

# Algorithmen und Berechenbarkeit

## Vorlesung 07

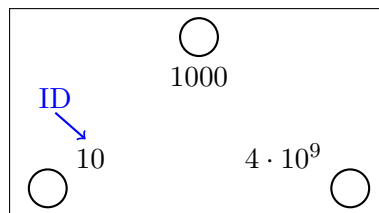
Letztes Update: 2018/02/25 - 18:13 Uhr

### Hashing

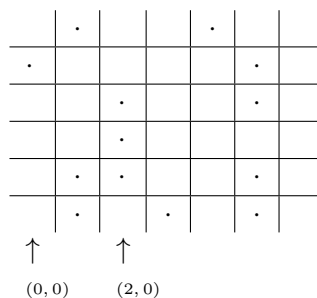
#### Einleitung

Bei der Implementierung des Algorithmus vom Closest-Pair Problem wird üblicherweise auf Hashing zurückgegriffen. Hashing beschreibt dabei eine Funktion bzw. Abbildung, *die eine große Eingabemenge (die Schlüssel) auf eine kleinere Zielmenge (die Hashwerte) abbildet*<sup>1</sup>.

Auch bei Telefon- und Wörterbüchern (vgl. dazu *Wörterbuchproblem*) oder für Anwendungen, die OpenStreetMap-Daten verwenden, wird oft auf Hashing zurückgegriffen.



Beim Closest-Pair-Algorithmus kann mittels Hashing die Gitterzelle errechnet werden. Jede Gitterzelle erhält Koordinaten.



Die Maschenweite des Gitters sei  $w$ . Dann fällt ein Punkt  $P(p_x, p_y)$  in die Gitterzelle

$$\left( \left\lfloor \frac{p_x}{w} \right\rfloor, \left\lfloor \frac{p_y}{w} \right\rfloor \right)$$

↙  
abgerundet

---

<sup>1</sup><https://de.wikipedia.org/wiki/Hashfunktion>

### Beispiel

Sie  $w = \frac{1}{2}$  und  $P(p_x, p_y) = \left(\frac{15}{10}, \frac{7}{10}\right)$ . Man erhält für das Gitter die Koordinaten

$$\left(\left\lfloor \frac{P_x}{w} \right\rfloor, \left\lfloor \frac{P_y}{w} \right\rfloor\right) = \left(\left\lfloor \frac{\frac{15}{10}}{\frac{1}{2}} \right\rfloor, \left\lfloor \frac{\frac{7}{10}}{\frac{1}{2}} \right\rfloor\right) = (3, 1)$$

Diese Koordinaten müssen noch ghasht werden.

### Hashing formal

Gegeben sei **erstens** das *Universum*  $U$ , das immer sehr groß gewählt wird und typischerweise eine Teilmenge der natürlichen Zahlen darstellt, **zweitens** sei auch die vergleichsweise kleine Menge  $S \subseteq U$  gegeben. Daraus folgt **drittens**  $n = |S|$ , also die Anzahl der Elemente in  $S$ .

Das Ziel ist nun

Finde  $h : U \rightarrow \{0, 1, \dots, m-1\}$

$$\text{sodass } \forall 0 \leq i < m : \left| \{x \in S \mid h(x) = i\} \right| \leq \left\lceil \frac{n}{m} \right\rceil$$

Man sucht also eine Funktion  $h$ , sodass keine zwei Elemente aus  $S$  auf dieselbe Zahl zeigen.

### Beispiel

Seien  $U = \mathbb{N}$ ,  $S = \{1, 7, 23, 99\}$ ,  $n = 4$  und  $m = 5$  gegeben. Dann ist die Funktion

$$h(x) = x \bmod 5$$

eine sehr gute Hashfunktion für  $S$ .

$$\begin{aligned} \rightarrow h(1) &= 1 \\ \rightarrow h(7) &= 2 \\ \rightarrow h(23) &= 3 \\ \rightarrow h(99) &= 4 \end{aligned}$$

Für die Menge  $S' = \{2, 17, 22, 32\}$  ist  $h(x)$  aber eine sehr schlechte Hashfunktion.

$$\begin{aligned} \rightarrow h(2) &= 2 \\ \rightarrow h(17) &= 2 \\ \rightarrow h(22) &= 2 \\ \rightarrow h(32) &= 2 \end{aligned}$$

**Satz:** Seien  $U, m$  und  $h$  gegeben, und sei  $k = |U|$  sowie  $n = |S| \Rightarrow S \in \binom{U}{n}$ . Dann gibt es für jedes  $n$  mit  $1 \leq n \leq \frac{k}{m}$  ein  $S \subseteq U$ , sodass alle Elemente aus  $S$  von  $h$  auf denselben Wert in  $\{0, 1, \dots, m-1\}$  abgebildet werden.

**Beweis:** Nach Schubfachsystem existiert ein  $i$  mit  $0 \leq i < m$ , sodass

$$\left| h^{-1}(i) \right| \geq \left\lceil \frac{k}{m} \right\rceil$$

Nach Annahme ist  $\frac{k}{m} \geq n$ .

Man kann nun  $S \subseteq h^{-1}(i)$  mit  $|S| = n$  wählen für ein beliebiges  $i$  mit  $|h^{-1}(i)| \geq \left\lceil \frac{k}{m} \right\rceil$ . □

Um eine gute Hashfunktion zu finden, müssen also immer auch die Daten betrachtet werden, die mit dieser Funktion ghasht werden.

## Datenstruktur

Man betrachte die Datenstruktur *Wörterbuch*, die folgende Operationen unterstützt.

<code>makeset()</code>	Erzeugt ein leeres Wörterbuch
<code>insert(x, S)</code>	Fügt <u>Item <math>x</math></u> in Wörterbuch ein, überschreibt falls vorhanden <div style="text-align: center; margin-left: 100px;"> <math>\underbrace{\text{Key} + \text{Information}}</math> </div>
<code>delete(x, S)</code>	Löscht Item $x$ aus $S$
<code>lookup(x, S)</code>	Gibt Item $x = \underbrace{(x, \text{Info})}_{\text{Paar}}$ aus, falls vorhanden

Man sucht eine Datenstruktur, die Zugriffe in  $\mathcal{O}(1)$  erlaubt und dabei nicht mehr als  $\mathcal{O}(n)$  Platz verbraucht.

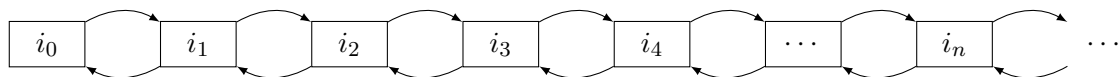
### Implementierungsansatz 1: Array für Items, Zähler für $|S|$

$i_0$	$i_1$	$i_2$	$i_3$	$i_4$	$\dots$	$i_n$
-------	-------	-------	-------	-------	---------	-------

- `makeset()` Diese Operation gelingt in  $\mathcal{O}(1)$ .
- `insert(x, S)` kostet  $\mathcal{O}(1)$ , falls bekannt ist, dass noch kein Item mit demselben Schlüssel in  $S$  existiert. Ansonsten muss zuerst in  $\mathcal{O}(n)$  geprüft werden, ob der Schlüssel bereits enthalten ist, bevor das Item in  $\mathcal{O}(1)$  eingefügt werden kann.
- `delete(x, S)` Da nicht mithilfe des Index gelöscht wird, müssen die Einträge von  $S$  durchlaufen werden, um das Item zu löschen. Das braucht  $\mathcal{O}(n)$ .
- `lookup(x, S)` Es müssen wie bei `delete(x, S)` die Einträge durchlaufen werden, um das gesuchte Element zu finden. Das braucht ebenfalls  $\mathcal{O}(n)$ .

Vorteile	Nachteile
<ul style="list-style-type: none"> <li>• Einfach</li> <li>• Platzsparend</li> </ul>	<ul style="list-style-type: none"> <li>• Performance bei <code>delete(x, S)</code></li> <li>• Performance bei <code>lookup(x, S)</code></li> </ul>

### Implementierungsansatz 2: Einfach/doppelt verkettete Listen



Im Allgemeinen verhält sich diese Datenstruktur für den Wörterbuchansatz recht ähnlich wie das Array aus dem ersten Ansatz.

### Implementierungsansatz 3: Direkte Adressierung

Für diese Datenstruktur wird angenommen, dass das Schlüsseluniversum endlich und nicht zu groß ist:

$$U := \{0, 1, 2, \dots, n-1\}$$

Man legt nun wie im ersten Ansatz ein Array an, diesmal mit der Größe  $k$ . An Position  $i$  steht die Information für Schlüssel  $i$ , falls eine Information für diesen Schlüssel abgelegt wurde. Ansonsten erhält man NIL.

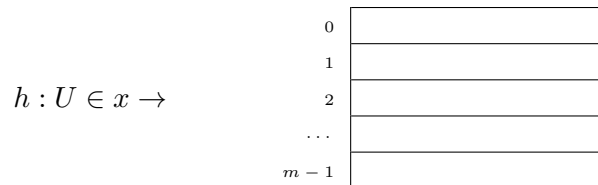
Vorteil	Nachteil
• Alle Operationen bis auf <code>makeset()</code> in $\mathcal{O}(1)$	• Platz und Größe des Schlüsseluniversums (nicht der Menge $S$ )

#### Implementierungsansatz 4: Suchstrukturen wie (2,3,4)-, AVL- oder RS-Bäume

Vorteil	Nachteil
• Platzverbrauch tatsächlich $\mathcal{O}(n)$	• Zugriffszeit nur in $\mathcal{O}(\log(n))$

#### Hashing mit Verkettung

Angenommen, ein gewähltes  $h$  ist nicht injektiv für  $S$ , das bedeutet, es gibt mehrere Elemente in  $x, y \in S$  für die gilt  $h(x) = h(y)$ . Man kann damit dennoch eine Hashdatenstruktur bauen:



Jeder Hascheintrag ist Kopf einer einfach verketteten Liste,  $x \in S$  wird in der  $h(x)$ -ten verketteten Liste gespeichert.

- **Platzbedarf:**

$$\mathcal{O}(m + n) = \mathcal{O}\left(n \cdot \left(1 + \frac{1}{\beta}\right)\right)$$

$\beta = \frac{n}{m}$  ist der *Belegungsfaktor*. Je kleiner  $\beta$ , desto ineffizienter ist die Datenstruktur, aber möglicherweise ist es dann einfacher, eine *gute* Hashfunktion zu finden.

- **Zugriffszeit:** Man nimmt an,  $h(x)$  kann in  $\mathcal{O}(1)$  ausgewertet werden. Dann ist der Zugriff  $x \in S$  in

$$\mathcal{O}(1 + \text{Position von } x \text{ in Liste } L_{h(x)})$$

und der Zugriff auf  $x \in U \setminus S$  in

$$\mathcal{O}(1 + \text{Länge von } L_{h(x)})$$

#### Erwartete Suchzeit

Man nimmt an,  $h$  verteilt  $U$  gleichmäßig über  $\{0, 1, 2, \dots, m-1\}$ , das bedeutet

$$\forall i \in \{0, 1, 2, \dots, m-1\} : |\{k \in U \mid h(k) = i\}| \leq \left\lceil \frac{|U|}{m} \right\rceil$$

Zum Beispiel  $h(x) = x \bmod m$

**Satz:** Sei  $x$  ein zufälliges (gleichverteiltes) Element aus  $U \setminus S$  und  $h \leq \frac{|U|}{2}$ . Die erwartete Suchzeit nach Element  $x$  ist dann  $\mathcal{O}(1 + \beta)$

**Beweis (erster Teil):** Sei  $l_i$  die Anzahl der Elemente aus  $S$ , die in  $L_i$  gespeichert wurde  $= |L_i|$ . Es gilt  $\sum_{i=0}^{m-1} l_i = n = |S|$ . Die erwartete Suchzeit ist damit

$$E := \left( \sum_{i=0}^{m-1} \underbrace{\Pr(h(i) = i)}_{\text{Beweis-Einschub}} \cdot l_i \right) + 1$$

**Beweis (Einschub):**

Hierbei sind die Elemente  $U_i$  diejenigen Elemente aus  $U$ , welche auf  $i$  gehasht werden.

$$\begin{aligned}
 \Pr(h(x) = i) &= \frac{|U_i \setminus S|}{|U \setminus S|} \\
 &\leq \frac{|U_i|}{\underbrace{|U \setminus S|}_U} \\
 &\geq \frac{2}{2} \\
 &\leq \frac{\left\lceil \frac{|U|}{m} \right\rceil}{\frac{|U|}{2}} \\
 &\leq \frac{\frac{|U|}{m} + 1}{\frac{|U|}{2}} \\
 &= \frac{2}{m} + \underbrace{\frac{2}{|U|}}_{n \leq \frac{|U|}{2}} \leq \underbrace{\frac{2}{m} + \frac{1}{n}}_{\substack{\text{Im zwei-} \\ \text{ten Teil} \\ \text{einsetzen}}}
 \end{aligned}$$

**Beweis (zweiter Teil):**

$$\begin{aligned}
 E &:= \left( \sum_{i=0}^{m-1} \Pr(h(i) = i) \cdot l_i \right) + 1 \\
 &\leq \left( \sum_{i=0}^{m-1} \left( \frac{2}{m} + \frac{1}{n} \right) \cdot l_i \right) + 1 \\
 &= 1 + \frac{2}{m} \underbrace{\sum_{i=0}^{m-1} l_i}_n + \frac{1}{n} \underbrace{\sum_{i=0}^{m-1} l_i}_n \\
 &= 1 + \frac{2n}{m} + 1 \\
 &= 2 + \frac{2n}{m} \\
 &= \mathcal{O}(1 + \beta)
 \end{aligned}$$

□