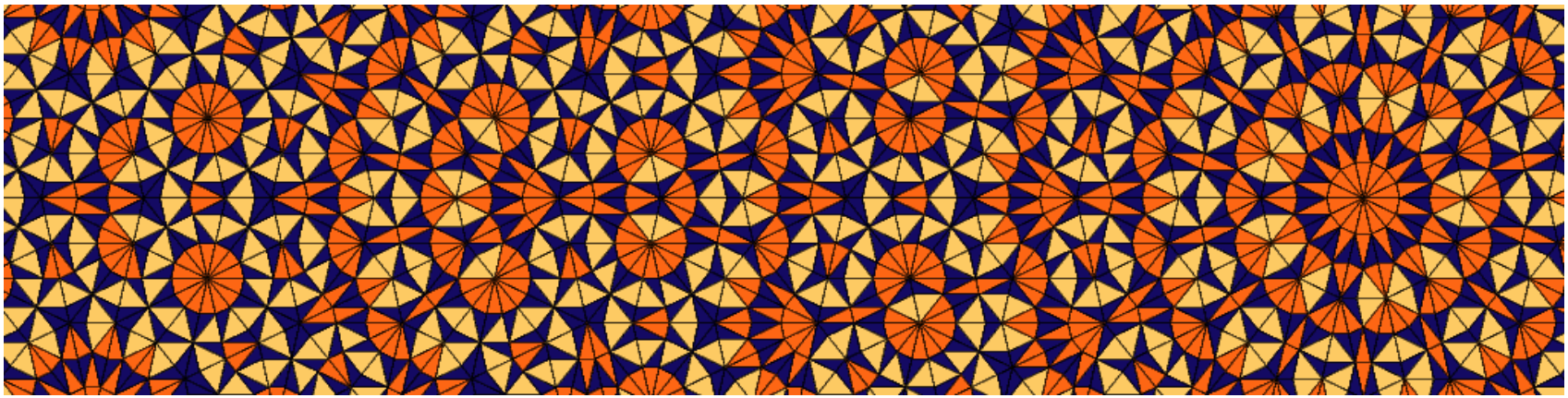
Panorama der Mathematik und Informatik

23: Algorithmen VI: Schnelle Multiplikation

Dirk Frettlöh
Technische Fakultät / Richtig Einsteigen



Eine weitere Anwendung der schnellen Fouriertransformation: Multiplizieren zweier Zahlen mit n Stellen in Zeit $O(n \log n)$.

Normaler Algorithmus: (siehe auch "Geschichte": "Rechenstäbe")

	123	·	456	
		6	12	18
	5	10	15	
4	8	12		
5	6	0	8	8

3 mal 3 Multiplikationen, 5 Additionen, bzw. allgemein:

n^2 Multiplikationen und ca. $2n - 1$ Additionen

Zur einfacheren Darstellung des über 40 Jahre alten und immer noch besten[1] Algorithmus von **Schönhage-Strassen**.

Dazu zunächst Mal:

Multiplikation von Polynomen:

$$
\begin{aligned}
&(x^3 + x^2 + x + 1)(x^3 + x^2 + 1) \\
&= x^6 + x^5 + x^4 + x^3 + x^5 + x^4 + x^3 + x^2 + x^3 + x^2 + x + 1 \\
&= x^6 + 2x^5 + 2x^4 + 3x^3 + 2x^2 + x + 1
\end{aligned}
$$

[1] seit 2007 gibt es theoretisch schneller, aber dieser ist der Standard für alles ab ca 50 000 Dezimalstellen.

Schreiben wir die Polynome als Koeffizientenvektoren, wird aus $1 + x + x^2 + x^3$ mal $1 + 0 \cdot x + x^2 + x^3$ gleich $1 + x + 2x^2 + 3x^3 + 2x^4 + 2x^5 + x^6$ dieses:

$$\begin{aligned} f \otimes g &= (1,1,1,1,0,0,0,0) \otimes (1,0,1,1,0,0,0,0) \\ &= (1,1,2,3,2,2,1,0) = h. \end{aligned}$$

Schreiben wir die Polynome als Koeffizientenvektoren, wird aus $1 + x + x^2 + x^3$ mal $1 + 0 \cdot x + x^2 + x^3$ gleich $1 + x + 2x^2 + 3x^3 + 2x^4 + 2x^5 + x^6$ dieses:

$$\begin{aligned} f \otimes g &= (1,1,1,1,0,0,0,0) \otimes (1,0,1,1,0,0,0,0) \\ &= (1,1,2,3,2,2,1,0) = h. \end{aligned}$$

Was ist das $\otimes$? Sind f und g die Koeffizientenvektoren, dann ist Eintrag Nummer 0 von h (also h_0) gleich $g_0 \cdot f_0$. Weiter ist $h_1 = f_0 \cdot g_1 + f_1 \cdot g_0$, $h_2 = f_0 \cdot g_2 + f_1 \cdot g_1 + f_2 \cdot g_0$ usw.

Schreiben wir die Polynome als Koeffizientenvektoren, wird aus $1 + x + x^2 + x^3$ mal $1 + 0 \cdot x + x^2 + x^3$ gleich $1 + x + 2x^2 + 3x^3 + 2x^4 + 2x^5 + x^6$ dieses:

$$\begin{aligned} f \otimes g &= (1,1,1,1,0,0,0,0) \otimes (1,0,1,1,0,0,0,0) \\ &= (1,1,2,3,2,2,1,0) = h. \end{aligned}$$

Was ist das $\otimes$? Sind f und g die Koeffizientenvektoren, dann ist Eintrag Nummer 0 von h (also h_0) gleich $g_0 \cdot f_0$. Weiter ist $h_1 = f_0 \cdot g_1 + f_1 \cdot g_0$, $h_2 = f_0 \cdot g_2 + f_1 \cdot g_1 + f_2 \cdot g_0$ usw.
Allgemein ist Eintrag Nummer n von $f \otimes g$:

$$h = f \otimes g, \quad h_n = \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

Schreiben wir die Polynome als Koeffizientenvektoren, wird aus $1 + x + x^2 + x^3$ mal $1 + 0 \cdot x + x^2 + x^3$ gleich $1 + x + 2x^2 + 3x^3 + 2x^4 + 2x^5 + x^6$ dieses:

$$\begin{aligned} f \otimes g &= (1,1,1,1,0,0,0,0) \otimes (1,0,1,1,0,0,0,0) \\ &= (1,1,2,3,2,2,1,0) = h. \end{aligned}$$

Was ist das $\otimes$? Sind f und g die Koeffizientenvektoren, dann ist Eintrag Nummer 0 von h (also h_0) gleich $g_0 \cdot f_0$. Weiter ist $h_1 = f_0 \cdot g_1 + f_1 \cdot g_0$, $h_2 = f_0 \cdot g_2 + f_1 \cdot g_1 + f_2 \cdot g_0$ usw. Allgemein ist Eintrag Nummer n von $f \otimes g$:

$$h = f \otimes g, \quad h_n = \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

Obacht: es tauchen $g_{-1}, g_{-2}, \ldots$ auf. Kein Problem: Vereinbaren wir $g_{-1} := g_{N-1}$, $g_{-2} := g_{N-2}$ usw für diese. Machen wir unsere Vektoren lang genug, so sorgen die vielen führenden Nullen dafür, dass kein Fehler passiert (nachprüfen!)

Also

$$h = f \otimes g, \quad h_n = \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

Also

$$h = f \otimes g, \quad h_n = \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

In der (diskreten) Fouriertheorie gibt es den Begriff der Faltung:

$$f * g(n) = \frac{1}{N} \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

Noch besser, es gibt einen **Faltungssatz:**

$$DFT(f * g) = N \cdot DFT(f) \cdot DFT(g)$$

Also

$$h = f \otimes g, \quad h_n = \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

In der (diskreten) Fouriertheorie gibt es den Begriff der Faltung:

$$f * g(n) = \frac{1}{N} \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \leq n \leq N-1)$$

Noch besser, es gibt einen **Faltungssatz:**

$$DFT(f * g) = N \cdot DFT(f) \cdot DFT(g)$$

Was heißt das zweite "mal"? Eintragsweise malnehmen!

Fakt: Da f ein Koeffizientenvektors eines Polynoms $p(x)$ war, ist $N{\cdot}\mathrm{DFT}(f)$ der Vektor der Funktionswerte $(p(1), p(\xi^{-1}), p(\xi^{-2}), , \ldots, p(\xi^{-(N-1)}))$.

Setzen wir der Einfachheit halber $\zeta = \xi^{-1}$. Dann ist $N{\cdot}\mathrm{DFT}(f)$ der Vektor der Funktionswerte $(p(1), p(\zeta), p(\zeta^2), , \ldots, p(\zeta^{N-1}))$.

Bsp: $p(x) = 1 + 2x - x^3$.

$$\mathrm{DFT}(p) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \zeta & \zeta^2 & \zeta^3 \\ 1 & \zeta^2 & \zeta^4 & \zeta^6 \\ 1 & \zeta^3 & \zeta^6 & \zeta^9 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 1+2-1 \\ 1+2\zeta-\zeta^3 \\ 1+2\zeta^2-\zeta^6 \\ 1+2\zeta^3-\zeta^9 \end{pmatrix}.$$

Damit kann man sich überlegen:

- $N \cdot (f * g)$ ist "Polynom mal Polynom" (f mal g)
- $N \cdot \mathrm{DFT}(f * g)$ ist dann der Vektor der Funktionswerte von f mal g, also $(f \cdot g)(1), (f \cdot g)(\zeta), \ldots (f \cdot g)(\zeta^{N-1})$.

Damit kann man sich überlegen:

- $N \cdot (f * g)$ ist "Polynom mal Polynom" (f mal g)
- $N \cdot \mathrm{DFT}(f * g)$ ist dann der Vektor der Funktionswerte von f mal g, also $(f \cdot g)(1), (f \cdot g)(\zeta), \ldots (f \cdot g)(\zeta^{N-1})$. Also $f(1) \cdot g(1), f(\zeta) \cdot g(\zeta), \ldots f(\zeta^{N-1}) \cdot g(\zeta^{N-1})$.

Damit kann man sich überlegen:

- $N \cdot (f * g)$ ist "Polynom mal Polynom" (f mal g)
- $N \cdot \mathrm{DFT}(f * g)$ ist dann der Vektor der Funktionswerte von f mal g, also $(f \cdot g)(1), (f \cdot g)(\zeta), \ldots (f \cdot g)(\zeta^{N-1})$. Also $f(1) \cdot g(1), f(\zeta) \cdot g(\zeta), \ldots f(\zeta^{N-1}) \cdot g(\zeta^{N-1})$.

Die DFT lässt sich auch umdrehen: IDFT. Berechnet sich fast genau wie die DFT (insbesondere auch schnell: FFT).

Matrix IDFT:

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \xi^1 & \xi^2 & \ldots & \xi^{(N-1)} \\ 1 & \xi^2 & \xi^4 & \ldots & \xi^{2(N-1)} \\ 1 & \xi^3 & \xi^6 & \ldots & \xi^{3(N-1)} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & \xi^{(N-1)} & \xi^{2(N-1)} & \ldots & \xi^{(N-1)^2} \end{pmatrix}$$

Idee: Berechne

$$N \cdot (f * g) = N \cdot IDFT\big(DFT(f * g)\big) = N \cdot IDFT\big(N \cdot DFT(f) \cdot DFT(g)\big)$$
$$= N^2 \cdot IDFT\big(DFT(f) \cdot DFT(g)\big)$$

Aufwand: Insgesamt $O(N \log N)$.

$$N \cdot (f * g) = N^2 \cdot IDFT\big(DFT(f) \cdot DFT(g)\big)$$

- Drei (I)DFTs: 3 mal $O(N \log N)$ Operationen.
- Eintragsweise multiplizieren: N Operationen.
- Evtl. im Ergebnis alles nochmal "mal N^2": N Operationen.

Idee: Berechne

$$N \cdot (f * g) = N \cdot IDFT\big(DFT(f * g)\big) = N \cdot IDFT\big(N \cdot DFT(f) \cdot DFT(g)\big)$$
$$= N^2 \cdot IDFT\big(DFT(f) \cdot DFT(g)\big)$$

Aufwand: Insgesamt $O(N \log N)$.

$$N \cdot (f * g) = N^2 \cdot IDFT\big(DFT(f) \cdot DFT(g)\big)$$

- Drei (I)DFTs: 3 mal $O(N \log N)$ Operationen.
- Eintragsweise multiplizieren: N Operationen.
- Evtl. im Ergebnis alles nochmal "mal N^2": N Operationen.

Also können wir nun zwei Polynome (vom Grad $\leq N/2$) multiplizieren mit Aufwand $O(N \log N)$ statt $O(N^2)$.

Also auch Binärzahlen...

...denn: Statt des Koeffizientenvektors eines Polynoms betrachte den Vektor der Binärdarstellung der Länge $2N$ zweier Zahlen mit der Länge N. (Man überlege sich: wenn $(p_0, p_1, p_2, \ldots)$ die Binärdarstellung ist, und p das Polynom zu diesem Koeffizientenvektor, was ist dann $p(2)$?)

Beispiel:

$$15 = (1,1,1,1,0,0,0,0), \quad 5 = (1,0,1,0,0,0,0,0)$$

Wir wenden den Algorithmus an:

$$15 \cdot 5 = (1,1,2,2,1,1,0,0)$$

Das ist so keine Binärzahl, aber Abarbeiten der Überträge (von links nach rechts) liefert die korrekte Binärzahl:

$$(1,1,2,2,1,1,0,0) \to (1,1,0,3,1,1,0,0) \to (1,1,0,1,2,1,0,0)$$
$$\to (1,1,0,1,0,2,0,0) \to (1,1,0,1,0,0,1,0) = 75 = 15 \cdot 5$$

Es gibt noch Nachteile:

- Bisher benutzten wir bei der DFT komplexe Einheitswurzeln: Lösungen der Gleichung $x^N = 1$ in $\mathbb{C}$. Also $x = e^{2\pi i/N}$. Daher
  - komplexe Zahlen
  - nicht immer ganzzahlig

Der Algorithmus von Schönhage-Strassen benutzt statt komplexer Einheitswurzeln (in $\mathbb{C}$) Einheitswurzeln in dem *Ring* $\{0, 1, \ldots, N-1\}$ mit $+ \mod N$ und $\cdot \mod N$: Lösungen von

$$x^n = 1 \mod N$$

Dadurch rechnet man ganzzahlig (und reell).

Es gibt noch Nachteile:

- Bisher benutzten wir bei der DFT komplexe Einheitswurzeln: Lösungen der Gleichung $x^N = 1$ in $\mathbb{C}$. Also $x = e^{2\pi i/N}$. Daher
  - komplexe Zahlen
  - nicht immer ganzzahlig

Der Algorithmus von Schönhage-Strassen benutzt statt komplexer Einheitswurzeln (in $\mathbb{C}$) Einheitswurzeln in dem *Ring* $\{0, 1, \ldots, N-1\}$ mit $+ \mod N$ und $\cdot \mod N$: Lösungen von

$$x^n = 1 \mod N$$

Dadurch rechnet man ganzzahlig (und reell).

Man wählt außerdem $N = 2^k + 1$, dann sind die Einheitswurzeln alle von der Form 2^m. "Mal Einheitswurzel" ist dann billig: shift der Binärzahl.
Außerdem ist $\mod 2^k + 1$ auch billig.

Laufzeit des Schönhage-Strassen-Algorithmus zur Multiplikation zweier Zahlen mit n Binärstellen ist $O(n \log n \log\log n)$.

Für Zahlen mit 1024 Bit bereits besser als "normale" Multiplikation: Statt ca. $1024^2 = 1.048.576$ Operationen für "naive" Multiplikation brauchen wir drei DFTs der Länge 2048: weniger als 180.000 Operationen.

Laufzeit des Schönhage-Strassen-Algorithmus zur Multiplikation zweier Zahlen mit n Binärstellen ist $O(n \log n \log\log n)$.

Für Zahlen mit 1024 Bit bereits besser als "normale" Multiplikation: Statt ca. $1024^2 = 1.048.576$ Operationen für "naive" Multiplikation brauchen wir drei DFTs der Länge 2048: weniger als 180.000 Operationen.

Der Schönhage-Strassen-Algorithmus war von 1971 bis 2007 der effizienteste bekannte Algorithmus zur Multiplikation großer Zahlen; 2007 veröffentlichte Martin **Fürer** *eine Weiterentwicklung des Algorithmus mit einer noch niedrigeren asymptotischen Komplexität.*
Diese Komplexität stellt eine Verbesserung sowohl gegenüber dem "naiven" aus der Schule bekannten Algorithmus der Laufzeit $O\left(n^2\right)$ als auch gegenüber dem 1962 entwickelten **Karatsuba-Algorithmus** *mit einer Laufzeit von $O\left(n^{\log_2(3)}\right)$ sowie dessen verbesserter Variante, dem* **Toom-Cook-Algorithmus** *mit $O(n^{1+\varepsilon})$ Laufzeit dar." (wikipedia)*

Karatsuba und Tom-Cook sind auch Divide-and-Conquer-Algorithmen.

In der Praxis ist Schönhage-Strassen ab etwa $n = 2^{2^{15}}$ bis $2^{2^{17}}$ (also ca 10 000 bis 40 000 Dezimalstellen) besser als Karatsuba und Tom-Cook.

"Fürer's algorithm currently only achieves an advantage for astronomically large values and is not used in practice." (wikipedia)