

Vorlesung

Effiziente Algorithmen und Komplexitätstheorie

Sommersemester 2008

Ingo Wegener

Raumverlegung

Die nächste Vorlesung (Montag, 05.01.) findet **vermutlich** (≤ 40 Personen) in OH14, E04 statt

Bitte vorher nochmal auf

<http://ls2-www.cs.uni-dortmund.de/lehre/sommer2008/ea/nachschauen>.

Was bisher geschah...

- starke Zusammenhangskomponenten ✓
- Matchings ✓
- Flussproblem ✓

jetzt Amortisierte Analyse

Zwischenspiel Amortisierte Analyse

jetzt kurze Pause von **Algorithmen**
etwas **Allgemeines** zur Analyse

Betrachte Datenstruktur D mit Operationen

übliche Worst-Case-Analyse bei n Daten in D
Worst-Case-Rechenzeit $t(n)$ je Operation

klar Gesamtrechenzeit für m Operationen
trivial durch $m \cdot t(n)$ nach oben beschränkt

alternativ bei $\leq n$ Daten in D und m Operationen
Worst-Case-Gesamtrechenzeit $t(n, m)$

Hoffnung $t(n, m) \ll m \cdot t(n)$

Notation $t(n, m)/m =$ **amortisierte Rechenzeit** einer Operation

Methoden und Anwendungsbeispiel

Plan drei Methoden zur amortisierten Analyse
abstrakt besprechen und
konkret anschauen am **Beispiel „Binärzähler“**

Beispiel Binärzähler

Aufgabe zu $z \in \mathbb{N}_0$ speichere Binärdarstellung $(z_{l-1}z_{l-2} \dots z_1z_0)_2$

$$\text{mit } z = \sum_{i=0}^{l-1} 2^i z_i$$

Operationen SetzeNull() und PlusEins()

Implementierung lineare Liste
 Listenanfang z_0 ; mit z_i Zeiger auf z_{i+1} verwaltet

Vereinbarung Manipulation eines Listeneintrags in Zeit 1

klar SetzeNull() in Zeit 1
 PlusEins() bis zu Zeit l (11111 \rightsquigarrow 100000)

also Worst-Case-Zeit von m Operationen $O(m \cdot \log m)$

Amortisierte Analyse mit der Potenzialmethode

Betrachte Folge von Datenstrukturen D_i und Operationen Op_i

$$D_0 \xrightarrow{\text{Op}_1} D_1 \xrightarrow{\text{Op}_2} D_2 \xrightarrow{\text{Op}_3} D_3 \xrightarrow{\text{Op}_4} \dots \xrightarrow{\text{Op}_m} D_m$$

T_i tatsächliche Rechenzeit von Op_i

Definiere **Potenzialfunktion** $\Phi: \mathcal{D} \rightarrow \mathbb{R}_0^+$

mit $\Phi(D_0) = 0$

Festlegung amort. Rechenzeit von $\text{Op}_i = T_i + \Phi(D_i) - \Phi(D_{i-1})$

Rechtfertigung

$$\begin{aligned} \sum_{i=1}^m T_i + \Phi(D_i) - \Phi(D_{i-1}) &= \left(\sum_{i=1}^m T_i \right) + \left(\sum_{i=1}^m \Phi(D_i) \right) - \left(\sum_{i=1}^m \Phi(D_{i-1}) \right) \\ &= \left(\sum_{i=1}^m T_i \right) + \Phi(D_m) - \Phi(D_0) \geq \sum_{i=1}^m T_i \end{aligned}$$

Beispiel Binärzähler mit Potenzialmethode

Definition $\Phi(z) := \sum_{i=0}^{l-1} z_i$

Bestimmung von $T_i + \Phi(D_i) - \Phi(D_{i-1})$

1. Fall vorher $z_0 = 0$

Beobachtungen $T_i = 1 \quad \Phi(D_i) = \Phi(D_{i-1}) + 1$

also $T_i + \Phi(D_i) - \Phi(D_{i-1}) = 2$

2. Fall vorher $z_0 = z_1 = \dots = z_k = 1 \quad (0 \leq k < l)$

Beobachtung $T_i = k + 1 \quad \Phi(D_i) = \Phi(D_{i-1}) - k + 1$

also $T_i + \Phi(D_i) - \Phi(D_{i-1}) = k + 1 - k + 1 = 2$

also Gesamtrechenzeit bei m Operationen $O(m)$

Beobachtung asymptotisch exakt

Amortisierte Analyse mit Kostenverteilung

Definiere **Kostenträger**
zum Beispiel Operationen, Teile der Datenstruktur, ...

Für jede Operation verteile reale Laufzeit T_i
beliebig auf Kostenträger

Berechne Gesamtkosten durch Addition über Kostenträger

Anmerkung schon gesehen beim Beweis von Lemma 4.17
Sperrflussberechnung mit Forward-Backward-Propagation
Verteilung der Laufzeit in der While-Schleife in Forward
auf betrachtete Kanten

Beispiel Binärzähler mit Kostenverteilung

Definiere verteile Kosten k einer PlusEins-Operation gleichmäßig auf die k berührten Stellen

Beobachtung Ziffer z_i in jeder (2^{i+1}) -ten Operation
also bei m Operationen
 Kosten $\lceil m/2^i \rceil$ bei Kostenträger z_i

also Gesamtkosten

$$\sum_{i=0}^{l-1} \lceil \frac{m}{2^i} \rceil$$

$$< l + m \cdot \sum_{i=0}^{l-1} 2^{-i} < 2m + l = O(m)$$

Beobachtung auch asymptotisch exakt

Amortisierte Analyse mit der Kontenmethode

Definiere **Konten**

zum Beispiel Operationen, Teile der Datenstruktur, ...

nach Belieben

- bei Operation Op_i mit Rechenzeit T_i auf Konto Betrag $b_i \in \mathbb{N}$ einzahlen und für Operation Rechenzeit $T_i + b_i$ als amortisierte Rechenzeit berücksichtigen
- bei Operation Op_i mit Rechenzeit T_i von Konto mit Kontostand $b \in \mathbb{N}_0$ Betrag $b_i \in \mathbb{N}$ mit $b_i \leq b$ abbuchen und für Operation Rechenzeit $T_i - b_i$ als amortisierte Rechenzeit berücksichtigen

Rechtfertigung \rightsquigarrow obere Schranke,
weil nie mehr abgehoben als eingezahlt

Beispiel Binärzähler mit der Kontenmethode

Definiere Konten $\hat{=}$ Stellen z_i

Definiere **Einzahlung** 1 für „0 \rightsquigarrow 1“

Beobachtung jetzt Kosten $1 + 1 = 2$ je 0-Stelle

Definiere **Abbuchung** 1 für „1 \rightsquigarrow 0“

Beobachtung korrekt, weil immer erst 1, dann 0

Beobachtung jetzt Kosten $1 - 1 = 0$ je 1-Stelle

Beobachtung amortisierte Kosten 2 für jedes PlusEins

also Gesamtkosten für m Operationen $\leq 2m$

Beobachtung auch asymptotisch exakt

Union-Find mit Pfadkompression („Nachlese“ aus DAP 2)

aus dem DAP 2-Skript (Wegener, Sommer 2007)

2.8 Datenstrukturen für Partitionen

n Objekte $1, \dots, n$ initial in n Mengen $1, \dots, n$

Operationen

- $\text{FIND}(x)$ Bestimme Namen der Menge, die x enthält.
- $\text{UNION}(A, B)$ Vereinige Mengen A und B .

baumorientierte DS für schnelle UNION

- Menge: wurzelgerichteter Baum, Name und Größe an Wurzel
- FIND sucht Namen an der Wurzel
- UNION: Wurzel kleinere Menge an Wurzel größere Menge

Pfadkompression (path compression)

- bei FIND Pfad zur Wurzel speichern
- dann alle Zeiger auf Wurzel umbiegen

Unsere Aufgabe. . .

aus dem DAP 2-Skript, S. 49:

Die Abschätzung der Kosten für m FIND-Befehle bleibt späteren Vorlesungen vorbehalten. Wir stellen nur das Ergebnis vor.

Definition 2.8.4: *Die Funktion Zweierturm Z ist rekursiv definiert durch*

$$Z(0) := 1 \text{ und } Z(i) := 2^{Z(i-1)},$$

dazu sei $\log^ n := \min\{k \mid Z(k) \geq n\}$.*

[. . .]

Satz 2.8.5: *Mit Hilfe der baumorientierten Datenstruktur und der Technik der Pfadkompression kann jede Folge von $n - 1$ UNION- und m FIND-Befehlen in Zeit $O((n + m) \log^* n)$ ausgeführt werden.*

Ein Wort zur Implementierung

Datenstrukturen Array $A[1..n]$ für Bäume
 Array $G[1..n]$ für Größe

initial $\forall i: A[i] = i$ { * zeigt Wurzel *}
 $G[i] = 1$

Union(i, j) If $G[i] > G[j]$ oder ($G[i] = G[j]$ und $i > j$)
 Then Vertausche i und j .
 $A[i] := j; G[j] := G[j] + G[i]$

Find(i) $k := i$; While $A[k] \neq k$
 $k := A[k]$
 $w := k; k := i$; While $A[k] \neq k$
 $k := A[k]; A[k] := w$
 Exit mit Ausgabe w

Über den Sinn von Pfadkompression

klar Union immer in Zeit $\Theta(1)$

klar Find (ohne und mit Pfadkompression) in Zeit $\Theta(l)$
 $l =$ Länge des Weges von i zur Wurzel

Beobachtung Bäume mit Tiefe $\Theta(\log n)$ können entstehen

also ohne Pfadkompression für $n - 1$ Union und m Find
Gesamtzeit $\Omega(n + m \cdot \log n)$ möglich

klar erstes Find(i) wird
Pfadkompression langsamer
aber nicht asymptotisch

Hoffnung zukünftige Find-Befehle schneller

Nochmal unser Ziel

Definition 5.1

$Z(0) := 1, Z(i) := 2^{Z(i-1)}$ für $i \in \mathbb{N}$

$\log^* n := \min\{k \mid Z(k) \geq n\}$

Theorem 5.2

Mit der Datenstruktur mit wurzelgerichteten Bäumen und Pfadkompression kann eine Folge von bis zu $n - 1$ Union-Befehlen und m Find-Befehlen in Zeit $O((n + m) \log^* n)$ ausgeführt werden.

Plan

- 1 Vorarbeit \rightsquigarrow wichtige Einsichten in Problemstruktur
- 2 geschickte amortisierte Analyse mit Kostenverteilung

Tiefe und Elementanzahlen

Lemma 5.3

Betrachte durch Anwendung von Union und Find entstandene wurzelgerichtete Bäume. Bezeichne **Tiefe** h eines Baumes die Länge eines längsten Weges von einem Blatt zu einer Wurzel. Ein Baum mit Tiefe h enthält mindestens 2^h Elemente.

Beweis. per vollständiger Induktion

Induktionsanfang $h = 0$

klar nur Wurzel, #Elemente = $1 = 2^0 = 2^h$ ✓

Induktionsschluss

Sei T Baum mit Tiefe $h > 0$ mit **Elementanzahl** $|T|$ **minimal**

Sei T entstanden durch $\text{Union}(T_1, T_2)$ mit $|T_1| \leq |T_2|$

Beobachtung $\text{Tiefe}(T_1) < \text{Tiefe}(T)$

also $\text{Tiefe}(T_1) \leq h - 1$ (\rightsquigarrow Induktionsvoraussetzung)

Beweis von Lemma 5.3

Wir haben T Baum mit Tiefe $h > 0$, $|T|$ minimal
 $T_1 = \text{Union}(T_1, T_2)$ mit $|T_1| \leq |T_2|$
 Tiefe(T_1) $\leq h - 1$

Annahme Tiefe(T_1) $< h - 1$

dann Tiefe(T) = Tiefe(T_2)

klar $|T| > |T_2|$ **Widerspruch**

also Tiefe(T_1) = $h - 1$

aus Induktionsvoraussetzung $|T_1| \geq 2^{h-1}$

Erinnerung $|T_2| \geq |T_1|$

also $|T| = |T_1| + |T_2| \geq 2^{h-1} + 2^{h-1} = 2^h$



Über Pfadkompression

Was ändert Find?

- Vorgänger von Knoten
- Tiefe von Bäumen

Was ändert Find nicht?

- Anzahl der Elemente im Baum
- Elemente im Baum
- Wurzel des Baumes

Folgerung Lemma 5.3 gilt **mit** und **ohne** Pfadkompression

Zentraler Analyse-Trick

Vergleiche Entwicklung der DS
 ohne Pfadkompression und mit Pfadkompression.

Betrachte Befehlsfolge der Länge r mit $\leq n - 1$ Union.

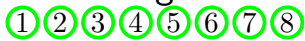
Notation

D_0^{mit} initiale DS mit Pfadkompression
 D_0^{ohne} initiale DS ohne Pfadkompression
 D_i^{mit} DS nach i Operationen mit Pfadkompression
 D_i^{ohne} DS nach i Operationen ohne Pfadkompression

Wir analysieren Union-Find mit Pfadkompression.

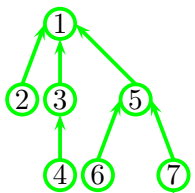
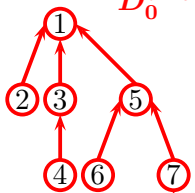
Wir betrachten dazu Union-Find ohne Pfadkompression.

Mit und ohne Pfadkompression im Vergleich



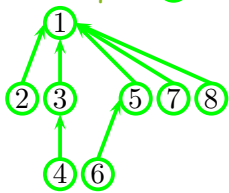
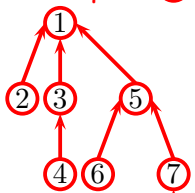
D_0^{ohne}

D_0^{mit}



D_7^{ohne}

D_7^{mit}



D_8^{ohne}

D_8^{mit}



Vergleich von D_i^{ohne} und D_i^{mit}

Definition 5.4

Wir betrachten eine Folge von bis zu $n - 1$ Union- und m Find-Befehlen. Dabei entstehen nacheinander die Datenstrukturen $D_0^{\text{mit}}, D_1^{\text{mit}}, \dots, D_r^{\text{mit}}$. Es bezeichne $D_0^{\text{ohne}}, D_1^{\text{ohne}}, \dots, D_r^{\text{ohne}}$ die Folge von Datenstrukturen, die bei Anwendung der gleichen Befehlsfolge bei Verzicht auf Pfadkompression entsteht.

Lemma 5.5

Betrachte die Folgen D_i^{mit} und D_i^{ohne} gemäß Definition 5.4. Für alle i gibt es zwischen D_i^{mit} und D_i^{ohne} eine bijektive Abbildung, die Bäume auf Bäume so abbildet, dass die enthaltenen Elemente und Wurzeln gleich sind.

Beweis von Lemma 5.5

per Induktion über die Anzahl Operationen i

Induktionsanfang $i = 0$ trivial ✓

Induktionsschluss

Voraussetzung D_i^{mit} und D_i^{ohne} so „gleich“

1. Fall $(i + 1)$ -te Operation ist Find

klar weder Wurzeln noch Elemente ändern sich ✓

2. Fall $(i + 1)$ -te Operation ist Union

gemäß Voraussetzung in D_i^{mit} und D_i^{ohne} Vereinigung zweier Mengen mit jeweils gleicher Wurzel und Größe

also gleiche Entscheidung, „Gleichheit“ erhalten



Ein genauerer Blick auf „Tiefe“

Definition 5.6

Betrachte die Folgen D_i^{mit} und D_i^{ohne} gemäß Definition 5.4. Für $v \in \{1, 2, \dots, n\}$ ist der *Rang* $\text{Rang}(v) := \max\{\text{Weglänge von } u \text{ nach } v \text{ in } D_i^{\text{ohne}} \mid \text{Find}(u) = \text{Find}(v) \text{ und } u \text{ Blatt}\}$.

Für $k \in \mathbb{N}$ ist die *Ranggruppe* R_k definiert durch

$R_k := \{i \in \mathbb{N}_0 \mid Z(k-1) < i \leq Z(k)\}$, außerdem ist $R_0 := \{0, 1\}$.

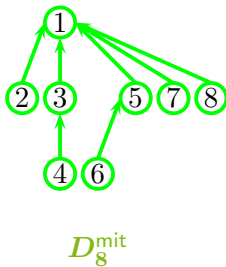
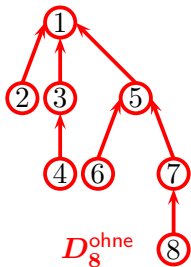
wichtig Rang immer in D_i^{ohne} definiert

Beobachtung in D_i^{ohne} wächst Rang streng monoton auf dem Weg zur Wurzel

Behauptung in D_i^{mit} auch

Beispiel für Ränge

Beispiel mit $n = 8$ und $r = 8$



Knoten	Rang
1	3
2	0
3	1
4	0
5	2
6	0
7	1
8	0

Rangwachstum

Lemma 5.7

Betrachte die Folgen D_i^{mit} und D_i^{ohne} gemäß Definition 5.4. Für alle $v \in \{1, 2, \dots, n\}$ und alle w , so dass v Elter von w in D_i^{mit} ist, gilt $\text{Rang}(v) > \text{Rang}(w)$.

Beweis. per vollständiger Induktion über Anzahl Operationen i

Induktionsanfang $i = 0$ trivial, weil keine Eltern ✓

Induktionsschluss

Voraussetzung in D_i^{mit} erfüllt

1. Fall $(i + 1)$ -te Operation ist Find

klar neue Eltern Wurzel, also Rang maximal ✓

Beweis von Lemma 5.7

Induktionsschluss

Voraussetzung in D_i^{mit} erfüllt

2. Fall $(i + 1)$ -te Operation ist Union

Beobachtung nur Kante zwischen
alter Wurzel und neuer Wurzel **neu**

Erinnerung Wurzeln in D_i^{mit} und D_i^{ohne} gleich (Lemma 5.5)

also erfüllt, weil in D_i^{ohne} erfüllt



Ränge und Anzahlen

Lemma 5.8

$$\forall k \in \mathbb{N}_0: |\{v \in \{1, 2, \dots, n\} \mid \text{Rang}(v) = k\}| \leq \frac{n}{2^k}$$

Beweis.

Betrachte D_i^{ohne}

klar $\text{Rang}(\text{Wurzel}) = \text{Tiefe des Baumes}$

also für T mit Wurzel v $|T| \geq 2^{\text{Rang}(v)}$ (Lemma 5.3)

klar w mit Rang k hat
 weder Vorgänger noch Nachfolger mit Rang k (Lemma 5.7)

also m Elemente mit Rang k
 $\Rightarrow \geq m \cdot 2^k$ Knoten in entsprechenden Teilbäumen

also Anzahl Elemente mit Rang $k \leq n/2^k$



Beweis von Theorem 5.2

Theorem 5.2

Mit der Datenstruktur mit wurzelgerichteten Bäumen und Pfadkompression kann eine Folge von bis zu $n - 1$ Union-Befehlen und m Find-Befehlen in Zeit $O((n + m) \log^* n)$ ausgeführt werden.

Beweis. mit amortisierter Analyse mit **Kostenverteilung**

Erinnerung jeder Union-Befehl in Zeit $O(1)$

also alle $\leq n - 1$ Union-Befehle in Zeit $O(n)$ ✓

noch offen Kosten der m Find-Befehle

Definiere **Kostenträger**

- Find-Befehle selbst
- Elemente

Vereinbarung „reale“ Kosten eines Find-Befehls,
 der genau k **Kanten** berührt
 $k + 1$ (also auch > 0 für Find(Wurzel))

Kostenverteilung für einen Find-Befehl

Betrachte $\text{Find}(v_k)$
berührt Kanten $v_k \rightarrow v_{k-1} \rightarrow v_{k-2} \rightarrow \dots \rightarrow v_1 \rightarrow v_0$
(tatsächliche) Rechenzeit $k + 1$

Kostenverteilung

immer Kosten 1 **auf Find-Befehl**

falls $k > 0$ Kosten 1 (für $v_1 \rightarrow v_0$) **auf Find-Befehl**

falls $k > 1$ Kosten 1 (für $v_i \rightarrow v_{i-1}$)

1. Fall Ranggruppe von v_i und v_{i-1} **gleich**
auf v_i

2. Fall Ranggruppe von v_i und v_{i-1} **verschieden**
auf Find-Befehl

Erinnerung Definition **Ranggruppe** (Definition 5.6)

$$R_k := \{i \in \mathbb{N}_0 \mid Z(k-1) < i \leq Z(k)\} \text{ für } k \in \mathbb{N}$$

$$R_0 := \{0, 1\}$$

Kostensummutation über Kostenträger: Find-Befehle

obere Schranke für Gesamtkosten je Find-Befehl

Teilkosten 1 immer

Teilkosten 1 falls nicht Wurzel gesucht

Teilkosten r für r Ranggruppenwechsel

Wieviele Ranggruppen gibt es?

Beobachtung $\leq n$ Ränge

also $\leq 1 + \log^* n$ Ranggruppen

also Teilkosten $\leq 1 + \log^* n$ für Ranggruppenwechsel

Gesamtkosten $\leq 1 + 1 + 1 + \log^* n = O(\log^* n)$ je Find-Befehl

also Gesamtkosten $O(m \log^* n)$ für m Find-Befehle

Kostensummation über Kostenträger: Elemente v_i

Betrachte Kante $v_i \rightarrow v_{i-1}$ mit Kosten **auf** v_i

klar v_0 wird Elter von v_i

Beobachtung v_0 war vorher **nicht** Elter von v_i

sonst Sonderfall $v_1 \rightarrow v_0 \rightsquigarrow$ Kosten **auf** Find

also v_i bekommt Elter mit größerem Rang als vorher

Erinnerung Kosten für $v_i \Leftrightarrow \exists g: \{\text{Rang}(v_i), \text{Rang}(v_{i-1})\} \subseteq R(g)$

also #Rangwachstum der Eltern $\leq Z(g) - Z(g-1) \leq Z(g)$

also jedes Element in Ranggruppe g trägt Kosten $\leq Z(g)$

Erinnerung #Elemente mit Rang $r \leq n/2^r$ (Lemma 5.8)

also Anzahl Elemente in Ranggruppe g

$$\begin{aligned} &\leq \sum_{r=Z(g-1)+1}^{Z(g)} \frac{n}{2^r} < n \cdot \sum_{r=Z(g-1)+1}^{\infty} 2^{-r} = \frac{n}{2^{Z(g-1)+1}} \cdot \sum_{r=0}^{\infty} 2^{-r} \\ &= \frac{n}{2^{Z(g-1)}} = \frac{n}{Z(g)} \end{aligned}$$

Kostesummation für alle Elemente

Wir haben Kosten für v_i nur bei Rangwachstum des Elter
 $\# \text{Rangwachstum} < n/Z(g)$

Summation über Ranggruppen

Beobachtung in Ranggruppe g'
 Gesamtkosten $\leq Z(g') \cdot \frac{n}{Z(g')} = n$

Erinnerung $\leq 1 + \log^* n$ Ranggruppen

also Gesamtkosten summiert über alle Elemente
 $\leq (1 + \log^* n) \cdot n = O(n \log^* n) \checkmark$

Zusammenfassung Kostensummutation

	Gesamtkosten $\leq n - 1$ Union	$O(n)$
+	Gesamtkosten m Finds verteilt auf Find-Befehle	$O(m \log^* n)$
+	Gesamtkosten m Finds verteile auf Elemente	$O(n \log^* n)$
<hr/>		
	Summe	$O((n + m) \log^* n)$



Asymptotisch exakt? Leider nein.

In Wahrheit sogar linear? Leider nein.

für uns jetzt genau genug