

7 Hashing

Algorithmen und Datenstrukturen 2
Sommer 2026

Prof. Dr. Sándor Fekete

Lehrevaluation!



Gruppe	Termin (siehe Semesterplan)	Raum	Tutor*in
01	Montag, 9:45 – 11:15	IZ 305	Henriette Watolla
02	Dienstag, 13:15 – 14:45	IZ 305	Bastian Bosse-Graf
03	Mittwoch, 13:15 – 14:45	IZ 305	Henrik Heitmann
04	Mittwoch, 16:45 – 18:15	IZ 161	Bastian Bosse-Graf
05	Mittwoch, 16:45 – 18:15	IZ 305	Simon Laschinger
06	Donnerstag*, 8:00 – 9:30	IZ 305	Henrik Heitmann
07	Freitag*, 11:30 – 13:00	IZ 305	Linus Claußen
08	Freitag*, 13:15 – 14:45	IZ 305	Henriette Watolla

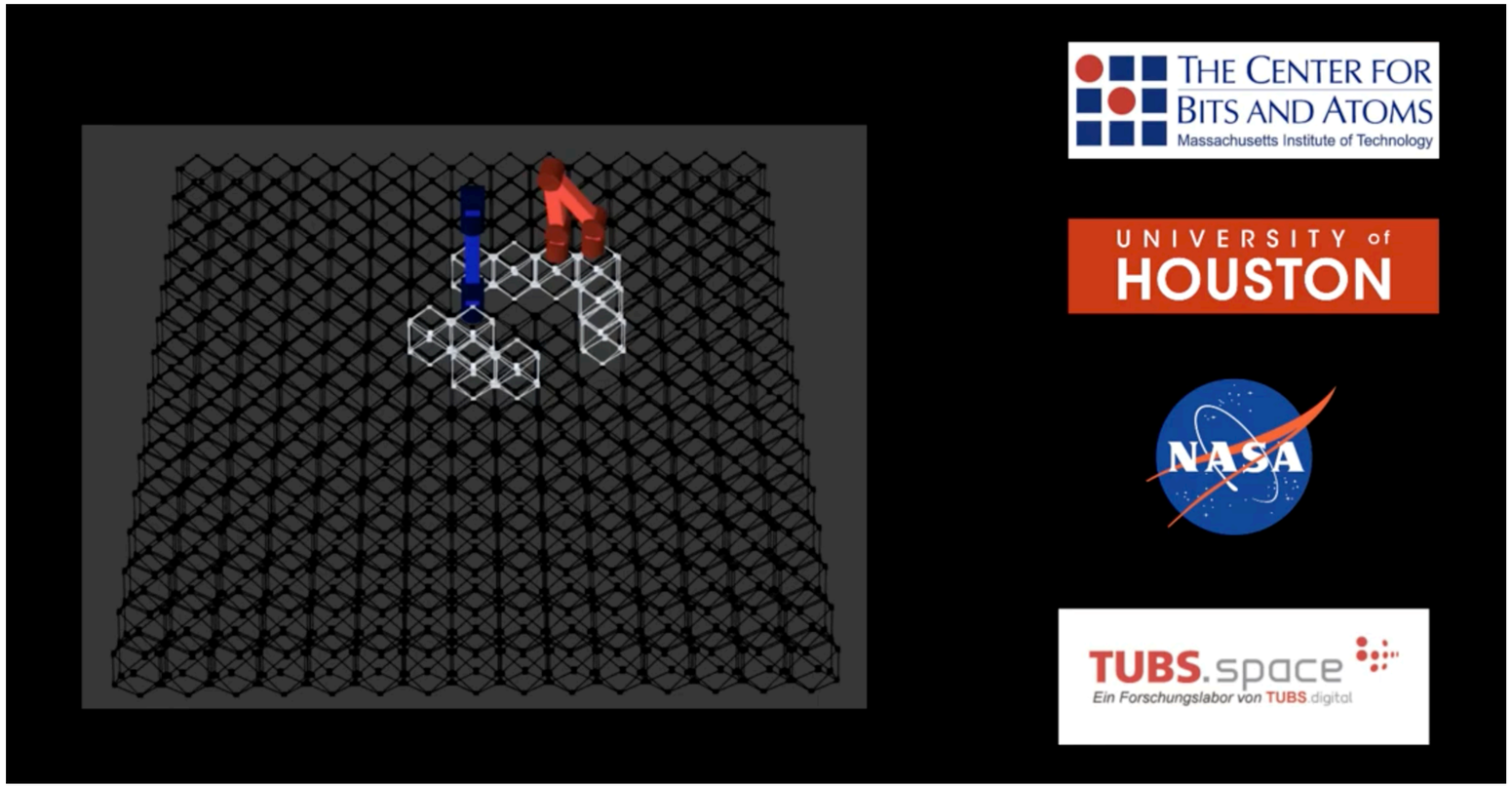
7.1 Motivation

Aufgabenstellung





Aufgabenstellung



THE CENTER FOR BITS AND ATOMS
Massachusetts Institute of Technology

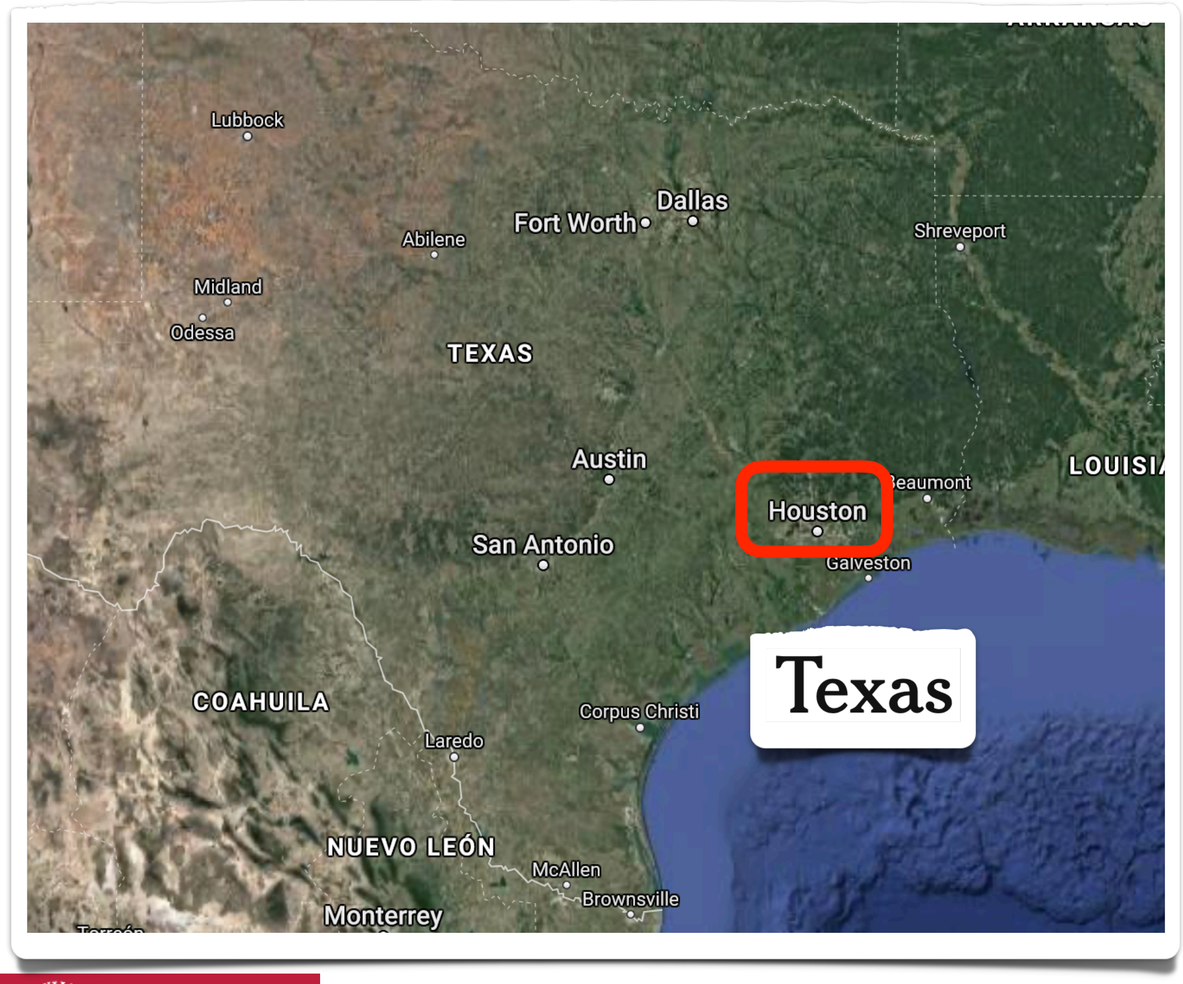
UNIVERSITY of HOUSTON

NASA

TUBS.space
Ein Forschungslabor von TUBS.digital

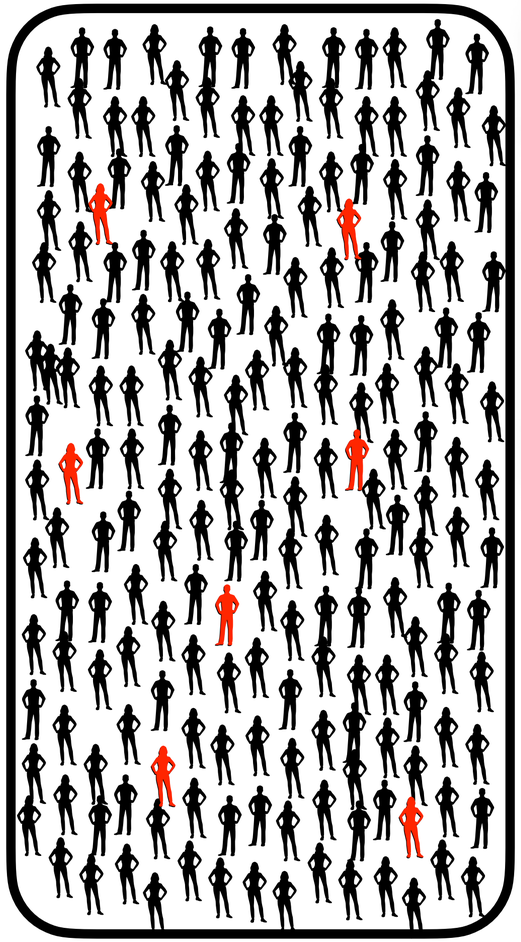


Aufgabenstellung





Aufgabenstellung



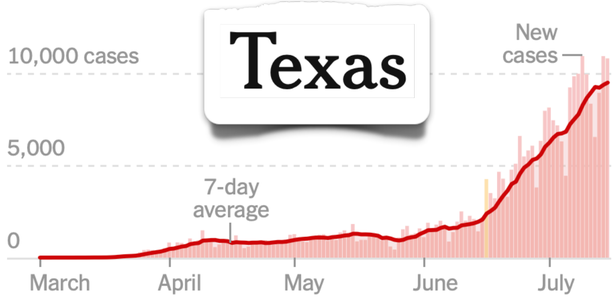
The New York Times

July 13, 2020

Bottleneck for U.S. Coronavirus Response: The Fax Machine

Before public health officials can manage the pandemic, they must deal with a broken data system that sends incomplete results in formats they can't easily use.

By The New York Times Updated July 16, 2020, 12:06 A.M. E.T.



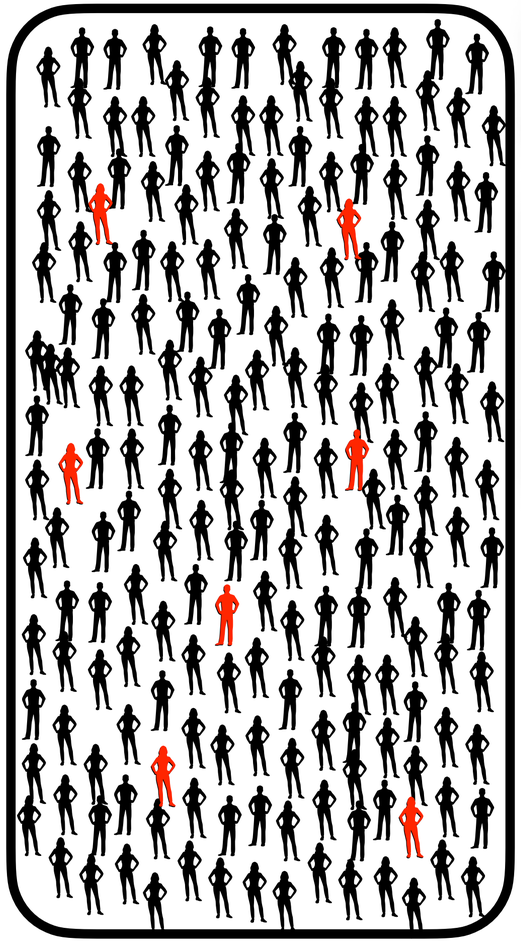
TOTAL CASES
295,423

DEATHS
3,582

Includes confirmed and probable cases where available



Aufgabenstellung



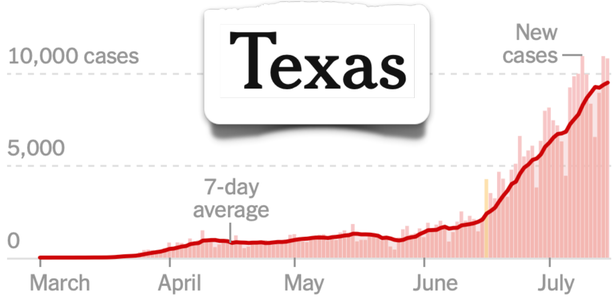
The New York Times

July 13, 2020

Bottleneck for U.S. Coronavirus Response: The Fax Machine

Before public health officials can manage the pandemic, they must deal with a broken data system that sends incomplete results in formats they can't easily use.

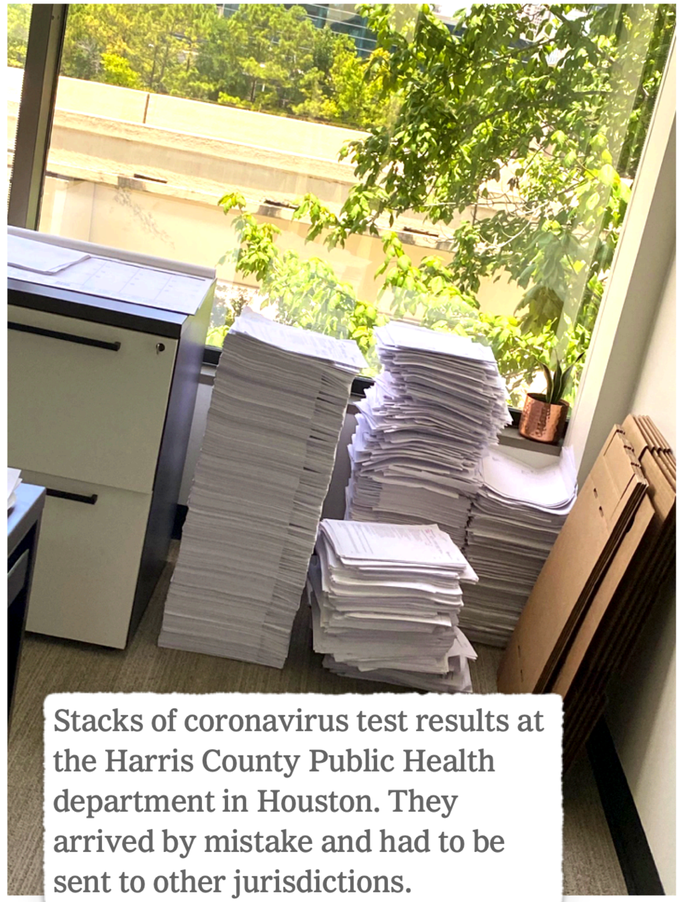
By The New York Times Updated July 16, 2020, 12:06 A.M. E.T.



TOTAL CASES
295,423

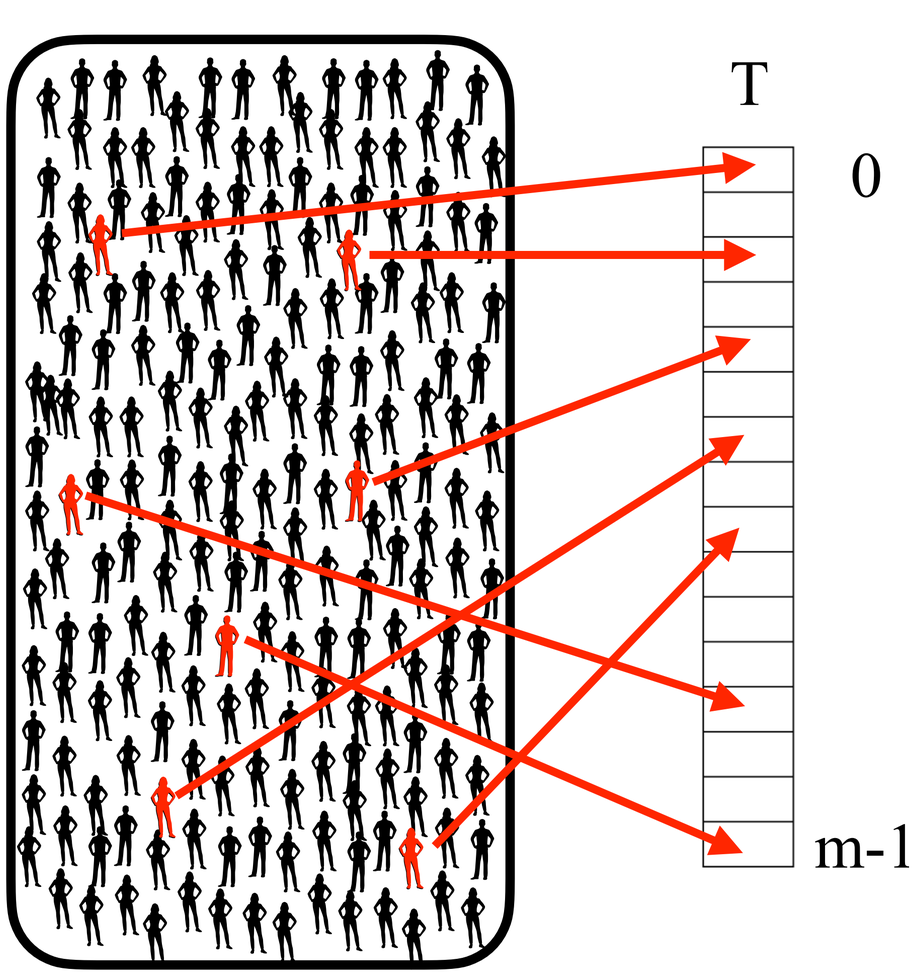
DEATHS
3,582

Includes confirmed and probable cases where available



Stacks of coronavirus test results at the Harris County Public Health department in Houston. They arrived by mistake and had to be sent to other jurisdictions.

Aufgabenstellung







Hashing

- Menge U potentieller Schlüssel sehr groß, aktuelle Schlüsselmenge S jeweils nur kleine Teilmenge des Universums (im allgemeinen S nicht bekannt)
- Idee: durch Berechnung feststellen, wo Datensatz mit Schlüssel x gespeichert
- Abspeicherung der Datensätze in einem Array T mit Indizes $\{0, 1, \ldots, m-1\}$: Hashtabelle
- Hashfunktion h liefert für jeden Schlüssel $x \in U$ eine Adresse in Hashtabelle, d.h. $h : U \to \{0, 1, \ldots, m-1\}$.

7.2 Aufgabenstellung

Aufgabenstellung

Aufgabe

Dynamische Verwaltung von Daten, wobei jeder Datensatz eindeutig durch einen Schlüssel charakterisiert ist

Viele Anwendungen benötigen nur einfache Daten-Zugriffsmechanismen (dictionary operations):

- Suche nach Datensatz bei gegebenem Schlüssel x
  search(x)
- Einfügen eines neuen Datensatzes d mit Schlüssel x
  insert(x, d) (abgekürzt insert(x))
- Entfernen eines Datensatzes bei gegebenem Schlüssel x
  delete(x)

Menge potentieller Schlüssel (Universum) kann sehr groß sein!

Alternativen

Sei $n := |S|$.

- Balancierte Suchbäume (AVL-Bäume, B-Bäume):
  dictionary operations haben Komplexität $O(\log n)$
- Hashing:
  für alle Operationen mittlere Komplexität $O(1)$

Belegungsfaktor

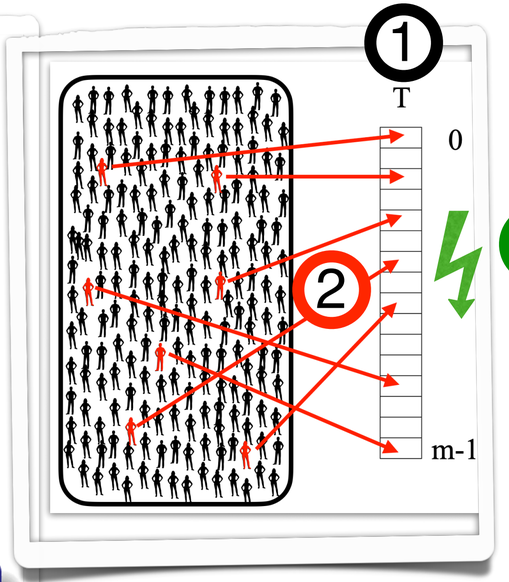
Quotient $\beta := n/m$ heißt Belegungsfaktor oder Auslastungsfaktor einer Hashtabelle der Größe m.

Mittlerer Aufwand für dictionary operations als Funktion in β abschätzbar
$\rightarrow$ Anzahl aktueller Schlüssel geht nur indirekt in Aufwand ein

Herausforderungen

Anzahl möglicher Schlüssel viel größer als Hashtabelle, also $|U| >> m$

- Hashfunktion muss verschiedene Schlüssel x_1 und x_2 auf gleiche Adresse abbilden.
- x_1 und x_2 beide in aktueller Schlüsselmenge
  $\rightarrow$ Adresskollision



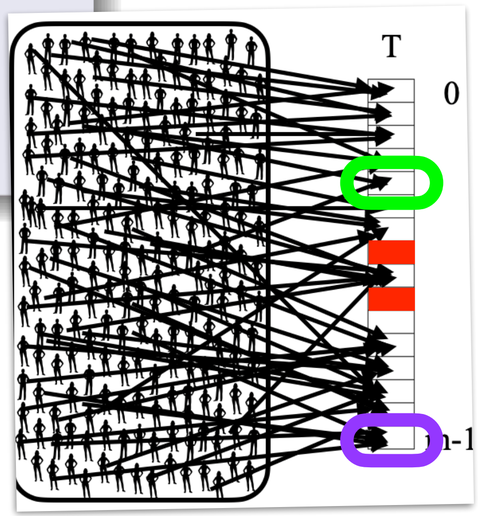
Hashverfahren gegeben durch:

1. eine Hashtabelle,
2. eine Hashfunktion, die Universum der möglichen Schlüssel auf Adressen einer Hashtabelle abbildet,
3. eine Strategie zur Auflösung möglicher Adresskollisionen.

Anforderungen

Gute Hashfunktionen sollten:

- surjektiv sein, d.h. den ganzen Wertebereich umfassen,
- die zu speichernden Schlüssel (möglichst) gleichmäßig verteilen, d.h. für alle Speicherplätze i und j sollte gelten $|h^{-1}(i)| \approx |h^{-1}(j)|$,
- effizient berechenbar sein.



7.3 Hashfunktionen

Divisions-Rest-Methode

Divisions-Rest-Methode

$$h(x) = x \bmod m := x - \left\lfloor \frac{x}{m} \right\rfloor \cdot m$$

Beispiel: m=11, S={49, 22, 6, 52, 76, 34, 13, 29}

Hashwerte:

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

Wahl von m

Problem: Daten oft nicht gleichverteilt!

Beispiel: Texte in Zahlen übertragen, oft viele Leerzeichen, bestimmte Wörter häufiger etc.

Wichtig: geeignete Wahl von m

- m Zweierpotenz: x mod m wählt nur die letzten $\log m$ Bits
- m Primzahl: x mod m *beeinflusst* alle Bits

7.4 Kollisionen

Auftreten von Kollisionen

Zur Erinnerung:

Im Allgemeinen unvermeidbar, dass Kollisionen auftreten, denn aus $N >> m$ folgt Existenz eines Speicherplatzes i mit $|h^{-1}(i)| \geq N/m$.

Frage:

Sei $n := |S|$. Wie wahrscheinlich sind Kollisionen bei $n << m$?

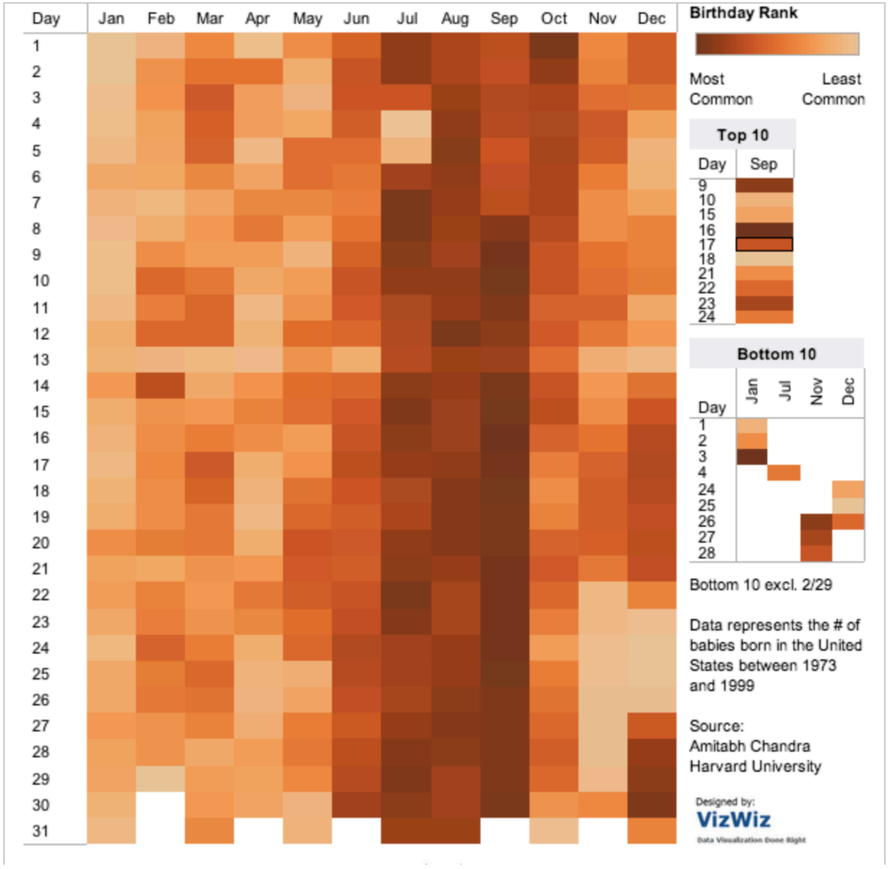
Geburtstagsparadoxon

Bei wie vielen (zufällig gewählten) Personen ist es *wahrscheinlich*, dass hiervon zwei am selben Datum (Tag und Monat) Geburtstag haben?

Geburtstagsparadoxon

Annahme:

- Daten unabhängig
- Prob($h(x) = j$)=$1/m$



Geburtstagsparadoxon

Annahme:

- Daten unabhängig
- Prob($h(x) = j$)=$1/m$

Prob(i-tes Datum kollidiert nicht mit den ersten $i-1$ Daten, wenn diese kollisionsfrei sind)= $\frac{m-(i-1)}{m}$

Intuition:

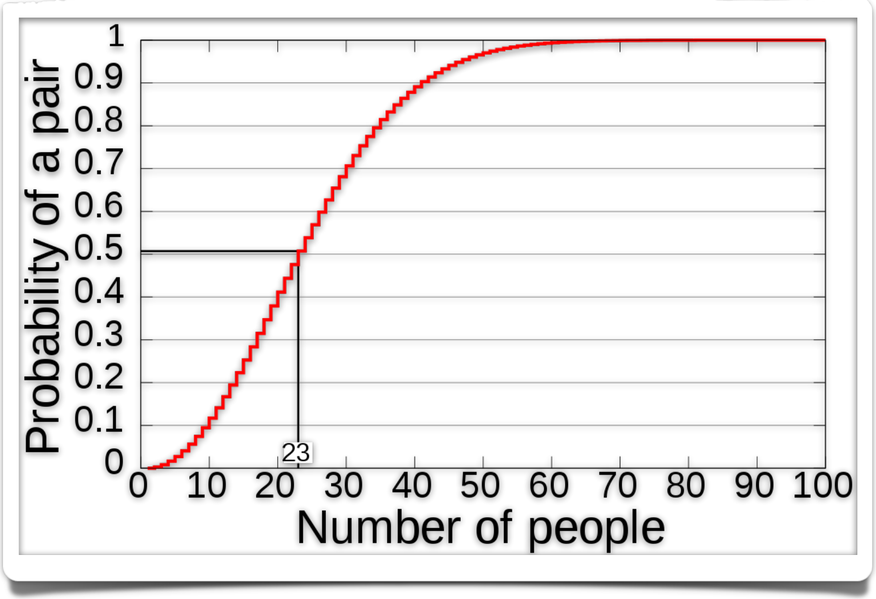
Egal welche Speicherplätze die ersten $i-1$ Daten belegen, $m-i+1$ der m Möglichkeiten sind *gut*.

Prob(n Daten kollisionsfrei) = $\frac{m-1}{m} \cdot \frac{m-2}{m} \ldots \frac{m-n+1}{m}$

Beispiel: $m = 365$

Prob(23 Daten kollisionsfrei) ≈ 0.49

Prob(50 Daten kollisionsfrei) ≈ 0.03



Geburtstagsparadoxon (verallgemeinert)

Prob($2m^{1/2}$ Daten kollisionsfrei) =

$$\underbrace{\frac{m-1}{m}}_{1} \cdots \underbrace{\frac{m-m^{1/2}}{m} \cdots \frac{m-2m^{1/2}+1}{m}}_{\left(\frac{m-m^{1/2}}{m}\right)^{m^{1/2}}}$$

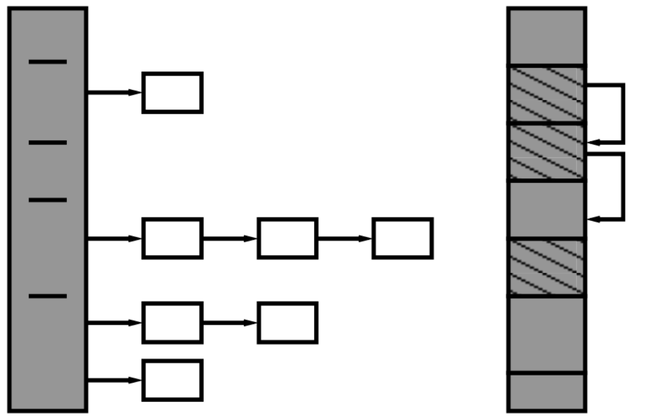
$$\leq 1 \cdot \left(\frac{m-m^{1/2}}{m}\right)^{m^{1/2}} = \left(1-\frac{1}{m^{1/2}}\right)^{m^{1/2}} \approx \frac{1}{e}$$

Hashing muss mit Kollisionen leben und benötigt Strategien zur Kollisionsbehandlung!

Kollisionbehandlung

Verschiedene Arten der Kollisionsbehandlung:

- mittels verketteter Listen (links)
- mittels offener Adressierung (rechts)



7.5 Verkettung von Überläufern

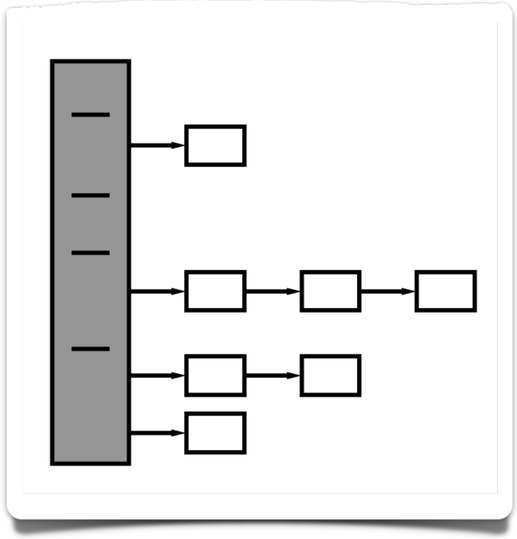
Verkettetung: Realisierung

Realisierung:

Jede Komponente der Hashtabelle enthält Zeiger auf paarweise disjunkte lineare Listen. Die i-te Liste $L(i)$ enthält alle Schlüssel $x \in S$ mit $h(x) = i$.

Vorteil: Alle Operationen werden unterstützt und $n > m$ ist möglich. (Für $n >> m$ jedoch Rehashing ratsam) Nachteil: Speicherplatzbedarf für Zeiger

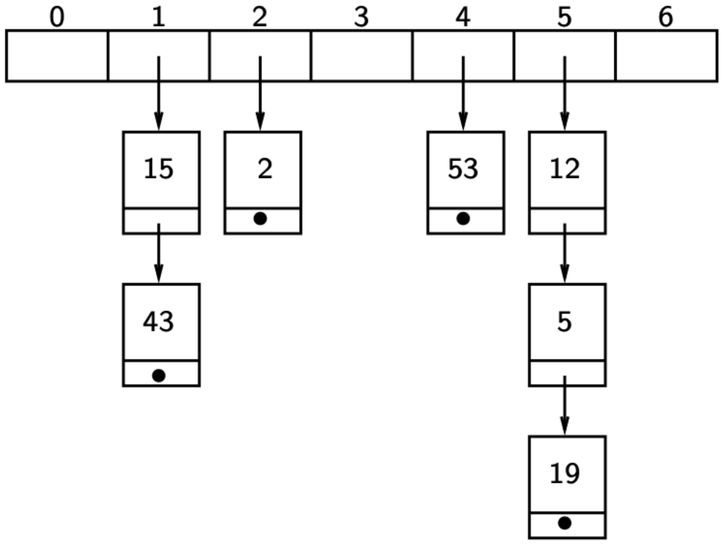
- search(x): Berechne $h(x)$ und suche in Liste $L(h(x))$.
- insert(x) (nach erfolgloser Suche): Berechne $h(x)$ und füge x in Liste $L(h(x))$ ein.
- delete(x) (nach erfolgreicher Suche): Berechne $h(x)$, suche x in Liste $L(h(x))$ und entferne x.



Beispiel

Beispiel: $m = 7$ und $h(x) = x \bmod m$

$$S = \{2, 12, 5, 15, 19, 43, 53\}$$



Analyse

Bei zufälligen Daten und ideal streuenden Hashfunktion gilt für

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

$\text{Prob}(X_{ij} = 1) = \frac{1}{m}$

$\rightarrow \text{E}(X_{ij}) = 1 \cdot \frac{1}{m} + 0 \cdot \frac{m-1}{m} = \frac{1}{m}$

$X_j = X_{1j} + \cdots + X_{nj}$ zählt Anzahl Daten in Liste $L(j)$.

$\text{E}(X_j) = \text{E}(X_{1j} + \cdots + X_{nj}) = \text{E}(X_{1j}) + \cdots + \text{E}(X_{nj}) = \frac{n}{m}$

Analyse (2)

- Erfolglose Suche in Liste $L(j)$:

  Inklusive nil-Zeiger durchschnittlich $1 + \frac{n}{m} = 1 + \beta$ Objekte betrachten
  Beispiel: Für $n \approx 0.95 \cdot m$ ist dies ≈ 1.95.

- Erfolgreiche Suche in Liste $L(j)$ der Länge ℓ:

  Jede Position in der Liste hat Wahrscheinlichkeit $1/\ell$, also $\frac{1}{\ell}(1 + 2 + \cdots + \ell) = \frac{\ell+1}{2}$.
  Durchschnittliche Listenlänge hier: $1 + \frac{n-1}{m}$
  (Liste enthält sicher das gesuchte Datum, und die anderen $n - 1$ Daten sind zufällig verteilt.)
  Also erwartete Suchdauer $\frac{1}{2}(1 + \frac{n-1}{m} + 1) = 1 + \frac{n-1}{2m} \approx 1 + \frac{\beta}{2}$
  Beispiel: Für $n \approx 0.95 \cdot m$ ist dies ≈ 1.475.

7.6 Offene Adressierung

Anforderungen

Zur Erinnerung:

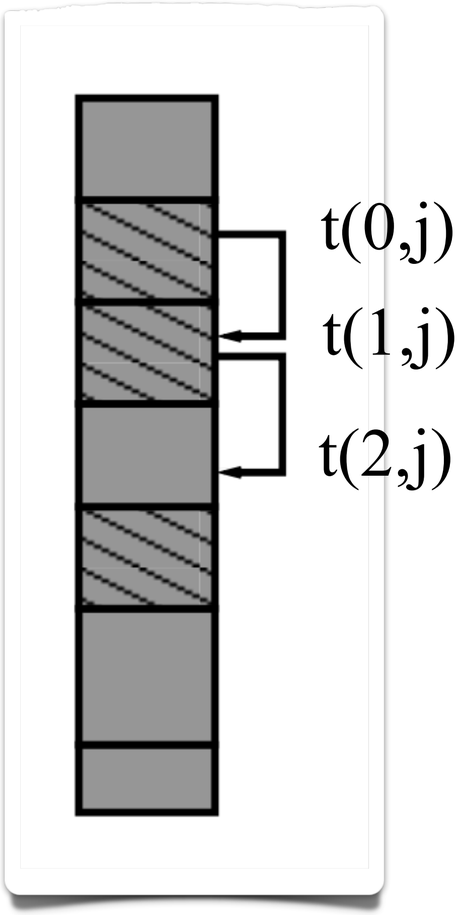
Im Kollisionsfall nach fester Regel alternativen freien Platz in Hashtabelle suchen (Sondierungsfolge).

Voraussetzung: Auswertung von h gilt als eine Operation.

$t(i,j)$:= Position des i-ten Versuchs zum Einfügen von Daten x mit $h(x) = j$

Anforderung an Funktion t:

- auch t in Zeit $O(1)$ berechenbar
- $t(0,j) = j$
- $t(\cdot,j) : \{0,\ldots,m-1\} \to \{0,\ldots,m-1\}$ bijektiv



Operationen

- search(x):
  - Berechne $j := h(x)$.
  - Suche x an den Positionen $t(0, j), \ldots, t(m-1, j)$.
  - Abbruch, wenn x gefunden oder freie Stelle entdeckt (kein Datum mit Schlüssel x).
- insert(x) nach erfolgloser Suche:
  Freien Platz finden (sonst Overflow) und
  x dort einfügen.
- delete(x) nach erfolgreicher Suche:
  Das Datum kann nicht einfach entfernt werden, da search frühzeitig Lücken finden würde und eine Suche fälschlicherweise als erfolglos abbrechen könnte.

Löschen?!

Problem: Datum kann bei Operation delete(x) nicht ohne weiteres gelöscht werden.

Ausweg: Speicherplatz/Position als besetzt, noch nie besetzt oder wieder frei markieren.

$\longrightarrow$ Suche wird nur an Positionen mit Markierung noch nie besetzt vorzeitig abgebrochen.

Problem: Im Laufe der Zeit keine Position mehr, die mit noch nie besetzt markiert ist.

$\longrightarrow$ Hashing wird ineffizient.

Offenes Hashing nur bei Anwendungen mit search und insert.

Sondieren

- Lineares Sondieren
- Quadratisches Sondieren
- Multiplikatives Sondieren
- Doppeltes Hashing

Hilfsmittel bei der Analyse: ideales Hashing

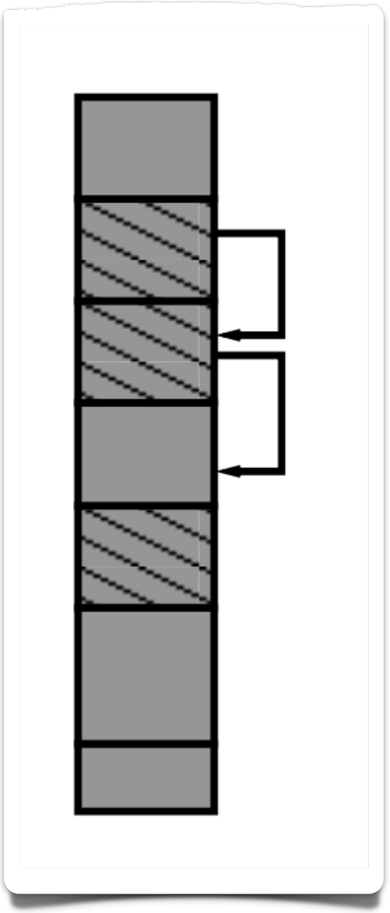
Lineares Sondieren

$t(i,j) := (i+j) \bmod m$

Beispiel: $m = 19$ und $j = h(x) = 7$

Sondierungsfolge:
7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 0, 1, 2, 3, 4, 5, 6

Problem: *Clusterbildung*
Tendenz, dass immer längere zusammenhängende, belegte Abschnitte in der Hashtabelle entstehen, sogenannte *Cluster*
$\rightarrow$ erhöhte Suchzeiten



Problem des Linearen Sondieren

Beispiel: $m = 19$, Positionen $2, 5, 6, 9, 10, 11, 12, 17$ belegt

$h(x)$:	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

Wunsch:

Zu jedem Zeitpunkt haben alle Positionen die gleiche Wahrscheinlichkeit, besetzt zu werden.

Beobachtung:

Das geht numerisch nicht genau, im Beispiel 11 freie Plätze, 19 mögliche Hashwerte, also alle Wahrscheinlichkeiten $k/19$, nicht $k/11$.

Modell des idealen Hashings:

Alle $\binom{m}{n}$ Möglichkeiten, die n besetzten Plätze für m Schlüssel auszuwählen, haben die gleiche Wahrscheinlichkeit.

Lineares Sondieren ist weit vom idealen Hashing entfernt.

Quadratisches Sondieren

$t(i,j) := j + (-1)^{i+1} \cdot \lfloor \frac{i+1}{2} \rfloor^2 \bmod m$

Sondierungsfolge:

$j,\ j+1^2,\ j-1^2,\ j+2^2,\ j-2^2, \ldots,\ j+(\frac{m-1}{2})^2,\ j-(\frac{m-1}{2})^2$

Beispiel: $m = 19$ und $j = h(x) = 7$

Sondierungsfolge:

$$7,\quad 8,\quad 6,\quad 11,\quad 3,\quad 16,\quad \underset{=17,}{-2}\quad \underset{=4,}{23}\quad \underset{=10,}{-9}\quad \underset{=13,}{32}\quad \underset{=1,}{-18}$$

$$\underset{=5,}{43}\quad \underset{=9,}{-29}\quad \underset{=18,}{56}\quad \underset{=15,}{-42}\quad \underset{=14,}{71}\quad \underset{=0,}{-57}\quad \underset{=12,}{88}\quad \underset{=2}{-74}$$

Quadratisches Sondieren (2)

Frage: Ist $t(\cdot, j)$ für alle j und m bijektiv?

Nein, aber immer wenn $m \equiv 3 \mod 4$ und m eine Primzahl ist

(Beweis: Zahlentheorie)

Besser als lineares Sondieren, aber für großes m sind die ersten Werte noch *nah* an j

Multiplikatives Sondieren

Hier: $h(x) = x \mod (m-1) + 1$ und damit in $\{1, \ldots, m-1\}$

$t(i,j) := i \cdot j \mod m$, $1 \leq i \leq m-1$

Hashwerte $1, \ldots, m-1$

Beispiel: $m = 19$ und $j = h(x) = 7$

Sondierungsfolge:

$i \cdot j$		7	14	21	28	35	42	49	56	63
$i \cdot j$	mod 19	7	14	2	9	16	4	11	18	6

$i \cdot j$		70	77	84	91	98	105	112	119	126
$i \cdot j$	mod 19	13	1	8	15	3	10	17	5	12

Multiplikatives Sondieren (2)

Frage: Ist $t(\cdot, j)$ für alle j und m bijektiv?

Nein, aber immer wenn m Primzahl und $j \neq 0$

Beweis: Durch Widerspruch!

Falls nicht, gibt es $1 \leq i_1 < i_2 \leq m-1$ mit

$$
\begin{aligned}
& i_1 \cdot j \equiv i_2 \cdot j \mod m \\
\Rightarrow \quad & j \cdot (i_2 - i_1) \equiv 0 \mod m \\
\Rightarrow \quad & j \cdot (i_2 - i_1) \text{ ist Vielfaches von } m \\
\Rightarrow \quad & \text{Primfaktorzerlegung von } j \cdot (i_2 - i_1) \text{ muss } m \text{ enthalten}
\end{aligned}
$$

Widerspruch zu $1 \leq j \leq m-1$, $1 \leq i_2 - i_1 \leq m-1$

Doppeltes Hashing

Sei $h_1(x) \equiv x \mod m$ und $h_2(x) \equiv x \mod (m-2)+1$.

i-te Position für x: $h_1(x) + i \cdot h_2(x) \mod m$, $1 \leq i \leq m-1$

Beispiel: $m = 19$ und $x = 47$

$h_1(47) \equiv 47 \mod 19 = 9$ und $h_2(47) \equiv 47 \mod 17 + 1 = 14$

Sondierungsfolge:

9, 4, 18, 13, 8, 3, 17, 12, 7, 2, 16, 11, 6, 1, 15, 10, 5, 0, 14

Doppeltes Hashing (2)

Beobachtung:

$i \cdot h_2(x)$ 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 Anfang zufällig verschoben.

Doppeltes Hashing kommt dem idealen Hashing am nächsten.

Universelles Hashing

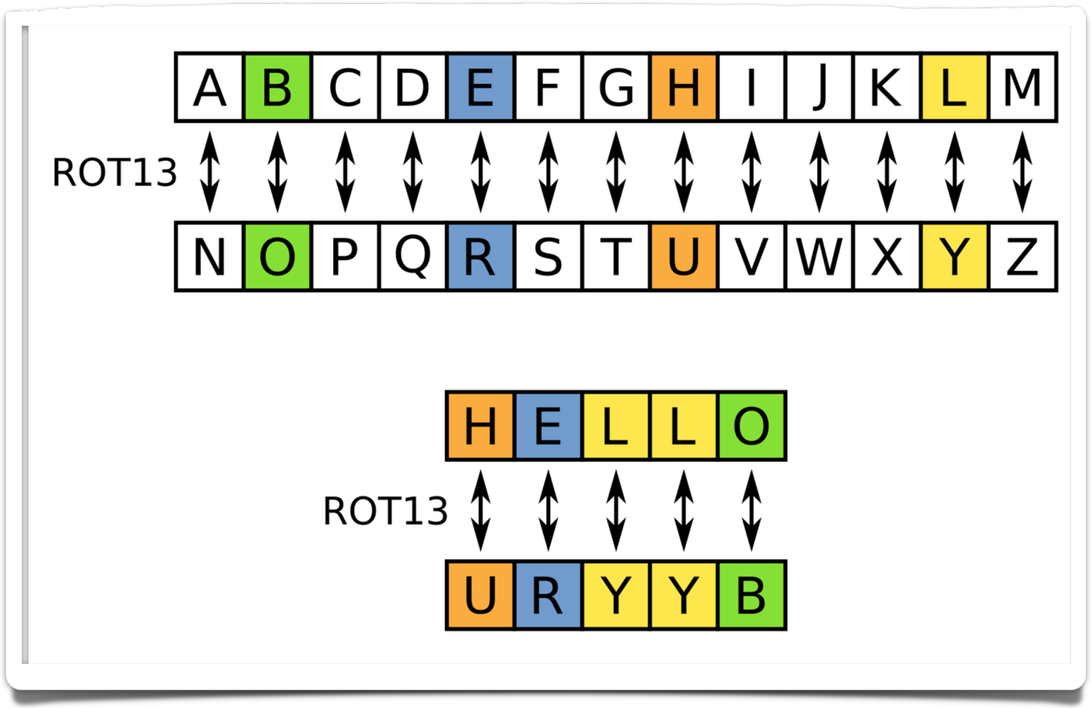
Zusammenfassung

- Auch die beste Hashfunktion kann Kollisionen nicht ganz vermeiden, deshalb sind Hashverfahren im *Worst Case* ineffiziente Realisierungen der Operationen search, insert, delete.
- Im Durchschnitt sind sie jedoch weitaus effizienter als Verfahren, die auf Schlüsselvergleichen basieren.
- Die Anzahl benötigter Schritte zum Suchen, Einfügen und Entfernen hängt (im Durchschnitt) im wesentlichen vom Belegungsfaktor, d.h. dem Verhältnis von Anzahl aktueller Schlüssel zur Größe der Hashtabelle, ab.

7.7 Kryptographie

Veränderte Ziele

- Text codieren:
  1. Berechtigter Empfänger soll Nachricht entschlüsseln können.
  2. Unberechtigter Empfänger soll Nachricht nicht entschlüsseln können.

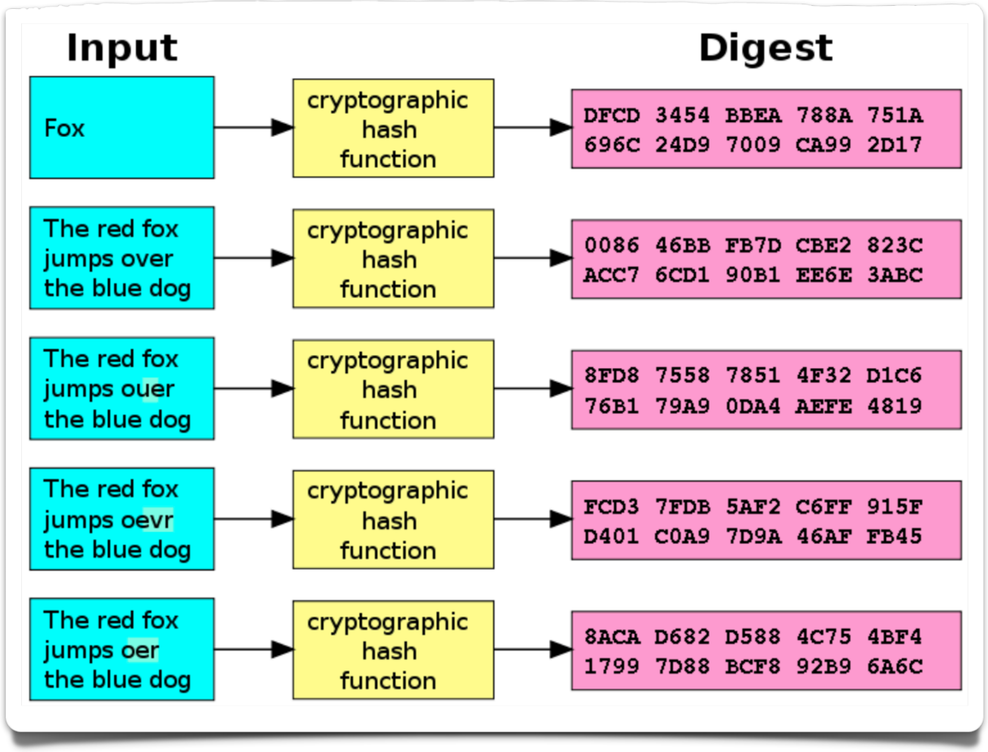






Veränderte Ziele

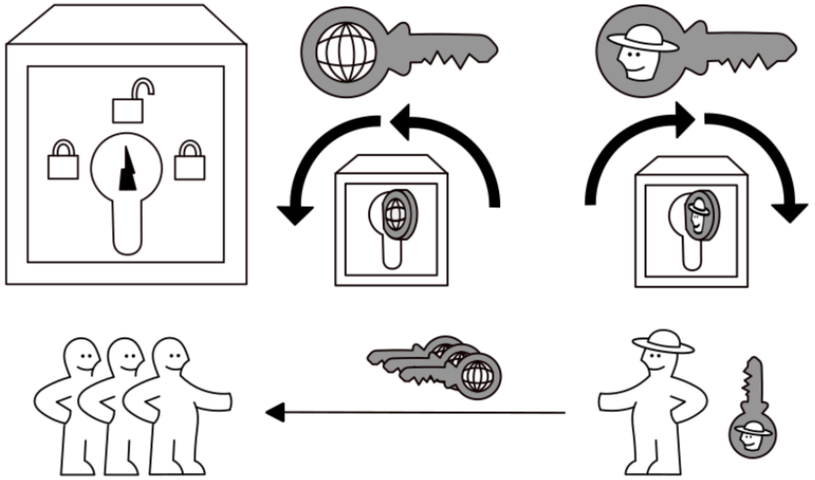
- Deterministisch, d.h. dieselbe Nachricht ergibt immer den selben Wert.
- Hashwert lässt sich schnell berechnen.
- SEHR schwer eine Nachricht mit demselben Hashwert zu generieren.
- SEHR schwer zwei Nachrichten mit demselben Hashwert zu finden.
- Ein kleine Änderung in der Nachricht ändert das Ergebnis so stark, dass man aus den Hashwerten nicht auf ähnliche Nachrichten schließen kann.

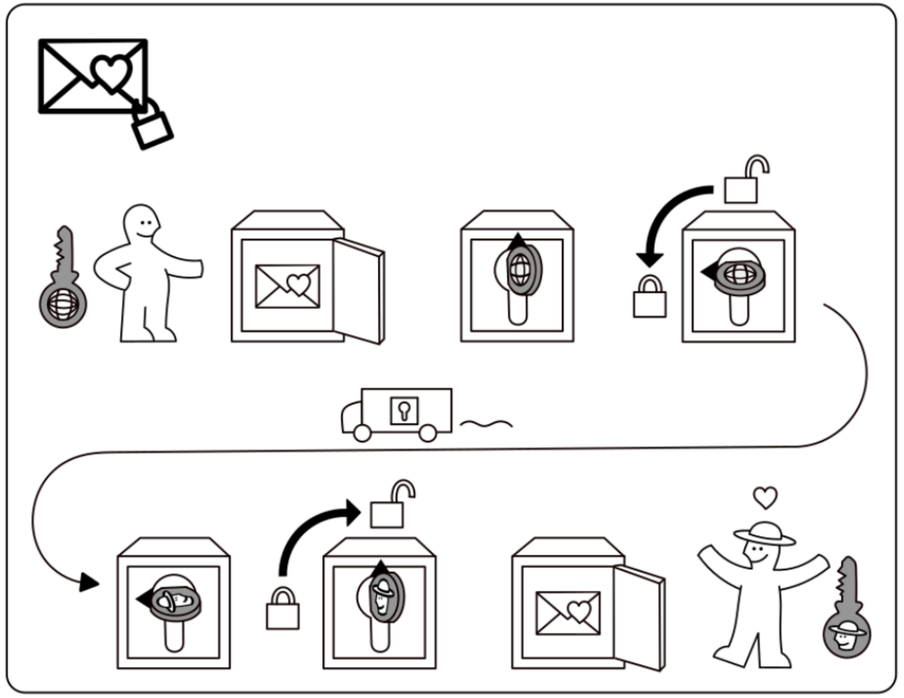


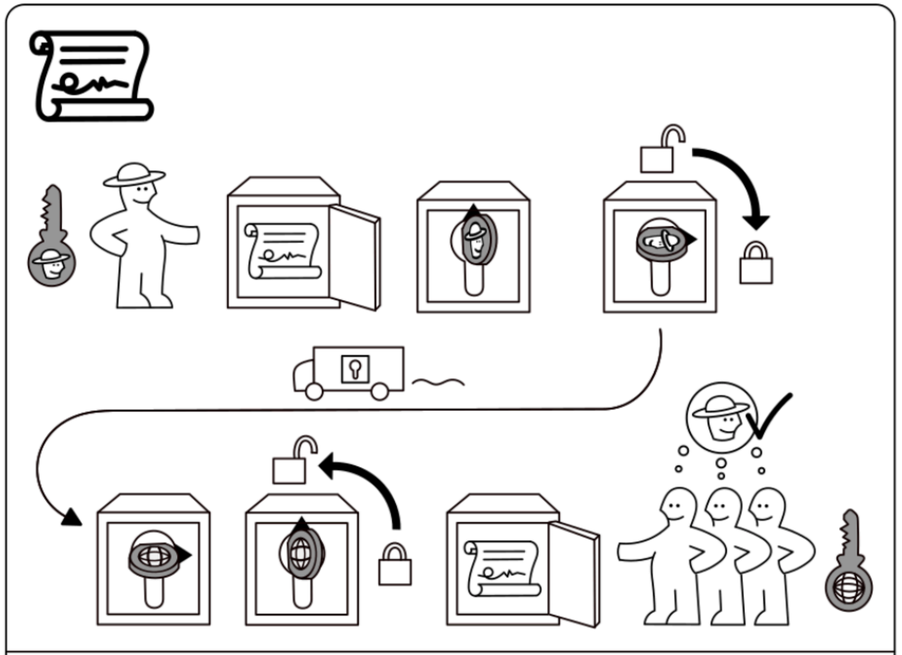
PUBLIK KEY KRYPTO

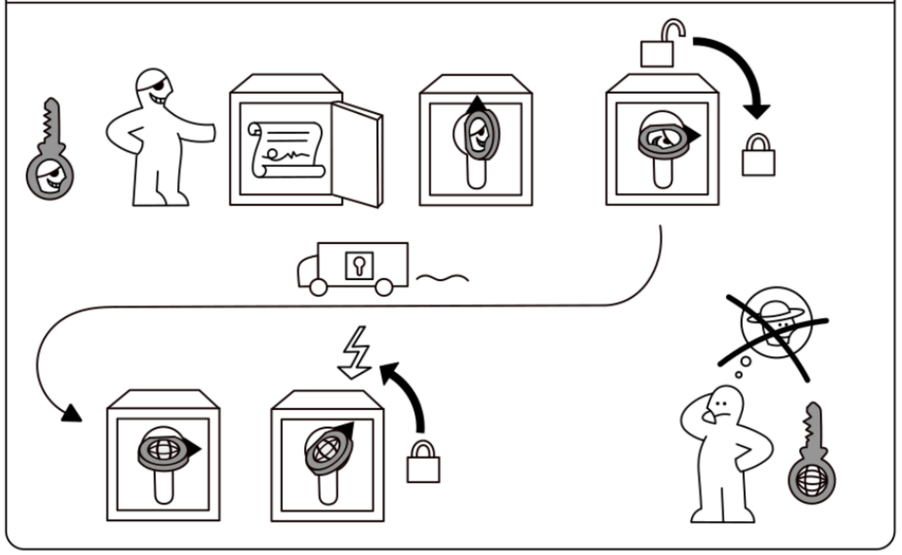
idea-instructions.com/public-key/
v1.2, CC by-nc-sa 4.0

IDEA









Vielen Dank!

s.fekete@tu-bs.de