

Kap. 4.3: AVL-Bäume Kap. 4.4: B-Bäume

Professor Dr. Petra Mutzel
Lehrstuhl für Algorithm Engineering, LS11

12. VO

16. Mai 2006

Motivation

„Warum soll ich heute hier bleiben?“

Müssen Sie nicht: DEMO

„Was gibt es heute Besonderes?“

Schönes Java-Applet!

2

Überblick

- Wiederholung AVL-Bäume

- Java-Applet AVL-Bäume

- Einführung von B-Bäumen

- Eigenschaften von B-Bäumen

3

Definitionen für AVL-Bäume

- **Höhe eines Knotens:** Länge eines längsten Pfades von v zu einem Nachkommen von v
- **Höhe eines Baumes:** Höhe seiner Wurzel
- **Höhe eines leeren Baumes:** -1
- **Balance eines Knotens:** $\text{bal}(v) = h_2 - h_1$, wobei h_1 und h_2 die Höhe des linken bzw. rechten Unterbaumes von v
- v heißt **balanciert:** wenn $\text{bal}(v) \in \{-1, 0, +1\}$, sonst heißt v unbalanciert
- **AVL-Baum:** Binärer Suchbaum, bei dem alle Knoten balanciert sind

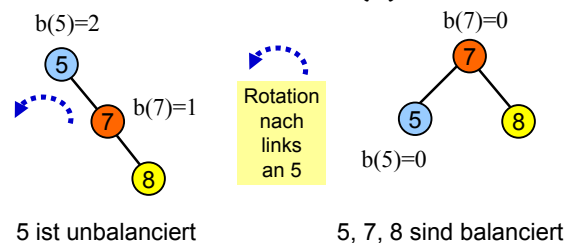
4

Implementierung der Operationen Insert und Delete

- **Idee:**
- zunächst wie bei den natürlichen binären Suchbäumen.
- Falls Baum nicht mehr balanciert ist, dann wissen wir, dass ein Knoten u auf dem Suchpfad existiert mit $\text{bal}(u) \in \{-2, +2\}$
- Wir rebalancieren den Baum nun an dieser Stelle, so dass er danach wieder balanciert ist.

5

Rotationen (1)



7

Rotationen (2)

$b(5)=2$

$b(7)=0$

5 ist unbalanciert

Rotation nach links an 5

falsch: die 6 muss mitwandern

5, 7, 8 sind balanciert

8

Rotationen (3)

$b(5)=2$

$b(7)=0$

5 ist unbalanciert

Rotation nach links an 5

so korrekt! Warum eigentlich?

5, 7, 8 sind balanciert

Die Suchbaumeigenschaft bleibt bei einer Rotation erhalten.

Nach einer Rotation entsteht wieder ein binärer Suchbaum

Achtung: dies ist nicht so wenn gleiche Schlüssel enthalten sind

Rotationen (4)

$b(5)=2$

$b(8)=-1$

5 ist unbalanciert

Rotation nach links an 5

falsch: Das war keine Rotation!!!

12

Rotationen (5)

$b(5)=2$

$b(8)=-1$

5 ist unbalanciert

Rotation nach links an 5

Diese Rotation hat nichts genützt!

14

Rotationen (6)

$b(5)=2$

$b(8)=-1$

Rotation nach rechts an 8

$b(5)=2$

$b(7)=1$

Rotation nach links an 5

$b(7)=0$

$b(5)=0$ $b(8)=0$

Doppelrotation Rechts-Links notwendig!

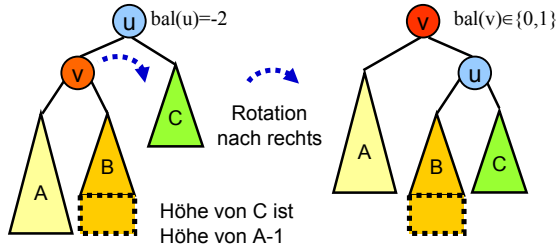
16

Doppelrotationen

18

1. Fall: $\text{bal}(u) = -2$

- Sei v das linke Kind von u (existiert!)
- **Fall 1.1:** $\text{bal}(v) \in \{-1, 0\}$:

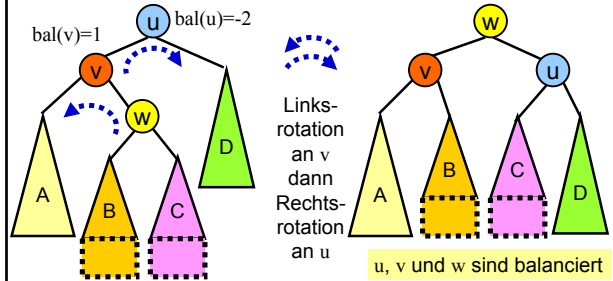


- Suchbaumeigenschaft bleibt erhalten; u und v sind balanciert
- Für die Knoten unterhalb hat sich nichts geändert.

19

1. Fall: $\text{bal}(u) = -2$ ff.

- Sei v das linke Kind von u (existiert!)
- **Fall 1.2:** $\text{bal}(v) = +1$:



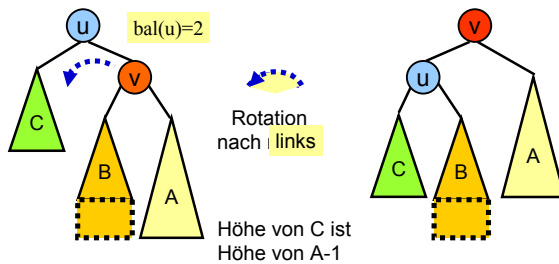
Höhe von (B oder C) und von D ist Höhe von A

u, v und w sind balanciert

20

2. Fall: $\text{bal}(u) = +2$

- Sei v das rechte Kind von u (existiert!)
- **Fall 1.1:** $\text{bal}(v) \in \{0, 1\}$: Rotation nach rechts an u

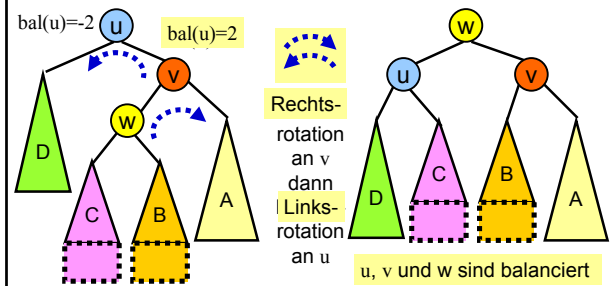


- Inorder-Reihenfolge bleibt erhalten und u und v sind balanciert
- Für die Knoten unterhalb hat sich nichts geändert.

21

2. Fall: $\text{bal}(u) = +2$ ff.

- Sei v das rechte Kind von u (existiert!)
- **Fall 1.2:** $\text{bal}(v) = -1$: Rechts-Links rotation an u



Höhe von (B oder C) und von D ist Höhe von A

u, v und w sind balanciert

22

Rebalancierung ff.

- Bestimme nun den nächsten unbalancierten Knoten maximaler Tiefe. Diese Tiefe ist nun kleiner als vorher.
- Wiederhole die Rebalancierung (AVL-Ersetzung) solange, bis alle Knoten balanciert sind.

- Das Verfahren konvergiert nach $O(h(T))$ Iterationen zu einem gültigen AVL-Baum.

23

Rebalancierung ff.

- Die Insert-Operation unterscheidet sich nur insofern von der Delete-Operation, dass hier ein einziger Rebalancierungsschritt genügt.
- Bei der Delete-Operation hingegen kann es sein, dass mehrere Rebalancierungsschritte notwendig sind.

Implementierungen:

24

RotateRight(u)

- (1) $v := u.left$ //Bestimme v
- (2) $u.left := v.right$
- (3) $v.right := u$
- (4) $u.height := \max(h(u.left), h(u.right)) + 1$
- (5) $v.height := \max(h(v.left), h(u)) + 1$
- (6) return(v) //neue Wurzel

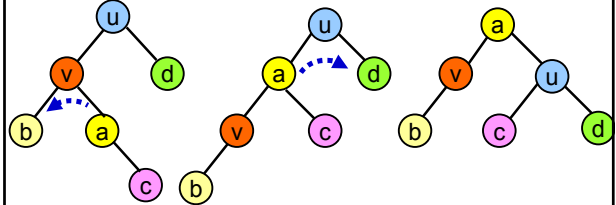


25

RotateLeftRight(u)

- (1) $u.left := \text{RotateToLeft}(u.leftson)$
- (2) return(RotateToRight(u))

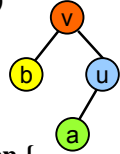
Beispiel:



26

INSERTAVL(p,q)

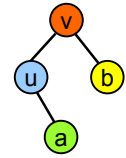
- (1) If $p == \text{NULL}$ then INSERT(p,q)
- (2) else
- (3) If $q.key < p.key$ then {
- (4) INSERTAVL(p.left,q)
- (5) if $h(p.right) - h(p.left) == -2$ then {
- (6) if $h(p.left.left) > h(p.left.right)$ then
- (7) $p = \text{RotateToRight}(p)$
- (8) else $p = \text{DoubleRotateLeftRight}(p)$
- (9) } }



27

INSERTAVL(p,q)

- (1) Else {
- (2) If $q.key > p.key$ then
- (3) INSERTAVL(p.right,q)
- (4) if $h(p.right) - h(p.left) == 2$ then {
- (5) if $h(p.right.right) > h(p.right.left)$ then
- (6) $p = \text{RotateToLeft}(p)$
- (7) else $p = \text{DoubleRotateRightLeft}(p)$
- (8) } }
- (9) $p.height := \max(h(p.left), h(p.right)) + 1$




28

Analyse

- Rotationen und Doppelrotationen können in konstanter Zeit ausgeführt werden.
- Eine Rebalancierung wird höchstens einmal an jedem Knoten auf dem Pfad vom eingefügten oder gelöschten Knoten zur Wurzel durchgeführt.
- Insert und Delete-Operationen sind in einem AVL-Baum in Zeit $O(\log n)$ möglich.
- AVL-Bäume unterstützen die Operationen Suchen, Insert, Delete, Minimum, Maximum, Successor, Predecessor, in Zeit $O(\log n)$.

Java-Applets

- Es gibt zu AVL-Bäumen sehr schöne Java-Applets, die die Doppelrotationen sehr schön darstellen, z.B. die Studienseiten von Math^e(Prism)²:
- <http://www.matheprisma.uni-wuppertal.de/Module/BinSuch/index.html> 

30

Kap. 4.4: B-Bäume

31

B-Bäume

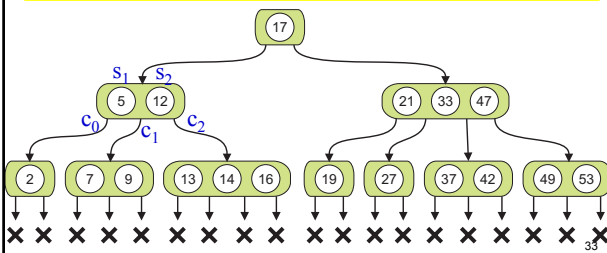
- Einführung von Rudolf Bayer und Eduard M. McCreight 1972
- Datenstruktur zur Verwaltung von Indizes für das relationale Datenmodell von Edgar F. Codd 1972
- → Entwicklung des ersten SQL-Datenbanksystems System R bei IBM

- B-Bäume sind ausgeglichene Mehrwegbäume
- **Idee:** jeder Knoten eines B-Baums der Ordnung m besitzt zwischen $\lceil m/2 \rceil$ und m Kinder.

32

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.



33

Definition B-Bäume

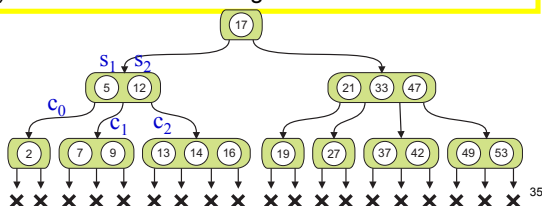
- Ein **B-Baum der Ordnung $m > 2$** ist ein Baum mit folgenden Eigenschaften (1)-(6):

- (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.

34

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.



35

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.

36

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.

37

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 $O(n)$.

38

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 $l+1$ Blättern hat l Schlüssel in inneren Knoten.

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

Schlüssel in Blättern

Schlüssel in inneren Knoten

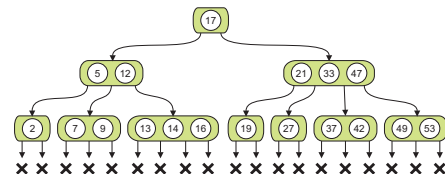
39

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



40

Datenstruktur eines B-Baums

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

Interne Repräsentation eines B-Baums:
var BTreeNode root

41

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!

42

SEARCH(p,x)

Suche in Baum mit Wurzel p den Schlüssel x

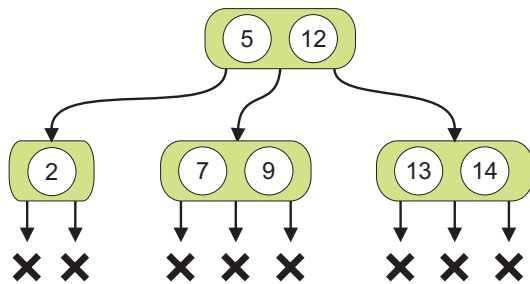
```

(1) If p==NULL then return NULL
(2) else {
(3)   var int i:=1
(4)   while i≤p.k and x>p.key do
(5)     i:=i+1
(6)   if i≤p.k and x==p.key[i] then
(7)     return p.data[i]
(8)   else return SEARCH(p.child[i-1],x)
(9) }
    
```

Analyse von SEARCH(p,x)

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

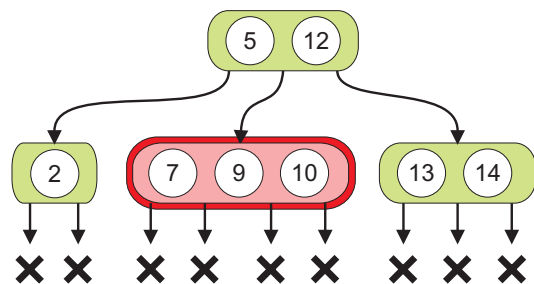
Einfügen eines Schlüssels in einen B-Baum



- Baum vor dem Einfügen von 10, m=3

45

Einfügen in B-Baum

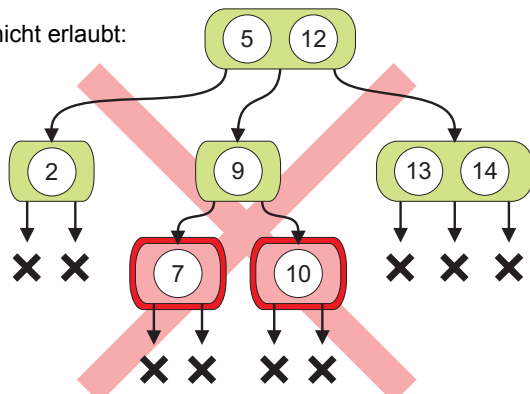


- Blatt wird zu groß: hat nun 3 Schlüssel!

46

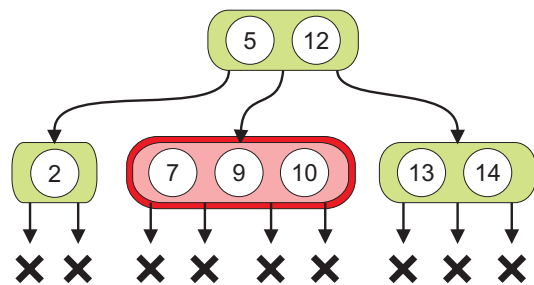
Einfügen in B-Baum

nicht erlaubt:



47

Einfügen in B-Baum



- Blatt wird zu groß: hat nun 3 Schlüssel!

48

