

Algorithmen und Datenstrukturen 1

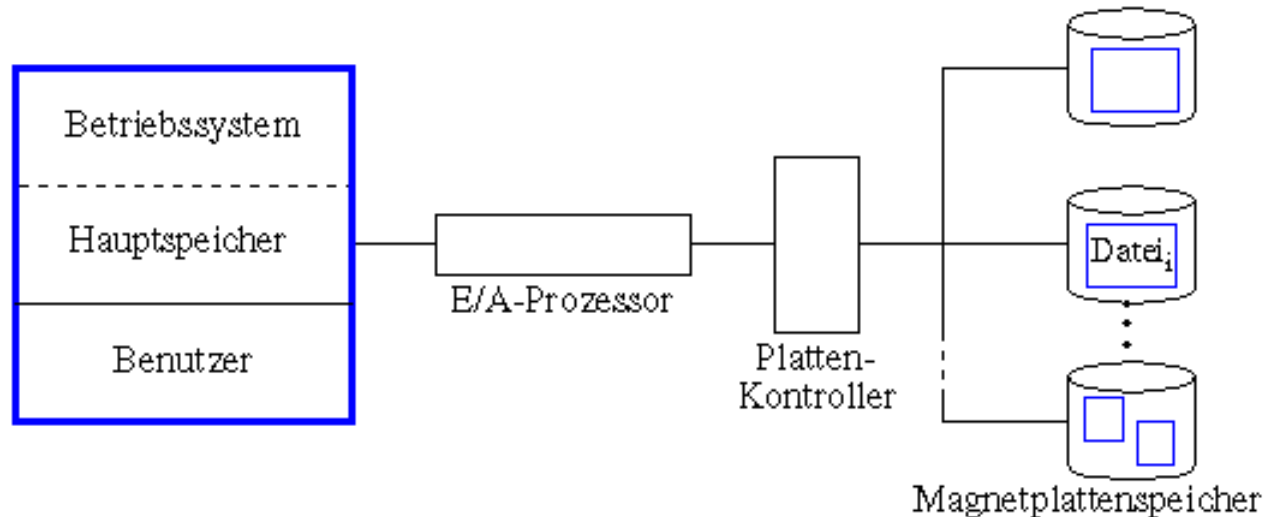
10. Vorlesung

Peter F. Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

Suchverfahren für große Datenmengen

- bisher betrachtete Datenstrukturen (Arrays, Listen, Binärbäume) und Algorithmen waren auf im Hauptspeicher vorliegende Daten ausgerichtet
- effiziente Suchverfahren für große Datenmengen auf Externspeicher erforderlich (persistente Speicherung)
 - große Datenmengen können nicht vollständig in Hauptspeicher-Datenstrukturen abgebildet werden
 - Zugriffsgranulat sind Seiten bzw. Blöcke von Magnetplatten: z.B. 4-16 KB
 - Zugriffskosten 5 Größenordnungen langsamer als für Hauptspeicher (5 ms vs. 50 ns)



Sequentieller Dateizugriff

Sequentielle Dateiorganisation

- Datei besteht aus Folge gleichartiger Datensätze
- Datensätze sind auf Seiten/Blöcken gespeichert
- ggf. bestimmte Sortierreihenfolge (bzgl. eines Schlüssels) bei der Speicherung der Sätze (sortiert-sequenzielle Speicherung)

Sequenzieller Zugriff

- Lesen aller Seiten / Sätze vom Beginn der Datei an
- sehr hohe Zugriffskosten, v.a. wenn nur ein Satz benötigt wird

Optimierungsmöglichkeiten

- "dichtes Packen" der Sätze innerhalb der Seiten (hohe Belegungsdichte)
- Clusterung zwischen Seiten, d.h. "dichtes Packen" der Seiten einer Datei auf physisch benachbarte Plattenbereiche, um geringe Zugriffszeiten zu ermöglichen

Schneller Zugriff auf einzelne Datensätze erfordert Einsatz von zusätzlichen *Indexstrukturen*, z.B. Mehrwegbäume

Alternative: gestreute Speicherung der Sätze (-> Hashing)

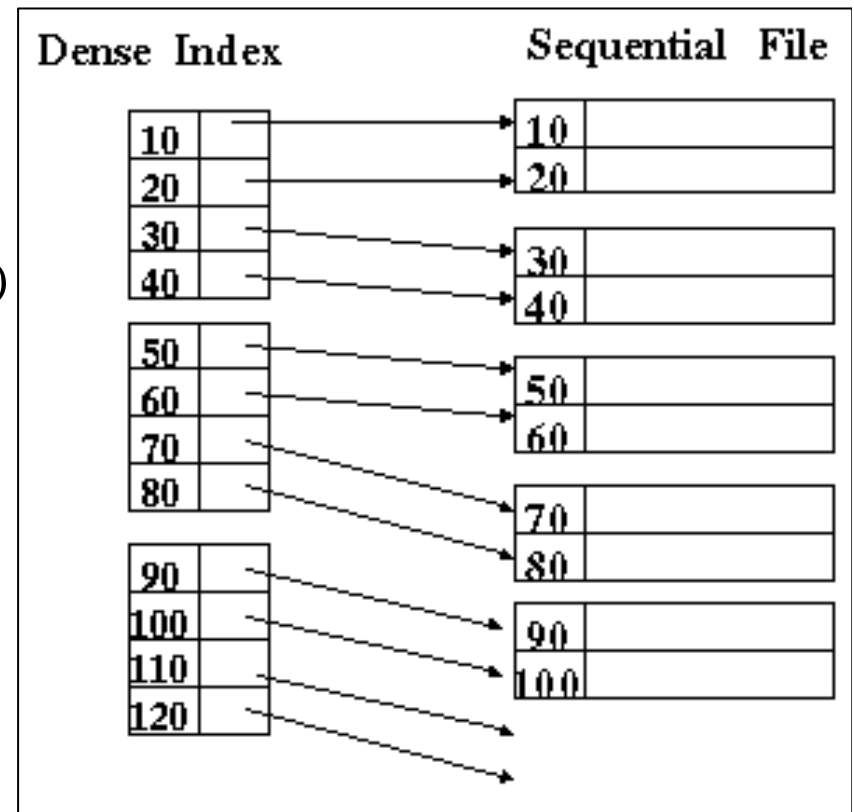
Dichtbesetzter vs. dünnbesetzter Index I

Dichtbesetzter Index (dense index)

- für jeden Datensatz existiert ein Eintrag in Indexdatei
- höherer Indexaufwand als bei dünnbesetztem Index
- breiter anwendbar, u.a. auch bei unsortierter Speicherung der Sätze
- einige Auswertungen auf Index möglich, ohne Zugriff auf Datensätze (Existenztest, Häufigkeitsanfragen, Min/Max-Bestimmung)

Anwendungsbeispiel

- 1 Million Sätze, $B=20$, 200 Indexeinträge pro Seite
- Dateigröße:
- Indexgröße:
- mittlere Zugriffskosten:



Dichtbesetzter vs. dünnbesetzter Index II

Dünnbesetzter Index (sparse index)

- nicht für jeden Schlüsselwert ex. Eintrag im Index
- sinnvoll v.a. bei Clustering gemäß Sortierreihenfolge des Indexattributes: ein Indexeintrag pro Datenseite

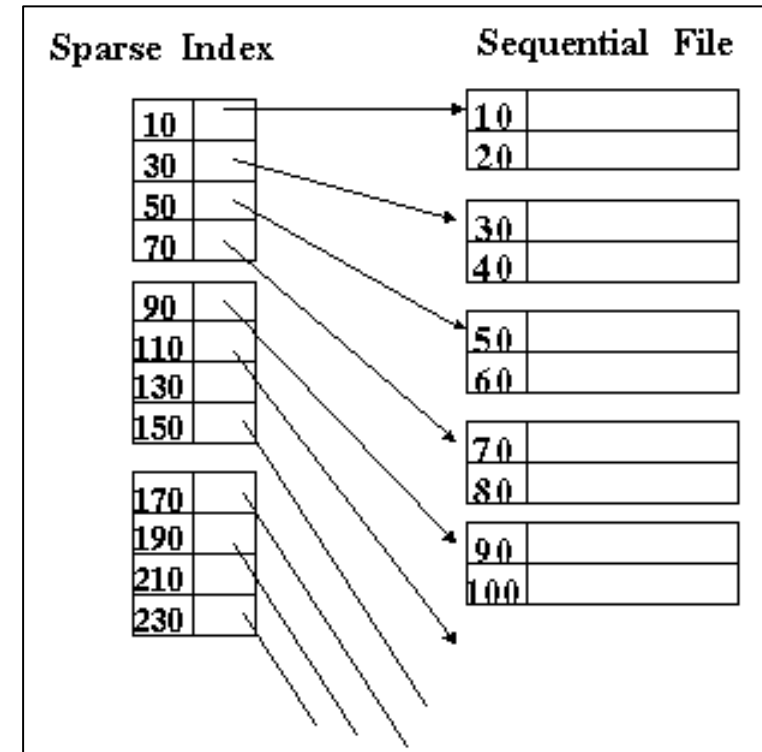
indexsequentielle Datei (ISAM): sortiertesequentielle Datei mit dünnbesetztem Index für Sortierschlüssel

Anwendungsbeispiel

- 1 Million Sätze, $B=20$, 2 Indexeinträge pro Seite
- Dateigröße:
- Indexgröße:
- mittlere Zugriffskosten:

Mehrstufiger Index

- Indexdatei entspricht sortiert sequentieller Datei -> kann selbst wieder indexiert werden
- auf höheren Stufen kommt nur dünnbesetzte Indexierung in Betracht
- beste Umsetzung im Rahmen von Mehrwegbäumen (B -/ B^* -Bäume)



Mehrwegebäume

Ausgangspunkt: Binäre Suchbäume (balanciert)

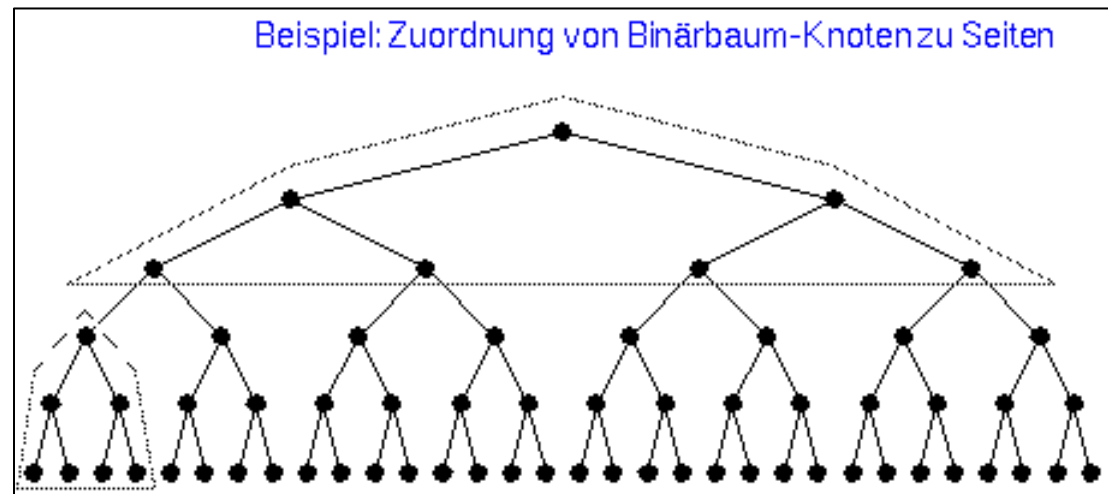
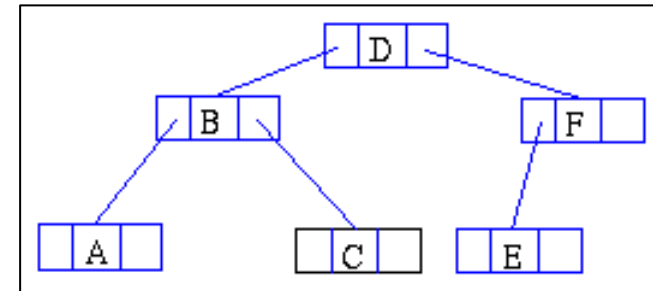
- entwickelt für Hauptspeicher
- ungeeignet für große Datenmengen

Externspeicherzugriffe erfolgen auf Seiten

- Abbildung von Schlüsselwerten/Sätzen auf Seiten
- Index-Datenstruktur für schnelle Suche

Alternativen:

- m-Wege-Suchbäume
- B-Bäume
- B*-Bäume



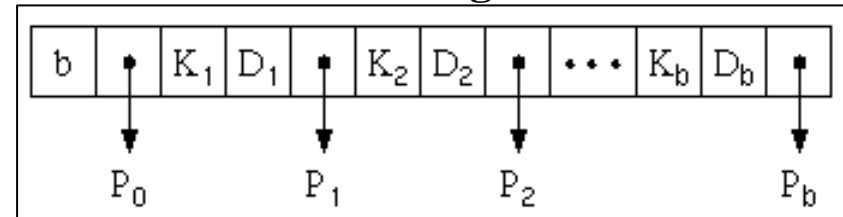
Grundoperationen: Suchen, Einfügen, Löschen

Kostenanalyse im Hinblick auf Externspeicherzugriffe

m-Wege-Suchbäume I

Def.: Ein m-Wege-Suchbaum oder ein m-ärer Suchbaum B ist ein Baum, in dem alle Knoten einen Grad $\leq m$ besitzen. Entweder ist B leer oder er hat folgende Eigenschaften:

- (1) Jeder Knoten des Baums mit b Einträgen, $b \leq m-1$, hat folgende Struktur:



Die P_i , $0 \leq i \leq b$, sind Zeiger auf die Unterbaume des Knotens und die K_i und D_i , $1 \leq i \leq b$ sind Schlüsselwerte und Daten.

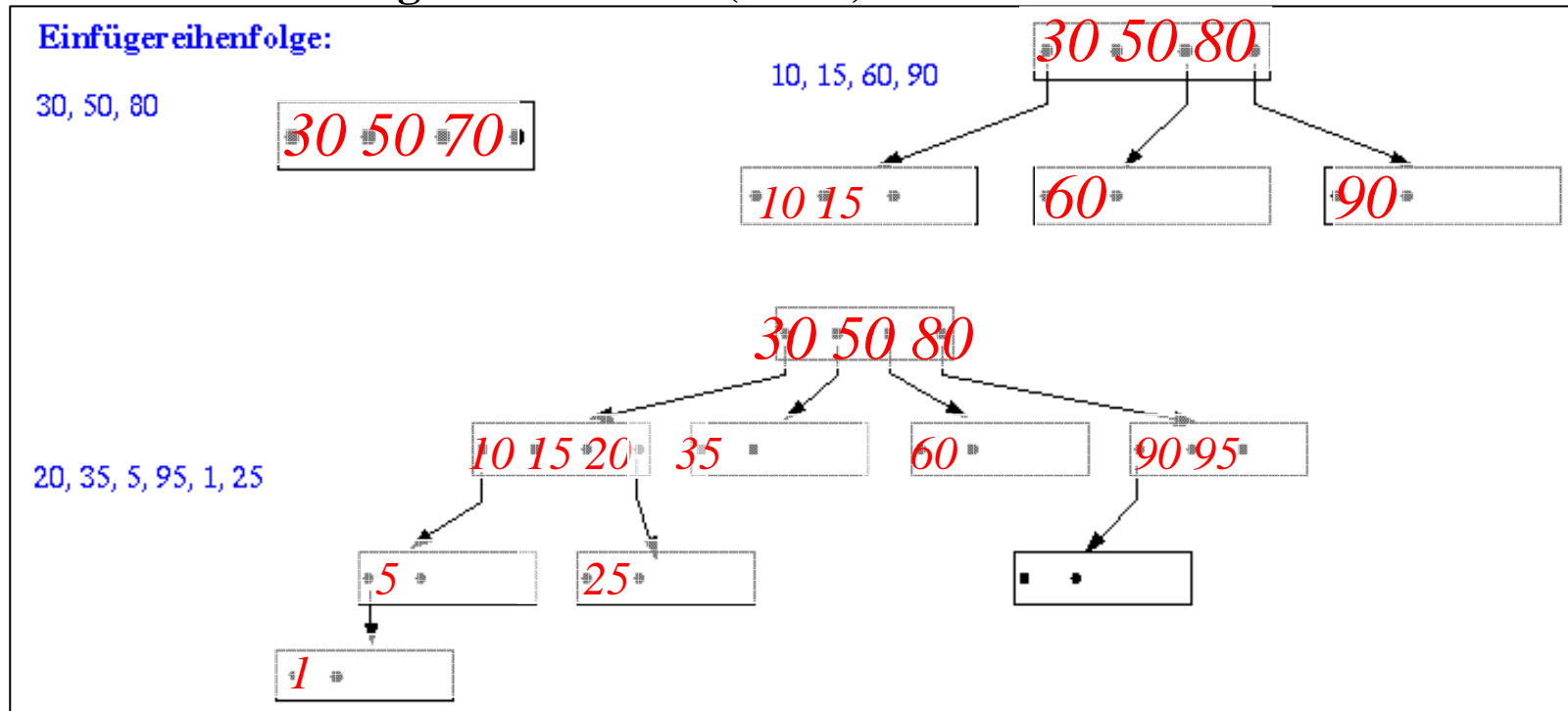
- (2) Die Schlüsselwerte im Knoten sind aufsteigend geordnet: $K_i \leq K_{i+1}$, $1 \leq i < b$.
(3) Alle Schlüsselwerte im Unterbaum von P_i sind kleiner als der Schlüsselwert K_{i+1} , $0 \leq i < b$.
(4) Alle Schlüsselwerte im Unterbaum von P_b sind größer als der Schlüsselwert K_b .
(5) Die Unterbäume von P_i , $0 \leq i \leq b$ sind auch m-Wege-Suchbäume.

Die D_i können Daten oder Zeiger auf die Daten repräsentieren

- direkter Index: eingebettete Daten (weniger Einträge pro Knoten; kleineres m)
- indirekter Index: nur Verweise; erfordert separaten Zugriff auf Daten zu dem Schlüssel

m-Wege-Suchbäume II

Beispiel: Aufbau eines m-Wege-Suchbaumes (m = 4)



Beobachtungen

- Die Schlüssel in den inneren Knoten besitzen zwei Funktionen. Sie identifizieren Daten(sätze) und sie dienen als Wegweiser in der Baumstruktur
- Der m-Wege-Suchbaum ist im allgemeinen nicht ausgeglichen

m-Wege-Suchbäume III

Wichtige Eigenschaften für alle Mehrwegbäume:

$S(P_i)$ sei die Seite, auf die P_i zeigt, und $K(P_i)$ sei die Menge aller Schlüssel, die im Unterbaum mit Wurzel $S(P_i)$ gespeichert werden können. Dann gelten folgende Ungleichungen:

- (1) $x \in K(P_0): x < K_1$
- (2) $x \in K(P_i): K_i < x < K_{i+1}$ für $i = 1, 2, \dots, b-1$
- (3) $x \in K(P_b): K_b < x$

Kostenanalyse

- Die Anzahl der Knoten N in einem vollständigen Baum der Höhe h , $h \geq 1$, ist

$$N = \sum_{i=0}^{h-1} m^i = \frac{m^h - 1}{m - 1}$$

- Im ungünstigsten Fall ist der Baum völlig entartet: $n = N = h$
- Schranken für die Höhe eines m-Wege-Suchbaums: $\log_m(n+1) \leq h \leq n$

Mehrwegbäume I

Ziel: Aufbau sehr breiter Bäume von geringer Höhe

- in Bezug auf Knotenstruktur vollständig ausgeglichen
- effiziente Grundoperationen auf Seiten (= Transporteinheit zum Externspeicher)
- Zugriffsverhalten weitgehend unabhängig von Anzahl der Sätze
- Einsatz als Zugriffs-/Indexstruktur für 10 als auch für 10^{10} Sätze

Grundoperationen:

- direkter Schlüsselzugriff auf einen Satz
- sortiert sequentieller Zugriff auf alle Sätze
- Einfügen eines Satzes; Löschen eines Satzes

Mehrwegbäume II

Varianten

- ISAM-Dateistruktur (1965; statisch, periodische Reorganisation)
- Weiterentwicklungen: B- und B*-Baum
- B-Baum: 1970 von R. Bayer und E. McCreight entwickelt
dynamische Reorganisation durch Splitten und Mischen von Seiten

Breites Spektrum von Anwendungen ("The Ubiquitous B-Tree")

- Dateiorganisation ("logische Zugriffsmethode", VSAM)
- Datenbanksysteme (Varianten des B*-Baumes sind in allen DBS zu finden!)
- Text- und Dokumentenorganisation ...

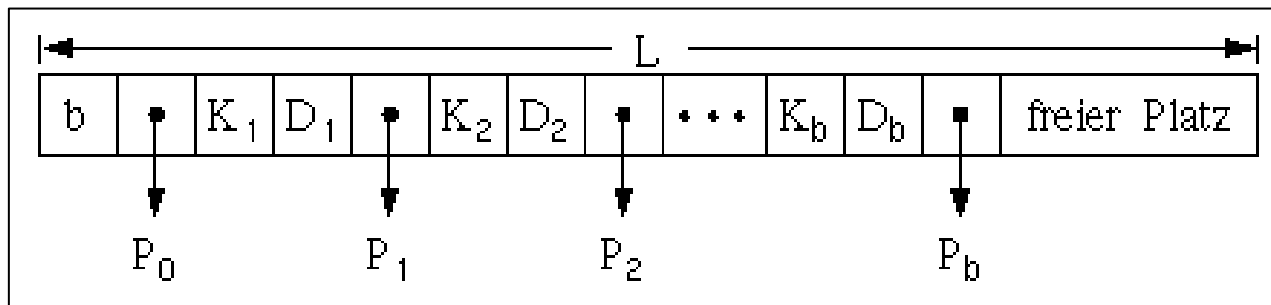
B-Bäume I

Def.: Seien k, h ganze Zahlen, $h \geq 0, k > 0$.

Ein **B-Baum** B der Klasse $\tau(k,h)$ ist entweder ein leerer Baum oder ein geordneter Baum mit folgenden Eigenschaften:

1. Jeder Pfad von der Wurzel zu einem Blatt hat die gleiche Länge $h-1$.
2. Jeder Knoten außer der Wurzel und den Blättern hat mindestens $k+1$ Söhne. Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne
3. Jeder Knoten hat höchstens $2k+1$ Söhne
4. Jedes Blatt mit der Ausnahme der Wurzel als Blatt hat mindestens k und höchstens $2k$ Einträge.

Für einen B-Baum ergibt sich folgendes Knotenformat:



B-Bäume II

Einträge

- Die Einträge für Schlüssel, Daten und Zeiger haben die festen Längen l_b , l_K , l_D und l_p .
- Die Knoten- oder Seitengröße sei L .
- Maximale Anzahl von Einträgen pro Knoten:

$$b_{\max} = \left\lfloor \frac{L - l_b - l_p}{l_K + l_D + l_p} \right\rfloor = 2k$$

Reformulierung der Definition

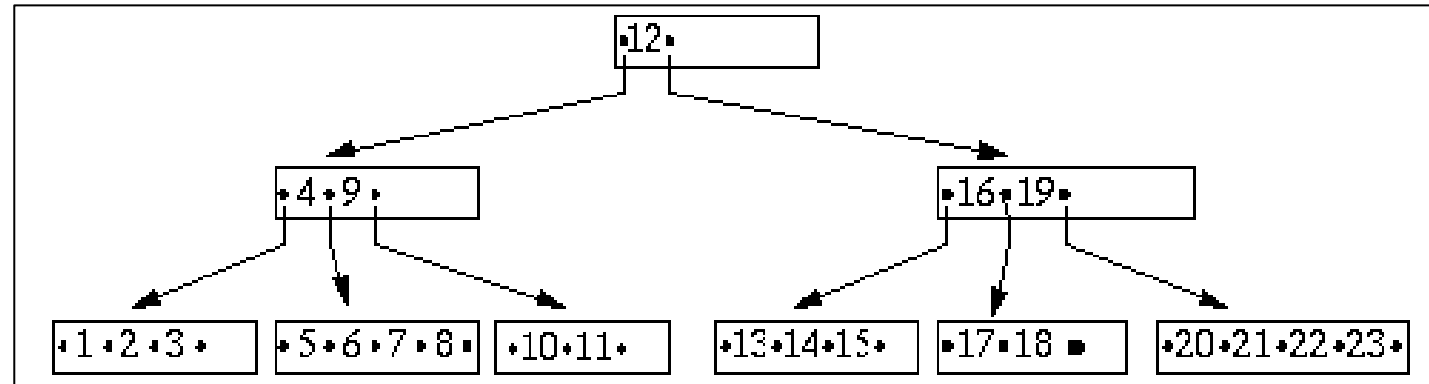
- (4) und (3). Eine Seite darf höchstens voll sein.
- (4) und (2). Jede Seite (außer der Wurzel) muss mindestens halb voll sein.
Die Wurzel enthält mindestens einen Schlüssel.
- (1) Der Baum ist, was die Knotenstruktur angeht, vollständig ausgeglichen

Balancierte Struktur:

- unabhängig von Schlüsselmenge
- unabhängig ihrer Einfügereihenfolge

B-Bäume III

**Beispiel: B-Baum
der Klasse $\tau(2,3)$**



- In jedem Knoten stehen die Schlüssel in aufsteigender Ordnung mit $K_1 < K_2 < \dots < K_b$
- Jeder Schlüssel hat eine Doppelrolle als Identifikator eines Datensatzes und als Wegweiser im Baum
- Die Klassen $\tau(k,h)$ sind nicht alle disjunkt. Beispielsweise ist ein maximaler Baum aus $\tau(2,3)$ ebenso in $\tau(3,3)$ und $\tau(4,3)$.

Höhe h: Bei einem Baum der Klasse $\tau(k,h)$ mit n Schlüsseln gilt für seine Höhe:

$$\log_{2k+1}(n+1) \leq h \leq \log_{k+1}((n+1)/2)+1 \quad \text{für } n \geq 1$$

und $h = 0$ für $n = 0$

Einfügen in B-Bäumen I

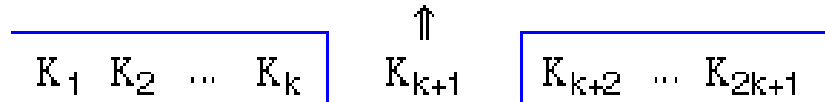
Was passiert, wenn Wurzel überläuft ?

• K_1 • K_2 • ... • K_{2k} •

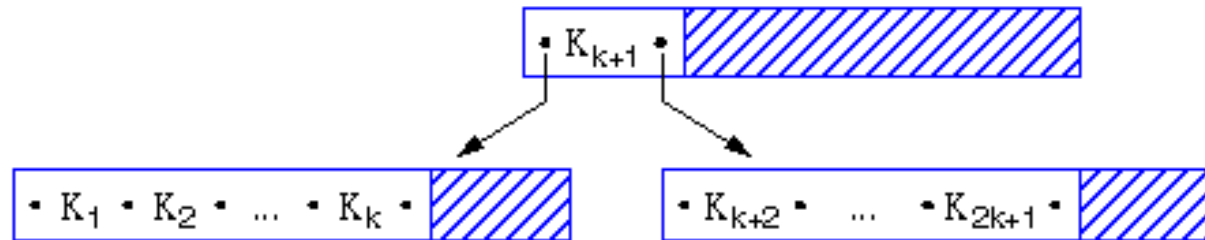
K_{2k+1}

Fundamentale Operation: Split-Vorgang

1. Anforderung einer neuen Seite und
2. Aufteilung der Schlüsselmenge nach folgendem Prinzip



- mittlere Schlüssel (Median) wird zum Vaterknoten gereicht
- Ggf. muß Vaterknoten angelegt werden (Anforderung einer neuen Seite).

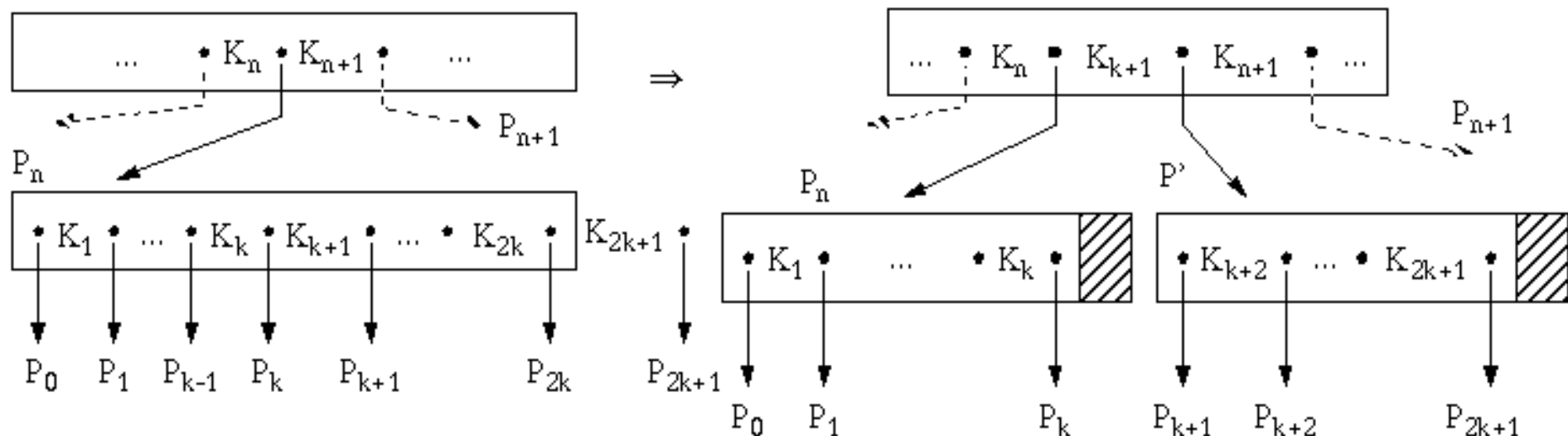


Split führt hier zu einer neuen Wurzel a

- Blattüberlauf erzwingt Split-Vorgang, was Einfügung in den Vaterknoten impliziert
- Wenn dieser überläuft, folgt erneuter Split-Vorgang
- Split-Vorgang der Wurzel führt zu neuer Wurzel: Höhe des Baumes erhöht sich um 1

Einfügen in B-Bäumen II

- Bei B-Bäumen ist Wachstum von den Blättern zur Wurzel hin gerichtet
- Einfügealgorithmus (ggf. rekursiv)
 - Suche Einfügeposition: Wenn Platz vorhanden ist, speichere Element, sonst schaffe Platz durch Split-Vorgang und füge ein
- Split-Vorgang als allgemeines Wartungsprinzip



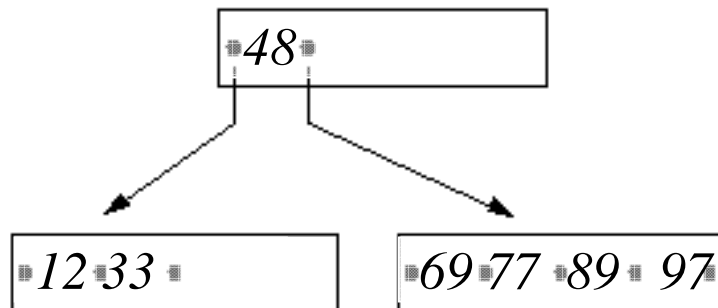
Beispiel: Aufbau eines B-Baumes der Klasse $\tau(2,h)$ I

Einfügereihenfolge:

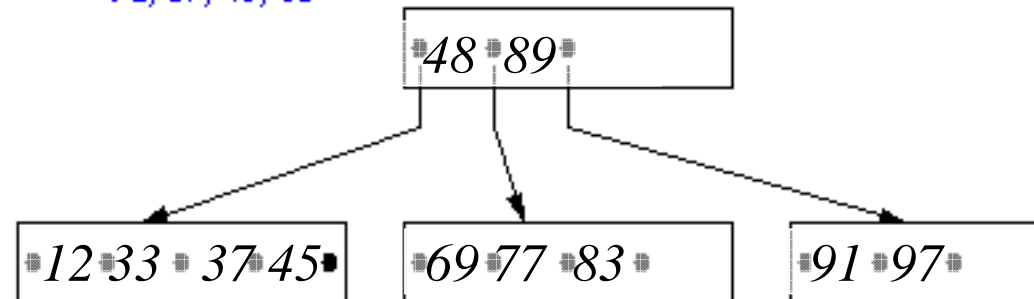
77, 12, 48, 69

12 48 69 77

33, 89, 97



91, 37, 45, 83



2, 5, 57, 90, 95

