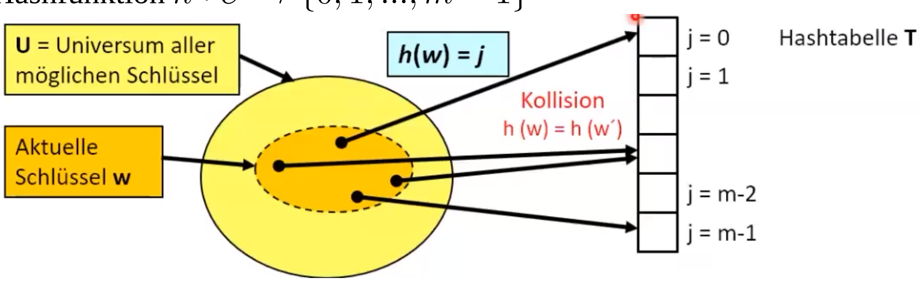
Motivation

- Wörterbuchproblem lösen
- Operationen
  - insert
  - search
  - remove
- Idee
  - Inhalt nicht suchen
  - Adresse berechnen in $O(1)$

Definition

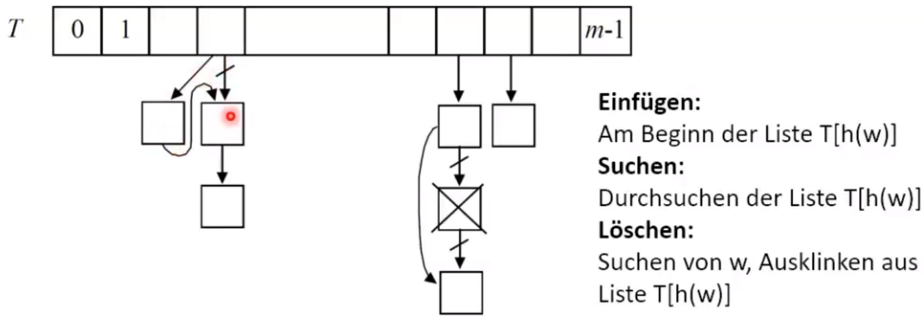
- lineares Feld $T[0...m-1]$
- Wert $w \in U$ wird in $T[h(w)]$ gespeichert
- Hashfunktion $h : U \rightarrow \{0, 1, ..., m-1\}$
- 

Kollisionsproblem

- endlich große Tabelle
- Menge der möglichen Schlüssel größer
- Kollision
  - unterschiedliche Schlüssel haben denselben Index
- Belegungsfaktor $\alpha = \frac{m}{n}$

Kollisionsbehandlung

- Überläuferlisten (Chaining)
  - Daten in verkettete Liste speichern bei Kollision

  - 

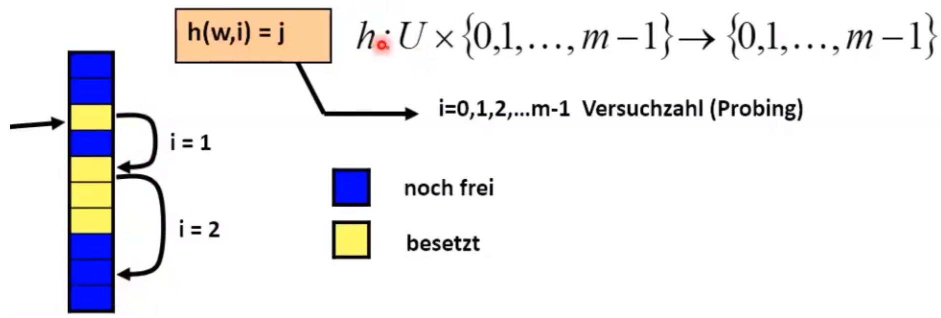
  - Laufzeiten
    * worst case $\Theta(n)$ für Suchen, Löschen
      - sehr unwahrscheinlich

        $$prob = \left(\frac{1}{m}\right)^{n-1}$$
      - **Erwartete Laufzeit**

        Einfügen: O(1)

        Suchen: O(1+α)

        Löschen: O(1+α)
    *
- Offene Adressierung
  - Werte direkt in Tabelle speichern
    * T[0..m-1] selbst gespeichert ⇒ $\alpha = n/m \leq 1$
  - Bei Kollision wird neue Adresse berechnet bis freie gefunden

* 

- Näherungen

  Linear Probing: $h(w,i) = [h'(w) + i] \bmod m$

  Problem: benachbarte Felder wahrscheinlicher belegt (**primary clustering**)

  Quadratic Probing: $h(w,i) = [h'(w) + f(i)] \bmod m$

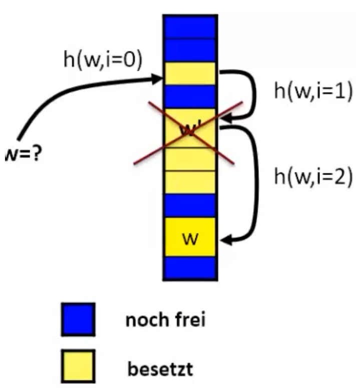
  f(i)...quadratische Funktion; bei einer Kollision immer noch dieselbe Indexfolge (**secondary clustering**)

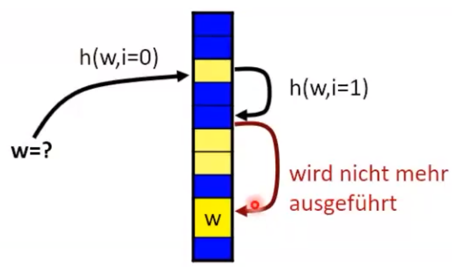
  Double Hashing: $h(w,i) = [h_1(w) + ih_2(w)] \bmod m$

  *

- Löschproblem

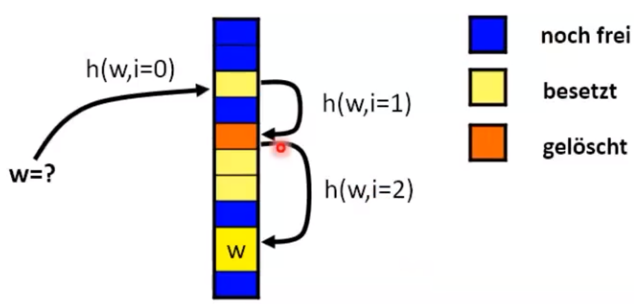
  - Löschen von w'
  - Suchen von w

  *

  * Lösung

  Anmerkung: Beim Einfügen werden gelöschte Felder gleich wie freie Felder behandelt

  ◆

- Operationen

```
EINFÜGE(T, w)
i ← 0
REPEAT
   ind ← h(w,i)
   IF T[ind] frei THEN
      T[ind] ← w
      return
   i ← i+1
UNTIL i=m
return „overflow“
```

```
SUCHE(T, w)
i ← 0
REPEAT
   ind ← h(w,i)
   IF T[ind] = w THEN
      return ind
   i ← i+1
UNTIL (T[ind] frei) or (i=m)
return „nicht gefunden“
```

Erwartete Laufzeit $O\left(\frac{1}{1-\alpha}\right)$

Hashfunktion

- Ziel: Werte gleich auf Feld zu verteilen
  - Verteilung der Werte meist unbekannt
- heuristische Wahl der Funktion
  - möglichst effizient
  - gleichwahrscheinliche Indizes
  - ähnliche Werte getrennt
    - unabhängig von Mustern in den Daten
- theoretische ideale Hashfunktion
  - jeder Index ist gleich wahrscheinlich

$$\Pr[h(w) = j] = \frac{1}{m} \quad \forall w \in U, j \in \{0,\dots,m-1\}$$

Arten von Hashfunktionen

- Divisionsmethode

$$h(w) = w \bmod m$$

  - schnell berechenbar
  - nicht für alle m geeignet
    - gutes m, wenn prim
    - e.g. $m = 2^k$
    - $h(w)$ hängt von letzten $k-1$ Stellen ab
- Multiplikationsmethode

- Nachkommastellen von Multiplikation mit Konstante A
- Multiplikation mit m abrunden

$$h(w) = \left\lfloor m \cdot \underbrace{\mathrm{frac}(w \cdot A)}_{\in [0,1)} \right\rfloor$$

-
- unabhängig vom m

Laufzeiten

-

	Lineares Feld	Lineare Liste	Gestreute Speicherung	
			Überläuferlisten	offene Adressierung
			α=n/m (z.B. 10)	α=n/m (z.B. 0.5)
Suchen	O(n)	O(n)	O(1+α)	≈ O(1/(1-α))
Einfügen	O(1)	O(1)	O(1)	wie oben
Suchen und Entfernen	O(n)	O(n)	O(1+α)	wie oben