Multiplication sous-quadratique

1. Diviser pour régner

On rappelle tout d'abord le lemme permettant d'estimer la complexité d'un algorithme récursif, de type « diviser pour régner » :

Lemme 1.1. *On fixe $a \geqslant 1$, $b > 1$ et $c \in \mathbb{R}$. Soit $f : \mathbb{R}^+ \to \mathbb{R}^+$ une fonction vérifiant, pour tout $x > x_0 \geqslant 1$ suffisamment grand, l'inégalité*

$$f(x) \leqslant af(x/b) + cx, \tag{1}$$

et $f(x) = 1$ pour $x \leqslant x_0$. Alors, il existe une constante effective $C(a, b, c, x_0)$ telle que pour tout $x \geqslant x_0$, on ait

$$f(x) \leqslant C(a, b, c, x_0) \begin{cases} x^{\log_b a} & \text{si } a > b, \\ x \log x & \text{si } a = b, \\ x & \text{si } a < b. \end{cases}$$

Preuve. Soit $x \geqslant x_0$, et soit $\ell := \lceil \log_b(x/x_0) \rceil$ le plus petit entier tel que $x/b^\ell \leqslant x_0$. En itérant l'inégalité (1), on obtient

$$\begin{aligned} f(x) &\leqslant af(x/b) + cx \leqslant a\big(af(x/b^2) + cx/b\big) + cx \\ &\leqslant \cdots \leqslant a^\ell f(x/b^\ell) + cx \sum_{i=0}^{\ell-1} (a/b)^i. \end{aligned}$$

Par le choix de ℓ, on a $f(x/b^\ell) = 1$; de plus, $\ell \leqslant \log_b x + 1$, soit pour tout $y \geqslant 1$,

$$y^l \leqslant y \times y^{\log_b x} = yx^{\log_b y}.$$

On majore donc a^ℓ par $ax^{\log_b a}$; la majoration de la somme géométrique dépend de la valeur de $r = a/b$:

- si $r > 1$, elle vaut $(r^\ell - 1)/(r - 1)$, et $r^l \leqslant rx^{\log_b r} = O(x^{\log_b a - 1})$;
- si $r = 1$, elle vaut $\ell \leqslant 1 + \log_b x$;
- si $r < 1$, elle vaut $(r^\ell - 1)/(r - 1) = O(1)$.

□

Il faut interpréter f comme une fonction de coût, fonction d'une taille x ; l'inégalité (1) est typique d'une fonction récursive coupant un problème de taille initiale x en a sous-problèmes de taille b fois plus faible, et utilisant un temps linéaire pour recombiner l'information ainsi obtenue.

L'exemple le plus simple est le tri-fusion : si n est une puissance de 2, pour trier un tableau d'au plus n entiers, on trie récursivement la première et la seconde moitié du tableau, avant de recombiner en temps linéaire les deux sous-tableaux triés. D'après le lemme (avec $a = b = 2$), cet algorithme utilise $O(n \log n)$ comparaisons.

On va maintenant appliquer ce principe à la multiplication des polynômes.

2. Karatsuba

Algorithme 2.1 (Multiplication de Karatsuba)
Entrée : $a, b \in R[X]$ de degrés $< n$, où R est un anneau commutatif, et n est une puissance de 2.
Sortie : $ab \in R[X]$.

(1) Si $n = 1$, renvoyer $ab \in R$.
(2) Soit $a = A_1X^{n/2} + A_0$ et $b = B_1X^{n/2} + B_0$, où $A_0, A_1, B_0, B_1 \in R[x]$ de degrés $< n/2$.
(3) Calculer $A_0 \times B_0$, $A_1 \times B_1$ et $(A_0 + A_1) \times (B_0 + B_1)$ en utilisant 3 appels récursifs.
(4) Renvoyer $A_1B_1X^n + ((A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1)X^{n/2} + A_0B_0$.

Puisque l'addition de deux polynômes de degré $\leqslant n$ utilise $O(n)$ opérations dans R, on peut appliquer le Lemme 1.1, avec $a = 3$ et $b = 2$; donc cet algorithme utilise $O(n^{\log_2 3})$ opérations dans R pour calculer le produit. Un net progrès par rapport à l'algorithme naïf en $O(n^2)$.

La condition « n est une puissance de 2 » n'est pas gênante, puisque on peut toujours remplacer n par $2^{\lceil \log_2 n \rceil}$ la puissance de 2 immédiatement supérieure, ce qui se traduit au pire par l'apparition dans la complexité d'un facteur $2^{\log_2 3} = 3$, une constante.

La formule de Karatsuba n'a rien de magique ; avec les mêmes notations, on aurait tout aussi bien pu calculer

$$A_1B_1X^n + (- (A_0 - A_1)(B_0 - B_1) + A_0B_0 + A_1B_1)X^{n/2} + A_0B_0.$$

Plus généralement, si on décide d'utiliser le produit des coefficients dominants $C(\infty) = A_1B_1$ (qu'on peut interpréter comme une évaluation à l'infini), et deux autres points d'interpolation $u \neq v$ tels que $u - v$ soit inversible, on a la « jolie » formule

$$C(\infty)X^n + \left(\frac{1}{u-v}C(u) - \frac{1}{u-v}C(v) - (u+v)C(\infty)\right)X^{n/2} + \left(-\frac{v}{u-v}C(u) + \frac{u}{u-v}C(v) + uvC(\infty)\right),$$

où $C(u) = (A_1u + A_0)(B_1u + B_0)$ et $C(v) = (A_1v + A_0)(B_1v + B_0)$. Ceci résulte du calcul de (C_0, C_1, C_2) dans l'identité $C(Y) = C_2Y^2 + C_1Y + C_0 = (A_1Y + A_0)(B_1Y + B_0)$, à partir de $C(\infty)$, $C(u)$ et $C(v)$, par identification ou par interpolation.

Pour u, v fixés, ces formules n'utilisent que

- 3 multiplications de polynômes de degré $\leqslant n/2$,
- $O(1)$ multiplications de polynômes de degré $\leqslant n$ par une constante, en $O(n)$ opérations dans R.
- $O(1)$ additions de polynômes de degré $\leqslant n$, en $O(n)$ opérations dans R.

Les algorithmes à la Karatsuba utilisant ces formules nécessitent donc tous $O(n^{\log_2 3})$ opérations.

3. Toom-Cook

Pour ne pas être embêté par les dénominateurs, on suppose dorénavant que R est un corps : tout élément non nul est donc inversible. En particulier, la condition « $u - v$ inversible » du paragraphe précédent est automatiquement vérifiée pour $u \neq v$.

Cette méthode se généralise sans difficulté en découpant, pour $n = r^k$ une puissance de $r \geqslant 2$, les polynômes a, b de degrés $< n$ en r blocs de taille n/r :

$$a = \sum_{0 \leqslant i < r} A_i X^{(n/r)i}, \quad b = \sum_{0 \leqslant i < r} B_i X^{(n/r)i},$$

où les $A_i, B_i \in R[X]$ sont de degré $< n/r$. On écrit

$$ab = c = \sum_{0 \leqslant i < 2r-1} C_i X^{(n/r)i},$$

où les $C_i \in R[X]$ sont de degré $< n/r$. Dans $R[X][Y]$, le polynôme

$$C(Y) = \sum_{0 \leqslant i < 2r-1} C_i Y^i = \left(\sum_{0 \leqslant i < r} A_i Y^i \right) \left(\sum_{0 \leqslant i < r} B_i Y^i \right) = A(Y)B(Y)$$

est de degré en Y inférieur ou égal à $2(r-1)$. Si on choisit $2r-1$ points d'évaluations distincts dans R – ce qui suppose que R est assez gros –, $(u_0, \ldots, u_{2r-2})$, on peut calculer $C(u_i) = A(u_i)B(u_i)$ pour tout i en utilisant

- $O(r)$ multiplications de polynômes de degré $< n/r$ par une constante ;
- $O(r)$ additions de polynômes de degré $< n/r$;
- $2r-1$ multiplications de polynômes de degré $< n/r$;

les 2 premiers points prennent en compte le calcul des polynômes $A(u_i)$ et $B(u_i)$ dans $R[X]$.

Par interpolation, les coefficients C_i sont déterminés de façon unique en fonction des $C(u_i)$, par une formule qui est linéaire à coefficients dans R. Explicitement, soit

$$Q(Y) = \prod_{0 \leqslant i < 2r-1} (Y - u_i), \quad \text{et} \quad L_i(Y) = \frac{Q(Y)}{Q'(u_i)(Y - u_i)}, \quad i = 0, \ldots, 2r-2,$$

les projecteurs de Lagrange, tous dans $R[Y]$; alors

$$C(Y) = \sum_{0 \leqslant i < 2r-1} C(u_i) L_i(Y) = \sum_{0 \leqslant i < 2r-1} C_i Y^i.$$

Pour des points d'évaluation fixés, on peut développer les $L_i(Y)$ dans cette formule pour obtenir une expression explicite des C_i.

Toujours d'après le Lemme, avec $a = 2r-1$ et $b = r$, le nombre d'opérations dans R utilisées pour calculer le produit ab par cette méthode est $O_r(n^{\log_r(2r-1)})$. Pour $r = 2$, on retrouve bien le résultat de Karatsuba.

Comme $\lim_{r \to +\infty} \log_r(2r-1) = 1$, on peut rendre l'exposant aussi proche de 1 que l'on veut. En travaillant plus précisément, on peut ecrire la dépendance explicite des constantes en r ; mais la complexité combinatoire des formules obtenues pour les C_i devient vite trop pénalisante. En pratique, seuls les cas $r = 2$ ou 3 sont utilisés.

4. FFT

Pour aller plus loin, on choisit des points d'évaluations astucieux, ce qui évite de précalculer d'horribles formules d'interpolation : on se contente de les générer implicitement lors de l'exécution de l'algorithme.

Pour multiplier deux polynômes de degré $< n/2$, où $n = 2^k$ est une puissance de 2, on fait l'hypothèse que R est un corps contenant une racine primitive n-ème de l'unité ω. La transformée de Fourier discrète

$$\begin{aligned}\mathcal{F}(\cdot,\omega) : R[X]/(X^n-1) &\to R^n \\ T &\mapsto \left(T(\omega^i)\right)_{0\leqslant i<n}\end{aligned}$$

établit un isomorphisme de R-algèbres, qui permet de calculer le produit ab comme $\mathcal{F}^{-1}(\mathcal{F}(a)\cdot\mathcal{F}(b))$. Bien sûr, si $a, b \in R[X]$ sont tels que $\deg ab < n$, il revient au même d'obtenir le représentant normalisé de la classe de ab dans $R[X]/(X^n-1)$, ou directement le produit ab.

Sous l'hypothèse que $\omega \in R$, il existe un algorithme simple utilisant $O(n \log n)$ opérations dans R pour calculer $\mathcal{F}$:

Algorithme 4.1
Entrée : $T \in R[X]$, $\deg T < n = 2^k$, ω d'ordre n dans R.
Sortie : $\mathcal{F}(T, \omega)$.

(1) Si $n < 1$, retourner le vecteur (T).
(2) Décomposer T en somme de ses parties paires et impaires $T(X) = T_0(X^2) + XT_1(X^2)$.
(3) Calculer $(a_0, \ldots, a_{n/2-1}) = \mathcal{F}(T_0, \omega^2)$, par un appel récursif.
(4) Calculer $(b_0, \ldots, b_{n/2-1}) = \mathcal{F}(T_1, \omega^2)$.
(5) Retourner $(a_i + \omega^i b_i)_{i<n}$.[*On étend a_i et b_i par périodicité : $a_{n/2+i} := a_i$.*]

La décomposition $T = T_0(X^2) + XT_1(X^2)$ n'utilise pas d'opérations dans R, c'est un jeu d'écriture sur les coefficients. La dernière étape utilise $O(n)$ opérations dans R, si on calcule les $\omega^i = \omega^{i-1} \cdot \omega$ successivement. (Alternativement, le vecteur $(1, \omega, \ldots, \omega^{n-1})$ peut être précalculé.) On peut donc appliquer le Lemme avec $a = b = 2$.

Calculer $\mathcal{F}^{-1}$ se fait avec la même complexité puisque on démontre que cette transformée inverse s'interprète comme une transformée directe pour un autre choix de ω ; explicitement, on vérifie que $\mathcal{F}(\mathcal{F}(T,\omega),\omega^{-1}) = nT$ pour tout $T \in R[X]/(X^n-1)$.

En rassemblant tous ces éléments, on multiplie deux polynômes de $R[X]$ de degré $< n$ en $O(n \log n)$ opérations dans R, sous l'hypothèse très restrictive de la présence de racines de l'unité d'ordre arbitrairement élevé dans R.

La méthode de Schönhage-Strassen permet de s'en affranchir, en remplaçant R par une extension convenable ; de même, il n'est pas nécessaire de supposer que R est corps. On démontre ainsi :

Théorème 4.2 (Cantor-Kaltofen)**.** *Si R est un anneau commutatif, on peut multiplier deux polynômes de degré $< n$ dans $R[X]$ en utilisant $O(n \log n \log\log n)$ opérations $(+, \times)$ dans R.*