Datenstrukturen für externe Daten

Grosse Datenmengen, die nicht in den Hauptspeicher passen, werden auf peripheren Speichern verwaltet. Datenstrukturen müssen für die langsamere Peripherie optimiert sein – da Teile der Datenstruktur für die Verarbeitung zunächst in den Hauptspeicher übertragen werden muss!
Bsp.: (relationale) Datenbanksysteme

Ziel: Minimierung der E/A-Vorgänge!

File: Zusammenfassung mehrerer peripher gespeicherter Records - möglichst kleine Anzahl der Peripheriezugriffe.

Datentransport erfolgt in Seiten/ Blöcken fester Größe – möglichst 1 Baumknoten!

Datenstrukturen für externe Daten

Operationen:

- Record im File suchen/ finden
- Record(s) zum HS übertragen und umgekehrt – möglichst als Teile eines Baumknotens in Seitengröße

Records werden durch Schlüssel identifiziert, Schlüssel müssen effizient in Recordnummern abbildbar sein

Speichermethoden für Files:

- sequentiell
- Indexsequentiell
- gestreut
- baumstrukturiert (Record ist Teil eines Baumknoten)

Baumstrukturierte Fileorganisation

Baumstrukturen unterstützen:

- den direkten Zugriff auf einen Record über seinen Schlüssel
- sequentielle Verarbeitung der Records in Schlüsselfolge

- durchschnittliche Seitengröße des Hauptspeichers: 2-8 KB
- Zugriff auf eine Datenseite, die sich bereits im HS befindet: ca. Faktor 100.000 mal schneller, als holen von der Platte -> Records dürfen nicht einzeln übertragen werden!

Für die Datenverwaltung auf mehrstufigen Speichern wurden Baumstrukturen entwickelt, bei denen jeder Knoten > 2 Nachfolger hat (Mehrwegbäume).

Ziel: möglichst viele Nachfolger pro Knoten zulassen und gleichzeitig mit dem Knoten ein- und auslagern!

Mehrwegbäume

- Mehrwegbäume sind buschiger und breiter

- Bei der Verarbeitung müssen Lokalitätseigenschaften ausgenutzt werden:
    - räumlich Lokalität (nahe Nachbarn)
    - zeitliche Lokalität (hochfrequent benutzte Knoten): Wurzel, höhere Baumebenen

- Höhe des Baumes bestimmt die max. Anzahl der Externzugriffe im Binärbaum (log N)

Baumstrukturierte Fileorganisation

Eine Speicherseite fasst mehrere (bis sehr viele) Knoten

Knotenmengen werden zu neuen Knoten (N = Node, P=Page)

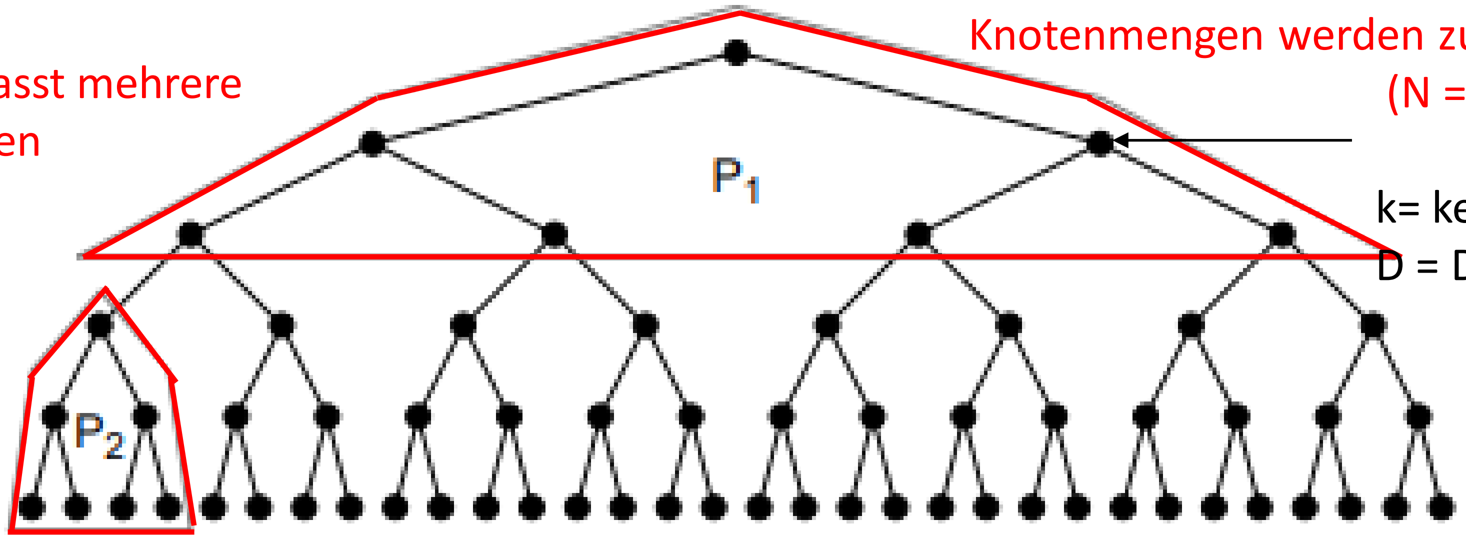


Bild 2: Unterteilung eines großen binären Suchbaumes in Seiten

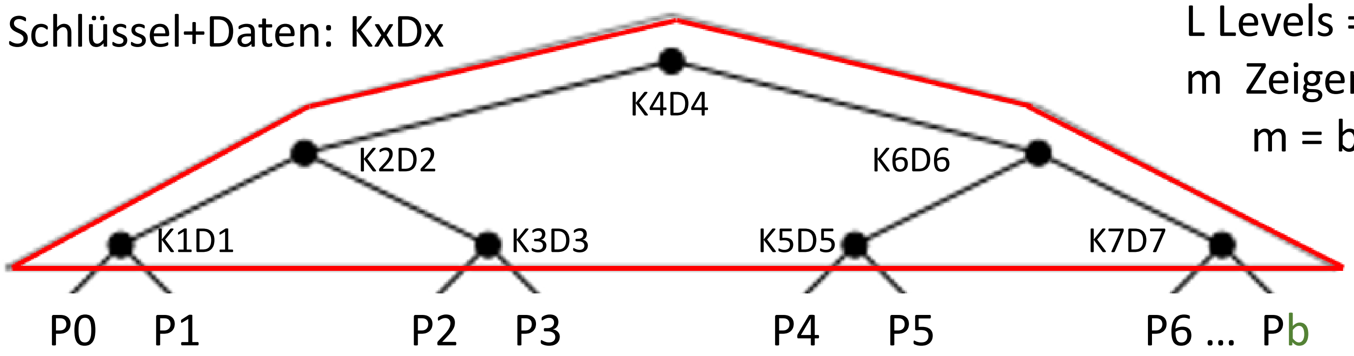
k= key (Schlüssel) +
D = Data (record)

Die Kosten für die internen Operationen auf einem Knoten können absolut vernachlässigt werden gegenüber den E-/A-Kosten.

Baumstrukturierte Fileorganisation: m-Wege-Suchbaum

		im Bsp:
b	Keys pro Knoten	(7)
L	Levels = Höhe = log (b)	(3)
m	Zeiger (zu Unter-Bäumen), $m = b+1 = 2^L$	(8)

neuer Knoten, Schlüssel+Daten: KxDx

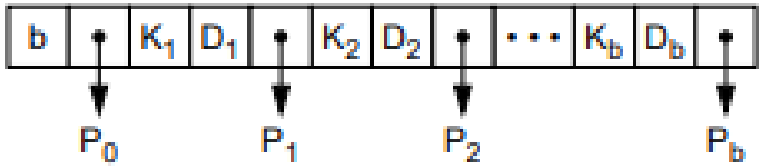


Weitere Pages bzw. Unter-Bäume

Bsp: m=8 Wege aus einem Knoten heraus

m-Wege-Suchbaum: Alle Knoten besitzen einen Grad <=m (**Ordnung**). Heißt: kein Knoten hat mehr als m Unterbäume.

I. Jeder Knoten hat folgende Struktur:



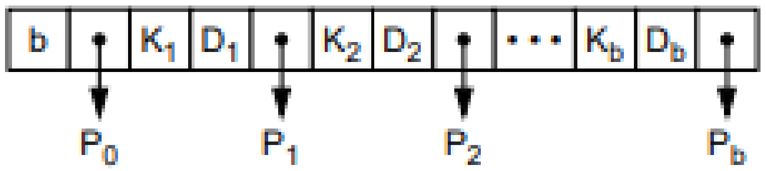
K_xD_x: Key_x/$Schlüssel_x$ + $Data_x$ (Record $_x$)

P_y: nächster $Page\text{-}Baum_y$

Baumstrukturierte Fileorganisation: m-Wege-Suchbaum

b Keys pro Knoten
L Levels = Höhe = log (b)
m Zeiger (zu Unter-Bäumen),
$m = b+1 = 2^L$

I. Jeder Knoten hat folgende Struktur:



K_xD_x: Key_x + $Data_x$ (Record $_x$)
P_y: Page-$Baum_y$

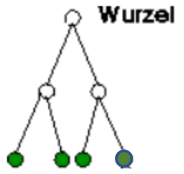
II. Die Schlüsselwerte K_i im Knoten sind aufsteigend geordnet.
III. Alle Schlüsselwerte im Unterbaum von P_i sind kleiner als der Schlüsselwert K_{i+1}
IV: Alle Schlüsselwerte im Unterbaum von P_i sind größer als der Schlüsselwert K_i
V: Die Unterbäume von P_i sind auch m-Wege-Suchbäume

Für den m-Wege-Suchbaum sind keine Balanzierungsmechanismen definiert, so dass Wildwuchs (Blätter auf verschiedenen Baumebenen) entstehen kann -> balancierte Bäume definieren...

Eigenschaften binärer Bäume (0..2 Kinder)

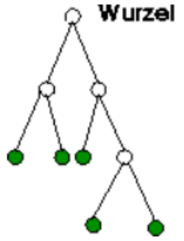
Vollständiger Binärbaum:

- jeder Knoten hat 2 oder keine Kinder
- alle Blätter haben den gleichen Abstand zur Wurzel



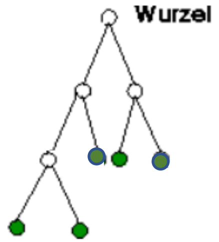
Voller Binärbaum:

- jeder Knoten hat 2 oder keine Kinder



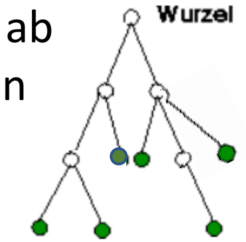
Heap-vollständiger Binärbaum:

- jeder Knoten hat 2 oder keine Kinder
- Kinder dürfen nur auf der untersten Ebene rechts fehlen
- Grund: Finden des letzten Knotens über dessen Index im Array möglich

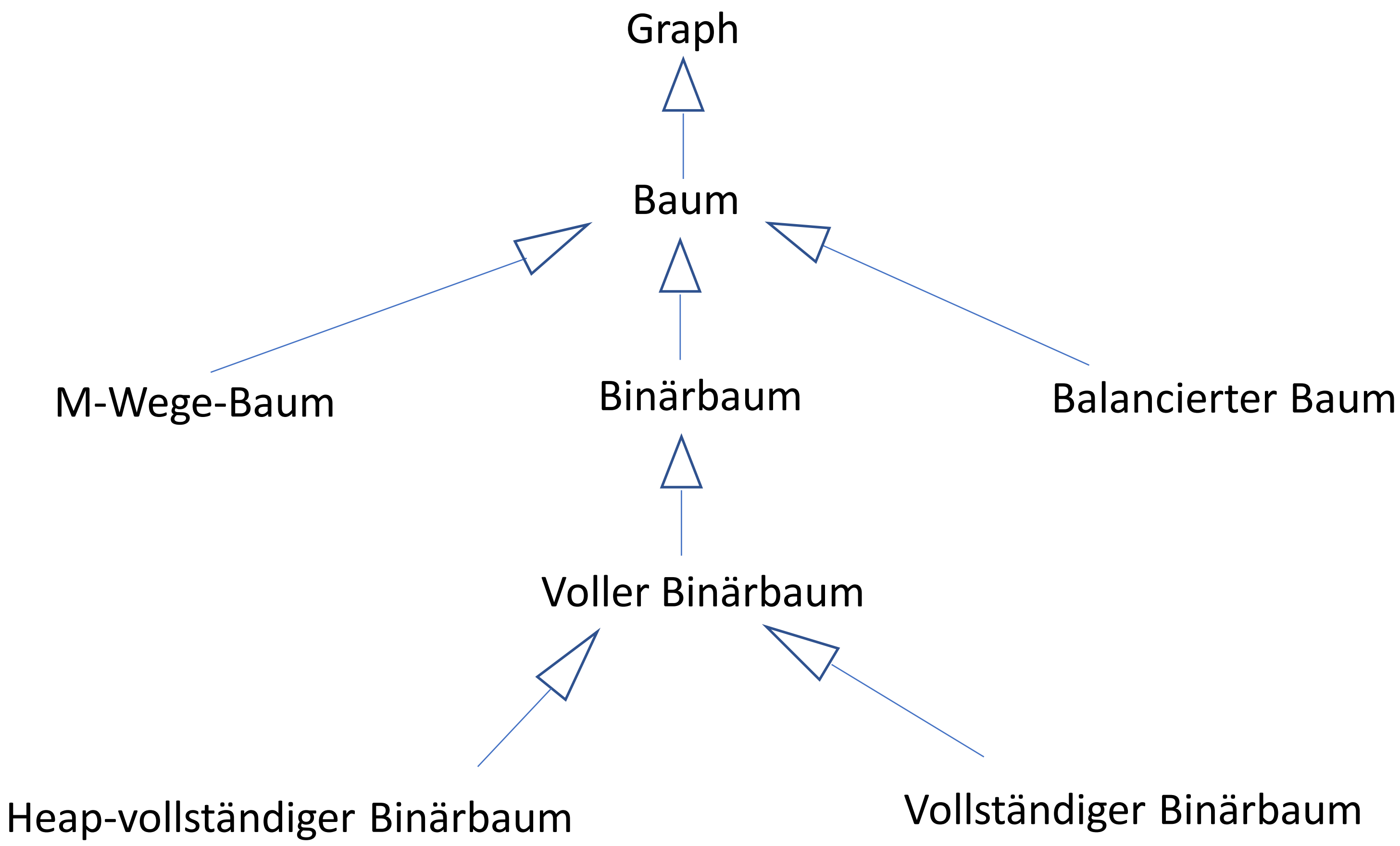


Balancierter m-Wege-Baum:

- alle Blätter weichen in ihrem Abstand zur Wurzel nur maximal um 1 ab
- in einem m-Wege-B-Baum mit n Knoten erreicht man ausgehend von der Wurzel alle Knoten in max. $\log_m(n)$ Schritten



Vererbungshierarchie binärer Bäume ("...ist ein...")



Balancierte Bäume (B-Bäume)

M-Wege-Suchbäume nennt man balanciert (ausgeglichen), wenn in einem Baum der Höhe h mit N Schlüsseln jede der 3 Operation

- Suchen
- Einfügen
- Entfernen

mit max. O ($\log_m$ N) Schritten ausführbar ist.

Heißt: Die Höhe eines Baumes mit N Knoten ist über jedem Blattknoten max. $\log_m$ N oder ($\log_m$ N)-1 (dafür müssen innere Knoten nicht die gleiche Menge an Nachfolgern haben).

Heißt: alle Blätter haben (fast) die gleiche Tiefe (Ebene) (Abweichung -1 erlaubt).

(Selbst-)Balanzierte Bäume (B-Bäume)

(Selbst-)Balanzierte Bäume (B-Bäume)

Die Neu-Balanzierung des m-Weg-Baumes würde zu einem Wartungsaufwand nach jeder Aktualisierung (Einfügen/ Löschen) von O(N) führen.

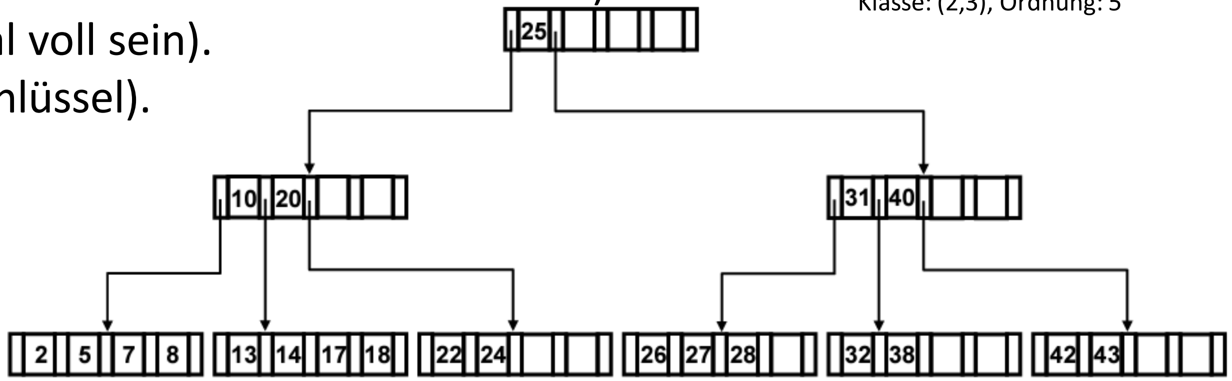
Gesucht: Balanzierungsmechanismus, der mit Hilfe lokaler Baumtransformationen den Baum fast ausgeglichen hält:
Rot-Schwarz-Bäume, AVL-Bäume

B-Bäume sind fast ausgeglichene, sehr breite Mehrwegbäume von geringer Höhe.
In Datei- und Datenbanksystemen dienen sie zur Organisation von Zugriffspfadstrukturen.

Definition:
Ein Baum der Klasse (k, h) ist ein geordneter Suchbaum mit folgenden Eigenschaften (k=min. keys pro Knoten, h=height der Knoten, Ordnung: 2k+1 (Folgeknoten)):

i. Jeder Pfad von der Wurzel zu einem Blatt hat (fast) die gleiche Länge.
ii. Jeder innere Knoten hat **mindestens k(ey) Schlüssel** (Seite muss mindestens halb voll sein)
iii. Jeder Knoten hat **höchstens 2k Schlüssel** (Seite darf maximal voll sein).
iv. Jedes Blatt hat mindestens k und max. 2k Einträge (keys/ Schlüssel).
v. Die Wurzel hat mind. 1 Schlüssel

Klasse: (2,3), Ordnung: 5



„Ordnung“ m: max. Anzahl der Schlüssel eines Knotens + 1
=max. Anzahl der Kindbäume eines Knoten
(D. Knuth, 1973, „The Art of Computer Programming“)

(Selbst-)Balanzierte Bäume: Suche

Binäre Suche innerhalb jedes Knotens!

(Selbst-)Balanzierte Bäume: Einfügen

Schlüssel werden immer **in Blätter** an die entsprechende Stelle **eingefügt**.

a) Wenn das Blatt einen weiteren Schlüssel aufnehmen kann: fertig.

b) Bei Überschreitung von 2k: aufspalten des Knoten in 2 - der mittlere Schlüssel rückt nach oben. Weiter bei Schritt 2. Wird der Wurzelknoten erreicht, wächst der Baum u.U. um 1 Level nach oben.

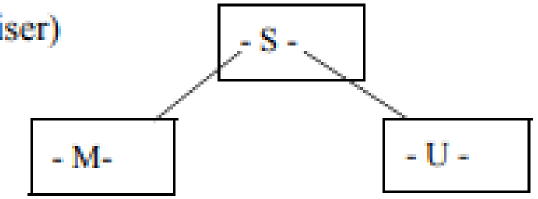
(Selbst-)Balanzierte Bäume: Einfügen

Wir fügen die Buchstaben des Wortes ALGORITHMUS in umgekehrter Reihenfolge ihres Auftretens im Wort in einen initial leeren B-Baum der Klasse (1, h) ein.

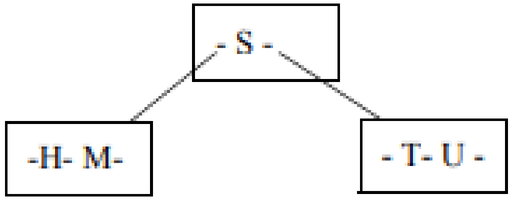
ALGORITHMUS in umgekehrter Reihenfolge: S, U, M, H, T, I, R, O, G, L, A

Einfügen: S,U

Einfügen: M (=> Split, S nach oben als Wegweiser)



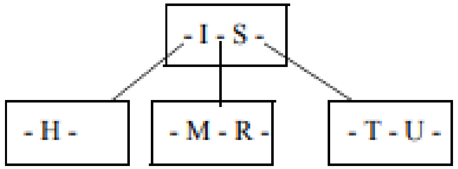
Einfügen: H, T



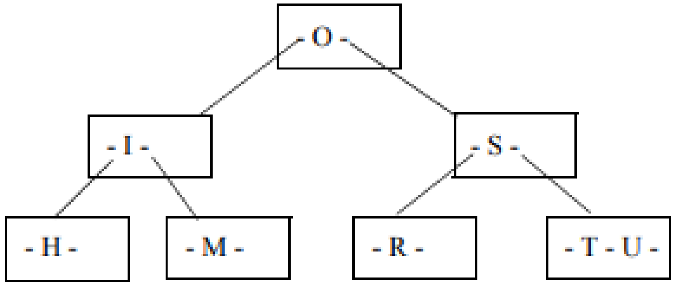
(Selbst-)Balanzierte Bäume (B-Bäume)

ALGORITHMUS in umgekehrter Reihenfolge: S, U, M, H, T, I, R, O, G, L, A

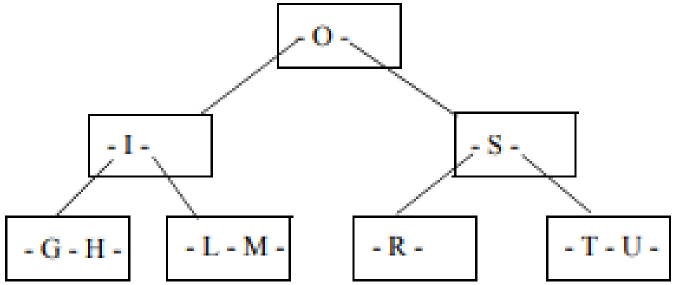
Einfügen: I (=> Spilit, I nach oben als Wegweiser), R



Einfügen: O (=> 2xSplit, jeweils O als Wegw.)



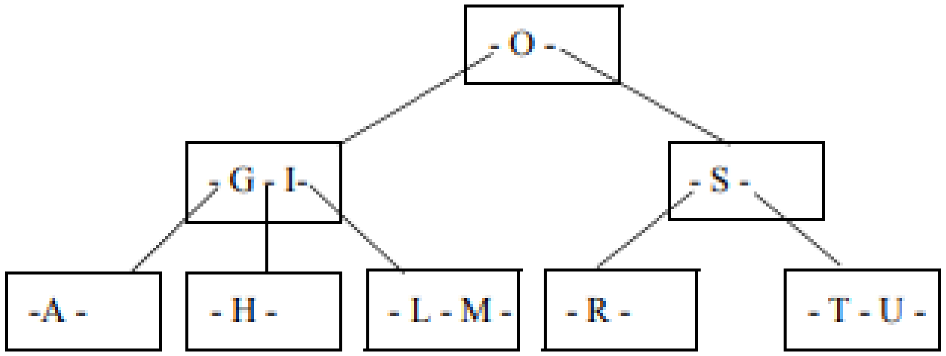
Einfügen: G, L



(Selbst-)Balanzierte Bäume (B-Bäume)

ALGORITHMUS in umgekehrter Reihenfolge: S, U, M, H, T, I, R, O, G, L, A

Einfügen: A



(Selbst-)Balanzierte Bäume: Löschen

1. Löschen im Blatt:
   Die Mindestanzahl an Schlüsseln muss erhalten bleiben:
   a) Ist die Anzahl der Schlüssel im Knoten, in dem der zu löschende Wert gefunden wurde, > k, dann ist das Löschen trivial: Wert wird gelöscht oder
   b) Löschen erzeugt Unterlauf im Knoten (zu wenig Schlüssel): Generell müssen Schlüssel aus dem betroffenen Knoten und einem Nachbarknoten neu „verteilt“ werden, um B-Baum-Kriterium zu erhalten:
      1) Rotation: Kann durchgeführt werden, wenn der Nachbarknoten danach mehr als k Knoten behält
      2) Mischen: Muß genommen werden, wenn durch Rotation im Nachbarknoten ein Unterlauf entstünde

2. Löschen im inneren Knoten: Der Wert kann nicht direkt gelöscht werden, da er Indikator für die an ihm „hängenden“ Unterbäume ist
   Er wird „getauscht“ gegen
   2. Den symmetrischen Vorgänger (sym. Vorgänger ist der größte Blattknoten im linken Unterbaum bzw. der größte Wert in diesem Blattknoten) oder
   3. Den symmetrischen Nachfolger (sym. Nachfolger ist der kleinste Blattknoten im rechten Unterbaum, bzw. der kleinste Wert)

(Selbst-)Balanzierte Bäume: Löschen der 40

Klasse: (2, h)

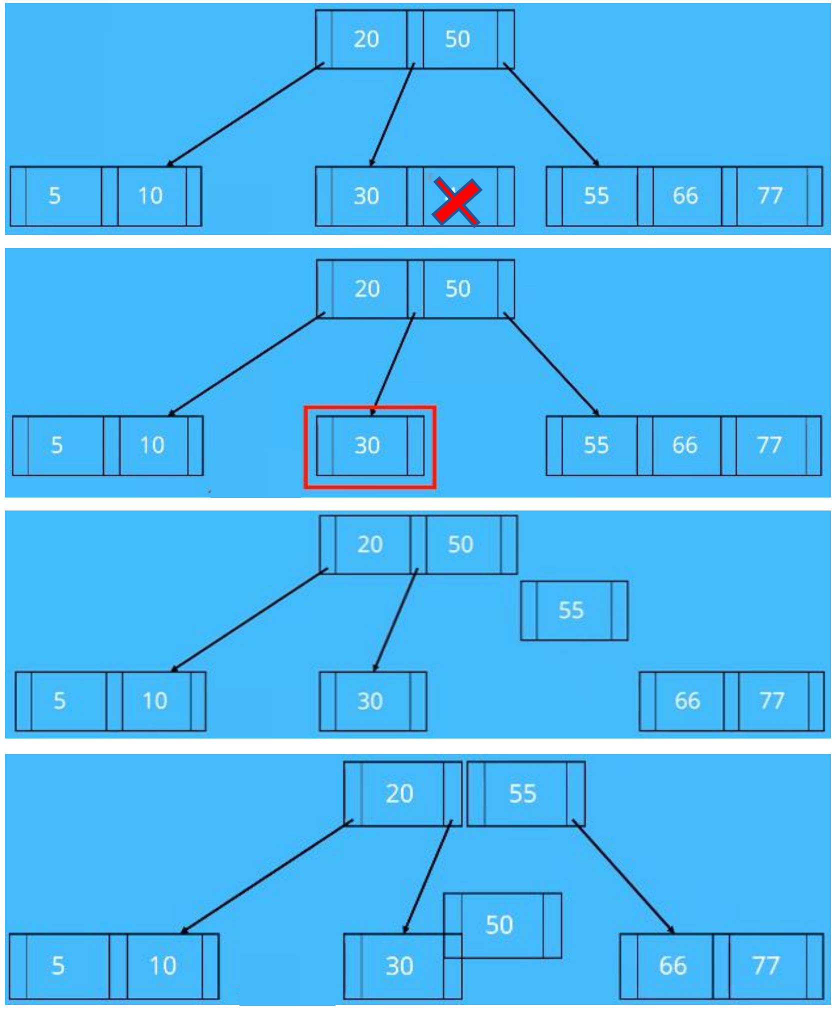
1. Löschen im Blatt:

   Die Mindestanzahl an Schlüsseln muss erhalten bleiben:

   a) Anzahl der Schlüssel im Knoten, in dem der zu löschende Wert gefunden wurde, ist > k:
   Schlüssel löschen

   b) Löschen erzeugt Unterlauf im Knoten (zu wenig Schlüssel):
   Generell müssen Schlüssel aus dem betroffenen Knoten und einem Nachbarknoten neu „verteilt“ werden, um B-Baum-Kriterium zu erhalten:

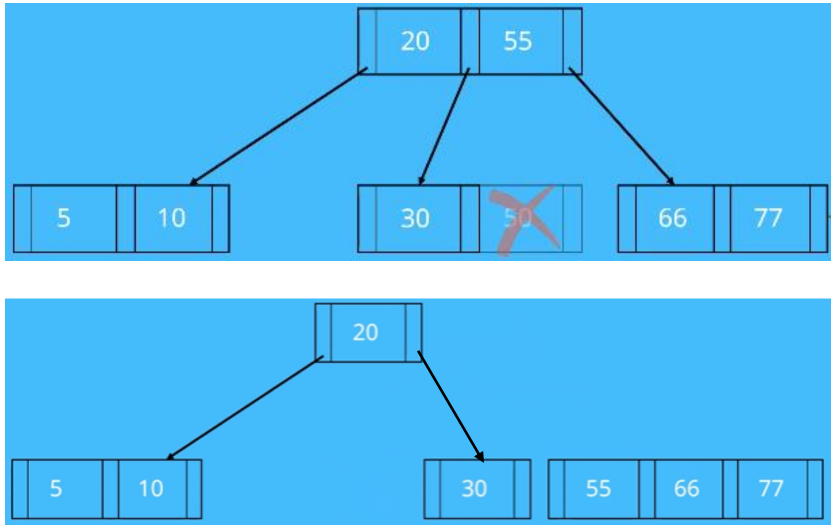
      1) Rotation: Kann durchgeführt werden, wenn der Nachbarknoten danach mehr als k Knoten behält



DHBW Duale Hochschule Baden-Württemberg Heidenheim

(Selbst-)Balanzierte Bäume: Löschen der 50

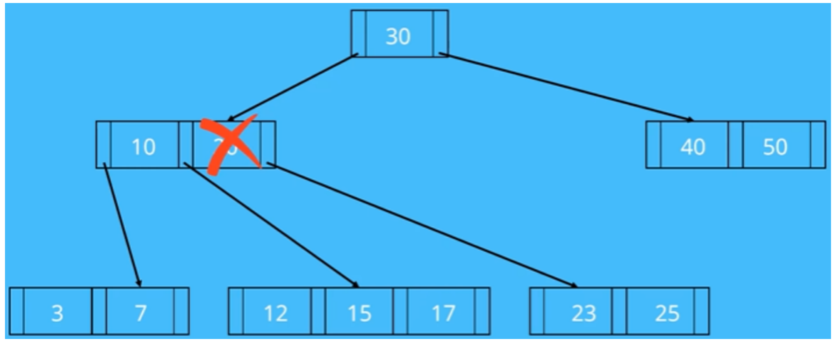
1. Löschen im Blatt:
   Die Mindestanzahl an Schlüsseln muss erhalten bleiben:
   a) Anzahl der Schlüssel im Knoten, in dem der zu löschende Wert gefunden wurde, ist > k: Schlüssel löschen
   b) Löschen erzeugt Unterlauf im Knoten (zu wenig Schlüssel): Generell müssen Schlüssel aus dem betroffenen Knoten und einem Nachbarknoten neu „verteilt“ werden, um B-Baum-Kriterium zu erhalten:
      1) Rotation: Kann durchgeführt werden, wenn der Nachbarknoten danach mehr als k Schlüssel behält
      2) Mischen: Muß genommen werden, wenn durch Rotation im Nachbarknoten ein Unterlauf entstünde



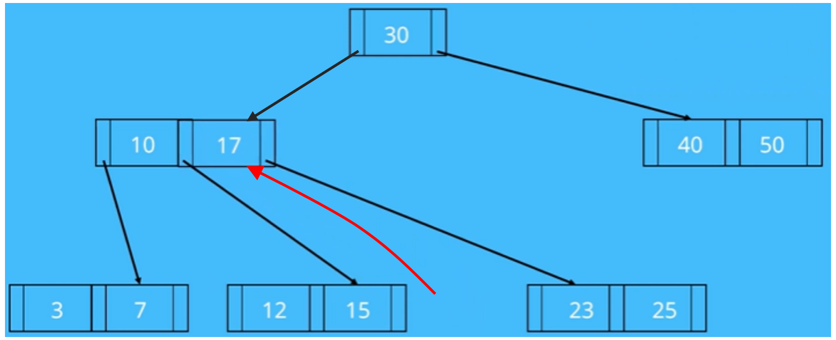
Wurzel muss nur 1 Schlüssel haben

(Selbst-)Balanzierte Bäume: Löschen der 20

2. Löschen im inneren Knoten: Der Wert kann nicht direkt gelöscht werden, da er Indikator für die an ihm „hängenden“ Unterbäume ist
 Er wird „getauscht“ durch
   a) Den symmetrischen Vorgänger (sym. Vorgänger ist der größte Blattknoten im linken Unterbaum bzw. der größte Wert in diesem Blattknoten) oder
   b) Den symmetrischen Nachfolger (sym. Nachfolger ist der kleinste Blattknoten im rechten Unterbaum, bzw. der kleinste Wert)

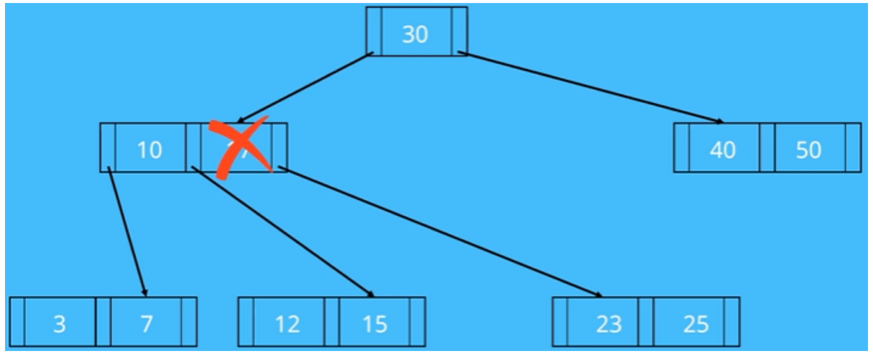


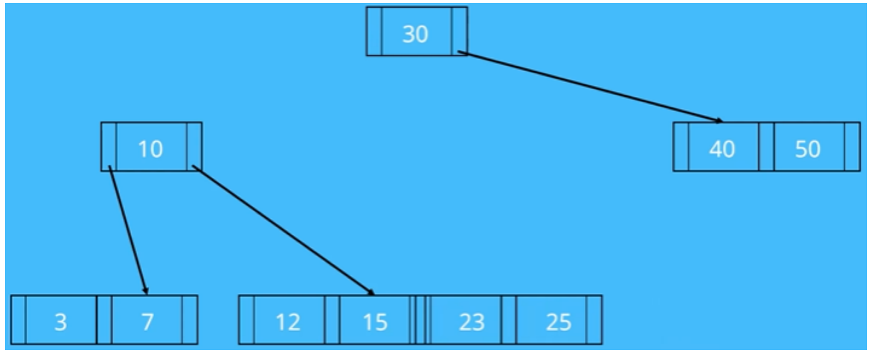
Mittlerer Nachfolger hat 3 Knoten.
Er ist Vorgänger des zu löschenden Knotens (a)



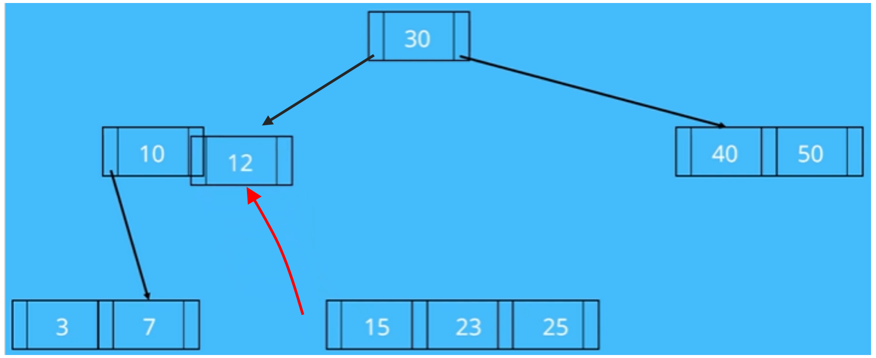
(Selbst-)Balanzierte Bäume: Löschen der 17

2. Löschen **im inneren Knoten**: Der Wert kann nicht direkt gelöscht werden, da er Indikator für die an ihm „hängenden“ Unterbäume ist
   Er wird „getauscht“ gegen
   a) Den symmetrischen Vorgänger (sym. Vorgänger ist der größte Blattknoten im linken Unterbaum bzw. der größte Wert in diesem Blattknoten) oder
   b) Den symmetrischen Nachfolger (sym. Nachfolger ist der kleinste Blattknoten im rechten Unterbaum, bzw. der kleinste Wert)





Rechter Nachfolger hat 4 Knoten.
Er ist Nachfolger des zu löschenden Knotens (b)



B*-Bäume

Die für den Einsatz wichtigste Variante des B-Baumes ist der B*-Baum.

Die Einträge (K_i, D_i, P_i) spielen in den inneren Knoten eine doppelte Rolle (Keys, Data, Pages=Nodes):

- Die zum Schlüssel K_i gehörenden **Daten D_i werden beim Schlüssel gespeichert**
- Der Schlüssel K_i dient als **Wegweiser im Baum**

In **B*-Bäumen** werden die zu speichernden **Daten (K_i, D_i) nur in den Blattknoten abgelegt** (für die K_i ergibt sich eine redundante Speicherung).
Heisst: die inneren Knoten gestatten einen schnellen Zugriff auf die Schlüssel –
die **Blätter allerdings enthalten alle Schlüssel mit ihren zugehörigen Daten in Sortierreihenfolge**.

Eine effiziente sequentielle Verarbeitung lässt sich durch Verkettung aller Blattknoten (sequence set) erreichen.

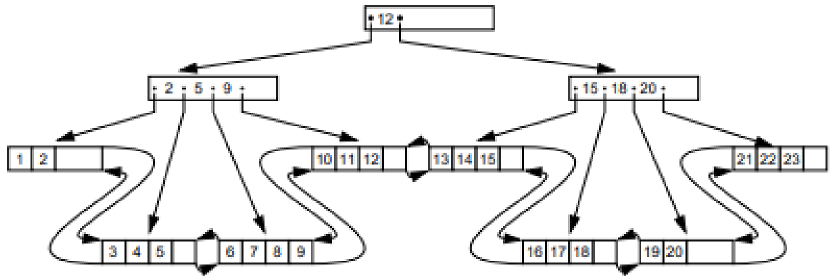
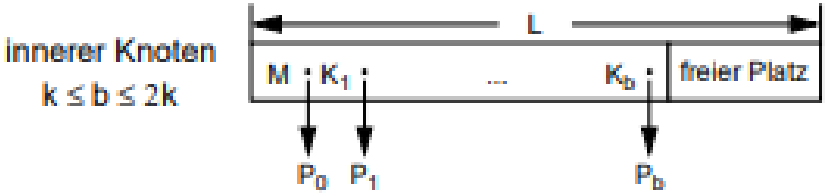


Bild 9: B*-Baum der Klasse τ(3,2,3)

B*-Bäume

Es sind hier 2 Knotenformate zu unterscheiden:

a) innerer Knoten:

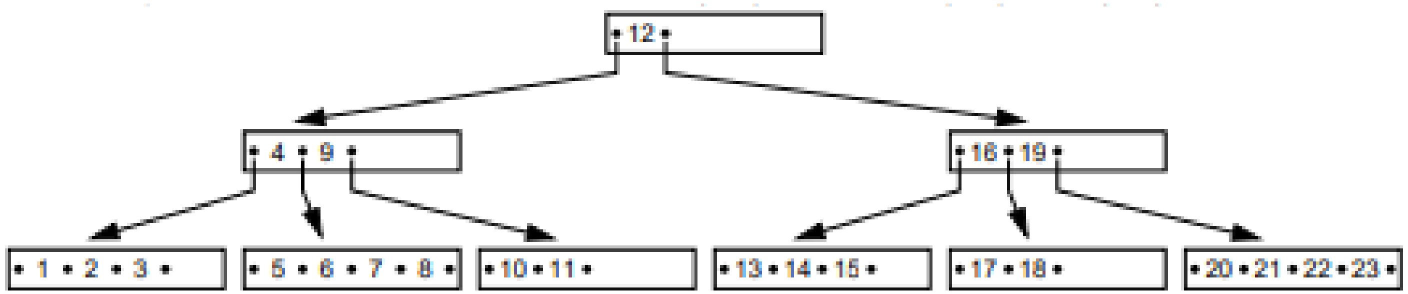


b) Blattknoten:

Die Pointer Prior und Next dienen zur Verkettung der Blattknoten. Das Feld M enthält die Zahl der aktuellen Einträge.

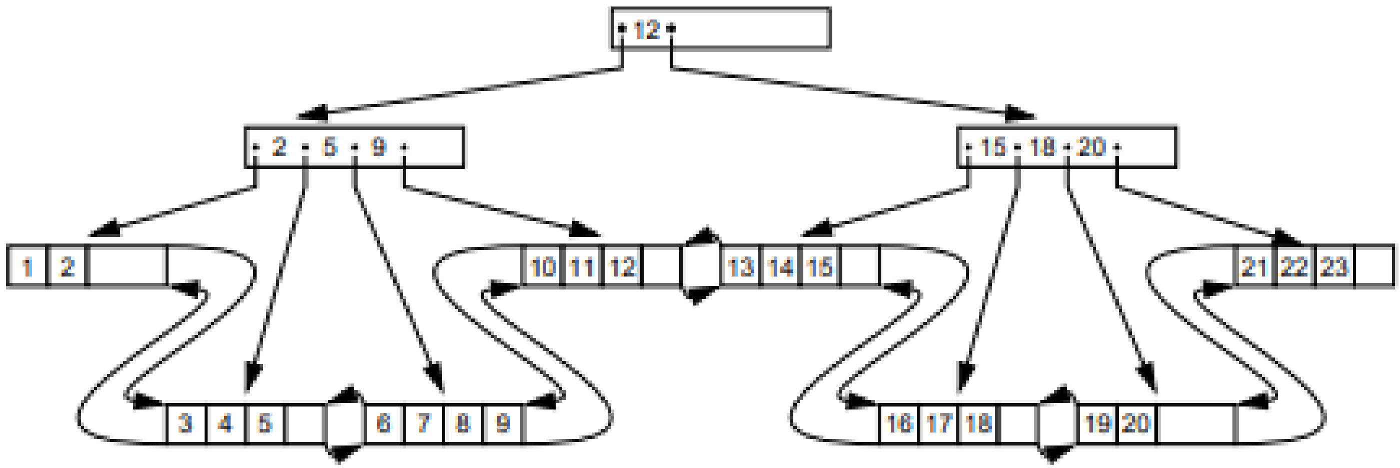
Da die inneren Knoten bei konstanter Länge L wesentlich weniger Information pro Eintrag enthalten, haben B*-Bäume eine wesentlich höhere Schlüsselzahl pro Knoten/ Seite, was zu geringerer Höhe des Baumes führt.

B*-Bäume: gleiche Schlüsselmenge



B-Baum der Klasse τ(2,3)

B-Baum (2-4 Keys/ Schlüssel, Höhe 3)



B*-Baum der Klasse τ(3,2,3)

B*-Baum,
Schlüssel in den Blättern redundant gespeichert
Klasse (k, k*, h*) k: keys der Innenknoten
k*: keys der Blätter
(Blattknoten hat mind. k*,
max. 2k* Schlüssel, Höhe h*)

B*-Bäume: Grundoperationen

Grundoperationen beim B*-Baum:

Aufzufassen wie eine gekettete sequentielle Datei von Blättern mit einem Indexteil, der selbst ein B-Baum ist.

Suche:
- direkte Suche kostet h* Einlagerungsschritte (Baumhöhe)

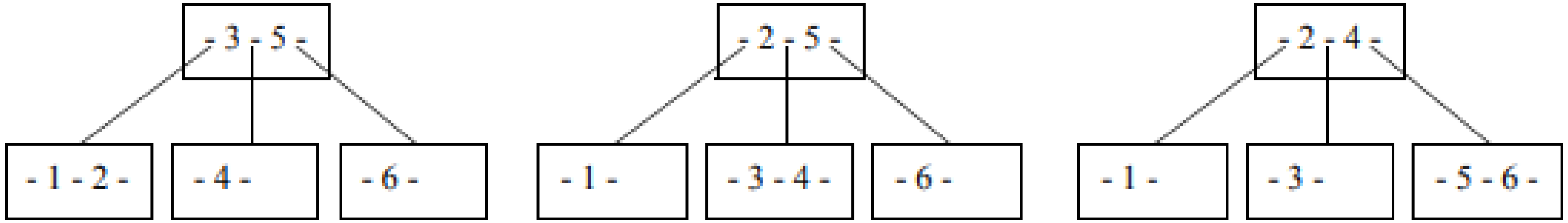
Löschen:
- nur die Daten in den Blättern müssen gelöscht werden

Vorteile des B*-Baumes:

- Strikte Trennung zwischen Datenteil und Indexteil
- Schlüssel in den inneren Knoten haben nur Wegweiserfunktion
- die redundant gespeicherten Schlüssel erhöhen den Speicherbedarf nur unwesentlich (< 1%)

Übung

Gegeben: B-Bäume der Klasse (1,2) bei Vorgabe folgender Schlüssel: 1, 2, 3, 4, 5, 6.



Wahr oder falsch?

1. Die Höhe des Baumes darf maximal 2 Level haben
2. Jeder Knoten muss mindestens 1 Schlüssel haben
3. Knoten und Schlüssel sind gleichbedeutend
4. Die linken Kind-Schlüssel eines Schlüssels müssen kleiner, die rechten größer sein
5. Ein Knoten hat bis zu einem Nachfolgeknoten mehr, als er Schlüssel hat
6. Der Wurzelknoten könnte in obigem Beispiel auch 1 Schlüssel haben
7. Diese Baumklasse kann 10 Schlüssel aufnehmen