TP8 MPSI: Programmation dynamique

Alexis Saurin
Alexis.Saurin@ens.fr
http://www.eleves.ens.fr/~saurin/tp_caml

11 juin 2003

1 Introduction

Nous allons nous intéresser aujourd'hui à la programmation dynamique. Il s'agit d'une méthode de conception d'algorithmes, un peu à la manière de la méthode "diviser pour régner" : le point commun avec la méthode diviser pour régner est qu'on cherche à résoudre un problème en résolvant d'abord des sous-problèmes puis en combinant leur solution. Là où la programmation dynamique diffère du diviser pour régner, c'est dans les sous-problèmes que l'on considère : avec le diviser pour régner, on a des sous problèmes qui sont indépendants (par exemple deux sous-tableaux à trier), et donc ils doivent tous les deux être traités. Dans la programmation dynamique, les sous-problèmes que l'on va considérer ne seront pas indépendants dans le sens qu'ils partageront des sous-sous-problèmes. Si l'on considérait un alogrithme diviser pour régner pour ce type de problème, on effectuerait trop de calcul.

Le principe de base est donc de ne calculer qu'une fois les sous-sous-problèmes. Pour cela, il va falloir bien identifier les sous-problèmes et leurs solutions de manière à faire le minimum de calcul possible.

On va s'intéresser ici à deux problèmes de programmation dynamique : la multiplication d'une suite de matrices et la recherche d'une plus longue sous séquence commune.

Avant de commencer, on va donner un plan général de travail pour la programmation dynamique :

1. déterminer la structure d'une solution optimale ;
2. déterminer de manière récursive la valeur d'une solution optimale
3. inverser la récursion : rendre itératif le procédé de calcul
4. construire une solution optimale.

2 Multiplication d'une suite de matrices

2.1 Le problème

Vous savez tous caluler le produit de deux matrices de tailles respectives mxn et nxp, et cela grâce à un algorithme très simple en $O(m * n * p)$. Il est donc très simple de multiplier une suite de matrices $(A_1, \ldots, A_n)$ pour obtenir le produit $A_1 A_2 \ldots A_n$.

Pour cela, il faut déteminer un parenthésage de notre suite puisque la multiplication est une opération binaire. Du point de vue mathématique, cela ne fait aucune différence puisque la multiplication matricielle est associative, mais du point de vue informatique, outre le fait qu'il soit essentiel que le résultat soit mathématiquement correct, il faut que le calcul soit faisable effectivement, et le parenthésage modifie potentiellement beaucoup le nombre de multiplications scalaires à effectuer pour calculer le produit. Considérons un exemple pour s'en convaincre : soient A_1, A_2 et A_3 des matrices de tailles respectives $10 * 100$, $100 * 5$ et $5 * 50$. Le parenthésage $((A_1 A_2) A_3)$ induit 10*100*5+10*5*50= 7500 multiplications scalaire, tandis que le parenthésage $(A_1 (A_2 A_3))$ induit le calcul de 100*5*50+10*100*50=75000 multiplications scalaires... On peut, selon le parenthésage, avoir à effectuer 10 fois plus de calculs ce qui augmente d'autant le temps de calcul !!

Il faut donc envisager de choisir, avant de se lancer dans les multiplications, un parenthésage qui épargne le plus de calcul possible. Notre but va donc être de calculer un parenthésage optimal dans le sens que le nombre de multiplications scalaires à effectuer pour arriver au résultat sera minimal.

Question 1: *Le nombre de parenthésages (à faire chez vous)* (∗∗) Ne serait-il pas possible de considérer tous les parenthésages les uns après les autres pour conserver celui qui minimise le nombre de multiplications ? Caluler le nombre de parenthésages d'une multiplication de n matrices.

Question 2: *Structure d'un parenthésage optimal*

Définir la forme d'un parenthésage optimal.

Question 3: *Solution récursive*

En déduire un algorithme récursif qui détermine le nombre de multiplications à faire avec le parenthésage optimal. Quelle est sa complexité ?

Question 4: *Calcul des coûts optimaux*

Le problème de l'algorithme récursif précédent est que les instances de base du problème sont calculées un grand nombre de fois, ce qui fait que le calcul prend très longtemps.

Décrire un algorithme itératif qui calcule la solution, on fait également calculer l'indice k auquel on fait le parenthésage le plus externe.

Question 5: *Multiplication des matrices*

Donner maintenant un algorithme utilisant les résultats du calcul précédent pour calculer la multiplication de la suite de matrice grâce à un parenthésage optimal.

3 Plus longue sous-séquence commune

On étudie maintenant un autre problème, qui l'on notera ici PLSC : étant données deux séquences X et Y (c'est-à-dire des suites finies prises dans un alphabet décidé par ailleurs, mais cela n'a que peu d'importance ici...), déterminer une sous-séquence[1] Z commune à X et Y qui soit de longueur maximale.

Par exemple, si X=(A, B, C, B, D, A, B) et Y=(B, D, C, A, B, A) sont les séquences considérées, Z=(B, C, B, A) est une solution, mais elle n'est pas forcément unique : (B, D, A, B) ou (B, C, A, B) sont d'autres PLSC...

Question 6: *Premier algorithme*

Donner un premier algorithme, inefficace, qui calculera le résultat.

Question 7: *Structure d'un solution*

Donner une caractérisation d'une PLSC de deux séquences.

Question 8: *Solution récursive au problème*

idem que la question 3.

Question 9: *Résolution itérative : calcul de la longueur d'une PLSC*

idem que la question 4. De même que pour la question 4, on calculera une information auxiliaire qui permettra de reconstruire à la fin du calcul une PLSC.

Question 10: *Construction d'une PLSC*

Écrire une fonction qui imprime une PLSC.

Question 11: *Application*

Déterminer une PLSC de (1, 0, 0, 1, 0, 1, 0, 1, 0) et (0, 1, 0, 1, 1, 0, 1, 1, 0, 0).

[1] on utilise le mot sous-séquence ici pour dire une suite de lettre apparaissant dans une séquence, dans cet ordre, mais pas nécessairement à la suite