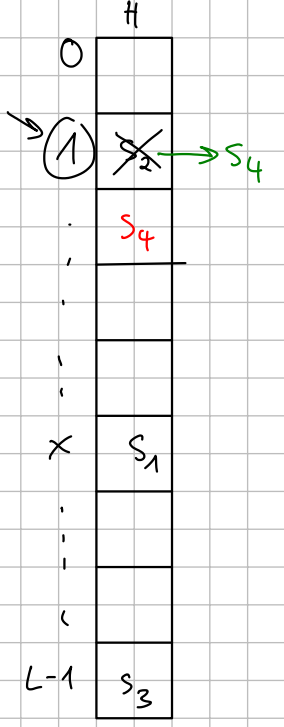
$h(s_1) = x$

$h(s_7) = 2$

$\underbrace{h(s_4) = h(s_2)}_{\text{Kollision}}$

$s_i \neq s_j$, $h(s_i) = h(s_j)$



Hashfunktionen sollten schnell (in $O(1)$) berechenbar sein

Wenn Universum $U = \{0, \dots, 2^w - 1\}$

Dann ist $h : U \to \{0, \dots, L-1\}$

$w :=$ Wortlänge

L häufig in der Form 2^d

Grundidee: s ist an der Stelle $H[h(s)]$ gespeichert

Behandlung von Kollisionen:

1. Verkettung: $H[i]$ ist verkettete Liste aller Schlüssel s mit $h(s) = i$
2. „offene Adressierung" (u.A. lineares Sondieren): Es wird nach einer Ersatzpos. gesucht nach der nächsten, also $h(i)+1$, $h(i)+2$ ...
3. „Kuckuckshashing"

(Auch bei Kuckuckshashing) Wollen Kollisionen so weit wie möglich vermeiden

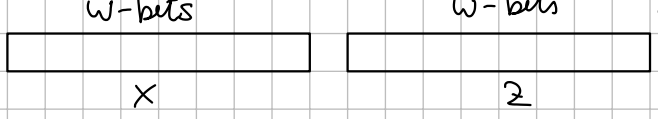
$\Rightarrow$ Hashfunktion gut wählen

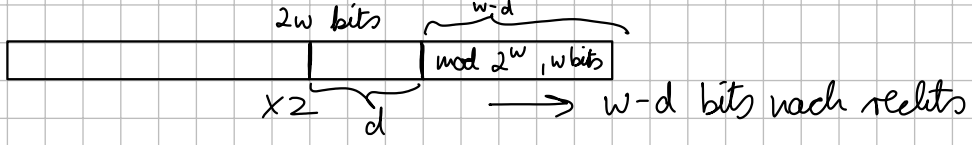
Multiplikative Hashfunktion

Schlüssel $x \in \{0, \dots, 2^w - 1\}$

zufälliger Parameter $z \in \{1, 3, 5, \dots, 2^w - 1\}$

$h(x) = \lfloor (x \cdot z) \bmod 2^w / 2^{w-d} \rfloor = \underbrace{(x \cdot z) \bmod 2^w}_{\text{AND } (2^w - 1)} >> (w-d)$





$$x \neq x' \Rightarrow Pr_z[h(x) = h(x')] \leq \frac{2}{2^d} = \frac{2}{L}$$

Anwendung dieser Hashfkt. bei Verkettung:

n Schlüssel bereits in H gespeichert $\quad n < L$

Belegungsfaktor $\alpha := \frac{n}{L}$ normalerweise will man $\alpha < 1$, aber für Verkettung nicht notwendig, bei „offener Adressierung" und Kuckuckshashing schon

Satz. Bei einer Hashtabelle mit Verkettung ist

$\mathbb{E}[$ Länge einer Liste $H[h(x)]] \leq 1 + 2\alpha$

(x beliebig aber fest)

Beweis: $S = \{x_1, \dots, x_n\}$ gespeicherte Schlüssel

$$\mathbb{E}[\text{Länge von } H[h(x)]] = \underbrace{1}_{\text{falls } x \in S} + \sum_{y \in S \setminus \{x\}} \underbrace{\Pr[h(y) = h(x)]}_{\leq \frac{2}{L}} = 1 + \frac{2(n-1)}{L} \leq 1 + \frac{2n}{L} = 1 + 2\alpha$$

Erwartete Laufzeit für Einfügen, Löschen, Suchen ist $O(1+\alpha)$

zurück zum linearen Sondieren: Beim Löschen darf man nicht einfach Einträge entfernen, sonst bricht die Suche zusammen. Lösung: als gelöscht markieren.

Erwartete Laufzeit für erfolglose Suche $O\left(\frac{1}{(1-\alpha)^2}\right)$

für erfolgreiche Suche $O\left(\frac{1}{1-\alpha}\right)$

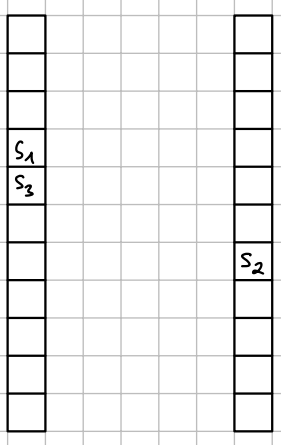
$\alpha < 0.7$

Kuckuckshashing

2 Hashfunktionen h_1 und h_2

Schlüssel x ist entweder in $H_1[h_1(x)]$ oder $H_2[h_2(x)]$ (oder gar nicht) enthalten

Suchen in $O(1)$ worst-case und Löschen



$h_1(s_1) = 3$

$h_2(s_1) = 6$

$h_1(s_2) = 3$

$h_2(s_2) = 6$

$h_1(s_3) = 4$

$h_2(s_3) = 6$

$h_1(s_4) = 4$

$h_2(s_4) = 6$

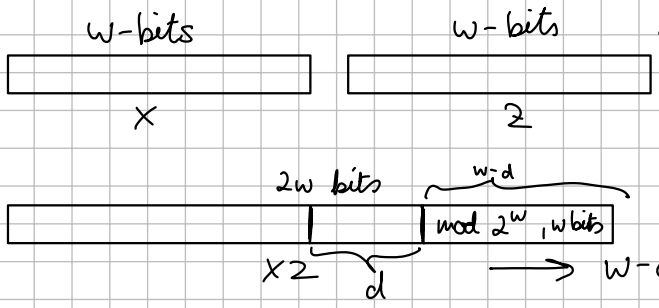
Einfügen: Kette von Austauschen, im Erwartungswert $O(1)$.

Im schlimmsten Fall (Pr. $\frac{1}{n}$) muss die gesamte Tabelle mit neuen h_1, h_2 aufgebaut werden ($O(n)$)

Schlüssel $x \in \{0, \dots, 2^w - 1\}$

zufälliger Parameter $z \in \{1, 3, 5, \dots, 2^w - 1\}$

$h(x) = \lfloor (xz) \bmod 2^w / 2^{w-d} \rfloor = (xz) \underbrace{\bmod 2^w}_{\text{AND } (2^w-1)} >> (w-d)$



$$x \neq x' \Rightarrow Pr_z[h(x) = h(x')] \leq \frac{2}{2^d} = \frac{2}{L}$$

Beweis:

- Lemma: sei $U = \{1, \dots, 2^w - 1\}$ und $z \in U$ ↓ dann ist $f: U \to U$ mit $f(z) = (z \cdot x) \bmod 2^w$ bijektiv

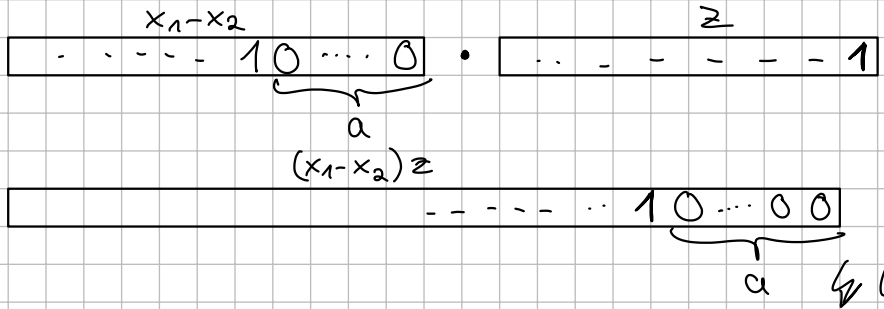
Beweis von Lemma: reicht zu zeigen, dass f injektiv ist

Durch Widerspruch: x_1 und x_2 mit $(x_1 z) \bmod 2^w = (x_2 z) \bmod 2^w$

$$(x_1 z \equiv x_2 z) \pmod{2^w}$$

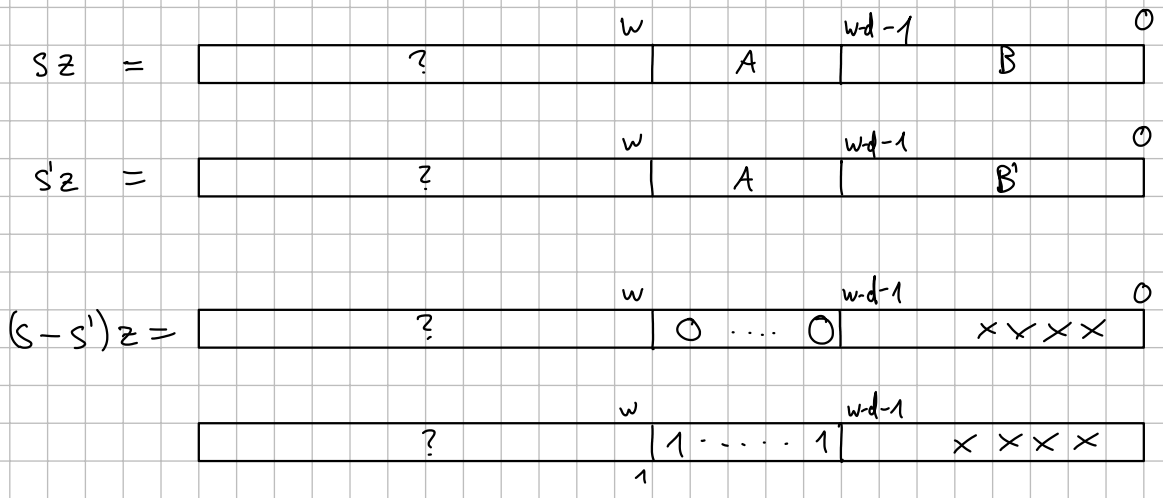
$$\Rightarrow ((x_1 - x_2) z \equiv 0) \pmod{2^w}$$

$\Rightarrow (x_1 - x_2)$ muss gerade sein

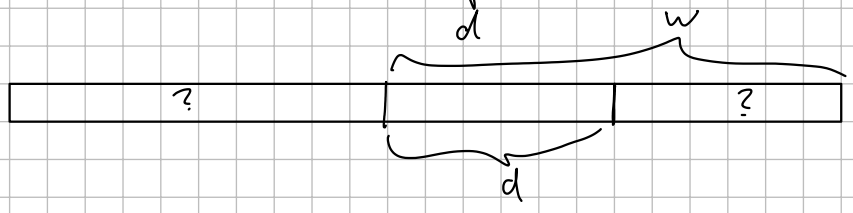


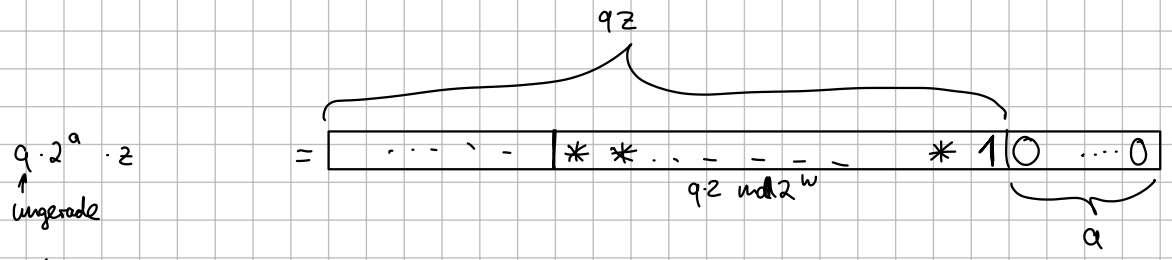
für ein beliebiges, aber festes x, und unter Gleichverteilung von z ist das äquivalent zur Wahl von $w-1$ bits zufällig

$$Pr_z[h(s) = h(s')] \text{ für } s > s' \leq \frac{2}{2^d} = \frac{2}{L}$$



$(s-s')z \bmod 2^w >> (w-d) = 0$ oder $\underbrace{1 \dots 1}_{d} = 2^d - 1$





$w - a < d$ (die d interessanten bits liegen zwischen den a '0' und dem $(qz) \bmod 2^w$)

: $Pr[\text{die d interessanten bits sin 0...0 oder 1...1}] = 0$

$w - a = d$ (die d interessanten bits fangen bei der '1' an)

: $Pr[\dots] = Pr[\dots 1 \dots 1] = \frac{1}{2^{d-1}} = \frac{2}{2^d}$

$w - a > d$: (— " — fangen nach der '1' an)

$Pr[\dots] = \frac{2}{2^d}$

Andere übliche Wahl für multiplikative Hashfunktion

Geg.: Primzahl p. Wähle z zufällig aus $\{0, \dots p-1\}$

$h(x) = (x \cdot z) \bmod p = ((x \bmod p)\, z) \bmod p$

Satz: Sei $s \neq s'$ beliebig aber fest, dann gilt $Pr_z[h(s) = h(s')] = \frac{1}{p}$

Beweis "analog" zu davor mit Lemma: $U = \{0, \dots, p-1\}$, $0 \neq q \in U$

$f: U \to U$ mit $f(z) = (z \cdot q) \bmod p$

dann ist f bijektiv

Beweis vom Lemma: jedes $x \in \{0, \dots, p-1\}$ besitzt ein eindeutiges y mit $(x\, y \equiv 1) \bmod p$

$$\left.\begin{array}{l} (x \cdot z \equiv y) \pmod p \\ (\underbrace{y \cdot z^{-1}}_{x} \cdot z \equiv y) \pmod p \end{array}\right\}$$

ggT von Euklid: $x, m \in \mathbb{Z}$, Satz: Wenn $ggT(x, m) = 1$, dann gibt es ein $y \in \mathbb{Z}$ mit $x \cdot y = 1$ (bzw $y = x^{-1}$)

$ggT(x, m) = s, t$, sodass $s \cdot x + t \cdot m = ggT(x, m)$

$ggT(z, p) = s, t$, sodass $s \cdot z + t \cdot p = 1$

$(s \cdot z \equiv 1 - t \cdot p) \bmod p$

$(s \cdot z \equiv 1) \bmod p$

ggT(u,v):

$a,b \leftarrow u,v$

$x_a, y_a, x_b, y_b \leftarrow 1,0,0,1$

while $b > 0$:

$q \leftarrow \lfloor \frac{a}{b} \rfloor$

$r \leftarrow a - q \cdot b$

$x_r, y_r \leftarrow x_a - q \cdot x_b, y_a - q y_b$

$a,b \leftarrow b,r$

$x_a, y_a, x_b, y_b \leftarrow x_b, y_b, x_r, y_r$

return a, x_a, y_a

Invarianten

$a = x_a \cdot u + y_a \cdot v$

$b = x_b \cdot u + y_b \cdot v$

$r = x_r \cdot u + y_r \cdot v$

Hashcodes für längere Objekte

$(s_1, s_2, \dots, s_k)$ k Stücke @ d Bits, $0 \le s_i < 2^d = L$

$z_1, \dots, z_k$ k unabhängig, gleichverteilt Multiplikatoren @ d Bits

z ungerade mit 2d Bits, gleichverteilt

Hashfunktion: 1. $t = \left(\sum_{i=1}^{k} s_i z_i \right) \bmod 2^{2d}$

2. $h(s_1, \dots, s_k) = u = \lfloor ((tz) \bmod 2^{2d}) / 2^d \rfloor$

Satz: $(s_1, \dots, s_k) \neq (s'_1, \dots, s'_k)$ beliebig aber fest

$Pr[u = u'] \le \frac{3}{2^d}$

Beweis: o.B.d.A.: $s'_1 > s_1$

① $Pr[t = t'] = Pr[t - t' \equiv 0 \pmod{2^{2d}}] = Pr[\underbrace{(s'_1 - s_1)}_{>0} z_1 + \sum_{i=2}^{k} (s'_i - s_i) z_i \equiv 0 \pmod{2^{2d}}]$

$A := \left(- \sum_{i=2}^{k} (s'_i - s_i) z_i \right) \pmod{2^{2d}}$

$= Pr[(s'_1 - s_1) z_1 = A] = Pr\left[z_1 = \frac{A}{s'_1 - s_1}\right] \le \frac{1}{2^d}$

② $Pr[u = u' \mid t \neq t'] \le \frac{2}{2^d}$ (nach früherer Analyse)

$Pr[u = u'] = Pr[t = t' \vee (t \neq t' \wedge u = u')] \le Pr[① \vee ②] \le Pr[①] + Pr[②] = \frac{3}{2^d}$

Use-cases

- Java: hashCode() vom Typ int (32 Bits)
  hängt mit equals() zusammen
  Vertrag: a.equals(b) ⇒ a.hashCode() == b.hashCode()

- Knuth-Morris-Pratt

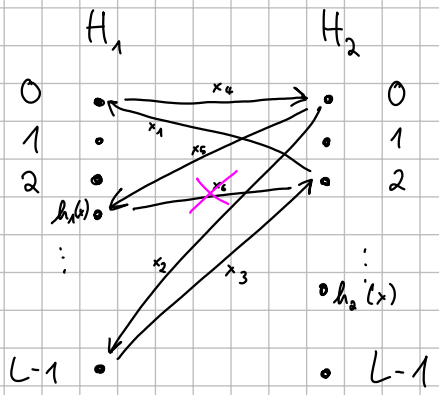
- Kryptographische Hashfunktionen
  Aus y lässt sich x mit $h(x) = y$ nicht effizient berechnen

SHA XXX
256
512
Linux: shaXXXsum

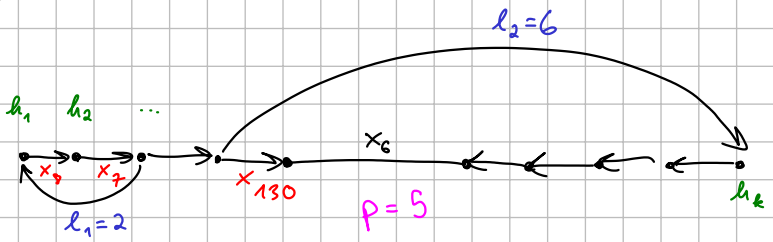
Kuckuckshashing

	h_1	h_2
x_1		
⋮	⋮	⋮
x_n		

$(L^2)^n = L^{2n}$



„Hindernis"



insgesamt $k+1$ Kanten

$l_1, l_2 \leq k$
$p \leq k$
$x \leq n^k$
$h \leq L^k$
Faktor 2 für $H_1, H_2, \ldots$
$H_2, H_1, \ldots$

$2 \cdot k^3 \cdot n^k \cdot L^k$ Möglichkeiten für ein Hindernis auf k Knoten (mit bereits $n-1$ eingefügten Schlüsseln)

$(L^2)^{n-(k+1)}$ restliche Einträge $= L^{2n-2k-2}$ Annahme $\alpha = \frac{n}{2L} \leq \frac{1}{4}$

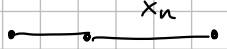
$\Leftrightarrow n \leq \frac{L}{2}$

$$\Pr[\exists \text{ Hindernis auf } k \text{ Knoten} \dots] \leq \frac{2 \cdot k^3 \cdot n^k \cdot L^k \cdot L^{2n-2k-2}}{L^{2n}} \leq \frac{2k^3 L^k L^k L^{2n-2k-2}}{2^k L^{2n}} = \frac{2k^3}{L^2 2^k}$$

$$\Pr[\text{Einfügen scheitert}] \leq \sum_{k \geq 2} \frac{2k^3}{L^2 2^k} = \frac{1}{L^2} \underbrace{\sum_{k \geq 2} \frac{2k^3}{2^k}}_{O(1)} = O\left(\frac{1}{L^2}\right) \in O\left(\frac{1}{n^2}\right)$$

⇒ Folge von n EinfügeOPs hat eine Wahrscheinlichkeit von $O(\frac{1}{n})$, dass irgendeine OP scheitert.

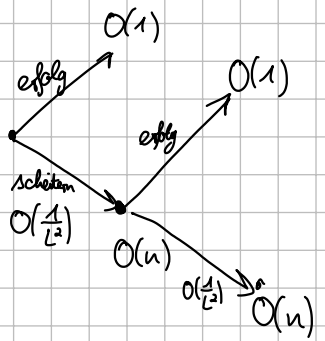
Erfolgreiches Einfügen:



$\Pr[\text{nicht leer}] \leq \frac{n-1}{L} \leq \frac{1}{2}$

⇒ Läuft auf Bernoulli Experiment aus (mit Erfolgs. Wsk $> \frac{1}{2}$)

⇒ Im Erwartungswert nur 2 Schritte



$O(1) + O(\frac{1}{L^2}) \cdot O(L) \cdot [$

$O(1) + O(\frac{1}{L^2}) \cdot O(L) \cdot [$

$\vdots$

$O(1) + O(\frac{1}{L}) + O(\frac{1}{L^2}) + O(\frac{1}{L^3}) + \dots$

$\Rightarrow O(1)$

<Implementierung von Einfügen

1. markieren
2. ohne markierungen, vgl. Aufgabe 95 (c)-(e)
3. ohne markierungen, draufgängerisch
   (Wir können nicht mehr als n Kanten traversieren, einfach Zähler einbauen)

Einfügen (x)

```
for i=1 ... n
    x ↔ H_1[h_1(x)]
    if x = null, return
    x ↔ H_2[h_2(x)]
    if x = null, return
return "ERFOLGLOS"
```