

Divide-and-Conquer / Teile und Herrsche

Wichtige Methode für den Entwurf von Algorithmen

Allgemeiner Ansatz:

0. Wenn Problem klein, löse es direkt
1. Zerlege Problem geeignet in Teilprobleme *divide*
2. Löse jedes Teilproblem rekursiv *conquer*
3. Vereinige die Teillösungen zu einer Gesamtlösung *„marry“*

Divide-and-Conquer / Teile und Herrsche

Wichtige Methode für den Entwurf von Algorithmen

Allgemeiner Ansatz:

0. Wenn Problem klein, löse es direkt
1. Zerlege Problem geeignet in Teilprobleme *divide*
2. Löse jedes Teilproblem rekursiv *conquer*
3. Vereinige die Teillösungen zu einer Gesamtlösung *„marry“*

Beispiel: MergeSort(L : Liste von Zahlen)

0. if $|L| \leq 1$ then return L
1. Teile L in Teillisten L_1 und L_2 möglichst gleicher Größe ($\sim n/2$) *divide*
2. $E_1 = \text{MergeSort}(L_1)$ $E_2 = \text{MergeSort}(L_2)$ *conquer*
3. $E = \text{Merge}(E_1, E_2)$ *„marry“*
return E

Divide-and-Conquer Laufzeitanalyse

$T(n)$ worst case Laufzeit von MergeSort wenn $|L| = n$

Beispiel: MergeSort(L : Liste von Zahlen)

- | | | |
|----|---|------------------|
| 0. | if $ L \leq 1$ then return L | $O(1)$ |
| 1. | Teile L in Teillisten L_1 und L_2 möglichst gleicher Größe ($\sim n/2$) | $O(n)$ |
| 2. | $E_1 = \text{MergeSort}(L_1)$ $E_2 = \text{MergeSort}(L_2)$ | $2 \cdot T(n/2)$ |
| 3. | $E = \text{Merge}(E_1, E_2)$
return E | $O(n)$ |

Divide-and-Conquer Recurrence

$$T(n) \leq d \quad \text{wenn } n \leq 1$$

$$T(n) \leq \beta n + 2 \cdot T(n/2) \quad \text{sonst}$$

Lösung: $T(n) = O(n \cdot \log n)$

Divide-and-Conquer Recurrences (“Mastertheorem”)

$a > 0, b > 1, c > 0$

$$T(n) \leq d \quad \text{wenn } n \leq 1$$

$$T(n) \leq a \cdot T(n/b) + \beta \cdot n^c \quad \text{sonst}$$

Lösung: $T(n) = O(n^c)$ wenn $a < b^c$
 $T(n) = O(n^c \cdot \log n)$ wenn $a = b^c$
 $T(n) = O(n^\lambda)$ mit $\lambda = \log_b a$ wenn $a > b^c$

Divide-and-Conquer Recurrences 2 ("Mastertheorem")

$a > 0, b > 1, c > 0, k \geq 0$

$$T(n) \leq d \quad \text{wenn } n \leq 1$$

$$T(n) \leq a \cdot T(n/b) + \beta \cdot n^c \cdot \log^k n \quad \text{sonst}$$

Lösung: $T(n) = O(n^c \cdot \log^k n)$ wenn $a < b^c$
 $T(n) = O(n^c \cdot \log^{k+1} n)$ wenn $a = b^c$
 $T(n) = O(n^\lambda)$ mit $\lambda = \log_b a$ wenn $a > b^c$

Beispiel: Polynommultiplikation

Polynome vom Grad n :

$$p(x) = p_0 + p_1x^1 + p_2x^2 + \dots + p_nx^n \quad \text{dargestellt durch Feld } P[0..n] = \langle p_0, p_1, p_2, \dots, p_n \rangle$$

$$q(x) = q_0 + q_1x^1 + q_2x^2 + \dots + q_nx^n \quad \text{dargestellt durch Feld } Q[0..n] = \langle q_0, q_1, q_2, \dots, q_n \rangle$$

Summenpolynom $(p+q)$ und Differenzpolynom $(p-q)$ lassen sich offensichtlich in $O(n)$ Zeit berechnen (addiere, bzw. subtrahiere die entsprechenden Koeffizienten).

Produktpolynom $r(x) = p(x) \cdot q(x)$ dargestellt durch Feld $R[0..2n] = \langle r_0, r_1, r_2, \dots, r_{2n} \rangle$

Lässt sich naiv in $O(n^2)$ Zeit berechnen:

```
for i = 0 to 2n do R[i] = 0
for i = 0 to n do
  for j = 0 to n do
    R[i+j] += P[i]·Q[j]
```

Schneller mit Divide-and-Conquer?

Beispiel: Polynommultiplikation via Divide-and-Conquer, einfältig

$$\begin{aligned}
 p(x) = p_0 + p_1x^1 + p_2x^2 + \dots + p_nx^n &= p_0 + \dots + p_{n/2-1}x^{n/2-1} + x^{n/2} [p_{n/2} + \dots + p_nx^{n/2}] \\
 &= p'(x) + x^{n/2} p''(x) \quad \text{grad}(p') = \text{grad}(p'') = n/2
 \end{aligned}$$

Analog

$$q(x) = q_0 + q_1x^1 + q_2x^2 + \dots + q_nx^n = q'(x) + x^{n/2} q''(x) \quad \text{grad}(q') = \text{grad}(q'') = n/2$$

Produktpolynom

$$r(x) = p(x) \cdot q(x) = p'(x) \cdot q'(x) + x^{n/2} [p'(x) \cdot q''(x) + p''(x) \cdot q'(x)] + x^n [p''(x) \cdot q''(x)]$$

Eine Multiplikation von Grad- n -Polynomen wird zurückgeführt auf 4 Multiplikationen von Grad- $(n/2)$ Polynomen plus ein paar Additionen und „Shifts“ (Multiplikation mit x^k)

Laufzeit: $T(n) = O(n) + 4 \cdot T(n/2)$
 Mastertheorem $\Rightarrow T(n) = O(n^2)$, keine Verbesserung!
 ($a=4, b=2, c=1$)

Beispiel: Polynommultiplikation via Divide-and-Conquer, schlau (Karatsuba-Ofman)

$$\begin{aligned}
 p(x) = p_0 + p_1x^1 + p_2x^2 + \dots + p_nx^n &= p'(x) + x^{n/2} p''(x) \quad \text{grad}(p') = \text{grad}(p'') = n/2 \\
 q(x) = q_0 + q_1x^1 + q_2x^2 + \dots + q_nx^n &= q'(x) + x^{n/2} q''(x) \quad \text{grad}(q') = \text{grad}(q'') = n/2
 \end{aligned}$$

Berechne $U = (p' + p'') \cdot (q' + q'')$ $V = p' \cdot q'$ $W = p'' \cdot q''$

$$\begin{aligned}
 r(x) = p(x) \cdot q(x) &= p'(x) \cdot q'(x) + x^{n/2} [p'(x) \cdot q''(x) + p''(x) \cdot q'(x)] + x^n [p''(x) \cdot q''(x)] \\
 &= V + x^{n/2} [U - V - W] + x^n [W]
 \end{aligned}$$

Eine Multiplikation von Grad- n -Polynomen wird zurückgeführt auf 3 Multiplikationen von Grad- $(n/2)$ Polynomen plus ein paar Additionen und „Shifts“ (Multiplikation mit x^k)

Laufzeit: $T(n) = O(n) + 3 \cdot T(n/2)$
 Mastertheorem $\Rightarrow T(n) = O(n^{1.59})$, klare Verbesserung!
 ($a=3, b=2, c=1$)

Weitere Verbesserungen sind möglich.

Multiplikation und Division von Polynomen, Ausblick

Satz:

$A(x)$, $B(x)$ Polynome vom Grad höchstens n

Produkt $P(x) = A(x) \cdot B(x)$

Quotient $Q(x) = A(x) \text{ div } B(x)$

Rest $R(x) = A(x) \text{ mod } B(x)$

können in Zeit $O(n \cdot \log n)$ berechnet werden

$Q(x)$ und $R(x)$ definiert durch eindeutige Darstellung von $A(x)$ als

$$A(x) = B(x) \cdot Q(x) + R(x)$$

mit $\text{grad}(R) < \text{grad}(B)$

Mit Hilfe der schnellen Diskreten Fourier Transformation (eine Divide-and-Conquer Methode) sowie iterativer Approximation

Mehrfach-Auswertung eines Polynoms für viele Argumente

Geg.: Polynom $f(x)$ vom Grad n sowie n Zahlen a_1, \dots, a_n

Ges.: $f(a_1), \dots, f(a_n)$

Naive Lösung benötigt $O(n^2)$ Zeit.

Lemma:

Sei $R(x) = A(x) \text{ mod } B(x)$ und a eine Nullstelle von B , also $B(a)=0$.

Dann gilt $A(a) = R(a)$.

Folgt direkt aus Definition von $Q(x) = A(x) \text{ div } B(x)$ und $R(x) = A(x) \text{ mod } B(x)$

definiert durch eindeutige Darstellung von $A(x)$ als

$$A(x) = B(x) \cdot Q(x) + R(x)$$

mit $\text{grad}(R) < \text{grad}(B)$

Divide-and-Conquer Algorithmus zur Mehrfachauswertung

Teile $\{a_1, \dots, a_n\}$ in $A_1 = \{a_1, \dots, a_{n/2}\}$ und $A_2 = \{a_{n/2+1}, \dots, a_n\}$

Berechne $B_1(x) = \prod_{a \in A_1} (x-a)$ $B_2(x) = \prod_{a \in A_2} (x-a)$ $O(n \cdot \log^2 n)$

Berechne $f_1 = f \text{ mod } B_1$ und $f_2 = f \text{ mod } B_2$ $O(n \cdot \log n)$

Löse rekursiv die Simultanauswertung von f_1 für A_1 $T(n/2)$

Löse rekursiv die Simultanauswertung von f_2 für A_2 $T(n/2)$

Beachte: f_1 und f_2 haben Grad $< n/2$

Divide-and-Conquer Algorithmus zur Mehrfachauswertung

Teile $\{a_1, \dots, a_n\}$ in $A1 = \{a_1, \dots, a_{n/2}\}$ und $A2 = \{a_{n/2+1}, \dots, a_n\}$

Berechne $B_1(x) = \prod_{a \in A1} (x-a)$ $B_2(x) = \prod_{a \in A2} (x-a)$ $O(n \cdot \log^2 n)$

Berechne $f1 = f \bmod B1$ und $f2 = f \bmod B2$ $O(n \cdot \log n)$

Löse rekursiv die Simultanauswertung von $f1$ für $A1$ $T(n/2)$

Löse rekursiv die Simultanauswertung von $f2$ für $A2$ $T(n/2)$

Beachte: $f1$ und $f2$ haben Grad $< n/2$

Laufzeit: $T(n) = 2T(n/2) + O(n \cdot \log^2 n)$

Aus dem Mastertheorem ergibt sich $T(n) = O(n \cdot \log^3 n)$

Verbesserung der Laufzeit auf $O(n \cdot \log^2 n)$ ist möglich.