

# Suchsäule

Verwalte dynamische Daten.

Unterstütze 3 Operationen:

- suchen
- einfügen schnell!
- löschen

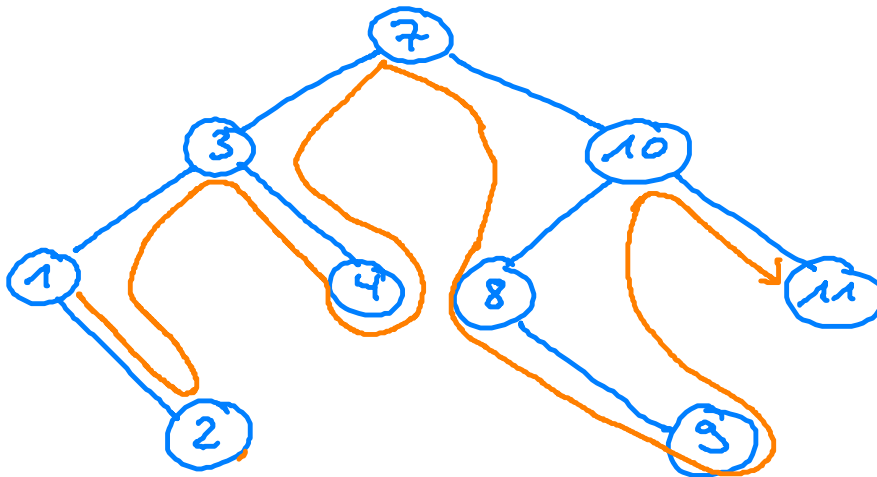
Suchsbäume unterstützen alle 3 Operationen mit Aufwand  $O(\log n)$ .

Idee:

- binäre Bäume
- Knoten eines Baumes repräsentieren Elemente mit Schlüsseln
- Für jeden Knoten  $v$  des Baumes haben alle Elemente im linken Teilbaum von  $v$  einen kleineren Schlüssel als  $v$  und die Elemente im rechten Teilbaum einen größeren Schlüssel:

Suchbaum-eigenschaft

Bsp:



äquivalent zur Suchbaum-eigenschaft:

inorder-Durchlauf des Baumes ergibt aufsteigende Sortierung der Schlüssel

Zur Erinnerung: inorder-Durchlauf bedeutet:

$\forall$  Knoten  $v$ : Besuche linken Teilbaum von  $v$

vor  $v$  vor rechten Teilbaum.

## Implementation der Operation im Suchbaum

I) Suchen: Vergleiche gesuchten Schlüssel mit Wurzel:

Falls " $=$ ", fertig

Falls " $<$ ", dann suche im linken Teilbaum

Falls " $>$ ", " " " rechten "

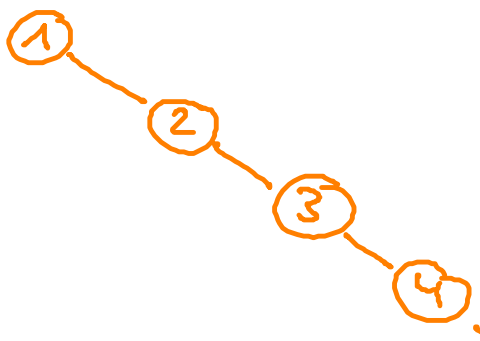
rekursiv anwenden!

Dies liefert die korrekte Antwort, sowohl im Erfolgsfall als auch im Fall, dass nichts gefunden wird.

Aufwand: Suche nach  $v$  erfordert  $h_T(v) + 1$  Vergleiche

$\Rightarrow$  im Worst Case:  $h(T) + 1$  Vergleiche

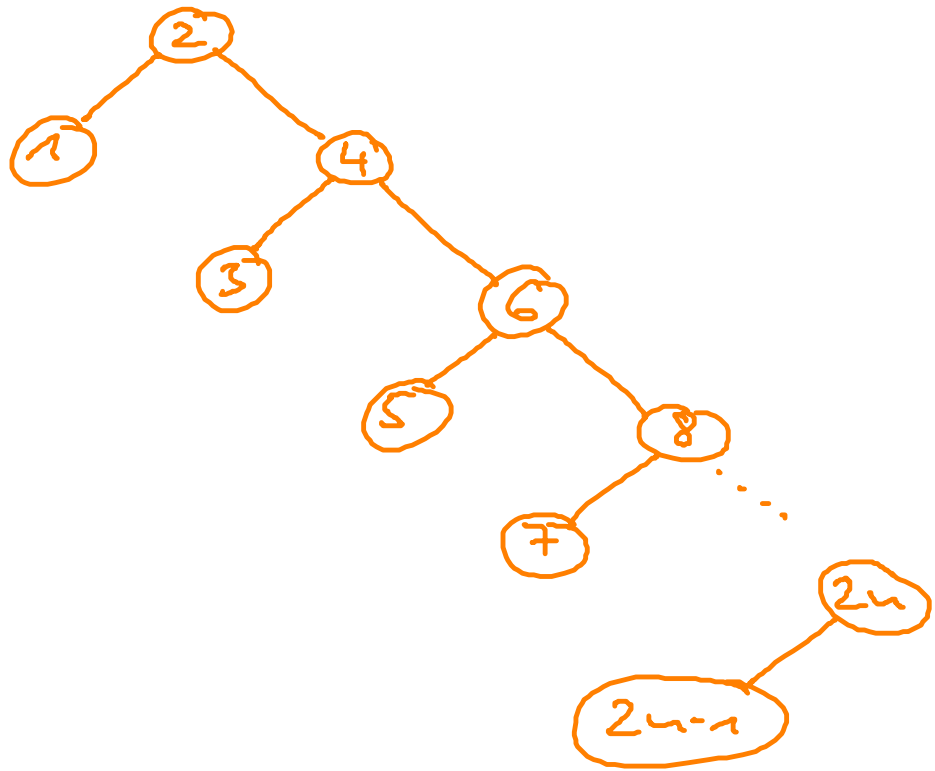
Problem: Falls  $T$  zu Liste entartet ist, benötigt man  $n$  Vergleiche:



=> wir benötigen mehr Struktur in unseren Suchbäumen, um solche Fälle zu vermeiden.

Z.B.: fordern vollständige Bäume (d.h. jeder innere Knoten hat genau zwei Kinder)

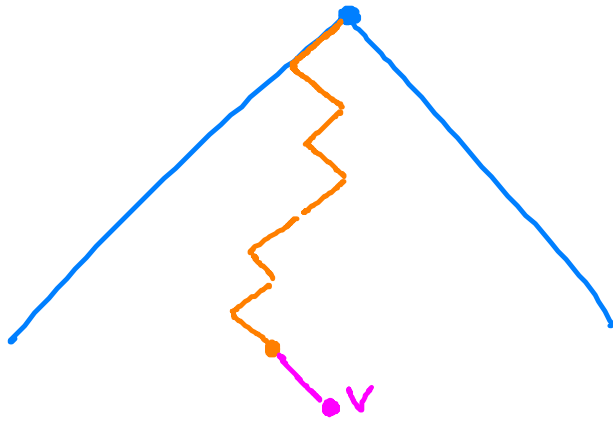
aber das reicht nicht, denn:



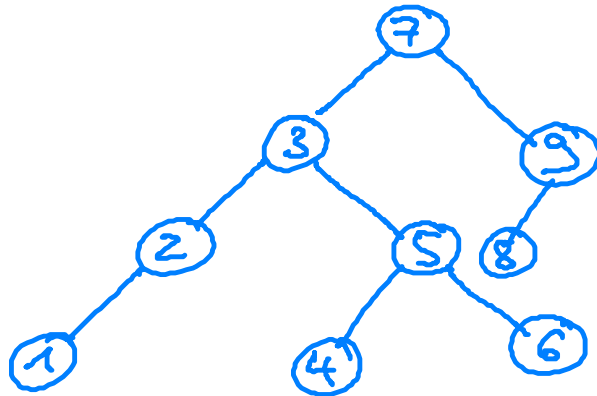
„Idee“ für

Lösung: Betrachte volle Bäume, bei denen alle Schichten außer der untersten voll besetzt sind.

II) Einfügen: neuen Knoten  $v$  gemäß Schlüssel an die richtige Stelle im Baum absinken lassen und dort als Blatt einfügen:



Beispiel: 7, 3, 5, 6, 2, 1, 9, 4, 8  
nacheinander in Suchbaum einfügen:

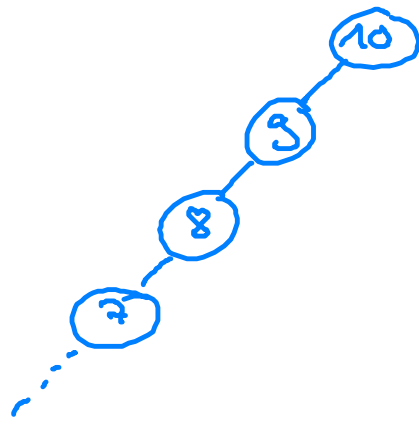


Aufwand für das Einfügen eines Elements:  
wie beim Suchen  $O(h(T))$ , also  $O(n)$   
im Worst Case.

→ brauche spezielle Bäume

Worst Case tritt auf, wenn man die  
Elemente in sortierter Reihenfolge einfügt:

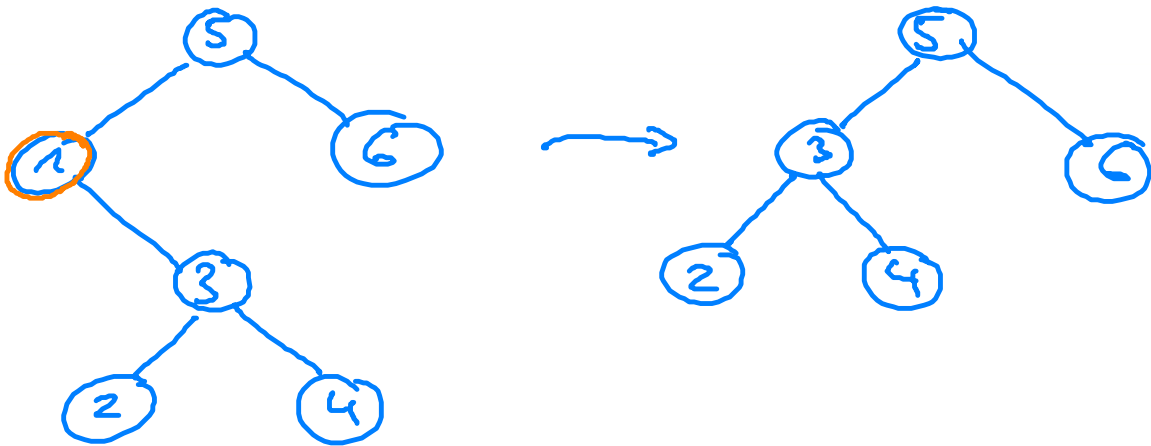
10, 9, 8, 7, ...



### III) Löschen

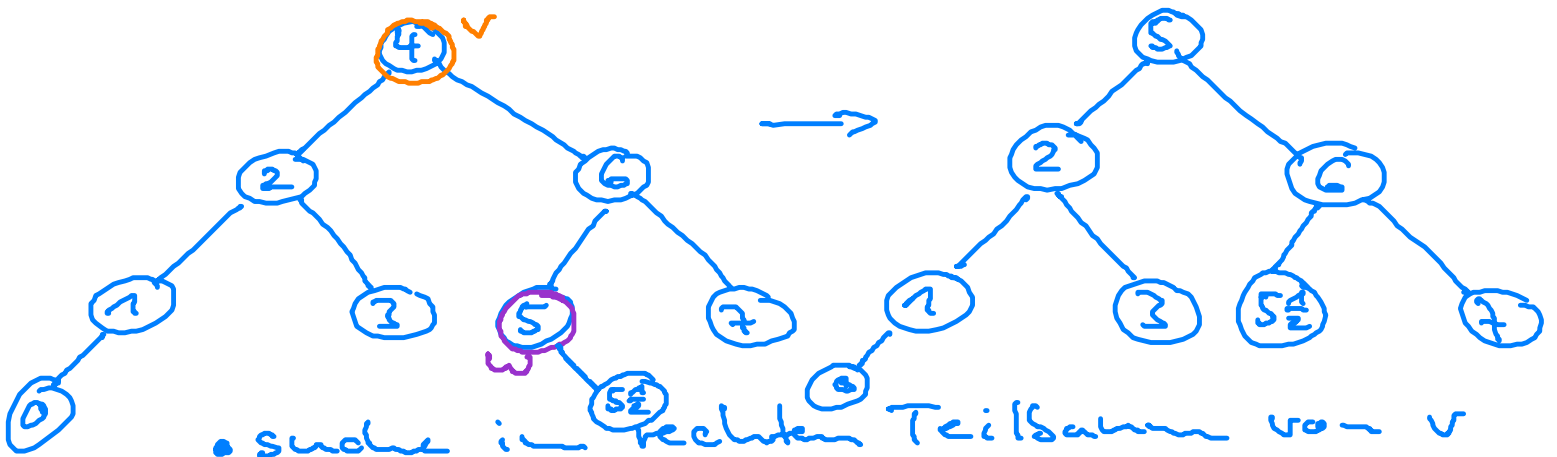
ein löschen Knoten suchen, drei Fälle unterscheiden:

- 1) Knoten ist Blatt  $\rightarrow$  einfach löschen, Suchbaum eigensch. bleibt erhalten
- 2) Knoten hat genau ein Kind:



$\rightarrow$  ersetze  $v$  durch sein Kind

- 3) Knoten  $v$  hat 2 Kinder



- suche im rechten Teilbaum von  $v$  den linkesten Knoten (der mit dem kleinsten Schlüssel im Teilbaum)
- vertausche Inhalte von  $v$  und  $w$
- lösche  $w$  (beachte:  $w$  hat kein linkes Kind)

Beachte:  $w$  ist Nachfolger von  $v$  beim in-order-Durchlauf, daher ist Suchbaumeneigenschaft weiterhin erfüllt.

Aufwand: suchen von  $v$  und  $w$ :  $O(h(T))$

Austauschen der Inhalte:  $O(1)$

Löschen von  $w$ :  $O(1)$

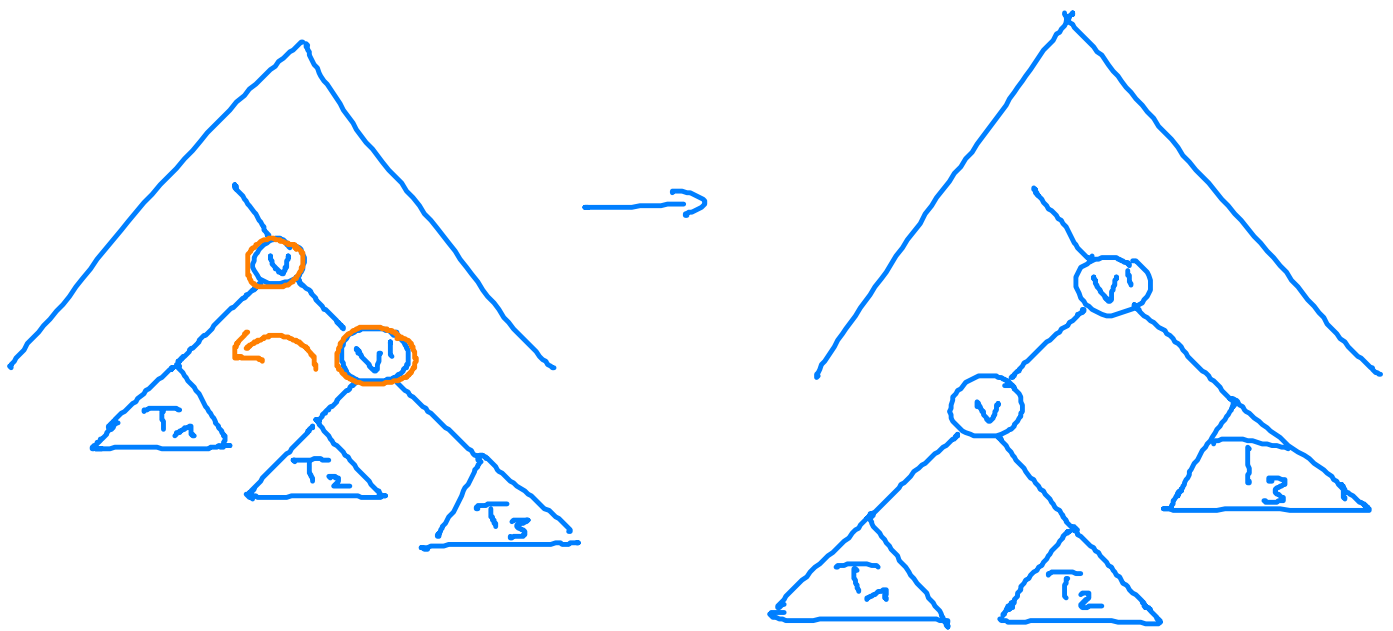
insgesamt:  $O(h(T))$

Alle 3 Operationen benötigen Aufwand  $O(h(T))$ . Versuche, Suchbäume mit spezieller Struktur zu verwenden, so dass  $h(T) \in O(\log n)$  für diese Bäume  $T$ .

Problem: Nach Einfügen und Löschen muss der Baum so modifiziert werden, dass er wieder diese Struktur besitzt.

Um dieses Problem zu lösen, verwenden wir im Folgenden zwei Operationen, die „Linksrotation“ und die „Rechtsrotation“:

Linksrotation:  $\text{RotLinks}(v, v')$



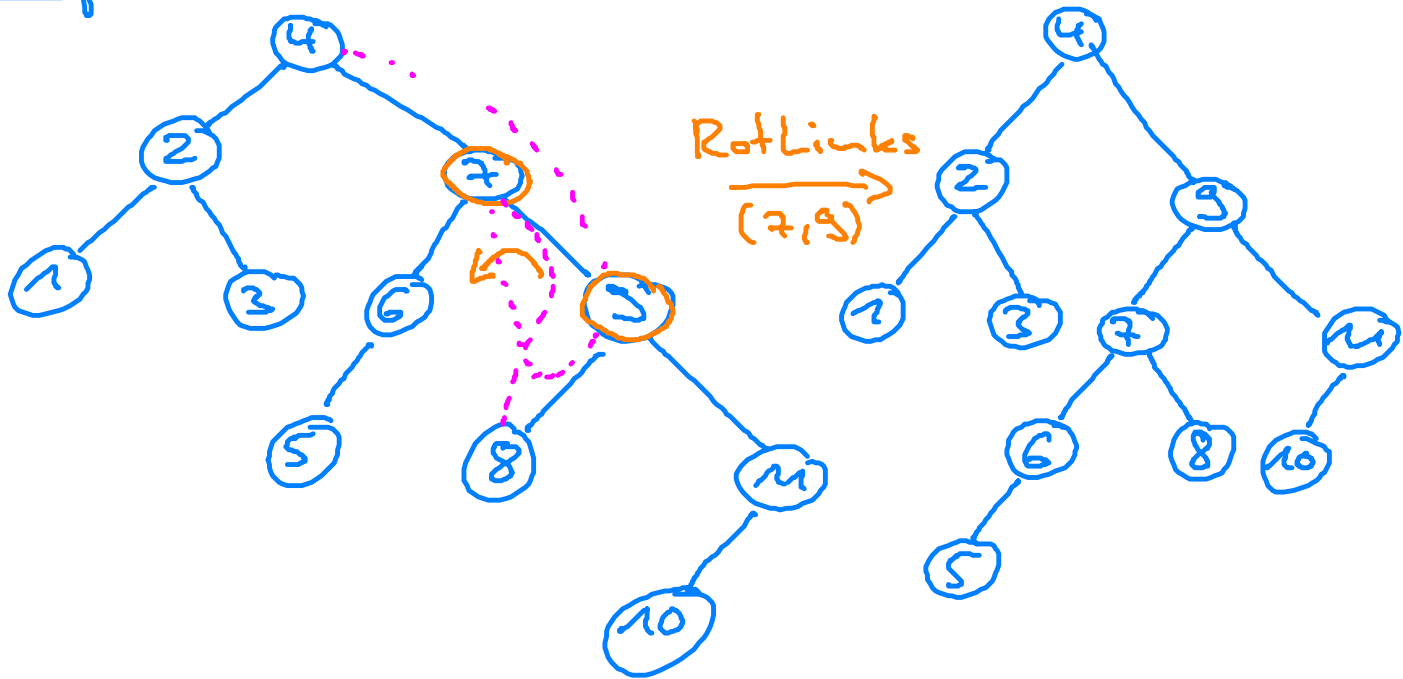
Beachte: Suchbaumeigenschaft ist weiterhin erfüllt, denn:

$$"T_1 < v < T_2 < v' < T_3"$$



Idee: Bei dieser Operation wird der linke Teilbaum höher und der rechte weniger hoch.

Bsp:

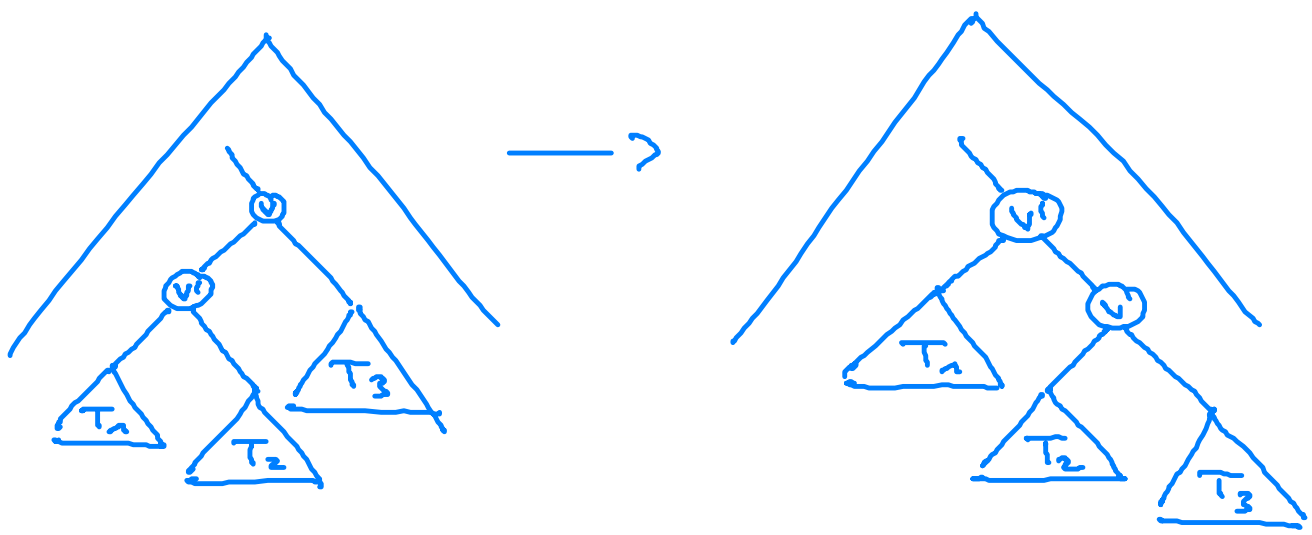


Aufwand: Referenz auf  $v$  gegeben:

$O(1)$  (nur 3 Referenzen aktualisieren)

symmetrisch dazu:

Rechtsrotation: Rot Rechts ( $v, v'$ )

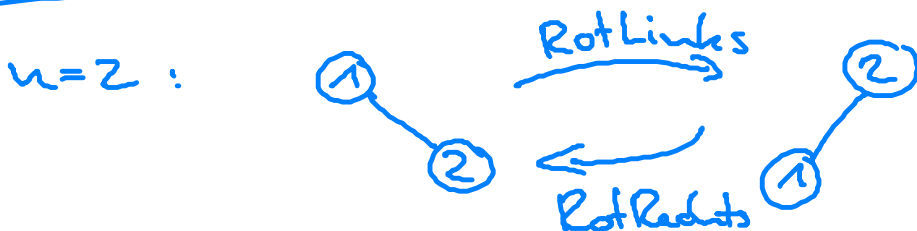


Wichtige Beobachtung: Die beiden Operationen Rot Links ( $v, v'$ ) und Rot Rechts ( $v', v$ ) sind zueinander invers.

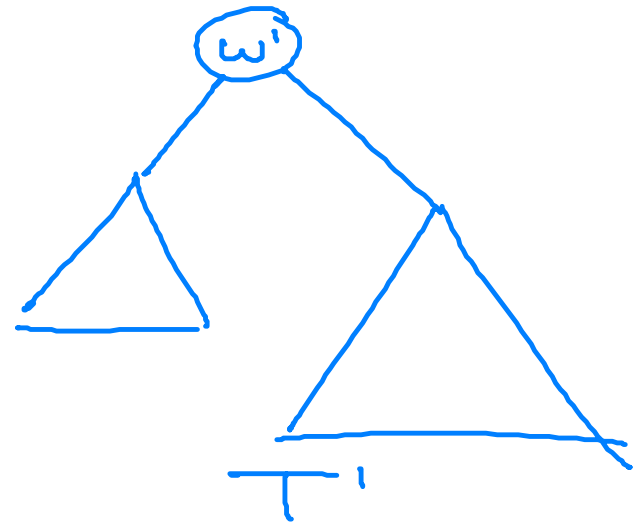
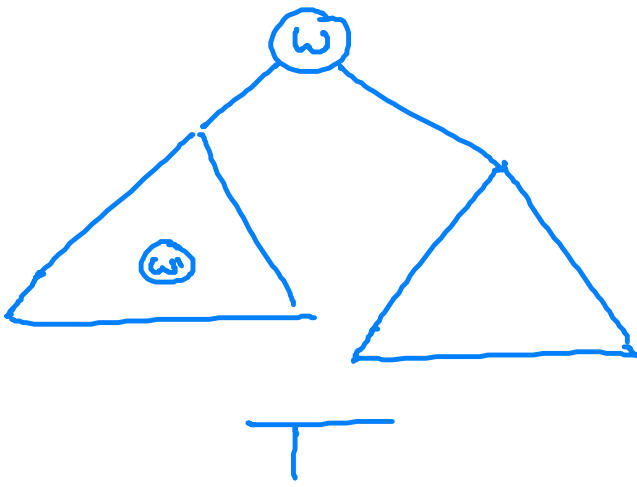
Die beiden Operationen sind mächtig genug um einen Suchbaum in jede gewünschte Gestalt zu bringen.

Satz: Seien  $T, T'$  Suchbäume zu derselben Schlüsselmenge. Dann gibt es eine endliche Folge von Rotationen, die  $T$  in  $T'$  überführen.

Beweis: Induktion über  $n$  ( $= \#$  Knoten)



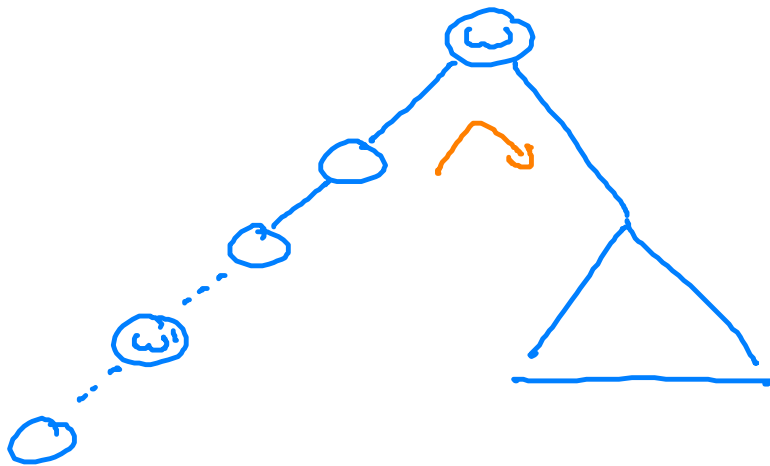
Ind. Schritt: Seien  $T, T'$  Suchbäume mit  $n$  Knoten



1. Fall:  $w' = w$ , fertig nach Induktion (angewandt auf Teilbäume)

2. Fall:  $w' \neq w$  und  $w'$  taucht in linkem Teilbaum von  $T$  auf.

Nach Induktion können wir den linken Teilbaum in die folgende Form bringen:



Wende nun so lange RotRechts auf Wurzel an, bis  $w'$  in Wurzel steht. Fahre fort wie in Fall 1

3. Fall:  $w' \neq w$  und  $w'$  in rechter Teilbaum  
→ symmetrisch zu 2. Fall.  $\square$