TP9 : programmation dynamique et graphes

AP3 Algorithmique et programmation — L2 mathématiques

8 décembre 2016

Exercice 1 (Produit de matrices). On rappelle que le produit de matrices est associatif, c'est-à-dire que $(AB)C = A(BC)$. Pour faire le produit de matrices $A_1 A_2 \dots A_n$, il existe de nombreuses façons de parenthéser correctement cette expression. Le but de cet exercice est de trouver le parenthésage optimal d'un produit de matrices $A_1 A_2 \dots A_n$ minimisant le nombre de multiplications scalaires.

(a) Combien de multiplications d'entiers faut-il faire pour calculer

$$(3) \times ((4) \times ((2) \times (6 \quad 7 \quad 8 \quad 9))) \quad ?$$

Et

$$(((3) \times (4)) \times (2)) \times (6 \quad 7 \quad 8 \quad 9) \quad ?$$

(b) À partir de maintenant, on suppose avoir n matrices $A_1, \dots, A_n$ multipliables, c'est-à-dire que : A_1 a dimension $p_0 \times p_1$, A_2 a dimension $p_1 \times p_2$, etc. jusque A_n qui a dimension $p_{n-1} \times p_n$. Remarquer que les coefficients des A_i importent peu et que seul les p_i sont nécessaires pour résoudre le problème du parenthésage optimal.

(c) Montrer que le nombre de parenthésages de $A_1 A_2 \dots A_n$ est supérieur ou égal à 2^{n-2}. Est-il efficace de tester tous les parenthésages possibles ?

(d) Pour calculer le parenthésage optimal, on va utiliser la méthode de programmation dynamique.

Notons $m_{i,j}$ $(1 \leq i \leqslant j \leqslant n)$ le nombre minimal de multiplications scalaires à effectuer pour calculer $A_i \dots A_j$. Démontrer que les $m_{i,j}$ sont définies par les relations suivantes :

$$m_{i,j} = \begin{cases} 0 & \text{si } i = j \\ \min_{i \leqslant k < j} \left(m_{i,k} + m_{k+1,j} + p_{i-1} p_k p_j \right) & \text{sinon} \end{cases}$$

(e) Écrire une fonction `matrixmult(p)` prenant en paramètre la liste p des dimensions p_i, $0 \leqslant i \leqslant n$ et renvoyant le nombre minimal de multiplications scalaires à effectuer pour calculer le produit $A_1 \dots A_n$.

(f) Modifier la fonction précédente en une fonction `matrixmult_ext(p)` prenant le même paramètre et renvoyant le couple $(m_{1,n}, s)$ où $m_{1,n}$ est comme avant et s est la chaîne de caractères comportant le parenthésage des A_i. Par exemple, pour les matrices de la question 2.(a), la fonction doit retourner :

```
(6, '((((A1)A2)A3)A4)')
```

Exercice 2 (Problème du sac à dos). Le but est d'implémenter le problème du sac à dos vu en cours. Étant donnés des objets $i \in \{1, \dots, n\}$, chacun ayant une valeur v_i et un poids w_i, il s'agit de remplir un sac pouvant contenir un poids W en maximisant la valeur total dans le sac. Formellement, on cherche à calculer

$$\max \left\{ v_{i_1} + \cdots + v_{i_k} \;\middle|\; \begin{array}{l} 1 \leq k \leq n \\ w_{i_1} + \cdots + w_{i_k} \leq W \end{array} \right\}$$

On note $m_{i,w}$ la valeur maximale atteignable avec un sac de taille w et en ne regardant que les objets $1, \dots, i$.

(i) Exprimer $m_{i,w}$ en fonction de $m_{i-1,w}$, $m_{i-1,w-w_i}$ et v_i. (Indice : on pourra dissocier deux cas, celui où $w_i > w$, et celui où $w_i \leq w$.)

(ii) En déduire un algorithme récursif naïf. Quelle est sa complexité ?

(iii) On passe maintenant à la conception de l'algorithme façon *programmation dynamique*. Écrire une fonction `python` :

```
def knapsack (values, weights, n, W): #...
```

qui prend en paramètres deux listes `values` et `weights` contenant respectivement les v_i et les w_i, un entier n et un entier W, et qui renvoie une matrice de taille $(n+1) \times (W+1)$ et contenant en case (i, w) la valeur $m_{i,w}$.

(iv) Modifier maintenant la fonction précédente de façon à renvoyer une matrice contenant en case (i, w) le couple $(m_{i,w}, b)$ où b est un booléen valant `True` si l'objet i est choisi pour atteindre la valeur $m_{i,w}$ et `False` sinon.

(v) En déduire une nouvelle fonction :

```
def knapsack (values, weights, n, W): #...
```

qui renvoie la liste des objets à choisir pour atteindre $m_{n,W}$.

Exercice 3 (Détection d'un cycle dans un graphe orienté). On représente un graphe orienté $G = (S, A)$ par sa matrice d'adjacence. Autrement dit, si $S = \{0, 1, \dots, n-1\}$, le graphe G est donné par la liste de listes M telle que $M[i][j] = 1$ si $(i, j) \in A$ et $M[i][j] = 0$ sinon.

(i) Ecrire une fonction prenant en entrée un graphe orienté G et renvoyant `True` si G est acyclique et `False` sinon.

(ii) Donner la complexité de cette procédure.