IIF
INSTITUT FÜR
INFORMATIK
FREIBURG

Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(21 - Balancierte Bäume, AVL-Bäume)

Prof. Dr. Susanne Albers

Balancierte Bäume

Eine Klasse von binären Suchbäumen ist balanciert, wenn jede der drei Wörterbuchoperationen

Suchen
Einfügen
Entfernen

von Schlüsseln für einen Baum mit n Schlüsseln stets (im worst case) in O(log n) Schritten ausführbar ist.

Mögliche Balancierungsbedingungen:

Höhenbedingung	→ AVL-Bäume
Gewichtsbedingung	→ BB[α]-Bäume
Strukturbedingungen	→ Bruder-, 2-3-, a-b-, B-Bäume

Ziel: Höhe eines Baumes mit n Schlüsseln bleibt stets in O(log n).

AVL-Bäume

Schöpfer: Adelson-Velskii und Landis (1962)

- Suchen, Einfügen und Entfernen eines Schlüssels in einem zufällig erzeugten natürlichen Suchbaum mit n Schlüsseln ist im Mittel in $O(\log_2 n)$ Schritten ausführbar.
- Der Worst Case liegt jedoch bei $\Omega(n)$.
- Idee von AVL-Bäumen: Modifizierte Prozeduren zum Einfügen und Löschen, die ein Degenerieren des Suchbaums verhindern.
- Ziel von AVL-Bäumen: Höhe sollte $O(\log_2 n)$ und das Suchen, Einfügen und Löschen sollte in logarithmischer Zeit möglich sein.

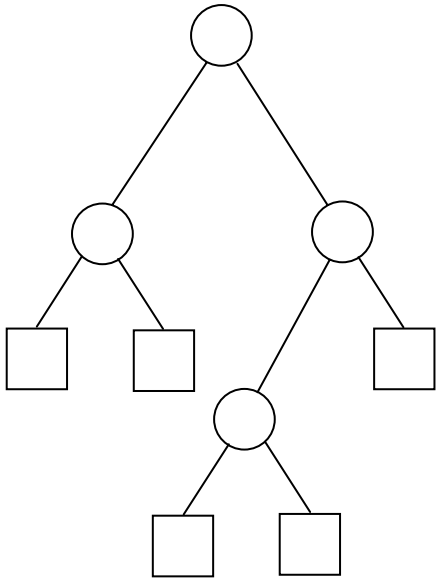
Definition von AVL-Bäumen

Definition: Ein binärer Suchbaum heißt AVL-Baum oder höhenbalanciert, wenn für jeden Knoten v gilt, dass sich die Höhe des rechten Teilbaumes $h(T_r)$ von v und die Höhe des linken Teilbaumes $h(T_l)$ von v um maximal 1 unterscheiden.

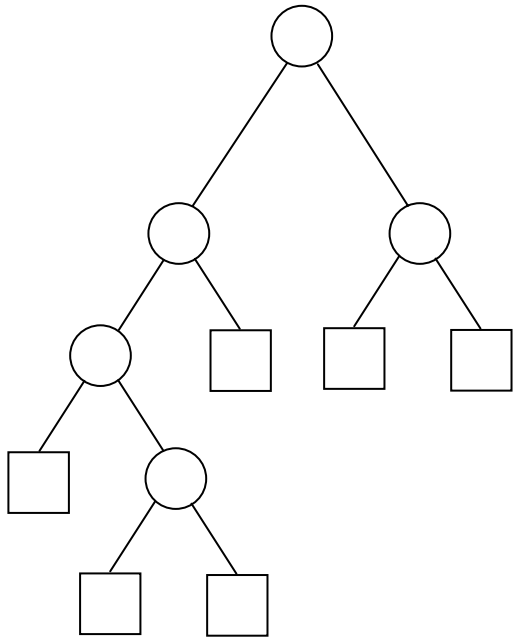
$$bal(v) = h(T_r) - h(T_l) \in \{-1, 0, +1\}$$

Balancegrad:

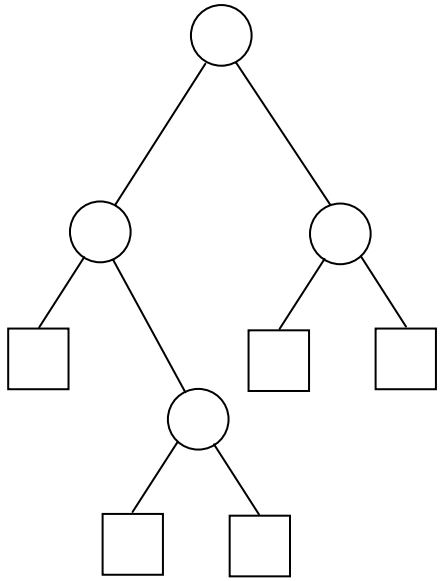
Beispiele



AVL-Baum



kein AVL-Baum



AVL-Baum

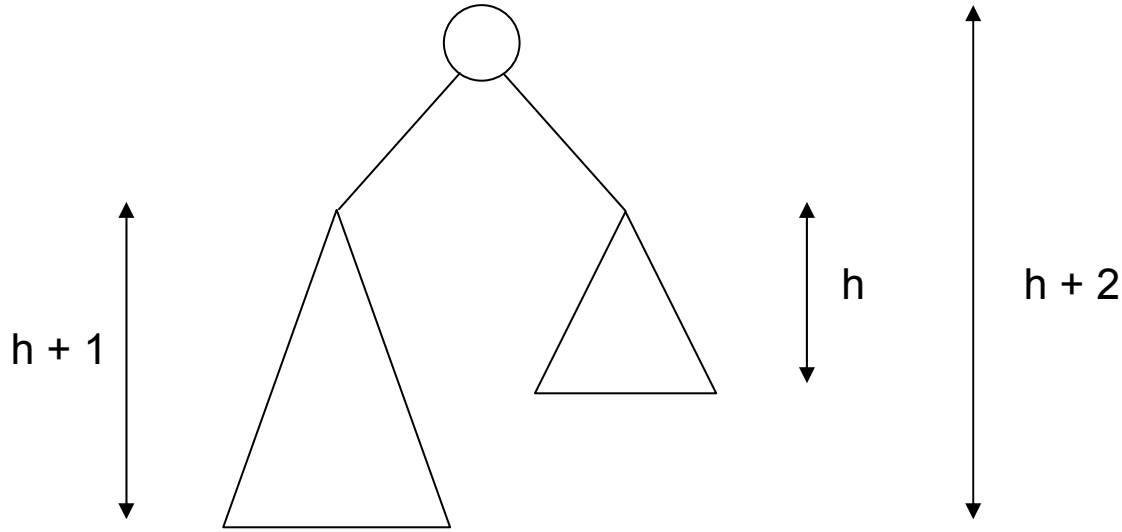
Eigenschaften von AVL-Bäumen

- AVL-Bäume können nicht zu linearen Listen degenerieren.
- AVL-Bäume mit *n* Knoten haben eine Höhe von O(log n).

Offenbar gilt:

- Ein AVL-Baum der Höhe 0 hat 1 Blatt
- Ein AVL-Baum der Höhe 1 hat 2 Blätter
- ein AVL-Baum der Höhe 2 mit minimaler Blattzahl hat 3 Blätter
- . . .
- Wie viele Blätter hat ein AVL-Baum der Höhe *h* mit minimaler Blattzahl?

Minimale Blattanzahl von AVL-Bäumen mit Höhe h



Folgerung: Ein AVL-Baum der Höhe *h* hat mindestens F_{h+2} Blätter mit

$$F_0 = 0$$

$$F_1 = 1$$

$$F_{i+2} = F_{i+1} + F_i$$

↝ F_i ist die i-te Fibonacci-Zahl.

Höhe eines AVL-Baumes

Satz: Die Höhe *h* eines AVL-Baumes mit *n* Blättern (und *n* - 1 inneren Knoten) beträgt höchstens

$$h \leq c \log_2 n, \text{ mit einer Konstanten c.}$$

Beweis: Für die Fibonacci-Zahlen gilt:

$$F_h = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^h - \left(\frac{1-\sqrt{5}}{2}\right)^h\right)$$

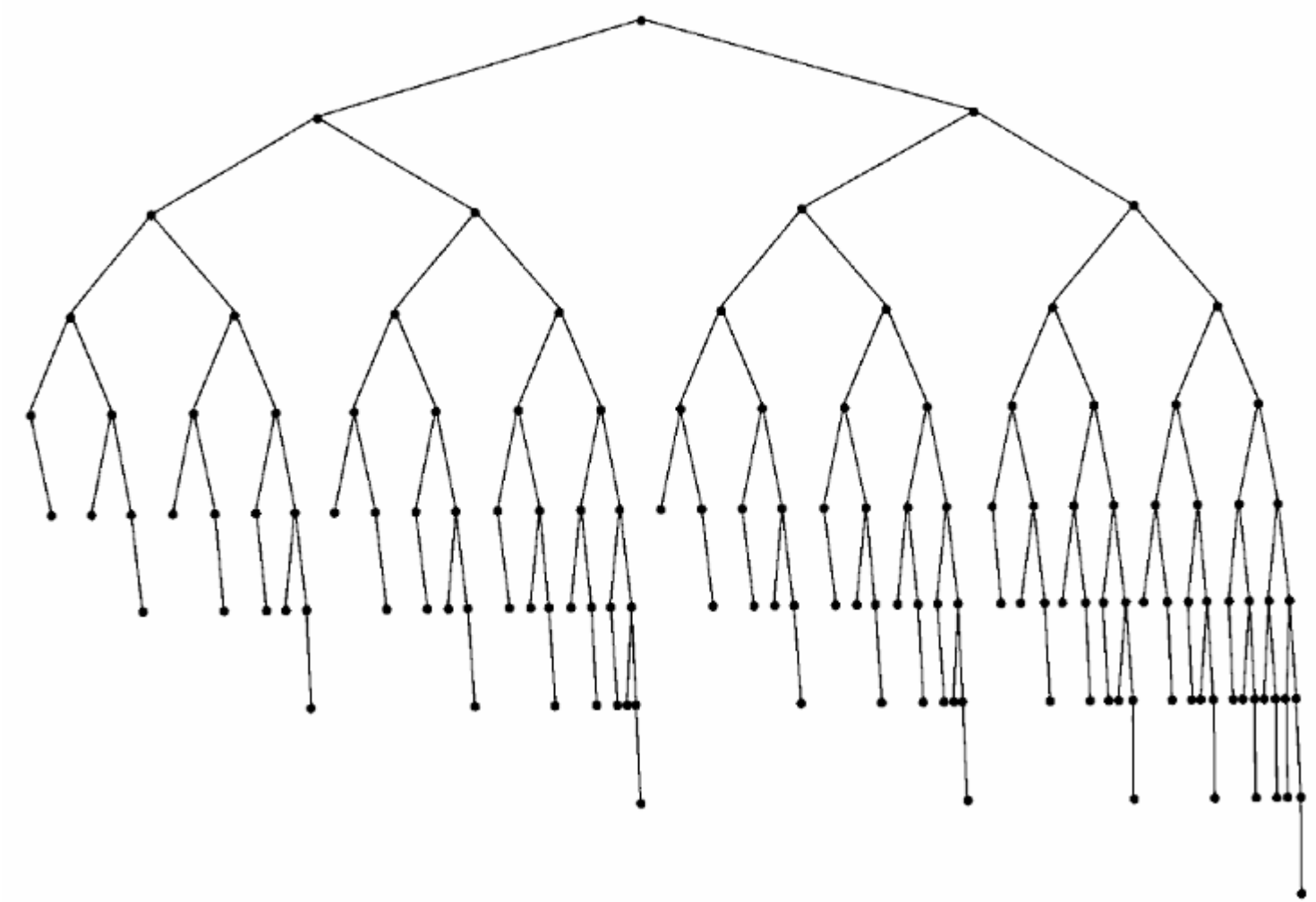
Wegen

$$n \geq F_{h+2} \geq (1.618\ldots)^h$$

folgt somit

$$h \leq \frac{1}{\log_2 1.618\ldots} * \log_2 n \leq 1.44..\log_2 n.$$

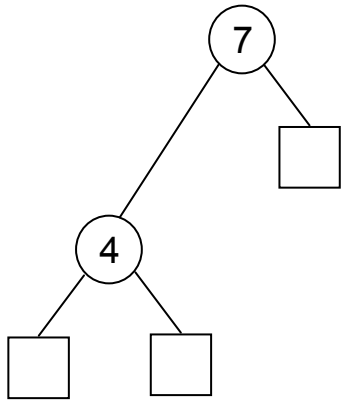
Minimaler AVL-Baum der Höhe 9



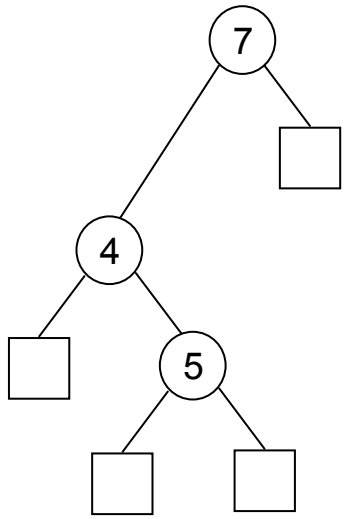
Einfügen in einen AVL-Baum

- Bei jeder Modifikation des Baums müssen wir garantieren, dass die AVL-Baum-Eigenschaft erhalten bleibt.

Ausgangssituation:

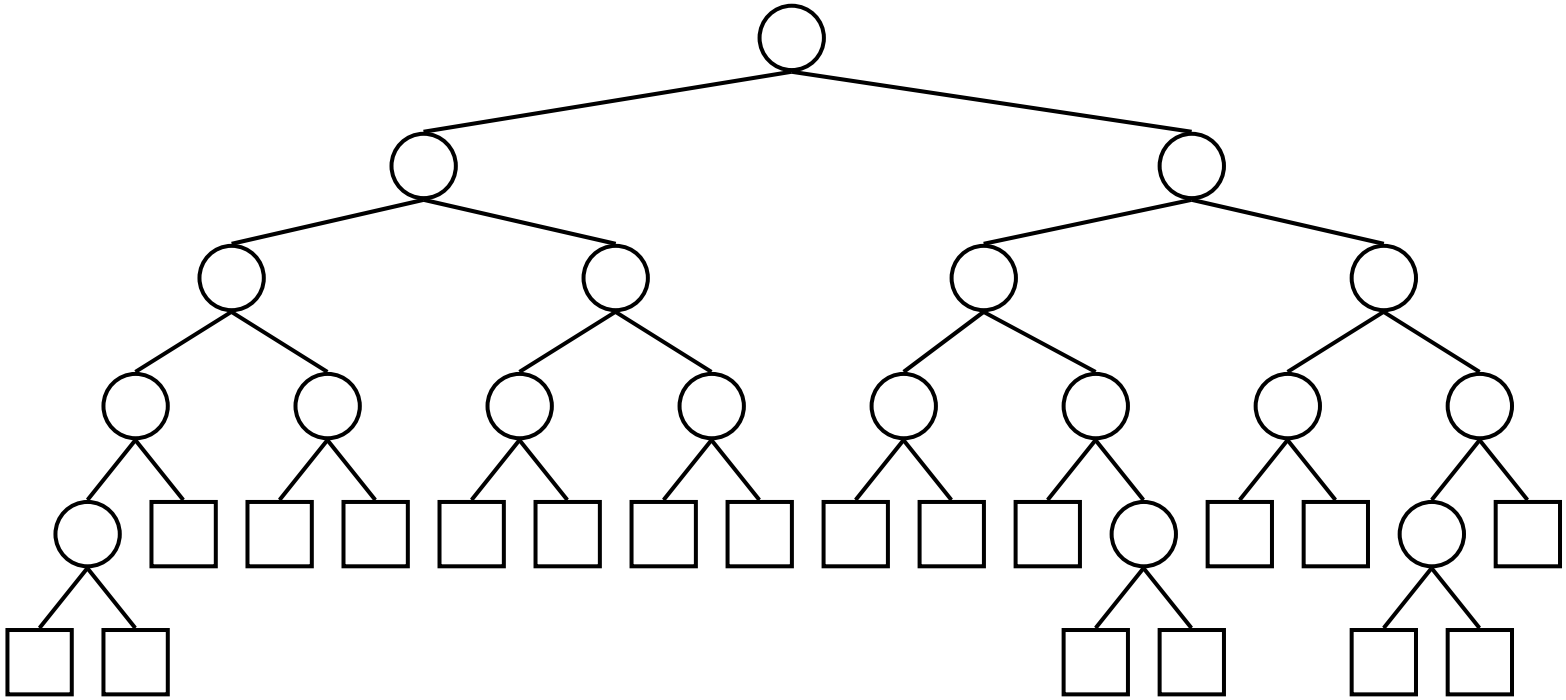


Nach Einfügen von 5:



Problem: Wie können wir den neuen Baum so modifizieren, dass ein AVL-Baum daraus entsteht?

Beispiel eines AVL-Baumes

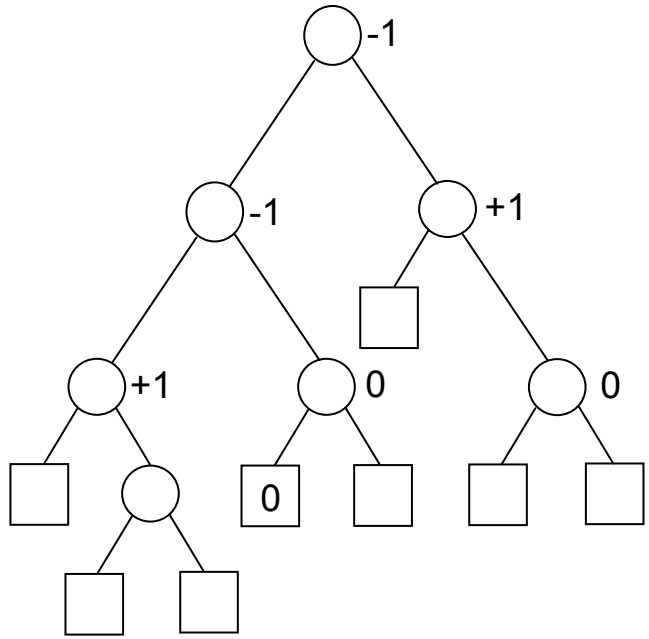


Speichern des Balancegrads in den Knoten

- Um die AVL-Baum-Eigenschaft wiederherzustellen, genügt es, in jedem Knoten den Balancegrad mitzuführen.
- Laut Definition gilt

$$bal(p) = h(p.right) - h(p.left) \in \{-1, 0, +1\}$$

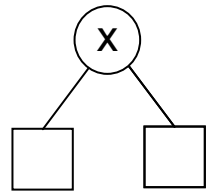
Beispiel:



Die verschiedenen Situationen beim Einfügen in den AVL-Baum

1. Der Baum ist leer: Schaffe einen einzigen Knoten mit zwei Blättern,

   speichere dort x und fertig!



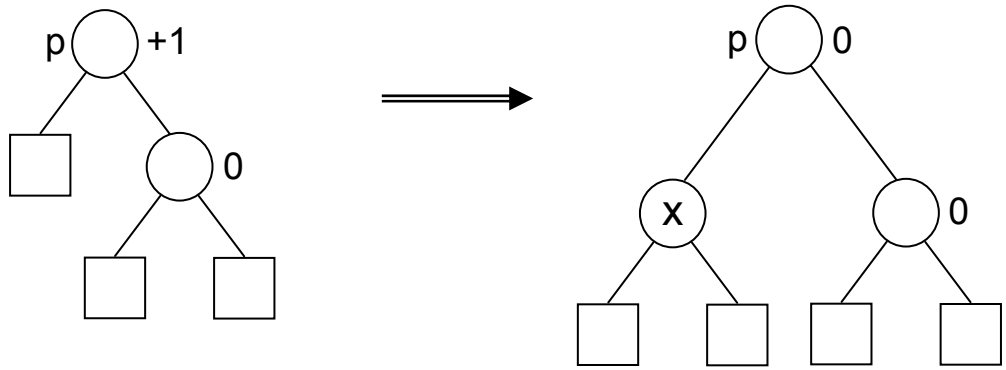
2. Der Baum ist nicht leer und die Suche endet bei einem Blatt.

   Sei Knoten *p* Vater des Blattes, bei dem Suche endet. Wegen bal(p) $\in$ {-1,0,1} muss gelten, dass entweder

- der linke Nachfolger von *p* ein Blatt ist, aber nicht der rechte (Fall 1) oder
- der rechte Nachfolger von *p* ein Blatt ist aber nicht der linke (Fall 2) oder
- beide Nachfolger von *p* sind Blätter (Fall 3).

Gesamthöhe unverändert (1)

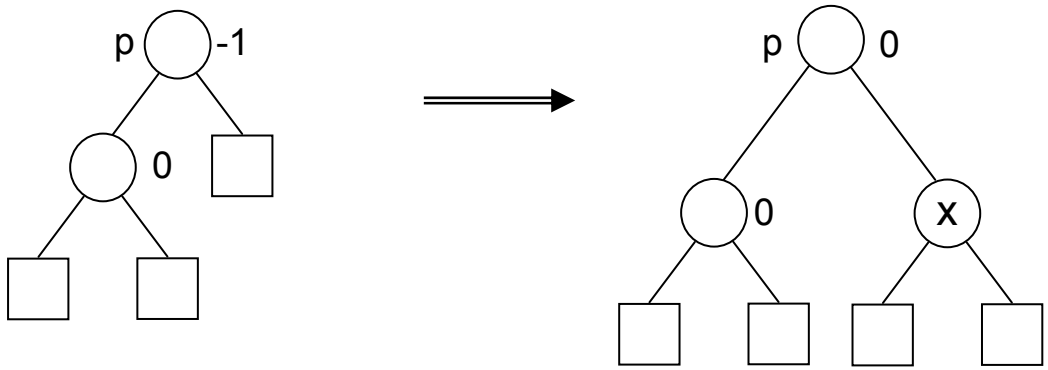
Fall 1: [*bal*(*p*) = + 1] und *x* < *p.key*, da Suche bei Blatt mit Vater *p* endet.



fertig!

Gesamthöhe unverändert (2)

Fall 2: [*bal*(*p*) = - 1] und *x* > *p.key*, da Suche bei Blatt mit Vater *p* endet.



fertig!

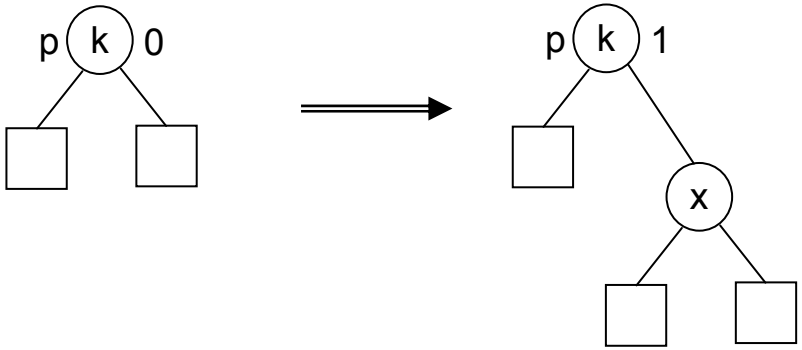
Beide Fälle sind unkritisch. Die Höhe des Teilbaums, in dem *p* sich befindet, ändert sich nicht.

Der kritische Fall

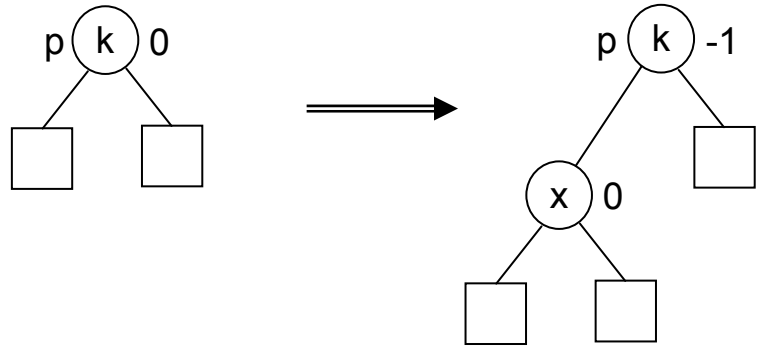
Fall 3: [*bal*(*p*) = 0] Dann sind beide Söhne von p Blätter. Die Höhe wächst!

Wir unterscheiden, ob wir den neuen Schlüssel x als rechten oder linken Nachfolger von p einfügen müssen:

[*bal(p) =0 und x > p.key*]



[*bal(p) =0 und x < p.key*]



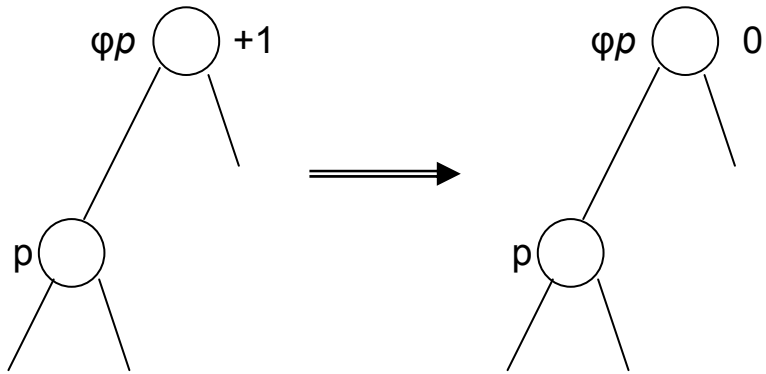
- In beiden Fällen benötigen wir eine Prozedur *upin*(*p*), die den Suchpfad zurückläuft, die Balancegrade prüft und Umstrukturierungen (so genannte Rotationen oder Doppelrotationen) durchführt.

Die Prozedur *upin*(*p*)

- Wenn *upin*(*p*) aufgerufen wird, ist stets $bal(p) \in \{-1, +1\}$ und die Höhe des Teilbaums mit Wurzel *p* ist um 1 gewachsen.
- *upin*(*p*) startet bei *p* und geht schrittweise nach oben (ggf. bis zur Wurzel).
- In jedem Schritt wird dabei versucht, die AVL-Baum-Eigenschaft wiederherzustellen.
- Wir konzentrieren uns im folgenden auf die Situation, dass *p* linker Nachfolger seines Vorgängers φ*p* ist.
- Die Situation, dass *p* rechter Nachfolger seines Vorgängers φ*p* ist, kann analog behandelt werden.

Fall1: *bal*(φ*p*) = 1

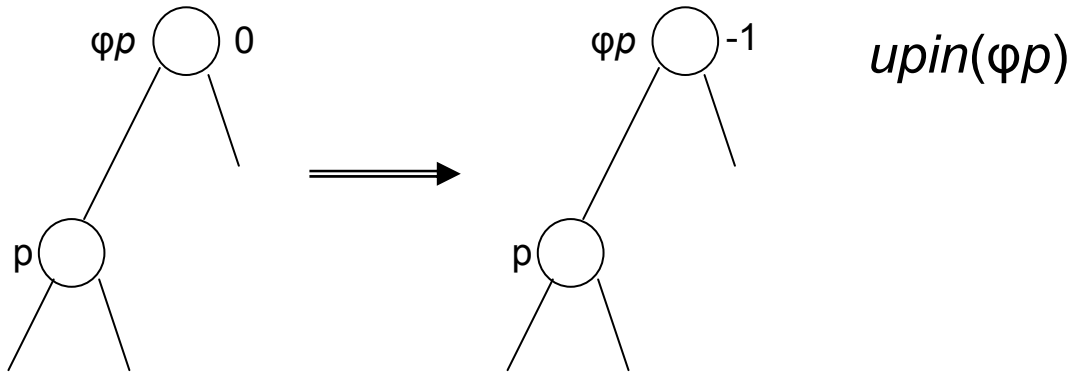
1. Der Vorgänger φ*p* hat den Balancegrad +1. Da sich die Höhe des Teilbaums mit Wurzel *p* als linker Nachfolger von φ*p* um 1 erhöht hat, genügt es, den Balancegrad von φ*p* auf 0 zu setzen:



fertig!

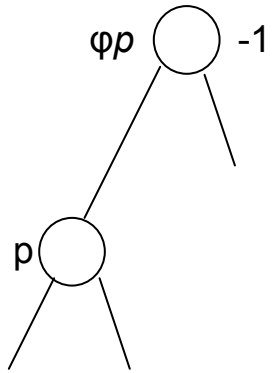
Fall 2: *bal*(φ*p*) = 0

2. Der Vorgänger φ*p* hat den Balancegrad 0. Da sich die Höhe des Teilbaums mit Wurzel *p* als linker Nachfolger von φ*p* sich um 1 erhöht hat, ändert sich der Balancegrad von φ*p* auf -1. Da sich gleichzeitig die Höhe des Teibaums mit Wurzel φ*p* verändert hat, müssen wir *upin* rekursiv mit φ*p* als Argument aufrufen.



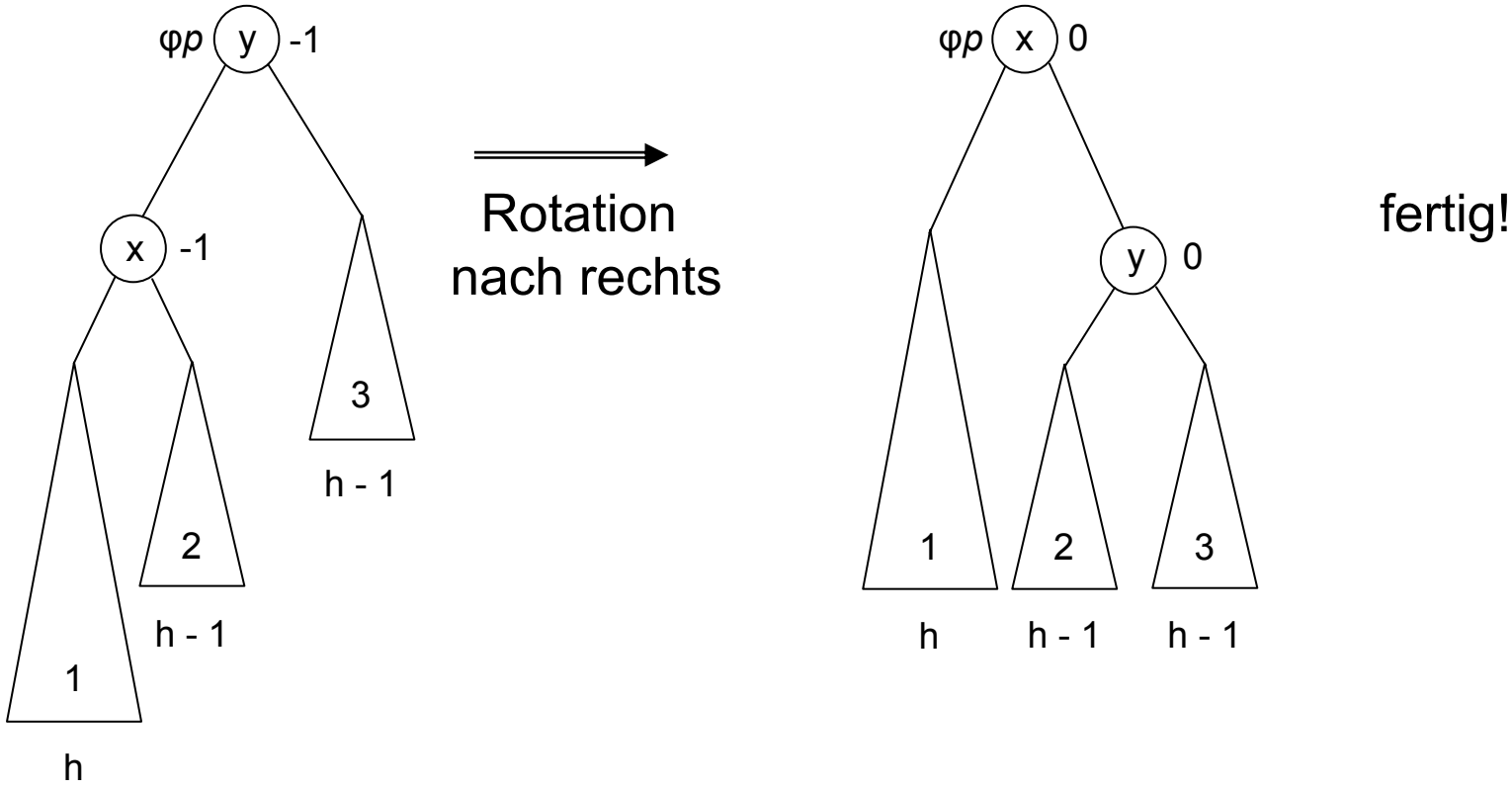
upin(φ*p*)

Der kritische Fall 3: *bal*(φ*p*) = -1



- Wenn *bal*(φ*p*) = -1 und die Höhe des linken Teilbaums *p* von φ*p* um 1 gewachsen ist, muss die AVL-Baum-Eigenschaft in φ*p* verletzt sein.
- In diesem Fall müssen wir den Baum umstrukturieren.
- Erneut unterscheiden wir zwei Fälle, nämlich *bal*(*p*) = -1 (Fall 3.1) und *bal*(*p*) = +1 (Fall 3.2).
- Die Invariante beim Aufruf von *upin(p)* bedeutet, dass *bal*(*p*) ≠ 0. Der Fall *bal*(*p*) = 0 kann also nicht vorliegen!

Fall 3.1: *bal*(φ*p*) = -1 und *bal*(*p*) = -1



Ist der resultierende Baum noch ein Suchbaum?

Es muss garantiert sein, dass der resultierende Baum die

1. Suchbaumeigenschaft und die
2. AVL-Baum-Eigenschaft erfüllt.

Suchbaumeigenschaft: Da der ursprüngliche Baum die Suchbaumeigenschaft erfüllt, muss gelten:

Alle Schlüssel in Baum 1 sind kleiner als x.

Alle Schlüssel in Baum 2 sind größer als x und kleiner als y.

Alle Schlüssel in Baum 3 sind größer als y (und x).

Daher erfüllt auch der resultierende Baum die Suchbaumeigenschaft.

Ist der resultierende Baum balanciert?

AVL-Baum-Eigenschaft: Da der ursprüngliche Baum ein AVL-Baum war, muss gelten:

- Wegen *bal*(φ*p*) = -1 haben Baum 2 und Baum 3 die gleiche Höhe h -1.
- Wegen *bal*(*p*) = -1 nach dem Einfügen, hat Baum 1 die Höhe *h*, während Baum 2 die Höhe h - 1 hat.

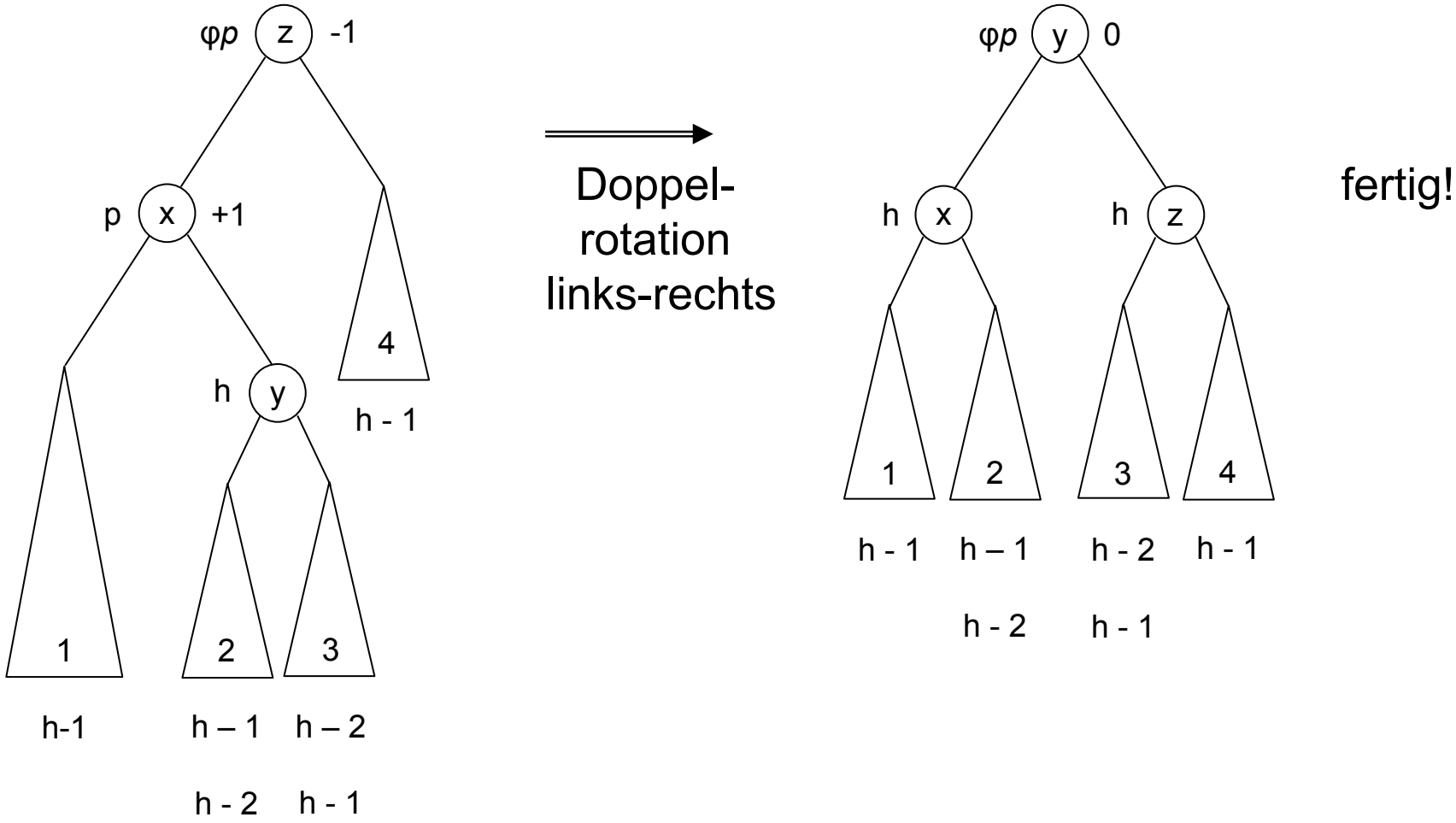
Damit gilt nach der Rotation:

- Der Knoten, der *y* enthält, hat Balancegrad 0.
- Der Knoten φ*p* hat Balancegrad 0.

Somit ist der AVL-Baum-Eigenschaft wieder hergestellt.

Fall 3.2: *bal*(φ*p*) = -1 und *bal*(*p*) = +1

Fall 3.2: *bal*(φ*p*) = -1 und *bal*(*p*) = +1



Eigenschaften der Teilbäume

1. Der neue Schlüssel muss in den rechten Teilbaum von *p* eingefügt worden sein.
2. Die Bäume 2 und 3 müssen unterschiedliche Höhe haben, weil sonst die Methode *upin* nicht aufgerufen worden wäre.
3. Die einzig mögliche Kombination der Höhen in den Bäumen 2 und 3 ist somit (h - 1,h - 2) und (h - 2,h - 1), sofern sie nicht leer sind.
4. Wegen $bal(p) = 1$ muss Baum 1 die Höhe h - 1 haben
5. Schließlich muss auch Baum 4 die Höhe h - 1 haben (wegen $bal(\varphi p) = -1$.

Somit erfüllt der resultierende Baum ebenfalls die AVL-Baum-Eigenschaft.

Suchbaumeigenschaft

Es gilt:

1. Die Schlüssel in Baum 1 sind sämtlich kleiner als *x*.
2. Die Schlüssel in Baum 2 sind sämtlich kleiner als *y* aber größer als *x*.
3. Die Schlüssel in Baum 3 sind alle größer als *y* und *x* aber kleiner als *z*.
4. Die Schlüssel in Baum 4 sind alle größer als *x*, *y* und *z*.

Daher hat auch der durch die Doppelrotation entstandene Baum die Suchbaumeigenschaft.

Hinweise

- Wir haben lediglich den Fall betrachtet, dass *p* linker Nachfolger seines Vorgängers φ*p* ist.
- Der Fall, dass *p* rechter Nachfolger seines Vorgängers φ*p* ist, kann analog behandelt werden.
- Um die Methode *upin*(*p*) effizient zu implementieren, müssen wir bei der Suche nach der Einfügestelle des neuen Schlüssels eine Liste aller besuchten Knoten anlegen.
- Dann können wir diese Liste bei den rekursiven Aufrufen nutzen, um jeweils zum Vorgänger überzugehen und ggf. die erforderlichen Rotationen oder Doppelrotationen auszuführen.

Das Einfügen in einen nicht leeren AVL-Baum

Suche nach x endet bei einem Blatt mit Vorgänger p

1. Rechter Sohn von p kein Blatt, $x < p.key$ → Anhängen links von *p*, fertig.
2. Linker Sohn von p kein Blatt, $x > p.key$ → Anhängen rechts von *p*, fertig.
3. Beide Söhne von *p* sind Blätter: Füge x als Sohn von p ein, *upin*(*p*)

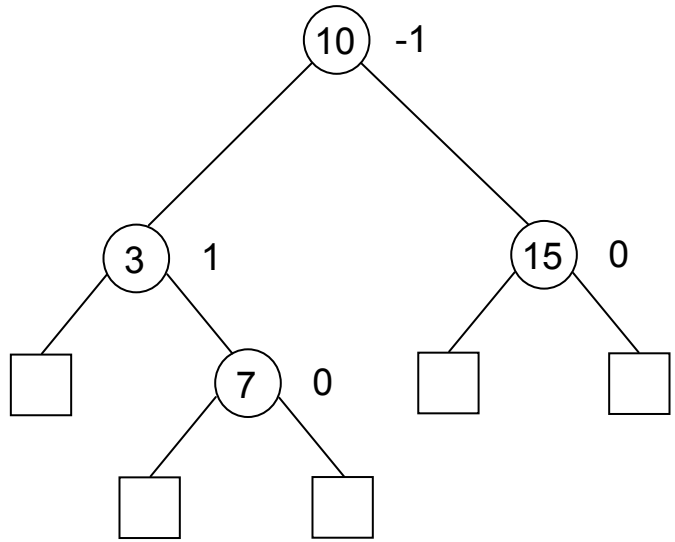
Die Methode *upin*(*p*):

1. *p* ist linker Nachfolger von φ*p*
   - (a) *bal*(φ*p*) = 1 → *bal*(φ*p*) = 0, fertig.
   - (b) *bal*(φ*p*) = 0 → *bal*(φ*p*) = -1, *upin*(φ*p*)
   - (c) i. *bal*(φ*p*) = -1 und *bal*(*p*) = -1 Rotation nach rechts, fertig.
   - ii. *bal*(φ*p*) = - und *bal*(*p*) = +1 Doppelrotation links-rechts, fertig.
2. *p* ist rechter Nachfolger von φ*p*.

   ...

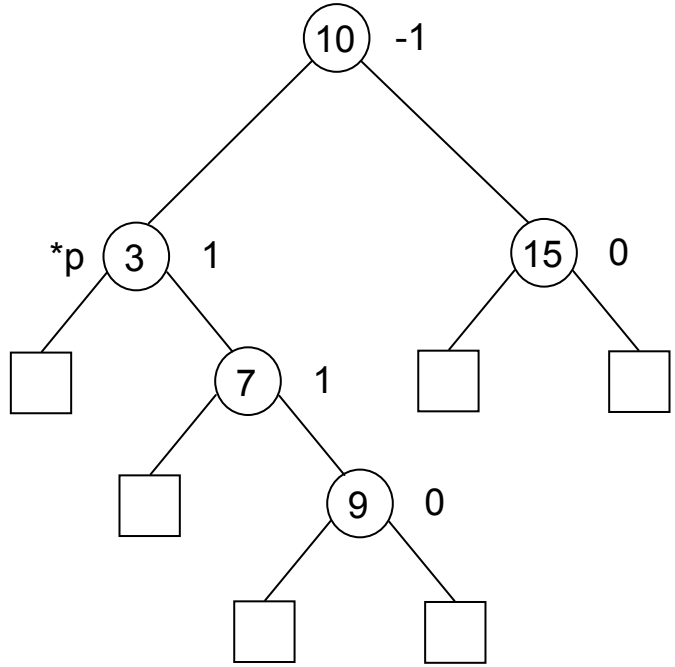
Ein Beispiel (1)

Ausgangssituation:



Ein Beispiel (2)

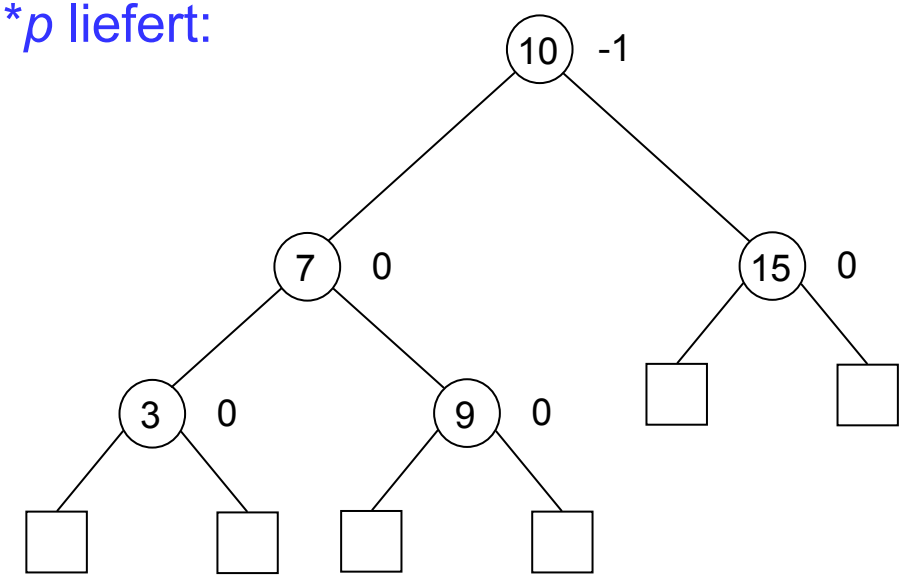
Einfügen von Schlüssel 9:



AVL-Baum-Eigenschaft ist verletzt!

Ein Beispiel (3)

Linksrotation bei **p* liefert:



Ein Beispiel (4)

Einfügen von 8 mit anschließender Doppelrotation liefert: