

TD: algorithmes sur les graphe orientés et pondérés:

Exercice 1: représentation d'un graphe orienté:

Un graphe orienté est représenté en Python par la liste de ses successeurs et est implémenté de la façon suivante:

```
0 class Arete:
1     def __init__(self, src, dest):
2         self.src = src
3         self.dest = dest
4
5     def get_source(self):
6         return self.src
7
8     def get_destination(self):
9         return self.dest
10
11    def __str__(self):
12        return str(self.src) + "->" +str(self.dest)
13
14 class Graph_or:
15     def __init__(self):
16         self.sommets = []
17         self.arestes = {}
18
19     def ajoute_sommet(self, s):
20         if s in self.sommets:
21             raise ValueError("sommet deja present")
22         else:
23             self.sommets.append(s)
24             self.arestes[s] = []
25
26     def ajoute_arete(self, a):
27         src = a.get_source()
28         dest = a.get_destination()
29         if not (src in self.sommets and dest in self.sommets):
30             raise ValueError("sommet absent")
31         self.arestes[src].append(dest)
32
33     def enfant_de(self, s):
34         return self.arestes[s]
35
36     def sommet(self, s):
37         return s in self.sommets
38
39     def __str__(self):
40         res = ""
41         for k in self.arestes:
42             for d in self.arestes[k]:
43                 res = res + str(k) + "->" +str(d) + "\n"
44         return res[:-1]
45
46
```

```

48 g = Graph_or()
sommets = ['a', 'b', 'c', 'd', 'e', 'f']
50 for s in sommets:
    g.ajoute_sommet(s)
52 g.ajoute_arete(Arete(sommets[0], sommets[1]))
g.ajoute_arete(Arete(sommets[1], sommets[2]))
54 g.ajoute_arete(Arete(sommets[2], sommets[3]))
g.ajoute_arete(Arete(sommets[2], sommets[4]))
56 g.ajoute_arete(Arete(sommets[3], sommets[4]))
g.ajoute_arete(Arete(sommets[3], sommets[5]))
58 g.ajoute_arete(Arete(sommets[0], sommets[2]))
g.ajoute_arete(Arete(sommets[1], sommets[0]))
60 g.ajoute_arete(Arete(sommets[3], sommets[1]))
g.ajoute_arete(Arete(sommets[4], sommets[0]))

```

objet_graphe_oriente.py

1. Expliquer avec précision les rôles des méthodes des classes Arete et Graph_or
2. Représenter sur une feuille le graphe g.

Exercice 2: parcours en profondeur et plus court chemin d'un graphe orienté:

1. Adapter la fonction parcours_dfs du TD sur les algorithmes sur les graphes non orientés afin d'obtenir le parcours en profondeur du graphe g.
2. Adapter la fonction pccs_dfs du TD sur les algorithmes sur les graphes non orientés afin d'obtenir le plus court chemin entre 'a' et 'f'.

Exercice 3: parcours en largeur et plus court chemin d'un graphe orienté:

On programme la recherche d'un plus court chemin sur un graphe orienté de la manière suivante:

```

0 def pcc_bfs(graphe, debut, fin, chemin_fin = []):
    chemin_deb = [debut]
    chemin_fin.append(chemin_deb)
    2 while len(chemin_fin) != 0:
        4 chemin_temp = chemin_fin.pop(0)
        dernier_s = chemin_temp[-1]
        6 #pour afficher la progression
        print("chemin courant: ", affiche(chemin_temp))
        8 if dernier_s == fin:
            return chemin_temp
        10 for s in graphe.enfant_de(dernier_s):
            if s not in chemin_temp:
                12 nouveau = chemin_temp + [s]
                chemin_fin.append(nouveau)

```

largeur_objet_graphe_oriente.py

Ce programme utilise l'implémentation objet des graphes de l'exercice 1 ainsi que la fonction affiche du TD sur les algorithmes sur les graphes non orientés.

1. Expliquer comment fonctionne ce programme. Tester cette fonction sur le graphe de l'exercice 1 entre les sommets a et f.

2. En quoi cette fonction utilise un parcours en largeur d'abord?

Exercice 4: algorithme de Dijkstra:

On programme cet algorithme de la manière suivante:

```
0 graphe = {'S1' : [( 'S2' , 2), ( 'S5' , 1)],
2         'S2' : [( 'S1' , 2), ( 'S3' , 2), ( 'S6' , 2), ( 'S7' , 3)],
4         'S3' : [( 'S2' , 2), ( 'S6' , 2), ( 'S7' , 5), ( 'S8' , 1)],
6         'S4' : [( 'S7' , 3), ( 'S8' , 2)],
8         'S5' : [( 'S1' , 1), ( 'S6' , 4)],
10        'S6' : [( 'S2' , 2), ( 'S5' , 4), ( 'S3' , 2)],
12        'S7' : [( 'S2' , 3), ( 'S3' , 5), ( 'S4' , 3), ( 'S8' , 1)],
14        'S8' : [( 'S3' , 1), ( 'S4' , 2), ( 'S7' , 1)]
16        }
18
20
22 from math import inf
24
26 def minimum(dico):
28     mini = inf #infini
30     s = -1
32     for k in dico:
34         if dico[k] < mini:
36             mini = dico[k]
38             s = k
40     return s
42
44 def dijkstra(G,s):
46     D = {} #tableau des distances minimales
48     d = {k:inf for k in G} #tableau des distances initiales
50     d[s] = 0 #s sommet de depart
52     while len(d) > 0: #fin quand d est vide
54         k = minimum(d) #sommet de distance min pour demarrer une etape
56         for j in range(len(G[k])): #on regarde les voisins de k
58             v, c = G[k][j] #v un voisin de k, c la distance
60             if v not in D:
62                 d[v] = min(d[v], d[k] + c)
64         D[k] = d[k] #on copie le sommet et la distance dans D
66         del d[k] #on supprime le sommet de d
68     return D
```

dijkstra.py

1. Représenter le graphe pondéré de ce programme.
2. Expliquer le fonctionnement de ce programme.
3. Le tester à partir du sommet S1.