

Algorithmique et Programmation  
Examen

*Notes de cours autorisées à l'exception de tout autre document.*

*Le résultat d'une question peut être utilisé dans la suite du problème même si elle n'a pas été résolue.*

**Durée : 3 heures**

**Exercice 1.** Soit  $A$  un anneau commutatif unitaire.

PARTIE I : DIVISION EUCLIDIENNE RAPIDE PAR LA MÉTHODE DE NEWTON

Soient  $S, T \in A[X]$  avec  $\deg(S) = n$ ,  $\deg(T) = m$  et  $T$  unitaire.

1. Montrer que l'algorithme classique de division euclidienne de  $S$  par  $T$  a une complexité arithmétique en  $O(n^2)$ .

**Réponse :** L'algorithme naïf pour calculer  $Q$  et  $R$  tels que  $S = QT + R$  avec  $\deg R < \deg T$  consiste à poser la division. Pour cela les termes dominants sont d'abord isolés :

$$S = aX^n + R_S, T = X^m + R_T \text{ avec } \deg R_S < n \text{ et } \deg R_T < m.$$

Si  $n < m$ , il n'y a rien à faire. Sinon la bouclé élémentaire de la division de  $S$  par  $T$  consiste à tuer le terme de plus haut degré de  $S$  en effectuant la soustraction

$$S - aX^\delta T = (aX^n + R_S) - (aX^{\delta+m} + aX^\delta R_T) = R_S - aX^\delta R_T,$$

où  $\delta = m - n$ . Le polynôme obtenu est congru à  $S$  modulo  $T$ , mais de degré strictement inférieur à  $n$ . Il n'y a plus qu'à itérer ce procédé jusqu'à obtenir un polynôme de degré strictement inférieur à  $n$  et à garder trace des quotients successifs.

La complexité de cet algorithme est facile à estimer. La boucle élémentaire s'effectue en  $O(m)$  opérations dans  $A$  ; dans le pire des cas, l'algorithme effectue  $(n - m + 1)$  passages dans cette boucle, de sorte que la complexité de la division de  $S$  par  $T$  est de  $O(m(n - m))$  opérations dans  $A$ . (avec une petite constante : 2).

2. Pour  $P \in A[X]$  et  $k \geq \deg(P)$ , nous notons  $\text{Rec}_k(P(X)) = X^k P(1/X)$ . Montrer que

$$\text{Rec}_{n-m}(Q) = \text{Rec}_n(S) \text{Rec}_m(T)^{-1} \text{ mod } X^{n-m+1},$$

où  $Q$  est le quotient de la division euclidienne de  $S$  par  $T$ .

**Réponse :** Nous avons

$$S(X) = Q(X)T(X) + R(X)$$

soit

$$X^n S(1/X) = X^n (Q(1/X)T(1/X) + R(1/X))$$

d'où

$$X^n S(1/X) = [X^{n-m}Q(1/X)][X^m T(1/X)] + X^{n-m+1}[X^{m-1}R(1/X)]$$

et le résultat.

3. Soit  $F \in A[X]$  avec  $F(0) = 1$  et soit  $\ell \geq 1$ . Considérons la suite de polynômes  $G_i \in A[X]$  définie par  $G_0 = 1$  et

$$G_{i+1} = 2G_i - F \cdot G_i^2 \text{ mod } X^{2^{i+1}}$$

pour  $i \geq 0$ . Montrer que pour tout entier  $i \geq 0$ , nous avons

$$F \cdot G_i \equiv 1 \text{ mod } X^{2^i}.$$

**Réponse :**

- $i = 0$  : trivial
- $i + 1 \geq 1$  et vrai pour  $i$  ( $F \cdot G_i = 1 + P(X)X^{2^i}$ ) :

$$\begin{aligned} F \cdot G_{i+1} &\equiv F \cdot (2G_i - F \cdot G_i^2) \text{ mod } X^{2^{i+1}} \\ &\equiv 2 + 2P(X)X^{2^i} - (1 + P(X)X^{2^i})^2 \text{ mod } X^{2^{i+1}} \\ &\equiv 1 + P(X)^2 X^{2^{i+1}} \text{ mod } X^{2^{i+1}} \end{aligned}$$

4. En déduire un algorithme pour calculer  $Q$  et le reste  $R$  de la division euclidienne de  $S$  par  $T$ .

**Réponse :**  $\text{Rec}_m(T)$  vérifie les conditions de l'énoncé (puisque  $T$  est unitaire). La méthode de Newton permet de calculer son inverse modulo  $X^{n-m}$  en itérant. À chaque étape de la boucle on effectue deux multiplications des polynômes de degré au plus  $2^i$ . Donc la complexité est :

$$2M(1) + 2M(2) + 2M(4) + \dots + 2M(2^\ell)$$

où  $2^\ell > m - n$ .

5. Montrer que la complexité arithmétique de ce nouvel algorithme de division euclidienne appliqué à deux polynômes de degré  $< n$  est en  $O(M(n))$  où  $M(n)$  est la complexité arithmétique du produit de deux polynômes de degré  $< n$  de  $A[X]$  (avec  $M(n+m) \geq M(n) + M(m)$  pour  $m, n \in \mathbb{N}$ ).

## PARTIE II : ÉVALUATION RAPIDE DE POLYNÔMES

Soit  $P \in A[X]$  unitaire avec  $\deg(P) = n$ . Soient  $a_1, \dots, a_n \in A$ .

- Supposons que  $n = 2^k$  est une puissance de 2 et considérons un l'arbre binaire complet  $T$  à  $n$  feuilles défini par :
  - chacune des  $n$  feuilles est associée à un polynôme  $X - a_j$  pour  $j \in \{1, \dots, n\}$ ;
  - pour chaque nœud interne  $u$  ayant les fils  $v$  et  $w$  associés aux polynômes  $M_v(X)$  et  $M_w(X)$  respectivement,  $u$  est associé au polynôme  $M_u(X) = M_v(X) \cdot M_w(X)$ .
  - Donner un algorithme pour construire l'arbre  $T$  avec une complexité arithmétique en  $O(M(n) \log n)$ .
  - Donner un algorithme qui prenant en entrée  $P$ ,  $(a_1, \dots, a_n)$  et  $T$ , calcule  $P(X) \bmod M_u(X)$  pour tout  $u \in T$ , avec une complexité arithmétique en  $O(M(n) \log n)$ .
- Déduire des questions précédentes un algorithme qui calcule  $P(a_1), \dots, P(a_n)$  pour *tout*  $n \in \mathbb{N}$  avec une complexité arithmétique en  $O(M(n) \log n)$ .

### PARTIE III : INTERPOLATION DE LAGRANGE RAPIDE

Soient  $a_1, \dots, a_n \in A$  deux à deux distincts et soient  $A_1, \dots, A_n \in A$ .

- Montrer qu'il existe une unique suite de polynômes  $\Delta_k(X) \in A[X]$ , avec  $k \in \{1, \dots, n\}$ , de degré  $n - 1$  telle que

$$P(X) := \sum_{k=1}^n A_k \cdot \Delta_k(X)$$

vérifie  $P(a_i) = A_i$  pour tout  $i \in \{1, \dots, n\}$ .

- Soit  $d_k$  le coefficient dominant de  $\Delta_k$  pour  $k \in \{1, \dots, n\}$  et soit  $M(X) = \prod_{i=1}^n (X - a_i)$ . En appliquant les résultats de la PARTIE II à  $M'(X)$ , donner un algorithme pour calculer  $d_1, \dots, d_n$  avec une complexité arithmétique en  $O(M(n) \log n)$ .
- Donner un algorithme qui prenant en entrée  $(a_1, \dots, a_n)$  et  $(A_1, \dots, A_n)$ , calcule  $P(X)$  avec une complexité arithmétique en  $O(M(n) \log n)$ .

**Exercice 2.** L'algorithme de Strassen de multiplication de matrices ne fonctionne que sur des anneaux et non sur le *quasi-anneau* des booléens ( $\{0, 1\}, \vee, \wedge, 0, 1$ ) puisque 1 n'a pas d'inverse pour  $\vee$ . Nous nous proposons de trouver des algorithmes de calcul du produit de matrices booléennes.

### PARTIE I : MULTIPLICATION DES QUATRE RUSSES

Soient  $A$  et  $B$  deux matrices booléennes  $n \times n$ . Nous supposons que  $\log_2(n)$  divise  $n$  et nous partitionnons  $A$  en sous-matrices  $A_i$  (avec  $1 \leq i \leq n/\log_2(n)$ ) de taille  $n \times \log_2(n)$  et  $B$  en sous-matrices  $B_j$  (avec  $1 \leq j \leq n/\log_2(n)$ ) de taille  $\log_2(n) \times n$ .

- Montrer que nous pouvons écrire  $AB = \sum_{i=1}^{n/\log_2(n)} A_i B_i$  où chaque produit  $A_i B_i$  est une matrice  $n \times n$ .
- Montrer que l'algorithme naturel pour calculer chaque matrice  $A_i B_i$  demande  $O(n^2 \log_2(n))$  opérations.

3. Proposer un algorithme calculant chaque matrice  $A_i B_i$  en  $O(n^2)$  opérations. et donc le produit de deux matrices booléennes en  $O(n^3 / \log_2(n))$ .

## PARTIE II : MULTIPLICATION HEURISTIQUE DE SHAMIR

Dans cette partie, l'idée (due à Shamir) est de travailler sur un anneau où on peut appliquer l'algorithme de Strassen et de relier les résultats pour qu'ils s'appliquent sur l'anneau des booléens. Pour ce faire, il propose de travailler avec l'anneau  $R = (\{0, 1\}, \oplus, \wedge, 0, 1)$  où  $\oplus$  représente le *ou exclusif*.

Soient  $A = (a_{i,j})$  et  $B$  deux matrices booléennes  $n \times n$  et soit  $C = (c_{i,j})$  leur produit dans le quasi-anneau des booléens. À partir de  $A$ , nous fabriquons une matrice  $A' = (a'_{i,j})$  en utilisant la procédure probabiliste suivante :

- si  $a_{i,j} = 0$ , alors  $a'_{i,j} = 0$  ;
  - si  $a_{i,j} = 1$ , alors  $a'_{i,j}$  est tiré uniformément aléatoirement dans  $\{0, 1\}$ . Les choix aléatoires pour chaque entrée étant indépendants.
1. Soit  $C' = (c'_{i,j})$  le produit  $A'B$  dans l'anneau  $R$ . Montrer que si  $c_{i,j} = 0$ , alors  $c'_{i,j} = 0$  et que si  $c_{i,j} = 1$ , alors  $c'_{i,j} = 1$  avec probabilité  $1/2$ .
  2. Montrer que pour tout  $\varepsilon > 0$ , la probabilité qu'un  $c'_{i,j}$  donné ne prenne jamais la valeur  $c_{i,j}$  est au plus de  $\varepsilon/n^2$  pour  $\log_2(n^2/\varepsilon)$  choix indépendants de la matrice  $A'$ . Montrer que la probabilité que tous les  $c'_{i,j}$  prennent au moins une fois la valeur correcte est minorée par  $1 - \varepsilon$ .
  3. Donner un algorithme probabiliste en  $O(M(n) \log(n))$  qui calcule le produit de deux matrices  $n \times n$  dans le quasi-anneau booléen avec une probabilité d'au moins  $1 - n^{-k}$  pour une constante  $k > 0$  quelconque. Les seules opérations autorisées sur les composantes des matrices sont  $\vee$ ,  $\wedge$  et  $\oplus$ .

**Exercice 3.** Soit  $\mathbb{G}$  un groupe dans lequel il est algorithmiquement possible de calculer le produit de deux éléments, de tester si deux éléments sont égaux et de déterminer si un élément est égal à 1 l'élément neutre de  $\mathbb{G}$  et soit  $g$  un élément de  $\mathbb{G}$  d'ordre fini  $n > 2$ .

1. Rappeler l'algorithme *pas-de-bébé pas-de-géant* de Shanks qui prend en entrée  $g$  et un majorant  $B$  de  $n$ , retourne  $n$  en  $2 \cdot \sqrt{B} + O(1)$  opérations de groupe.

Le but de cet exercice est de présenter un algorithme de complexité  $O(\sqrt{n})$  sans connaissance *a priori* d'un majorant de  $n$ .

2. Soit  $v \in \mathbb{N}$ ,  $v \neq 0$ . Considérons la suite  $(t_j)_{j \in \mathbb{N}}$  définie par récurrence :

$$t_0 = 2v \text{ et } t_{j+1} = t_j + j + v + 1 \text{ pour } j \geq 0.$$

Donner un algorithme qui prenant en entrée  $v$  et  $j \in \mathbb{N}$ , retourne  $g^i$  pour tout entier  $i \in \{1, \dots, j + v - 1\} \cup \{t_1, \dots, t_j\}$  en  $2j + v + 1$  opérations de groupe.

3. En remarquant que tout entier  $m \in [t_{j-1}, t_j]$  peut s'écrire comme une différence  $t_j - i$  avec  $0 \leq i < j + v$ , donner un algorithme (paramétré par  $v$ ) qui calcule  $n$  en  $O(\sqrt{n})$  opérations. On donnera une estimation précise du nombre d'opérations dans  $\mathbb{G}$  en fonction de  $v$  et le choix optimal pour  $v$ .