Parenthésages optimaux de produits de matrices

Nous savons que la multiplication de matrices est associative. C'est-à-dire que, lorsque cela a un sens, les produits $A \times (B \times C)$ et $(A \times B) \times C$ sont égaux, ce qui fait qu'on note $A \times B \times C$ ou ABC sans préciser la place des parenthèses. En revanche, les choses changent quand il s'agit de réaliser le calcul. Lorsque $A \in \mathscr{M}_{p,q}$, $B \in \mathscr{M}_{q,r}$, $C \in \mathscr{M}_{r,s}$, on a toujours $A(BC) = (AB)C \in \mathscr{M}_{r,s}$ mais le premier calcul aura coûté $pqs + qrs$ multiplications alors que le second en aura coûté $pqr + prs$. Par exemple, avec $p = 2$, $q = 6$, $r = 3$ et $s = 5$. On effectue respectivement 150 ou 66 multiplications.
Dans certaines applications, où l'on a à effectuer des produits matriciels en cascade avec des matrices de grandes tailles, il y a un grand intérêt à planifier l'organisation des calculs et donc à choisir un parenthésage optimal.

1 Sous-structures optimales

Considérons une suite de n matrices telle que les produits A_iA_{i+1} soit définis. Pour $0 \leqslant i < n-1$, on note l_i le nombre de lignes de A_i et on pose l_n le nombre de colonnes de A_{n-1} : on a donc $A_i \in \mathscr{M}_{l_i,l_{i+1}}$. Nous voulons savoir quels parenthésages permettront d'optimiser le nombre de multiplications pour calculer le produit des A_i. Imaginons qu'un tel parenthésage optimale conduise à effectuer comme dernière opération le produit de deux matrices : $(A_0 \cdots A_k) \times (A_{k+1} \cdots A_{n-1})$. Ce dernier produit matriciel demandera à lui seul $l_0 \times l_{k+1} \times l_n$ multiplications. Le nombre total de multiplications sera donc égal à la somme de trois termes :

- le nombre de multiplication pour le calcul de $A_0 \cdots A_k$
- le nombre de multiplication pour le calcul de $A_{k+1} \cdots A_{n-1}$
- $l_0 \times l_{k+1} \times l_n$

On note $m(i,j)$ le nombre minimal de multiplications pour calculer $A_i \times \cdots \times A_j$, et posons $m(i,i) = 0$.

1. Justifier que si cette somme de trois termes est optimale, le premier et le second sont aussi obtenus avec des parenthèsages optimaux.
2. Que vaut $m(i, i+1)$?
3. Pour calculer le produit $A_i \cdots A_j$ on peut calculer séparément $A_i \cdots A_k$ et $A_{k+1} \cdots A_j$, avec $i \leqslant k < j$. Justifier, qu'avec ce découpage, le nombre de multiplications est $m(i,k) + m(k+1,j) + l_i l_{k+1} l_{j+1}$.

Ainsi, $m(i,j)$ est la valeur minimale parmi les $m(i,k) + m(k+1,j) + l_i l_{k+1} l_{j+1}$ avec $i \leqslant k < j$.

2 Version récursive

On rappelle que le nombre optimal de multiplications pour le calcul du produit matriciel $A_i \cdots A_j$ vérifie la relation :

$$m(i,j) = \min\big(m(i,k) + m(k+1,j) + l_i l_{k+1} l_{j+1}\big),\ i \leqslant k < j$$

Soit $L = [l_0, l_1, \cdots, l_n]$ une suite de $n+1$ entiers que l'on interprète comme les nombres de lignes et colonnes de n matrices.

1. On suppose qu'une fonction récursive `choix_parenthesage_rec(L, i, j)` calcule $m(i,j)$. Que serait l'arbre des appels avec $i = 0$, $j = 4$, avec une programmation naïve sans utiliser de dictionnaire pour stocker les résultats intermédiaires lorsqu'ils sont calculés ?
2. Le dictionnaire `D` de mémoïsation contient les associations `{(i, j) : m(i,j) }`. Écrire une fonction récursive `choix_parenthesage_rec(L, i, j)` qui utilise le dictionnaire de mémoïsatoin `D` et qui renvoie le nombre minimal d'opérations pour le produit $A_i \cdots A_j$. *Aide : on pourra calculer tous les $m(i,k) + m(k+1,j) + l_i l_{k+1} l_{j+1}$ pour $i \leqslant k < j$ et ne conserver en mémoire que le plus petit. Aide bis : écrire au brouillon l'algorithme classique de recherche de minimum dans une liste.*

3 Version ascendante

On se propose d'écrire un algorithme qui calcule les $m(i,j)$ et les stocke dans un tableau `M` par une approche ascendante. Cet algorithme remplit parallèlement un tableau `S` de même taille que le précédent, initialisé à -1 dans lequel `S[i][j]` est l'entier k tel que $m(i,j) = m(i,k) + m(k+1,j) + l_i l_{k+1} l_{j+1}$, (*i.e.* celui pour lequel on fait le découpage optimal).

1. On suppose les valeurs de $m(i,j)$ calculées et stockées. Que dire de la matrice `M` ? Où trouvera-t-on le résultat $m(0,n)$ correspondant à l'optimum cherché ?
2. Pour d donné, à quoi les termes `M[i][i+d]` correspond-ils ?
3. On reprend l'exemple du produit de 3 matrices donné en exemple plus haut. On a `L = [2, 6, 3, 5]`. Remplir les tableaux `S` et `M` en expliquant dans quel ordre et comment ils peuvent être remplis.
4. Écrire une fonction d'entête `choix_parenthesage(L : list)` qui remplit et renvoie le tuple `(S, M)`. Conseil : remplir dans l'ordre les sur-diagonales `M[i, i+1]` puis `M[i, i+2]`, etc.
5. Écrire une fonction `cout_cascade(L : list)` qui prend en argument une liste `L` et qui renvoie le nombre de multiplications effectuées lors du calcul en cascade de $A_0 \cdots A_{n-1}$. C'est-à-dire en effectuant les multiplications dans l'ordre d'écriture : d'abord A_0A_1 puis $(A_0A_1)A_2$ puis $((A_0A_1)A_2)A_3$ etc.
6. On écrit le script suivant avec les fonctions définies plus haut :

```
L = [1200, 2000, 345, 560, 1000, 489]
S, M = choix_parenthesage(L)
c = cout_cascade(L)
print(S)
print(c, M[0, len(L)-2]/c)
```

   Son exécution provoque les affichages suivants :

```
[[-1.  0.  1.  1.  1.]
 [-1. -1.  1.  1.  1.]
 [-1. -1. -1.  2.  3.]
 [-1. -1. -1. -1.  3.]
 [-1. -1. -1. -1. -1.]]
2318640000 0.6005033122865128
```

   Placer les parenthèses qui donnent l'optimum pour $A_0A_1A_2A_3A_4$.
7. Écrire une fonction récursive `parenthese_opt(S, i, j)` qui prend le tableau `S` en argument, `i` et `j` des entiers et renvoie la chaîne représentant l'expression parenthésée de façon optimale. Par exemple `(A0A1)((A2A3)A4)`. On rappelle que `str(i)` convertit l'entier stocké dans la variable `i` en chaîne de caractère. *Aide : posez-vous la question des cas de base (valeurs de i et j...). Utilisez la concaténation des chaînes de caractères.*