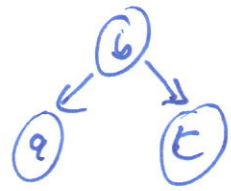
12.01.10

Gemeinsames Muster:

$a < b < c$ wird zu



!

(Vgl. Fälle!)

Also:

Algorithmus 4.8 (Umstrukturierung im Binärbaum)

Restructure (x)

Eingabe: Knoten x eines binären Suchbaumes T, Vaterknoten y, Großvaterknoten z

Ausgabe: Binärer Suchbaum T nach Umstrukturierung mit x, y, z.

1 Sei (a, b, c) die Größensortierung der Knoten x, y, z;
sei (T_0, T_1, T_2, T_3) die Größensortierung der vier Teilbäume unter x, y, z, die nicht Wurzeln x, y, z haben.

2 Ersetze den Teilbaum mit Wurzel z durch einen neuen Teilbaum mit Wurzel b.

3 Setze a als linkes Kind von b, T_0 und T_1 als linken und rechten Teilbaum unter a.

4 Setze c als rechtes Kind von b und T_2 und T_3 als linken und rechten Teilbaum von C.

Satz 4.9

Betrachte einen AVL-Baum T. Wird T durch Einfügen eines Knotens v unbalanciert, so ist T nach Aufruf von Restructure(x) wieder höhenbalanciert.

Beweis:

Siehe Fallunterscheidung; beachte, dass die Tiefen von T_0, T_1, T_2, T_3 sich nur um eins unterscheiden können, da T vor Einfügen von v höhenbalanciert war.

Satz 4.10

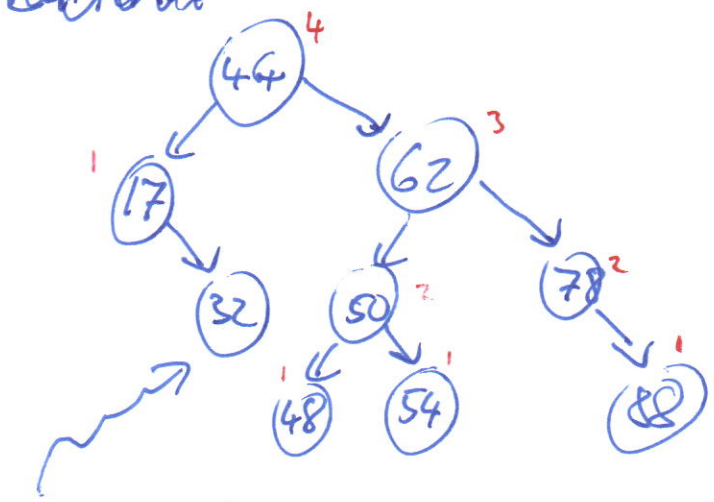
Einfügen in einen AVL-Baum kann man (unter Bewahrung der Höhenbalanciertheit) in $O(\log n)$

Beweis:

„Normales" Einfügen geht in $O(h)$, also $O(\log n)$; Restrukturieren geht in $O(1)$.

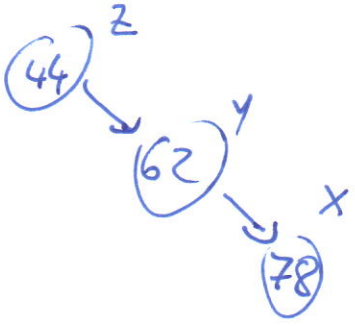
~~Einfügen:~~

Löschen:

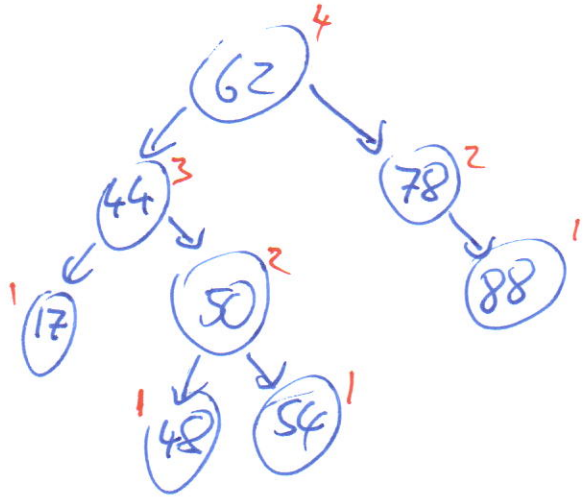


Löschen macht Baum unbalanciert!

Neu:



also



→ höhenbalanciert!

Ähnliche Idee:

Lösche ~~Entfernen~~ ✓

Sei z der erste ~~unbalancierte~~ Knoten auf dem Weg von v zur Wurzel, der nach Entfernung von v unbalanciert wird.

Sei y das Kind von z mit größerer Höhe (offenbar kein Vorfahre von v).

Sei x ein Kind von y mit größter Höhe. (Ggf. Unentschieden, dann Wahl egal!)

Dann Restructure(x)

Problem: Höhe des Teilbaums von b kann sich reduzieren → Vorfahre von b kann unbalanciert werden.

Idee: Immer wieder Restructure auf dem Weg zur Wurzel, bis nicht mehr notwendig!

Satz 4.4

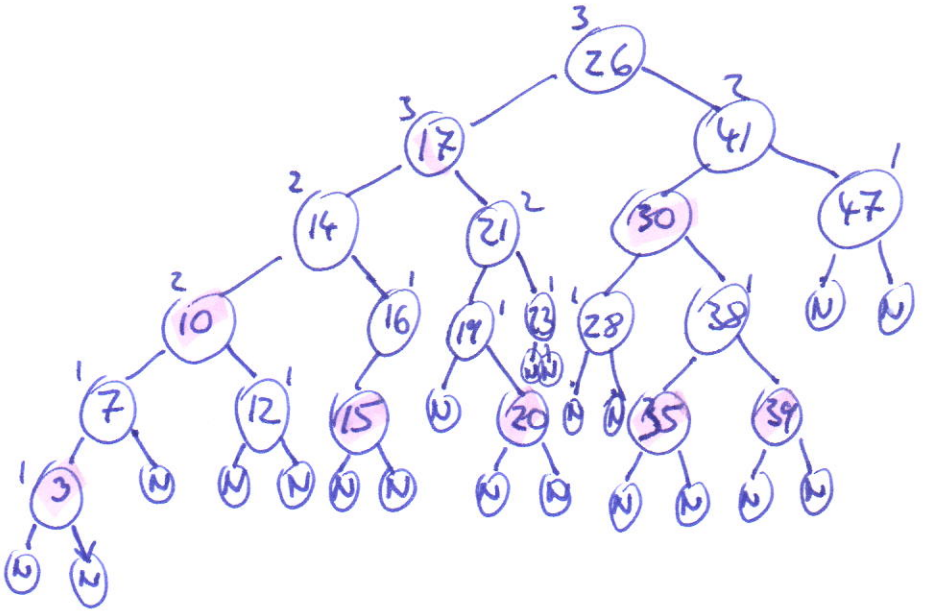
Löschen in einem AVL-Baum lässt sich (unter Bewahrung der Höhenbalanciertheit in $O(\log n)$ ausführen).

4.6 Andere Baumstrukturen

4.6.1 Rot-Schwarz-Bäume

Eigenschaften

0. Binärer Suchbaum, alle Blätter sind „NIL"



Kein AVL-Baum!

1. Jeder Knoten ist entweder rot oder schwarz
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.
4. Wenn ein Knoten rot ist, dann sind seine beiden Kinder schwarz
5. Für jeden Knoten enthalten alle Pfade, die an einem Knoten starten und in einem Blatt des Teilbaumes dieses Knotens enden, die gleiche Anzahl schwarzer Knoten.

Man kann folgende Dinge beweisen:

Satz 4.12

Ein Rot-Schwarz-Baum mit n inneren Knoten hat höchstens die Höhe $2 \log(n+1)$.

Beweis: Nicht hier!

Satz 4.13

Löschen und Einfügen (samt notwendiger Reparaturoperationen zum Erhalt der Eigenschaften 0.–5.) lassen sich in einem Rot-Schwarz-Baum in $O(\log n)$ durchführen, wobei n die Zahl der inneren Knoten ist

Historisch:

Bayer (1972) – Grundidee

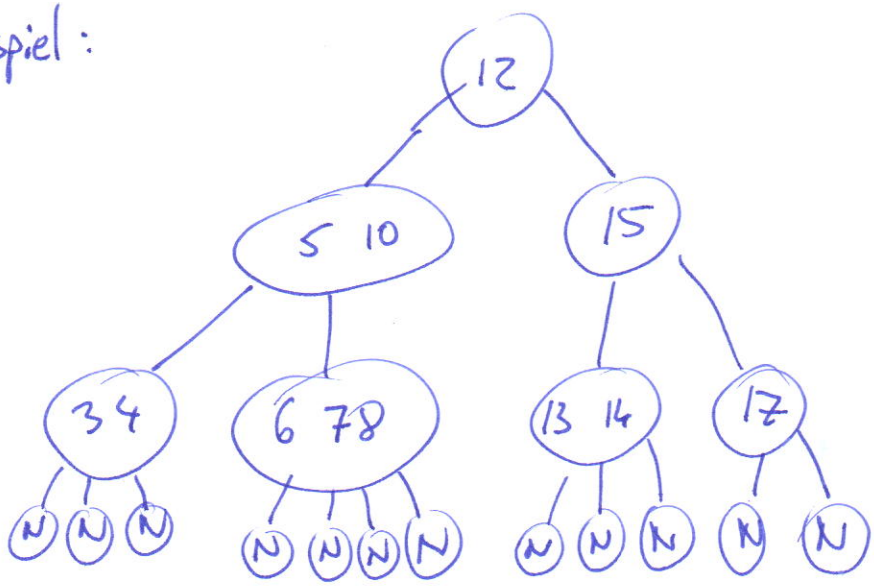
Guibas/Sedgwick (1978) – Farben + Eigenschaften

4.6.2 (2,4)-Bäume

Grundideen: (I) Erlaube mehr als ein Objekt pro Knoten (z.B. zwei oder drei)

(II) Erlaube mehr als zwei Kinder (z.B. bis zu vier)

Beispiel:



Eigenschaften:

(i) Jeder Knoten hat höchstens 3 Objekte,

(ii) Jeder Knoten hat höchstens 4 Kinder

Sowie

(iii) Jeder innere Knoten hat mindestens zwei Kinder

(iv) Alle Blätter haben dieselbe Tiefe