

Algorithmen und Datenstrukturen

B7. Balancierte Bäume¹

Marcel Lüthi and Gabriele Röger

Universität Basel

10. April 2019

¹Folien basieren auf Vorlesungsfolien von Sedgewick & Wayne

<https://algs4.cs.princeton.edu/lectures/33BalancedSearchTrees-2x2.pdf>

Algorithmen und Datenstrukturen

10. April 2019 — B7. Balancierte Bäume^a

^aFolien basieren auf Vorlesungsfolien von Sedgewick & Wayne

<https://algs4.cs.princeton.edu/lectures/33BalancedSearchTrees-2x2.pdf>

B7.1 Einführung

B7.2 2-3 Bäume

B7.3 Rot-Schwarz Bäume

B7.1 Einführung

Informatiker des Tages : Donald Knuth



Donald E. Knuth

- ▶ Autor: "The art of computer programming"
 - ▶ Autor des Textsatzsystems \TeX
- ▶ Gewinner Turing Award (1974) und vieler anderer Preise
 - ▶ Arbeit an Analyse von Algorithmen
- ▶ Entwickelte erste Sprache für "Literate programming"

Knuth D. The art of computer programming 1: Fundamental algorithms 2: Seminumerical algorithms 3: Sorting and searching. MA: Addison-Wesley. 1968.

Balancierte Bäume

Implementation	suchen	Worst-case		Average-case		
		einfügen	löschen	suchen (hit)	einfügen	löschen
Verkettete Liste	N	N	N	$N/2$	N	$N/2$
Binäre suche	$\log_2(N)$	N	N	$\log_2(N)$	$N/2$	N
BST	N	N	N	$\log_2(N)$	$\log_2(N)$	\sqrt{N}
Ziel	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$

Frage

Können wir eine Implementation finden, bei der alle Operationen logarithmische Komplexität haben?

B7.2 2-3 Bäume

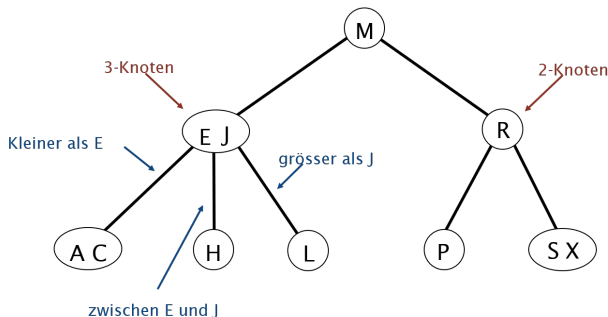
2-3 Bäume

Wir unterscheiden zwei Knotentypen

2-Knoten 1 Schlüssel, zwei Kinder

3-Knoten 2 Schlüssel, drei Kinder

- ▶ Wir verlangen **symmetrische Ordnung**
- ▶ Zusätzlich muss Baum **perfekt balanciert** sein.
 - ▶ Jeder Pfad von Wurzel zu Blatt hat dieselbe Länge.

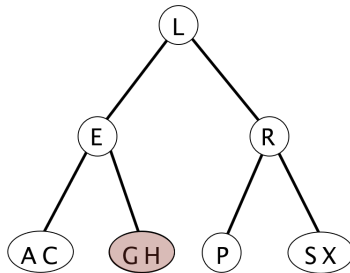
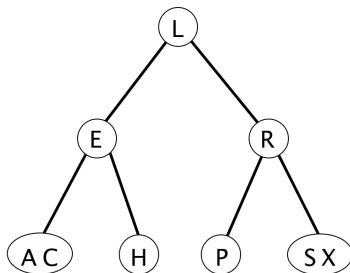


Einfügen in 2-3 Baum

Einfügen in 2-Knoten auf letzter Ebene

- ▶ Neuer Schlüssel zu 2-Knoten hinzufügen. Knoten wird zu 3-Knoten.

G Einfügen

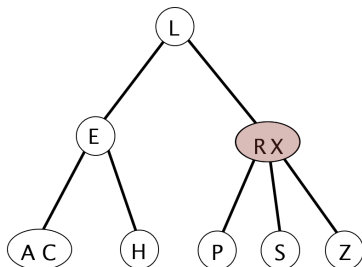
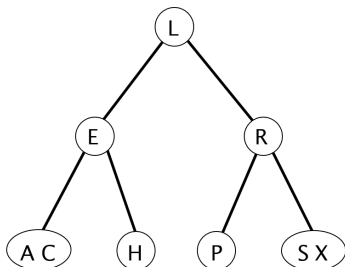


Einfügen in 2-3 Baum

Einfügen in 3-Knoten auf letzter Ebene

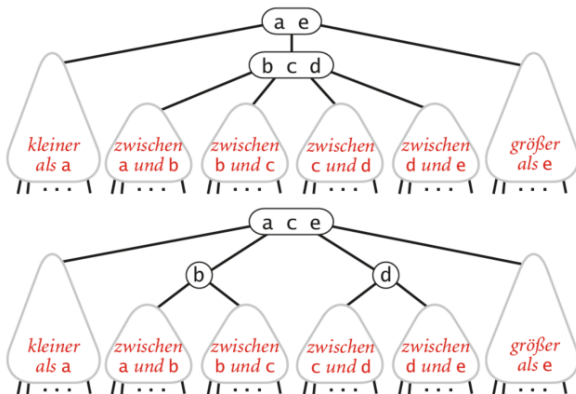
- ▶ Neuer Schlüssel zu 3-Knoten hinzufügen. Knoten wird temporär zu 4-Knoten.
- ▶ Mittlerer Knoten in Parent einfügen.
- ▶ Falls nötig, rekursiv fortsetzen.
- ▶ Falls Wurzel erreicht wird, und diese zu 4-Knoten wird, wird diese zu zwei 2-Knoten.

Z Einfügen



Lokale Transformationen

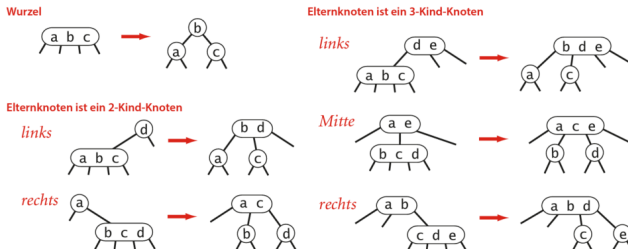
- ▶ Teilen eines 4 Knotens ist **lokale** Operation
 - ▶ Unterbäume nicht davon betroffen
 - ▶ Konstante Anzahl Operationen



Quelle: Abb. 3.30, Algorithmen, Wayne & Sedgewick

Globale Eigenschaften

- ▶ Invariante: Jede Operation belässt Baum perfekt balanciert.
- ▶ Ordnung der Teilbäume bleibt erhalten.



Quelle: Abb. 3.31, Algorithmen, Wayne & Sedgewick

2-3 Baum: Quiz: Performance

- Bäume sind perfekt balanciert!



Baumhöhe:

Worst Case

Best Case

Übersicht

Implementation	suchen	Worst-case		Average-case		
		einfügen	löschen	suchen (hit)	einfügen	löschen
Verkettete Liste	N	N	N	$N/2$	N	$N/2$
Binäre suche	$\log_2(N)$	N	N	$\log_2(N)$	$N/2$	N
Binärer Suchbaum	N	N	N	$\log_2(N)$	$\log_2(N)$	\sqrt{N}
2-3 Baum	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$

Problem

2-3 Bäume sind mühsam zu implementieren.

- ▶ Wir müssen viele Spezialfälle unterscheiden.
- ▶ Code wird unelegant und fehleranfällig.
- ▶ Elegante Lösung: Rot-Schwarz Bäume

B7.3 Rot-Schwarz Bäume

Informatiker des Tages : Robert Sedgewick



Robert Sedgewick

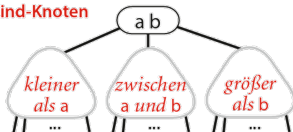
- ▶ Professor in Princeton
- ▶ Doktorand von Donald Knuth.
- ▶ "Erfinder" der Rot-Schwarz Bäume
- ▶ Autor von unserem Lehrbuch.

Guibas, Leo J., and Robert Sedgewick. "A dichromatic framework for balanced trees", IEEE Foundations of Computer Science, 1978.

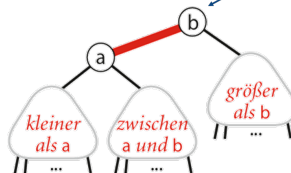
Rot-Schwarz Bäume: Idee

- ▶ 2-3 Baum wird als binärer Suchbaum repräsentiert
- ▶ 3-Knoten werden mit speziellen "roten" links markiert.

3-Kind-Knoten

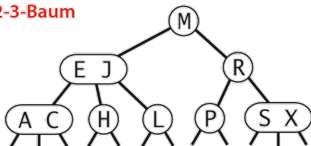


Grösster Knoten
wird zur Wurzel

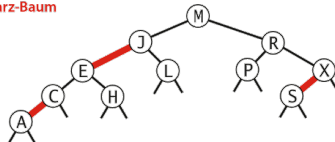


Quelle: Abb. 3.34, Algorithmen, Wayne & Sedgwick

2-3-Baum



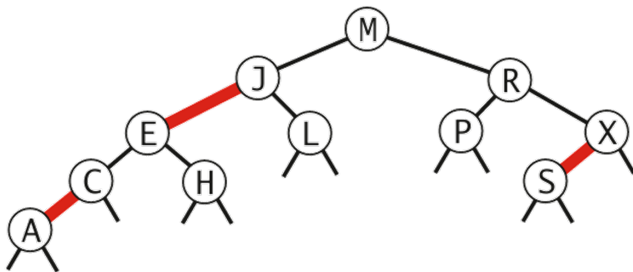
Rot-Schwarz-Baum



Rot-Schwarz Bäume - Definition

Ein Rot-Schwarz Baum ist ein binärer Suchbaum, mit der Eigenschaft:

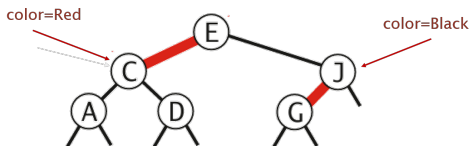
- ▶ Rote Referenzen zeigen nach links
- ▶ Von keinem Knoten gehen zwei rote Referenzen aus
 - ▶ (Keine 4-Knoten im 2-3 Baum)
- ▶ Jeder Pfad von der Wurzel zu einem Blatt hat die gleiche Anzahl von schwarzen Referenzen.
 - ▶ (Gleiche Tiefe im 2-3 Baum)



Repräsentation in Code

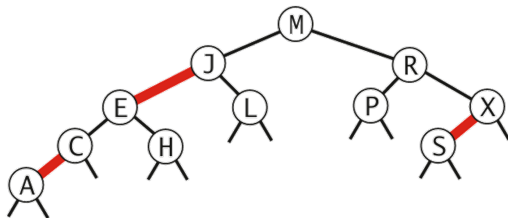
- ▶ Jeder Knoten hat genau eine Referenz von Parent
 - ▶ 1 Feld in Knoten genügt um Farbe speichern

```
class Node[Key, Value]:  
    Node(key : Key, value : Value)  
  
    key : Key  
    value : Value  
    left : Node[Key, Value]  
    right : Node[Key, Value]  
    color : Color # Red or Black
```



Suchen und ordnungsbasierte Operationen

- ▶ RB-Tree ist ein binärer Suchbaum - einfach mit Farbe
- ▶ Implementation von Suche und ordnungsbasierten Operationen bleibt gleich.
 - ▶ Farbe wird ignoriert.



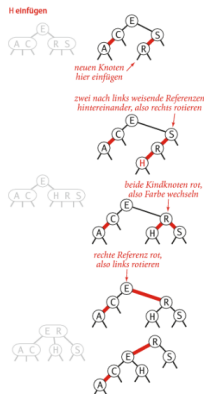
Quelle: Abb. 3.36, Algorithmen, Wayne & Sedgewick

Einfügen: Idee

Grundidee

Alle Operationen werden auf Operationen in entsprechendem 2-3 Baum zurückgeführt

- ▶ Neuer Link wird immer Rot
 - ▶ Führt zu potentiell 4 Knoten in 2-3 Baum
- ▶ Lokale Operationen um 2-3 Baum wiederherzustellen
 - ▶ Farb wechseln
 - ▶ Rotation links
 - ▶ Rotation rechts



Einfügen: Details

- ▶ Unterscheidung aller möglichen Fälle
- ▶ Pro Fall: Eigene Strategie um 2-3 Baum wiederherzustellen

Am besten in Ruhe selber lesen / anschauen

- ▶ Relevante Teile aus dem Buch auf Adam
 - ▶ Gute, schrittweise Erklärung mit Ablaufprotokoll
- ▶ Details nicht prüfungsrelevant

Animation:

<https://algs4.cs.princeton.edu/lectures/33DemoRedBlackBST.mov>

Einfügen: Implementation

► Trügerisch einfache Implementation

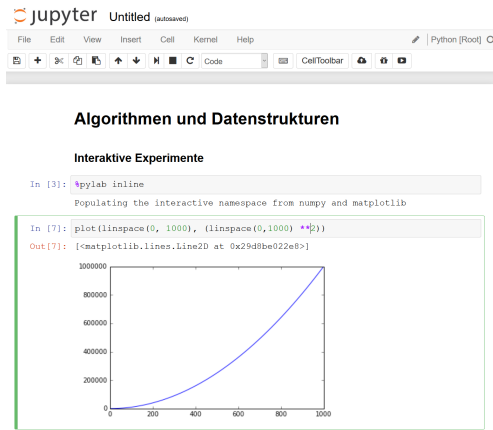
```
def _put(self, key, value, node):
    if (node == None):
        return RedBlackBST.Node(key, value, Color.RED, 1)
    elif key < node.key:
        node.left = self._put(key, value, node.left)
    elif key > node.key:
        node.right = self._put(key, value, node.right)
    elif key == node.key:
        node.value = value

    if self._isRed(node.right) and not self._isRed(node.left):
        node = self._rotateLeft(node)
    if self._isRed(node.left) and self._isRed(node.left.left):
        node = self._rotateRight(node)
    if self._isRed(node.left) and self._isRed(node.right):
        self._flipColors(node)

    node.count = 1 + self._size(node.left) + self._size(node.right)

    return node
```

Implementation



Jupyter-Notebook: RedBlackBST.ipynb

Analyse

Theorem

Die Höhe eines Rot-Schwarz-Baums mit N Knoten ist nicht höher als $2 \log_2(N)$.

Intuition:

- ▶ Jeder Pfad von Wurzel zu Blatt hat gleiche Anzahl von Schwarzen Referenzen
 - ▶ Korrespondenz mit 2-3 Baum
- ▶ Es gibt nie zwei rote Referenzen hintereinander.



Übersicht

Implementation	Worst-case			Average-case		
	suchen	einfügen	löschen	suchen (hit)	einfügen	löschen
Verkettete Liste	N	N	N	$N/2$	N	$N/2$
Binäre suche	$\log_2(N)$	N	N	$\log_2(N)$	$N/2$	N
Binärer Suchbaum	N	N	N	$\log_2(N)$	$\log_2(N)$	\sqrt{N}
Rot-Schwarz Baum	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$	$\log_2(N)$

Wir haben logarithmische Komplexität aller Operationen mit einer kleinen Konstante.