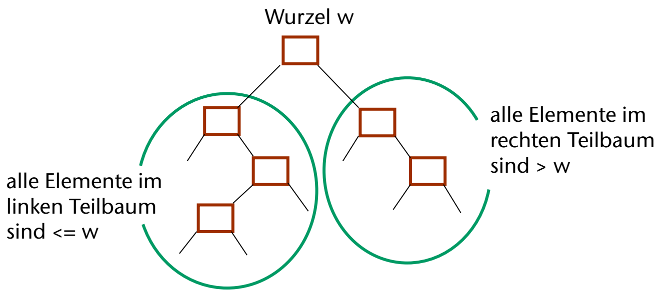
Informatik II
Bäume zum effizienten Information Retrieval

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

Binäre Suchbäume (binary search tree, BST)

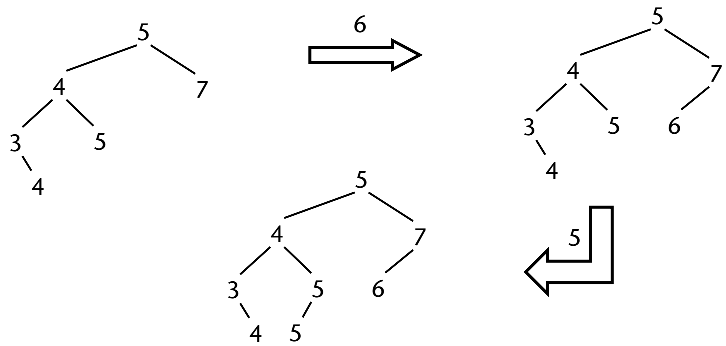
- Speichere wieder Daten als "Schlüssel + Nutzdaten"
- Sei eine Ordnungsrelation für die Schlüssel im Baum definiert
- Ziel: Binärbäume zur Speicherung von Mengen von Schlüsseln, so daß folgende Operationen effizient sind:
  - Suchen (find)
  - Einfügen (insert)
  - Entfernen (remove, delete)

- Definition:
  Ein binärer Baum mit Suchbaumeigenschaft ist von folgender Form

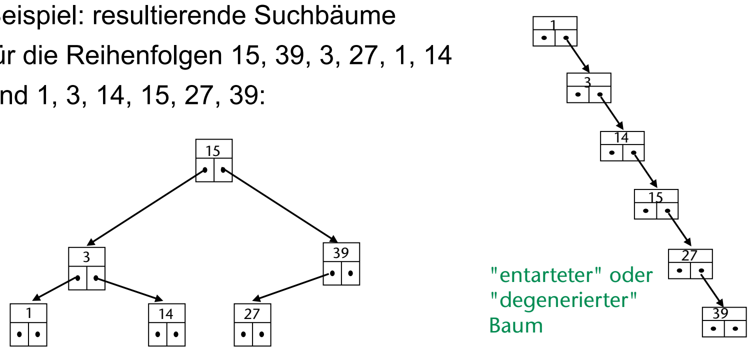


Einfügen

- Hinzufügen von x zum Baum B
  - wenn B leer ist, erzeuge Wurzel mit x
  - wenn x <= Wurzel, füge x rekursiv zum linken Teilbaum hinzu
  - wenn x > Wurzel, füge x rekursiv dem rechten Teilbaum hinzu
- Beispiel:

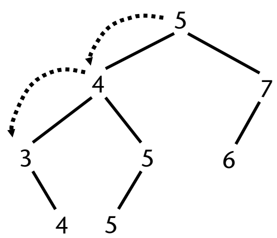


- Achtung:
  - Baum-Struktur hängt von Einfügereihenfolge im anfangs leeren Baum ab!
  - Dito für Höhe: Höhe kann, je nach Reihenfolge, zwischen n und $\lceil \log_2(n+1) \rceil$ liegen
- Beispiel: resultierende Suchbäume für die Reihenfolgen 15, 39, 3, 27, 1, 14 und 1, 3, 14, 15, 27, 39:



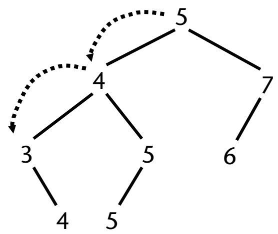
Suchen

- Aufgabe: Key x im BST B suchen:
  - wenn B leer ist, dann ist x nicht in B
  - wenn x = Wurzelelement gilt, haben wir x gefunden
  - wenn x < Wurzelelement, suche im linken Teilbaum
  - wenn x > Wurzelelement, suche im rechten Teilbaum
- Beispiel: suche 3 im Baum



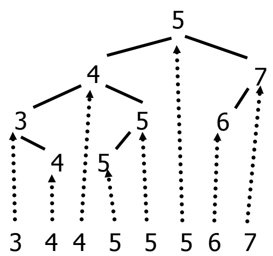
- Bemerkung: gibt es mehrere Knoten mit gleichem Wert , wird offenbar derjenige mit der geringsten Tiefe gefunden

- Aufgabe: suche kleinstes Element
  - Folge dem linken Teilbaum, bis Knoten gefunden wurde, dessen linker Teilbaum leer ist



- Analog: größtes Element finden

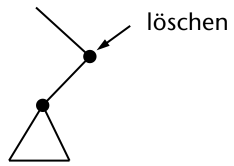
- Aufgabe: Nachfolger in Sortierreihenfolge suchen
  - Wenn der Knoten einen rechten Teilbaum hat, ist der Nachfolger das kleinste Element des rechten Teilbaumes
  - Ansonsten: Aufsteigen im Baum, bis ein Element gefunden wurde, das größer oder gleich dem aktuellen Knoten ist
    - Falls man die Wurzel erreicht hat, gibt es kein solches Element
  - Dazu sollte der Knoten eine Referenz auf seinen Vater beinhalten
- Beispiel:



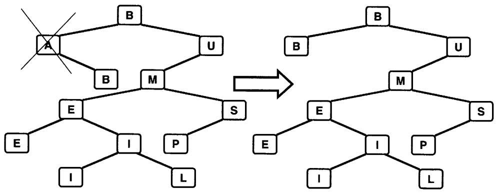
Löschen

- Ist die aufwendigste Operation
  - Auch hier muß sichergestellt werden, daß der verbleibende Baum ein gültiger binärer Suchbaum ist

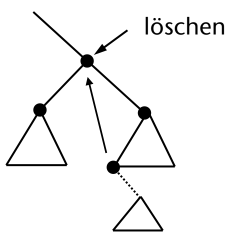
1. Fall: Knoten hat keinen Teilbaum (Blatt)
   - Knoten einfach löschen
2. Fall: nur ein Teilbaum
   - Teilbaum eine Etage höher schieben



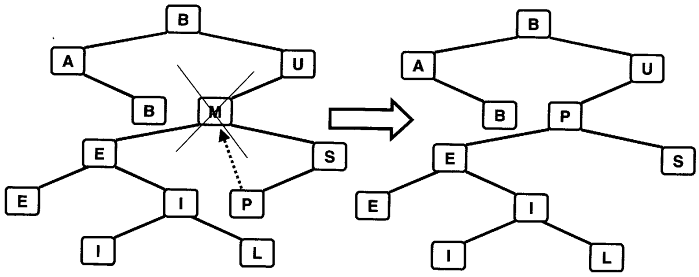
- Beispiel:



3. Fall: Knoten hat zwei Teilbäume
   - Suche kleinstes Element des rechten Teilbaumes
     - Dieses kleinste Element hat höchstens einen rechten Teilbaum
   - Kopiere Inhalt (Schlüssel + Daten) dieses kleinsten Knotens in den Inhalt des zu löschenden Knotens
   - Lösche den Knoten, der vorher dieses kleinste Element beinhaltete → Fall 2

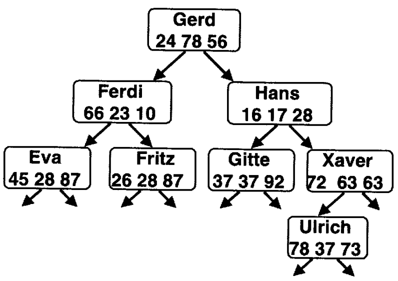


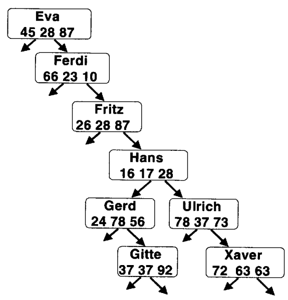
- Beispiel:



Balancierte Bäume

- Aufwand, ein Element zu finden, entspricht der Tiefe des gefundenen Knotens
  - Im worst case = Tiefe des Baumes
  - Diese liegt zwischen $\lfloor \log N \rfloor + 1$ und N





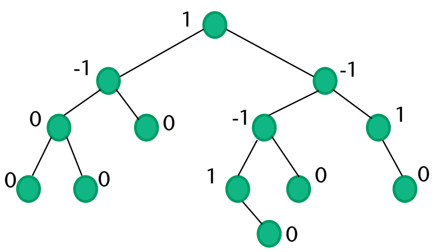
- Definition für "balanciert":
  - Es gibt verschiedene Definitionen!
  - Allgemein: kein Blatt ist “wesentlich weiter“ von der Wurzel entfernt als irgendein anderes
  - Hier: Für alle Knoten unterscheidet sich Anzahl der Knoten in linkem und rechtem Teilbaum höchstens um 1
  - Folge: ein balancierter BST hat die Tiefe $\lfloor \log N \rfloor + 1$
- Schlecht balancierte Bäume erhält man, wenn die Elemente in sortierter Reihenfolge angeliefert werden
- Der 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 eines Knotens x:

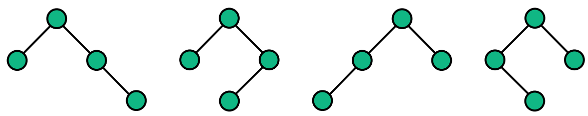
  bal(x) = (Höhe des rechten Unterbaumes von x) -
  (Höhe des linken Unterbaumes von x)
- Definition AVL-Baum:

  Ein binärer Baum, wobei
  für jeden Knoten x gilt:
  $bal(x) \in \{-1,0,1\}$



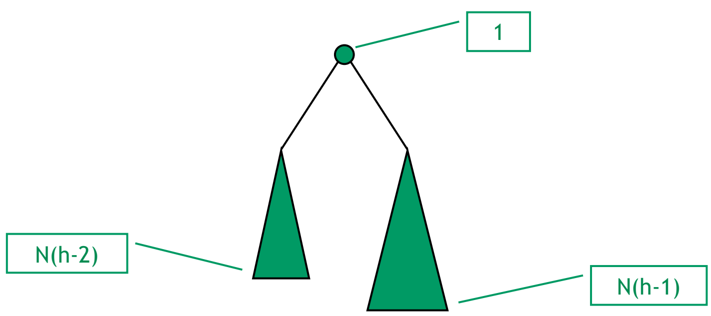
Minimale Knotenanzahl von AVL-Bäumen

- Bezeichne $N(h)$ 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:

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

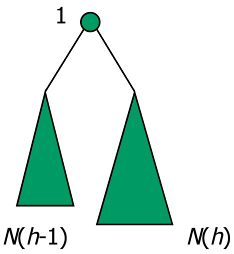
Beweis:

1. Induktionsanfang: $h = 1$

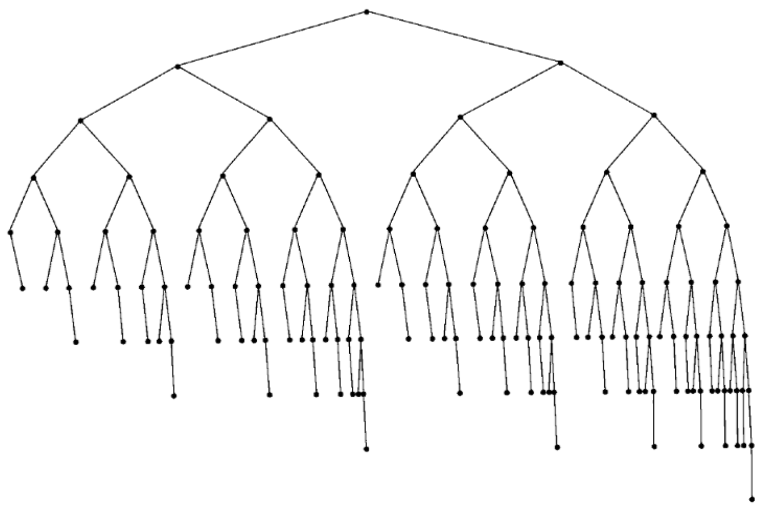
   $$F_{1+2} - 1 = F_3 - 1 = 2 - 1 = 1$$

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

$$N(h+1) = 1 + N(h) + N(h-1)$$
$$= 1 + F_{h+2} - 1 + F_{h+1} - 1$$
$$= F_{h+3} - 1$$
$$= F_{[h+1]+2} - 1$$



Beispiel: ein minimaler AVL-Baum der Höhe 10



Die maximale Höhe von AVL-Bäumen

- Erinnerung bzgl. Fibonacci-Zahlen :

$$F_n \approx \frac{1}{\sqrt{5}} \Phi^n \quad , \quad \Phi = \frac{1+\sqrt{5}}{2} \approx 1.61803398875\ldots$$

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

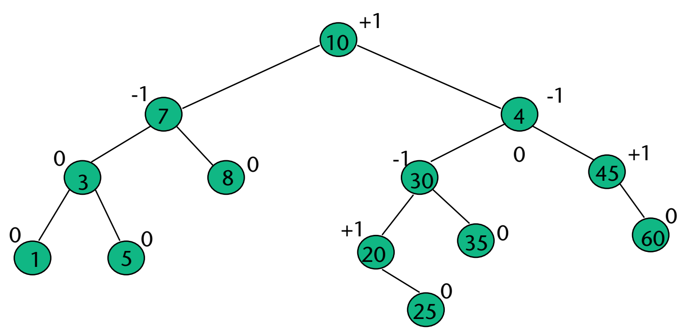
  Ein AVL-Baum mit N Knoten hat höchstens die Höhe

  $h \leq 1.44\ldots * \log(N) + \text{const}$

- Erinnerung: Die Höhe jedes binären Baumes mit N Knoten beträgt mindestens $\log(N+1)$

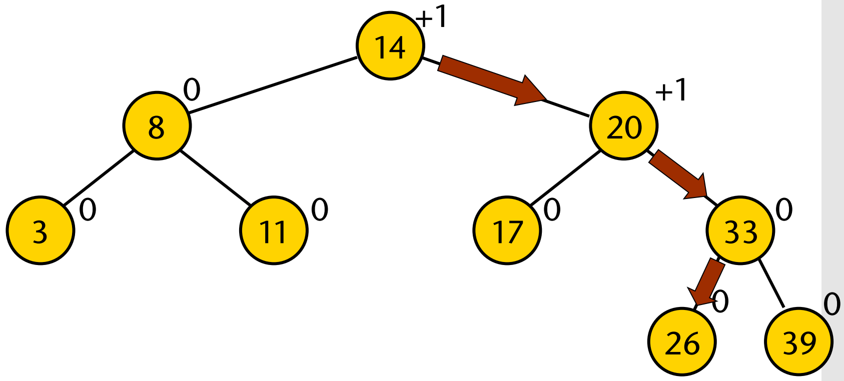
Der AVL Search Tree

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



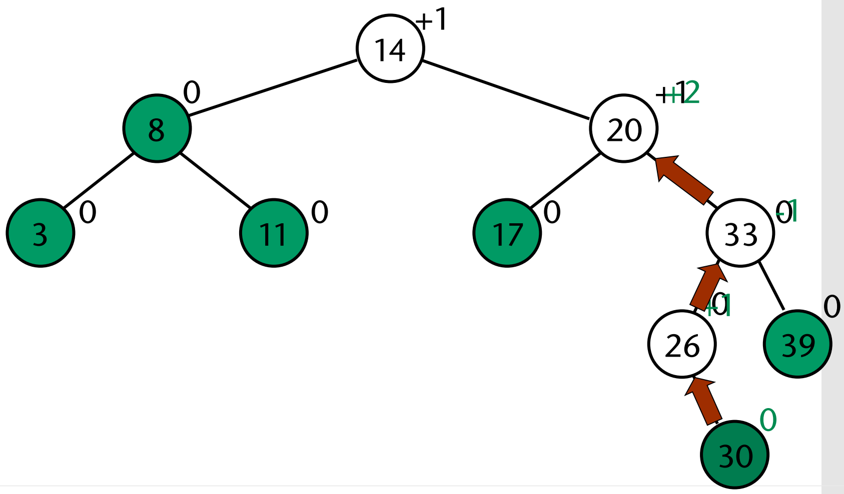
Einfügen von Knoten

Beispiel: Einfügen von $k = 30$

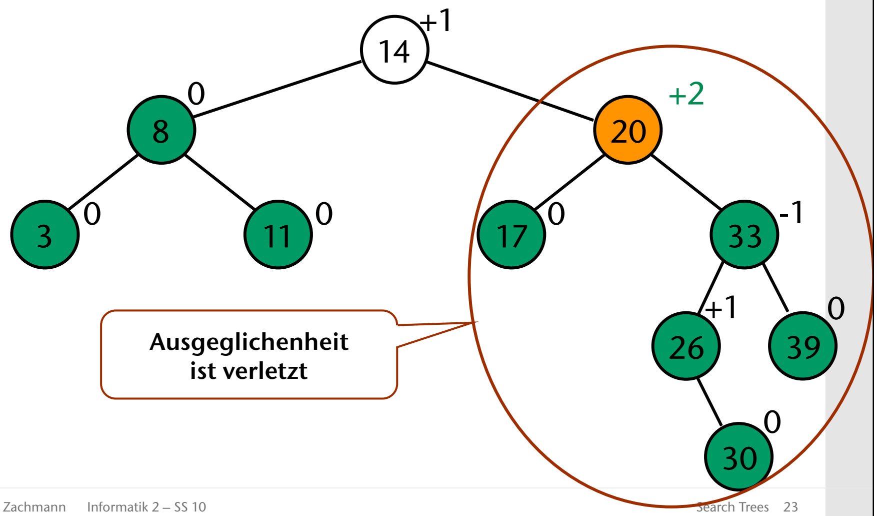


Einfügen von Knoten

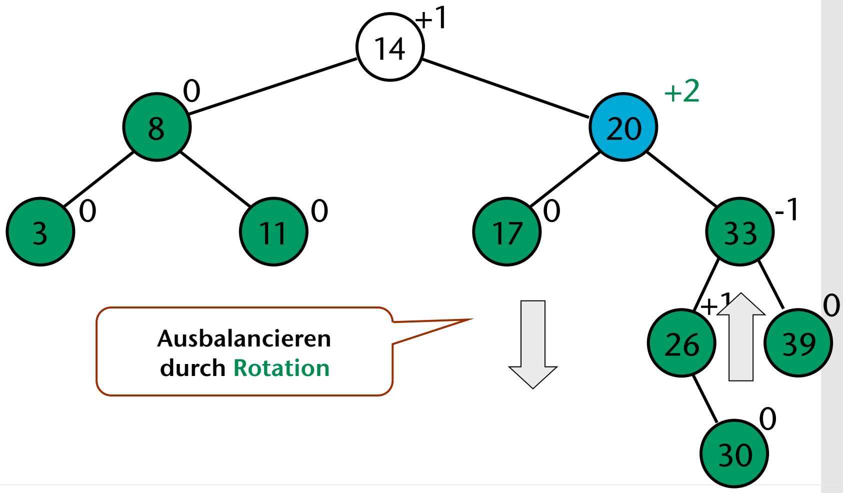
Beispiel: Einfügen von $k = 30$



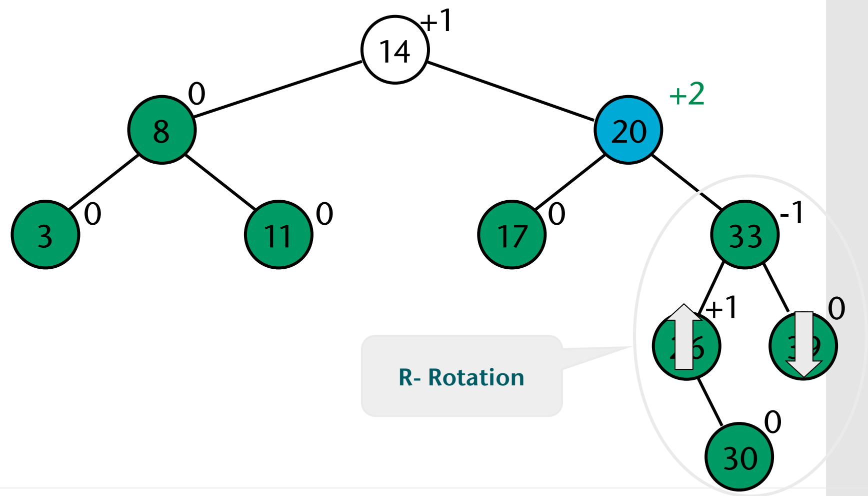
Beispiel: Einfügen von k = 30



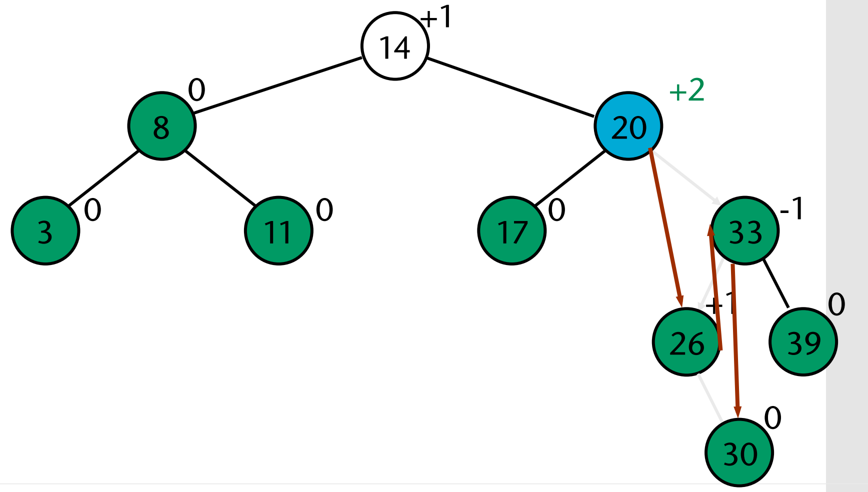
Beispiel: Einfügen von k = 30



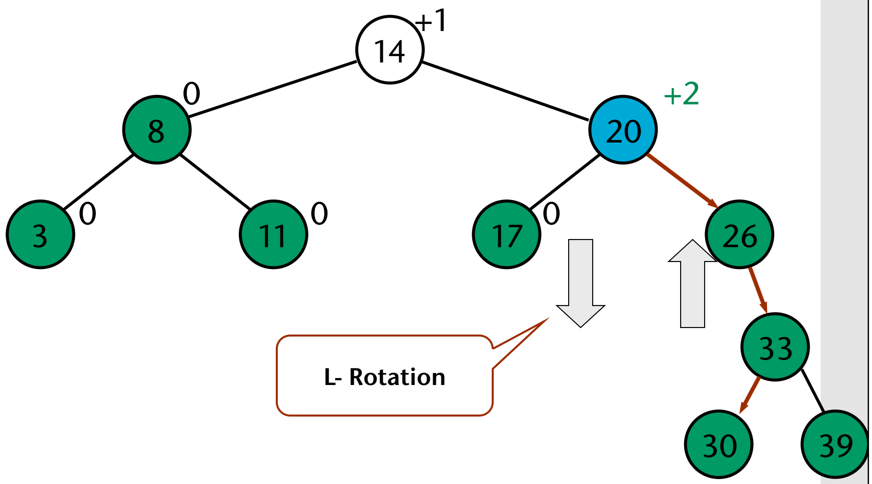
Beispiel: Einfügen von k = 30



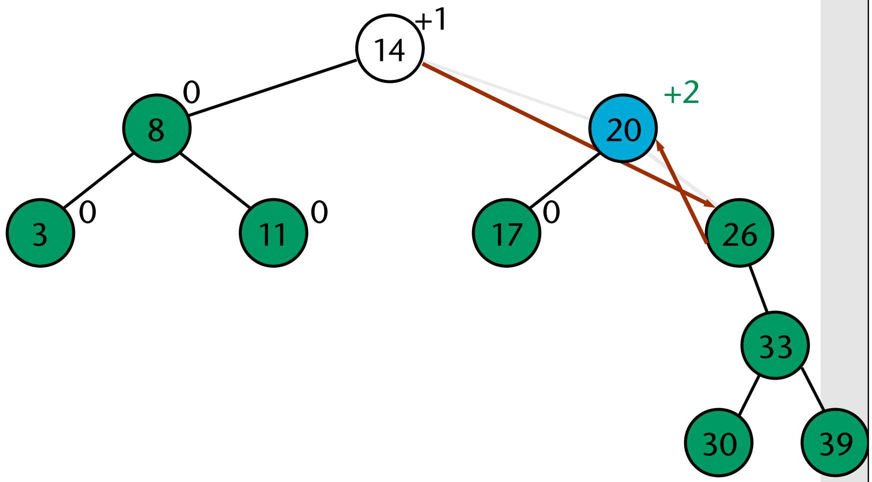
Beispiel: Einfügen von k = 30



Beispiel: Einfügen von $k = 30$



Beispiel: Einfügen von $k = 30$



Beispiel: Einfügen von k = 30