# Kapitel 5
# Amortisierte Analyse

Effiziente Algorithmen, SS 2018

Professor Dr. Petra Mutzel

VO 11/12/13 am 22./24./29. Mai 2018

# Amortisierte Analyse

**Betrachte** Datenstruktur $D$ mit Operationen

**Übliche Worst-Case-Analyse**

**klar** bei $n$ Daten in $D$
Worst-Case-Rechenzeit $t(n)$ je Operation
**dann** 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

**Beispiel Binärzähler**

**Aufgabe** zu $z \in \mathbb{N}_0$ speichere Binärdarstellung $(z_{l-1}z_{l-2}\dots z_1 z_0)_2$
mit $z = \sum_{i=0}^{l-1} 2^i z_i$

**Operationen** SetzeNull() und PlusEins() (Beginn bei $z = 0$)

**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 $l$ $(11111 \leadsto 100000)$

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

# 5.1 Amortisierte Analyse mit der Potenzialmethode

**Betrachte** Folge von Datenstrukturen $D_i$ und Operationen $\mathsf{Op}_i$

$$D_0 \overset{\mathsf{Op}_1}{\leadsto} D_1 \overset{\mathsf{Op}_2}{\leadsto} D_2 \overset{\mathsf{Op}_3}{\leadsto} D_3 \overset{\mathsf{Op}_4}{\leadsto} \cdots \overset{\mathsf{Op}_m}{\leadsto} D_m$$

$T_i$ tatsächliche Rechenzeit von $\mathsf{Op}_i$

**Definiere** Potenzialfunktion $\Phi\colon \mathcal{D} \to \mathbb{R}_0^+$

Amortisierte Rechenzeit von $\mathsf{Op}_i$:

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

**Forderung** für die Potenzialfunktion: $\Phi(D_i) \geq \Phi(D_0)$ für alle $i$

**Rechtfertigung**

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} (T_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{m} T_i + \sum_{i=1}^{m} (\Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{m} T_i + (\Phi(D_m) - \Phi(D_0)) \geq \sum_{i=1}^{m} T_i$$

# Beispiel Binärzähler mit Potenzialmethode

**Betrachte** Operation PlusEins(), Beginn bei $z = 0$

**Definition** $\Phi(z) := \sum_{i=0}^{l-1} z_i$, also $\Phi(D_0) = 0$

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

**1. Fall** vorher $z_0 = 0$
**Beobachtungen** $T_i = 1 \quad \Phi(D_i) = \Phi(D_{i-1}) + 1$
**also** $a_i = T_i + \Phi(D_i) - \Phi(D_{i-1}) = 2$

**2. Fall** vorher $z_0 = z_1 = \cdots = z_{k-1} = 1$ ($0 < k < l$, $k$ max.)
**Beobachtung** $T_i = k + 1 \quad \Phi(D_i) = \Phi(D_{i-1}) - k + 1$
**also** $a_i = T_i + \Phi(D_i) - \Phi(D_{i-1}) = k + 1 - k + 1 = 2$

**also** Gesamtrechenzeit bei $m$ Operationen $O(m)$

**Beobachtung** asymptotisch exakt

# 5.2 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 $\left(2^i\right)$-ten Operation
also bei $m$ Operationen
Kosten $\lceil m/2^i \rceil$ bei Kostenträger $z_i$

**also Gesamtkosten**

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

**Beobachtung** auch asymptotisch exakt

# 5.3 Amortisierte Analyse mit der Kontenmethode

- Verschiedenen Operationen werden verschiedene Kosten zugeordnet, die evtl. auch größer oder kleiner als die tatsächlichen Kosten sein können.
- Wenn die amortisierten Kosten die tatsächlichen Kosten übersteigen, dann wird die Differenz speziellen Objekten in der Datenstruktur als Kredit (Guthaben) zugewiesen, der später verwendet werden kann.
- Dieser dient für Operationen, deren tatsächliche Kosten höher sind als deren amortisierte Kosten. Der Gesamtkredit darf niemals negativ sein.
- Es muss dabei zu jedem Zeitpunkt und für jede Sequenz gelten: die Summe der tatsächlichen Kosten ist kleiner gleich der Summe der amortisierten Kosten.

# Amortisierte Analyse mit der Kontenmethode

Definiere Konten
zum Beispiel Operationen, Teile der Datenstruktur, ...

nach Belieben

- bei Operation $\mathrm{Op}_i$ 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 $\mathrm{Op}_i$ 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 $\widehat{=}$ Stellen $z_i$

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

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

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

Beobachtung korrekt, weil immer erst „$0 \rightsquigarrow 1$“, dann „$1 \rightsquigarrow 0$“

Beobachtung jetzt amort. 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

# 5.4 Union-Find mit Pfadkompression

„Nachlese“ aus DAP 2

Zunächst: Wiederholung aus DAP2 (s.ppt-Folien)

**Bemerkung:** Damit der Beweis identisch zu demjenigen im Skript ist, bauen wir ihn auf der Realisierung des ADT UNION-FIND aus der DAP2-Vorlesung von Ingo Wegener (SS 07) auf. Dort gibt es zwei Unterschiede zu DAP2 vom SS 09:

- Die Basisversion entspricht der aus SS 09 mit der gewichteten Vereinigungsregel nach Größe. UNION vereinigt also zwei Bäume nach der Größe (Anzahl der Knoten) statt der Höhe der Bäume.
- Zur Abschätzung wurde eine etwas andere Funktion verwendet ($Z(k)$ statt $A_k(1)$) Damit ist auch die Inverse eine leicht andere; aber die Wachstumseigenschaften sind ähnlich.

# Kurze Wiederholung aus DAP 2 (vom SS 07)

**Datenstrukturen für Partitionen: ADT UNION-FIND**

$n$ Objekte $1, \ldots, n$ initial in $n$ Mengen $\{1\}, \ldots, \{n\}$

Operationen

- FIND($x$) Bestimme Repräsentanten der Menge, die $x$ enthält.
- UNION($A, B$) Vereinige Mengen $A$ und $B$.

Baumorientierte Datenstruktur für schnelles UNION

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

Pfadkompression

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

# Ein Wort zur Implementierung (mit Pfadkompression)

**Datenstrukturen** Array $\pi[1..n]$ für Elternverweis in Bäumen
Array $G[1..n]$ für Größe

**initial** $\forall i\colon \pi[i] = i$ {* *zeigt Wurzel* *}
$G[i] = 1$

**Union**$(i, j)$ If $G[i] \geq G[j]$
Then $\pi[j] := i$; $G[i] := G[i] + G[j]$
Else $\pi[i] := j$; $G[j] := G[i] + G[j]$

**Find**$(i)$ $k := i$; While $\pi[k] \neq k$ { $k := \pi[k]$ }
$w := k$; $k := i$;
While $\pi[k] \neq k$ { $k' := \pi[k]$; $\pi[k] := w$; $k := k'$ }
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 durch
Pfadkompression langsamer, aber nicht asymptotisch

**Hoffnung** zukünftige Find-Befehle schneller

# Das Theorem aus DAP2 (SS 07)

## Definition 5.1

$Z(0) := 1,\ Z(i) := 2^{Z(i-1)}$ für $i \in \mathbb{N}$
$\log^* n := \min\{k \mid Z(k) \geq n\}$

Also: $Z(0) = 1, Z(1) = 2, Z(2) = 4, Z(3) = 2^4 = 16, Z(4) = 2^{16} = 65536, Z(5) = 2^{65536}$

## 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 für Beweis

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 Union$(T_1, T_2)$ mit $|T_1| \leq |T_2|$

Beobachtung Tiefe$(T_1)$ $<$ Tiefe$(T)$ (weil $T_1$ an Wurzel hängt)
also Tiefe$(T_1) \leq h - 1$ ($\rightsquigarrow$ Induktionsvoraussetzung)

# Beweis von Lemma 5.3

Wir haben $T$ Baum mit Tiefe $h > 0$, $|T|$ minimal
$T =$ 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 ($T$ war minimal)

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 (DatenStruktur)
ohne Pfadkompression und mit Pfadkompression.

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

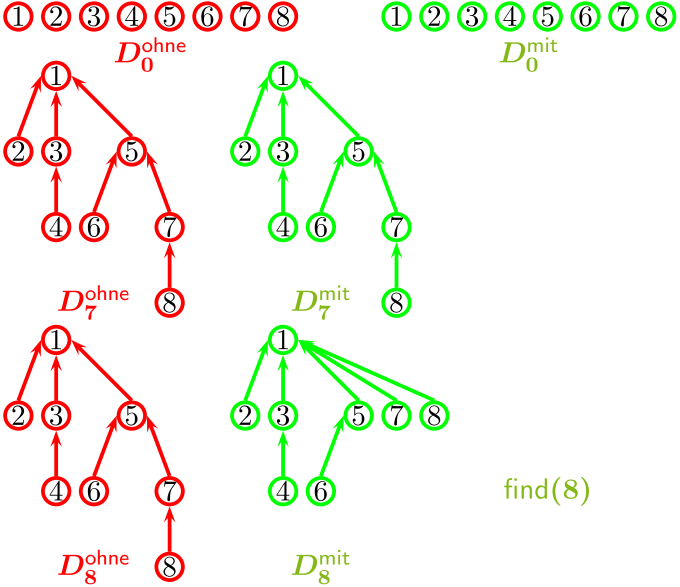
Notation

| | |
|---|---|
| $D_0^{\text{mit}}$ | initiale DS mit Pfadkompression |
| $D_0^{\text{ohne}}$ | initiale DS ohne Pfadkompression |
| $D_i^{\text{mit}}$ | DS nach $i$ Operationen mit Pfadkompression |
| $D_i^{\text{ohne}}$ | 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




find(8)

# Vergleich von $D_i^{\text{ohne}}$ und $D_i^{\text{mit}}$

## Definition 5.4

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

## Lemma 5.5

Betrachte die Folgen $D_i^{\text{mit}}$ und $D_i^{\text{ohne}}$ gemäß Definition 5.4. Für alle $i$ gibt es zwischen $D_i^{\text{mit}}$ und $D_i^{\text{ohne}}$ 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 $\checkmark$

Induktionsschluss
Voraussetzung $D_i^{\text{mit}}$ und $D_i^{\text{ohne}}$ so „gleich“

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

klar weder Wurzeln noch Elemente ändern sich $\checkmark$

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

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

also gleiche Entscheidung, „Gleichheit“ erhalten

$\square$

# Ein genauerer Blick auf „Tiefe“

## Definition 5.6

Betrachte die Folgen $D_i^{\text{mit}}$ und $D_i^{\text{ohne}}$ gemäß Definition 5.4.
Für $v \in \{1, 2, \ldots, n\}$ 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 $R_0 := \{0, 1\}$.

$R_0 = \{0, 1\}, R_1 = \{2\}, R_2 = \{3, 4\}, R_3 = \{5, 6, \ldots, 16\}, R_4 = \{17, \ldots, 65536\}, \ldots$

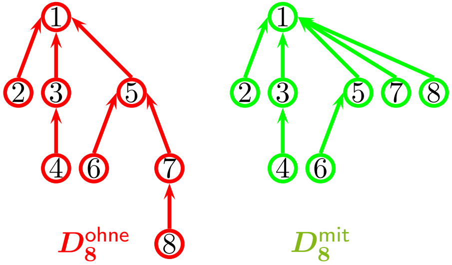
wichtig   Rang immer in $D_i^{\text{ohne}}$ definiert

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

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

Beweis. Per vollständiger Induktion über Anzahl Operationen $i$

Induktionsanfang $i = 0$ trivial, weil keine Eltern $\checkmark$

Induktionsschluss

Voraussetzung in $D_i^{\text{mit}}$ erfüllt

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

klar neuer Elternknoten ist nur Wurzel,

und für Wurzel ist Rang maximal $\checkmark$

# Beweis von Lemma 5.7

Induktionsschluss

Voraussetzung in $D_i^{\text{mit}}$ erfüllt

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

Beobachtung nur Kante zwischen
alter Wurzel und neuer Wurzel neu

Erinnerung Wurzeln in $D_{i+1}^{\text{mit}}$ und $D_{i+1}^{\text{ohne}}$ gleich (Lemma 5.5)

also erfüllt, weil in $D_{i+1}^{\text{ohne}}$ erfüllt

(Rang in $D_{i+1}^{\text{ohne}}$ definiert)

□

# Ränge und Anzahlen

## Lemma 5.8

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

Beweis.

Betrachte $D_i^{\text{ohne}}$

klar Rang(Wurzel) = 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 $k$
$\Rightarrow$ mind. $m \cdot 2^k$ 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

Zur Analyse betrachten wir diejenige Rang-Funktion, die am Ende aller Operationen (insbesondere nach dem letzten Union-Befehl) entsteht.

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

# 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.* Kosten der $m$ Find-Befehle

Definiere Kostenträger

- Find-Befehle selbst
- Elemente

Vereinbarung: tatsächliche Kosten eines Find-Befehls, der genau $k$ Kanten berührt, ist $k+1$ (also auch $> 0$ für Find(Wurzel))

# Kostenverteilung für einen Find-Befehl

**Betrachte** Find($v_k$)
berührt Kanten $v_k \to v_{k-1} \to v_{k-2} \to \cdots \to v_1 \to v_0$
(tatsächliche) Rechenzeit $k+1$

**Kostenverteilung**

**immer** Kosten 1 auf Find-Befehl

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

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

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

# Kostensummation 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 $v_i \to v_{i-1}$ mit Kosten auf $v_i$

klar $v_0$ wird Elternknoten von $v_i$

Beobachtung $v_0$ war vorher nicht Elternknoten von $v_i$
sonst Sonderfall $v_1 \to v_0 \rightsquigarrow$ Kosten auf Find

also $v_i$ bekommt Elternknoten mit größerem Rang als vorher

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

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 $g$

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

# Kostensummation für alle Elemente

Wir haben Kosten für $v_i$ nur bei Rangwachstum des Elternknoten
#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
$\leq (1 + \log^* n) \cdot n = O(n \log^* n)$ ✓

# Zusammenfassung Kostensummation

| | | |
|---|---|---|
| | Gesamtkosten $\leq n-1$ Union | $O(n)$ |
| $+$ | Gesamtkosten $m$ Finds<br>verteilt auf Find-Befehle | $O(m \log^* n)$ |
| $+$ | Gesamtkosten $m$ Finds<br>verteilt auf Elemente | $O(n \log^* n)$ |
| | Summe | $O\left((n+m)\log^* n\right)$ |

□

Asymptotisch exakt? Man kann $O\left(n + m \log^* n\right)$ zeigen

für uns jetzt genau genug
(weitergehender Beweis nicht interessant für uns)