Suchsäume

Verwalte dynamische Daten.
Unterstütze 3 Operationen:

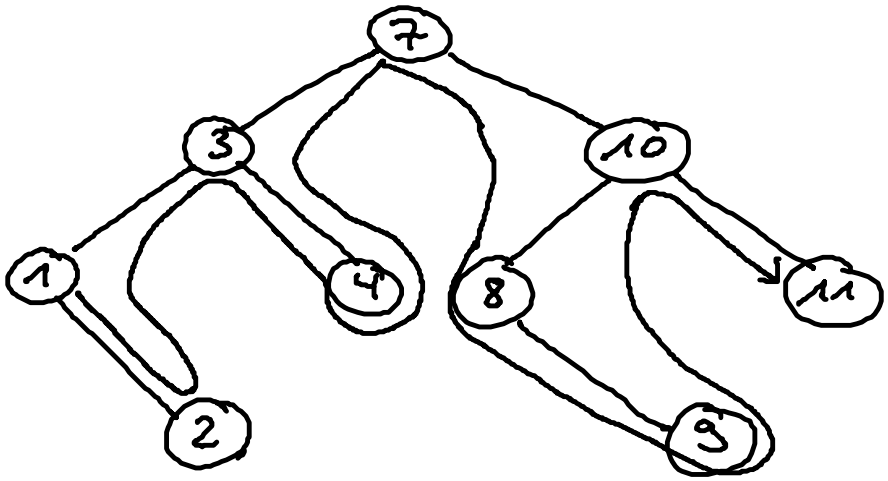
- suchen
- einfügen schnell!
- löschen

Suchbäume 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:

Suchbaumeigenschaft

Bsp:



äquivalent zur Suchbaumeigenschaft:
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 Operationen im Suchbaum

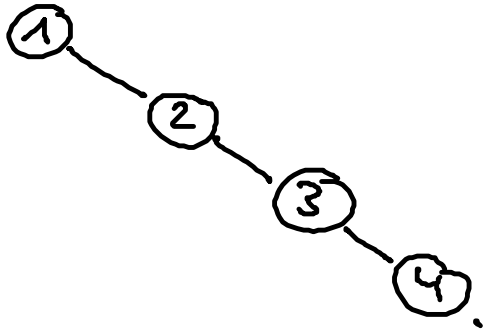
I) Suchen: Vergleiche gesuchten Schlüssel mit Wurzel:
falls „=", fertig
falls „<", dann suche im linken Teilbaum
falls „>", " " " rechten "

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

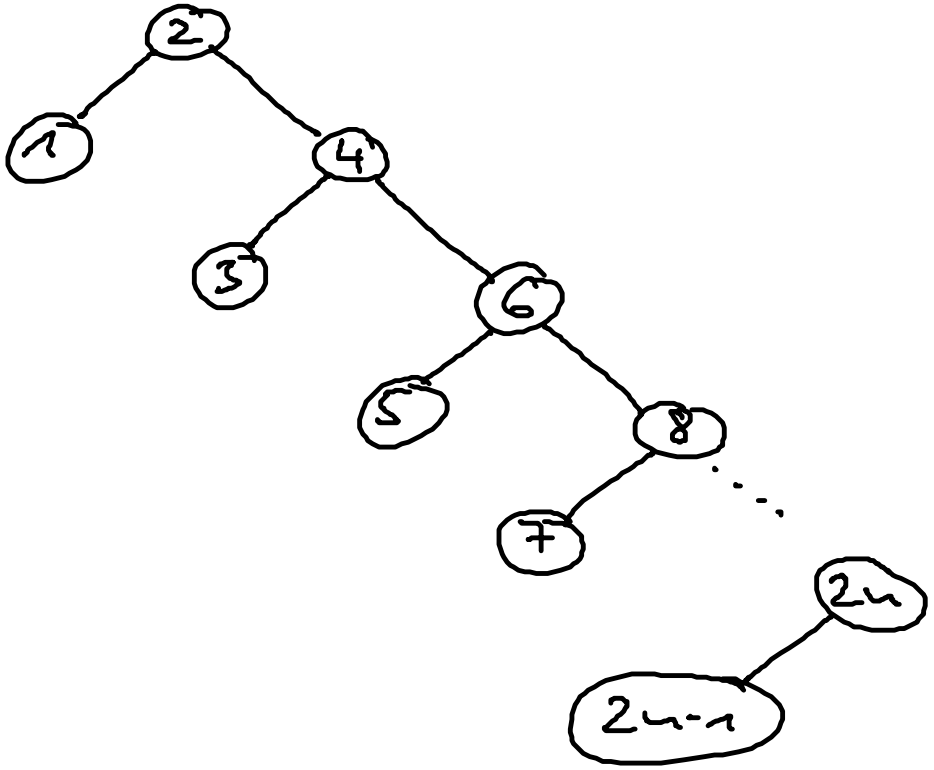
Problem: Falls T zu Liste entartet ist, benötigt man n Vergleiche:



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

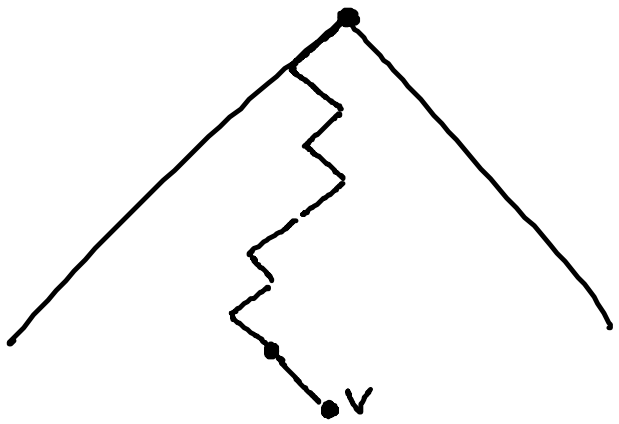
z.B.: fordern <u>vollständige</u> Bäume (d.h. jeder innere Knoten hat genau zwei Kinder)

aber das reicht nicht, denn:

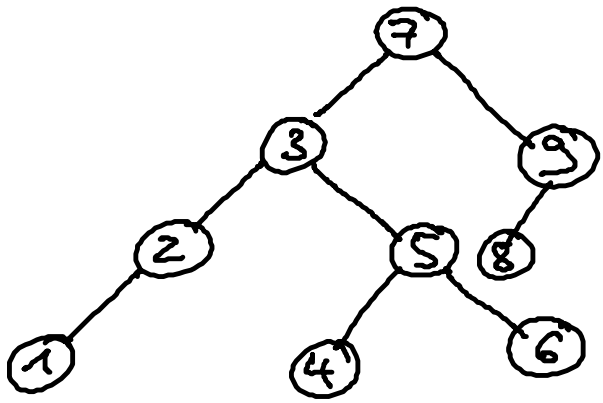


<u>„Idee" für Lösung:</u> Betrachte <u>volle Bäume</u>, bei denen alle Schichten außer der untersten voll besetzt sind.

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:

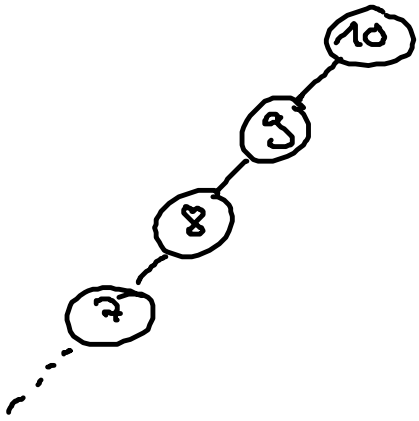


Aufwand für das Einfügen eines Elements: wie beim Suchen $O(h(T))$, also $O(n)$ im Worst Case.

→ brauche spezielle Bäume

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

10, 9, 8, 7, ...

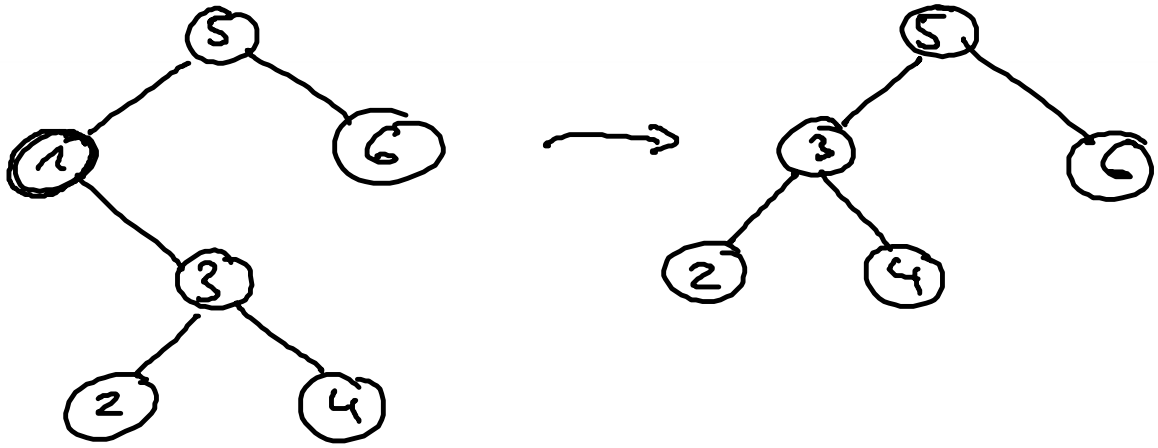


III) Löschen

Zu löschenden Knoten suchen, drei Fälle unterscheiden:

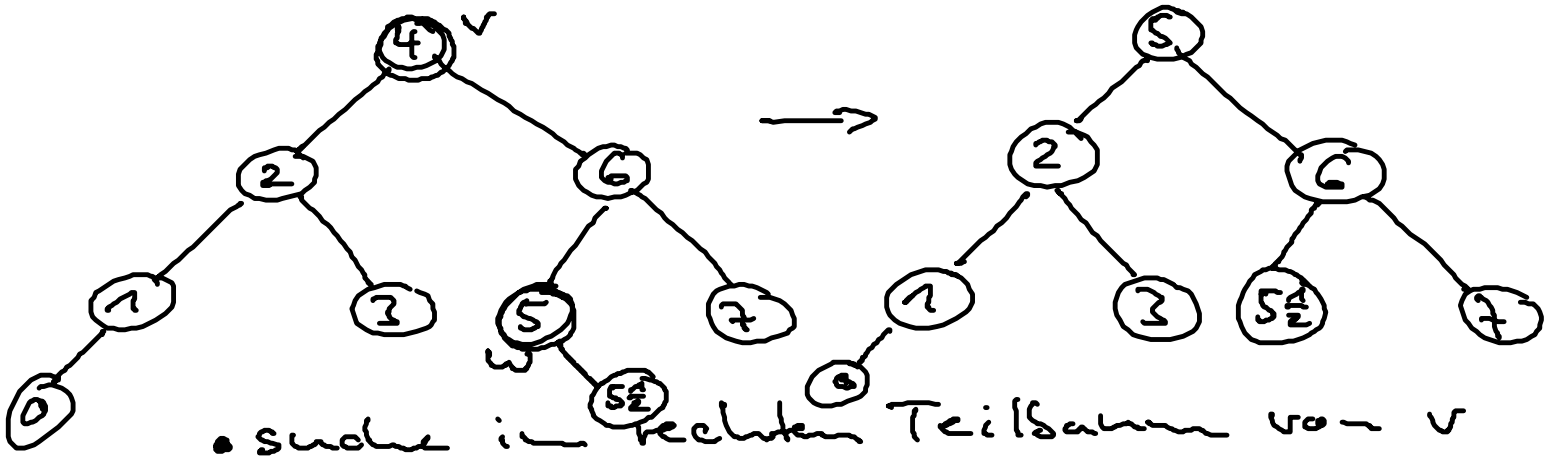
1) Knoten ist Blatt → einfach löschen, Suchbaumeigensch. bleibt erhalten

2) Knoten hat genau ein Kind:



→ ersetze v durch sein Kind

3) Knoten v hat 2 Kinder



- suche im rechten Teilbaum von v den linkesten Knoten (oder 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 inorder-Durchlauf, daher ist Suchbaumeigenschaft weiterhin erfüllt.

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

Austauschen der Inhalte: $O(1)$

Löschen von w: $O(1)$

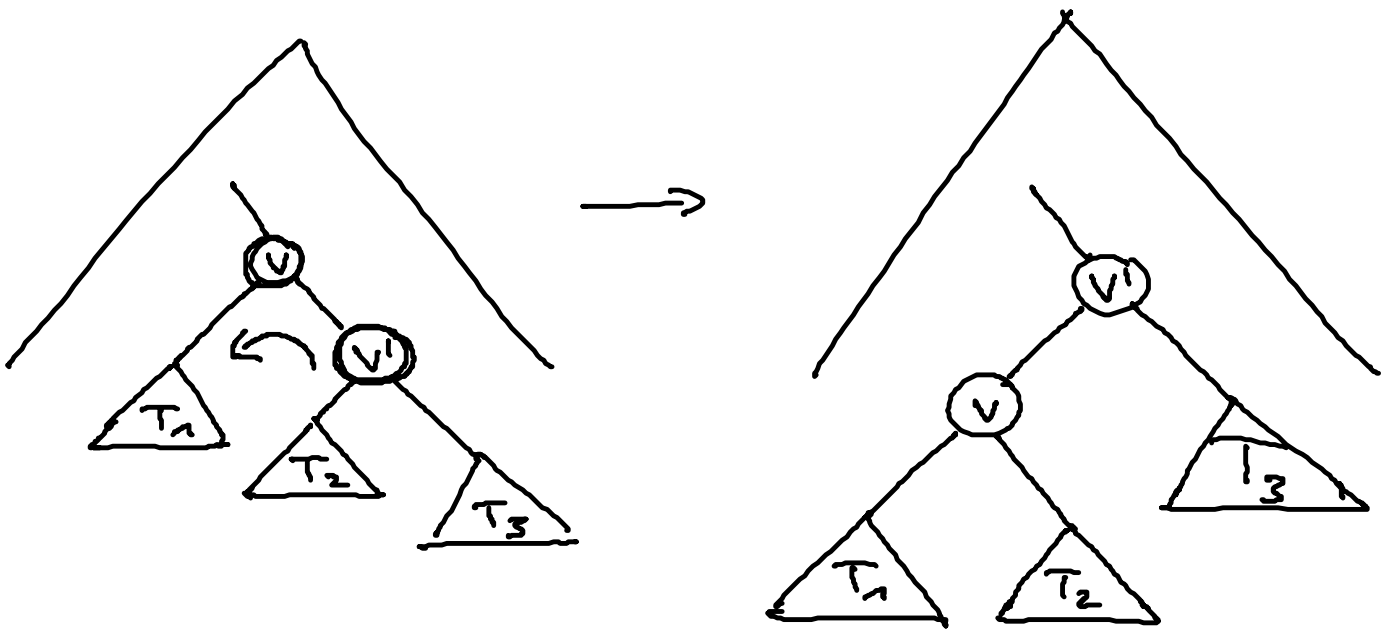
insgesamt: $O(h(T))$

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 Einfüge- und Lösch-
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: RotLinks(v, v')

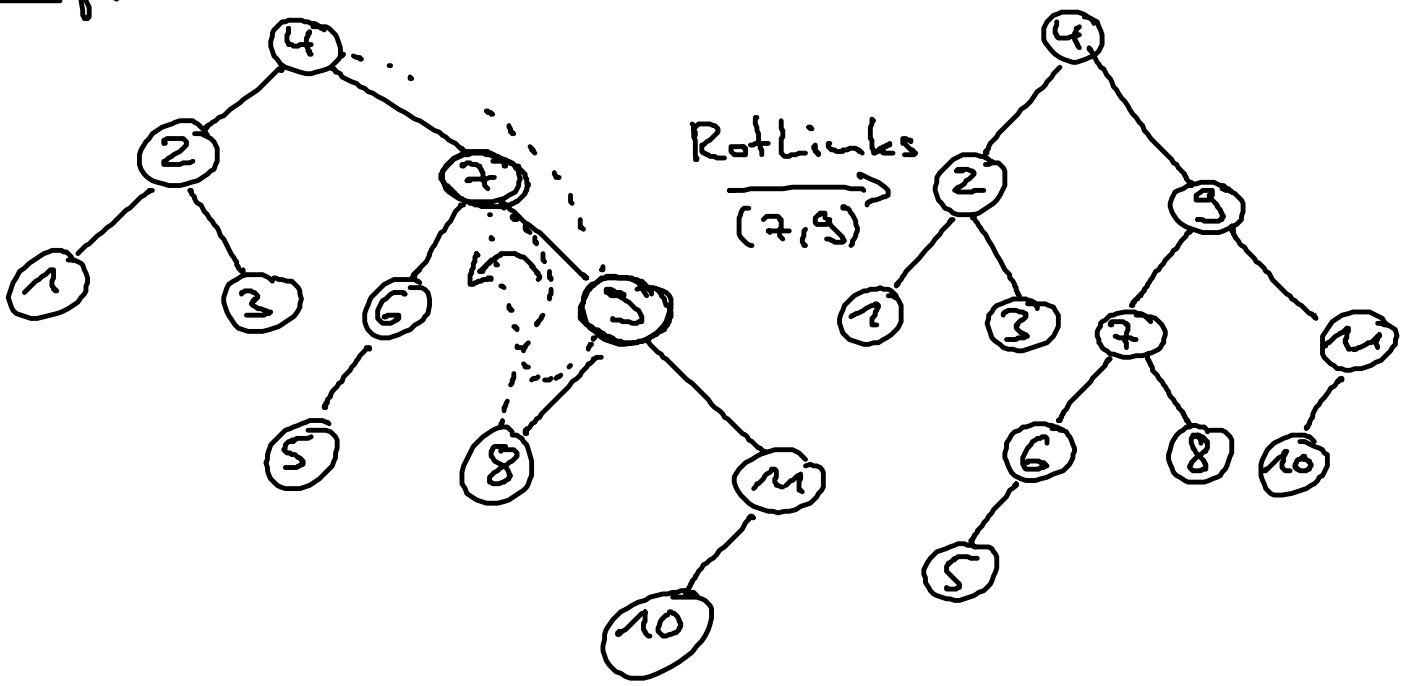


Beachte: Suchbaumeigenschaft ist
weiterhin erfüllt, denn:

„$T_1 < v < T_2 < v' < T_3$"

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

Bsp:

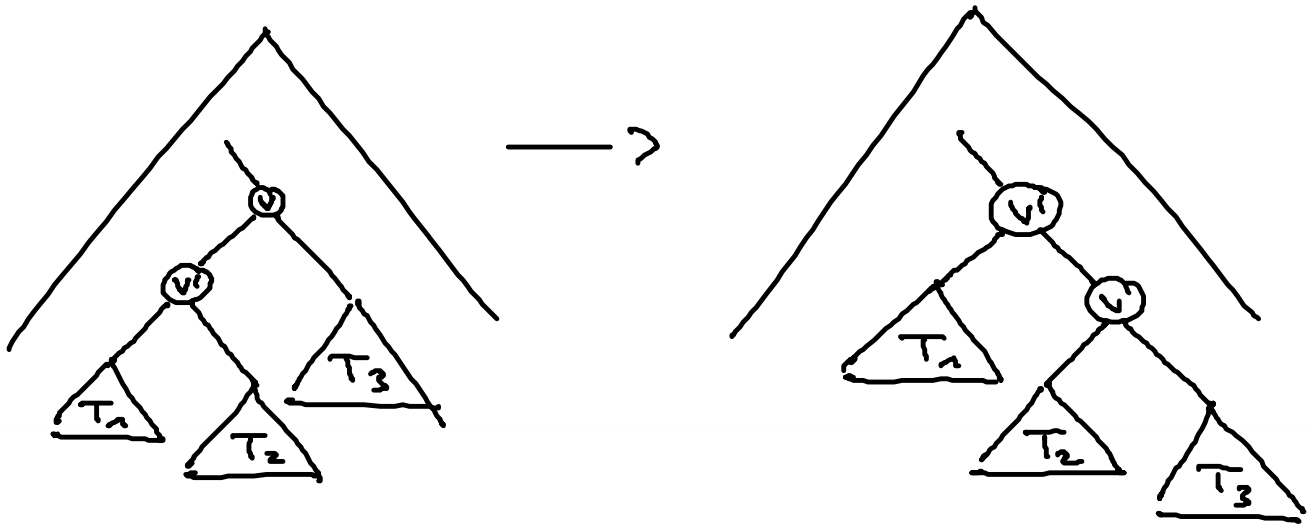


Aufwand: Referenz auf v gegeben:

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

symmetrisch dazu:

Rechtsrotation: RotRechts(v,v')



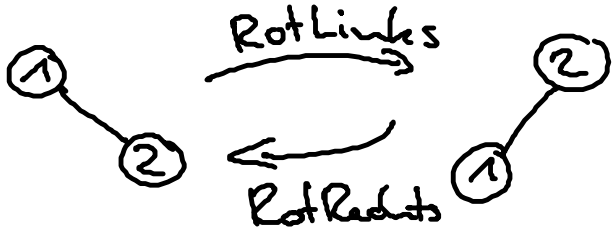
Wichtige Beobachtung: Die beiden Operationen RotLinks(v, v') und RotRechts(v', v) sind zueinander invers.

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

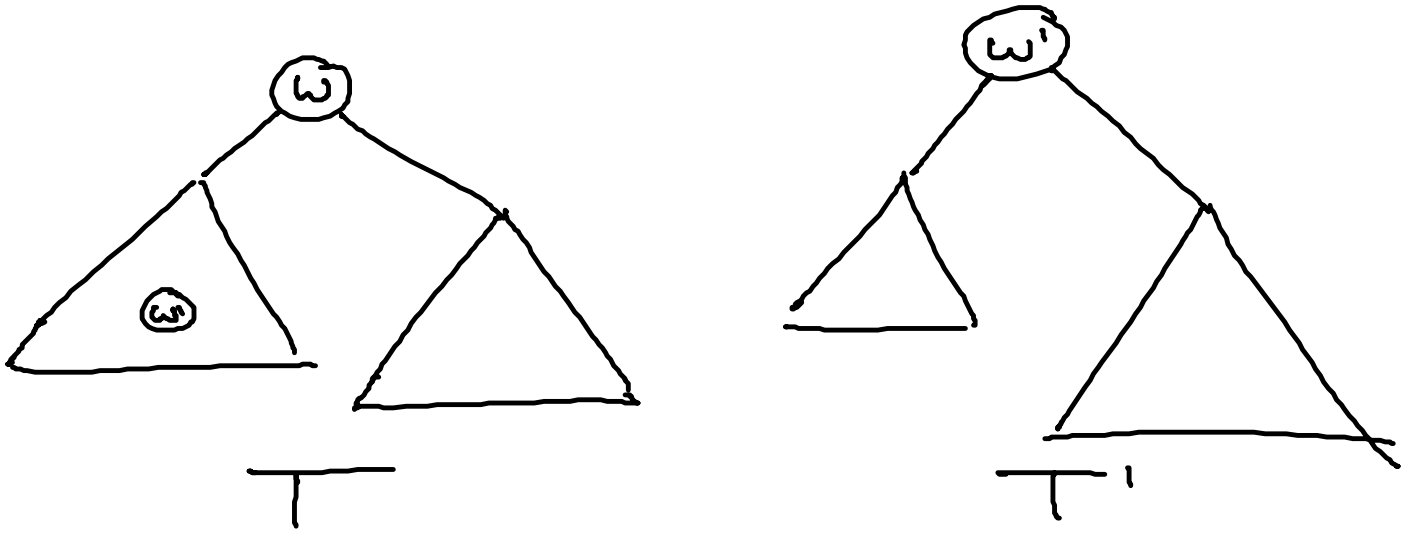
Satz: Seien T, T' Suchbäume zu derselben Schlüsselmenge. Dann gibt es eine endliche Folge von Rotationen, die T in T' überführen.

Beweis: Induktion über n (= # Knoten)

n=2:



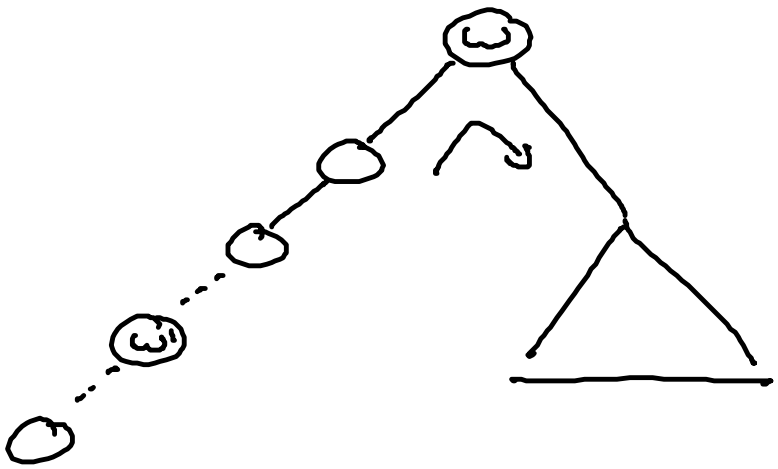
Ind. Schritt: Seien T, T' Suchbäume mit n Knoten



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

2. Fall: $w' \neq w$ und w' taucht im linken 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' im rechten Teilbaum
$\rightarrow$ symmetrisch zu 2. Fall. $\square$