

Vorlesung

# Effiziente Algorithmen und Komplexitätstheorie

Sommersemester 2008

Ingo Wegener

# Was bisher geschah...

## Thema Hashing

Verwalte **Wörterbuch** mit **Schlüsseln**.  
Operationen Search, Insert, Delete

## Begriffe und Variablen

- Universum alle Schlüssel  $\mathcal{U} = \{0, 1, \dots, |\mathcal{U}| - 1\}$   
(dazu Primzahl  $p \geq |\mathcal{U}|$ )
- Menge der zu speichernden Schlüssel  $S$  (mit  $|S| = n$ )
- Größe der Hashtabelle  $M$
- Hashfunktion  $h: \mathcal{U} \rightarrow \{0, 1, \dots, M - 1\}$
- Kollision ( $x \neq y \in \mathcal{U}$  mit  $h(x) = h(y)$ )
- Lastfaktor  $\alpha = n/M$
- ideales Hashing (oder uniformes Hashing):  $h(x)$  uniform zufällig verteilt

# Beobachtungen, Erkenntnisse, Begriffe

- bei idealem Hashing erwartete Zeit für erfolgreiche/erfolglose Suche  $\Theta(1 + \alpha)$
- im Worst Case Zeit  $\Theta(n)$
- $\rightsquigarrow$  Hashklassen
- $\mathcal{H}$  heißt  $c$ -universell, wenn  $\text{Prob}(\text{Kollision}) \leq c/M$  (Definition 15.1)
- $\mathcal{H}_l = \{h_{a,b} \mid 0 < a < p, 0 \leq b < p\}$  mit  $h_{a,b}(x) := ((a \cdot x + b) \bmod p) \bmod m$  ist 1-universell. (Definition 15.2, Theorem 15.3)
- erwartete Suchzeit bei  $h \in \mathcal{H}$  uniform zufällig und  $\mathcal{H}$   $c$ -universell ist  $\Theta(1 + \alpha)$  ( $c = O(1)$ , Theorem 15.4)

# Perfektes Hashing

perfektes Hashing Worst-Case Zugriffszeit  $O(1)$   
z.B. für **statisches** Wörterbuch (z. B. CD-ROM)

triviale Idee Wähle  $M$  **groß genug** für Kollisionsfreiheit.

Fazit Hashtabellengrößen  $M = o(n^2 / \log n)$  **chancenlos**

also bessere **Idee** benötigt

# Idee für statisches perfektes Hashing

**Beobachtung** Hashtabellengröße  $M' \rightsquigarrow k \cdot M'$   
 $\hat{=}$  Platz  $k$  je ursprünglichem Bucket

**Idee** Bucketgröße individuell und adaptiv bestimmen

**Erinnerung** Bucketgröße  $b_i^2$  ausreichend (Lemma 15.5)

Aber ist  $\sum_{i=0}^{M-1} b_i^2$  nicht auch schon quadratisch?

## Lemma 15.6

Sei  $M = n$ ,  $h \in \mathcal{H}_l$  uniform zufällig gewählt,

$b_i := |\{x \in S \mid h(x) = i\}|$ .

$$\mathbb{E} \left( \sum_{i=0}^{M-1} b_i^2 \right) < 2n$$

## Beweis von Lemma 15.6

klar  $\sum_{i=0}^{M-1} b_i = n$

klar  $a^2 = 2\binom{a}{2} + a$

also 
$$\begin{aligned} \mathbb{E} \left( \sum_{i=0}^{M-1} b_i^2 \right) &= \mathbb{E} \left( \sum_{i=0}^{M-1} b_i \right) + 2 \sum_{i=0}^{M-1} \mathbb{E} \left( \binom{b_i}{2} \right) \\ &= n + 2 \sum_{i=0}^{M-1} \mathbb{E} \left( \binom{b_i}{2} \right) \end{aligned}$$

Beobachtung  $\sum_{i=0}^{M-1} \binom{b_i}{2} = |\{x \neq y \in S \mid h(x) = h(y)\}|$

Definiere  $X_{x,y} := \begin{cases} 1 & \text{falls } h(x) = h(y) \\ 0 & \text{sonst} \end{cases}$

damit  $\sum_{i=0}^{M-1} \binom{b_i}{2} = \sum_{x \neq y \in S} X_{x,y}$

## Beweis von Lemma 15.6 (Fortsetzung)

Wir haben 
$$\mathbb{E} \left( \sum_{i=0}^{M-1} b_i^2 \right) = n + 2 \sum_{i=0}^{M-1} \mathbb{E} \left( \binom{b_i}{2} \right)$$
 mit 
$$\sum_{i=0}^{M-1} \binom{b_i}{2} = \sum_{x \neq y \in S} X_{x,y}$$

Erinnerung 
$$\mathbb{E}(X_{x,y}) = 1/M \text{ (Theorem 15.3)}$$

also 
$$\begin{aligned} \mathbb{E} \left( \sum_{i=0}^{M-1} b_i^2 \right) &= n + 2 \cdot \binom{n}{2} \cdot \frac{1}{M} \\ &= n + \frac{n(n-1)}{n} < 2n \end{aligned}$$



# Aufbau der Datenstruktur

## Algorithmus 15.7

1. Repeat
2.     Wähle  $h \in \mathcal{H}_l$  uniform zufällig.
3.     Für alle  $i \in \{0, 1, \dots, M - 1\}$   $b_i := 0$ ;  $B_i := \emptyset$
4.     Für alle  $x \in S$   $b_{h(x)} := b_{h(x)} + 1$ ;  $B_i := B_i \cup \{x\}$
5.     Until  $\sum_{i=0}^{M-1} b_i^2 < 4n$ .
6.     Für alle  $i \in \{0, 1, \dots, M - 1\}$
7.         Repeat
8.         gut := true
9.         Initialisiere leere Hashtabelle  $H_i$  der Größe  $b_i^2$ .
10.         Wähle  $h_i \in \mathcal{H}_l$  uniform zufällig.
11.         Für alle  $x \in B_i$ 
  - If  $H_i[h_i(x)]$  belegt Then gut := false
  - Else Speichere  $x$  in  $H_i[h_i(x)]$ .
12.     Until gut

# Über statisch perfektes Hashing

## Theorem 15.8

Algorithmus 15.7 konstruiert in erwarteter Zeit  $O(n)$  eine Datenstruktur der Größe  $O(n)$  mit zusätzlichem Speicherbedarf  $O(n)$  und wählte dabei Hashfunktionen so aus, dass jede anschließende Search-Operationen nur konstante Zeit braucht.

Beweis.

klar Search( $x$ ) mit Berechnung von  $h(x)$  und  $h_{h(x)}$  ✓

klar Speicherplatz explizit so beschränkt ✓

offen erwartete Laufzeit

# Erwartete Laufzeit der ersten Phase

Erste Phase (Zeilen 1–5)

Erinnerung  $E \left( \sum_{i=0}^{M-1} b_i^2 \right) < 2n$  (Lemma 15.6)

klar  $\sum_{i=0}^{M-1} b_i^2 \geq 0$

also  $\text{Prob} \left( \sum_{i=0}^{M-1} b_i^2 > 2 \cdot 2n \right) < 1/2$  (Markow)

also im Erwartungswert  $\leq 2$  Durchläufe

also erwartete Laufzeit  $O(n)$

## Erwartete Laufzeit der zweiten Phase

### Zweite Phase (Zeilen 6–12)

**Betrachte** einen Repeat-Until-Schleifendurchgang

**Erinnerung** W'keit für Ende  $\geq 1/2$  (Lemma 15.5)

**also** erwartete Laufzeit  $O\left(\sum_{i=0}^{M-1} b_i^2\right)$

**Erinnerung** Vorbedingung  $\sum_{i=0}^{M-1} b_i^2 < 4n$

**also** Gesamtlaufzeit  $O(n)$



**Fazit** statisches perfektes Hashing **praktikabel**  
und **leicht zu implementieren**

# Dynamisches Perfektes Hashing

Geht das auch bei dynamischen Wörterbüchern?

bei etwas reduzierten Wünschen ja

konkret Worst-Case-Zeit  $O(1)$  je Search und  
amortisierte erwartete Worst-Case-Zeit  $O(1)$  sonst

## Zentrale Ideen

- benutze gleichen Ansatz wie für statisches perfektes Hashing
- benutze Verdopplungsstrategie für Größe der Hashtabelle
- nach  $n$  Updates oder bei ungünstiger Verteilung, kompletter Neuaufbau der Hashtabelle
- offen Werte passend konkret definieren und nachrechnen, dass alles passt

Details Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, Tarjan (1994): Dynamic perfect hashing: upper and lower bounds

# Cuckoo Hashing

Pagh, Rodler (2004): Cuckoo Hashing

## Ideen

- Benutze zwei Hashtabellen  $T_1$  und  $T_2$  der Größe jeweils  $M = (1 + \varepsilon)n$ . ( $\varepsilon$  konstant)
- Falls  $n$  nicht bekannt, benutze übliche Verdopplungstaktik.
- Benutze zwei Hashfunktionen  $h_1$  und  $h_2$ , für jede Hashtabelle eine.
- Speichere  $x$  **entweder** in  $T_1[h_1(x)]$  **oder** in  $T_2[h_2(x)]$ .
- Führe komplettes Rehashing durch, wenn es Schwierigkeiten gibt.

**klar** Search( $x$ ) **einfach**

$x$  vorhanden  $\Leftrightarrow x$  steht in  $T_1[h_1(x)]$  oder in  $T_2[h_2(x)]$

**klar** Delete = Search + „markiere als gelöscht“

# Cuckoo Hashing: Insert

Eingabe  $x$

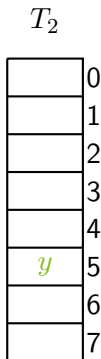
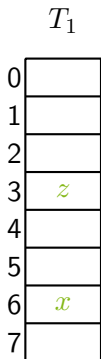
1. Search( $x$ ). Falls vorhanden, STOP.
2. Für  $k \in \{1, 2, \dots, k_{\max}\}$
3. Vertausche  $x$  und den Inhalt von  $T_1[h_1(x)]$ .
4. Falls  $x$  leer, STOP.
5. Vertausche  $x$  und den Inhalt von  $T_2[h_2(x)]$ .
6. Falls  $x$  leer, STOP.
7. RehashAll
8. Insert( $x$ )

Wahl des Parameters  $k_{\max}$  **später**

nach  $M^2$  Insert auf jeden Fall RehashAll

**RehashAll** wählt  $h_1$  und  $h_2$  randomisiert neu,  
fügt alle Elemente neu ein

# Beispiel Insert (erfolgreich)



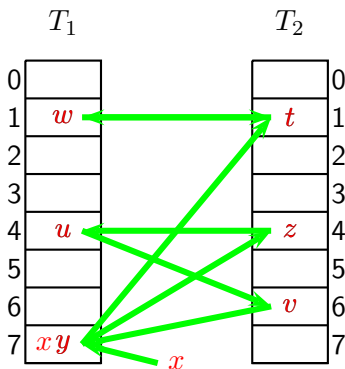
$$h_1(y) = 6, h_2(y) = 5$$

$$h_1(z) = 3, h_2(z) = 5$$

**Insert( $x$ )**

$$h_1(x) = 6, h_2(x) = 7$$

# Beispiel Insert (erfolglos)



$$h_1(t) = 1, h_2(t) = 1$$

$$h_1(u) = 4, h_2(u) = 6$$

$$h_1(v) = 7, h_2(v) = 6$$

$$h_1(w) = 1, h_2(w) = 1$$

$$h_1(y) = 7, h_2(y) = 4$$

$$h_1(z) = 4, h_2(z) = 4$$

**Insert( $x$ )**

$$h_1(x) = 7, h_2(x) = 1$$

# Auf dem Weg zur Analyse

klar entscheidend von Hashfunktionen abhängig

## Definition 15.9

Klasse von Hashfunktionen  $\mathcal{H} \subseteq \{h: \mathcal{U} \rightarrow \{0, 1, \dots, M-1\}\}$  heißt  $(c, k)$ -universell, wenn für alle paarweise verschiedenen  $x_1, x_2, \dots, x_k \in \mathcal{U}$  und für alle  $y_1, y_2, \dots, y_k \in \{0, 1, \dots, M-1\}$

$$\text{Prob}(h(x_1) = y_1, h(x_2) = y_2, \dots, h(x_k) = y_k) \leq \frac{c}{M^k}$$

gilt, wobei die W'keit über die zufällige Wahl von  $h \in \mathcal{H}$  bestimmt ist.

klar  $(c, k)$ -Universalität ist Verallgemeinerung von  $c$ -Universalität  
 $(c, 2)$ -universell  $\hat{=}$   $c$ -universell

# Die Hashfunktionen

**Fakt** Es gibt Klasse von Hashfunktionen  $\mathcal{H}$ ,

- deren Funktionen mit  $o(n)$  Worten beschreibbar sind,
- deren Funktionen in konstanter Zeit auswertbar sind und
- die eingeschränkt auf  $M^2$  Schlüssel  $(1, n^\delta)$ -universell ist für eine Konstante  $\delta > 0$  mit Wahrscheinlichkeit  $1 - O(1/n^2)$ .

**klar** mit  $W$ 'keit  $O(1/n^2)$  gewählte Hashfunktionen unbrauchbar

**klar** mit  $k_{\max} \leq n^\delta$  verhalten sich  $h_1$  und  $h_2$   
wie rein zufällige (also ideale) Hashfunktionen  
mit  $W$ 'keit  $1 - O(1/n^2)$

# Was bei Insert passiert. . .

## Beobachtungen

- Insert erzeugt Kette von „nestlosen“ Schlüsseln, beginnend mit  $x_1$ , das verdrängt  $x_2$ , das verdrängt  $x_3$ , . . . , bis leeres Feld erreicht oder  $i > k_{\max}$
- Es kann geschlossene Kreise geben, die mit Sicherheit zum Abbruch führen.

## Lemma 15.10

Sei  $x_1, x_2, \dots$  eine Sequenz von „nestlosen“ Schlüsseln bei Insert, die nicht Teil eines geschlossenen Kreises sind.

Für jeden Präfix  $x_1, x_2, \dots, x_p$  gibt es eine Teilfolge von  $\geq p/3$  verschiedenen Schlüsseln, die mit  $x_1$  beginnt.

## Beweis des Lemmas

1. Fall  $x_1, x_2, \dots$  ist kreisfrei.

dann  $x_1, x_2, \dots, x_p$  Sequenz mit geforderten Eigenschaften

2. Fall Es gibt  $i < j$  mit  $x_i = x_j$ ; seien  $i, j$  min.

1. Unterfall  $p < i + j$

erste  $j - 1 \geq \frac{i+j-1}{2} \geq \frac{p}{2}$  Schlüssel

sind Sequenz mit geforderten Eigenschaften

2. Unterfall  $p \geq i + j$

Sequenz  $x_1, x_2, \dots, x_{j-1}$  oder

Sequenz  $x_{i+j-1}, x_{i+j}, \dots, x_p$

hat geforderte Eigenschaften



## Zur Länge von Insert-Sequenzen

wollen wissen Prob (Insert-Sequenz hat Länge  $\geq t$ )

- 1. Fall:  $t > k_{\max}$  W'keit 0
- 2. Fall:  $t \leq k_{\max}$  und Hashfunktionen nicht  $(1, k_{\max})$ -universell W'keit  $O(1/n^2)$
- 3. Fall:  $t = k_{\max}$  und geschlossener Kreis aufgetreten W'keit ?
- 4. Fall:  $t \leq k_{\max}$  und mindestens  $(2t - 1)/3$  verschiedene Schlüssel beginnend mit  $x_1$  angefasst W'keit ?

### 3. Fall: geschlossener Kreis

**Notation**  $x_l$  schließt Kreis  
am Anfang  $v \leq l$  verschiedene Schlüssel

#Möglichkeiten, Kreis zu schließen

$$\leq \underbrace{v^2} \cdot \underbrace{v} \cdot \underbrace{M^{v-1}} \cdot \underbrace{n^{v-1}}$$

Werte für  $i$  und  $j$     Positionen für  $x_l$     Wahlen der Zellen    Wahlen der Schlüssel

$$= v^3 M^{v-1} n^{v-1}$$

**klar** jede Möglichkeit hat W'keit  $M^{-2v}$ , da  $h_1$  und  $h_2$  ideal

**also** W'keit  $\leq \sum_{v=3}^l \frac{v^3 M^{v-1} n^{v-1}}{M^{2v}} < \frac{1}{Mn} \sum_{v=3}^{\infty} v^3 \left(\frac{n}{M}\right)^v = O(1/n^2)$

## 4. Fall: kein geschlossener Kreis

Es gibt  $v = \lceil (2t - 1)/3 \rceil$  verschiedene Schlüssel  $b_1, \dots, b_v$ ,

$b_1 = x_1$  ursprünglich einzufügender Schlüssel,

$h_{\beta_1}(b_1) = h_{\beta_1}(b_2)$ ,  $h_{\beta_2}(b_2) = h_{\beta_2}(b_3)$ ,  $h_{\beta_1}(b_3) = h_{\beta_1}(b_4)$ , ...

mit  $(\beta_1, \beta_2) \in \{(1, 2), (2, 1)\}$ .

**Beobachtung**  $b_1 = x_1$  steht fest, darum noch  $n^{v-1}$  Wahlen der  $b_i$  möglich

**klar** W'keit je Möglichkeit  $M^{-(v-1)}$ , weil  $h_1, h_2$   $(1, k_{\max})$ -universell

**also** insgesamt W'keit  $2 \cdot \left(\frac{n}{M}\right)^{v-1} \leq 2(1 + \varepsilon)^{-(2t-1)/3+1}$

## Erwartete Länge von Insert-Sequenzen

$$\begin{aligned}
 E(T) &= \sum_{t=1}^{\infty} t \cdot \text{Prob}(T = t) = \sum_{t=1}^{\infty} \text{Prob}(T \geq t) \\
 &\leq 1 + \sum_{t=2}^{k_{\max}} \text{Prob}(T \geq t) \\
 &= 1 + \sum_{t=2}^{k_{\max}} \left( 2(1 + \varepsilon)^{-(2t-1)/3+1} + O(1/n^2) \right) \\
 &\leq 1 + O\left(\frac{k_{\max}}{n^2}\right) + 2 \sum_{t=2}^{\infty} \left( (1 + \varepsilon)^{-2/3} \right)^t \\
 &= O\left(1 + \frac{1}{1 - (1 + \varepsilon)^{-2/3}}\right) = O(1 + 1/\varepsilon)
 \end{aligned}$$

# Kosten des Rehashings

klar Laufzeit von RehashAll  $O(n)$

Beitrag durch schlechte Wahl von  $h_1, h_2$

$$O(n) \cdot O(1/n^2) = O(1/n)$$

Beitrag durch geschlossene Schleifen  $O(n) \cdot O(1/n^2) = O(1/n)$

Beitrag durch anderen Abbruch  $O(n) \cdot 2(1 + \varepsilon)^{-(2k_{\max}-1)/3+1}$

$$\text{Wahl } k_{\max} := \lceil 3 \log_{1+\varepsilon} M \rceil$$

$$\text{also Beitrag } O(n) \cdot O(1/n^2) = O(1/n)$$

**Fazit** Cuckoo-Hashing realisiert dynamisch perfektes Hashing  
einfach und auch praktisch effizient

# Bioinformatik

## Hauptproblem Computergestützte Sequenzanalyse

Sequenzen typischerweise auf Alphabeten  $\Sigma = \{A, C, G, T\}$  (für die vier Basen Adenin, Guanin, Cytosin und Thymin),

$\Sigma = \{A, C, G, U\}$  (mit Uracil statt Thymin), oder

$\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$   
(Aminosäuren)

**Aber auch** Datenbanken, Visualisierung, Modellierung, Simulation, Netzwerke, . . .

**Viele Teilprobleme kommen z.B. in der SNP-Analyse vor**

**Was sind SNPs?**

**Ein Ziel:** Identifizierung von SNPs und SNP-Interaktionen die zu einem höheren Krankheitsrisiko führen

## Single Nucleotide Polymorphism

- Variationen einzelner Basenpaare eines DNA-Strangs
- Auftreten bei mehr als 1% der Population
- Stellen etwa 90% aller genetischen Varianten

## Beispiel eines SNPs

