

Einführung in die Informatik 1

– Datenorganisation und Datenstrukturen –

Sven Kosub

AG Algorithmik/Theorie komplexer Systeme
Universität Konstanz

E 202 | Sven.Kosub@uni-konstanz.de | Sprechstunde: Freitag, 12:30-14:00 Uhr, o.n.V.

Wintersemester 2008/2009

Datenstrukturen

- Konzeptionen zur Organisation von Daten
- bauen auf elementaren Datentypen auf
- für die Effizienz von Algorithmen fundamental

Grundbausteine von Datenstrukturen

- **Reihung** (*array*)
zur Verwaltung homogener Daten, d.h. für Daten, die vom gleichen Datentyp sind
- **Verbund** (*record*)
zur Verwaltung heterogener Daten, d.h. für Daten, die nicht notwendig vom gleichen Typ sein müssen

eindimensionale Reihung

- repräsentiert eine Folge von Daten gleichen Datentyps
- speichert Daten als Zeile einer Tabelle

- Repäsentierung von Funktionswerten an den Stellen i
- z.B. Messung der Signalstärke zu den Zeitpunkten $t = 1, \dots, k$

Zeitpunkt	1	2	3	4	...	30	31
Signalstärke	10,5	10,5	12,2	9,8	...	13,1	13,3

- Zugriff auf jedes Element der Folge mit gleichem Zeitaufwand, z.B. in der Form $a[i]$ auf das i -te Element
- sind die Daten vom Typ T , so ist die Reihung vom Typ T

zweidimensionale Reihung

- repräsentiert eine Folge von eindimensionalen Reihungen vom Typ T
- speichert Daten in Matrixform, deren Zeilen Reihungen sind
- Zugriff durch $a[i, j]$ auf das j -te Element der i -ten Zeile

- Temperaturverteilung über Gitterkoordinaten $t : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R}$
- Temperaturwert $t(1, 1)$ an Stelle $(1, 1)$ steht in Array t an Stelle $t[1, 1]$

- alternativer Zugriff in Java: $a[i][j]$

n -dimensionale Reihungen als Verallgemeinerung

Zeichenketten (*strings*)

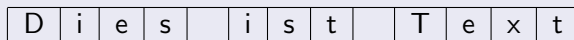
- Reihungen vom Datentyp `char` (Buchstabe)
- typischerweise spezielle Syntax für Zeichenketten

Zeichenketten in Java:

- Verwendung von Anführungszeichen, z.B.

```
"Dies ist Text"
```

- zugehöriges Speicherbild:



- alternative Notation als (Standard-)Reihung:

```
{'D','i','e','s','','i','s','t','','T','e','x','t'}
```

Verbund

- häufig Beziehungen zwischen Daten unterschiedlichen Typs zu modellieren

Stammdatenblatt s:

- `s.Name = "Mustermann"`
- `s.Vorname = "Martin"`
- `s.GebTag = 10`
- `s.GebMonat = 5`
- `s.GebJahr = 1930`
- ...

Name	"Mustermann"
Vorname	"Martin"
GebTag	10
GebMonat	05
GebJahr	1930
Familienstand	"verheiratet"
...	...

Typkombinationen

- Komponenten eines Verbundes dürfen von beliebigem Typ sein
- insbesondere: vom Typ Verbund oder Reihung
- hierarchische Datenmodellierung

Komplexe Verbünde

Name	Nachname	"Mustermann"
	Vorname	"Martin"
GebDatum	Tag	10
	Monat	05
	Jahr	1930
Familienstand	"verheiratet"	

Name	Nach	Mustermann
	Vor	Martin
Matrikel	123456	
Punkte	1	...
	2	
	14	

Wie ist ein Verbund in einem anderen Verbund enthalten?

- **Inklusion durch Wert** (*by value*)

Daten des Unterverbundes werden im Verbund gespeichert

- + vollständige Speicherung an einem Ort
- große Datenmengen pro Verbund zu speichern

- **Inklusion durch Referenz** (*by reference*)

Daten des Unterverbundes werden an einem anderen Ort gespeichert und durch einen Verweis (Zeiger, *pointer*) auf die Speicheradresse gespeichert (referenziert)

- + flexibel; mehrere Zeiger auf gleiche Speicheradresse
- (De)Referenzierung eine Hauptquelle für Fehler

Beispiel-Szenario

- Heidi Mustermann besucht Übungen zu zwei verschiedenen Vorlesungen
- jede Übung wird durch einen Verbund (Name, Punkte, etc.) realisiert
- d.h. Stammdaten für Heidi Mustermann in zwei Verbänden repräsentiert

Inklusion durch Wert

- zwei separate Exemplare des Stammdatenblatts „Heidi Mustermann“
- doppelter Speicherplatz
- bei Änderung von Stammdaten müssen alle Exemplare gesucht und geändert werden

Kurs = „Informatik 1“

Stammdaten =

Vorname = „Heidi“
Nachname = „Mustermann“
GebTag =

Punkte = 17

Kurs = „Analysis 1“

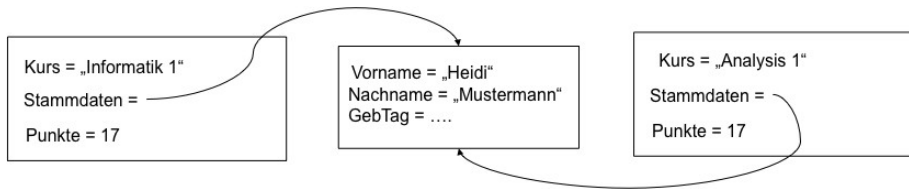
Stammdaten =

Vorname = „Heidi“
Nachname = „Mustermann“
GebTag =

Punkte = 20

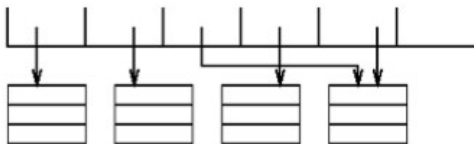
Inklusion durch Referenz

- ein Exemplar für Stammdatenblatt „Heidi Mustermann“
- beide Verbände verweisen darauf
- zusätzliche Dereferenzierung notwendig



Inklusion durch Referenz (Forts.)

- Stammdatenblätter der Übungsteilnehmer durch Referenz
- Student hat sich zweimal angemeldet



Wie sollten Inklusionen modelliert werden - durch Wert oder Referenz?

- möglichst realitätsnah, d.h. im Beispiel: es gibt nur ein Exemplar einer Person, also nur ein Stammdatenblatt
- Methodik der objektorientierten Modellierung (demnächst)

Wie sollten Inklusionen modelliert werden - durch Wert oder Referenz?

- Effizienzüberlegung als weiterer Modellierungsaspekt
- wichtige Operationen auf Reihungen: [Sortieren](#)
- Typ T kann i.A. komplex sein
- im Schnitt $\approx n \cdot \log n$ Vertauschungen (bei n Elementen)
- Inklusion durch Wert: alle Daten müssen physikalisch vertauscht werden (i.A. viele Byte pro Vertauschung)
- Inklusion durch Referenz: nur Referenzen müssen vertauscht werden (4 Byte bei 32-Bit-Prozessoren)

Wie erfolgt Zugriff auf referenzierten Verbund?

C++

- Inklusion durch Wert mittels Punktnotation, z.B.

`B[5].GebDatum.Tag`

- Inklusion durch Referenz mittels Pfeilnotation, z.B.

`B[5]->GebDatum.Tag`

Java

- ausschließlich Inklusion durch Referenz möglich
- Punktnotation wird verwendet

Verbundhierarchien:

- Verbünde dürfen rekursiv geschachtelt sein, d.h. Verbund hält Referenz auf Verbünde mit den gleichen Komponenten
- typische Verwendung: Verwaltung dynamischer Datensammlungen, Navigation

Spezialfälle der rekursiven Referenzierung:

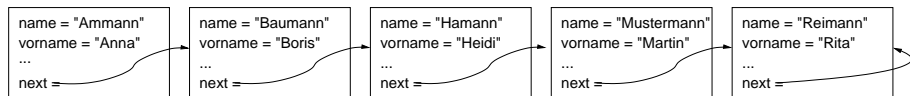
- **Listen**
- **Bäume**

Liste:

- lineare Form der Datenorganisation
- erster Verbund der Liste heißt **Kopf** (`head`)
- jeder Verbund der Liste mit Referenz auf einen **Nachfolger**verbund (`next`)
- jeder Verbund hat höchstens einen Verbund, der auf ihn referenziert
- letzter Verbund der Liste referenziert auf sich selbst oder auf speziellen Bezeichner für den **leeren Verbund** (`nil`)

Suchliste mit den studentischen Stammdaten:

- Verbände der Liste enthalten Komponenten `name`, `vorname`, `...`, `next`
- Verbände in Liste aufsteigend lexikographisch nach `name` sortiert:
 - links von jedem Verbund sind nur Verbände mit kleinerem `name` gespeichert
 - rechts von jedem Verbund sind nur Verbände mit größerem `name` gespeichert



Algorithmus zum Suchen in Suchliste:

Algorithmus: `listSearch(stud)`

Eingabe: `stud` ist vom Verbundstyp der Liste

Ausgabe: „Gefunden!“, falls Student mit Namen `stud.name` in Liste
enthalten; „Nicht gefunden!“ sonst

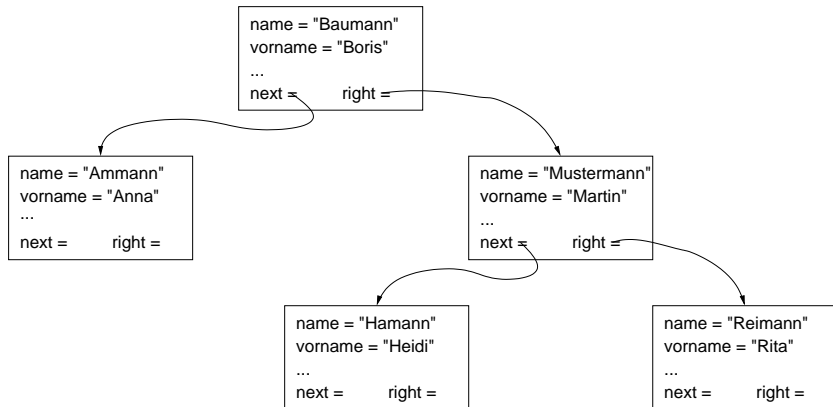
```
(1) s=head;  
(2) while (s≠nil und s.name≠stud.name) {  
(3)   s=s.next;  
(4) }  
(5) if (s≠nil) { return(„Gefunden!“) }  
(6) else { return(„Nicht gefunden!“) }
```

(binärer) Baum:

- zweidimensionale Form der Datenorganisation
- oberster Verbund des Baumes heißt **Wurzel** (root)
- jeder Verbund des Baumes mit Referenzen auf einen **linken Kindverbund** (left) und einen **rechten Kindverbund** (right)
- jeder Verbund des Baumes hat höchstens einen Verbund, der auf ihn referenziert
- unterste Verbünde des Baumes referenziert auf sich selbst oder auf speziellen Bezeichner für den **leeren Verbund** (nil)

Suchbaum mit den studentischen Stammdaten:

- Verbände des Baumes enthalten Komponenten `name`, `vorname`, `...`, `left`, `right`
- Verbände des Baumes wie folgt nach `name` sortiert:
 - links unterhalb jedes Verbundes sind nur Verbände mit kleinerem `name` gespeichert,
 - rechts unterhalb jedes Verbundes sind nur Verbände mit größerem `name` gespeichert



Algorithmus zum Suchen im Suchbaum:

Algorithmus: `treeSearch(stud)`

Eingabe: `stud` vom Typ `Student`

Ausgabe: „Gefunden!“, falls `Student` mit Namen `stud.name` in Baum enthalten, „Nicht gefunden!“ sonst

```
(1) s=root;
(2) while (s≠nil und s.name≠stud.name) {
(3)     if (s.name≤stud.name) { s=s.right; }
(4)     else { s=s.left; }
(5) }
(6) if (s≠nil) { return(„Gefunden!“) }
(7) else { return(„Nicht gefunden!“) }
```

Abstrakte Datentypen und Objekte

Abstrakte Datentypen (ADT):

- Einheit von Datenstrukturen und Manipulationsoperationen
- Zugriff nur über eine abstrakte Aufrufchnittstelle
- Interna bleiben verdeckt (*information hiding*)

Abstrakter Datentyp für komplexe Zahlen

- interne Information:

`re, im, ...`

- öffentliche Information:

`getRe(), getIm(), set(re,im), getAngle(), getRadius(),
setPolar(angle,radius), add(comp), ...`

- Java unterstützt ADT durch Zwang
- C/C++ überlässt Programmierern Verwendung von ADT