Grundlagen und Karatsuba Algorithmus

Qianli Wang 13. Juni 2020 Katharina Klost

1 Einführung

Heutzutage kennen fast alle Multiplikation, sofern man die Grundschule besucht hat. Die Multiplikation von zwei Zahlen ist eine der gebräuchlichsten Operationen für einen Computer, aber mit zunehmender Größe von zwei Multiplikatoren nimmt die Komplexität des Algorithmus zur Berechnung ihres Produkts dramatisch zu. Die Multiplikation ist wirklich brauchbar und relevant mit unserem Leben. z.B. die Verschlüsselung von Nachrichten, Kryptographie, polynomielle Multiplikation oder Lösung von geometrischen Problemen. Jedoch ist die Methode zum Multiplizieren, die wir in der Grundschule gelernt haben, nicht so effizient, wie wir uns vorstellen bzw erwarten, was im Hauptteil nochmal erklärt wird. Um diese Beschränkung zu umgehen, wird der Karatsuba Algorithmus eingeführt, der uns ermöglicht, die Effizienz der Multiplikation bis zu einem gewissen Grad zu verbessern.

2 Grundlegende Definitionen und Ergebnisse

Definition 1. Eine Grundoperation ist eine Operation, die der Prozessor direkt unterstützt, damit wir mit Hilfe von der Anzahl von Grundoperationen auf Umwegen den Zeitaufwand eines Algorithmus besser darstellen können.

Beobachtung 2. *Die Addition und Multiplikation kostet nur konstante Zeit, also* $\mathcal{O}(1)$.

2.1 Addition zweier Zahlen mit Länge n

Wie viele Grundoperationen muss man mindestens durchführen, damit man zwei Zahlen mit Länge n miteinander addieren kann? Schauen wir zuerst ein Beispiel dafür an:

```
    6789
+   5678
    1111
-------------
   12467
```

Um die beiden Zahlen zu addieren, sollen wir sie im ersten Schritt untereinander schreiben und ziffernweise von rechts nach links addieren(möglicherweise noch Überträge, schreiben wir dann Übertrag eine Stelle davor).

Satz 3. *Laufzeit von der Addition, die n Grundoperationen braucht, ist* $\mathcal{O}(n)$.

Beweis. Um die Analyse zu vereinfachen, nehmen wir an: Wir haben zwei Zahlen mit Länge n. (Eventuell ist die Länge einer Zahl kleiner als n. Aber wir können einfach 0's vor der Zahl hinzufügen, damit die Länge auch $n$ist. Das gilt auch für spätere Analysen). Da wir alle Ziffer von rechts nach links einmal durchgehen und miteinander addieren sollen, entspricht die Länge n genau dann n ziffernweisen Additionen. Obwohl es im Prinzip maximal $(n+1)$ Spalten geben kann, bekommt man trotzdem die linkeste Ziffer ohne weitere Berechnung. Daraus folgt, dass die Laufzeit von Addition $\mathcal{O}(n)$ ist. □

2.2 Multiplikation einer Zahl mit einer Ziffer

Hier kann man die Schulmethode zur Multiplikation anwenden. Wir führen ziffernweise Multiplikationen von rechts nach links durch und schreiben die Zwischenergebnisse schräg untereinander hin. Anschließend addieren wir alle Ziffer, die in einer selben Spalte stehen, was uns das Teilprodukt liefert. Schauen wir mal folgendes Beispiel an:

```
      1234
*        6
----------
       24
      18
     12
+    6
----------
     7404
```

Zwischenschritte:

- Berechne alle Teilprodukte von ziffernweise jeweiligen Multiplikationen.
- Schreiben Teilprodukte von rechts nach links schräg auf.
- Addieren alle Ziffer in der selben Spalte.

Satz 4. *Die Multiplikation einer Zahl mit einer Ziffer benötigt 2n Grundoperationen. Die Laufzeit liegt also in* $\mathcal{O}(n)$

Beweis. Angenommen: a ist eine Zahl mit Länge n, b ist eine Ziffer. Jede Ziffer von a soll jeweils einmal mit b multiplizieren, was dann n Multiplikationen fordert. Außerdem sollen alle Teilprodukte addiert werden, was damit n Additionen entspricht. Auch wie zuvor erwähnt, kann es hier auch $(n+1)$ Spalten geben, aber die Argumentation hier ist etwas anders. Denn die rechteste Ziffer kann man direkt hinschreiben. Also werden tatsächlich n Additionen benötigt. D.h. Insgesamt werden $2n$ Grundoperationen durchgeführt. Als Laufzeit ergibt sich $\mathcal{O}(n)$. □

2.3 Grundschulmethode zur Multiplikation

Angenommen: Wir haben zwei Zahlen a und b, die beide aus n Ziffern bestehen. Beim Multiplizieren ist es ebenso wichtig, dass man ordentlich arbeitet und genügend Platz lässt. Es werden die Produkte nebeneinander geschrieben(a multipliziert mit jeder Ziffer von b). Beginnt mit der niedrigsten Stelle der rechten Zahl, dann kann man es anschließend leicht addieren. Ein Beispiel ist als folgendes gegeben:

```
      1234
*     5678
-----------
      9872
     8638
    7404
+  6170
-----------
   7006652
```

Zwischenschritte:

- 1234 multipliziert mit jeder Ziffer von 5678. Es werden die Teilprodukte nebeneinander geschrieben.
- Beginnt mit der niedrigsten Stelle und wiederum von rechts nach links durchgehen und miteinander addieren.

Satz 5. *Laufzeit von der Grunschulmethode ist* $\mathcal{O}(n^2)$.

Beweis. Jetzt konzentrieren wir uns auf die Anzahl der Grundoperationen. Wie viele Grundoperationen sind das? Betrachten wir zuerst die Multiplikation: Aus dem Beispiel sehen wir uns schon, dass 1234 jeweils mit jeder Ziffer von 5678 multipliziert, was genau die Multiplikation "Zahl mal Ziffer" ist, sodass man alle Teilprodukte errechnen kann. Entsprechend brauchen wir n Multiplikationen. Also werden dann $2n^2$ Grundoperationen zur Multiplikation durchgeführt. Anschließend werden wir alle Teilprodukte spaltenweise miteinander addieren. Zur Vereinfachung der Analyse fügen wir einfach 0'en hinzu, wo es in der Spalte mindestens eine Zahl gibt. Das Produkt von einer n stelligen Zahl mit einer Ziffer kann maximal $n+1$ Ziffern haben wegen des Übertrags. Da alle Teilprodukte schräg untereinander stehen und es insgesamt n Teilprodukte gibt, werden höchstens$(n+1+n-1) = 2n$ spaltenweise Additionen durchgeführt. Hier würde ich noch kurz erklären. $(n+1)$ bedeutet, dass wir höchstens $(n+1)$ Ziffern haben. Außerdem bedeutet $(n-1)$, dass es $(n-1)$ Stellen zwischen dem linkesten Teilprodukt und dem rechtesten Teilprodukt gibt. Darüber hinaus enthält eine Spalte n Ziffern. D.h. eine spaltenweise Addition benötigt $(n-1)$ ziffernweise Additionen . Nach der obigen Untersuchung über die Addition folgt es:
$\rightarrow (n-1) \cdot 2n = 2n^2 - 2n$ Grundoperationen
Also insgesamt: $4n^2 - 2n$ Grundoperationen.
$\Rightarrow Laufzeit : \mathcal{O}(n^2)$ □

Definition 6. Teile und Herrsche ist ein Algorithmenentwurfsparagdima, das auf einer mehrfach verzweigten Rekursion basiert. Der "Teile und Herrsche"Algorithmus arbeitet, in dem er ein Problem rekursiv in zwei oder mehrere Unterprobleme derselber oder verwandter Art zerlegt, bis diese einfach genug werden, um direkt gelöst werden zu können.

Hier wird noch eine andere Methode zur Multiplikation vorgestellt, die dieses Prinzip Divide and Conquer anwendet und auch hilfreich ist, mit der Idee davon den Karatsuba Algorithmus besser verstehen zu können.

2.4 Divide and Conquer

Angenommen, dass wir zwei Zahlen p und q haben mit Länge n. Wir definieren $m = \lfloor \frac{n}{2} \rfloor$. Dann können wir p und q so darstellen:

$$p = 10^m \cdot a + b$$
$$q = 10^m \cdot c + d$$
$$(wobei\ a,\ b,\ c,\ d \in \mathbb{Z})$$

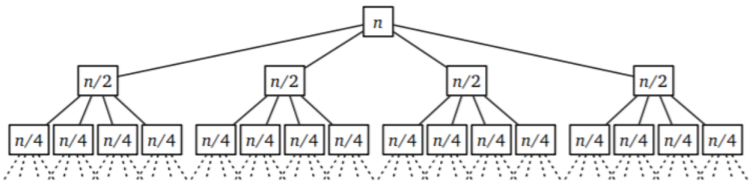
Der Grund dafür, warum wir so die Zahlen quasi in der Mittel aufteilen, ist, dass wir das Prinzip "Divide and Conquer" nutzen wollen. Die typische Lösung dafür ist dann, dass wir dieses Problem in 2 kleineren aber eher gleichmäßigen Unterprobleme zerlegen. So kann das Produkt von p und q durch Distributivitätsgesetz entfaltet.

$$p \cdot q = (10^m \cdot a + b) \cdot (10^m \cdot c + d) \quad (1)$$
$$= a \cdot c \cdot 10^{2m} + (a \cdot d + b \cdot c) \cdot 10^m + b \cdot d \quad (2)$$
$$= a \cdot c \cdot 10^n + (a \cdot d + b \cdot c) \cdot 10^{\frac{n}{2}} + b \cdot d \quad (3)$$

Beobachtung 7. *Aus der Formel (3) kann man leicht erkennen, dass es dringend gefordert ist, 4 unterschiedlichen Multiplikationen durchzuführen.*

Satz 8. *Laufzeit des einfachen divide and conquer Ansatzes ist $\mathcal{O}(n^2)$.*

Beweis. Von der Beobachtung 5 erfahren wir, dass wir dazu 4 Multiplikationen brauchen. Dann können wir die Rekursionsformel so formulieren:
$\mathrm{T}(n) = 4 \cdot \mathrm{T}(\lfloor \frac{n}{2} \rfloor) + \mathcal{O}(n)$
$\mathrm{T}(\frac{n}{2}) = 4 \cdot \mathrm{T}(\lfloor \frac{n}{4} \rfloor) + \mathcal{O}(\frac{n}{2})$
...
$\mathrm{T}(2) = 4 \cdot \mathrm{T}(1) + \mathcal{O}(1)$ (wobei $\mathrm{T}(1) = \mathcal{O}(1)$)
Um die Rekursionsformel zu lösen, können wir einen Rekursionsbaum aufbauen:



[**3**]

Die Tiefe des Baums ist gleich $log_2 n$.
Auf der **ersten** Ebene: 4^1 Rekursionen. Auf der **zweiten** Ebene: 4^2 Rekursionen, usw. Das heißt, auf der **letzten** Ebene gibt es dann $4^{\log_2 n}$ Rekursionen. Und die Laufzeit jeder Rekursion auf der letzten Ebene ist $T(1)$ *bzw.* $\mathcal{O}(1)$.
$\rightarrow$ Laufzeit: $4^{\log_2 n} \cdot \mathcal{O}(1) = \mathcal{O}(4^{\log_2 n}) = \mathcal{O}(\ n^{\log_2 4}) = \mathcal{O}(n^2)$

□

3 Karatsuba Algorithmus

Der **Karatusba Algorithmus**, der von Anatoli Alexejewitsch Karazuba im Jahr 1962 veröffentlicht wurde, ist besonders geeignet für die Multiplikation zweier großer ganzer Zahlen. Dieser Algorithmus, der asymptotisch schneller als der quadratische "Grundschule" Algorithumus ist, reduziert die Multiplikation von zwei n-stelligen Zahlen auf höchsten $n^{\log_2 3} \approx n^{1.585}$ einstellige Multiplikationen im Allgemein (und genau dann $n^{\log_2 3}$, wenn n eine Potenz von 2 ist)(Beweis und Analyse siehe unten).

Die Kernidee besteht darin, dass vier Unterprobleme bei der Multiplikation auf drei reduziert werden können. Daher beschleunigt sich die Berechnung.

Im Allgemein gibt es folgende zwei Fälle zu betrachten:
Fall 1: Multiplikation zweier einstelligen Zahlen($n = 1$).
Fall 2: Multiplikation zweier Zahlen mit Länge n, wobei n größer 1 ist.

Beobachtung 9. *Im ersten Fall kann man sofort Ergebnis kriegen, weil man dafür genau nur eine Grundoperation braucht, die vom Prozessor direkt untertützt ist.*

Angenommen, wir haben zwei Zahlen p und q mit Länge n. Wir definieren $m = \lfloor \frac{n}{2} \rfloor$. Dann können wir p und q so darstellen:

$$p = 10^m \cdot a + b$$
$$q = 10^m \cdot c + d$$
$$(wobei\ a,\ b,\ c,\ d \in \mathbb{Z})$$

Der Grund dafür, warum wir so die Zahlen quasi in der Mittel aufteilen, ist, dass wir das Prinzip "Divide and Conquer" nutzen wollen. Die typische Lösung dafür ist dann, dass wir dieses Problem in 2 kleineren aber eher gleichmäßigen Unterprobleme zerlegen.
Um p mit q zu multiplizieren, kann man $p \cdot q$ wie oben dargestellt, wie folgt ein wenig umformen:

$$\begin{aligned} p \cdot q &= (10^m \cdot a + b) \cdot (10^m \cdot c + d) && (4)\\ &= a \cdot c \cdot 10^{2m} + (a \cdot c + b \cdot d - (b-a) \cdot (d-c)) \cdot 10^m + b \cdot d && (5)\\ &= a \cdot c \cdot 10^n + (a \cdot c + b \cdot d - (b-a) \cdot (d-c)) \cdot 10^{\frac{n}{2}} + b \cdot d && (6) \end{aligned}$$

Aus der Formel 6 können wir bemerken, dass ein paar Terme gleich sind. Es ist deswegen ausreichend, dass wir solche Terme nur ein mal rechnen sollen. Wir definieren:

$$\begin{aligned} u &= a \cdot c \\ v &= (b-a) \cdot (d-c) \\ w &= b \cdot d \end{aligned}$$

Also wir können $p \cdot q$ wie folgt umschreiben:

$$p \cdot q = u \cdot 10^n + (u + w - v) \cdot 10^{\frac{n}{2}} + w$$

Daraus ist ersichtlich, dass wir dazu mithilfe des Karatsuba Algorithmus nur 3 Multiplikationen mit Länge $\lfloor \frac{n}{2} \rfloor$ brauchen. Im Vergleich zur Grundschulmethode können wir dann eine Multiplikation einsparen. Durch ein konkretes Beispiel wird es klarer: Wir wollen das Produkt von 12345 und 6789 berechnen. Seien a = 12, b = 345 und c = 6, d = 789, nämlich:

$$\begin{aligned} 12345 &= 12 \cdot 1000 + 345 \\ 6789 &= 6 \cdot 100 + 789 \end{aligned}$$

Dann sind Ergebnisse von u,v, w wie folgt

$$\begin{aligned} u &= a \cdot c = 12 \cdot 6 = 72 \\ v &= (b-a) \cdot (d-c) = (345-12) \cdot (789-6) = 260739 \\ w &= b \cdot d = 345 \cdot 789 = 272205 \end{aligned}$$

$\Rightarrow 12345 \cdot 6789 = u \cdot 10^6 + (u + w - v) \cdot 10^3 + v = 83810205$

Schauen wir uns den Pseudocode [**1**] von diesem Algorithums an:

```
1  procedure karatsuba(num1, num2)
2      if (num1 < 10) or (num2 < 10)
3          return num1 * num2
4
5      m = max(size_base10(num1), size_base10(num2))
6      m2 = floor(m / 2)
7
8      high1, low1 = split_at(num1, m2)
9      high2, low2 = split_at(num2, m2)
10
11     u = karatsuba(high1, high2)
```

```
12      v = karatsuba((high1-low1), (high2-low2))
13      w = karatsuba(low1, low2)
14
15      return (u * 10 ^ (m2 * 2)) + ((u + w - v) * 10 ^ m2) + w
```

Zuerst wird geprüft, ob mindestens eine von beiden Zahlen größer als 10 ist, also ob die Multiplikatoren nur eine Ziffer sind. Wenn ja, ist es doch genug effizient, die beiden Zahlen direkt miteinander zu multiplizieren. Darüber hinaus ist dies die Abbruchbedingung der Rekursion. Ansonsten wenden wir den Algorithmus an. In der 5. und 6. Zeile wird die maximale Länge der Zahlen berechnet und abgerundet. Danach werden die vorgegebenen Zahlen in die Form $p = 10^m \cdot a + b$ umgeformt, also die Ziffernfolgen in der Mitte aufgeteilt. In den Zeilen 11 - 13 werden 3 Karatsuba-Funktion rekursiv aufgerufen, die genau 3 Multiplikationen mit Länge $\lfloor \frac{n}{2} \rfloor$ entsprechen. Sobald u, v, w die kleinstmöglichen Einheiten haben, werden sie in die endgültigen Gleichung zurückgegeben und die Zahl mit einer Basis aufgefüllt(im obigen Pseudocode zur Basis 10). Am Ende kriegt man dann das Endergebnis.

Beobachtung 10. *Außer den Multiplikationen werden noch $\mathcal{O}(n)$ Additionen benötigt.*

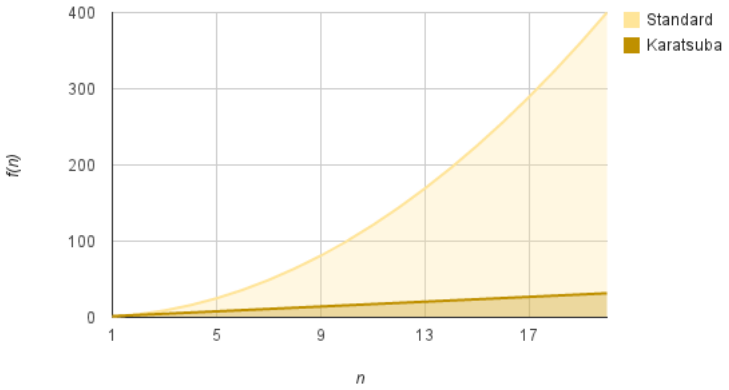
Satz 11. *Die Laufzeit von Karatsuba Algorithmus ist $\mathcal{O}(n^{\log_2 3})$.*

Beweis. **Initialisierung:** $\mathrm{T}(1) = 1$
$\mathrm{T(n)} = 3 \cdot \mathrm{T}(\frac{n}{2}) + \mathcal{O}(\mathrm{n})$
Angenommen: $\mathrm{n} = 2^k \Leftrightarrow \log_2 n$ und $f(k) = \mathrm{T}(2^k)$
$\Rightarrow f(k) = 3 \cdot f(k-1) + 2^k$
$\Leftrightarrow \frac{f(k)}{3^k} = \frac{f(k-1)}{3^{k-1}} + (\frac{2}{3})^k$
$\Rightarrow f(k) \leqslant 3^{k+1}$ (*) Beweis siehe unten

Dann können wir dies in die Definition von f einsetzen:
$T(n) = \mathrm{T}(2^k) = f(k) = f(\log_2 n) \leqslant 3^{\log 2n+1}$
$\Rightarrow T(n) = \mathcal{O}(3^{\log_2 n}) = \mathcal{O}(2^{\log_2 3 \cdot \log_2 n}) = \mathcal{O}(n^{\log_2 3})$

(*) $\frac{f(k)}{3^k} = \frac{f(k-1)}{3^{k-1}} + (\frac{2}{3})^k$
$\frac{f(k-1)}{3^{k-1}} = \frac{f(k-2)}{3^{k-2}} + (\frac{2}{3})^{k-1}$
...
$\frac{f(1)}{3^1} = \frac{f(1)}{3^0} + (\frac{2}{3})^1$
$\Rightarrow \frac{f(k)}{3^k}$ - $\frac{f(0)}{3^0} = \sum_{i=1}^{k} (\frac{2}{3})^i = \frac{\frac{2}{3} \cdot (1-\frac{2}{3})^k}{1-\frac{2}{3}} = 2 \cdot (\frac{1}{3})^k$
$\Leftrightarrow f(k) = 3^k \cdot (1 + 2 \cdot \frac{1}{3^k}) = 3^k + 2$
$\Rightarrow f(k) \leqslant 3^{k+1}$ □

Mit der Darstellung von Komplexität ist es nicht so deutlich und intuitiv.



[**2**]

Durch obiges Bild wird veranschaulicht, dass der Karatsuba Algorithmus mit der Laufzeit $\mathcal{O}(n^{\log_2 3})$ viel schneller als der naive Algorithmus mit der Laufzeit $\mathcal{O}(n^2)$ ist. Schauen wir uns ein konkretes Beispiel dafür an: Wir wollen zwei 1024 stelligen Zahlen miteinander multiplizieren (In der Tabelle [**1**] stehen die benötigten Anzahlen von Multiplikation).

Länge	Karatsuba	Grundschulmethode
$1024{=}2^{10}$	$3^{10} = 59049$	$(2^{10})^2 = 1048675$

Aus der Tabelle kann man leicht davon sehen, dass der Karatsuba-Algorithmus viel effizienter ist, da die Anzahl der benötigten Multiplikation bei 1024 Stellen nur ungefähr 5% von der Grundschulmethode ist.

4 Andere Methoden

Es gibt neuere und schnellere Methoden wie:

1) **Schönhage-Strassen-Algorithmus** in $\mathcal{O}(n \cdot \log n \cdot \log\log n)$ (FNTT)[**3**]
2) **Toom-Cook-Algorithmus** in $\mathcal{O}(n \cdot \log n \cdot 2^{\sqrt{log(n)}})$[**4**]

Die obigen Methoden werden hier leider nicht berücksichtigt, nur als Hinweise betrachtet.

5 Zusammenfassung

In der Ausarbeitung wurde der Karatsuba Algorithmus vorgestellt, der Multiplikation zweier Zahlen mit Länge n auf kleinere Unterprobleme zurückführt, was uns ermöglicht, dass nur 3 anstatt 4 Multiplikationen zur Berechnung benötigt werden. Das verwendete Prinzip ist **Divide and Conquer**(auf deutsch Teile und Herrsche). Wiederhole solange, bis wir direkt das Produkt berechnen können, also mit Länge = 1.

Literatur

[1] *https://en.wikipedia.org/wiki/Karatsuba_algorithm*

[2] *https://iq.opengenus.org/content/images/2018/05/Karatsuba − Complexity.png*

[3] *https://de.wikipedia.org/wiki/Sch%C3%B6nhage − Strassen − Algorithmus*

[4] *https://de.wikipedia.org/wiki/Toom − Cook − Algorithmus*