Diviser pour régner

■ ■ ■

Exercice 1. (Multiplication de polynômes, Algorithme de Knuth) On représente informatiquement un polynôme $P = a_0 + \cdots + a_n X^{n-1}$ avec $a_{n-1} \neq 0$ par le vecteur `P = [|`$a_0; \ldots ; a_{n-1}$`|]`. Le polynôme nul sera représenté par le vecteur vide `[||]`.

1. a) Écrire un algorithme naïf `mult_pol` permettant de calculer le produit de deux polynômes.
`mult_pol : int array -> int array -> int array`

b) Si on multiplie un polynôme P de degré n avec un polynôme Q de degré m, quel est le nombre d'opérations (additions + multiplications) effectuées dans l'algorithme précédent ?
Quelle est la complexité d'un produit PQ lorsque $\deg P = \deg Q = n$.

2. Soient a, b, c, d des éléments d'un anneau commutatif. Montrer que l'on peut calculer $\alpha = ac$, $\beta = bd$ et $\gamma = ad + bc$ en ne faisant que trois multiplications.

3. a) On suppose que P et Q sont de même degré n. On note $m = \left\lfloor \frac{n}{2} \right\rfloor$ le quotient de la division euclidienne de n par 2. Montrer qu'il existe deux polynômes A et B, de degrés inférieurs à $m+1$, dont la recherche des coefficients ne nécessite aucun calcul tels que $P(X) = A(X) + X^m B(X)$. On posera de même $Q(X) = C(X) + X^m D(X)$.

b) Écrire un algorithme `coupe` qui, étant donné un polynôme, renvoie les polynômes A et B définis précédemment.
`coupe : int array -> int array * int array`

c) En déduire un algorithme récursif `mult_rec` calculant le produit de polynômes de mêmes degrés n.
`mult_rec : int array -> int array -> int array`
On supposera données des fonctions `add`, `oppose`, `translate` *qui permettent respectivement d'additionner deux polynômes, de renvoyer leur opposé et de les multiplier par un monôme.*

```
add : int array -> int array -> int array
oppose : int array -> int array
translate : int array -> int -> int array
```

Expliquez (sans écrire de programme) comment modifier la fonction précédente pour qu'elle puisse multiplier des polynômes P et Q de degrés éventuellement distincts.

d) En notant $T(n)$ le coût de calcul du produit PQ, montrer qu'il existe un algorithme qui permet d'aboutir à la relation de récurrence

$$T(n) = 3T(n/2) + \Theta(n).$$

4. On écrit $\Theta(n) = an$, où $a > 0$ est une constante. On est ainsi ramené à résoudre l'équation fonctionnelle

$$T(1) = 1,\ T(n) = 3T(n/2) + an,\ \forall\ n \geqslant 2.$$

a) Soit β un réel. En posant $u_k = 2^{-\beta k} T(2^k)$, écrire la relation de récurrence entre u_k et u_{k-1}.

b) Pour quelle valeur de β cette relation est-elle de la forme $u_k = u_{k-1} + K 2^{\alpha k}$?
Préciserer alors la valeur des constantes K et α en fonction de a et β.

c) En déduire l'expression de u_k en fonction de k, puis l'expression de $T(n)$ en fonction de n, lorsque $n = 2^k$ est une puissance de 2.

d) En déduire une expression de $T(n)$ lorsque $n \to \infty$ sous une forme $\Theta(\cdot)$.

e) Conclure.

Exercice 2. (Algorithme de Strassen, 1969) Les matrices seront définies en lignes et représentées par des objets de type `'a array array`

1. Évaluer la complexité d'un algorithme naïf de multiplication matricielle en nombre de multiplications scalaires, puis en nombre d'additions.

En 1969, Strassen a conçu un algorithme permettant de faire mieux asymptotiquement. Pour cela, il a remarqué que multiplier deux matrices d'ordre 2 permet de se faire à l'aide de 7 multiplications et 18 additions. En effet, si

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \text{ et } B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

alors

$$AB = \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix},$$

où

$m_1 = (a_{11} + a_{22})(b_{11} + b_{22})$	$m_5 = (a_{11} + a_{12})b_{22}$
$m_2 = (a_{21} + a_{22})b_{11}$	$m_6 = (a_{21} - a_{11})(b_{11} + b_{12})$
$m_3 = a_{11}(b_{12} - b_{22})$	$m_7 = (a_{12} - a_{22})(b_{21} + b_{22})$
$m_4 = a_{22}(b_{21} - b_{11})$	

2. Écrire les formules de récurrences satisfaites par le nombre de multiplications M_n et le nombre d'additions A_n nécessaires pour multiplier des matrices d'ordre n.

3. On suppose que $n = 2^k$ est une puissance de 2.

a) Déterminer M_n.

b) En étudiant la suite $(M_{2^k}/7^k)$, déterminer une majoration de M_n.