COMMENT FAIRE LE PRODUIT DE DEUX DÉCIMAUX LE PLUS RAPIDEMENT POSSIBLE ?

ALEX6OKO

Abstract. Nous allons réduire au maximium le nombre d'opérations élementaire pour effectuer le produit de deux décimaux.

Introduction

En primaire, on nous a appris que pour multiplier deux nombres décimaux (disons 32 et 58) nous devons procéder ainsi:

$$\underbrace{\begin{array}{r} 32 \\ \times\underline{58} \end{array}}_{\text{Opération initiale}} \Rightarrow \underbrace{\begin{array}{l} 8\times 2 = 16 \Rightarrow \text{on pose 6, retient 1} \\ 8\times 3 = 24+1 = 25 \Rightarrow \text{Ligne 1: } 256 \end{array}}_{\text{Multiplication par les unités (8)}} \Rightarrow \underbrace{\begin{array}{l} 5\times 2 = 10 \Rightarrow \text{on pose 0, retient 1} \\ 5\times 3 = 15+1 = 16 \Rightarrow \text{Ligne 2: } 1600 \end{array}}_{\text{Multiplication par les dizaines (5)}}$$

$$\Rightarrow \underbrace{\begin{array}{r} 256 \\ +1600 \\ \hline 1856 \end{array}}_{\text{Addition finale}}$$

Ici on a fait 7 opérations élémentaires. Mais pourquoi ces opérations aboutissent effectivement au résultat. Et puis, après avoir compris cela, on peut se demander si l'on peut avoir un procédé encore plus rapide.

1. Convolution discrète de deux suites à support fini

Soient $x = (x_n)_{n\in\mathbb{Z}}$ et $h = (h_n)_{n\in\mathbb{Z}}$ deux suites à support fini, c'est-à-dire :

$$\exists N_1, N_2 \in \mathbb{Z}, \quad x_n = 0 \text{ pour } n \notin [N_1, N_2]$$
$$\exists M_1, M_2 \in \mathbb{Z}, \quad h_n = 0 \text{ pour } n \notin [M_1, M_2]$$

La **convolution discrète** de x et h, notée $(x * h)_n$, est définie pour tout $n \in \mathbb{Z}$ par :

$$(x * h)_n = \sum_{k\in\mathbb{Z}} x_k \cdot h_{n-k}$$

Comme x et h ont un support fini, la somme est en réalité finie.

Exemple :

$$x = (1, 2, 3), \quad h = (4, 5)$$

Alors, la convolution $y_n = (x * h)_n$ est donnée par :

$$\begin{aligned} y_0 &= x_0 h_0 = 1 \cdot 4 = 4 \\ y_1 &= x_0 h_1 + x_1 h_0 = 1 \cdot 5 + 2 \cdot 4 = 13 \\ y_2 &= x_1 h_1 + x_2 h_0 = 2 \cdot 5 + 3 \cdot 4 = 22 \\ y_3 &= x_2 h_1 = 3 \cdot 5 = 15 \end{aligned}$$

$$\Rightarrow (x * h) = (4,\ 13,\ 22,\ 15)$$

Théorème 1. *Tout nombre réel $x \in \mathbb{R}$ se décompose de façon unique en base 10 sous la forme :*

$$x = \pm \sum_{k=-\infty}^{n} a_k \cdot 10^k$$

où :

- *$n \in \mathbb{Z}$ est l'exposant le plus élevé (lié à la partie entière),*
- *$a_k \in \{0, 1, 2, \ldots, 9\}$ sont les chiffres décimaux,*
- *la suite des a_k est nulle pour tout $k > n$,*
- *et la décomposition est **unique**, sauf pour les cas particuliers où le développement décimal est infini avec uniquement des 9 ou des 0 (par exemple : $1 = 0{,}999\ldots$).*

Proof. Nous pouvons prendre $x \in [0, 1[$, car pour tout $x \in \mathbb{R}$, on a :

$$x = \lfloor x \rfloor + \{x\} = 10^0 \lfloor x \rfloor + \{x\},$$

où $\lfloor x \rfloor$ désigne la partie entière de x, et $\{x\}$ sa partie fractionnaire.

Ensuite, il existe un entier $n \in \mathbb{Z}$ tel que :

$$\lfloor 10^n \{x\} \rfloor \neq 0 \quad \text{et} \quad \lfloor 10^{n-1} \{x\} \rfloor = 0.$$

On construit alors par récurrence une suite $(a_m)_{m \in \mathbb{Z}}$, telle que pour tout $m \leq n$, on ait :

$$a_{m-1} = \lfloor 10^m \{x\} \rfloor - a_m \cdot 10^{m-1}.$$

avec $a_n = \lfloor 10^n \{x\} \rfloor$.

Posons :

$$x_p := \sum_{m=n-p}^{n-1} a_m \cdot 10^m$$

et montrons par récurrence que :

$$\lfloor 10^{n-1} x \rfloor = a_{n-1}, \quad \lfloor 10^{n-2} x \rfloor = a_{n-2} + 10 \cdot a_{n-1}, \quad \text{etc.}$$

Ainsi, on suppose avoir au rang k:

$$\lfloor 10^{n-k} x \rfloor = \sum_{j=0}^{k-1} a_{n-1-j} \cdot 10^j$$

Autrement dit, on a :

$$x_p = 10^{n-1} x - \theta_p, \quad \text{avec } 0 \leq \theta_p < 1$$

En divisant par 10^{n-1}, on obtient :

$$\sum_{m=n-p}^{n-1} a_m \cdot 10^m = x - r_p, \quad \text{avec } 0 \leq r_p < 10^{n-p}$$

Ainsi, en prenant la limite quand $p \to \infty$, on obtient :

$$x = \sum_{m=-\infty}^{n-1} a_m \cdot 10^m$$

ce qui montre que la suite (a_m) représente bien le développement décimal de x.
Montrons à présent l'unicité du développement décimal.

Supposons que $x = \sum_{m=-\infty}^{n_1} a_m \cdot 10^m = \sum_{m=-\infty}^{n_2} b_m \cdot 10^m$. Premièrement, $n_1 = n_2$. Et oui ! En supposant par l'absurde et sans perdre en généralité que $n_1 < n_2$ et x¿0, alors en divisant par 10^{n_2}, on a $\frac{x}{10^{n_2}} < 1$ et $\frac{x}{10^{n_2}} > 1$ (Ceci est absurde). Donc $n_1 = n_2 = n$.

Or on avait construit les coefficients du développement décimal par récurrence à partir de l'exposant, donc les suites (a_n) et (b_n) vue comme deux suites verifiant la même relation de récurrence et de termes initiaux identiques. □

On appelle *nombre décimal* tout nombre de la forme $\frac{a}{10^r}$, où $r \in \mathbb{N}$ et $a \in \mathbb{Z}$. On note $\mathbb{D}$ l'ensemble des décimaux.

Tout élement de $\mathbb{D}$ a un développement décimal fini.

Observation clé :

Faire le produit de deux nombres décimaux comme en primaire revient exactement à effectuer la **convolution discrète** des suites de leurs chiffres.

En effet, si l'on écrit deux nombres A et B sous forme de suites finies de chiffres en base 10 :

$$A = \sum_{k=0}^{n} a_k \cdot 10^k \quad \leftrightarrow \quad \text{suite } (a_0, a_1, \ldots, a_n)$$

$$B = \sum_{k=0}^{m} b_k \cdot 10^k \quad \leftrightarrow \quad \text{suite } (b_0, b_1, \ldots, b_m)$$

alors le produit $A \cdot B$ peut s'obtenir en convoluant ces deux suites :

$$c_k = \sum_{i+j=k} a_i b_j \quad \Rightarrow \quad A \cdot B = \sum_{k=0}^{n+m} c_k \cdot 10^k$$

On retrouve ainsi, chiffre par chiffre, les résultats intermédiaires que l'on pose à la main en procédant comme à l'école primaire. Puis ensuite, on somme

Exemple : Pour $A = 32$ et $B = 58$, on identifie :

$$a = (2, 3), \quad b = (8, 5)$$

Le produit correspond alors à la convolution :

$$c_0 = 2 \cdot 8 = 16$$
$$c_1 = 3 \cdot 8 + 2 \cdot 5 = 24 + 10 = 34$$
$$c_2 = 3 \cdot 5 = 15$$

On obtient donc la suite des coefficients convolués :

$$(ab) = (c_0, c_1, c_2) = (16,\ 34,\ 15)$$

Ainsi, le produit brut $A \cdot B$ s'écrit d'abord comme une somme pondérée par les puissances de 10 :

$$A \cdot B = 16 \cdot 10^0 + 34 \cdot 10^1 + 15 \cdot 10^2$$

Mais ceci n'est pas une écriture décimale standard, car les "chiffres" (les c_k) dépassent 9.

On applique alors un **réajustement** pour réécrire cette somme sous la forme canonique d'un développement décimal, où chaque coefficient est un chiffre de 0 à 9, et les dépassements sont propagés vers les puissances supérieures (comme les retenues à l'école primaire).

$$\begin{aligned} &c_0 = 16 \Rightarrow \text{on pose } 6, \text{ et on retient } 1 \text{ vers } c_1 \\ &c_1 = 34 + 1 = 35 \Rightarrow \text{on pose } 5, \text{ et on retient } 3 \text{ vers } c_2 \\ &c_2 = 15 + 3 = 18 \end{aligned}$$

On obtient finalement :

$$A \cdot B = 6 \cdot 10^0 + 5 \cdot 10^1 + 8 \cdot 10^2 + 1 \cdot 10^3 = \boxed{1856}$$

C'est exactement le même résultat qu'avec la méthode scolaire. La convolution donne les produits partiels, et la gestion des retenues permet de convertir ce résultat en une écriture décimale classique.

On obtient donc :

$$\Rightarrow \boxed{1856}$$

C'est bien le résultat du produit 32×58.

Soient $A = (a_n)_{n\in\mathbb{Z}}$ et $B = (b_n)_{n\in\mathbb{Z}}$ deux suites à support fini de taille maximale N, c'est-à-dire qu'il existe $N \in \mathbb{N}$ tel que $a_n = b_n = 0$ pour tout $n \notin \{0, 1, \ldots, N-1\}$.

Alors, le calcul de leur convolution discrète $C = A * B$, donné par

$$c_k = \sum_{i+j=k} a_i b_j,$$

peut se faire en $\mathcal{O}(N^2)$ opérations élémentaires.

Proof. La suite $C = A * B$ résultant de la convolution aura un support contenu dans $\{0, 1, \ldots, 2N-2\}$, soit $2N-1$ termes à calculer.

Pour chaque $k \in \{0, \ldots, 2N-2\}$, on calcule :

$$c_k = \sum_{i=0}^{k} a_i b_{k-i},$$

où seuls les termes avec $i < N$ et $k - i < N$ peuvent être non nuls. Cela signifie que chaque somme contient au plus N produits $a_i b_{k-i}$.

Ainsi, pour chaque k, on effectue au plus N multiplications et $N-1$ additions. Comme il y a $2N-1$ indices k, le nombre total d'opérations est majoré par une constante fois $(N \cdot (2N-1) \sim 2N^2$. D'où la complexité en $\mathcal{O}(N^2)$. □

On voudrait une méthode qui soit alors plus rapide que $\mathcal{O}(N^2)$

2. FFT d'une suite à une support fini et convolution

La **transformation de Fourier discrète** d'une suite à support fini $x = (x_0, x_1, \ldots, x_{N-1})$ de longueur N est définie par :

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1,$$

où X_k représente le coefficient de la transformée pour la fréquence k.

La **transformée de Fourier inverse** d'une suite $X = (X_0, X_1, \ldots, X_{N-1})$ est donnée par :

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{j\frac{2\pi}{N}kn}, \quad n = 0, 1, \ldots, N-1.$$

Elle permet de récupérer la suite originale à partir de sa transformée de Fourier, et possède la propriété suivante : la transformée de Fourier et sa transformée inverse sont des opérations réciproques, c'est-à-dire que l'on retrouve la suite d'origine si l'on applique successivement la transformée de Fourier et la transformée de Fourier inverse.

Soient $x = (x_0, x_1, \ldots, x_{N-1})$ et $y = (y_0, y_1, \ldots, y_{N-1})$ deux suites de longueur N. La convolution de ces deux suites, notée $z = x * y$, est définie par :

$$z_n = \sum_{m=0}^{N-1} x_m \cdot y_{n-m}, \quad n = 0, 1, \ldots, N-1.$$

Si l'on applique la transformation de Fourier à chaque suite, nous avons la relation suivante entre la convolution dans le domaine temporel et la multiplication dans le domaine de Fourier :

$$\mathcal{F}(x * y) = \mathcal{F}(x) \cdot \mathcal{F}(y),$$

où $\mathcal{F}(x)$ représente la transformée de Fourier de la suite x. En d'autres termes, la transformée de Fourier de la convolution de deux suites est égale au produit **termes à termes** des transformées de Fourier de ces suites.

De plus, pour calculer la convolution de deux suites dans la pratique, il est courant de compléter chaque suite par des zéros afin d'obtenir une longueur commune, puis de calculer la FFT de chaque suite, de multiplier les résultats, et enfin d'appliquer la transformée de Fourier inverse pour retrouver la convolution.

Soit $x = (x_0, x_1, \ldots, x_{N-1})$ une suite de longueur N. La Transformée de Fourier discrète (DFT) de cette suite est donnée par :

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1.$$

Calculer cette DFT directement, en appliquant la définition de manière naïve, donne une complexité de $O(N^2)$, car l'expression nécessite deux boucles imbriquées de taille N.

Cependant, l'algorithme de la Transformation de Fourier rapide (FFT) de Cooley-Tukey permet de réduire cette complexité en $O(N \log N)$, en utilisant une approche de **diviser pour régner**. L'idée principale de la FFT est de diviser la DFT de taille N en deux DFT de taille $N/2$, en séparant les termes pairs et impairs de la suite d'entrée. Cette division se poursuit récursivement jusqu'à atteindre des DFT de taille 1, qui sont triviales à calculer.

En d'autres termes, l'algorithme FFT de Cooley-Tukey réduit la taille des sous-problèmes à chaque étape de manière exponentielle. À chaque niveau de la division, il y a $O(N)$ opérations pour combiner les résultats des sous-problèmes. Le nombre total de niveaux est $\log_2 N$, donc la complexité totale de l'algorithme est donnée par :

$$O(N) \times O(\log N) = O(N \log N).$$

Ainsi, l'algorithme FFT permet de calculer la DFT en un temps beaucoup plus court que la méthode brute, surtout pour des valeurs grandes de N. Pour plus de détails, je vous conseille cette vidéo: https://www.youtube.com/watch?v=htCj9exbGo0.

L'idée est alors la suivante : on applique la FFT à chacune des suites des coefficients de la décomposition en base 10 des deux réels A et B, puis on multiplie les images obtenues terme à terme. Grâce à la remarque précédente, il suffit ensuite d'appliquer la transformée de Fourier rapide inverse (IFFT) au résultat pour obtenir le produit convolué de A et B via la méthode FFT.

Grâce à la bibliothèque Numpy nous pourrons comparer les deux méthodes de calcul de produit: convolution ou FFT

```
import numpy as np
import time
##Fast fourier
start = time.time()


x = np.random.randint(0, 10, 10000)
X = np.fft.fft(x)
y=np.random.randint(0, 10, 10000)
Y = np.fft.fft(y)

Z=X*Y

Res=np.fft.ifft(Z)
end = time.time()
print(f"Temps d'execution avec FFT: {end - start:.6f} secondes")

## vs Convolution
start=time.time()
def convolution(a, b):
    len_a = len(a)
    len_b = len(b)
    result = [0] * (len_a + len_b - 1)

    # Calcul de la convolution
    for i in range(len_a):
        for j in range(len_b):
            result[i + j] += a[i] * b[j]

    return result

Res2=convolution(x,y)
end=time.time()
print(f"Temps d'execution convolution : {end - start:.6f} secondes")
```

On trouve :
Temps d'exécution avec FFT: 0.001901 secondes
Temps d'exécution convolution : 27.670099 secondes

Conclusion

On a trouvé une méthode vraiment rapide pour effectuer le produit de deux décimaux (et donc d'à peu-près tout les réels car $\mathbb{D}$ est dense dans $\mathbb{R}$. Seulement, la compléxité algorithmique n'est pas identique à la complexité cerebrale d'un calcul. En pratique, surtout pour des enfants et surtout pour des décimaux à écriture courte, faire une FFT de tête ou bien posée ne sera jamais plus rapide, qu'une bonne vieille convolution des familles.

Email address: alexislcrd@gmail.com