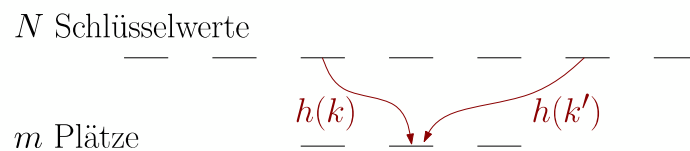


# Hashing

- ▶  $U = \{0, 1, \dots, N - 1\}$  mit  $N \in \mathbb{N}$  ist o.B.d.A. das Universum der möglichen Schlüsselwerte
- ▶ Speichern des Elements  $x$  in Feld der Größe  $N$  an Stelle  $\text{Schlüssel}(x)$ 
  - ▶ ist effizient (Einfügen, Löschen, Suchen in  $O(1)$ )
  - ▶ benötigt aber oft zu viel Speicherplatz
- ▶ Idee von Hashing
  - ▶ Definiere *Hashfunktion*  $h : U \rightarrow \{0, 1, \dots, t - 1\}$  mit  $t < N \in \mathbb{N}$
  - ▶ Annahme: Hashfunktion kann in  $O(1)$  evaluiert werden
  - ▶ *Hashtabelle* = Feld  $T$  der Größe  $t$
  - ▶ Speichere Element  $x$  an Stelle  $h(\text{Schlüssel}(x))$

## Schubfachprinzip

- ▶ Problem: wir möchten  $N > t$  verwenden



- ▶ Schubfachprinzip
  - ▶ Es gibt Schlüssel  $k \neq k'$ , so dass  $h(k) = h(k')$ : *Hashkonflikt*
  - ▶ Stärker: es gibt einen Platz  $i \in \{0, 1, \dots, t - 1\}$ , so dass  $h(k) = i$  für mindestens  $\lceil \frac{N}{t} \rceil$  Schlüssel  $k$

# Hashing mit Verkettung

- ▶ Mögliche Art, Hashkonflikte zu lösen
- ▶ Statt Feld von Elementen wird Feld von Listen von Elementen verwendet
- ▶ Worst-Case Analyse:  $T$  enthält  $n$  Elemente
  - ▶ Suche: Suche Schlüssel  $k$  in Liste  $T[h(k)]: O(n)$
  - ▶ Einfügen: Füge Element  $x$  am Anfang von  $T[h(\text{Schlüssel}(k))]$  ein:  $O(1)$
  - ▶ Löschen: Lösche Element  $x$  aus Liste  $T[h(k)]:$   
Mit doppelt verlinkten Listen: Laufzeit Suche +  $O(1)$

## Worst-Case Analyse

Im schlechtesten Fall sind alle  $n$  Elemente in einer Liste gespeichert, und die Laufzeit der Suche ist  $\Theta(n)$  – nicht besser als bei einer Liste.

# Average-Case Analyse

Laufzeit hängt davon ab, wie gut die Hashfunktion die Schlüssel verteilt.

## Einfaches gleichverteiltes Hashing

**Annahme** Wahrscheinlichkeit, dass ein beliebiges Element in einen der  $t$  Plätze hasht, ist gleichverteilt

- ▶  $n_j = |T[j]|$  für  $j \in \{0, 1, \dots, t - 1\}$
- ▶ Erwartungswert  $E[n_j] = \frac{n}{t} := \alpha$
- ▶  $\alpha$  nennen wir *Belegungsfaktor*

**Satz** In einer Hash-Tabelle, in der Konflikte durch Verkettung gelöst werden, und unter Verwendung des einfachen gleichverteilten Hashings, benötigt eine Suche durchschnittlich Zeit  $\Theta(1 + \alpha)$ .

**Beweis** In Vorlesung besprochen.

## “Facts of life in probability land”

$n$  Bälle werden in  $t$  Eimer geworfen, unabhängig, jeder gleichverteilt auf die Eimer  
(oder die “zufällige” Hashfunktion bildet  $n$  Schlüssel in  $t$  Tabellenplätze ab)

### Fakten

- ▶ Im Erwartungswert enthält jeder Eimer  $n/t$  Bälle.
- ▶ Wenn  $n \geq \Omega(\sqrt{n})$  dann ist eine Kollision recht wahrscheinlich, d.h. es gibt einen Eimer mit mehr als einem Ball.
- ▶ Wenn  $n = t$  dann sind im Erwartungswert  $n/e = n/2.71$  Eimer leer.
- ▶ Erst wenn  $n = \Omega(n \log n)$  kann man damit rechnen, dass es keinen leeren Eimer gibt.
- ▶ Wenn  $n = t$  dann ist es sehr wahrscheinlich, dass ein Eimer mindestens  $\log n / \log \log n$  Bälle enthält.

## Einfaches gleichverteiltes Hashing

### Problem:

- ▶ Wir machen Annahmen über die Eingabe, die nicht immer realistisch sind
- ▶ Für eine gegebene Hashfunktion können besonders “schwierige” Elemente gewählt werden, die zu hohen Laufzeiten führen

# Universelles Hashing

**Idee** Hashfunktion wird zufällig und unabhängig von Schlüsselwerten gewählt

- ▶ Das heisst, das mehrfache Ausführen mit denselben Elementen führt zu unterschiedlichen Hashtabellen
- ▶ Wir nehmen den Zufall nun selbst in die Hand

**Def.** Sei  $\mathcal{H}$  eine endliche Menge an Hashfunktionen von  $U$  nach  $\{0, 1, \dots, t - 1\}$ .  $\mathcal{H}$  ist *universell* falls für  $k_i, k_j \in U$  mit  $k_i \neq k_j$  die Anzahl der Hashfunktionen  $h \in \mathcal{H}$  mit  $h(k_i) = h(k_j)$  höchstens  $\frac{|\mathcal{H}|}{t}$  ist.

# Universelles Hashing

**Satz** Sei  $h$  eine zufällig gewählte Hashfunktion aus einer Menge  $\mathcal{H}$  an universellen Hashfunktionen. Sei  $T$  eine Hashtabelle der Größe  $t$ , in der Konflikte mit Verkettung gelöst werden, und in die  $n$  Schlüssel mit Hilfe von  $h$  eingefügt wurden. Der Erwartungswert der Länge der Liste  $T[h(k)]$ , die nach Schlüssel  $k$  durchsucht wird, ist höchstens  $1 + \alpha$ .

**Kor.** Universelles Hashing mit Verkettung zur Lösung von Konflikten erlaubt es, in  $O(1 + \alpha)$  erwarteter Zeit in einer Hashtabelle der Größe  $t$ , die  $n$  Elemente enthält, zu suchen.

⇒ Für  $t = O(n)$  brauchen die Operationen Suche, Einfügen und Löschen durchschnittlich Zeit  $O(1)$ .

# Universelles Hashing

Wie wird universelles Hashing implementiert?

- ▶ Klasse `UniversellesHashing`
- ▶ Zufallsgenerator im Konstruktor erzeugt eine Hashfunktion  $h \in \mathcal{H}$
- ▶ Destruktor löscht  $h$
- ▶ So lange eine Instanz existiert, wird die Hashfunktion  $h$  nicht verändert

# Universelles Hashing

Zu zeigen: es gibt Mengen von universellen Hashfunktionen

- Bsp.
- ▶  $t$  ist Primzahl, z.B. 257
  - ▶  $x \in U$  kann eindeutig als  $r + 1$ -Tupel  $x = \langle x_0, x_1, \dots, x_r \rangle$  mit  $r > 0$  und  $0 \leq x_i < t$  geschrieben werden, z.B. bitweise geschrieben
  - ▶ Hashfunktion: Für jedes  $a = \langle a_0, a_1, \dots, a_r \rangle \in \{0, 1, \dots, t - 1\}^{r+1}$  definieren wir die Hashfunktion  $h_a : U \rightarrow \{0, 1, \dots, t - 1\}$

$$h_a(x) = \left( \sum_{i=0}^r a_i x_i \right) \text{ mod } t$$

# Universelles Hashing

Bsp. Fortsetzung:

**Lemma** Die Menge  $\mathcal{H} = \{h_a \mid a \in \{0, \dots, t-1\}^{r+1}\}$  ist universell.

**Beweis**

- ▶ Betrachte  $x, y \in U$  mit  $x \neq y$  und o.B.d.A.  $x_0 \neq y_0$
- ▶ Es ist  $h_a(x) = h_a(y)$  falls
$$a_0 \underbrace{(x_0 - y_0)}_{\neq 0} = \left( \sum_{i=1}^r a_i (x_i - y_i) \right) \pmod t$$
- ▶ Sind  $a_1, \dots, a_r$  fest, so gibt es nur eine Wahl für  $a_0$  da  $t$  prim ist
- ▶ Es gibt  $t^r$  Möglichkeiten, um  $a_1, \dots, a_r$  zu wählen
- ▶ Es gibt also  $t^r \leq \frac{|\mathcal{H}|}{t} = \frac{t^{r+1}}{t}$  Funktionen in  $\mathcal{H}$ , so dass  $h_a(x) = h_a(y)$