

COMPUTERORIENTIERTE MATHEMATIK I



Christof Schütte

Wintersemester 2016/17

Computer-orientierte Mathematik

9. Vorlesung - Christof Schuette

16.12.16



Werkzeugkiste

Definitionen und Berechnungsmethoden zu Kondition und Stabilität

Auswertungsbäume zur systematischen Stabilitätsabschätzung

Von der Wurzel zu den Blättern: Auswertungsbaum

Von den Blättern zur Wurzel: rekursive Stabilitätsabschätzung

Blätter: Eingabewerte (Stabilität $\sigma = 1$),

Beispiele.

Polynom-Desaster revisited

Niedrige Auflösung (einfaches Modell zur Stabilitätsabschätzung) versus höhere Auflösung (komplexeres Modell zur Stabilitätsabschätzung).



bisher: Teil I

Auswirkung von (Rundungs)-Fehlern auf das Ergebnis:

Kondition eines Problems: Eingabefehler

Stabilität eines Algorithmus: Auswertungsfehler

bisher: Teil I

Auswirkung von (Rundungs)-Fehlern auf das Ergebnis:

Kondition eines Problems: Eingabefehler

Stabilität eines Algorithmus: Auswertungsfehler

jetzt: Teil II

Rechenaufwand:

Komplexität eines Problems

Effizienz eines Algorithmus



Wie lange muß ich auf das Ergebnis warten?

a) Ist das Problem schwierig?

(Komplexität)

b) Ist mein Algorithmus zu langsam?

(Effizienz)

Theoretische Informatik/Diskrete Mathematik:

→ Komplexitätstheorie, Berechenbare Funktionen



hängt ab von: Algorithmus, Eingabedaten ("Größe" des Problems)

Option 1: **Aufwandsmaß:** reine Rechenzeit

Nachteil: Abhängigkeit von zusätzlichen Parametern:

Implementierung, Programmiersprache, Compiler, Prozessor,...



hängt ab von: Algorithmus, Eingabedaten ("Größe" des Problems)

Option 1: **Aufwandsmaß:** reine Rechenzeit

Nachteil: Abhängigkeit von zusätzlichen Parametern:

Implementierung, Programmiersprache, Compiler, Prozessor,...

Option 2: **Aufwandsmaß:**

Anzahl dominanter Operationen



hängt ab von: Algorithmus, Eingabedaten ("Größe" des Problems)

Option 1: **Aufwandsmaß:** reine Rechenzeit

Nachteil: Abhängigkeit von zusätzlichen Parametern:

Implementierung, Programmiersprache, Compiler, Prozessor,...

Option 2: **Aufwandsmaß:**

Anzahl dominanter Operationen

Wir wollen allgemeingültige Resultate!



Problemklasse: Sortieren von n ganzen Zahlen

Problemgröße: n

Aufwandsmaß: Anzahl der Vergleiche

Laut Kapitel 2.3 des Skripts:

Merge-Sort: Anzahl der benötigten Vergleiche: $\leq n \log_2 n$



Definition: (Landau-Symbol \mathcal{O} und das Symbol Θ)

Seien $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+ = \{n \in \mathbb{R} \mid n \geq 0\}$ und $g : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ zwei Funktionen.

Wir schreiben: $f(n) = \mathcal{O}(g(n))$ für $n \rightarrow \infty$, falls

$$\exists C > 0 \text{ und } \exists n_0 \in \mathbb{R} \text{ mit } f(n) \leq C \cdot g(n) \text{ für alle } n \geq n_0$$



Definition: (Landau-Symbol \mathcal{O} und das Symbol Θ)

Seien $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+ = \{n \in \mathbb{R} \mid n \geq 0\}$ und $g : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ zwei Funktionen.

Wir schreiben: $f(n) = \mathcal{O}(g(n))$ für $n \rightarrow \infty$, falls

$$\exists C > 0 \text{ und } \exists n_0 \in \mathbb{R} \text{ mit } f(n) \leq C \cdot g(n) \text{ für alle } n \geq n_0$$

Wir schreiben: $f(n) = \Theta(g(n))$ für $n \rightarrow \infty$, falls

$$f(n) = \mathcal{O}(g(n)) \quad \text{und} \quad g(n) = \mathcal{O}(f(n))$$



Satz:

$$\text{a) } f(n) = O(g(n)) \text{ für } n \rightarrow \infty \iff \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c, \quad c \in \mathbb{R}$$

$$\text{b) } f(n) = \Theta(g(n)) \text{ für } n \rightarrow \infty \iff \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \neq 0$$



Definition:(Komplexität)

Sei $(P(n))$ eine Familie von Problemen, die für große $n \in \mathbb{N}$ gelöst werden sollen und \mathbf{A} die Menge aller Algorithmen, die $(P(n))$ lösen. Der Aufwand von einem Algorithmus $\mathcal{A} \in \mathbf{A}$ sei $L_{\mathcal{A}}(n)$. Dann heißt die größte Zahl $\mathcal{K}(n)$ mit

$$\mathcal{K}(n) \leq L_{\mathcal{A}}(n) \quad \forall \mathcal{A} \in \mathbf{A}$$

die Komplexität des Problems / der Problemfamilie (P) .



Definition: (Komplexität)

Sei $(P(n))$ eine Familie von Problemen, die für große $n \in \mathbb{N}$ gelöst werden sollen und \mathbf{A} die Menge aller Algorithmen, die $(P(n))$ lösen. Der Aufwand von einem Algorithmus $\mathcal{A} \in \mathbf{A}$ sei $L_{\mathcal{A}}(n)$. Dann heißt die größte Zahl $\mathcal{K}(n)$ mit

$$\mathcal{K}(n) \leq L_{\mathcal{A}}(n) \quad \forall \mathcal{A} \in \mathbf{A}$$

die Komplexität des Problems / der Problemfamilie (P) .

Definition (Effizienz) in der Numerischen Mathematik

Ein Algorithmus $\mathcal{A} \in \mathbf{A}$ heißt **effizient**, falls

$$L_{\mathcal{A}}(n) = \Theta(\mathcal{K}(n))$$



Definition: (Komplexität)

Sei $(P(n))$ eine Familie von Problemen, die für große $n \in \mathbb{N}$ gelöst werden sollen und \mathbf{A} die Menge aller Algorithmen, die $(P(n))$ lösen. Der Aufwand von einem Algorithmus $\mathcal{A} \in \mathbf{A}$ sei $L_{\mathcal{A}}(n)$. Dann heißt die größte Zahl $\mathcal{K}(n)$ mit

$$\mathcal{K}(n) \leq L_{\mathcal{A}}(n) \quad \forall \mathcal{A} \in \mathbf{A}$$

die Komplexität des Problems / der Problemfamilie (P) .

Definition (Effizienz) in der Numerischen Mathematik

Ein Algorithmus $\mathcal{A} \in \mathbf{A}$ heißt effizient, falls

$$L_{\mathcal{A}}(n) = \Theta(\mathcal{K}(n))$$

Bem.: Definition in der Theoretischen Informatik: $L_{\mathcal{A}}(n) = \mathcal{O}(n^p)$



Problemklasse: Sortieren von n ganzen Zahlen

Problemgröße: n

Aufwandsmaß: Anzahl der Vergleiche



Problemklasse: Sortieren von n ganzen Zahlen

Problemgröße: n

Aufwandsmaß: Anzahl der Vergleiche

Laut Kapitel 2.3 des Skripts:

Algorithmus Merge-Sort:

$$L_{\mathcal{A}}(n) \leq n \log_2 n$$

Komplexität:

$$\mathcal{K}(n) \geq \frac{1}{4} n \log_2 n$$



Problemklasse: Sortieren von n ganzen Zahlen

Problemgröße: n

Aufwandsmaß: Anzahl der Vergleiche

Laut Kapitel 2.3 des Skripts:

Algorithmus Merge-Sort:

$$L_{\mathcal{A}}(n) \leq n \log_2 n$$

Komplexität:

$$\mathcal{K}(n) \geq \frac{1}{4} n \log_2 n$$

Merge-Sort ist ein effizienter Algorithmus!



Definition: (größter gemeinsamer Teiler – ggT)

Eine Zahl d , die zwei ganze Zahlen $a, b \in \mathbb{N}$ teilt ($d|a$ und $d|b$), heißt **gemeinsamer Teiler** von a und b .

Die größte positive Zahl d , die gemeinsamer Teiler von a und b ist, heißt **größter gemeinsamer Teiler** von a und b oder kurz **$ggT(a, b)$** .

Problem (P):

Berechne $ggT(a, b)$ für Eingabedaten $a \geq b \in \mathbb{N} \setminus \{0\}$



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

$ggT := 1$

for $i = 2:b$

if $i|a$ and $i|b$

$ggT := i$

endif

endfor

return ggT



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

$ggT := 1$

for $i = 2:b$

if $i|a$ and $i|b$

$ggT := i$

endif

endfor

return ggT

Aufwandsmaß: Anzahl der Divisionen



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

$ggT := 1$

for $i = 2:b$

if $i|a$ and $i|b$

$ggT := i$

endif

endfor

return ggT

Aufwandsmaß: Anzahl der Divisionen

Aufwand: $2(b - 1)$



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

for $i := b:2$

if $i|a$ and $i|b$

return i

endif

endfor

return 1



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

for $i := b:2$

if $i|a$ and $i|b$

return i

endif

endfor

return 1

Aufwandsmaß: Anzahl der Divisionen



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

for $i := b:2$

if $i|a$ and $i|b$

return i

endif

endfor

return 1

Aufwandsmaß: Anzahl der Divisionen

Aufwandsschranke: $2(b - 1)$

**Definition** (Kongruenzen)

Der bei Division von a durch b bleibende Rest heißt

$$a \bmod b = a - \lfloor a/b \rfloor b .$$

$m, n \in \mathbb{Z}$ heißen **kongruent modulo b** , falls $m \bmod b = n \bmod b$.

**Definition** (Kongruenzen)

Der bei Division von a durch b bleibende Rest heißt

$$a \bmod b = a - \lfloor a/b \rfloor b .$$

$m, n \in \mathbb{Z}$ heißen **kongruent modulo b** , falls $m \bmod b = n \bmod b$.

Satz: (Rekursionsatz für den ggT)

Es gilt

$$\text{ggT}(a, b) = \text{ggT}(b, a \bmod b) \quad \forall a, b \in \mathbb{N} .$$



Aus der Definition $\text{mod}(a, b) = a - \lfloor a/b \rfloor b$ ergibt sich

$$a = \lfloor a/b \rfloor b + \text{mod}(a, b) .$$

Damit erhält man

$$d|a \text{ und } d|b \iff d|b \text{ und } d|\text{mod}(a, b) .$$

Die Mengen aller gemeinsamen Teiler von a , b und b , $\text{mod}(a, b)$ stimmen also überein und damit auch deren größtes Element, d.h.

$$\text{ggT}(a, b) = \text{ggT}(b, a \bmod b) \quad \forall a, b \in \mathbb{N} .$$



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

$m = a;$

$n = b;$

while $n > 0$

$r = m \text{ modulo } n$

$m = n$

$n = r$

endwhile

return m



Input: positive Zahlen $a \geq b > 0$

Output: $ggT(a,b)$

$m = a;$

$n = b;$

while $n > 0$

$r = m \text{ modulo } n$

$m = n$

$n = r$

endwhile

return m

Satz:

Der Euklidische Algorithmus liefert nach endlich vielen Schritten das richtige Ergebnis.

**Definition** (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} - F_n - F_{n-1} = 0, \quad F_0 = 0, \quad F_1 = 1$$

definierten Zahlen F_k , $k = 0, 1, \dots$ heißen **Fibonacci-Zahlen**.

**Definition** (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} - F_n - F_{n-1} = 0, \quad F_0 = 0, \quad F_1 = 1$$

definierten Zahlen F_k , $k = 0, 1, \dots$ heißen **Fibonacci-Zahlen**.

Lemma

Es sei $a > b$. Terminiert der Euklidische Algorithmus nach genau $k \geq 0$ Schritten, so gilt $a \geq F_{k+1}$ und $b \geq F_k$.



Induktionsanfang: Termination nach $k = 0$ Schritten impliziert $b = 0 = F_0$. Nach Voraussetzung ist dann $a > b = 0$ also $a \geq 1 = F_1$.

Induktionsschritt: Im ersten Schritt werden $m = b$ und $n = \text{mod}(a, b)$ berechnet. Nach Voraussetzung sind zur Bestimmung von $\text{ggT}(m, n)$ dann noch weitere $k - 1$ Schritte nötig. Nach Induktionsvoraussetzung gilt

$$b = m \geq F_k, \quad \text{mod}(a, b) = n \geq F_{k-1} .$$

Also ist $a \geq F_{k+1}$ zu zeigen. Aus $a > b > 0$ folgt $\lfloor a/b \rfloor \geq 1$ und damit

$$b + \text{mod}(a, b) = b + (a - \lfloor a/b \rfloor b) \leq a.$$

Daraus folgt

$$a \geq b + \text{mod}(a, b) \geq F_k + F_{k-1} = F_{k+1}.$$

**Definition** (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} - F_n - F_{n-1} = 0, \quad F_0 = 1, \quad F_1 = 1$$

definierten Zahlen F_k , $k = 0, 1, \dots$ heißen **Fibonacci-Zahlen**.

**Definition** (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} - F_n - F_{n-1} = 0, \quad F_0 = 1, \quad F_1 = 1$$

definierten Zahlen F_k , $k = 0, 1, \dots$ heißen **Fibonacci-Zahlen**.

Lemma

Es sei $a > b$. Terminiert der Euklidische Algorithmus nach genau $k \geq 1$ Schritten, so gilt $a \geq F_{k+1}$ und $b \geq F_k$.

**Definition** (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} - F_n - F_{n-1} = 0, \quad F_0 = 1, \quad F_1 = 1$$

definierten Zahlen F_k , $k = 0, 1, \dots$ heißen **Fibonacci-Zahlen**.

Lemma

Es sei $a > b$. Terminiert der Euklidische Algorithmus nach genau $k \geq 1$ Schritten, so gilt $a \geq F_{k+1}$ und $b \geq F_k$.

Satz (Lamé, 1848)

Gilt $a > b$ und $b < F_{k+1}$ mit $k \in \mathbb{N}$, so terminiert der Euklidische Algorithmus nach höchstens k Schritten.

**Definition** (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} - F_n - F_{n-1} = 0, \quad F_0 = 1, \quad F_1 = 1$$

definierten Zahlen F_k , $k = 0, 1, \dots$ heißen **Fibonacci-Zahlen**.

Lemma

Es sei $a > b$. Terminiert der Euklidische Algorithmus nach genau $k \geq 1$ Schritten, so gilt $a \geq F_{k+1}$ und $b \geq F_k$.

Satz (Lamé, 1848)

Gilt $a > b$ und $b < F_{k+1}$ mit $k \in \mathbb{N}$, so terminiert der Euklidische Algorithmus nach höchstens k Schritten.

Bemerkung:

Im Falle $b = F_k$, $a = F_{k+1}$ braucht man k Schritte.



Formel von Moivre-Binet

$$F_k = \frac{1}{\sqrt{5}} \left(\phi^k - (1 - \phi)^k \right), \quad \phi = \frac{1 + \sqrt{5}}{2}.$$

Satz:

Der Euklidische Algorithmus terminiert nach höchstens $\log_{\phi}(b) + 1$ Schritten.



Formel von Moivre-Binet

$$F_k = \frac{1}{\sqrt{5}} \left(\phi^k - (1 - \phi)^k \right), \quad \phi = \frac{1 + \sqrt{5}}{2}.$$

Satz:

Der Euklidische Algorithmus terminiert nach höchstens $\log_{\phi}(b) + 1$ Schritten.

Bemerkung:

Das ist ein qualitativer Sprung gegenüber dem naiven Algorithmus!

Beispiel: $\log_{\phi}(10.000) < 20 \ll 10.000$



Lesen Sie Abschnitt 2.3 im Skript!