I.T.C. D.M. 2 : Programmation dynamique : deux exemples

D.M. pour le jeudi 5 janvier

1 Fin du T.P. 3 : algorithme de Seam Carving

Rédiger les solutions aux questions 16 à 19 du T.P.3. Le corrigé des questions 1 à 15 est disponible sur la page Informatique 2ème année.

2 Problème des parenthésages pour le produit matriciel

D'après un exemple du cours de Vincent Puyhaubert.

2.1 Introduction

Nous savons que la multiplication matricielle 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 lorsqu'on ne s'intéresse qu'au produit lui-même. Par contre, les choses changent quand il s'agit de réaliser concrètement le calcul et qu'on compte les multiplications.

En effet, si $A \in \mathcal{M}_{p,q}, B \in \mathcal{M}_{q,r}$ alors le calcul de chacune des entrées de AB par la formule :

$$(A,B)(i,j) = \sum_{k=1}^{q} a_{i,k}b_{k,j}$$

coûte q multiplications, et donc le calcul de la matrice AB qui a $p \times r$ entrées coûte pqr multiplications.

Q 1) On rajoute une troisième matrice $C \in \mathcal{M}_{r,s}$, on a toujours $A(BC) = (AB)C \in \mathcal{M}_{p,s}$ mais déterminer le nombre de multiplications de scalaires pour

a) le produit $(AB)C$

b) le produit $A(BC)$

Illustration :

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

Il y aura dans l'exemple illustré $qs(p+r) = 150$ ou $\mathrm{pr}(q+s) = 66$ multiplications selon le parenthésage.

Dans certaines applications (à trouver ! !), où l'on aurait à effectuer des produits matriciels en cascade avec des matrices de grande tailles, il y a un grand intérêt à planifier l'organisation des calculs et donc à choisir un parenthésage optimal.

2.2 Excursion mathématique

Q 2) **Une jolie application des séries entières via la notion de *série génératrice* :** Pour n matrices $A_1, \dots, A_n$ on appelle $P(n)$ le nombre de façon de mettre les parenthèses pour calculer "$A_1 \times \cdots \times A_n$ " Par convention on pose naturellement que $P(1) = 1$, et on a $P(2) = 1$ et que $P(3) = 2$.

a) Montrer que, pour tout $n \geq 2$: $P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$.

b) On pose $P(0) = 0$ et on suppose que la série $\sum P(n)x^n$ a un rayon de convergence R non nul. On note $S(x)$ sa somme pour $x \in]-R, R[$.
Déduire du a) une équation vérifiée par $S(x)$ et $S(x)^2$.

c) Résoudre cette équation et en déduire que la série $\sum P(n)x^n$ a effectivement un rayon de convergence non nul et une formule explicite pour ses coefficients $P(n)$.
Culturel : On note plutôt $C_n = P(n+1)$ et sauf erreur de calcul, vous aurez montré que $C_n = \dfrac{(2n)!}{(n+1)!n!}$: on appelle ce nombre le n-ième nombre de Catalan. C'est donc pour nous ici le nombre de façon de parenthéser une expression formée de $n+1$ termes.

d) Justifier qu'un algorithme qui testerait tous les choix de parenthèsage possibles n'est pas raisonnable pour n grand, avec un équivalent plus « simple » de C_n.

2.3 Sous-structures optimales dans les parenthésages optimaux

Considérons donc une suite de n matrices $(A_0, A_1, \ldots, A_{n-1})$ telle que tous les produits A_iA_{i+1} soient définis. Pour $0 \leq i < n-1$, on note ℓ_i le nombre de lignes de A_i et pose $\ell_n = c_{n-1}$, nombre de colonnes de A_{n-1}. On a donc $A_i \in \mathcal{M}_{\ell_i, \ell_{i+1}}$.

Nous voulons savoir quels parenthésages permettront d'optimiser le nombre de multiplications pour calculer le produit des A_i. Imaginons qu'un tel parenthésage optimal conduise à effectuer comme *dernière opération* le produit :

$$(A_0 \ldots A_k).(A_{k+1} \ldots A_{n-1})$$

Q 3) **a)** Estimer le nombre total de multiplications en fonction des tailles de ces matrices et de nombres de multiplications N_1 et N_2 déjà effectuées pour calculer les deux termes entre parenthèse.

> **Idée clef, à la base de la programmation dynamique :** si le nombre total de produits effectués est optimal alors les deux nombres N_1 et N_2 sont aussi optimaux .

b) Justifier l'affirmation du cartouche

c) On note $m(i,j)$ le nombre minimal de multiplications pour calculer $A_i \times \cdots \times A_j$. Si on *choisit* d'effectuer ce produit avec comme dernière étape $(A_i \times \cdots \times A_k) \times (A_{k+1} \ldots A_j)$, donner le nombre minimal de multiplications pour effectuer ce produit avec ce choix, en fonction de $m(i,k)$, $m(k+1,j)$ et des tailles des matrices.

d) En déduire une formule donnant $m(i,j)$ comme un minimum.

2.4 Implémentation impérative

On se donne une liste `L` $= [\ell_0, \ldots, \ell_n]$ d'entiers correspondant au nombre de lignes des matrices dans le produit $(A_0 \ldots A_{n-1})$ avec la convention précédente pour $\ell_n = c_{n-1}$.

On va stocker les $m(i,j)$ comme valeur d'un dictionnaire `m` de clef `(i,j)`. On va aussi utiliser un dictionnaire `K` où pour la clef `(i,j)` la valeur associée est l'entier $k = k_{i,j}$ tel que :

$$m(i,j) = m(i,k) + m(k+1,j) + \ell_i\ell_{k+1}\ell_{j+1}$$

Q 4) **a)** Le but est d'exploiter la relation du d) du § 2.3 pour fabriquer le tableau des $m(i,j)$. Pour cela , on écrit une fonction auxiliaire :

```
trouve_min(m :dict, i : int, j: int, L :list)-> (r,s) : tuple
```

où `r` contient la valeur $m(i,j)$ et `s` le premier indice entre i et $j-1$ réalisant l'égalité $m(i,j) = m(i,s) + m(s+1,j) + \ell_i\ell_{s+1}\ell_{j+1}$.
Implémenter cette fonction.

b) La fonction précédente va prendre en argument un dictionnaire `m` incomplet, qu'on va remplir au fur et à mesure justement grâce à elle. Ici, on va le faire de manière impérative, dans l'ordre suivant :

- toutes les valeurs $m_{i,i}$ pour $i = 0, \ldots, n-1$ sont nulles,
- on calcule ensuite les valeurs pour les produits de deux matrices consécutives, à savoir tous les $m_{i,i+1}$,
- on enchaîne avec les produits de trois matrices et ainsi de suite.

Implémenter cette idée en une fonction : `construit_K_m(L : list)-> K,m`

c) Pour obtenir la valeur minimale de multiplications pour le produit cherché, il suffit de renvoyer `m[(0,n-1)]`. En revanche, pour obtenir le parenthésage optimal, on se sert seulement du dictionnaire `K`. On va écrire une fonction :

`parenthesage(L)-> s : str`

qui renvoie une chaîne de caractères correspondant à un parenthésage optimal, en commençant par calculer `K` avec la fonction précédente. La chaîne de caractères sera de la forme donnée dans les exemples suivants :

```
>>>L=[2, 6, 3, 5]
>>>parenthese(L)
'(A0A1)A2'
>>> L=[2,6,3,5,4]
>>> parenthese(L)
'((A0A1)A2)A3'
>>> L=[2,6,3,5,2]
>>> parenthese(L)
'(A0A1)(A2A3)'
```

Pour programmer cette fonction, une idée possible est d'utiliser une sous-fonction récursive pour la constitution de la chaîne de caractères que nous appellerons `aux(i,j)` qui renvoie une chaîne de caractères pour la tranche de i à j et la fonction `parenthesage` réalisera l'appel principal `aux(0,n-1)`.

Pour comprendre l'écriture de `aux` il faut déjà avoir compris qu'en notant `K[(i,j)]=r`

- si `r==i` alors la chaîne de caractères codant le parenthésage optimal pour $A_i \ldots A_j$ commence par `Ai(`
- si `r>i` alors cette chaîne de caractères commencera par `(` concaténée avec l'appel récursif de `aux(i,r)` suivi d'une parenthèse.
- On distinguera de même suivant le fait que `r==j-1` ou pas pour la fin de la chaîne de caractères.

On pourra utiliser deux variables `gauche` et `droite` pour les deux parties de la chaîne de caractères renvoyée par `aux`.

N.B. Rappelons pour cette qu'en Python, les chaînes de caractères ne sont pas modifiables, donc pour fabriquer de telles chaînes par concaténations successives, on doit utiliser `c=c+...`. Noter aussi l'importance de la fonction `str(i)` qui fabrique la chaîne de caractère contenant *la valeur* de la variable `i`.