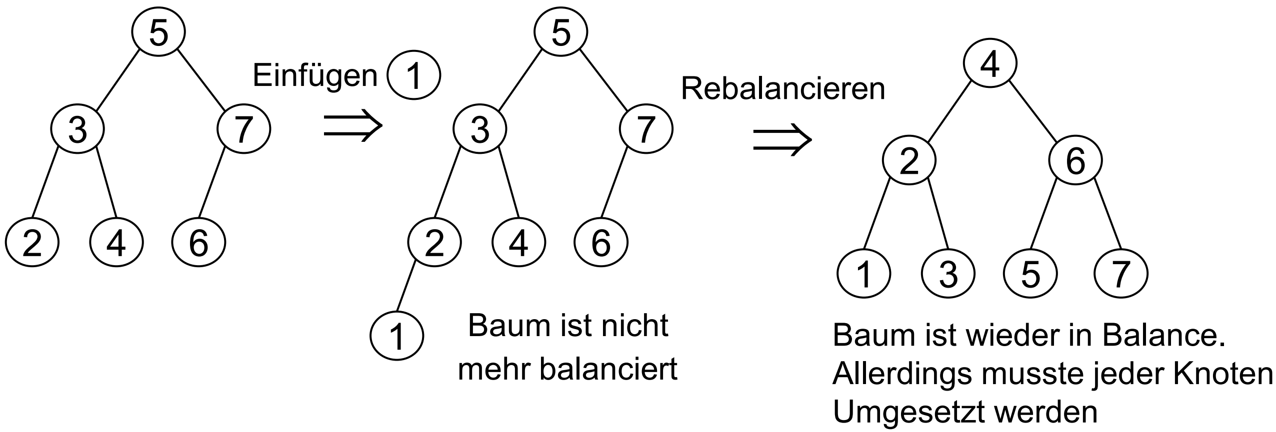
Algorithmen & Datenstrukturen

Balancierte Bäume

Wolfgang Auer

Motivation

- Laufzeit der Operationen auf binären Suchbäumen wird durch die Höhe bestimmt.
- Die Höhe des Baumes wird durch die Reihenfolge des Einfügens der Knoten bestimmt $\Rightarrow$ Sortiertes Einfügen führt zur Entartung!
- Idee: Schon beim Einfügen dafür sorgen, dass der Baum in Balance bleibt.

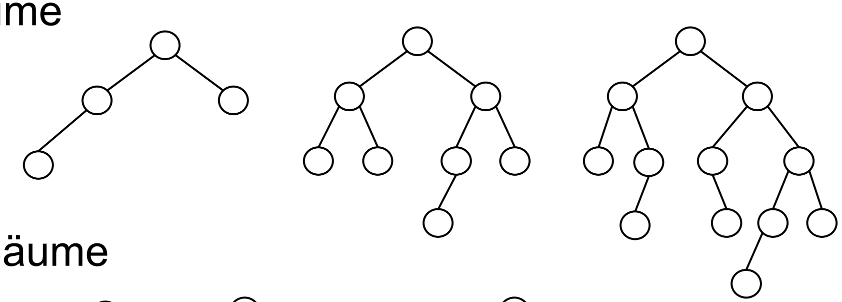


Durch das Rebalancieren ist wieder eine Laufzeit von O(log n) garantiert. Allerdings können die Kosten für das Rebalancieren O(n) betragen

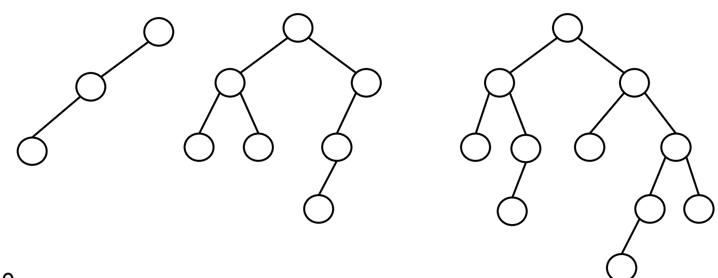
AVL-Bäume

- Adelson-Velskii und Landis entwickelten eine Methode annähernd balancierte Bäume mit einer Laufzeit O(log n) zu erstellen.
- Die AVL-Eigenschaft besagt, dass die Differenz der Höhe des linken Unterbaums und des rechten Unterbaums eines Knotens maximal 1 sein darf.

Gültige AVL-Bäume



Ungültige AVL-Bäume

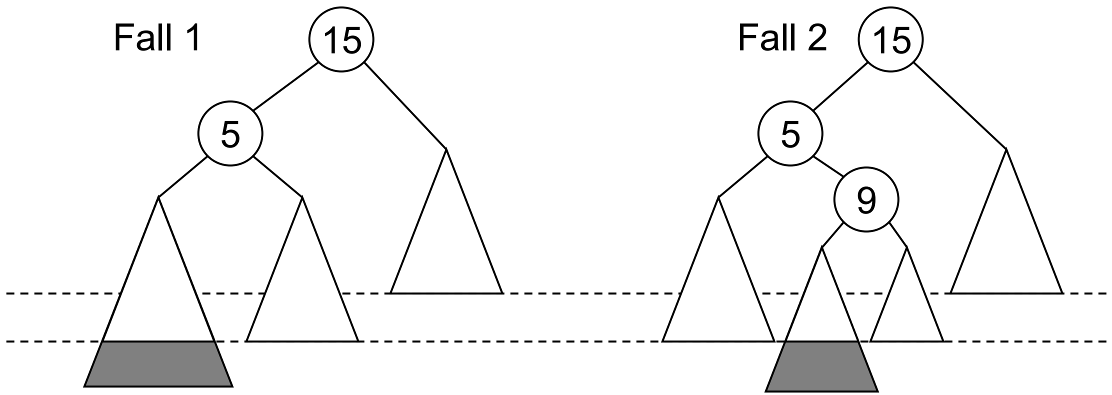


Einfügen (1)

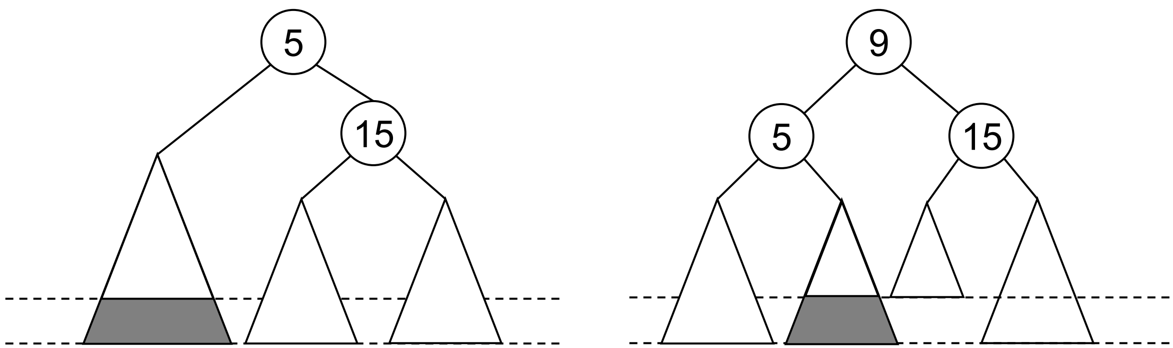
- Das Einfügen in einen balancierten Baum kann die Balance des Baumes zerstören
- Beispiel:
  - Gegeben ist ein Baum *T* mit den Unterbäumen *L* und *R*, die die Höhen h_L und h_R besitzen. Beim Einfügen in *L* (linken Unterbaum) können drei Fälle auftreten
    - $h_L = h_R \Rightarrow$ Durch das Einfügen wird *T* linkslastig, dennoch bleibt die Balance gewahrt.
    - $h_L < h_R \Rightarrow$ *T* war rechtslastig, durch das Einfügen wird $h_L = h_R$ und Balance bleibt bewahrt.
    - $h_L > h_R \Rightarrow$ *T* war linkslastig, durch das Einfügen wird Balance zerstört. Baum muss **umstrukturiert** werden

Einfügen (2)

- Beim Einfügen in den linkslastigen Baum sind zwei Situationen möglich



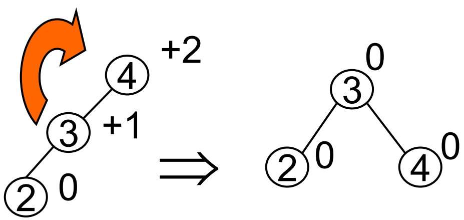
Resultat der Umstrukturierung



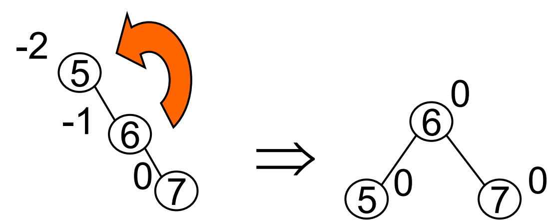
Rotation (1)
(Einfache Rotation)

- Umstrukturieren des Baums wird als Rotation bezeichnet

- Einfacher Fall (einfache Rotation)



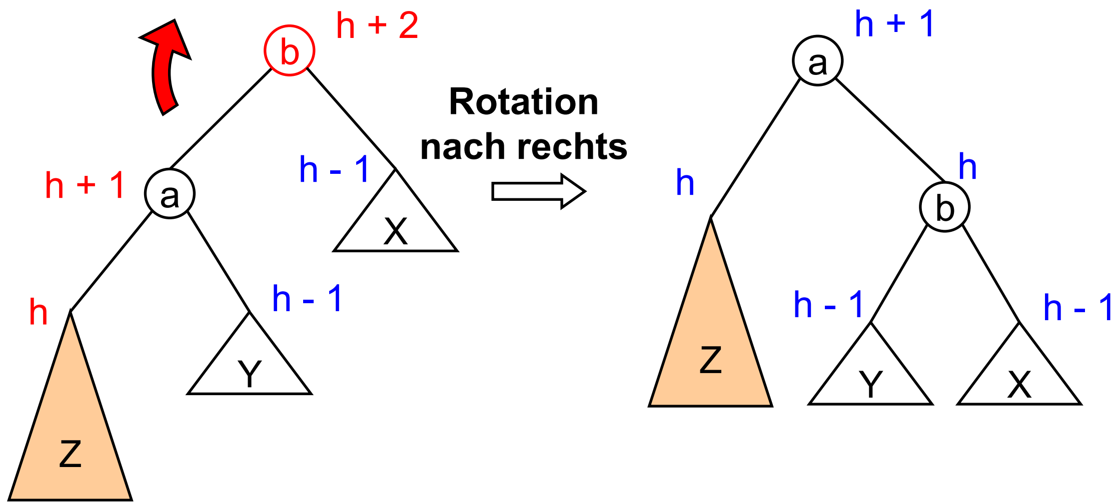
Links-lastiger Baum =>
Rechtsrotation



Rechts-lastiger Baum =>
Linksrotation

Rotation (2)
(Einfache Rotation)

- Schema der einfachen Rotation

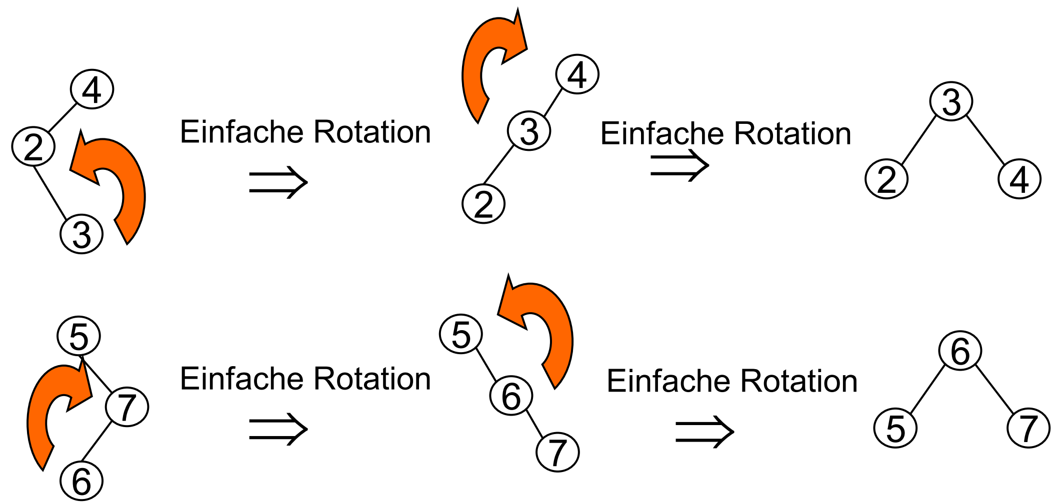


h…Höhe des Unterbaumes *Z*

- Höhe des linken Teilbaums bleibt identisch
- Höhe des Vorfahren des Teilbaums bleibt gleich

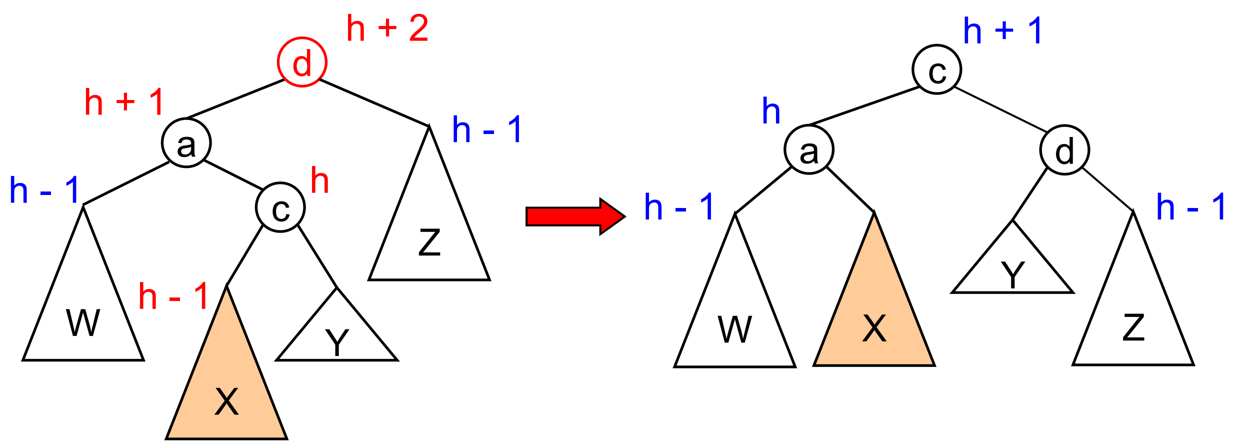
Rotation (3)
(Doppelrotation)

- In verschiedenen Fällen reicht eine Einfachrotation nicht aus, um den Baum auszugleichen



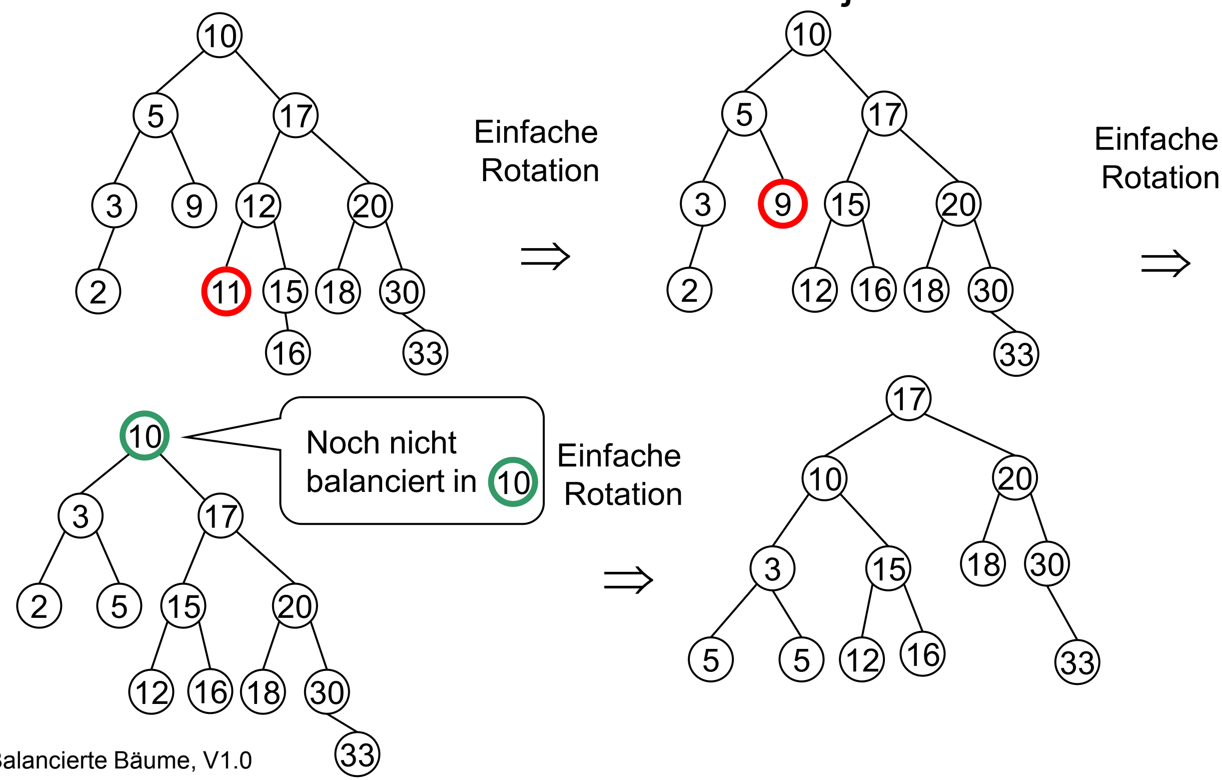
Rotation (4)
(Doppelrotation)

- Schema der Doppelrotation



Löschen

- Obwohl die Schritte zum Ausgleich im wesentlichen identisch sind, ist das Löschen im ausgeglichenen Baum komplizierter als das Einfügen.
  - Das Löschen kann eine Rotation für jeden Knoten bedeuten!



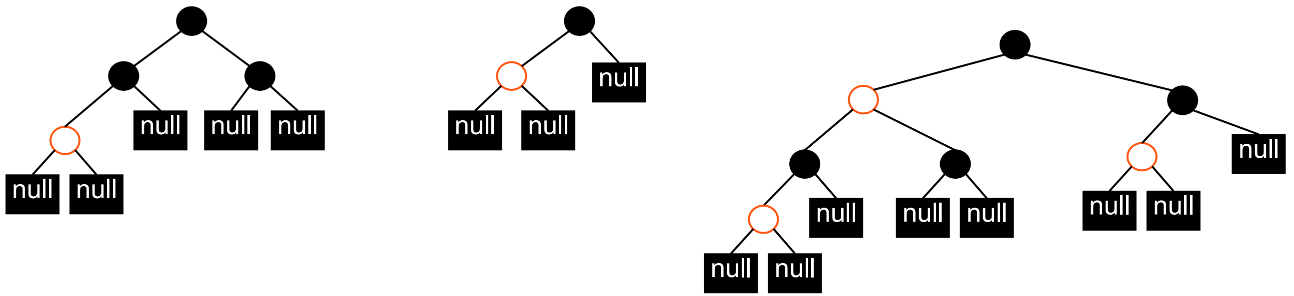
Rot-Schwarz-Bäume
(Red-Black trees)

- R. Bayer entwickelte symmetrische binäre B-Bäume. Guibas und Sedgewick adaptierten diese zu Rot-Schwarz-Bäume.
- *Rot-Schwarz-Bäume* sind binäre Suchbäume, die zusätzlich folgende Bedingungen erfüllen:
  - Jeder Knoten enthält die Felder *color, key, left, right, parent*
  - Jeder Knoten ist entweder *rot* oder *schwarz* gefärbt
  - Die Wurzel des Baumes ist immer *schwarz*
  - Die Kinder eines *roten* Knotens sind immer *schwarz*
  - Externe Knoten sind immer *schwarz*
  - Jeder Pfad von einem Knoten zu allen externen Knoten enthält die gleiche Anzahl von *schwarzen* Knoten

$\boxed{\Rightarrow \text{Maximale Höhe} = 2 \operatorname{ld}(n + 1)}$

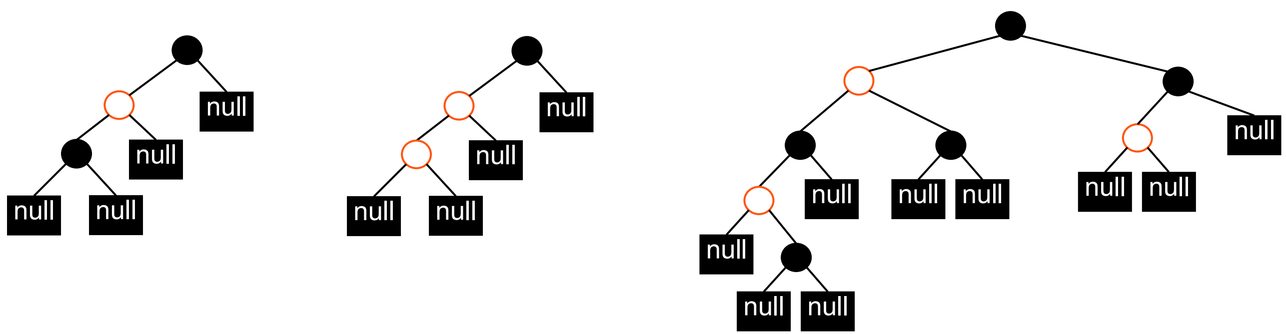
Rot-Schwarz-Bäume
(Beispiele)

- Gültige RB-Bäume



rot eigenfärbter Knoten   schwarz eigenfärbter Knoten

- Ungültige RB-Bäume



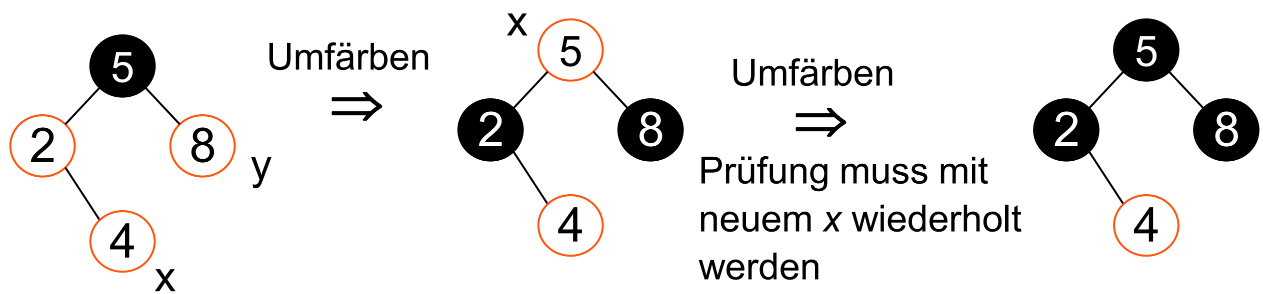
Einfügen (1)

- Einfügen kann die Balance des Baumes zerstören => rebalancieren

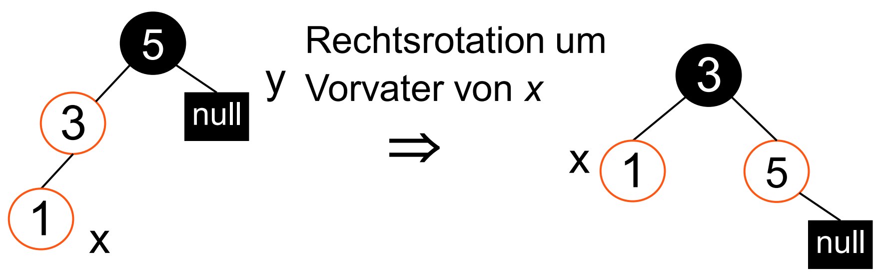
```
RBInsert(root, x)  {
   Insert(root, x)
   x.color = red;
   Rebalance(root, x)
   root.color = black;
}
```

Einfügen (2)
Rebalancieren

- Fall 1: Vater von x und Onkel von x sind beide *rot*

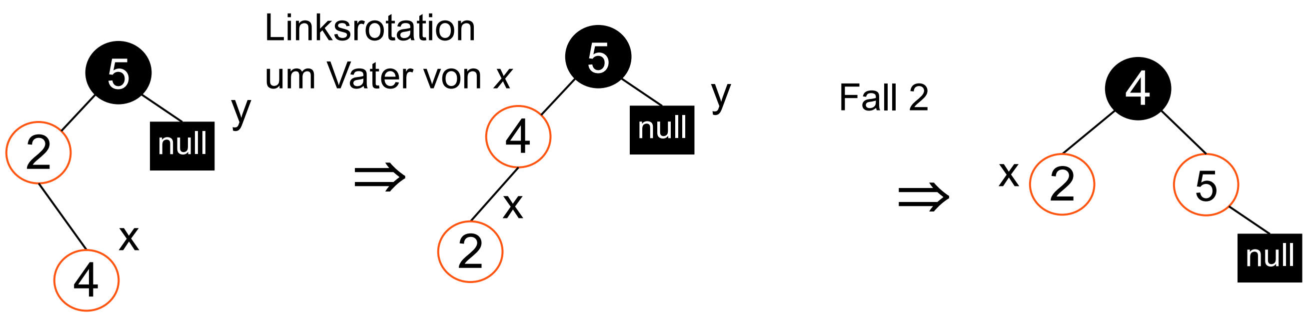


- Fall 2: *x* ist linkes Kind, Vater von *x* ist *rot* und Onkel von *x* ist *schwarz*



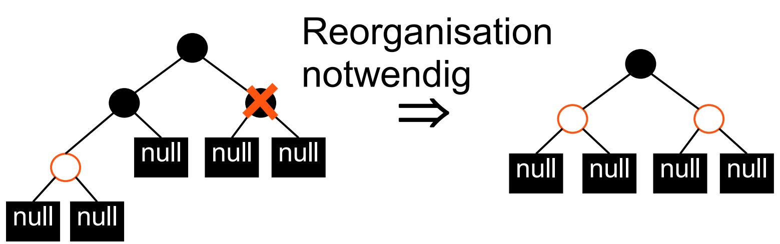
Einfügen (3)
Rebalancieren

- Fall 3: *x* ist rechtes Kind, Vater von *x* ist *rot* und Onkel von *x* ist *schwarz*



Löschen eines Knotens

- Prinzipiell gleiches Vorgehen wie beim herkömmlichen Löschen in BST.
  Nur wenn der zu löschende Knoten durch einen *schwarzen* Knoten ersetzt wird, muss reorganisiert werden



Reorganisation beim Löschen unterscheidet sich nur minimal von der beim Einfügen

Vergleich AVL und RB-Bäume

- Suchen im Baum
  - AVL und RB => O(log n)
- Maximale Höhe des Baumes:
  - AVL: $ld\ (n + 1)$
  - RB: $2\ ld\ (n + 1)$
- Laufzeit für Balancierung
  - AVL: zwischen $c_1\ ld\ n$ und $c_2\ ld\ n$, wobei $1 <= c_1 <= c_2$
  - RB: $c\ ld\ n$, wobei $1<= c$

Bewertung