Kapitel 2: Algebraische und zahlen-theoretische Algorithmen

(Effiziente Algorithmen, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

Organisatorisches

- Nächste Vorlesung:
  Donnerstag, Oktober 25, 16:30–18:00 Uhr, AH II
- Webseite:
  `http://algo.rwth-aachen.de/Lehre/WS1819/Effi.php`

Algebraische und zahlen-theoretische Algorithmen

- Multiplikation von ganzen Zahlen
- Multiplikation von Matrizen
- Multiplikation von Polynomen

- Der grösste gemeinsame Teiler
- Modulare Potenzfunktion
- Primzahltest

Zum Aufwärmen: Multiplikation von ganzen Zahlen

Multiplikation von ganzen Zahlen: Schulmethode

In der Grundschule haben wir gelernt,
wie man zwei positive ganze Zahlen x und y mit einander multipliziert:

1	1	9	5	8	3	3	$*$	2	8	5	1
2	3	9	1	6	6	6					
	9	5	6	6	6	6	4				
		5	9	7	9	1	6	5			
			1	1	9	5	8	3	3		
3	4	0	9	3	1	9	8	8	3		

Diese Schulmethode verwendet $\Theta(n^2)$ Operationen.
(Wir betrachten hier die Bit-Komplexität von Algorithmen.)

Frage: Geht das irgendwie besser?

Multiplikation von ganzen Zahlen: Eine erste Idee

Wir teilen die Ziffernfolgen der beiden n-stelligen Zahlen $x = 10^{n/2}x_1 + x_0$ und $y = 10^{n/2}y_1 + y_0$ in zwei gleich lange Teile auf.

x:	x_1	x_0
y:	y_1	y_0

Divide and Conquer

- Wir berechnen rekursiv die vier Produkte x_0y_0, x_0y_1, x_1y_0, x_1y_1.
- Wir geben $10^n x_1y_1 + (x_0y_1 + x_1y_0)10^{n/2} + x_0y_0$ aus.

Ergo: $T(n) = 4T(n/2) + \Theta(n)$

Multiplikation von ganzen Zahlen: Die zweite Idee

Idee von Anatoly Alexeevitch Karatsuba (1962)

Statt den vier Produkten x_0y_0, x_0y_1, x_1y_0, x_1y_1,
berechnen wir nur die drei Produkte x_0y_0, x_1y_1, und $(x_0+x_1)(y_0+y_1)$

Damit können wir den gemischten Term in der Form

$$x_0y_1 + x_1y_0 \;=\; (x_0+x_1)(y_0+y_1) - x_0y_0 - x_1y_1$$

schreiben, und das gewünschte Produkt wie folgt berechnen:

$$xy \;=\; 10^n x_1y_1 + ((x_0+x_1)(y_0+y_1) - x_0y_0 - x_1y_1)10^{n/2} + x_0y_0$$

Multiplikation von ganzen Zahlen: Resultat

Ergo: $T(n) = 3T(n/2) + \Theta(n)$

Satz (Karatsuba, 1962)

Das Produkt von zwei n-stelligen Zahlen
kann mit $\Theta(n^{\log_2 3})$ Bit-Operationen berechnet werden.

Anmerkung: $\log_2 3 \approx 1.585$

Matrix-Multiplikation

Matrix-Multiplikation: Schulmethode

Problem: Matrix-Multiplikation

Eingabe: Zwei $n \times n$ Matrizen A und B

Gesucht: Das Produkt $C := AB$

Trivialer Algorithmus: Berechne Eintrag C_{ij} als $\sum_{k=1}^{n} a_{ik} b_{kj}$

Satz

Das Produkt von zwei $n \times n$ Matrizen
kann in kubischer Zeit $O(n^3)$ berechnet werden.

Anmerkung: Wir zählen die Anzahl der Multiplikationen.

Frage: Geht das irgendwie besser?

Multiplikation von 2×2 Matrizen (1)

Mit acht Multiplikationen und vier Additionen:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} w & x \\ y & z \end{pmatrix} = \begin{pmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{pmatrix}$$

Multiplikation von 2 × 2 Matrizen (2)

Mit sieben Multiplikationen und achtzehn Additionen:

$$\begin{aligned}
m_1 &= (b-d)(y+z) \\
m_2 &= (a+d)(w+z) \\
m_3 &= (a-c)(w+x) \\
m_4 &= (a+b)z \\
m_5 &= a(x-z) \\
m_6 &= d(y-w) \\
m_7 &= (c+d)w
\end{aligned}$$

$$\begin{pmatrix} aw+by & ax+bz \\ cw+dy & cx+dz \end{pmatrix} = \begin{pmatrix} m_1+m_2-m_4+m_6 & m_4+m_5 \\ m_6+m_7 & m_2-m_3+m_5-m_7 \end{pmatrix}$$

Multiplikation von 2×2 Matrizen (3)

Mit sechs Multiplikationen und 8.128 Additionen:

Nein, das geht nicht.
Bei sieben Multiplikationen ist bereits Schluss.

Shmuel Winograd hat 1971 gezeigt, dass man zur Multiplikation von 2×2 Matrizen mindestens sieben Multiplikationen braucht (auch wenn man noch so viele Additionen und Subtraktionen zur Verfügung hat).

Schnelle Matrix-Multiplikation nach Strassen (1)

Idee von Volker Strassen (1969)

- Konstruiere einen Divide-and-Conquer Algorithmus für die Multiplikation von $n \times n$ Matrizen.
- Verwende die sieben Multiplikationen und achtzehn Additionen (aus dem Multiplikationsschema für 2×2 Matrizen) im Conquer Schritt.

Schnelle Matrix-Multiplikation nach Strassen (2)

- Im Divide Schritt unterteilen wir die beiden Matrizen $n \times n$ Matrizen A und B jeweils in vier $n/2 \times n/2$ Matrizen:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- Im Conquer Schritt berechnen wir durch zehn Matrix-Additionen und durch sieben rekursive Aufrufe die sieben Matrix-Produkte $M_1, \ldots, M_7$ für diese $n/2 \times n/2$ Matrizen $A_{11}, A_{21}, \ldots, B_{22}$.
- Wir fügen die Ergebnisse durch acht Matrix-Additionen zur Ergebnismatrix zusammen:

$$\begin{pmatrix} M_1 + M_2 - M_4 + M_6 & M_4 + M_5 \\ M_6 + M_7 & M_2 - M_3 + M_5 - M_7 \end{pmatrix}$$

Schnelle Matrix-Multiplikation nach Strassen (3)

Ergo: $T(n) = 7T(n/2) + \Theta(n^2)$

Satz (Strassen, 1969)

Das Produkt von zwei $n \times n$ Matrizen
kann in $O(n^{\log_2 7})$ Zeit berechnet werden.

Anmerkung: $\log_2 7 \approx 2.807$

Anmerkungen

- Der konstante Faktor (in der O-Notation versteckt) von Strassen's Algorithmus ist grösser als der konstante Faktor in der $O(n^3)$ Schulmethode.
- In der Praxis: Schulmethode für kleine n, Strassen für grosse n. Der Cross-over Punkt liegt meistens um $n = 20$.
- In der Praxis: Wenn Matrizen dünn besetzt sind (= mit sehr vielen Nullen), dann gibt es schnellere Spezialalgorithmen aus der Numerik

- Don Coppersmith und Shmuel Winograd (1990) haben den Exponenten von Strassen's 2.807 auf 2.376 verbessert
- Virginia Vassilevska Williams (2011) hat den Exponenten weiter auf 2.373 verbessert

- Als untere Schranke für die Multiplikation von zwei $n \times n$ Matrizen kennen wir nur die triviale Schranke $\Omega(n^2)$.

Verifikation von Matrix-Multiplikationen

Verifikation von Matrix-Multiplikationen

Problem: Matrix-Multiplikation Verifikation

Eingabe: Drei $n \times n$ Matrizen A, B, C

Frage: Gilt $AB = C$?

Trivialer Lösungsansatz:

- Berechne das Produkt AB
- Vergleiche die n^2 Einträge mit den Einträgen in C

Frage

Geht das irgendwie besser? Schneller? Einfacher?

Ein probabilistischer Ansatz

Idee von Rusins Martins Freivalds (1977)

- Wähle zufällig einen Vektor $x \in \{0, 1\}^n$.
- Wenn $ABx = Cx$, dann return "$AB = C$".
  Wenn $ABx \neq Cx$, dann return "$AB \neq C$".

Anmerkungen:

- Wenn Freivalds Ungleichheit behauptet, dann gilt wirklich $AB \neq C$.
- Wenn Freivalds Gleichheit behauptet, dann gilt aber nicht notwendigerweise $AB = C$.
- Beispiel: Falls der zufällig gewählte Vektor $x = 0$ ist, dann behauptet Freivalds Gleichheit völlig unabhängig von A, B, C

Fehleranalyse (1)

Satz

Wenn D eine $n \times n$ Matrix mit $D \neq 0$ ist
und x ein zufällig gewählter Vektor in $\{0, 1\}^n$,
dann ist die Wahrscheinlichkeit von $Dx = 0$ höchstens $1/2$.

Beweisskizze:

- Wähle Indizes k und ℓ, sodass $d_{k\ell} \neq 0$ in Matrix D gilt.
- Setze $y := Dx$ und betrachte die k-te Komponente y_k in y.
- Dann gilt $y_k \;=\; d_{k1}x_1 + d_{k2}x_2 + \cdots + d_{kn}x_n \;=\; d_{k\ell}x_\ell + R$
- Zufallsexperiment: Wir setzen der Reihe nach die Komponenten x_i mit $i \neq \ell$ zufällig und unabhängig von einander auf ± 1, und erhalten so die entsprechenden Restsumme R.
- Dann ist mindestens einer der beiden Werte $R + d_{k\ell}$ mit $x_\ell = +1$ und R mit $x_\ell = 0$ ungleich 0.
- Daher gilt $y_k \neq 0$ mit Wahrscheinlichkeit mindestens $1/2$.

Fehleranalyse (2)

Satz

Wenn der Freivalds Algorithmus behauptet,

- dass $AB \neq C$, dann stimmt das auf jeden Fall;
- dass $AB \neq C$, dann stimmt das mit Wahrscheinlichkeit $\geq 1/2$.

Beweis:
Wende Satz von vorhergehender Seite auf Matrix $D := C - AB$ an

Wiederholt man den Freivalds Algorithmus 100-mal (unabhängig) so sinkt die Fehlerwahrscheinlichkeit von $1/2$ auf $2^{-100} \approx 10^{-30}$

Laufzeit

Satz

Der Freivalds Algorithmus kann in $O(n^2)$ Zeit implementiert werden.

Beweis: Das Produkt ABx berechnet man als $A(Bx)$.

Übung

Ein viel einfacherer Ansatz ("Ingenieursmethode")

- Wähle zufällig Zeilenindex i und Spaltenindex j
- Berechne Eintrag $[AB]_{ij}$ im Produkt AB
- Wenn $[AB]_{ij} = C_{ij}$, dann return "$AB = C$".
  Wenn $[AB]_{ij} \neq C_{ij}$, dann return "$AB \neq C$".

- Lineare Laufzeit $O(n)$
- Frage: Wie gross ist die Fehlerwahrscheinlichkeit?
- Frage: Wie oft muss man diesen einfachen Algorithmus wiederholen, damit die Fehlerwahrscheinlichkeit unter $1/2$ sinkt?

Multiplikation von Polynomen

Polynom-Multiplikation: Schulmethode (1)

In der Mittelschule haben wir gelernt,
wie man zwei Polynome $A(x)$ und $B(x)$ mit einander multipliziert:

$$(x^3 + 5x^2 - 2x + 1) \cdot (3x^3 - x^2 + x + 2)$$

$$\begin{array}{rrrrrrrr}
= & 3x^6 & +15x^5 & -6x^4 & +3x^3 & & & \\
 & & -x^5 & -5x^4 & +2x^3 & -x^2 & & \\
 & & & x^4 & +5x^3 & -2x^2 & +x & \\
 & & & & +2x^3 & +10x^2 & -4x & +2 \\
 & & & & & & & \\
= & 3x^6 & +14x^5 & -10x^4 & +12x^3 & +7x^2 & -3x & +2
\end{array}$$

Die Schulmethode verwendet $\Theta(n^2)$ Operationen,
um zwei Polynome vom Grad n mit einander zu multiplizieren.

Polynom-Multiplikation: Genaue Problemstellung

$$\begin{aligned}
A(x) &= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2x^2 + a_1x + a_0 \\
B(x) &= b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_2x^2 + b_1x + b_0
\end{aligned}$$

$$\begin{aligned}
C(x) &= A(x)\,B(x) \\
&= c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \cdots + c_2x^2 + c_1x + c_0
\end{aligned}$$

Problem: Polynom-Multiplikation

Eingabe: Ganze Zahlen $a_0, \ldots, a_{n-1}$ und $b_0, \ldots, b_{n-1}$

Gesucht: Zahlen $c_0, \ldots, c_{2n-2}$ mit $c_k = \sum_{i=0}^{k} a_i b_{k-i}$

Anmerkung: Wir nehmen an, dass $a_i = b_i = 0$ für $i \geq n$

Unser Ziel: Polynom-Multiplikation in sub-quadratischer Zeit

Polynom-Multiplikation: Illustration

$$\begin{array}{cccccccc}
a_0 b_0 & a_1 b_0 & a_2 b_0 & a_3 b_0 & a_4 b_0 & a_5 b_0 & \color{red}{a_6 b_0} & a_7 b_0 \\
a_0 b_1 & a_1 b_1 & a_2 b_1 & a_3 b_1 & a_4 b_1 & \color{red}{a_5 b_1} & a_6 b_1 & a_7 b_1 \\
a_0 b_2 & a_1 b_2 & a_2 b_2 & a_3 b_2 & \color{red}{a_4 b_2} & a_5 b_2 & a_6 b_2 & a_7 b_2 \\
a_0 b_3 & a_1 b_3 & a_2 b_3 & \color{red}{a_3 b_3} & a_4 b_3 & a_5 b_3 & a_6 b_3 & a_7 b_3 \\
a_0 b_4 & a_1 b_4 & \color{red}{a_2 b_4} & a_3 b_4 & a_4 b_4 & a_5 b_4 & a_6 b_4 & a_7 b_4 \\
a_0 b_5 & \color{red}{a_1 b_5} & a_2 b_5 & a_3 b_5 & a_4 b_5 & a_5 b_5 & a_6 b_5 & a_7 b_5 \\
\color{red}{a_0 b_6} & a_1 b_6 & a_2 b_6 & a_3 b_6 & a_4 b_6 & a_5 b_6 & a_6 b_6 & a_7 b_6 \\
a_0 b_7 & a_1 b_7 & a_2 b_7 & a_3 b_7 & a_4 b_7 & a_5 b_7 & a_6 b_7 & a_7 b_7
\end{array}$$

$$c_6 = a_0 b_6 + a_1 b_5 + a_2 b_4 + a_3 b_3 + a_4 b_2 + a_5 b_1 + a_6 b_0$$

Darstellung von Polynomen (1)

Koeffizienten-Darstellung

Das Polynom $A(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ wird
durch die Folge $a_0, \ldots, a_{n-1}$ der Koeffizienten spezifiziert.

Punkt-Wert-Darstellung

Das Polynom $A(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ wird
durch n Punkte $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ spezifiziert,
wobei $y_i = A(x_i)$ für $0 \leq i \leq n-1$ gilt.

Darstellung von Polynomen (2)

Zur Erinnerung

Für n paarweise verschiedene Zahlen $x_0, \ldots, x_{n-1}$, und
für n Punkte $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$
existiert genau ein Polynom $A(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$
mit $y_i = A(x_i)$ für $0 \leq i \leq n-1$.

Beweis für "es existiert höchstens ein Polynom $A(x)$":
Wenn ein Polynom vom Grad $n-1$ mehr als $n-1$ Nullstellen hat, so ist es identisch gleich 0

Beweis für "es existiert mindestens ein Polynom $A(x)$":
Lagrange Interpolation

$$A(x) \;=\; \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

Darstellung von Polynomen (3a): Übersetzung/Hin

Gegeben: Koeffizienten-Darstellung $a_0, \ldots, a_{n-1}$ eines Polynoms $A(x)$;
n paarweise verschiedene Zahlen $x_0, \ldots, x_{n-1}$

Gesucht: Punkt-Wert-Darstellung von $A(x)$ für Stützstellen $x_0, \ldots, x_{n-1}$

Jeder Wert $y_i = A(x_i)$ kann in $O(n)$ Zeit berechnet werden:

```
y= a[n-1];
for i= n-2 downto 0 do
    y= x*y + a[i]

return y;
```

Gesamtzeit: $O(n^2)$

Darstellung von Polynomen (3b): Übersetzung/Rück

Gegeben: Punkt-Wert-Darstellung $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ für $A(x)$;
Gesucht: Koeffizienten-Darstellung $a_0, \ldots, a_{n-1}$ von $A(x)$

Die Lagrange'sche Interpolationsformel kann in $O(n^2)$ Zeit ausgewertet werden

Zusammenfassend:
Koeffizienten-Darstellung und Punkt-Wert-Darstellung können in quadratischer Zeit $O(n^2)$ in einander übergeführt werden

Zurück zur Polynom-Multiplikation

Unser Hauptziel: Polynom-Multiplikation in sub-quadratischer Zeit

Problem: Polynom-Multiplikation (in Koeffizienten-Darstellung)

Eingabe: Ganze Zahlen $a_0, \ldots, a_{n-1}$ und $b_0, \ldots, b_{n-1}$

Gesucht: Zahlen $c_0, \ldots, c_{2n-2}$ mit $c_k = \sum_{i=0}^{k} a_i b_{k-i}$

Nebenproblem: Polynom-Multiplikation in sub-quadratischer Zeit

Problem: Polynom-Multiplikation (in Punkt-Wert-Darstellung)

Eingabe: Punkt-Wert-Darstellung $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ für $A(x)$
Punkt-Wert-Darstellung $(x_0, y_0'), \ldots, (x_{n-1}, y_{n-1}')$ für $B(x)$

Gesucht: Punkt-Wert-Darstellung $(x_0, y_0''), \ldots, (x_{n-1}, y_{n-1}'')$ für $A(x)B(x)$

Einfach in linearer Zeit $O(n)$ zu lösen

Arbeitsplan

Schritt 1:
Übersetze die beiden Polynome $A(x)$ und $B(x)$
von Koeffizienten-Darstellung nach Punkt-Wert-Darstellung

Schritt 2:
Multipliziere $A(x)$ und $B(x)$ in Punkt-Wert-Darstellung

Schritt 3:
Übersetze das in Schritt 2 berechnete Produkt $A(x)B(x)$
von Punkt-Wert-Darstellung nach Koeffizienten-Darstellung

Intermezzo:
Rechnen mit komplexen Zahlen

Komplexe Zahlen

Eine komplexe Zahl z kann dargestellt werden:

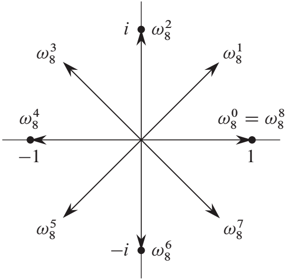
- Algebraisch durch Zerlegung in Realteil und Imaginärteil: $z = a + i\,b$
- Polar durch Radius r und Winkel ϕ: $\quad z = r\cos\phi + i\,r\sin\phi$
- Exponentiell: $\quad z = r \cdot e^{i\phi}$

Rechenoperationen mit $z = r \cdot e^{i\phi}$ und $z' = r' \cdot e^{i\phi'}$

- Multiplikation: $(r \cdot e^{i\phi})(r' \cdot e^{i\phi'}) \;=\; rr' \cdot e^{i(\phi+\phi')}$
- Potenzierung: $z^n \;=\; r^n \cdot e^{i n\phi}$

Einheitswurzeln (1)

Die n-ten Einheitswurzeln $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$
sind die n (komplexen) Lösungen der Gleichung $\omega^n = 1$.



$\omega_n^1 = e^{2\pi i/n}$ und $\omega_n^k = e^{2k\pi i/n}$ für $0 \leq k \leq n$

Einheitswurzeln (2)

- $\omega_n = e^{2\pi i/n}$ ist die n-te Haupt-Einheitswurzel
- Die n-ten Einheitswurzeln bilden eine multiplikative Gruppe, die zur zyklischen Gruppe $\mathbb{Z}_n$ (Restklassengruppe modulo n) isomorph ist
- Alle Einheitswurzeln sind Potenzen der Haupt-Einheitswurzel ω_n
- Es gelten die üblichen Rechenregeln $\omega_n^k \omega_n^\ell = \omega_n^{k+\ell}$

Halbierungslemma

Für gerades $n \geq 2$ fallen die Quadrate der n-ten Einheitswurzeln mit den $(n/2)$-ten Einheitswurzeln zusammen.

Summenlemma

Jede n-te Einheitswurzel $\omega_n^k \neq 1$ erfüllt die Gleichung

$$(\omega_n^k)^{n-1} + (\omega_n^k)^{n-2} + \cdots + (\omega_n^k)^2 + (\omega_n^k) + 1 \ = \ 0$$

Arbeitsplan: Schritt 1

Schritt 1: Zielsetzung

Zur Erinnerung:

Schritt 1:
Übersetze die beiden Polynome $A(x)$ und $B(x)$
von Koeffizienten-Darstellung nach Punkt-Wert-Darstellung

- Von jetzt an nehmen wir an, dass $n = 2^q$ eine Zweierpotenz ist
- Die Punkt-Wert-Darstellung von $A(x)$ und $B(x)$ werden wir mit den n-ten Einheitswurzeln $\langle x_0, \ldots, x_{n-1} \rangle = \langle \omega_n^0, \ldots, \omega_n^{n-1} \rangle$ als Stützstellen bestimmen
- Wir beschreiben das Verfahren nur für $A(x)$. Das Polynom $B(x)$ wird analog behandelt.
- Wir verwenden Divide & Conquer

Divide & Conquer (1)

Divide & Conquer Ansatz:

$$A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3 + \cdots + a_{n-2} x^{n/2-1}$$

$$A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3 + \cdots + a_{n-1} x^{n/2-1}$$

Koeffizienten-Darstellung:

$$A_{even}(x): \quad \langle a_0,\ a_2,\ a_4,\ a_6,\ \ldots,\ a_{n-2} \rangle$$

$$A_{odd}(x): \quad \langle a_1,\ a_3,\ a_5,\ a_7,\ \ldots,\ a_{n-1} \rangle$$

$$A(x) \;=\; A_{even}(x^2) + x \cdot A_{odd}(x^2)$$

Divide & Conquer (2)

Wir beobachten, dass die Zahlen $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$ exakt mit den $(n/2)$-ten Einheitswurzeln zusammen fallen

- Wir bestimmen rekursiv die Punkt-Wert-Darstellung von $A_{even}(x)$ für die $n/2$ Stützstellen $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$
- Wir bestimmen rekursiv die Punkt-Wert-Darstellung von $A_{odd}(x)$ für die $n/2$ Stützstellen $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$
- Wir berechnen aus diesen beiden Punkt-Wert-Darstellungen die Punkt-Wert-Darstellung von $A(x)$ für die n Stützstellen $\omega_n^0, \ldots, \omega_n^{n-1}$ mit Hilfe von $A(x) \;=\; A_{even}(x^2) + x \cdot A_{odd}(x^2)$

Ergo: $T(n) = 2T(n/2) + \Theta(n)$ und daher $T(n) \in O(n \log n)$

Was haben wir in Schritt 1 eigentlich berechnet? (1)

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} & \cdots & \omega^{3(n-1)} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} & \cdots & \omega^{4(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \omega^{4(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Wir haben den Koeffizientenvektor a mit einer Matrix V multipliziert, und als Resultat den Vektor y mit den gewünschten y-Koordinaten erhalten.

Was haben wir in Schritt 1 eigentlich berechnet? (2)

Wir haben den Koeffizientenvektor a mit einer Matrix V multipliziert,
und als Resultat den Vektor y erhalten.

Der resultierende Vektor y ist
die *Diskrete Fourier Transformierte (DFT)* des Vektors a
bezüglich der Einheitswurzel ω

Anmerkung:
Wir hätten genauso gut (in der selben Laufzeit) die DFT des Vektors a bezüglich der Einheitswurzeln ω^3 oder ω^5 oder ω^{-1} berechnen können

Arbeitsplan: Schritt 3

Schritt 3: Zielsetzung

Zur Erinnerung:

Schritt 3:
Übersetze das in Schritt 2 berechnete Produkt $A(x)B(x)$
von Punkt-Wert-Darstellung nach Koeffizienten-Darstellung

- Unser Startpunkt ist die Punkt-Wert-Darstellung eines Polynoms $C(x)$ an n Stützstellen $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$, wobei $x_k = \omega_n^k$ für $0 \leq k \leq n-1$
- Wir suchen die Koeffizienten-Darstellung $c_0, \ldots, c_{n-1}$ von $C(x)$

Illustration (1)

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} & \cdots & \omega^{3(n-1)} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} & \cdots & \omega^{4(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \omega^{4(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Lineares Gleichungssystem $Vc = y$ (und wir suchen den Vektor c)

Illustration (2)

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} & \cdots & \omega^{3(n-1)} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} & \cdots & \omega^{4(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \omega^{4(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Lineares Gleichungssystem $c = V^{-1} y$ (und wir suchen den Vektor c)

Das lineare Gleichungssystem (1)

Es sei $\omega = e^{2\pi i/n}$ die n-te Haupt-Einheitswurzel.
Es sei $V = (v_{j,k})$ die $n \times n$ Matrix mit $v_{j,k} = \omega^{jk}$.
Es seien $y_0, \ldots, y_{n-1}$ die vorgegebenen Funktionswerte
des Polynoms $C(x)$ an den Stützstellen $\omega^0, \ldots, \omega^{n-1}$

Unser Ziel: Löse das Gleichungssystem $Vc = y$ nach dem Vektor c auf

Das lineare Gleichungssystem (2)

Satz

Die inverse Matrix $V^{-1} = (w_{j,k})$ ist gegeben durch $w_{j,k} = \omega^{-jk}/n$.

Beweis: Wir verifizieren, dass $V^{-1}V = I$ gilt.
Der Eintrag in der r-ten Zeile und s-ten Spalte von $V^{-1}V$ ist

$$[V^{-1}V]_{r,s} = \sum_{k=0}^{n-1} (\omega^{-rk}/n)\,(\omega^{ks}) = \frac{1}{n}\sum_{k=0}^{n-1} (\omega^{r-s})^k$$

- Für $r = s$ ist $\omega^{r-s} = 1$, und daher $\sum_{k=0}^{n-1} (\omega^{r-s})^k = n$
- Für $r \neq s$ ist $\omega^{r-s} \neq 1$, und daher $\sum_{k=0}^{n-1} (\omega^{r-s})^k = 0$

Das lineare Gleichungssystem (3)

Altes Ziel:

Löse das Gleichungssystem $Vc = y$ nach dem Vektor c auf

Neues Ziel:

Berechne den Vektor $c = V^{-1}y$, wobei $[V^{-1}]_{j,k} = \omega^{-jk}/n$ gilt.

Neues Ziel, besser formuliert:

Berechne die Diskrete Fourier Transformierte (DFT) d des Vektors y bezüglich der Einheitswurzel ω^{-1}.
Der gewünschte Vektor ist dann $c = d/n$.

Ergo: Schritt 3 kann in $O(n \log n)$ Zeit erledigt werden

Polynom-Multiplikation: Zusammenfassung

Arbeitsplan

Schritt 1:
Übersetze die beiden Polynome $A(x)$ und $B(x)$
von Koeffizienten-Darstellung nach Punkt-Wert-Darstellung

Laufzeit: $O(n \log n)$

Schritt 2:
Multipliziere $A(x)$ und $B(x)$ in Punkt-Wert-Darstellung

Laufzeit: $O(n)$

Schritt 3:
Übersetze das in Schritt 2 berechnete Produkt $A(x)B(x)$
von Punkt-Wert-Darstellung nach Koeffizienten-Darstellung

Laufzeit: $O(n \log n)$

Hauptresultat

Satz

Das Produkt zweier Polynome vom Grad $n-1$ in Koeffizienten-Darstellung kann in $O(n \log n)$ Zeit berechnet werden.

Anmerkungen:

- Da das Ergebnispolynom $C(x)$ den Grad $2n-2$ hat, müssen wir ganz am Anfang die Koeffizientenvektoren $a_0, \ldots, a_{n-1}$ und $b_0, \ldots, b_{n-1}$ durch Hinzufügen von Komponenten mit Wert 0 auf die Dimension $2n-2$ bringen.
- Durch Hinzufügen von noch mehr Komponenten mit Wert 0 machen wir die Dimension zu einer Zweierpotenz

Anmerkungen

- Die Diskrete Fourier Transform wurde von James William Cooley und John Wilder Tukey 1965 algorithmisch untersucht. (*"An algorithm for the machine calculation of complex Fourier series"*, Mathematics of Computation 11, pp 297–301)
- Der Numeriker Carl David Tolmé Runge verwendete bereits 1903 die Reduktion der n-dimensionalen DFT auf zwei $(n/2)$-dimensionale DFTs
- Der rekursive Algorithmus wurde bereits von Carl Friedrich Gauss im Jahre 1805 benutzt, um die Flugbahnen der Asteroiden Pallas und Juno zu interpolieren. (*"Theoria interpolationis methodo nova tractata"*, verfasst in Neu-Latein)
- Die FFT (Fast Fourier Transformation) wurde als einer der *Top 10 Algorithms of the 20th Century* gelistet (Algorithms with the greatest influence on the development and practice of science and engineering in the 20th century)

Die Liste der Top 10 Algorithmen

- Metropolis Algorithm for Monte Carlo
- Simplex Method for Linear Programming
- Krylov Subspace Iteration Methods
- The Decompositional Approach to Matrix Computations
- The Fortran Optimizing Compiler
- QR Algorithm for Computing Eigenvalues
- Quicksort Algorithm for Sorting
- Fast Fourier Transform
- Integer Relation Detection
- Fast Multipole Method