

Exercice 1

```
## Exercice 1

def chaine(a, t, e, x):
    """ entrées : a liste de deux listes d'entiers
        t liste de deux listes d'entiers (un de moins que pour a)
        e, x listes de deux entiers
        sortie : délai optimal d'après le protocole de la chaîne de montage
    """
    n = len(a[0])
    # 1) on définit le tableau "vide" des f_i(j)
    f = [[0 for j in range(n)] for i in range(2)]

    # 2) on initialise avec les cases que l'on connaît
    f[0][0] = e[0] + a[0][0]
    f[1][0] = e[1] + a[1][0]

    # 3) on remplit le tableau avec les formules de récurrence
    for j in range(1, n):
        f[0][j] = min(f[0][j-1] + a[0][j], f[1][j-1] + t[1][j-1] + a[0][j])
        f[1][j] = min(f[1][j-1] + a[1][j], f[0][j-1] + t[0][j-1] + a[1][j])

    # 4) on exploite les cases f_0(n-1) et f_1(n-1)
    fe = min(f[0][n-1] + x[0], f[1][n-1] + x[1])

    return fe

# exemple
ex_e = [3, 5]
ex_a = [[6, 8, 2, 4, 9], [8, 4, 7, 7, 3]]
ex_t = [[2, 1, 5, 3], [3, 2, 1, 2]]
ex_x = [2, 3]

assert chaine(ex_a, ex_t, ex_e, ex_x) == 32
```

```
# version avec un chemin optimal

def chaine_bis(a, t, e, x):
    n = len(a[0])
    # 1) on définit les tableaux "vide"
    f = [[0 for j in range(n)] for i in range(2)]
    l = [[-1 for j in range(n)] for i in range(2)]

    # 2) on initialise avec les cases que l'on connaît
    f[0][0] = e[0] + a[0][0]
    f[1][0] = e[1] + a[1][0]

    # 3) on remplit les tableaux avec les formules de récurrence
    for j in range(1, n):
        u, v = f[0][j-1] + a[0][j], f[1][j-1] + t[1][j-1] + a[0][j]
        if u < v:
            f[0][j] = u
            l[0][j] = 0
        else:
            f[0][j] = v
            l[0][j] = 1

        w, y = f[1][j-1] + a[1][j], f[0][j-1] + t[0][j-1] + a[1][j]
        if w < y:
            f[1][j] = w
            l[1][j] = 1
        else:
            f[1][j] = y
            l[1][j] = 0

    # 4) on exploite les cases f_0(n-1) et f_1(n-1)
    d0, d1 = f[0][n-1] + x[0], f[1][n-1] + x[1]
    if d0 < d1:
        fe = d0
        last = 0
    else:
        fe = d1
        last = 1

    # 5) on exploite le tableau l
    ch = [last]
    j = n-1
    while j > 0:
        v = l[last][j]
        ch.append(v)
        last = v
        j = j - 1

    return fe, ch[::-1]
```

Exercice 2

1. Si l'on part de la ligne $n-1$ alors on se contente de lire l'entier dans t :

$$\forall j \in [0, n-1], S_{n-1,j} = t[n-1][j].$$

Considérons deux entiers i et j avec $0 \leq j \leq i \leq n-2$.

Si l'on part de la case à la i -ème ligne et j -ème colonne alors il y a déjà la valeur de $t[i][j]$ plus la plus grande des deux valeurs correspondant à un départ sur la case en dessous à gauche (ligne $i+1$, colonne j) ou à un départ sur la case en dessous à droite (ligne $i+1$, colonne $j+1$) donc :

$$S_{i,j} = t[i][j] + \max(S_{i+1,j}, S_{i+1,j+1}).$$

2.

```
def triangle(t):
    n = len(t)
    # 1) création du tableau
    S = [[0 for j in range(n)] for i in range(n)]

    # 2) initialisation / cases "faciles"
    for j in range(n):
        S[n-1][j] = t[n-1][j]

    # 3) boucle pour remplir le tableau
    i = n-2
    while i >= 0:
        for j in range(i+1):
            S[i][j] = t[i][j] + max(S[i+1][j], S[i+1][j+1])
        i = i - 1

    # 4) on exploite la ou les case(s) utile(s)
    return S[0][0]
```

3. Une première version en déterminant d'abord le tableau S puis le chemin optimal :

```
def chemin(t):
    n = len(t)
    S = [[0 for j in range(n)] for i in range(n)]
    for j in range(n):
        S[n-1][j] = t[n-1][j]
    i = n-2
    while i >= 0:
        for j in range(i+1):
            S[i][j] = t[i][j] + max(S[i+1][j], S[i+1][j+1])
        i -= 1
    C = [0]
    j = 0
    for i in range(n-1):
        if S[i+1][j] < S[i+1][j+1]:
            j += 1
        C.append(j)
    return S[0][0], C
```

Une autre version en calculant, en même temps que S, un tableau des chemins optimaux en partant de chaque emplacement.

```
def chemin(t):
    n = len(t)
    S = [[0 for j in range(n)] for i in range(n)] # tableau de valeurs de S
    C = [[[[] for j in range(n)] for i in range(n)] # tableau des chemins correspondants
    for j in range(n): # initialisation de S et C
        S[n-1][j] = t[n-1][j]
        C[n-1][j] = [j]
    i = n-2
    while i >= 0: # on "remonte" les lignes
        for j in range(i+1):
            a = S[i+1][j] + t[i][j]
            b = S[i+1][j+1] + t[i][j]
            if a > b:
                S[i][j] = a # si S[i+1][j] > S[i+1][j+1] alors...
                ch = [c for c in C[i+1][j]] # on "récupère" le chemin partant de (i+1,j)
            else:
                S[i][j] = b # sinon...
                ch = [c for c in C[i+1][j+1]] # on "récupère" le chemin partant de (i+1,j+1)
            ch.append(j) # et on ajoute j à la fin
            C[i][j] = ch # puis on met à jour le tableau C
        i -= 1 # on remonte d'une ligne
    return S[0][0], list(reversed(C[0][0])) # on "retourne" le chemin pour le donner de haut en bas
```