

# Datenstrukturen und effiziente Algorithmen

Markus Vieth, David Klopp, Christian Stricker

1. Dezember 2015



# 1 Vorlesung 12

## 1.1 (a,b)-Suchbäume

Blattorientierte Speicherung der Elemente

Innere Knoten haben mindestens a und höchstens b Kinder und tragen entsprechende Schlüsselwerte, um die Suche zu leiten.

**Beispiel:**

$$h \hat{=} \text{Tiefe} \Rightarrow a^h \leq n \leq b^h \Rightarrow \log_b n \leq h \leq \log_a n$$

### 1.1.1 Aufspaltung bei Einfügen

### 1.1.2 Verschmelzen von Knoten beim Löschen

Aufspalte- und Verschmelze-Operationen können sich von der Blattebene bis zur Wurzel kaskadenartig fortpflanzen. Sie bleiben aber auf den Suchpfad begrenzt.

$\Rightarrow$  Umbaukosten sind beschränkt durch die Baumtiefe  $= O(\log n)$

## 1.2 Amortisierte Analyse

	000	
	001	Kosten(1) = 1
	010	= 2
	011	= 1
<b>Beispiel: Binärzähler</b>	100	= 3
	101	= 1
	110	= 2
	111	= 1
		$\overline{11}$

Naive Analyse  $2^k = n$

$$1 \cdot \frac{n}{2} + 2 \cdot \frac{n}{4} + 3 \cdot \frac{n}{8} + \dots + k \cdot \frac{n}{2^k} = \frac{n}{2} \sum_{i=1}^k i \left(\frac{1}{2}\right)^{i-1} = 2^{k+1} - k - 2 = 2n - k - 2$$

Von 0 bis  $n$  im Binärsystem zu zählen kostet  $\leq 2n$  Bit-Flips

**Sprechweise:** amortisierte Kosten einer Inkrement-Operation sind 2

Folge von  $n$ -Ops kostet  $2n$

### 1.2.1 Bankkonto-Methode

$$\text{Konto}(i+1) = \text{Konto}(i) - \text{Kosten}(i) + \text{Einzahlung}(i)$$

$$\sum_{i=1}^n \text{Kosten}(i) = \text{tatsächliche Gesamtkosten} = \sum_{i=1}^n (\text{Einzahlung}(i) + \text{Konto}(i) - \text{Konto}(i+1))$$

$$= \sum_{i=1}^n \text{Einzahlung}(i) + \text{Konto}(1) - \text{Konto}(n+1)$$

000	
001€	Kosten(1) = 1
01€0	= 2
01€1€	= 1
1€00	= 3
1€01€	= 1
1€1€0	= 2
1€1€1€	= 1
	$\overline{11}$

### Kontoführungsschema: für Binärzähler

1€ pro 1 in der Binärdarstellung

Jeder Übergang  $1€ \rightarrow 0$  kann dann mit dem entsprechenden Euro Betrag auf dieser 1 bezahlt werden.

Es gibt pro Inkrement Operation nur einen  $0 \rightarrow 1$  Übergang

2€ Einzahlung für jede Inc-Operation reichen aus um:

1. diesen  $0 \rightarrow 1$  Übergang zu bezahlen
2. die neu entstandene 1€ mit einem Euro zu besparen.

$$\text{GK} = 2(2^k - 1) + 0^{\text{I}} - k^{\text{II}} = 2n - k - 2$$

---

<sup>I</sup>Zählerstand(000)

<sup>II</sup>Zählerstand( $\overbrace{111 \dots 1}^k$ )

## 2 Vorlesung 13

**Satz:** Ausgehend von einem leeren 2-5-Baum betrachten wir die Rebalancierungskosten  $C$  (Split- und Fusionsoperationen) für eine Folge von  $m$  Einfüge- oder Löschooperationen. Dann gilt:  $C \in O(m)$  d.h. Amortisierte Kosten der Split- und Fusionsoperationen sind konstant.  
! Dies bezieht sich nicht auf die Suchkosten, die in  $O(\log n)$  liegen.

**Beweisidee:**

**Kontoführung:**

1	2	3	4	5	6
2€	1€	0€	0€	1€	2€

regelmäßige Einzahlung: 1€

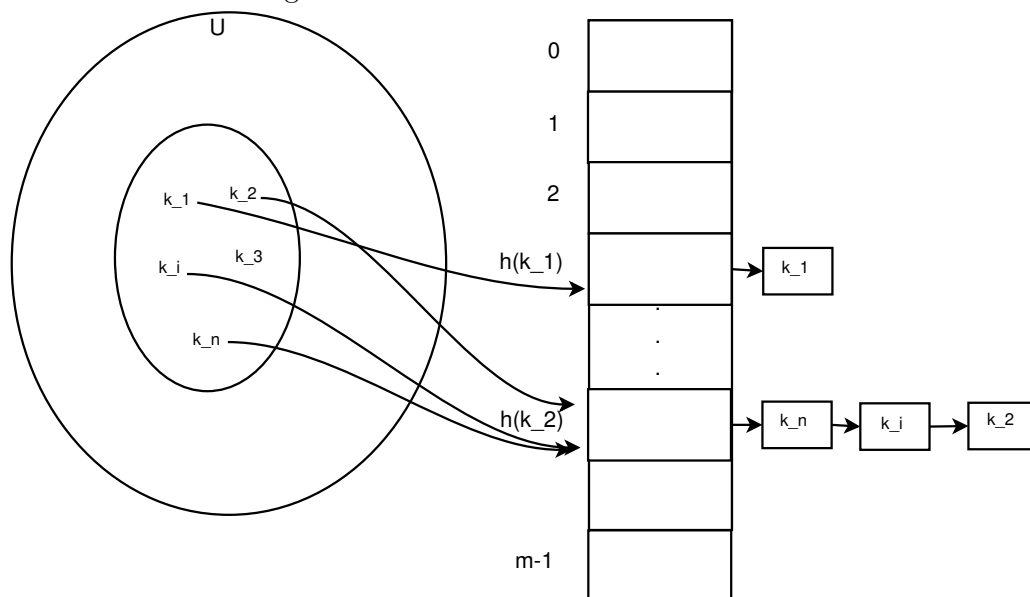
Durch eine Einfüge- oder Löschooperation steigt oder fällt der Knotengrad des direkt betroffenen Knotens um höchstens 1.  $\Rightarrow$  1€ Einzahlung reicht zur Aufrechterhaltung dieses Sparplanes.

Jetzt Beseitigung der temporären 1- und 6-Knoten:

Ein 6-Knoten nutzt 1€ um seinen Split zu bezahlen. Die beiden neu entstehenden 3-Knoten benötigen kein Kapital. Der Vaterknoten des gesplitteten 6-Knotens benötigt ggf. den zweiten verfügbaren €. Analoge Betrachtung für Fusion eines temp. 1-Knotens.

### 2.1 Hashing

Abbildung 2.1: Universum und Hashtabelle der Größe  $m$



$U \subseteq \mathbb{N}$  z.B. 64-Bit-Integer

$n$  = Zahl der zu verwaltenden Schlüssel

$$|U| \gg n$$

Hashfunktion  $h$ :

$$h : U \rightarrow [0, \dots, m-1]$$

$$\text{z.B. } k \mapsto k \bmod m$$

Einfache Annahme: (einfaches uniformes Hashing)

$$\forall k_i, k_j \in U : \Pr(h(k_i) = h(k_j)) = \frac{1}{m}$$

### Analyse der Laufzeit zum Einfügen eines neuen Elementes $k$

- $h(k)$  berechnen  $\rightarrow O(1)$
- Einfügen am Listenanfang in Fach  $h(k)$ .  $\rightarrow O(1)$

### Analyse der Suchzeit für einen Schlüssel $k$

- $h(k) \rightarrow O(1)$
- Listenlänge zum Fach  $h(k)$  sei  $n_{h(k)}$  also beim Durchlauf der kompletten Liste  $\rightarrow O(n_{h(k)})$

$$E(n_{h(k)}) = \frac{n}{m} = \alpha^I$$

$$\text{Suchzeit(Einfügen)} \in O(1 + \alpha)$$

### Laufzeit beim Löschen von Schlüssel $k$

- $h(k) \rightarrow O(1)$
- Durchlaufen der Liste  $\rightarrow O(n_{h(k)})$
- Löschen durch „Pointer-Umbiegen“  $\rightarrow O(1)$

## 2.1.1 Universelles Hashing

**Idee** Arbeite nicht mit einer festen Hashfunktion sondern wähle am Anfang eine zufällige Hashfunktion aus einer Klasse von Hashfunktionen aus.

**z.B.**

$$h_{a,b}(k) = ((a \cdot k + b) \bmod p) \bmod m$$

$p$  sei eine hinreichend große Primzahl  $0 < a < p, 0 \leq b < p$

$$\mathcal{H}_{p,m} = \{h_{a,b}(k) | 0 < a < p, 0 \leq b < p\}$$

$$|\mathcal{H}_{p,m}| = p(p-1)$$

**Definition**  $\mathcal{H}$  heißt universell  $\Leftrightarrow \forall k, l \in U : \Pr(h(k) = h(l)) \leq \frac{1}{m}$

---

<sup>I</sup>Belegungsfaktor

**Suchzeit**

$$\mathcal{X}_{k,l} = \begin{cases} 1 & \text{für } h(k) = h(l) \\ 0 & \text{sonst} \end{cases}$$

$$E(n_{h(k)}) = E\left(\sum_{l \in T, l \neq k}\right) = \sum_{l \in T, l \neq k} E(X_{k,l}) = \sum_{l \in T, l \neq k} Pr(h(k) = h(l)) = \sum_{l \in T, l \neq k} \frac{1}{m} = \frac{n-1}{m} = \alpha$$