Algorithmen und Datenstrukturen

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Algorithmen und Datenstrukturen

Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, $\mathcal{O}$-Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

Binärbäume: Wozu?

Datenstruktur zum schnellen (logarithmischen) Zugriff auf abgelegte Daten.
. . . zumindest, wenn “Balance” gewahrt bleibt. . .

Wie werden denn Daten abgespeichert ?

1. *knotenorientiert* Zu Jedem Knoten ist ein Datum assoziiert.

2. *blattorientiert* Die eigentliche Information findet sich lediglich an den Blättern. Die inneren Knoten sollen lediglich einen schnellen Weg zur Information anzeigen.

Die in natürlicher Weise zu geordneten Mengen gehörige Speicherung ist die blattorientierte.

AVL-Bäume Adelson-Velskij/Landis 1962

Ein *AVL-Baum* ist ein 1-ausgeglichener Suchbaum, d.h., es handelt sich um einen binären Suchbaum, in dem sich für jeden inneren Knoten die Höhen der beiden anhängenden Teilbäume höchstens um Eins unterscheiden.

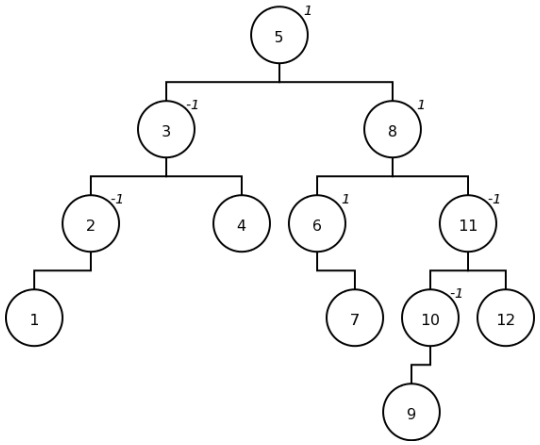
Das Suchen nach einem Element funktioniert weiter wie bei “normalen” Suchbäumen (die Suchbaumeigenschaft ist ja erfüllt).

Beim Einfügen und Löschen muss evtl. durch nachträgliches *Rebalancieren* die 1-Ausgeglichenheit wiederhergestellt werden.

Das wollen wir zunächst beobachten in: `http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm`.

AVL-Bäume: Balance

Jedem Knoten können wir als *Balancewert* die Differenz der Höhen des rechten und linken darunterhängenden Teilbaumes zuordnen:



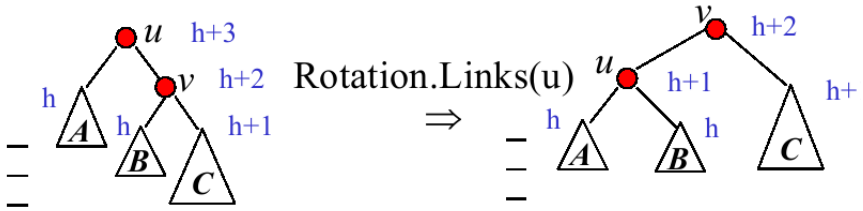
Satz: Alle (wichtigen) Operationen lassen sich auf AVL-Bäumen in Laufzeit $\mathcal{O}(\log(n))$ realisieren.

AVL-Bäume: Aus der Balance:

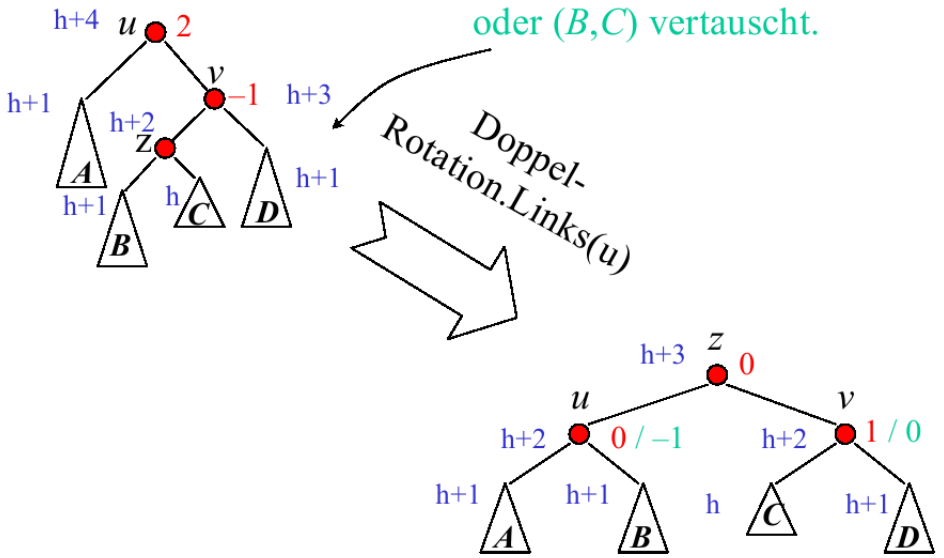
Zwei Grundfälle

(bei Einfügen oder auch beim Löschen)

1. (einfache) Rotation

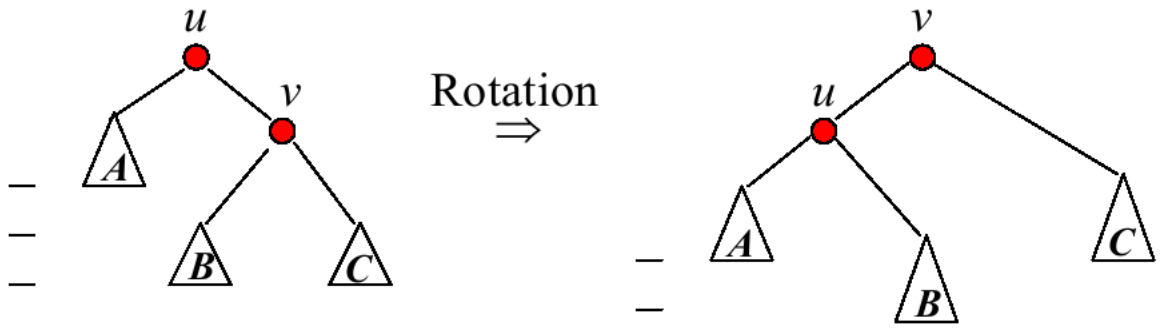


2. Fall: Doppelrotation



Ein günstiger Sonderfall

Beim Löschen kann folgende Situation auftreten:



Diese ist besonders günstig, da keine neuen Imbalance-Situationen höher im Baum erzeugt werden können.

Anwendung von balancierten Suchbäumen

Schnitt von achsenparallelen Liniensegmenten (*Segmentschnittproblem*)

Gegeben: Menge von insgesamt n vertikalen und horizontalen Liniensegmenten in der Ebene

Gesucht: Alle Paare von sich schneidenden Elementen

Trivialer Algorithmus: $\mathcal{O}(n^2)$

Wie geht's ? Paarweiser Vergleich der Segmente

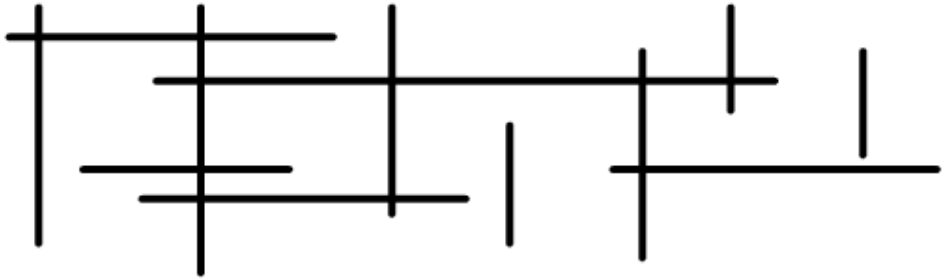
Anwendung z.B. bei VLSI-Schaltkreisen:

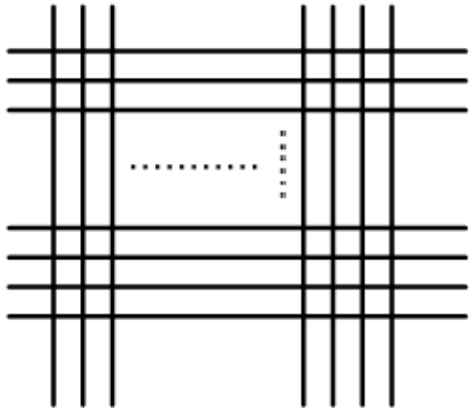
VLSI-Designer will keine unerwarteten "Drahtüberschneidungen"

Hinweis: *Algorithmische Geometrie* ist ein Gebiet, welches sich mit solchen Problemen befasst und in Trier von Herrn Näher vertreten wird.

Zur Anschauung des Problems

Bestimme alle Paare sich schneidender Segmente





Vereinfachende Annahme nachfolgend:
Alle Anfangs- und Endpunkte horizontaler Segmente und alle vertikalen Segmente haben paarweise verschiedene x-Koordinaten. (*Allgemeine Lage*)

Idee: Vertikale *Sweepline* geht Ebene von links nach rechts durch; jedes aktuell von der Sweepline abgedeckte vertikale Segment kann höchstens Schnittpunkte mit gerade *aktiven* horizontalen Segmenten haben.

Alternativ: Divide-and-Conquer-Ansatz, siehe auch: `http://download.informatik.uni-freiburg.de/lectures/Algorithmentheorie/2006-2007WS/LectureReFlash/01_3/Segmentschnittproblem.html`

Algorithmus-Idee: Der Anfang

$Q \leftarrow \{x \mid x$ ist x-Koordinate von Anfangs- oder Endpunkt eines horizontalen Segmentes oder x-Koordinate von einem vertikalen Segment $\}$;

Q merkt sich auch das Segment, zu dem es gehört.

Q verwaltet also die Menge der *Ereignisse*, das sind die für die Sweepline interessanten x-Koordinaten.

Zwischen zwei unmittelbar aufeinander folgenden Punkten aus x sind keine neuen Segmentschnitte zu verzeichnen.

Sortiere Q (bzgl. x-Werten);

Bei n Segmenten kostet dies (allein) bereits $\mathcal{O}(n \log(n))$ Zeit.

Die Sweepline L enthält die jeweils aktiven horizontalen Segmente in aufsteigender y-Koordinaten-Reihenfolge.

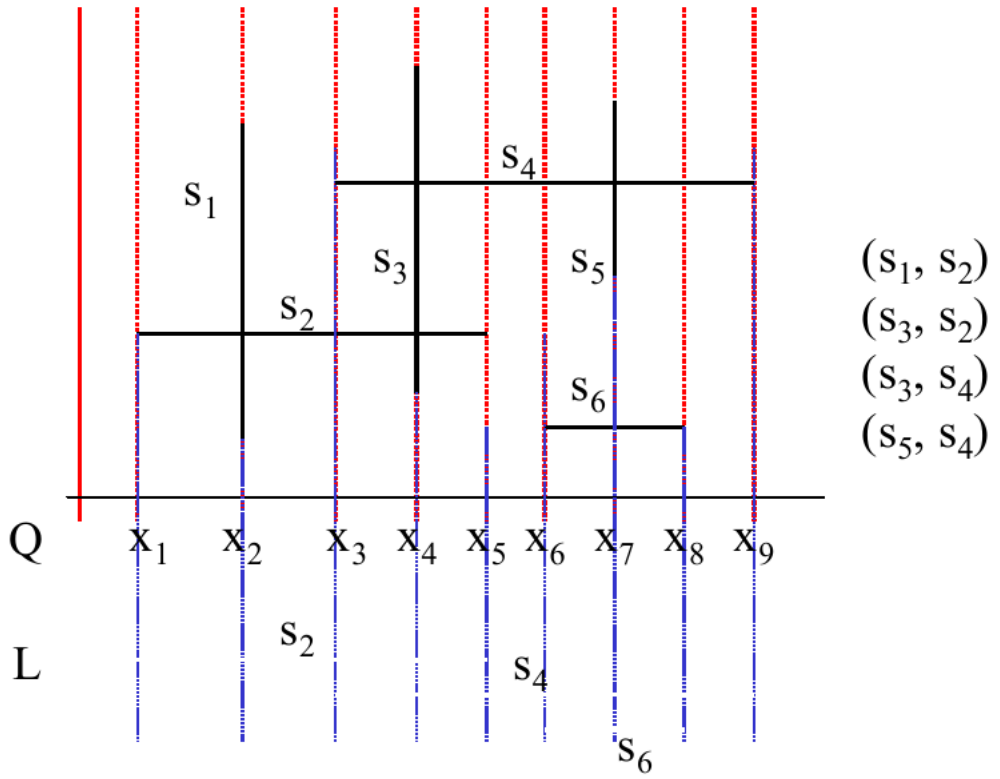
Eingangs gelte: $L \leftarrow \emptyset$.

Algorithmus-Idee: Der Durchführungsteil

```
while Q ≠ ∅ do
    p ← kleinstes Element in Q ;
    if p linker Endpunkt eines horizontalen Segments s
        then L ← L ∪ {s}
        else if p rechter Endpunkt eines horizontalen Segments s
                then L ← L \ {s}
                else /* p ist x-Wert eines vertikalen
                        Segments s = [ (p, y_unten), (p, y_oben)] */
                     Bestimme alle horizontalen Segmente t ∈ L
                     mit y_unten ≤ y(t) ≤ y_oben und gib (s,t) aus.
             fi;
    fi;
    Q ← Q \ {p}
```

Algorithmus-Idee:

- Beispiel: Sweepline-Algorithmus



Zur Implementierung der Sweepline

Wir benötigen folgende Operationen auf L:
—Einfügen & Entfernen
—Bestimmen aller Elemente, die in gegebenem Bereich $[y_u, y_o]$ fallen: *Bereichsanfrage* (Range Query)
$\rightsquigarrow$ Implementation von L mit balanciertem Suchbaum mit *blattorientierter Speicherung*, wobei benachbarte Blätter verkettet sind.
$\rightsquigarrow$ Mit AVL-Bäumen lassen sich die Sweepline-Operationen in $\mathcal{O}(\log(n))$ pro "Ereignis" realisieren.
Die Sweepline ändert sich höchstens $\mathcal{O}(n)$ oft ($\rightsquigarrow$ *Ereignisse*).
Zusammen: Laufzeit $\mathcal{O}(n \log(n) + k)$, wobei k die Gesamtzahl sich schneidender Segmente ist.
Gilt also $k = \Theta(n^2)$, ist der Algorithmus nicht besser als der naive, er ist aber *ausgabesensitiv*.
Hinweis: D&C liefert asymptotisch dieselbe Laufzeit.

Höhenbalancierte Bäume

Wir haben mit AVL-Bäumen ein Beispiel für sogenannte *höhenbalancierte Bäume* kennengelernt.
Dies ist nicht das einzige Balance-Kriterium, wie wir gleich sehen werden.
Es gibt viele weitere höhenbalancierte Baum-Modelle, z.B. Rot-Schwarz-Bäume.
Diese werden (gemeinsam mit AVL) in der folgenden Semesterarbeit (an der Uni Leipzig) vorgestellt: `http://leechuck.de/ginfin/baum1/index.html`
(auf den Seiten findet sich übrigens auch JAVA-Quellcode hierzu sowie ein nettes JAVA-Applet zu Mergesort)
Wie im Buch von Sedgewick ausgeführt, gestatten Rot-Schwarz-Bäume einen Top-Down-Ansatz zur Rebalancierung, was insbesondere für ihre parallele Ausführung erhebliche Vorteile bringt.

Wurzelbalancierte Bäume

$V(T)$ bezeichne die Blattmenge eines Baums T.
Sei T ein binärer Baum mit linkem Teilbaum T_ℓ und rechtem Teilbaum T_r, also $T = (T_\ell, r, T_r)$.

$$\rho(r) := \rho(T) := \frac{|V(T_\ell)| + 1}{|V(T)| + 1} = 1 - \frac{|V(T_r)| + 1}{|V(T)| + 1}$$

ist die *Wurzelbalance* von r bzw. von T.
Gilt für jeden Teilbaum T' von T

$$\alpha \leq \rho(T') \leq 1 - \alpha,$$

so heißt T von *beschränkter Wurzelbalance* α.
$B[\alpha]$ sei die Menge aller Bäume von beschränkter Wurzelbalance α.

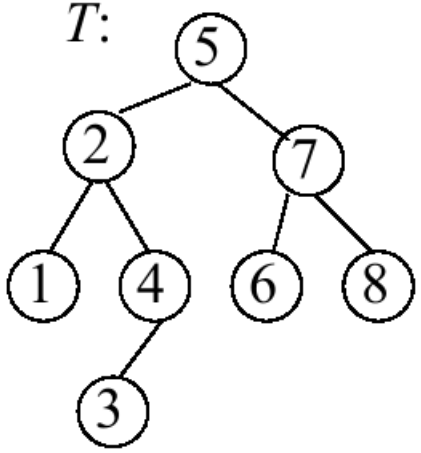
Wurzelbalancierte Bäume: Beispiel

Sei T_x Unterbaum mit Wurzel x

l sei die # Knoten im linken Unterbaum

x	$\rho(T_x) = \frac{l+1}{n+1}$
1	1/2 = (0+1)/(1+1)
2	2/5 = (1+1)/(4+1)
3	1/2
4	2/3
5	5/9
6	1/2
7	2/4 = 1/2
8	1/2

T:



T ist BB[α]-Baum für $\alpha \leq \frac{1}{3}$.

T ist kein BB[α]-Baum für $\alpha > \frac{1}{3}$.

Satz: Sei T ein $BB[\alpha]$-Baum mit n Knoten und Höhe $h(T)$. Dann gilt

$$h(T) \leq \frac{\log(n+1)-1}{\log(1-\alpha)}$$

Ebenso ist die Pfadlänge von T im Mittel logarithmisch in n.

Beweis: Wir beweisen nur die erste Satzaussage.
Für die zweite verweisen wir auf das Buch von Mehlhorn mit der Warnung, dass M. eine abgewandelte Definition von Wurzelbalance benutzt.
$n = 1$: Dann hat T Höhe null ✓
Als Induktionsvoraussetzung nehmen wir die Gültigkeit der Aussage für Bäume mit höchstens n Knoten an. Betrachte einen Baum T' mit $n+1$ Knoten.
Sei T'' derjenige echte Unterbaum von T' mit den meisten Knoten, nämlich n''.
Da T'' echter Unterbaum, hat T'' höchstens n Knoten.

$$\text{IV} \rightsquigarrow h(T'') \leq \frac{\log(n''+1)-1}{\log(1-\alpha)}$$

Der Baum T' ist genau um Eins höher als T'', und außerdem gilt die Balance-Bedingung für T' d.h.

$$h(T') \leq \frac{\log((n+1))(1-\alpha)-1}{\log(1-\alpha)} + 1$$

Die Induktions-Behauptung folgt sofort durch leichtes Logarithmen-Rechnen.

Zusatz: Eine besonders günstige Wahl ist $\alpha = 1 - \sqrt{2}/2 \approx 0,2929$.
Dann liegt die mittlere Suchzeit (mittlere Pfadlänge) um höchstens 15% über dem theoretischen Optimum, und die maximale Suchzeit liegt um höchstens einen Faktor von zwei über dem theoretischen Optimum.

Problem: Zerstören der Wurzelbalance durch Einfügen und Löschen.

Rebalancierung von $\mathrm{BB}[\alpha]$-Bäumen

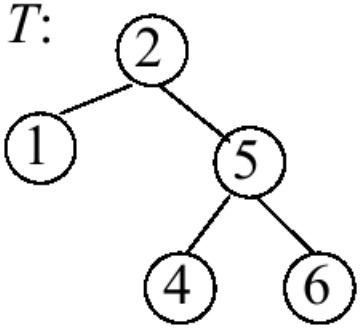
Problem: Durch Einfügen und Löschen in $\mathrm{BB}[\alpha]$-Bäumen kann deren Balance zerstört werden.
Lösung: Die Balance kann nach solch einer Operation nur auf dem Suchpfad (Weg von der Wurzel zur Stelle der Änderung) zerstört werden. Längs dieses Suchpfades muss rebalanciert werden. Der Pfad hat logarithmische Länge.
Unter der Voraussetzung

$$\frac{2}{11} \leq \alpha \leq 1 - \frac{1}{2}\sqrt{2} \approx 0.293$$

kann man zeigen, dass die lokalen Operationen Rotation und Doppelrotation auf dem Suchpfad zur Rebalancierung ausreichen.

Beispiel zur Rebalancierung: Sei $\alpha = 0.29$, Einfügen(7)

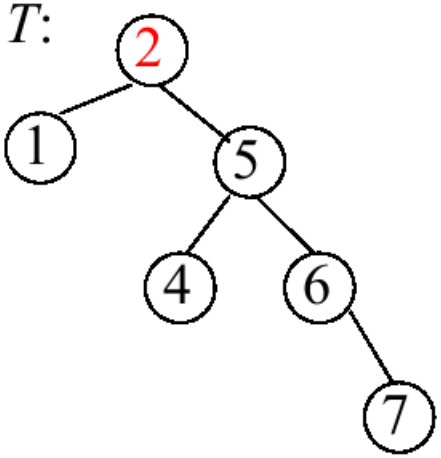
T:



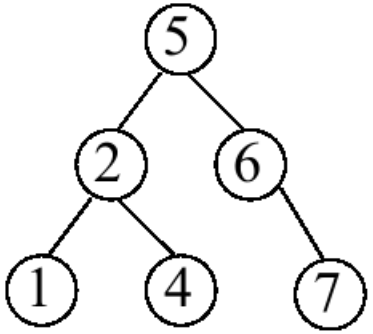
i	2	5	1	4	6
ρ_i	2/6	2/4	1/2	1/2	1/2

i	2	5	1	4	6	7	
ρ_i	2/7	2/5	1/2	1/2	1/3	1/2	vor R.
	2/4	4/7	...	...	...	...	nach R

T:

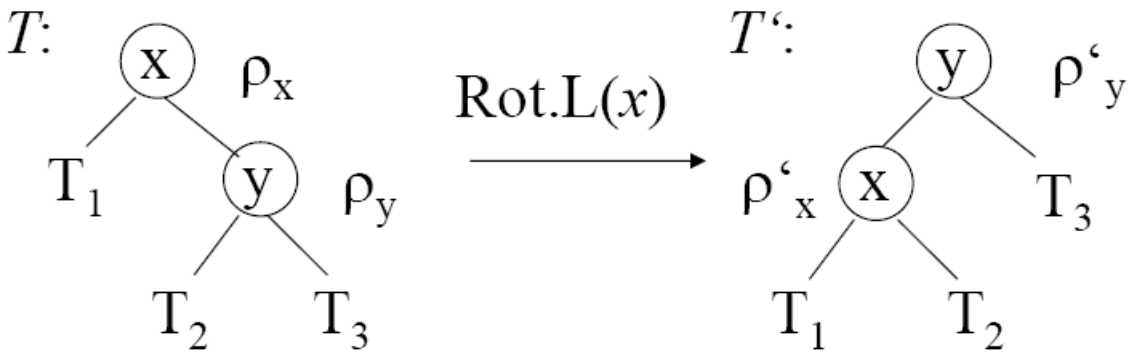


Rot.L(2)



Lemma 1: Wirkung von Rotation und Doppelrotation

a) Bei Rotation nach links um x

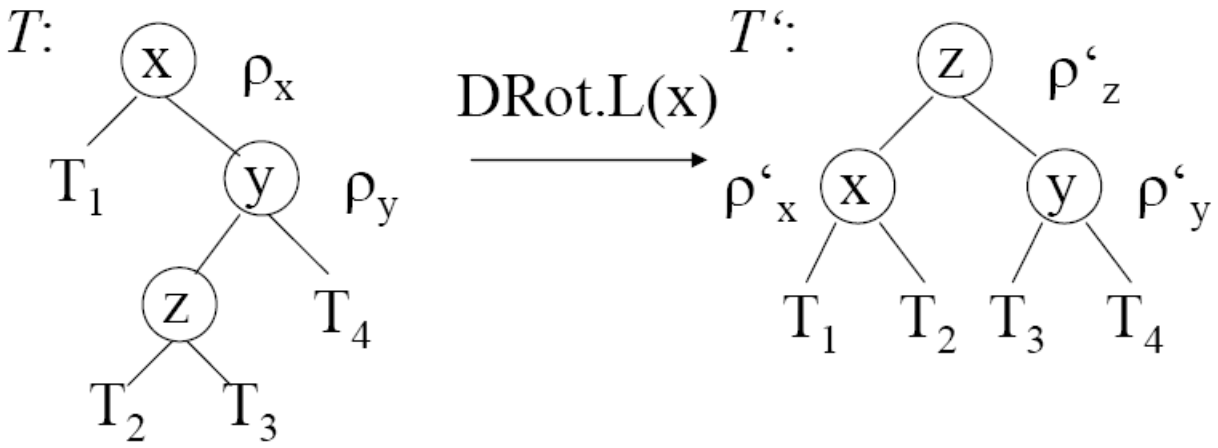


gilt für die Wurzelbalance

$$\rho'_x = \frac{\rho_x}{\rho_x + (1-\rho_x)\cdot\rho_y}$$

$$\rho'_y = \rho_x + (1-\rho_x)\cdot\rho_y$$

b) Bei Doppelrotation nach links



gilt für die Wurzelbalance

$$\rho'_x = \frac{\rho_x}{\rho_x + (1-\rho_x)\cdot\rho_y\cdot\rho_z}$$

$$\rho'_y = \frac{\rho_y(1-\rho_z)}{1-\rho_y\rho_z}$$

$$\rho'_z = \rho_x + (1-\rho_x)\cdot\rho_y\cdot\rho_z$$

Beweis des Lemmas 1:

Sei a_i-1 die Anzahl der Knoten im Teilbaum T_i:

a) Es gilt $\rho_x = \frac{a_1}{a_1 + a_2 + a_3}$ $\qquad \rho'_x = \frac{a_1}{a_1 + a_2}$

$\rho_y = \frac{a_2}{a_2 + a_3}$ $\qquad \rho'_y = \frac{a_1 + a_2}{a_1 + a_2 + a_3}$

Damit gilt

$$\frac{\rho_x}{\rho_x + (1 - \rho_x) \cdot \rho_y} = \frac{\frac{a_1}{a_1 + a_2 + a_3}}{\frac{a_1}{a_1 + a_2 + a_3} + \frac{a_2 + a_3}{a_1 + a_2 + a_3} \cdot \frac{a_2}{a_2 + a_3}} = \frac{a_1}{a_1 + a_2} = \rho'_x$$

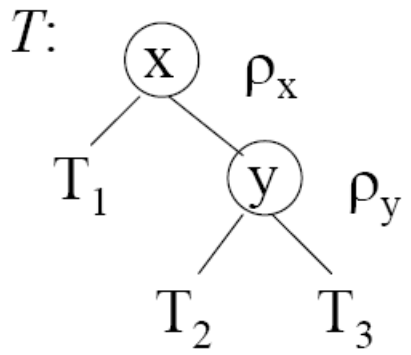
$$\rho_x + (1 - \rho_x) \cdot \rho_y = \frac{a_1}{a_1 + a_2 + a_3} + \frac{a_2 + a_3}{a_1 + a_2 + a_3} \cdot \frac{a_2}{a_2 + a_3} = \frac{a_1 + a_2}{a_1 + a_2 + a_3} = \rho'_y$$

b) analog

Lemma 2: Sei T ein binärer Suchbaum, der bei x nur ein wenig außer Balance geraten ist, d.h.

i) Jeder echte Unterbaum von T ist BB[α]-Baum

ii) $\alpha(1-\alpha) \leq \rho_x < \alpha$



Dann gilt:

a) Ist $\rho_y > \frac{1-2\alpha}{1-\alpha}$ und entsteht T' aus T durch eine Doppelrot. nach links um x, so ist T' ein BB[α]-Baum.

b) Ist $\rho_y \leq \frac{1-2\alpha}{1-\alpha}$ und entsteht T' aus T durch eine Rotation nach links um x, so ist T' ein BB[α]-Baum.

Beweisidee des Lemmas:

a) Es muss gezeigt werden, dass eine Doppelrotation möglich ist (T_2 nicht leer) und dass T' ein BB[α]-Baum ist, d.h. $\alpha \le \rho'_x, \rho'_y, \rho'_z \le 1-\alpha$.

b) analog

Bemerkung: Sei T ein BB[α]-Baum, in dem ein einzelnes Element eingefügt oder gelöscht wird, wodurch T' entstehe. Man kann zeigen, dass

i) die Balancen außerhalb des Suchpfades unverändert sind, also im zulässigen Bereich [α, 1-α] liegen.

ii) die Balancen auf dem Suchpfad unverändert oder nur wenig außerhalb von [α, 1-α] liegen und T' gemäß Lemma 2 durch Rotation und Doppelrot. balanciert werden kann.

Algorithmus zur Rebalancierung von BB[α]-Bäumen nach dem Einfügen/Löschen eines einzelnen Knotens.

Sei $v_0, v_1, \ldots v_k$ der Weg von der Wurzel bis zu der Stelle, an der die Struktur des Baumes geändert wurde,
Sei T_i der Teilbaum mit Wurzel v_i:

```
for i := k downto 0 do                    /* d.h. vom Blatt zur Wurzel! */
  begin if  ρ(T_i) ≤ α  then              /* T' sei rechter Teilb. von T_i */
              if ρ(T') ≤ (1-2α)/(1-α) then Rotation.Links(v_i)
                                      else DoppelRot.Links(v_i);
            if ρ(T_i) > 1-α  then       /* T' sei linker Teilb. von T_i */
              if  ρ(T') > α/(1-α)  then Rotation.Rechts(v_i)
                                      else DoppelRot.Rechts(v_i);
  end;
```

Bemerkungen zur Realisierung

a) In jedem Knoten des Baumes wird neben Inhalt, RSohn, LSohn auch die Anzahl der Nachfolger gespeichert. Dann lässt sich die Wurzelbalance in jedem Knoten berechnen.

b) Die Knoten des Suchpfades werden beim Abstieg auf einem Stack (Keller) abgelegt. Dann kann man den Suchpfad leicht hochsteigen.

c) Zeitbedarf einer Rebalancierung: $O(\log n)$,
denn es ist ein Ab- und ein Aufstieg des Suchpfades nötig, der Pfad hat Länge $O(\log n)$.