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:
       def __init__(self, src, dest):
2          self.src = src
           self.dest = dest
4
       def get_source(self):
6          return self.src

8      def get_destination(self):
           return self.dest
10
       def __str__(self):
12         return str(self.src) + "->" +str(self.dest)


14
   class Graph_or:
16     def __init__(self):
           self.sommets = []
18         self.aretes = {}

20     def ajoute_sommet(self, s):
           if s in self.sommets:
22             raise ValueError("sommet deja present")
           else:
24             self.sommets.append(s)
               self.aretes[s] = []
26
       def ajoute_arete(self, a):
28         src = a.get_source()
           dest = a.get_destination()
30         if not (src in self.sommets and dest in self.sommets):
               raise ValueError("sommet absent")
32         self.aretes[src].append(dest)

34     def enfant_de(self, s):
           return self.aretes[s]
36
       def sommet(self, s):
38         return s in self.sommets

40     def __str__(self):
           res = ""
42         for k in self.aretes:
               for d in self.aretes[k]:
44                 res = res + str(k) + "->" +str(d) + "\n"
           return res[:-1]
46
```

```
g = Graph_or()
sommets = ['a', 'b', 'c', 'd', 'e', 'f']
for s in sommets:
    g.ajoute_sommet(s)
g.ajoute_arete(Arete(sommets[0], sommets[1]))
g.ajoute_arete(Arete(sommets[1], sommets[2]))
g.ajoute_arete(Arete(sommets[2], sommets[3]))
g.ajoute_arete(Arete(sommets[2], sommets[4]))
g.ajoute_arete(Arete(sommets[3], sommets[4]))
g.ajoute_arete(Arete(sommets[3], sommets[5]))
g.ajoute_arete(Arete(sommets[0], sommets[2]))
g.ajoute_arete(Arete(sommets[1], sommets[0]))
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'ontenir 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'ontenir 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:

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

```
graphe = {'S1' : [('S2', 2), ('S5', 1)],
          'S2' : [('S1', 2), ('S3', 2), ('S6', 2), ('S7', 3)],
          'S3' : [('S2', 2), ('S6', 2), ('S7', 5), ('S8', 1)],
          'S4' : [('S7', 3), ('S8', 2)],
          'S5' : [('S1', 1), ('S6', 4)],
          'S6' : [('S2', 2), ('S5', 4), ('S3', 2)],
          'S7' : [('S2', 3), ('S3', 5), ('S4', 3), ('S8', 1)],
          'S8' : [('S3', 1), ('S4', 2), ('S7', 1)]
          }

from math import inf

def minimum(dico):
    mini = inf #infini
    s = -1
    for k in dico:
        if dico[k] < mini:
            mini = dico[k]
            s = k
    return s


def dijkstra(G,s):
    D = {}                              #tableau des distances minimales
    d = {k:inf for k in G}              #tableau des distances initiales
    d[s] = 0                            #s sommet de depart
    while len(d) > 0:                   #fin quand d est vide
        k = minimum(d)                  #sommet de distance min pour demarrer une etape
        for j in range(len(G[k])):      #on regarde les voisins de k
            v, c = G[k][j]              #v un voisin de k, c la distance
            if v not in D:
                d[v] = min(d[v], d[k] + c)
        D[k] = d[k]                     #on copie le sommet et la distance dans D
        del d[k]                        #on supprime le sommet de d
    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.