3. Effiziente Algorithmen und ihre Komplexität

3.1. Multiplikation ganzer Zahlen

Der "klassische" Algorithmus zur Multiplikation zweier ganzer Zahlen x und y hat die Komplexität $l(x) \cdot l(y)$, da im wesentlichen jede Ziffer von x mit jeder Ziffer von y multipliziert werden muß. Um zwei Zahlen der Länge n zu multiplizieren, braucht man also $\mathcal{O}(n^2)$ Zeit.

Im Jahre 1962 hatten A. Karatsuba und Yu. Ofman [1)] eine Idee, wie man einen Multiplikationsalgorithmus mit besserem Komplexitätsverhalten konstruieren könnte. Sei β der Radix der Zahldarstellung. Der Einfachheit halber nehmen wir an, die Länge n der beiden Multiplikanden x und y wäre eine Potenz von 2. Wir zerschlagen die Zahlen jeweils in zwei Teile

$$\begin{aligned} x &= a \cdot \beta^{n/2} + b, \\ y &= c \cdot \beta^{n/2} + d, \end{aligned} \tag{3.1.1}$$

wobei jeder dieser Teile die Länge $n/2$ hat. Das Produkt ergibt sich dann als

$$\begin{aligned} z = x \cdot y &= ac \cdot \beta^n + (ad + bc) \cdot \beta^{n/2} + bd \\ &= ac \cdot \beta^n + \big((a+b)(c+d) - ac - bd\big) \cdot \beta^{n/2} + bd. \end{aligned} \tag{3.1.2}$$

Zur Berechnung von xy genügen also drei Multiplikationen $(n/2)$–stelliger Zahlen und einige Additionen und Verschiebungen um Potenzen von β. Diese Operationen brauchen nur $\mathcal{O}(n)$ Zeit. Wir erhalten das Programm

$$\begin{array}{l} \textbf{begin} \\ \quad u \leftarrow (a+b) * (c+d); \\ \quad v \leftarrow a * c; \\ \quad w \leftarrow b * d; \\ \quad z \leftarrow v * \beta^n + (u - v - w) * \beta^{n/2} + w \\ \textbf{end} \end{array} \tag{3.1.3}$$

Zur Berechnung der Produkte u, v und w wird die Multiplikationsprozedur rekursiv angewandt.

Wir nehmen bis auf weiteres an, daß $a + b$ und $c + d$ jeweils $(n/2)$–stellige Zahlen wären und ignorieren somit den möglichen Übertrag, der bei der Addition von a und

[1)] A. Karatsuba und Yu. Ofman: *Dokl. Akad. Nauk SSSR* 145, 293–294 (1962) [englische Übersetzung: *Multiplication of Multidigit Numbers on Automata, Soviet Phys. Dokl.* 7, 595–596 (1963)].

b entstehen und zu einer $(n/2\ +1)$–stelligen Zahl führen kann. Die Zeit für die Multiplikation zweier n–stelliger Zahlen ist also nach oben beschränkt durch

$$T(n) = \begin{cases} k, & \text{für } n = 1 \\ 3T(n/2) + kn, & \text{für } n > 1, \end{cases} \tag{3.1.4}$$

wobei kn eine obere Schranke für die Additionen und Verschiebungen in (3.1.3) ist. Als Lösung von (3.1.4) erhalten wir

$$T(n) = 3k\,n^{\log 3} - 2kn, \tag{3.1.5}$$

was durch Induktion leicht zu beweisen ist [2]:

> die Basis, für $n = 1 = 2^0$, ist trivial. Erfüllt $T(\overline{n}) = 3k\,\overline{n}^{\log 3} - 2k\overline{n}$ die Beziehung (3.1.4), so gilt auch
>
> $$\begin{aligned} T(2\overline{n}) &= 3T(\overline{n}) + 2k\overline{n} \\ &= 3(3k\,\overline{n}^{\log 3} - 2k\overline{n}) + 2k\overline{n} \\ &= 3k \cdot 2^{\log 3} \cdot \overline{n}^{\log 3} - 6k\overline{n} + 2k\overline{n} \\ &= 3k(2\overline{n})^{\log 3} - 2k(2\overline{n}). \end{aligned}$$

Wir kehren zurück zum Problem, daß $a+b$ und $c+d$ eventuell die Länge $n/2\ +1$ haben könnten. In diesem Fall schreiben wir $a+b$ als $a_1\beta^{n/2}+b_1$, wobei a_1 das führende Bit von $a+b$ ist und b_1 die restlichen Bits. Ebenso schreiben wir $c+d$ als $c_1\beta^{n/2}+d_1$. Das Produkt $(a+b)(c+d)$ ergibt sich nun als

$$a_1c_1 \cdot \beta^n + (a_1d_1 + b_1c_1) \cdot \beta^{n/2} + b_1d_1. \tag{3.1.6}$$

Der Term b_1d_1 wird durch eine rekursive Anwendung der Multiplikationsprozedur auf Zahlen der Länge $n/2$ berechnet. Die übrigen Multiplikationen in (3.1.6) sind entweder Multiplikationen mit einem einzigen Bit oder Verschiebungen, haben also die Komplexität $\mathcal{O}(n)$.

Somit ist die Zeitkomplexität des Karatsuba Algorithmus $\mathcal{O}(n^{\log 3}) \simeq \mathcal{O}(n^{1.59})$. Bisher haben wir nur Potenzen von 2 als Längen n zugelassen. Ist n keine Potenz von 2, so erhöhen wir die Länge (durch Hinzufügen führender Nullen) zur nächsten Potenz von 2. Dabei wird die Länge höchstens verdoppelt. Der Faktor 2 kann aber in der asymptotischen Komplexität vernachlässigt werden, denn $\mathcal{O}((2n)^{\log 3}) = \mathcal{O}(n^{\log 3})$.

---

[2] log ist dabei der Logarithmus zur Basis 2.

Satz 3.1.1 (Karatsuba): *Zwei Zahlen der Länge n können in Zeit $\mathcal{O}(n^{\log 3})$ multipliziert werden.* ■

Beispiel 3.1.1: Wir betrachten das folgende Zahlenbeispiel für den Karatsuba Algorithmus. Der Radix β der Zahlendarstellung sei dabei 10.

$$\begin{array}{lll} x = 3141 & a = 31 & b = 41 \\ y = 5927 & c = 59 & d = 27 \end{array}$$

$$\begin{aligned} u &= (a+b)(c+d) = 72 \times 86 = 6192 \\ v &= ac = 31 \times 59 = 1829 \\ w &= bd = 41 \times 27 = 1107 \\ xy &= 1829 \times 10000 + (6192 - 1829 - 1107) \times 100 + 1107 \\ &= 18616707 \end{aligned}$$ ■

Der Karatsuba Algorithmus ist ein Beispiel für einen "divide–and–conquer" Algorithmus. Dabei wird ein Problem dadurch gelöst, daß es in mehrere Teilprobleme geringerer Problemgröße zerlegt wird. Die Zeitkomplexität eines solchen Algorithmus ist bestimmt von der Anzahl und Größe der Teilprobleme und zu einem geringeren Ausmaß vom Arbeitsaufwand für die Zerlegung des Problems. Rekursionsrelationen der Art (3.1.4) treten bei der Analyse rekursiver "divide–and–conquer" Algorithmen häufig auf. Der folgende Satz gibt Auskunft über die Lösung solcher Rekursionsrelationen. Dabei betrachten wir den Fall, daß ein Problem der Größe n zerlegt wird in a Probleme der Größe n/c. Der Aufwand für die Zerlegung und anschließende Kombination der Teilergebnisse sei $b \cdot n$.

Satz 3.1.2: *Seien a, b und c natürliche Zahlen. Die Lösung der Rekursionsrelation*

$$T(n) = \begin{cases} b, & \text{für } n = 1, \\ aT(n/c) + bn, & \text{für } n > 1 \end{cases}$$

für eine Potenz n von c ist

$$T(n) = \begin{cases} \mathcal{O}(n), & \text{falls } a < c, \\ \mathcal{O}(n \log n), & \text{falls } a = c, \\ \mathcal{O}(n^{\log_c a}), & \text{falls } a > c. \end{cases}$$

Beweis: Wir zeigen zunächst für alle Potenzen n von c

$$T(n) = bn \sum_{i=0}^{\log_c n} r^i, \qquad \text{wobei } r = a/c.$$

Die Basis, für $n = c^0 = 1$, ist klar. Erfüllt nun $T(\overline{n}) = b\overline{n} \sum_{i=0}^{\log_c \overline{n}} r^i$ die Rekursionsrelation, so gilt auch

$$\begin{aligned} T(c\overline{n}) &= aT(\overline{n}) + bc\overline{n} \\ &= a\big(b\overline{n} \textstyle\sum_{i=0}^{\log_c \overline{n}} r^i\big) + bc\overline{n} \\ &= b(c\overline{n})\big(\tfrac{a}{c} \cdot \textstyle\sum_{i=0}^{\log_c \overline{n}} r^i \ + 1\big) \\ &= b(c\overline{n}) \cdot \textstyle\sum_{i=0}^{\log_c (c\overline{n})} r^i. \end{aligned}$$

Ist $a < c$, dann konvergiert die geometrische Reihe $\sum_{i=0}^{\infty} r^i$, also $T(n)$ ist $\mathcal{O}(n)$.

Ist $a = c$, so ist jeder der $\mathcal{O}(\log n)$ Summanden in $T(n)$ 1, also $T(n)$ ist $\mathcal{O}(n \log n)$.

Ist $a > c$, dann ist die partielle Summe der geometrischen Reihe

$$T(n) \ = \ bn \cdot \frac{r^{1+\log_c n} - 1}{r - 1} \ \leq \ bn \cdot \frac{r}{r-1} \cdot r^{\log_c n} \ \leq \ 2bn \cdot \frac{a^{\log_c n}}{\underbrace{c^{\log_c n}}_{=n^{\log_c c} = n}} \ = \ 2b \cdot a^{\log_c n},$$

also $\mathcal{O}(a^{\log_c n})$, oder anders ausgedrückt $\mathcal{O}(n^{\log_c a})$. Dabei haben wir das allgemeine Gesetz $u^{\log_s v} = v^{\log_s u}$ verwendet. ■

Ist die Problemgröße n keine Potenz von c, so kann man gewöhnlich ein Problem der Größe n einbetten in ein Problem der Größe n', wobei n' die nächst größere Potenz von c ist. Die asymptotische Wachstumsrate aus Satz 2.1.3 gilt also für beliebiges n.

Aus Satz 3.1.2 sehen wir, daß man durch die Zerlegung eines Problems (bei linearem Aufwand für die Zerlegung) in zwei Unterprobleme der halben Größe einen $\mathcal{O}(n \log n)$ Algorithmus erhält. Ist die Anzahl der Unterprobleme 3, 4 oder 8, dann ist der Algorithmus von der Ordnung $n^{\log 3}$, n^2 oder n^3. Zerlegt man das Problem in 4, 9 oder 16 Unterprobleme der Größe $n/4$, so erhält man einen Algorithmus der Ordnung $n \log n$, $n^{\log 3}$ oder n^2. Man könnte also einen asymptotisch schnelleren Muliplikationsalgorithmus dadurch erhalten, daß man die Multiplikanden in 4 Teile zerlegt und das Produkt ausdrückt in Termini von 8 oder weniger Produkten der Teile. D.E. Knuth [3)] beweist

[3)] D.E. Knuth: *The Art of Computer Programming, Vol. 2,* Addison–Wesley, Reading, Massachusetts (1969).

Satz 3.1.3: *Für jede positive reelle Zahl ϵ gibt es einen Multiplikationsalgorithmus für natürliche Zahlen, welcher zur Multiplikation zweier Zahlen der Länge $\leq n$ Zeit $\mathcal{O}(n^{1+\epsilon})$ braucht.* ∎

Den bisher bekannten Algorithmus mit der besten asymptotischen Komplexität erhält man durch Verwendung der schnellen Fourier Transformation. Es ist dies der Algorithmus von Schönhage und Strassen; seine Komplexität ist $\mathcal{O}(n \log n \log\log n)$. Wir verweisen dazu auf [AHU], [Lip] und [Sed].

Jede Verbesserung der Komplexität der Multiplikation zieht auch eine Komplexitätsverbesserung anderer Operationen auf den ganzen Zahlen nach sich. Sei $M(n)$ die Zeit für die Multiplikation zweier n–stelliger Zahlen, $D(n)$ die Zeit um eine $2n$–stellige Zahl durch eine n–stellige Zahl zu dividieren, so gilt der folgende Satz.

Satz 3.1.4: *Die Komplexitätsfunktionen $M(n)$ und $D(n)$ unterscheiden sich nur um einen konstanten Faktor.*

Beweis: [AHU] Theorem 8.5. ∎

3.2. Multiplikation von Polynomen

Die “klassische Methode zur Multiplikation von Polynomen $p(x) = \sum_{i=0}^{m} p_i x^i$ und $q(x) = \sum_{j=0}^{n} q_j x^j$ über einem Ring R benutzt die Formel

$$r(x) = p(x) \cdot q(x) = \sum_{l=0}^{m+n} \Big(\sum_{i+j=l} p_i \cdot q_j \Big) x^l.$$

Sind p und q Polynome in ν Variablen in dichter Darstellungsweise, so ist der Zeitaufwand für die Multiplikation von p und q abschätzbar durch

$$M(p_0, q_o) \cdot \prod_{i=1}^{\nu} (1 + \deg_i p)(1 + \deg_i q),$$

wobei p_0 und q_0 die größten Koeffizienten in p und q sind und $M(r, s)$ eine Komplexitätsabschätzung für die Multiplikation von Elementen r, s in R ist. Ist etwa $M(r, s)$ von der Ordnung 1, so ist die Zeit für die Multiplikation zweier Polynome vom Grad $\leq n$ in jeder der ν Variablen $\mathcal{O}(n^{2\nu}) = \mathcal{O}((n^\nu)^2)$.

Für den Fall dichter Polynome ist es auch sinnvoll, die Karatsuba Methode zur Multiplikation anzuwenden. Dazu zerlegen wir

$$p(x) = p_1(x) \cdot x^{\lceil n/2 \rceil} + p_0(x) \quad \text{und} \quad q(x) = q_1(x) \cdot x^{\lceil n/2 \rceil} + q_0(x).$$

Es gilt nun

$$r(x) = p(x) \cdot q(x) = p_1 q_1 x^{2\lceil n/2 \rceil} + \big((p_1 + p_0)(q_1 + q_0) - p_1 q_1 - p_0 q_0 \big) x^{\lceil n/2 \rceil} + p_0 q_0.$$

Man kommt also mit drei Teilmultiplikationen aus, wobei die Multiplikanden jeweils vom Grad $\leq \lceil n/2 \rceil$ sind. Daraus ergibt sich (wiederum unter der Annahme $M(r, s) = \mathcal{O}(1)$) die Zeilkomplexität $\mathcal{O}(n^{\nu \cdot \log_2 3}) = \mathcal{O}((n^\nu)^{\log_2 3})$ für die Multiplikation zweier Polynome mit Grad $\leq n$ in jeder der ν Variablen.

3.3. Evaluierung von Polynomen

Es sei das Polynom $p(x) = \sum_{i=0}^{n} p_i x^i$ in der Variablen x über dem Ring R und ein Element x_0 aus R gegeben. Gesucht ist der Wert von p an der Stelle x_0, also $p(x_0)$.

Man könnte nun sukzessive p_0, $p_1 x, \ldots, p_n x^n$ berechnen und bräuchte auf diese Weise $2n-1$ Multiplikationen und n Additionen im Ring R zur Berechnung von $p(x_0)$. Eine wesentliche Verbesserung der Komplexität läßt sich jedoch mit der *Hornerschen Regel* erreichen. Dabei wird $p(x_0)$ nach dem Schema

$$p(x_0) = (\ldots(p_n x_0 + p_{n-1}) \cdot x_0 + \ldots) \cdot x_0 + p_0$$

berechnet. Dazu sind n Multiplikationen und n Additionen in R notwendig. Die Hornersche Regel ergibt sich aus der Division von $p(x)$ durch $x - a$.

Zerlegt man das Polynom $p(x)$ in der Art

$$p(x) = \underbrace{\sum_{j=0}^{\lfloor n/2 \rfloor} a_j x^{2j}}_{p'} + \underbrace{\sum_{k=0}^{\lceil n/2 \rceil - 1} b_k x^{2k+1}}_{p''},$$

so ist

$$p'(x_0) = (\ldots(a_{\lfloor n/2 \rfloor} x_0^2 + a_{\lfloor n/2 \rfloor - 1}) x_0^2 + \ldots) x_0^2 + a_0 \qquad \text{und}$$
$$p''(x_0) = ((\ldots(b_{\lceil n/2 \rceil - 1} x_0^2 + b_{\lceil n/2 \rceil - 2}) x_0^2 + \ldots) x_0^2 + b_0) x_0.$$

Diese Methode nennt man die *Hornersche Regel zweiter Ordnung*. Um nun $p(x_0)$ auf diese Weise zu berechnen, braucht man $1 + \lfloor n/2 \rfloor + \lceil n/2 \rceil = n+1$ Multiplikationen und $\lfloor n/2 \rfloor + (\lceil n/2 \rceil - 1) + 1 = n$ Additionen in R. Das ist zwar keine Verbesserung gegenüber der Hornerschen Regel erster Ordnung, hat man aber sowohl $p(x_0)$ als auch $p(-x_0)$ zu berechnen, so kann man das durch eine einzige zusätzliche Addition erreichen. Die Hornersche Regel zweiter Ordnung ergibt sich aus der Division von $p(x)$ durch $x^2 - a^2$.