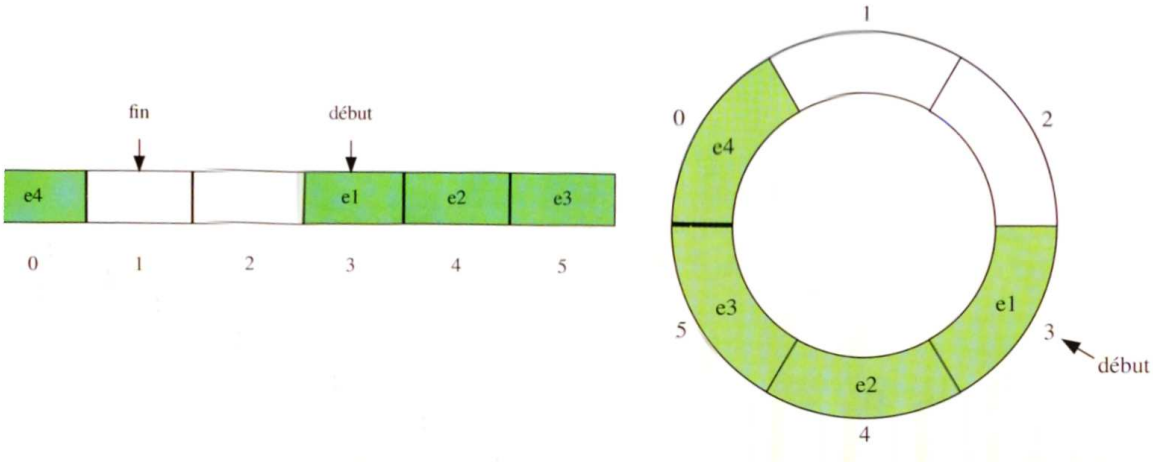
4. Implémentation d'une file par un tableau circulaire

L'implémentation d'une file est couteuse si la file est grande puisque qu'à chaque retrait, il faut supprimer un élément et décaler tous les autres d'un indice. Une implémentation plus efficace consiste à considérer le tableau comme un anneau dont la file occupe les éléments de l'indice `debut` à l'indice `fin - 1`.

La file a donc ici une taille maximum.

- Lorsqu'on enfile, on ajoute l'élément à l'indice `fin`
- Lorsqu'on doit enfiler un élément et qu'on a atteint la fin du tableau, on continue à le remplir au début si le tableau n'est pas plein.
- Lorsqu'on retire un élément, on retire celui de l'indice `debut` et lorsqu'on arrive à la fin du tableau au remet `debut` à 0.
- Les indice `debut` et `fin` vont donc varier au fur et à mesure qu'on ajoute ou qu'on supprimer des éléments.

En réalité, les éléments ne sont pas vraiment supprimés, ce sont les indices `debut` et `fin` qui délimitent la file.

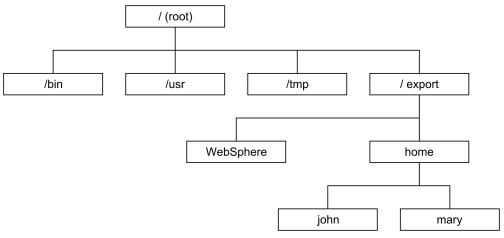


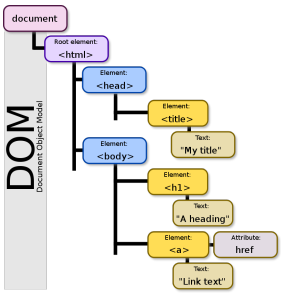
SD03 – Les Arbres

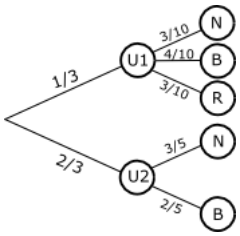
1. Introduction

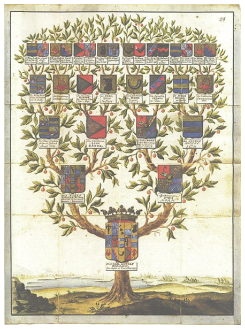
Les arbres sont utilisés pour représenter des situations de hiérarchie. Ainsi on peut les utiliser pour :

- le découpage d'un livre en chapitres, sections, paragraphes ...
- l'arborescence d'un système de fichier
- les formats de fichier XML (par balisage) tel que le HTML
- l'organigramme d'une organisation





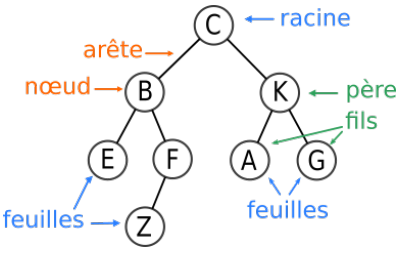




Définition 1: Arbre comme Type abstrait de données

Un arbre est une structure de données hiérarchique qui possède:

- une **racine** (premier nœud)
- des **nœuds** (ou sommets)
- des **branches** (ou arêtes)
- des **feuilles** (nœuds sans fils)



Propriété 1: Les caractéristiques d'un arbre

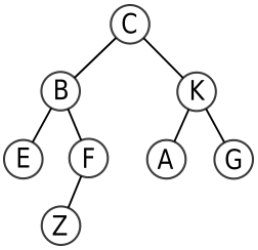
- la **taille** d'un arbre est son nombre total de nœuds.
- l'**arité** d'un nœud est son nombre de fils.
- la **profondeur** d'un nœud est le nombre de nœuds de son chemin le plus court vers la racine.
- un **chemin** est une liste de nœuds reliés par des arêtes.
- la **hauteur** d'un arbre et la profondeur de son nœud le plus profond. nous prenons comme convention:
  - si un arbre est réduit à un seul nœud, sa hauteur sera 1
  - si un arbre est vide sa hauteur est 0

Remarque 1:

Cette convention sur la hauteur d'un arbre n'est pas toujours celle adoptée dans les sujets de bac. Dans certains sujets, l'arbre vide a pour hauteur -1

Exercice 1:

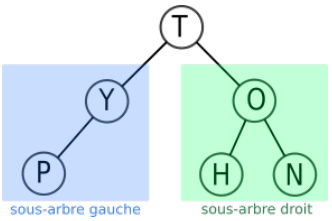
- Quelle est la taille de cet arbre? .....
- Quelle est l'arité de B? de F? de Z? .....
- Quelle est la profondeur de G? de B? de Z? de C? .....
- Quelle est la hauteur de l'arbre? .....
- Donner un chemin entre F et A .....



2. Les arbres binaires

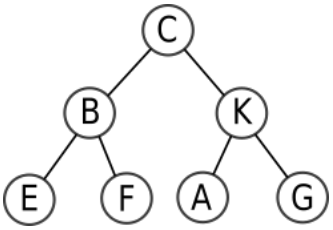
Définition 2: Les arbres binaires

Un **Arbre binaire** est un arbre dont tous les nœuds ont pour arité 2. Cela signifie que chaque nœud a au maximum deux enfants. On parlera alors de **sous arbre gauche (noté SAG)** et de **sous arbre droit (noté SAD)**.



Définition 3: Les arbres binaires complet

Un **Arbre binaire** est **complet** lorsqu'il ne manque aucun fils gauche ni aucun fils droit
Dans le cas où un arbre n'est pas complet, on pourra le rendre complet en ajoutant des nœuds valant `None` et qu'on identifiera par Δ.



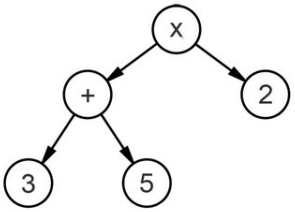
Propriété 2: Caractéristiques d'un arbre binaire complet

- Un arbre complet de hauteur h aura pour taille: $S = 1 + 2^1 + 2^2 + ... + 2^{h-1} = \sum_{k=0}^{h-1} 2^k = 2^h - 1$
- Un arbre de taille n aura une hauteur telle que: $h \leqslant n \leqslant 2^h - 1$. S'il est complet alors $n = 2^h - 1$

Exemple 1: Représentation d'une expression arithmétique

Les expressions arithmétiques sont représentables sous forme d'arbre binaire. En effet les opérateurs d'addition, de soustraction, de multiplication ou de division sont d'arité 2, c'est-à-dire qu'ils s'appliquent à deux **opérandes**. Les arbres binaires, par définition d'arité 2, sont donc tout désignés pour les représenter.

Représenter l'expression $\dfrac{2-5}{7+1} \times 3 \times \dfrac{4+6+8}{9}$ sous forme d'un arbre binaire.



Arbre binaire représentant l'expression arithmétique : $(3+5) \times 2$

3. Implémentation en python

3.1. En utilisant la programmation orientée objet

Ce que nous allons appeler *arbre* est en fait un *nœud* et ses deux fils SAG et SAD.

Exercice 2:

1. Dessiner l'arbre créé par les instructions suivantes:

Code Python

```
>>> a = Arbre(4)
>>> a.left = Arbre(3)
>>> a.right = Arbre(1)
>>> a.right.left = Arbre(2)
>>> a.right.right = Arbre(7)
>>> a.left.left = Arbre(6)
>>> a.right.right.left = Arbre(9)
```

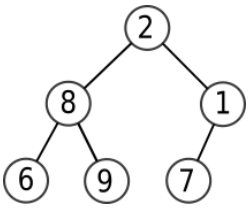
2. Nous allons créer une classe `Arbre` qui contiendra 3 attributs:
   - `data`: la valeur du nœud (int)
   - `gauche`: le sous arbre gauche (de type Arbre)
   - `droite`: le sous arbre droite (de type Arbre)

   Par défaut les attributs `gauche` et `droite` seront initialisés à `None`

   Prévoyez également des getters et setters pour manipuler les attributs `gauche` et `droite`.
3. Implémenter l'arbre de la question 1.

3.2. Implémentation à partir de tuples imbriqués

Un arbre peut se représenter par le tuple `(valeur, gauche, droite)`. L'arbre ci-contre peut donc être représenté par:



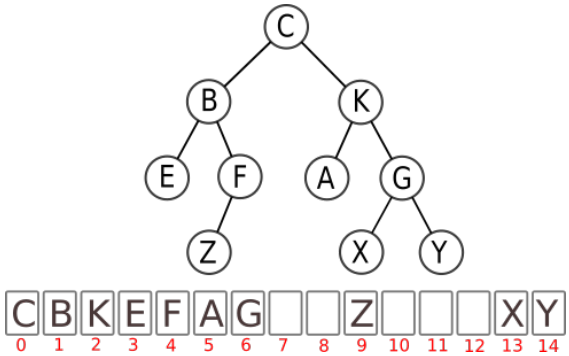
Code Python

```
>>> a = (2, (8, (6,(),()), (9,(),())), (1, (7, (),()), ()))
```

Le sous-arbre gauche est donc `a[1]` et le sous arbre droit `a[2]`

3.3. Implémentation à partir d'une liste

Les fils du nœud d'indice `i` sont placés aux indices `2i+1` et `2i+2`.
Cette méthode est utilisée en généalogie pour numéroter les individus.
Pour comprendre facilement la numérotation, il suffit de s'imaginer l'arbre complet (en rajoutant les fils vides) et de faire une numérotation en largeur, niveau par niveau.



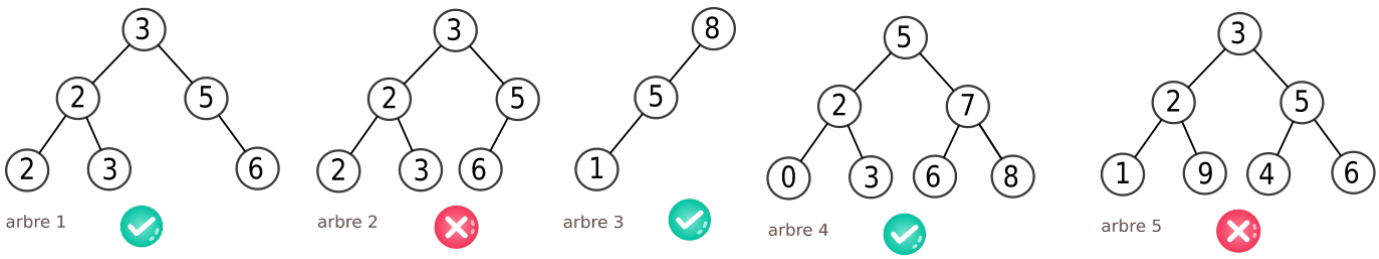
Si on note Δ le sous arbre vide, dessiner l'arbre représenté par la liste:
$a = [3, 4, \Delta, 7, 5]$

4. Arbre binaire de recherche

Définition 4: Arbre binaire de recherche

Un **arbre binaire de recherche** est un arbre binaire dont les valeurs des nœuds (valeurs qu'on appelle étiquettes, ou clés) vérifient la propriété suivante:

- l'étiquette d'un nœud est supérieure ou égale à celle de chaque nœud de son sous-arbre gauche.
- l'étiquette d'un nœud est strictement inférieure à celle du chaque nœud de son sous-arbre droit.



Exercice 3:

1. Proposer l'ordre dans lequel les valeurs ont été insérées dans les arbres binaires de recherche précédents.
2. A partir d'un ABR vide, ajouter successivement les valeurs `5, 3, 4, 6, 8, 7, 2, 9, 1`
3. Combien de comparaisons s'effectuent pour la recherche de la valeur 9 dans cet ABR
4. A partir d'un ABR vide, ajouter successivement les valeurs `6, 8, 9, 1, 4, 2, 7, 3, 5`
5. Combien de comparaisons s'effectuent pour la recherche de la valeur 9 dans cet ABR

SD04 – Les graphes

1. Généralités sur les graphes

Définition 1: Les graphes en mathématiques

Un **graphe** est un objet mathématique. Cet objet est constitué de:

- un ensemble fini de points appelés **sommets** ou **nœuds**.
- un ensemble fini de lignes (**arêtes**) ou de flèches (**arcs**) reliant les sommets.

Les **graphes** sont une modélisation mathématique efficace car simple et adaptable permettant de modéliser, en vue de leur étude et de leur amélioration, par exemple:

- des réseaux de communication (téléphone, internet, sociaux);
- des réseaux de transport (routiers, aériens, maritimes, ferroviaires, ...)
- des circuits électriques;
- des relations biologiques (proie-prédateur, évolution, ...)

Définition 2: Graphe orienté ou non orienté

Un graphe est dit **orienté** si les liens entre les sommets sont des arcs. Il est **non orienté** si ce sont des arêtes.

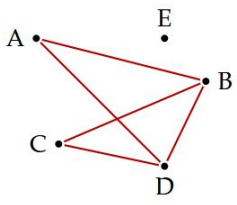
Définition 3: Graphe pondéré

Un graphe est dit **pondéré** si on affecte des valeurs ou des poids (ou tout autre indication) sur les arcs ou les arêtes. Ces valeurs peuvent indiquer des distances, des coûts etc.

Définition 4: Eléments d'un graphe

- un graphe d'ordre n est un ensemble de n points, appelés sommets, reliés entre eux par des liens;
- un arc qui relie un sommet à lui-même est appelé une boucle;
- deux sommets sont **adjacents** s'ils sont reliés par une arête ou un arc;
- un sommet est **isolé** si aucune arête ou arc ne le relie aux autres sommets;
- le **degré** d'un sommet est le nombre d'arêtes ou d'arcs dont ce sommet est une extrémité (un boucle comptant pour 2);
- un graphe est **complet** si tous les sommets sont adjacents entre eux.

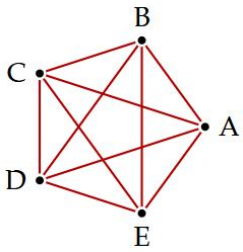
Figure 1



Non orienté
Non pondéré

A	B	C	D	E

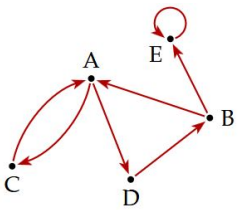
Figure 2



Non orienté
Non pondéré

Complet degré 4

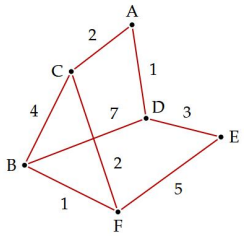
Figure 3



Orienté
Non pondéré

A	B	C	D	E

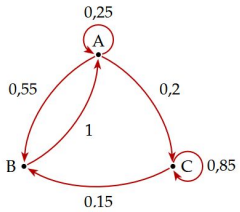
Figure 4



Non orienté
Pondéré

A	B	C	D	E	F

Figure 5



Orienté
Pondéré

A	B	C

Définition 5: Parcours d'un graphe

- une **chaine** est une liste de sommets telle que chaque sommet de la liste soit adjacent au suivant;
- un graphe est **connexe** s'il existe une chaine entre deux sommets quelconques de ce graphe;
- une **chaine fermée** est une chaine dont l'origine et l'extrémité sont confondues;
- une chaine fermée composée d'arêtes toutes distinctes forme un **cycle**.

Remarque 2

Dans un graphe orienté, on parle de **chemin** pour une chaine et de **circuit** pour un cycle.

- Figure 1: A-B-D-C-B est une chaine de longueur 4 et D-A-B-C-D un cycle de longueur 4.
- Figure 3: A-D-B-E est un chemin de longueur 3 et B-A-C-A-D-B un circuit de longueur 5

Définition 6: Vocabulaire

- deux sommets sont **voisins** (ou **adjacents**) s'ils sont reliés par une arête;
- dans le cas où le graphe est orienté: $X \to Y$, Y est adjacent à X mais X n'est pas adjacent à Y;
- un graphe est **connexe** lorsque tous ses sommets peuvent être reliés par un chemin. La figure 1 n'est pas connexe.

Exercice 1:

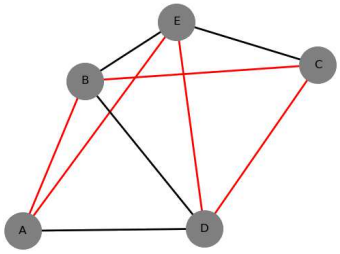
1. Ce graphe est-il complet? Que faut-il faire pour le rendre complet?

   .....

2. les arêtes repérées en rouge forment-elles un cycle? Pourquoi?

   .....

3. Ce graphe est-il connexe? En rajoutant un sommet, est-il de le rendre non-connexe? .....

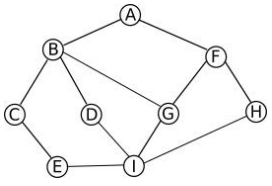


Définition 7: Les graphes connexes

- la **distance** entre deux sommets est la longueur minimale d'un chemin joignant ces sommets (ou la somme des poids la plus petite). La plus grande distance entre deux sommets est la **diamètre** du graphe;
- l'**excentricité** d'un sommet est la distance maximale entre ce sommet et tous les autres sommets du graphe;
- les **centre** d'un graphe sont les sommets qui possèdent l'excentricité minimale. On note cette valeur le **rayon** du graphe.

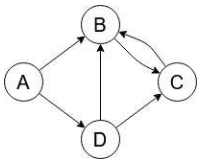
Exercice 2:

1. Le graphe est-il pondéré? orienté? .....
2. Quel est le degré du sommet G? de F? .....
3. Donner deux chemins entre D et F. .....
4. Quelle est la distance entre C et H? .....
5. Quelle est l'excentricité de G? .....
6. Quels sont les centres de ce graphe? .....



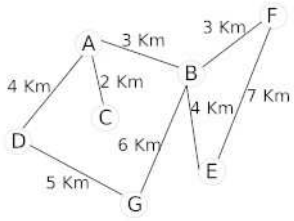
Exercice 3:

1. Le graphe est-il pondéré? orienté? .....
2. Quel est le degré du sommet C? de A? .....
3. Donner deux chemins entre A et C. .....
4. Quelle est la distance entre A et C? .....
5. Le graphe est-il fortement connexe?? .....



Exercice 4:

1. Le graphe est-il pondéré? orienté? .....
2. Quel est le degré du sommet B? de C? .....
3. Donner deux chemins entre D et F. .....
4. Quelle est la distance entre D et F? .....



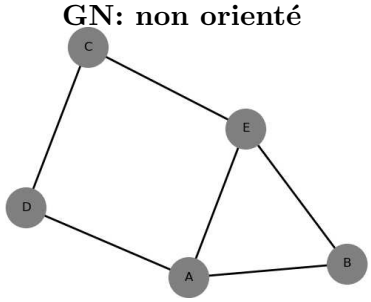
2. Représenter un graphe: la liste d'adjacence

Propriété 1: Liste d'adjacence

Pour un graphe donné, on associe à chaque sommet la liste de ses voisins (sommets adjacents). Dans le cas d'un graphe orienté, on peut aussi parler de **liste de successeurs** (flèches partant du sommet considéré) ou **liste de prédécesseurs** (flèches arrivant au sommet considéré).

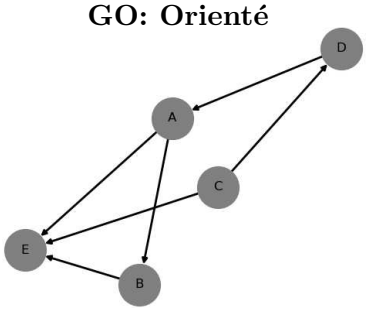
Lorsque le graphe est pondéré, on associe à chaque sommet une liste de tuples.

GN: non orienté



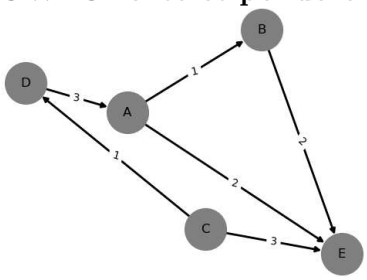

```
GN = {A:[B,D,E],
      B:[A,E],
      C:[D,E],
      D:[A,C],
      E:[A,B,C]      }
```

GO: Orienté



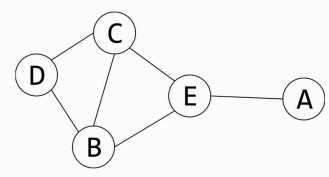

```
GO = {A:[B,E],
      B:[E],
      C:[D,E],
      D:[A],
      E:[]      }
```

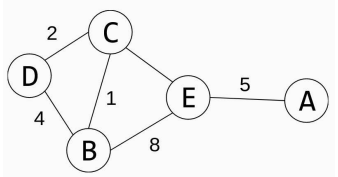
GW: Orienté et pondéré

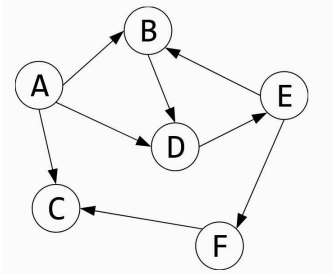


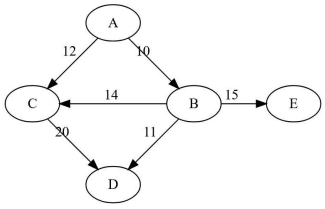

```
GW = {A:[(B,1),(E,2)],
      B:[(E,2)],
      C:[(D,1),(E,3)],
      D:[(A,3)],
      E:[]      }
```

Exercice 5: Etablir la liste d'adjacence des graphes suivants









Exercice 6: Dessiner les graphes correspondants aux listes d'adjacences suivantes

```
G1 = {A:[B,C],
      B:[A],
      C:[B,C,D],
      D:[A,C]}
```

```
G2 = {A:[B,C],
      B:[A,D],
      C:[A],
      D:[B]}
```

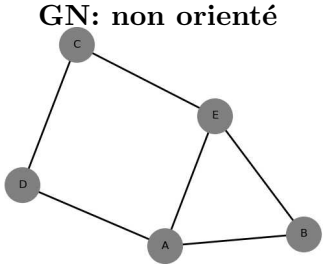
```
G3 = {A:[(C,2),(D,3)],
      B:[(A,2),(C,1)],
      C:[(D,4),(B,1)],
      D:[(C,1)]}
```

3. Représenter un graphe: la matrice d'adjacence

Définition 8: La matrice d'adjacence

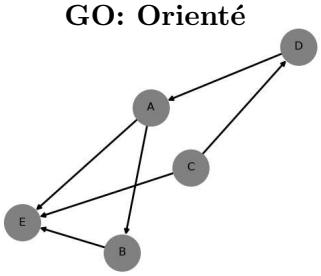
La **matrice d'adjacence** est un tableau carré (à double entrée). Les lignes et les colonnes correspondent aux sommets (ordonnés dans l'ordre alphanumérique si rien n'est précisé).
Si le sommet X et adjacent à Y ($X \to Y$), on remplit dans la matrice, à l'intersection de la ligne de X et de la colonne de Y, un 1, ou bien une donnée si le graphe est pondéré (il y aura alors une matrice d'adjacence selon chaque type de donnée). S'il n'y a pas de relation entre X et Y, on note un 0.
Lorsque le graphe n'est pas orienté, la matrice d'adjacence est symétrique par rapport à sa diagonale (diagonale principale: de en haut à gauche à en bas à droite); aussi, les boucles apparaissent sur la diagonale.

GN: non orienté



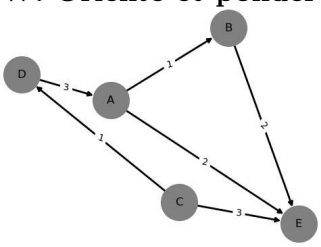

```
GN = [ [0, 1, 0, 1, 1],
       [1, 0, 0, 0, 1],
       [0, 0, 0, 1, 1],
       [1, 0, 1, 0, 0],
       [1, 1, 1, 0, 0] ]
```

GO: Orienté



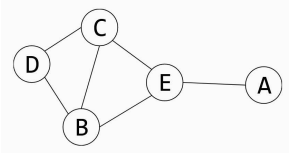

```
GO = [ [0, 1, 0, 0, 1],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 1, 1],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0] ]
```

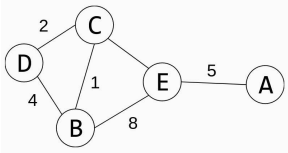
GW: Orienté et pondéré

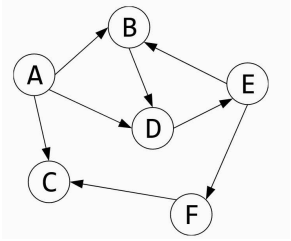


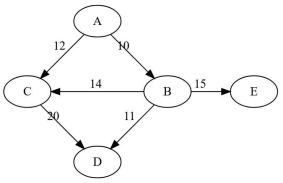

```
GW = [ [0, 1, 0, 0, 2],
       [0, 0, 0, 0, 2],
       [0, 0, 0, 1, 3],
       [3, 0, 0, 0, 0],
       [0, 0, 0, 0, 0] ]
```

Exercice 7: Etablir la matrice d'adjacence des graphes suivants









Exercice 8: Dessiner les graphes correspondants aux matrices d'adjacences suivantes

```
G1 = [[0,2,3,0],
      [2,0,1,2],
      [3,1,0,2],
      [0,2,2,0]]
```

```
G2 = [[0,1,0,0],
      [1,0,0,1],
      [0,0,0,1],
      [0,1,1,0]]
```

```
G3 = [[0,1,0,1,0,0,0],
      [0,0,1,0,0,0,0],
      [0,0,0,1,0,0,0],
      [1,0,0,1,0,0,0],
      [0,1,0,0,0,1,0],
      [0,0,0,0,0,0,1],
      [0,0,0,0,1,0,1]]
```

Voici la matrice d'adjacence d'un graphe H :

```
H = [ [0, 1, 0, 1, 0, 0, 0],
      [0, 0, 1, 0, 0, 0, 0],
      [0, 0, 0, 1, 0, 0, 0],
      [1, 0, 0, 1, 0, 0, 0],
      [0, 1, 0, 0, 0, 1, 0],
      [0, 0, 0, 0, 0, 0, 1],
      [0, 0, 0, 0, 1, 0, 1] ]
```

1. Le graphe H est-il orienté ?
2. Donner une représentation graphique de H.
3. Donner sa liste d'adjacence (de successeurs) et sa liste de prédécesseurs.
4. En reprenant ce même graphe, mais en le rendant non-orienté H' (remplacer les flèches par des arcs simples): le graphe obtenu est-il connexe? Complet? Quels sont le diamètre, les centres et le rayon de G?