

## Kap 4.2: Binäre Suchbäume Kap. 4.3: AVL-Bäume

Professor Dr. Petra Mutzel  
Lehrstuhl für Algorithm Engineering, LS11

11. VO

11. Mai 2006

## Motivation

„Warum soll ich heute hier bleiben?“

Balancierte Bäume brauchen Sie immer wieder!

„Was gibt es heute Besonderes?“

Schönes Java-Applet!

2

## Überblick

- Wiederholung binäre Suchbäume (in Webversion ohne Folien)
- Analyse der Suchzeit, etc.
- Einführung von AVL-Bäumen
- Implementierung der Operationen
- Schönes Java-Applet

3

## Analyse der Operationen

1. Alle Operationen benötigen eine Laufzeit von  $O(h(T))$  für binäre Suchbäume, wobei  $h(T)$  die Höhe des gegebenen Suchbaumes  $T$  ist.

Frage: Wie hoch kann  $h(T)$  für einen binären Suchbaum mit  $n$  Knoten sein?

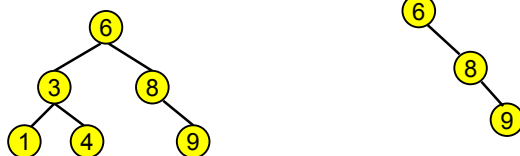
4

## Abhängigkeit der Höhe von der Einfügereihenfolge

Einfügereihenfolge: 1,3,4,6,8,9

Baum ist zu linearer Liste degeneriert  
Problem: Suchzeit ist linear,  
da  $h(T)=n-1$

Einfügereihenfolge: 6,8,3,4,1,9



5

## Diskussion

- Ein binärer Suchbaum  $T$  kann im Extremfall zu einer **linearen Liste** ausarten (z.B. Einfügereihenfolge sortiert).
- Dann dauert die erfolglose Suche  $\Theta(h(T))=\Theta(n)$
- Der andere Extremfall sind **vollständig balancierte Bäume** (d.h. alle Ebenen bis auf evtl. die unterste sind voll besetzt); diese haben die Höhe  $h(T)=\Theta(\log n)$ .
- Hier dauert die erfolglose Suche  $\Theta(h(T))=\Theta(\log(n))$
- Wir werden sehen: für dieses gute Zeitverhalten genügen auch „hinreichend“ balancierte Bäume

## Durchschnittliche Suchpfadlänge

Man kann zeigen:

- Die erwartete Suchpfadlänge (über alle möglichen Permutationen von  $n$  Einfügeoperationen) ist  
 $I(n) = 2 \ln n - O(1) = 1.38629 \dots \log n - O(1)$
- D.h. für große  $n$  ist die durchschnittliche Suchpfadlänge nur ca. 38% länger als im Idealfall.
- Allerdings ist diese Annahme, dass die Daten in einer zufälligen Reihenfolge eingegeben werden in vielen Anwendungen nicht gerechtfertigt.

7

## Kap. 4.3: AVL-Bäume

8

## Balancierte Bäume

- AVL-Bäume sind sogenannte balancierte Bäume.
- Balancierte Bäume** versuchen sich regelmäßig wieder „auszubalancieren“, um zu garantieren, dass die Höhe logarithmisch bleibt.
- Hierfür gibt es verschiedene Möglichkeiten:
  - **höhenbalancierte Bäume**
  - **gewichtsbalancierte Bäume**
  - **(a,b)-Bäume**
- Um die bisherigen binären Suchbäume von den balancierten Bäumen abzugrenzen, nennt man erstere auch **natürliche binäre Suchbäume**.

9

## Balancierte Bäume

- Höhenbalancierte Bäume:** Die Höhen der Unterbäume eines Knotens unterscheiden sich um höchstens eine Konstante
- Gewichtsbalancierte Bäume:** Die Anzahl der Knoten in den Unterbäumen jedes Knotens unterscheidet sich höchstens um eine Konstante.
- (a,b)-Bäume ( $2 \leq a \leq b$ ):** Jeder innere Knoten (außer der Wurzel) hat zwischen  $a$  und  $b$  Kinder und alle Blätter haben den gleichen Abstand zur Wurzel.

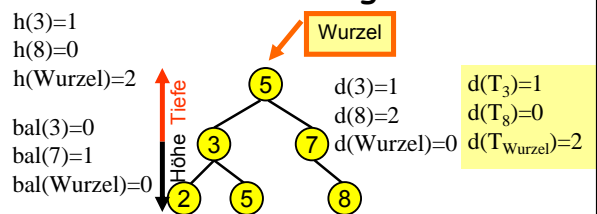
10

## 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:**  $bal(v) = h_2 - h_1$ , wobei  $h_1$  und  $h_2$  die Höhe des linken bzw. rechten Unterbaumes von  $v$
- $v$  heißt **balanciert**: wenn  $bal(v) \in \{-1, 0, +1\}$ , sonst heißt  $v$  unbalanciert
- AVL-Baum:** Binärer Suchbaum, bei dem alle Knoten balanciert sind

11

## Bezeichnungen



Beispiele für balanciert üben!

Höhe  $h(v)$ : Länge eines längsten Pfades von  $v$  zu einem Nachkommen von  $v$

Höhe  $h(T_r)$  eines (Teil-)baumes  $T_r$  mit Wurzel  $r$ :  
 $\max \{ d(v) : v \text{ ist Knoten in } T_r \} = \text{Höhe seiner Wurzel}$

## Definitionen für AVL-Bäume

- AVL-Bäume wurden 1962 eingeführt von G.M. Adelson-Velskii und Y.M. Landis
- AVL-Bäume sind also höhenbalanciert.
- Man kann zeigen:
  - **Theorem:** Ein AVL-Baum mit  $n$  Kindern hat Höhe  $O(\log n)$ .

13

## Implementierungen der Operationen

- **Search**
- **Minimum**
- **Maximum**
- **Successor**
- **Predecessor**
- genau wie bei den natürlichen binären Suchbäumen.

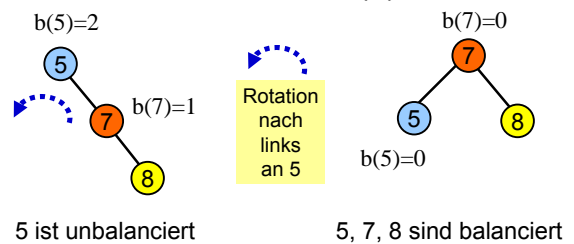
14

## 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.

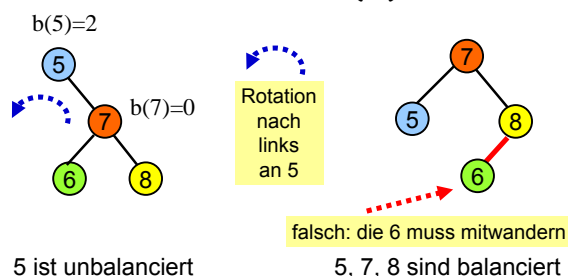
15

## Rotationen (1)



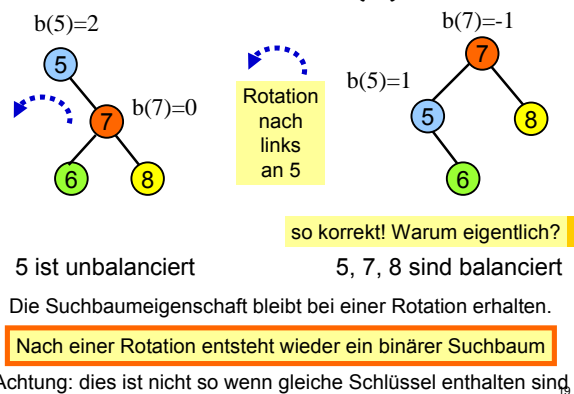
17

## Rotationen (2)



18

## Rotationen (3)



## Definition

- **AVL-Ersetzung:** Operation (z.B. Insert, Delete), die einen Unterbaum T eines Knotens z durch einen modifizierten AVL-Baum ersetzt, dessen Höhe um höchstens 1 von der Höhe von T abweicht.

21

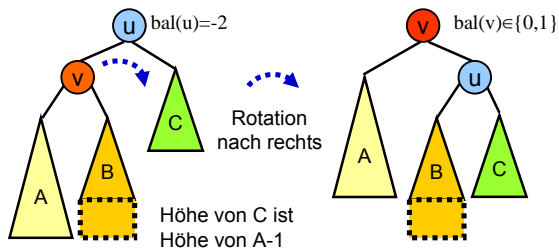
## Rebalancierung

- Im folgenden sei  $T_0$  ein AVL-Baum vor der AVL-Ersetzung und  $T_1$  der unbalancierte Baum hinterher.
- Sei  $u$  ein unbalancierter Knoten **maximaler Tiefe**, d.h.  $\text{bal}(u) \in \{-2, +2\}$ .
- Seien  $T$  und  $T^*$  die maximalen Unterbäume mit Wurzel  $u$  in  $T_0$  bzw. neuer Wurzel in  $T_1$ .
- Wir unterscheiden vier verschiedene Situationen.

22

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

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

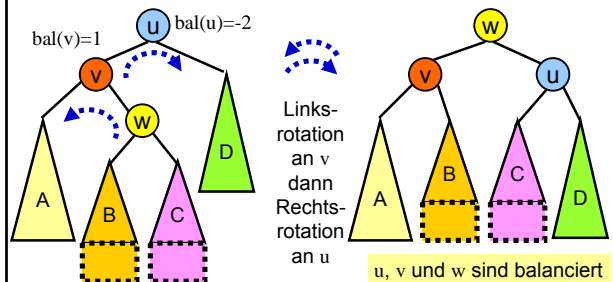


- Suchbaumeigenschaft bleibt erhalten;  $u$  und  $v$  sind balanciert
- Für die Knoten unterhalb hat sich nichts geändert.

23

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

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

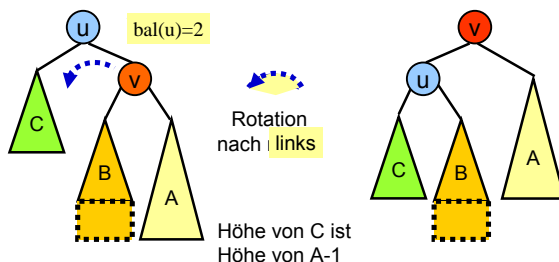


Höhe von (B oder C) und von D ist Höhe von A

24

### 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 links an  $u$

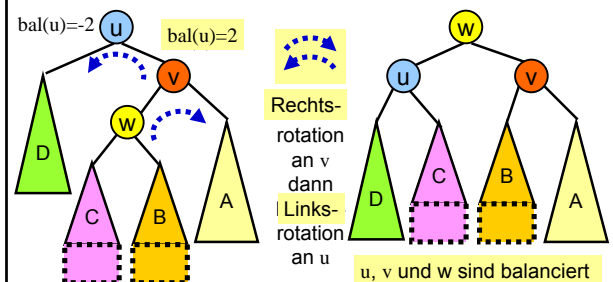


- Inorder-Reihenfolge bleibt erhalten und  $u$  und  $v$  sind balanciert
- Für die Knoten unterhalb hat sich nichts geändert.

25

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

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



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

26

## Rebalancierung ff.

- Für die einfache Rotation gilt:  $h(T)-h(T^*) \in \{0,1\}$
- Im Falle der Doppelrotation gilt:  $h(T)-h(T^*) \in \{0,1\}$
- Alle Transformationen (Einfach- und Doppelrotationen) kann man als eine AVL-Ersetzung auffassen.

27

## Rebalancierung ff.

- Nach der AVL-Ersetzung gilt:
- Die Unterbäume mit Wurzel  $u$  und  $v$  sind danach balanciert. Für die Unterbäume unterhalb hat sich die Balancierung nicht geändert.

- 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.

## Rebalancierung ff.

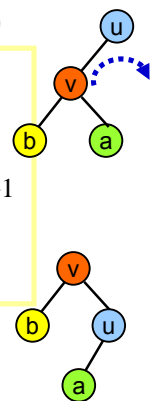
- Die Insert-Operation unterscheidet sich nur insofern von der Delete-Operation, dass hier ein einziger Rebalancierungsschritt genügt.
- 
- Bei der Delete-Operation hingegen kann es sein, dass mehrere Rebalancierungsschritte notwendig sind.

Implementierungen:

29

## RotateToRight(u)

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

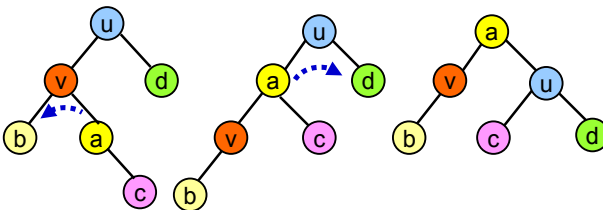


30

## DoubleRotateLeftRight(u)

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

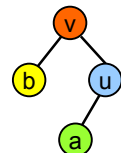
Beispiel:



31

## INSERTAVL(p,q)

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



33

### 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)  $p.height := \max(h(p.left), h(p.right)) + 1$

34

### 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  $O(\log n)$ .

### Diskussion

- Wenn oft gesucht und selten umgeordnet wird, sind AVL-Bäume günstiger als natürliche binäre Suchbäume.
- Falls jedoch fast jeder Zugriff ein Einfügen oder Entfernen ist (hinreichend zufällig), dann sind natürliche binäre Suchbäume vorzuziehen.

36

### Java-Applets

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

37