TSI2. Devoir libre 02

CORRECTION Informatique

Problème 01 : tranches maximales

Les tableaux qui peuventêtre utilisés dans ce problème sont fournis sous forme de liste Python (donc indéxés à partir de 0). Le but du problème est d'étudier diverses méthodes de calcul du maximum des sommes d'éléments consécutifs d'un tableau donné (non trié) de nombres réels. Plus précisément, t étant donné, contenant n valeurs indexées de 0 à $n-1$, on cherche la plus grande des sommes des valeurs des « tranches » non vides $t[g:d]$, c'est- à -dire des sommes de la forme $\sum_{k=g}^{d-1} t[k]$, où les entiers g et d vérifient $0 \leqslant g < d \leqslant n$.
Cette valeur sera notée $S(t)$; elle est bien définie en tant que plus grand élément d'un ensemble ordonné fini.

Partie A. Les préliminaires c'est important !

1. Calculons $S(t)$ pour : a) $t = [-7, -8]$.
   On a $t[0] = -7$, $t[1] = -8$, on peut faire trois sommes -7, -8 et -15 dont la plus grande est $S(t) = -7$ pour $g = 0$ et $d = 1$. b) $t = [-3, 5, -2]$.
   On a : $t[0] = -3$, $t[1] = 5$, $t[2] = -2$. On peut faire six sommes.
   $$-3, 5, -2, -3+5, -3+5-2, 5-2 = -3, 5, -2, 2, 0, 3.$$
   La plus grande est $S(t) = 5$ pour $g = 1$ et $d = 2$.
2. Écrivons une fonction $max(a, b)$ renvoyant le plus grand des deux réels a et b. Cette fonction pourra être utilisée dans la suite. (Attention, on sait que Python a une fonction prédéfinie qui calcule le maximum mais il n'est pas question de l'utiliser ici.)

```
>>> def max(a,b) :
if a > b :

    return a

else :

    return b
```

Partie B. Un algorithme naif et première amélioration

La fonction ci-dessous détermine $S(t)$ en calculant successivement toutes les sommes décrites ci-dessus :

```
>>> def som_max1(t) :
        n = len(t) ; s_max = t[0]
        for g in range(n) :
            for d in range(g+1,n+1) :
                s = t[g]
                for k in range(g+1,d) :
                    s+ = t[k]
                s_max = max(s_max,s)
        return(s_max)
```

1. Appliquons som_max1 à la liste $t = [-4, 0, -1, 2]$, en écrivant à la main toutes les étapes décrites par l'algorithme.

   On a : $t[0] = -4$, $t[1] = 0$, $t[2] = -1$, $t[3] = 2, n = 4$. Puis $n = len(t) = 4$, $s\_max = -4$.
   • $g = 0$. Then d is in range$(1, 5)$.
   $d = 1$, $s = -4 : k$ is in range$(1, 1)$. $NULL$ $s\_max = max(-4, -4) = -4$.
   $d = 2$, $s = -4 : k$ is in range$(1, 2)$. $k = 1$ $s\_max = -4 + t[1] = -4$.
   $d = 3$, $s = -4 : k$ is in range$(1, 3)$. $s = -4 + 0 - 1 = -5$ $s\_max = max(-4, -5) = -4$.
   $d = 4$, $s = -4 : k$ is in range$(1, 4)$. $s = -4 - 1 + 2 = -3$ $s\_max = max(-4, -3) = -3$.

   • $g = 1$. Then d is in range$(2, 5)$.
   $d = 2$, $s = 0 : k$ is in range$(2, 2)$. $NULL$ $s\_max = 0$.
   $d = 3$, $s = 0 : k$ is in range$(2, 3)$. $s = 0 - 1 = -1$ $s\_max = max(-3, -1) = -1$.
   $d = 4$, $s = t[1] = 0 : k$ is in range$(2, 4)$. $s = 0 - 1 + 2 = 1$. $s\_max = max(-1, 1) = 1$.

   • $g = 2$. Then d is in range$(3, 5)$.
   $d = 3$, $s = t[2] = -1 : k$ is in range$(3, 3)$. $NULL$ $s\_max = -1$.
   $d = 4$, $s = -1 : k$ is in range$(3, 4)$. $s = -1 + t[3] = 1$. $s\_max = max(-1, 1) = 1$.

   • $g = 3$. Then d is in range$(4, 5)$.
   $d = 3$, $s = t[3] = 2 : k$ is in range$(4, 4)$. $NULL$ $s\_max = max(1, 2) = 2$.

   We return $s\_max = 2$. Il y a 10 étapes.

2. On peut montrer (et **on l'admettra**) que le nombre $T_1(n)$ d'additions de réels effectuées au cours de l'appel de $som\_max1(t)$ est de l'ordre de n^3 lorsque t contient n valeurs.
   On peut supprimer l'une des boucles for imbriquées de la fonction som_max1, en mettant à jour la variable s_max au fur et à mesure du calcul des $\sum_{k=g}^{d-1} t[k]$, pour g fixé et d variant de $g + 1$ à n.
   Justifions cela en écrivant une fonction $som\_max2(t)$ renvoyant $S(t)$, au prix d'un nombre $T_2(n)$ d'additions de réels qui devra être de l'ordre de n^2 lorsque t contient n valeurs.
   (On supprimera la boucle indexée par d.)

```
>>> def som_max2(t) :
        n = len(t); s_max = t[0]
        for g in range(n) :
            s = t[g]
            s_max = max(s_max, s)
            for k in range(g + 1, n) :
                s+ = t[k]
                s_max = max(s_max, s)
        return(s_max)
```

3. Appliquons som_max2 à la liste $t = [-4, 0, -1, 2]$, en écrivant à la main toutes les étapes décrites par l'algorithme et comparons avec som_max1.

   • $g = 0$. $s = t[0] = -4$, $s\_max = max(-4, -4) = -4$. Then k is in range$(1, 4)$.
   $k = 1 : s = -4 + t[1] = -4$, $s\_max = max(-4, -4) = -4$.
   $k = 2 : s = -4 - 1 = -5$, $s\_max = max(-4, -5) = -4$.
   $k = 3 : s = -5 + 2 = -3$, $s\_max = max(-4, -3) = -3$.

   • $g = 1$. $s = t[1] = 0$, $s\_max = max(-3, 0) = 0$. Then k is in range$(2, 4)$.
   $k = 2 : s = 0 + t[2] = -1$, $s\_max = max(0, -1) = 0$.
   $k = 3 : s = -1 + t[3] = 1$, $s\_max = max(0, 1) = 1$.

   • $g = 2$. $s = t[2] = -1$, $s\_max = max(1, -1) = 1$. Then k is in range$(3, 4)$.
   $k = 3 : s = -1 + t[3] = -1 + 2 = 1$, $s\_max = max(1, 1) = 1$.

   • $g = 3$. $s = t[3] = 2$, $s\_max = max(1, 2) = 2$. Then k is in range$(4, 4)$. NULL.

   We return 2.

   Il y a eu 7 étapes.

4. Justifions que $T_2(n) = \sum_{g=0}^{n-1}(n-g-1)$.

   Pour g fixé dans $[\![0, n-1]\!]$, la boucle interne du programme *som_max*2 effectue $n-g-1$ additions.
   Donc : $T_2(n) = \sum_{g=0}^{n-1}(n-g-1)$. On a alors :

$$T_2(n) = n\sum_{g=0}^{n-1} 1 - \sum_{g=0}^{n-1} g - \sum_{g=0}^{n-1} 1 = n^2 - \frac{n(n-1)}{2} - n.$$

   Finalement, $T_2(n) = \dfrac{n(n-1)}{2}$.

Partie C. Une version récursive

On remarque (et **l'on admet**) que, pour $n \geqslant 2$ et pour t contenant n valeurs, $S(t)$ est le plus grand des $n+1$ nombres suivants : les $\sum_{k=0}^{d-1} t[k]$ avec $d \in [\![1, n]\!]$ et la valeur de $S(t')$, où t' désigne t privé de sa première valeur.

1. Calculons $S(t')$ pour $t = [-4, 0, -1, 2]$ et calculons les quatre valeurs $\sum_{k=0}^{d-1} t[k]$ avec $d \in [\![1, 4]\!]$.

   On a $S(t') = S([0, -1, 2]) = 2$. Puis :

$$d = 1 \Rightarrow \sum_{k=0}^{0} t[k] = t[0] = -4,\ d = 2 \Rightarrow \sum_{k=0}^{1} t[k] = -4 + 0 = -4,$$

$$d = 3 \Rightarrow \sum_{k=0}^{2} t[k] = -4 + 0 - 1 = -5,\ d = 4 \Rightarrow \sum_{k=0}^{3} t[k] = -3.$$

   Alors le maximum de $2, -4, -4, -5$ et -3 est 2. On retrouve le résultat de la partie B.

2. Écrivons une fonction Python **récursive** nommée *som_max*3(t) renvoyant $S(t)$.
   (On rappelle que $t[1:]$ est l'écriture Python de t'.)

   Si le tableau contient une seule valeur, il y a une seule somme que l'on renvoie qui est $t[0]$. Sinon, on applique le principe de l'énoncé, en déterminant la plus grande des sommes des tranches contenant $t[0]$ puis en la comparant à $S(t')$.

```
>>> def som_max3(t) :
        if len(t) == 1 :
            return t[0]
        else :
            s_max = t[0]
            for k in range(1,len(t)) :
                s+ = t[k]
                s_max = max(s_max, s)
            return max(s_max, som_max3(t[1 :]))
```

3. On note $T_3(n)$ d'additions de réels effectuées au cours de l'appel de *som_max*3(t) lorsque t contient n valeurs.

   (a) Justifieons que $T_3(1) = 0$ et que pour tout $n \geqslant 2$, $T_3(n) = n - 1 + T_3(n-1)$.

   Comme dans le cas $len(t) = 1$, on ne procède à aucune addition, $T_3(1) = 0$. Puis on ajoute dans la boucle *for*, $len(t) - 1 = n - 1$ additions, comme on fait appel à *som_max*3$(t[1:])$, cela fait bien au total :

$$T_3(n) = n - 1 + T_3(n-1).$$

(b) On va en déduire $T_3(n)$ en fonction de n :

On a $T_3(2) = 1 + T_3(1) = 1$, $T_3(3) = 2 + T_3(2) = 3 = 2 + 1$, $T_3(4) = 3 + T_3(3) = 3 + 2 + 1$.

De façon générale, $T_3(n) = \sum_{k=1}^{n}(k-1) = \dfrac{n(n-1)}{2}$.

Partie D. Un algorithme linéaire

On note dans cette partie, pour tout $p \in [\![1, n]\!]$:

$$\varepsilon_p = \left\{ \sum_{k=g}^{p-1} t[k],\ 0 \leqslant g < p \right\} \text{ et } \mathcal{F}_p = \left\{ \sum_{k=g}^{d-1} t[k],\ 0 \leqslant g < d \leqslant p \right\}$$

et l'on pose $\delta_p = \max\limits_{g \in [\![1,p]\!]} \sum_{k=g}^{p-1} t[k]$ et $\sigma_p = \max\limits_{(g,d) \in [\![1,p]\!]^2,\, g<d} \sum_{k=g}^{d-1} t[k]$.

1. Déterminons ε_i, $\mathcal{F}_i$, δ_i et σ_i pour tout $i \in [\![1, 4]\!]$ dans le cas de $t = [-4, 0, -1, 2]$.

On a : $\varepsilon_1 = \left\{ \sum_{k=g}^{0} t[k],\ 0 \leqslant g < 1 \right\} = \{t[0]\} = \{-4\}$.

Puis : $\varepsilon_2 = \left\{ \sum_{k=g}^{1} t[k],\ 0 \leqslant g < 2 \right\} = \{t[0] + t[1], t[1]\} = \{-4, 0\}$.

Puis : $\varepsilon_3 = \left\{ \sum_{k=g}^{2} t[k],\ 0 \leqslant g < 3 \right\} = \{t[0] + t[1] + t[2], t[1] + t[2], t[2]\} = \{-5, -1, -1\}$.

Puis : $\varepsilon_4 = \left\{ \sum_{k=g}^{3} t[k],\ 0 \leqslant g < 4 \right\} = \{-3, 1, 1, 2\}$.

Ensuite : $\mathcal{F}_1 = \left\{ \sum_{k=g}^{d-1} t[k],\ 0 \leqslant g < d \leqslant 1 \right\} = \{t[0]\} = \{-4\}$ car $(g, d) = (0, 1)$ obligatoirement.

Ensuite : $\mathcal{F}_2 = \left\{ \sum_{k=g}^{d-1} t[k],\ 0 \leqslant g < d \leqslant 2 \right\} = \{t[0], t[0] + t[1], t[1]\}$ car (g, d) prend alors les valeurs $(0, 1)$, $(0, 2)$ et $(1, 2)$. Il reste :

$$\mathcal{F}_2 = \{-4, -4, 0\} = \{-4, 0\}.$$

De même, $\mathcal{F}_3$ est : $\left\{ \sum_{k=g}^{d-1} t[k],\ 0 \leqslant g < d \leqslant 3 \right\}$ et (g, d) prend les valeurs $(0, 1)$, $(0, 2)$, $(0, 3)$, $(1, 2)$, $(1, 3)$ et $(2, 3)$. Il reste :

$$\mathcal{F}_3 = \{-4, -5, 0, -1, -1\} = \{-5, -4, -1, 0\}.$$

Enfin, $\mathcal{F}_4$ est : $\left\{ \sum_{k=g}^{d-1} t[k],\ 0 \leqslant g < d \leqslant 4 \right\}$ et finalement :

$$\mathcal{F}_4 = \{-4, -5, -3, -1, 1, 1, 2\} = \{-5, -4, -3, -1, 0, 1, 2\}.$$

Puis $\delta_1 = \max \varepsilon_1 = -4$, $\delta 2 = \max \varepsilon_2 = 0$, $\delta_3 = \max \varepsilon_3 = -1$, $\delta_4 = \max \varepsilon_4 = 2$, $\sigma_1 = \max \mathcal{F}_1 = -4$, $\sigma_2 = \max \mathcal{F}_2 = 0$, $\sigma_3 = \max \mathcal{F}_3 = 0$, $\sigma_4 = \max \mathcal{F}_4 = 2$.

2. Soit $p \in [\![1, n-1]\!]$.

(a) Justifions $\varepsilon_{p+1} = \{t[p]\} \cup \{x + t[p],\ x \in \varepsilon_p\}$.

Pour passer de ε_p à ε_{p+1}, on ajoute des termes en $t[p]$, soit il est tout seul, c'est $\sum_{k=g}^{p} t[k]$ avec $g = p$, soit on l'ajoute à des termes du type $\sum_{k=g}^{p-1} t[k]$ qui sont dans ε_p. Donc ε_{p+1} est bien la réunion de $t[p]$ et de tous les termes de la forme $x + t[p]$, où x parcourt ε_p.

(b) Montrons que $\delta_{p+1} = \max(t[p],\, \delta_p + t[p])$.

On a : $\max\{x + t[p], x \in \varepsilon_p\} = \delta_p = t[p]$. Dons si $\delta_p \leqslant 0$, $\delta_{p+1} = t[p]$ sinon $\delta_{p+1} = \delta_p + t[p]$. Alors :

$$\delta_{p+1} = \max(t[p],\, \delta_p + t[p]).$$

(c) Justifions $\mathcal{F}_{p+1} = \mathcal{F}_p \cup \varepsilon_{p+1}$.

Pour passer de $\mathcal{F}_p$ à $\mathcal{F}_{p+1}$, on ajoute des sommes $\displaystyle\sum_{k=g}^{d-1} t[k]$, où d peut valoir $p+1$ donc des sommes du type $\displaystyle\sum_{k=g}^{p} t[k]$. Ces sommes sont celles de ε_{p+1}. Donc :

$$\mathcal{F}_{p+1} = \mathcal{F}_p \cup \varepsilon_{p+1}.$$

(d) Montrons que $\sigma_{p+1} = \max(\sigma_p,\, \delta_{p+1})$.

On sait que $\sigma_{p+1} = \max \mathcal{F}_{p+1} = \max\left[\mathcal{F}_p \cup \varepsilon_{p+1}\right]$, d'après la question précédente. Or $\sigma_p = \max \mathcal{F}_p$ et $\delta_{p+1} = \max \varepsilon_{p+1}$ et donc :

$$\sigma_{p+1} = \max(\sigma_p, \delta_{p+1}).$$

Remarque : vérifions sur $t = [-4, 0, -1, 2]$. Alors :
$\sigma_2 = \max(\sigma_1, \delta_2) = \max(-4, 0) = 0$. OK
$\sigma_3 = \max(\sigma_2, \delta_3) = \max(0, -1) = 0$. OK
$\sigma_4 = \max(\sigma_3, \delta_4) = \max(0, 2) = 2$. OK

3. Il est clair que $S(t) = \sigma_n$.
4. On fera l'initialisation triviale $\sigma_1 = \delta_1 = t[0]$. Écrivons une fonction $som\_max4(t)$ renvoyant $S(t)$ en terminaison d'une boucle simple calculant les quantités δ_p et σ_p à chaque pas p de la boucle.

```
>>> def som_max4(t) :
        sigma == t[0]
        delta == t[0]
        for p in range(1,len(t)) :
            delta = max(t[p],delta + t[p])
            sigma = max(sigma, delta)
        return sigma
```

5. Calculons $T_4(n)$ le nombre d'additions effectuées.

Il y a au plus une addition à chaque passage de la boucle donc $T_4(n) = n - 1$. Ainsi, $T_4(n) = O(n)$, c'est beaucoup mieux que $T_2(n)$ et $T_3(n)$.

Problème 02 : chemins de poids minimum

Soit M un tableau de n lignes et p colonnes de nombres réels, les lignes étant numérotées de 0 à $n-1$ de haut en bas et les colonnes de 0 à $p-1$, de gauche à droite. On note $M[i,j]$ la valeur située dans la case de coordonnées (i,j).

*Un **chemin** est une liste de couples d'indices, coordonnées de cases du tableau, telles que deux couples consécutifs de la liste vérifient la règle de déplacement suivante : « si l'on se trouve dans la case de coordonnées (i,j), on peut aller soit dans la case de droite $(i,j+1)$, soit dans la case du dessous $(i+1,j)$, à condition de ne pas sortir du tableau.*

*Le **poids** d'un chemin est égal à la somme des nombres contenus dans les cases parcourues.*

*On se donne comme point de départ la case D de coordonnées $(0,0)$, située en haut à gauche, et comme point d'arrivée la case A de coordonnées $(n-1,p-1)$, située en bas à droite. On cherche à déterminer un **chemin optimal**, c'est-à-dire de poids miminum parmi tous les chemins qui mènent de D à A.*

Partie A. Nombre de chemins possibles

On note $\gamma_{n,p}$ le nombre de chemins possibles.

1. On suppose $n = p = 3$. Trouvons $\gamma_{3,3}$.

   Depuis la case $D(0,0)$, on a 6 chemins possibles qui sont :
   • D(0,0) $\longrightarrow (0,1) \longrightarrow (0,2) \longrightarrow (1,2) \longrightarrow A(2,2)$.
   • D(0,0) $\longrightarrow (0,1) \longrightarrow (1,1) \longrightarrow (1,2) \longrightarrow A(2,2)$.
   • D(0,0) $\longrightarrow (0,1) \longrightarrow (1,1) \longrightarrow (2,1) \longrightarrow A(2,2)$.
   • D(0,0) $\longrightarrow (1,0) \longrightarrow (1,1) \longrightarrow (1,2) \longrightarrow A(2,2)$.
   • D(0,0) $\longrightarrow (1,0) \longrightarrow (1,1) \longrightarrow (2,1) \longrightarrow A(2,2)$.
   • D(0,0) $\longrightarrow (1,0) \longrightarrow (2,0) \longrightarrow (2,1) \longrightarrow A(2,2)$.

2. Justifions que choisir un chemin de D à A revient à choisir $n + p - 2$ déplacements élémentaires successifs (en ne distinguant pas un déplacement vers le bas et un déplacement vers la droite).

   Choisir un chemin de D à A revient à choisir $n + p - 2$ déplacements élémentaires successifs parmi lesquels $n - 1$ déplacements vers le bas et $p - 1$ vers la droite. En ne distinguant pas le sens, cela fait bien $n - 1 + p - 1 = n + p - 2$.

3. Écrivons now $\gamma_{n,p}$ sous la forme $\binom{n+p-2}{l}$, où l est un entier que l'on détermine.

   $\gamma_{n,p}$ est le nombre de façons de placer $n-1$ déplacements vers le bas parmi les $n+p-2$ déplacements (les déplacements vers la droite occupant les places restantes). Il s'agit donc du nombre de parties à $n - 1$ éléments d'un ensemble à $n + p - 2$ éléments. C'est $\binom{n+p-2}{n-1}$. On peut aussi prendre $\binom{n+p-2}{p-1}$ qui donne le même nombre.

   Remarque : si $n = p = 3$, on a : $\binom{4}{2} = 6$ et on retrouve le résultat de la question 1.

4. L'ensemble des poids des différents chemins possibles est une partie finie non vide de $\mathbb{R}$. Elle admet un plus petit élément, poids d'au moins l'un des chemins possibles.

Partie B. Détermination du poids d'un chemin optimal

Énumérer tous les chemins possibles s'avère très coûteux pour de grandes valeurs de n, p (on peut montrer par exemple (ce n'est pas demandé ici) que $\gamma_{n,n}$ est un $O(4^n/\sqrt{n})$. Donc on ne va pas chercher le chemin optimal en les trouvant tous. On va s'appuyer sur le principe suivant : toute politique optimale est composée de « sous-politiques » optimales, comme a dit un célèbre enarque (je sais plus qui !).

1. Un exemple. Soit le tableau suivant : $\begin{array}{|c|c|c|}\hline 1 & 4 & 6 \\ \hline 2 & 4 & 0 \\ \hline 3 & 5 & 0 \\ \hline 8 & 0 & 7 \\ \hline \end{array}$.

   Déterminer un chemin optimal et son coût.

   On doit aller de D de poids 1 à A de poids 7. On part de $D(1)$ puis on va à $(1,0)$ de poids 2 puis à $(2,0)$ de poids 3 puis à $(2,1)$ de poids 5 puis on a le choix entre $(2,2)$ et $(3,1)$ de même poids 0 et enfin A de poids 7. Le coût est

   $$1 + 2 + 3 + 5 + 0 + 7 = 18.$$

   On peut imaginer profiter des deux zéros consécutifs en allant de D à $(1,0)$ de poids 2 puis à $(1,1)$ de poids 4 et ensuite $(1,2)$ de poids 0 et $(2,2)$ de poids 0 et enfin A de poids 7. Le coût est :

   $$1 + 2 + 4 + 0 + 0 + 7 = 14.$$

   Ce coût est inférieur et en conclusion, choisir un chemin optimal n'est pas toujours faire le choix à chaque étape élémentaire du poids le plus léger. On remarque par ailleurs que le coût est toujours inférieur ou égal à celui des poids de D et de A et donc supérieur ou égal à 8 ici.

2. Pour tout couple $(i,j) \in [\![0, n-1]\!] \times [\![0, p-1]\!]$, on note $cout(i,j)$ le minimum des poids des chemins menant de la case de coordonnées (i,j) à la case A de coordonnées $(n-1, p-1)$.

   (a) On a immédiatement : $cout(n-1, p-1) = M[n-1, p-1]$.

   (b) On fixe $j \in [\![0, p-1]\!]$. Écrivons $cout(n-1, j)$ sous forme d'une somme.

   Dans ce cas, il n'y a qu'un seul chemin possible de la case numéro $(n-1, j)$ à A car on doit rester sur la dernière ligne. Le coût optimal est celui de ce chemin unique.

   $$\forall\, j \in [\![0, p-1]\!],\ cout(n-1, j) = \sum_{k=j}^{p-1} M[n-1, k].$$

(c) On fixe $i \in [\![0, n-1]\!]$. Écrivons de même $cout(i, p-1)$ sous forme d'une somme.

Dans ce cas aussi, il n'y a qu'un seul chemin possible de la case $(i-1, p)$ à la case $(n-1, p-1) = A$ car on doit rester sur la dernière colonne.

$$\forall\, i \in [\![0, n-1]\!],\ cout(i, p-1) = \sum_{k=i}^{n-1} M[k, p-1].$$

(d) On suppose que $i \in [\![0, n-2]\!]$ et $j \in [\![0, p-2]\!]$. Établissons la relation :

$$cout(i, j) = M[i, j] + \min(cout(i, j+1), cout(i+1, j)).$$

Soit C la case (i, j). Tout chemin optimal menant de C à A est composé de C elle-même suivi soit d'un chemin menant de la case numérotée $(i+1, j)$ à A, soit d'un chemin menant de la case numérotée $(i, j+1)$ à A. De plus, on peut choisir un chemin optimal menant de la case $(i+1, j)$ (resp. $(i, j+1)$) à A sinon le chemin de C à A ne serait pas optimal. Enfin, le choix entre $(i+1, j)$ et $(i, j+1)$ est tranché par la comparaison entre $cout[i+1, j]$ et $cout[i, j+1]$. Donc on a bien :

$$cout(i, j) = M[i, j] + \min(cout(i, j+1), cout(i+1, j)).$$

(e) Écrivons en Python une fonction récursive $cout(i, j)$ qui renvoie le côut d'un chemin optimal partant de (i, j).

```
>>> def cout(i,j) :
        if i == n-1 and j == p-1 :
            return M[i,j]
        if i == n-1 :
            return ( M[i,j] + cout(i,j+1) )
        if j == p-1 :
            return ( M[i,j] + cout(i+1,j) )
        return ( M[i,j] + min(cout(i+1,j) , cout(i,j+1)) )
```

3. Écrivons now une fonction Python nommée $poids\_min(M)$ recevant comme argument M, tableau correspondant aux données et renvoyant le poids d'un chemin optimal. On utilisera $cout(i, j)$ comme sous-procédure.
On rappelle qu'après avoir importé *numpy* avec l'alias *np*, la commande $n, p = np.shape(M)$ donne la dimension du tableau M, c'est-à -dire son nombre de lignes n et son nombre de colonnes p.

```
>>> def poids_min(M) :
        n , p = np.shape(M)
        def cout(i,j) :
            if i == n-1 and j == p-1 :
                return M[i,j]
            if i == n-1 :
                return ( M[i,j] + cout(i,j+1) )
            if j == p-1 :
                return ( M[i,j] + cout(i+1,j) )
            return ( M[i,j] + min(cout(i+1,j) , cout(i,j+1)) )
        return cout(0,0)
```

En effet, on descend jusqu'à $cout(0, 0)$ qui est bien le minimum des poids des chemins menant de la case $(0, 0)$ c'est-à-dire D à la case $(n-1, p-1)$ c'est-à-dire A.