

Binäre Suchbäume – Löschen

In Searchtree $\langle K, D \rangle$ jetzt korrekt:

Java

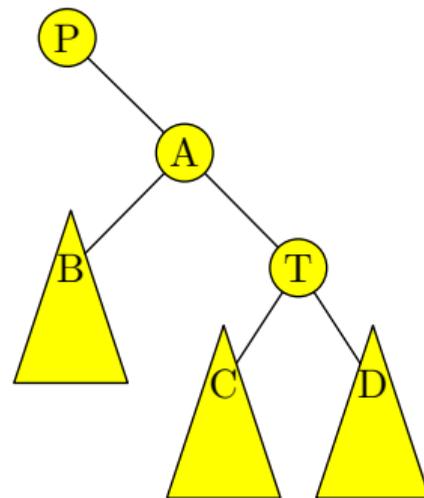
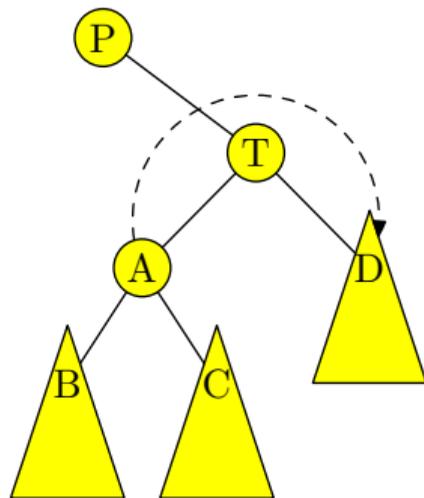
```
public void delete(K k) {  
    if(root == null) return;  
    if(root.key.equals(k))  
        if(root.left == null && root.right == null) {  
            root = null; return;  
        }  
        else if(root.left == null) {  
            root = root.right; root.parent = null; return;  
        }  
    Searchtreenode $\langle K, D \rangle$  n = root.findsubtree(k);  
    if(n  $\neq$  null) n.delete();  
}
```

Binäre Suchbäume – Beispiel

Die Schlüssel von 1 bis 40 werden zufällig eingefügt oder gelöscht.



Umstrukturierung durch Rotationen



Eine Rechtsrotation um T.

Die Suchbaumeigenschaft bleibt erhalten.

B, C, D können nur aus externen Knoten bestehen.

Laufzeit: Konstant!

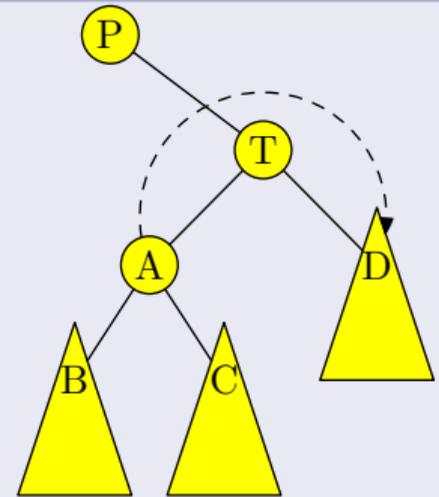
Rotationen

Wir rotieren rechts um 5 und dann zurück links um 3.

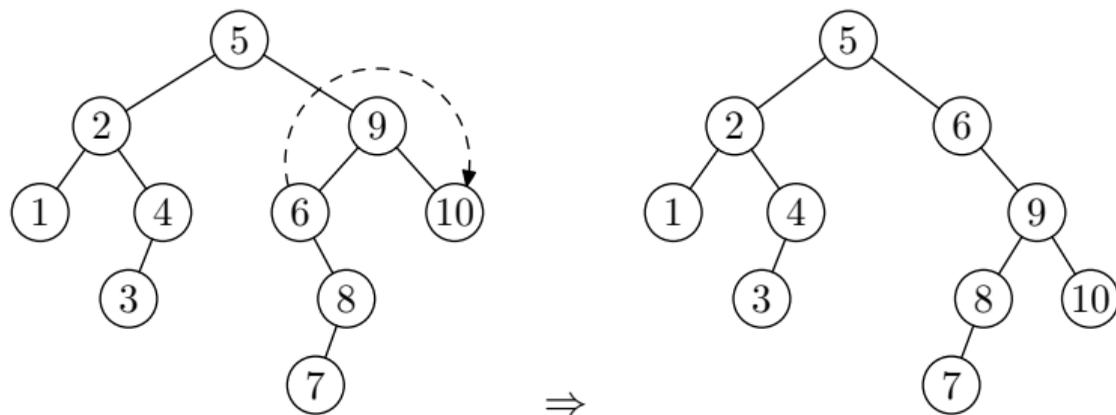


Java

```
void rotateright() {  
    Searchtreenode<K, D> p, a, b, c, d;  
    p = this.parent;  
    a = this.left; d = this.right;  
    b = a.left; c = a.right;  
    if (p ≠ null) {  
        if (p.left == this) p.left = a;  
        else p.right = a;  
    }  
    a.right = this; a.parent = p;  
    this.left = c; this.parent = a;  
    if (c ≠ null) c.parent = this;  
}
```

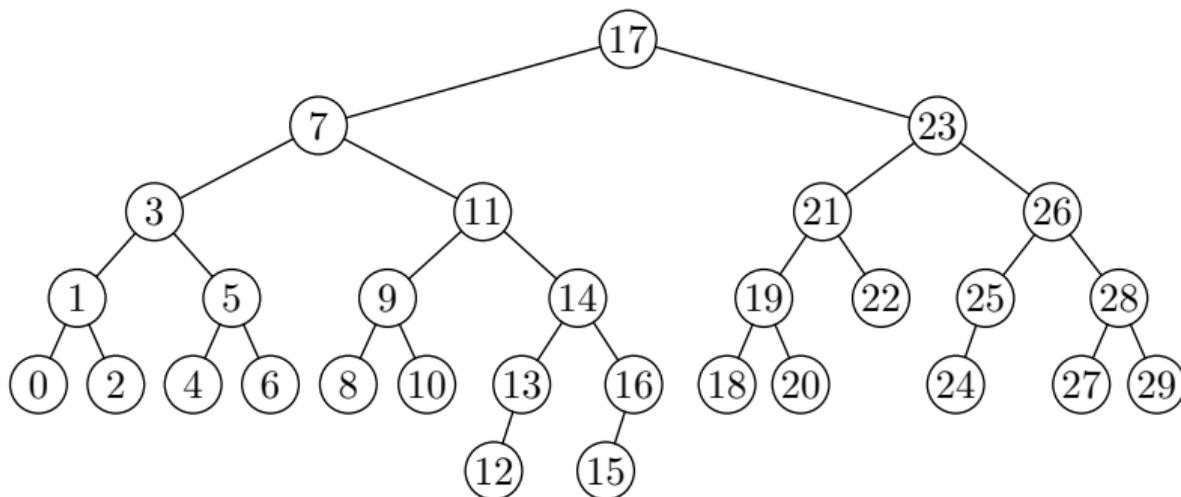


Beispiel



Frage: Kann man einen Knoten durch Rotationen zu einem Blatt machen?

AVL-Bäume

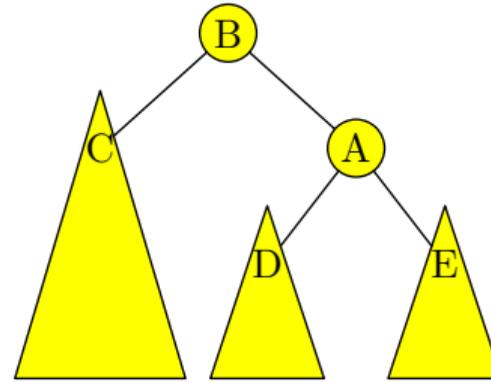
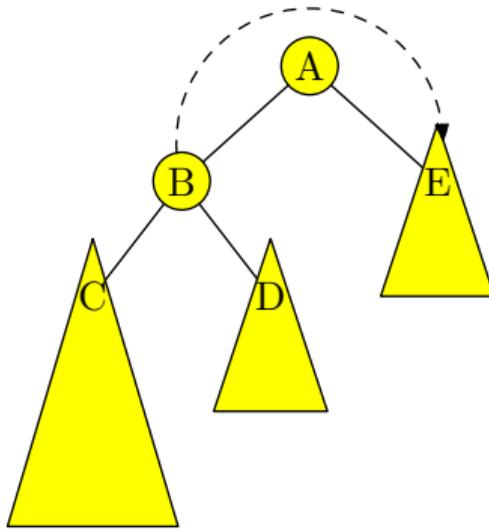


AVL-Bäume werden annähernd balanziert gehalten.

AVL-Eigenschaft:

Die Höhen des rechten und linken Unterbaums jedes Knotens unterscheiden sich höchstens um 1.

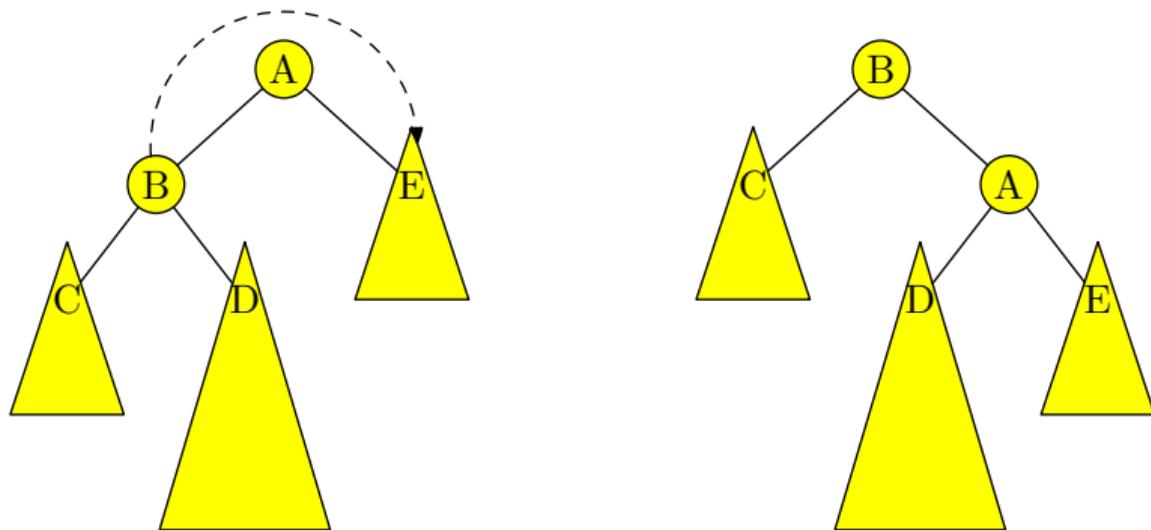
AVL-Bäume – Einfügen



Durch Einfügen in C ist die AVL-Bedingung verletzt.

Reparieren durch Rechtsrotation um A.

AVL-Bäume – Einfügen

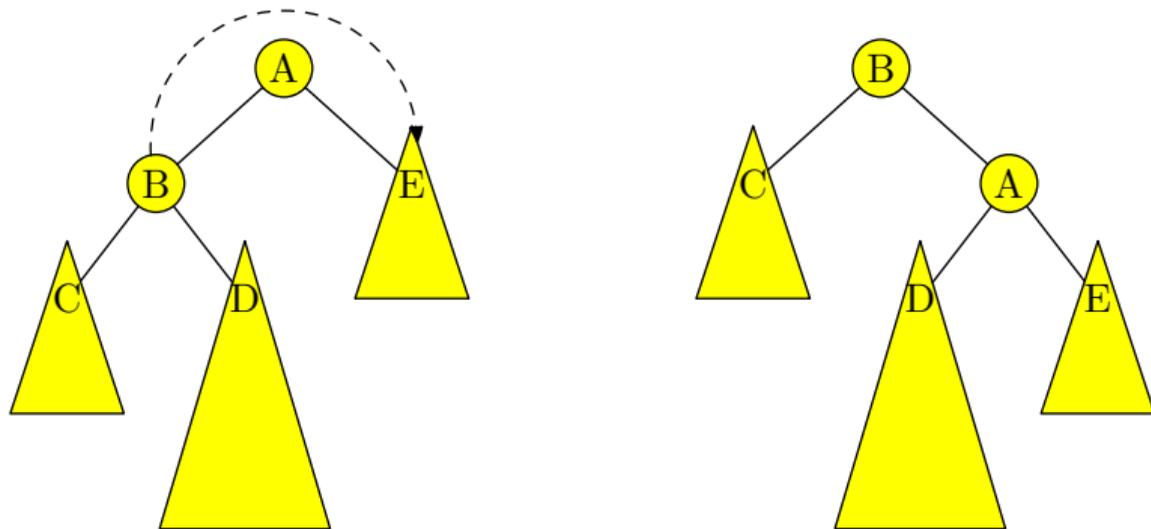


Durch Einfügen in D ist die AVL-Bedingung verletzt.

Reparieren durch Rechtsrotation um A?

Lösung: Erst um B links, dann um A rechts rotieren!

AVL-Bäume – Einfügen

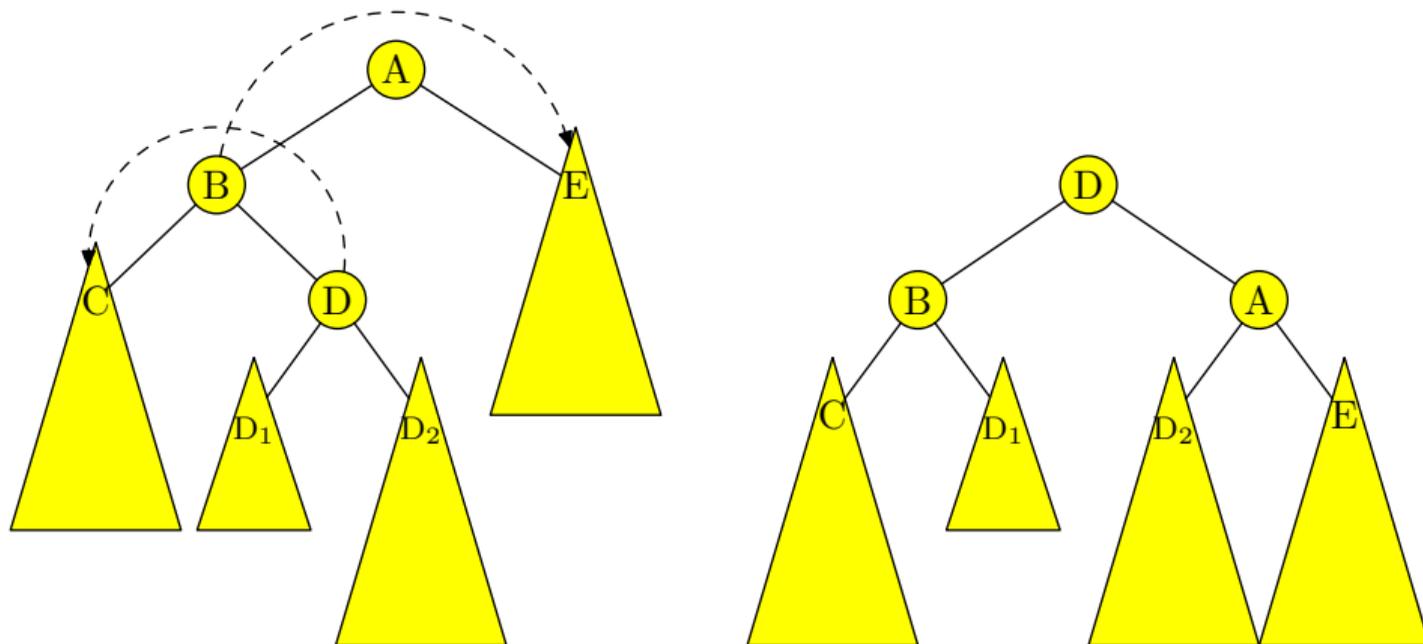


Durch Einfügen in D ist die AVL-Bedingung verletzt.

Reparieren durch Rechtsrotation um A?

Lösung: **Erst um B links, dann um A rechts rotieren!**

AVL-Bäume – Einfügen



Durch Einfügen in D ist die AVL-Bedingung verletzt.

Erst um B links, dann um A rechts rotieren!

AVL-Bäume – Einfügen

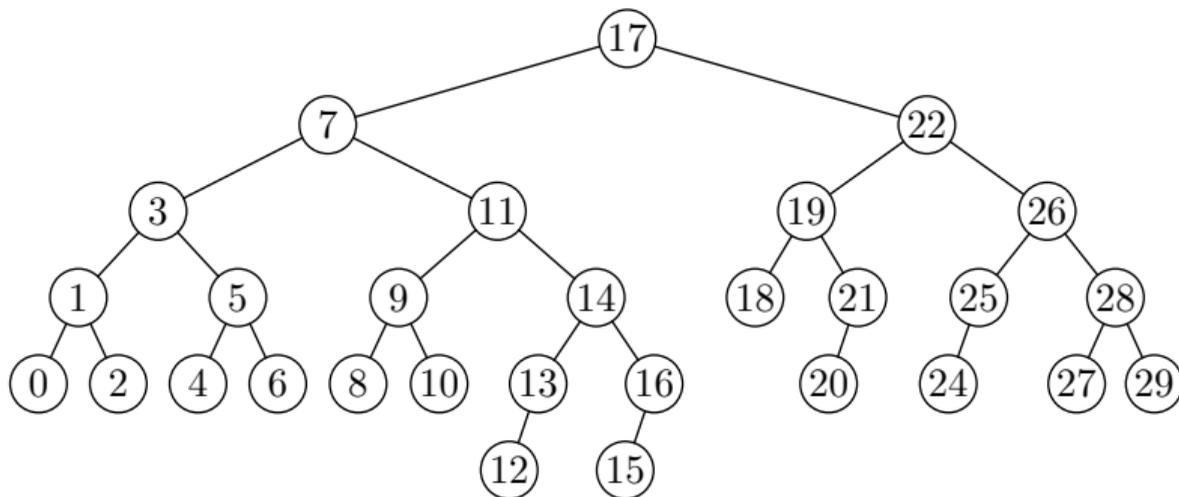
Durch Einfügen können nur Knoten auf dem Pfad von der Wurzel zum eingefügten Blatt unbalanziert werden.

Algorithmus

```
procedure avl – insert(key k) :  
  insert(k);  
  n := findnode(k);  
  while n  $\neq$  root do  
    if n unbalanced then rebalance n fi;  
    n := parent(n);  
  od
```

Es werden höchstens zwei Rotationen durchgeführt.

Beispiel



Java

```
void rebalance() {
    computeheight();
    if(height(left) > height(right) + 1) {
        if(height(left.left) < height(left.right)) left.rotateleft();
        rotateright();
    }
    else if(height(right) > height(left) + 1) {
        if(height(right.right) < height(right.left)) right.rotateright();
        rotateleft();
    }
    if(parent ≠ null)((AVLtreeNode<K, D>) parent).rebalance();
}
```

Java

```
public void insert(K k, D d) {  
    if(root == null) root = newNode(k, d);  
    else root.insert(newNode(k, d));  
    ((AVLTreeNode<K, D>) root.findsubtree(k)).rebalance();  
    repair_root();  
}
```

Java

```
public void repair_root() {  
    if(root == null) return;  
    while(root.parent != null) root = root.parent;  
}
```

AVL-Bäume – Einfügen

- Nur Knoten auf Pfad zur Wurzel können unbalanziert werden.
- Durch Rotation oder Doppelrotation reparieren.
- Dadurch nimmt Höhe ab!
- \Rightarrow Danach wieder balanziert.
- \Rightarrow Es muß nur **einmal** repariert werden.
- Einfügen benötigt maximal zwei Rotationen.

AVL-Bäume – Löschen

Wiederholung

Drei Möglichkeiten beim Löschen:

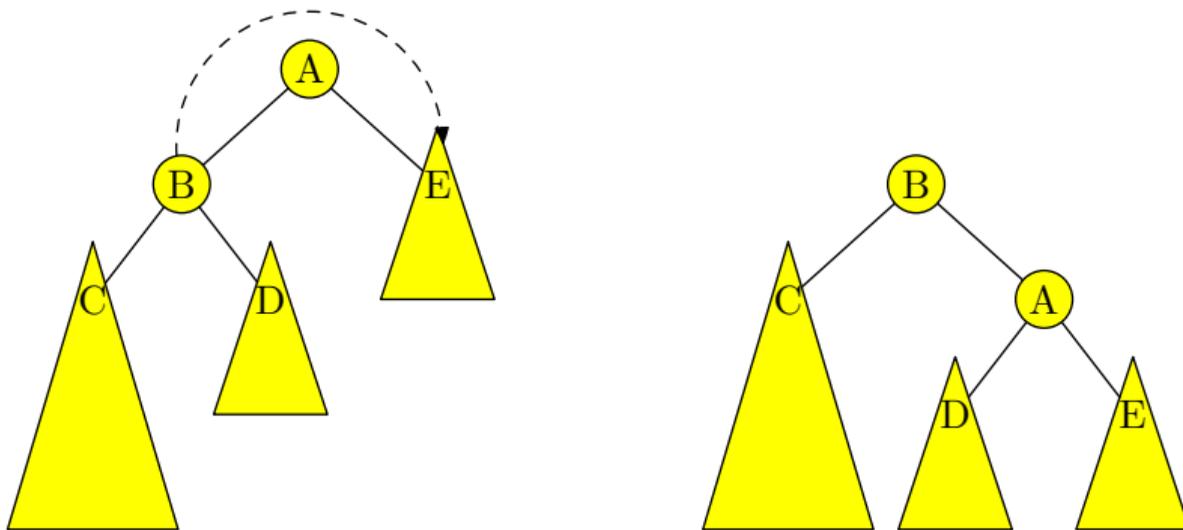
- 1 Ein Blatt
- 2 Kein linkes Kind
- 3 Es gibt linkes Kind

Nur bei den ersten beiden Fällen ändert sich die Höhe direkt!

Nur Knoten auf dem Pfad zur Wurzel können unbalanziert werden.

⇒ Wieder durch Rotationen reparieren.

AVL-Bäume – Löschen



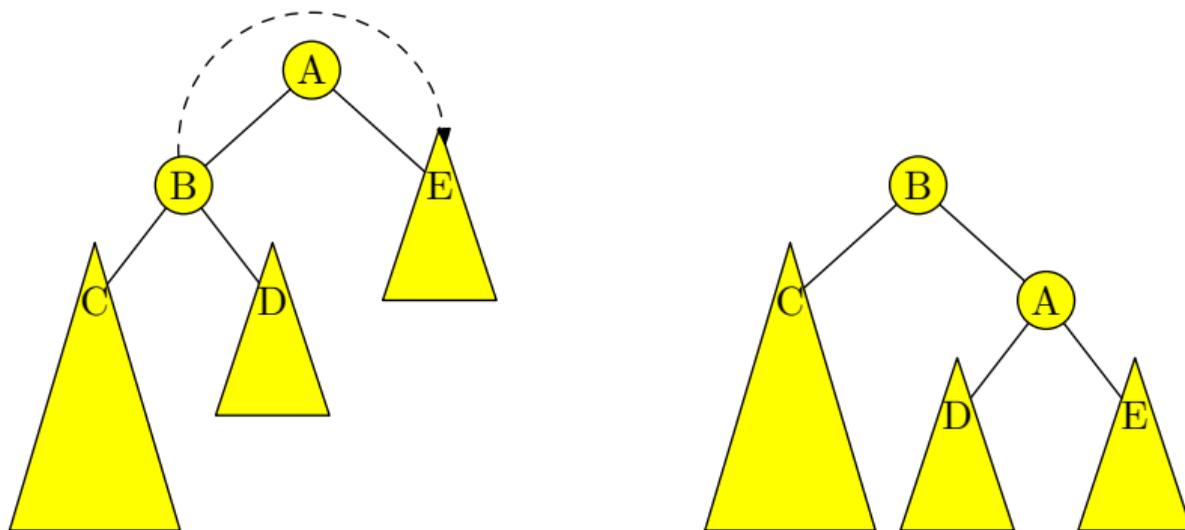
Durch Löschen aus E ist AVL-Bedingung in A verletzt.

Reparieren durch Rechtsrotation um A.

Die Höhe ist dadurch gesunken!

Elternknoten von A kann **wieder** unbalanziert werden!

AVL-Bäume – Löschen



Durch Löschen aus E ist AVL-Bedingung in A verletzt.

Reparieren durch Rechtsrotation um A.

Die Höhe ist dadurch gesunken!

Elternknoten von A kann **wieder** unbalanziert werden!

Java

```
void delete() {  
    if(left == null && right == null) {  
        if(parent.left == this) parent.left = null;  
        else parent.right = null;  
        ((AVLTreeNode<K, D>) parent).rebalance(); }  
    else if(left == null) {  
        copy(right);  
        if(right.left != null) right.left.parent = this;  
        if(right.right != null) right.right.parent = this;  
        left = right.left; right = right.right;  
        rebalance(); }  
    else {  
        SearchTreeNode<K, D> max = left;  
        while(max.right != null) max = max.right;  
        copy(max); max.delete(); }  
}
```

AVL-Bäume – Löschen

Java

```
public void delete(K k) {  
    if(root == null) return;  
    AVLtreenode<K, D> n;  
    n = (AVLtreenode<K, D>) root.findsubtree(k);  
    if(n == null) return;  
    if(n == root && n.left == null && n.right == null) {  
        root = null;  
    }  
    else {  
        n.delete();  
    }  
    repair_root();  
}
```

AVL-Bäume – Beispiel

Schlüssel von 1 bis 40 werden zufällig eingefügt und gelöscht.



AVL-Bäume – Analyse

Theorem

Ein AVL-Baum der Höhe h besitzt zwischen F_h und $2^h - 1$ viele Knoten.

Definition

Wir definieren die n te Fibonaccizahl:

$$F_n = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ F_{n-1} + F_{n-2} & \text{falls } n > 2 \end{cases}$$

AVL-Bäume – Analyse

Theorem

Ein AVL-Baum der Höhe h besitzt zwischen F_h und $2^h - 1$ viele Knoten.

Definition

Wir definieren die n te Fibonaccizahl:

$$F_n = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ F_{n-1} + F_{n-2} & \text{falls } n > 2 \end{cases}$$

Beweis.

Trivial: Ein vollständiger Binärbaum der Höhe h hat **genau** $2^h - 1$ viele interne Knoten (und ist ein AVL-Baum).

Ein Baum der Höhe 0 hat keinen internen Knoten und $F_0 = 0$.

Ein Baum der Höhe 1 hat genau einen internen Knoten und $F_1 = 1$.

Falls $h > 1$, dann hat der linke oder rechte Unterbaum Höhe $h - 1$ und damit mindestens F_{h-1} viele interne Knoten.

Der andere Unterbaum hat mindestens Höhe $h - 2$ und damit mindestens F_{h-2} viele interne Knoten.

Insgesamt macht das mindestens $F_{h-1} + F_{h-2} = F_h$ viele Knoten. □

Beweis.

Trivial: Ein vollständiger Binärbaum der Höhe h hat **genau** $2^h - 1$ viele interne Knoten (und ist ein AVL-Baum).

Ein Baum der Höhe 0 hat keinen internen Knoten und $F_0 = 0$.

Ein Baum der Höhe 1 hat genau einen internen Knoten und $F_1 = 1$.

Falls $h > 1$, dann hat der linke oder rechte Unterbaum Höhe $h - 1$ und damit mindestens F_{h-1} viele interne Knoten.

Der andere Unterbaum hat mindestens Höhe $h - 2$ und damit mindestens F_{h-2} viele interne Knoten.

Insgesamt macht das mindestens $F_{h-1} + F_{h-2} = F_h$ viele Knoten. □

Beweis.

Trivial: Ein vollständiger Binärbaum der Höhe h hat **genau** $2^h - 1$ viele interne Knoten (und ist ein AVL-Baum).

Ein Baum der Höhe 0 hat keinen internen Knoten und $F_0 = 0$.

Ein Baum der Höhe 1 hat genau einen internen Knoten und $F_1 = 1$.

Falls $h > 1$, dann hat der linke oder rechte Unterbaum Höhe $h - 1$ und damit mindestens F_{h-1} viele interne Knoten.

Der andere Unterbaum hat mindestens Höhe $h - 2$ und damit mindestens F_{h-2} viele interne Knoten.

Insgesamt macht das mindestens $F_{h-1} + F_{h-2} = F_h$ viele Knoten. □

AVL-Bäume – Analyse

Theorem

Die Höhe eines AVL-Baums mit n Knoten ist $\Theta(\log n)$.

Beweis.

Es gilt $F_h = \Theta(\phi^h)$ mit $\phi = (1 + \sqrt{5})/2$.

$$F_h \leq n \Rightarrow h \log(\phi) + O(1) \leq \log n$$

$$n \leq 2^h - 1 \Rightarrow \log n \leq h + O(1)$$



AVL-Bäume – Analyse

Theorem

Die Höhe eines AVL-Baums mit n Knoten ist $\Theta(\log n)$.

Beweis.

Es gilt $F_h = \Theta(\phi^h)$ mit $\phi = (1 + \sqrt{5})/2$.

$$F_h \leq n \Rightarrow h \log(\phi) + O(1) \leq \log n$$

$$n \leq 2^h - 1 \Rightarrow \log n \leq h + O(1)$$



AVL-Bäume – Analyse

Theorem

Einfügen, Suchen und Löschen in einen AVL-Baum mit n Elementen benötigt $O(\log n)$ Schritte.

Beweis.

Die Höhe des AVL-Baumes ist $\Theta(\log n)$.

Einfügen, Suchen und Löschen benötigt $O(h)$ Schritte, wenn h die Höhe des Suchbaums ist.

Das Rebalanzieren benötigt ebenfalls $O(h)$ Schritte. □

AVL-Bäume – Analyse

Theorem

Einfügen, Suchen und Löschen in einen AVL-Baum mit n Elementen benötigt $O(\log n)$ Schritte.

Beweis.

Die Höhe des AVL-Baumes ist $\Theta(\log n)$.

Einfügen, Suchen und Löschen benötigt $O(h)$ Schritte, wenn h die Höhe des Suchbaums ist.

Das Rebalanzieren benötigt ebenfalls $O(h)$ Schritte. □