Devoir n°2

Problème : programmation dynamique

I La problématique

On s'intéresse au produit de plusieurs matrices. On rappelle que le produit matriciel AB nécessite que le nombre de colonnes de A soit égal au nombre de lignes de B. On rappelle que le produit matriciel est associatif, c'est à dire que l'on peut parenthéser comme l'on veut. Par contre, il n'est pas commutatif en général. On rappelle aussi que si $A = (a_{i,j})_{n,p}$ et $B = (B_{i,j})_{p,m}$, on a

$$AB = \left(\sum_{k=1}^{p} a_{i,k} b_{k,j} \right)_{\substack{1 \leqslant i \leqslant n \\ 1 \leqslant j \leqslant m}}$$

1. Soit $A \in \mathcal{M}_{n,p}(\mathbb{K})$ et $B \in \mathcal{M}_{p,m}(\mathbb{K})$, montrer que le nombre de multiplications nécessaires de coefficients pour effectuer le produit AB est npm.

2. La problématique apparaît sur l'exemple qui suit : on considère un triple produit matriciel

   $$A_{2,5} B_{5,3} C_{3,1}$$

   où les couples d'indices désignes le nombre de lignes et colonnes de la matrice.

   (a) On effectue le produit matriciel de la façon suivante : $(A_{2,5} B_{5,3}) C_{3,1}$. Combien y-a-t-il de multiplications ?

   (b) On effectue le produit matriciel avec l'autre parenthésage possible : $A_{2,5}(B_{5,3} C_{3,1})$. Combien y-a-t-il de multiplications ?

   (c) Conclusion ?

L'objectif du problème est de trouver une façon optimale (le moins de multiplications possibles de coefficients) pour effectuer un produit matriciel quelconque $M_1 \cdots M_n$, en choisissant un parenthésage optimal.

II Force brute

La méthode de la force brute consiste à déterminer tous les parenthésages possibles et à compter pour chacun d'entre eux le nombre de multiplications, et de choisir un parenthésage qui conduit à un nombre minimum de multiplications de coefficients.

3. Expliquer pourquoi un tel parenthésage optimal existe.

4. Nous allons déterminer le nombre de parenthésages possible d'un produit de n matrices $M_1 \cdots M_n$. On note pour $n \geqslant 1$, C_n le nombre de parenthésages possibles d'un produit de $n+1$ éléments. On convient que $C_0 = 1$.

5. Montrer que $C_1 = 1$, $C_2 = 2$, $C_3 = 5$.

6. Il semble difficile de déterminer C_n directement de manière générale. On se propose de déterminer une relation de récurrence entre les coefficients C_i. Pour cela, on considère un produit de $n+1$ matrices $M_1 \cdots M_n M_{n+1}$ effectué avec un certain parenthésage et nous regardons le dernier produit, soit une forme
$$(M_1 \cdots M_k) \times (M_{k+1} \cdots M_{n+1})$$
avec $k \in \{1, \ldots, n\}$. Justifier que (désormais classique)
$$C_n = \sum_{k=1}^{n} C_{k-1} C_{n-k} = \sum_{k=0}^{n-1} C_k C_{n-1-k} \qquad (*)$$

7. Calculer C_4 et C_5. La formule ci-dessus est-elle efficace ?

8. On reconnaît dans la formule (*) un quasi-produit de Cauchy. On pose alors la série entière associée à la suite (C_n) dite aussi série génératrice
$$f(x) = \sum_{n=0}^{+\infty} C_n x^n$$
On suppose que le rayon de convergence est $R > 0$. Montrer que $f(x) = 1 + x f(x)^2$ sur $]-R, R[$.

9. Montrer qu'il existe une fonction ε à valeurs dans $\{-1, 1\}$ de sorte que si $x \neq 0$
$$f(x) = \frac{1 + \varepsilon(x)\sqrt{1-4x}}{2x}$$

10. Exprimer $\varepsilon(x)$ à l'aide de f et en déduire que ε est constante égale à -1.

11. En déduire que si $x \neq 0$ et
$$f(x) = \frac{1 - \sqrt{1-4x}}{2x}$$

12. On rappelle que
$$\sqrt{1+x} = 1 + \sum_{n=0}^{+\infty} \frac{(-1)^n}{(n+1)2^{2n+1}} \binom{2n}{n} x^{n+1}$$
En déduire que
$$f(x) = \sum_{n=0}^{+\infty} \frac{1}{n+1} \binom{2n}{n} x^n$$
et conclure que pour tout n, $C_n = \frac{1}{n+1}\binom{2n}{n}$. Quel est finalement de rayon de convergence de f ?

13. En déduire un équivalent de C_n lorsque n tend vers $+\infty$. Que peut-on en déduire quant à la méthode de force brute ? On trouvera $C_n \sim \dfrac{4^n}{\sqrt{\pi} n^{3/2}}$.

III Sous-problèmes optimaux

Pour cela, on considère un produit de n matrices $M_1 \cdots M_n$ effectué avec un certain parenthésage optimal (P) pour le nombre de multiplications de coefficients et nous regardons le dernier produit, soit une forme

$$\overbrace{\underbrace{(M_1 \cdots M_k)}_{(P_1)} \times \underbrace{(M_{k+1} \cdots M_n)}_{(P_2)}}^{(P)}$$

avec $k \in \{1, \ldots, n-1\}$.

14. Montrer que le parenthésage (P_1) utilisé pour effectuer le produit $M_1 \cdots M_k$ est aussi optimal ainsi que (P_2) pour le produit $M_{k+1} \cdots M_n$.

15. Quelle méthodologie programmative pourra-t-on donc utiliser ?

IV La relation récursive

On note $m_{i,j}$ le nombre minimum de multiplications de coefficients permettant de calculer le produit $M_i \cdots M_j$, pour $1 \leqslant i \leqslant j \leqslant n$. Nous avons donc pour tout i entre 1 et n, $m_{i,i} = 0$ et nous cherchons ici donc à déterminer $m_{1,n}$.
On définit aussi $p_0, \ldots, p_n$ de sorte que pour tout i entre 1 et n, M_i ait p_{i-1} lignes et p_i colonnes.
On remarquera que si $1 \leqslant i \leqslant n-1$, p_i est à la fois le nombre de colonne de M_i et le nombre de ligne de M_{i+1}.

16. Justifiez, par optimalité, en regardant le dernier produit possible, que si $1 \leqslant i < j$,

$$m_{i,j} = \min_{i \leqslant k < j} (m_{i,k} + m_{k+1,j} + p_{i-1} p_k p_j)$$

Montrer précisément sur un schéma quels sont les $m_{u,v}$ utilisés pour la calcul de $m_{i,j}$.

17. Écrire une fonction récursive qui permet de déterminer le nombre minimum de multiplications de coefficients pour effectuer un produit de matrice.

18. Quel est la problématique d'un tel algorithme ?

V Méthode dynamique

19. Quelle est la «bonne façon» de construire le tableau des $m_{u,v}$?
On note aussi $I_{i,j}$ l'indice k de la formule (*) qui permet de minimiser le calcul de $m_{i,j}$.
Construire les tableaux m et I associé au produit $A_{2,5} B_{5,3} C_{3,2} D_{2,1}$ et en déduire le nombre minimal de multiplications de coefficients ainsi qu'un parenthésage associé.
Vérifier le résultat en comparant à tous les autres parenthésages possibles (il y en a $C_3 = 5$, ça va).

20. Écrire une fonction de paramètre **p** la liste $[p_0, \ldots, p_n]$ qui construit les tableaux m et I et qui renvoie donc $m_{1,n}$.
On donne le squelette d'une telle fonction, à vous de remplir les ? :

```python
import math as m
def parenthese(p):
    nn = len(p)                                  # c'est n+1 du sujet
    t_m = [[None]*nn for _ in range(nn)] # tableau m indice 0 - n
    t_I = [[None]*nn for _ in range(nn)] # tableau I indice 0 - n
    """ les valeurs utilisées seront pour les indices 1<=i<=j<=n
    on rend ainsi les notations indicielles compatibles python """
    for i in range(1,nn):
        t_m[i][i] =  ?????
    for d in range(nn-1): # nn-1=n diagonales à remplir
        for i in range(1,nn-1-d): # balayage de la diagonale
            # calcul de m[i][i+d+1]
            # avec recherche du minimum
            mini = m.inf # initialisation à +infini
            j = i+d+1
            for l in range(i,j):
                m = t_m[?][?] + t_m[l+1][?] + p[?]*p[l]*p[?]
                ?????
                ?????
                ?????
            t_m[i][j] = mini
            t_I[i][j] = indice
    return t_m, t_I, t_m[1][?]
```

21. Écrire une fonction de paramètre la liste **p** qui renvoie par un backtracing un parenthésage optimal (en renvoyant sur l'exemple $A_{2,5}B_{5,3}C_{3,2}D_{2,1}$ la chaîne de caractère $*(*(**))$ modélisant le produit parenthésé optimal $A_{2,5}(B_{5,3}(C_{3,2}D_{2,1}))$).
Indications : procédez par une sous-procédure récursive ; et il sert à quoi le tableau I ?

VI Une optimation hackée

22. Un espion s'est glissé dans l'équipe informatique du laboratoire Google, et dans le but de faire perdre de l'argent à l'entreprise, il modifie tous les algorithmes de produit matriciel dans le but de les rendre le plus lent possible en reprogrammant selon le coût maximum en produits de coefficients. Que doit-il modifier ?

23. Comme à la question 19, construire les tableaux m' et I' associé au produit $A_{2,5}B_{5,3}C_{3,2}D_{2,1}$ et en déduire le nombre maximum de multiplications de coefficients ainsi qu'un parenthésage associé. Vérifier le résultat en comparant à tous les autres parenthésages possibles.

Définition de dynamisme :
1. Philosophie de l'énergie créatrice.
2. Énergie, vitalité.

Synonymes :
vitalité, pep, tonus, ressort, force vitale.

Contraires :
débilité, marasme, mollesse, statisme, aboulie, nonchaloir, adynamie, atonie, passivité

VII La problématique

1. Soit $A \in \mathcal{M}_{n,p}(\mathbb{K})$ et $B \in \mathcal{M}_{p,m}(\mathbb{K})$, le nombre de multiplications nécessaires pour calculer le coefficient $c_{i,j} = \sum_{k=1}^{p} a_{i,k}b_{k,j}$ est p, et il y a nm coefficients, et donc le nombre cherché est npm.

2. La problématique apparaît sur l'exemple qui suit : on considère un triple produit matriciel

$$A_{2,5}B_{5,3}C_{3,1}$$

   (a) On effectue le produit matriciel de la façon suivante : $(A_{2,5}B_{5,3})C_{3,1}$. Il y a $2 \times 5 \times 3 = 30$ multiplications pour $A_{2,5}B_{5,3}$ qui est de taille 2,3 puis $2 \times 3 \times 1 = 6$ multiplications pour le produit avec $C_{3,1}$ soit en tout 36 multiplications.

   (b) On effectue le produit matriciel avec l'autre parenthésage possible : $A_{2,5}(B_{5,3}C_{3,1})$. Il y a $5 \times 3 \times 1 = 15$ multiplications pour $B_{5,3}C_{3,1}$ qui est de taille 5,1 puis $2 \times 5 \times 1 = 10$ multiplications pour le produit avec $A_{2,5}$ soit en tout 25 multiplications.

   (c) On utilisera donc plutôt le second parenthésage pour effectuer notre produit matriciel.

L'objectif du problème est de trouver une façon optimale (le moins de multiplications possibles de coefficients) pour effectuer un produit matriciel quelconque $M_1 \cdots M_n$, en choisissant un parenthésage optimal.

VIII Force brute

La méthode de la force brute consiste à déterminer tous les parenthésages possibles et à compter pour chacun d'entre eux le nombre de multiplications, et de choisir un parenthésage qui conduit à un nombre minimum de multiplications de coefficients.

3. Un tel parenthésage optimal existe puisque l'ensemble constitué des nombres de multiplications nécessaires associé à un parenthésage est une partie non vide de $\mathbb{N}$ (elle est ici finie mais ce n'est pas important ici, par contre pour la dernière question, oui, pour avoir l'existence d'un plus grand élément), et donc elle admet un plus petit élément, qui est donc le nombre de multiplications d'un parenthésage optimal. Il n'y a pas forcément unicité.

4. Nous allons déterminer le nombre de parenthésages possible d'un produit de n matrices $M_1 \cdots M_n$. On note pour $n \geqslant 1$, C_n le nombre de parenthésages possibles d'un produit de $n+1$ éléments. On convient que $C_0 = 1$.

5. Pour $n = 1$, une seule façon de faire le produit AB : $C_1 = 1$.
   Pour $n = 2$, nous avons pour le produit ABC : $(AB)C$ ou $A(BC)$: $C_2 = 2$.
   Pour $n = 3$, nous avons pour le produit $ABCD$: $(AB)(CD)$, $A(B(CD))$, $A((BC)D)$, $((A(BC))D$, $((AB)C)D$ et $(AB)(CD)$: $C_3 = 5$.

6. On considère un produit de $n+1$ matrices $M_1 \cdots M_n M_{n+1}$ effectué avec un certain parenthésage et nous regardons le dernier produit, soit une forme

$$(M_1 \cdots M_k) \times (M_{k+1} \cdots M_{n+1})$$

avec $k \in \{1, \ldots, n\}$. Alors pour une valeur de k donnée, il y a C_{k-1} parenthésages possibles de $M_1 \cdots M_k$ et $C_{n+1-(k+1)}$ parenthésages possibles de $M_{k+1} \cdots M_{n+1}$, et donc y a $C_{k-1} \times C_{n+1-(k+1)}$ parenthésages possibles de ce type ; comme les cas $k = 1$ ou bien $k = 2$, ..., ou bien $k = n$ soit disjoints, et recouvrants, nos avons

$$C_n = \sum_{k=1}^{n} C_{k-1} C_{n-k} = \sum_{k=0}^{n-1} C_k C_{n-1-k} \qquad (*)$$

7. On peut vérifier déjà :
– pour $n = 1$: $C_1 = C_0 C_0 = 1$.
– pour $n = 2$: $C_2 = C_0 C_1 + C_1 C_0 = 1 + 1 = 2$.
– pour $n = 3$: $C_3 = C_0 C_2 + C_1 C_1 + C_2 C_0 = 2 + 1 + 2 = 5$.
– pour $n = 4$: $C_4 = C_0 C_3 + C_1 C_2 + C_2 C_1 + C_3 C_0 = 5 + 2 + 2 + 5 = 14$.
– pour $n = 5$: $C_5 = C_0 C_4 + C_1 C_3 + C_2 C_2 + C_3 C_1 + C_4 C_0 = 14 + 5 + 4 + 5 + 14 = 52$.

La formule ci-dessus n'est pas très efficace. Une programmation récursive est possible mais avec beaucoup de chevauchement de sous-problèmes ; on peut cependant obtenir une programmation plus efficace avec mémoïsation ou par utilisation d'un tableau pour stocker les valeurs à réutiliser.

8. On reconnaît dans la formule (*) un quasi-produit de Cauchy. On pose alors la série entière associée à la suite (C_n) dite aussi série génératrice

$$f(x) = \sum_{n=0}^{+\infty} C_n x^n$$

On suppose que le rayon de convergence est $R > 0$. Alors on multiplie dans $(*)$ par x^n, puis on somme pour $n \geqslant 1$,

$$\begin{aligned} \sum_{n=1}^{+\infty} C_n x^n &= \sum_{n=1}^{+\infty} \sum_{k=0}^{n-1} C_k C_{n-1-k} x^n \\ &= x \sum_{n=1}^{+\infty} \sum_{k=0}^{n-1} C_k C_{n-1-k} x^{n-1} \\ &= x f(x)^2 \end{aligned}$$

et en ajoutant 1,

$$f(x) = 1 + x f(x)^2$$

9. Pour tout $x \neq 0$, $f(x)$ est donc racine d'une équation du second degré en u $xu^2 - u + 1 = 0$, de discriminant $\Delta = 1 - 4x$, et au voisinage de 0, $\Delta > 0$ et ainsi il existe une fonction ε à valeurs dans $\{-1, 1\}$ de sorte que si $x \neq 0$

$$f(x) = \frac{1 + \varepsilon(x)\sqrt{1 - 4x}}{2x}$$

Pour $x = 0$, nous avons $f(0) = 1$.

10. On a en fait si $x \neq 0$,
$$\varepsilon(x) = \frac{2xf(x) - 1}{\sqrt{1-4x}}$$
et donc ε est une fonction continue (en prolongeant en 0 par $\varepsilon(0) = -1$), sur $]-R, R[\cap]-1/4, 1/4[$ par opérations, sachant que f est la somme d'une série entière, donc comme elle prend ses valeurs dans $\{-1, 1\}$, et comme elle ne peut pas prendre les 2 valeurs (puisque sinon, elle devrait prendre toutes les valeurs intermédiaires), elle est constante égale à -1.

11. Ainsi, si $x \neq 0$ et
$$f(x) = \frac{1 - \sqrt{1-4x}}{2x}$$

12. On rappelle que
$$\sqrt{1+x} = \sum_{n=0}^{+\infty} \frac{(-1)^n}{(n+1)2^{2n}} \binom{2n}{n} x^n$$
Ainsi
$$f(x) = \sum_{n=0}^{+\infty} \frac{1}{n+1} \binom{2n}{n} x^n$$
et donc par unicité des coefficients d'une série entière, pour tout n, $C_n = \frac{1}{n+1}\binom{2n}{n}$. Comme
$$\frac{C_{n+1}}{C_n} = \frac{\frac{1}{n+2}\binom{2n+2}{n+1}}{\frac{1}{n+1}\binom{2n}{n}} = \frac{n+1}{n+2}\frac{(2n+2)!n!n!}{(n+1)!(n+1)!} \sim 4$$
et ainsi finalement de rayon de convergence de f est $R = 1/4$.

13. On a aussi par la formule de Stirling
$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{1}{n+1} \times \frac{\sqrt{2\pi.2n}\mathrm{e}^{-2n}(2n)^{2n}}{2\pi n \mathrm{e}^{-2n} n^{2n}} \frac{4^n}{\sqrt{\pi} n \sqrt{n}}$$
Nous obtenons une complexité en $\mathcal{O}(4^n n^{-3/2})$. La méthode de la force brute sera inadaptée pour n grand.

IX Sous-problèmes optimaux

14. Si le parenthésage (P_1) utilisé pour effectuer le produit $M_1 \cdots M_k$ n'était pas optimal, on pourrait le remplacer par un autre (P_1') plus efficace, et alors
$$\overbrace{(\underbrace{M_1 \cdots M_k}_{(P_1')}) \times (\underbrace{M_{k+1} \cdots M_n}_{(P_2)})}^{(P')}$$
serait plus efficace que le parenthésage (P), ce qui est contradictoire.
De même pour (P_2). Nous sommes bien dans une situation où les sous-problématiques d'une problématique optimale sont encore optimales.

15. On pourra donc procéder par une méthode récursive (de complexité trop élevée par les chevauchements de sous-problèmes, sauf à utiliser la mémoïsation), mais aussi surtout par une méthode dynamique.

X La relation récursive

16. On étudie le produit $M_i \cdot M_j$ et on regarde le parenthésage de dernier niveau, de la forme donc

$$(\underbrace{M_i \cdots M_k}) \times (\underbrace{M_{k+1} \cdots M_j})$$

avec k, de sorte que $i \leqslant k < j$. Alors :
– ou $k = i$;
– ou $k = i + 1$;
⋮ – ou $k = j - 1$;
et pour une valeur de k, nous avons le nombre de multiplications minimum de coefficients pour ce parenthésage qui vaut

$$m_{i,k} + m_{k+1,j} + p_{i-1}p_k p_j$$

avec :
– $m_{i,k}$ pour $M_i \cdots M_k = A$
– $m_{k+1,j}$ pour $M_{k+1} \cdots M_j = B$
– $p_{i-1}p_k p_j$ pour AB avec les tailles p_{i-1}, p_k, p_j.
Ainsi, par optimalité,

$$m_{i,j} = \min_{i \leqslant k < j} (m_{i,k} + m_{k+1,j} + p_{i-1}p_k p_j)$$

Ainsi pour calculer $m_{i,j}$, nous utilisons un certain nombre de valeurs de type $m_{u,v}$. On précise cela, puisque le «remplissage» du tableau m doit s'effectuer de façon particulière pour aboutir. On rappelle que la valeur finale qui nous intéresse est $m_{1,n}$.
– pour $k = i$: $m_{i,i}$ et $m_{i+1,j}$
– pour $k = i + 1$: $m_{i,i+1}$ et $m_{i+2,j}$
⋮
– pour k : $m_{i,k}$ et $m_{k+1,j}$
⋮
– pour $k = j - 1$: $m_{i,j-1}$ et $m_{j,j}$
Graphiquement,

17. De façon récursive :

```python
def parenthese_r(p, i, j):  # la fonction récursive
    global c
    c += 1
    # test d'arrêt
    if i == j :
        return 0
    # calcul de m[i][j]
    # avec recherche du min
    mini = ma.inf
    for l in range(i,j):
        m = parenthese_r(p,i,l)+parenthese_r(p,l+1,j)+p[i-1]*p[l]*p[j]
        if m < mini :
            mini = m

    return mini

def parenthese_recursif(p):  # la fonction appelante
    i = 1                    # avec les paramètres
    j = len(p)-1             # pour obtenir la valeur cherchée
    return parenthese_r(p, i, j)
```

18. Un tel algorithme est peu efficace, il y a trop de recouvrement, la complexité selon n sera trop grande. On pourrait palier à ce problème avec l'utilisation de mémoïsation à l'aide d'un dictionnaire ou d'une table. Et bien justement, la programmation dynamique se fera par le remplissage d'une table, sans recouvrement, avec la remarque de la question précédente.

XI Méthode dynamique

19. On a vu que l'on devait remplir le tableau par diagonale successive, la première la diagonale principale remplie de 0, qui «remontent» pour finir par le calcul de $m_{1,n}$ en dernier, ce qui est un peu logique.

    On construit les tableaux m et I associé au produit $A_{2,5}B_{5,3}C_{3,2}D_{2,1}$.

i \ j	1	2	3	4
1	0	30	42	31
2		0	30	21
3			0	6
4				0

i \ j	1	2	3	4
1		1	2	1
2			2	2
3				3
4				

    Ainsi le nombre minimal de multiplications de coefficients est 31 avec le parenthésage associé $A_{2,5}B_{5,3}C_{3,2}D_{2,1}$

    Nous avons $C_3 = 5$ parenthésages possibles :
    - $(A_{2,5}B_{5,3})(C_{3,2}D_{2,1})$: 30 + 6 + 6 = 42
    - $A_{2,5}(B_{5,3}(C_{3,2}D_{2,1}))$: 6 + 15 + 10 = 31
    - $(A_{2,5}((B_{5,3}C_{3,2})D_{2,1})$: 30 + 10 + 10 = 50
    - $((A_{2,5}B_{5,3})C_{3,2})D_{2,1}$: 30 + 12 + 4 46
    - $(A_{2,5}(B_{5,3}C_{3,2}))D_{2,1}$: 30 + 20 + 4 = 54

20. On complète le squelette :

```python
import math as m

def parenthese(p):
    nn = len(p)                          # c'est n+1 du sujet
    t_m = [[None]*nn for _ in range(nn)] # tableau m indice 0 - n
    t_I = [[None]*nn for _ in range(nn)] # tableau I indice 0 - n
    """ les valeurs utilisées seront pour les indices 1<=i<=j<=n
    on rend ainsi les notations indicielles compatibles python """
    for i in range(1,nn):
        t_m[i][i] =  0
    for d in range(nn-1): # nn-1=n diagonales à remplir
        for i in range(1,nn-1-d): # balayage de la diagonale
            # calcul de m[i][i+d+1]
            # avec recherche du minimum
            mini = m.inf # initialisation à +infini
            j = i+d+1
            for l in range(i,j):
                m = t_m[i][l] + t_m[l+1][j] + p[i-1]*p[l]*p[j]
                if m < mini:   # on a trouvé plus petit
                    mini = m   # nouvelle valeur minimale temporaire
```

```
                    indice = l # la nouvelle position du parenthésage interne
            t_m[i][j] = mini
            t_I[i][j] = indice

    return t_m, t_I, t_m[1][nn-1]
```

21. On donne la procédure pour l'affichage symbolique d'un parenthésage optimal :

```
def representation(p):
    n = len(p)

    mul = [[None]*n for _ in range(n)]
    par = [[None]*n for _ in range(n)]
    for i in range(1,n):
        mul[i][i] = 0

    for d in range(n-1): # n-1 diagonales à remplir
        for i in range(1,n-1-d):
            # calcul de m[i][i+d+1]
            # avec recherche du min
            min = ma.inf
            j = i+d+1
            for l in range(i,j):
                m = mul[i][l] + mul[l+1][j] + p[i-1]*p[l]*p[j]
                if m < min :
                    min = m
                    indice = l
            mul[i][j] = min
            par[i][j] = indice

    # construction de la représentation d'un parenthésage optimal
    # par récursivité

    def parenthesage(par, i, j):
        if i == j:
            return "*"

        c1 = parenthesage(par, i, par[i][j])
        c2 = parenthesage(par, par[i][j]+1, j)
        return '(' + c1 + c2 + ')'

    # on supprime les parenthèses extérieures
    return parenthesage(par, 1, n-1)[1:-1]
```

XII Une optimation hackée

22. Il suffit de calculer un maximum au lieu d'un minimum dans la formule (*), soit :

```
import math as m

def parenthese_maxi(p):
    nn = len(p)                          # c'est n+1 du sujet
    t_m = [[None]*nn for _ in range(nn)] # tableau m indice 0 - n
    t_I = [[None]*nn for _ in range(nn)] # tableau I indice 0 - n
    """ les valeurs utilisées seront pour les indices 1<=i<=j<=n
    on rend ainsi les notations indicielles compatibles python """
    for i in range(1,nn):
        t_m[i][i] =  0
    for d in range(nn-1): # nn-1=n diagonales à remplir
        for i in range(1,nn-1-d): # balayage de la diagonale
            # calcul de m[i][i+d+1]
            # avec recherche du maximum
            maxi = 0 # initialisation à 0
            j = i+d+1
            for l in range(i,j):
                m = t_m[i][l] + t_m[l+1][j] + p[i-1]*p[l]*p[j]
                if m > maxi:   # on a trouvé plus grand
                    maxi = m   # nouvelle valeur maximale temporaire
```

```
                    indice = l # la nouvelle position du parenthésage interne
                t_m[i][j] = maxi
                t_I[i][j] = indice

    return t_m, t_I, t_m[1][nn-1]
```

23. On obtient les tables :

i \ j	1	2	3	4
1	0	30	50	**54**
2		0	30	40
3			0	6
4				0

i \ j	1	2	3	4
1		1	1	3
2			2	3
3				3
4				