

Algorithmen und Datenstrukturen

Sommersemester 2020

Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 9

20. Mai 2020

Anwendung stabiler Sortialgorithmen

Sortieren nach mehreren Kriterien

Beispiel:

Sortiere die Studierenden der Uni Trier nach ihrem Alter in Jahren (1. Kriterium), Studierende mit gleichem Alter sollen nach ihrer Matrikelnummer (2. Kriterium) sortiert werden.

Lösung:

Sortiere zunächst nach Matrikelnummer und dann nach Alter, d.h. zunächst nach dem weniger wichtigen dann nach dem wichtigsten Kriterium. Für den zweiten Schritt sollte ein stabiles Sortiervorgehen verwendet werden, damit bei gleichem Alter die Ordnung gemäß der Matrikelnummer erhalten bleibt.

Allgemeiner: Sortieren von Zeichenketten (Strings)

Radix Sort: Sortieren von Strings

Es handelt sich um eine iterierte Variante von Bucketsort.

z.B. kann man bei natürlichen Zahlen als “Fächer” die möglichen Werte der Buchstaben ansehen. Sortiert man nun zunächst mit Bucketsort gemäß dem letzten (also unwichtigsten) Buchstaben, dann nach dem zweitletzten usf. bis zum ersten Buchstaben, erhält man am Ende eine vollständig alphabetisch geordnete Liste.

Formaler unterscheidet man zwischen sich abwechselnden **Partitionsphasen** und **Sammelphasen**.

Beispiel: Sortieren der Strings (Zahlen) 124, 523, 483, 128, 923, 584.

0	1	2	3	4	5	6	7	8	9
			523	124				128	
			483	584					
			923						

--> 523, 483, 923, 124, 584, 128

0	1	2	3	4	5	6	7	8	9
		523						483	
		923						584	
		124							
		128							

--> 523, 923, 124, 128, 483, 584

0	1	2	3	4	5	6	7	8	9
	124			483	523				923
	128				584				

--> 124, 128, 483, 523, 584, 923 OK!

Kapitel 3: Wörterbücher

Verwaltung einer Menge S von n Schlüsseln unter den Operationen

Initialisierung (INIT/CLEAR)

Nachschlagen (LOOKUP)

Einfügen (INSERT)

Löschen (DELETE)

Datenstrukturen für Wörterbücher

1. **Hashing**
2. **Binäre Suchbäume**
3. **AVL-Bäume**

Hashing

Spezielle Variante des Wörterbuchproblems mit ganzzahligen Schlüsseln.

Genauer:

Verwalte eine Menge S von n Schlüsseln aus dem Universum $U = \{0, \dots, N-1\}$ unter den Wörterbuchoperationen LOOKUP, INSERT, DELETE wobei $n \ll N$, d.h. die Anzahl der abgespeicherten Schlüssel ist viel kleiner als die Menge der möglichen Schlüssel

~> Speicherung von dünn besetzten Tabellen

Dabei soll nur Platz $O(n)$ verwendet werden und die Operationen möglichst effizient, im Idealfall in $O(1)$, realisiert werden.

Bitvektor

Einfache Lösung mit Speicherplatz $O(N)$ (Größe des Universums)

Verwende ein Feld (Tafel) $T[0..N - 1]$ von Boole'schen Werten
d.h. **Bitvektor** über das gesamte Universum.

INIT: **for** $x = 0$ **to** $N - 1$ **do** $T[x] \leftarrow \text{false}$; **od**

INSERT(x): $T[x] \leftarrow \text{true}$;

DELETE(x): $T[x] \leftarrow \text{false}$;

LOOKUP(x): **return** $T[x]$;

Hashing

Hashtafel $T[0..m]$ ($m = \mathcal{O}(n)$).

Hashfunktion $h : \mathcal{U} \rightarrow [0..m - 1]$

Beispiel:

$S = \{ 3, 5, 9, 16, 15, 23 \}$ d.h. $n = 6$

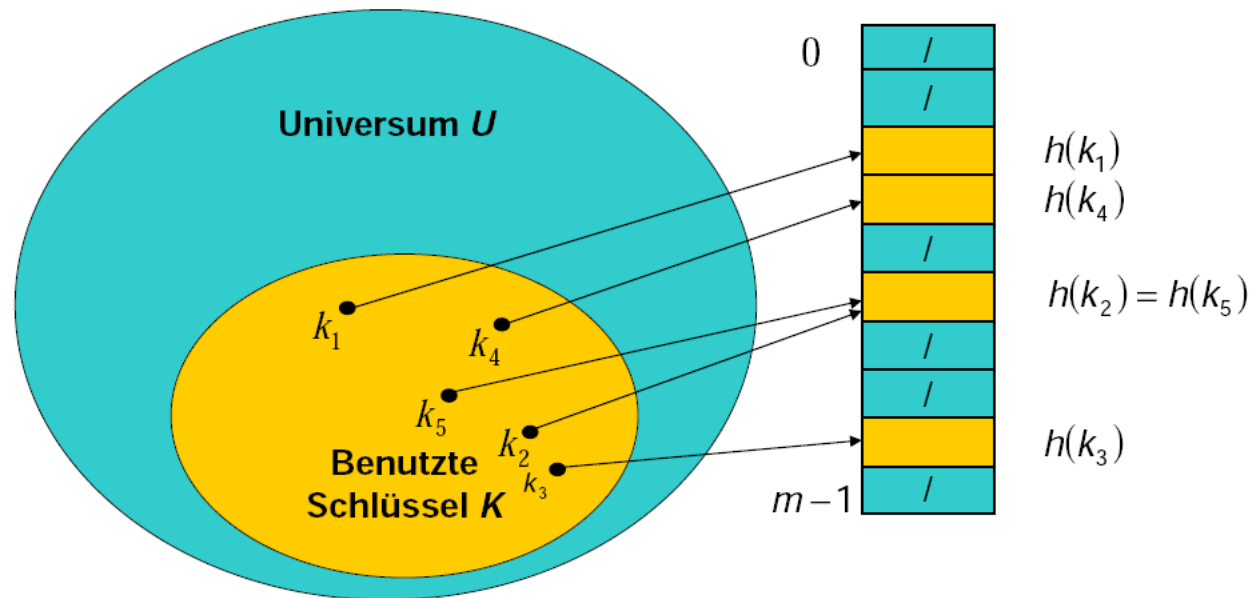
$m = 7$

$h(x) = x \bmod m$

Problem: Kollisionen

Die Schlüssel 9, 16, 23 werden alle auf dieselbe Position 2 abgebildet.

Hashing - Illustration



Zum Kollisionsproblem

Angenommen, unsere Hashfunktion verteilt die n tatsächlich Schlüssel völlig gleichmäßig auf die m Positionen der Tafel ($m \geq n$).

Sei $P(i)$ die Wahrscheinlichkeit dafür, dass der i -te Schlüssel auf eine freie Position abgebildet wird, wenn alle “vorigen” Schlüssel ebenfalls kollisionsfrei abgebildet wurden.

→ Wahrscheinlichkeit, dass keine Kollision nach i Zuweisungen erfolgte, beträgt $P(1) \cdot P(2) \cdot \dots \cdot P(i)$.

Für $n = 23$ und $m = 365$ entspricht das dem **Geburtstagsparadoxon**:

Lediglich 23 zufällig ausgewählte Personen genügen, um mit Wahrscheinlichkeit $> 0,5$ zu gewährleisten, dass zwei von ihnen am selben Tag Geburtstag haben.

Für uns bedeutet es: Kollisionsfreiheit ist nicht völlig zu umgehen.

Kollisionsbehandlung

(1) Hashing mit Verkettung

Jede Tafelposition wird durch eine beliebig erweiterbare Liste dargestellt.
Die Hashtabelle enthält daher Listenköpfe.

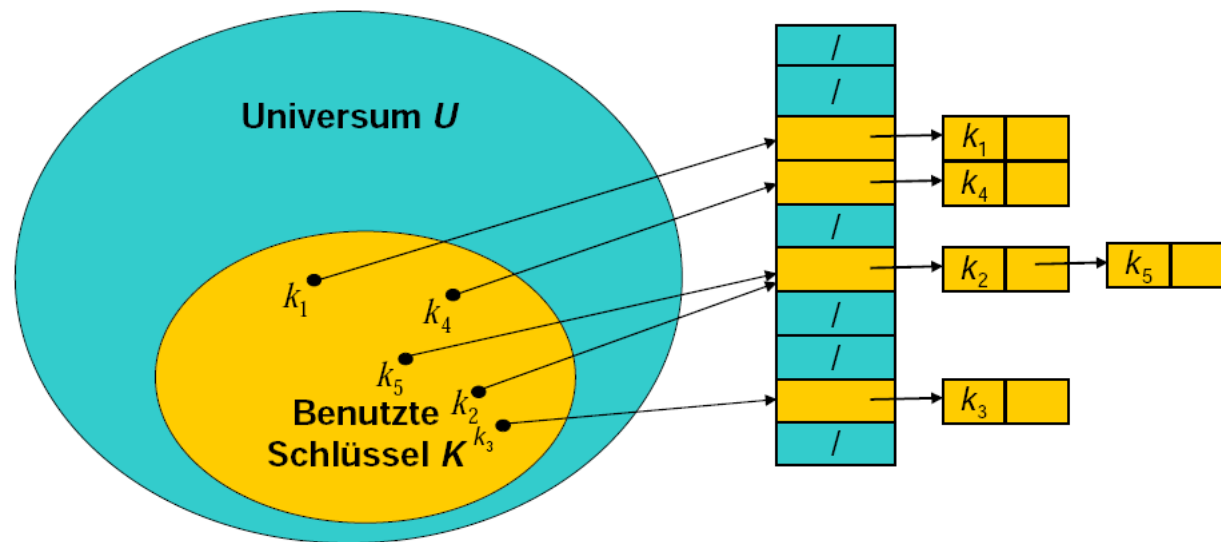
(2) Hashing mit offener Adressierung

Neben der Hashfunktion $h = h_0$ werden noch weitere Hashfunktionen h_i definiert, die eine Kollisionsbehandlung ermöglichen.

(3) Perfektes Hashing

Verwende eine injektive Hashfunktion
d.h. Kollisionen werden komplett vermieden.

Hashing mit Listen - Illustration



Hashing mit Verkettung

Betrachte $h : U \rightarrow [0..m - 1]$.

$$\delta_h(x, y) := \begin{cases} 1 & \text{falls } x \neq y \text{ und } h(x) = h(y) \\ 0 & \text{sonst} \end{cases}$$

Wir erweitern auf Schlüsselmengen $S \subseteq U$ mit

$$\delta_h(x, S) = \sum_{y \in S} \delta_h(x, y).$$

Ist S die Menge der abgespeicherten Schlüssel, so ist $\delta_h(x, S)$ die Länge der Liste an der Stelle $h(x)$.

Wie groß ist $\delta_h(x, S)$?

Wir betrachten zwei Fälle:

Im schlechtesten Fall: $n = |S|$, dann generiert Hashing zu linearer Suche.

Im mittleren Fall: Analyse folgt...

Hashing mit Verkettung (mittlerer Fall)

Betrachte $h : U \rightarrow S = \{0, \dots, m - 1\}$.

Annahmen:

- (1) h verteilt S gleichmäßig.
- (2) Die Schlüsselmenge $S = \{x_1, \dots, x_n\}$ wird gemäß Gleichverteilung und unabhängig voneinander “gezogen”

Frage: Wie groß ist $\delta_h(x, S)$ im Mittel ?

Sei $h_i = h(x_i)$. Die ZV h_i nimmt nach Annahmen die Zahlen in $[0..m - 1]$ gleichwahrscheinlich an.