

Vorbesprechung zur ersten Aufgabe

Effizientes Hashing

Daniel Stein

Software-Praktikum "Muster- und Bilderkennung" – 05.04.2007

**Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik VI
Computer Science Department
RWTH Aachen University, Germany**

Prinzip

- ▶ **Zur Erinnerung:** Beim *Hashing* wird ein Datensatz bezüglich seines Schlüssels $k \in \mathcal{K}$ in einem linearen Feld mit Indizes $0, \dots, m - 1$ gespeichert. Dieses Feld bezeichnet man als *Hashtabelle* der Größe m .
- ▶ **Prinzip:**
 - ▶ Zur Verfügung stehen m Speicherplätze in einer *Hashtabelle* T :
var T: array [0.. $m - 1$] of element .
 - ▶ Dann wird ein Schlüssel $x \in \text{Universum } U = \{0, \dots, N - 1\}$ auf einen Speicherplatz in der Hashtabelle abgebildet. Dazu dient die *Hashfunktion* $h(x)$ (deutsch: Schlüsseltransformation, Streuspeicherung):

$$\begin{aligned} h : U &\rightarrow \{0, 1, \dots, m - 1\} \\ x &\rightarrow h(x) \end{aligned}$$

- ▶ Nach dieser Adressberechnung kann das Element mit dem Schlüssel x an der Stelle $T[h(x)]$ gespeichert werden, falls dieser Platz noch frei ist.

▶ Prinzip:

- ▶ Da in der Regel $(m \cong n) \ll N$ ist, kann es zu *Kollisionen* kommen, d.h.

$$h(x) = h(y) , \text{ für } x \neq y .$$

- ▶ x wird also nicht notwendigerweise in $T[h(x)]$ selbst gespeichert. Entweder enthält $T[h(x)]$ dann einen Verweis auf eine andere Adresse (offene Hashverfahren), oder mittels einer *Sondierfunktion* muß ein anderer Speicherplatz berechnet werden (geschlossene Hashverfahren).
- ▶ Dementsprechend hat die Operation *Search* (x,S) dann zwei Teile. Zunächst muß $h(x)$ berechnet werden, dann ist x in $T[h(x)]$ zu suchen.

▶ Anforderungen an die Hashfunktion:

- ▶ Die ganze Hashtabelle sollte abgedeckt werden, d.h. $h(x)$ ist surjektiv .
- ▶ $h(x)$ soll die Schlüssel x möglichst gleichmäßig über die Hashtabelle verteilen.
- ▶ Die Berechnung soll effizient, also nicht zu rechenaufwendig sein.

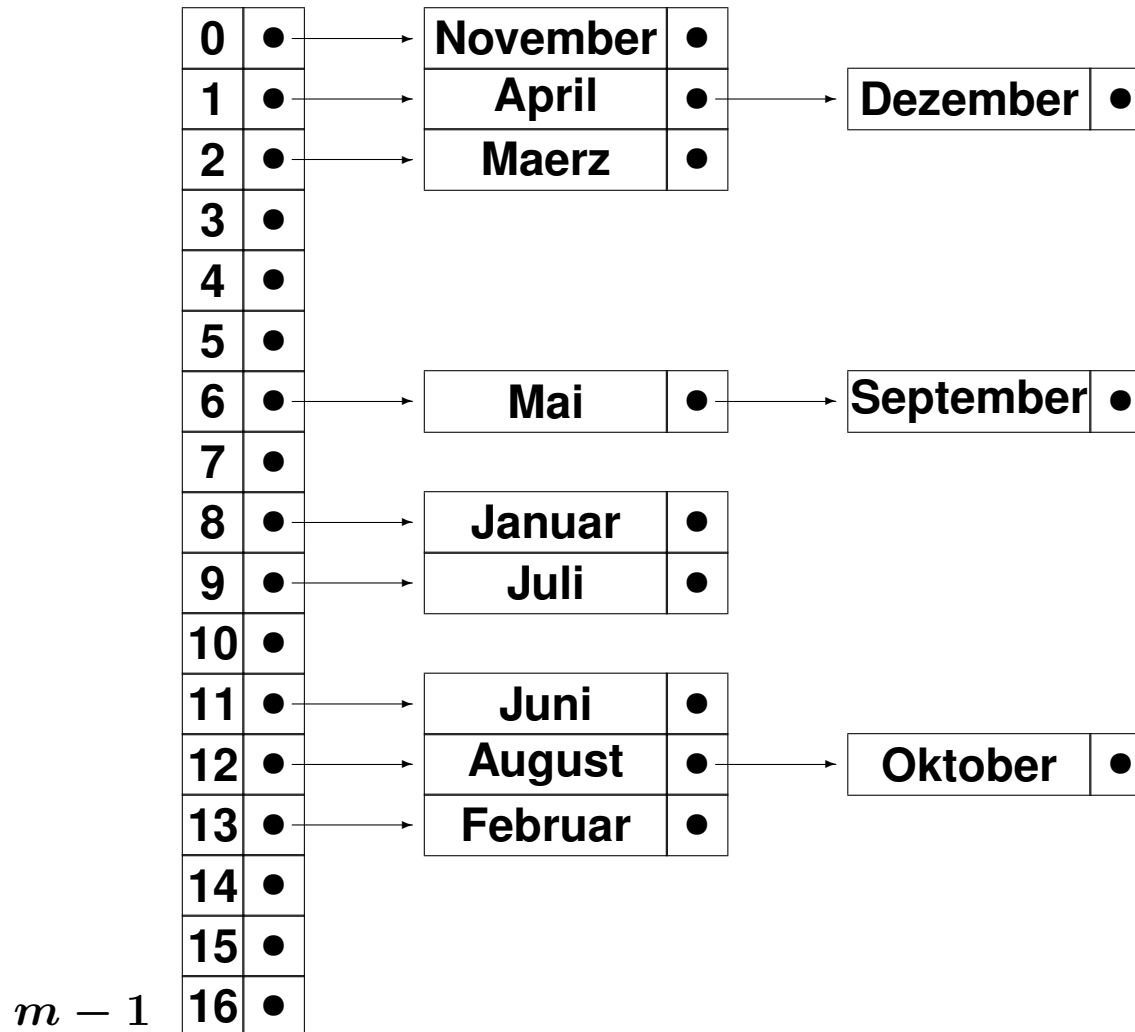
Verteilen von Monatsnamen über eine Tabelle mit m Elementen, die Hashfunktion sei geeignet gewählt!

Beachte: Es gibt drei Kollisionen!

0:	November
1:	April, Dezember
2:	März
3:	
4:	
5:	
6:	Mai, September
7:	
8:	Januar
9:	Juli
10:	
11:	Juni
12:	August, Oktober
13:	Februar
14:	
15:	
16:	

Open Hashing (Hashing mit Verkettung)

Jeder Behälter wird durch eine beliebig erweiterbare Liste von Schlüsseln dargestellt. Ein Array von Zeigern verwaltet die Behälter.
 Für obiges Beispiel mit den Monatsnamen ergibt sich:



- ▶ Diese Verfahren arbeiten mit einem statischen Array und müssen daher Kollisionen mit einer neuerlichen Adressberechnung behandeln (Offene Adressierung). Dazu dient eine Sondierungsfunktion oder Rehashing-Funktion, die eine Permutation aller Hashtabellen-Plätze und damit die Reihenfolge der zu sondierenden Plätze angibt:
 - ▷ jedes $x \in U$ definiert eine Folge $h(x, j)$, $j = 0, 1, 2, \dots$ von Positionen in der Hashtabelle
 - ▷ diese Folge muß bei jeder Operation durchsucht werden
 - ▷ Definition der zusätzlichen Hashfunktionen $h(x, j)$, $j = 0, 1, 2, \dots$ erforderlich
- ▶ **Achtung:** Die Operation "Delete(x, S)" erfordert eine Sonderbehandlung, weil die Elemente, die an x mittels Rehashing vorbeigeleitet wurden, auch nach Löschen von x noch gefunden werden müssen.
- ▶ Zwei Arten von Kollisionen können unterschieden werden für $x \neq y$:
 1. Primärkollisionen: $h(x, 0) = h(y, 0)$
 2. Sekundärkollisionen: $h(x, j) = h(y, 0)$ für $j = 0, 1, 2, \dots$

Lineares Sondieren

- ▶ Die (Re)Hashfunktionen $h(x, j)$, $j = 0, 1, 2, \dots$ sollen so gewählt werden, daß für jedes x der Reihe nach sämtliche m Zellen der Hashtabelle inspiziert werden. Dabei muss jedoch beachtet werden, daß für zwei Werte x und y mit gleichen Hashwerten $h(x, 0) = h(y, 0)$ auch deren Sondierbahnen gleich sind. Auf diese Weise können lange Sondierketten entstehen.
- ▶ Einfachster Ansatz: Lineares Sondieren

$$h(x, j) = (h(x) + j) \mod m \quad 0 \leq j \leq m - 1$$

- ▶ Ein Problem beim linearen Sondieren ist die Clusterbildung: Es besteht die Tendenz, daß immer längere zusammenhängende, belegte Abschnitte in der Hashtabelle entstehen, so genannte Cluster. Diese Cluster verschlechtern deutlich die mittlere Suchzeit im Vergleich zu einer Hashtabelle mit demselben Belegungsfaktor, in der aber die Elemente zufällig gestreut sind.

Quadratisches Sondieren

- ▶ **Vorstufe:** Um die oben angesprochene Häufung zu vermeiden, kann man mit folgender (Re-)Hashfunktion quadratisch sondieren:

$$h(x, j) = (h(x) + j^2) \pmod{m} \quad \text{für } 0 \leq j \leq m - 1$$

Warum j^2 ?

Wenn m eine Primzahl ist, dann sind die Zahlen $j^2 \pmod{m}$ alle verschieden für $j = 0, 1, \dots, \left\lfloor \frac{m}{2} \right\rfloor$.

- ▶ **Verfeinerung:** $1 \leq j \leq \frac{m-1}{2}$

$$\begin{aligned} h(x, 0) &= h(x) \pmod{m} \\ h(x, 2j-1) &= (h(x) + j^2) \pmod{m} \\ h(x, 2j) &= (h(x) - j^2) \pmod{m} \end{aligned}$$

Wähle m als Primzahl mit $m \pmod{4} = 3$ (**Basis: Zahlentheorie**).

- ▶ **Quadratisches Sondieren ergibt keine Verbesserung für Primärkollisionen ($h_0(x) = h_0(y)$), aber es vermeidet Clusterbildung bei Sekundärkollisionen ($h_0(x) = h_k(x)$ für $k > 0$), das heißt die Wahrscheinlichkeit für die Bildung längerer Ketten wird herabgesetzt.**