

L'objectif de ce travail est de créer les classes *Gmatrice* et *Gliste* qui permettent d'implémenter un graphe en python.

1- UTILISATION DE GRAPHVIZ POUR VISUALISER LE GRAPHE :

Graphviz est un logiciel qui permet de tracer graphiquement un graphe.

```
from graphviz import Digraph , Graph
graphe = Graph(format = 'png' , filename = 'graphe')

# -----
graphe.edge('Lyon', 'Paris', '488 km')
graphe.clear()
graphe.edge('Lyon', 'Marseille', '316 km')
graphe.edge('Marseille', 'Toulouse', '405 km')
graphe.edge('Toulouse', 'Lyon')
graphe.view()
```

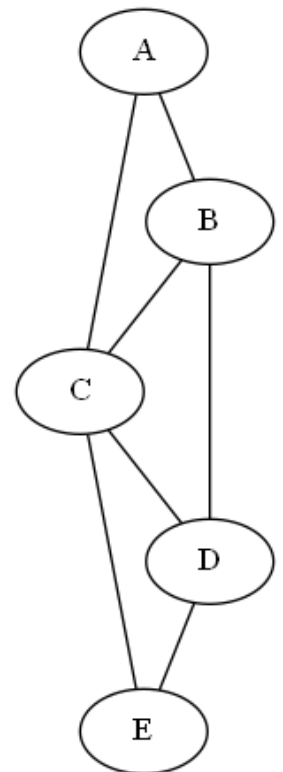
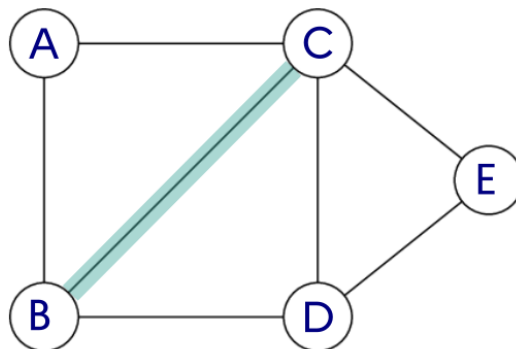
⇒ Ecrire et exécuter le script ci-dessous. Il utilise la classe *Graph* qui est importée.

Dans ce script, on utilise les méthodes *edge()*, *clear()* et *view()* de la classe *Graph*.

⇒ Qu'obtient-on si on utilise la classe importée *Digraph* à la place de *Graph* ?

Avec la classe *Digraph*, le graphe est orienté.

⇒ Modifier le script pour pouvoir tracer le graphe donné ci-dessous :



⇒ Enregistrer ce fichier sous le nom *visuGraphviz.py* et l'uploader sur *nsibrantly.fr*.

```
graphe.edge('A', 'B')
graphe.edge('A', 'C')
graphe.edge('B', 'C')
graphe.edge('B', 'D')
graphe.edge('C', 'D')
graphe.edge('C', 'E')
graphe.edge('D', 'E')
graphe.view()
```

2- CLASSE Gmatrice QUI IMPLEMENTE UN GRAPHE AVEC DES MATRICES D'ADJACENCE :

On donne ci-dessous le script de la classe *Gmatrice* qui permettra d'implémenter des graphes en utilisant la technique des matrices d'adjacence.

```
from graphviz import Digraph , Graph
graphe = Graph(format = 'png' , filename = 'graphe')

class Gmatrice :
    def __init__(self , nd , oriente = False , pondere = False) :
        self.noeud = [str(nd)]
        self.matrice = [[0]]
```

⇒ Copier ce script dans un nouveau fichier que vous nommerez *implementationMatrice.py* , fichier qui sera à uploader sur *nsibrantly.fr* en fin de tp.

a. METHODE AJOUT ARETE():

On donne ci-dessous le script incomplet de la méthode *ajout_arete()* :

```
def ajout_arete(self,nd1,nd2,valeur = None) :
    nd1,nd2 = str(nd1) , str(nd2)
```

⇒ Compléter ce script pour obtenir les exécutions suivantes. Pour ne pas compliquer le code, on travaille sur des arbres non orientés et non pondérés.

```
>>> g = Gmatrice('A')

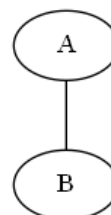
>>> g.noeud
['A']

>>> g.matrice
[[0]]
```

```
>>> g.ajout_arete('A','B')

>>> g.noeud
['A', 'B']

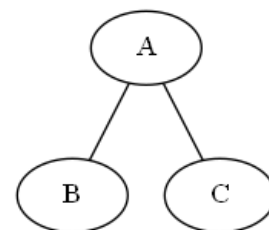
>>> g.matrice
[[0, 1], [1, 0]]
```

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$


```
>>> g.ajout_arete('A','C')

>>> g.noeud
['A', 'B', 'C']

>>> g.matrice
[[0, 1, 1], [1, 0, 0], [1, 0, 0]]
```

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$


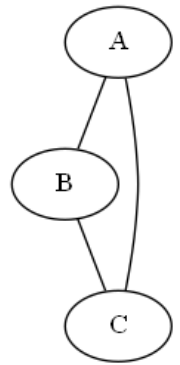
```

>>> g.ajout_arete('B','C')

>>> g.noeud
['A', 'B', 'C']

>>> g.matrice
[[0, 1, 1], [1, 0, 1], [1, 1, 0]]

```

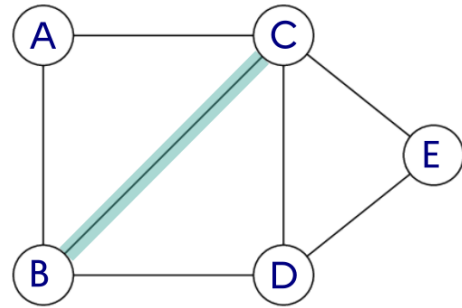
$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$


En exécutant les appels suivants, on modélise le graphe vu ci-dessous :

```

# Main
g1 = Gmatrice('A')
g1.ajout_arete('A','B')
g1.ajout_arete('A','C')
g1.ajout_arete('B','C')
g1.ajout_arete('B','D')
g1.ajout_arete('C','D')
g1.ajout_arete('C','E')
g1.ajout_arete('D','E')

```



```

def ajout_arete(self,nd1,nd2,valeur = None) :
    nd1,nd2 = str(nd1) , str(nd2)
    if nd1 not in self.noeud :
        self.noeud.append(nd1)
        for i in range(len(self.matrice)):
            self.matrice[i].append(0)
        l = [0 for i in range(len(self.noeud))]
        self.matrice.append(l)
    if nd2 not in self.noeud :
        self.noeud.append(nd2)
        for i in range(len(self.matrice)):
            self.matrice[i].append(0)
        l = [0 for i in range(len(self.noeud))]
        self.matrice.append(l)
    i1 = self.noeud.index(nd1)
    i2 = self.noeud.index(nd2)
    self.matrice[i1][i2] = 1
    self.matrice[i2][i1] = 1

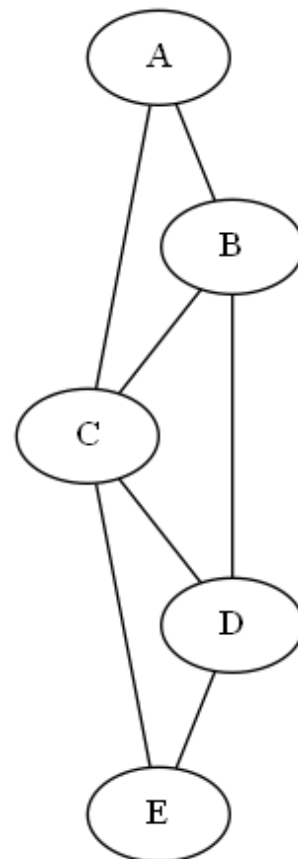
```

b. METHODE TRACE():

Ecrire le script de la méthode `trace()`. Elle permet de visualiser le graphe avec `graphviz`. L'exécution du programme principal ci-dessous, permet de visualiser le graphe :

```
# Main
g1 = Gmatrice('A')
g1.ajout_arete('A','B')
g1.ajout_arete('A','C')
g1.ajout_arete('B','C')
g1.ajout_arete('B','D')
g1.ajout_arete('C','D')
g1.ajout_arete('C','E')
g1.ajout_arete('D','E')

g1.trace()
```



```
def trace(self) :
    for i in range(len(self.matrice)) :
        for j in range(i , len(self.matrice)) :
            if self.matrice[i][j] == 1 :
                graphe.edge(self.noed[i] , self.noed[j])
    graphe.view()
```

3- CLASSE GLISTE QUI IMPLEMENTE UN GRAPHE AVEC DES LISTES D'ADJACENCE :

On donne ci-dessous le script de la classe `Gliste` qui permettra d'implémenter des graphes en utilisant la technique des listes d'adjacence.

```
from graphviz import Digraph , Graph
graphe = Graph(format = 'png' , filename = 'graphe')

class Gliste :
    def __init__(self, oriente = False) :
```

⇒ Copier ce script dans un nouveau fichier que vous nommerez `implementationListe.py`, fichier qui sera à uploader sur `nsibrantly.fr` en fin de tp.

a. METHODE AJOUT_ARETE():

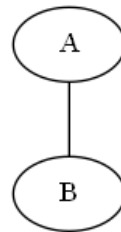
On donne ci-dessous le script incomplet de la méthode *ajout_arete()* :

```
def ajout_arete(self,nd1,nd2,valeur = None) :  
    nd1,nd2 = str(nd1) , str(nd2)
```

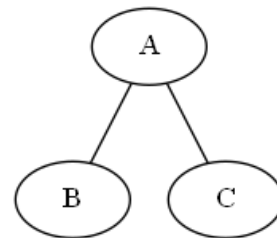
⇒ Compléter ce script pour obtenir les exécutions suivantes. Pour ne pas compliquer le code, on travaille sur des arbres non orientés et non pondérés.

```
>>> g1 = Gliste()  
>>> g1.dic  
{}
```

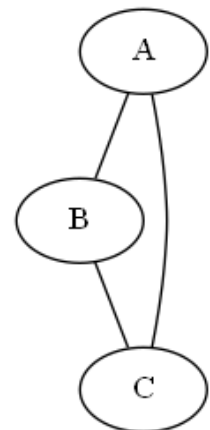
```
>>> g1.ajout_arete('A','B')  
>>> g1.dic  
{'A': ['B'], 'B': ['A']}
```



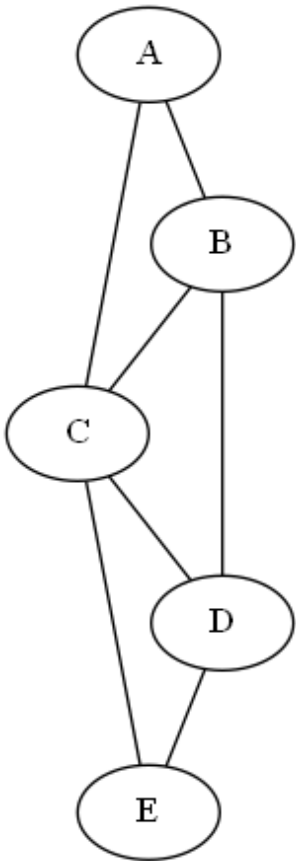
```
>>> g1.ajout_arete('A','C')  
>>> g1.dic  
{'A': ['B', 'C'], 'B': ['A'], 'C': ['A']}
```



```
>>> g1.ajout_arete('B','C')  
>>> g1.dic  
{'A': ['B', 'C'], 'B': ['A', 'C'], 'C': ['A', 'B']}
```



```
def ajout_arete(self,nd1,nd2,valeur = None) :  
    nd1,nd2 = str(nd1) , str(nd2)  
    if nd1 in self.dic :  
        self.dic[nd1].append(nd2)  
    else : self.dic[nd1] = [nd2]  
    if nd2 in self.dic :  
        self.dic[nd2].append(nd1)  
    else : self.dic[nd2] = [nd1]
```



b. METHODE TRACE():

Ecrire le script de la méthode `trace()`. Elle permet de visualiser le graphe avec `graphviz`. L'exécution du programme principal ci-dessous, permet de visualiser le graphe :

```
# Main
g1 = Gliste()
g1.ajout_arete('A', 'B')
g1.ajout_arete('A', 'C')
g1.ajout_arete('B', 'C')
g1.ajout_arete('B', 'D')
g1.ajout_arete('C', 'D')
g1.ajout_arete('C', 'E')
g1.ajout_arete('D', 'E')

g1.trace()
```

Info utile : Pour ne pas créer avec `graphviz` 2 fois une même arête, on peut mémoriser les couples `nd1, nd2` pour lesquels une arête a déjà été tracée. Il existe une entité en python appelé « le set » qui permet de créer des sortes de tuples mais pour lesquels l'ordre de rangement n'a pas d'importance. On utilise les accolades pour créer un set :

```
>>> s = {'A' , 'B'}

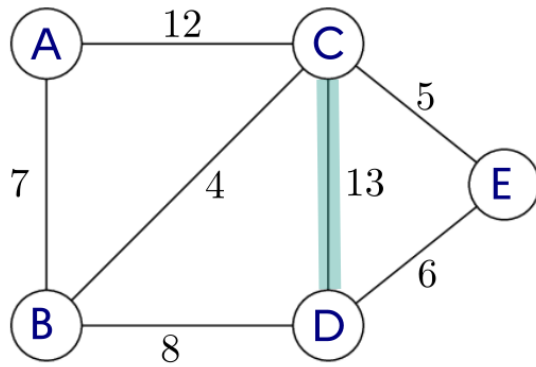
>>> s == {'B' , 'A'}
True
```

```
def trace(self) :
    deja_trace = []
    for nd1 in self.dic :
        for nd2 in self.dic[nd1] :
            if {nd1,nd2} not in deja_trace :
                graphe.edge(nd1,nd2)
                deja_trace.append({nd1,nd2})
    graphe.view()
```

4- ENRICHISSEMENT DE LA CLASSE GMATRICE :

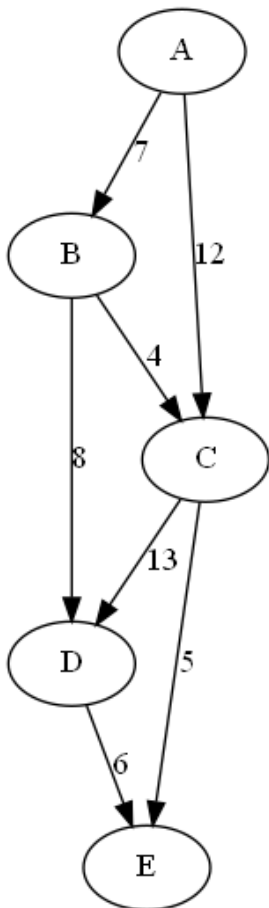
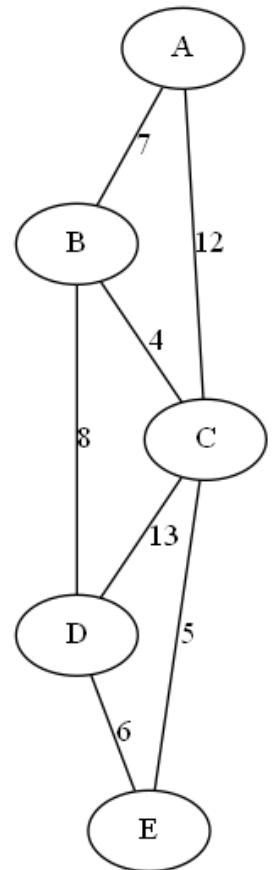
La méthode constructeur de la classe `Gmatrice` a 2 paramètres que l'on a pas exploiter jusqu'à présent. Si `orienté` prend la valeur `True` le graphe devient un graphe orienté. Si `pondere` prend la valeur `True`, le graphe devient pondéré.

```
class Gmatrice :
    def __init__(self , nd , oriente = False , pondere = False) :
        self.noeud = [str(nd)]
        self.matrice = [[0]]
```



```
# Main
g1 = Gmatrice('A', False, True)
g1.ajout_arete('A', 'B', 7)
g1.ajout_arete('A', 'C', 12)
g1.ajout_arete('B', 'C', 4)
g1.ajout_arete('B', 'D', 8)
g1.ajout_arete('C', 'D', 13)
g1.ajout_arete('C', 'E', 5)
g1.ajout_arete('D', 'E', 6)

g1.trace()
```



```
# Main
g1 = Gmatrice('A', True, True)
g1.ajout_arete('A', 'B', 7)
g1.ajout_arete('A', 'C', 12)
g1.ajout_arete('B', 'C', 4)
g1.ajout_arete('B', 'D', 8)
g1.ajout_arete('C', 'D', 13)
g1.ajout_arete('C', 'E', 5)
g1.ajout_arete('D', 'E', 6)

g1.trace()
```

```

class Gmatrice :
    def __init__(self , nd , oriente = False , pondere = False) :
        self.oriente = oriente
        self.pondere = pondere
        if oriente == False :
            self.graphe = Graph(format = 'png' , filename = 'graphe')
        else :
            self.graphe = Digraph(format = 'png' , filename = 'graphe')
        self.noed = [str(nd)]
        self.matrice = [[0]]

    def ajout_arete(self,nd1,nd2,valeur = None) :
        nd1,nd2 = str(nd1) , str(nd2)
        if nd1 not in self.noed :
            self.noed.append(nd1)
            for i in range(len(self.matrice)):
                self.matrice[i].append(0)
            l = [0 for i in range(len(self.noed))]
            self.matrice.append(l)
        if nd2 not in self.noed :
            self.noed.append(nd2)
            for i in range(len(self.matrice)):
                self.matrice[i].append(0)
            l = [0 for i in range(len(self.noed))]
            self.matrice.append(l)
        i1 = self.noed.index(nd1)
        i2 = self.noed.index(nd2)
        if valeur == None : valeur = 1
        self.matrice[i1][i2] = valeur
        if self.oriente == False : self.matrice[i2][i1] = valeur
        print(self.matrice)

    def trace(self) :
        for i in range(len(self.matrice)) :
            if self.oriente == False : iDeb = i
            else : iDeb = 0
            for j in range(iDeb , len(self.matrice)) :
                if self.pondere == False and self.matrice[i][j] == 1 :
                    self.graphe.edge(self.noed[i] , self.noed[j])
                if self.pondere == True and self.matrice[i][j] != 0 :
                    self.graphe.edge(self.noed[i] , self.noed[j] , str(self.matrice[i][j]))

        self.graphe.view()

```

```

# Main
g1 = Gmatrice('A',True,True)
g1.ajout_arete('A','B',7)
g1.ajout_arete('A','C',12)
g1.ajout_arete('B','C',4)
g1.ajout_arete('B','D',8)
g1.ajout_arete('C','D',13)
g1.ajout_arete('C','E',5)
g1.ajout_arete('D','E',6)

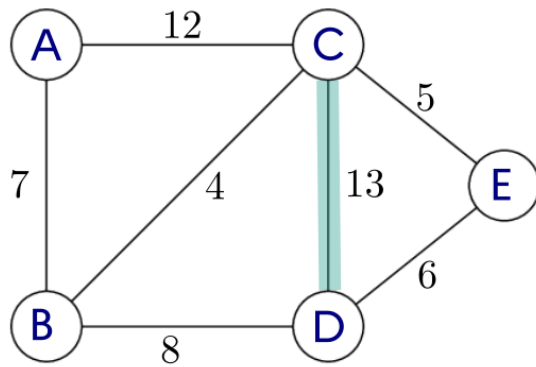
g1.trace()

```

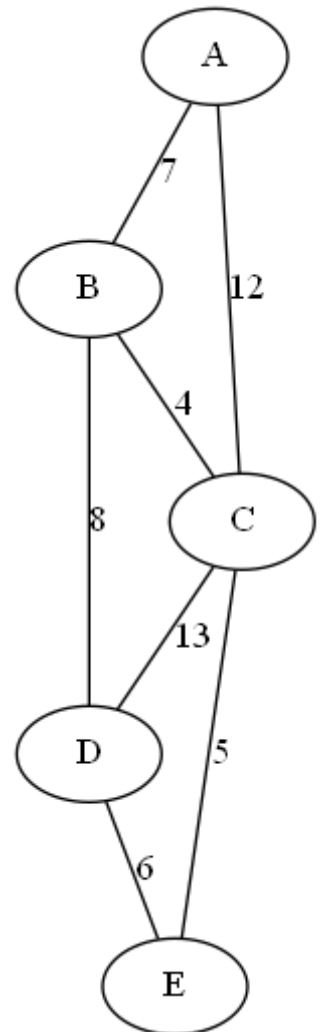

5- ENRICHISSEMENT DE LA CLASSE GLISTE :

La méthode constructeur de la classe Gmatrice a 2 paramètres que l'on a pas exploiter jusqu'à présent. Si *orienté* prend la valeur *True* le graphe devient un graphe orienté. Si *pondere* prend la valeur *True*, le graphe devient pondéré.

```
class Gliste :  
    def __init__(self, oriente = False) :  
        self.dic = {}
```



```
# Main  
g1 = Gliste(False)  
g1.ajout_arete('A', 'B', 7)  
g1.ajout_arete('A', 'C', 12)  
g1.ajout_arete('B', 'C', 4)  
g1.ajout_arete('B', 'D', 8)  
g1.ajout_arete('C', 'D', 13)  
g1.ajout_arete('C', 'E', 5)  
g1.ajout_arete('D', 'E', 6)  
  
g1.trace()
```



```

from graphviz import Digraph , Graph

class Gliste :
    def __init__(self, oriente = False) :
        self.oriente = oriente
        if oriente == False :
            self.graphe = Graph(format = 'png' , filename = 'graphe')
        else :
            self.graphe = Digraph(format = 'png' , filename = 'graphe')
        self.dic = {}

    def ajout_arete(self,nd1,nd2,valeur = None) :
        nd1,nd2 = str(nd1) , str(nd2)
        if nd1 in self.dic :
            self.dic[nd1].append((nd2,valeur))
        else : self.dic[nd1] = [(nd2,valeur)]
        if nd2 in self.dic :
            self.dic[nd2].append((nd1,valeur))
        else : self.dic[nd2] = [(nd1,valeur)]

    def trace(self) :
        deja_trace = []
        for nd1 in self.dic :
            for nd2 in self.dic[nd1] :
                if {nd1,nd2[0]} not in deja_trace or self.oriente :
                    v = ''
                    if nd2[1] != None : v = str(nd2[1])
                    self.graphe.edge(nd1,nd2[0],v)
                    deja_trace.append({nd1,nd2[0]})
        self.graphe.view()

# Main
g1 = Gliste(False)
g1.ajout_arete('A','B',7)
g1.ajout_arete('A','C',12)
g1.ajout_arete('B','C',4)
g1.ajout_arete('B','D',8)
g1.ajout_arete('C','D',13)
g1.ajout_arete('C','E',5)
g1.ajout_arete('D','E',6)

g1.trace()

```