

# 16. Natürliche Suchbäume

[Ottman/Widmayer, Kap. 5.1, Cormen et al, Kap. 12.1 - 12.3]

# Wörterbuchimplementationen

Hashing: Implementierung von Wörterbüchern mit erwartet sehr schnellen Zugriffszeiten.

Nachteile von Hashing: im schlechtesten Fall lineare Zugriffszeit.

Manche Operationen gar nicht unterstützt:

- Aufzählen von Schlüssel in aufsteigender Anordnung
- Nächst kleinerer Schlüssel zu gegebenem Schlüssel

# Bäume

Bäume sind

- Verallgemeinerte Listen: Knoten können mehrere Nachfolger haben
- Spezielle Graphen: Graphen bestehen aus Knoten und Kanten. Ein Baum ist ein zusammenhängender, gerichteter, azyklischer Graph.

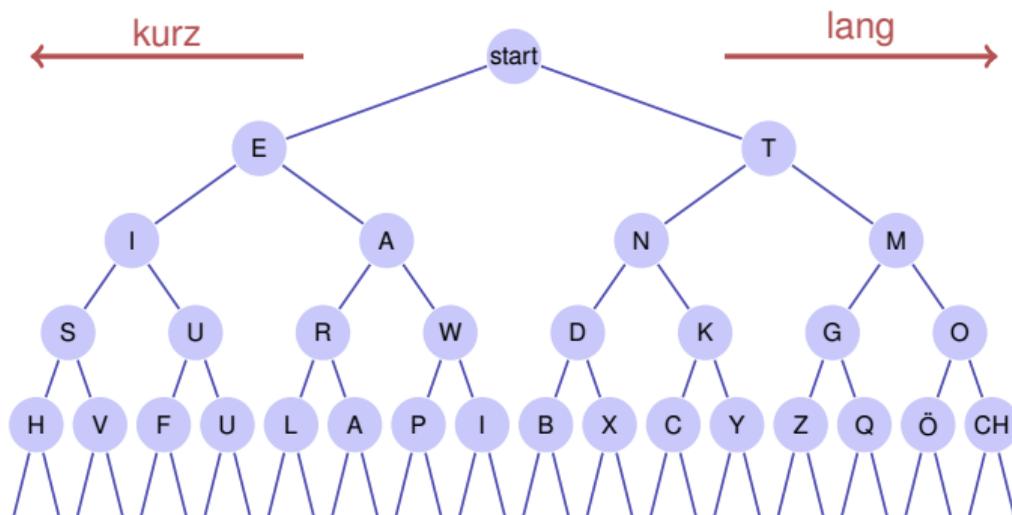
# Bäume

## Verwendung

- Entscheidungsbäume: Hierarchische Darstellung von Entscheidungsregeln
- Syntaxbäume: Parsen und Traversieren von Ausdrücken, z.B. in einem Compiler
- Codebäume: Darstellung eines Codes, z.B. Morsealphabet, Huffman Code
- Suchbäume: ermöglichen effizientes Suchen eines Elementes



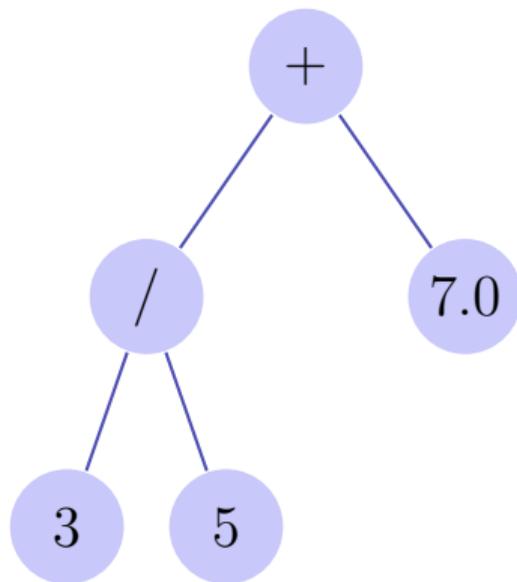
# Beispiele



Morsealphabet

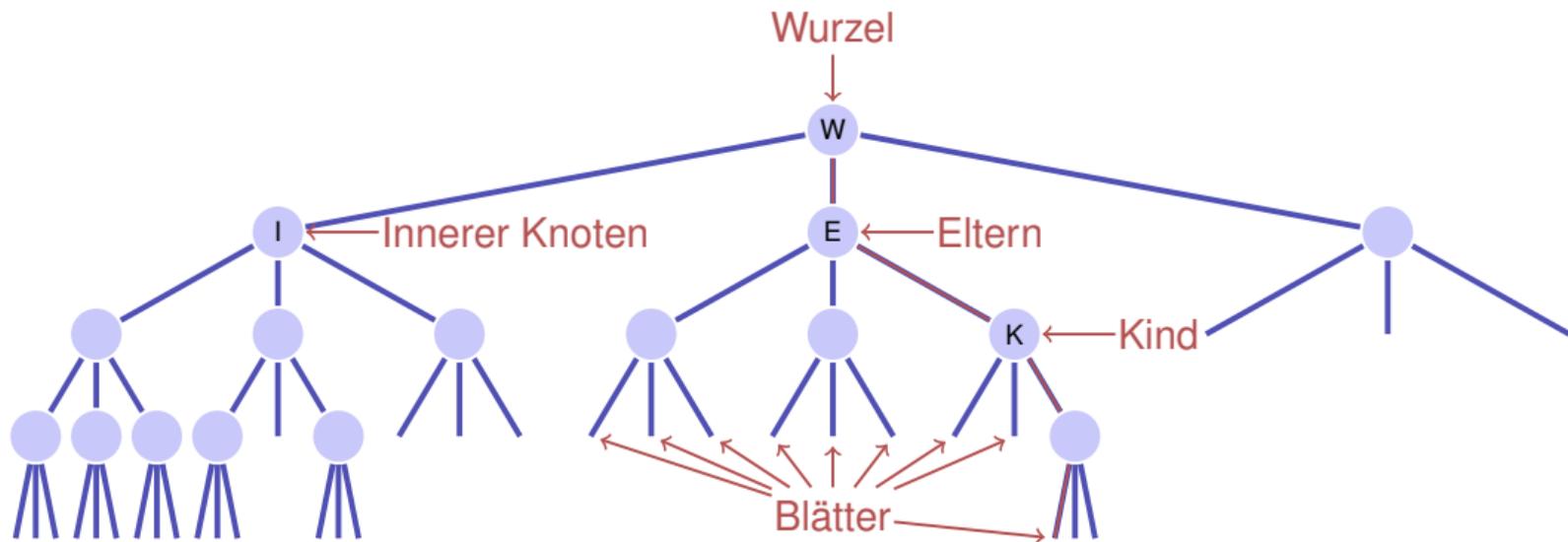
# Beispiele

$3/5 + 7.0$



Ausdrucksbaum

# Nomenklatur



- Ordnung des Baumes: Maximale Anzahl Kindknoten, hier: 3
- Höhe des Baumes: maximale Pfadlänge Wurzel – Blatt (hier: 4)

# Binäre Bäume

Ein binärer Baum ist entweder

- ein Blatt, d.h. ein leerer Baum, oder
- ein innerer Knoten mit zwei Bäumen  $T_l$  (linker Teilbaum) und  $T_r$  (rechter Teilbaum) als linken und rechten Nachfolger.

In jedem Knoten  $v$  speichern wir

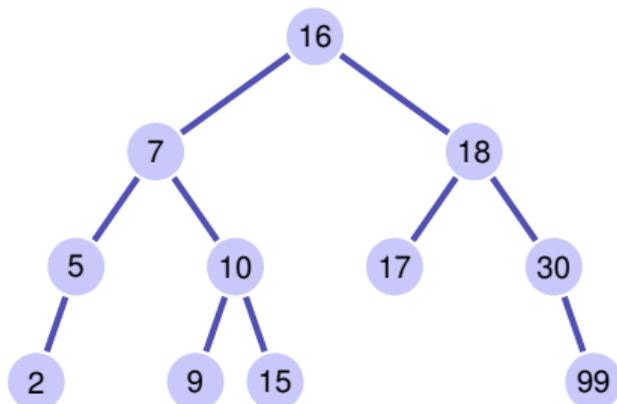


- einen Schlüssel  $v.key$  und
- zwei Zeiger  $v.left$  und  $v.right$  auf die Wurzeln der linken und rechten Teilbäume.
- Ein Blatt wird durch den **null**-Zeiger repräsentiert

# Binärer Suchbaum

Ein binärer Suchbaum ist ein binärer Baum, der die Suchbaumeigenschaft erfüllt:

- Jeder Knoten  $v$  speichert einen Schlüssel
- Schlüssel im linken Teilbaum  $v.left$  von  $v$  sind kleiner als  $v.key$
- Schlüssel im rechten Teilbaum  $v.right$  von  $v$  sind grösser als  $v.key$



# Suchen

**Input** : Binärer Suchbaum mit Wurzel  $r$ ,  
Schlüssel  $k$

**Output** : Knoten  $v$  mit  $v.\text{key} = k$  oder **null**

$v \leftarrow r$

**while**  $v \neq \text{null}$  **do**

**if**  $k = v.\text{key}$  **then**

        | **return**  $v$

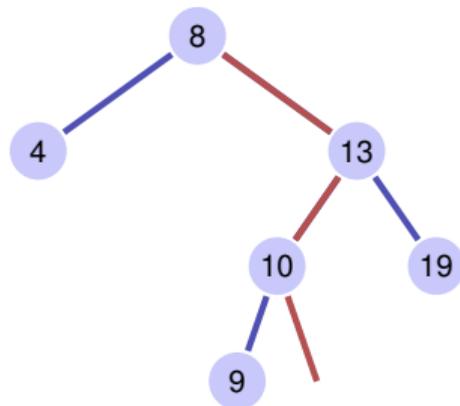
**else if**  $k < v.\text{key}$  **then**

        |  $v \leftarrow v.\text{left}$

**else**

        |  $v \leftarrow v.\text{right}$

**return null**



Search (12)  $\rightarrow$  **null**

# Höhe eines Baumes

Die Höhe  $h(T)$  eines Baumes  $T$  mit Wurzel  $r$  ist gegeben als

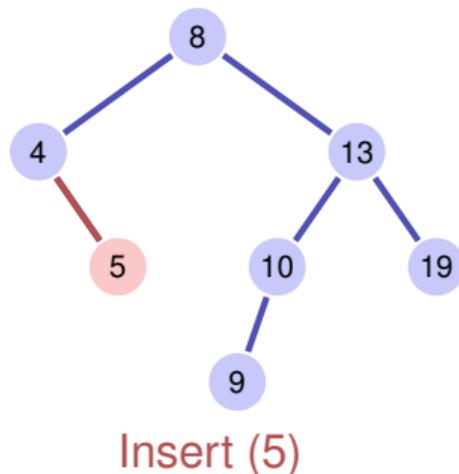
$$h(r) = \begin{cases} 0 & \text{falls } r = \mathbf{null} \\ 1 + \max\{h(r.\text{left}), h(r.\text{right})\} & \text{sonst.} \end{cases}$$

Die Laufzeit der Suche ist somit im schlechtesten Fall  $\mathcal{O}(h(T))$

# Einfügen eines Schlüssels

Einfügen des Schlüssels  $k$

- Suche nach  $k$ .
- Wenn erfolgreich:  
Fehlerausgabe
- Wenn erfolglos: Einfügen des  
Schlüssels am erreichten Blatt.

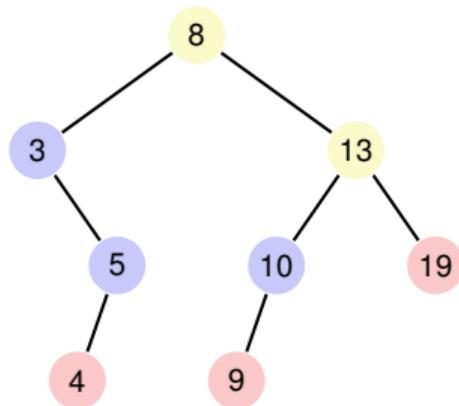


# Knoten entfernen

Drei Fälle möglich

- Knoten hat keine Kinder
- Knoten hat ein Kind
- Knoten hat zwei Kinder

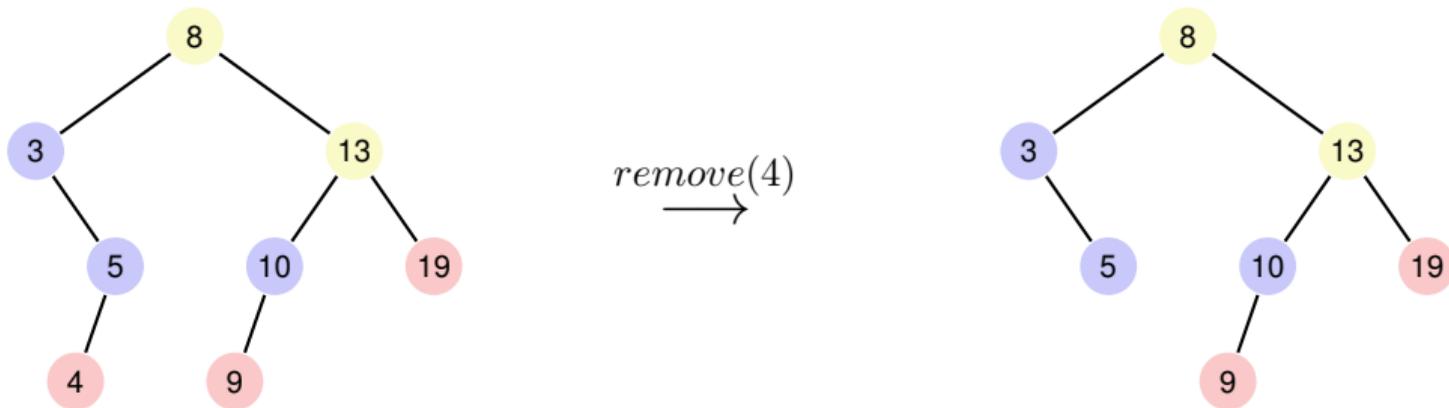
[Blätter zählen hier nicht]



# Knoten entfernen

Knoten hat keine Kinder

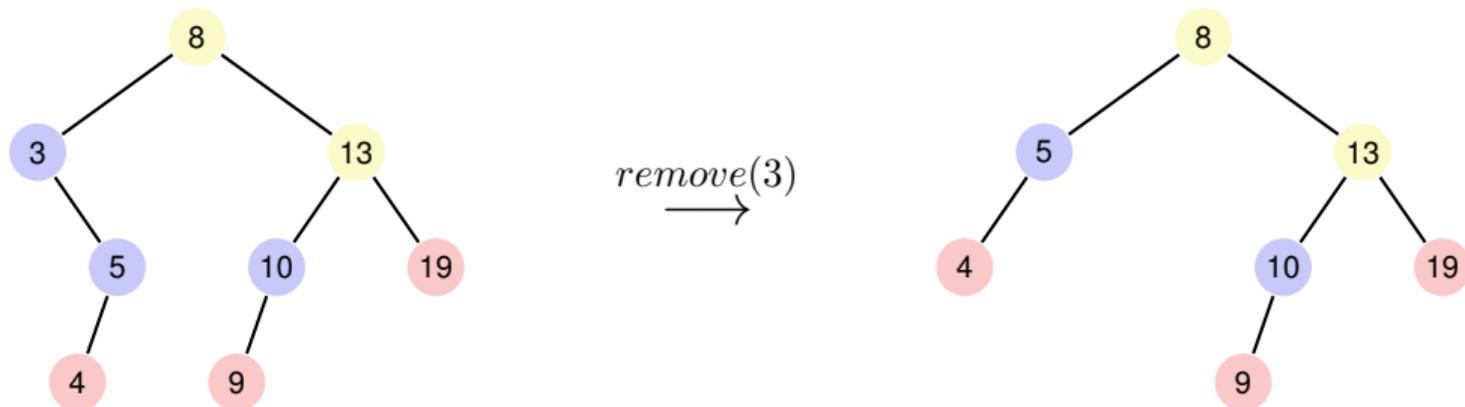
Einfacher Fall: Knoten durch Blatt ersetzen.



# Knoten entfernen

Knoten hat ein Kind

Auch einfach: Knoten durch das einzige Kind ersetzen.



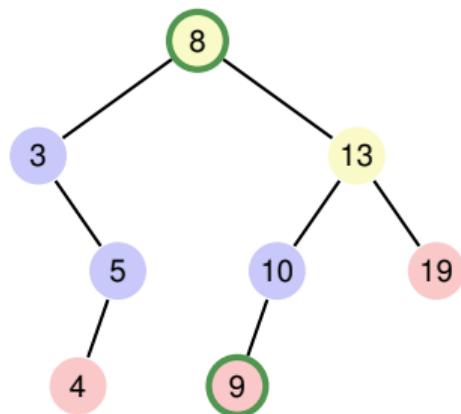
# Knoten entfernen

Knoten  $v$  hat zwei Kinder

Beobachtung: Der kleinste Schlüssel im rechten Teilbaum  $v.right$  (der *symmetrische Nachfolger* von  $v$ )

- ist kleiner als alle Schlüssel in  $v.right$
- ist grösser als alle Schlüssel in  $v.left$
- und hat kein linkes Kind.

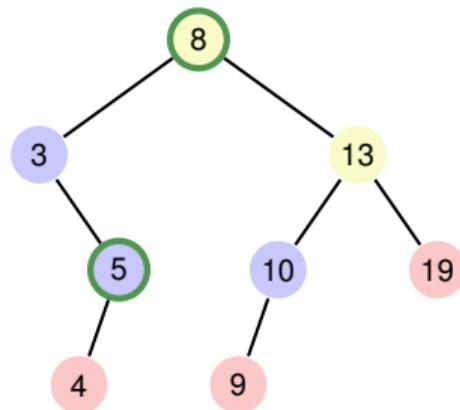
Lösung: ersetze  $v$  durch seinen symmetrischen Nachfolger



# Aus Symmetriegründen...

Knoten  $v$  hat zwei Kinder

Auch möglich: ersetze  $v$  durch seinen symmetrischen Vorgänger



# Algorithmus SymmetricSuccessor( $v$ )

**Input** : Knoten  $v$  eines binären Suchbaumes

**Output** : Symmetrischer Nachfolger von  $v$

$w \leftarrow v.\text{right}$

$x \leftarrow w.\text{left}$

**while**  $x \neq \text{null}$  **do**

$w \leftarrow x$   
     $x \leftarrow x.\text{left}$

**return**  $w$

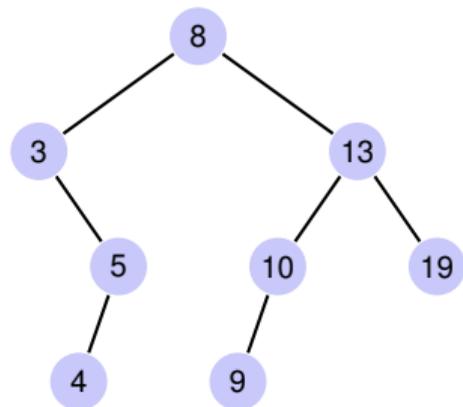
# Analyse

Löschen eines Elementes  $v$  aus einem Baum  $T$  benötigt  $\mathcal{O}(h(T))$   
Elementarschritte:

- Suchen von  $v$  hat Kosten  $\mathcal{O}(h(T))$
- Hat  $v$  maximal ein Kind ungleich **null**, dann benötigt das Entfernen  $\mathcal{O}(1)$
- Das Suchen des symmetrischen Nachfolgers  $n$  benötigt  $\mathcal{O}(h(T))$  Schritte. Entfernen und Einfügen von  $n$  hat Kosten  $\mathcal{O}(1)$

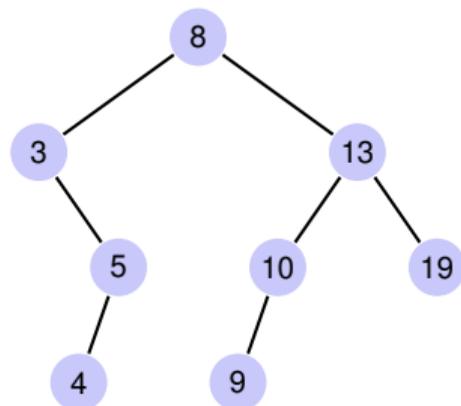
# Traversierungsarten

- Hauptreihenfolge (preorder):  $v$ , dann  $T_{\text{left}}(v)$ , dann  $T_{\text{right}}(v)$ .  
8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder):  $T_{\text{left}}(v)$ , dann  $T_{\text{right}}(v)$ , dann  $v$ .  
4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder):  
 $T_{\text{left}}(v)$ , dann  $v$ , dann  $T_{\text{right}}(v)$ .  
3, 4, 5, 8, 9, 10, 13, 19

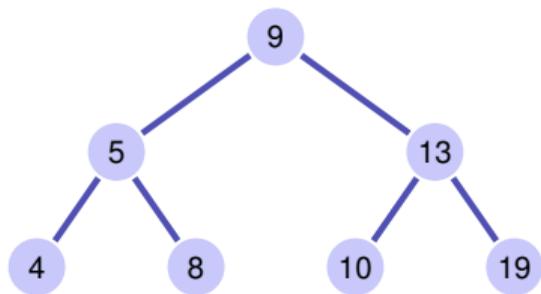


# Weitere unterstützte Operationen

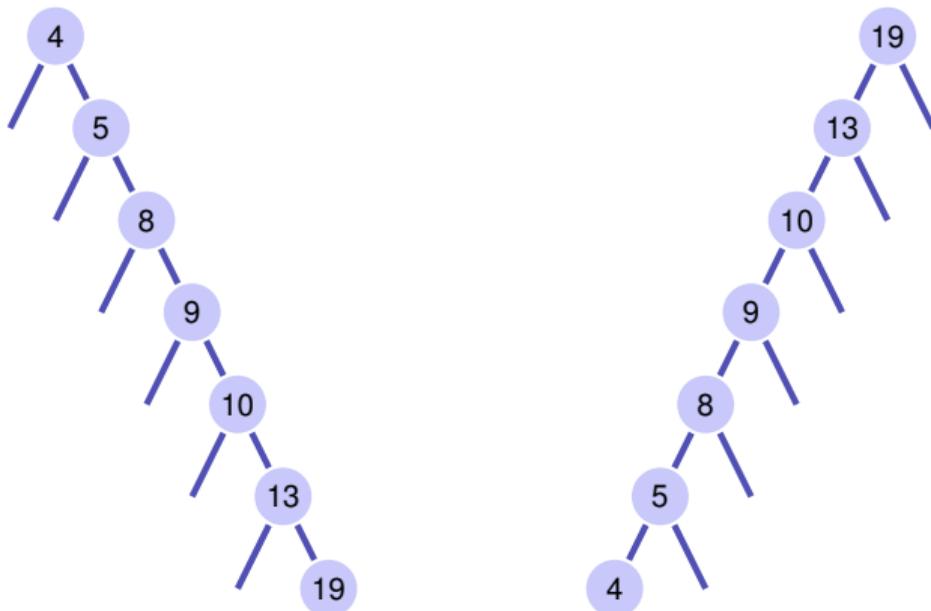
- $\text{Min}(T)$ : Auslesen des Minimums in  $\mathcal{O}(h)$
- $\text{ExtractMin}(T)$ : Auslesen und Entfernen des Minimums in  $\mathcal{O}(h)$
- $\text{List}(T)$ : Ausgeben einer sortierten Liste der Elemente von  $T$
- $\text{Join}(T_1, T_2)$ : Zusammenfügen zweier Bäume mit  $\max(T_1) < \min(T_2)$  in  $\mathcal{O}(n)$ .



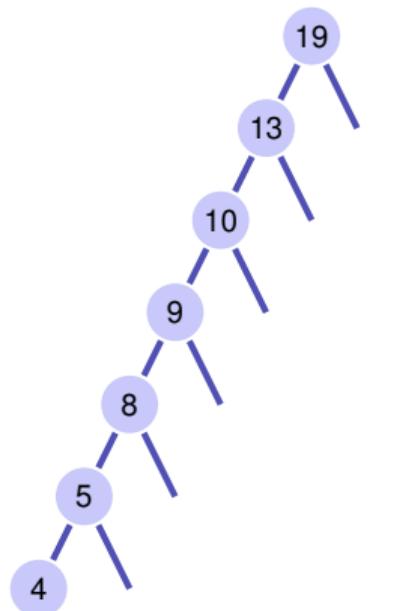
# Degenerierte Suchbäume



Insert 9,5,13,4,8,10,19  
bestmöglich  
balanciert



Insert 4,5,8,9,10,13,19  
Lineare Liste



Insert 19,13,10,9,8,5,4  
Lineare Liste

# 17. AVL Bäume

Balancierte Bäume [Ottman/Widmayer, Kap. 5.2-5.2.1, Cormen et al, Kap. Problem 13-3]

# Ziel

Suchen, Einfügen und Entfernen eines Schlüssels in Baum mit  $n$  Schlüsseln, welche in zufälliger Reihenfolge eingefügt wurden im Mittel in  $\mathcal{O}(\log_2 n)$  Schritten.

Schlechtester Fall jedoch:  $\Theta(n)$  (degenerierter Baum).

**Ziel:** Verhinderung der Degenerierung. Künstliches, bei jeder Update-Operation erfolgtes Balancieren eines Baumes

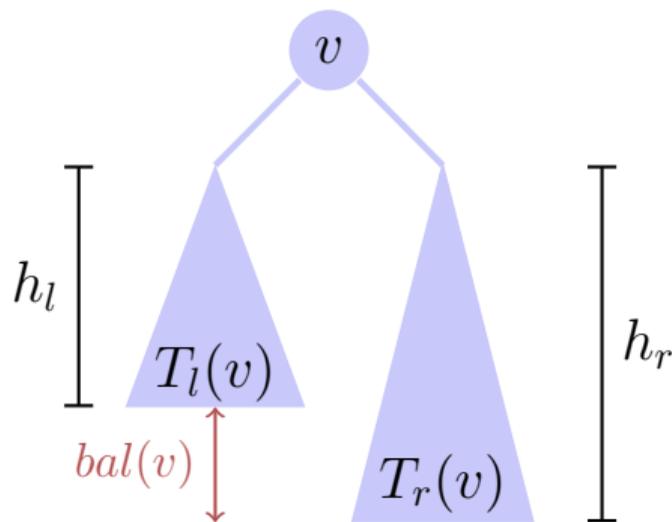
Balancierung: garantiere, dass ein Baum mit  $n$  Knoten stets eine Höhe von  $\mathcal{O}(\log n)$  hat.

**Adelson-Venskii und Landis (1962): AVL-Bäume**

# Balance eines Knotens

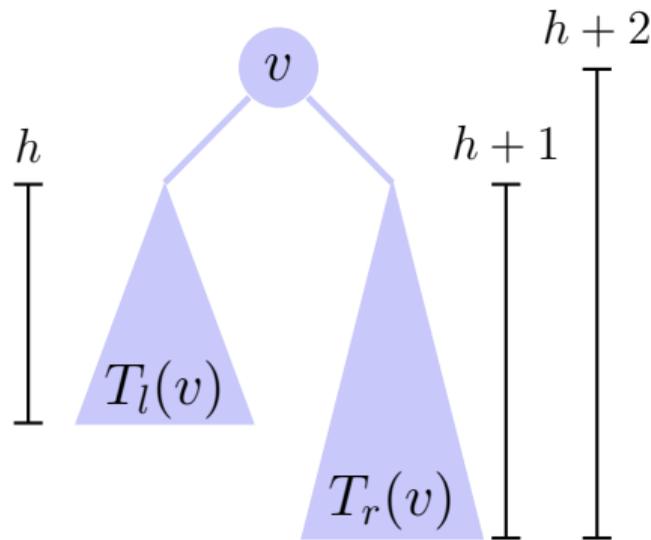
Die *Balance* eines Knotens  $v$  ist definiert als die Höhendifferenz seiner beiden Teilbäume  $T_l(v)$  und  $T_r(v)$

$$\text{bal}(v) := h(T_r(v)) - h(T_l(v))$$

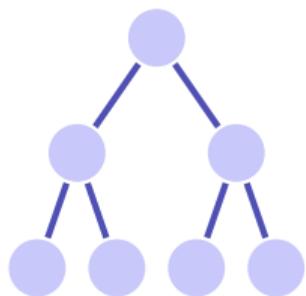


# AVL Bedingung

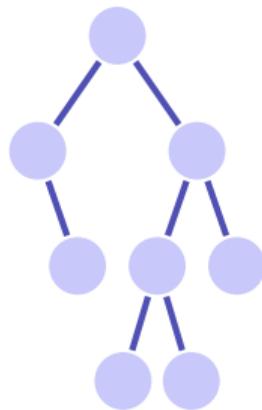
*AVL Bedingung: für jeden Knoten  $v$  eines Baumes gilt  $\text{bal}(v) \in \{-1, 0, 1\}$*



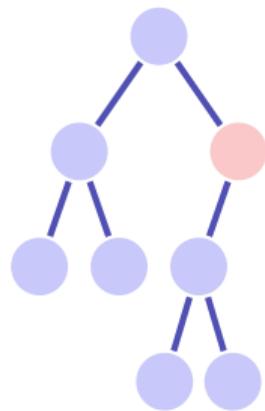
# (Gegen-)Beispiele



AVL Baum der Höhe  
2



AVL Baum der Höhe  
3

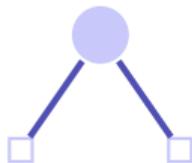


Kein AVL Baum

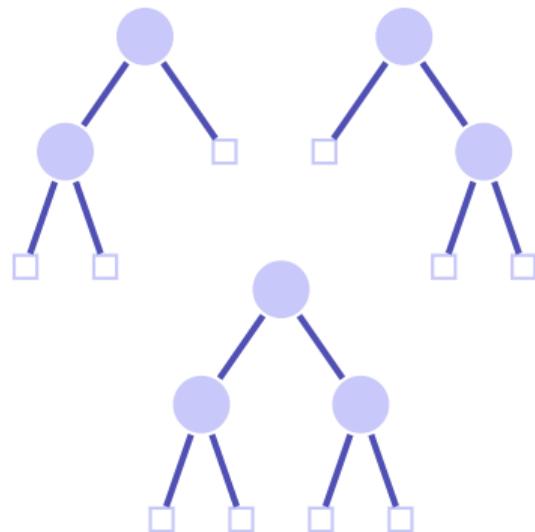
# Anzahl Blätter

- 1. Beobachtung: Ein Suchbaum mit  $n$  Schlüsseln hat genau  $n + 1$  Blätter. Einfaches Induktionsargument.
- 2. Beobachtung: untere Grenze für Anzahl Blätter eines Suchbaums zu gegebener Höhe erlaubt Abschätzung der maximalen Höhe eines Suchbaums zu gegebener Anzahl Schlüssel.

# Untere Grenze Blätter



AVL Baum der Höhe 1 hat  
 $M(1) := 2$  Blätter



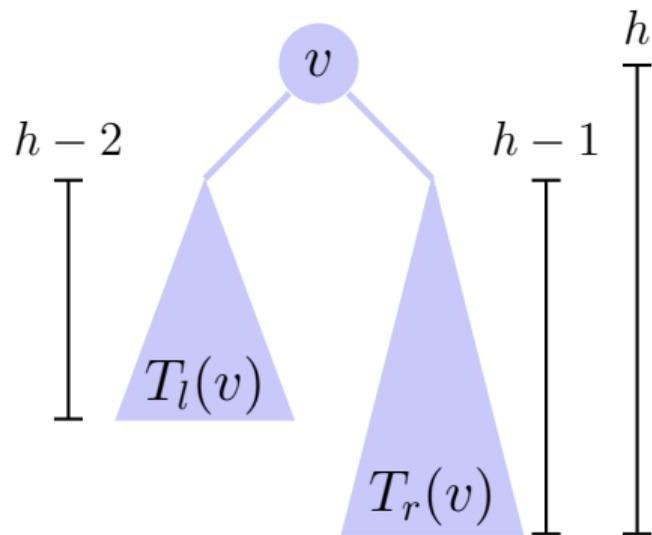
AVL Baum der Höhe 2 hat  
mindestens  $M(2) := 3$   
Blätter

# Untere Grenze Blätter für $h > 2$

- Höhe eines Teilbaums  $\geq h - 1$ .
- Höhe des anderen Teilbaums  $\geq h - 2$ .

Minimale Anzahl Blätter  $M(h)$  ist

$$M(h) = M(h - 1) + M(h - 2)$$



Insgesamt gilt  $M(h) = F_{h+2}$  mit **Fibonacci-Zahlen**  $F_0 := 0$ ,  $F_1 := 1$ ,  
 $F_n := F_{n-1} + F_{n-2}$  für  $n > 1$ .

# [Fibonacci Zahlen: geschlossene Form]

Geschlossene Form der Fibonacci Zahlen: Berechnung über erzeugende Funktionen:

## 1 Potenzreihenansatz

$$f(x) := \sum_{i=0}^{\infty} F_i \cdot x^i$$

# [Fibonacci Zahlen: geschlossene Form]

- 2 Für Fibonacci Zahlen gilt  $F_0 = 0$ ,  $F_1 = 1$ ,  
 $F_i = F_{i-1} + F_{i-2} \forall i > 1$ . Daher:

$$\begin{aligned} f(x) &= x + \sum_{i=2}^{\infty} F_i \cdot x^i = x + \sum_{i=2}^{\infty} F_{i-1} \cdot x^i + \sum_{i=2}^{\infty} F_{i-2} \cdot x^i \\ &= x + x \sum_{i=2}^{\infty} F_{i-1} \cdot x^{i-1} + x^2 \sum_{i=2}^{\infty} F_{i-2} \cdot x^{i-2} \\ &= x + x \sum_{i=0}^{\infty} F_i \cdot x^i + x^2 \sum_{i=0}^{\infty} F_i \cdot x^i \\ &= x + x \cdot f(x) + x^2 \cdot f(x). \end{aligned}$$

# [Fibonacci Zahlen: geschlossene Form]

3 Damit:

$$\begin{aligned} f(x) \cdot (1 - x - x^2) &= x. \\ \Leftrightarrow f(x) &= \frac{x}{1 - x - x^2} \\ \Leftrightarrow f(x) &= \frac{x}{(1 - \phi x) \cdot (1 - \hat{\phi} x)} \end{aligned}$$

mit den Wurzeln  $\phi$  und  $\hat{\phi}$  von  $1 - x - x^2$ .

$$\begin{aligned} \phi &= \frac{1 + \sqrt{5}}{2} \\ \hat{\phi} &= \frac{1 - \sqrt{5}}{2}. \end{aligned}$$

# [Fibonacci Zahlen: geschlossene Form]

4 Es gilt:

$$(1 - \hat{\phi}x) - (1 - \phi x) = \sqrt{5} \cdot x.$$

Damit:

$$\begin{aligned} f(x) &= \frac{1}{\sqrt{5}} \frac{(1 - \hat{\phi}x) - (1 - \phi x)}{(1 - \phi x) \cdot (1 - \hat{\phi}x)} \\ &= \frac{1}{\sqrt{5}} \left( \frac{1}{1 - \phi x} - \frac{1}{1 - \hat{\phi}x} \right) \end{aligned}$$

# [Fibonacci Zahlen: geschlossene Form]

5 Potenzreihenentwicklung von  $g_a(x) = \frac{1}{1-a \cdot x}$  ( $a \in \mathbb{R}$ ):

$$\frac{1}{1 - a \cdot x} = \sum_{i=0}^{\infty} a^i \cdot x^i.$$

Sieht man mit Taylor-Entwicklung von  $g_a(x)$  um  $x = 0$  oder so: Sei  $\sum_{i=0}^{\infty} G_i \cdot x^i$  eine Potenzreihenentwicklung von  $g$ . Mit der Identität  $g_a(x)(1 - a \cdot x) = 1$  gilt

$$1 = \sum_{i=0}^{\infty} G_i \cdot x^i - a \cdot \sum_{i=0}^{\infty} G_i \cdot x^{i+1} = G_0 + \sum_{i=1}^{\infty} (G_i - a \cdot G_{i-1}) \cdot x^i$$

Also  $G_0 = 1$  und  $G_i = a \cdot G_{i-1} \Rightarrow G_i = a^i$ .

# [Fibonacci Zahlen: geschlossene Form]

6 Einsetzen der Potenzreihenentwicklung:

$$\begin{aligned} f(x) &= \frac{1}{\sqrt{5}} \left( \frac{1}{1 - \phi x} - \frac{1}{1 - \hat{\phi} x} \right) = \frac{1}{\sqrt{5}} \left( \sum_{i=0}^{\infty} \phi^i x^i - \sum_{i=0}^{\infty} \hat{\phi}^i x^i \right) \\ &= \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i) x^i \end{aligned}$$

Koeffizientenvergleich mit  $f(x) = \sum_{i=0}^{\infty} F_i \cdot x^i$  liefert

$$F_i = \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i).$$

# Fibonacci Zahlen

Es gilt  $F_i = \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)$  mit den Wurzeln  $\phi, \hat{\phi}$  der Gleichung  $x^2 = x + 1$  (goldener Schnitt), also  $\phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$ .

Beweis (Induktion). Klar für  $i = 0, i = 1$ . Sei  $i > 2$ :

$$\begin{aligned} F_i &= F_{i-1} + F_{i-2} = \frac{1}{\sqrt{5}}(\phi^{i-1} - \hat{\phi}^{i-1}) + \frac{1}{\sqrt{5}}(\phi^{i-2} - \hat{\phi}^{i-2}) \\ &= \frac{1}{\sqrt{5}}(\phi^{i-1} + \phi^{i-2}) - \frac{1}{\sqrt{5}}(\hat{\phi}^{i-1} + \hat{\phi}^{i-2}) = \frac{1}{\sqrt{5}}\phi^{i-2}(\phi + 1) - \frac{1}{\sqrt{5}}\hat{\phi}^{i-2}(\hat{\phi} + 1) \\ &= \frac{1}{\sqrt{5}}\phi^{i-2}(\phi^2) - \frac{1}{\sqrt{5}}\hat{\phi}^{i-2}(\hat{\phi}^2) = \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i). \end{aligned}$$

# Baumhöhe

Da  $\hat{\phi} < 1$ , gilt insgesamt

$$M(h) \in \Theta \left( \left( \frac{1 + \sqrt{5}}{2} \right)^h \right) \subseteq \Omega(1.618^h)$$

und somit

$$h \leq 1.44 \log_2 n + c.$$

AVL Baum ist asymptotisch nicht mehr als 44% höher als ein perfekt balancierter Baum.

# Einfügen

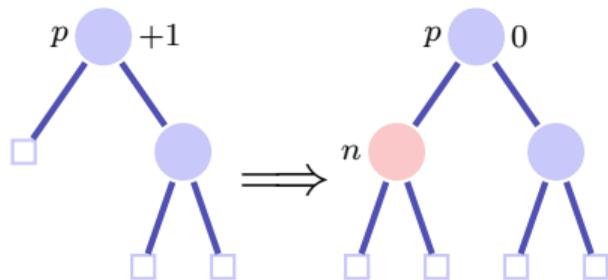
## Balancieren

- Speichern der Balance für jeden Knoten
- Baum Re-balancieren bei jeder Update-Operation

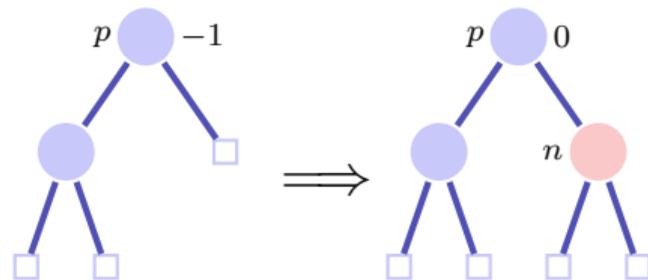
Neuer Knoten  $n$  wird eingefügt:

- Zuerst einfügen wie bei Suchbaum.
- Prüfe die Balance-Bedingung für alle Knoten aufsteigend von  $n$  zur Wurzel.

# Balance am Einfügeort



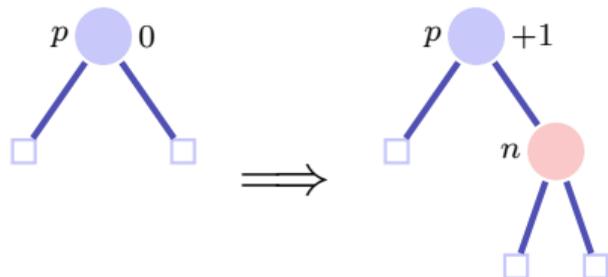
Fall 1:  $\text{bal}(p) = +1$



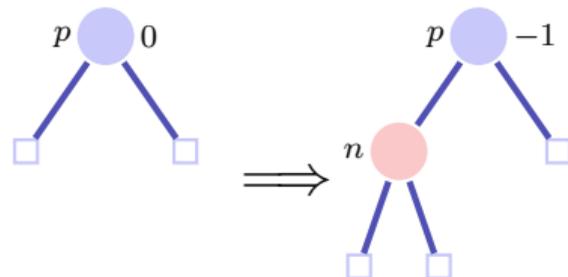
Fall 2:  $\text{bal}(p) = -1$

Fertig in beiden Fällen, denn der Teilbaum ist nicht gewachsen.

# Balance am Einfügeort



Fall 3.1:  $\text{bal}(p) = 0$  rechts



Fall 3.2:  $\text{bal}(p) = 0$ , links

In beiden Fällen noch nicht fertig. Aufruf von `upin(p)`.

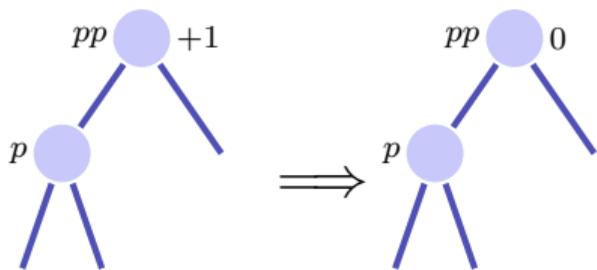
# upin(p) - Invariante

Beim Aufruf von `upin(p)` gilt, dass

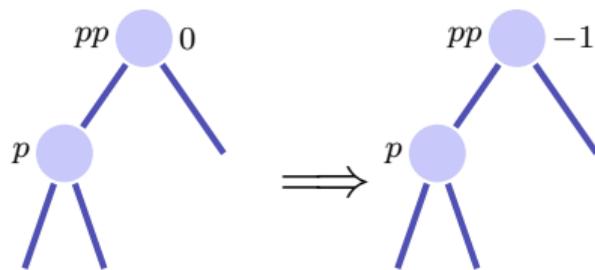
- der Teilbaum ab  $p$  gewachsen ist und
- $\text{bal}(p) \in \{-1, +1\}$

# upin(p)

Annahme:  $p$  ist linker Sohn von  $pp$ <sup>17</sup>



Fall 1:  $\text{bal}(pp) = +1$ , fertig.



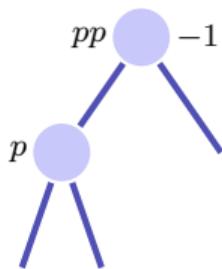
Fall 2:  $\text{bal}(pp) = 0$ , **upin(pp)**

In beiden Fällen gilt nach der Operation die AVL-Bedingung für den Teilbaum ab  $pp$

<sup>17</sup>Ist  $p$  rechter Sohn: symmetrische Fälle unter Vertauschung von  $+1$  und  $-1$

# upin(p)

Annahme:  $p$  ist linker Sohn von  $pp$



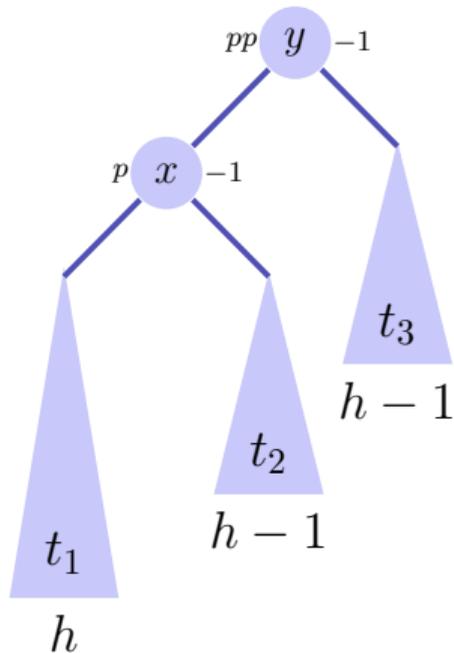
Fall 3:  $\text{bal}(pp) = -1$ ,

Dieser Fall ist problematisch: das Hinzufügen von  $n$  im Teilbaum ab  $pp$  hat die AVL-Bedingung verletzt. Rebalancieren!

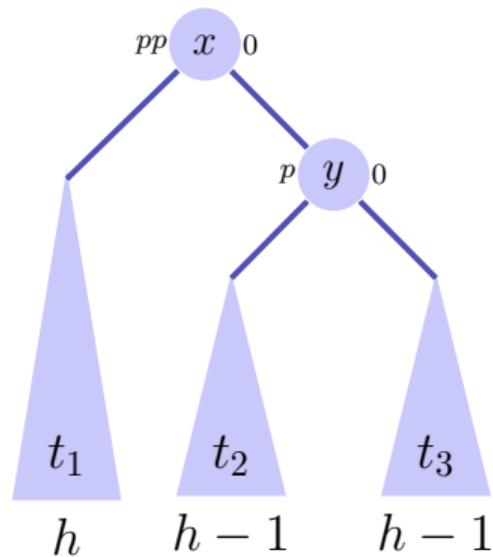
Zwei Fälle  $\text{bal}(p) = -1$ ,  $\text{bal}(p) = +1$

# Rotationen

Fall 1.1  $\text{bal}(p) = -1$ .<sup>18</sup>



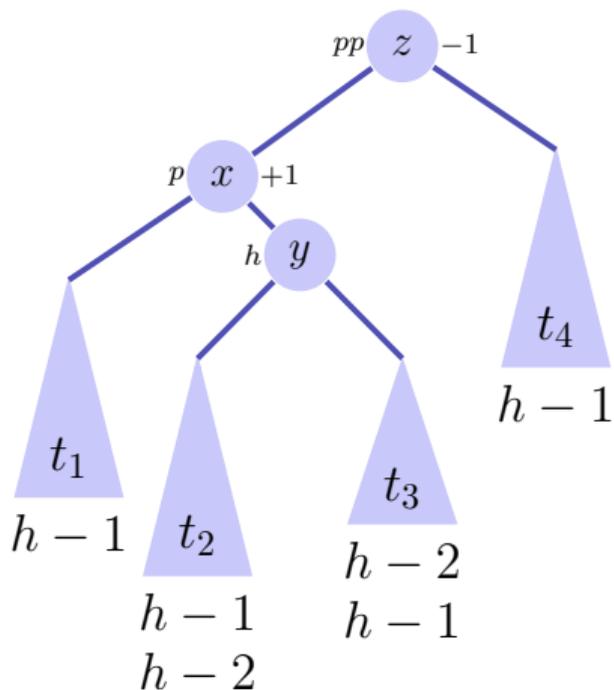
$\implies$   
Rotation  
nach  
rechts



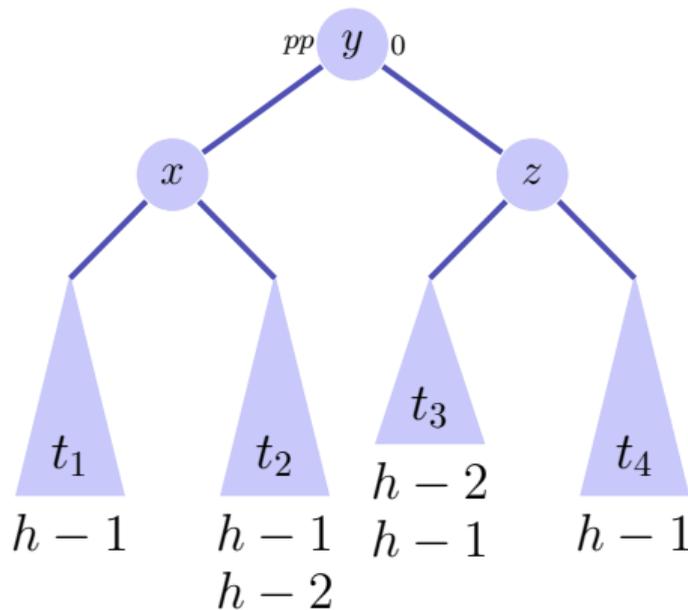
<sup>18</sup> $p$  rechter Sohn:  $\text{bal}(pp) = \text{bal}(p) = +1$ , Linksrotation

# Rotationen

Fall 1.2  $\text{bal}(p) = +1$ .<sup>19</sup>



$\implies$   
Doppel-  
rotation  
links-  
rechts



<sup>19</sup> $p$  rechter Sohn:  $\text{bal}(pp) = +1$ ,  $\text{bal}(p) = -1$ , Doppelrotation rechts links

# Analyse

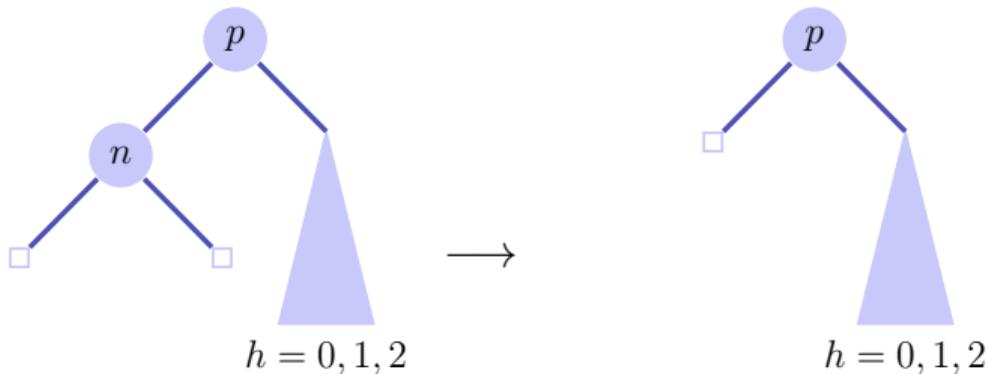
- Höhe des Baumes:  $\mathcal{O}(\log n)$ .
- Einfügen wie beim binären Suchbaum.
- Balancieren durch Rekursion vom Knoten zur Wurzel. Maximale Pfadlänge  $\mathcal{O}(\log n)$ .

Das Einfügen im AVL-Baum hat Laufzeitkosten von  $\mathcal{O}(\log n)$ .

# Löschen

Fall 1: Knoten  $n$  hat zwei Blätter als Kinder Sei  $p$  Elternknoten von  $n$ .  
 $\Rightarrow$  Anderer Teilbaum hat Höhe  $h' = 0, 1$  oder  $2$

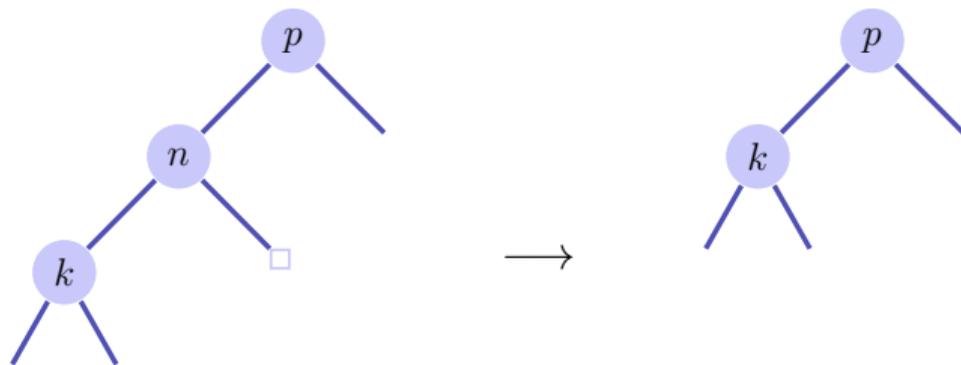
- $h' = 1$ :  $\text{bal}(p)$  anpassen.
- $h' = 0$ :  $\text{bal}(p)$  anpassen. Aufruf  $\text{upout}(p)$ .
- $h' = 2$ : Rebalancieren des Teilbaumes. Aufruf  $\text{upout}(p)$ .



# Löschen

Fall 2: Knoten  $n$  hat einen inneren Knoten  $k$  als Kind

- Ersetze  $n$  durch  $k$ . `upout(k)`



Fall 3: Knoten  $n$  hat zwei inneren Knoten als Kinder

- Ersetze  $n$  durch symmetrischen Nachfolger. `upout(k)`
- Löschen des symmetrischen Nachfolgers wie in Fall 1 oder 2.

# upout(p)

Sei  $pp$  der Elternknoten von  $p$

(a)  $p$  linkes Kind von  $pp$

1  $\text{bal}(pp) = -1 \Rightarrow \text{bal}(pp) \leftarrow 0$ . **upout(pp)**

2  $\text{bal}(pp) = 0 \Rightarrow \text{bal}(pp) \leftarrow +1$ .

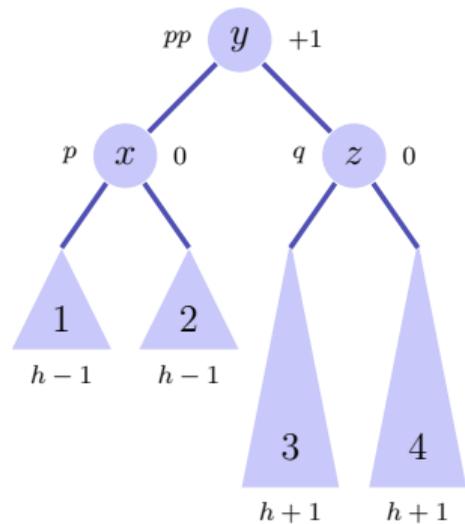
3  $\text{bal}(pp) = +1 \Rightarrow$  nächste Folien.

(b)  $p$  rechtes Kind von  $pp$ : Symmetrische Fälle unter Vertauschung von  $+1$  und  $-1$ .

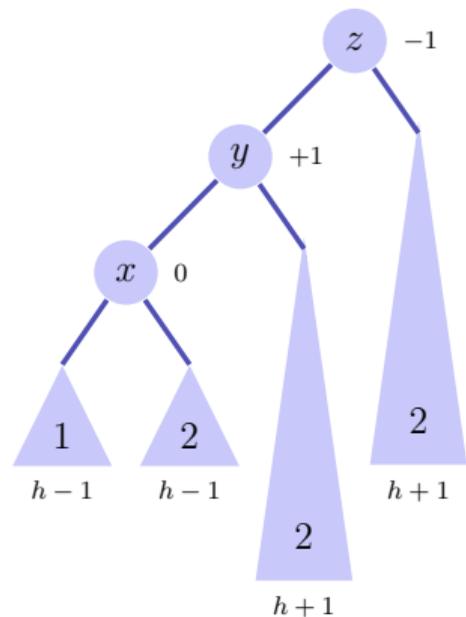
# upout (p)

Fall (a).3:  $\text{bal}(pp) = +1$ . Sei  $q$  Bruder von  $p$

(a).3.1:  $\text{bal}(q) = 0$ .<sup>20</sup>



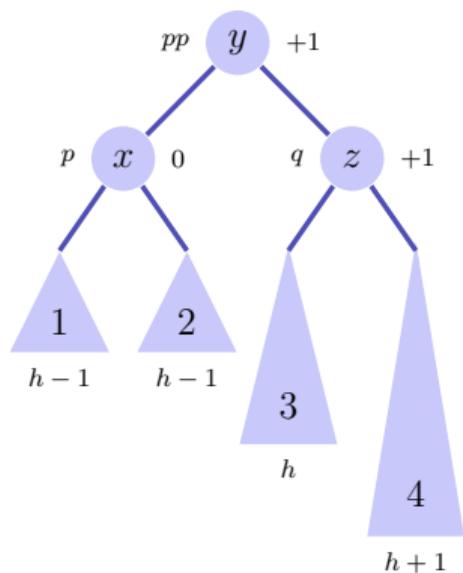
$\implies$   
Linksrotation  
(y)



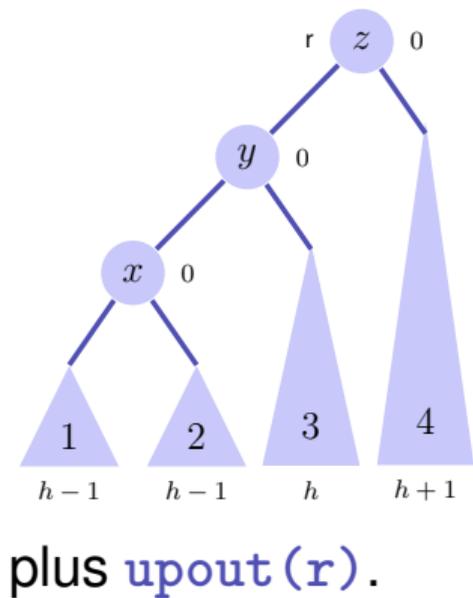
<sup>20</sup>(b).3.1:  $\text{bal}(pp) = -1$ ,  $\text{bal}(q) = -1$ , Rechtsrotation.

# upout (p)

Fall (a).3:  $\text{bal}(pp) = +1$ . (a).3.2:  $\text{bal}(q) = +1$ .<sup>21</sup>



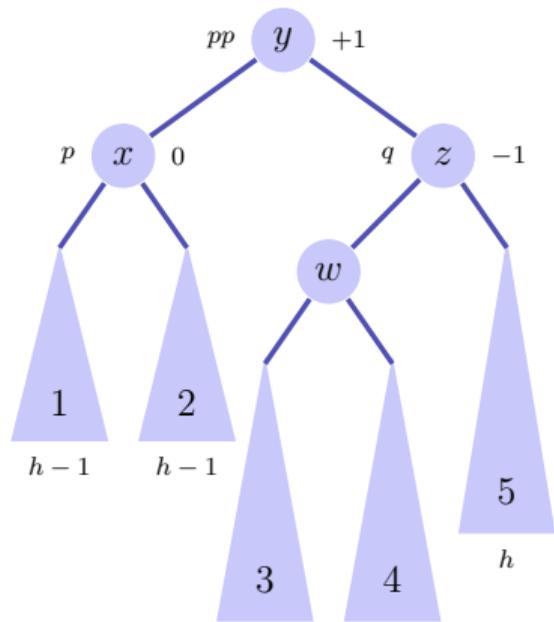
$\Rightarrow$   
Linksrotation  
(y)



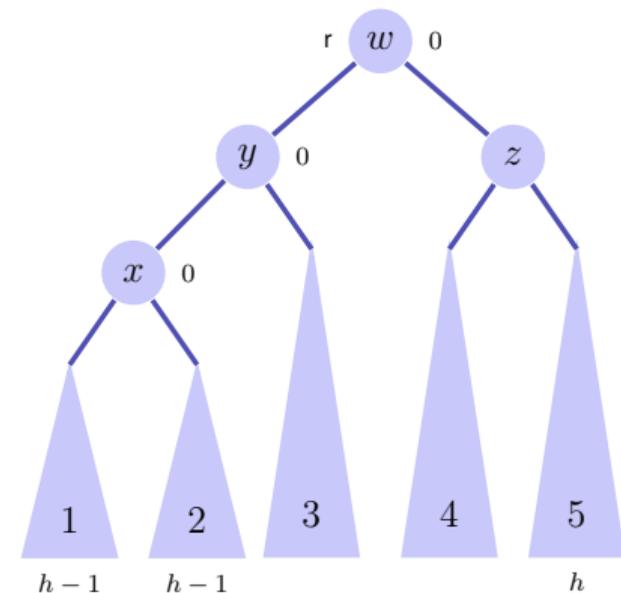
<sup>21</sup>(b).3.2:  $\text{bal}(pp) = -1$ ,  $\text{bal}(q) = +1$ , Rechtsrotation+upout

# upout (p)

Fall (a).3:  $\text{bal}(pp) = +1$ . (a).3.3:  $\text{bal}(q) = -1$ .<sup>22</sup>



$\implies$   
Doppelrotation  
rechts (z)  
links (y)



plus **upout (r)**.

<sup>22</sup>(b).3.3:  $\text{bal}(pp) = -1$ ,  $\text{bal}(q) = -1$ , Links-Rechts-Rotation + upout

# Zusammenfassung

- AVL-Bäume haben asymptotische Laufzeit von  $\mathcal{O}(\log n)$  (schlechtester Fall) für das Suchen, Einfügen und Löschen von Schlüsseln
- Einfügen und Löschen ist verhältnismässig aufwändig und für kleine Probleme relativ langsam.