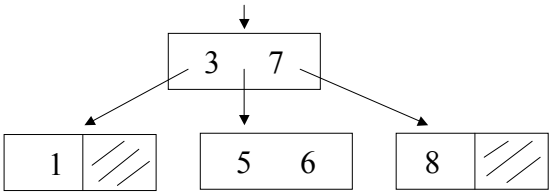
Kap. 6.2 Binäre B-Bäume

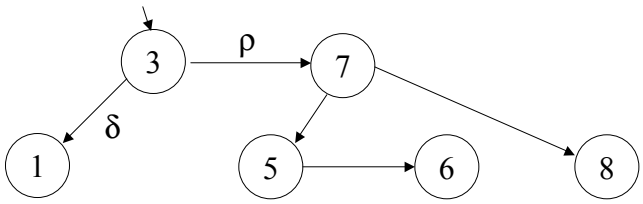
Ullmann: (2, 3) - Bäume

τ (1, h) anpassen für HS ?



1

darstellen als:



Def: ***binärer B-Baum*** ist binärer Baum
mit δ-Kanten u. ρ-Kanten und:

i) alle linken Kanten sind δ
ii) jeder Pfad von Wurzel zu Blatt hat gleichviel δ-Kanten
iii) rechte Kanten können. ρ oder δ sein, aber keine aufeinanderfolgenden. ρ -Kanten
iv) Knoten mit ausgehender δ-Kante hat zwei Nachfolger

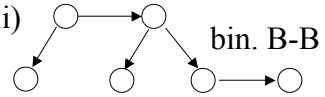
2

Vergleich mit AVL ?

Hilfsinfo: Kantenart bzw. Balanzierung?

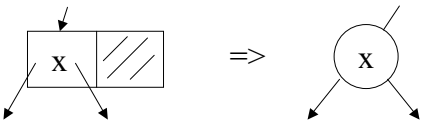
Lemma: Die Klassen binäre B-Bäume u. AVL-Bäume sind nicht vergleichbar

i)  AVL

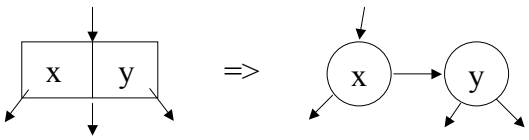
ii)  bin. B-B

3

Algorithmen: direkt von B-Baum übernehmen
nur Wechsel für Seitendarstellung:



2 Verweise



3 Verweise

4

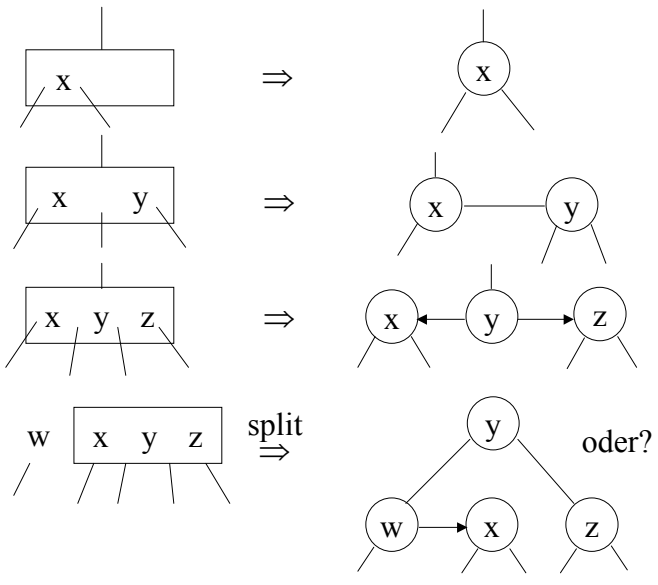
Symmetrische binäre B-Bäume

Unsymmetrie bzgl. ρ Kanten aufheben,

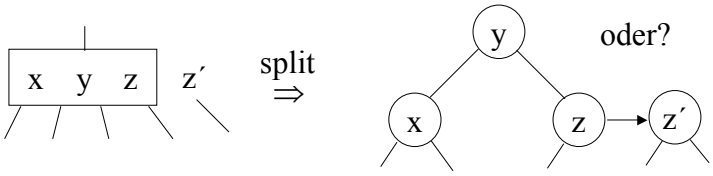
$\xrightarrow{\rho}$, $\xleftarrow{\rho}$, keine aufeinanderfolgenden ρ -Kanten;

bzw. Seite mit 1, 2 oder 3 Indexelementen:

5



6



Lemma: Die symm. binären B-Bäume sind echte Oberklasse der AVL-Bäume.

Hinw.: Vergleich der Balancierungs-Kriterien?

Verallg.: Erlaube c > 0 aufeinanderfolgende ρ-Kanten

7

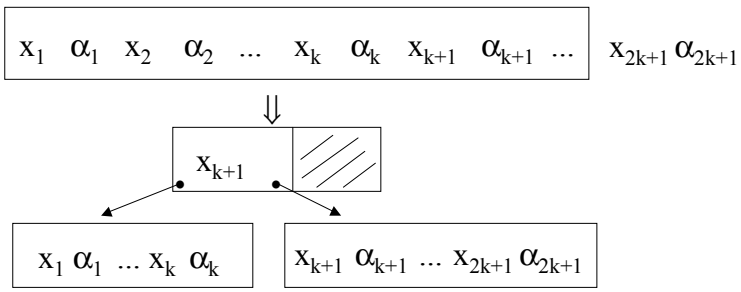
Kap. 6.3 B*-Bäume

Standard Zugriffsstruktur in allen heutigen DB-Systemen.

Doppelrolle von (x, α):

i) Identifikation von (x, α) über x

ii) Steuerung der Suche durch x



8

Spalten von Blatt: übernehme x_{k+1} in Vater
x_{k+1} redundant gespeichert

Spalten von internem Knoten:
wie bisher: x_{k+1} in Vater, ohne Redundanz

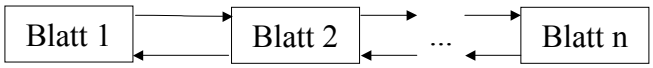
Eigenschaften, Vorteile von B*-Bäumen:

i) (x, α) nur auf Blättern
ii) redundante Info in Zwischenknoten nur 1-2 % des gesamten Speichers
iii) in Zwischenknoten mehr (x, p) als (x, α, p)

9

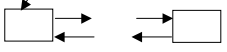
$\Rightarrow$ höherer Verzweigungsgrad
niedrigerer Baum
schnellere Suche u. Manipulation

iv) einfacherer Löschalg.: (x, α) auf Blatt!
v) Blätter des B*-B ~ sortierte, sequentielle Datei



vi)

reiner Index Teil
= B-Baum von Schlüsseln



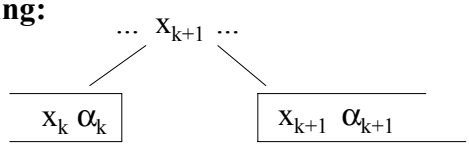
sequentielle sortierte Datei

10

Kap. 6.4 Präfix B-Bäume

Grundidee: In Index-Teil eines B*-B nur minimale Info, um Suche zu steuern.

Spaltung:



$x < x_{k+1} \leq y$

Def.: ***Separator*** zwischen x_k und x_{k+1} ist beliebiges s mit $x_k < s \leq x_{k+1}$

Ziel: möglichst kurze Separatoren im Indexteil, z.B. für

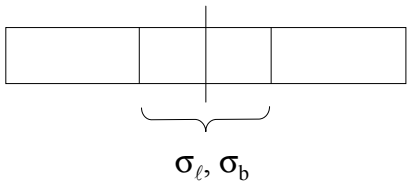
Bigbird, Burt, Cookiemonster, Ernie, Snuffleopogus
Bu C D, E F,G, ..., S

Präfix Eigenschaft: Seien $x, y \in X^+$
und < lexikographisch, $x < y$.
$\Rightarrow \exists!$ kürzester Präfix $\overline{y}$ von y
mit: a) $x < \overline{y} \leq y$
b) kein anderer Separator ist kürzer.

Def.: Ein ***einfacher Präfix-B-Baum*** ist ein B*-Baum, in dem bei Blatt-Split die kürzesten Präfix-Separatoren verwendet werden.

Hinw.: - variabel lange Separatoren in Index
- **delete** darf „kürzeste Separatoren“ verletzen.

Spaltintervalle:



ℓ : leaves
b: branching

σ_ℓ, σ_b

Hinw.: kleine Intervalle, gute Verbesserung

Algorithmen: wie B* plus Berechnung des kürzesten Separators

13

Echte Präfix-B-Bäume: $x_i \; p_i \; x_{i+1}$

$\Rightarrow \quad x_i = yz_1$
$x_{i+1} = yz_2$

$\forall\, w \in K(p_i)$: $x_i \le w < x_{i+1}$, d.h. $w = yw'$
$\Rightarrow$ in T (p_i) ist y überflüssig,
ergibt sich aus Suchpfad

weitere Themen: siehe Spezialliteratur u. Vorl. DB-Systeme.
- Parallelverarbeitung
- Chiffrierung

14

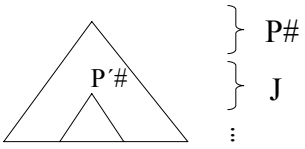
Beispiel: Anwendung von Präfix B-Bäumen

1. Zeitreihen von Wertpapierkursen:

P# Jahr Monat Tag Zeit Kurs

zusammenges. Schlüssel

(x, α): x = (P#, J, M, T, Z); α = Kurs



2. Zeitreihen von Meßwerten, z.B. in Energie-Verteilernetz, viele Meßpunkte

15

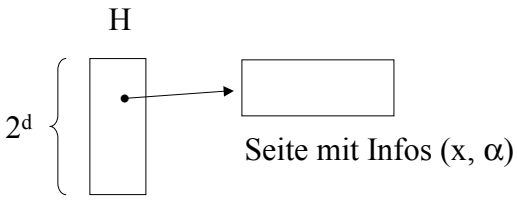
3. Komplexe Systeme ...

Marke	Typ	ccc
BMW	700	3000
		⋮
		3500
	300	⋮

Beobachtung: lange zusammengesetzte Schlüssel haben lange gemeinsame Präfixe

16

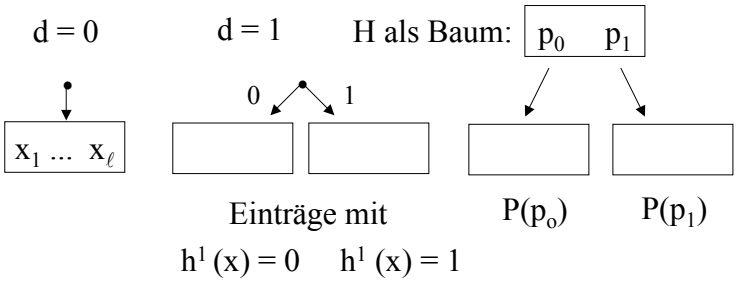
Kap. 6.5 Erweiterbares Hashing



17

Idee: Splits ähnlich wie bei B-Bäumen:

h(x) Hashfunktion; $h^d(x)$: vordere d Bits

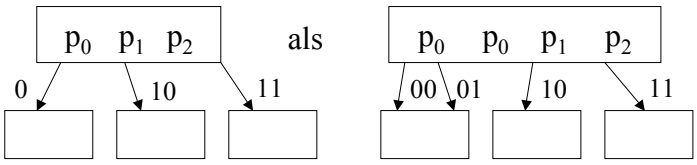


also: (x, α) auf $P(p_o)$ falls $h^1(x) = 0$

" $P(p_1)$ " $h^1(x) = 1$

18

Spaltung von P (p_1): in P (p_1) u. P (p_2)
mit Verdoppelung von H:



Spaltung von P (p_1) in P (p_1) u. P (p_3):

Schlüsselverteilung von P (p_1) entspr.

$h^3(x)$ = 100 auf P (p_1)
101 " P (p_3)

19

H: p0 p0 p0 p0 p1 p3 p2 p2

000 001 ... 111

Spaltungen von p_0, p_2, p_1 ?

Search x:
1. h (x)
2. Hole H-Seite entspr. h^d
3. Verweis aus H-Seite
4. Hole Datenseite
5. Suche (x, α) auf Datenseite

20

Insert (x, α): 1. Suche u. hole Blatt
2. füge (x, α) ein
3. bei Überlauf spalte
4. ändere H-Seite,
bei Einzelverweis:
verdopple H !!

Delete (x, α): ...
Füllung der Datenseiten in H speichern,
wegen Konkatenation?

21

Probleme:

1. Speicherausnutzung: entsprechend h^d,
keine Mindestgarantie
2. Redundanz der p_i in H, Balanzierung?
3. keine Verarbeitung nach <
4. keine Ausgleichstechnik
5. Streufunktion?

Vorteil gegenüber B-B:
nur 2 Plattenzugriffe

22