Karlsruhe Institute of Technology

# Tutorium 8: Wiederholungstutorium

Matthias Schimek | 23. Juni 2017

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS17

# Gliederung

# O-Kalkül Definitionen

$g \in \mathcal{O}(f)$:

- $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \leq c \cdot f(n)$

$g \in o(f)$:

- $\forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \leq c \cdot f(n)$

$g \in \Omega(f)$:

- $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \geq c \cdot f(n)$

$g \in \omega(f)$:

- $\forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \geq c \cdot f(n)$

$g \in \Theta(f)$?

# O-Kalkül Definitionen

$g \in \mathcal{O}(f)$:

- $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \leq c \cdot f(n)$

$g \in o(f)$:

- $\forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \leq c \cdot f(n)$

$g \in \Omega(f)$:

- $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \geq c \cdot f(n)$

$g \in \omega(f)$:

- $\forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0\ :\ g(n) \geq c \cdot f(n)$

$g \in \Theta(f)$?

# O-Kalkül - Aufgaben

Beweise oder widerlege:

- $n^2 \in o(n^3)$
- $n^3 \in \Omega(n^2)$
- $n \in \Theta(\sqrt{n})$,
- $3^n \in \mathcal{O}(2^n)$

# Mastertheorem

Mittels *Mastertheorem* lassen sich folgende Rekurrenzen lösen:

- $a, b, c$ und $d$ positive Konstanten, $n \in \mathbb{N}$

$$T(n) = \begin{cases} a & \text{für } n = 1 \\ d \cdot T(n/b) + c \cdot n & \text{für } n > 1 \end{cases}$$

- Es gilt:

$$T(n) \in \begin{cases} \Theta(n) & \text{für } d < b \\ \Theta(n \cdot \log n) & \text{für } d = b \\ \Theta(n^{\log_b d}) & \text{für } d > b \end{cases}$$

# Mastertheorem

Mittels *Mastertheorem* lassen sich folgende Rekurrenzen lösen:

- $a, b, c$ und $d$ positive Konstanten, $n \in \mathbb{N}$

$$T(n) = \begin{cases} a & \text{für } n = 1 \\ d \cdot T(n/b) + c \cdot n & \text{für } n > 1 \end{cases}$$

- Es gilt:

$$T(n) \in \begin{cases} \Theta(n) & \text{für } d < b \\ \Theta(n \cdot \log n) & \text{für } d = b \\ \Theta(n^{\log_b d}) & \text{für } d > b \end{cases}$$

# Mastertheorem - Aufgaben

Gib möglichst scharfe asymptotische Schranken mit Hilfe des Mastertheorems an:

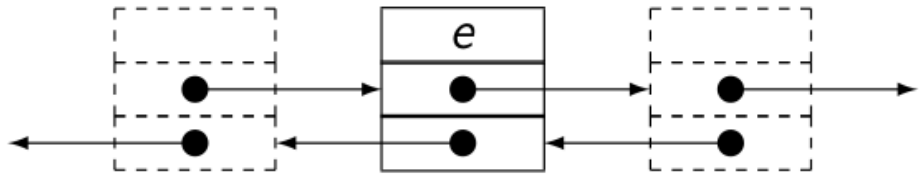
i) $A(1) = 1$ und für $n \in \mathbb{N}_{>1}$ : $A(n) = 5 \cdot A(\lceil n/5 \rceil) + 12 \cdot n$

ii) $B(1) = 1$ und für $n \in \mathbb{N}_{>1}$ :
$B(n) = 2 \cdot B(\lceil n/4 \rceil) + n + 3 \cdot \log n + 10$

# Listen - doppelt verkettet

```
Class Handle = Pointer to Item

Class Item of Element                    // one link in a doubly linked list
    e : Element
    next : Handle                        //
    prev : Handle
    invariant next→prev = prev→next = this
```




[1]

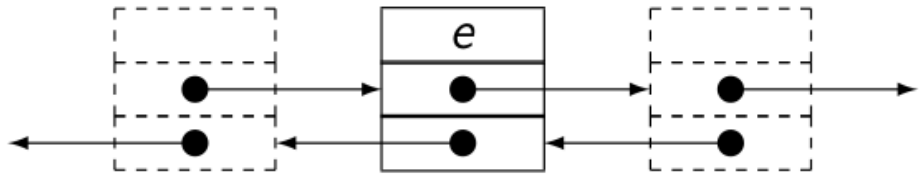
- Problem: Vorgänger erstes Element? Nachfolger von letztem Element?
- Lösung: *dummy header*
- **Aufgabe** in Pseudocode Item *C* in Liste zwischen Item *A* und Item *B* einfügen

[1] Quelle: Vorlesungsfolien, Algo I, KIT

# Listen - doppelt verkettet

```
Class Handle = Pointer to Item

Class Item of Element                    // one link in a doubly linked list
    e : Element
    next : Handle                        //
    prev : Handle
    invariant next→prev = prev→next = this
```

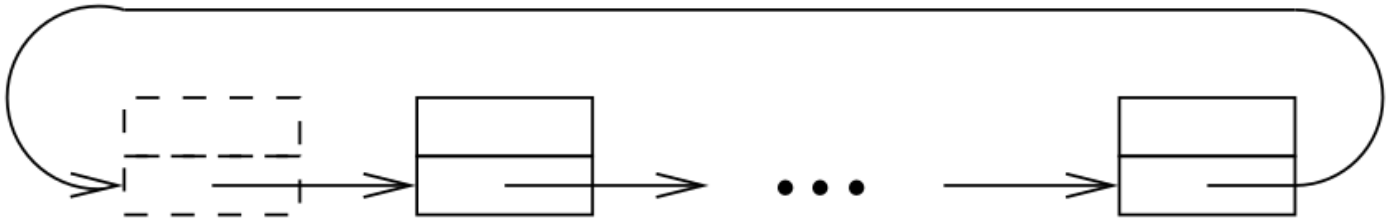



[1]

- Problem: Vorgänger erstes Element? Nachfolger von letztem Element?
- Lösung: *dummy header*
- **Aufgabe** in Pseudocode Item *C* in Liste zwischen Item *A* und Item *B* einfügen

[1] Quelle: Vorlesungsfolien, Algo I, KIT

# Listen - einfach verkettet



[2]

- Nur ein Zeiger
- Dummy Header
- Zeiger auf letztes Element für *pushBack*

[2] Quelle: Vorlesungsfolien, Algo I, KIT

# Unbounded Array

Unbounded Array = Bounded Array mit *Füllstand*

- $w$ allozierte Größe des Bounded Arrays
- $n$ Anzahl der gespeicherten Elemente im Array
- $\alpha$ Konstante $> 1$
- *Invariante:* $n \leq w < \alpha \cdot n$

$\Rightarrow$ Wenn zu voll $\rightarrow$ umkopieren in größeres Array

$\Rightarrow$ Wenn zu leer $\rightarrow$ umkopieren in kleineres Array

Wann *genau* wird umkopiert?

# Unbounded Array

Unbounded Array = Bounded Array mit *Füllstand*

- $w$ allozierte Größe des Bounded Arrays
- $n$ Anzahl der gespeicherten Elemente im Array
- $\alpha$ Konstante $> 1$
- *Invariante:* $n \leq w < \alpha \cdot n$

$\Rightarrow$ Wenn zu voll $\rightarrow$ umkopieren in größeres Array

$\Rightarrow$ Wenn zu leer $\rightarrow$ umkopieren in kleineres Array

Wann *genau* wird umkopiert?

# Hashtabelle

- Speichere Menge $M \subset$ Universum
- $key(e)$ eindeutig für $e \in M$
- unterstützt folgende Operationen in $mathcalO(1)$:
  - $M.insert(e)$ : $M := M \cup \{e\}$
  - $M.remove(k : key)$ : $M := M \setminus \{e\},\ key(e) = k$
  - $M.find(k : key)$ : return $e \in M$ with $key(e) = k$; $\perp$ falls $e \notin M$

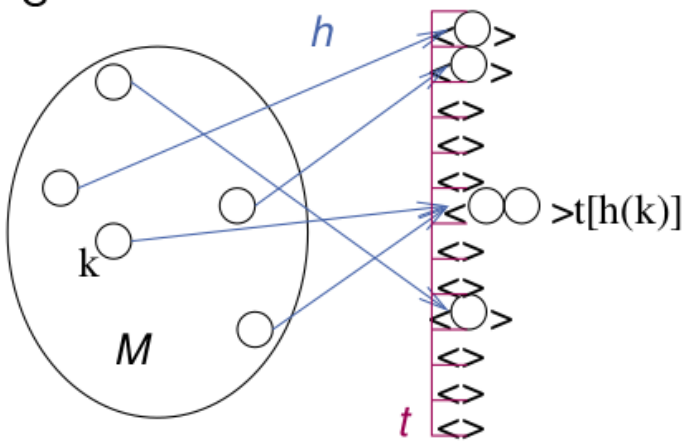
# Hashtabellen - Verkettete Listen

Implementiere die Folgen in den Tabelleneinträgen durch einfach verkettete Listen

insert($e$): Füge $e$ am Anfang von $t[h(\text{key}(e))]$ ein.

remove($k$): Durchlaufe $t[h(k)]$.
Element $e$ mit $\text{key}(e) = k$ gefunden?
$\rightsquigarrow$ löschen und zurückliefern.

find($k$) : Durchlaufe $t[h(k)]$.
Element $e$ mit $\text{key}(e) = k$ gefunden?
$\rightsquigarrow$ zurückliefern.
Sonst: $\perp$ zurückgeben.




3

[2] Folien 'Algorithmen I', KIT

# Hashtabellen - Linear Probing

*insert* : axe, chop, clip, cube, dice, fell, hack, hash, lop, slash

| | an | bo | cp | dq | er | fs | gt | hu | iv | jw | kx | ly | mz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *tt* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | ⊥ | ⊥ | ⊥ | ⊥ | axe | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| | ⊥ | ⊥ | chop | ⊥ | axe | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | dice | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | dice | ⊥ | ⊥ | ⊥ | ⊥ | fell | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | dice | ⊥ | ⊥ | ⊥ | hack | fell | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | dice | hash | ⊥ | ⊥ | ⊥ | fell | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | dice | hash | lop | ⊥ | hack | fell | ⊥ |
| | ⊥ | ⊥ | chop | clip | axe | cube | dice | hash | lop | slash | hack | fell | ⊥ |

*remove* ⇩ clip

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ⊥ | ⊥ | chop | ~~clip~~ | axe | cube | dice | hash | lop | slash | hack | fell | ⊥ |
| | ⊥ | ⊥ | chop | lop | axe | cube | dice | hash | ~~lop~~ | slash | hack | fell | ⊥ |
| | ⊥ | ⊥ | chop | lop | axe | cube | dice | hash | slash | ~~slash~~ | hack | fell | ⊥ |
| | ⊥ | ⊥ | chop | lop | axe | cube | dice | hash | slash | ⊥ | hack | fell | ⊥ |

# Hashing

Es sei die Hashfunktion $h : \mathbb{N} \to \{0...13\},\ h(x) = x \bmod 14$ und eine Hashtabelle mit 14 Einträgen gegeben.
Füge die Zahlen
$75, 44, 46, 53, 14, 2, 40, 61, 87, 86, 13, 28$ mittels

- Hashing mit verketteten Listen
- Hashing mit linearer Suche

ein

# Sortieralgorithmen

**Sortieralgorithmen:**

- Insertionsort
- Quicksort
- Mergesort
- Heapsort
- Bogosort

Funktionsweise? Laufzeit?

# Aufgabe

Gegeben: Array mit $n$ Zahlen $a_i$ und $0 \leq a_i \leq (n-1)$.
Sortiert das Array in $\mathcal{O}(n)$ Zeit

# Ganzzahliges Sortieren

**Satz** Vergleichsbasiertes Sortieren benötigt $\Omega(n \log n)$ Vergleiche

- Geht es anders?
- Wie funktioniert Bucketsort?
- Wie funktioniert LSD-Radixsort
- Was ist der unterschied zu MSD-Radixsort?

# Ganzzahliges Sortieren

**Satz** Vergleichsbasiertes Sortieren benötigt $\Omega(n \log n)$ Vergleiche

- Geht es anders?
- Wie funktioniert Bucketsort?
- Wie funktioniert LSD-Radixsort
- Was ist der unterschied zu MSD-Radixsort?