

# Index- und Speicherungsstrukturen

Bernhard Seeger

Fachgebiet Informatik  
Fachbereich Mathematik  
Universität Marburg

email: seeger informatik.uni-marburg.de

Tel.: 28-1526

Sprechstunde: nach Vereinbarung

---

1.

## Übersicht

1. Einführung  
Speicherarchitekturen, Dateisysteme, Pufferorganisation, Seitenformate
2. Eindimensionale Zugriffsstrukturen  
statische u. dynamische Hashverfahren,  
B+-Bäume und viele Varianten
3. Mehrdimensionale Zugriffsstrukturen  
a) für Hauptspeicher: kd-Baum, Quadranten-Baum  
b) Verallgemeinerungen des B+-Baums  
c) Mehrdimensionale Hashverfahren
4. Anwendungen von Zugriffsstrukturen  
a) Historische Datenverwaltung  
b) Text-Datenbanken  
c) Ähnlichkeitssuche  
d) Geo-Datenbanken  
e) OO-Datenbanksysteme  
...
5. Integration in Datenbanksysteme  
Mehrbenutzerbetrieb von B+-Bäumen

---

3.

# Organisation

- ☐ Voraussetzungen
  - gute Kenntnisse im Bereich Datenstrukturen
- ☐ Literatur
  - es gibt derzeit kein empfehlenswertes Buch!!
  - Ordner in der Bibliothek enthält aber viele wichtigsten Originalarbeiten
- ☐ Übungen
  - zweiwöchentlich
  - geplant sind: 3 Implementierungs- und 3 Blattaufgaben
  - Implementierungssprache: C
- ☐ Scheinkriterien
  - ohne Note:  
50% der Blattaufgaben und 2 von 3 Implementierungsaufgaben
  - mit Note:  
zusätzlich ein Kolloquium (Dauer: 20-20 Minuten)

---

2.

## 1. Einführung

allgemeine und häufig auftretende Problemstellung:

- ☐ Gegeben:
    - homogene Menge  $O$  von Objekten/Datensätzen
    - Prädikat  $p: O \rightarrow \{\text{true}, \text{false}\}$
  - ☐ Gesucht:
    - Menge aller Objekte aus  $O$ , die  $p$  erfüllen:
$$\{o \mid o \in O \wedge p(o)\}$$

- ☐ *Datensätze* in einer homogenen Menge
  - gleichen Datensatztyp
- ☐ *Datensatztyp* besteht aus verschiedenen *Attributen*
  - Annahme: der Datentyp eines Attributs ist *einfach*, z. B. *INTEGER*

---

4.

## Beispiel

- ❑ Datensatztyp:  
 TYPE LinieHalt = RECORD  
     Linie: INTEGER;  
     Haltestelle: STRING(80);  
     Zeit: INTEGER;  
 END;
- ❑ Menge der Datensätze:  
 (7, Hans-Meerwein-Str., 516), (5, Wilhelmsplatz, 615), ...
- ❑ Anfragen:
  - Finde alle Linien, die an der Haltestelle Hans-Meerwein-Str. halten.
  - Finde alle Linien, die zwischen 8 und 10 Uhr am Wilhelmsplatz halten.

### Lösungsvorschlag

- ❑ Durchlaufe alle Datensätze in der Menge und teste das Prädikat
- ❑ Funktioniert zwar, aber sehr teuer

5.

## Anforderung an Zugriffsstrukturen

### effiziente Unterstützung von Suchoperationen:

- ❑ Suchoperationen:
  - **Exact Match Query** ( $x_1, \dots, x_k$ )  
spezifiziert k Attribute exakt.
  - **Partial Match Query** ( $*, x_{i1}, *, \dots, x_{i2}, \dots, x_{is}$ )  
spezifiziert Werte für  $s < k$  Attribute; k-s Attribute bleiben unspezifiziert (\*).
  - **Range Query** ( $[u_1, o_1], \dots, [u_k, o_k]$ )  
spezifiziert k Bereiche mit  $u_i \leq o_i, 1 \leq i \leq k$ .
  - **Partial Range Query** ( $*, [u_{i1}, o_{i1}], *, \dots, [u_{is}, o_{is}]$ )
- ❑ weitergehende Suchoperationen
  - **k-nächste Nachbarn**  
Zu  $X = (x_1, \dots, x_k)$  und einer Metrik  $d(.,.)$  sind die k Datensätze aus der Menge O gesucht, die minimalen Abstand zu X haben.
- ❑ mögliche Nebenbedingungen:
  - **sortierte** Ausgabe (bzgl. einem oder mehreren Attributen)

7.

## Indexstrukturen

- ❑ effiziente Datenstrukturen zur Unterstützung von "einfachen" Suchoperationen.
- ❑ i.a. werden nur Anfragen bzgl. bestimmter Attribute unterstützt
  - Anzahl dieser Attribute = 1: *eindimensionale Zugriffsstrukturen*
  - Anzahl > 1: *mehrdimensionale Zugriffsstrukturen*
- ❑ Datensatz: (K, info), *Schlüssel*  $K = (K_1, K_2, \dots, K_d)$ ,  $d = \text{Dimension}$
- ❑ Beispiele:
  - binäre Suchbäume (AVL-Baum, 2-3 Baum), Hashtabellen

### in dieser Vorlesung

- ❑ Indexstrukturen für **Externspeicher**
  - eine homogene Datenmenge wird in einer Datei abgespeichert.
- ❑ Unterschied zu Hauptspeicherzugriffsstrukturen auf Grund der Eigenschaften von Externspeicher:
  - niedrige Speicherkosten: für 1 DM gibt es 3 MB Plattenspeicher
  - hohe Zugriffskosten, niedrige Transferkosten  
 ==> CPU-Kosten können (nahezu) vernachlässigt werden.

6.

### Dynamisches Einfügen, Löschen und Verändern von Datensätzen

- ❑ globale Reorganisation der gesamten Datenmenge ist oft nicht akzeptabel.
- ❑ *Beispiel: sortierte sequentielle Datei*
  - Das Einfügen eines Datensatzes erfordert im schlechtesten Fall, daß alle Datensätze um eine Position verschoben werden müssen.
  - Folge: auf alle Seiten der Datei muß zugegriffen werden.
 Das Einfügen, Löschen und Verändern von Datensätzen darf daher nur **lokale Änderungen** bewirken ("think global act local")

### Komplettaufbau einer Zugriffsstruktur

- ❑ zu einer gegebenen Datenmenge O soll eine Zugriffsstruktur aufgebaut werden.
  - einfaches Hintereinanderausführen der gewöhnlichen Einfügeoperation ist ineffizient
- Kosteneinsparung durch Vorverarbeitung** der Datenmenge (z. B. Sortieren)

8.

## Hohe Speicherplatzausnutzung

- ❑ Datenmengen können sehr groß werden.
- ❑ Eine möglichst *hohe Speicherplatzausnutzung* ist wichtig:
  - Speicherkosten für die Zugriffsstruktur << Speicherkosten für die Datenmenge.
  - Speicherplatzausnutzung der Datei soll hoch sein. (Datei soll möglichst gut gefüllt sein)

## Leichte Integration in bestehende Systeme (z. B. DBS)

- ❑ Verwendung bereits vorliegender Schnittstellen (z. B. zum Dateisystem)
- ❑ obere Schnittstelle muß kompatibel zu den Systemanforderungen sein  
==> einige Optimierungstechniken sind dann teilweise nicht möglich

## Weitere Anforderungen

- ❑ **Implementierbarkeit:** je einfacher die Struktur desto besser ist ihre Akzeptanz.
- ❑ **Robustheit:** gutes Verhalten im worst-case.
- ❑ **Implementierungskosten-Nutzen** Verhältnis der Zugriffsstrukturen sollte möglichst niedrig sein.

9.

- ❑ Speicherung auf **Externspeicher**
  - i. a. Magnetplattenspeicher
  - Bänder und optische Platten
- ❑ nur kleiner Teil der Daten im *Hauptspeicher*:
  - i.a. keine Persistenz
  - Hauptspeicher sind zu klein
  - Kosten

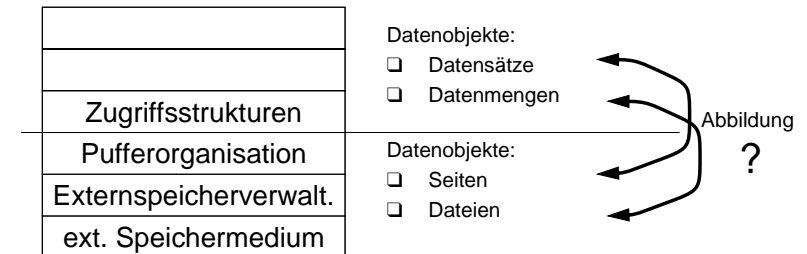
Es ist i.a. nicht effektiv, eine Datei vollständig vom Externspeicher in den Hauptspeicher zu laden, um beispielsweise nur eine Anfrage zu beantworten

11.

## 1.1 Physische Speicherung von Daten

### Anforderungen

- ❑ Persistenz
  - dauerhafte Speicherung der Daten, d.h. Daten überdauern das Programmende.
- ❑ Verwaltung sehr großer Datenmengen
  - Datenvolumen im GigaByte-Bereich.



10.

### 1.1.1 Aufbau eines Magnetplattenspeichers

- ❑ ein Stapel übereinanderliegender rotierender Magnetplatten.
- ❑ Strukturierung:
  - Zylinder, Spur und Sektor (Seite)
  - Zugriff über einen Kamm mit Schreib-/Leseköpfen zugegriffen, der quer zur Rotation bewegt wird

### Zugriff auf Seiten

- *Positionierung des Schreib-/Lesekopfes*  
Ø Zeit für die Armbewegung [10 ms]
- *Warten auf den Sektor / Seite* (Rotationsverzögerung)  
Ø halbe Rotationszeit der Platte [5,5 ms]
- *Übertragung der Seite (Transferzeit)*  
Zeit für Schreiben bzw. Lesen einer 4 KByte Seite [1 ms]
- *Kontrolle der Übertragung*  
Zeit des Platten-Controllers [ < 1 ms]

Zeit für Zugriff auf eine Seite >> Zeit für Operation im Hauptspeicher !

12.

## Entwicklungstendenzen

- ☐ Zugriffszeit hat sich kaum verändert (Faktor 2 in den letzten 10 Jahren).
  - ☐ Plattenspeicherkapazität hat sich erheblich erhöht (Faktor 10)
  - ☐ Prozessorgeschwindigkeit erhöht sich jedes Jahr um Faktor 1,5.
- ==> eine E/A-Krise droht!

### neue Entwicklungen in den letzten 10 Jahren:

- ☐ Hardware
  - parallele Plattensysteme: RAID (Redundant and Independent Disks)
  - Plattencaches mit Prefetching von Seiten
  - effizientere Abbildung vom linearen Adreßraum auf die Platte
- ☐ Software (Externspeicherverwaltung)
  - effizientere Prefetching-Strategien
  - asynchrone E/A-Operationen
  - Anforderung mehrerer Seiten in einer E/A-Operation

13.

### wichtige Operationen in Dateisystemen:

- ☐ Öffnen (und Schließen) von Dateien
  - Anbindung eines Dateinamens an eine Programmvariable
- ☐ Lesen und Schreiben von einer Seite
  - möglicherweise vom bzw. in einen Puffer
  - Umsetzung der logischen in physische Adressen (z.B. Zylinder-, Spur- und Sektornummer) erfolgt durch das Dateisystem.
- ☐ Allokation von neuen Datenseiten für Dateien
  - UNIX-Dateisystem erlaubt nur, daß eine Datei am Ende erweitert wird.
  - möglichst gute Clusterung von Dateien: logisch zusammenhängende Seiten sollen auch physisch nah beieinander liegen.
  - Freispeicherverwaltung
- ☐ Erzeugen und Löschen von Dateien

15.

## 1.1.2 Seitenorientiertes Dateisystem

- ☐ bietet die Grundfunktionalität für die Organisation von externen Datenstrukturen.

### wichtige Datentypen

- ☐ Datei
    - bietet einen logischen Adreßraum von Seiten an
- $$\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & & N-1 \\ \text{logischer Adreßraum:} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \boxed{\phantom{0}} & \dots\dots & \boxed{\phantom{0}} \end{array}$$
- ☐ Seite (Block)
    - *kleinste Transfereinheit* zwischen Haupt- und Sekundärspeicher
    - *Wahlfreier (direkter) Zugriff.*
    - *Feste Größe für das gesamte Dateisystem* (i. a. zwischen 1 KByte und 8 KByte)
- Größe der Seiten:
- große Seiten = hohe Datenrate (gemessen in MBytes pro Sek.)
  - kleine Seiten = gute Plattenausnutzung
- ☐ Datei besteht aus mehreren exklusiv genutzten Seiten.

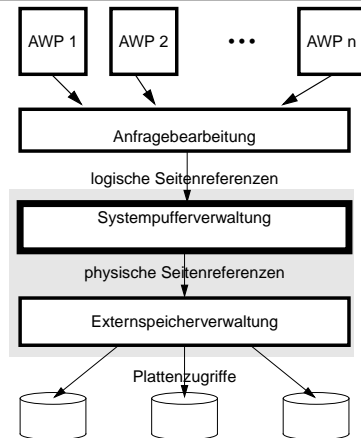
14.

## 1.1.3 Seitenpuffer

- ☐ Puffer besteht aus Frames
  - Annahme: ein Frame kann genau eine Seite aufnehmen.
  - Puffer liegt im tatsächlichen HSP und nicht nur im virtuellen HSP.
- ☐ Ein Puffer kann für
  - eine Datei
  - eine Seitengröße
  - einen Dateityp
 verwendet werden.
  - Annahme: Dateisystem besitzt genau einen Puffer.
- ☐ Puffer wird gemeinsam von verschiedenen Programmen benutzt.
- ☐ Puffer liest direkt von der Platte (ohne daß die Seite nochmals in einem anderen Puffer gespeichert wird.)
- ☐ Zeitpunkt des Zurückschreibens einer modifizierten Seite wird allein durch den Pufferorganisator und nicht durch das einzelne Programm bestimmt.

16.

## Pufferorganisation



### Schnittstelle:

- ☐ Bereitstellen einer DB-Seite im DB-Puffer (zur exklusiven oder gemeinsamen Benutzung).
- ☐ Bereitstellen einer neuen Seite.
- ☐ Freigeben einer Seite.

### intern verwendete Funktionen:

- ☐ effiziente Suche im Puffer
- ☐ Suche nach freien Frames
- ☐ Bestimmen einer Seite, die aus dem Puffer entfernt wird.
- ☐ Schreiben modifizierter Seiten

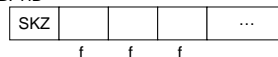
17.

## Abspeicherungsformen

- ☐ Annahme: Satzlänge < Seitenlänge

### 1. Satz mit Attributen fester Länge:

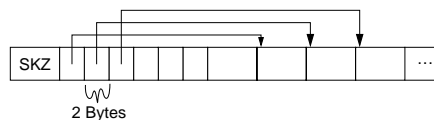
z.B. TID



- ☐ für Attribute variabler Länge

- speicheraufwendig
- unflexibel

### 2. Satz mit Attributen variabler Länge (mit Zeiger im Vorspann):

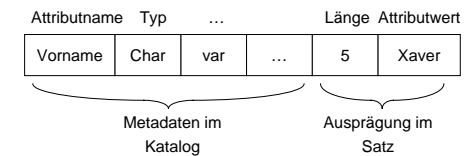


- unflexibel

19.

## 1.2 Abbildung von Sätzen

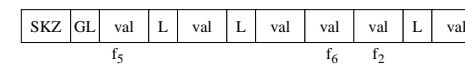
- ☐ Anforderungen:
  - physische Abspeicherung von Sätzen in Seiten
  - Operationen: Lesen, Einfügen, Modifizieren, Löschen
- ☐ Satzbeschreibung
  - pro Attribut:



- Satzbeschreibung separat in einem Katalog

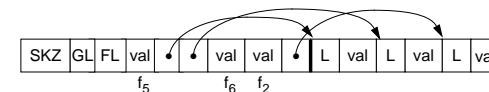
18.

- ☐ Flexiblere Abspeicherungsformen mit Zugriffsmöglichkeit auf einzelne Attribute
- ☐ Betrachte einen Datensatz der Form (f5, v, v, f6, f2, v) (f = fest, v = variabel)
  - eingebettete Längfelder



- dynamische Erweiterung möglich

- Optimierung: eingebettete Längfelder mit Zeigern



Adresse des n-ten Attributs kann berechnet werden

### Annahme (im Rest der Vorlesung):

- ☐ Attribute mit fester Länge.

20.

## Klassifikation

### mögliche Klassifikationskriterien für externe Zugriffsstrukturen

- ☐ eindimensionale und mehrdimensionale Zugriffsstrukturen
- ☐ nach unterstützen Anfragetypen
- ☐ nach unterstützen Datentypen (Zeit, Geo, Text, ...)
- ☐ nach den benutzten Techniken
  - Hashing                      Interpolation                      Baumstruktur
  - transformationsraumteilend      raumteilend                      datenteilend
  - ordnungszerstörend              ordnungserhaltend              ordnungserhaltend

### Bemerkung

Es ist überaus schwierig, einfache und gehaltvolle Klassifikationskriterien für alle Zugriffsstrukturen zu finden. In diesem Kurs wenden wir primär die Kriterien 1-3 an (siehe auch Übersicht der Vorlesung) und werden in den jeweiligen Kapiteln versuchen, eine Verfeinerung vorzunehmen.

## Zusammenfassung

- ☐ Zugriffsstrukturen für Externspeicher
  - vielfältige Anforderungen:
    - hohe Speicherplatzausnutzung
    - effiziente Suche
    - dynamische Unterstützung von Änderungsoperationen
  - wichtige Klassen
    - eindimensional - mehrdimensional
    - statisch - dynamisch
- ☐ Abbildung von Sätzen
  - Speicherung variabel langer Felder
  - dynamische Erweiterungsmöglichkeiten
  - Berechnung von Feldadressen