Julius-Maximilians-
UNIVERSITÄT WÜRZBURG

Lehrstuhl für
INFORMATIK I
Effiziente Algorithmen und wissensbasierte Systeme

Institut für Informatik

Algorithmen und Datenstrukturen

Wintersemester 2018/19
16. Vorlesung

Amortisierte Analyse

Special Guest: Fabian Feitsch

Prof. Dr. Alexander Wolff *Lehrstuhl für Informatik I*

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Verhindere, dass die Tabelle überläuft
oder dass Operationen ineffizient werden.

Problem: Was tun, wenn man die maximale Anzahl zu speichernder Elemente vorab nicht kennt?

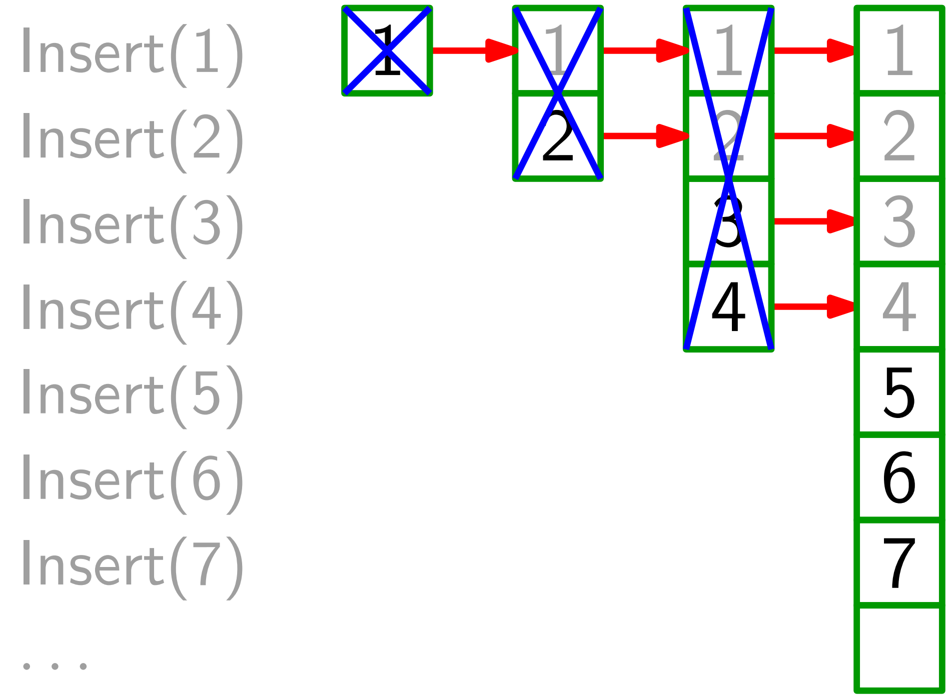
Lösung: *Dynamische* Tabellen!

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:

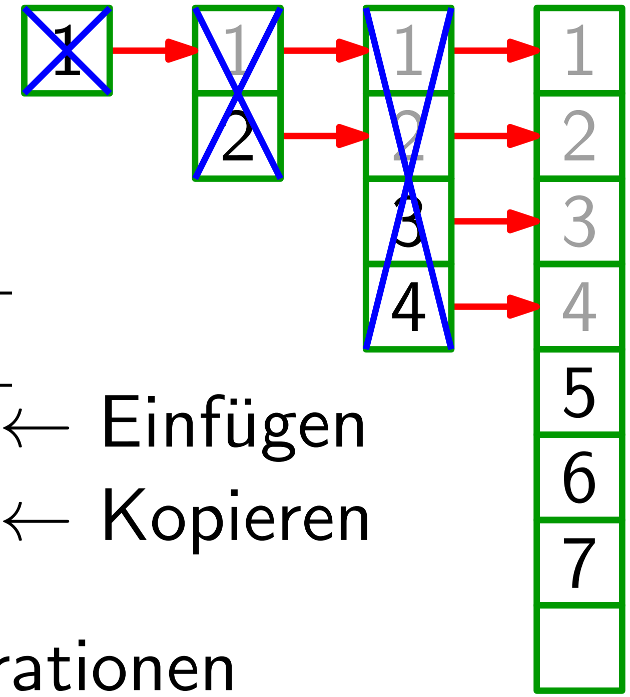
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

Also ist der Gesamtaufwand $\underset{O}{\cancel{\Theta}}(n \log n)$, genauer $\Theta(n)$.

Let's see why...

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = \boxed{n} + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \leq n + \frac{2^{\log_2(n-1)+1} - 1}{2-1}$$

$$= n + 2(n-1) - 1$$

$$< 3n \in \Theta(n)$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1} - 1}{q - 1} \quad \left[\text{endl. geom. Reihe}\right]$$

D.h. die durchschnittlichen („amortisierten“) Kosten sind $\Theta(1)$.

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben –
auch wenn einzelne Operationen in der Folge teuer sind!

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle Ereignisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Bei amortisierter Analyse geht es jedoch um die durchschnittliche Laufzeit *im schlechtesten Fall* –
nicht im Erwartungswert!

Wir betrachten 3 verschiedene Typen von amortisierter Analyse:

– Aggregationsmethode ✓
– Buchhaltermethode
– Potentialmethode

Buchhaltermethode

- Tendentiell genauer als die Aggregationsmethode
- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow\;$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow\;$ Wir bezahlen teure Operationen mit vorher Beiseitegelegtem.

- Damit's klappt: wir dürfen nie in die Miesen kommen –

  Guthaben $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$ darf nicht negativ werden!

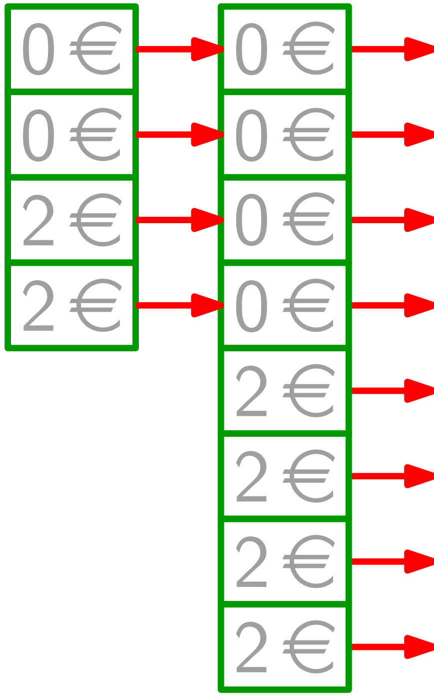
  Dann gilt $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.

D.h. amortisierte Kosten sind obere Schranke für tatsächliche Kosten!

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.

Damit wird deutlich, dass die DS nie Miese macht.

$\Downarrow$

Also sind amortisierte Kosten obere Schranke für tatsächliche Kosten!

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = 3n = \Theta(n)$$

D.h. die (tats.) Kosten für n Einfügeoperationen betragen $\Theta(n)$.

Buchhaltermethode: noch'n Beispiel

Stapel

verwaltet sich ändernde Menge nach *LIFO-Prinzip*

Abs. Datentyp	**Implementierung**
boolean Empty() Push(key k) key Pop() key Top()	

Multipop(int k)
neu!
	while not Empty() **and** $k > 0$ **do**
	    Pop()
	    $k = k - 1$

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation_i	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, \mathit{size}_i\}$	0

Geht das gut??? – *Ja!* D.h. Folge von n Op. dauert $\Theta(n)$ Zeit.

Zeige: Amortisierte Kosten „bezahlen" immer für die echten!

- Jede Push-Operation legt einen Teller auf den Stapel. Dafür bezahlt sie 1 € und legt noch 1 € auf den Teller.
- Jede (Multi-)Pop-Operation wird mit den Euros auf den Tellern, die sie wegnimmt, komplett bezahlt.

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi \colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

Folge: $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$ *teleskopierende Summe*

$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \geq \sum_{i=1}^{n} c_i$

D.h. amortisierte Kosten „bezahlen“ für echte Kosten.

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, \mathit{size}_i\}$$
$$c_i = +\min\{k_i, \mathit{size}_i\}$$

$\hat{c}_i = c_i + \Delta\Phi D_i = 0$, dito mit Pop $(k_i = 1)$.

Also: Amortisierte Kosten pro Operation $O(1)$.
$\Rightarrow$ Echte Kosten für n Oper. im worst case $O(n)$.

Zusammenfassung

Zeige mit amortisierter Analyse, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Drei Typen von amortisierter Analyse:

- Aggregationsmethode ✓

  Summiere tatsächliche Kosten (oder obere Schranken dafür) auf.

- Buchhaltermethode ✓

  Verbinde Extrakosten mit konkreten Objekten der DS und bezahle damit teure Operationen.

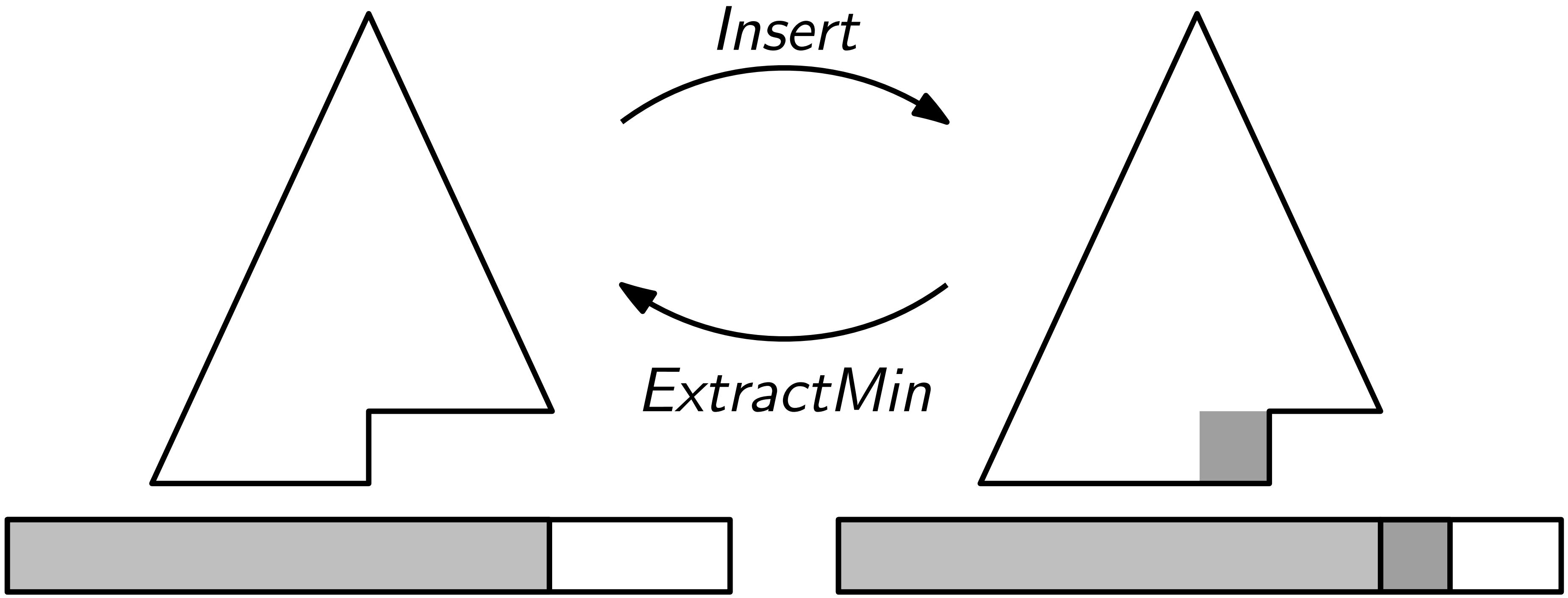
- Potentialmethode ✓

  Definiere Potential der gesamten DS, so dass mit der Potentialdifferenz teure Operationen bezahlt werden können.

Übungsaufgaben zur amortisierten Analyse (I)

Gegeben sei ein gewöhnlicher MinHeap, dessen Methoden *Insert* und *ExtractMin* im schlechtesten Fall $O(\log n)$ Zeit brauchen.

Zeigen Sie mit der Potentialmethode, dass *Insert* amortisiert $O(\log n)$ Zeit und *ExtractMin* amortisiert $O(1)$ Zeit benötigt.



Übungsaufgaben zur amortisierten Analyse (II)

Entwerfen Sie eine Datenstruktur zum Verwalten einer dynamischen Menge von Zahlen. Die DS soll 2 Methoden haben:

- *Insert* zum Einfügen einer Zahl und
- *DeleteLargerHalf* zum Löschen aller Zahlen aus der Datenstruktur, die größer oder gleich dem aktuellen Median der Zahlenmenge sind.

Beide Methoden sollen amortisiert $O(1)$ Zeit benötigen.
Tipp: Verwenden Sie eine Liste!

1. Beschreiben Sie Ihren Entwurf der Datenstruktur einschließlich der beiden Methoden in Worten.

2. Analysieren Sie mithilfe der Buchhaltermethode.
   Geben Sie die amortisierten Kosten, die Sie mit *Insert* und *DeleteLargerHalf* verbinden, exakt an.