Devoir n°2

3 heures

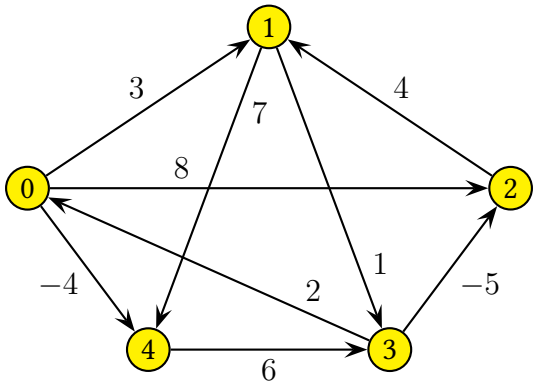
Problème : distance dans un graphe orienté pondéré

L'algorithme de Dijkstra vu peut-être en première année permet de trouver toutes les distances minimales à partir d'un seul noeud (sommet) vers tous les autres. En appliquant cela à tous les noeuds, il permettait de déterminer l'ensemble des distances minimales entre deux noeuds quelconque.

L'algorithme de Floyd-Warshall va permettre de déterminer toutes ces distances, dans le cas de graphe dont les arcs ont des poids quelconques, même négatifs, mais avec la contrainte que le graphe ne présente pas de cycle (un chemin de sommet départ et sommet arrivé identiques) de poids total négatif ou nul. **On se place dans le cas d'un tel graphe**.

Graphe exemple

Nous considèrerons le graphe pondéré orienté exemple suivant pour se familiariser avec l'algorithme :



L'indexation commence à 0. On note $E = \{0, \dots, n-1\}$ l'ensemble des sommets du graphe.
On définit la matrice d'adjacence G comme suit : du sommet i vers le sommet j :
– $G_{i,j} = p_{ij}$ si l'arc orienté de i vers j existe avec un poids p_{ij}
– $G_{i,j} = 0$ si $i = j$
– $G_{i,j} = \infty$ si l'arc orienté de i vers j n'existe pas.
Nous modéliserons une matrice par une liste de listes de la façon suivante : par exemple, la matrice

$$\begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}$$

sera modélisée par la liste python

```
A = [[1,2], [0,-1]]
```

Nous importons le nombre infini, que l'on pourra ensuite utilisé, par la commande :

```
from math import inf
```

On pose $W^{(0)} = G$. Pour k, avec $1 \leqslant k \leqslant n$, on définit la matrice $W^{(k)}$ en définissant ses coefficients $W^{(k)}_{i,j}$ de la façon suivante :
$W^{(k)}_{i,j}$ est égal au poids minimum des chemins du sommet i au sommet j (s'ils existent, sinon le poids est ∞) qui n'empruntent que les sommets intermédiaires dans $\{0, \ldots, k-1\}$.

1. Donner des exemples concrets d'applications de recherche de chemins de poids minimal dans un graphe pondéré orienté.

2. Expliquez pourquoi, lorsqu'il existe des chemins du sommet i vers le sommet j, il existe un tel chemin de poids minimal.

3. Donner la matrice d'adjacence G associée au graphe exemple et écrire le code python permettant de définir G.

4. Vérifier que $W^{(0)}_{i,j}$ représente le poids minimal des chemins (s'ils existent) du sommet i vers le sommet j empruntant aucun sommet intermédiaire, ∞ sinon.

5. Quelle matrice doit-on déterminer pour répondre à l'objectif ?

6. À chaque itération de l'algorithme, $k \in \{1, \ldots, n\}$, une mise à jour de la matrice $W^{(k-1)}$ est effectuée vers la matrice $W^{(k)}$. Pour cela, nous devons établir les formules liant les coefficients $W^{(k-1)}_{i,j}$ et $W^{(k)}_{i,j}$.

   On considère pour cela, s'il existe, un chemin c de poids minimal de i vers j empruntant les sommets intermédiaires dans $\{0, \ldots k-1\}$.

   (a) Montrer que si c passe de façon intermédiaire par le sommet $k-1$, alors il ne peut y passer qu'une seule fois.
   En déduire que le chemin c se décompose en un chemin c_1 de i vers $k-1$ et un chemin c_2 de $k-1$ vers j, avec c_1 et c_2 des chemins dont les sommets intermédiaires sont dans $\{0, \ldots, k-2\}$ (éventuellement vide si $k = 1$).

   (b) Si c ne passe pas de façon intermédiaire par le sommet $k-1$, que dire de ses sommets intermédiaires ?

   (c) En déduire précisément que l'on a alors la relation de récurrence

   $$W^{(k)}_{i,j} = \min(W^{(k-1)}_{i,k-1} + W^{(k-1)}_{k-1,j}, W^{(k-1)}_{i,j})$$

   (d) Vérifier que la relation précédente est encore valide lorsqu'il n'existe pas de chemin de i vers j empruntant les sommets intermédiaires dans $\{0, \ldots k-1\}$.

   (e) Expliquez en quoi cette formule permet de répondre à l'objectif.

7. Appliquez l'algorithme de Floyd-Warshall sur le graphe exemple. On trouvera

$$W^{(\boxed{?})} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Quel est le chemin de poids minimal et son poids du sommet 0 vers le sommet 1 ?

8. Toto, un peu naïf, voyant ces belles formules de récurrence, se lance dans une programmation récursive. Comme lui, donnez le programme python récursif permettant de déterminer $W^{(n)}$. Toto procède comme souvent en récursivité, en deux fonctions, la seconde étant la vraie procédure récursive : on donne le squelette :

```
def FWR(G):
    n = ?
    W = ?              # initialisation
    for i in range(n):
        for j in range(n):
            W[i][j] = FW_r(G, i, j, n)
    return W

def FW_r(G, i,j,k):
    if k == 0:
        return ?
    return min(FW_r(G, ? , ?, ?), FW_r(G, ? , ? , ? ) + FW_r(G, ?, ?, ?))
```

Quelle est la complexité d'une telle programmation ? Commentez.
Comment pourrait-on simplifier simplement la complexité en mémoire, sans changer le principe de récursivité ?

9. Toto pense finalement à la méthode dynamique. Il écrit alors

```
def FW(G):
    n = len(G)
    W = [G]

    for k in range(0,n):
        Wk = [[0]*n for _ in range(n)]
        for i in range(n):
            for j in range(n):
                Wk[i][j] = min(W[-1][i][j], W[-1][i][k] + W[-1][k][j])
        W.append(Wk)
    return W[-1]
```

Expliquez précisément chaque ligne de ce programme (avec le choix des indices, des bornes).
Quelle est la complexité en opérations ? en mémoire ? (en fonction de n) On pourra ce référer aux calculs effectués pour le graphe exemple.

10. En schématisant sur des tableaux matriciels la formule de récurrence permettant les calculs de $W_{i,j}^{(k)}$, montrer que si l'on souhaite uniquement obtenir $W^{(n)}$, on peut en fait effectuer les calculs en utilisant un unique tableau et donner le code python associé (*).

11. On souhaite aussi déterminer un chemin optimal de i vers j.

(a) Quelle est la problématique en cas d'utilisation comme dans la question précédente d'un unique tableau ?

(b) Dans le graphe exemple, comment obtient-on à l'aide des matrices $W^{(k)}$ un chemin optimal par exemple du sommet 0 vers le sommet 2 ?

(c) Écrire une fonction python qui détermine un chemin optimal du sommet i au sommet j ainsi que son poids.

1. Donner des exemples concrets d'applications de recherche de chemins de poids minimal dans un graphe pondéré orienté.

   Système de navigation routier (les poids sont positifs à priori ; poids négatif pour pénaliser éventuellement).
   Système de routage du traffic internet.
   Déplacement d'un robot en robotique.

2. Expliquez pourquoi, lorsqu'il existe des chemins du sommet i vers le sommet j, il existe un tel chemin de poids minimal.

   Si on considère l'ensemble des chemins de i vers j, cet ensemble peut être infini en effectuant des boucles et du coup l'ensemble des longueurs de ces chemins va être une partie quelconque à priori de $\mathbb{Z}$ qui ne peut donc pas forcément être minimisé.

   Par contre, on remarque que si un tel chemin admet des boucles (si on passe plusieurs fois par un même noeuds), comme il n'y a pas de boucle de poids négatif, en supprimant toutes les boucles de ce chemin, on obtient un chemin de poids inférieur, et allant de i vers j.

   Or il y a un nombre **fini** de chemin allant de i vers j qui ne passe qu'une seul fois par un noeud (grossièrement, au plus $(n-2)!$: le chemin est de taille intermédiaire p, $p \leqslant n-2$, et on choisit le premier noeud intermédiaire parmi $n-2$, le second parmi $n-3$, etc). On peut donc choisir un tel chemin de poids minimal.

3. Écrire le code permettant de définir la matrice d'adjacence W associée au graphe exemple. Par exemple,

```
G = [[0,3,8,inf,-4]]
G.append([inf,0,inf,1,7])
G.append([inf,4,0,inf,inf])
G.append([2,inf,-5,0,inf])
G.append([inf,inf,inf,6,0])
```

4. Vérifier que $W_{i,j}^{(0)}$ représente le poids minimal des chemins (s'ils existent) du sommet i vers le sommet j empruntant aucun sommet intermédiaire, ∞ sinon.

   Il s'agit donc des chemins directs du sommet i vers le sommet j, et donc soit il y a un chemin, il y en a un seul, de poids $G_{i,j}$, soit il n'y a pas de chemin et alors le poids est ∞. Ainsi $W_{i,j}^{(0)}$ représente bien le poids minimal des chemins (s'ils existent) du sommet i vers le sommet j empruntant aucun sommet intermédiaire, ∞ sinon.

5. On cherche donc à déterminer la matrice $W^{(n)}$, dont le coefficient $W_{i,j}^{(n)}$ est égal au poids minimal des chemins de i vers j empruntant des sommets intermédiaires dans $\{0,\ldots,n-1\}$, donc tous, s'il existent, ou ∞ sinon.

6. À chaque itération de l'algorithme, $k \in \{1,\ldots,n\}$, une mise à jour de la matrice W^{k-1} est effectuée vers la matrice $W^{(k)}$. Pour cela, nous devons établir les formules liant les coefficients $W_{i,j}^{(k-1)}$ et $W_{i,j}^{(k)}$.

   On considère pour cela, s'il existe, un chemin c de poids minimal de i vers j empruntant les sommets intermédiaires dans $\{0,\ldots k\}$.

   (a) Si c passe de façon intermédiaire par le sommet k, alors dans le cas où il passerait deux fois, on serait un présence d'un cycle, de poids donc positif, et ainsi on pourrait supprimer ce cycle (neutre si le poids est nul), et se ramener à un chemin de poids strictement inférieur si le poids est strictement positif, ce qui est absurde dans ce cas.
   Ainsi le chemin passe une unique fois de façon intermédiaire par le sommet k (et dans le cas de cycle de poids nul, le chemin c peut se réduire à un chemin ne passant qu'une seule fois de façon intermédiaire par k, en gardant le même poids minimal).

   Ainsi le chemin c se décompose en un chemin c_1 de i vers k et un chemin c_2 de k vers j, avec c_1 et c_2 des chemins dont les sommets intermédiaires sont dans $\{0,\ldots,k-1\}$ puisque le chemin initial ne passe pas de nouveau par le sommet k, ces deux chemins étant de poids minimal, par sous-structure optimale.

   (b) En déduire précisément que l'on a alors la relation de récurrence

   $$W_{i,j}^{(k)} = \min(W_{i,k}^{(k-1} + W_{k,j}^{(k-1)}, W_{i,j}^{(k-1)})$$

   – ou bien le chemin c se décompose en un chemin c_1 de i vers k et un chemin c_2 de k vers j, avec c_1 et c_2 des chemins de poids minimal dont les sommets intermédiaires sont dans $\{0,\ldots,k-1\}$
   – ou bien le chemin c ne passe pas de façon intermédiaire par le sommet k, et donc c'est un chemin qui passe de façon intermédiaire par les sommets $,\ldots,k-1$, de poids minimal.
   On en déduit la formule par minimalisation.

   (c) S'il n'existe pas de chemin de i vers j empruntant les sommets intermédiaires dans $\{0,\ldots k\}$, nous avons $W_{i,j}^{(k)} = \infty$. Mais dans ce cas, il n'existe pas non plus de chemins de i vers j empruntant les sommets intermédiaires dans $\{0,\ldots k-1\}$ (qui serait un chemin de i vers j empruntant les sommets intermédiaires dans $\{0,\ldots k\}$), et donc $W_{i,j}^{(k-1)} = \infty$. De plus, si on avait $W_{i,k}^{(k-1}$ et $W_{k,j}^{(k-1)}$ non infini, on aurait une chemin de i vers k et un chemin de k vers j empruntant les sommets intermédiaires dans $\{0,\ldots k-1\}$, ce qui permettrait par concaténation de créer u chemin de i vers j empruntant les sommets intermédiaires dans $\{0,\ldots k-1\}$, ce qui est contradictoire. Ainsi, on a $W_{i,k}^{(k-1}$ ou $W_{k,j}^{(k-1)}$ infini et donc $W_{i,k}^{(k-1} + W_{k,j}^{(k-1)} = \infty$, et la formule est encore bien valide.

   (d) Elle permet de construire successivement les matrices $W^{(1)}$, $W^{(2)}$,...,$W^{(n)}$ et donc d'atteindre l'objectif.

7. Résoudre le problème des chemins de poids minimum pour le graphe exemple. On obtient successivement, avec un certain courage et de la ténacité à toute épreuve,

$$W^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad W^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$W^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad W^{(5)} = \begin{pmatrix} 0 & 1 & \boxed{-3} & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Quel est le chemin de poids minimal et son poids du sommet 0 vers le sommet 1 ? Le poids est 1, et en reprenant le graphe exemple, nous voyons **un** chemin de poids 1 : 0 > 4 > 3 > 2 > 1. C'est le seul.

8. Toto, un peu naïf, voyant ces belles formules de récurrence, se lance dans une programmation récursive. Comme lui, donnez le programme python récursif permettant de déterminer $W^{(n)}$. Toto procède comme souvent en récursivité, en deux fonctions, la seconde étant la vraie procédure récursive :

```
def FWR(G):
    n = len(G)
    W = [[0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            W[i][j] = FW_r(G, i, j, n)
    return W

def FW_r(G, i,j,k):
    if k == 0:
        return G[i][j]
    return min(FW_r(G, i, j, k-1), FW_r(G, i, k-1,k-1) + FW_r(G, k-1,j,k-1))
```

Quelle est la complexité d'une telle programmation ? Commentez.
Comment pourrait-on simplifier simplement la complexité en mémoire, sans changer le principe de récursivité ?

La récursivité a une profondeur de n, avec 3 sous-appels pour chaque appel, et n^2 coefficients à déterminer. Nous obtenons une complexité de $3^n n^2$, soit plus qu'exponentielle. La méthode n'est pas utilisable si n grand.
Les appels se croisent. Par mémoïsation, on pourrait éviter les «doublons» de calculs et soulager l'occupation mémoire.

9. Toto pense finalement à la méthode dynamique. Il écrit alors

```
def FW(G):
    n = len(G)  # taille du graphe
    W = [G]     # G=W^(0) initialisation de la liste des matrices W^(k)
```

```
    for k in range(0,n):                  # de 0 à n-1 : W^(1) à W^(n)
        Wk = [[0]*n for _ in range(n)]    # tableau de 0 de taille n
        for i in range(n):                # i indice ligne de 0 à n-1
            for j in range(n):            # j indice colonne de 0 à n-1
                # calcul de W^(k+1)_(i,j) par la formule à l'aide de W[-1] qui est W^(k)
                # attention au décalage de k
                Wk[i][j] = min(W[-1][i][j], W[-1][i][k] + W[-1][k][j])
        W.append(Wk)                      # ajout de W^(k) à la liste
    return W[-1]                          # c'est W^(n)
```

Expliquez précisément chaque ligne de ce programme (avec le choix des indices, des bornes). Quelle est la complexité en opérations ? en mémoire ? (en fonction de n) On pourra ce référer aux calculs effectués pour le graphe exemple.

Nous avons trois boucles imbriquées indépendantes de taille n, avec une somme et un calcul de minimum, soit une complexité de l'ordre de n^3.
La taille mémoire utilisée est de n tableau de taille n, soit n^3.

10. En schématisant sur des tableaux matriciels la formule de récurrence permettant les calculs de $W_{i,j}^{(k)}$, montrer que si l'on souhaite uniquement obtenir $W^{(n)}$, on peut en fait effectuer les calculs en utilisant un unique tableau et donner le code python associé (*).

Pas facile à comprendre, et c'est le prélude à une optimisation. On remarque que (sur l'exemple) lors du passage de $W^{(k-1)}$ à $W^{(k)}$ qui utilise les coefficients de $W^{(k-1)}$ de la ligne $k-1$ et colonne $k-1$, ceux-ci restent inchangés et effectivment, c'est bien ce que l'on constate lorsque $i = k-1$ ou $j = k-1$ sur la formule (à écrire). Il n'y aura pas de conflit si on les «écrase» puisque en fait inchangés. On peut alors appliquer la formule **en place**, en utilisant une seule matrice, que l'on modifie toujours à l'aide de la formule :

```
def FW(G):
    n = len(G)
    W = G[:]

    for k in range(0,n):

        for i in range(n):
            for j in range(n):
                W[i][j] = min(W[i][j], W[i][k] + W[k][j])

    return W
```

11. On souhaite aussi déterminer un chemin optimal de i vers j.

    (a) Quelle est la problématique en cas d'utilisation comme dans la question précédente d'un unique tableau ?

    On ne pourra pas faire du backtracing pour retrouver un chemin de poids minimal.

    (b) Dans le graphe exemple, comment obtient-on à l'aide des matrices $W^{(k)}$ un chemin optimal par exemple du sommet 0 vers le sommet 2 ?

$$W^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad W^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad W^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Dans la dernière matrice, nous lisons le poids minimal -3 ; on voit alors directement sur le graphe un tel chemin : $0 \to 4 \to 3 \to 2$.
Pour le retrouver en général par l'intermédiaire des matrices, c'est plus délicat, puisqu'il faut procéder par backtracing. Au départ, le chemin passe par 0 et 2.

$$W^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & \boxed{-4} \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & \boxed{1} & 6 & 0 \end{pmatrix} \qquad W^{(5)} = \begin{pmatrix} 0 & 1 & \boxed{-3} & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Ce dernier -3 a été obtenu à partir de la matrice $W^{(4)}$, comme minimum de $W^{(4)}_{0,2} = -1$ et de $W^{(4)}_{0,4} + W^{(4)}_{4,2} = -4 + 1 = -3$. Il y a ajout du noeud 4 : $0 \to 4 \to 2$.

$$W^{(3)} = \begin{pmatrix} 0 & 3 & 8 & \boxed{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & \boxed{-2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad W^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & \boxed{-4} \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Ce dernier $W^{(4)}_{0,4} = -4$ a été obtenu à partir de la matrice $W^{(3)}$, comme minimum de $W^{(3)}_{0,4} = -4$ et de $W^{(3)}_{0,3} + W^{(3)}_{3,4} = 4 - 2 = 2$. Pas d'ajout de noeud.

Ce dernier $W^{(4)}_{4,2} = 1$ a été obtenu à partir de la matrice $W^{(3)}$, comme minimum de $W^{(3)}_{4,2} = \infty$ et de $W^{(3)}_{4,3} + W^{(3)}_{3,2} = 6 - 5 = 1$. Ajout du noeud 3, chemin $0 \to 4 \to 3 \to 2$.

Pas d'ajout par la suite, les valeurs 6 et -5 étant inchangées jusqu'à $W^{(3)}$.
Ceci constitue l'algorithme de la question suivante, la gestion de la construction de la liste des noeuds par backtracing étant la difficulté.

(c) Écrire une fonction python qui détermine un chemin optimal du sommet i au sommet j ainsi que son poids.

```
def FW_b(G,a,b):
    n = len(G)
    W = [G]
```

```
    # construction dynamique des matrices

    for k in range(0,n):
        Wk = [[0]*n for _ in range(n)]
        for i in range(n):
            for j in range(n):
                Wk[i][j] = min(W[-1][i][j], W[-1][i][k] + W[-1][k][j])
        W.append(Wk)

    # chemin optimal de a vers b : backtracing

    L = [a,b]            # chemin temporaire
    for k in range(n):
        p = len(L)
        L_bis = []
        for i in range(p-1):
            L_bis.append(L[i])
            u = L[i]
            v = L[i+1]
            kk = n-1-k
            if W[kk][u][v] > W[kk][u][kk]+W[kk][kk][v]: # on passe par kk
                L_bis.append(kk)                        # ajout du noeud intermédiaire
        L_bis.append(L[-1])                             # ajout du dernier noeud b
        L = L_bis[:]
    return L, W[n][a][b]
```