KONSTRUKTION VON ROT-SCHWARZ-BÄUMEN

RALF HINZE

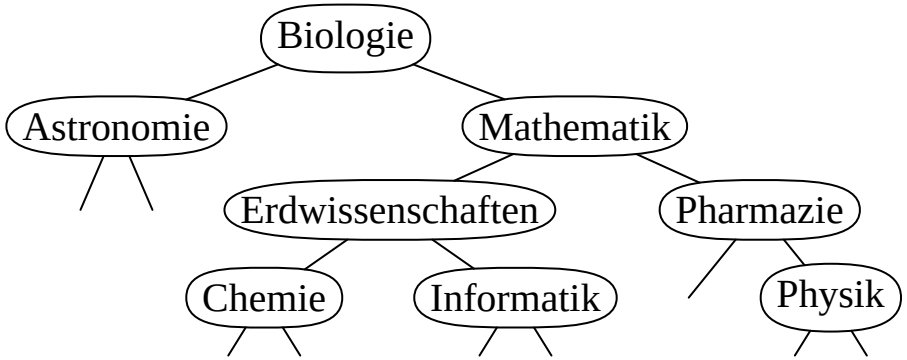
Institut für Informatik III
Universität Bonn
Email: ralf@informatik.uni-bonn.de
Homepage: http://www.informatik.uni-bonn.de/~ralf

Februar, 2001

Binäre Suchbäume

Eine geeignete Datenstruktur zur Verwaltung einer *Menge von Schlüsseln* sind binäre Suchbäume.

$$\textbf{data}\ \mathit{Tree} \quad = \quad \mathit{Empty} \mid \mathit{Node}\ \mathit{Tree}\ \mathit{Key}\ \mathit{Tree}$$



Um die Degeneration der Bäume zu vermeiden, sind Balancierungsschemata entwickelt worden: AVL-Bäume (Adel'son-Vel'skiĭ und Landis, 1962), 2-3-Bäume (Hopcroft, 1970), Rot-schwarz-Bäume (Bayer, 1972) usw.

Rot-schwarz-Bäume

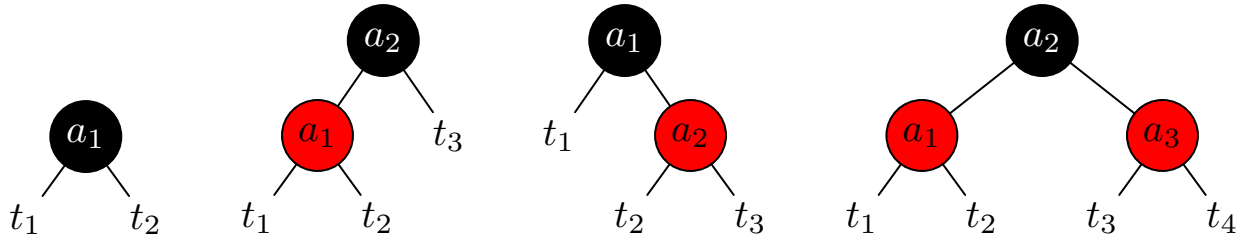
Ein Rot-schwarz-Baum ist ein Binärbaum, dessen innere Knoten entweder rot oder schwarz eingefärbt sind (Blätter sind definitionsgemäß schwarz).

$$\begin{array}{lcl} \textbf{data}\ \mathit{Color} & = & \mathit{Red} \mid \mathit{Black} \\ \textbf{data}\ \mathit{Tree} & = & \mathit{Empty} \mid \mathit{Node}\ \mathit{Color}\ \mathit{Tree}\ \mathit{Key}\ \mathit{Tree} \end{array}$$

Zur Historie von Rot-schwarz-Bäumen

Rot-schwarz-Bäume wurden von Bayer unter dem Namen *symmetric binary B-trees* als binäre Darstellung von 2-3-4-Bäumen entwickelt.

In einem Rot-schwarz-Baum werden die 3- und die 4-Knoten durch kleine Binärbäume repräsentiert, die aus einer schwarzen Wurzel und ein oder zwei roten Hilfsknoten bestehen.

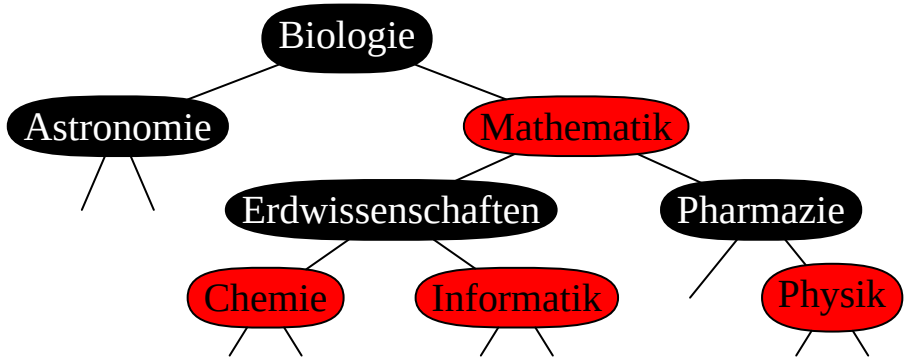


Balancierungsbedingungen

Die historischen Wurzeln erklären die beiden folgenden Balancierungsbedingungen.

Rot-Bedingung: Jeder rote Knoten hat einen schwarzen Vorgänger.

Schwarz-Bedingung: Jeder Pfad von der Wurzel zu einem Blatt enthält die gleiche Anzahl schwarzer Knoten (Schwarzhöhe).



Problemstellung

Gegeben sei eine *aufsteigende* Folge von Schlüsseln. Konstruiere in linearer Zeit einen Rot-schwarz-Baum, der die Schlüssel in symmetrischer Reihenfolge enthält.

Ansatz #1: Analysiere und verbessere die naive Methode des wiederholten Einfügens in einen anfangs leeren Baum.

Ansatz #2: Verwende bekannte Algorithmen zur Konstruktion von Binärbäumen minimaler Höhe. *Teilproblem:* Gegeben sei ein *beliebiger* Binärbaum. Kann dieser Baum so eingefärbt werden, daß ein Rot-schwarz-Baum entsteht?

Ansatz #1: Einfügen

Der folgende Algorithmus geht auf Okasaki (1999) zurück.

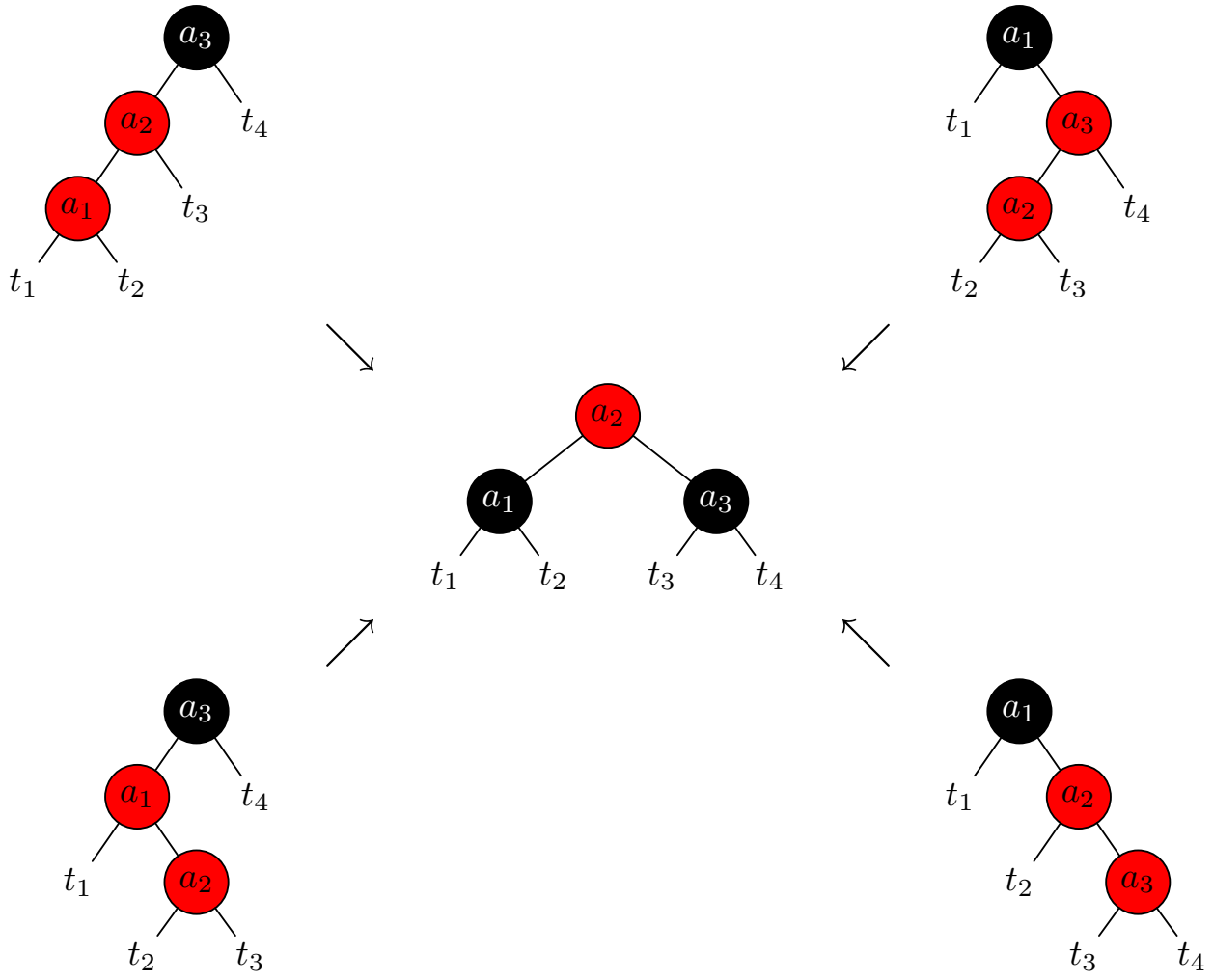
$$
\begin{array}{lcl}
insert & :: & Key \rightarrow Tree \rightarrow Tree \\
insert\ a\ t & = & blacken\ (ins\ a\ t) \\
blacken\ (Node\ c\ l\ k\ r) & = & Node\ Black\ l\ k\ r
\end{array}
$$

$$
\begin{array}{lcl}
ins\ a\ Empty & = & Node\ Red\ Empty\ a\ Empty \\
ins\ a\ (Node\ c\ l\ k\ r) & & \\
\quad \mid a < k & = & balance\ c\ (ins\ a\ l)\ k\ r \\
\quad \mid a == k & = & Node\ c\ l\ a\ r \\
\quad \mid a > k & = & balance\ c\ l\ k\ (ins\ a\ r)
\end{array}
$$

Ansatz #1: Balancierung

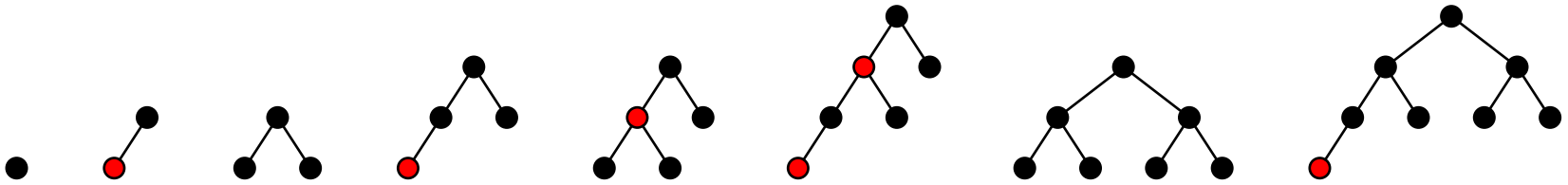
Da ein neu eingefügter Knoten rot eingefärbt ist, kann nur die Rot-Bedingung verletzt werden.

$$
\begin{aligned}
&\mathit{balance}\ \mathit{Black}\ (\mathit{Node}\ \mathit{Red}\ (\mathit{Node}\ \mathit{Red}\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4 \\
&\quad = \quad \mathit{Node}\ \mathit{Red}\ (\mathit{Node}\ \mathit{Black}\ t_1\ a_1\ t_2)\ a_2\ (\mathit{Node}\ \mathit{Black}\ t_3\ a_3\ t_4) \\
&\mathit{balance}\ \mathit{Black}\ (\mathit{Node}\ \mathit{Red}\ t_1\ a_1\ (\mathit{Node}\ \mathit{Red}\ t_2\ a_2\ t_3))\ a_3\ t_4 \\
&\quad = \quad \mathit{Node}\ \mathit{Red}\ (\mathit{Node}\ \mathit{Black}\ t_1\ a_1\ t_2)\ a_2\ (\mathit{Node}\ \mathit{Black}\ t_3\ a_3\ t_4) \\
&\mathit{balance}\ \mathit{Black}\ t_1\ a_1\ (\mathit{Node}\ \mathit{Red}\ (\mathit{Node}\ \mathit{Red}\ t_2\ a_2\ t_3)\ a_3\ t_4) \\
&\quad = \quad \mathit{Node}\ \mathit{Red}\ (\mathit{Node}\ \mathit{Black}\ t_1\ a_1\ t_2)\ a_2\ (\mathit{Node}\ \mathit{Black}\ t_3\ a_3\ t_4) \\
&\mathit{balance}\ \mathit{Black}\ t_1\ a_1\ (\mathit{Node}\ \mathit{Red}\ t_2\ a_2\ (\mathit{Node}\ \mathit{Red}\ t_3\ a_3\ t_4)) \\
&\quad = \quad \mathit{Node}\ \mathit{Red}\ (\mathit{Node}\ \mathit{Black}\ t_1\ a_1\ t_2)\ a_2\ (\mathit{Node}\ \mathit{Black}\ t_3\ a_3\ t_4) \\
&\mathit{balance}\ c\ l\ a\ r \\
&\quad = \quad \mathit{Node}\ c\ l\ a\ r
\end{aligned}
$$


a_3
a_2
t_4
a_1
t_3
t_1
t_2
a_1
t_1
a_3
a_2
t_4
t_2
t_3
a_2
a_1
a_3
t_1
t_2
t_3
t_4
a_3
a_1
t_4
t_1
a_2
t_2
t_3
a_1
t_1
a_2
t_2
a_3
t_3
t_4

Ansatz #1: Analyse der naiven Methode

Die folgenden Bäume entstehen, wenn i Schlüssel beginnend mit dem größten in einen anfangs leeren Baum eingefügt werden ($1 \leqslant i \leqslant 8$).

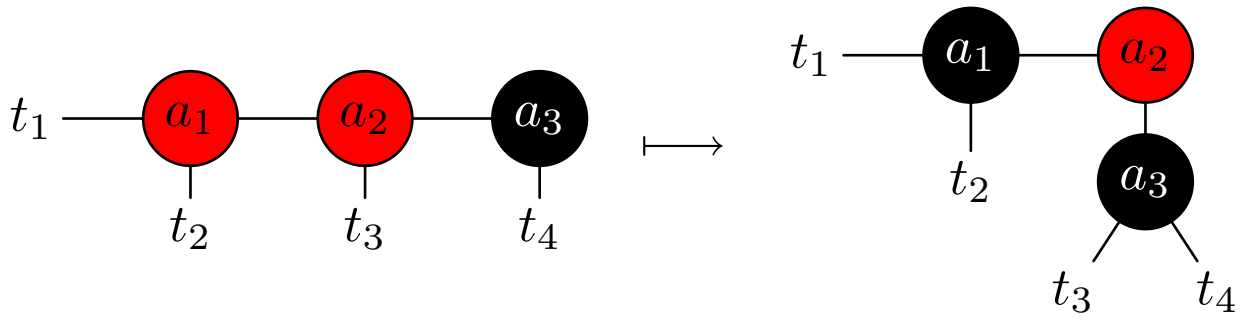


Beim Einfügen wird immer das *linke Rückgrat* des Baumes bis zum linkesten Blatt durchlaufen.

Ansatz #1: Analyse der naiven Methode

Beobachtung: Alle Knoten unterhalb des Rückgrats sind schwarz.

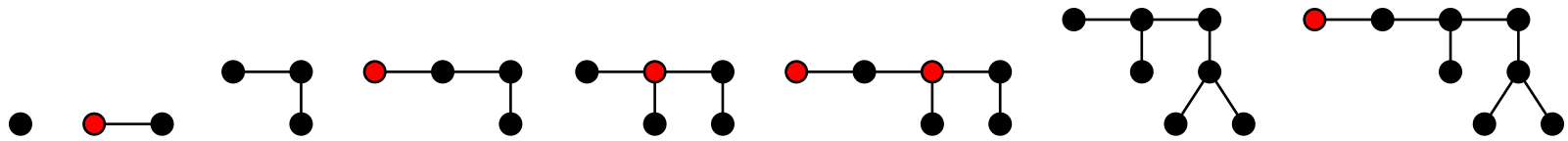
Betrachten wir noch einmal die Balancierungsoperation – zur Verdeutlichung zeichnen wir das linke Rückgrat waagerecht.



☞ Aufgrund der Schwarz-Bedingung müssen die Bäume unterhalb des Rückgrats vollständig ausgeglichen sein. Damit entsprechen die generierten Rot-schwarz-Bäume Listen sogenannter *Wimpel*.

Ansatz #1: Analyse der naiven Methode

Zeichnen wir die obigen Beispiele noch einmal neu.



Sei r die Höhe des rechtesten Wimpels. Aufgrund der Schwarz-Bedingung müssen Wimpel der Höhe i entweder ein- oder zweimal auftreten für alle i mit $0 \leqslant i \leqslant r$.

☞ Da ein Wimpel der Höhe r genau 2^r Schlüssel enthält, korrespondieren die generierten Rot-schwarz-Bäume zu Binärzahlen mit den Ziffern 1 und 2.

Ansatz #1: Das 1-2-Zahlensystem

Die Binärzahl $(b_0 \dots b_{n-1})_2$ bezeichnet die natürliche Zahl $\sum_{i=0}^{n-1} b_i 2^i$.

Jede natürliche Zahl hat eine eindeutige Darstellung im 1-2-Zahlensystem.

$$()_2, (1)_2, (2)_2, (11)_2, (21)_2, (12)_2, (22)_2, (111)_2, (211)_2 \dots$$

Binäres Inkrement:

$$\begin{aligned} add\ 1\ () &= 1 \\ add\ 1\ (1s) &= 2s \\ add\ 1\ (2s) &= 1(add\ 1\ s)\ . \end{aligned}$$

Ansatz #1: Verbesserung der naiven Methode

Die Analogie zum 1-2-Zahlensystem kann ausgenutzt werden, um die naive Methode zu verbessern (für den Fall, daß die Schlüssel geordnet vorliegen).

Aus den Ziffern werden Wimpel:

$$\begin{array}{lcl} \textbf{data}\ \mathit{Digit} & = & \mathit{One\ Key\ Tree} \\ & \mid & \mathit{Two\ Key\ Tree\ Key\ Tree}\ . \end{array}$$

Ein Rot-schwarz-Baum wird durch eine Liste von Ziffern dargestellt:

$$\textbf{type}\ \mathit{Tree}' \quad = \quad [\mathit{Digit}]\ .$$

Ansatz #1: Verbesserung der naiven Methode

Das Einfügen eines Schlüssels entspricht dem binären Inkrement.

$$
\begin{array}{lcl}
insert' & :: & Key \to Tree' \to Tree' \\
insert'\ a\ ps & = & add\ (One\ a\ Empty)\ ps \\
add\ (One\ a\ t)\ [\,] & = & [One\ a\ t] \\
add\ (One\ a_1\ t_1)\ (One\ a_2\ t_2 : ps) & = & Two\ a_1\ t_1\ a_2\ t_2 : ps \\
add\ (One\ a_1\ t_1)\ (Two\ a_2\ t_2\ a_3\ t_3 : ps) & & \\
\quad = \quad One\ a_1\ t_1 : add\ (One\ a_2\ (Node\ Black\ t_2\ a_3\ t_3))\ ps & &
\end{array}
$$

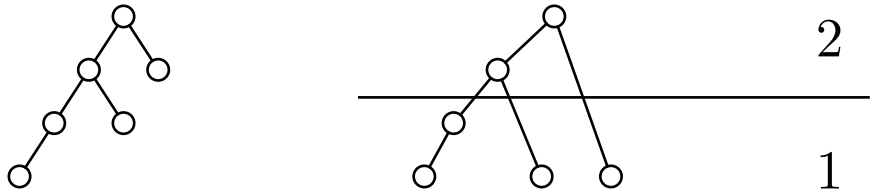
Wie bisher wird durch wiederholtes Einfügen ein Baum konstruiert (am Ende wird die Wimpelliste dann in einen Rot-schwarz-Baum überführt).

☞ Mithilfe der amortisierten Analyse läßt sich zeigen, daß der Algorithmus lineare Zeit benötigt.

Ansatz #2: Einfärben von Binärbäumen

Problem: Gegeben sei ein *beliebiger* Binärbaum. Kann dieser Baum so eingefärbt werden, daß ein Rot-schwarz-Baum entsteht?

Der erste Testfall:

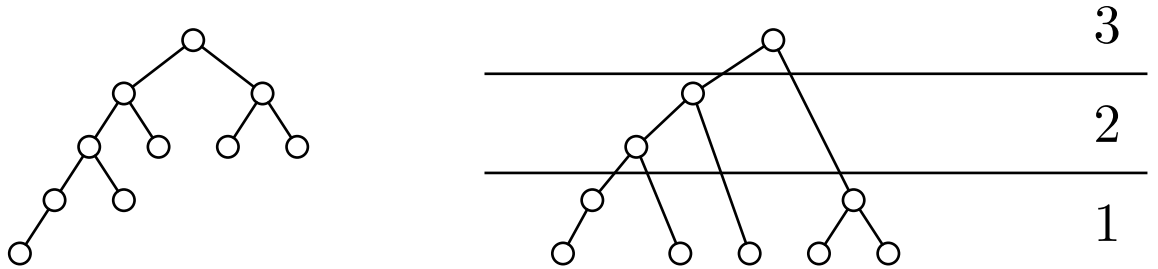


Auf der rechten Seite sind die Knoten entsprechend ihrer Höhe eingezeichnet. Wir annotieren die Knoten mit einer *Schichtnummer*: ein Knoten der Höhe h erhält die Schichtnummer $\lceil h \,/\, 2 \rceil$.

☞ Ein Knoten wird genau dann rot eingefärbt, wenn er einen Vorgänger mit der gleichen Schichtnummer hat.

Ansatz #2: Einfärben von Binärbäumen

Der zweite Testfall:



☞ Der rechte Nachfolger der Wurzel muß in die zweite Schicht gehoben werden (die Schichtnummer 1 wird zu 2 korrigiert).

Auf diese Weise läßt sich der Baum in einem Durchlauf einfärben. Hat ein *Blatt* eine korrigierte Schichtnummer > 1, dann ist ein Einfärben nicht möglich.

Eigenschaften der Algorithmen

☞ Der erste Algorithmus generiert Rot-schwarz-Bäume mit einer *minimalen* Anzahl roter Knoten (unter allen Rot-schwarz-Bäumen der gleichen Größe).

☞ Der zweite Algorithmus generiert Rot-schwarz-Bäume mit einer minimalen Schwarzhöhe.

☞ Wenn der Algorithmus auf linksvollständige Bäume angewendet wird, ergeben sich Rot-schwarz-Bäume mit einer *maximalen* Anzahl roter Knoten (unter allen Rot-schwarz-Bäumen der gleichen Größe).

Zur Historie: Der zweite Algorithmus verallgemeinert ein Verfahren von Bayer zur Einfärbung von AVL-Bäumen.

Resümee

- Bei der Entwicklung der Algorithmen haben wir verschiedene Einsichten in die Struktur von Rot-schwarz-Bäumen gewonnen (und deren Verwandtschaft zu anderen Baumklassen).

- Zahlensysteme sind ein ausgezeichnetes Hilfsmittel, um Datenstrukturen zu analysieren (und auch um neue Datenstrukturen zu entwickeln).

- Die optische Darstellung der Bäume spielt eine wichtige Rolle bei der Entwicklung der Algorithmen.

Anhang: Einfärben von Binärbäumen (Programm)

Wir nehmen an, daß die einzufärbenden Bäume als Elemente des folgenden Datentyps gegeben sind

$$\begin{array}{lcl} \textbf{type}\ \mathit{Level} & = & \mathit{Int} \\ \textbf{data}\ \mathit{LTree} & = & E \mid N\ \mathit{Level}\ \mathit{LTree}\ \mathit{Key}\ \mathit{LTree}\ , \end{array}$$

wobei $\lceil height\ (N\ h\ l\ a\ r)\ /\ 2 \rceil = h$.

Anhang: Einfärben von Binärbäumen (Programm)

```
rbtree                  ::  LTree → Tree
rbtree E                =   Empty
rbtree (N h l a r)      =   Node Black (make h l) a (make h r)

make                    ::  Level → LTree → Tree
make hp E
      | hp == 1         =   Empty
      | otherwise       =   error "not a red-black tree"
make hp (N h l a r)     =   Node color (make h' l) a (make h' r)
    where
    h'                  =   h `max` (hp - 1)
    color | hp == h     =   Red
          | otherwise   =   Black
```