

Représentation d'un graphe en POO

Christophe Viroulaud

Terminale - NSI

Algo 22

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

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.

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Construire une bibliothèque **graphe**.

1. Conception

1.1 Différents graphes

1.2 Les ensembles

2. Implémentation

3. Application

4. Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

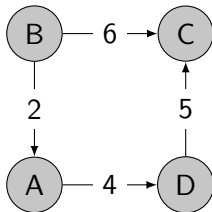
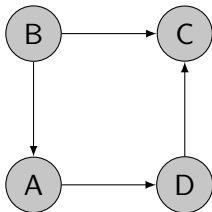
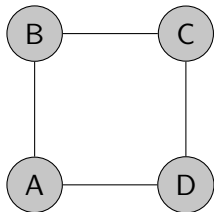
Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception - Différents graphes



Observation

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

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

- ▶ 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.

1. Conception

1.1 Différents graphes

1.2 Les ensembles

2. Implémentation

3. Application

4. Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

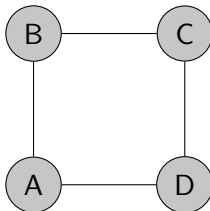
Méthodes de parcours

Application

Pour aller plus loin

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

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Remarque

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

1. Conception

2. Implémentation

2.1 Classe Graphe

2.2 Méthodes de parcours

3. Application

4. Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

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.

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

```
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.

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Avant de regarder la correction



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

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

```
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:
6      int = 1) -> None:
7      # ajout éventuel des sommets
8      self.ajouter_sommet(s1)
9      self.ajouter_sommet(s2)
10     # création arête
11     self.sommets[s1].add((s2, d))
12     if not self.orienté:
13         self.sommets[s2].add((s1, d))
```

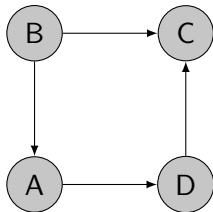


FIGURE 1 – Exemple

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) → 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.

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

```
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

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

Code 3 – Méthode d'affichage

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

Code 4 – Appel de la méthode

1. Conception

2. Implémentation

2.1 Classe Graphe

2.2 Méthodes de parcours

3. Application

4. Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

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) → list` qui renvoie la liste des sommets visités depuis `s` avec un parcours en profondeur.
3. **Pour les plus avancés :** Écrire la méthode récursive `dfs_rec(self, s: str, parcours: list) → list`.

Avant de regarder la correction



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

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes
Les ensembles

Implémentation

Classe Graphe
Méthodes de parcours

Application

Pour aller plus loin

```
1 def dfs_it(self, s: str) -> list:
2     parcours = []
3     p = Pile()
4     p.empiler(s)
5     while not p.est_vide():
6         en_cours = p.depiler()
7         if en_cours not in parcours:
8             parcours.append(en_cours)
9
10            for voisin in self.sommets[en_cours]:
11                p.empiler(voisin[0])
12
13     return parcours
```

```
1 >>> exemple.dfs_it("A")
2 ['A', 'D', 'C']
```

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

```
1 def dfs_rec(self, s: str, parcours: list) -> None:
2     """
3     version réursive du parcours en profondeur
4
5     Args:
6         s (str): départ
7         parcours (list): sommets traversés
8     """
9     if s not in parcours:
10         parcours.append(s)
11         for voisin in self.sommets[s]:
12             self.dfs_rec(voisin[0], parcours)
```

```
1 >>> parcours_rec = []
2 >>> exemple.dfs_rec("A", parcours_rec)
3 >>> parcours_rec
4 ['A', 'D', 'C']
```

1. Conception

2. Implémentation

3. Application

4. Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

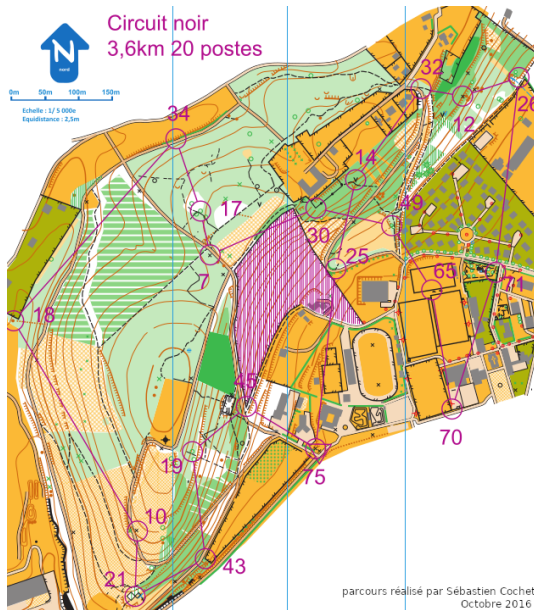
Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Application



Représentation
d'un graphe
en POO

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

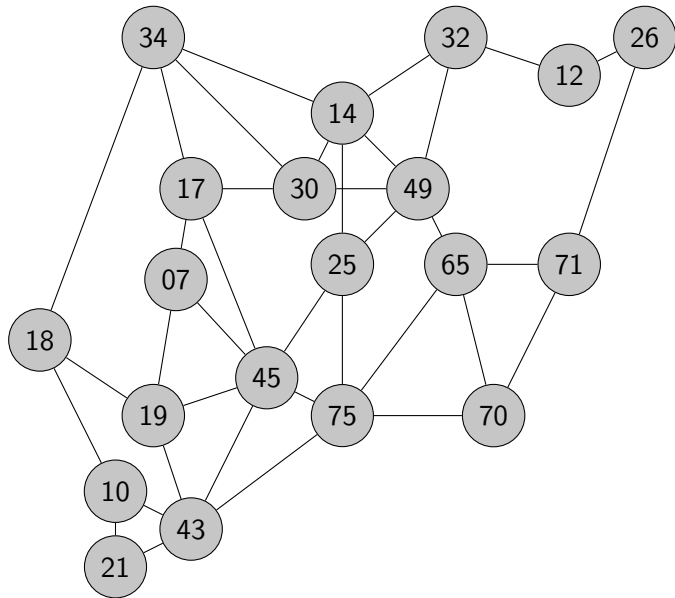


FIGURE 2 – Graphe représentatif de la carte de CO

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Activité 5 :

1. Placer le fichier `parcours_noir.json` dans le même dossier que la bibliothèque.
2. Créer un fichier `parcours_co.py`
3. Construire une instance de **Graphe** représentative de la carte de CO.

Avant de regarder la correction



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

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

```
1 import json
2 from biblio_graphe import Graphe
3
4 # chargement du json
5 f = open("parcours_noir.json")
6 list_graphe = json.load(f)
7
8 graphe = Graphe(False)
9 # parcours du graphe
10 for couple in list_graphe:
11     sommet = couple["sommet"]
12     adjacents = couple["adjacents"]
13     for voisin in adjacents:
14         # les sommets sont ajoutés automatiquement
15         graphe.ajouter_arete(sommet, voisin)
16
17 f.close()
```

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

1. Conception

2. Implémentation

3. Application

4. Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Considérons un graphe non orienté et non pondéré. Un parcours en largeur permet de déterminer le plus court chemin entre deux sommets. En effet, le parcours *s'éloigne* de l'origine niveau après niveau. Il suffit alors de maintenir un dictionnaire des sommets visités associés à leur prédécesseur pour retrouver le chemin le plus court partant de l'origine jusqu'à n'importe quel sommet.

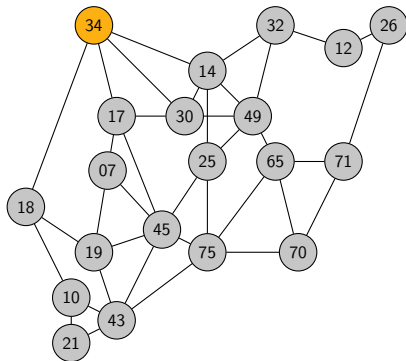


FIGURE 3 – Plus court chemin entre 34 et 19.

► 34 : niveau 0

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

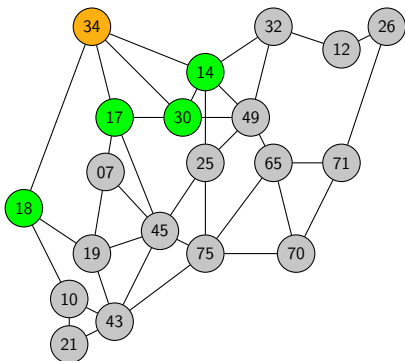


FIGURE 4 – Plus court chemin entre 34 et 19.

- 34 : niveau 0
- 14 - 30 - 17 - 18 : niveau 1

Conception

Différents graphes

Les ensembles

Implémentation

Classe Graphe

Méthodes de parcours

Application

Pour aller plus loin

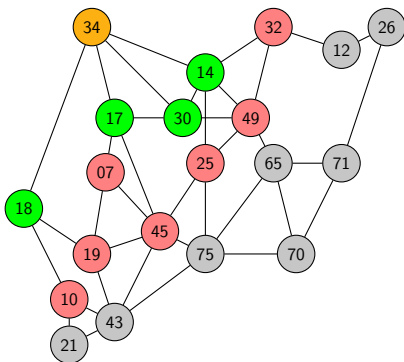


FIGURE 5 – Plus court chemin entre 34 et 19.

- 34 : niveau 0
- 14 - 30 - 17 - 18 : niveau 1
- 32 - 49 - 25 - 45 - 07 - 19 - 10 : niveau 2