COURS 2

Multiplication rapide

Résumé

Le produit de polynômes et d'entiers est une opération élémentaire, qui intervient dans un nombre impressionnant d'algorithmes du calcul formel. L'efficacité de ces algorithmes repose donc sur celle du produit. Pour multiplier deux polynômes de degré n à coefficients dans un anneau A, la méthode classique requiert $O(n^2)$ opérations dans A. De même, l'algorithme scolaire de multiplication de deux entiers à n chiffres nécessite un nombre d'opérations binaires en $O(n^2)$. Nous présentons dans ce cours plusieurs algorithmes de multiplication rapide, dont celui de Karatsuba, de complexité $O(n^{1,59})$, ainsi que ceux utilisant la transformée de Fourier rapide, dont la complexité est essentiellement linéaire en n.

1. Introduction, résultats principaux

Les problèmes abordés dans ce cours concernent la complexité arithmétique de la multiplication des polynômes à une variable et la complexité binaire de la multiplication des entiers. Au vu de l'exemple suivant, il est facile de se convaincre de la similitude des deux questions :

Polynômes : Soient à multiplier $3X^2 + 2X + 1$ et $6X^2 + 5X + 4$ dans $\mathbb{Z}[X]$.

$$(3X^2 + 2X + 1) \times (6X^2 + 5X + 4)$$

$$= (3 \cdot 6)X^4 + (3 \cdot 5 + 2 \cdot 6)X^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)X^2 + (2 \cdot 4 + 1 \cdot 5)X + (1 \cdot 4)$$

$$= 18X^4 + 27X^3 + 28X^2 + 13X + 4.$$

Nombres entiers : Soient à multiplier 321 et 654 en base 10.

$$(3 \cdot 10^2 + 2 \cdot 10 + 1) \times (6 \cdot 10^2 + 5 \cdot 10 + 4)$$

$$= (3 \cdot 6)10^4 + (3 \cdot 5 + 2 \cdot 6)10^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)10^2 + (2 \cdot 4 + 1 \cdot 5)10 + (1 \cdot 4)$$

$$= 18 \cdot 10^4 + 27 \cdot 10^3 + 28 \cdot 10^2 + 13 \cdot 10 + 4$$

$$= 2 \cdot 10^5 + 9 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10 + 4 = 209934.$$

Dans les deux cas, nous avons retranscrit l'algorithme naïf, et la suite des calculs est essentiellement la même, si ce n'est que, dans le cas des entiers, il faut en outre gérer les retenues (dernière égalité de l'exemple). On ne sera donc pas surpris que les résultats obtenus dans les deux cas soient très semblables.

Résultats. Dans toute la suite $(A, +, \times)$ désignera un anneau (commutatif et unitaire). Tout d'abord, nous considérons le cas arithmétique ; il s'agit de minimiser le coût, en termes du nombre d'opérations $(+, -, \times)$ dans A, du produit des polynômes en degré borné. Les premiers résultats à retenir de ce cours sont les suivants.

La multiplication des polynômes de degré au plus n dans $A[X]$ requiert :
- $O(n^2)$ *opérations dans A par* l'algorithme naïf *;*
- $O(n^{1,59})$ *opérations dans A par* l'algorithme de Karatsuba *;*
- $O(n \log n \log\log n)$*, voire dans certains cas $O(n \log n)$ opérations dans A, via* la transformée de Fourier rapide (FFT).

Ainsi, la multiplication des polynômes peut se faire en un coût arithmétique *essentiellement linéaire* en leur degré.

Cette diversité d'algorithmes motive l'introduction de la notion de *fonctions de multiplication* (notées usuellement $\mathsf{M}(n)$), qui estiment le nombre d'opérations suffisantes pour multiplier des polynômes : une application $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ est une fonction de multiplication si on peut multiplier les polynômes de degré au plus n en au plus $\mathsf{M}(n)$ opérations (à quelques détails techniques près ; ces fonctions sont définies plus précisément plus loin). De la sorte, le coût de la multiplication devient un mètre étalon pour mesurer le coût d'autres algorithmes.

La multiplication des polynômes est omniprésente : les algorithmes de calcul de pgcd (plus grand commun diviseur), de pgcd étendu, de factorisation en une ou plusieurs variables, de composition des séries formelles, d'évaluation multipoint, d'interpolation, font tous intervenir des produits de polynômes, et à ce titre, leur complexité s'énonce naturellement en termes de fonctions de multiplication M.

L'analogie entre les entiers et les polynômes va très loin ; la plupart des réponses apportées dans le cadre de la complexité arithmétique trouvent un équivalent en complexité binaire. Cependant, aucun théorème d'équivalence n'est connu ; il se trouve que les mêmes idées algorithmiques s'adaptent plus ou moins facilement dans les deux cadres. Ainsi, on dispose des résultats suivants dans le modèle binaire.

On peut multiplier des entiers de n chiffres binaires par :
- l'algorithme naïf *en $O(n^2)$ opérations binaires ;*
- l'algorithme de Karatsuba *en $O(n^{1,59})$ opérations binaires ;*
- l'algorithme de Schönhage-Strassen *en $O(n \log n \log\log n)$ opérations binaires.*

Les preuves de ces résultats de complexité binaire sont plus délicates que celles de leurs analogues polynomiaux, à cause des problèmes de gestion des retenues. Ainsi, dans la suite, nous ne traitons d'abord en détail que les versions polynomiales de ces résultats, le cas entier étant ensuite brièvement passé en revue.

En pratique. Les constantes cachées dans les $O(\cdot)$ sont déterminantes pour l'efficacité pratique de tels algorithmes. Parlons tout d'abord du cas polynômial, par exemple lorsque A est un corps fini de taille « raisonnable » (typiquement, dont les éléments sont représentés sur quelques mots machine). Dans les meilleures implantations actuelles (magma, NTL) :

- l'algorithme de Karatsuba bat l'algorithme naïf pour des degrés d'environ 20 ;
- les méthodes à base de FFT en $O(n \log n)$ gagnent pour des degrés de l'ordre de 100, mais ne peuvent pas être utilisées pour des degrés arbitrairement grands (vient un moment où on manque de racines de l'unité, voir plus loin) ;

- l'algorithme de type FFT en $O(n \log n \log\log n)$ est utilisé pour des degrés de l'ordre de quelques dizaines ou centaines de milliers.

Certains problèmes, en cryptologie ou en théorie des nombres, nécessitent de manipuler des polynômes de degré de l'ordre de 100 000, tailles auxquelles les algorithmes rapides sont indispensables.

L'implantation des algorithmes rapides pour les entiers est délicate en raison des retenues. Dans les meilleures implantations actuelles (magma, GMP) :

- l'algorithme de Karatsuba bat l'algorithme naïf pour des nombres de l'ordre de 100 chiffres binaires ;
- les méthodes à base de FFT (Schönhage-Strassen) gagnent pour des nombres d'environ 10 000 chiffres binaires.

À nouveau, des problèmes venus de cryptologie ou de théorie des nombres demandent de manipuler des nombres de taille colossale (de l'ordre de 100 000 000 de chiffres ; il faut 10 Mo pour stocker un tel nombre). Ceci justifie amplement les efforts d'implantation d'algorithmes rapides.

Fixons les notations : on travaille avec des polynômes F et G à coefficients dans un anneau A, ayant un degré au plus $n-1$, formellement

$$F = f_0 + \cdots + f_{n-1}X^{n-1} \quad \text{et} \quad G = g_0 + \cdots + g_{n-1}X^{n-1} \ ;$$

le problème est alors de calculer (les coefficients de)

$$H = FG = h_0 + \cdots + h_{2n-2}X^{2n-2}.$$

2. Algorithme naïf

Cet algorithme consiste à développer le produit, c'est-à-dire à écrire

$$H = FG = \sum_{i=0}^{2n-2} h_i X^i \quad \text{avec} \quad h_i = \sum_{j+k=i} f_j g_k.$$

Ainsi, calculer tous les h_i demande $O(n^2)$ opérations dans A. C'est un algorithme de complexité arithmétique *quadratique*.

Exercice 1. Montrer que, pour multiplier deux polynômes de degrés m et n, l'algorithme naïf demande au plus $(m+1)\times(n+1)$ multiplications dans A et mn additions dans A.

Exercice 2. Montrer que, pour multiplier deux entiers à n chiffres chacun, l'algorithme naïf demande $O(n^2)$ opérations binaires.

Exercice 3. Estimer la complexité binaire de la méthode naïve lorsque les polynômes ont degré n et des coefficients entiers bornés par H..

Exercice 4. Shigeru Kondo a calculé 25 000 000 000 décimales de π sur un pc de bureau[1]. Ce type de calcul repose sur la multiplication d'entiers. En supposant que la machine de Kondo était capable d'effectuer 10^{12} opérations à la seconde, montrer que Kondo n'a pas utilisé l'algorithme naïf.

[1]Calcul achevé le 7 mars 2003 sur un Pentium 4 3,2 Gz, 2 Go, après 17 jours et 14 heures, avec 100 Go d'espace disque utilisé. Le record actuel sur un super-ordinateur est de $1{,}2 \times 10^{12}$ chiffres, obtenu par Kanada et son équipe en 2002.

3. Algorithme de Karatsuba

Un premier raffinement de l'algorithme naïf se base sur la remarque suivante : il est possible de gagner *une* multiplication pour le produit des polynômes de degré 1. Soient en effet à multiplier les polynômes

$$F = f_0 + f_1 X \quad \text{et} \quad G = g_0 + g_1 X.$$

Le produit $H = FG$ s'écrit

$$H = f_0 g_0 + (f_0 g_1 + f_1 g_0)X + f_1 g_1 X^2.$$

Effectuer tous les 4 produits $f_0 g_0, f_0 g_1, f_1 g_0, f_1 g_1$ correspond à l'algorithme quadratique. Mais on peut faire mieux en remarquant que le coefficient de X s'écrit

$$f_0 g_1 + f_1 g_0 = (f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1.$$

Cette écriture mène à un algorithme qui effectue au total 3 multiplications et 4 additions. On a perdu quelques additions par rapport à l'algorithme naïf, mais le gain d'une multiplication va se transformer en gain dans l'*exposant* de l'algorithme, par application récursive.

Passons en effet au cas général des degrés quelconques. Inspirés par l'observation précédente, on va scinder F et G en deux. On suppose donc que F et G sont de degré au plus $n-1$, et que l'entier n est pair, $n = 2k$. On pose alors

$$F = F^{(0)} + F^{(1)} X^k, \quad G = G^{(0)} + G^{(1)} X^k,$$

$F^{(0)}, F^{(1)}, G^{(0)}, G^{(1)}$ ayant des degrés au plus $k-1$. Le produit $H = FG$ s'écrit

$$H = F^{(0)}G^{(0)} + (F^{(0)}G^{(1)} + F^{(1)}G^{(0)})X^k + F^{(1)}G^{(1)}X^{2k}.$$

Pour écrire l'algorithme, on suppose que n est une puissance de 2 afin de pouvoir effectuer tous les appels récursifs que l'on souhaite. On obtient alors, avec les notations ci-dessus :

Algorithme de Karatsuba

Entrée : F, G de degré au plus $n-1$, n étant une puissance de 2.

Sortie : $H = FG$.

1. Si $n = 1$, renvoyer FG.
2. Calculer $A_1 = F^{(0)}G^{(0)}$ et $A_2 = F^{(1)}G^{(1)}$ récursivement.
3. Calculer $A_3 = F^{(0)} + F^{(1)}$ et $A_4 = G^{(0)} + G^{(1)}$.
4. Calculer $A_5 = A_3 A_4$ récursivement.
5. Calculer $A_6 = A_5 - A_1$ et $A_7 = A_6 - A_2$.
6. Renvoyer $A_1 + A_7 X^{n/2} + A_2 X^n$.

On peut maintenant établir la complexité de cet algorithme.

THÉORÈME 1. *Si n est une puissance de 2, l'algorithme de Karatsuba calcule le produit de deux polynômes de degreé au plus $n-1$ en $9n^{\log_2 3}$ opérations dans A.*

DÉMONSTRATION. Lors de l'appel en degré $< n$, on effectue 3 appels récursifs en degré $< n/2$, et quelques additions. Le coût $K(n)$ satisfait donc à la récurrence :

$$K(n) \leq 3K(n/2) + 4n,$$

où le terme $4n$ vient estimer le nombre d'additions. Le lemme « diviser pour régner » permet alors de conclure. □

On en déduit le résultat général suivant en degré quelconque.

COROLLAIRE 1. *On peut multiplier les polynômes de degré n (quelconque) en $O(n^{\log_2 3}) = O(n^{1,59})$ opérations dans A.*

En effet, soit N la plus petite puissance de 2 telle que $N > n$. Alors pour calculer le produit de F et de G il suffit de multiplier $\tilde{F} = X^{N-\deg(F)-1}F$ et $\tilde{G} = X^{N-\deg(G)-1}G$ via l'algorithme vu précédemment ; on perd au pire un facteur constant par rapport à l'estimation du Théorème 1.

EXERCICE 5. Estimer la constante cachée dans l'estimation asymptotique du Corollaire 1.

Observons que pour obtenir une bonne implantation en degré quelconque, la solution précédente (basée sur l'insertion artificielle de coefficients nuls) n'est pas forcément la meilleure, car elle induit la perte d'un facteur constant. Une solution alternative est d'adapter le découpage vu plus haut au cas où l'un des polynômes est de degré pair.

EXERCICE 6. Déterminer la complexité d'une variante de l'algorithme de Karatsuba où tout polynôme de degré impair $n = 2k-1$ est découpé en deux tranches de degré au plus $k-1$ et tout polynôme de degré pair $n = 2k$ est découpé en deux tranches de degré au plus k.

Par ailleurs, en fonction de la nature de l'anneau ou du corps de base, on peut vouloir arrêter les appels récursifs avant d'avoir atteint le degré 0. Ce choix dépend des vitesses relatives de l'addition et de la multiplication.

EXERCICE 7. Soit n une puissance de 2. Établir un algorithme hybride, qui fait appel à l'algorithme de Karatsuba pour $n > 2^d$ et à l'algorithme naïf pour $n \leq 2^d$. Montrer que la complexité arithmétique $\mathsf{C}(n)$ de cet algorithme vérifie $\mathsf{C}(n) \leq \gamma(d)n^{\log_2 3} - 8n$ pour tout $n \geq 2^d$, où $\gamma(d)$ est une fonction qui dépend uniquement de d. Trouver la valeur de d qui minimize $\gamma(d)$ et comparer le résultat avec celui du Théorème 1.

EXERCICE 8. Pensez-vous qu'il soit possible de multiplier deux polynômes de degré au plus 1 en utilisant seulement 2 multiplications dans l'anneau de base ?

EXERCICE 9. Soit A un anneau et soit P et Q deux polynômes de degré au plus 4 dans $A[X]$.

1. Estimer le nombre de multiplications de A requises par l'algorithme de Karatsuba pour calculer le produit AB ;
2. Donner un algorithme qui multiplie P et Q en utilisant au plus 7 multiplications dans A ;
3. En déduire un algorithme de multiplication polynomiale dans $A[X]$ de complexité arithmétique $O(n^{1,4})$;
4. Montrer que, pour tout entier $\alpha \geq 2$, il existe un algorithme de multiplication polynomiale dans $A[X]$ de complexité arithmétique $O(n^{\log_\alpha(2\alpha-1)})$;
5. Montrer que pour tout $\varepsilon > 0$, il existe un algorithme de multiplication polynomiale dans $A[X]$ de complexité arithmétique $O(n^{1+\varepsilon})$, où la constante dans le $O(\cdot)$ dépend de ε, mais pas de n.

4. Transformée de Fourier rapide

Les méthodes à base de transformée de Fourier sont ce que l'on sait faire de mieux pour multiplier les polynômes. Pour simplifier la présentation, on suppose ici que l'on cherche à multiplier des polynômes F et G dans $A[X]$, de degrés strictement inférieurs à $n/2$ (ou plus généralement tels que $\deg FG < n$).

On fait (provisoirement) l'hypothèse que l'anneau A est de cardinal au moins n. On se donne des points distincts $a_0, \ldots, a_{n-1}$ dans A. Le principe de la multiplication par transformation de Fourier est le suivant :

1. *Évaluation.* On calcule les valeurs :
$$\mathsf{Ev}(F) = (F(a_0), \ldots, F(a_{n-1}))\,; \quad \mathsf{Ev}(G) = (G(a_0), \ldots, G(a_{n-1}))\,.$$
2. *Produit point à point.*
$$\mathsf{Ev}(F), \mathsf{Ev}(G) \mapsto \mathsf{Ev}(FG) = (FG(a_0), \ldots, FG(a_{n-1}))\,.$$
3. *Interpolation.*
$$\mathsf{Ev}(FG) \mapsto FG.$$
Cette opération revient à retrouver les coefficients de FG à partir de ses valeurs en $a_0, \ldots, a_{n-1}$ (à supposer que cette opération d'interpolation soit possible, voir ci-dessous).

En termes matriciels, l'opération $F \mapsto \mathsf{Ev}(F)$ est linéaire, et sa matrice (pour des polynômes F de degré au plus $n-1$, dans la base monomiale $\{1, X, \ldots, X^{n-1}\}$) est la matrice de Vandermonde
$$V_{a_0,\ldots,a_{n-1}} = \begin{bmatrix} 1 & a_0 & \cdots & a_0^{n-1} \\ \vdots & & & \vdots \\ 1 & a_{n-1} & \cdots & a_{n-1}^{n-1} \end{bmatrix}.$$
Pour l'instant nous n'avons pas réellement progressé ; il reste la question importante de trouver des jeux de valeurs $a_0, \ldots, a_{n-1}$ pour lesquels les opérations d'évaluation et d'interpolation sont possibles (*i. e.* pour lesquels $V_{a_0,\ldots,a_{n-1}}$ est inversible), et réalisables rapidement. Pour ce faire, nous allons choisir des *racines de l'unité*.

On dit que $\omega \in A$ est une racine n-ième de l'unité si $\omega^n = 1$; ω est une racine n-ième *primitive* de l'unité si en plus $\omega^t - 1$ est non diviseur de zéro dans A pour tout diviseur strict t de n (c'est-à-dire que $\alpha(\omega^t - 1) = 0$ implique $\alpha = 0$).

Remarquons que si A est un corps, cette dernière condition revient simplement à dire que w^t est différent de 1 pour tout diviseur strict t de n. Par exemple, dans $\mathbb{C}$, -1 n'est pas une racine 4-ième primitive de l'unité, et i l'est.

À un tel ω, on associe la suite des points d'évaluation $1, \omega, \ldots, \omega^{n-1}$, et la matrice de Vandermonde associée sera notée V_ω. Un premier intérêt de ce choix apparaît dans le théorème suivant.

Théorème 2. *Soit $\omega \in A$ une racine n-ième primitive de l'unité. Alors $\omega^{-1} = \omega^{n-1}$ est également une racine n-ième primitive de l'unité, et $V_{\omega^{-1}} V_\omega = nI_n$.*

Exercice 10. Prouver le théorème précédent.

Autrement dit, l'interpolation sur une racine primitive de l'unité se ramène à une évaluation sur une racine primitive (dite *conjuguée*). Ceci explique qu'on se consacre donc maintenant à ce dernier problème uniquement, que l'on appelle traditionnellement DFT (Discrete Fourier Transform).

Supposons que n est pair, $n = 2k$; on peut alors regrouper les puissances de ω en puissances paires et impaires, d'une part $1, \omega^2, (\omega^2)^2, \ldots, (\omega^2)^{k-1}$ et d'autre part $\omega, \omega\cdot\omega^2, \omega\cdot(\omega^2)^2, \ldots, \omega\cdot(\omega^2)^{k-1}$. On va utiliser ce groupement pour mettre en place un algorithme de type « diviser pour régner ».

Pour appliquer cette idée, écrivons les divisions euclidiennes

$$F = Q_0(X^k - 1) + R_0 \quad \text{et} \quad F = Q_1(X^k + 1) + R_1,$$

avec $\deg R_0 < k$ et $\deg R_1 < k$. Les racines de $X^k - 1$ sont précisément les puissances paires de ω, et les racines de $X^k + 1$ les puissances impaires. Pour prouver ce point, il suffit d'utiliser l'hypothèse que $\omega^k - 1$ n'est pas un diviseur de zéro. Soit ensuite $\bar{R}_1(X) = R_1(\omega X)$. On a alors les égalités :

$$F(\omega^{2\ell}) = R_0((\omega^2)^\ell) \quad \text{et} \quad F(\omega^{2\ell+1}) = R_1(\omega^{2\ell+1}) = \bar{R}_1((\omega^2)^\ell).$$

Il convient de remarquer que R_0 et R_1 s'obtiennent très facilement à partir de F : pour $0 \leq \ell < k$, le coefficient en X^ℓ de R_0 est la somme des coefficients en X^k et $X^{k+\ell}$ de F ; le coefficient en X^ℓ de R_1 est la différence des coefficients en X^k et $X^{k+\ell}$ de F. Ensuite, passer de R_1 à $\bar{R}_1$ s'effectue en multipliant pour chaque ℓ le coefficient de X^ℓ de R_1 par ω^ℓ.

On dispose maintenant de tous les éléments pour écrire l'algorithme de DFT. Pour mettre en place les idées ci-dessus de manière récursive, il est plus simple de supposer que n est une puissance de 2 ; on obtient alors l'algorithme suivant :

Transformée de Fourier Discrète (DFT_ω)

Entrée : $F = f_0 + \cdots + f_{n-1}X^{n-1}$, et les puissances $1, \omega, \cdots, \omega^{n-1}$ d'une racine n-ième primitive de l'unité, n étant une puissance de 2.

Sortie : $F(1), \ldots, F(\omega^{n-1})$.

1. Si $n = 1$, renvoyer f_0.
2. Calculer R_0, R_1 et $\bar{R}_1$.
3. Soit $k = n/2$. Calculer récursivement $R_0(1), R_0(\omega^2), \ldots, R_0((\omega^2)^{k-1})$ et $\bar{R}_1(1), \bar{R}_1(\omega^2), \ldots, \bar{R}_1((\omega^2)^{k-1})$.
4. Renvoyer $R_0(1), \bar{R}_1(1), R_0(\omega^2), \bar{R}_1(\omega^2), \ldots, R_0((\omega^2)^{k-1}), \bar{R}_1((\omega^2)^{k-1})$.

La complexité de cet algorithme fait l'objet du théorème suivant.

THÉORÈME 3. *L'algorithme* DFT_ω *ci-dessus requiert au plus* $\frac{3n}{2}\log n$ *opérations dans* A.

DÉMONSTRATION. Supposant connues les puissances de ω, le coût de l'appel en degré n est d'au plus $2 \times n/2$ additions et soustractions (pour le calcul de R_0 et R_1), et $n/2$ multiplications (pour le calcul de $\bar{R}_1$), plus 2 appels récursifs en degré $n/2$. Sa complexité $T(n)$ satisfait donc à la récurrence :

$$T(n) \leq \frac{3n}{2} + 2T\left(\frac{n}{2}\right)$$

et le lemme « diviser pour régner » permet de conclure. □

EXERCICE 11. Montrer que l'algorithme DFT_ω ci-dessus requiert $n\log n$ additions dans A et $\frac{1}{2}n\log n$ multiplications d'éléments de A par des puissances de ω.

L'algorithme de transformée de Fourier discrète permet de multiplier les polynômes à coefficients dans A, pourvu que A contienne « suffisamment » de racines primitives de l'unité, d'ordre « suffisamment » élevé.

De manière explicite, si A contient une racine de l'unité ω d'ordre $n = 2^k$, on obtient l'algorithme suivant. Cette algorithme requiert l'inversibilité de n dans A, il ne fonctionne donc que sous l'hypothèse supplémentaire que 2 est inversible dans A.

Multiplication par Transformée de Fourier Rapide (FFT)

Entrée : F et G de degrés au plus $n/2-1$ et ω, une racine primitive n-ième de l'unité, n étant une puissance de 2.

Sortie : FG.

1. Calculer les n premières puissances de ω (qui donnent également les puissances de ω^{-1}, à permutation près).
2. Calculer $\mathsf{DFT}_\omega(F)$ et $\mathsf{DFT}_\omega(G)$.
3. Effectuer les multiplications de ces valeurs point-à-point ; on définit $(h_0, \ldots, h_{n-1})$ le vecteur résultat et H le polynôme $h_0 + \cdots + h_{n-1}X^{n-1}$.
4. Calculer $(k_0, \ldots, k_{n-1}) = \mathsf{DFT}_{\omega^{-1}}(H)$.
5. Renvoyer le polynôme $\frac{k_0}{n} + \cdots + \frac{k_{n-1}}{n}X^{n-1}$.

Remarquons que, en pratique, on manipule uniquement des tableaux d'éléments de A. La présentation du pseudo-code ci-dessus, qui fait explicitement intervenir le polynôme H, est trompeuse de ce point de vue. Par ailleurs, on a tout intérêt à stocker les valeurs des puissances de ω d'un appel à l'autre.

La complexité de cet algorithme est de 3 DFT en degré n, soit $\frac{9}{2}n\log n$ opérations, plus $O(n)$ divisions par n et $O(n)$ multiplications pour calculer les puissances de ω. Le coût pour la multiplication de polynômes de degré au plus $n-1$ s'en déduit aisément.

COROLLAIRE 2. *Soit $n = 2^k$, et supposons que 2 est inversible dans A et qu'il existe dans A une racine $2n$-ième primitive de l'unité. Cette racine étant connue, on peut multiplier les polynômes de degré au plus $n-1$ en $9n\log n + O(n)$ opérations dans A.*

EXERCICE 12. Montrer que sous les hypothèses du corollaire précédent, on peut multiplier les polynômes de degré au plus $n-1$ en utilisant $6n\log n + O(n)$ additions dans A, $3n\log n + O(n)$ multiplications par des puissances de ω, $2n$ multiplications arbitraires dans A et $2n$ divisions par $2n$.

EXERCICE 13. Soit n dans $\mathbb{N}$, soit n_0 la plus petite puissance de 2 supérieure ou égale à n, et supposons qu'il existe dans A une racine $2n_0$-ième primitive de l'unité. Cette racine étant connue, on peut multiplier les polynômes de degré au plus $n-1$ en $18n\log n + O(n)$ opérations dans A.

EXERCICE 14. Soit $n = 2^k$, et supposons qu'on dispose d'une racine n-ième primitive de l'unité $\omega \in A$. Soit P et Q deux polynômes dans $A[X]$ de degré au plus $n-1$. Supposons que les coefficients de $X^0, X^1, \ldots, X^{n-1}$ du produit $R = PQ$ sont connus. Montrer que R peut être calculé en $\frac{9}{2}n\log n + O(n)$ opérations dans A.

Même dans le cas favorable où A est un corps, il n'y existe pas nécessairement toutes les racines primitives de l'unité que l'on souhaite. Dans le cas particulier des corps finis, on sait donner une réponse précise à cette question d'existence.

THÉORÈME 4. *Soient $\mathbb{F}_q$ le corps fini à q éléments et $n \in \mathbb{N}$. Le corps $\mathbb{F}_q$ contient une racine n-ième primitive de l'unité si et seulement si n divise $q-1$.*

EXERCICE 15.
1. Prouver le théorème précédent.
2. Montrer que si n divise $q-1$ et si α est un élément primitif de $\mathbb{F}_q$ (*i. e.* tel que α engendre le group multiplicatif $(\mathbb{F}_q \setminus \{0\}, \times)$) alors $\alpha^{(p-1)/n}$ est une racine n-ième primitive de l'unité.

Ce résultat mène à la notion de *premier de Fourier*, qui sont les nombres premiers p tels que $p-1$ soit divisible par une grande puissance de 2, c'est-à-dire des nombres premiers de la forme $\ell 2^k + 1$, avec k « sufisamment grand ». Par exemple,

$$4179340454199820289 = 29 \times 2^{57} + 1$$

est un tel nombre premier. Ainsi, dans $\mathbb{Z}/4179340454199820289\,\mathbb{Z}$, on dispose de racines primitives 2^{57}-ièmes de l'unité (21 en est une) ; on peut donc y multiplier des polynômes de degrés colossaux par l'algorithme en $O(n \log n)$.

Nous donnons dans la Figure 1 la courbe de complexité pratique[2] de la multiplication polynomiale à coefficients dans le corps fini $A = \mathbb{Z}/4179340454199820289\,\mathbb{Z}$.

L'allure de cette courbe confirme que les estimations théoriques de complexité sont respectées en pratique. Le comportement quasi-linéaire *par morceaux*, agrémenté de sauts entre les puissances successives de 2, est typique pour les implantations actuelles de la FFT.

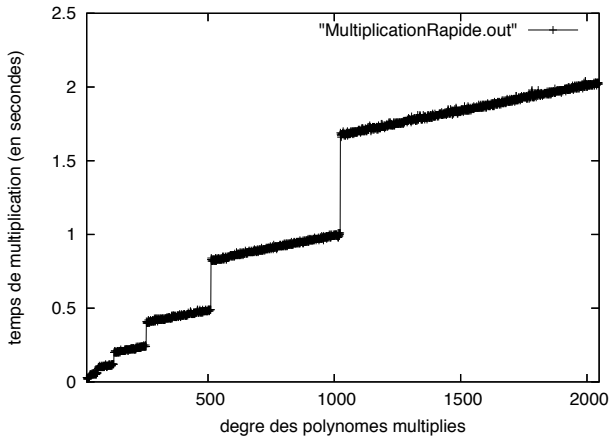

FIG. 1. Courbe de complexité pratique de la multiplication dans $A[X]$, où $A = \mathbb{Z}/4179340454199820289\,\mathbb{Z}$.

[2]Calculs effectués avec le logiciel Magma, version V2.12-1, sur un Opteron 150 (2,4 GHz)

5. L'algorithme de Schönhage et Strassen

Quand les racines de l'unité font défaut, il reste possible de faire fonctionner les idées à base de transformée de Fourier. Ceci est réalisé par l'algorithme de Schönhage et Strassen, qui fait l'objet de cette section. Cet algorithme s'applique quel que soit l'anneau de base, pourvu que 2 y soit inversible ; l'idée est de rajouter les racines de l'unité qui manquent en étendant l'anneau de base de manière judicieuse.

Soient F et G des polynômes de degré strictement inférieur à $n/2$. L'algorithme a en fait pour vocation de calculer le produit FG modulo X^n+1 ; ce n'est pas une limitation ici, car le produit a un degré inférieur à n, et cela a l'avantage d'assurer la cohérence dans les appels récursifs.

La première idée est de réécrire F et G comme des polynômes $\bar{F}$ et $\bar{G}$ en deux variables X, Y, de degré strictement inférieur à $d = \sqrt{n}$ en chacune des deux variables. Par exemple, si $n = 9$, on voit le polynôme $a_0 + a_1X + \cdots + a_8X^8$ comme $(a_0+a_1X+a_2X^2)+(a_3+a_4X+a_5X^2)Y+(a_6+a_7X+a_8X^2)Y^2$. Savoir multiplier les polynômes dans cette représentation est suffisant, puisque $FG = (\bar{F}\bar{G})(X, X^d)$, et cette dernière évaluation se fait en complexité linéaire. Ici, la restriction de calculer FG modulo $X^n + 1$ se traduit par le fait qu'on calcule $\bar{F}\bar{G}$ modulo $Y^d + 1$.

Ensuite, on pose $B = A[X]/(X^{2d}+1)$ et on va multiplier $\bar{F}$ et $\bar{G}$ dans $B[Y]$: cela permet de retrouver $\bar{F}\bar{G} \mod (Y^d + 1)$ dans $A[X, Y]$ puisque tous les coefficients du produit ont un degré en X strictement plus petit que $2d$.

Dans B, $\omega = X^2$ est une racine primitive de l'unité d'ordre $2d$. On peut donc appliquer l'algorithme de FFT afin de multiplier des polynômes de degré inférieurs strictement à d dans $B[Y]$ en $O(d \log d)$ opérations dans B. En regardant de près l'algorithme précédent, on voit que celui-ci nécessite :

- $O(d \log d)$ opérations $(+, -)$ dans B (chacune demande $O(d)$ opérations dans A) ;
- $O(d)$ divisions par d dans B (chacune demande $O(d)$ opérations dans A) ;
- $O(d \log d)$ multiplications par une puissance de ω dans B (chacune demande $O(d)$ opérations dans A, car elle revient simplement à décaler les indices et éventuellement changer un signe) ;
- d produits dans B, qui sont gérés par des appels récursifs.

On en déduit que la coût $T(n)$ obéit à la récurrence :

$$T(n) \leq Cn \log n + \sqrt{n}T(2\sqrt{n}),$$

où C est une constante universelle indépendante de n, bien sûr, mais aussi de A.

Exercice 16. Montrer que si α, β, γ sont des constantes positives telles que $T(n) \leq \alpha n \log n + \beta\sqrt{n}T(\gamma\sqrt{n})$, pour tout $n \geq 2$, alors

1. $T(n) \in O(n \log n)$, si $\beta\gamma < 2$;
2. $T(n) \in O(n \log n \log\log n)$, si $\beta\gamma = 2$;
3. $T(n) \in O(n \log^{\beta\gamma} n)$, si $\beta\gamma > 2$.

En admettant le résultat de l'exercice prédédent, nous pouvons donc établir le résultat suivant.

Théorème 5. *Soit A un anneau dans lequel 2 est inversible (d'inverse connu). On peut multiplier des polynômes de $A[X]$ de degré au plus n en $O(n \log n \log\log n)$ opérations $(+, -, \times)$ dans A.*

Il est possible d'étendre cette idée au cas où 3 est inversible (FFT triadique), et plus généralement à un anneau quelconque. On obtient alors l'algorithme de complexité $O(n \log n \log\log n)$ mentionné dans l'introduction.

6. Algorithmes pour les entiers

Les algorithmes ainsi que les résultats présentés ci-dessus s'étendent à la multiplication des entiers[3]. Nous allons brièvement présenter cette problématique à travers un exemple. Soient à multiplier les entiers 2087271 et 1721967, qu'on suppose donnés en base 2,

$$A = 11111110100101100111 \quad \text{et} \quad B = 11010010000110011101111,$$

chacun ayant $D = 21$ chiffres binaires. On peut ramener leur multiplication à un produit de polynômes. Plus exactement, on associe à A et B les polynômes de degré strictement inférieur à D :

$$P = X^{20}+X^{19}+X^{18}+X^{17}+X^{16}+X^{15}+X^{14}+X^{12}+X^{11}+X^8+X^6+X^5+X^2+X+1$$

et

$$Q = X^{20} + X^{19} + X^{17} + X^{14} + X^{10} + X^9 + X^6 + X^5 + X^3 + X^2 + X + 1.$$

La stratégie est la suivante : on calcule $R = PQ$ dans $\mathbb{Z}[X]$, et ensuite on évalue R en $X = 2$. Pour multiplier P et Q dans $\mathbb{Z}[X]$, il suffit d'effectuer leur produit dans $A[X] = \mathbb{Z}/p\mathbb{Z}[X]$, où p est un nombre premier tel que $2D > p > D$. Par le Théorème 5, cette multiplication polynomiale en degré D peut se faire en $O(D \log D \log\log D)$ opérations $(+, -, \times)$ dans $A = \mathbb{Z}/p\mathbb{Z}$.

Puisque chaque opération de A nécessite au plus $O(\log^2 D)$ opérations binaires, il s'ensuit que le coût binaire du calcul de R est de $O(D \log^3 D \log\log D)$. Ici $p = 23$ et R (écrit en base 2) vaut $R = X^{40} + 10X^{39} + 10X^{38} + 11X^{37} + 11X^{36} + 11X^{35} + 100X^{34} + 11X^{33} + 11X^{32} + 100X^{31} + 11X^{30} + 100X^{29} + 101X^{28} + 11X^{27} + 101X^{26} + 1000X^{25} + 101X^{24} + 101X^{23} + 1000X^{22} + 1001X^{21} + 1010X^{20} + 1000X^{19} + 111X^{18} + 1000X^{17} + 110X^{16} + 110X^{15} + 110X^{14} + 11X^{13} + 100X^{12} + 110X^{11} + 100X^{10} + 11X^9 + 100X^8 + 100X^7 + 100X^6 + 11X^5 + 10X^4 + 11X^3 + 11X^2 + 10X + 1$. Enfin, l'évaluation de R en 2 revient à gérer les retenues et cela peut se faire en un coût négligéable. Ici $AB = R(2) = 11010001001101011110110111011010110110101001$, ou encore, en écriture décimale $AB = 3594211782057$.

Une légère amélioration est possible si l'on choisit pour p un premier de Fermat supérieur à $2D$. Dans notre exemple, on peut prendre $p = 193$. En effet, il existe dans $A = \mathbb{Z}/193\,\mathbb{Z}$ une racine primitive $\omega = 125$, d'ordre 64, donc supérieur à $2D = 40$. Ce choix mène à un algorithme de complexité binaire $O(D \log^3 D)$.

Une idée alternative est de calculer PQ dans $\mathbb{Z}[X]$ en faisant ce calcul dans $\mathbb{C}[X]$. En estimant les erreurs numériques, on peut montrer qu'il suffit de calculer en précision fixe $O(\log^2 D)$. La complexité binaire est donc toujours $O(D \log^3 D)$ via cette approche.

On peut faire un peu mieux, en remplaçant la base 2 par une base B telle que $B \log B$ soit de l'ordre de $\log D$ et en appliquant l'un des raisonnements précédents ; on peut aboutir ainsi à un algorithme de complexité binaire $O(D \log^2 D)$. Qui plus est, en appelant récursivement l'algorithme pour multiplier les petits entiers, on peut descendre cette complexité à $O(D \log D \log\log D \log\log\log D \ldots)$.

[3]Historiquement, les algorithmes pour la multiplication des entiers ont été introduits *avant* leurs homologues polynomiaux, alors que ces derniers sont souvent bien plus simples à énoncer.

Le record est cependant detenu par la version entière, à base de FFT dans des anneaux de type $\mathbb{Z}/(2^{2^\kappa}+1)\mathbb{Z}$, de l'algorithme de Schönhage-Strassen, dont nous nous contentons de mentionner l'existence.

THÉORÈME 6. *On peut multiplier deux entiers de D chiffres binaires en utilisant $O(D \log D \log\log D)$ opérations binaires.*

7. Un concept important : les fonctions de multiplication

Un bon nombre des résultats de complexité donnés dans la suite du cours reposent sur la notion de *fonction de multiplication*. Une fonction $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ sera dite fonction de multiplication (pour un anneau A) si :

- On peut multiplier les polynômes de $A[X]$ de degré au plus n en au plus $\mathsf{M}(n)$ opérations dans A ;
- M vérifie l'inégalité $\mathsf{M}(n+n') \geq \mathsf{M}(n) + \mathsf{M}(n')$.

Ainsi, on sait d'après ce qui précède que des fonctions de la forme $n^{\log_2 3}$ ou $n \log n$ sont (à des constantes près) des fonctions de multiplication (respectivement pour tous les anneaux, ou ceux qui permettent la transformée de Fourier). L'intérêt de cette notion est qu'elle permet d'énoncer des résultats de complexité indépendants du choix de l'algorithme utilisé pour multiplier les polynômes (même si parfois cela mène à des estimations légèrement trop pessimistes).

La seconde condition est utilisée pour écarter l'hypothèse d'une fonction qui croît trop lentement (si M est constante, elle ne vérifie pas cette condition ...). Concrètement, elle permettra d'établir que dans des algorithmes du type « inversion de série formelle par l'opérateur de Newton », le coût total est essentiellement celui de la dernière étape.

De la même manière, on est amenés à introduire la notion de *fonction de multiplication* pour les entiers. Une fonction $\mathsf{M}_\mathbb{Z} : \mathbb{N} \to \mathbb{N}$ est une fonction de multiplication pour les entiers si :

- On peut multiplier des entiers de n chiffres en $\mathsf{M}_\mathbb{Z}(n)$ opérations binaires.
- $\mathsf{M}_\mathbb{Z}$ vérifie l'inégalité $\mathsf{M}_\mathbb{Z}(n+n') \geq \mathsf{M}_\mathbb{Z}(n) + \mathsf{M}_\mathbb{Z}(n')$.

Ce concept est très proche de son analogue polynomial, et les contextes d'utilisation sont souvent les mêmes : on utilise la seconde condition pour contrôler les coûts de multiplication lors de l'analyse d'algorithmes récursifs.

EXERCICE 17. Soit A un anneau, soit $a \in A$ et soit P un polynôme de $A[X]$, de degré au plus n. On se propose de calculer le polynôme $Q(X) = P(X+a)$.

1. Donner un algorithme pour le calcul de Q et estimer sa complexité en termes de n ;
2. Montrer qu'il est possible de calculer $Q(X)$ en utilisant seulement $O(\mathsf{M}(n)\log n)$ opérations $(+,-,\times)$ dans A.

EXERCICE 18. Soit A un anneau, soit $\kappa, n \in \mathbb{N}$ et soit P un polynôme de $A[X]$, de degré au plus n.

1. Donner un algorithme pour le calcul de $P(X)^\kappa$ et estimer sa complexité en fonction de κ et de n ;
2. Montrer qu'il est possible de calculer $P(X)^\kappa$ en utilisant seulement $O(\mathsf{M}(n\kappa))$ opérations $(+,-,\times)$ dans A.

Notes

Le mathématicien russe A. N. Kolmogorov avait conjecturé au début des années 1960 qu'il serait impossible de multiplier deux entiers de n chiffres en un coût binaire sous-quadratique. En 1962 cette conjecture fut infirmée par Karatsuba et Ofman [9]. Une généralisation de l'algorithme de [9] a été proposée quelques années plus tard par Toom [14] et Cook [3]. L'algorithme de Toom-Cook a une complexité binaire de $O(n\,32^{\sqrt{\log n}})$ pour multiplier deux entiers de taille binaire n ; ce résultat peut être importé dans le monde polynomial (voir Exercice 9 pour une version plus faible).

L'algorithme DFT a une longue histoire, voir par exemple [6, 8]. Il s'agit d'un progrès algorithmique très fameux : J. Dongarra et F. Sullivan [7] le placent parmi les dix algorithmes qui ont marqué le plus le développement des sciences de l'ingénieur du 20$^{\text{e}}$ siècle. L'article fondateur de Cooley et Tukey [5] est sans doute l'un des plus cités en informatique[4]. L'article [1] constitue une bonne référence pour le lecteur désireux de se familiariser avec la myriade de techniques de type FFT.

Une avancée importante a été la découverte de Schönhage et Strassen [13] du résultat équivalent pour la multiplication des nombres entiers. Pendant longtemps, on a cru que cet algorithme ne pourrait présenter qu'un intérêt purement théorique. À ce jour, la plupart des logiciels généralistes de calcul formel disposent d'une multiplication rapide d'entiers : Maple et Mathematica utilisent la bibliothèque GMP, Magma dispose de sa propre implantation.

L'analogue polynomial de l'algorithme de Schönhage-Strassen traité en Section 5 a été suggéré dans [11]. Le cas particulier de la caractéristique 2 est traité dans l'article [12]. Plus récemment, Cantor et Kaltofen [2] ont donné un algorithme qui multiplie des polynômes de degré n sur une algèbre A (non nécessairement commutative, ni associative) en $O(n \log n \log\log n)$ opérations dans A.

Un problème ouvert est de trouver un algorithme de multiplication polynomiale en complexité $O(n)$, ou de prouver qu'un tel algorithme n'existe pas. Morgenstern a donné une réponse partielle, en montrant que dans un modèle simplifié où $A = \mathbb{C}$ et où les seules opérations admises sont les additions et les multiplications par des nombres complexes de module inférieur à 1, au moins $\frac{1}{2}n \log n$ opérations sont *nécessaires* pour calculer une DFT en taille n. Concernant les entiers, Cook et Aanderaa [4] ont prouvé une borne inférieure de la forme $cn \log n/(\log\log n)^2$ pour la multiplication en taille binaire n sur une machine de Turing.

Une autre question importante est de savoir si les n coefficients du *produit court* $FG \bmod X^n$ de deux polynômes F et G de degré au plus n peuvent être calculés plus efficacement que l'ensemble des $2n$ coefficients du produit FG. L'article [10] apporte une réponse positive au cas où l'algorithme de multiplication est celui de Karatsuba. Par contre, on ne connaît aucun élément de réponse dans le cas où la multiplication polynomiale repose sur la DFT.

Une faiblesse de l'algorithme DFT est son comportement quasi-linéaire *par morceaux* (voir la Figure 1), dû aux sauts de complexité entre deux puissances successives de 2. Récemment, van der Hoeven [15] a donné un nouvel algorithme, appelé TFT (de *Truncated Fourier Transform* en anglais), qui coïncide avec la DFT si n est une puissance de 2 et ayant une courbe de complexité « plus lisse » que la DFT.

[4] Plus de 2000 citations, d'après la base bibliographique Science Citation Index (SCI ®).

Un nombre premier p de la forme $2^e k+1$ est appelé nombre premier de Fourier d'exposant e. Ces premiers sont utiles pour la FFT sur des corps finis, voir Ex. 15. On peut montrer qu'il y a approximativement $(x/\log(x))/2^{e-1}$ premiers de Fourier d'exposant e inférieurs à x. Il s'ensuit qu'il y a au environ 130 corps finis de la forme $k = \mathbb{F}_p$ tel que p a la taille d'un mot machine (32 bits) et tel que dans k on puisse multiplier par FFT des polynômes de degré de l'ordre d'un million. Les racines de l'unité de $k = \mathbb{F}_p$ peuvent être calculées à partir des éléments primitifs, cf. Ex. 15. On peut montrer que si on choisit au hasard un élément de $\mathbb{F}_p$, la probabilité qu'il soit primitif est égale à $6/\pi^2$, soit plus de 0,6.

Bibliographie

[1] Bernstein (D. J.). – Multidigit multiplication for mathematicians. – Preprint, available from http://cr.yp.to/papers.html.

[2] Cantor (D. G.) and Kaltofen (E.). – On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, vol. 28, n° 7, 1991, pp. 693–701.

[3] Cook (S. A.). – *On the minimum computation time of functions*. – PhD thesis, Harvard, 1966.

[4] Cook (S. A.) and Aanderaa (S. O.). – On the minimum computation time of functions. *Transactions of the American Mathematical Society*, vol. 142, 1969, pp. 291–314.

[5] Cooley (James W.) and Tukey (John W.). – An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, vol. 19, 1965, pp. 297–301.

[6] Cooley (James W.) and Tukey (John W.). – On the origin and publication of the FFT paper. *Current Contents*, vol. 33, n° 51–52, 1993, pp. 8–9.

[7] Dongarra (J.) and Sullivan (F.). – Top Ten Algorithms. *Computing in Science & Engineering*, vol. 2, n° 1, 2000.

[8] Duhamel (P.) and Vetterli (M.). – Fast Fourier transforms : a tutorial review and a state of the art. *Signal Processing. An Interdisciplinary Journal*, vol. 19, n° 4, 1990, pp. 259–299.

[9] Karatsuba (A.) and Offman (Y.). – Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, vol. 7, 1963, pp. 595–596.

[10] Mulders (Thom). – On short multiplications and divisions. *Applicable Algebra in Engineering, Communication and Computing*, vol. 11, n° 1, 2000, pp. 69–88.

[11] Nussbaumer (Henri J.). – Fast polynomial transform algorithms for digital convolution. *Institute of Electrical and Electronics Engineers. Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n° 2, 1980, pp. 205–215.

[12] Schönhage (A.). – Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Numerische Mathematik*, vol. 20, 1973, pp. 409–417.

[13] Schönhage (A.) and Strassen (V.). – Schnelle Multiplikation großer Zahlen. *Computing*, vol. 7, 1971, pp. 281–292.

[14] Toom (A. L.). – The complexity of a scheme of functional elements simulating the multiplication of integers. *Doklady Akademii Nauk SSSR*, vol. 150, 1963, pp. 496–498.

[15] van der Hoeven (Joris). – The truncated Fourier transform and applications. In *ISSAC 2004*, pp. 290–296. – ACM, New York, 2004.