Karlsruher Institut für Technologie

Algorithmen 1

Teile und Herrsche
Karatsubas Algorithmus und Master-Theorem

Multiplikation: Idee → Code

Wie geht das nochmal?

$$
\begin{array}{r}
\overbrace{5\,6\,7\,8}^{a} \cdot \overbrace{1\,2\,3\,4}^{b} \\
\hline
5\,6\,7\,8 \\
1\,1\,3\,5\,6 \\
1\,7\,0\,3\,4 \\
+ \quad 2\,2\,7\,1\,2 \\
\hline
7\,0\,0\,6\,6\,5\,2
\end{array}
$$

- berechne $a \cdot b_i$ für jede Ziffer b_i von $b = (b_{n-1} \dots b_0)$
- verschiebe Ergebnis um i Stellen nach links
- addiere Ergebnisse

Hohe Abstraktionsebene

- leicht zu verstehen
- schwer zu implementieren

Konkretisierung mittels Pseudocode

mult(a, b) // $\Theta(n^2)$

  total $:= 0$ // $\Theta(1)$

  for i from 0 to $n-1$ **do** // $\Theta(n^2)$

    prod $:= a \cdot b_i \cdot 10^i$ // $\Theta(n)$

    total $:=$ total $+$ prod // $\Theta(n)$

  return total // $\Theta(n)$

$\boldsymbol{a \cdot b_i}$ // $\Theta(n)$

  carry $:= 0$ // $\Theta(1)$

  for j from 0 to $n-1$ **do** // $\Theta(n)$

    (carry, c_j) $:= a_j \cdot b_i +$ carry // $\Theta(1)$

  $c_n :=$ carry // $\Theta(1)$

  return $(c_n, \dots, c_0)$ // $\Theta(n)$

Einschub: Pseudocode

Nutzen

- interpoliert zwischen hoher und niedriger Abstraktionsebene
- viele mächtige Operationen $\rightarrow$ hohe Abstraktionsebene $\rightarrow$ Ideenvermittlung
- nur primitive Operationen $\rightarrow$ tiefe Abstraktionsebene $\rightarrow$ tatsächliche Implementierung
- kann helfen von der high-level Idee zu einer Implementierung zu kommen
- kann bei der Analyse (Laufzeit und Korrektheit) helfen

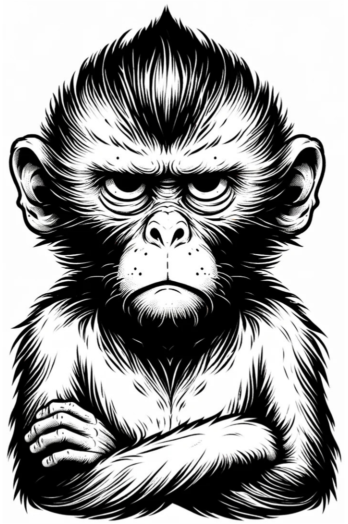
Regeln

- in der Vorlesung: Syntax wird immer on-the-fly miterklärt
- flexibler als bei richtiger Programmiersprache (z.B. mathematische Formeln)
- man kann auch etwas kreativ sein, sollte es aber konsistent halten
- Hauptziele:
  - gut lesbar (für Menschen)
  - richtige Abstraktionsebene

Wie schnell multipliziert ein Computer Zahlen?

Typische Annahme

- $\Theta(1)$ für die Multiplikation zweier Zahlen
- passt meist gut zur Realität
- macht die Analyse angenehm



Warum lernen wir dann hier etwas über Multiplikations-Algorithmen?
Warum erlauben wir uns pro Schritt nur Ziffern zu multiplizieren?

- schönes Beispiel für Laufzeitanalyse
- schönes Beispiel für Rekursion
- schönes Beispiel für Teile und Herrsche
- blödes Beispiel, weil ihr dann denkt Zahlen zu multiplizieren wäre teuer

Also: Der folgende Algo ist nicht relevant, außer ihr multipliziert sehr sehr große Zahlen. In der Vorlesung gilt sonst immer: multiplizieren in $\Theta(1)$.

Teile und Herrsche

Schriftliche Multiplikation

$$5678 \cdot 1234 = 5678 \cdot (1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0)$$

$+$

$5678 \cdot 1 \cdot 10^3$ $\quad 5678 \cdot 2 \cdot 10^2$ $\quad 5678 \cdot 3 \cdot 10^1$ $\quad 5678 \cdot 4 \cdot 10^0$

$+$

$5 \cdot 10^3 \cdot 2 \cdot 10^2$ $\quad 6 \cdot 10^2 \cdot 2 \cdot 10^2$ $\quad 7 \cdot 10^1 \cdot 2 \cdot 10^2$ $\quad 8 \cdot 10^0 \cdot 2 \cdot 10^2$

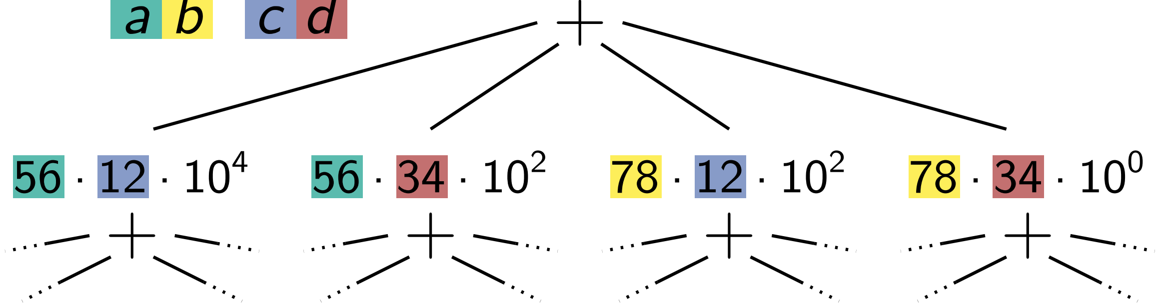
Idee im Kern

- spalte schwieriges Problem in mehrere leichtere Probleme
- wiederhole das, bis die Instanz trivial ist
- hier: Problem selbst wird substanziell leichter (Zahl $\rightarrow$ Ziffer)
- allgemein: Problem bleibt gleich, aber Instanz wird leichter (z.B. kleineres n)

Teile und Herrsche

Multiplikation mit anderer Zerlegung

$$\underset{a\ b}{5678} \cdot \underset{c\ d}{1234} = (56 \cdot 10^2 + 78) \cdot (12 \cdot 10^2 + 34)$$



mult(x, y)

// assumption: $n = |x| = |y| = 2^k$ for $k \in \mathbb{N}$

if $n = 1$ **return** $x \cdot y$

$a, b :=$ first and second half of x

$c, d :=$ first and second half of y

return mult$(a, c) \cdot 10^n +$ **mult**(b, d)

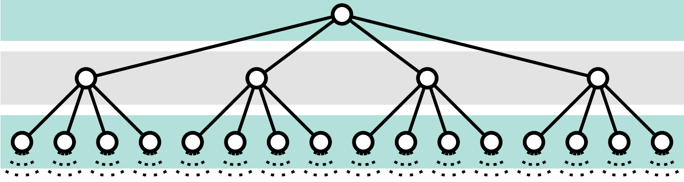
$+($**mult**$(a, d) +$ **mult**$(c, b)) \cdot 10^{n/2}$

Größe des Rekursionsbaums

- jeder Knoten hat 4 Kinder $\rightarrow$ Lage i hat 4^i Knoten
- n wird mit jeder Lage halbiert $\rightarrow$ $\log_2(n)$ Lagen
- Knoten insgesamt: $\sum_{i=0}^{\log_2(n)} 4^i \in \Theta\left(4^{\log_2(n)}\right) = \Theta\left(n^2\right)$
- Achtung: das heißt noch nicht, dass die Laufzeit auch $\Theta\left(n^2\right)$ ist (stimmt in diesem Fall aber)

Einschub: Exponentielle Summen & geometrische Reihen

Exponentiell wachsende Lagen



Lage i	Anzahl Knoten
0	$4^0 = n^2 \cdot 1/4^k$
1	$4^1 = n^2 \cdot 1/4^{k-1}$
2	$4^2 = n^2 \cdot 1/4^{k-2}$
$\vdots$	$\vdots$
$k-1$	$4^{k-1} = n^2 \cdot 1/4^1$
$k = \log_2(n)$	$4^k = n^2 \cdot 1/4^0$

Summe der Knoten

klein $\to$ groß

$$\sum_{i=0}^{k} 4^i$$

groß $\to$ klein

$$\sum_{i=0}^{k} n^2 \cdot \left(\tfrac{1}{4}\right)^i = n^2 \cdot \sum_{i=0}^{k} \left(\tfrac{1}{4}\right)^i$$

geometrische Summe $\in \Theta(1)$

$$\in \Theta(n^2)$$

Mentaler Shortcut: Summen, die exponentiell schrumpfen/wachsen

- $\sum_i c^i$ wird asymptotisch vom größten Summanden dominiert
- also: $\displaystyle\sum_{i=a}^{b} c^i \in \Theta\left(c^a + c^b\right)$

($c \in (0, 1) \cup (1, \infty)$ konstant)
($a, b > 0$ können von n abhängen)

Ziel: weniger rekursive Aufrufe

Karlsruher Institut für Technologie

Bisher

- $xy = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$
  $= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd$
- vier rekursive Multiplikationen

mult(x, y)

// assumption: $n = |x| = |y| = 2^k$ for $k \in \mathbb{N}$
if $n = 1$ **return** $x \cdot y$
$a, b :=$ first and second half of x
$c, d :=$ first and second half of y
return mult$(a, c) \cdot 10^n +$ **mult**(b, d)
$+($**mult**$(a, d) +$ **mult**$(c, b)) \cdot 10^{n/2}$

Idee zur Verbesserung

- ignoriere zunächst die 10er-Potenzen
- $(a + b) \cdot (c + d) = ac + ad + bc + bd$
- wenn wir $(a + b) \cdot (c + d)$, ac und bd kennen, können wir daraus $ad + bc$ erhalten
- berechne $(a + b) \cdot (c + d)$, ac und bd
- berechne dann $(ad + bc) = (a + b) \cdot (c + d) - ac - bd$
- nur drei rekursive Multiplikationen
- dafür etwas mehr Additionen

Karatsubas Algorithmus

mult(x, y)

```
// assumption: n = |x| = |y| = 2^k for k ∈ ℕ
if n = 1 return x · y
a, b := first and second half of x
c, d := first and second half of y
ac := mult(a, c)
bd := mult(b, d)
sum := mult((a + b), (c + d)) − ac − bd
return ac · 10^n + sum · 10^(n/2) + bd
```

Erinnerung

- $xy = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd$
- $(ad + bc) = (a + b) \cdot (c + d) - ac - bd$

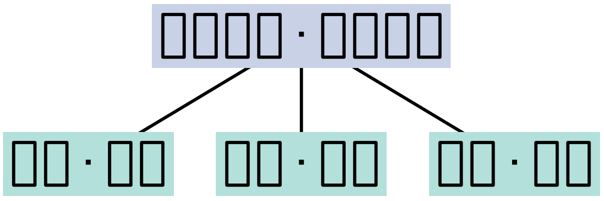
Anzahl Knoten im Rekursionsbaum

- 3 Kinder $\to 3^i$ Knoten auf Lage i
- n wird immer halbiert $\to \log_2(n)$ Lagen
- $\sum_{i=0}^{\log_2(n)} 3^i \in \Theta\left(3^{\log_2(n)}\right) = \Theta\left(n^{\log_2(3)}\right)$
  $\subseteq O\left(n^{1{,}585}\right)$

Offene Fragen

- Wie viel Arbeit macht man in jedem Knoten? Sind die zusätzlichen Additionen schlimm?
- Können wir annehmen, dass n eine Zweierpotenz ist? Könnte nicht $|a + b| > \frac{n}{2}$ sein?

Karatsubas Algorithmus – Laufzeitanalyse



Arbeit pro Knoten

- linear in der Größe der Zahlen
- also: $\Theta\left(n \cdot \frac{1}{2^i}\right)$ für einen Knoten auf Lage i

Arbeit pro Lage

- 3^i Knoten auf Lage i
- Gesamtarbeit für Lage i: $3^i \cdot \Theta\left(n \cdot \frac{1}{2^i}\right) = \Theta\left(n \cdot \left(\frac{3}{2}\right)^i\right)$

Arbeit insgesamt

- exponentielle Summe $\rightarrow$ dominiert durch letzte Lage $i = \log_2(n)$
- also: $\Theta\left(n \cdot \left(\frac{3}{2}\right)^{\log_2(n)}\right) = \Theta\left(n \cdot \left(\frac{n^{\log_2(3)}}{n}\right)\right) = \Theta\left(n^{\log_2(3)}\right)$
- Sanity Check: letzte Lage hat $n^{\log_2(3)}$ Knoten und konstanten Aufwand pro Knoten

Rundungsfehler?

Annahme: $n = |x| = |y|$ ist Zweierpotenz
- Problem: $a + b$ hat ggf. $\frac{n}{2} + 1$ Ziffern
- also: keine Zweiterpotenz mehr im rekursiven Aufruf

Umgang mit ungeradem n
- n ungerade $\Rightarrow |a| = \lceil \frac{n}{2} \rceil = \frac{n}{2} + \frac{1}{2}$
- damit hat $a + b$ bis zu $\frac{n}{2} + \frac{3}{2}$ Ziffern

Was ändert sich bei der Analyse?
- $n_i = $ #Ziffern auf Lage i ($n_0 = n$)
- bisher: $n_i = \frac{n}{2^i}$
- jetzt: $n_{i+1} \leq \frac{n_i}{2} + \frac{3}{2} \leq \frac{n}{2^i} + 3$
- kein asymptotischer Unterschied $\to$ Laufzeit $\Theta\left(n^{\log_2(3)}\right)$

mult(x, y)

...
$a, b :=$ first, second half of x
...
sum $:=$ **mult**$((a + b), (\ldots))\ldots$

Nebenrechnung

$$n_0 = n$$
$$n_1 = \frac{n}{2} + \frac{3}{2} \qquad \left(\cdot\tfrac{1}{2} + \tfrac{3}{2}\right)$$
$$n_2 = \frac{n}{2^2} + \frac{3}{2^2} + \frac{3}{2} \qquad \left(\cdot\tfrac{1}{2} + \tfrac{3}{2}\right)$$
$$n_3 = \frac{n}{2^3} + \frac{3}{2^3} + \frac{3}{2^2} + \frac{3}{2} \qquad \left(\cdot\tfrac{1}{2} + \tfrac{3}{2}\right)$$
$$n_i = \frac{n}{2^i} + 3 \cdot \sum_{j=1}^{i} \frac{1}{2^j} \leq \frac{n}{2^i} + 3$$

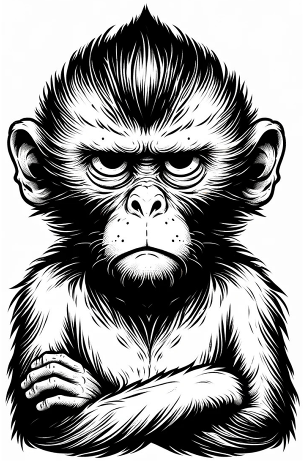
Achtung: Abbruch der Rekursion müsste man hier nochmal genauer anschauen.
Möglichkeit 1: Zeige, dass es für konstant viele Ziffern nach konstanter Zeit terminiert.
Möglichkeit 2: Brich die Rekursion nicht erst bei $n = 1$, sondern etwas früher ab.

Karatsubas Algorithmus

Theorem
Karatsubas Algo. multipliziert zwei n-Ziffern Zahlen in $\Theta\left(n^{\log_2(3)}\right) \subseteq O\left(n^{1{,}585}\right)$ Zeit.

Analyse via Rekursionsbaum
- Wie viele Knoten sind auf Lage i?
- Wie groß ist das n auf Lage i?
- Wie viel Zeit kostet ein Knoten in Lage i?

mult(x, y)
- **if** $n = 1$ **return** $x \cdot y$
- $a, b :=$ first and second half of x
- $c, d :=$ first and second half of y
- $ac :=$ **mult**(a, c)
- $bd :=$ **mult**(b, d)
- $\text{sum} :=$ **mult**$((a + b), (c + d)) - ac - bd$
- **return** $ac \cdot 10^n + \text{sum} \cdot 10^{n/2} + bd$



Warum ist das so viel Arbeit?
Muss ich das für jeden rekursiven Algorithmus so analysieren?
Kann man da nicht irgendwie abkürzen?

Rekurrenzen

Karlsruher Institut für Technologie

Laufzeit als Formel

Laufzeit für Basisfall → Kosten in den Blättern

$$T(n) = \begin{cases} \Theta(1) & \text{wenn } n = 1, \\ 3 \cdot T\left(\frac{n}{2}\right) + \Theta(n) & \text{wenn } n > 1. \end{cases}$$

Anzahl rekursiver Aufrufe → Anzahl Kinder im Baum

Laufzeit für den Aufruf selbst → Kosten für einen Knoten

neue Instanzgrößen → Änderung von n zwischen Lagen

$$\Rightarrow T(n) \in \Theta\left(n^{\log_2(3)}\right)$$

mult(x, y)

- **if** $n = 1$ **return** $x \cdot y$
- $a, b :=$ first and second half of x
- $c, d :=$ first and second half of y
- $ac :=$ **mult**(a, c)
- $bd :=$ **mult**(b, d)
- sum $:=$ **mult**$((a + b), (c + d)) - ac - bd$
- **return** $ac \cdot 10^n + \text{sum} \cdot 10^{n/2} + bd$

Können wir das verallgemeinern?

$$T(n) = \begin{cases} \Theta(1) & \text{wenn } n = 1, \\ a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c) & \text{wenn } n > 1. \end{cases}$$

$$\Rightarrow T(n) \in \Theta(??)$$

Analyse via Rekursionsbaum

- Wie viele Knoten sind auf Lage i?
- Wie groß ist das n auf Lage i?
- Wie viel Zeit kostet ein Knoten in Lage i?

Lösen allgemeiner Rekurrenzen

Können wir das verallgemeinern?

$$T(n) = \begin{cases} \Theta(1) & \text{wenn } n = 1, \\ a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c) & \text{wenn } n > 1. \end{cases}$$

Analyse via Rekursionsbaum

- Wie viele Knoten sind auf Lage i?
- Wie groß ist das n auf Lage i?
- Wie viel Zeit kostet ein Knoten in Lage i?

Situation für Lage *i*

- #Knoten: a^i
- Instanzgröße: $\frac{n}{b^i}$
- Kosten pro Knoten: $\frac{n^c}{b^{ci}}$

Gesamtkosten für Lage *i*

$$a^i \cdot \frac{n^c}{b^{ci}} = \left(\frac{a}{b^c}\right)^i \cdot n^c$$

Anzahl Lagen

$$\log_b(n)$$

Gesamtkosten für den Baum

$$\sum_i \left(\frac{a}{b^c}\right)^i \cdot n^c$$

exponentielle Summe
$\rightarrow$ dominiert von größtem Summanden
(außer wenn $a/b^c = 1$)

$a < b^c$	$a = b^c$	$a > b^c$
$T(n) \in \Theta(n^c)$	$T(n) \in \Theta(n^c \log n)$	$T(n) \in \Theta\left(n^{\log_b(a)}\right)$
(Kosten der Wurzel dominiert)	(Kosten auf jeder der $\log_b(n)$ Lagen gleich)	(Kosten der $n^{\log_b(a)}$ Blätter dominiert)

Master-Theorem

Theorem
Sei $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ mit $f(n) \in \Theta(n^c)$ und $T(1) \in \Theta(1)$. Dann gilt

$$T(n) \in \begin{cases} \Theta\left(n^c\right) & \text{wenn } a < b^c, \\ \Theta\left(n^c \log n\right) & \text{wenn } a = b^c, \\ \Theta\left(n^{\log_b(a)}\right) & \text{wenn } a > b^c. \end{cases}$$

Ein Name, viele Theoreme

- es gibt viele Varianten des Master-Theorems
- Einschränkung $f(n) \in \Theta(n^c)$ kann man aufweichen
- asymmetrische Verzweigung: z.B. $T(n) = T(\frac{n}{2}) + 2T(\frac{n}{4})$
- Runden auf ganze Zahlen

aktuelles Papier zum Thema:
William Kuszmaul, Charles E. Leiserson
Floors and Ceilings in Divide-and-Conquer Recurrences
Symposium on Simplicity in Algorithms (SOSA 2021)

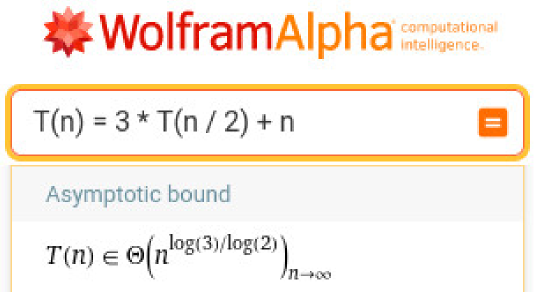
Zusammenfassung

Karatsubas Algorithmus [**Karatsuba**; 1962]

- algorithmische Technik: Teile und Herrsche
- Multiplikation zweier n-Ziffern Zahlen in $\Theta\left(n^{\log_2(3)}\right) \subseteq O\left(n^{1{,}585}\right)$ Zeit
- noch schneller:
  - $\Theta\left(n^{\log_3(5)}\right) \subseteq O\left(n^{1{,}465}\right)$ [**Toom, Cook**; 1963–1966]
  - $O\left(n \cdot \log n \cdot \log\log n\right)$ [**Schönhage, Strassen**; 1971]
  - $O\left(n \cdot \log n \cdot 2^{O(\log^* n)}\right)$ [**Fürer**; 2007]

Auflösung von Rekurrenzen

- Analyse des Rekursionsbaums + exponentielle Summen $\to$ Master-Theorem
- Master-Theorem funktioniert nicht immer (z.B. $T(n) = 2T(n-1)$)
  - weitere Techniken: branching vector, vollständige Induktion
  - nützlich: WolframAlpha



Pseudocode

Erinnerung

meldet euch bis **Freitag 12 Uhr** für die Tutorien an
(Infos auf der Homepage)

kommt in den Discord-Server
(Link im Ilias)