

Cours d'Algorithmique

Quatrième fiche de TD — Divide and conquer, programmation dynamique

Département Info ESIL — usage interne

2006-07

1 Divide and conquer

1.1 Conversion binaire vers décimal

La conversion classique de binaire vers décimal a une complexité en $\Theta(n)$ car elle utilise n étapes de multiplication. En utilisant le principe du diviser pour régner, nous pouvons atteindre une complexité en $\Theta(\log n)$ (Indication : utiliser la même approche que pour l'accélération de l'addition binaire vue en cours).

1.2 Multiplication de polynômes

Montrer que les coefficients de polynôme produit de $a*x+b$ et $c*x+d$ peuvent être obtenus à l'aide de 3 multiplications (Indication : l'un des produits est $(a+b)*(c+d)$). En déduire un algorithme qui multiplie deux polynômes de degré n en $\Theta(n^{\log n})$ étapes (n est supposé être une puissance de 2 afin de simplifier l'algorithme). En déduire un algorithme qui multiplie deux entiers binaires en $\Theta(n^{\log n})$ étapes de complexité constante.

2 Programmation dynamique

L'approche "diviser pour régner" conduit parfois à résoudre plusieurs fois le même sous-problème, devenant alors très inefficace. Avec l'approche de "la programmation dynamique", les solutions des sous-problèmes sont mémorisées au fur et à mesure et on a le moyen de les retrouver lorsque le sous-problème est rencontré à nouveau. Évidemment, il faut déterminer l'ordre dans lequel on va résoudre les problèmes et les structures de données qui serviront à mémoriser les solutions intermédiaires. Le développement d'un algorithme de programmation dynamique peut être planifié en suivant les quatre étapes suivantes : 1) Caractériser la structure d'une solution (optimale), 2) définir récursivement la valeur d'une solution (optimale), 3) calculer la valeur d'une solution (optimale) en remontant jusqu'à l'énoncé du problème initial et 4) construire une solution (optimale) du problème donné en se servant des informations recueillies dans l'étape précédente.

2.1 Multiplication d'une suite de matrices

On veut calculer le produit $A = A_1 \times A_2 \dots A_n$ d'une suite de matrices. Le produit étant associatif, on peut procéder de plusieurs manières. Pour $n = 4$, on a 5 façons de faire, chacune associée à l'un parenthésages suivants : $(A_1(A_2(A_3 A_4))) = (A_1((A_2 A_3)A_4)) = ((A_1 A_2)(A_3 A_4)) = ((A_1(A_2 A_3))A_4) = (((A_1 A_2)A_3)A_4)$. On se propose de déterminer une solution qui minimise le nombre d'opérations. Quel est le nombre de multiplications scalaires nécessaires effectuer le produit de deux matrices de dimensions $p \times q$ et $q \times r$? Considérez le cas où les dimensions de A_1, \dots, A_4 sont $10 \times 100, 100 \times 5, 5 \times 50, 50 \times 8$. Quel est le nombre des multiplications nécessaires pour chacun des parenthésages ci-dessus, quelle solution est optimale ? On montre que le nombre de parenthésages du produit de n matrices est le nombre de Catalan de rang n , $T(n) = \frac{1}{n+1} * \text{binomial}(2n, n) \simeq \frac{4^n}{n^{3/2}}$. Le passage en revue de tous les parenthésages possibles n'est donc pas efficace.

Étape 1 : Caractérisons la structure de la solution optimale. Supposons obtenir une solution optimale pour le produit $A = A_1 \dots A_n$ en coupant entre A_k et A_{k+1} , c'est-à-dire en calculant $B = A_1 \dots A_k$ et $C = A_{k+1} \dots A_n$ et ensuite $A = B \times C$.

Étape 2 : Donnons une définition récursive de la solution optimale. Notons $m_{i,j}$ le nombre minimum de multiplications scalaires nécessaires pour calculer le produit de matrices $A_i \dots A_j$ et supposons que chaque matrice A_k est de dimension $p_{k-1} \times p_k$. Montrons d'abord qu'une séparation arbitraire du produit $A_i \dots A_j$ entre A_q et A_{q+1} donne $m_{i,j} = m_{i,q} + m_{q+1,j} + p_{i-1} p_q p_j$. Nous en déduisons que coût minimal pour le calcul du produit $A_i \dots A_j$ vaut 0 si $i = j$ et $\min_{i \leq q \leq j} m_{i,q} + m_{q+1,j} + p_{i-1} p_q p_j$ lorsque $i < j$ et que le coût le plus faible pour calculer $A = A_1 \dots A_n$ sera $m_{1,n}$.

Étape 3 : Au lieu de calculer récursivement la solution de la récurrence ci-dessus, nous adopterons une approche ascendante en calculant successivement les coûts $m_{i,j}$ que nous rangeons dans un tableau $M[1..n, 1..n]$ tout en gardant l'indice q correspondant au minimum dans un tableau $S[1..n, 1..n]$. Écrivons la procédure `ORDONNER(p)` qui remplit les tableaux M et S et dont le paramètre p est la séquence (p_0, \dots, p_n) telle que, pour $i \in \{1, \dots, n\}$, la matrice A_i soit de dimension $p_{i-1} \times p_i$. Quel est le coût de l'algorithme obtenu ?

5) Étape 4 : Construisons une solution optimale du problème donné. A partir de l'information calculée et stockée dans les tableaux M et S , nous sommes en mesure de calculer effectivement le produit de matrices avec un coût optimal. Chaque élément $S[i, j]$ contient la valeur de q pour laquelle le parenthésage optimal de $A_i \dots A_j$ implique une séparation entre A_q et A_{q+1} . Nous savons donc que, lors du calcul de $A_1 \dots A_n$, la dernière opération se fera entre $A_1 \dots A_{S[1,n]}$ et $A_{S[1,n]+1} \dots A_n$. De même, $S[1, S[1, n]]$ détermine la dernière multiplication effectuée lors du calcul de $A_1 \dots A_{S[1,n]}$ et ainsi de suite récursivement. Écrivons la procédure `MULTIPLIER(i, j)` qui implante ce processus.

2.2 Sac à dos généralisé

Le problème du sac à dos introduit pendant le cours considérait que chaque objet pouvait être pris une fois, ou ne pas être pris. On peut généraliser le problème en associant à chaque objet i un entier strictement positif m_i . Cet entier donne le nombre maximal d'exemplaires de i que nous sommes autorisés à prendre. Donner la nouvelle formulation du problème. Est-ce que les choses changent fondamentalement par rapport au problème initial ?