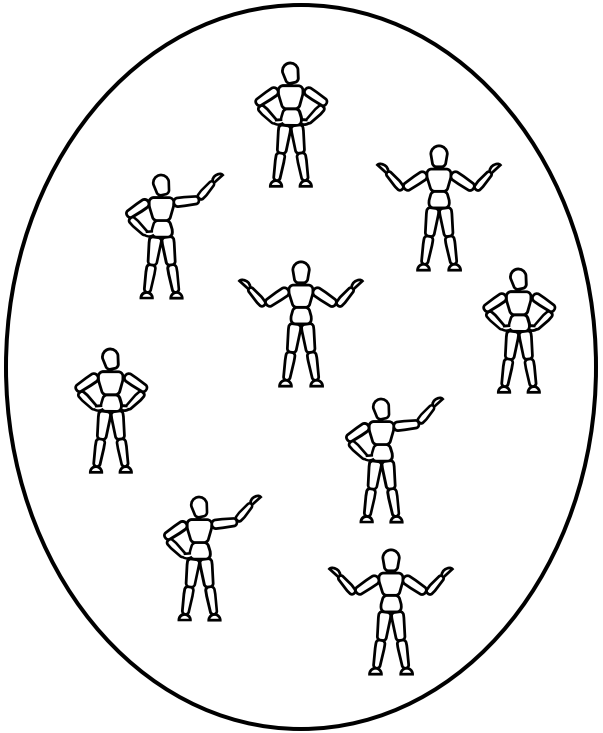
Technische Universität Braunschweig

Algorithmen und Datenstrukturen 2

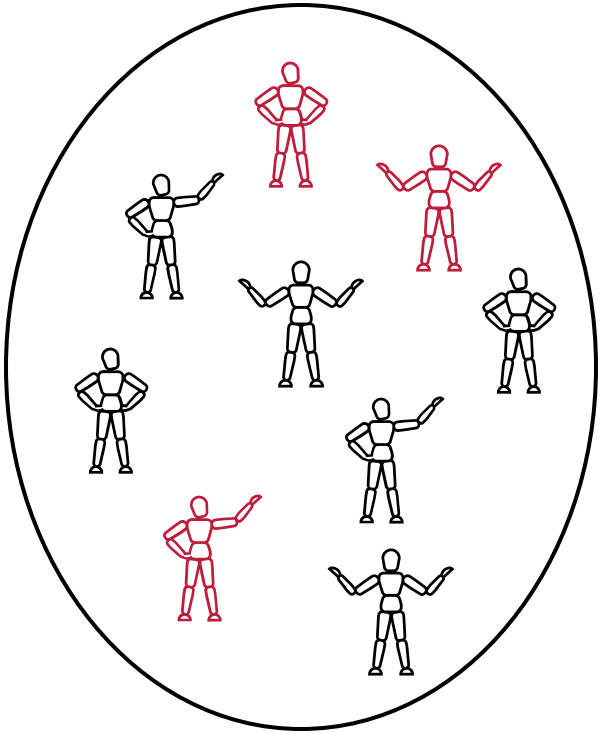
Arne Schmidt

Großes Universum, kleine Welt



Sehr große Anzahl an möglichen Datensätzen.

Großes Universum, kleine Welt



Sehr große Anzahl an möglichen Datensätzen.

Am Ende wird nur ein ganz kleiner Teil benötigt!

Was passiert mit diesen Datensätzen?

Aus AuD 1, dynamische Verwaltung der Daten:

- Search(x), insert(x), delete(x)
- Datenstrukturen:
  - Binäre Suchbäume
  - Heaps
  - ...

Kapitel 7 – Hashing

Hashtabelle

Definition 7.1
Eine **Hashtabelle** der Größe m besteht aus einem Array T mit Speicherzellen $T[0], \dots, T[m-1]$. Die Datensätze werden in dieser Hashtabelle gespeichert.

Eine **Hashfunktion** $h: U \to \{0, \dots, m-1\}$ liefert für jeden Schlüssel x aus einer Menge U eine Adresse in der Hashtabelle

Für gut gewähltes m und h können Operationen (search, insert, delete) in erwarteter konstanter Zeit durchgeführt werden.

Datenstrukturen aus AuD 1 benötigen im Mittel $O(\log n)$ Zeit.

Definition 7.2
Der Quotient $\beta = \frac{n}{m}$ einer Hashtabelle mit m Speicherzellen und n eingefügten Datensätzen heißt **Belegungsfaktor**.

Hashing Aufwand

Die Laufzeit einen Datensatz in die Tabelle einzufügen, hängt von vielen Dingen ab...

Wie schnell kann man h berechnen?

Was machen wir, wenn ein bestimmter Platz schon belegt ist?

Kapitel 7.1 – Hashfunktionen

Hashverfahren

Definition 7.3
Ein Hashverfahren ist gegeben durch:
- Eine Hashtabelle
- Eine Hashfunktion
- Strategie zur Auflösung möglicher Adresskollisionen.

Die Hashfunktion sollte dabei
- **Surjektiv** sein (ganzer Wertebereich wird abgedeckt)
- Schlüssel möglichst gleichmäßig verteilen, d.h. für $i, j \in \{0, \dots, m-1\}$ ist $|h^{-1}(i)| \approx |h^{-1}(j)|$
- Effizient berechenbar sein

Wir nehmen vereinfacht an, dass $U = \{0, 1, \dots, N-1\}$.
Eine einfache Hashfunktion ist dann $h(x) = x \bmod m$.

Beispiel 7.4

Sei $m = 11$ und $S = \{49, 22, 6, 52, 76, 34, 13, 29\}$, sowie $h(x) = x \bmod m$.

x	49	22	6	52	76	34	13	29
h(x)	5	0	6	8	10	1	2	7

Nun kommt der Wert 33 hinzu.
h(33) = 0, müsste also auch in die Zelle 0 eingetragen werden. Eine Kollision!

So etwas ist unvermeidbar, da $|U| \gg m$ ist.

Kapitel 7.2 – Kollisionen

Geburtstagsparadoxon

Wie wahrscheinlich ist es, dass zwei Personen von n zufällig gewählten am gleichen Tag Geburtstag haben?

Wie viele Personen benötigt man für eine Chance von ~50%?

JANUARY

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

FEBRUARY

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

MARCH

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

APRIL

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

MAY

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUNE

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

JULY

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

AUGUST

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

SEPTEMBER

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

OCTOBER

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

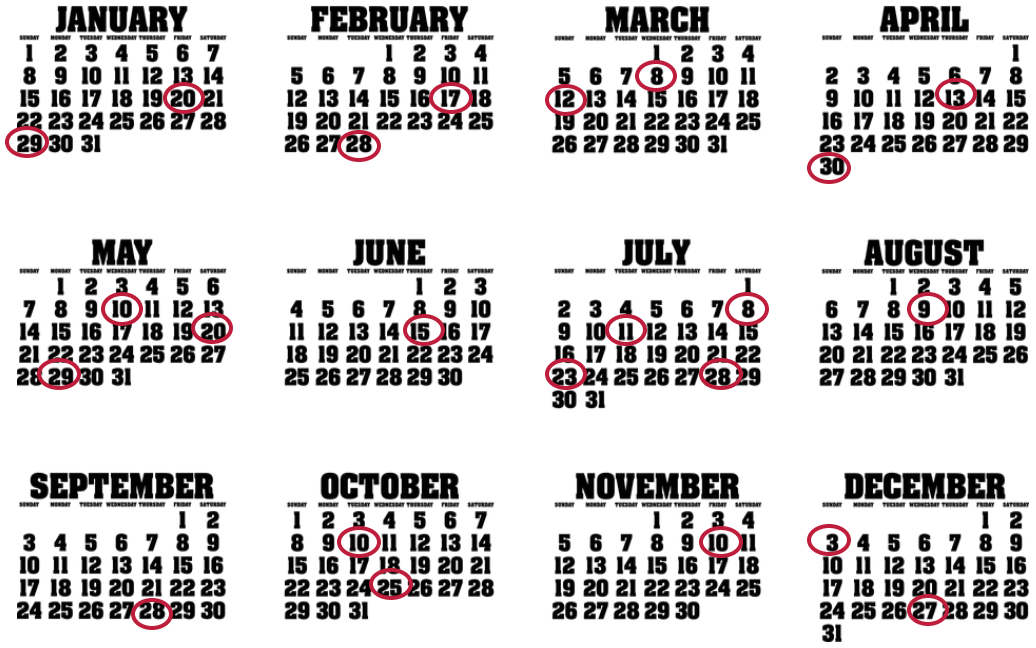
NOVEMBER

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

DECEMBER

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Geburtstagsparadoxon



Angenommen:

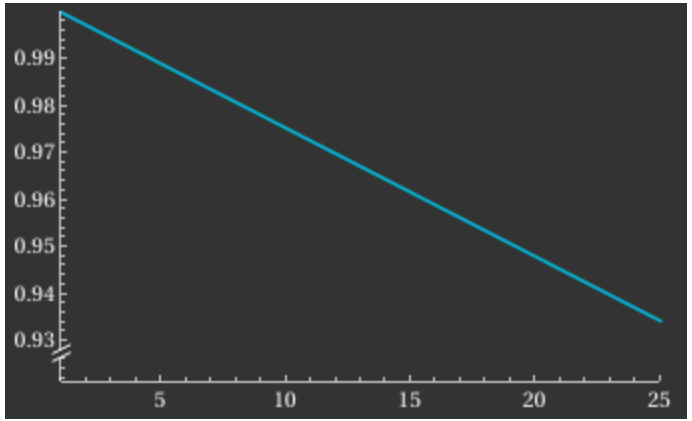
- Geburtstage sind gleichverteilt, d.h. $\text{Prob}(h(x) = j) = \frac{1}{m}$
- Es sind bereits i-1 Personen mit unterschiedlichen Tagen ausgewählt worden.

Wie hoch ist die W‘keit, dass auch Person i an einem unterschiedlichen Tag Geburtstag hat?

$$\frac{m-(i-1)}{m} = \frac{366\ -i}{365}$$

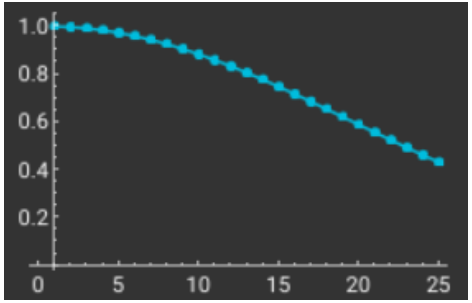
Wahrscheinlichkeiten

$$\frac{366 - i}{365}$$



W‘keit, dass die i-te Person kollisionsfrei ist

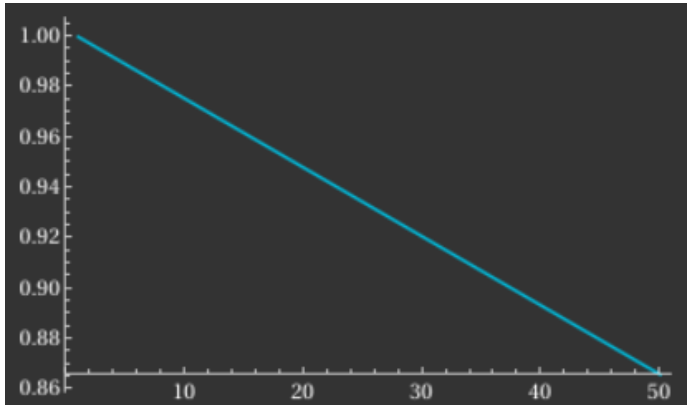
$$\prod_{x=1}^{i} \frac{366 - x}{365}$$



W‘keit, dass alle i Person kollisionsfrei sind.
Für i = 23 ist das ~0,49

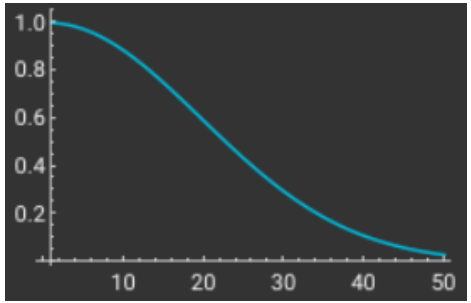
Wahrscheinlichkeiten

$$\frac{366 - i}{365}$$



W‘keit, dass die i-te Person kollisionsfrei ist

$$\prod_{x=1}^{i} \frac{366 - x}{365}$$



W‘keit, dass alle i Person kollisionsfrei sind.
Für i = 23 ist das ~0,03

Kollisionswahrscheinlichkeit

Satz 7.7
Die Wahrscheinlichkeit, dass $2\sqrt{m}$ Daten kollisionsfrei eingefügt werden können, liegt bei höchstens 36,79%.

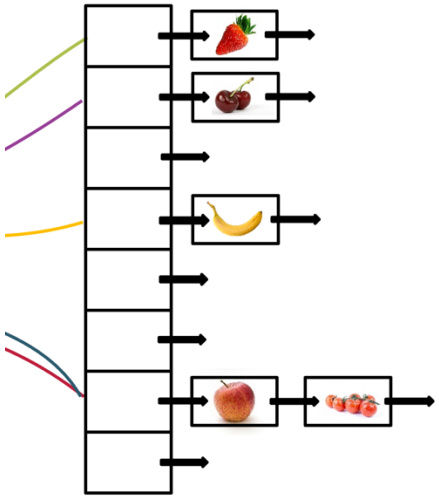
Beweis:

$$Prob(2\sqrt{m}\text{ Daten kollisionsfrei}) = \frac{m-1}{m} \cdot \ldots \cdot \frac{m-\sqrt{m}}{m} \cdot \ldots \cdot \frac{m-2\sqrt{m}+1}{m}$$
$$\leq \frac{m-\sqrt{m}}{m} \cdot \ldots \cdot \frac{m-2\sqrt{m}+1}{m}$$
$$\leq \left(\frac{m-\sqrt{m}}{m}\right)^{\sqrt{m}} = \left(1-\frac{1}{\sqrt{\mathrm{m}}}\right)^{\sqrt{m}} \approx \frac{1}{e} \approx 0.3679$$

Etwas genauer analysiert kann man zeigen, dass die W‘keit höchstens etwa 13.53% beträgt.

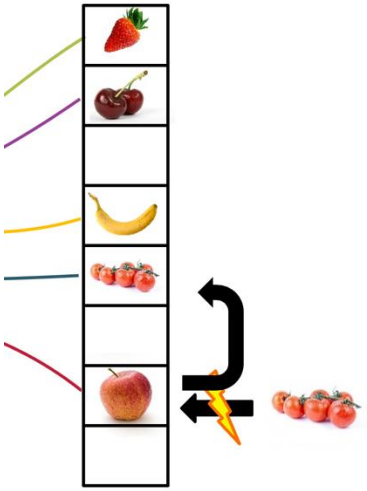
Kollisionsbehandlung

Listen



Daten mit gleichem Hashwert werden in einer verketteten Liste gespeichert.

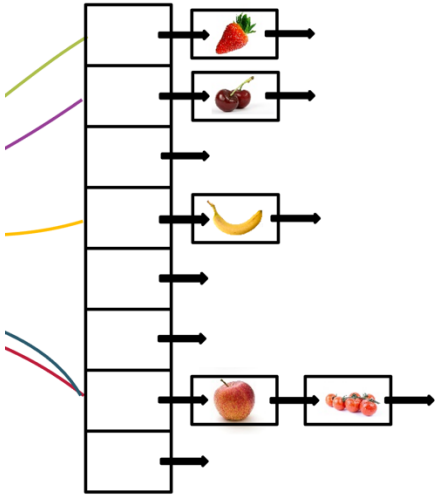
Offene Adressierung



Suche nach fester Regel (Sondierungsregel) einen freien Platz

Kapitel 7.2.1 – Verkettete Listen

Verkettete Listen



Daten mit gleichem Hashwert werden in einer verketteten Liste gespeichert.

Vorteil:
n darf größer als m sein.
(Ist $n \gg m$ ist Rehashing ratsam)

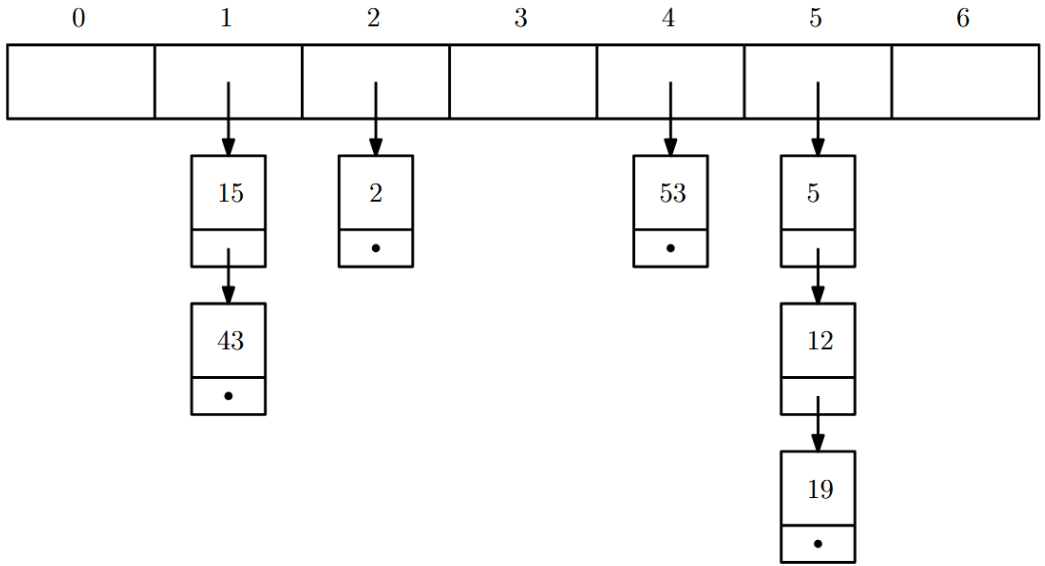
Nachteil:
Zusätzlicher Speicher für die Zeiger nötig

Operationen:

- **search(x)**: Berechne h(x) und suche in Liste L(h(x)) nach x
- **insert(x)**: Nach erfolgloser Suche: füge x in Liste L(h(x)) ein.
- **delete(x)**: Nach erfolgreicher Suche: entferne x aus L(h(x)).

Beispiel 7.8

Sei $m = 7$ und $h(x) = x \bmod m$.
Betrachte Schlüsselmenge $S = \{2, 5, 12, 15, 19, 43, 53\}$. Einfügen in die Hashtabelle liefert



Listengröße

Satz 7.9
Bei zufälligen Daten und ideal streuenden Hashfunktionen enthält eine beliebige Liste L(j) im Erwartungsfall $\beta = \frac{n}{m}$ viele Elemente.

Sei

$$X_{ij} := \begin{cases} 1, & i\text{-tes Datum kommt in Liste } L(j) \\ 0, & \text{sonst} \end{cases}$$

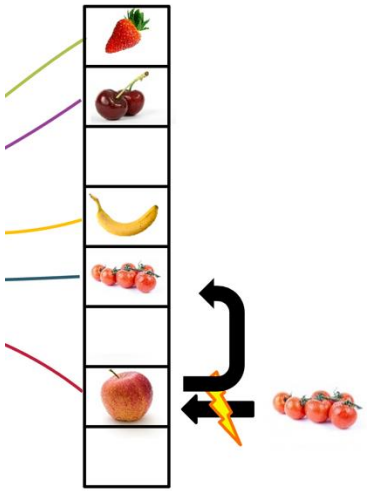
Es gilt $\text{Prob}(X_{ij} = 1) = \frac{1}{m}$ und damit $E(X_{ij}) = 1 \cdot \frac{1}{m} + 0 \cdot \frac{m-1}{m} = \frac{1}{m}$.
Mit $X_j = \sum_{i=1}^{n} X_{ij}$, also wieviele Daten in Liste L(j) kommen, ist $E(X_j) = \sum_{i=1}^{n} E(X_{ij}) = \frac{n}{m} = \beta$.

Laufzeit

Für $O(m)$ Daten haben wir also eine erwartete Laufzeit von $O(1)$ für alle drei Operatoren!

Kapitel 7.2.2 – Offene Adressierung

Offene Adressierung



Suche nach fester Regel (Sondierungsregel) einen freien Platz

Sei $t(i,j)$ die Position des i-ten Versuchs zum Einfügen von Datum x mit $h(x) = j$.

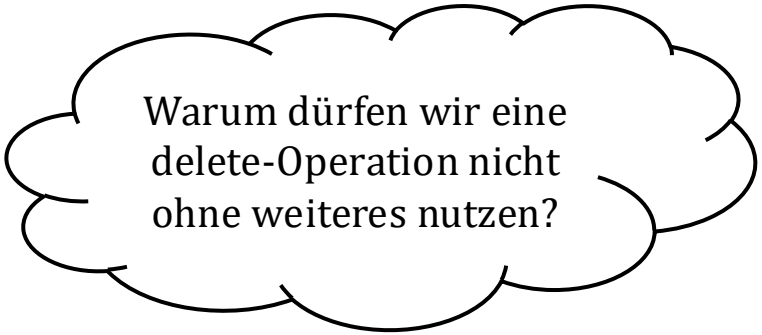
Dazu fordern wir:

- t ist in $O(1)$ berechenbar.
- $t(0,j) = j$
- $t(\cdot,j): \{0,\dots,m-1\} \to \{0,\dots,m-1\}$ bijektiv

Operationen:

- **search(x)**: Berechne $j = h(x)$ und suche an Positionen $t(0,j),\dots,t(m-1,j)$ nach x. Brich ab, wenn x oder freie Stelle gefunden.
- **insert(x)**: Nach erfolgloser Suche: Suche freie Stelle und füge x dort ein.

Delete



Löschen erzeugt Lücken, sodass eine Suche vorzeitig abbricht, weil in der Sondierung ein freies Feld gefunden wurde.

„Ausweg“: Markiere Felder als „schon mal belegt“.
Das müllt allerdings die Tabelle voll und die Suche wird ineffizient, sodass ein Rehashing notwendig wird.

Kapitel 7.2.2 – Offene Adressierung (Sondierungsregeln)

Lineares Sondieren

Bei linearem Sondieren ist $t(i,j)$ in der Form $t(i,j) = ci + j \bmod m$ mit $c \in \mathbb{N}$.

Beispiel 7.10
Sei $m = 19, j = h(x) = 7$ und $t(i,j) = i + j \bmod 19$. Dann ist die Sondierungsfolge:
$7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 0, 1, 2, 3, 4, 5, 6$

Beispiel 7.11
Betrachte eine Hashtabelle mit $m = 19$ Feldern, wobei Positionen $2, 5, 6, 9, 10, 11, 12, 17$ bereits belegt sind.

$h(x)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Landet an Position	0	1	3	3	4	7	7	7	8	13	13	13	13	13	14	15	16	18	18
W'keit	$\frac{1}{19}$	$\frac{1}{19}$	0	$\frac{2}{19}$	$\frac{1}{19}$	0	0	$\frac{3}{19}$	$\frac{1}{19}$	0	0	0	0	$\frac{5}{19}$	$\frac{1}{19}$	$\frac{1}{19}$	$\frac{1}{19}$	0	$\frac{2}{19}$

Ideales Hashing

Definition 7.13
Beim **idealen Hashing** haben alle $\binom{m}{n}$ Möglichkeiten, die n besetzten Plätze für m Schlüssel auszuwählen, die gleiche Wahrscheinlichkeit.

Lineares Hashing ist weit vom idealen Hashing entfernt!

$h(x)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Landet an Position	0	1	3	3	4	7	7	7	8	13	13	13	13	13	14	15	16	18	18
W'keit	$\frac{1}{19}$	$\frac{1}{19}$	0	$\frac{2}{19}$	$\frac{1}{19}$	0	0	$\frac{3}{19}$	$\frac{1}{19}$	0	0	0	0	$\frac{5}{19}$	$\frac{1}{19}$	$\frac{1}{19}$	$\frac{1}{19}$	0	$\frac{2}{19}$

Quadratisches Sondieren

Bei quadratischem Sondieren wächst der additive Term quadratisch statt linear.

Beispiel 7.14

Sei $m = 19, j = h(x) = 7$ und $t(i,j) = j + (-1)^{i+1} \cdot \left\lfloor \frac{i+1}{2} \right\rfloor^2 \mod 19$. Dann ist die Sondierungsfolge:
7, 8, 6, 11, 3, 16, 17, 4, 10, 13, 1, 5, 9, 18, 15

Frage: $t(\cdot, j)$ für alle j, m bijektiv?
Nein, aber immer wenn $m \equiv 3 \mod 4$.

Das ist insgesamt schon besser geeignet als lineares Sondieren.

Multiplikatives Sondieren

Bei multiplikativem Sondieren werden Hashwert und Zähler miteinander Multipliziert.
Dabei nimmt der Zähler nur die Werte $1, \dots, m-1$ an.
Achtung: $j = h(x) \neq 0$ für alle Schlüssel x. (Warum?)

Beispiel 7.15
Sei $m = 19, j = h(x) = 7$ und $t(i,j) = i \cdot j \bmod 19$. Dann ist die Sondierungsfolge:

ij	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126
$ij \bmod 19$	7	14	2	9	16	4	11	18	6	13	1	8	15	3	10	17	5	12

Frage: $t(\cdot, j)$ für alle j, m bijektiv?
Nein, aber immer wenn m eine Primzahl ist und $j \neq 0$.

Doppeltes Hashing

Nutze zwei Hashfunktionen $h_1(x), h_2(x)$, welche additiv verknüpft werden.
Einer der beiden wird multiplikativ mit dem Versuchszähler $0 \leq i \leq m-1$ verknüpft.

Beispiel 7.16
Sei $m = 19, x = 47$ und $h(i,x) = h_1(x) + i \cdot h_2(x) \bmod 19$ mit

$$h_1(47) = 47 \bmod 19 = 9 \text{ und } h_2(47) = 47 \bmod 17 + 1 = 14.$$

Dann ist die Sondierungsfolge:
9, 4, 18, 13, 8, 3, 17, 12, 7, 2, 16, 11, 6, 1, 15, 10, 0, 14

Beobachtung 7.17
$i \cdot h_2(x)$ durchläuft für $1 \leq i \leq m-1$ die Werte $1, \ldots, m-1$ in irgendeiner Reihenfolge.
Ergänzt wird $0 = 0 \cdot h_2(x)$.
Durch den Summanden $h_1(x)$ wird der Start zufällig verschoben.
Doppeltes Hashing kommt dem idealen Hashing am nächsten.

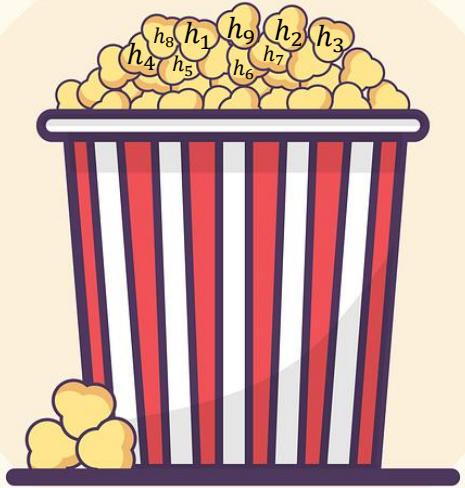
Kapitel 7.3 – Universelles Hashing

Worst-Case

Ich kann für jede Hashfunktion schnell Instanzen bauen, die sehr schlecht performen.

Wie kann ich dafür sorgen, dass auch für solche Worst-Case-Instanzen eine gute Performance zu erwarten ist?

Universelles Hashing



Wähle zur Laufzeit eine Hashfunktion aus einer Familie von Hashfunktionen mit besonderen Eigenschaften.

Damit hat man für jede Schlüsselmenge eine zufällige Hashfunktion und im Erwartungsfall eine gute Laufzeit.