Effiziente Algorithmen und Datenstrukturen I

Kapitel 10: Lineare Algebra

Christian Scheideler

WS 2008

Überblick

- Notation
- Arithmetik auf großen Zahlen (Addition und Multiplikation)
- Effiziente Matrix-Vektor-Multiplikation
- Effiziente Matrixmultiplikation (4-Russen Algo und Strassens Algo)
- Transitive Hülle

Notation

- Ganze Zahl: Ziffernfolge zur Basis B
- “Zur Basis B”: Ziffern aus [B] = {0,…,B-1}
- Zahl $a=(a_{n-1}\ldots a_0)_B$ zur Basis B hat den Wert

$$\sum_{i=0}^{n-1} a_i B^i$$

Wichtige Beispiele:

- B=2: Binärzahlen ($(101)_2 = 1\cdot 2^2 + 1\cdot 2^0 = 5$)
- B=10: Dezimalzahlen
- B=16: Hexadezimalzahlen ($(A)_{16} = 10$)
  Ziffern: $\{0,\ldots,9\} \cup \{A,B,C,D,E,F\}$

Notation

Elementare Operationen:

Seien $a,b,c,x,y \in [B]$.

- $(x\ y)_B := a+b+c$
- $(x\ y)_B := a \cdot b$

Beispiele für $B=10$:

- $(12)_{10} = 5+6+1$
- $(42)_{10} = 6 \cdot 7$

Basis B beliebig aber fest: lassen wir weg

Addition

- Gegeben: zwei Zahlen $a=(a_{n-1}\ldots a_0)$ und $b=(b_{n-1}\ldots b_0)$
- Gesucht: $s=(s_n\ldots s_0)$ mit $s=a+b$

Schulmethode:

	a_{n-1}	…	a_1	a_0	
	b_{n-1}	…	b_1	b_0	
Übertrag	c_n c_{n-1}	…	c_1	c_0	$\in \{0,1\}$
	s_n	s_{n-1}	…	s_1	s_0

Addition

Es gilt:

- $c_0 = 0$
- $(c_{i+1}\ s_i) := a_i + b_i + c_i$ für alle $i >= 0$
- $s_n = c_n$

Algorithmus:

- $c := 0$
- for $i := 0$ to $n-1$ do
  $(c\ s_i) := a_i + b_i + c$ Elementaroperation
- $s_n := c$

Addition

Theorem 10.1: Die Addition zweier Zahlen der Länge n benötigt höchstens n Elementaradditionen

Beobachtung: Jede Addition n-stelliger Zahlen a und b benötigt mindestens n Elementaradditionen, d.h. unser Algo ist optimal.

Multiplikation

- Gegeben: zwei Zahlen $a=(a_{n-1}\ldots a_0)$ und $b=(b_{n-1}\ldots b_0)$
- Gesucht: $s=(s_{2(n-1)}\ldots s_0)$ mit $s = a \cdot b$

Schulmethode:

- berechne $a \cdot b_j$ für alle $j \geq 0$
- addiere Ergebnisse versetzt zusammen

Multiplikation

$(a_{n-1} \ldots a_0) \cdot b_j$ $\xrightarrow{(c_i\ d_i)=a_i \cdot b_j}$

$c_{n-1}\ c_{n-2} \ldots c_0\ \ 0$

$+ \quad d_{n-1} \ldots d_1\ d_0$

Übertrag $e_n\ \ e_{n-1} \ldots e_1\ e_0$

$p_n\ p_{n-1} \ldots \ p_1\ p_0$

Algorithmus:

$e:=0;\ c_{-1}:=0$

for $i:=0$ to $n-1$ do

$(c_i\ d_i):=a_i \cdot b_j;\ (e\ p_i):=c_{i-1}+d_i+e$

$p_n:=c_{n-1}+e$

Multiplikation

Lemma 10.2: Wir können eine Zahl der Länge n mit einer Ziffer in $2n$ primitiven Operationen multiplizieren.

Sei $p_j := a \cdot b_j$ mit $p_j = (p_{j,n} \ldots p_{j,0})$.

Dann müssen wir noch

$$p := \sum_{j=0}^{n-1} p_j B^j$$

berechnen.

Multiplikation

Bildlich: $p_{0,n} \cdots p_{0,2}\ p_{0,1}\ p_{0,0}$

$+ \quad p_{1,n}\ p_{1,n-1} \cdots p_{1,1}\ p_{1,0}$

… ……………………………

$+ \quad p_{n-1,n} \cdots\ p_{n-1,1}\ p_{n-1,0}$

$p_{2n-1} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ p_2 \quad p_1 \quad p_0$

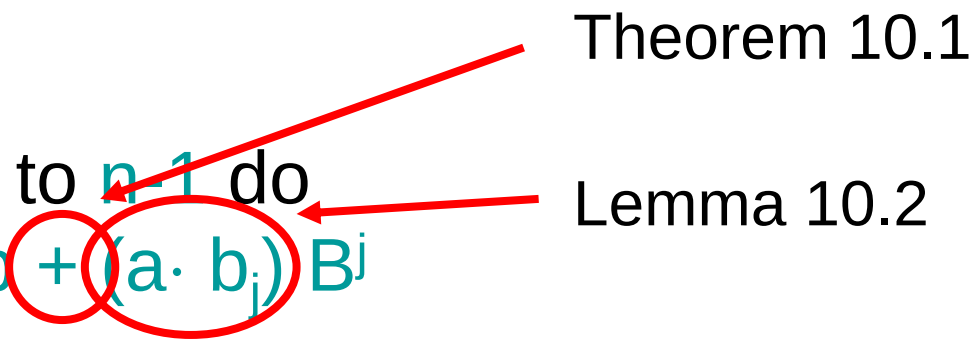
Multiplikation

Algorithmus:

p:=0

For j:=0 to n-1 do

p:=p + (a· b_j) B^j



Anzahl Elementaroperationen maximal

- 2n wegen Theorem 10.1, insgesamt $2n^2$
- 2n wegen Lemma 10.2, insgesamt $2n^2$

Also insgesamt max. $4n^2$ Elementaroperationen

Multiplikation

Mit einer besseren Abschätzung für die Additionen kann gezeigt werden:

Theorem 10.3: Die Schulmethode multipliziert zwei Zahlen der Länge n mit höchstens $3n^2+n$ primitiven Operationen.

Können wir besser werden?

Rekursive Multiplikation

Rekursive Version der Schulmethode:

Gegeben: zwei n-stellige Zahlen a, b zur Basis B, n gerade

Sei $a=a_1 \cdot B^k + a_0$ und $b=b_1 \cdot B^k + b_0$ mit $k=n/2$

Dann gilt:

$$a \cdot b = (a_1 \cdot B^k + a_0) \cdot (b_1 \cdot B^k + b_0)$$
$$= a_1 \cdot b_1 B^{2k} + (a_1 \cdot b_0 + a_0 \cdot b_1) B^k + a_0 \cdot b_0$$

4 rekursive Aufrufe zur Multiplikation $n/2$-stelliger Zahlen

Rekursive Multiplikation

Annahme: $|a|=|b|=n=2^c$ für ein $c \in IN$

Algorithmus Produkt(a,b):

if $|a|=|b|=1$ then return $a \cdot b$

else

$k:=|a|/2$

return Produkt(a_1,b_1)$\cdot B^{2k}$ + (Produkt(a_1,b_0) + Produkt(a_0,b_1))B^k + Produkt(a_0,b_0)

Rekursive Multiplikation

Anzahl Elementaroperationen ($\cdot$, +):

$$T(n) = \begin{cases} 1 & n=1 \\ 4T(n/2)+3(2n) & n>1 \end{cases}$$

Daraus ergibt sich $T(n) < 7n^2 + 6n$

Beweis: durch Induktion

Karatsuba Multiplikation

Karatsubas Idee:

$$a \cdot b = (a_1 \cdot B^k + a_0) \cdot (b_1 \cdot B^k + b_0)$$
$$= a_1 b_1 B^{2k} + (a_1 b_0 + a_0 b_1) B^k + a_0 b_0$$
$$= a_1 b_1 B^{2k} + ((a_1 + a_0)(b_1 + b_0) - (a_1 b_1 + a_0 b_0)) \cdot B^k + a_0 b_0$$

Nur noch drei rekursive Aufrufe für

$a_1 b_1$, $a_0 b_0$, $(a_1 + a_0)(b_1 + b_0)$

Karatsuba Multiplikation

Algorithmus Karatsuba(a,b):

if $|a|=|b|=1$ then return $a \cdot b$

else

$k:=|a|/2$

$p_1:=$Karatsuba(a_1,b_1)

$p_2:=$Karatsuba(a_1+a_0,b_1+b_0)

$p_3:=$Karatsuba(a_0,b_0)

return $p_1 \cdot B^{2k} + (p_2-(p_1+p_3))B^k + p_3$

Karatuba Multiplikation

Anzahl Elementaroperationen:

$$T(n) = \begin{cases} 1 & n=1 \\ 3T(n/2+1)+6(2n) & n>1 \end{cases}$$

Daraus ergibt sich $T(n) \sim n^{\log 3} = n^{1,58\ldots}$

Problem: Karatsuba erst für $n>1.000.000$ besser als Schulmethode

Bester bekannter Algo für Multiplikation:
Laufzeit $O(n \log n \log\log n)$

Matrix-Vektor-Multiplikation

Seien $m,n \in \mathbb{N}$. Eine $m \times n$-Matrix A hat die Form

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,n} \end{pmatrix}$$

Für ein $x=(x_i)_{1 \leqslant i \leqslant n}$ ist $y = A \cdot x$ definiert als

$$y_i = \sum_{j=1}^{n} a_{i,j}\, x_i \quad \text{für alle } 1 \leqslant i \leqslant m$$

Matrix-Vektor-Multiplikation

Schulmethode:

- Berechne $y_i = \sum_{j=1}^{n} a_{i,j} x_i$ für alle $1 \leqslant i \leqslant m$

Laufzeit: $O(n \cdot m)$ (falls $+$ und $\cdot$ Einheits-kosten haben)

Bessere Laufzeit möglich für festes A und beliebiges $x \in IR^n$, falls alle $a_{i,j} \in [B]$ für ein festes und nicht zu großes B sind.

Matrix-Vektor-Multiplikation

- Wir betrachten zunächst eine $m \times n$-Matrix A mit $a_{i,j} \in \{0,1\}$ für alle i,j und $m = \log_2 n$.
- Definiere U_n als die $m \times n$-Matrix, deren i-te Spalte die Binärdarstellung der Zahl $i-1$ repräsentiert (von oben nach unten)
- Beispiel für $n=8$:

$$U_8 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Matrix-Vektor-Multiplikation

- Sei $U^{(i)}$ die i-te Spalte von U_n und $A^{(j)}$ die j-te Spalte von A.
- Definiere $P=(p_{i,j})$ als die $n \times n$-Matrix mit $p_{i,j} = \delta(U^{(i)}, A^{(j)})$, wobei δ die Kronecker Funktion ist mit $\delta(U^{(i)}, A^{(j)}) = 1 \Leftrightarrow U^{(i)} = A^{(j)}$ und 0 sonst.
- Es gilt:

$$\begin{aligned}(U_n \cdot P)_{i,j} &= \sum_{k=1}^{n} u_{i,k} \cdot p_{k,j} \\ &= \sum_{k=1}^{n} U_i^{(k)} \, \delta(U^{(k)}, A^{(j)}) \\ &= A_i^{(j)} = a_{i,j}\end{aligned}$$

- D.h. $A \cdot x = (U_n \cdot P) \cdot x = U_n \cdot (P \cdot x)$

Matrix-Vektor-Multiplikation

- Bei Darstellung von P als Folge von n Spaltenpositionen, in denen P eine 1 hat (P hat nur eine 1 pro Spalte!), kann $y:=P\cdot x$ in $O(n)$ Zeit berechnet werden.
- Wir müssen noch $U_n \cdot y$ berechnen. Dazu verwenden wir eine rekursive Def. von U_n

$$U_1 = (\,0 \quad 1\,) \qquad U_n = \left(\begin{array}{c|c} \mathbf{O}^T_{n/2} & \mathbf{1}^T_{n/2} \\ \hline U_{n/2} & U_{n/2} \end{array}\right)$$

Matrix-Vektor-Multiplikation

- $U_n \cdot y$ ergibt dann

$$\left(\begin{array}{c|c} \mathbf{O}^T_{n/2} & \mathbf{1}^T_{n/2} \\ \hline U_{n/2} & U_{n/2} \end{array}\right) \left(\begin{array}{c} y_1 \\ \hline y_2 \end{array}\right) = \left(\begin{array}{c} \mathbf{O}^T_{n/2}\, y_1 + \mathbf{1}^T_{n/2}\, y_2 \\ \hline U_{n/2}\,(y_1 + y_2) \end{array}\right)$$

- Berechnung von y_1+y_2, $\mathbf{O}^T_{n/2} \cdot y_1$ und $\mathbf{1}^T_{n/2} \cdot y_2$ benötigt höchstens $2n$ arithm. Operationen. Sei $T(n)$ die Anzahl der arithm. Operationen für n. Dann gilt

  $T(2) = 2$ und $T(n) \leqslant T(n/2) + 2n$

- Auflösung der Rekurrenz ergibt $T(n) \leqslant 4n$.

Matrix-Vektor-Multiplikation

- Schulmethode: Zeit $O(n \cdot m) = O(n \log n)$
- Liberty-Zucker-Methode: nur Zeit $O(n)$

Beliebige $m \times n$-Matrix A:

- Unterteile A in $m' \times n$-Matrizen A_1, A_2,… mit $m' = \log_2 n$
- Wende Verfahren oben auf $A_i \cdot x$ an für jedes i, setze die Ergebnisse zum Ergebnisvektor zusammen

Laufzeit: $O(n \cdot m / \log n)$ (statt $O(n \cdot m)$)

Matrix-Vektor-Multiplikation

Erweiterung auf Matrizen A mit $a_{i,j} \in [B]$:

- Unterteile A in $m' \times n$-Matrizen $A_1, A_2, \ldots$ mit $m' = \log_B n$
- Verwende für $A_i \cdot x$ die Matrix

$$U_n = \left(\begin{array}{c|c|c|c} \mathbf{O}^T_{n/B} & \mathbf{1}^T_{n/B} & \cdots & \mathbf{B\text{-}1}^T_{n/B} \\ \hline U_{n/B} & U_{n/B} & \cdots & U_{n/B} \end{array} \right)$$

Laufzeit: $O(n \cdot m/\log_B n)$

Matrixmultiplikation

Restlicher Inhalt (über pdf-Folien):

- Der 4-Russen-Algorithmus
- Matrixmultiplikation a la Strassen
- Transitive Hülle