

TD n°6

Programmation dynamique

Multiplication chaînée de matrices

On souhaite multiplier n matrices $M_1 \cdot M_2 \cdot \dots \cdot M_n$, où M_i est une matrice de dimension $d_{i-1} \times d_i$.

On suppose qu'un algorithme `mult(A,B)` multiplie A et B en temps $k \cdot l \cdot m$ où A est de dimension $k \times l$ et B est de dimension $l \times m$.¹

Comme la multiplication matricielle est associative, $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$. L'ordre des multiplications est important pour la complexité. Prenez l'exemple suivant :

$$\begin{array}{ccccccc}
 & & & & \circ & & 1 \\
 a & b & c & & g & h & i & j & 1 & \circ & s & 1 \\
 & & & & @ & k & l & m & n & A & @ & t & C \\
 d & e & f & & & o & p & q & r & & & u & A \\
 & & & & & & & & & & & v &
 \end{array}$$

Si on multiplie d'abord les deux premières matrices, le coût est $2 \times 3 \times 4 = 24$, puis on multiplie le résultat (une matrice 2×4) par la troisième, ce qui ajoute un coût de $2 \times 4 \times 1 = 8$, on a un coût total de $24 + 8 = 32$. Si on multiplie d'abord les deux dernières pour obtenir une matrice 3×1 , le temps sera de $(3 \times 4 \times 1) + (2 \times 3 \times 1) = 12 + 6 = 18$, une économie considérable.

Vous devez faire un algorithme de type programmation dynamique pour trouver le coût optimal pour multiplier les matrices $M_1 \cdot \dots \cdot M_n$. On note $T_{i,j}$ le coût optimal pour multiplier les matrices $M_i \cdot \dots \cdot M_j$.

1. Expliquez comment décomposer le problème de calculer $T_{i,j}$ en problèmes plus simples.
2. Donnez une récurrence qui permet de calculer $T_{i,j}$.
3. Donnez les conditions initiales.
4. Donnez l'algorithme qui calcule le coût optimal pour multiplier des matrices de dimensions $(d_0 \times d_1); (d_1 \times d_2); \dots; (d_{n-1} \times d_n)$. Les dimensions sont données dans un tableau D indicé de 0 à n .
5. Donnez la complexité de votre algorithme (ne comptez pas dans la complexité le temps de multiplication des matrices, mais seulement le temps pour trouver le coût optimal).
6. Expliquez comment retrouver l'ordre optimal de multiplication de matrices (de préférence en donnant l'algorithme complet qui multiplie les matrices dans l'ordre optimal).

Distance d'edition en genomique

On considère qu'un **gène** est un mot sur l'alphabet $\{A; T; G; C\}$.

On considère trois opérations de base permettant de transformer un gène en un autre :

1. *Insertion* d'une lettre. Exemple AAA devient AATA par insertion d'un T en troisième position
2. *Suppression* d'une lettre. ATCC devient ACC par suppression d'un T en deuxième position

1. C'est le cas de l'algorithme simple qui consiste à calculer la matrice résultat en coordonnée i, j , pour $1 \leq i \leq k$ et $1 \leq j \leq m$ en multipliant la i ème ligne de A avec la j ème colonne de B utilise l multiplications élémentaires par coordonnée, soit un total de $(km)l$ multiplications élémentaires. Il existe des algorithmes plus efficaces qui multiplient deux matrices $n \times n$ en temps beaucoup plus petit, $n^{2.3727}$.

3. *Substitution.* Exemple AAAT devient ACAT en substituant un C au deuxième A. .

Une **séquence d'édition** entre un gène A et un gène B est une suite $A = G_0; G_1; G_2; \dots; G_{k-1}; G_k = B$ de gènes telle que l'on passe de G_i à G_{i+1} par l'une des trois opérations élémentaires autorisées (insertion, suppression, substitution). k est la longueur de la séquence d'édition.

La **distance d'édition** (aussi appelée distance de Levenshtein) entre deux gènes A et B est la longueur d'une plus petite séquence d'édition entre A et B .

On se propose de calculer la distance d'édition entre deux gènes A et B par récurrence sur leur longueur. Posons quelques notations :

- $A[i]$ est la i ème lettre du gène A
- $A[1::i]$ est le gène constitué des i premières lettres de A
- $n = |A|$ est le nombre de lettres de A et $m = |B|$ celui de B .
- $dp(i; j)$ est la distance d'édition entre $A[1::i]$ et $B[1::j]$.

Ainsi ce que l'on veut est calculer $dp(n; m)$.

Par exemple pour $A = \text{ATATCCCG}$ et $B = \text{ATTTCGC}$ on a la suite de gènes

$\text{ATATCCCG} \rightarrow \text{ATTTCCCG} \rightarrow \text{ATTTCGCG} \rightarrow \text{ATTTCGC}$

(substitution de A par T ; substitution de C par G ; suppression G) donc $dp(8; 7) = 3$.

Exercice 1 Que vaut $dp(0; 0)$?

Exercice 2 Montrer que si $A[i] = B[j]$ alors $dp(i; j) = dp(i - 1; j - 1)$

Exercice 3 Montrer que si $A[i] \neq B[j]$ alors $dp(i; j) = 1 + \min(dp(i - 1; j); dp(i; j - 1); dp(i - 1; j - 1))$.

Exercice 4 En déduire une fonction récursive `dp(entier i, entier j, gene A, gene B)` calculant $dp(i; j)$.

Exercice 5 Donner un pire cas pour le nombre d'appels récursifs effectués par `dp(n, m, A, B)`.

Exercice 6 En remarquant qu'un grand nombre d'appels récursifs sont identiques, et en utilisant un tableau de mémorisation, donner une version de `dp` qui marche en temps polynomial.

Exercice 7 Préciser les complexités en temps et en espace de ce nouvel algorithme.

Exercice 8 On veut aussi retrouver la séquence d'édition correspondant à la longueur calculée. Modifier votre algorithme pour qu'il affiche la séquence qui transforme A en B . L'affichage ressemblera, par exemple, à : "Insertion de T en 2ème position. Substitution de C par G en 4ème position. Suppression de T en 5ème position".

Exercice 9 On prend maintenant une modélisation plus réaliste au niveau biologique. Les opérations ont un *coût* :

- Substituer un 'A' par un 'T', ou un 'T' par un 'A', ou un 'C' par un 'G', ou un 'G' par un 'C', coûte 1.
- Les autres substitutions (entre l'ensemble {'A', 'T'} et l'ensemble {'C', 'G'}) donc coûtent 3.
- Insertions et suppressions coûtent 5.

Le coût d'une séquence d'édition est la somme des coûts de ses opérations élémentaires. La distance d'édition entre deux gènes est maintenant le coût minimum d'une séquence d'édition.

Expliquer rapidement les modifications à apporter à l'algorithme de pour qu'il résolve cette variante du problème (on veut juste le coût, pas la séquence d'édition).