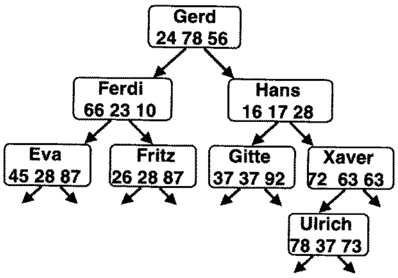
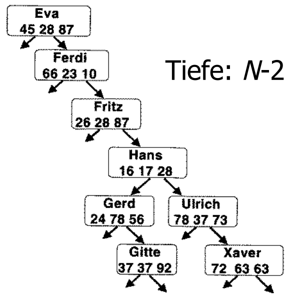
Balancierte Bäume

- Aufwand, ein Element zu finden, entspricht der Tiefe des gefundenen Knotens
  - im worst case = Tiefe des Baumes
  - liegt zwischen ⌊log N⌋+1 und N



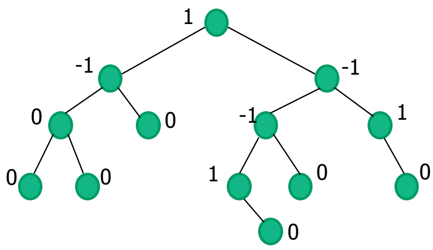


Tiefe: *N*-2

- Definition für "balanciert":
  - es gibt verschiedene Definitionen
  - Allgemein: kein Blatt ist "wesentlich weiter" von der Wurzel entfernt als irgendein anderes
  - Hier: Für alle Knoten unterscheidet sich Anzahl der Knoten in linkem und rechtem Teilbaum höchstens um 1
  - Folge: ein binärer Baum der Tiefe ⌊log N⌋+1
- schlecht balancierte Bäume
  - erhält man, wenn die Elemente in sortierter Reihenfolge angeliefert werden
  - Aufwand, einen optimal balancierten Baum nach Einfüge- und Löschoperationen zu erzwingen, ist sehr groß

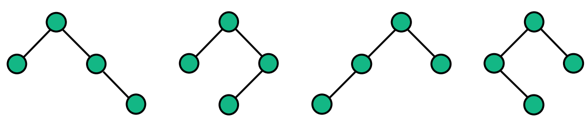
AVL-Bäume

- AVL-Baum
  - 1962 von Adelson, Velskij und Landis eingeführt
  - schwächere Form eines balancierten Baumes
- Definition Balance-Faktor:
  - bal(x) = (Höhe des rechten Unterbaumes von x) -
    (Höhe des linken Unterbaumes von x)
- Definition AVL-Baum:
  binärer Baum, wobei
  für jeden Knoten x gilt:
  bal(x) $\in$ {-1,0,1}



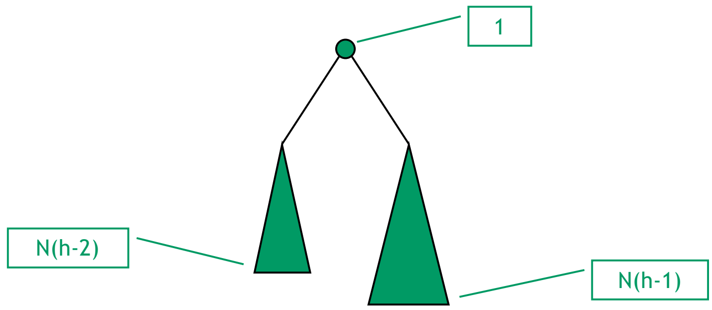
Minimale Knotenanzahl von AVL-Bäumen

- N(h) sei die minimale Anzahl von Knoten eines AVL-Baumes der Höhe h

Höhe	mögliche AVL-Bäume dieser Höhe	Knotenzahl
h = 1		N(1) = 1
h = 2		N(2) = 2
h = 3		N(3) = 4

- Allgemeiner worst case Fall bei Höhe h:

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



Satz: $N(h) = F_{h+2} - 1$

Beweis:

1) Induktionsanfang: h = 1

$$F_{1+2} - 1 = F_3 - 1 = 2 - 1 = 1$$

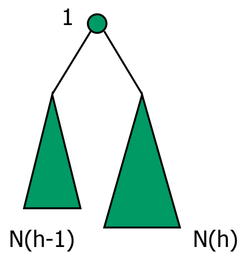
2) Induktionsschritt: $h \to h+1$

$$N(h+1) = 1 + N(h) + N(h-1)$$

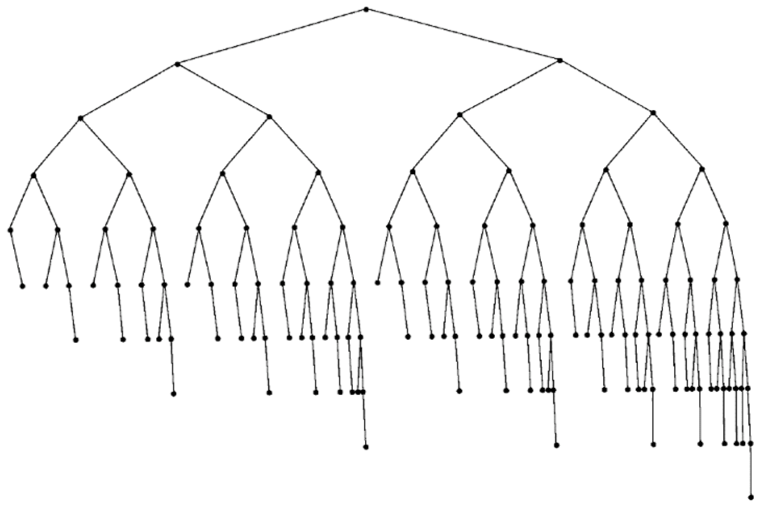
$$= 1 + F_{h+2} - 1 + F_{h+1} - 1$$

$$= F_{h+3} - 1$$

$$= F_{[h+1]+2} - 1$$



Minimaler AVL-Baum der Höhe 10



Maximale Höhe von AVL-Bäumen

- Erinnerung: Fibonacci-Zahlen

$$F_n \approx \frac{1}{\sqrt{5}} \Phi^n$$

$$\Phi = \frac{1+\sqrt{5}}{2} \approx 1.61803398875\ldots$$

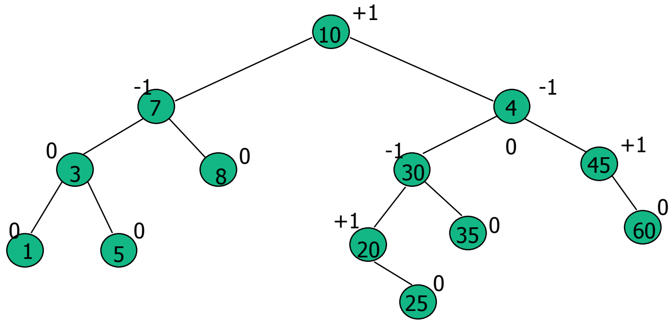
- Aus $N(h) = F_{h+2} - 1$ folgt nach Umformung und Abschätzung von F_n die …
- Wichtige Eigenschaft von AVL-Bäumen:

  Ein AVL-Baum mit N Knoten hat höchstens die Höhe

  $h \leq 1.44\ldots * \log(N) + \text{const}$
- Erinnerung: Die Höhe jedes binären Baumes mit N Knoten beträgt mindestens $\log(N+1)$

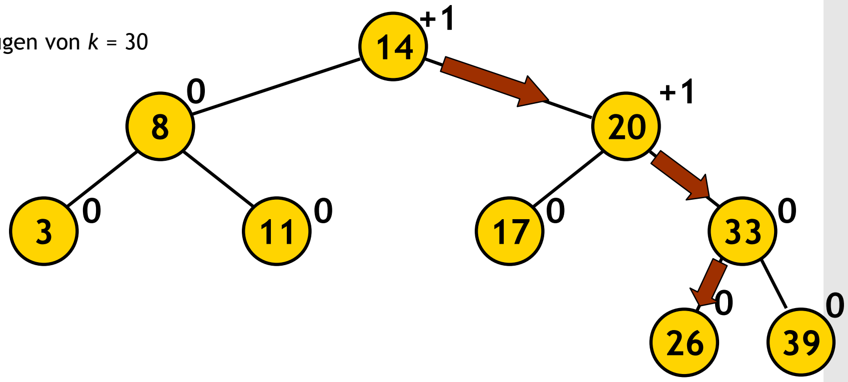
AVL Search Tree

- Problem: wir wollen BST, der auch über viele Insert- und Delete-Operationen halbwegs gut balanciert bleibt
- Idee: verwende BST, der zusätzlich AVL-Eigenschaften hat
- Problem: wie erhält man AVL-Eigenschaften bei Einfügen/Löschen?



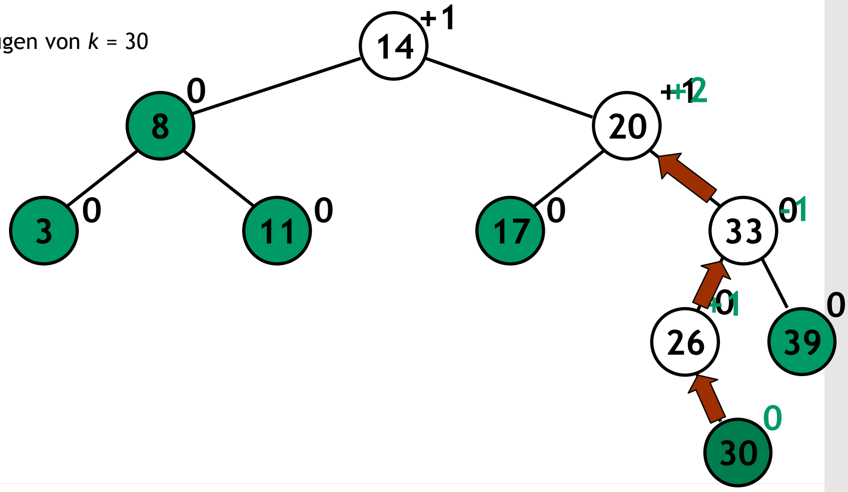
Einfügen von Knoten

Einfügen von k = 30



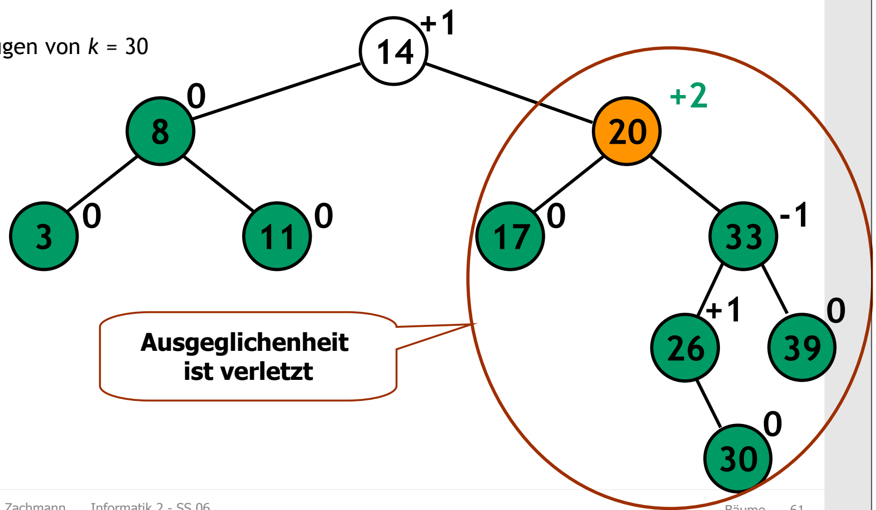
Einfügen von Knoten

Einfügen von k = 30



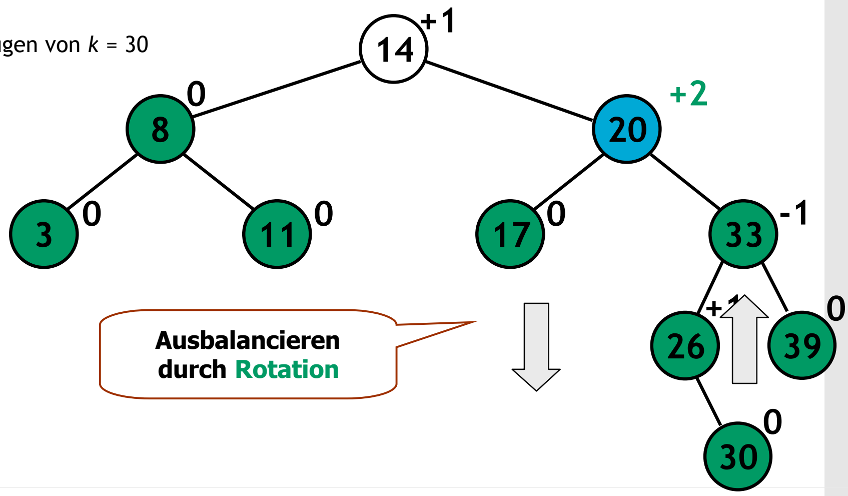
Einfügen von Knoten

Einfügen von k = 30



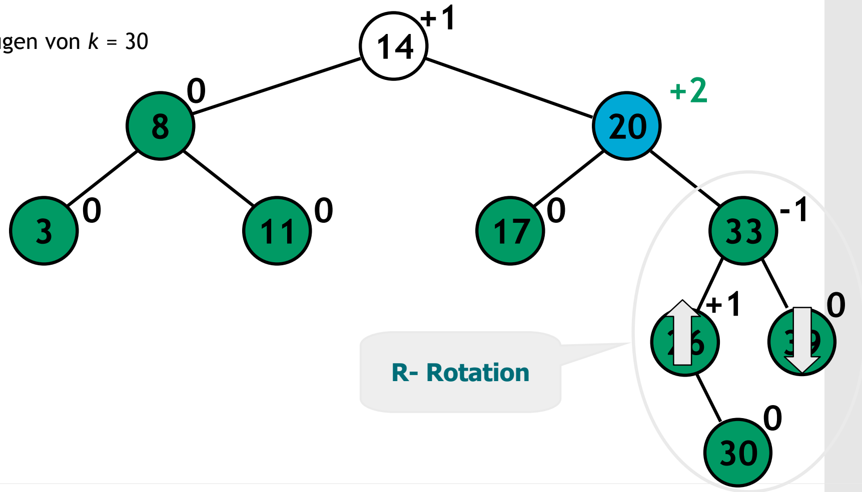
Einfügen von Knoten

Einfügen von k = 30



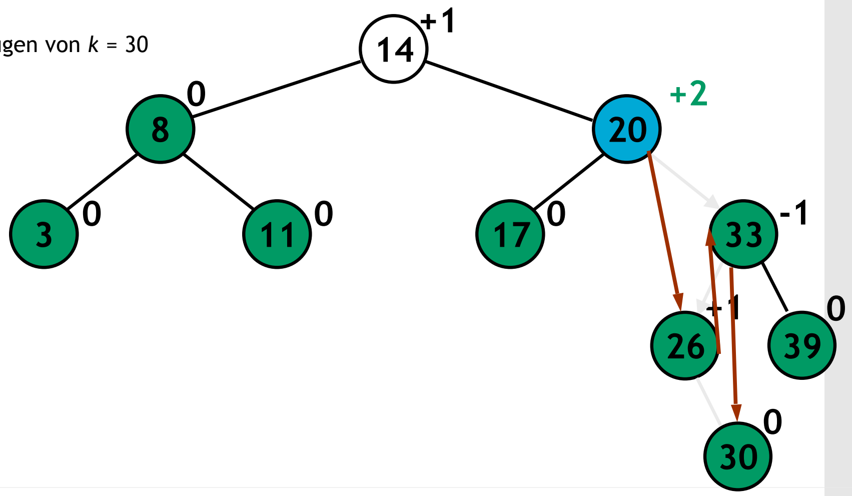
Einfügen von Knoten

Einfügen von k = 30



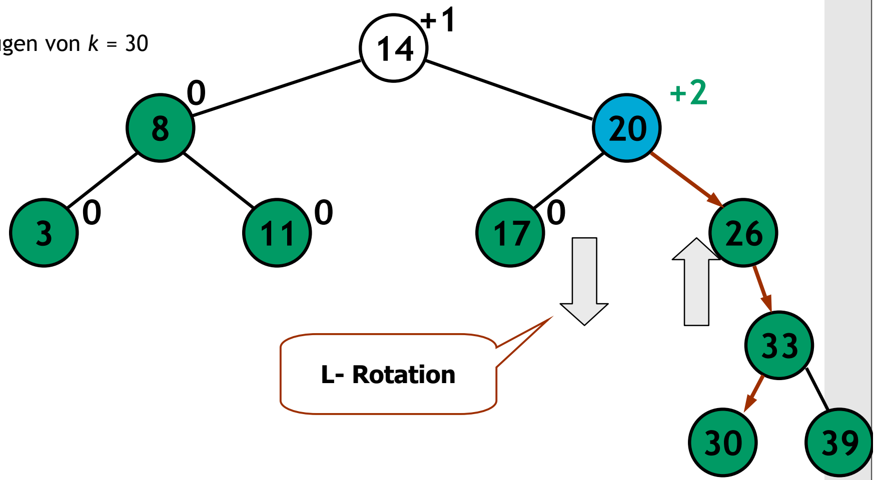
Einfügen von Knoten

Einfügen von k = 30



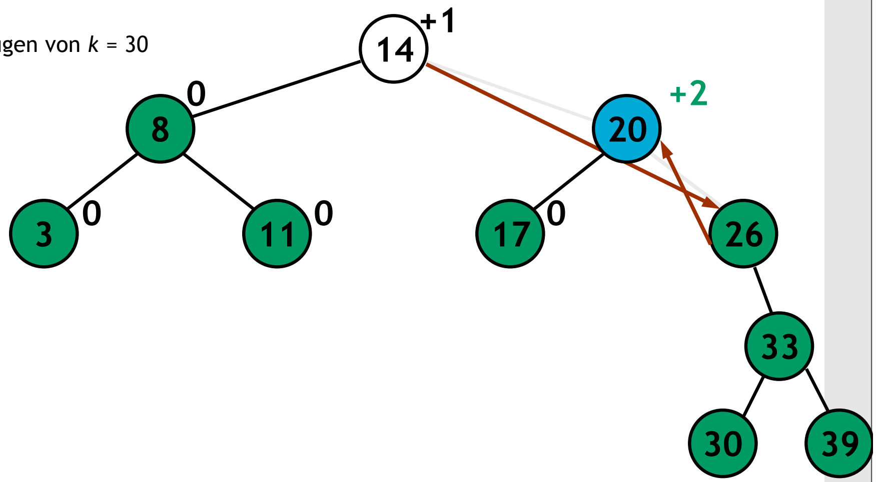
Einfügen von Knoten

Einfügen von k = 30



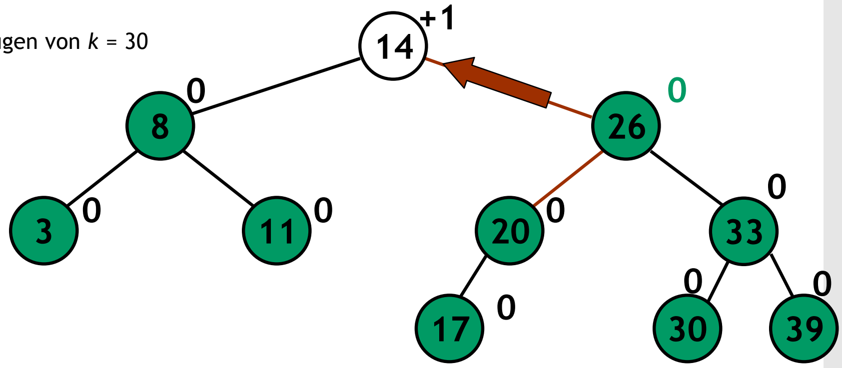
Einfügen von Knoten

Einfügen von k = 30



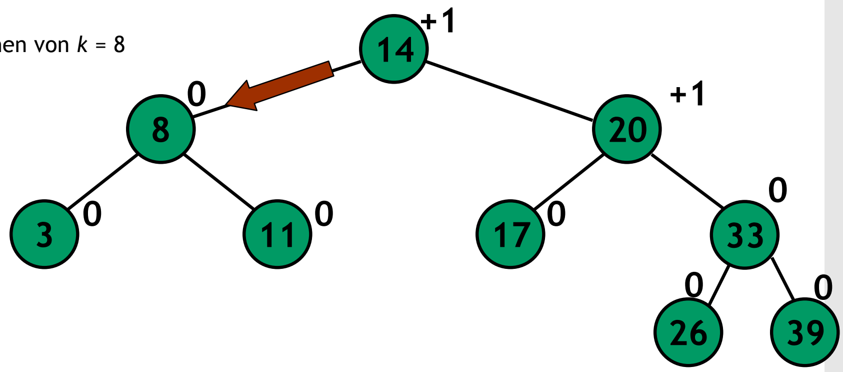
Einfügen von Knoten

Einfügen von k = 30



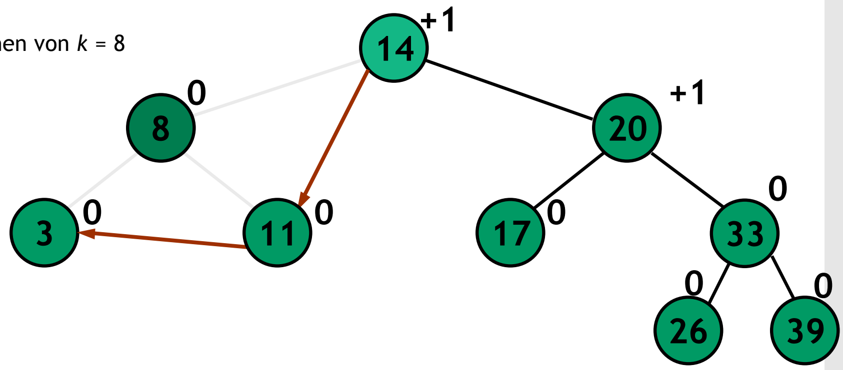
Löschen von Knoten

löschen von k = 8



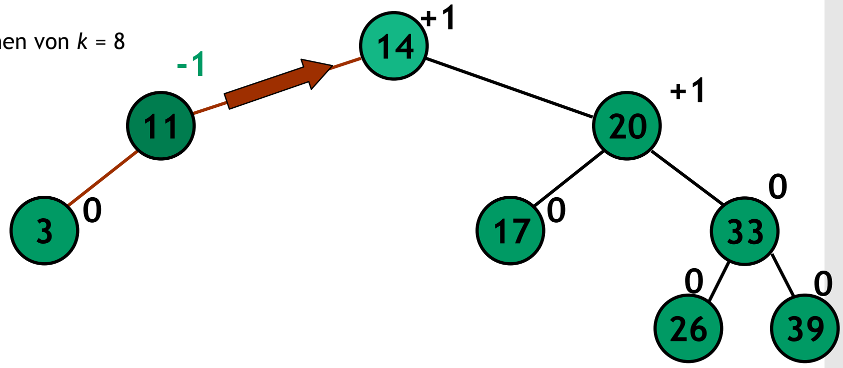
Löschen von Knoten

löschen von k = 8



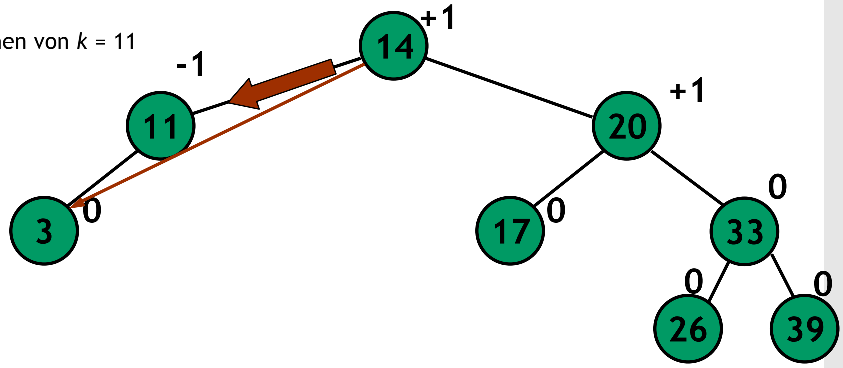
Löschen von Knoten

löschen von k = 8



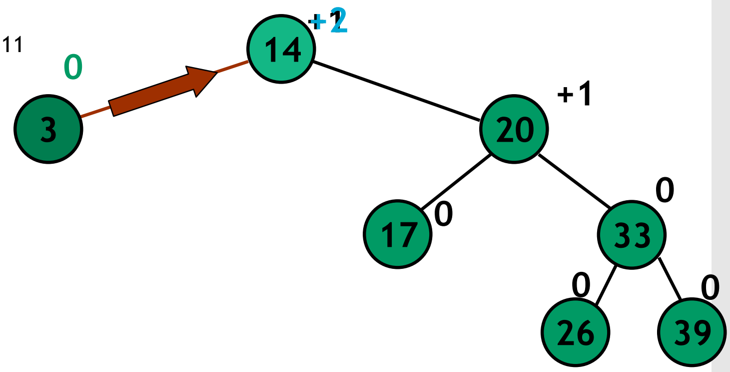
Löschen von Knoten

löschen von k = 11



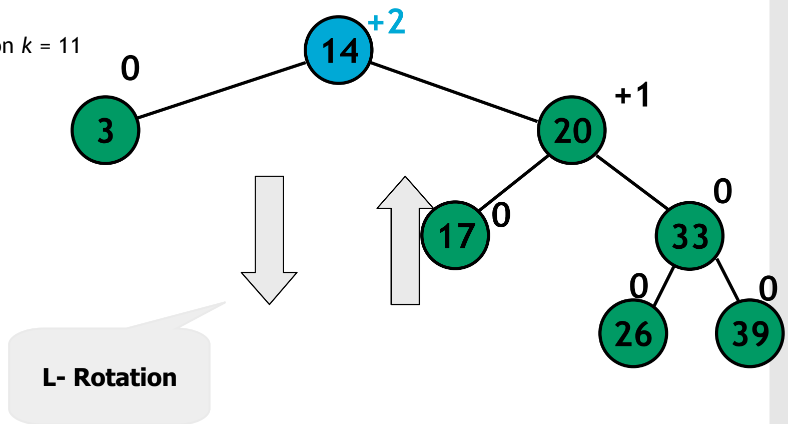
Löschen von Knoten

löschen von k = 11



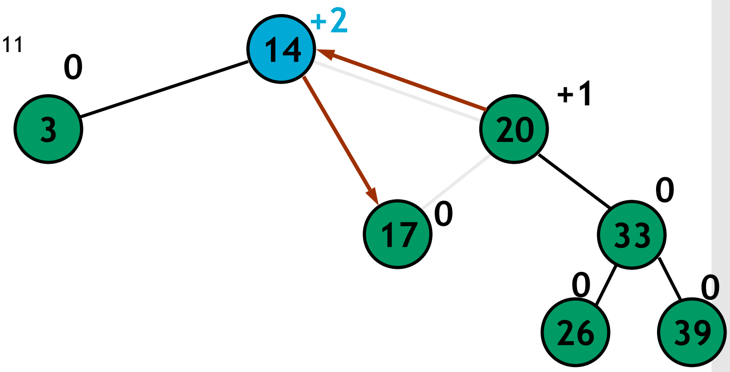
Löschen von Knoten

löschen von k = 11



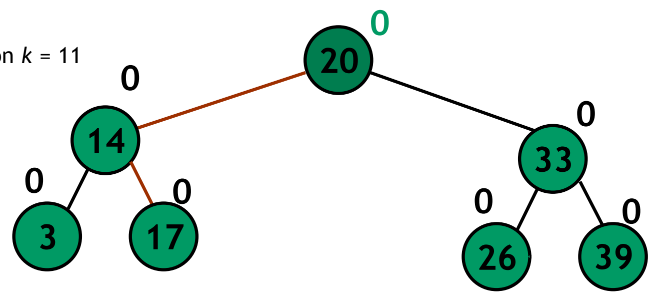
Löschen von Knoten

löschen von k = 11



Löschen von Knoten

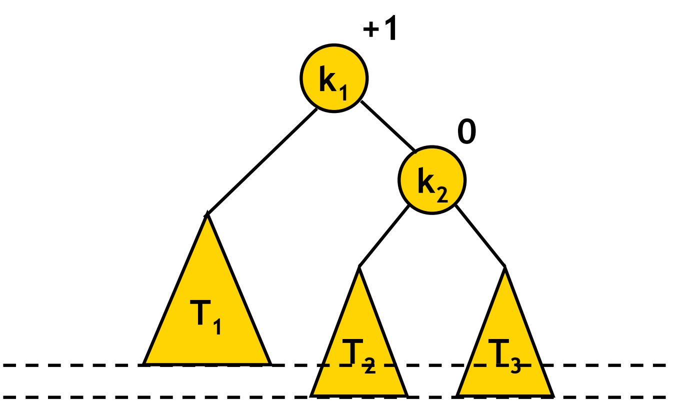
löschen von k = 11



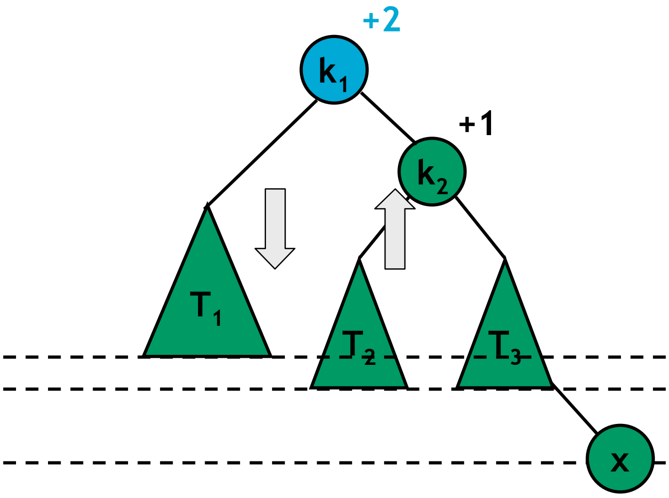
AVL-Rotationen

- Operationen auf AVL-Bäumen zur Erhaltung der AVL-Eigenschaft
- Bestehen ausschließlich aus "Umhängen" von Zeigern
- Es gibt 2 verschiedene Arten von Rotationen
  - *Single Rotation*: RR und LL
    - RR = der neue Knoten befindet sich im rechten Teilbaum des rechten Teilbaums vom (jetzt) unbalancierten Knoten aus
    - LL = analog
    - wird manchmal auch einfach nur R- bzw. L-Rotation genannt
  - *Double Rotation*:
    - RL = neuer Knoten im linken Unterbaum des rechten Unterbaumes
    - m.a.W.: vom Knoten mit dem "schlechten" Balancefaktor muß man in den rechten Teilbaum gehen, dann von da aus in den linken Teilbaum, dann kommt man zu dem neu eingefügten Knoten
    - LR = analog

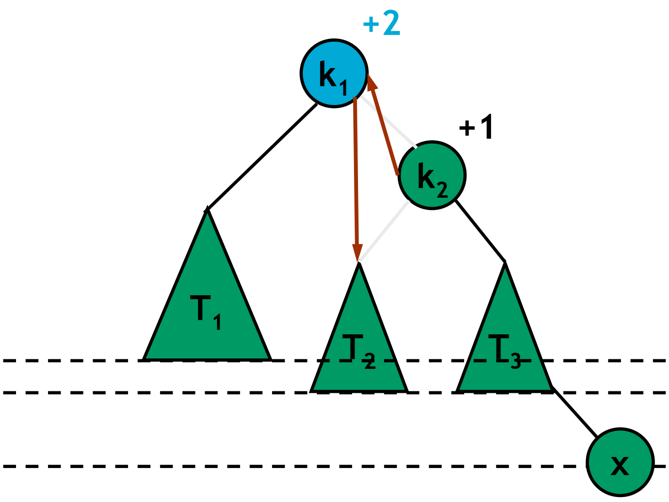
RR-Rotation



RR-Rotation

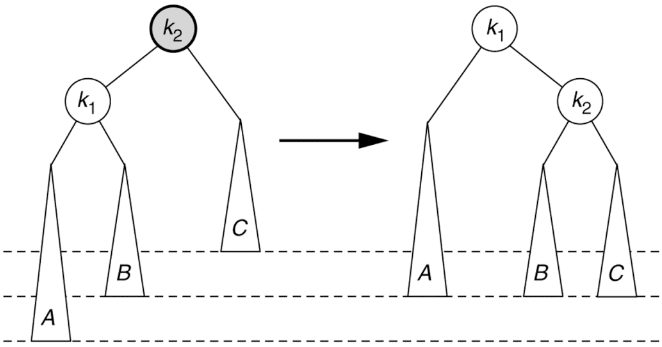


RR-Rotation



LL Rotation Algorithm

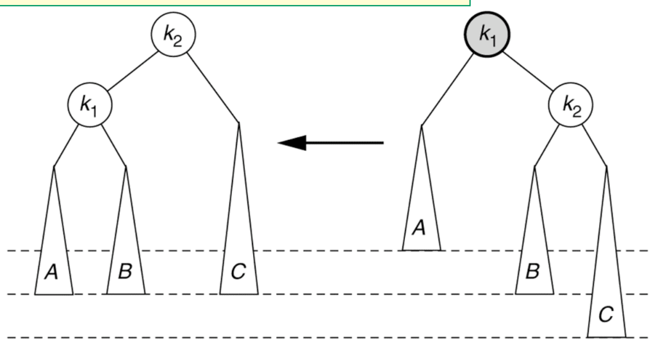
```
def LL_Rotate (k2):
  k1 = k2.left
  k2.left = k1.right
  k1.right = k2
  return k1
```



(a) Before rotation (b) After rotation

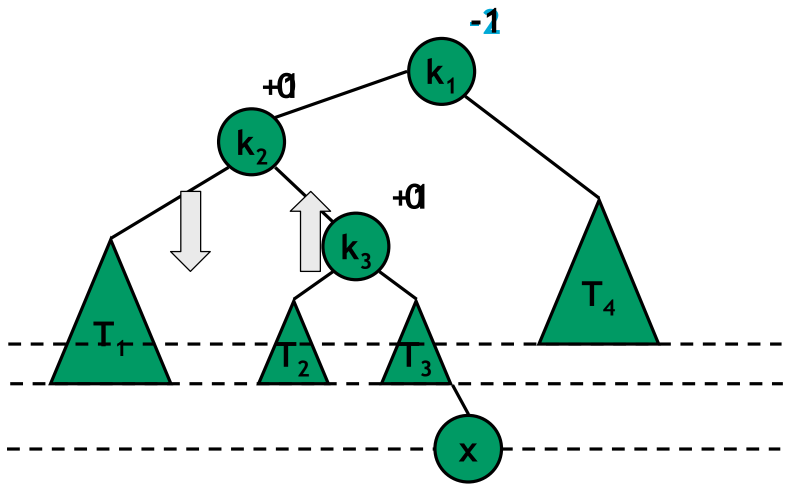
RR Rotation Algorithm

```
def RR_Rotate (k1):
k2 = k1.left
k1.right = k2.left
k2.left = k2
return k2
```

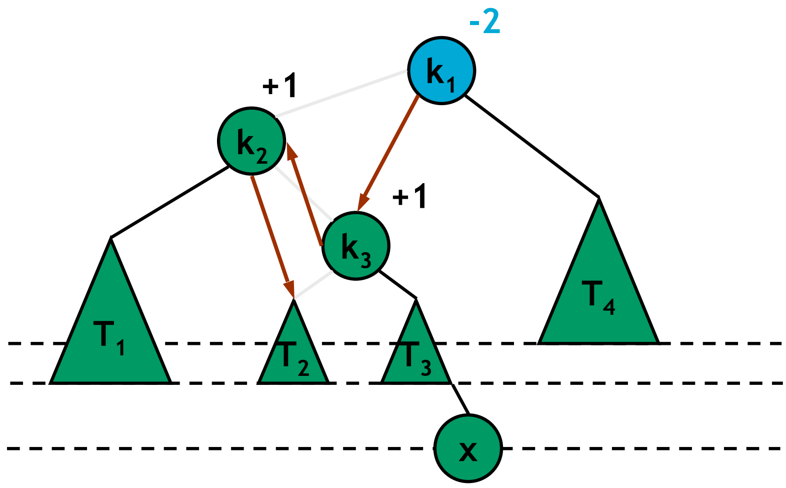


(a) After rotation (b) Before rotation

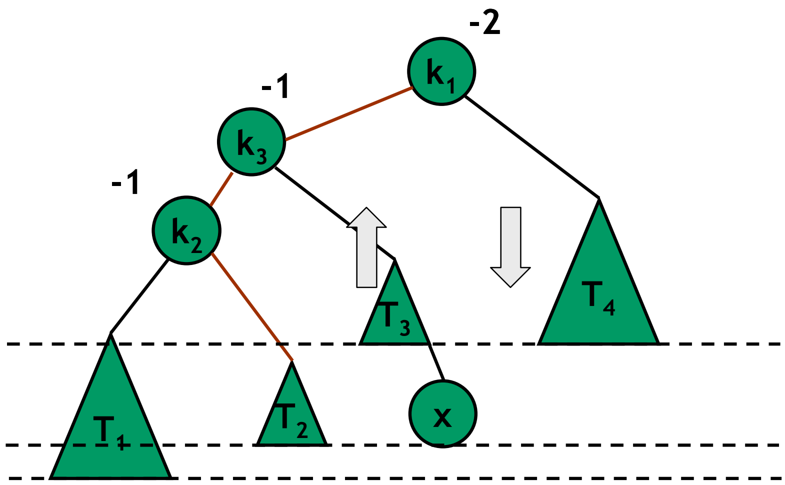
LR-Rotation



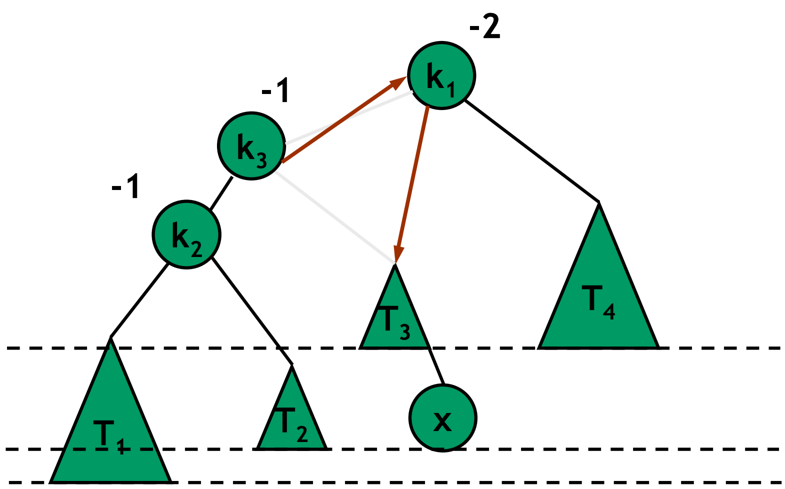
LR-Rotation



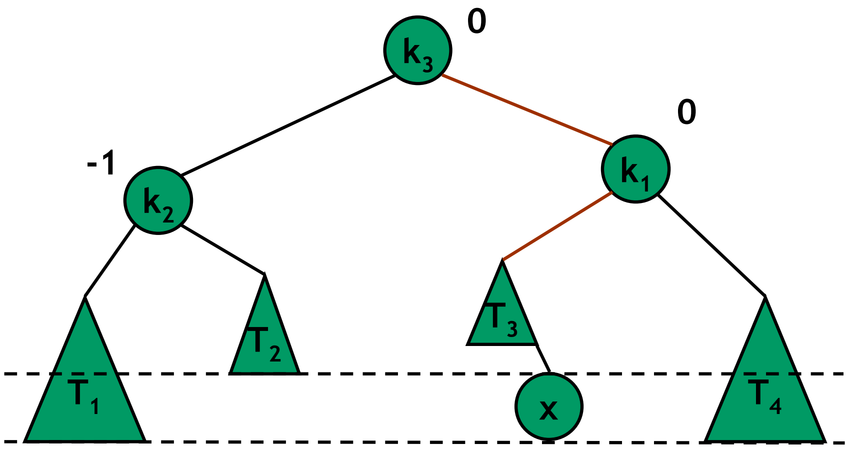
LR-Rotation



LR-Rotation

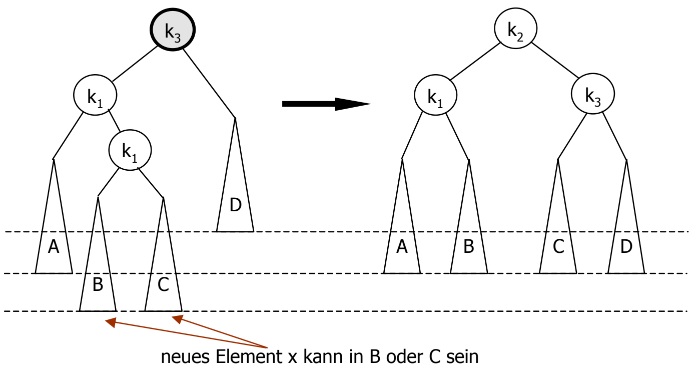


LR-Rotation



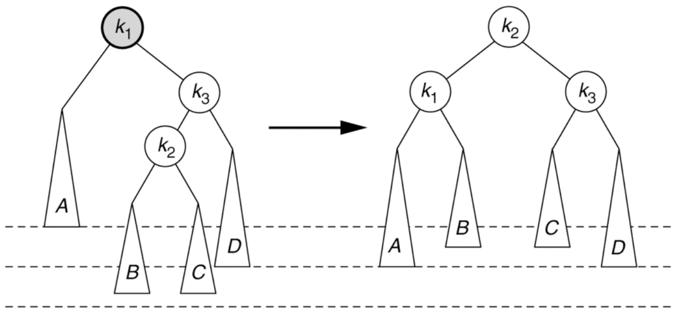
Code

```
def LR_doubleRotate ( k3 ):
  k3.left = RR_Rotate ( k3.left )
  return LL_Rotate ( k3 )
```



(a) Before rotation (b) After rotation

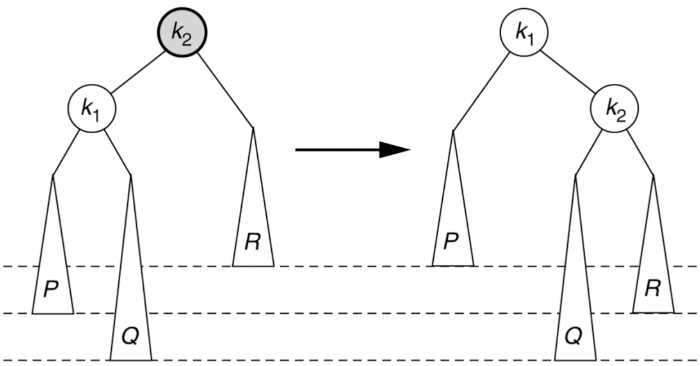
```
def RL_doubleRotate( k1 ):
  k1.right = LL_Rotate( k1.right )
return RR_Rotate( k1 )
```



(a) Before rotation (b) After rotation

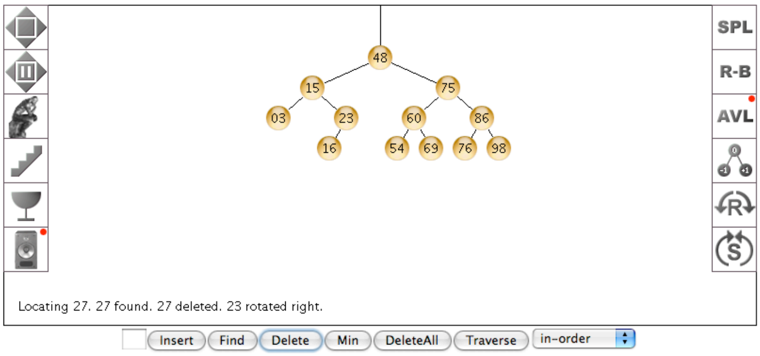
Warum Double Rotation?

- Single Rotation kann LR oder RL nicht lösen:



(a) Before rotation (b) After rotation

Algo-Animation



http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm