

TD: graphes:

Exercice 1: implémentation d'un graphe (liste des successeurs): On s'intéresse au programme suivant:

```
0 graphe = {  
2     "A": ["B", "E"],  
     "B": ["A", "C", "E"],  
4     "C": ["B"],  
     "D": [],  
     "E": ["A", "B"],  
6     }  
  
8 def deg(g, s):  
10     if g.get(s) is not None:  
        return len(g[s])
```

graphe1.py

1. Quel type est utilisé pour implémenter le graphe dans ce programme?
2. Représenter le graphe.
3. Expliquer le fonctionnement de deg() et tester cette fonction.

Exercice 2: implémentation d'un graphe (matrice): On s'intéresse au programme suivant:

```
0 G = [  
2     [0, 1, 0, 0, 1]  
     [1, 0, 1, 0, 1]  
4     [0, 1, 0, 0, 0]  
     [0, 0, 0, 0, 0]  
6     [1, 1, 0, 0, 0]  
     ]
```

graphe2.py

1. Quel type est utilisé pour implémenter le graphe dans ce programme?
2. Représenter le graphe.

Exercice 3: implémentation d'un graphe orienté (classes): On s'intéresse au programme suivant:

```
0 class Arete:
1     def __init__(self ,src ,dest):
2         self.src = src
3         self.dest = dest
4
5
6     def get_source(self):
7         return self.src
8
9     def get_destination(self):
10        return self.dest
11
12    def __str__(self):
13        return str(self.src) + "->" + str(self.dest)
14
15 class Graphe_Or:
16    def __init__(self):
17        self.sommets = []
18        self.arestes = {}
19
20    def ajoute_sommet(self ,s):
21        if s not in self.sommets:
22            self.sommets.append(s) #on ajoute dans la liste
23            self.arestes[s] = []    #on ajoute dans le dictionnaire
24
25    def ajoute_arete(self ,a):
26        src = a.get_source()      #on utilise la classe Arete
27        dest = a.get_destination()
28        if src in self.sommets and dest in self.sommets:
29            if not dest in self.arestes[src]:
30                self.arestes[src].append(dest) #on ajoute la destination liee a la
31                source via l'arete
32
33    def get_sommets(self):
34        return self.sommets
35
36    def __str__(self):
37        res = ""
38        for k in self.arestes:
39            for d in self.arestes[k]:
40                res = res + str(k) + "->" + str(d) + "\n"
41        return res[:-1]
```

graphe3.py

1. Expliquer avec précision les méthodes et attributs utilisés dans les classes Arete et Graphe_Or.
2. Implémenter le graphe suivant à l'aide des classes de ce programme.

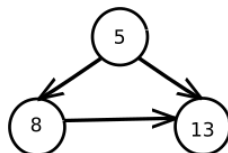


Figure 1: Graphe orienté

Exercice 4: implémentation d'un graphe non orienté (classes): On ajoute cette classe au programme de l'exercice 3:

```
0 class Graphe(Graphe_Or):
1     def ajoute_arete(self, a):
2         Graphe_Or.ajoute_arete(self, a)
3         retour = Arete(a.get_destination(), a.get_source())
4         Graphe_Or.ajoute_arete(self, retour)
```

graphe4.py

1. Expliquer brièvement le fonctionnement de cette nouvelle classe.
2. Implémenter le graphe suivant à l'aide des classes de ce programme.

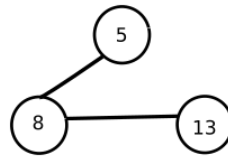


Figure 2: Graphe non orienté

Exercice 5: matrices d'adjacence: Tracer les graphes associés aux matrices d'adjacence suivante:

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

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

3. $M_3 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

Exercice 6: matrices d'adjacence: Ecrire la matrice d'adjacence du graphe suivant:

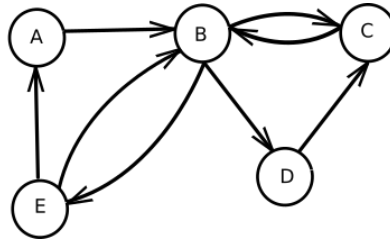


Figure 3: Graphe orienté

Exercice 7: Soit le graphe suivant:

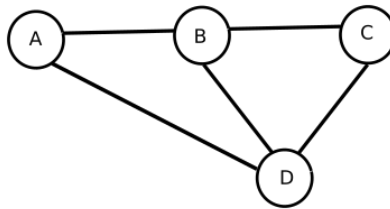


Figure 4: Graphe orienté

1. Écrire un dictionnaire Python où les clés sont les sommets pour représenter ce graphe.
2. Écrire une fonction `sommets()` qui prend en paramètres un sommet `s` et un graphe `g`, sous la forme d'un dictionnaire et qui renvoie la liste des sommets liés par une arête au sommet `s`.