FernUniversität in Hagen

Klausurvorbereitung
Kurs 1661 „Datenstrukturen I“

11.Juli 2015

Dirk Friedenberger

Agenda

- Algebren und Abstrakte Datentypen
- Hashing und AVL-Bäume
- Sortieralgorithmen
- Graphen

Algebra

Syntax

- Signatur
  - Objektmengen
  - Operationen

algebra *intset*
sorts *intset, int, bool*
ops

empty:		$\rightarrow$ *intset*
insert:	*intset* $\times$ *int*	$\rightarrow$ *intset*
delete:	*intset* $\times$ *int*	$\rightarrow$ *intset*
contains:	*intset* $\times$ *int*	$\rightarrow$ *bool*
isempty:	*intset*	$\rightarrow$ *bool*

Semantik

- Trägermengen
- Funktionen

sets $intset = F(\mathbb{Z}) = \{M \subset \mathbb{Z} \mid M \text{ endlich}\}$
functions

empty	$= \emptyset$
insert (M, i)	$= M \cup \{i\}$
delete (M, i)	$= M \setminus \{i\}$
contains (M, i)	$= \begin{cases} true & \text{falls } i \in M \\ false & \text{sonst} \end{cases}$
isempty (M)	$= (M = \emptyset)$

end *intset*.

Abstrakter Datentyp

Syntax

- Signatur
  - Objektmengen
  - Operationen

algebra *intset*
sorts *intset, int, bool*
ops

empty:		→ *intset*
insert:	*intset* × *int*	→ *intset*
delete:	*intset* × *int*	→ *intset*
contains:	*intset* × *int*	→ *bool*
isempty:	*intset*	→ *bool*

Semantik

- Axiome

axs

isempty (*empty*)	= *true*
isempty (*insert* (*x*, *i*))	= *false*
insert (*insert* (*x*, *i*), *i*)	= *insert* (*x*, *i*)
contains (*insert* (*x*, *i*), *i*)	= *true*
contains (*insert* (*x*, *j*), *i*)	= *contains* (*x*, *i*) $(i \neq j)$
...	

end *intset*.

Übungsausgabe

- Abstrakte Datentypen, HK 2008 Aufgabe 1

Aufgabe 1 Spezifikationen

Sei K eine Menge, $\oplus$: $K \times K \to K$ und $\otimes$: $K \times K \to K$ seien zwei Funktionen auf K. (K, $\oplus$, $\otimes$, n, e) heißt *Körper* über K, wenn gilt:

1. $\oplus$ ist assoziativ und kommutativ,
2. $\otimes$ ist assoziativ und kommutativ,
3. n (genannt „Nullelement") ist das neutrale Element bezüglich $\oplus$,
4. e (genannt „Einselement") ist das neutrale Element bezüglich $\otimes$,
5. zu jedem Element a aus K existiert ein bezüglich $\oplus$ inverses Element in K (das „Negative" zu a),
6. zu jedem Element a aus K gibt es ein bezüglich $\otimes$ inverses Element in K (den „Kehrwert" zu a),
7. für alle Elemente a, b, c aus K gilt das Distributivitätsgesetz: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$.

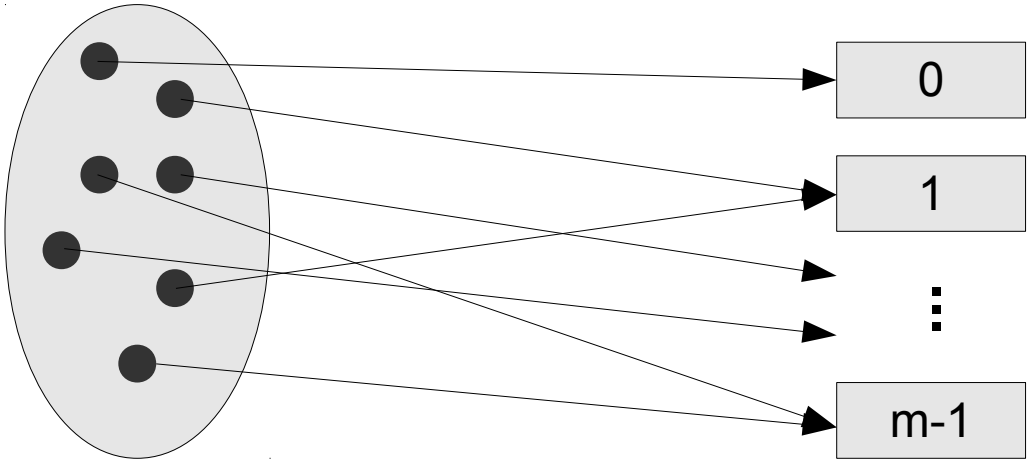
(a) Geben Sie eine Spezifikation für Körper über beliebige, frei bleibende Mengen K als abstrakten Datentyp an.

(b) Warum können Sie hier stattdessen keine algebraische Spezifikation vornehmen?

Hashing

- Eine Hashfunktion ist eine totale Abbildung

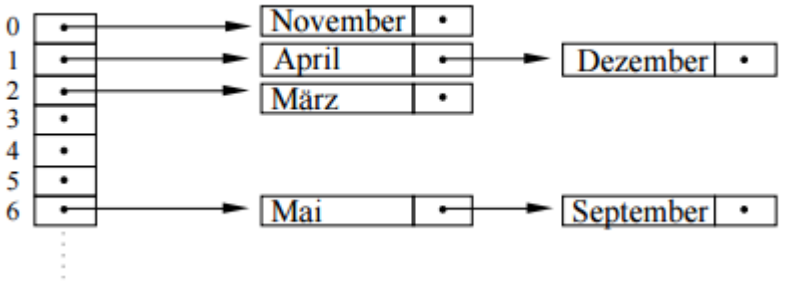
$$h : D \rightarrow \{0, \ldots, m-1\}$$



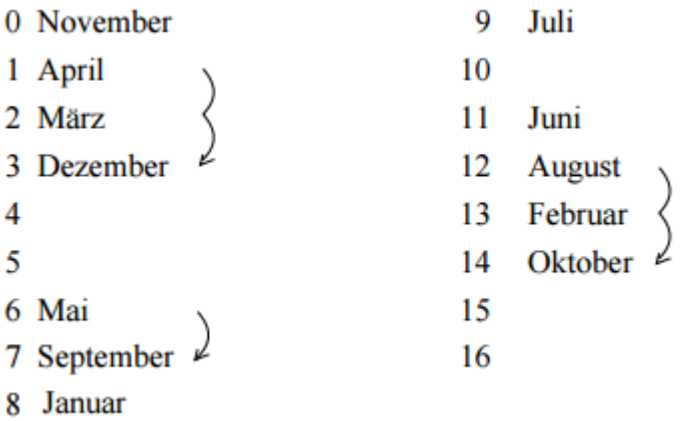
- Möglichst surjektiv
- Möglichst geichmäßig verteilend
- Möglichst effizient

Hashing

- Offenes Hashing

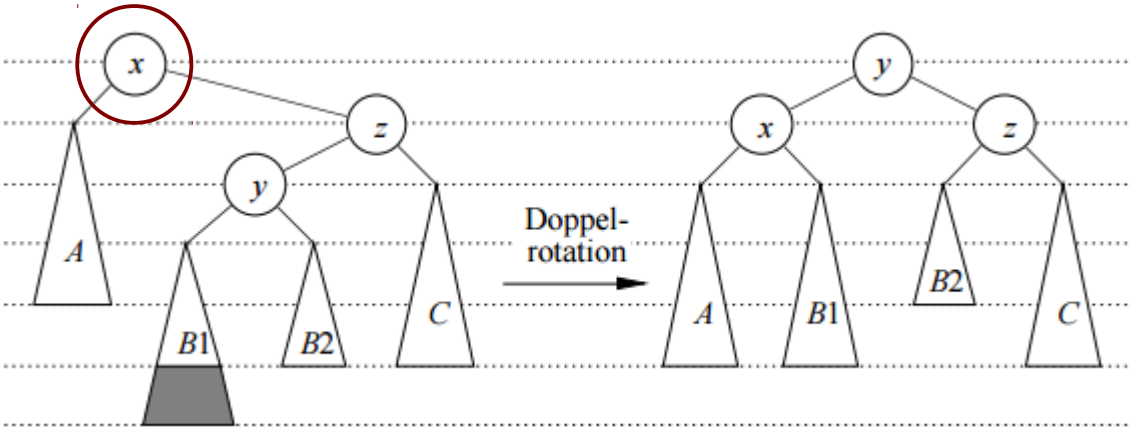


- Geschlossenes Hashing
  - Benötigt Kollisionsstrategie (lineares Sondieren, Quadratisches Sondieren, Doppelhashing)



AVL-Bäume

Ein AVL-Baum ist ein binärer Suchbaum, in dem sich für jeden Knoten die Höhen seiner zwei Teilbäume höchstens um 1 unterscheiden.



Ist bei einem aus der Balance geratenen Knoten

- ein äußerer Teilbaum am tiefsten -> Rotation
- der mittlere Teilbaum am tiefsten -> Doppelrotation

Übungsausgabe

- Hashing, NK 2006 Aufgabe 3

Aufgabe 3 Hashing

(a) Der Datentyp *dictionary* soll mittels offenem Hashing mit m Behältern implementiert werden. Implementieren Sie Algorithmen für die Operationen *insert*, *delete* und *member*. Dabei soll eine Reorganisation der Hashtabelle stattfinden, sobald $n / m > c$ gilt (n: Anzahl der im Dictionary gespeicherten Elemente, c: eine beliebige Konstante). Geben Sie zusätzlich eine Funktion t für die durchschnittliche Laufzeit der Operation *member* an, wenn sich Anfragen mit der Wahrscheinlichkeit p auf einen in der Hashtabelle gespeicherten Wert beziehen.

Benutzen Sie in den Algorithmen die folgenden Typen, Variablen und Konstanten:

type *listelem* = **record** *value*: *elem*; *next*: ↑*listelem* **end**;
type *set* = ↑**array of** ↑*listelem*;
const *c*; **var** *n*, *m*;

(b) Beim geschlossenen Hashing mit m Behältern, die genau ein Element aufnehmen können, ist die Kollisionswahrscheinlichkeit P_k ein bedeutender Faktor für die durchschnittliche Laufzeit der *dictionary* Operationen. Leiten Sie, unter der Annahme, daß die Hashfunktion ideal ist, eine Formel für P_k her.

Führen Sie eine Abschätzung durch, die es Ihnen ermöglicht, zu einer gegebenen Anzahl n ein m zu bestimmen, so daß $P_k < x$ gilt, für ein vorgegebenes $x \in]0, 1]$. Gesucht ist also eine Beziehung zwischen m, n und x.

Hinweis: Bei einer idealen Hashfunktion ist die Wahrscheinlichkeit $P(i)$ für das Ereignis, daß der i-te Schlüssel auf einen freien Behälter abgebildet wird gegeben durch $P(i) = (m - i + 1) / m$. Ersetzen Sie bei der Berechnung an geeigneter Stelle einen Term durch eine möglichst große untere Schranke, die es Ihnen erlaubt, die entstehende Ungleichung nach m aufzulösen.

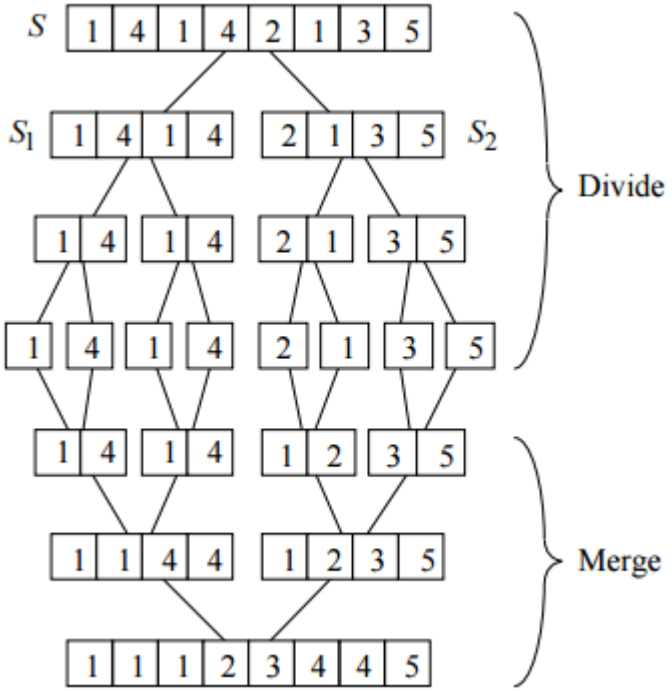
Devide-and-Conquer

Mergesort

Divide: Menge S in S1, S2 aufteilen

Conquer: S1' := MergeSort (S1); S2' := MergeSort (S2);

Merge: **Merge (S1', S2')**

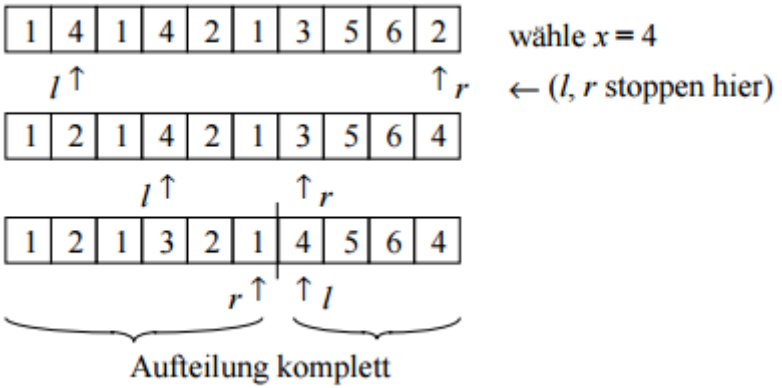


Quicksort

Divide: **Partinioniere S in S1, S2**

Conquer: S1' := QuickSort (S1); S2' := QuickSort (S2);

Merge: return concat (S1', S2')

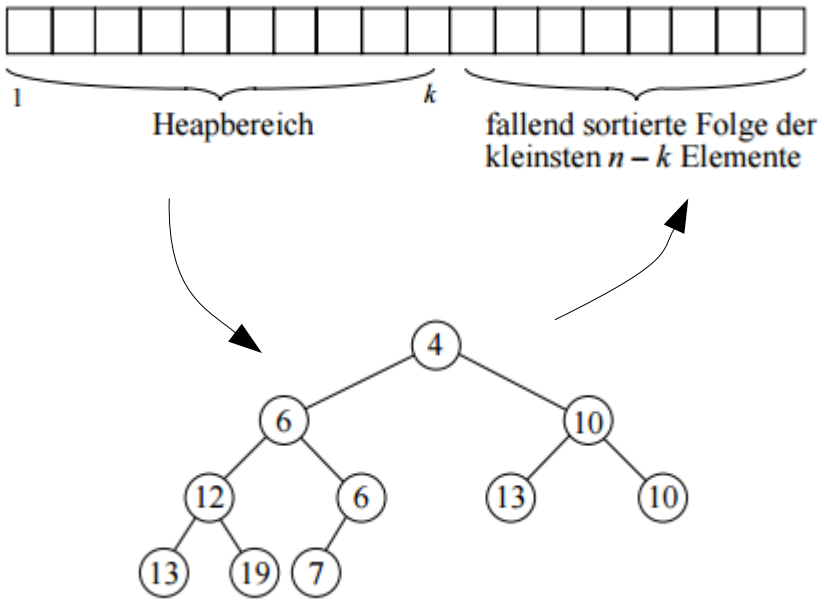


Verfeinertes Auswählen und Einfügen

Heapsort

Schritt 1: Aufbau des Heap

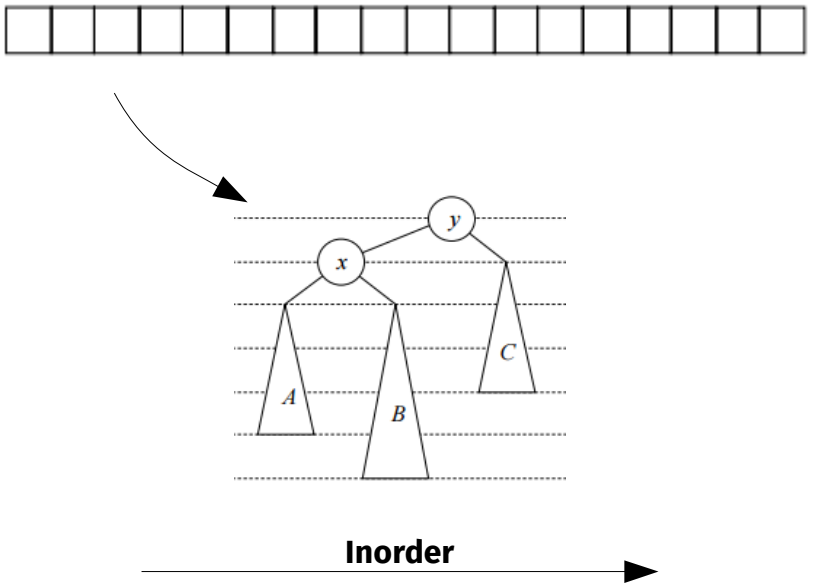
Schritt 2: Aufbau sortierter Folge



Baumsortieren

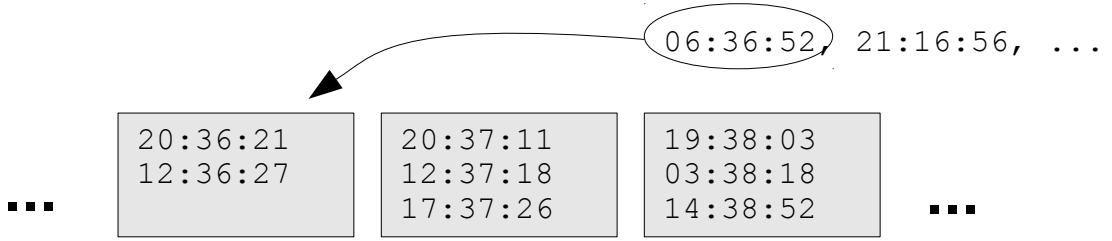
Schritt 1: Liste in AVL-Baum eintragen

Schritt 2: Inorder-Durchlauf

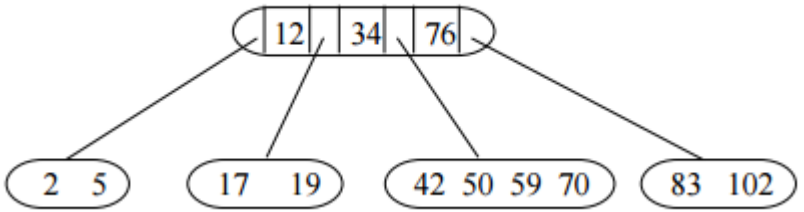


Spezialfälle

- Bucketsort und Radixsort ->Laufzeit von O(n)



- Externes Suchen: B-Bäume -> große Datenmengen



Übungsausgabe

- Radixsort, HK 2007 Aufgabe 3

Aufgabe 3 Radixsort

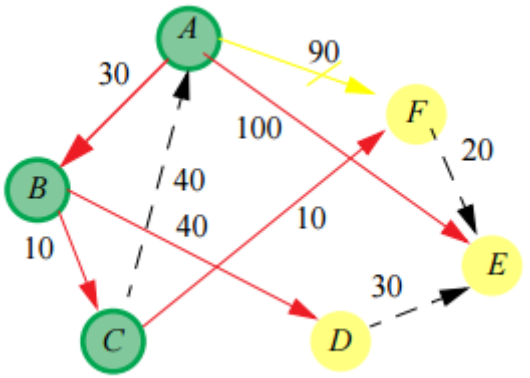
(a) Es sei *keytype* die Menge aller Zeitstempel der Form *hh*:*mm*:*ss* mit *hh* $\in$ { 00, 01, ..., 23 } und *mm*, *ss* $\in$ { 00, 01, ..., 59 }. Sortieren Sie die unten angegebene Folge mittels Radixsort in drei Phasen.

23:59:01, 13:31:14, 12:11:01, 03:27:01,17:23:14, 12:11:03, 17:15:59, 13:28:01

(b) Wenn Sie 1000 Zahlen aus dem Bereich 1 - 10^{100} sortieren sollen, würden Sie Heapsort oder Radixsort bevorzugen? Begründen Sie Ihre Antwort.

Graphen

- Graph ist ein Paar aus Knoten und Kanten G = (V, E)
- Datenstruktur: Adjazenzmatrix und Adjazenzlisten
- Algorithmen:
  - Tiefen- und Breitendurchlauf
  - Algorithmus von Dijkstra



Übungsausgabe

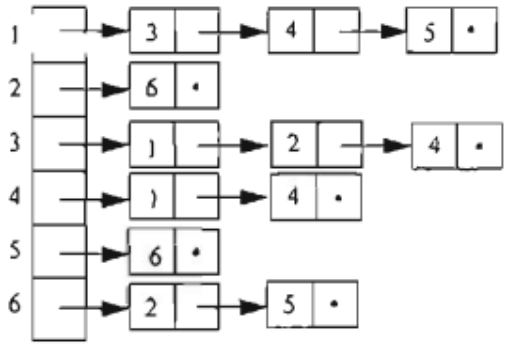
- Adjazentzmatrix und -liste, HK 2009 Aufgabe 3

Aufgabe 3 Graphen

Analysieren Sie im folgenden die Graphen G1 und G2. Wenn Sie die Wahl haben, fahren Sie mit der Bearbeitung beim Knoten mit der kleinsten Nummer fort.

	1	2	3	4	5	6
1	∞	9	∞	4	∞	5
2	9	∞	3	∞	∞	∞
3	∞	3	5	∞	6	7
4	4	∞	∞	∞	9	∞
5	∞	∞	6	9	∞	∞
6	5	∞	8	∞	∞	∞

Adjazenzmatrix G1



Adjazenzliste G2

(a) Könnten die Graphen als ungerichtete Graphen aufgefaßt werden? Wenn ja, warum?

(b) Enthält G2 Zyklen? Wenn ja, geben Sie diese bitte an.

(c) Geben Sie die stark verbundenen Komponenten von G1 und G2 an.

(d) Geben Sie folgende Spannbäume an:
- Tiefendurchlauf durch G1 ausgehend von Knoten 1.
- Breitendurchlauf durch G1 ausgehend von Knoten 1.
- Tiefendurchlauf durch G2 ausgehend von Knoten 2.
- Breitendurchlauf durch G2 ausgehend von Knoten 2.

Quellen

Ralf Hartmut Güting und Stefan Dieker. "Datenstrukturen I" Fernuniversität Hagen, 2012.

Klausuren und Musterlösungen, http://www.fernuni-hagen.de/FACHSCHINF/, (abgerufen Juli 2015)