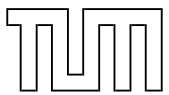
Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2010



Übersicht

1. Effizienz
   - Laufzeitanalyse
   - Durchschnittliche Laufzeit

Beispiel: Bresenham-Algorithmus

Algorithmus 11 : Bresenham1

$(x, y) = (0, R)$;	$\mathcal{O}(1)$
plot$(0, R)$; plot$(R, 0)$; plot$(0, -R)$; plot$(-R, 0)$;	$\mathcal{O}(1)$
$F = \frac{5}{4} - R$;	$\mathcal{O}(1)$
while $x < y$ **do**	$\mathcal{O}(\sum_{i=1}^{k} T(l))$
if $F < 0$ **then**	
$F = F + 2x + 1$;	
else	
$F = F + 2x - 2y + 2$;	alles $\mathcal{O}(1)$
$y = y - 1$;	
$x = x + 1$;	
plot(x, y); plot(y, x); plot$(-x, y)$; plot$(y, -x)$;	
plot$(x, -y)$; plot$(-y, x)$; plot$(-x, -y)$; plot$(-y, -x)$;	

$$\mathcal{O}\left(\sum_{i=1}^{k} 1\right) = \mathcal{O}(k)$$

Wie groß ist Anzahl Schleifendurchläufe k?

Beispiel: Bresenham-Algorithmus

- Betrachte dazu die Entwicklung der Wert der Funktion

$$\varphi(x, y) = y - x$$

- Anfangswert: $\varphi_0(x, y) = R$
- Monotonie: verringert sich pro Durchlauf um mindestens 1
- Beschränkung: durch die **while**-Bedingung $x < y$ bzw. $0 < y - x$

$\Rightarrow$ maximal R Runden

Beispiel: Fakultätsfunktion

Algorithmus 12 : fakultaet(n)

Input : $n \in \mathbb{N}$
Output : $n!$

if $(n == 1)$ **then**	$\mathcal{O}(1)$
⌊ **return** 1	$\mathcal{O}(1)$
else	
⌊ **return** $n * fakultaet(n-1)$	$\mathcal{O}(1 + \ldots?)$

- $T(n)$: Laufzeit von fakultaet(n)
- $T(1) = \mathcal{O}(1)$
- $T(n) = T(n-1) + \mathcal{O}(1)$

$\Rightarrow$ $T(n) = \mathcal{O}(n)$

Average Case Complexity

Uniforme Verteilung:
(alle Instanzen gleichwahrscheinlich)

$$t(n) = \frac{1}{|I_n|} \sum_{i \in I_n} T(i)$$

Tatsächliche Eingabeverteilung kann in der Praxis aber stark von uniformer Verteilung abweichen.

Dann

$$t(n) = \sum_{i \in I_n} p_i \cdot T(i)$$

Aber: meist schwierig zu berechnen!

Beispiel: Binärzahl-Inkrementierung

Algorithmus 13 : increment(A)

Input : Array A mit Binärzahl in $A[0] \ldots A[n-1]$,
in $A[n]$ steht eine 0

Output : inkrementierte Binärzahl in $A[0] \ldots A[n]$

$i = 0;$
while $(A[i] == 1)$ **do**
  $A[i] = 0;$
  $i = i + 1;$
$A[i] = 1;$

Durchschnittliche Laufzeit für Zahl mit n Bits?

Beispiel: Binärzahl-Inkrementierung

Analyse:

- Sei I_n die Menge der n-Bit-Instanzen.
- Für die Hälfte ($1/2$) der Zahlen $x_{n-1} \ldots x_0 \in I_n$ ist $x_0 = 0$

$\Rightarrow$ 1 Schleifendurchlauf

- Für die andere Hälfte gilt für die Hälfte (also insgesamt $1/4$) der Zahlen $x_1 x_0 = 01$

$\Rightarrow$ 2 Schleifendurchläufe

- Für den Anteil $(1/2)^k$ der Zahlen gilt $x_{k-1} x_{k-2} \ldots x_0 = 01 \ldots 1$

$\Rightarrow$ k Schleifendurchläufe

Beispiel: Binärzahl-Inkrementierung

Durchschnittliche Laufzeit:

$$
\begin{aligned}
t(n) &= \frac{1}{|I_n|} \sum_{i \in I_n} T(i) \\
&= \frac{1}{|I_n|} \sum_{k=1}^{n} \frac{|I_n|}{2^k} \cdot O(k) \\
&= \sum_{k=1}^{n} O\left(\frac{k}{2^k}\right) \\
&= O\left(\sum_{k=1}^{n} \frac{k}{2^k}\right) \stackrel{?}{=} O(1)
\end{aligned}
$$

Beispiel: Binärzahl-Inkrementierung

Lemma

$$\sum_{k=1}^{n} \frac{k}{2^k} \leq 2 - \frac{n+2}{2^n}$$

Beweis.

Induktionsanfang: Für $n = 1$ gilt:

$$\sum_{k=1}^{1} \frac{k}{2^k} = \frac{1}{2} \leq 2 - \frac{1+2}{2^1} \qquad \checkmark$$

Induktionsvoraussetzung: Für n gilt:

$$\sum_{k=1}^{n} \frac{k}{2^k} \leq 2 - \frac{n+2}{2^n}$$

Beispiel: Binärzahl-Inkrementierung

Beweis.

Induktionsschritt: $n \to n+1$

$$\begin{aligned}
\sum_{k=1}^{n+1} \frac{k}{2^k} &= \sum_{k=1}^{n} \frac{k}{2^k} + \frac{n+1}{2^{n+1}} \\
&\leq 2 - \frac{2+n}{2^n} + \frac{n+1}{2^{n+1}} \qquad \text{(laut Ind.vor.)} \\
&= 2 - \frac{4+2n}{2^{n+1}} + \frac{n+1}{2^{n+1}} = 2 - \frac{4+2n-n-1}{2^{n+1}} \\
&= 2 - \frac{n+3}{2^{n+1}} \\
&= 2 - \frac{(n+1)+2}{2^{n+1}}
\end{aligned}$$

□

Beispiel: Suche in selbstorganisierender Liste

Move-to-Front Rule:

Verschiebe nach jeder erfolgreichen Suche das gefundene Element an den Listenanfang

Bsp.: Ausführung von search(4) ergibt

Beispiel: Suche in selbstorganisierender Liste

Analyse:

- l_n Probleminstanzen bestehend aus n search-Operationen
- s_i Position des Elements i in der Liste (1 bedeutet am Anfang)
- p_i Wahrscheinlichkeit für Operation search(i)

Erwartete Laufzeit der Operation search(i) mit zufälligem i:

$$\mathbb{E}(T(1 \times \text{search}(i))) = O\left(\sum_i p_i s_i\right)$$

Erwartete Laufzeit $t(n)$ bei statischer Liste:

$$t(n) = \mathbb{E}(T(n \times \text{search}(i))) = O\left(n \sum_i p_i s_i\right)$$

Beispiel: Suche in selbstorganisierender Liste

Erwartete Laufzeit $t(n)$ bei dynamischer Liste:

$$\mathbb{E}(T(1 \times \text{search}(i))) = O\left(\sum_i p_i \cdot \mathbb{E}[s_i]\right)$$

$$t(n) = \mathbb{E}(T(n \times \text{search}(i))) = O\left(\sum_{j=1}^{n} \sum_i p_i \cdot \mathbb{E}[s_i]\right)$$

Beispiel: Suche in selbstorganisierender Liste

Optimale Anordnung?

$\Rightarrow$ wenn für alle Elemente i, j mit $p_i > p_j$ gilt, dass $s_i < s_j$, d.h. die Elemente nach Zugriffswahrscheinlichkeit sortiert sind

o.B.d.A. seien die Indizes so, dass $p_1 \geq p_2 \geq \ldots \geq p_m$

- Optimale Anordnung: $s_i = i$
- Optimale erwartete Laufzeit: $\text{opt} = \sum_i p_i \cdot i$

Problem: wir kennen die Wahrscheinlichkeiten nicht!

Satz

Die erwartete Laufzeit für n search-Operationen bei Verwendung der Move-to-Front Rule ist maximal 2opt für genügend große n.

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Betrachte zwei feste Elemente i und j

t aktuelle Operation

t_0 letzte Suchoperation auf i oder j

- bedingte Wahrscheinlichkeit: $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$
- $\Pr[A \mid (A \vee B)] = \frac{\Pr[A \wedge (A \vee B)]}{\Pr[A \vee B]} = \frac{\Pr[A]}{\Pr[A \vee B]}$
- $\Pr[\text{search}(j) \text{ bei } t_0 \mid \text{search}(i \vee j) \text{ bei } t_0] = \frac{p_j}{p_i + p_j}$

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Betrachte festes Element i

- Definiere Zufallsvariablen $X_j \in \{0, 1\}$:

$$X_j = 1 \quad \Leftrightarrow \quad j \text{ vor } i \text{ in der Liste}$$

- Erwartungswert:

$$\begin{aligned} \mathbb{E}[X_j] &= 0 \cdot \Pr[X_j = 0] + 1 \cdot \Pr[X_j = 1] \\ &= \Pr[\text{letzte Suche nach } j] \\ &= \frac{p_j}{p_i + p_j} \end{aligned}$$

Beispiel: Suche in selbstorganisierender Liste

Beweis.

- Listenposition von Element i:

$$1 + \sum_{j \neq i} X_j$$

- Erwartungswert der Listenposition von Element i:

$$\begin{aligned}
\mathbb{E}[s_i] &= \mathbb{E}\left[1 + \sum_{j \neq i} X_j\right] \\
&= 1 + \mathbb{E}\left[\sum_{j \neq i} X_j\right] = 1 + \sum_{j \neq i} \mathbb{E}\left[X_j\right] \\
\mathbb{E}[s_{i,\mathrm{MTF}}] &= 1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j}
\end{aligned}$$

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Erwartete Laufzeit für Operation t für genügend großes t:

$$
\begin{aligned}
\mathbb{E}[T_{\mathrm{MTF}}] &= \sum_i p_i \left(1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j}\right) \\
&= \sum_i \left(p_i + \sum_{j \neq i} \frac{p_i p_j}{p_i + p_j}\right) = \sum_i \left(p_i + 2 \sum_{j<i} \frac{p_i p_j}{p_i + p_j}\right) \\
&= \sum_i p_i \left(1 + 2 \sum_{j<i} \frac{p_j}{p_i + p_j}\right) \leq \sum_i p_i \left(1 + 2 \sum_{j<i} 1\right) \\
&\leq \sum_i p_i \cdot (2i - 1) < \sum_i p_i \cdot 2i = 2 \cdot \mathrm{opt}
\end{aligned}
$$

□