

Kap. 4.4: B-Bäume Kap. 4.5: Dictionaries in der Praxis



Professor Dr. Petra Mutzel
Lehrstuhl für Algorithm Engineering, LS11
Fakultät für Informatik, TU Dortmund

13./14. VO DAP2 SS 2009 2./4. Juni 2009

2. Übungstest

- **Termin:** Di **16. Juni** 2009 im AudiMax, Beginn: 12:15 Uhr (bitte um 12:00 Uhr anwesend sein)
- **Dauer:** 30 Minuten
- **Stoff:** aus VO-Folien, Skript und Übungen bis inkl. B-Bäume (schwerpunktmäßig ab Heap-Sort inkl.)
- Dann ab ca. 12:50 Uhr: Vorlesung bis 13:45 Uhr

Motivation

„Warum soll ich heute hier bleiben?“

Mal ganz andere Suchbäume

„Was ist daran Besonderes?“

praxis-relevant für Datenbanken

Überblick

- Einführung von B-Bäumen
- Eigenschaften von B-Bäumen
- Realisierung und Analyse
- Varianten von B-Bäumen

B-Bäume

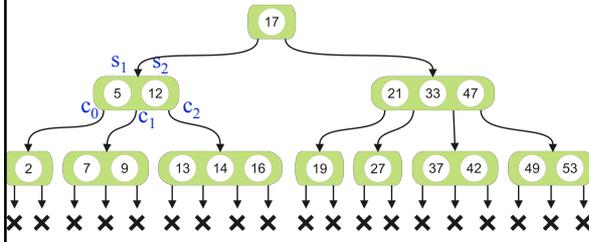
- Einführung von Rudolf Bayer und Eduard M. McCreight 1972
- Datenstruktur zur Verwaltung von Indizes für das relationale Datenmodell von Edgar F. Codd 1972
- → Entwicklung des ersten SQL-Datenbanksystems System R bei IBM
- B-Bäume sind ausgeglichene Mehrwegbäume
- **Idee:** jeder Knoten eines B-Baums der Ordnung m besitzt zwischen $\lceil m/2 \rceil$ und m Kinder.

Motivation für B-Bäume

- Minimierung von externen Speicherzugriffen (Festplatten)
- Man wählt i.A. die Ordnung m gerade so gross, dass jeweils alle Schlüssel eines B-Knotens genau einer *page* (Blockgröße bei einem Lesezugriff) entsprechen
- typische Größen, z.B. $m=100$ oder $m=5000$
- Beispiel:
 - 1 Mio. Datensätze, $m=100$: Höhe eines B-Baumes: 2
 - also nur 2 Speicherzugriffe (bei Wurzel im Hauptspeicher)
 - im Vergleich hierzu Binärbäume: Höhe: 20

B-Bäume

- Ein Knoten mit $k+1$ Zeigern c_0, c_1, \dots, c_k auf die Unterbäume besitzt k Schlüssel s_1, \dots, s_k
- Diese sind in sortierter Reihenfolge gespeichert und trennen die jeweiligen Unterbäume.



Definition B-Bäume

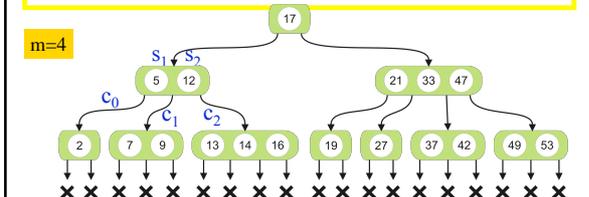
- Ein **B-Baum der Ordnung $m > 2$** ist ein Baum mit folgenden Eigenschaften (1)-(6):

- (1) Kein Schlüsselwert kommt mehrfach vor
- (2) Für jeden Knoten α mit $k+1$ Zeigern c_0, c_1, \dots, c_k auf Unterbäume gilt:
 - a) α hat k Schlüssel s_1, \dots, s_k , wobei gilt: $s_1 < \dots < s_k$
 - b) Für jeden Schlüssel s im Teilbaum mit Wurzel c_i gilt: $s_i < s < s_{i+1}$, wobei $s_0 = -\infty$ und $s_{k+1} = +\infty$
 - c) Ein Schlüssel s_i wird als Trennschlüssel der Teilbäume mit Wurzel c_{i-1} und c_i bezeichnet.

→ jeder Schlüssel entweder in Blatt oder Trennschlüssel

Definition B-Bäume ff

- (3) Der Baum ist leer, oder die Wurzel hat mindestens 2 Zeiger auf Kinder.
- (4) Jeder Knoten mit Ausnahme der Wurzel enthält mindestens $\lceil m/2 \rceil$ Zeiger.
- (5) Jeder Knoten hat höchstens m Zeiger.
- (6) Alle Blätter haben die gleiche Tiefe.



Existenz von B-Bäumen

Theorem: Für jede beliebige Menge von Schlüsseln und jedes beliebige $m > 2$ existiert ein gültiger B-Baum der Ordnung m .

Beweisidee: Besteht die Schlüsselmenge aus weniger als m Schlüsseln, dann besteht der B-Baum einfach aus einem einzigen Wurzelknoten, der alle Schlüssel enthält. Sonst: benutze folgendes Lemma.

Beweisidee der Existenz von B-Bäumen

Lemma: Gegeben sei eine Menge M aus n Schlüsseln und $m \geq 3$ mit $n \geq m$. Wir setzen $l := \lfloor n/m \rfloor$.

Wir können l Trennschlüssel aus M so auswählen, dass die übrigen Schlüssel in $l+1$ Teilmengen S_0, \dots, S_l zerfallen, die jeweils eine gültige Befüllung eines Knotens in einem B-Baum der Ordnung m darstellen.

$n=33$
 $m=4$ x
 $l=8$ Durchschnittliche Größe einer Teilmenge: $(n+1)/(l+1)-1$

Weiter mit Beweis zu Theorem: Diese $l+1$ Teilmengen S_i bilden die Blätter unseres B-Baums. Wiederhole diese Aufteilung rekursiv für die Trennschlüssel.

Größe von B-Bäumen

Lemma: Die Größe eines B-Baums ist linear in n , der Anzahl der Schlüssel.

Beweis:

- Jeder Schlüssel kommt im Baum vor, deshalb: Größe ist in $\Omega(n)$.
- Sei t die Anzahl der Knoten im Baum. Da jeder Knoten mindestens einen Schlüssel enthält, gilt $t = O(n)$.
- Da in jedem Knoten mit k Schlüsseln gilt, dass er $k+1$ Zeiger enthält, haben wir insgesamt $n+t = O(n)$ viele Zeiger.
- Also gezeigt: $\Omega(n)$ und $O(n)$.

Minimale Höhe von B-Bäumen

Lemma: Die minimale Höhe eines B-Baums mit n Schlüsseln ist $\lceil \log_m(n+1) \rceil - 1$

Beweis: Ein B-Baum hat die kleinste Höhe, wenn alle Knoten $m-1$ Schlüssel enthalten.

Bei Höhe h des Baums enthält er dann m^h Blätter.

Ein B-Baum mit $h+1$ Blättern hat h Schlüssel in inneren Knoten.

Insgesamt: $n = m^h(m-1) + (m^h-1) = m^{h+1} - 1$

Schlüssel in Blättern

Schlüssel in inneren Knoten

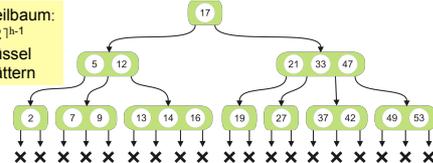
Maximale Höhe von B-Bäumen

Lemma: Die maximale Höhe eines B-Baums mit n Schlüsseln ist $\lfloor \log_{\lceil m/2 \rceil}((n+1)/2) \rfloor$

Beweis: Ein B-Baum hat die größte Höhe, wenn die Wurzel nur einen einzigen Schlüssel enthält und die beiden Teilbäume und deren Teilbäume (rekursiv) jeweils nur $\lceil m/2 \rceil - 1$ Schlüssel.

Insgesamt bei Höhe h : $n = 1 + 2(\lceil m/2 \rceil^h - 1) = 2 \lceil m/2 \rceil^h - 1$

linker Teilbaum:
 $\lceil m/2 \rceil^{h-1}$
Schlüssel
in Blättern



Datenstruktur eines B-Baums

struct BTreeNode

```
int k // Anzahl der Schlüssel
KeyType key[1..k]
DataType data[1..k]
BTreeNode child[0..k]
```

Interne Repräsentation eines B-Baums:
BTreeNode root

Implementierung von SEARCH(r,s) in B-Bäumen

Im Wesentlichen wie im Binärbaum:

1. Vergleiche s mit allen Schlüsseln in der Wurzel (des Teilbaums)
2. Falls gefunden: STOP!
3. Sonst: Folge dem Zeiger, der sich zwischen den beiden im Wurzelknoten benachbarten Schlüsseln befindet. Gehe zu 1.
4. Ausgabe: nicht gefunden!

SEARCH(p,s)

Suche in Baum mit Wurzel p den Schlüssel s

- (1) If $p == \text{NULL}$ then return NULL
- (2) else {
- (3) $i := 1$
- (4) while $i \leq p.k$ and $s > p.key[i]$ do
- (5) $i := i + 1$
- (6) if $i \leq p.k$ and $s == p.key[i]$ then
- (7) return $p.data[i]$
- (8) else return SEARCH($p.child[i-1], s$)
- (9) }

Analyse von SEARCH(r,s)

- In Implementierung: lineare Suche
- besser: binäre Suche
- Aber asymptotisch: beide Male Suche innerhalb eines Knotens konstant (wegen m konstant)
- Insgesamt: Laufzeit: $O(\text{Höhe von B})$

Einfügen eines Schlüssels in einen B-Baum

• Baum vor dem Einfügen von 10, $m=3$

tu technische universität dortmund Petra Mutzel DAP2 SS09 19

Einfügen in B-Baum

Blatt wird zu groß: hat nun 3 Schlüssel!

tu technische universität dortmund Petra Mutzel DAP2 SS09 20

Einfügen in B-Baum

nicht erlaubt:

tu technische universität dortmund Petra Mutzel DAP2 SS09 21

Einfügen in B-Baum

Blatt wird zu groß: hat nun 3 Schlüssel!

tu technische universität dortmund Petra Mutzel DAP2 SS09 22

Einfügen in B-Baum

Elternknoten wird zu groß:

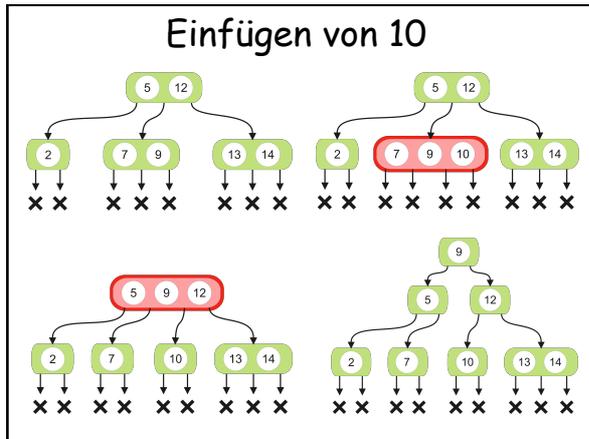
Aufspalten: mittlerer Schlüssel geht in den Elternknoten

tu technische universität dortmund Petra Mutzel DAP2 SS09 23

Einfügen in B-Baum

B-Bäume wachsen „nach oben“

Aufspalten der Wurzel → neue Wurzel



Implementierung von INSERT(r,s) in B-Bäumen

1. Einfügeposition suchen: SEARCH(r,s)
2. Einfügen in Blatt
3. Wiederhole
 - Falls dieser Knoten zu viele Schlüssel enthält, dann **Aufspalten**: der mittlere Schlüssel wird zum Elterknoten verschoben. bis Knoten nicht mehr zu groß.
4. Falls wir die Wurzel aufspalten mußten, setze: neue Wurzel.

tu technische universität dortmund Petra Mutzel DAP2 SS09 26

Analyse von INSERT(r,s)

Laufzeit: $O(\text{Höhe des Baums}) = \Theta(\log n)$

tu technische universität dortmund Petra Mutzel DAP2 SS09 27

Entfernen in B-Bäumen

- B-Baum der Ordnung 5
- Baum vor dem Entfernen von 33

Idee: überschreiben mit einem Schlüssel aus dem Blatt:

Problem: zu wenig Schlüssel

tu technische universität dortmund Petra Mutzel DAP2 SS09 28

Entfernen in B-Bäumen

- Entfernen von 33

Lösung: Rotation

Problem: zu wenig Schlüssel

tu technische universität dortmund Petra Mutzel DAP2 SS09 29

Entfernen in B-Bäumen

- B-Baum nach der Rotation

Problem: zu wenig Schlüssel

Lösung: Rotation

tu technische universität dortmund Petra Mutzel DAP2 SS09 30

Problem: zu wenig Schlüssel nach Entfernen von 33 → 31

Lösung: Rotation

tu technische universität dortmund Petra Mutzel DAP2 SS09 31

Rotation

Entfernen in B-Bäumen

Beispiel 2:

- Entfernen von 7

tu technische universität dortmund Petra Mutzel DAP2 SS09 35

Entfernen in B-Bäumen

Problem: zu wenig Schlüssel
Rotation nicht möglich ⊗
Lösung: Verschmelzen

tu technische universität dortmund Petra Mutzel DAP2 SS09 36

Entfernen in B-Bäumen

Problem: wieder zu wenig Schlüssel
Lösung: Verschmelzen

- Entfernen von 7

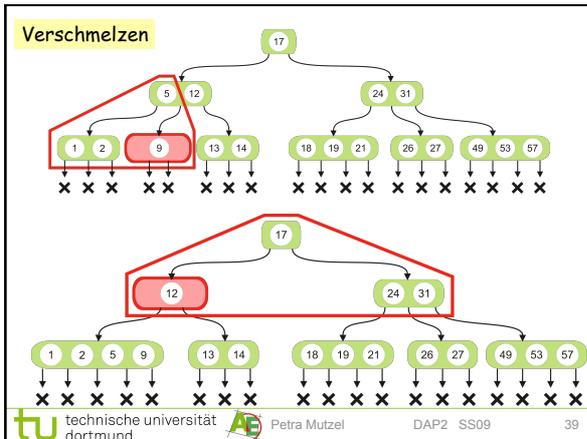
tu technische universität dortmund Petra Mutzel DAP2 SS09 37

Entfernen in B-Bäumen

- Entfernen von 7

B-Bäume schrumpfen „von oben“

tu technische universität dortmund Petra Mutzel DAP2 SS09 38



Entfernen in B-Bäumen

- (1) Den zu entfernenden Schlüssel s suchen
- (2) Falls s in einem Blatt ist: einfach entfernen
- (3) Sonst: Suche direkten Vorgänger (dieser ist Blatt), überschreibe s mit dessen Schlüssel und entferne das Blatt.
- (4) Falls dieses Blatt β nun zu wenige Schlüssel besitzt, dann entweder (5) oder (6)
- (5) **Reparatur A: Rotation:** Falls ein Geschwisterblatt von β genug Schlüssel enthält, dann wird ein Schlüssel daraus verwendet um die Größe von β wiederherzustellen.
- (6) **Reparatur B: Verschmelzen:** Sei γ ein Geschwisterblatt von β . Wir verschmelzen γ , β und ihren Trennschlüssel in einen neuen großen Knoten
- (7) Setze die Reparatur rekursiv fort (evtl. nach (6) nötig)

Diskussion zum Entfernen (1)

- Warum ist Reparatur B korrekt?
- Wir wissen, dass
 - β nur $\lceil m/2 \rceil - 2$ Schlüssel enthält
 - beide Geschwister (bzw. γ) von β nur $\lceil m/2 \rceil - 1$ Schlüssel enthalten
 - damit: der neue Knoten höchstens $m-1$ Schlüssel besitzt.
 - Denn: $\leq \lceil m/2 \rceil - 2 + \lceil m/2 \rceil - 1 + 1$ Schlüssel
 - Fall: m ungerade: $= 2((m+1)/2) - 2 = m-1$
 - Fall: m gerade: $= 2(m/2) - 2 = m-2$

Analyse von Delete(r,s)

Laufzeit: $\Theta(\text{Höhe des Baums}) = \Theta(\log n)$

Diskussion zum Entfernen (2)

- Alternative zu **Reparatur B** wäre eine erweiterte Rotation:
- Falls ein Geschwister γ' von einem Geschwisterknoten γ von β genügend Schlüssel besitzen würde, wäre eine doppelte Rotation denkbar: zunächst von γ' nach γ , danach von γ nach β .
- Also wäre es denkbar solange alle Nachbarn abzusuchen, bis ein Knoten gefunden wird.
- Die Zeit hierfür wäre $O(m)$.
- Aber: **Reparatur B** ist besser, weil Zeit: $O(\log n)$, und in der Praxis $\log n < m$

Varianten zum Einfügen/Entfernen

- Wir durchlaufen für diese Operationen den Baum einmal von oben nach unten, einmal von unten nach oben.
- Es existieren auch Alternativen bei denen beim Einfügen und Entfernen der Baum jeweils nur einmal von oben nach unten durchwandert wird:
- Hierbei werden die besuchten Knoten des Baumes gleich „auf Verdacht“ gespalten (falls sie groß sind) bzw. verschmolzen (falls sie klein sind).

Varianten von B-Bäumen

- Alternative Realisierungen garantieren jeweils eine 2/3 Füllung der Knoten.
- oder erweitern den Baum durch Zeiger zwischen den Geschwistern.
- oder akkumulieren „falsche“ Balancierungen auf und reparieren diese später gemeinsam.
- Es gibt immer wieder neue gute Einfälle!
- Auch in der praktischen Umsetzung gibt es immer wieder Neues!

bis heute: aktuelles Forschungsthema

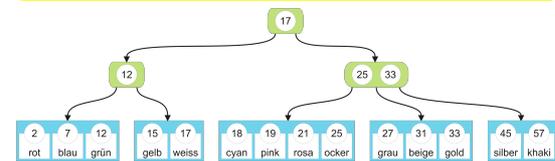
Spezielle B-Bäume

- 2-3 Baum = B-Baum der Ordnung 3 [Hopcroft '70]
- 2-3-4 Baum = B-Baum der Ordnung 4
- Rot-Schwarz-Baum = 2-3-4 Baum in dem große Knoten durch binäre Teilbäume simuliert werden

B⁺-Bäume

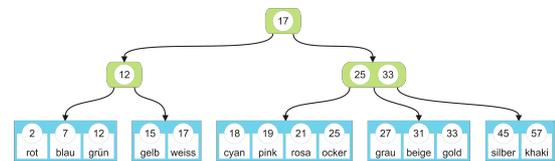
In Literatur auch: „B⁺-Bäume“

- Im Wesentlichen B-Bäume bei denen die Datensätze nur in den Blättern stehen.
- Die inneren Knoten enthalten ausschliesslich Schlüssel (und Zeiger) zur Suchsteuerung.



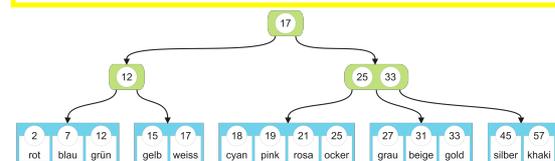
B⁺-Bäume

- Vorteil: Ein innerer Knoten kann mehr Schlüssel aufnehmen (da er keine info-Daten speichern muß)
- Dadurch wird Verzweigungsgrad erhöht und die Höhe nimmt ab.



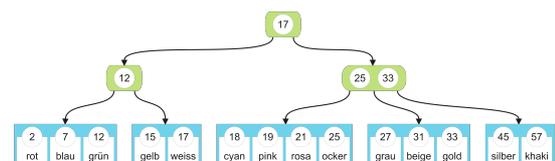
B⁺-Baum der Ordnung (m,m⁺)

- Die inneren Knoten besitzen zwischen $\lceil m/2 \rceil - 1$ und $m-1$ viele Schlüssel.
- In jedem Blatt sind zwischen $\lceil m^+/2 \rceil - 1$ und $m^+ - 1$ viele Schlüssel-Daten-Paare gespeichert.
- Dies führt zu besseren Externspeicherbedingungen.



Operationen in B⁺-Bäumen

- Suchen, Einfügen, Entfernen: sehr ähnlich wie in B-Bäumen.



Kap. 4.6: Dictionary-Realisierungen in der Praxis

Quelle: K. Mehlhorn, S. Näher:
LEDA: A platform for combinatorial and geometric computing,
Cambridge University Press, 1999, S. 127

Untersuchte Realisierungen

- bin_tree: binäre Suchbäume
- rs_tree: zufällige binäre Suchbäume bin_tree ∪ heap für zufällige prio-Werte
- avl_tree: AVL-Bäume
- bb_tree: B-Bäume der Ordnung m=?
- ab_tree: 2-3-4 Bäume
- rb_tree: rot-schwarz Bäume Simulationen von 2-3-4 Bäumen auf binären Suchbäumen
- skiplist: Skiplisten

