

# ***Algorithmik***

Sebastian Iwanowski  
FH Wedel

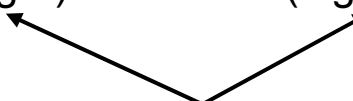
## 3. Lösungen für das Wörterbuchproblem 3.1 Hashing

# Algorithmik 3

## Implementierung von Dictionaries

Ein Dictionary ist eine Datenstruktur für mit einem Schlüssel vergleichbare Elemente, welche die Funktionen member (key), insert (key, newdata) und delete (key) zur Verfügung stellt

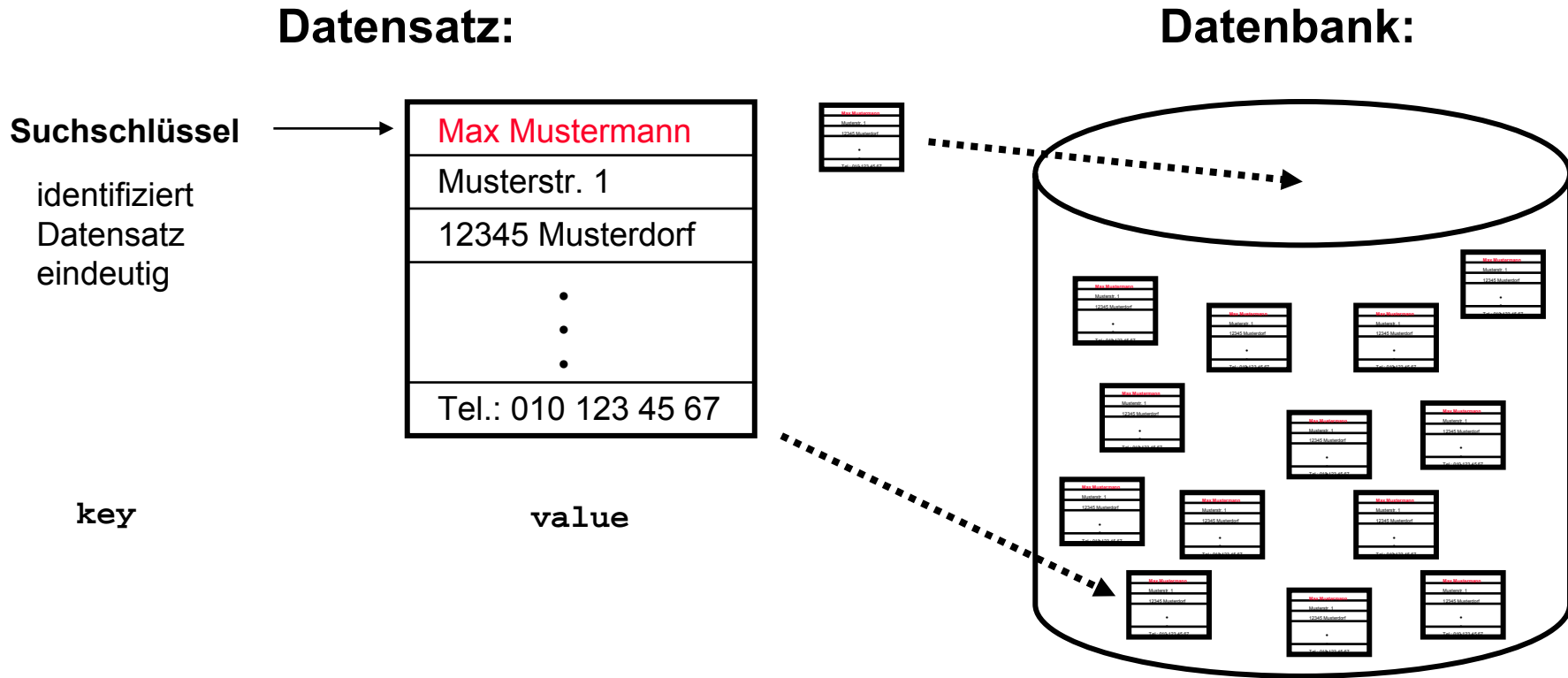
### Sortiertes Feld als Dictionary:

member (key)	Laufzeit $\Theta(\log n)$ w.c. und $\Theta(\log \log n)$ a.c. erreichbar
	
	nicht vom selben Algorithmus
insert (key, newdata)	Laufzeit $\Theta(n)$ w.c. und a.c.
insert (key, newdata)	Laufzeit $\Theta(n)$ w.c. und a.c.

### Referenzen zum Nacharbeiten und Vertiefen:

Skript Alt, S. 30 – 35 (für member, Wiederholung vom letzten Mal)

# Welches Problem löst Hashing ?



## Datenverwaltungs-Operationen: Map operations

- **Suchen** get (key)
- **Einfügen** put (key, value)
- **Entfernen** remove (key)



**Hashing ist ein Verfahren,  
das diese Operationen effizient löst.**

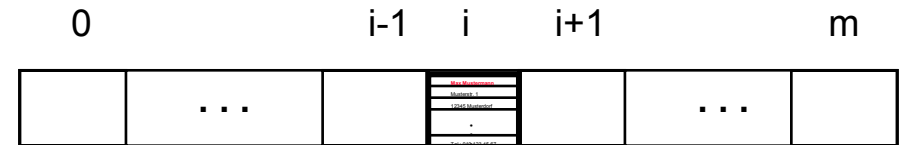


# Aufwerfen von Detailfragen

Datensatz:

Max Mustermann
Musterstr. 1
12345 Musterdorf
•
•
•
Tel.: 010 123 45 67

Hashtabelle T:



1) Wie definiert man eine gute Hashfunktion ?

2) Wo speichert man den Datensatz in der Hashtabelle ab ?

# 1) Wie definiert man eine gute Hashfunktion ?

**Fall 1:** Es gibt mindestens so viele Plätze in der Hashtabelle wie verschiedene Schlüssel

**Ziel:** Jeder Schlüssel wird auf eine *andere* Hashzahl abgebildet.

## *Perfektes Hashing*

**Lösung:** Ordne die Schlüssel nach einer Reihenfolge (z.B. lexikographisch) !  
Bilde jeden Schlüssel auf seine Ordnungszahl ab !

**Beispiel:**  
(für Strings  
als Schlüssel)

„Max Mustermann“ → (13 1 24 0 13 21 19 20 5 18 13 1 14 14)

$$\begin{aligned} \text{hash („Max Mustermann“)} &= 13 \cdot 27^{13} + 1 \cdot 27^{12} + 24 \cdot 27^{11} + 0 \cdot 27^{10} + 13 \cdot 27^9 + 21 \cdot 27^8 + 19 \cdot 27^7 \\ &\quad + 20 \cdot 27^6 + 5 \cdot 27^5 + 18 \cdot 27^4 + 13 \cdot 27^3 + 1 \cdot 27^2 + 14 \cdot 27^1 + 14 \cdot 27^0 \\ &\approx 52966834350000000000 \text{ (20-stellige Zahl)} \end{aligned}$$

Im Allgemeinen sind sehr viele *verschiedene* Schlüssel möglich !

**Fazit:** **Fall 1 ist unrealistisch !**

# 1) Wie definiert man eine gute Hashfunktion ?

**Fall 2:** Es gibt weniger Plätze in der Hashtabelle als verschiedene Schlüssel.

$m$

$k$

Fall 2:  $m < k$

**Folgerung:** Verschiedene Schlüssel müssen auf die gleiche Hashzahl abgebildet werden

*Kollision*

**Ziel:**

Auf jede Hashzahl zwischen 0 und  $m-1$  werden ungefähr gleich viele Schlüssel abgebildet.  
(nämlich ungefähr  $k/m$ )

**Lösung:** Ordne die Schlüssel nach einer Reihenfolge (z.B. lexikographisch) !

Bilde jeden Schlüssel auf seine Ordnungszahl modulo  $m$  ab !

# 1) Wie definiert man eine gute Hashfunktion ?

## Beispiel:

(für Strings  
als Schlüssel)

$m = 1000$       „Antje“  $\rightarrow$  (1 14 20 10 5)

$$\begin{aligned}\text{hash („Antje“)} &= (1 * 27^4 + 14 * 27^3 + 20 * 27^2 + 10 * 27^1 + 5) \bmod 1000 \\ &= 821858 \bmod 1000 \\ &= 858\end{aligned}$$

## Algorithmus für eine gute Hashfunktion (nach *Hornerschema*) :

$$\begin{aligned}\text{hash („Antje“)} &= (((((1 * 27 + 14) * 27 + 20) * 27 + 10) * 27 + 5) \bmod 1000 \\ &= (((((((1 \bmod 1000 * 27 + 14) \bmod 1000) * 27 + 20) \bmod 1000) * 27 + 10) \bmod 1000) * 27 + 5) \bmod 1000\end{aligned}$$

## Java-Code:

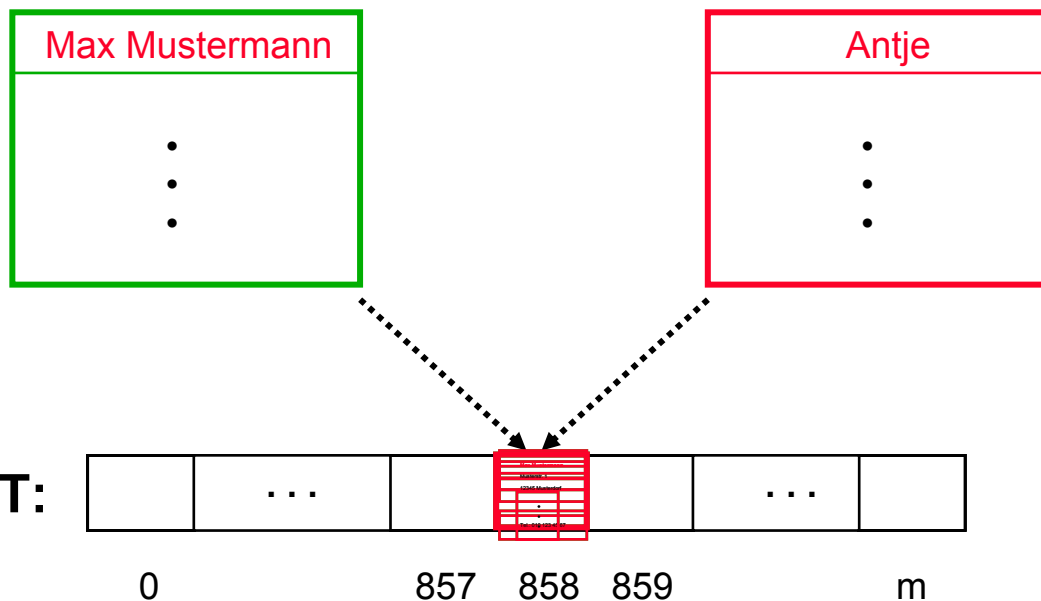
```
static int hash (String key, int m)
{
    int result = 0, numberSymbols = 27;
    for (int i = 0; i < key.length(); i++)
        result = (result*numberSymbols + order (key.charAt(i))) % m;
    return result;
}
```



## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

**Beispiel:**  $\text{hash}(\text{„Max Mustermann“}) = 858$   
 $\text{hash}(\text{„Antje“}) = 858$



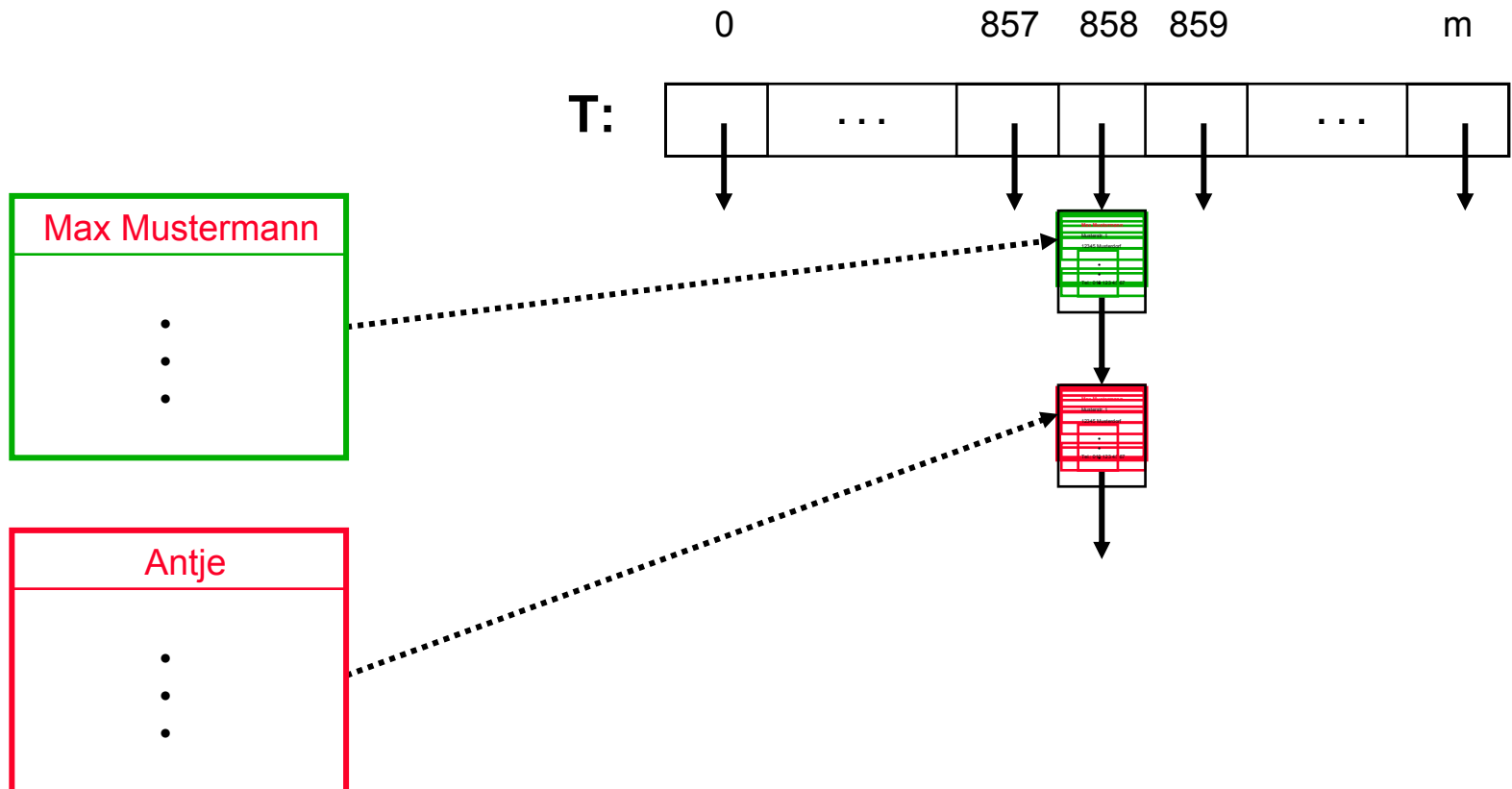
**Hashtabelle T:**

**Hat jemand eine bessere Idee ?**

## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

**Lösung:**  $T[i]$  enthält Zeiger auf verkettete Liste von Datensätzen, deren Schlüssel alle die Hashzahl  $i$  haben.



## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

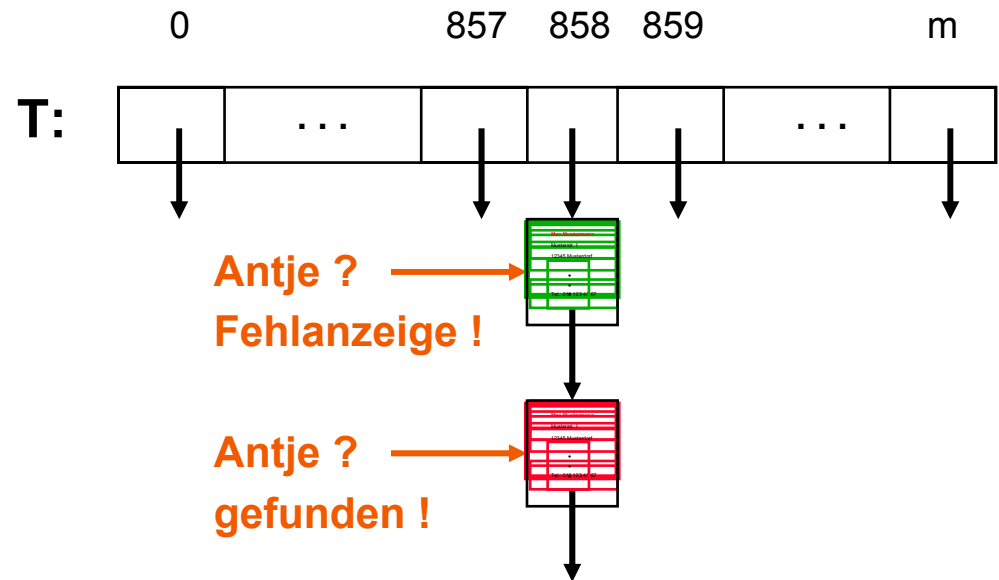
**Lösung:**  $T[i]$  enthält Zeiger auf verkettete Liste von Datensätzen, deren Schlüssel alle die Hashzahl  $i$  haben.

**Suchen:** Antje ?



1) Berechne hash („Antje“)=858

2) Durchsuche Liste von  $T[858]$

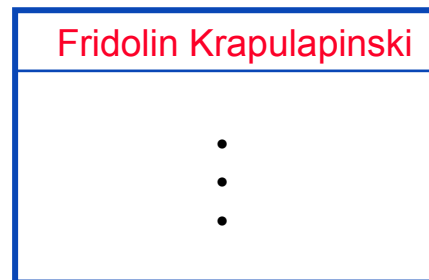


## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

**Lösung:**  $T[i]$  enthält Zeiger auf verkettete Liste von Datensätzen, deren Schlüssel alle die Hashzahl  $i$  haben.

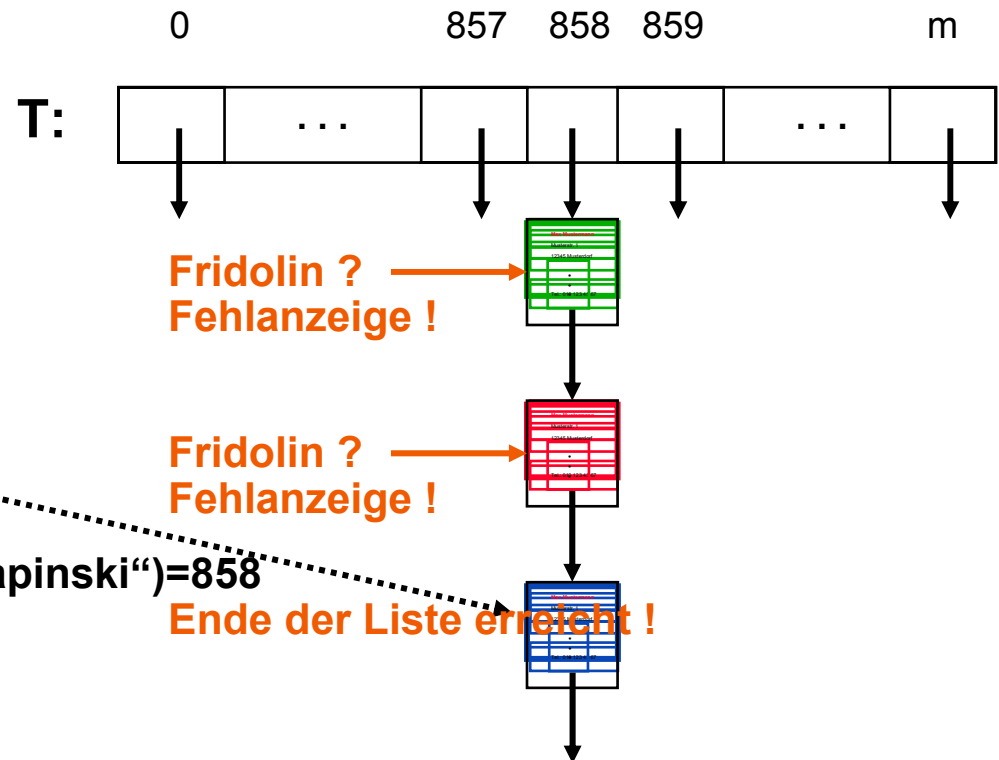
**Einfügen:**



1) Berechne hash („Fridolin Krapulapinski“)=858

2) Durchlaufe Liste von  $T[858]$

3) Füge am Ende ein



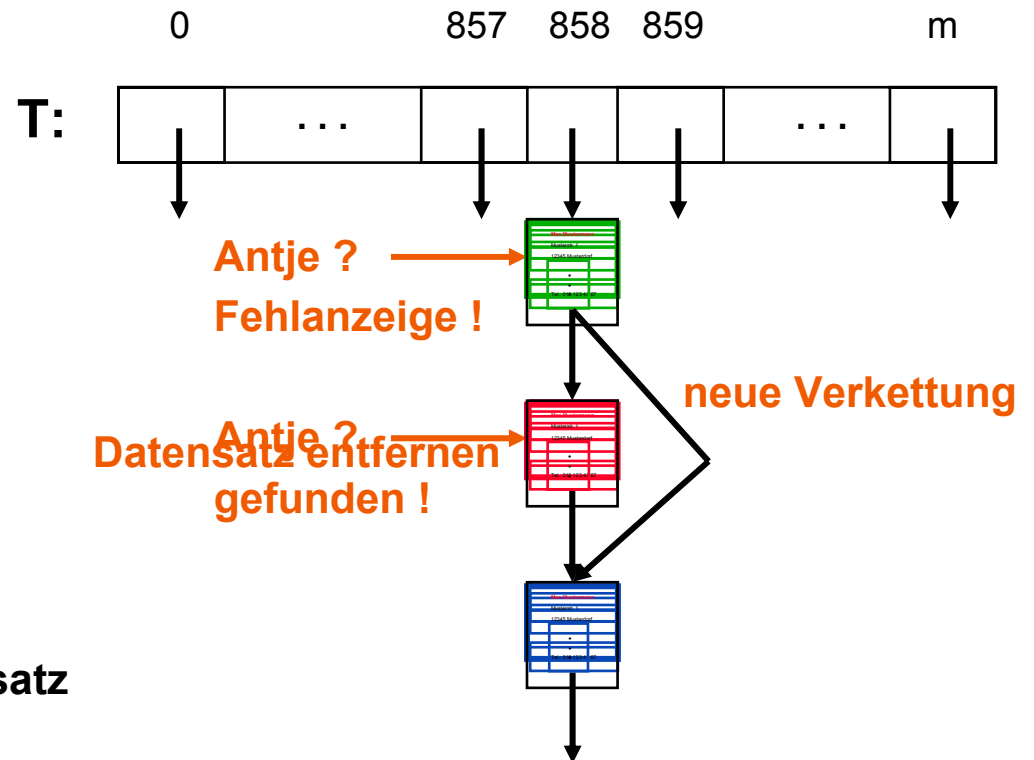
## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

**Lösung:**  $T[i]$  enthält Zeiger auf verkettete Liste von Datensätzen, deren Schlüssel alle die Hashzahl  $i$  haben.

**Entfernen:** Antje 

- 1) Berechne hash („Antje“)=858
- 2) Durchlaufe Liste von  $T[858]$
- 3) Entferne entsprechenden Datensatz



## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

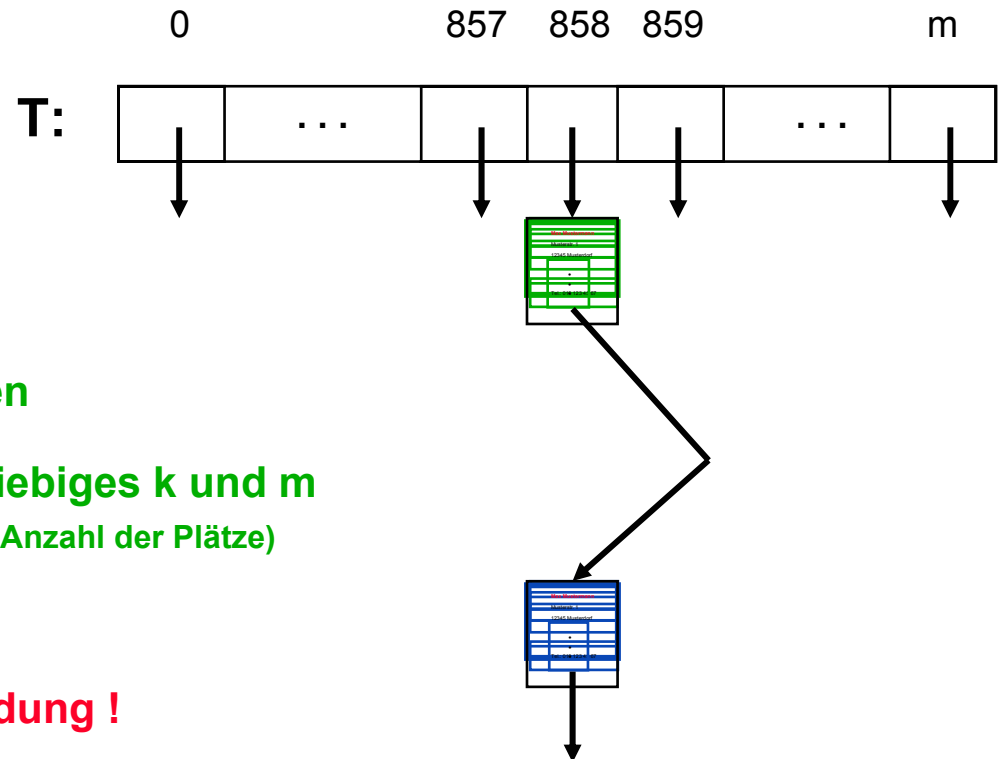
**Lösung:**  $T[i]$  enthält Zeiger auf verkettete Liste von Datensätzen, deren Schlüssel alle die Hashzahl  $i$  haben.

**Bewertung des bisher vorgestellten Verfahrens zur Abspeicherung:**

- **einfach zu implementieren**
- **realisiert Hashing für beliebiges  $k$  und  $m$**   
( $k$ : Anzahl der Schlüssel,  $m$ : Anzahl der Plätze)

**Kritik:**

- **Speicherplatzverschwendung !**



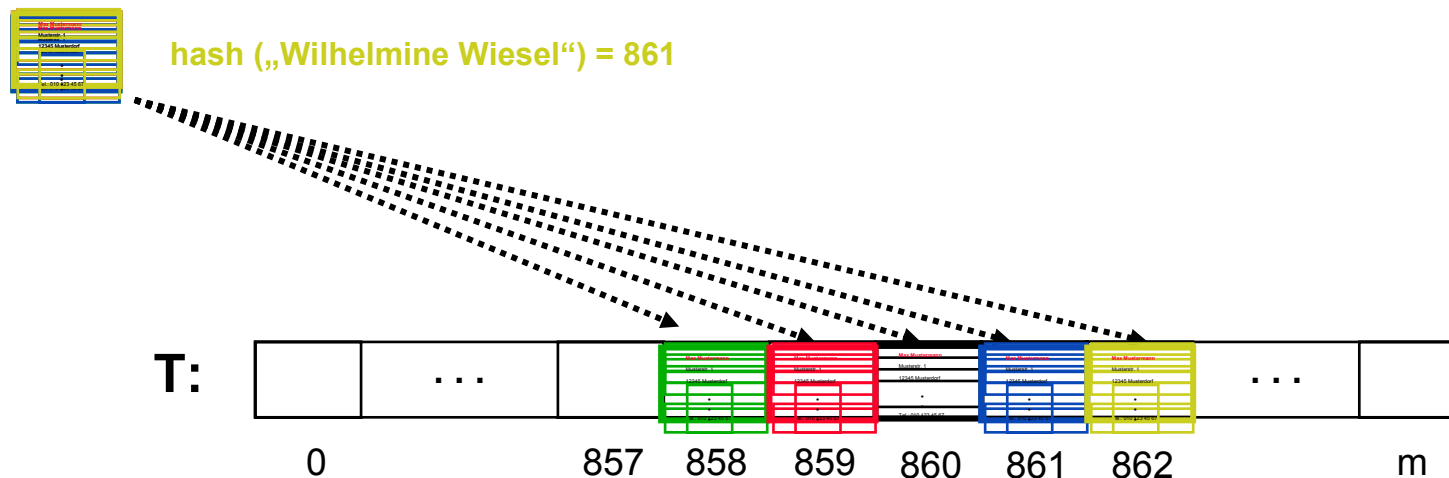
## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

**Alternative Lösung:** Suche anderen leeren Platz in der Hashtabelle:  
*(Offenes Hashing)* Rücke von  $T[i]$  jeweils nach einer festen Regel weiter, bis ein freier Platz gefunden wird

↓  
*Sondierungsregel*

**Beispiel für Sondierungsregel:** rücke um eins nach rechts



## 2) Wo speichert man den Datensatz in der Hashtabelle ab ?

**Problem: Wie verfährt man bei Kollisionen ?**

**Alternative Lösung:** Suche anderen leeren Platz in der Hashtabelle:  
*(Offenes Hashing)* Rücke von  $T[i]$  jeweils nach einer festen Regel weiter,  
bis ein freier Platz gefunden wurde

↓  
*Sondierungsregel*

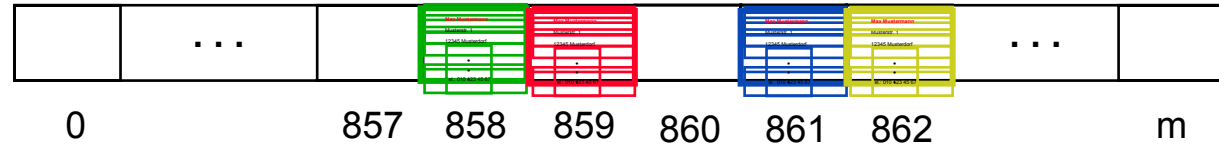
**Weitere Möglichkeiten für Sondierungsregeln:**

1. Weiterrücken in quadratischen Abständen
2. Weiterrücken gemäß einer zweiten Hashfunktion *(double hashing)*
3. Viele weitere Regeln in der Literatur und Praxis



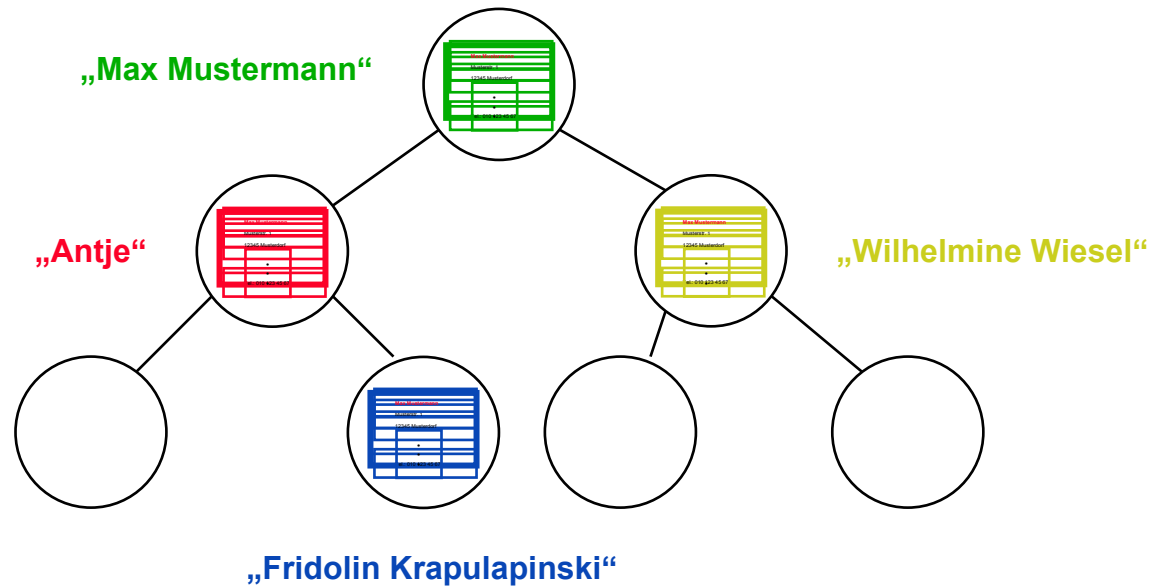
# Vergleich mit anderen Verfahren

Hashtabellen



*Was ist besser ?*

Suchbäume



# Vergleich mit anderen Verfahren

m = 1000

n = 500

n = 1000

n = 2000

n = 1 000 000

	<b>Suchbäume</b> (mit n Einträgen)	<b>Hashtabellen</b> (mit n Einträgen und m Hashplätze)	
<b>Speicherplatz</b>	$O(n)$ ≈ 500 ≈ 1000 ≈ 2000	$O(m+n)$ ≈ 1500 ≈ 2000 ≈ 2500	<b>Verbesserung durch Offenes Hashing</b>
<b>durchschnittliche Laufzeit einer Operation</b> (Suchen / Einfügen / Entfernen)	$O(\log n)$ ≈ 9 ≈ 10 ≈ 11 ≈ 20	$O(n/m)$ ≈ 1,2 ≈ 1,3 ≈ 2,1 ≈ 1000	
<b>Anwendbarkeit</b>	für beliebig viele Daten	nur für konstant viele Daten sinnvoll (n ≈ m)	<b>Verbesserung durch Dynamisches Hashing</b>
<b>Einsatz - empfehlung</b>	bei überwiegendem <b>Einfügen und Entfernen</b>	bei überwiegendem <b>Suchen</b>	

# Algorithmik 3

## Implementierung von Dictionaries

Ein Dictionary ist eine Datenstruktur für mit einem Schlüssel vergleichbare Elemente, welche die Funktionen member (key), insert (key, newdata) und delete (key) zur Verfügung stellt

### 3.1 Hashing

Datentyp: Indiziertes Array mit  $m$  Speicherplätzen

- Arbeitsweise:
- Es gibt hash-Funktion  $h: \text{Keys} \rightarrow \{0, \dots, m-1\}$
  - Jedes Element wird auf Position  $h(k)$  abgespeichert, wenn diese Position noch frei ist (wobei  $k$  der Schlüssel des Elements ist)
  - Wenn die Position  $h(k)$  besetzt ist, muss eine Kollisionsvermeidung gemacht werden (verschiedene Strategien)

Alle 3 Dictionary-Funktionen      Laufzeit  $\Theta(n)$  w.c. und  $\Theta(n/m)$  a.c.  
⇒ für  $n \in O(m)$ : Laufzeit  $\Theta(1)$  a.c.

### Referenzen zum Nacharbeiten und Vertiefen:

Cormen, Kap. 11

# Algorithmik 3

## Implementierung von Dictionaries

Ein Dictionary ist eine Datenstruktur für mit einem Schlüssel vergleichbare Elemente, welche die Funktionen member (key), insert (key, newdata) und delete (key) zur Verfügung stellt

### 3.1 Hashing: Kollisionsvermeidung

Datentyp: Indiziertes Array mit m Speicherplätzen

#### Hashlisten

- An Position  $h(k)$  wird anstelle des Datums ein Pointer auf eine lineare Liste vorgehalten. Alle Daten, die auf  $h(k)$  abgebildet werden, werden sequentiell in die Liste eingefügt.

#### Offenes Hashing

- Wenn die Position  $h(k)$  besetzt ist, wird mit spezieller Sondierungsregel eine andere Position bestimmt
- Für die Sondierungsregel gibt es viele Strategien

### Referenzen zum Nacharbeiten und Vertiefen:

Cormen, Kap. 11