Stabilitätsabschätzungen Vorlesung vom 8.12.17

Auswertungsbäume zur systematischen Stabilitätsabschätzung

Auswertungsbaum: Knoten, gerichtete Kanten, Wurzel, Blätter

Zerlegung in Teilbäume

Von den Blättern zur Wurzel:

rekursive Funktionsauswertung und Stabilitätsabschätzung

Theoretische Grundlagen: Satz 7.6 und Satz 7.9

Beispiele.

Summationsalgorithmen

Rekursive Summation, Auswertungsbaum, Stabilitätsanalyse

Hierarchische Summation.

Kondition: Klassifizierung von Problemen

Problem: Auswertung von $f(x)$

Definition: Auswirkung von Eingabefehlern auf das Ergebnis (Formel!)

Ausrechnen: Formeln!

Bedeutung: Grenzen der Genauigkeit (Formel!)

Beispiele

Stabilität: Klassifizierung von Algorithmen

Algorithmus: $f(x) = g_n \circ \cdots g_2 \circ g_1(x)$

Definition: Auswirkung von sukzessivem Runden auf das Ergebnis (Formel!)

Elementarfunktionen: Modell für den tatsächlichen Ablauf im Rechner

Ausrechnen: Formeln!

Bedeutung: vermeidbare(?) Grenzen der Genauigkeit (Formel!)

Beispiele

Weitere Klassifizierung: Komplexität und Effizienz

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

Aufwand (Laufzeit) eines Algorithmus

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

Aufwandsmaß: reine Rechenzeit

zusätzliche Parameter:

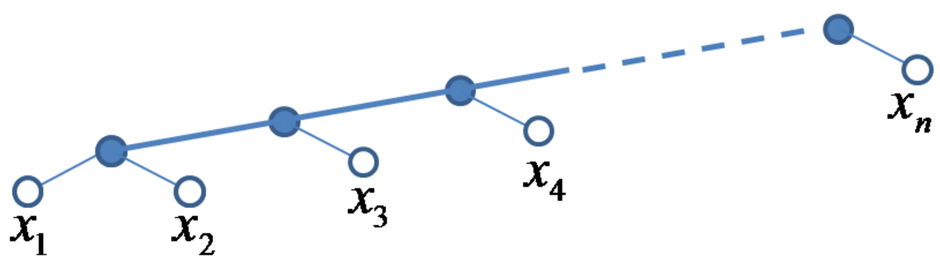
Implementierung, Programmiersprache, Compiler, Prozessor,...

Aufwandsmaß: Anzahl dominanter Operationen (problemabhängig!)

implementierungsunabhängige Resultate

Rekursive Summation positiver Zahlen

rekursive Summation: `S = a[1]; for i=2:1:m S = S + a[i]; end`



Aufwandsmaß: Anzahl der Additionen

Aufwand des rekursiven Algorithmus: $n - 1$

Hierarchische Summation

$$
\begin{array}{ccccccccccccccc}
x_1 & & x_2 & & x_3 & & x_4 & & x_5 & & x_6 & & x_7 & & x_8 \\
& \searrow\swarrow & & & & \searrow\swarrow & & & & \searrow\swarrow & & & & \searrow\swarrow & \\
& s_1^{(1)} & & & & s_2^{(1)} & & & & s_3^{(1)} & & & & s_4^{(1)} & \\
& & \searrow & & \swarrow & & & & & & \searrow & & \swarrow & & \\
& & & s_1^{(2)} & & & & & & & & s_2^{(2)} & & & \\
& & & & \searrow & & & & & & \swarrow & & & & \\
& & & & & & & s_1^{(3)} & & & & & & &
\end{array}
$$

Aufwandsmaß: Anzahl der Additionen

Aufwand des hierarchischen Algorithmus: $(n = 2^J)$:

$$2^{J-1} + 2^{J-2} + \cdots + 1 = \sum_{j=0}^{J-1} 2^j = \frac{2^J - 1}{2 - 1} = 2^J - 1 = n - 1$$

Komplexität der Summation positiver Zahlen

Kann es einen Algorithmus geben, der weniger als $n-1$ Additionen benötigt?

Nein!

Folgerung: Die Komplexität der Addition von n Zahlen ist $n - 1$.

Aufwand und Komplexität

Definition 2.4

Problem (P), Algorithmen $\mathcal{A}(P)$ zur Lösung von (P)

Eingabedaten der Länge n, Referenzoperation (problemspezifisch!)

Der Aufwand $T_A(n)$ eines Algorithmus $A \in \mathcal{A}(P)$ ist das

Maximum der benötigten Anzahl von Referenzoperationen
über alle zulässigen Eingabedaten der Länge n.

Definition 2.5

Die Komplexität $\mathcal{K}_P(n)$ von (P) ist

$$\mathcal{K}_P(n) = \inf_{A \in \mathcal{A}(P)} T_A(P)$$

Effiziente Algorithmen

Problem (P), Algorithmus $A \in \mathcal{A}(P)$ zur Lösung von (P)

Effizienz eines Algorithmus' A:

Theoretische Informatik: $T_A(n) = O(n^p)$

Numerische Mathematik: $T_A(n) = O(\mathcal{K}_P(n))$

Definition: Landau-Symbol $O(\cdot)$ für $f(n)$, $g(n) \to \infty$ für $n \to \infty$

$$f(n) = O(g(n)) \quad \text{für } n \to \infty \qquad \Longleftrightarrow \qquad \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

Beispiel: $18n^3 + 3n^2 + \sin(e^n) = O(n^3)$

Sortieren

gegeben: $z_1,\ z_2,\ \ldots,\ z_n \in \mathbb{R}$

gesucht: Permutation (Umordnung) π: $z_{\pi(1)} \leq z_{\pi(2)} \leq \cdots \leq z_{\pi(n)}$

Algorithmus: TumbSort: Alle Umordnungen durchprobieren.

Aufwandsmaß: Vergleich zweier Zahlen

Aufwand von TumbSort: $O((n-1)n^n)$

BubbleSort

Algorithmus: BubbleSort: Sukzessive Maximierung

Beispiel:

$$
\begin{array}{lll}
S = \{7,1,5\}\ ,\ k=3: & SMax = \max\{7,1,5\} = 7, & iMax = 1 \\
S = \{5,1,7\}\ ,\ k=2: & SMax = \max\{5,1\} = 5, & iMax = 1 \\
S = \{1,5,7\} & &
\end{array}
$$

Aufwand von BubbleSort:

$$
(n-1)+(n-2)+\cdots+1 = \sum_{k=1}^{n-1} k = \tfrac{1}{2}n(n-1) = O(n^2) \ll O(n^n)
$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 1$$

$$S = S_x \cup S_y$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 1$$

$$S = S_x \cup S_y = \{1,$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 7$$

$$S = S_x \cup S_y = \{1, 2, 3, 4, 6, 7,$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 7$$

$$S = S_x \cup S_y = \{1, 2, 3, 4, 6, 7, 8,$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 7$$

$$S = S_x \cup S_y = \{1, 2, 3, 4, 6, 7, 8,$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 7$$

$$S = S_x \cup S_y = \{1, 2, 3, 4, 6, 7, 8, 9\}$$

Strukturelle Einsicht

Lemma 2.7: Gegeben sind zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_m\}$$

Dann lässt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Reißverschluss-System:

$$S_x = \{1, 3, 6, 9\}, \qquad S_y = \{2, 4, 7, 8\} \qquad \text{Zähler:} \quad 7$$

$$S = S_x \cup S_y = \{1, 2, 3, 4, 6, 7, 8, 9\}$$

MergeSort

$$
\begin{array}{ccccccccccccccc}
 & & & & & & & U & & & & & & & \\
 & & & \swarrow & & & & & & & & \searrow & & & \\
 & & & U_1^{(1)} & & & & & & & & U_2^{(1)} & & & \\
 & \swarrow & & & & \searrow & & & & \swarrow & & & & \searrow & \\
 & U_1^{(2)} & & & & U_2^{(2)} & & & & U_3^{(2)} & & & & U_4^{(2)} & \\
 & \swarrow\searrow & & & & \swarrow\searrow & & & & \swarrow\searrow & & & & \swarrow\searrow & \\
U_1^{(3)} & & U_2^{(3)} & & U_3^{(3)} & & U_4^{(3)} & & U_5^{(3)} & & U_6^{(3)} & & U_7^{(3)} & & U_8^{(3)}
\end{array}
$$

Aufwand von MergeSort: $\quad n = 2^J$

$$
\begin{aligned}
T_{\mathrm{MergeSort}}(n) &\leq 1 \cdot 2^{J-1} + 4 \cdot 2^{J-2} + \cdots + 2^J \cdot 1 \\
&\leq 2 \cdot 2^{J-1} + 2^2 \cdot 2^{J-2} + \cdots + 2^J \cdot 1 \leq n \log_2 n
\end{aligned}
$$

optimal!

Berechnung von $ggT(a, b)$

Definition 4.1 (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 $a > b \in N$

Ausprobieren: TumbGGT (Algorithmus 4.3)

Eingabe: a, $b \in \mathbb{N}$ mit $a > b \geq 0$

```
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 mit Rest Aufwand: $2(b-1)$

Geschickteres Ausprobieren: TumbGGT++

Eingabe: $a,\ b \in \mathbb{N}$ mit $a > b \geq 0$

```
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: $2(b-1)$

Strukturelle Einsicht

Definition 4.6 (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 $(m \equiv n)$, falls $m \bmod b = n \bmod b$.

modulo-Funktion: $\mathsf{mod}(a, b) = a \bmod b$

Lemma 4.12 (Rekursionssatz für den ggT)
Es gilt

$$ggT(a, b) = ggT(b, \mathsf{mod}(a, b)) \qquad \forall a, b \in \mathbb{N} \; .$$

Umsetzung: Der Euklidische Algorithmus

Eingabe: $a, b \in \mathbb{N}$ mit $a > b \geq 0$

```
Output: ggT(a,b)

m = a;
n = b;
while n>0
    r = m modulo n
    m = n
    n = r
endwhile

return m
```

Satz: Der Euklidische Algorithmus terminiert nach höchstens b Schritten.

Aufwandsanalyse

Definition (Fibonacci–Zahlen)
Die durch die Drei–Term–Rekursion

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

definierten Zahlen F_k, $k = 0, 1, \ldots$, heißen Fibonacci–Zahlen.

Lemma 4.14
Es sei $a > b \geq 0$. Terminiert der Euklidische Algorithmus nach genau $k \geq 1$ Schritten, so gilt $a \geq F_{k+1}$ und $b \geq F_k$.

Satz 4.15 (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.

Aufwandsschranke für den Euklidischen Algorithmus

Moivre-Binet: $F_k = (\phi^k - (1-\phi)^k)/\sqrt{5} \quad \phi = \frac{1+\sqrt{5}}{2}$

Satz 4.16
Für den Aufwand $T_{\mathrm{E}}(n)$ des Euklidische Algorithmus
bei Anwendung auf eine n-stellige Zahl b und $a > b \geq 0$ gilt

$$T_{\mathrm{E}}(n) \leq 1 + \log(\phi)^{-1} n$$

Bemerkung:

Das ist eine exponentielle Verbesserung gegenüber TUMBGGT!

Beispiel: n$=\log(10.000) = 4 \ll 10.000$

Primzahltests

Problem: Ist eine gegebene Zahl $a \in \mathbb{N}$ eine Primzahl?

Anwendungen: Kryptographie (RSA Algorithmus)

Polynomielle Laufzeit:

AKS-Primzahltest $\mathcal{O}(n^{7,5+\varepsilon})$

mit $n = \log(a)$ und $\varepsilon > 0$.

Alternativen: stochastische Algorithmen,...