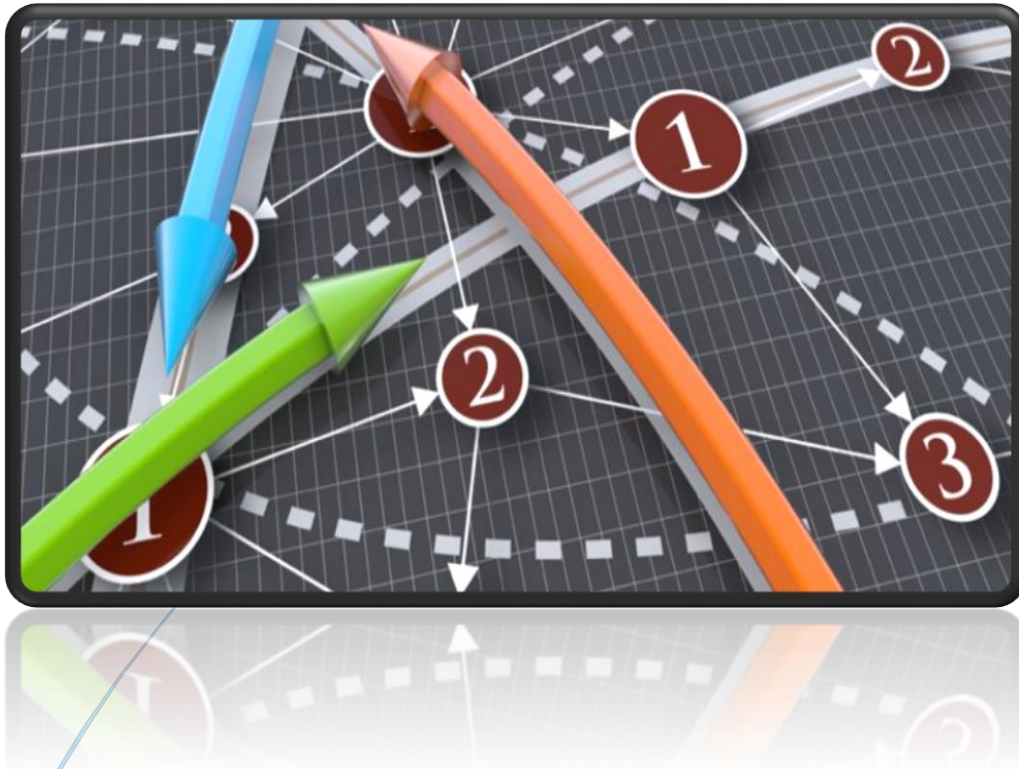


1/2/2023

# COMPTE RENDU TP RECHERCHE OPERATIONNELLE

CI1 : GEM-MSEI



Réalisé par :

**SALH EDDIN MAIMOUNI**

**HAMZA CHAIB**

**ES-SALHY MOHAMED**

Encadré par :

**Pr. Abdessamad KAMOISS**

**Année universitaire 2022-2023**

# REMERCIEMENT

À notre professeur Mr. KAMMOUS.

Nous tenons à vous exprimer notre profonde gratitude pour votre engagement et votre dévouement à notre égard. Vous êtes un professeur exceptionnel, toujours prêt à nous aider et à nous soutenir dans nos travaux pratiques et dans le cours. Votre patience, votre expertise et votre disponibilité ont été précieuses pour notre progression et notre réussite dans ce travail. Nous vous remercions pour votre précieuse contribution et espérons pouvoir continuer à bénéficier de votre enseignement.

Sincèrement.

# TABLE DES MATIERES

<b>I. Introduction :</b>	1
<b>II. Manipulation des graphes avec Python :</b>	2
<b>A. Exemple d'un graphe symétrique valué :</b>	2
<b>B. Implémentation de la matrice d'adjacence avec python :</b>	2
<b>C. Détermination des degrés des sommets d'un graphe :</b>	3
<b>D. Type du graph « eulérien, semi eulérien ou ni eulérien ni semi eulérien » :</b>	5
<b>E. Implémentation d'algorithme de Dijkstra :</b>	6
<b>F. Détermination de la matrice d'adjacence d'arborescence :</b>	8
<b>G. Traçage d'un arborescence en python « nx + plt » :</b>	9
<b>H. Algorithme de Floyd-Warshall :</b>	10
1. Présentation :	10
2. Implémentation d'algorithme de FW en Python :	11
<b>I. Implémentation de l'algorithme de Bellman Ford :</b>	12
<b>J. Exemple d'un graphe dont la longueur du chemin d'un sommet à un autre est le produit des poids des arcs de ce chemin :</b>	14
<b>K. Utilisation de l'algorithme de Bellman Ford pour le plus long chemin et produit:</b>	15
<b>L. Conclusion :</b>	18
<b>III. Application de traitement des graphes :</b>	19
<b>A. Introduction.</b>	19
<b>B. Vue principale.</b>	20
<b>C. Description de vue principale.</b>	21
<b>D. Alerts :</b>	22
<b>E. Dessiner un graphe.</b>	22
<b>F. Conclusion :</b>	23
<b>IV. Conclusion générale :</b>	24

# LISTE DES FIGURES

Figure 1 : Graphe G symétrique valué.....	2
Figure 2 : Code d'affichage de la matrice d'adjacence du graphe G. ....	3
Figure 3 : Résultat obtenu de la matrice d'adjacence.....	3
Figure 4: Méthode degré du graphe non orienté non valué.....	4
Figure 5:Methode degré du graphe orienté non valué.....	4
Figure 6: Méthode degré du graphe non orienté valué.....	4
Figure 7: Méthode degré du graphe orienté valué.....	4
Figure 8: Les degrés de chaque sommet de graphe G.....	5
Figure 9: Méthode qui détermine les types de graphe eulérien ou semi-eulérien.....	5
Figure 10: Résultat d'exécution des méthodes E(self) et ES (self) sur le graph G. ....	6
Figure 11 : Code d'implémentation d'algorithme de Dijkstra.....	7
Figure 12 : Résultat d'exécution de Dijkstra (M, S) sur le graphe G depuis le sommet 0.	7
Figure 13 : Code d'affichage de la matrice d'adjacence d'arborescence : .....	8
Figure 14 : Matrice adjacente arborescence de graphes G.....	9
Figure 15: Code d'affichage de l'arborescence : .....	9
Figure 16 : Arborescence de graphes G depuis le sommet 0. ....	10
Figure 17 : Code d'algorithme de FW en python. ....	11
Figure 18 : Exécution de Floyd Warshall sur le graph G. ....	11
Figure 19: Code d'algorithme de Bellman Ford. ....	13
Figure 20:Résultat d'exécution de Bellman Ford Standard sur le graphe G depuis le sommet 0.....	13
Figure 21: Graphe G2 orienté valué.....	14
Figure 22: Graph G2 après l'application de log ( poids ) .....	16
Figure 23: Code de Bellman Ford ; produit + plus long chemin .....	17
Figure 24: Résultat d'exécution de Bellman Ford produit + plus long chemin sur le graphe G2 depuis le sommet 0 .....	17
Figure 25: Arborescence du plus long chemin de graph G2 depuis le sommet 0.....	18
Figure 26 : Vue principale de l'application.....	20
Figure 27: dessineur des graphes.....	23

## **I. Introduction :**

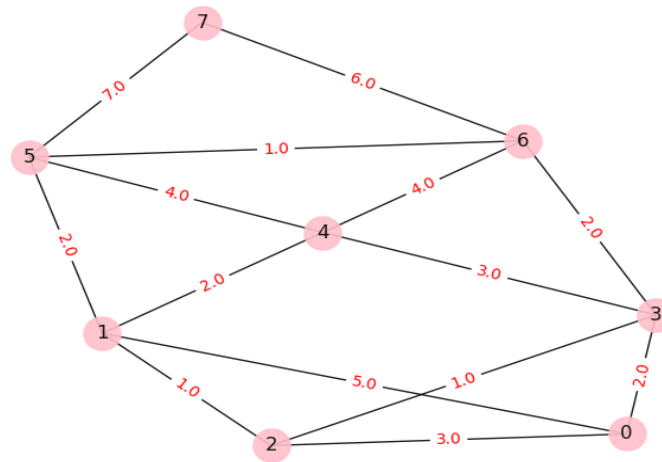
Python est un langage de programmation largement utilisé dans de nombreux domaines, y compris la recherche opérationnelle. La recherche opérationnelle est une discipline qui vise à trouver des solutions optimales à des problèmes de gestion et de décision en utilisant des méthodes mathématiques et informatiques. Dans ce contexte, Python peut être utilisé pour résoudre des problèmes complexes tels que le calcul du plus court ou du plus long chemin, l'optimisation de la production ou la gestion des ressources.

En raison de sa flexibilité, de sa puissance de calcul et de sa communauté active de développeurs, Python est devenu un choix populaire pour la recherche opérationnelle. De nombreux outils et bibliothèques dédiés à l'optimisation sont disponibles pour Python, ce qui en fait un langage idéal pour la modélisation et la résolution de problèmes complexes.

Dans ce TP, nous allons étudier l'implémentation de Python dans la recherche opérationnelle en nous concentrant sur les graphes : le calcul du plus court chemin et le calcul du plus long chemin.... Nous verrons comment utiliser Python et les bibliothèques dédiées pour modéliser ces problèmes et trouver des solutions optimales.

## II. Manipulation des graphes avec Python :

### A. Exemple d'un graphe symétrique valué :



*Figure 1 : Graphe G symétrique valué*

La matrice d'adjacence du graphe ci-dessus :

$$\begin{pmatrix} 0 & 5 & 3 & 2 & +\infty & +\infty & +\infty & +\infty \\ 5 & 0 & 1 & +\infty & 2 & 2 & +\infty & +\infty \\ 3 & 1 & 0 & 1 & +\infty & +\infty & +\infty & +\infty \\ 2 & +\infty & 1 & 0 & 3 & +\infty & 2 & +\infty \\ +\infty & 2 & +\infty & 3 & 0 & 4 & 4 & +\infty \\ +\infty & 2 & +\infty & +\infty & 4 & 0 & 1 & 7 \\ +\infty & +\infty & +\infty & 2 & 4 & 1 & 0 & 6 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 6 & 0 \end{pmatrix}$$

### B. Implémentation de la matrice d'adjacence avec python :

Pour réaliser cette phase de travaux pratiques en utilisant le langage Python, vous devrez tout d'abord donner la matrice d'adjacence du graphe G. Cette matrice est un tableau de n lignes et m colonnes, où n est le nombre de sommets du graphe et m est le nombre d'arêtes. Chaque case (i, j) de la matrice contient le poids (la valeur portée par un arrêt ou un arc entre deux sommets) de l'arête reliant le sommet i au sommet j. Pour implémenter la matrice d'adjacence du graphe G en utilisant le module Numpy.

```
matric.py - C:/Users/LENOVO/matric.py (3.11.1)
File Edit Format Run Options Window Help
import numpy as np

A=[ [0,5,3,2,float("inf"),float("inf"),float("inf"),float("inf")],
    [5,0,1,float("inf"),2,2,float("inf"),float("inf")],
    [3,1,0,1,float("inf"),float("inf"),float("inf"),float("inf")],
    [2,float("inf"),1,0,3,float("inf"),2,float("inf")],
    [float("inf"),2,float("inf"),3,0,4,4,float("inf")],
    [float("inf"),2,float("inf"),float("inf"),4,0,1,7],
    [float("inf"),float("inf"),float("inf"),2,4,1,0,6],
    [float("inf"),float("inf"),float("inf"),float("inf"),float("inf"),7,6,0]]

G = np.array(A)
print("La matrice d'adjacence", "\n", G)
```

Figure 2 : Code d'affichage de la matrice d'adjacence du graphe G.

```
IDLE Shell 3.11.1
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/LENOVO/matric.py =====
La matrice d'adjacence
[[ 0.  5.  3.  2. inf inf inf inf]
 [ 5.  0.  1. inf 2. 2. inf inf]
 [ 3.  1.  0.  1. inf inf inf inf]
 [ 2. inf 1.  0.  3. inf 2. inf]
 [inf 2. inf 3.  0.  4.  4. inf]
 [inf 2. inf inf 4.  0.  1.  7.]
 [inf inf inf 2.  4.  1.  0.  6.]
 [inf inf inf inf inf 7.  6.  0.]]
>>>
```

Figure 3 : Résultat obtenu de la matrice d'adjacence.

### C. Détermination des degrés des sommets d'un graphe :

Pour écrire une fonction **degre (M, S)** qui prend en argument la matrice d'adjacence M d'un graphe et un sommet S et qui retourne le degré de ce sommet, nous avons créé successivement quatre classes correspondent aux quatre types de graphes.

- Graphe non orienté non valué.
  - Graphe orienté non valué.
  - Graphe orienté valué.
  - Graphe non orienté valué.
- ✓ **Remarque :** les graphes orientés ont des degrés entrants et des degrés sortants, leur degré total est la somme de ces derniers.



```
def degre(self,A): # methode qui retourne le degre du sommet A
    D=0
    for e in self.matrix[A]:
        if e==1:
            D=D+1 # le degre est egale aux nombre des 1 dans la ligne corespondante à A
    print(A,"son degre",D)
    return D
```

**Figure 4: Méthode degré du graphe non orienté non valué**

```
def degreS(self,A): # methode qui renvoi le degre sortant d'un sommet
    D=0
    for e in self.matrix[A]:
        if e==1:
            D=D+1 # si l'element dans la ligne de A egale à 1 donc incrementer le degre sortant
    return D # renvoyer le degre sortant

def degreE(self,A): # methode qui renvoi le degre sortant d'un sommet
    D=0
    Tmatrix=self.matrix.T
    for e in Tmatrix[A]:
        if e==1:
            D=D+1 # si l'element dans la colone de A egale à 1 donc incrementer le degre sortant
    return D # renvoyer le degre sortant

def degre(self,A):
    print(A,"son degre",D)
    return self.degreE(A)+self.degreS(A) # methode qui renvoi le degre totale du sommet A
```

**Figure 5: Méthode degré du graphe orienté non valué**

```
def degreS(self,A): # methode qui renvoi les degre sortant du sommet A
    D=0
    for e in self.matrix[A]:
        if e!=0 and e!=float('inf'):
            D=D+1 # si l'element non nulle et finie donc incrementer le degre
    return D

def degreE(self,A): # methode qui renvoi les degre entrant du sommet A
    D=0
    for e in self.matrix.T[A]: # maintenant on travail avec la matrice transposé
        if e!=0 and e!=float('inf'):
            D=D+1 # si l'element non nulle et finie donc incrementer le degre
    return D

def degre(self,A): # methode qui renvoi le degre totale du sommet A
    print(A,"son degre",D)
    return self.degreE(A)+self.degreS(A)
```

**Figure 7: Méthode degré du graphe orienté valué**

```
def degre(self,A): # methode qui renvoi le degre sortant d'un sommet
    D=0
    for e in self.matrix[A]:
        if e!=0 and e!=float("inf"):
            # si l'element dans la colone de A egale à un nombre non nulle et finie donc incrementer le degre sortant
            D=D+1
    print(A,"son degre",D)
    return D # renvoyer le degre sortant
```

**Figure 6: Méthode degré du graphe non orienté valué**



0	son degré	3
1	son degré	4
2	son degré	3
3	son degré	4
4	son degré	4
5	son degré	4
6	son degré	4
7	son degré	2

Ln: 23 Col: 13

Figure 8: Les degrés de chaque sommet de graphe G.

#### D. Type du graph « eulérien, semi eulérien ou ni eulérien ni semi eulérien » :

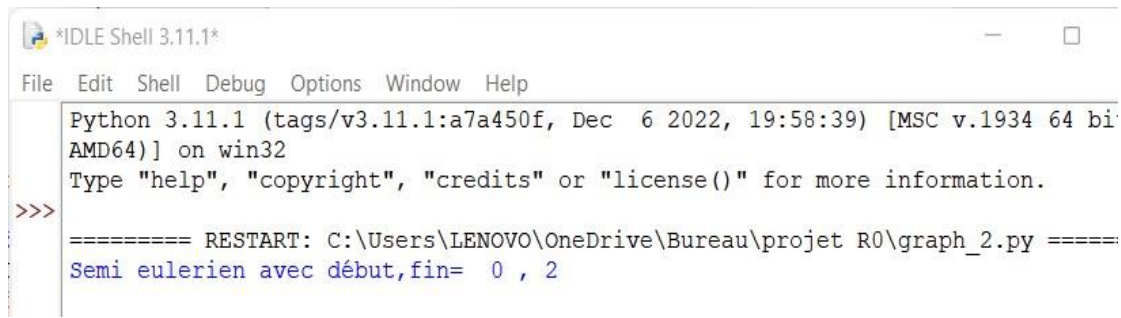
La première méthode **E(self)** renvoie True si le graphe est eulérien et False si no, pour la deuxième **SE (self)**, si le graphe est semi-eulérien elle renvoie une liste qui contient ses extrémités si non elle renvoie False.

- ✓ **Remarque :** La notion des graphes eulérien et semi-eulérien dépend seulement des graphes symétriques pour c'est deux méthodes.

```
def E(self): # methode qui test si le graph est eulerien ou non
    D=[]
    for i in range(len(self.matrix)):
        D.append(self.degre(i))
    if all(d%2==0 for d in D):
        print("le graph est eulerien")
        return True # si tous les degré sont paire donc True
    else:
        print("le graph n'est pas eulerien")
        return False # si non False

def SE(self): # methode qui test le graph s'il est semi eulerien
    D,X,impaire,extrimite=[],[],0,[]
    for i in range(len(self.matrix)):
        D.append(self.degre(i)) # ajouter à la list D les degré de chaque sommet
    for d in D:
        X.append(d%2) # ajouter à la liste X les restes de division de chaque degré
    for x,i in zip(X,range(len(X))):
        if x==1:
            extrimite.append(i+1)
            impaire=impaire+1
    if impaire==2:
        print("semi eulerien avec",extrimite)
        return extrimite # s'il ya exactement deux degré impaire et les autres paires donc le graph est semi eulerien
    else:
        print("n'est pas semi eulerien")
        return False # si non le graph n'est pas semi eulerien
```

Figure 9: Méthode qui détermine les types de graphe eulérien ou semi-eulérien.



```
*IDLE Shell 3.11.1*
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bi
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\LENOVO\OneDrive\Bureau\projet R0\graph_2.py =====
Semi eulerien avec debut, fin= 0 , 2
```

*Figure 10: Résultat d'exécution des méthodes  $E(self)$  et  $ES(self)$  sur le graph  $G$ .*

### **E. Implémentation d'algorithme de Dijkstra :**

La fonction **dijkstra** implémente l'algorithme de Dijkstra pour trouver le plus court chemin dans un graphe. L'algorithme de Dijkstra est un algorithme de recherche de graphe qui fonctionne en identifiant itérativement le sommet du graphe qui est le plus proche du sommet source et en mettant à jour la distance du sommet source vers tous les autres sommets du graphe sur la base de ce sommet.

L'algorithme commence par initialiser une table avec les distances du sommet source vers tous les autres sommets, et une table avec le sommet prédécesseur (c'est-à-dire le sommet qui le précède) pour chaque sommet dans le chemin du sommet source vers ce sommet. L'algorithme sélectionne alors itérativement le sommet qui n'est pas dans l'ensemble des sommets visités et qui a la plus petite distance du sommet source, et met à jour la distance du sommet source vers tous les autres sommets sur la base de ce sommet. Ce processus continue jusqu'à ce que tous les sommets aient été visités.

Dans cette implémentation, l'entrée matrix représente la matrice d'adjacence du graphe, et S est le sommet source. La fonction renvoie trois tables : **M\_tab**, **L\_tab** et **P\_tab**, qui représentent l'ensemble des sommets visités, la table des distances du sommet source vers tous les autres sommets, et la table des sommets prédécesseurs, respectivement

```

59 def dijkstra(matrix,S):
60     M_tab, L_tab, P_tab, Ind = [], [], [], 0 # 3 lists qu'on utilise pour retourner le tableau à afficher dans GUI
61     # initialisation
62     M = np.array([S]) # initialiser M comme un vecteur d'une seule dimension qui contient le point de début
63     M_tab.append(M) # ajouter M dans M_tab
64     matrix[S][S]=0 # initialiser la petite longueur de S vers S comme 0
65     L = np.array([matrix[S]]) # initialiser L comme la ligne de S dans la matrice adjacente
66     L_tab.append(L[0]) # ajouter L dans L_tab
67     p = [] # p c'est une list intermédiaire pour initialiser P
68     for e in L[0]: # e est la distance entre S et les autres sommets
69         if e<float("inf"):
70             p.append(S) # si e est finie alors S est le prédécesseur
71         else:
72             p.append(None) # si non S n'est pas un prédécesseur
73     P = np.array(p) # initialiser P
74     P_tab.append(P) # ajouter P dans P_tab
75     # traitement
76     for i in range(len(L[0])): # tant que M != V # boucle initial, i va de sommet 0 vers sommet n-1
77
78         # choisir x C V\M tq pour tout y C V\M, L[x]<=L[y]
79         Min_lst = [] # list intermédiaire qui va contenir le minimum pour chaque valeur de i
80         for j in range(len(L[0])): # j va de 0 à n-1
81             if j in M:
82                 pass # si non a traité le sommet ne faire aucune chose
83             else:
84                 Min_lst.append(L[0][j]) # si non ajouter sa distance à Min_lst
85         if Min_lst == []:
86             break # si Min_lst est vide alors on a traité tout les sommets donc sortir de la boucle
87         Min_lst = np.array(Min_lst) # Transformer Min_lst à un module numpy
88         Min_val = Min_lst.min() # affecter la distance minimal de Min_lst à Min_val
89         var = np.where(L[0]==Min_val)
90         try:
91             Index_min = int(var[0]) # affecter l'index de la valeur minimal à Index_min s'il y'un un seul minimal
92         except:
93             for l in range(len(var[0])): # si non choisir le premier minimum
94                 if not(int(var[0][l]) in M):
95                     Index_min = int(var[0][l])
96                     break
97             else:
98                 pass
99         M = np.append(M, [Index_min]) # ajouter l'index de la distance minimal au tableau M
100        M_tab.append(M) # ajouter M à M_tab
101
102        # L_t1 tableau L temporisé "c'est les plus petites distances entre le sommet S et les autres sommets à travers le premier sommet choisi "
103        L_t1 = matrix[Index_min]+Min_val
104        L_t2 = np.array([]) # L_t2 tableau L temporisé vide
105        for j in range(len(L[0])):
106            # ajouter à L_t2 la plus petite distance entre la distance déjà trouvée et la nouvelle distance trouvée
107            L_t2 = np.append(L_t2, [np.array([L[0][j], L_t1[j]]).min()])
108        L_tab.append(L_t2) # ajouter L_t2 à L_tab
109        L = np.array([L_t2]) # affecter L_t2 à L
110        P_t = np.array([]) # P_t tableau P temporisé
111        for j in range(len(L[0])):
112            if L_tab[i][j]==L_tab[i+1][j]:
113                P_t = np.append(P_t, P_tab[i][j]) # si on n'a pas modifié la distance donc reste le même prédécesseur
114            else:
115                P_t = np.append(P_t, [Index_min]) # si non le prédécesseur sera Index_min
116        P_tab.append(P_t) # ajouter P_t à P_tab
117        print(M_tab[-1], "M", L_tab[-1], "L", P_tab[-1], "P") # afficher le tableau de dijkstra
118    return [M_tab, L_tab, P_tab] # renvoyer une list qui est le tableau complet de dijkstra

```

Figure 11 : Code d'implémentation d'algorithme de Dijkstra.

```

Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>
===== RESTART: C:\Users\LENOVO\OneDrive\Bureau\projet R0\graph_2.py =====
[0] [[ 0.  5.  3.  2. inf inf inf inf]] [0 0 0 0 None None None None]
[0 3] M [ 0.  5.  3.  2.  5. inf  4. inf] L [0.0 0.0 0.0 0.0 3.0 None 3 None] P
[0 3 2] M [ 0.  4.  3.  2.  5. inf  4. inf] L [0.0 2.0 0.0 0.0 3.0 None 3 None] P
[0 3 2 1] M [ 0.  4.  3.  2.  5.  6.  4. inf] L [0.0 2.0 0.0 0.0 3.0 1.0 3.0 None] P
[0 3 2 1 6] M [ 0.  4.  3.  2.  5.  5.  4. 10.] L [0. 2. 0. 0. 3. 6. 3. 6.] P
[0 3 2 1 6 4] M [ 0.  4.  3.  2.  5.  5.  4. 10.] L [0. 2. 0. 0. 3. 6. 3. 6.] P
[0 3 2 1 6 4 5] M [ 0.  4.  3.  2.  5.  5.  4. 10.] L [0. 2. 0. 0. 3. 6. 3. 6.] P
[0 3 2 1 6 4 5 7] M [ 0.  4.  3.  2.  5.  5.  4. 10.] L [0. 2. 0. 0. 3. 6. 3. 6.] P

```

Figure 12 : Résultat d'exécution de Dijkstra (M, S) sur le graphe G depuis le sommet 0.

## F. Détermination de la matrice d'adjacence d'arborescence :

La fonction **dijkstra\_arboressance** prend la table de plus courte distance et de prédécesseur obtenues de la fonction **dijkstra** et construire une matrice d'adjacence pour une arborescence (un arbre orienté) sur la base de ces tables.

Une arborescence est un arbre orienté qui a un chemin unique de la racine (le sommet source) vers chaque autre sommet de l'arbre. Elle peut être représentée sous forme de matrice d'adjacence, où chaque élément (i, j) représente le poids de l'arête orientée de i vers j. Dans cette implémentation, la matrice d'adjacence de l'arborescence est construite en parcourant les tables de distance et de prédécesseur, et en définissant l'élément (i, j) sur la distance entre les sommets i et son sommet prédécesseur j si j est le prédécesseur de i. Sinon, l'élément (i, j) est défini sur l'infini.

Les entrées **L** et **P** sont les tables de distance et de prédécesseur, respectivement, et Matrice est la matrice d'adjacence du graphe original. La fonction renvoie la transposée de la matrice d'adjacence de l'arborescence.

**Remarque :** cet algorithme est valable si on donne un tableau **L** et **P** soit depuis l'algorithme de dijkstra ou l'algorithme de Bellman Ford.

```

1454 # algorithm de passage de la derniere ligne d'algorithme de dijksta vers l'arboressance
1455 def dijkstra_arboressance(L,P,Matrice):
1456     # L et P sont la derniere ligne de tableaux de dijkstra et Matrice c'est la matrice agjacente du graph
1457     ARBR=[] # initialiser ARBR
1458     # pour dst dans les distance et pred dans les predecesseurs et i comme index
1459     for dst,Pred,i in zip(L,P,range(len(L))):
1460         ARBR.append([])
1461         for j in range(len(L)):
1462             try: #essayer
1463                 if j == int(Pred)-1:
1464                     ARBR[i].append(Matrice[j][i]) # si j = predecesseur ajouter au ARBR la distance entre eux
1465             else:
1466                 ARBR[i].append(float('inf')) # si non ajouter infinie
1467             except:
1468                 ARBR[i].append(float('inf')) # s'il y'a un erreur ajouter linfinie
1469     # ARBR est maintenant la matrice adjacente de l'arboressance
1470     return np.array(ARBR).T

```

**Figure 13 : Code d'affichage de la matrice d'adjacence d'arborescence :**



```
*IDLE Shell 3.11.1*
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f,
AMD64) on win32
Type "help", "copyright", "credits" <
>>>
===== RESTART: C:\Users\LENOVO\Or
[[ 0. inf  3.  2. inf inf inf inf]
 [inf inf inf inf inf inf inf inf]
 [inf  1. inf inf inf inf inf inf]
 [inf inf inf inf  3. inf  2. inf]
 [inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf  1. inf  6.]
 [inf inf inf inf inf inf inf inf]]
```

Figure 14 : Matrice adjacente arborescence de graphes G.

### G. Tracage d'un arborescence en python « nx + plt » :

La fonction ci-dessous **trace\_arborescence (ARBR,matrice)** à partir d'une matrice d'adjacence **matrice** et une matrice **ARBR** d'arborescence. L'arborescence est créée depuis la matrice renvoyer par la fonction **dijkstra\_arboressance** en utilisant la bibliothèque **networkx**, et est tracée en utilisant les fonctions **nx.draw** et **nx.draw\_networkx\_edge\_labels** de **networkx**. La fonction commence par créer un objet **DiGraph** vide en utilisant **nx.DiGraph()**. Elle initialise également un dictionnaire **edge\_labels** qui stockera les étiquettes c'est-à-dire les poids des arêtes du graphe. Ensuite, la fonction parcourt la matrice d'adjacence **matrice** et ajoute les sommets et les arêtes au graphe **g** en utilisant les méthodes **add\_node** et **add\_edge** de **g**. Si un sommet **i** est relié à un sommet **j** par une arête de poids **a[i][j]**, cette arête est ajoutée au graphe **g** et son poids est ajouté au dictionnaire **edge\_labels**. Enfin, la fonction utilise la fonction **nx.draw** pour tracer le graphe **g**, et utilise la fonction **nx.draw\_networkx\_edge\_labels** pour ajouter les étiquettes des arêtes du graphe.

La fonction affiche le résultat en utilisant la fonction **plt.show** de **matplotlib**.

```
4 import networkx as nx
5 import matplotlib.pyplot as plt
6
7 def trace_arboressance (ARBR,matrice):
8     g=GraphOV(ARBR) # g est l'arboressance -> graph orienté valué
9     n = g.Ordre() # n est l'ordre de ce graph
10    a = g.matrix # a est sa matrice
11    g = nx.DiGraph() # g maintenant est Directer Graph depuis la biblioteque networkx
12    edge_labels={} # initialiser le dictionnaire des distances entre les sommets
13    for i in range(n):
14        for j in range(n):
15            if a[i][j] <float("inf"):
16                if i==j and A[i][i]==0:
17                    pass # si nous somme dans la diagonal pass et A[i][i]==0 ne faire aucune chose
18                elif i==j and a[i][j]!=0: # si nous somme dans la diagonal pass et a[i][i]!=0
19                    g.add_edge(i,j) # ajouter les sommet
20                    edge_labels[(i,j)]=str(a[i][j]) # ajouter la distance entre les sommet
21                elif not((j,i) in edge_labels.keys()): # si on a deja ajouter la distance ne rajouter pas
22                    g.add_edge(i,j)
23                    edge_labels[(i,j)]=str(a[i][j])
24    pos = nx.spring_layout(g) # posiotions des sommets
25    # dessiner les sommet et les arcs
26    nx.draw(g, pos, edge_color='black', width=1, linewidths=1,node_size=500, node_color='pink', alpha=0.9,labels={node: node for node in g.nodes()})
27    nx.draw_networkx_edge_labels(g, pos, edge_labels = edge_labels,font_color='red')
28    plt.show() #afficher le graph
```

Figure 15: Code d'affichage de l'arborescence :

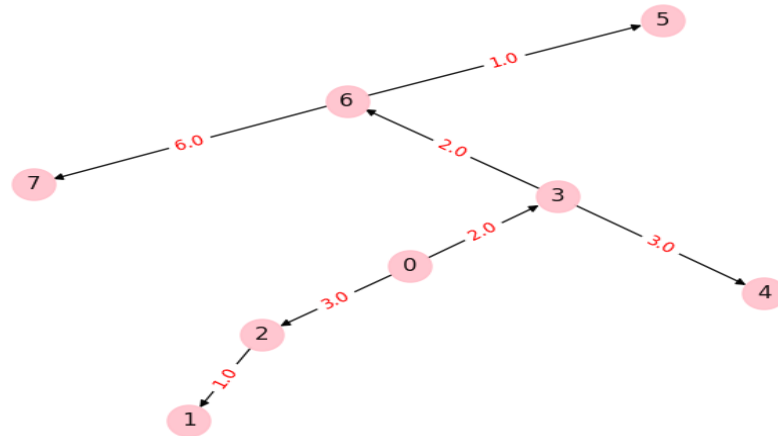


Figure 16 : Arborescence de graphes G depuis le sommet 0.

## H. Algorithme de Floyd-Warshall :

### 1. Présentation :

L'algorithme de Floyd-Warshall est un algorithme de recherche de graphe qui permet de trouver les plus courts chemins entre toutes les paires de sommets dans un graphe pondéré (valué) avec des poids négatifs ou positifs. Il fonctionne en itérativement mettant à jour la distance entre chaque paire de sommets en utilisant la distance de chaque sommet intermédiaire entre ces deux sommets.

Pour utiliser l'algorithme de Floyd-Warshall, on commence par initialiser une table de distances avec les poids des arêtes du graphe. Ensuite, pour chaque sommet intermédiaire  $i$ , on met à jour la distance entre chaque paire de sommets  $(u, v)$  en utilisant la distance de  $i$  entre ces deux sommets. Si la distance obtenue est inférieure à la distance déjà enregistrée, elle est mise à jour. Ce processus est répété jusqu'à ce que tous les sommets aient été utilisés comme sommets intermédiaires.

A la fin de l'algorithme, la table de distances contient les plus courts chemins entre toutes les paires de sommets. Cependant, si le graph contient un circuit absorbant ceci rend l'algorithme de Floyd-Warshall inapplicable.

Voici l'algorithme de Floyd-Warshall :

Pour chaque sommet  $i$  de 0 à  $n-1$  :

Pour chaque sommet  $u$  de 0 à  $n-1$  :

Pour chaque sommet  $v$  de 0 à  $n-1$  :

Si la distance entre  $u$  et  $v$  est plus grande que la distance entre  $u$  et  $i$  plus la distance entre  $i$  et  $v$  :

Mettre à jour la distance entre  $u$  et  $v$  en utilisant la distance entre  $u$  et  $i$  plus la distance entre  $i$  et  $v$  ;

## 2. Implémentation d'algorithme de FW en Python :

Il s'agit d'une implémentation Python de l'algorithme de Floyd-Warshall pour trouver les plus courts chemins entre toutes les paires de sommets dans un graphe. L'entrée est une matrice d'adjacence **self.matrix** représentant le graphe.

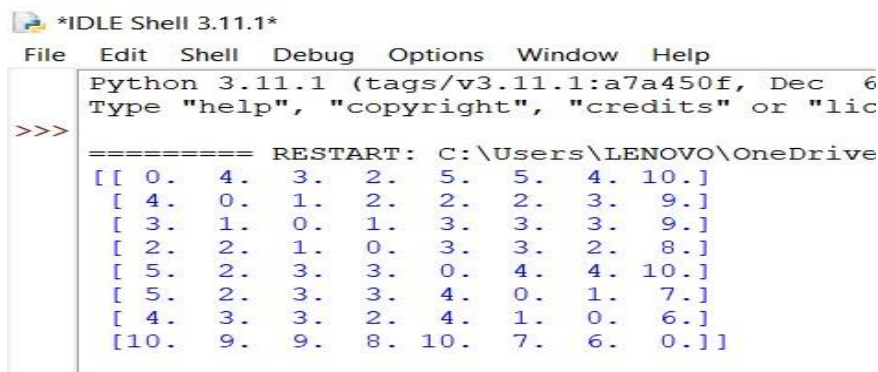
La fonction commence par initialiser une copie de la matrice d'adjacence **self.matrix**. Elle parcourt ensuite tous les sommets  $i$ ,  $j$  et  $k$  du graphe et met à jour la distance entre  $j$  et  $k$  en prenant le minimum de la distance actuelle et de la distance entre  $j$  et  $i$  plus la distance entre  $i$  et  $k$ . Ce processus est répété jusqu'à ce que toutes les paires de sommets aient été prises en compte.

```

498 def FloydWarshall(self): # mthode qui execute l'algorithme de Floyd Warshall
499     n = len(self.matrix)
500     M = copy.copy(self.matrix)
501     for i in range(n):
502         for j in range(n):
503             for k in range(n):
504                 # remplacer par la plus petite distance trouvé entre le sommet j et le sommet k
505                 M[j][k] = min(M[j][k], M[j][i] + M[i][k])
506     return M #renvoyer la matrice de floyd warshall

```

Figure 17 : Code d'algorithme de FW en python.



```

*IDLE Shell 3.11.1*
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6
Type "help", "copyright", "credits" or "lic
>>>
===== RESTART: C:\Users\LENOVO\OneDrive
[[ 0.  4.  3.  2.  5.  5.  4. 10.]
 [ 4.  0.  1.  2.  2.  2.  3.  9.]
 [ 3.  1.  0.  1.  3.  3.  3.  9.]
 [ 2.  2.  1.  0.  3.  3.  2.  8.]
 [ 5.  2.  3.  3.  0.  4.  4. 10.]
 [ 5.  2.  3.  3.  4.  0.  1.  7.]
 [ 4.  3.  3.  2.  4.  1.  0.  6.]
 [10.  9.  9.  8. 10.  7.  6.  0.]]

```

Figure 18 : Exécution de Floyd Warshall sur le graph G.



## I. Implémentation de l'algorithme de Bellman Ford :

On a implémenté l'algorithme de Bellman-Ford qui peut être utilisé pour trouver le chemin le plus court entre un sommet source unique et tous les autres sommets dans un graphe.

La fonction **initBF** initialise les tables **L** et **P** pour être utilisées dans la fonction **bellmanford**. La liste **L** stocke la longueur du chemin le plus court entre le sommet source et chaque sommet dans le graphe, et **P** stocke le sommet prédécesseur de chaque sommet dans le chemin le plus court. La fonction **bellmanford** met à jour les valeurs de **L** et **P** de manière itérative jusqu'à ce qu'elles terminent une itération, puis s'il y'a une mise à jour dans la table **L** il renvoi une nouvelle fois la fonction **bellmanford** d'une manière récursive jusqu'à stabilisation c.-à-d. qu'il n'y'a pas une mise à jour dans la table **L**.

La fonction prend en entrée une matrice d'adjacence **matrix** représentant le graphe, et les listes **sortie1** et **sortie2** qui sont utilisées pour stocker les valeurs intermédiaires de **L** et **P** lors de chaque itération de l'algorithme. S'il y'a un circuit absorbant, la fonction s'exécute jusqu'à une erreur de récursivité, si non elle retournera les listes **sortie1** et **sortie2** en tant que résultat.

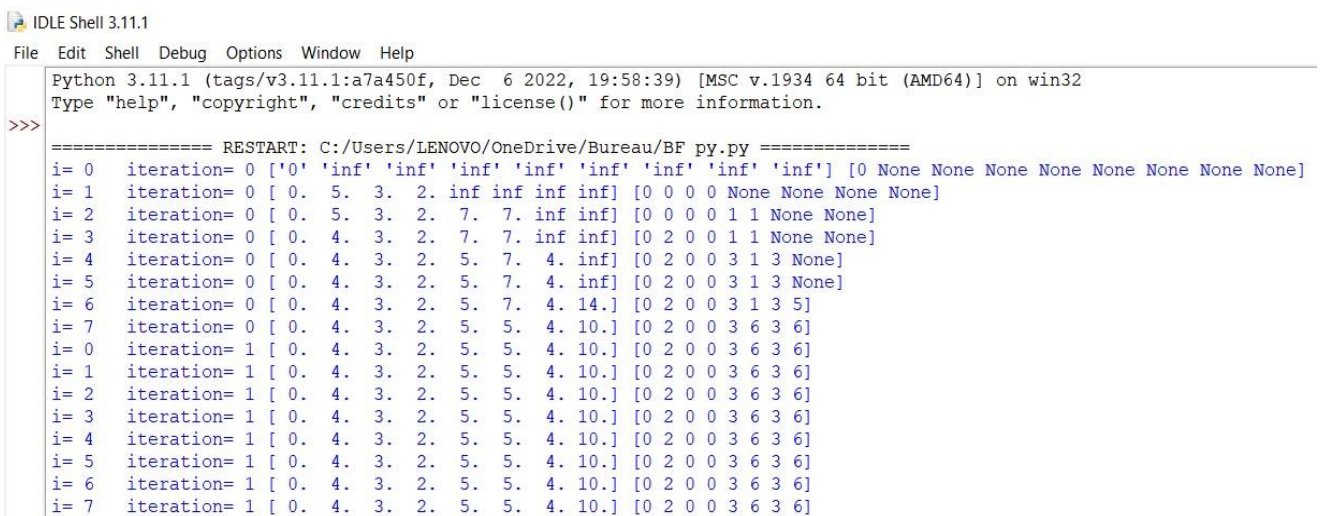
**Remarque :** un circuit absorbant dans le cas d'un problème de plus court chemin est un circuit qui est négatif, contrairement dans le plus long chemin il est positif.

```

34 def initBF(matrix,A):
35     L=[["inf"]*(A-1)+[0]+["inf"]*(len(matrix)-A)] # initialiser la table L
36     P=[["None"]*(A-1)+[A]+["None"]*(len(matrix)-A)] # initialiser la table P
37     return [L,P] # renvoyer la fonction Bellman_ford(L,P,Matrix,[],[])
38 iteration=0
39 # fonction bellmanford qui utilise les lists exemple: matrix = [[0,2,3],[inf,0,1],[3,inf,2]]
40 # fonction bellmanford qui prend L,P,matrix comme arguments + sortiel,sortie2 initialiser par des lists vides pour les renvoyer à la fin des itérations
41 def bellmanford(L,P,matrix,sortiel,sortie2):
42     T,l=[],[] # lists intermediaires
43     global iteration
44     l=L[0].copy()
45     for i in range(len(matrix)): # boucle initial
46         print("BF",i," ",iteration,np.array(L[-1]),np.array(P[-1])) # afficher la table de bellman ford
47         L.append([])
48         P.append([]) # ajouter des lists vides à L et P pour les remplis
49         for j in range(len(matrix)): # boucle secondaire
50             if L[i][i]=="inf": # si la distance L[i][i] est infinie ajouter L P de [i] et L P de [i+1] restent les memes puis sortir de la boucle
51                 L[i+1]=L[i]
52                 P[i+1]=P[i]
53                 break
54             elif L[i][j]=="inf" or (matrix[i][j]!="inf" and L[i][j]>matrix[i][j]+L[i][i]): #si la distance est grande
55                 if matrix[i][j]=="inf": # si il est infinie ajouter à L[i+1] la distance entre le sommet i et le sommet j et à P[i+1] None
56                     L[i+1].append(matrix[i][j])
57                     P[i+1].append(None)
58                 else: # si non ajouter à L[i+1] la distance entre le sommet i et le sommet j + la distance L[i][i]
59                     L[i+1].append(matrix[i][j]+L[i][i])
60                     if matrix[i][j]+L[i][i]<float("inf"):
61                         P[i+1].append(i+1)
62                     else:
63                         P[i+1].append(None)
64                 else: # si non ajouter à L[i+1] la distance entre le sommet i et le sommet j et à P[i+1] le predecesseur qui est déjà dans P[j][j]
65                     L[i+1].append(L[i][j])
66                     P[i+1].append(P[i][j])
67         if all(L[i]==l for i in range(len(matrix))): # si non a terminé une itération sans modification donc arreter
68             iteration = 0 # reset iteration
69             return [sortiel,sortie2]
70         else: # si non refaire la meme fonction
71             sortiel.append(L)
72             sortie2.append(P)
73             iteration +=1 # incrementer le nombre d'iteration
74     return bellmanford([L[-1]], [P[-1]],matrix,sortiel,sortie2)

```

**Figure 19: Code d'algorithme de Bellman Ford.**



```

===== RESTART: C:/Users/LENOVO/OneDrive/Bureau/BF py.py =====
i= 0 iteration= 0 [ 0. inf inf inf inf inf inf inf] [0 None None None None None None None]
i= 1 iteration= 0 [ 0. 5. 3. 2. inf inf inf inf] [0 0 0 0 None None None None]
i= 2 iteration= 0 [ 0. 5. 3. 2. 7. 7. inf inf] [0 0 0 0 1 1 None None]
i= 3 iteration= 0 [ 0. 4. 3. 2. 7. 7. inf inf] [0 2 0 0 1 1 None None]
i= 4 iteration= 0 [ 0. 4. 3. 2. 5. 7. 4. inf] [0 2 0 0 3 1 3 None]
i= 5 iteration= 0 [ 0. 4. 3. 2. 5. 7. 4. inf] [0 2 0 0 3 1 3 None]
i= 6 iteration= 0 [ 0. 4. 3. 2. 5. 7. 4. 14.] [0 2 0 0 3 1 3 5]
i= 7 iteration= 0 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 0 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 1 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 2 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 3 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 4 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 5 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 6 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]
i= 7 iteration= 1 [ 0. 4. 3. 2. 5. 5. 4. 10.] [0 2 0 0 3 6 3 6]

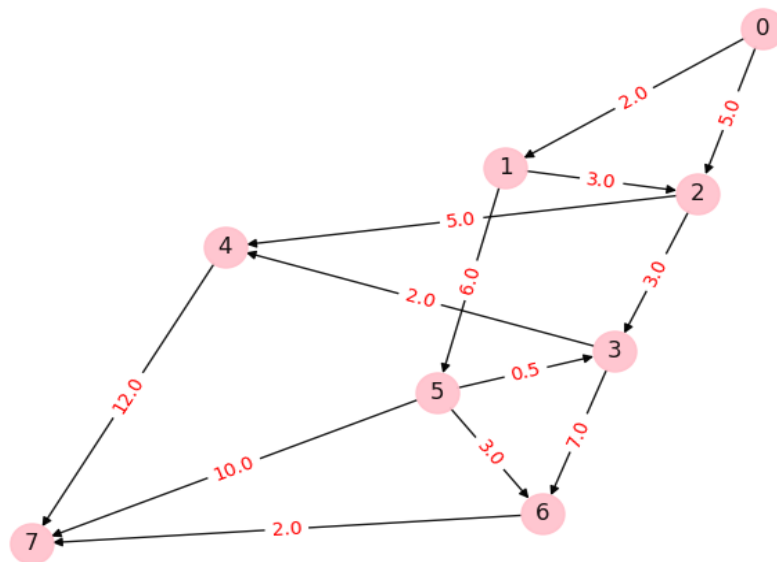
```

**Figure 20: Résultat d'exécution de Bellman Ford Standard sur le graphe G depuis le sommet 0.**

**J. Exemple d'un graphe dont la longueur du chemin d'un sommet à un autre est le produit des poids des arcs de ce chemin :**

Dans un parcours de certification, chaque étudiant dispose d'un score de 100 points au départ. Par la suite, il peut faire des transitions d'un niveau vers un autre niveau de certification selon les possibilités données par le graphe ci-dessous. A chaque transition son score est multiplié par la valeur présente sur l'arc. Par exemple, si un étudiant réalise le parcours 0 -> 1 -> 5 -> 7, son score final sera comme suit :

$$\text{score final} = 100 \times 2 \times 6 \times 10 = 12000.$$



*Figure 21: Graphe G2 orienté valué.*

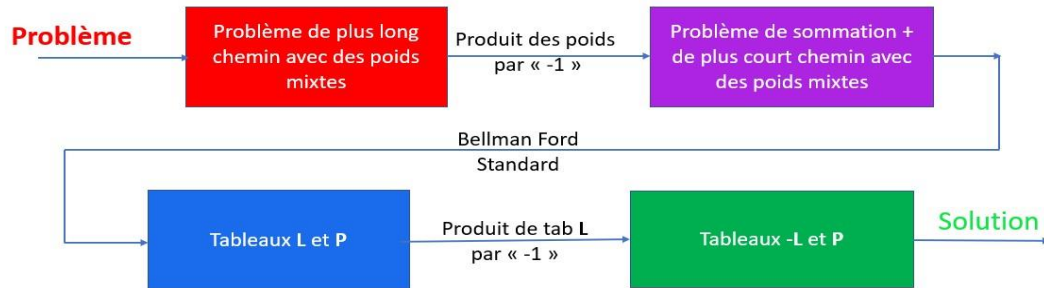
**Remarque :** le graph G2 ne contient pas de circuits.

La matrice d'adjacence du graphe G2 :

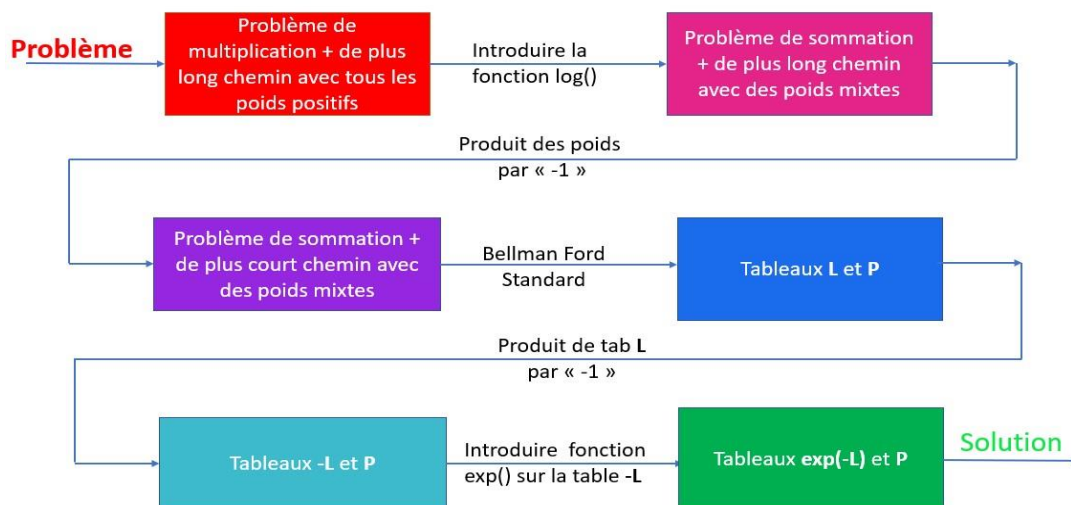
$$\begin{pmatrix} +\infty & 2 & 5 & +\infty & +\infty & +\infty & +\infty & +\infty \\ +\infty & +\infty & 3 & +\infty & +\infty & 6 & +\infty & +\infty \\ +\infty & +\infty & +\infty & 3 & 5 & +\infty & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & 2 & +\infty & 7 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 12 \\ +\infty & +\infty & +\infty & 0.5 & +\infty & +\infty & 3 & 10 \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 2 \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & +\infty \end{pmatrix}$$

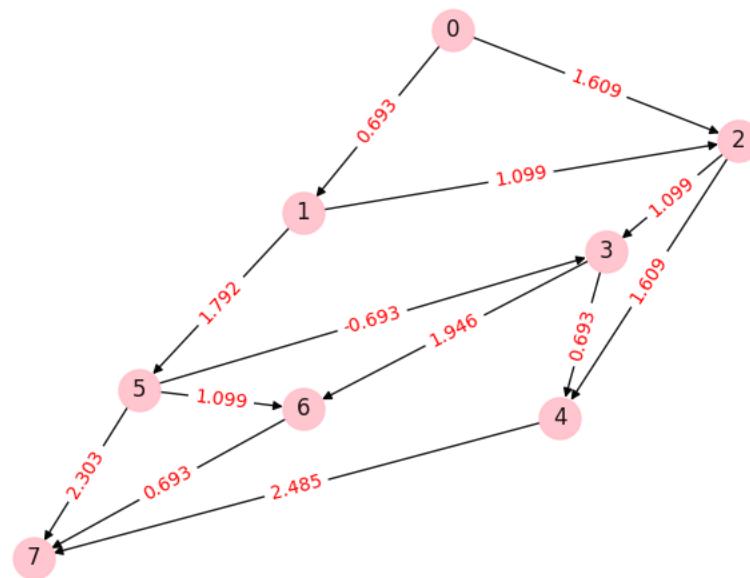
## K. Utilisation de l'algorithme de Bellman Ford pour le plus long chemin et produit:

Afin d'utiliser l'algorithme de Bellman Ford pour déterminer le plus long chemin depuis un sommet donné vers le reste des autres sommets, on suit la méthode décrite ci-dessous :



Si le graph est de la famille dont la longueur du chemin d'un sommet à un autre est le produit des poids des arcs de ce chemin et on cherche le plus long chemin alors on suit la méthode ci-dessous :





*Figure 22: Graph G2 après l'application de log ( poids )*

Après l'application de log ( poids ) sur les poids de graphe G2 on remarque qu'il y'a des nombres de signes mixtes et tel que ce circuit ne contient pas de circuits donc nous pouvons utiliser l'algorithme de Bellman Ford avec les modifications nécessaires qu'on a déjà décrit pour trouver le meilleur chemin.

Dans le code ci-dessous il y'a les fonctions suivants :

- **applog** : une fonction qui prend en argument une matrice d'adjacence **matrix** et introduire le  $\log(x)$  à ses elements.
- **appng** : une fonction qui prend en argument une matrice d'adjacence **matrix** et inversa le signe de ses elements.
- **appexp** : une fonction qui prend en argument une liste des distance **L** et applique l'exponentielle  $e^{\text{signe} \cdot x}$  à ses elements, selon le signe souaité, ici negatif.

Après on introduit le melange de ces fonctions avec l'algorithme de Bellman Ford à fin de trouver le plus court ou le plus long dans le cas de sommation entre les sommet ainsi que le cas



de produit, pour notre cas on a utilisé ce mélange à fin de trouver le meilleur score dans le probleme donner dans le graph G2 .

```

66 def applog(matrix): # fonction qui applique le logarithme sur la matrice
67     for i in range(len(matrix[0])):
68         for j in range(len(matrix[0])):
69             if not(matrix[i][j]=="inf"):
70                 try:
71                     # si le nombre n'est pas infinie et striquement positive appliquer le log
72                     matrix[i][j] = math.log(matrix[i][j])
73                 except:
74                     # si le nombre n'est pas striquement positive on peut pas appliquer le log
75                     print("erreur")
76                     return None
77             else: # si le nombre est infinie -> log(inf)=inf
78                 matrix[i][j] = "inf"
79     return matrix # renvoyer la matrice
80
81 def appng(matrix): # fonction qui inverse le signe de la matrice
82     for i in range(len(matrix[0])):
83         for j in range(len(matrix[0])):
84             if not(matrix[i][j]=="inf"):
85                 # si le nombre n'est pas infinie inverser son signe
86                 matrix[i][j] = -matrix[i][j]
87             else:
88                 # si non reste infinie
89                 matrix[i][j] = "inf"
90     return matrix # renvoyer la matrice
91
92 def appexp(L,signe): # fonction qui inverse le signe de la table L
93     for i in range(len(L)):
94         if not(L[i]=="inf"):
95             #adaptation avec les nombres entiers si la longueur est finie
96             if math.exp(signe*L[i])-int(math.exp(signe*L[i]))>0.5:
97                 L[i] = int(math.exp(signe*L[i]))+1 # majorer la distance
98             else:
99                 L[i] = int(math.exp(signe*L[i])) # minorer la distance
100         else:
101             L[i] = "inf" # reste infinie
102     return L # renvoyer la table L
103
104 matrice=applog(matrice) # appliquer le log sur la matrice
105 matrice=appng(matrice) # inverser le signe de la matrice
106 B = initBF(matrice,0) # initialiser pour bellman ford
107 B = bellmanford(B[0],B[1],matrice,[1],[1]) # appliquer l'algorithme de bellman ford
108 for i in range(len(B[0][0])):
109     B[0][0][i] = appexp(B[0][0][i],-1) # appliquer l'exp(-distance)
110 for i in range(len(B[0][0])):
111     print(B[0][0][i],B[1][0][i]) # afficher le resultat

```

Figure 23: Code de Bellman Ford ; produit + plus long chemin

```

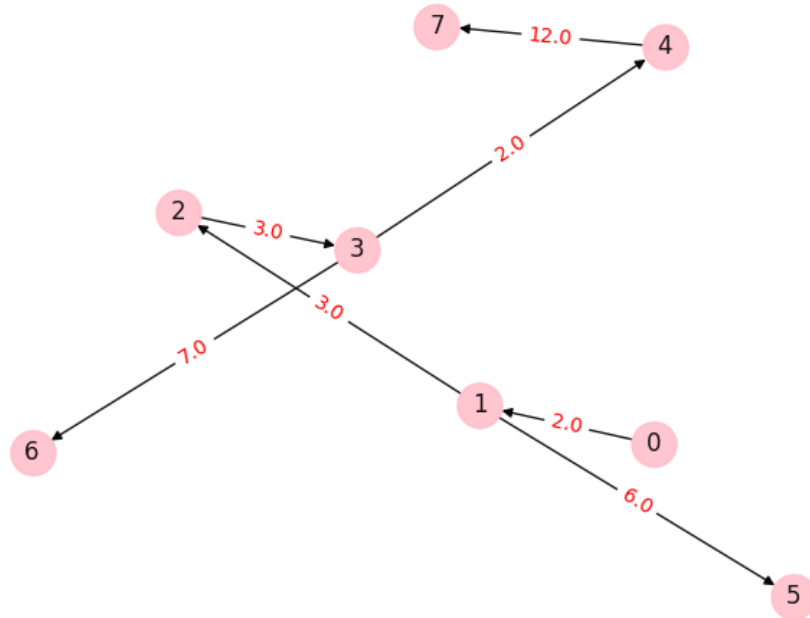
IDLE Shell 3.11.1
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/LENOVO/OneDrive/Bureau/BF py.py =====
[1, 'inf', 'inf', 'inf', 'inf', 'inf', 'inf', 'inf', 'inf'] [0, None, None, None, None, None, None, None, None]
[1, 2, 5, 'inf', 'inf', 'inf', 'inf', 'inf'] [0, 0, 0, None, None, None, None, None]
[1, 2, 6, 'inf', 'inf', 12, 'inf', 'inf'] [0, 0, 1, None, None, 1, None, None]
[1, 2, 6, 18, 30, 12, 'inf', 'inf'] [0, 0, 1, 2, 2, 1, None, None]
[1, 2, 6, 18, 36, 12, 126, 'inf'] [0, 0, 1, 2, 3, 1, 3, None]
[1, 2, 6, 18, 36, 12, 126, 432] [0, 0, 1, 2, 3, 1, 3, 4]
[1, 2, 6, 18, 36, 12, 126, 432] [0, 0, 1, 2, 3, 1, 3, 4]
[1, 2, 6, 18, 36, 12, 126, 432] [0, 0, 1, 2, 3, 1, 3, 4]
[1, 2, 6, 18, 36, 12, 126, 432] [0, 0, 1, 2, 3, 1, 3, 4]

```

Figure 24: Résultat d'exécution de Bellman Ford produit + plus long chemin sur le graphe G2 depuis le sommet 0

**Remarque :** il faut multiplier le résultat de la figure 24 avec le score initial afin de trouver le score final.

D'après la **figure 22** on peut utiliser la fonction **dijkstra\_arboressance** et la fonction **trace\_arboressance** à fin d'afficher l'arborescence de plus long chemin de sommet **0** vers les autres sommets, le résultat est le suivant :



*Figure 25: Arborescence du plus long chemin de graph G2 depuis le sommet 0.*

### L. Conclusion :

Selon les résultats obtenus il est possible d'utiliser l'algorithme de plus court chemin pour un graphe sans circuits absorbants de poids négatif. Dans le cas d'un graphe ne contenant que des circuits absorbants de poids négatif, l'algorithme de plus long chemin peut être utilisé. Si un graphe contient à la fois des circuits absorbants de poids négatifs et positifs, il n'est pas possible d'utiliser les algorithmes de plus court chemin ou de plus long chemin. Ces algorithmes ne peuvent être utilisés que pour les graphes ne contenant pas de circuits absorbants.



### III. Application de traitement des graphes :

#### A. Introduction.

Dans cet axe, nous résumons tout ce que nous avons fait dans notre TP dans une seule application a base de langage Python qui regroupe toutes les implémentations sur la théorie des graphes. Notre application a pour objectif de calculer le plus court et le plus long chemin dans un graphe à l'aide des algorithmes de Bellman-Ford et Dijkstra. Elle permet également de visualiser le graphe grâce aux bibliothèques **matplotlib**, **networkx**, **Tkinter** et **numpy**, ainsi que de fournir un dictionnaire de successeurs et de prédécesseurs pour chaque nœud.

Cette application est utile dans divers domaines où il est nécessaire de trouver le meilleur chemin entre deux points dans un graphe, comme la navigation, la gestion de réseaux ou l'optimisation de routes. Elle offre également la possibilité de mieux comprendre la structure d'un graphe en le visualisant de manière claire et concise.

## B. Vue principale.

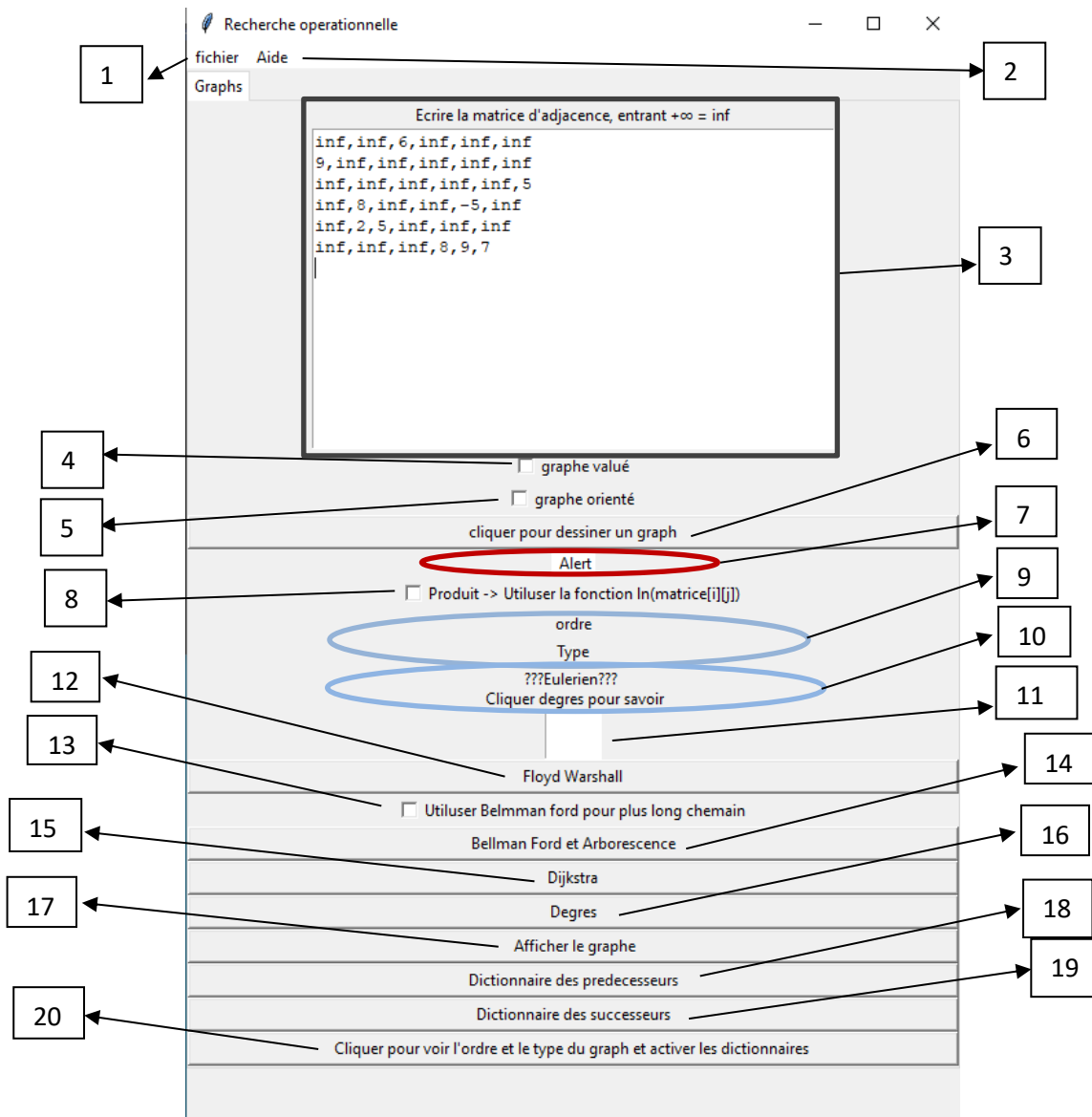


Figure 26 : Vue principale de l'application.

### C. Description de vue principale.

Menu bar	
1	Gestion des fichiers : <ul style="list-style-type: none"> <li>Ouvrir un fichier <b>txt</b> qui contient une matrice d'adjacence.</li> <li>Enregistrer la matrice qu'on a.</li> <li>Reset tous à 0.</li> </ul>
2	Aide : <ul style="list-style-type: none"> <li>Afficher l'aide.</li> <li>A propos de logiciel.</li> </ul>
Entrer du graph	
3	Vous pouvez ici écrire la matrice d'adjacence du graphe entrant ' , ' entres les nombre et 'inf' à la place de $\infty$ .
4	Choisir si le graph à dessiner est value ou non.
5	Choisir si le graph à dessiner est orienté ou non.
6	Dessiner le graphe <u>« plus d'information par la suite ».</u>
Alert	
7	Tous les alerts s'affiches ici <u>« plus d'information par la suite ».</u>
Produit entre les sommets	
8	Choisir s'il y'a une loi de multiplication ou de sommation entre les sommets
Information sur le graphe	
9	Affichage d'ordre et de type de graphe.
10	Afficher si le graph est eulérien, semi eulérien, ou non.
Plus court et plus long chemin et arborescence	
11	Entrer le sommet de début pour l'algorithme de <b>dijkstra</b> et aussi <b>Bellman Ford</b> . L'indexation se fait de 0 jusqu'à (ordre-1).
12	Utiliser l'algorithme de <b>Floyd Warshall</b> pour plus court chemin.
13	Choisir le mode de fonctionnement de l'algorithme de <b>Bellman Ford</b> , plus court ou plus long chemin.
14	Utiliser l'algorithme de <b>Bellman Ford</b> ainsi que l'affichage d' <b>arborescence</b> .
15	Utiliser l'algorithme de <b>Dijkstra</b> pour plus court chemin.
Autres fonctions	
16	Afficher le tableau de degrés des sommets et aussi eulerianité du graphe.
17	afficher le graphe.
18	Afficher le dictionnaire des prédécesseurs.
19	Afficher le dictionnaire des successeurs.
20	Ordre et type du graphe ainsi qu'activation des dictionnaire.

## D. Alerts :

Alert	S'il n'y a aucun Alert.
Fichier enregistré	Si le fichier enregistré sans problèmes.
Ecrire un Sommet de début	Si on n'a pas écrit le sommet de début.
Dijkstra ne marche pas avec les valeurs négatifs	Si on veut dijkstra mais il y'a des poids négatifs.
log ne marche pas avec les 0 ou les nombres négatifs	Si on veut la loi de multiplication mais il y'a des poids négatifs.
erreur de syntax	Si on a fait une erreur dans la syntaxe de la matrice.
la matrice n'est pas carrée	Si la matrice n'est pas carrée.

## E. Dessiner un graphe.

Le dessinateur des graphes est une classe appelée "Draw\_Graph", utilisant la bibliothèque **Tkinter** pour créer une interface utilisateur graphique. Cette classe est utilisée pour créer une visualisation de graphe sur un canevas, où l'utilisateur peut ajouter des sommets et des arêtes en cliquant sur le canevas. Le graphe peut être orienté ou non orienté, et peut également avoir des valeurs associées à ses arêtes. La classe comprend plusieurs méthodes pour ajouter des sommets et des arêtes, afficher la matrice, lier des événements et réinitialiser le graphe. La classe utilise également la bibliothèque **numpy**, ainsi qu'une grande collection de fonctions mathématiques de haut niveau pour opérer sur ces matrices.

Pour ajouter des sommets dans le graphe, vous devez cliquer bouton gauche sur la zone de dessin. Pour ajouter des arêtes, vous devez cliquer d'abord sur un sommet par bouton droite, puis sur un autre sommet par bouton gauche pour relier les deux. Si vous voulez enregistrer la matrice d'adjacence, vous pouvez cliquer sur le bouton 'enregistrer dans matrice.txt'.

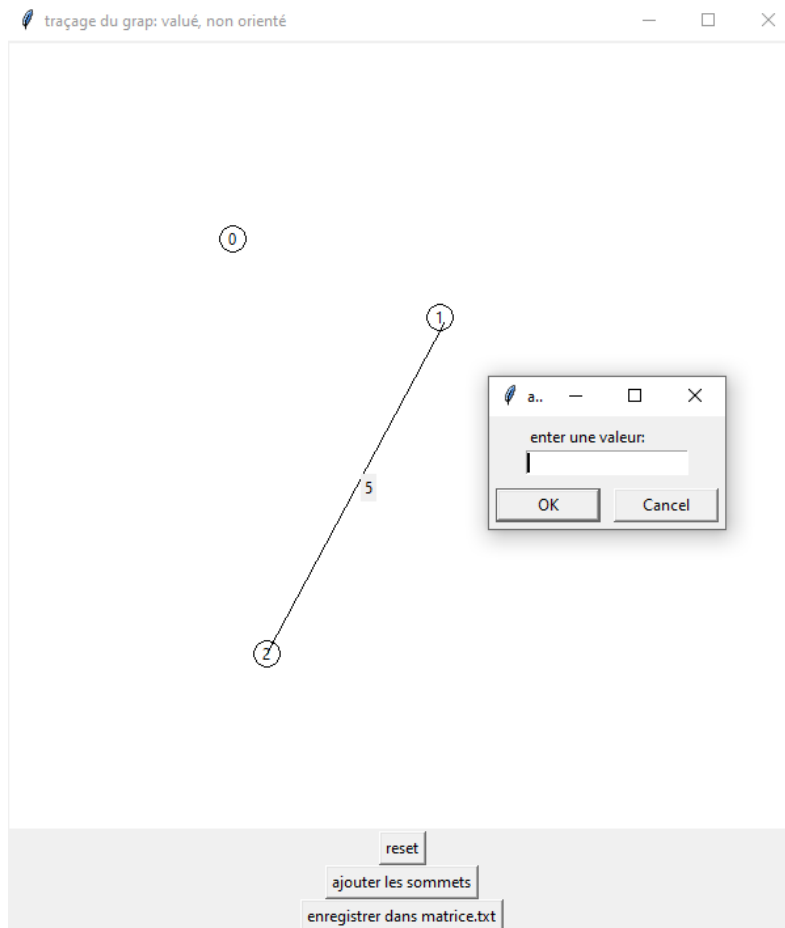


Figure 27: dessineur des graphes

## F. Conclusion :

Ce TP a été très riche en informations soit à propos de théorie des graphes ou pour la programmation en langage python. Par ce que nous sommes maintenant capable de traiter tous les types des graphes, soit orienté ou non, value ou non, trouver le plus court ou le plus long chemin soit par loi de multiplication ou de sommation, aussi des autres acquis.

Concernant le langage python, on a vraiment augmenté nos compétences, nous pouvons maintenant créer des interfaces graphiques à l'aide de **tkinter** et utiliser les diverses classes dans cette bibliothèque, ses méthodes et ses fonctions, ainsi que nous pouvons manipuler les matrices et l'algèbre linéaire en générale utilisant **numpy**, aussi que **matplotlib** pour de tracer des divers types des graphes, et **networkx** qui est une bibliothèque puissante pour la manipulation des graphes.

#### **IV. Conclusion générale :**

En général, la théorie des graphes est un domaine important de la mathématique et de l'informatique qui étudie les relations entre les objets sous forme de nœuds et les liens entre eux sous forme d'arêtes. Les graphes sont utilisés dans de nombreuses applications telles que la recherche d'itinéraire, la modélisation de réseaux sociaux et la résolution de problèmes d'optimisation. Le TP pouvait avoir permis de comprendre les différents types de graphes, les algorithmes de parcours de graphes et les propriétés des graphes.

En utilisant Python, il est possible d'implémenter des algorithmes de théorie des graphes de manière efficace et facile à comprendre. Il existe de nombreux paquets Python tels que Networkx qui offrent des fonctionnalités avancées pour la manipulation de graphes. Il est également possible d'utiliser Python pour visualiser les graphes à l'aide de paquets tels que Matplotlib pour créer des graphiques attrayants et faciles à comprendre, ainsi que nombreuses bibliothèque , numpy, tkinter .....

En résumé, Python est un outil puissant pour la théorie des graphes car il permet de faciliter l'implémentation des algorithmes et la visualisation des résultats.