Algorithmen und Datenstrukturen (für ET/IT)

Sommersemester 2014

Dr. Tobias Lasser

Computer Aided Medical Procedures
Technische Universität München

Programm heute

7. Fortgeschrittene Datenstrukturen
8. Such-Algorithmen
9. Graph-Algorithmen
10. Numerische Algorithmen
    - Matrizen
    - Lineare Gleichungen
    - Die LUP-Zerlegung
    - Least Squares Probleme
    - Fast Fourier Transformation

Fourier Transformation

- Signale $f : \mathbb{R} \to \mathbb{R}$ sind im Zeitbereich
  - jedem Zeitpunkt wird ein Wert (z.B. Amplitude) zugeordnet
  - Beispiele: Audio, Video, Bilder
- Fourier-Transformation führt Signal in Frequenzbereich über
  - betrachte Signal als Überlagerung von Sinuskurven
- Anwendungen:
  - Filterung von bestimmten Frequenzen
  - Datenkompression, z.B.
    - Audio: MP3
    - Video: MPEG
    - Bilder: JPEG
  - Bild- und Signalverarbeitung
- überaus nützlich, und effizient dank FFT (Fast Fourier Transformation)

Polynome

Polynome

Sei $n \in \mathbb{N}$. Eine Funktion $A : \mathbb{R} \to \mathbb{R}$ der Form

$$A(x) = a_0 + a_1 x^1 + \ldots + a_{n-1} x^{n-1}$$

heißt Polynom in x vom Grad $n-1$. Hierbei heißen die $a_i \in \mathbb{R}$, $i = 1, \ldots, n-1$, Koeffizienten von A, $a_{n-1} \neq 0$.

- A hat Gradschranke n, falls der Grad von A zwischen 0 und $n-1$ liegt.
- Summen-Notation für A:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

- Die Koeffizientendarstellung von A ist der Vektor

$$a = (a_0, a_1, \ldots, a_{n-1})^T \in \mathbb{R}^n$$

Operationen auf Polynomen

Seien A, B Polynome.

- Auswertung von Polynom: berechne für $x_0 \in \mathbb{R}$

$$A(x_0)$$

- Summe von Polynomen: berechne Polynom C für $x \in \mathbb{R}$ mit

$$C(x) = A(x) + B(x)$$

- Multiplikation von Polynomen: berechne Polynom C für $x \in \mathbb{R}$ mit

$$C(x) = A(x)B(x)$$

Auswertung von Polynomen

Sei A Polynom mit Koeffizientendarstellung $a = (a_0, \ldots, a_{n-1})$.

- Auswertung von A an $x_0 \in \mathbb{R}$ mit Horner-Schema:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \ldots + x_0(a_{n-2} + x_0(a_{n-1}))\cdots))$$

- Aufwand: $\Theta(n)$
- Beispiel: $A(x) = 5x^2 - 3x + 4$ an Stelle $x_0 = 2$:

$$A(x_0) = 4 + x_0(-3 + x_0(5)) = 4 + 2(-3 + 2(5)) = 18$$

Addition von Polynomen

Seien A, B Polynome mit Gradschranke $n \in \mathbb{N}$.

- Koeffizientendarstellungen $a = (a_0, \ldots, a_{n-1})$, $b = (b_0, \ldots, b_{n-1})$
- Summe $A(x) + B(x)$ ist Polynom C mit Gradschranke n,

$$C(x) = \sum_{j=0}^{n-1} c_j x^j$$

  mit $c_j = a_j + b_j$ für $j = 0, \ldots, n-1$.
- Aufwand: $\Theta(n)$
- Beispiel: $A(x) = 5x^2 - 3x + 4$, $B(x) = 4x - 2$, dann ist

$$C(x) = 5x^2 + x + 2$$

Multiplikation von Polynomen

Seien A, B Polynome mit Gradschranke $n \in \mathbb{N}$.

- Koeffizientendarstellungen $a = (a_0, \ldots, a_{n-1})$, $b = (b_0, \ldots, b_{n-1})$
- Produkt $A(x)B(x)$ ist Polynom C mit Gradschranke $2n - 1$,

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j$$

mit

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} \qquad \text{für } j = 0, \ldots, 2n-2$$

- Aufwand: $\Theta(n^2)$
- Beispiel: $A(x) = 5x^2 - 3x + 4$, $B(x) = 4x - 2$, dann sind

$$c_0 = -8, \; c_1 = 22, \; c_2 = -22, \; c_3 = 20, \; c_4 = 0$$

Faltung

Seien A, B Polynome mit Gradschranke $n \in \mathbb{N}$.

- Koeffizientendarstellung $a = (a_0, \ldots, a_{n-1})$, $b = (b_0, \ldots, b_{n-1})$
- Sei $c = (c_0, \ldots, c_{2n-2})$ die Koeffizientendarstellung des Produktpolynoms $C(x) = A(x)B(x)$ mit

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}$$

- c wird auch Faltung der Vektoren a, b genannt,

$$c = a * b$$

- Faltung ist sehr häufige Operation auf Signalen (z.B. Anwendung von Filtern)

Stützstellendarstellung

Stützstellendarstellung von Polynom

Sei A Polynom mit Gradschranke $n \in \mathbb{N}$. Die Stützstellendarstellung von A ist ein Menge von n Stützstellenpaaren

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

mit x_k paarweise verschieden für $k = 0, \ldots, n-1$ sowie

$$y_k = A(x_k).$$

- Die Stützstellendarstellung ist nicht eindeutig
  - jede Menge von verschiedenen Punkten $x_0, \ldots, x_{n-1}$ kann verwendet werden

Auswertung

Sei A Polynom mit Gradschranke $n \in \mathbb{N}$ mit Koeffizienten $a = (a_0, \ldots, a_{n-1})$.

- Konversion von Koeffizientendarstellung in Stützstellendarstellung erfolgt mittels Auswertung:
  - wähle Stützstellen $x_0, \ldots, x_{n-1}$
  - werte $A(x_k)$ aus für $k = 0, \ldots, n-1$
- Auswertung von n Stützstellen mit Horner-Schema: $\Theta(n^2)$
- später: Auswertung mittels FFT: $\Theta(n \log n)$

Interpolation

Eindeutigkeit interpolierendes Polynom

Für jede Menge von n Stützstellenpaaren

$$\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$$

mit paarweise verschiedenen x_k existiert ein eindeutiges Polynom $A(x)$ mit Gradschranke n, so daß gilt

$$y_k = A(x_k) \qquad \text{für } k = 0, \ldots, n-1.$$

- Konversion von Stützstellendarstellung in Koeffizientendarstellung heißt Interpolation.

Interpolation: Beweisidee

- Ähnlich zu Least Squares Problemen kann Matrix-Gleichung aufgestellt werden:

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}}_{=:V(x_0,\ldots,x_{n-1})} \underbrace{\begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{=:a} = \underbrace{\begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}}_{=:y}$$

- $V(x_0, \ldots, x_{n-1})$ heißt Vandermonde-Matrix
- da x_k paarweise verschieden ist

$$\det V(x_0, \ldots, x_{n-1}) = \prod_{0 \le j < k \le n-1} (x_k - x_j) \neq 0$$

- also ist $V(x_0, \ldots, x_{n-1})$ invertierbar und

$$a = V(x_0, \ldots, x_{n-1})^{-1} y$$

ist die Koeffizientendarstellung von A.

Interpolation: Algorithmen

Sei A Polynom mit n Stützstellenpaaren $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$

- Berechnung mit LUP-Zerlegung:
  - $a = V(x_0, \ldots, x_{n-1})^{-1} y$
  - Aufwand: $\Theta(n^3)$
- Berechnung mit Lagrange-Interpolationsformel:

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

  - Aufwand: $\Theta(n^2)$
- später: Berechnung mit IFFT: $\Theta(n \log n)$

Addition von Polynomen

Seien A, B Polynome mit Gradschranke $n \in \mathbb{N}$.

- Stützstellendarstellung für A:

$$\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$$

- Stützstellendarstellung für B mit selben x_k:

$$\{(x_0, y'_0), \ldots, (x_{n-1}, y'_{n-1})\}$$

- Stützstellendarstellung für $C(x) = A(x) + B(x)$:

$$\{(x_0, y_0 + y'_0), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})\}$$

- Aufwand: $\Theta(n)$

Multiplikation von Polynomen

Seien A, B Polynome mit Gradschranke $n \in \mathbb{N}$.

- Produkt $C(x) = A(x)B(x)$ hat Gradschranke $2n$
  - benötigt $2n$ Stützstellen
  - erweitere Darstellung von A, B auf $2n$ Stützstellen vor Multiplikation
- Stützstellendarstellungen für A, B:

$$\{(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})\}$$

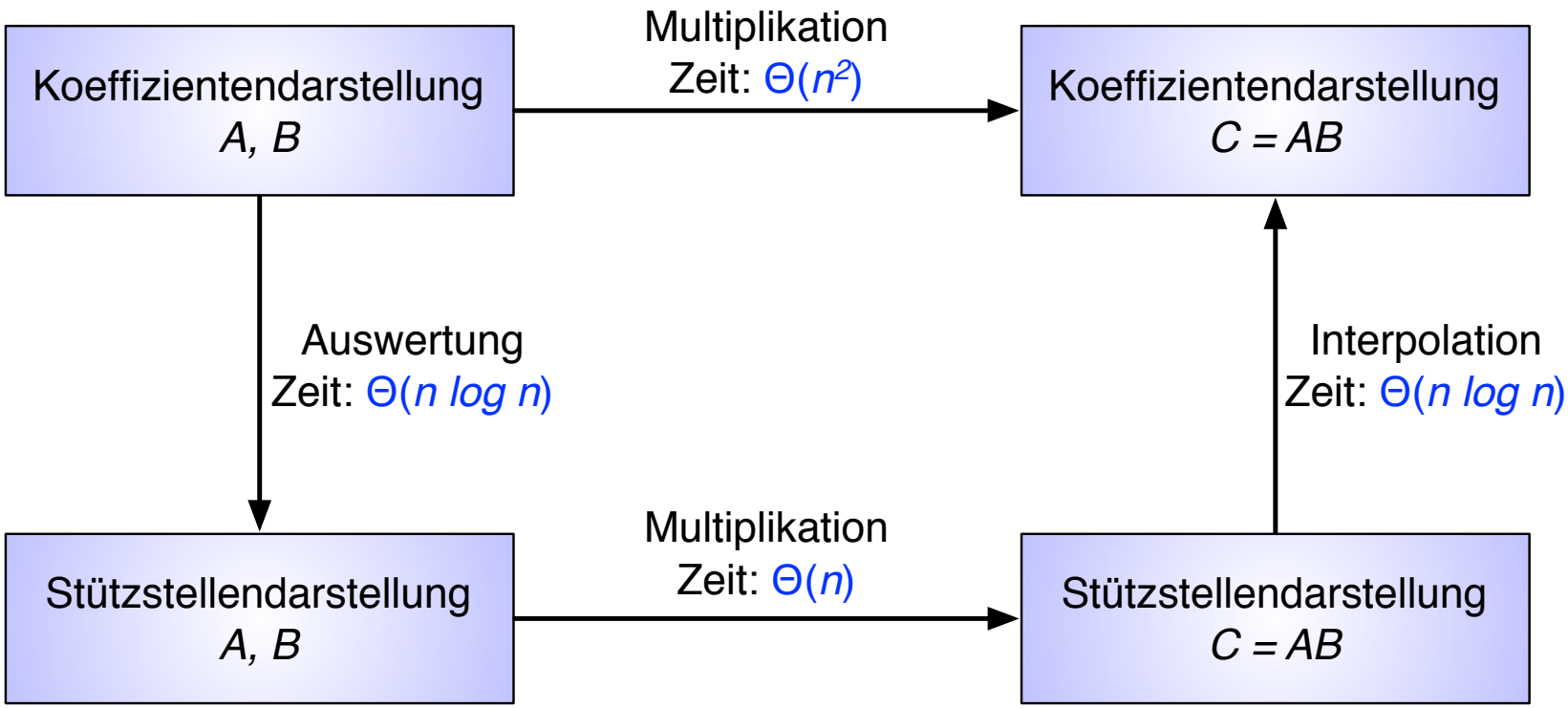
$$\{(x_0, y'_0), \ldots, (x_{2n-1}, y'_{2n-1})\}$$

- Stützstellendarstellung für C(x)=A(x)B(x):

$$\{(x_0, y_0 y'_0), \ldots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\}$$

- Aufwand: $\Theta(n)$

Effiziente Multiplikation von Polynomen



- Multiplikation von Polynomen in Koeffizientendarstellung via Stützstellendarstellung: $\Theta(n \log n)$
- Auswertung und Interpolation mit FFT/IFFT

Komplexe Einheitswurzeln

Komplexe Einheitswurzeln

Sei $n \in \mathbb{N}$. Die komplexe n-te Einheitswurzel ist eine Zahl $\omega \in \mathbb{C}$ mit

$$\omega^n = 1.$$

- Es gibt n komplexe n-te Einheitswurzeln: $e^{2\pi ik/n}$ für $k = 0, \ldots, n-1$.

Komplexe Einheitswurzeln

Komplexe Einheitswurzeln

Sei $n \in \mathbb{N}$. Die komplexe n-te Einheitswurzel ist eine Zahl $\omega \in \mathbb{C}$ mit

$$\omega^n = 1.$$

- Es gibt n komplexe n-te Einheitswurzeln: $e^{2\pi ik/n}$ für $k = 0, \ldots, n-1$.

n-te Haupteinheitswurzel

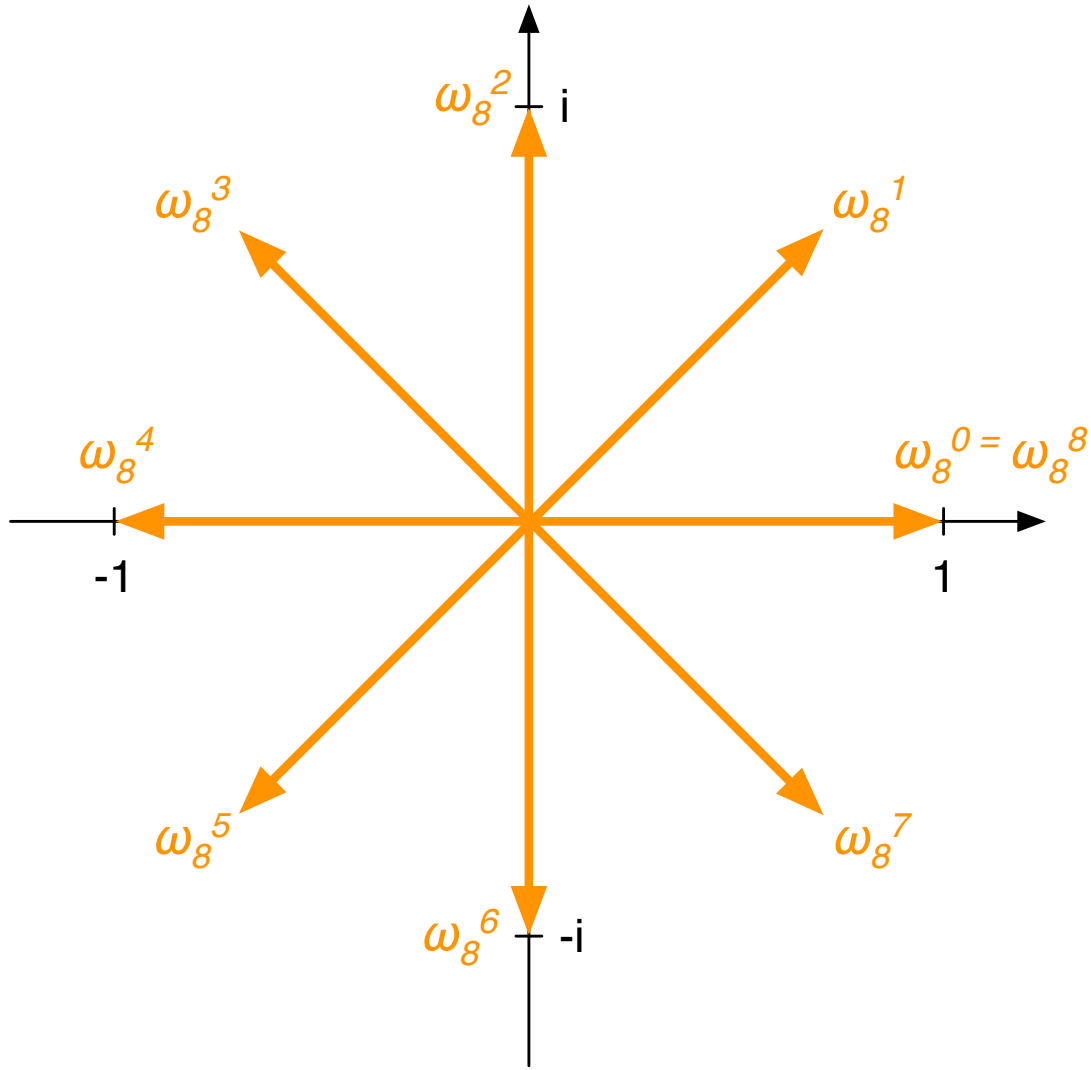
Die Zahl $\omega_n \in \mathbb{C}$ mit

$$\omega_n = e^{2\pi i/n}$$

heißt n-te Haupteinheitswurzel.

- Die anderen komplexen n-ten Einheitswurzeln sind Potenzen von ω_n.

Beispiel: die komplexen 8-ten Einheitswurzeln



$$\omega_8 = e^{2\pi i/8}$$

Diskrete Fourier Transformation

Diskrete Fourier Transform

Sei A Polynom mit Gradschranke n mit Koeffizienten $a = (a_0, \ldots, a_{n-1})$,

$$A(x) = \sum_{j=0}^{n-1} a_j x^j.$$

Die Auswertung von A an den Stellen $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$, d.h.

$$y_k = A(\omega_n^k) = \sum_{j=1}^{n-1} a_j \omega_n^{kj}$$

für $k = 0, \ldots, n-1$, heißt Diskrete Fourier Transformation des Koeffizientenvektors a. Kurz notiert auch als

$$y = DFT_n(a)$$

mit $y = (y_0, \ldots, y_{n-1})$.

Diskrete Fourier Transformation

- Aufwand der DFT: $\Theta(n^2)$
- zur Multiplikation von Polynomen benötigen wir tatsächlich $2n$ Koeffizienten statt n
- Zur effizienten Berechnung der DFT verwenden wir einen Divide & Conquer Ansatz $\rightarrow$ FFT

Eigenschaften komplexer Einheitswurzeln

Halbierungslemma

Sei $n \in \mathbb{N}$ gerade Zahl. Dann gilt für $k = 0, \ldots, n-1$

$$(\omega_n^k)^2 = \omega_{n/2}^k = \left(\omega_n^{k+n/2}\right)^2.$$

Eigenschaften komplexer Einheitswurzeln

Halbierungslemma

Sei $n \in \mathbb{N}$ gerade Zahl. Dann gilt für $k = 0, \ldots, n-1$

$$(\omega_n^k)^2 = \omega_{n/2}^k = \left(\omega_n^{k+n/2}\right)^2.$$

Beweis-Idee:

- Es ist

$$(\omega_n^k)^2 = \left(e^{2\pi ik/n}\right)^2 = \left(e^{2\pi i \cdot 2/n}\right)^k = \omega_{n/2}^k$$

Eigenschaften komplexer Einheitswurzeln

Halbierungslemma

Sei $n \in \mathbb{N}$ gerade Zahl. Dann gilt für $k = 0, \ldots, n-1$

$$(\omega_n^k)^2 = \omega_{n/2}^k = \left(\omega_n^{k+n/2}\right)^2.$$

Beweis-Idee:

- Es ist
$$(\omega_n^k)^2 = \left(e^{2\pi ik/n}\right)^2 = \left(e^{2\pi i \cdot 2/n}\right)^k = \omega_{n/2}^k$$
- Ausserdem:
$$\left(\omega_n^{k+n/2}\right)^2 = \omega_n^{2k+n} = \omega_n^{2k}\omega_n^n = \omega_n^{2k} = (\omega_n^k)^2$$

DFT mit Divide and Conquer 1

Sei A Polynom mit Koeffizienten $a = (a_0, \ldots, a_{n-1})$, $n \in \mathbb{N}$.

- Annahme: n sei eine Zweier-Potenz ($n = 2^m$)
- Divide-Schritt: teile A auf entlang Koeffizienten mit geradem und ungeradem Index in $A^{[0]}$ und $A^{[1]}$:

$$\begin{aligned} A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1} \\ A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1} \end{aligned}$$

- $A^{[0]}$ und $A^{[1]}$ haben Gradschranke $n/2$
- Conquer-Schritt: es gilt

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

DFT mit Divide and Conquer 2

Auswertung von $A(x)$ für $x = \omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ also berechnet als:

- Divide: werte $A^{[0]}$ und $A^{[1]}$ mit Gradschranke $n/2$ aus an

$$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$$

  Laut Halbierungslemma sind das nur $n/2$ verschiedene Einheitswurzeln!

- Rekursion: weitere Aufteilung der Polynome in kleinere Probleme
- Conquer: führe Ergebnisse zusammen

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

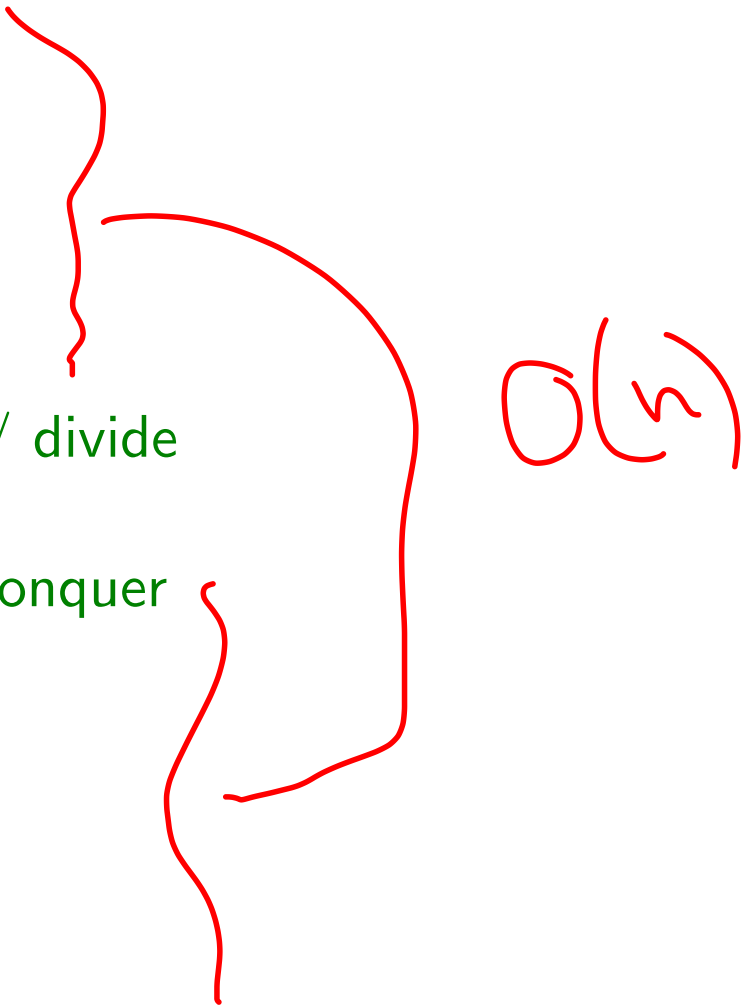
Der Divide Schritt halbiert also (gemäß Halbierungslemma) die Problemgrösse, und eine DFT_n wird mittels 2 $DFT_{n/2}$ berechnet.

Algorithmus: FFT (rekursiv)

Input: Vektor $a = (a_0, \ldots, a_{n-1})^T \in \mathbb{R}^n$, $n \in \mathbb{N}$ Zweier-Potenz
Output: $DFT_n(a)$

FFTrekursiv(a):

```
1    if (n == 1) return a;
2    ω_n = e^{2πi/n};
3    ω = 1;
4    a^[0] = (a_0, a_2, a_4, ..., a_{n-2});
5    a^[1] = (a_1, a_3, a_5, ..., a_{n-1});
6    y^[0] = FFTrekursiv(a^[0]);  // divide
7    y^[1] = FFTrekursiv(a^[1]);
8    for k = 0 to n/2 − 1 {  // conquer
9       y_k = y_k^[0] + ω y_k^[1];
10      y_{k+n/2} = y_k^[0] − ω y_k^[1];
11      ω = ωω_n;    // ω = ω_n^k
12   }
13   return y;
```



FFT (rekursiv): Korrektheit und Laufzeit 1

- Zeile 1: DFT von einem Element ist das Element selbst,

$$y_0 = a_0 \omega_1^0 = a_0 \cdot 1 = a_0$$

FFT (rekursiv): Korrektheit und Laufzeit 1

- Zeile 1: DFT von einem Element ist das Element selbst,

$$y_0 = a_0 \omega_1^0 = a_0 \cdot 1 = a_0$$

- Zeilen 6/7: wegen Halbierungslemma ist

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k})$$

$$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k})$$

FFT (rekursiv): Korrektheit und Laufzeit 1

- Zeile 1: DFT von einem Element ist das Element selbst,

$$y_0 = a_0\omega_1^0 = a_0 \cdot 1 = a_0$$

- Zeilen 6/7: wegen Halbierungslemma ist

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k})$$

$$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k})$$

- Zeile 9: berechnet $y_0, y_1, \ldots, y_{n/2-1}$ da

$$y_k = y_k^{[0]} + \omega_n^k y_k^{[1]} = A^{[0]}(\omega_n^{2k}) + \omega_n^k A^{[1]}(\omega_n^{2k}) = A(\omega_n^k)$$

FFT (rekursiv): Korrektheit und Laufzeit 2

- Zeile 10: berechnet $y_{n/2}, y_{n/2+1}, \ldots, y_{n-1}$ da

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k y_k^{[1]} = y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]}$$

da $\omega_n^{n/2} = -1$ und damit $\omega_n^{k+n/2} = -\omega_n^k$, und somit weiter

$$\begin{aligned} y_{k+n/2} &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) = A(\omega_n^{k+n/2}) \end{aligned}$$

da $\omega_n^{2k+n} = \omega_n^{2k}$.

FFT (rekursiv): Korrektheit und Laufzeit 2

- Zeile 10: berechnet $y_{n/2}, y_{n/2+1}, \ldots, y_{n-1}$ da

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k y_k^{[1]} = y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]}$$

da $\omega_n^{n/2} = -1$ und damit $\omega_n^{k+n/2} = -\omega_n^k$, und somit weiter

$$\begin{aligned} y_{k+n/2} &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) = A(\omega_n^{k+n/2}) \end{aligned}$$

da $\omega_n^{2k+n} = \omega_n^{2k}$.

- also tatsächlich $y = DFT_n(a)$

FFT (rekursiv): Korrektheit und Laufzeit 2

- Zeile 10: berechnet $y_{n/2}, y_{n/2+1}, \ldots, y_{n-1}$ da

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k y_k^{[1]} = y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]}$$

  da $\omega_n^{n/2} = -1$ und damit $\omega_n^{k+n/2} = -\omega_n^k$, und somit weiter

$$\begin{aligned} y_{k+n/2} &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) = A(\omega_n^{k+n/2}) \end{aligned}$$

  da $\omega_n^{2k+n} = \omega_n^{2k}$.

- also tatsächlich $y = DFT_n(a)$
- Laufzeit: ohne Rekursion $\Theta(n)$, Rekursionsgleichung:

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$

Interpolation mit komplexen Einheitswurzeln

Sei A Polynom mit Koeffizienten $a = (a_0, \ldots, a_{n-1})$.

- DFT lässt sich auch als Matrix-Gleichung mit Vandermonde-Matrix V_n mit ω_n-Potenzen als Einträge:

$$\underbrace{\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}}_{=y} = \underbrace{\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}}_{=:V_n} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{=a}$$

- Inverse Operation $a = V_n^{-1} y$ entspricht Interpolation, auch geschrieben als

$$a = IDFT_n(y)$$

Inverse Diskrete Fourier Transformation

- Es ist also $IDFT_n(y) = V_n^{-1} y$
- V_n^{-1} hat Einträge ω_n^{-kj}/n an Stelle (j, k) für $j, k = 0, \ldots, n-1$
- $IDFT_n(y)$ ist also für $j = 0, \ldots, n-1$

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}$$

- IFFT lässt sich aus FFT gewinnen durch
  - vertauschen von a und y
  - ersetzen von ω_n durch ω_n^{-1}
  - jedes Element mit $\frac{1}{n}$ multiplizieren
- IFFT Laufzeit also auch $\Theta(n \log n)$

Faltungstheorem

Faltungstheorem

Seien $a, b \in \mathbb{R}^n$ mit $n \in \mathbb{N}$ und $n = 2^m$. Dann ist

$$a * b = IDFT_{2n}\big(DFT_{2n}(a) \cdot DFT_{2n}(b)\big)$$

- a, b müssen mit Nullen zur Länge $2n$ aufgefüllt werden
- $\cdot$ ist das komponentenweise Produkt zweier Vektoren
- Gesamtlaufzeit: $\Theta(n \log n)$

Algorithmus: FFT (rekursiv)

Input: Vektor $a = (a_0, \ldots, a_{n-1})^T \in \mathbb{R}^n$, $n \in \mathbb{N}$ Zweier-Potenz
Output: $DFT_n(a)$
FFTrekursiv(a):

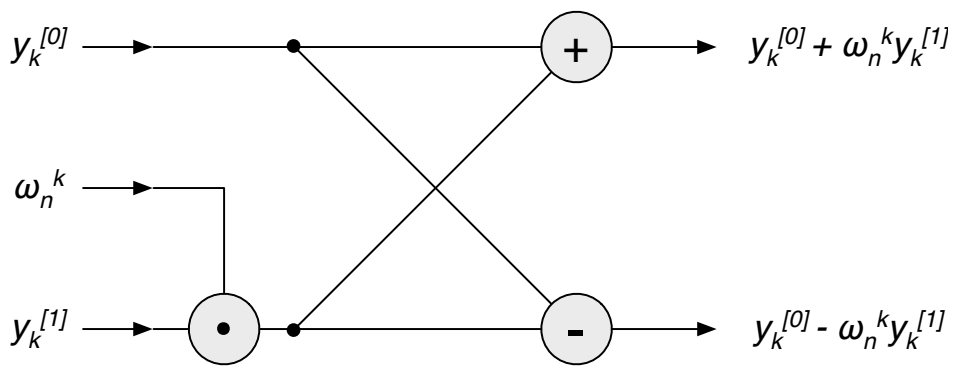
```
1    if (n == 1) return a;
2    ω_n = e^{2πi/n};
3    ω = 1;
4    a^[0] = (a_0, a_2, a_4, ..., a_{n-2});
5    a^[1] = (a_1, a_3, a_5, ..., a_{n-1});
6    y^[0] = FFTrekursiv(a^[0]);  // divide
7    y^[1] = FFTrekursiv(a^[1]);
8    for k = 0 to n/2 - 1 {  // conquer
9       y_k = y_k^[0] + ω y_k^[1];
10      y_{k+n/2} = y_k^[0] - ω y_k^[1];
11      ω = ω ω_n;    // ω = ω_n^k
12   }
13   return y;
```

FFT effizienter

- **FFTrekursiv** berechnet in Zeile 9,10 zweimal $\omega y_k^{[1]}$
- effizienter mit Hilfsvariable t:

for $k = 0$ **to** $n/2 - 1$ {
$t = \omega y_k^{[1]}$;
$y_k = y_k^{[0]} + t$;
$y_{k+n/2} = y_k^{[0]} - t$;
$\omega = \omega\omega_n$;
}

- dieses Schema ist bekannt als Butterfly-Operation:



FFT: weitere Optimierungen

Zusätzlich zu Butterfly-Operationen:

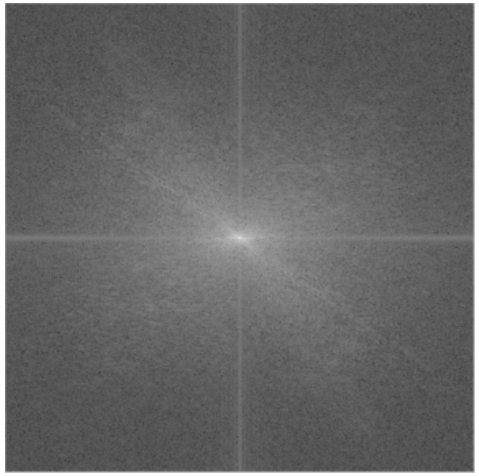
- Entrekursivierung möglich mittels Bitreversierung
- Komplexität immer noch $\Theta(n \log n)$, aber kleinere Konstanten
- Implementierung als parallele Schaltkreise möglich

FFT: Ausblick

- bisher behandelt: eindimensionale DFT
- auch möglich: mehrdimensionale DFT

FFT: Beispiel 1

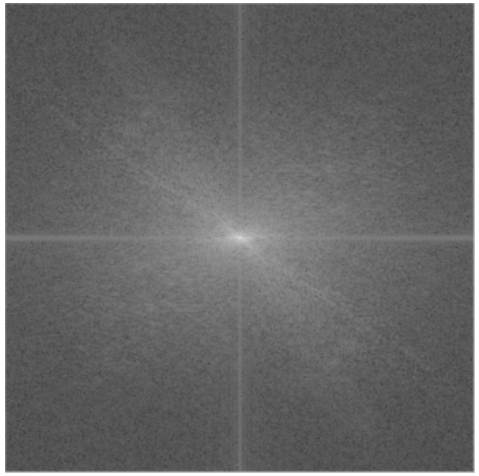
Original



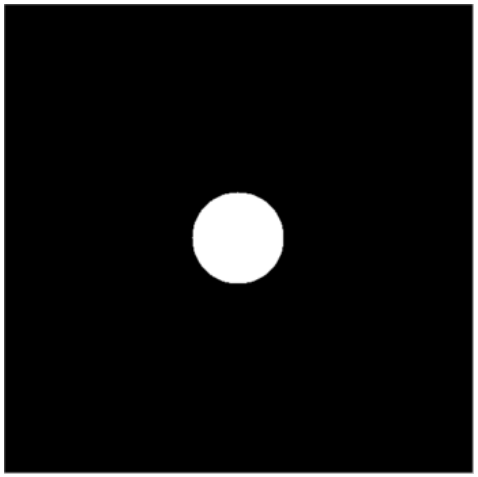
DFT

FFT: Beispiel 1

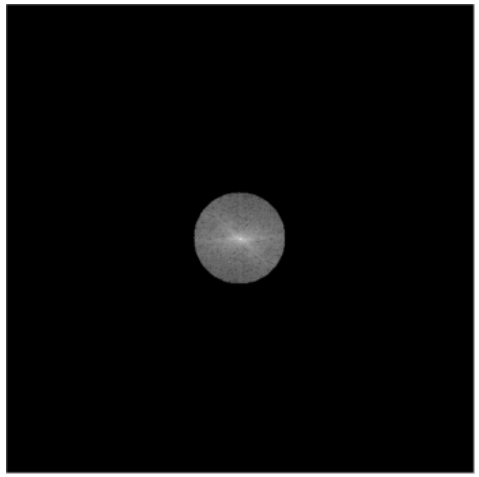
Original



DFT



Filter

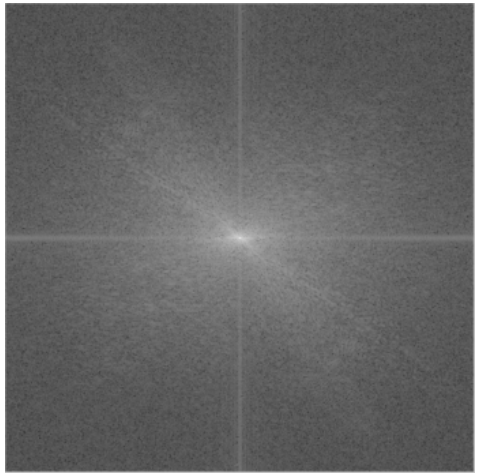


Filter·DFT

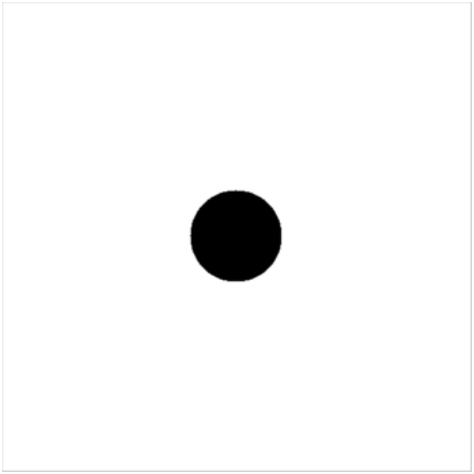
IDFT(Filter·DFT)

FFT: Beispiel 2

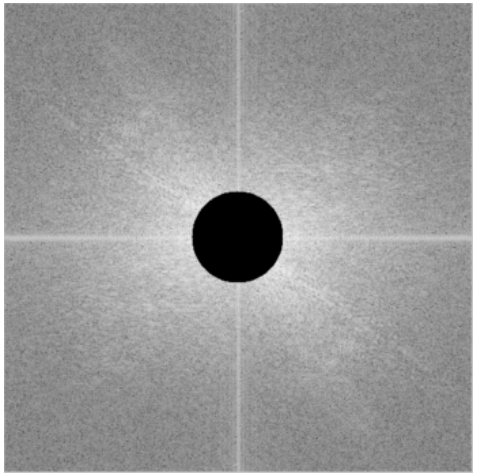
Original



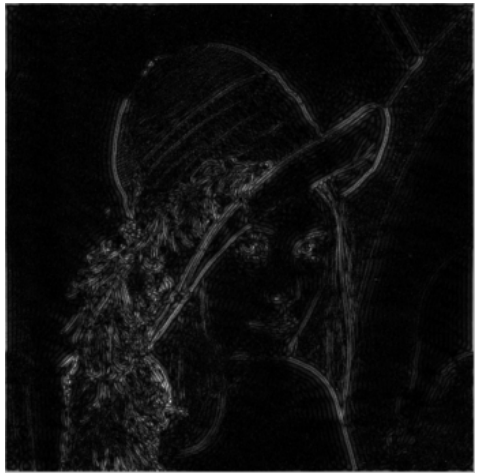
DFT



Filter



Filter·DFT



IDFT(Filter·DFT)

Zusammenfassung