



2.5 Bäume



- 2.5.1 Binäre Suchbäume
- 2.5.2 Optimale Suchbäume
- 2.5.3 **Balancierte Bäume**
- 2.5.4 Skip-Listen
- 2.5.5 Union-Find-Strukturen

1  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Keine Zugriffswahrscheinlichkeiten gegeben => Versuche Baum balanciert zu halten!

Balancierte Bäume

- Nachteil bei normalen Suchbäumen:
Worst-case Aufwand ist $O(n)$
- Tritt bei degenerierten Bäumen auf
- Kann durch Balancierung verhindert werden
- **Problem:** Stelle Balance in $O(\log n)$ wieder her

2  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

logarithmische Höhe -> Zugriff in logarithmischer Zeit

Balancierte Bäume

- **Gewichtsbalance:** Für jeden Knoten unterscheidet sich die Anzahl der Knoten im linken und rechten Teilbaum um maximal eins.
- **Höhenbalance:** Für jeden Knoten unterscheidet sich die Höhe des linken und rechten Teilbaums um maximal eins.



Hier interessant: Höhenbalance! => Kurze Pfade!

Balancierte Bäume

- Zur Erinnerung:
 - Maximale Zahl von Knoten in einem Baum der Höhe h ist $O(2^h)$
 - Minimale Zahl von Knoten in einem Baum der Höhe h ist $\Omega(h)$
 - Minimale Zahl von Knoten in einem balancierten Baum der Höhe h ist $\Omega(2^{h/2})$



Balancierte Bäume

- Zur Erinnerung:
 - Minimale Höhe eines Baums mit n Knoten ist $\Omega(\log n)$
 - Maximale Höhe eines Baums mit n Knoten ist $O(n)$
 - Maximale Höhe eines balancierten Baums mit n Knoten ist $O(2 \times \log n) = O(\log n)$



2.5 Bäume

- 2.5.1 Binäre Suchbäume
- 2.5.2 Optimale Suchbäume
- 2.5.3 Balancierte Bäume
 - 2.5.3.1 AVL-Bäume
 - 2.5.3.2 Rot-Schwarz-Bäume
 - 2.5.3.3 B-Bäume
- 2.5.4 Skip-Listen
- 2.5.5 Union-Find-Strukturen



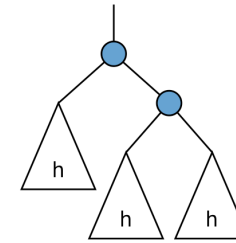
AVL: Adelson-Velski-Landis

AVL-Bäume

- Werden wie normale binäre Suchbäume behandelt
- Jeder Knoten speichert sein Ungleichgewicht (+1,0,-1)
- Nach Insert() oder Delete() Wiederherstellung der Balance durch **Rotation**



AVL-Bäume



AVL-Bäume

The diagram shows a binary tree structure. The root node is a blue circle. It has a left child and a right child, both also blue circles. The left child has a left subtree of height h . The right child has a left subtree of height h and a right subtree of height $h+1$. The subtrees are represented by triangles.

9

Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

AVL-Bäume

The diagram shows a binary tree structure after a simple rotation. The root node is a blue circle. It has a left child and a right child, both also blue circles. The left child has a left subtree of height h . The right child has a left subtree of height h and a right subtree of height $h+1$. The subtrees are represented by triangles.

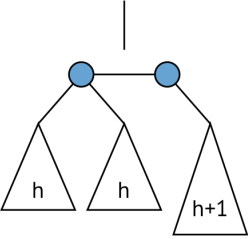
Einfache
Rotation

10

Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

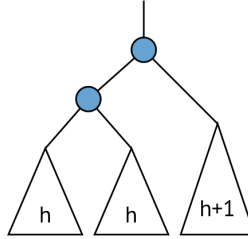
AVL-Bäume



Einfache
Rotation

11 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

AVL-Bäume

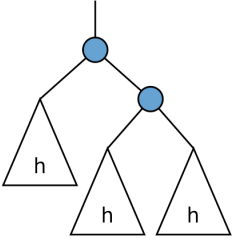


Einfache
Rotation



12 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Imbalance wird durch Rotation wieder ausgeglichen.

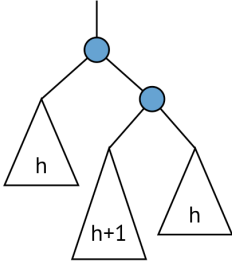
AVL-Bäume





The diagram shows a binary tree with a root node (blue circle) and a left child (blue circle). The root's left child has a left child (triangle labeled 'h'). The root's right child has a left child (triangle labeled 'h') and a right child (triangle labeled 'h').

13  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

AVL-Bäume



The diagram shows a binary tree with a root node (blue circle) and a left child (blue circle). The root's left child has a left child (triangle labeled 'h'). The root's right child has a left child (triangle labeled 'h+1') and a right child (triangle labeled 'h').

14  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

AVL-Bäume

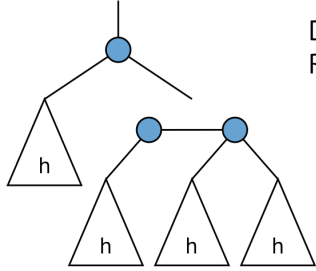
15 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

AVL-Bäume

Doppelte
Rotation

16 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer


AVL-Bäume



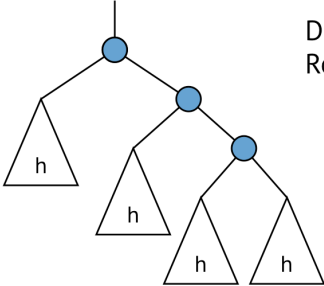
Doppelte
Rotation

17

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer


 FRIEDRICH-ALEXANDER
UNIVERSITÄT


AVL-Bäume



Doppelte
Rotation

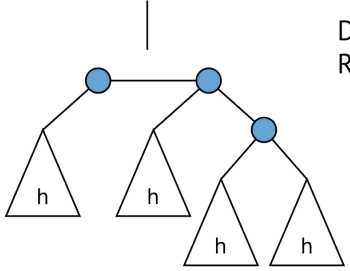
18

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer


 FRIEDRICH-ALEXANDER
UNIVERSITÄT

1. Rotation: Bringe „Problem“ nach außen


AVL-Bäume



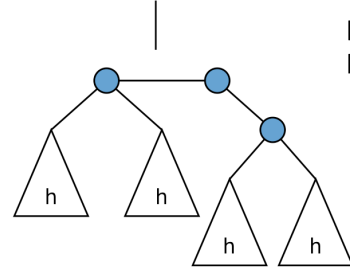
Doppelte
Rotation

19

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




AVL-Bäume

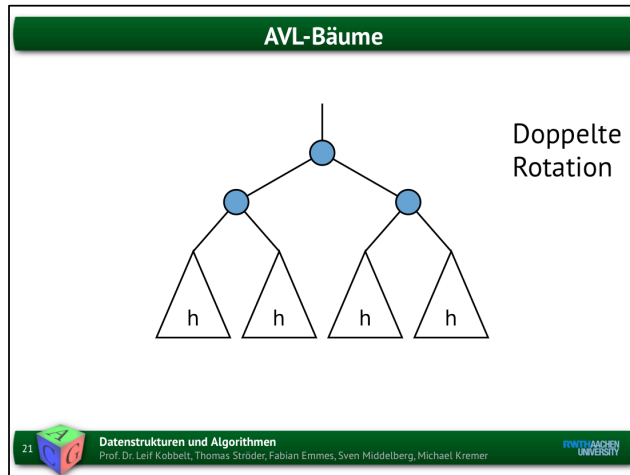


Doppelte
Rotation

20

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





2. Rotation: Behebe „Problem“. Beachte: Es sind zwei Rotationen nötig, um Imbalancezustand in einen Balancezustand zu überführen.

AVL-Bäume

- Rotationen müssen ggf. nach oben propagiert werden
- Im schlechtesten Fall $O(\log n)$ Rotationen bei Delete()
- Nur praktikabel, wenn gesamte Datenstruktur im Hauptspeicher

22 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

2.5 Bäume

- 2.5.1 Binäre Suchbäume
- 2.5.2 Optimale Suchbäume
- 2.5.3 Balancierte Bäume
 - 2.5.3.1 AVL-Bäume
 - 2.5.3.2 Rot-Schwarz-Bäume
 - 2.5.3.3 B-Bäume
- 2.5.4 Skip-Listen
- 2.5.5 Union-Find-Strukturen



Rot-Schwarz-Bäume

- Binärer Suchbaum, jeder Knoten hat zwei Söhne oder keinen
- Rote und schwarze Knoten
- Re-Balancing durch Rotationen und Umfärben



Verwendung z.B. in C++-STL

Rot-Schwarz-Bäume

- Konsistenzregeln
 1. Jeder Knoten ist rot oder schwarz
 2. Die Wurzel ist schwarz
 3. Die Blätter sind schwarz
 4. Die Söhne eines roten Knotens sind schwarz.
 5. Alle Pfade von einem Knoten zu den nachfolgenden Blättern haben gleich viele schwarze Knoten

25

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Z.B.: Perfekt balancierter Suchbaum mit nur schwarzen Knoten

Rot-Schwarz-Bäume



Blätter werden im Folgenden nicht mehr dargestellt.

26

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rot-Schwarz-Bäume

- Eigenschaften
 - Maximales Pfadlängenverhältnis 2:1
 - Pfadlängen $\Theta(\log n)$



27  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  FRIEDRICH-ALEXANDER
UNIVERSITÄT

Kürzester Pfad: Nur schwarze Knoten

Längster Pfad: Füge zwischen zwei schwarzen Knoten jeweils einen roten ein

Rot-Schwarz-Bäume

- Lemma:
Die Höhe eines Rot-Schwarz-Baums
mit n inneren Knoten ist höchstens
 $2 \times \lceil \lg(n+1) \rceil$

28  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  FRIEDRICH-ALEXANDER
UNIVERSITÄT

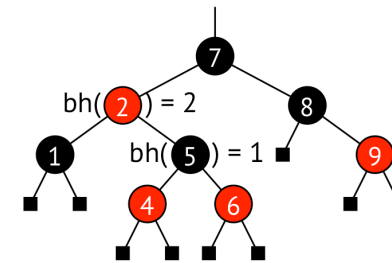
Rot-Schwarz-Bäume

- Die Zahl der schwarzen Knoten auf einem (dann jeden) Pfad von einem Knoten x (ausschließlich) bis zu einem Blatt (einschließlich) sei $bh(x)$



$bh(x)$: Anzahl der schwarzen Knoten auf allen Pfaden ab Knoten x

Rot-Schwarz-Bäume



Rot-Schwarz-Bäume

- Der Baum unter einem Knoten x enthält mindestens $2^{bh(x)}-1$ innere Knoten
 - I.A. x ist ein Blatt $\rightarrow 2^{bh(x)}-1=2^0-1=0$
 - I.V. Die Beh. gelte für alle y mit $height(y) < height(x)$
 - I.S. x sei ein innerer Knoten mit Söhnen x_1, x_2
 - $\rightarrow bh(x_1), bh(x_2) \geq bh(x)-1$
 - \rightarrow der Baum unter x enthält mindestens $(2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)+1 = 2^{bh(x)}-1$ innere Knoten



Rot-Schwarz-Bäume

- Es sei $h = height(root)$ die Höhe des Baumes
 - Mindestens die Hälfte der Knoten auf einem Pfad von der Wurzel zu einem Blatt müssen schwarz sein.
 - $\rightarrow bh(root) \geq h / 2$
 - $\rightarrow n \geq 2^{h/2}-1$
 - $\rightarrow h \leq 2 \times \lceil \lg(n+1) \rceil$



Rot-Schwarz-Bäume

- Insert(), Delete()
 - Wie bei normalen binären Suchbäumen mit anschließender Wiederherstellung der Konsistenzbedingungen
 - Insert() → drei verschiedene Fälle
 - Delete() → vier verschiedene Fälle
 - Pseudo-Code: siehe Cormen et al.



Rot-Schwarz-Bäume: Insert()

- Jeder eingefügte Knoten wird zunächst rot gefärbt
- Bei Wiederherstellung durch Umfärben oder Rotation kann genau eine weitere Konsistenzverletzung auftreten.
- Propagiere Modifikation nach oben.



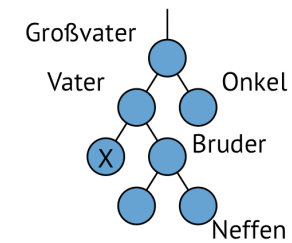
Onkel rot: Fall 1; Onkel schwarz + innen: Fall 2; Onkel schwarz + außen: Fall 3

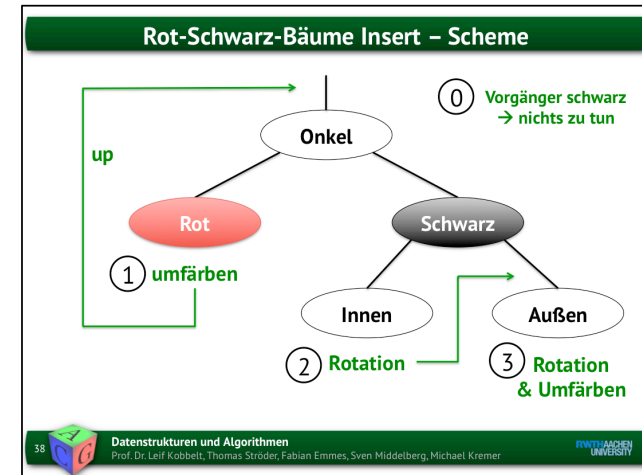
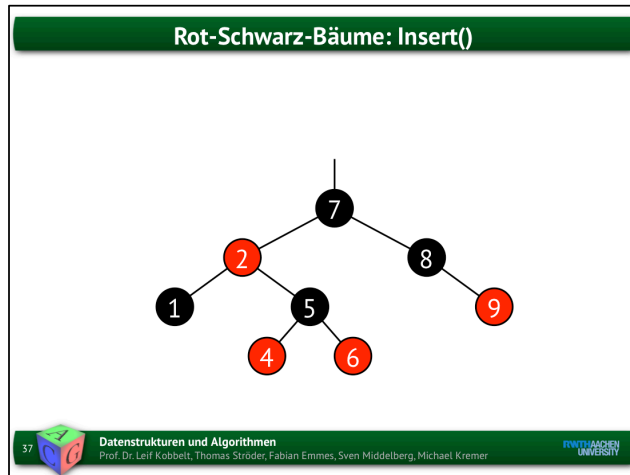
Rot-Schwarz-Bäume: Insert()

- Mögliche Konsistenzverletzungen
 - Neuer Knoten ist Wurzel
 - Neuer Knoten ist Nachfolger eines roten Knotens
- Fallunterscheidung nach der Farbe des **Onkels**
 - Konsistenzregeln
 1. Jeder Knoten ist rot oder schwarz.
 2. Die Wurzel ist schwarz.
 3. Die Blätter sind schwarz.
 4. Die Söhne eines roten Knotens sind schwarz.
 5. Alle Pfade von einem Knoten zu den nachfolgenden Blättern haben gleich viele schwarze Knoten.



Rot-Schwarz-Bäume: Insert()

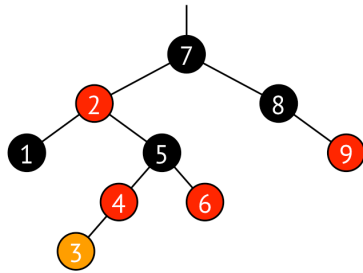




Minimiere erwartete Anzahl der Knotenzugriffe

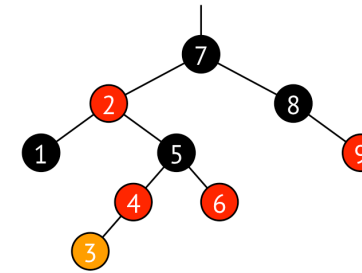
Rot-Schwarz-Bäume: Insert()

Einfügen zunächst wie bei normalen binären Suchbäumen



Rot-Schwarz-Bäume: Insert()

Fall 1: Der Onkel ist rot



Rot-Schwarz-Bäume: Insert()

Fall 1: Der Onkel ist rot → umfärben

41 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rot-Schwarz-Bäume: Insert()

Fall 1: Der Onkel ist rot → umfärben und weiter mit Großvater

42 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rot-Schwarz-Bäume: Insert()

Fall 1: Der Onkel ist rot → umfärben und weiter mit Großvater

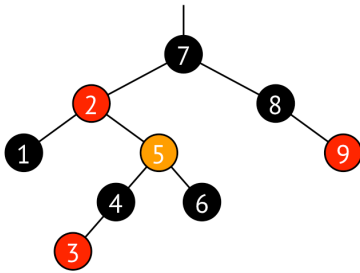
43 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rot-Schwarz-Bäume: Insert()

44 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

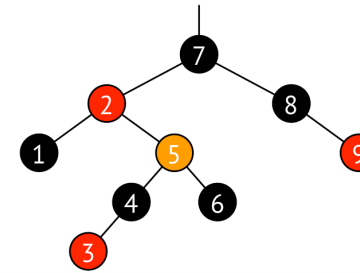
Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn



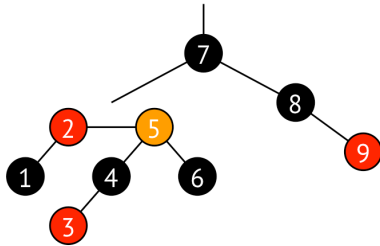
Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn
→ Rotation nach außen



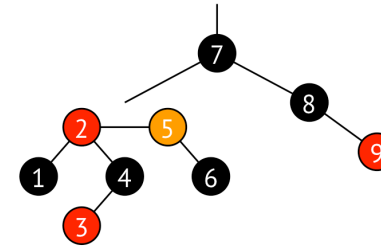
Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn
→ Rotation nach außen



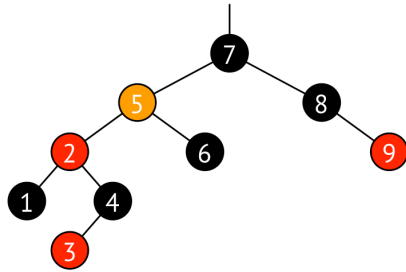
Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn
→ Rotation nach außen



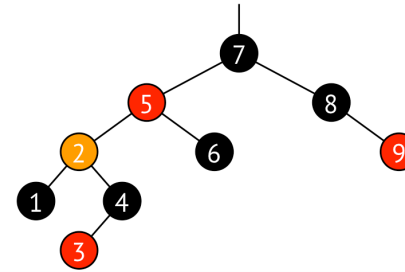
Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn
→ Rotation nach außen



Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn
→ Rotation nach außen und weiter mit äußeren Sohn (Fall 3)



Rot-Schwarz-Bäume: Insert()

Fall 2: Der Onkel ist schwarz und ● ist ein innerer Sohn
 → Rotation nach außen und weiter mit äußeren Sohn (Fall 3)

51 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

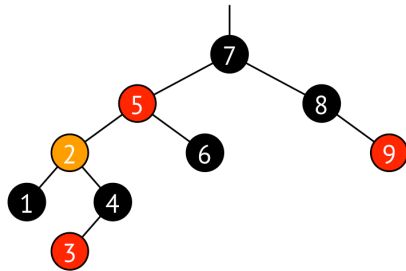
Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn

52 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn
→ Rotation nach innen



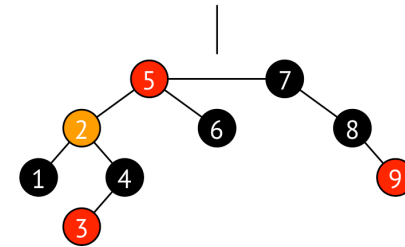
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FWTH AACHEN
UNIVERSITY

Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn
→ Rotation nach innen



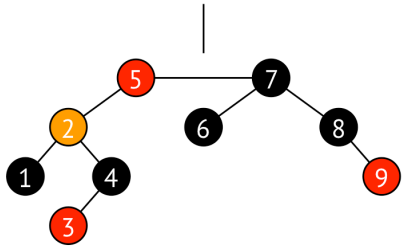
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FWTH AACHEN
UNIVERSITY

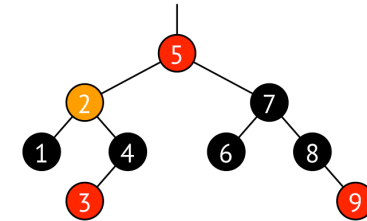
Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn
→ Rotation nach innen



Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn
→ Rotation nach innen



Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn
 → Rotation nach innen und umfärben

57 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Intuition für Umfärbung: Schwarze Marke bleibt oben liegen

Rot-Schwarz-Bäume: Insert()

Fall 3: Der Onkel ist schwarz und ● ist ein äußerer Sohn
 → Rotation nach innen und umfärben

58 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Fallunterscheidung Insert()

1. Onkel ist rot
 - Umfärben, Rekursion nach oben
2. Sohn < Vater > Großvater oder Sohn > Vater < Großvater
 - Rotation, weiter mit Fall 3
3. Sohn < Vater < Großvater oder Sohn > Vater > Großvater
 - Rotation, Umfärben, Fertig



Aufwand Insert()

- Suchen der Einfügestelle $O(\log n)$
- Einfügen $O(1)$
- Konsistenzwiederherstellung
 - Umfärbe-Schritte $O(\log n)$
(Schritt vom Enkel zum Großvater)
 - Rotationen $O(1)$ (maximal zwei)
- Insgesamt: $O(\log n)$



maximal 2 Rotationen, da nach Fall 3 das Problem ganz gelöst ist (keine Propagation von rot-rot Kollisionen möglich, da oberster Knoten dann schwarz)

Rot-Schwarz-Bäume: Delete()

- Wie bei normalen binären Suchbäumen wird der Knoten entweder direkt gelöscht oder sein Successor wird gelöscht und der Inhalt wird kopiert.
- Konsistenzwiederherstellung
 - Wenn der Knoten rot war, ist nichts zu tun
 - Wenn der Knoten schwarz war, gibt es **4** Fälle



Bruder rot: Fall 1; Bruder schwarz + Neffen beide schwarz: Fall 2; Bruder schwarz + innerer Neffe rot + äußerer Neffe schwarz: Fall 3; Bruder schwarz + äußerer Neffe rot (innerer Neffe egal): Fall 4

Rot-Schwarz-Bäume: Delete()

- Falls der **tatsächlich** entfernte Knoten schwarz ist, fehlt auf allen Pfaden innerhalb des entsprechenden Teil-Baums eine schwarze Marke
- Weise diese schwarze Marke dem (einen) Nachfolger des gelöschten Knotens zu. Dadurch wird dieser **schwarz-rot** oder **doppel-schwarz**.



Rot-Schwarz-Bäume: Delete()

- Erinnerung

63 Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Knoten mit nur einem Nachfolger: Wir haben die Blätter ausgeblendet! Tatsächlich haben alle Knoten zwei oder null Nachfolger, aber die Blätter enthalten keine Informationen und können daher beliebig gelöscht oder eingefügt werden, um diese Bedingung wiederherzustellen. Nur ein Nachfolger bezieht sich also auf „nur ein Nicht-Blatt-Nachfolger“.

Rot-Schwarz-Bäume: Delete()

- Erinnerung

64 Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Rot-Schwarz-Bäume: Delete()

- Erinnerung

65 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer **FRIEDRICH-ALEXANDER
UNIVERSITÄT**

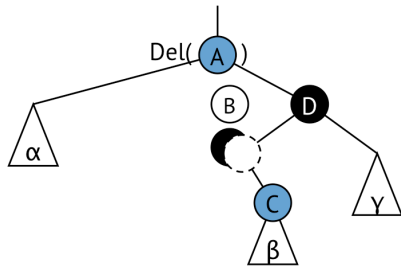
Rot-Schwarz-Bäume: Delete()

- Erinnerung

66 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer **FRIEDRICH-ALEXANDER
UNIVERSITÄT**

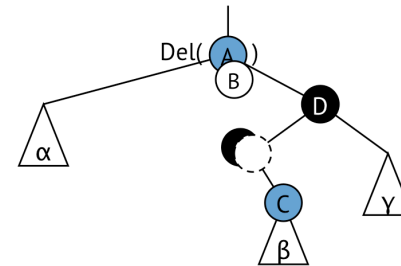
Rot-Schwarz-Bäume: Delete()

- Erinnerung



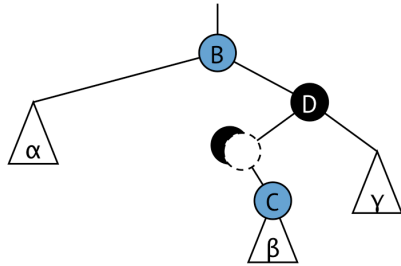
Rot-Schwarz-Bäume: Delete()

- Erinnerung



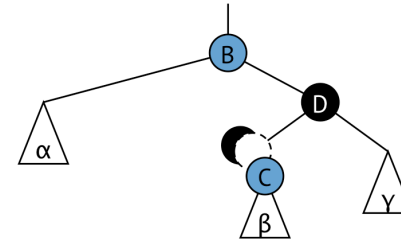
Rot-Schwarz-Bäume: Delete()

- Erinnerung



Rot-Schwarz-Bäume: Delete()

- Erinnerung



Rot-Schwarz-Bäume: Delete()

- Erinnerung

71 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer FRIEDRICH-ALEXANDER
UNIVERSITÄT

Rot-Schwarz-Bäume: Delete()

- Erinnerung

72 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer FRIEDRICH-ALEXANDER
UNIVERSITÄT

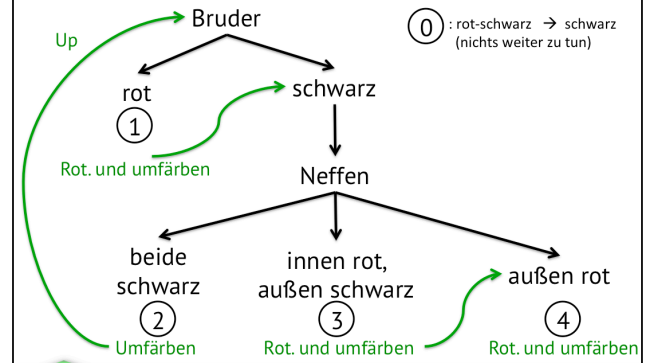
Bei schwarzen Knoten müssen wir aufpassen, weil die Anzahl der schwarzen Knoten in jedem Teilbaum gleich sein muss.

Rot-Schwarz-Bäume: Delete()

- Umfärben und Rotationen
 - Erhalte die Zahl schwarzer Marken der Sub-Bäume
- Abbruchbedingungen
 - Ein **schwarz-roter** Knoten wird **schwarz** gefärbt
 - Ein Wurzelknoten wird erreicht (**doppel-schwarz** kann einfach wegfallen, da die Wurzel auf allen Pfaden liegt)

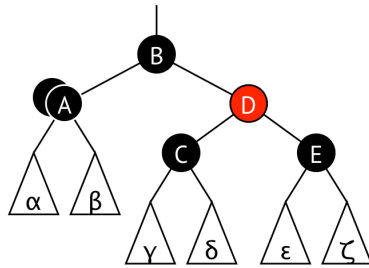


Rot-Schwarz-Bäume – Delete - Scheme



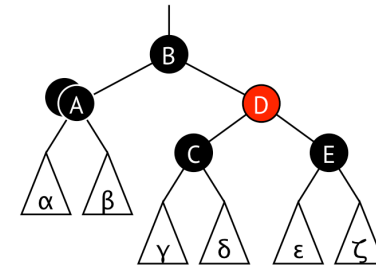
Rot-Schwarz-Bäume: Delete()

Fall 1: Der Bruder ist rot



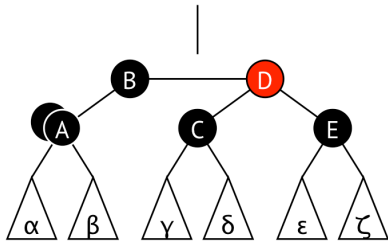
Rot-Schwarz-Bäume: Delete()

Fall 1: Der Bruder ist rot → Rotation nach links



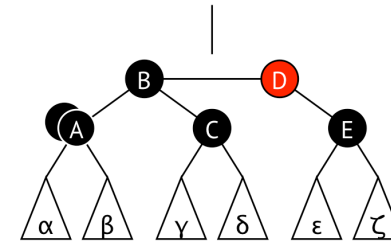
Rot-Schwarz-Bäume: Delete()

Fall 1: Der Bruder ist rot → Rotation nach links




Rot-Schwarz-Bäume: Delete()

Fall 1: Der Bruder ist rot → Rotation nach links



Rot-Schwarz-Bäume: Delete()


Fall 1: Der Bruder ist rot → Rotation nach links


Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Es gibt nur zwei Elementare Operationen: umfärben und rotieren.

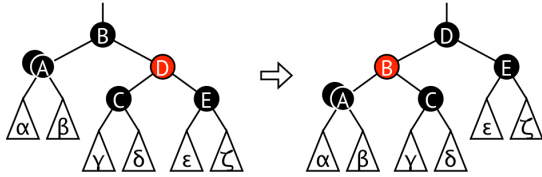
Rot-Schwarz-Bäume: Delete()

Fall 1: Der Bruder ist rot → Rotation nach links und umfärben


Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rot-Schwarz-Bäume: Delete()

Fall 1: Der Bruder ist rot → Rotation nach links und umfärben

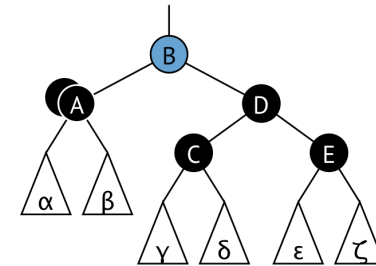


weiter mit Fall 2,3,4: der Bruder ist **nicht** rot



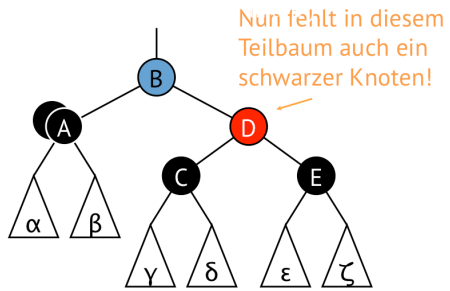
Rot-Schwarz-Bäume: Delete()

Fall 2: Der Bruder ist schwarz und dessen Söhne sind beide schwarz



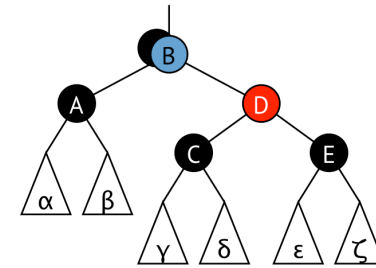
Rot-Schwarz-Bäume: Delete()

Fall 2: Der Bruder ist schwarz und dessen Söhne sind beide schwarz
→ umfärben



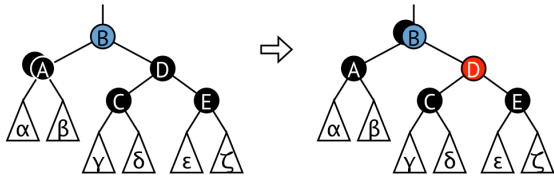
Rot-Schwarz-Bäume: Delete()

Fall 2: Der Bruder ist schwarz und dessen Söhne sind beide schwarz
→ umfärben und Marke nach oben weitergeben



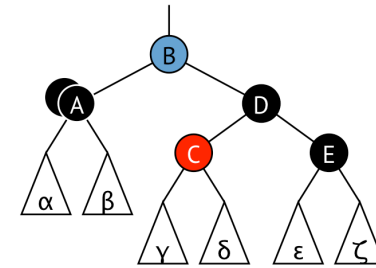
Rot-Schwarz-Bäume: Delete()

Fall 2: Der Bruder ist schwarz und dessen Söhne sind beide schwarz
→ umfärben und Marke nach oben weitergeben
(Ende, falls "B" rot war, sonst weiter mit "B")



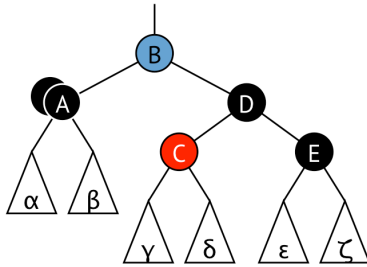
Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot
und dessen äußerer Sohn ist schwarz



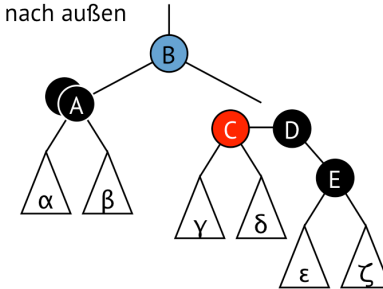
Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot und dessen äußerer Sohn ist schwarz
→ Rotation nach außen



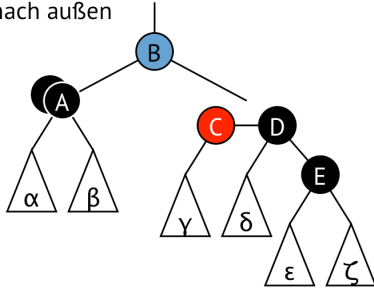
Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot und dessen äußerer Sohn ist schwarz
→ Rotation nach außen



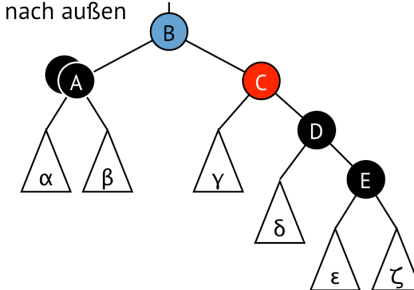
Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot und dessen äußerer Sohn ist schwarz
→ Rotation nach außen



Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot und dessen äußerer Sohn ist schwarz
→ Rotation nach außen



Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot und dessen äußerer Sohn ist schwarz
 → Rotation nach außen und umfärben

91 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer FRIEDRICH-ALEXANDER
UNIVERSITÄT

Rot-Schwarz-Bäume: Delete()

Fall 3: Bruder ist schwarz, dessen innerer Sohn ist rot und dessen äußerer Sohn ist schwarz
 → Rotation nach außen und umfärben

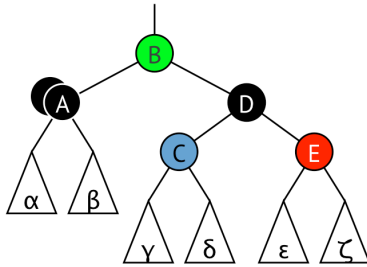
... weiter mit **Fall 4**

92 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer FRIEDRICH-ALEXANDER
UNIVERSITÄT

Jetzt haben wir Fall 3 in Fall 4 überführt.

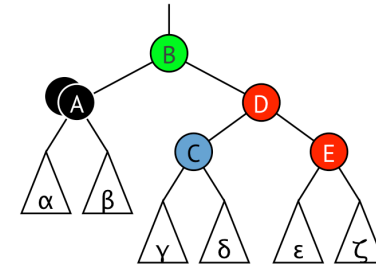
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot



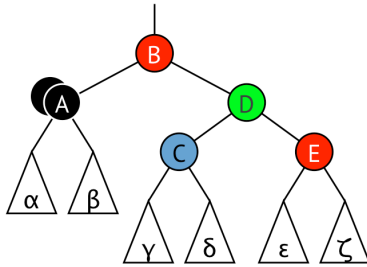
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot



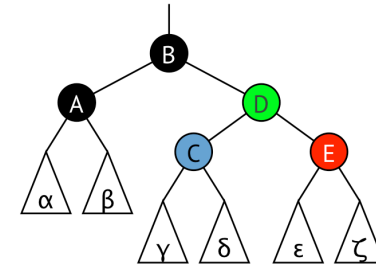
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben



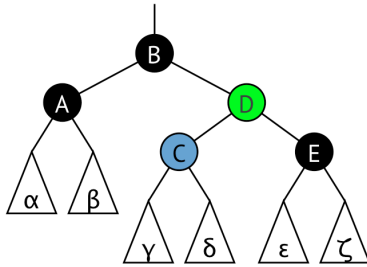
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben



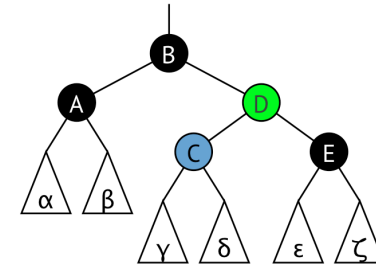
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben



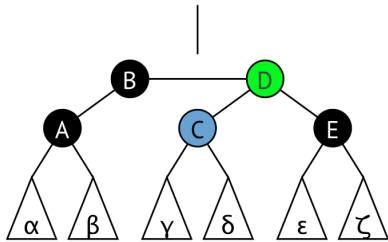
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben und Rotation nach innen



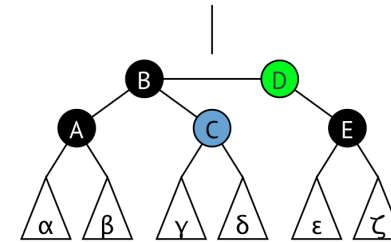
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben und Rotation nach innen



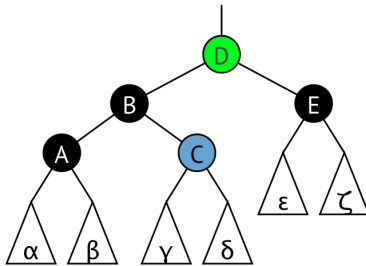
Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben und Rotation nach innen



Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben und Rotation nach innen



101



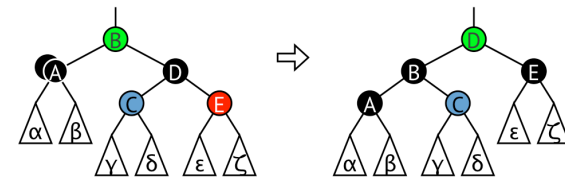
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

UNIVERSITÄT
DUISBURG
ESSEN

Rot-Schwarz-Bäume: Delete()

Fall 4: Der Bruder ist schwarz, dessen äußerer Sohn ist rot
→ umfärben und Rotation nach innen



102





Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

UNIVERSITÄT
DUISBURG
ESSEN

Aufwand Delete()



- Suchen des Löschknotens $O(\log n)$
- Löschen $O(1)$
- Konsistenzwiederherstellung
 - Umfärbe-Schritte $O(\log n)$
 - Rotationen $O(1)$ (maximal drei)
- Insgesamt: $O(\log n)$

103  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  FRIEDRICH-ALEXANDER
UNIVERSITÄT

Maximal 3 Rotationen, da die Folge Fall 1, Fall 2 zur Lösung aller Konflikte führt (der oberste Knoten, der die schwarze Marke erhält, muss rot sein und damit sind alle Konflikte durch Schwarzfärbung dieses Knotens zu beheben) und ansonsten die maximale Fallfolge Fall 1, Fall 3 und Fall 4 mit jeweils einer Rotation ist (nach Fall 4 ist sowieso alles gelöst)

2.5 Bäume

- 2.5.1 Binäre Suchbäume
- 2.5.2 Optimale Suchbäume
- 2.5.3 Balancierte Bäume
 - 2.5.3.1 AVL-Bäume
 - 2.5.3.2 Rot-Schwarz-Bäume
 - 2.5.3.3 B-Bäume
- 2.5.4 Skip-Listen
- 2.5.5 Union-Find-Strukturen

104  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  FRIEDRICH-ALEXANDER
UNIVERSITÄT

B-Bäume

- Alternative Interpretation/Implementierung der Rot-Schwarz-Bäume führt auf 2,4-Bäume
- Diese sind ein Spezialfall des allgemeineren Konzepts von B-Bäumen
- Vorteil: größere Mengen von Knoten werden zusammengefasst und passen dadurch besser in einen Festplatten-Block
- **Hysterese:** Nicht in jedem Schritt muss rebalanciert werden.



Speicherhierarchie

- Mit zunehmendem Abstand vom Prozessor ...
 - ... steigt die Speicherkapazität
 - ... wächst die zugreifbare Blockgröße
 - ... sinkt die Zugriffsgeschwindigkeit



Speicherhierarchie

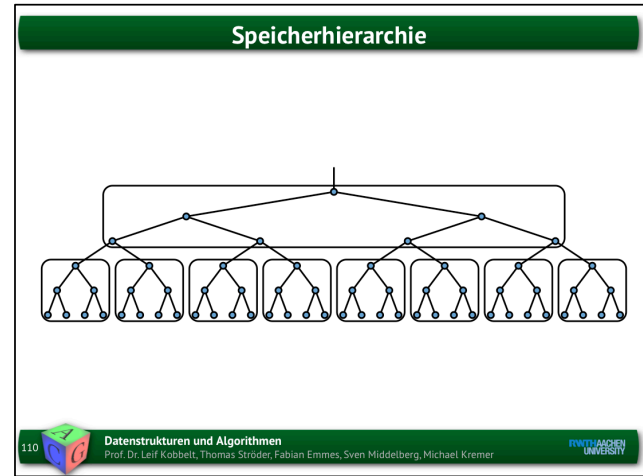
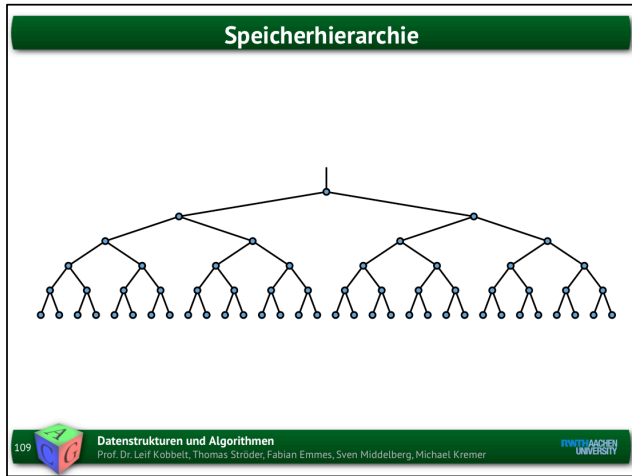
- Effiziente Algorithmen ...
 - ... führen häufige Berechnungen auf kleinen Datenmengen durch ("innere Schleife")
 - ... minimieren die Zugriffe auf externe Speicherebenen
 - ... passen die Größe der Datenstrukturen an die jeweiligen Blockgrößen an

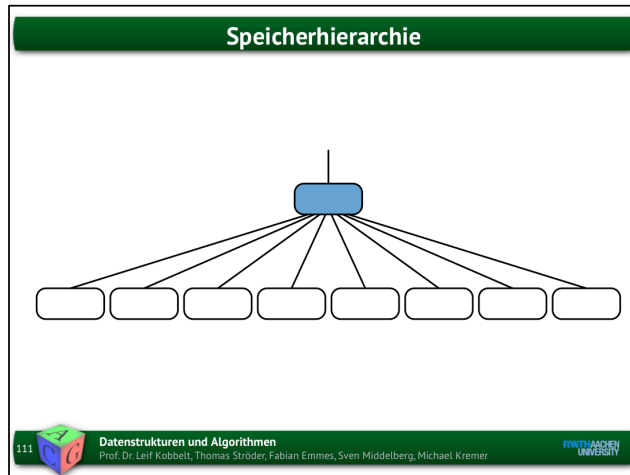


Speicherhierarchie

- Bei der Suche in **externen** Binärbaumstrukturen hängt die Zugriffszeit im Wesentlichen von der Verteilung der Knoten auf Festplatten-Sektoren ab.
- Fasse Teilbäume in Sektoren zusammen







B-Bäume können so konfiguriert werden, dass sie an die Anforderungen spezifischer Hardware angepasst werden können (z.B. bestimmte Speicherblockgröße optimal ausfüllen).

Definition : B-Bäume

- Jeder Knoten x hat die folgenden Felder
 - Die Zahl $n[x]$ der aktuell im Knoten gespeicherten Schlüssel ("Füllgrad")
 - $n[x]$ Schlüssel
 $key_1[x] \leq \dots \leq key_{n[x]}[x]$
 - $n[x]+1$ Zeiger auf die Söhne
 $c_0[x], c_1[x], \dots, c_{n[x]}[x]$
 - Der Bool-Wert $leaf[x]$ zeigt an, ob x ein Blatt ist

The slide contains a green header bar with the text 'Definition : B-Bäume' and a footer bar with the text 'Datenstrukturen und Algorithmen' and 'FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG'.

Definition : B-Bäume

- Bedingungen
 - Ist k_i ein Schlüssel im Unterbaum mit Wurzel $c_i[x]$ so ist
$$k_0 \leq \text{key}_1[x] \leq k_1 \leq \text{key}_2[x] \leq \dots \leq \text{key}_{n[x]}[x] \leq k_{n[x]}$$
 - Alle Blätter haben dieselbe Tiefe h



Definition : B-Bäume

- Bedingungen (Fortsetzung)
 - Es gibt eine feste Zahl $t \geq 2$, den minimalen Grad, mit
 - Jeder Knoten *außer der Wurzel* enthält mindestens $t-1$ **Schlüssel**
 - Jeder Knoten enthält höchstens $2 \times t - 1$ **Schlüssel**



Definition : B-Bäume

- Bedingungen (Fortsetzung, alternativ)
 - Es gibt eine feste Zahl $t \geq 2$, den minimalen Grad, mit
 - Jeder Knoten *außer der Wurzel* hat mindestens t **Söhne**
 - Jeder Knoten hat höchstens $2 \times t$ **Söhne**

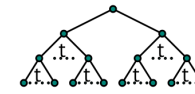


B-Bäume

- Für die Höhe h eines B-Baums mit n Schlüsseln gilt

$$h \leq \log_t \frac{n+1}{2}$$

#Knoten Tiefe	$n \geq 1 + (t-1) \sum_{i=0}^{h-1} 2t^i$ $= 1 + 2(t-1) \frac{t^h - 1}{t-1}$ $= 2t^h - 1$
------------------	--



B-Bäume

- Halte den Wurzelknoten immer im Hauptspeicher (da für jeden Zugriff notwendig)
- Erwartete Zugriffe $\approx \log_t n$
- Beispiel
 - 10^8 Telefonnummern in Deutschland
 - Blockgröße 512
 - maximal 2 Festplattenzugriffe



B-Bäume

- Zusammenhang mit Rot-Schwarz-Bäumen
 - Jeder schwarze Knoten absorbiert seine roten Söhne
 - Minimaler Füllgrad: $t = 2$ Söhne
 - Maximaler Füllgrad: $2t = 4$ Söhne
 - (2,4)-Bäume

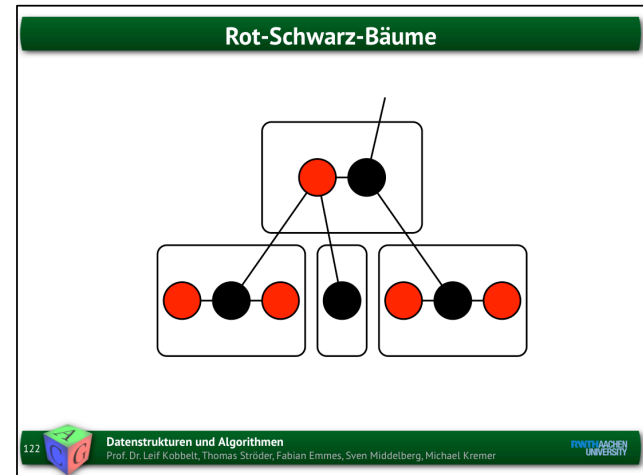
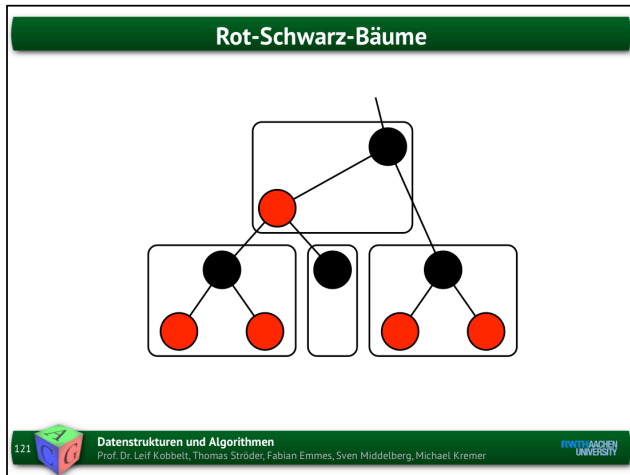


Rot-Schwarz-Bäume

119 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Rot-Schwarz-Bäume

120 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





Rot-Schwarz-Bäume

The diagram shows a root node with two circles, one red and one black. It has three children. The left child has three circles (red, black, red). The middle child has one black circle. The right child has three circles (red, black, red).

123  **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



Rot-Schwarz-Bäume

The diagram shows a root node with two circles, one red and one black. It has three children. The left child has three circles (red, black, red). The middle child has one black circle. The right child has three circles (red, black, red).

124  **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



Rot-Schwarz-Bäume

The diagram shows a tree node (black circle) with two children (red circles). To its right is a bitset representation in a box: [1, 0, 1], where the first and third positions are red and the second is black.

125  **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



Rot-Schwarz-Bäume

The diagram shows a tree node (black circle) with a red left child and a black right child. To its right is a bitset representation in a box: [1, 0], where the first position is red and the second is black.

126  **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



Rot-Schwarz-Bäume

The diagram shows two binary tree nodes. The left node is a root node (black) with two children: a left child (black) and a right child (red). The right node is a root node (black) with a left child (black) and a right child (red). The root node of the right tree is enclosed in a white box, and the red node is also enclosed in a white box.

127  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Rot-Schwarz-Bäume

The diagram shows two binary tree nodes. The left node is a root node (black) with two children: a left child (black) and a right child (black). The right node is a root node (black) with a left child (black) and a right child (black). The root node of the right tree is enclosed in a white box.

128  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

B-Bäume

- Operationen
 - Search()
 - Insert()
 - Delete()
- In den meisten Fällen kommen die Operationen ohne Restrukturierung des Baums aus
- Sonst: $O(1)$ verallgemeinerte Rotationen





Zugriff auf externen Speicher

- DiskRead(x), DiskWrite(x)
- $x \leftarrow$ pointer to some object
DiskRead(x);
access/modify the contents of x
DiskWrite(x) // only if modified
access but don't modify the contents of x





Create()

- Create()
 - $x \leftarrow \text{AllocateNode}()$
 - $\text{leaf}[x] \leftarrow \text{true}$
 - $n[x] \leftarrow 0$
 - $\text{DiskWrite}(x)$

131  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Search()



- SEARCH(x, k)
 - $i \leftarrow 1$
 - while $i \leq n[x]$ and $k > \text{key}_i[x]$
 - $i \leftarrow i + 1$
 - if $i \leq n[x]$ and $k = \text{key}_i[x]$ then
 - return (x, i)
 - if leaf[x] then
 - return NIL
 - else
 - DiskRead($c_{i-1}[x]$)
 - Search($c_{i-1}[x], k$)

132  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

while statt if, da der Verzweigungsgrad jetzt größer als 2 sein kann und nicht konstant ist

Insert()



- Bei Binärbäumen wird immer ein neuer Blattknoten eingefügt
- Bei B-Bäumen ist kein neuer Knoten nötig, solange der Blattknoten noch nicht voll ist
- Steigt die Zahl der Schlüssel über $2 \times t - 1$, so wird das Blatt geteilt

133  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Binärbäume wachsen an den Blättern – B-Bäume wachsen an der Wurzel

Insert()

- Naiver Ansatz
 - Suche das entsprechende Blatt
 - Split() falls überfüllt
 - Propagiere nach oben, falls Vaterknoten ebenfalls überläuft
 - Problem: Auf die Knoten entlang des Pfades wird jeweils zweimal zugegriffen

134  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Insert()

- One-Pass Algorithmus
 - Teile die Knoten entlang des Suchpfades bereits auf dem Hinweg wenn diese schon voll sind
 - Evtl. unnötige Split-Operationen amortisieren sich durch den schnelleren Average-Case



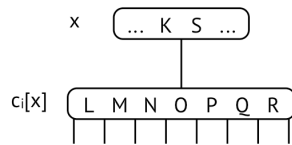
SplitChild()

- SplitChild(x,i)
 - // x : Vaterknoten
 - // i : Index des Sohns
 - // Vorbedingung: $c_i[x]$ ist voll



SplitChild()

- Beispiel $t = 4$



137



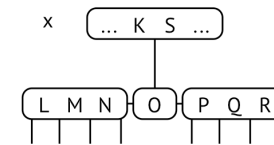
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SplitChild()

- Beispiel $t = 4$



138



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SplitChild()

- Beispiel $t = 4$

x ... K S ...
|
O
/ \
L M N P Q R
| | | | | |

139

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

SplitChild()

- Beispiel $t = 4$

x ... K O S ...
|
O
/ \
L M N P Q R
| | | | | |

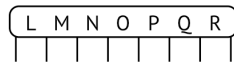
$C_i[x]$ $C_{i+1}[X]$

140

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

SplitChild()

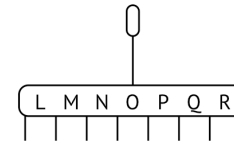
- Spezialfall: Wurzel



SplitChild()

- Spezialfall: Wurzel

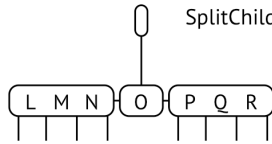
Erzeuge zuerst einen
leeren Knoten ...



SplitChild()

- Spezialfall: Wurzel

Erzeuge zuerst einen
leeren Knoten ... dann
SplitChild()



143



Datenstrukturen und Algorithmen

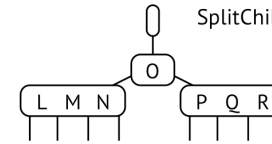
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FWTH AACHEN
UNIVERSITY

SplitChild()

- Spezialfall: Wurzel

Erzeuge zuerst einen
leeren Knoten ... dann
SplitChild()



144



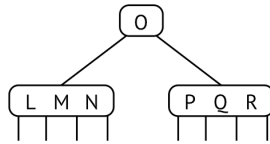
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FWTH AACHEN
UNIVERSITY

SplitChild()

- Spezialfall: Wurzel
- B-Bäume wachsen an der Wurzel



Insert()

- Insert(T,k)
r ← root[T]
if $n[r] = 2 \times t - 1$ then
s ← AllocateNode()
root[T] ← s
leaf[s] ← false
n[s] ← 0
co[s] ← r
SplitChild(s,0)
InsertNonFull(s,k)
else
InsertNonFull(r,k)

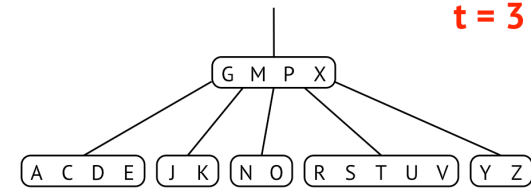


InsertNonfull()

- InsertNonFull(x,k)
 - if leaf[x] then
 - find correct position for k and insert
 - DiskWrite(x)
 - else
 - find correct subtree $c_i[x]$
 - DiskRead($c_i[x]$)
 - if $n[c_i[x]] = 2 \times t - 1$ then
 - SplitChild(x,i)
 - if $k > \text{key}_i[x]$ then
 - $i \leftarrow i + 1$
 - InsertNonFull($c_i[x], k$)



Beispiel



Beispiel

- Insert(B)

```
graph TD; Root["G M P X"]; Leaf1["A C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["R S T U V"]; Leaf5["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5;
```

149 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Beispiel

- Insert(B)


```
graph TD; Root["G M P X"]; Leaf1["A C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["R S T U V"]; Leaf5["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5;
```

150 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Beispiel

- Insert(B)


```
graph TD; Root["G M P X"]; Leaf1["A C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["R S T U V"]; Leaf5["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5;
```

151  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Beispiel

- Insert(B)


```
graph TD; Root["G M P X"]; Leaf1["A B C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["R S T U V"]; Leaf5["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5;
```

152  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Beispiel

- Insert(Q)


```
graph TD; Root["G M P X"]; Leaf1["A B C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["R S T U V"]; Leaf5["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5;
```

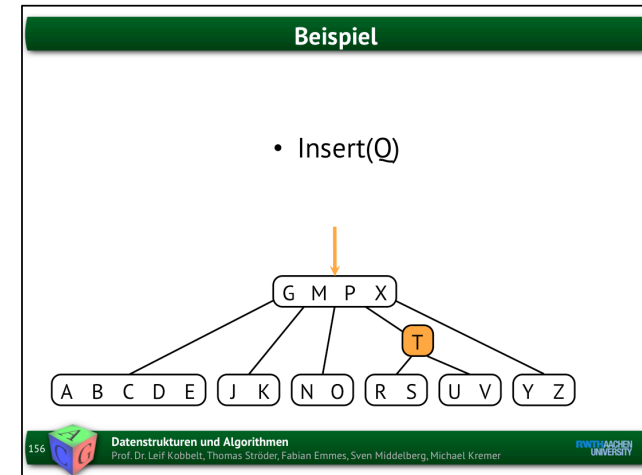
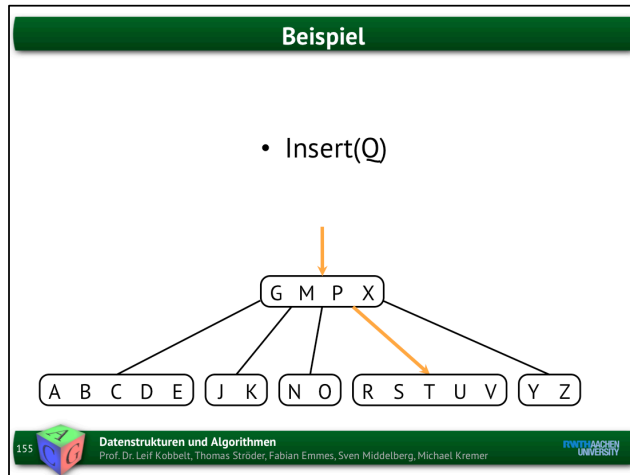
153  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

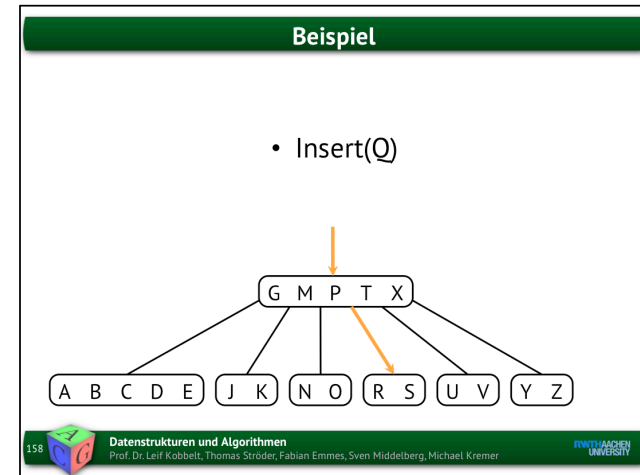
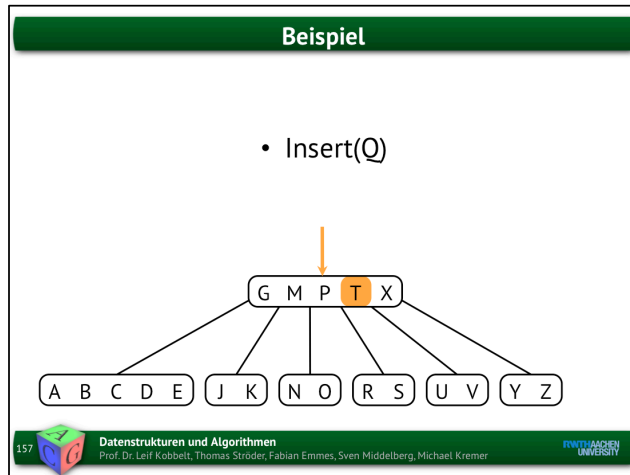
Beispiel

- Insert(Q)

```
graph TD; Root["G M P X"]; Leaf1["A B C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["R S T U V"]; Leaf5["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5;
```

154  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer







Beispiel

• Insert(Q)



```
graph TD; Root["G M P T X"]; Leaf1["A B C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["Q R S"]; Leaf5["U V"]; Leaf6["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5; Root --- Leaf6;
```

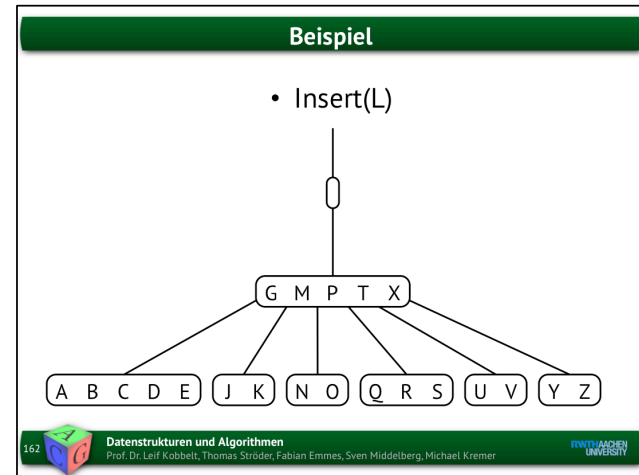
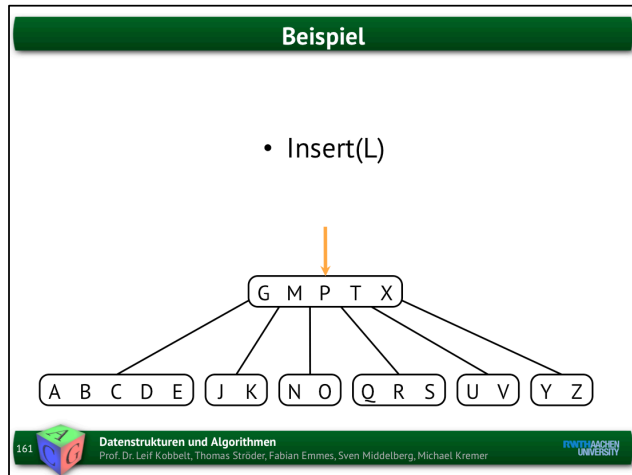
159  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

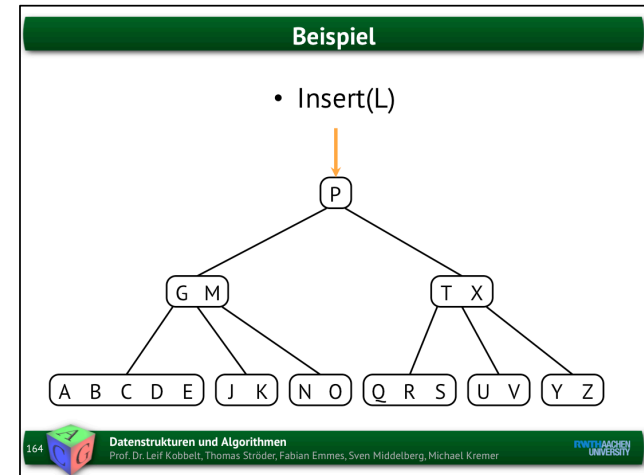
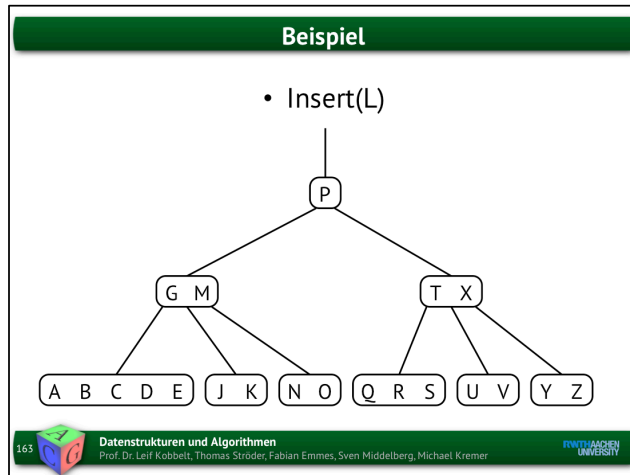
Beispiel

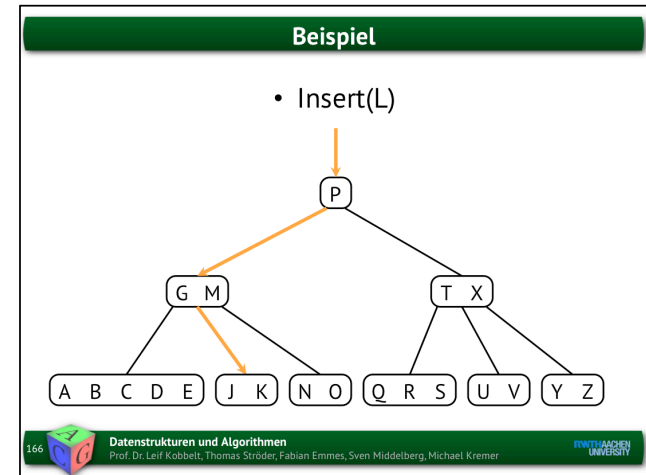
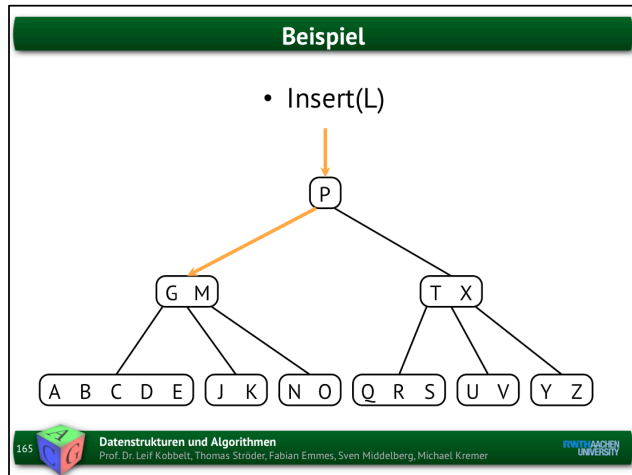
• Insert(L)

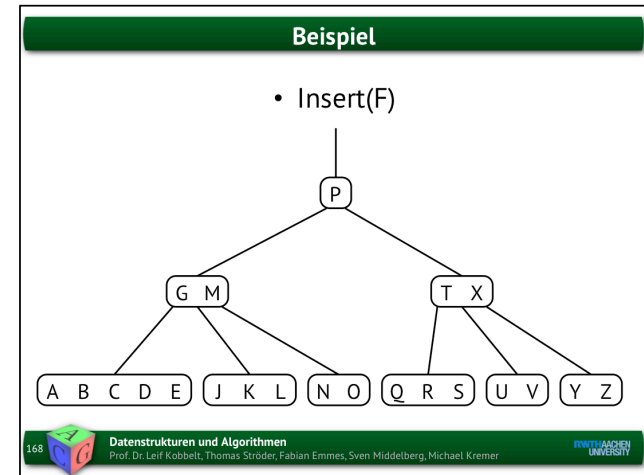
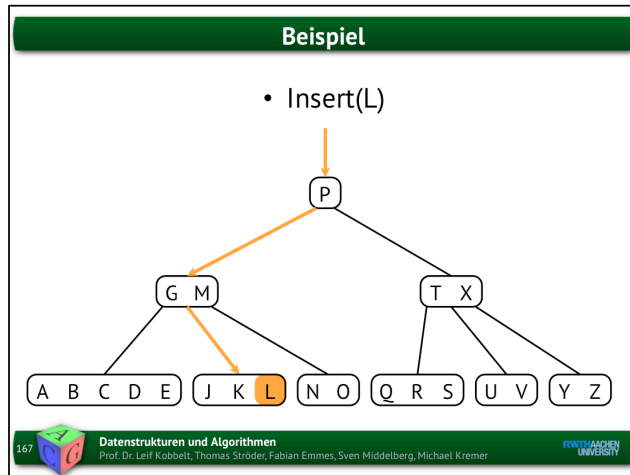
```
graph TD; Root["G M P T X"]; Leaf1["A B C D E"]; Leaf2["J K"]; Leaf3["N O"]; Leaf4["Q R S"]; Leaf5["U V"]; Leaf6["Y Z"]; Root --- Leaf1; Root --- Leaf2; Root --- Leaf3; Root --- Leaf4; Root --- Leaf5; Root --- Leaf6;
```

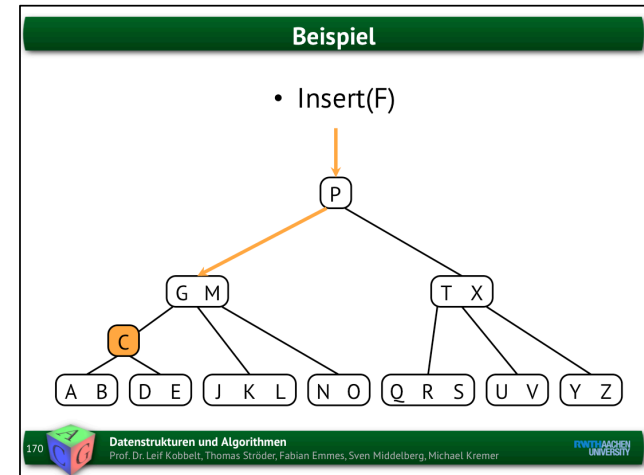
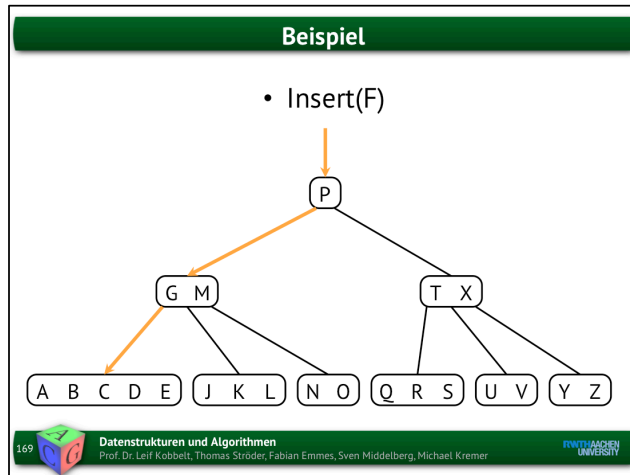
160  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

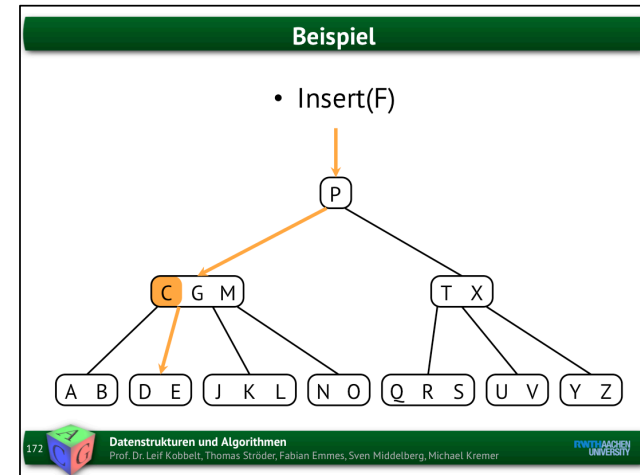
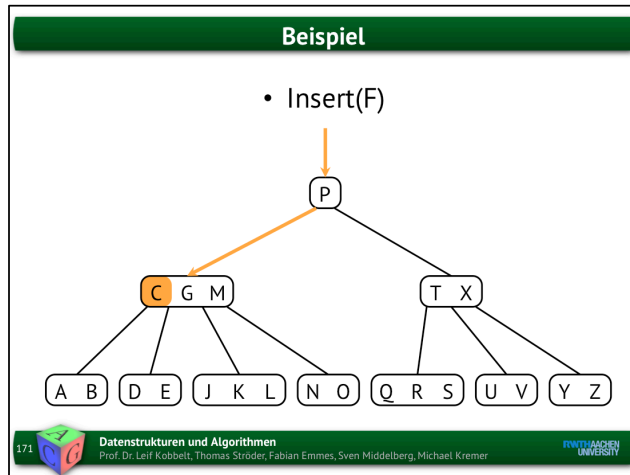


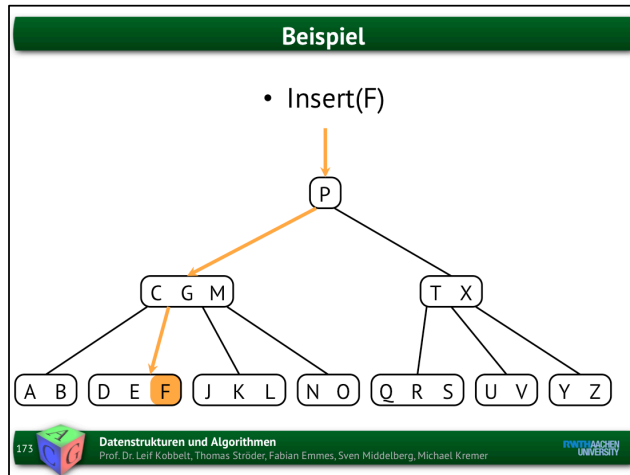




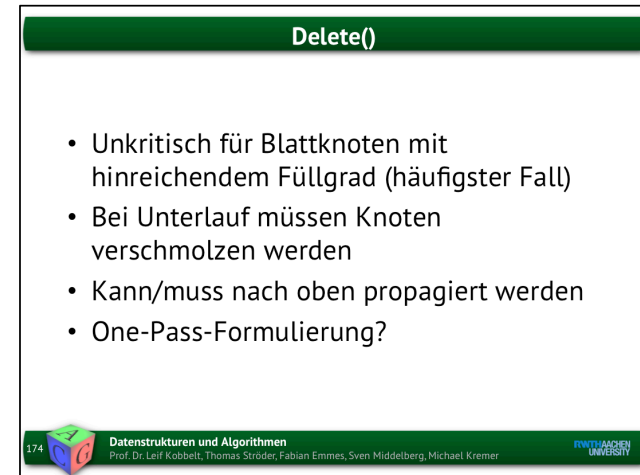








Sobald in rekursivem Abstieg festgestellt wird, dass ein Knoten voll ist, wird dieser zunächst einmal gesplittet, bevor weiter abgestiegen wird.



Delete()

- Delete() sucht den Schlüssel entlang eines Pfades
- Eine Verschmelzung findet nur statt, wenn die entsprechenden Knoten beide einen Füllgrad $\leq t-1$ haben.
- Teste vor dem Abstieg, ob der Nachfolger Füllgrad $\geq t$ hat (dann ist Verschmelzung ausgeschlossen)
- Falls dies nicht der Fall ist, erzwinge dies durch direkte Umstrukturierung



Delete()

- Wie bei Insert() sorgen wir dafür, dass der aktuelle Knoten hinreichend gefüllt ist, so dass keine Verschmelzung notwendig wird (Schleifeninvariante)
- Falls ein Knoten nicht hinreichend gefüllt wäre, hätten wir das bereits bei der Bearbeitung des Vaterknotens bemerkt und behoben.



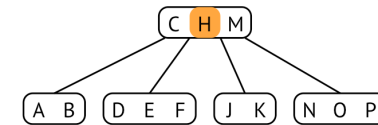
Delete()

- Hilfsoperationen
 - Steal()
 - Merge()
 - Rotate()



Steal()

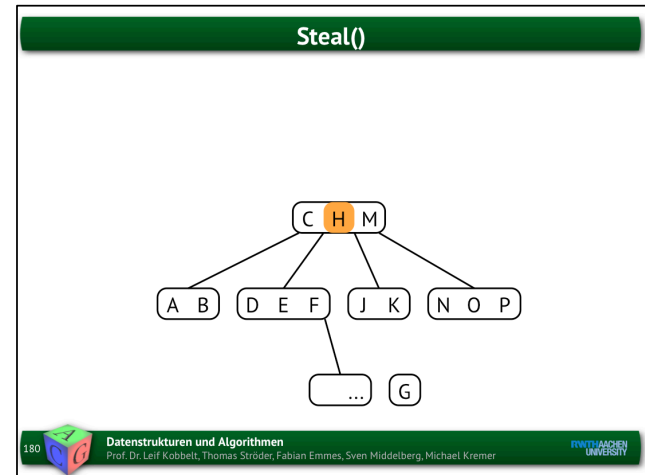
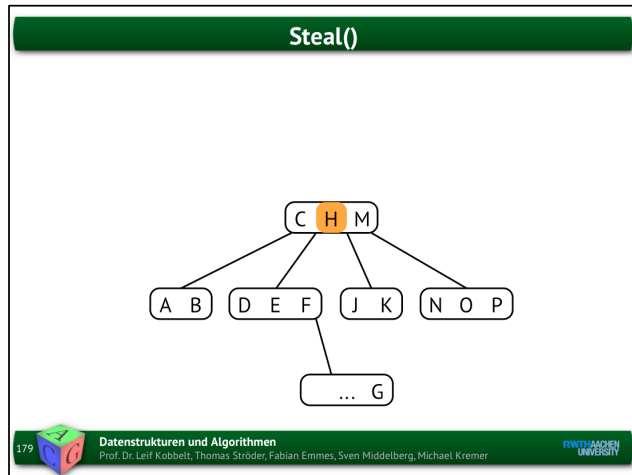
- Lösche Element aus **aktuellem** Knoten



$t = 3$

mehr als $t-1$ Knoten → rekursiver Abstieg möglich





Steal()

[A B] [D E F] [J K] [N O P]
[...]

181 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

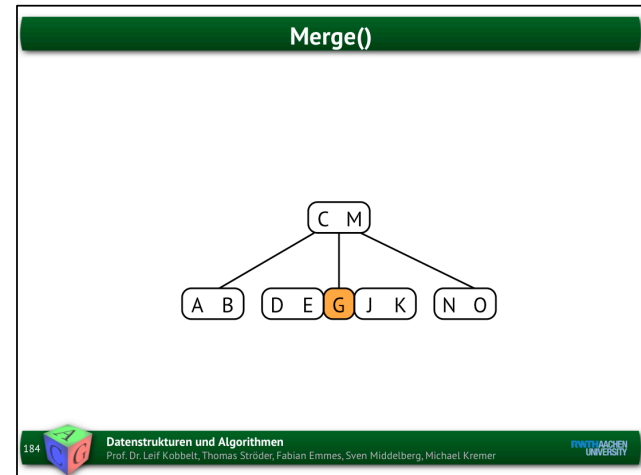
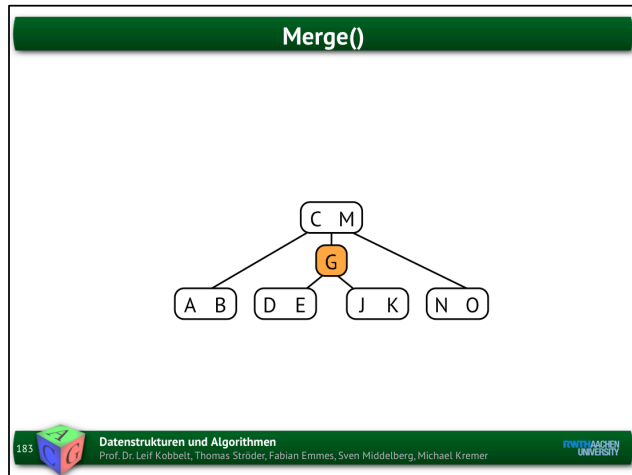
Merge()

- Lösche Element aus **aktuellem** Knoten

[A B] [D E] [J K] [N O]

nur $t-1$ Knoten → rekursiver Abstieg **nicht** möglich

182 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Merge()

- Lösche Element aus **aktuellem** Knoten

```

    graph TD
      Root["C M"] --- Leaf1["A B"]
      Root --- Leaf2["D E G J K"]
      Root --- Leaf3["N O"]
      style Leaf2 stroke:#f96
    
```

185

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Rotate()

- Lösche Element aus **rechtem** Teilbaum

```

    graph TD
      Root["C G M"] --- Leaf1["A B"]
      Root --- Leaf2["D E F"]
      Root --- Leaf3["J K"]
      Root --- Leaf4["N O"]
      style Leaf3 stroke:#f96
      style Leaf4 stroke:#f96
      Leaf2 --> Leaf3
    
```

nur $t-1$ Knoten → rekursiver Abstieg nicht möglich

186

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Rotate()

- Lösche Element aus **rechtem** Teilbaum

nur $t-1$ Knoten \rightarrow rekursiver Abstieg nicht möglich



187 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Rotate()

188 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Rotate()

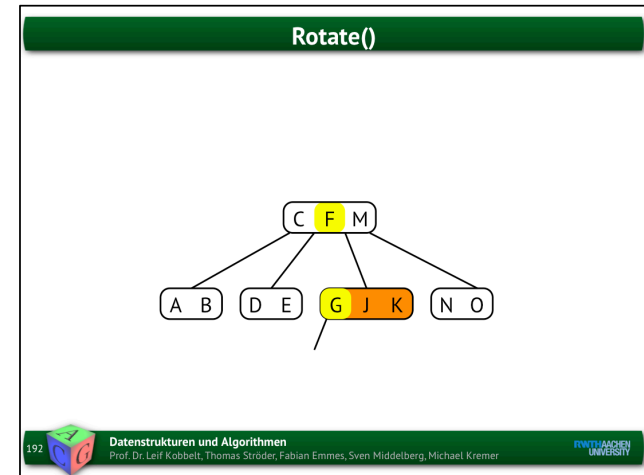
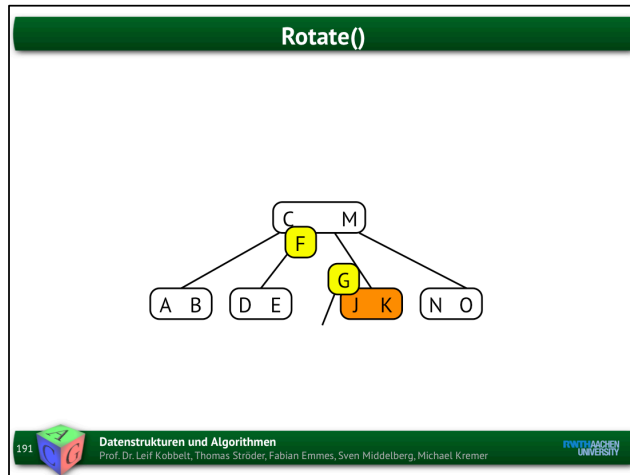
```
graph TD; CM[C M]; CM --- AB[A B]; CM --- F[F]; CM --- G[G]; CM --- NO[N O]; F --- DE[D E]; F --- JK[J K]; style F fill:#ffff00; style G fill:#ffff00; style JK fill:#ffa500;
```

189  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Rotate()

```
graph TD; CM[C M]; CM --- AB[A B]; CM --- F[F]; CM --- G[G]; CM --- NO[N O]; F --- DE[D E]; F --- JK[J K]; style F fill:#ffff00; style G fill:#ffff00; style JK fill:#ffa500;
```

190  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

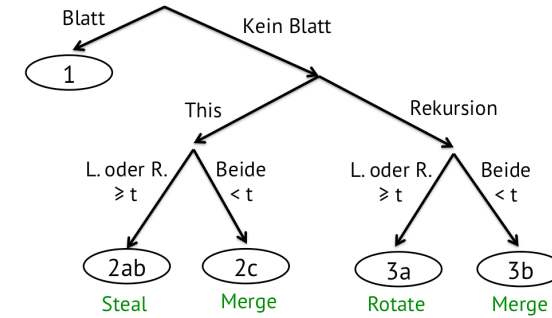


Delete()

- Standardfall: Schlüssel nicht im Knoten, Rekursion in den entsprechenden Teilbaum
- Nur in Knoten absteigen, die hinreichend gefüllt sind → wenn nicht, vorher den Baum mit Merge(), Steal(), Rotate() umstrukturieren.
- Nicht-Standardfälle: 1, 2a, 2b, 3a, 3b (s. Cormen)



B-Bäume – Delete – Scheme



Delete()

- **Fall 1:** Schlüssel in einem Blatt

Del(F, P)

C G M T X

A B D E F J K L N O Q R S U V Y Z

195 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 1:** Schlüssel in einem Blatt

P

Del(F, C G M) T X



A B D E F J K L N O Q R S U V Y Z

196 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 1:** Schlüssel in einem Blatt



```
graph TD; P((P)) --- CGM((C G M)); P --- TX((T X)); CGM --- DEF((D E F)); CGM --- JKL((J K L)); CGM --- NO((N O)); TX --- QRS((Q R S)); TX --- UV((U V)); TX --- YZ((Y Z)); style DEF stroke:#f96
```

197  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Delete()

- **Fall 1:** Schlüssel in einem Blatt
- Schlüssel wird gelöscht

```
graph TD; P((P)) --- CGM((C G M)); P --- TX((T X)); CGM --- AB((A B)); CGM --- DE((D E)); CGM --- JKL((J K L)); TX --- NO((N O)); TX --- QRS((Q R S)); TX --- UV((U V)); style AB stroke:#f96
```

198  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Delete()

- **Fall 2a:** Schlüssel in innerem Knoten, linker Sohn hat $\geq t$ Schlüssel

199 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 2a:** Schlüssel in innerem Knoten, linker Sohn hat $\geq t$ Schlüssel

200 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 2a:** Schlüssel in innerem Knoten, linker Sohn hat $\geq t$ Schlüssel
- Stehle **Vorgänger** aus linkem Teilbaum

201 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

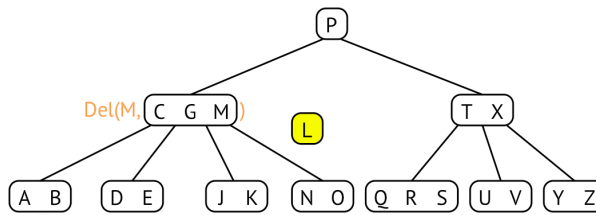
Delete()



- **Fall 2a:** Schlüssel in innerem Knoten, linker Sohn hat $\geq t$ Schlüssel
- Stehle **Vorgänger** aus linkem Teilbaum

202 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

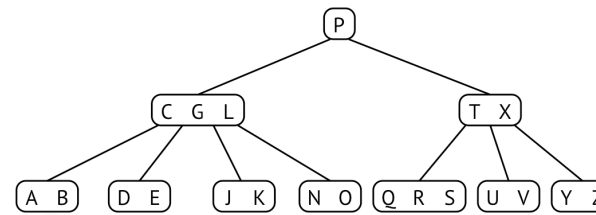
- **Fall 2a:** Schlüssel in innerem Knoten, linker Sohn hat $\geq t$ Schlüssel
- Stehle **Vorgänger** aus linkem Teilbaum





203  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Delete()

- **Fall 2a:** Schlüssel in innerem Knoten, linker Sohn hat $\geq t$ Schlüssel
- Stehle **Vorgänger** aus linkem Teilbaum



204  **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

Delete()

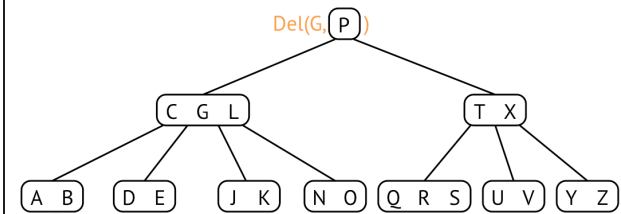
- **Fall 2b:** Schlüssel in innerem Knoten, rechter Sohn hat $\geq t$ Schlüssel

→ analog zu **Fall 2a**



Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel



Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel

207

Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel
- Stehlen nicht möglich \rightarrow Merge()

208

Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel
- Stehlen nicht möglich \rightarrow Merge()

209 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel
- Stehlen nicht möglich \rightarrow Merge()

210 **Datenstrukturen und Algorithmen**
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel
- Stehlen nicht möglich \rightarrow Merge()

Del(G, C L)

A B

D E G J K

N O

Q R S

U V

Y Z

211

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel
- Stehlen nicht möglich \rightarrow Merge()
- Rekursion

Del(G, D E G J K)

D E G J K

N O

Q R S

U V

Y Z

212

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Delete()

- **Fall 2c:** Schlüssel in innerem Knoten, beide Söhne haben $t-1$ Schlüssel
- Stehlen nicht möglich \rightarrow Merge()
- Rekursion

```

graph TD
    P((P)) --- CL((C L))
    P --- TX((T X))
    CL --- AB((A B))
    CL --- DEJK((D E J K))
    CL --- NO((N O))
    TX --- QRST((Q R S))
    TX --- UV((U V))
    TX --- YZ((Y Z))
  
```

213 Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
 FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel

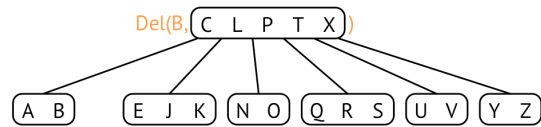
```

graph TD
    CLPTX((C L P T X)) --- AB((A B))
    CLPTX --- EJK((E J K))
    CLPTX --- NO((N O))
    CLPTX --- QRST((Q R S))
    CLPTX --- UV((U V))
    CLPTX --- YZ((Y Z))
  
```

214 Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
 FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel



215



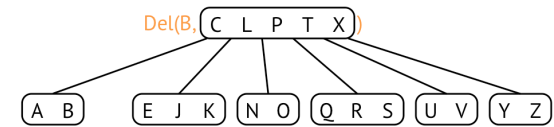
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FWTH AACHEN
UNIVERSITY

Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate()



216



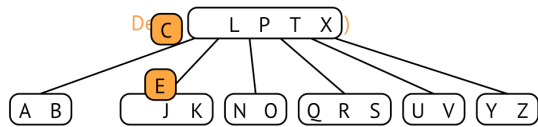
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FWTH AACHEN
UNIVERSITY

Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate()



217



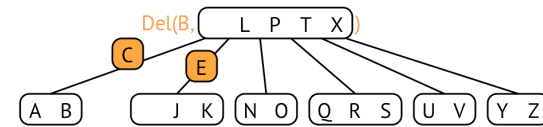
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate()



218



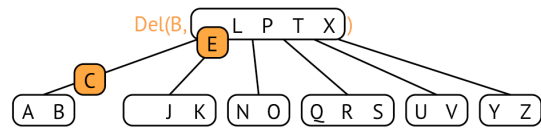
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate()



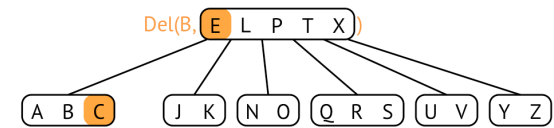
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate()



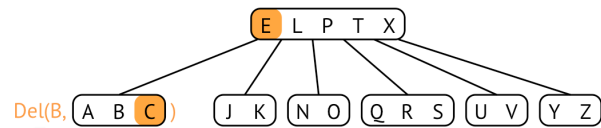
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



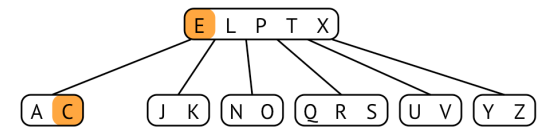
Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate(), Rekursion



Delete()

- **Fall 3a:** Schlüssel nicht im aktuellen Knoten, der entspr. Unterbaum hat nur $t-1$ Schlüssel, sein Bruder aber t Schlüssel
- Rotate(), Rekursion



Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel

223
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel

224
Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel

Abstieg nicht möglich!

225

Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel
- Merge zwei der Unterbäume

Abstieg nicht möglich!

226

Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel
- Merge zwei der Unterbäume

Del(D,)

227

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel
- Merge zwei der Unterbäume

Del(D, C L P T X)

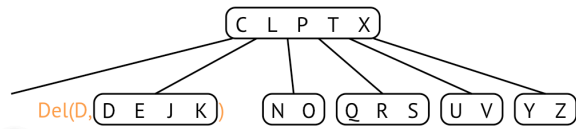
228

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel
- Merge zwei der Unterbäume



229



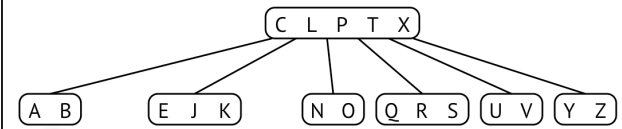
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRONTMÄCHER
UNIVERSITY

Delete()

- **Fall 3b:** Schlüssel nicht im aktuellen Knoten, sowohl der entspr. Unterbaum als auch dessen Brüder haben $t-1$ Schlüssel
- Merge zwei der Unterbäume



230



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRONTMÄCHER
UNIVERSITY

Zusammenfassung

- B-Bäume
 - Optimierter Zugriff auf externen Speicher durch Blockung mehrerer Schlüssel
 - Minimaler/maximaler Füllgrad
 - Garantierte Balancierung
 - Hysterese bei der Umstrukturierung
 - Steal(), Merge(), Rotate()

