# Représentation d'un graphe en POO

Christophe Viroulaud

Terminale - NSI

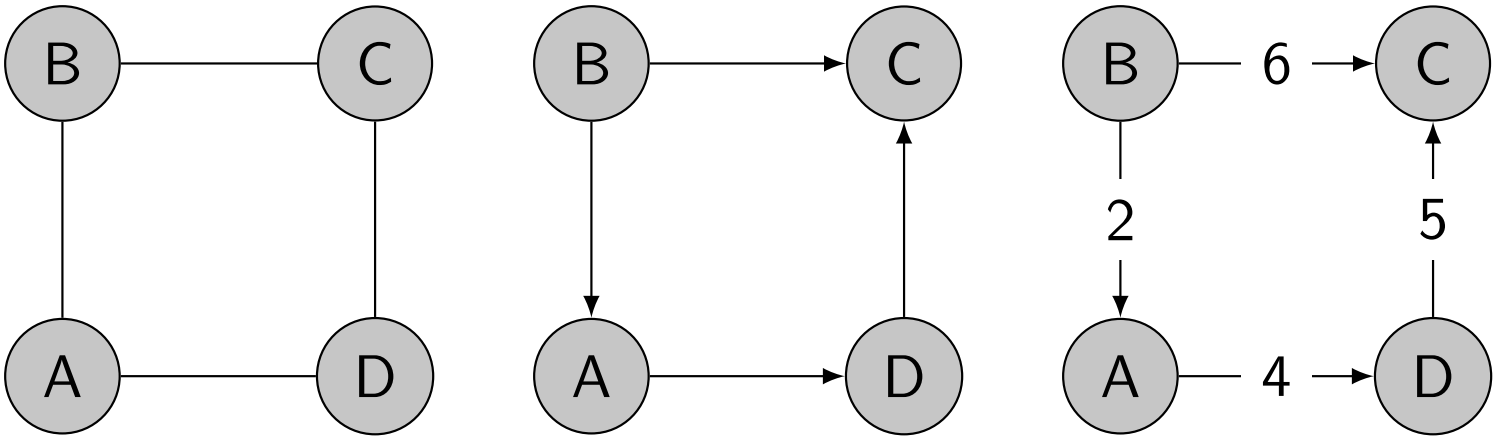
**Algo 22**

Les graphes sont des outils très utilisés en informatique. Il semble donc intéressant de disposer d'une bibliothèque fournissant les outils nécessaires à la création et la manipulation d'un graphe.

Construire une bibliothèque **`graphe`**.

# Sommaire

# Conception - Différents graphes




## Observation

La bibliothèque **`graphe`** doit prendre en compte les différents cas de figures.

- Une arête non-orientée est équivalente à deux arêtes orientées de sens opposé.
- Un graphe non pondéré est équivalent à un graphe où toutes les pondérations valent 1.

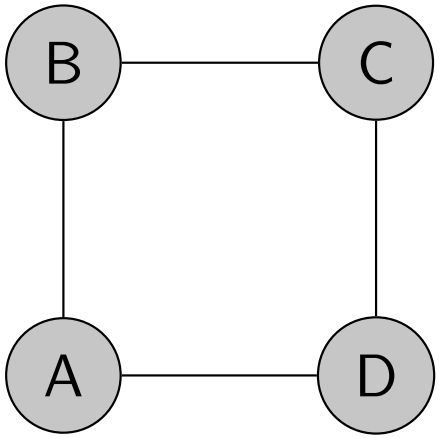
# Sommaire

# Les ensembles



## À retenir

Un ensemble est une collection non ordonnée sans éléments en double. https://tinyurl.com/set-pyt




```
1 voisins_B = { ("A", 1), ("D", 1) }
```

Code 1 – Ensemble des voisins de B

## Remarque

```
1 # Crée un ensemble vide
2 e = set()
3
4 # Crée un dictionnaire vide
5 d = {}
```

# Sommaire

# Implémentation - Classe `Graphe`



**Activité 1 :**

1. Créer le fichier `biblio_graphe.py`
2. Construire la classe `Graphe` et son constructeur qui initialisera :
   - un attribut booléen `oriente` qui sera initialisé par une valeur booléenne passée en paramètre.
   - un dictionnaire vide nommé `sommets`.

# Avant de regarder la correction



- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

# Correction

```
1  class Graphe:
2      def __init__(self, oriente: bool):
3          self.sommets = {}
4          self.oriente = oriente
```

**Activité 2 :**

1. Écrire la méthode `ajouter_sommet(self, s: str)` → `None` qui ajoute un nouveau sommet dans le dictionnaire `sommets` s'il n'est pas déjà présent. Un ensemble vide sera associé au sommet.
2. Écrire la méthode `ajouter_arete(self, s1: str, s2: str, d: int = 1)` → `None` qui crée une arête, éventuellement orientée de `s1` vers `s2`, de pondération `d`. Si la pondération n'est pas précisée la valeur `1` est utilisée par défaut. Si les sommets n'existent pas, la méthode devra les créer préalablement.

# Avant de regarder la correction



- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

# Correction

```
1  def ajouter_sommet(self, s: str) -> None:
2      if not(s in self.sommets):
3          self.sommets[s] = set()
4
5  def ajouter_arete(self, s1: str, s2: str, d:
       int = 1) -> None:
6      # ajout éventuel des sommets
7      self.ajouter_sommet(s1)
8      self.ajouter_sommet(s2)
9      # création arête
10     self.sommets[s1].add((s2, d))
11     if not self.oriente:
12         self.sommets[s2].add((s1, d))
```

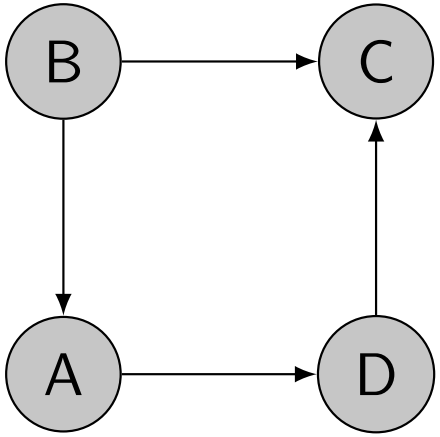



FIGURE 1 – Exemple

**Activité 3 :**

1. Créer un fichier **`exemple.py`**
2. Importer la bibliothèque et créer une instance de **`Graphe`** représentant le graphe (figure 1).
3. Dans la classe **`Graphe`** créer la méthode **`affiche_voisins(self, s: str)`** $\rightarrow$ **`None`** qui affiche dans la console les voisins de **`s`** ainsi que la distance entre les 2 sommets.

# Avant de regarder la correction



- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

# Correction



```
1 exemple = Graphe()
2 exemple.ajouter_arete("A", "D")
3 exemple.ajouter_arete("D", "C")
4 exemple.ajouter_arete("B", "A")
5 exemple.ajouter_arete("B", "C")
```

Code 2 – Instanciation

```python
def affiche_voisins(self, s: str) -> None:
    for v in self.sommets[s]:
        print(f"{s} --> {v[0]} : {v[1]}")
```

Code 3 – Méthode d'affichage

```
>>> exemple.affiche_voisins("B")
B --> A : 1
B --> C : 1
```

Code 4 – Appel de la méthode

# Sommaire

# Méthodes de parcours



**Activité 4 :**

1. Dans le même dossier que la bibliothèque, placer le fichier `structures.py` construit dans les cours précédents.
2. Écrire la méthode itérative `dfs_it(self, s: str)` $\rightarrow$ `list` qui renvoie la liste des sommets visités depuis `s` avec un parcours en profondeur.