ETH zürich

Felix Friedrich, Manuela Fischer

Datenstrukturen und Algorithmen

Vorlesung am D-MATH der ETH Zürich

Frühjahr 2023

16. Rot-Schwarz-Bäume

2-3-Bäume, Rot-Schwarz-Bäume
[Sedgewick/Wayne, Kap. 3.3, Cormen et al, Kap. 13]

Hintergrund

- Suchbaum: Suchen, Einfügen und Entfernen eines Schlüssels in $\mathcal{O}(h)$ Schritten (bei einem Baum der Höhe h)
- Schlechtester Fall: $\Theta(n)$ (degenerierter Baum)

Hintergrund

- Suchbaum: Suchen, Einfügen und Entfernen eines Schlüssels in $\mathcal{O}(h)$ Schritten (bei einem Baum der Höhe h)
- Schlechtester Fall: $\Theta(n)$ (degenerierter Baum)

Ziel: Verhindern der Degenerierung, durch Balancieren des Baumes nach jeder Update-Operation.

Balancierung: Garantiere, dass ein Baum mit n Knoten stets eine Höhe von $\Theta(\log n)$ hat.

Hintergrund

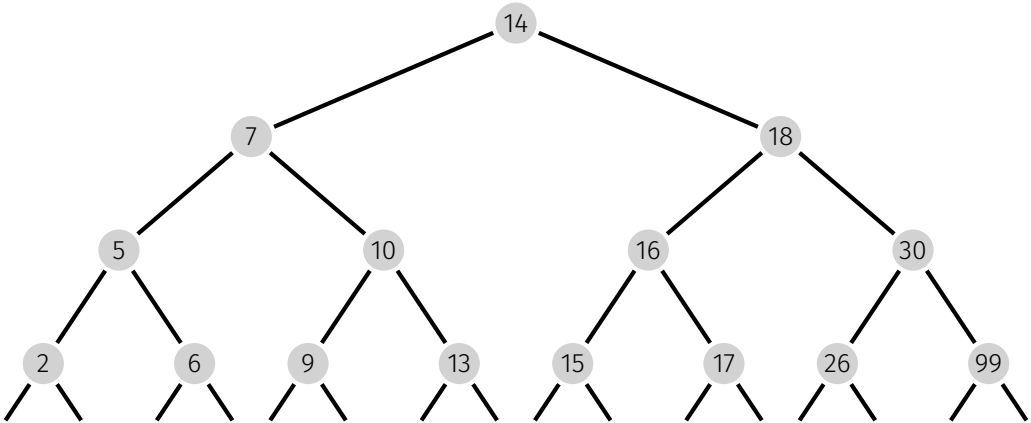
- Suchbaum: Suchen, Einfügen und Entfernen eines Schlüssels in $\mathcal{O}(h)$ Schritten (bei einem Baum der Höhe h)
- Schlechtester Fall: $\Theta(n)$ (degenerierter Baum)

Ziel: Verhindern der Degenerierung, durch Balancieren des Baumes nach jeder Update-Operation.

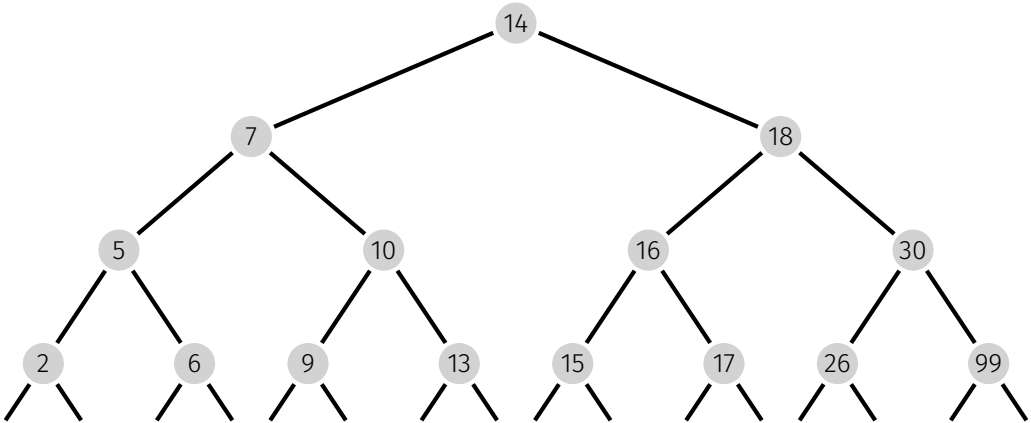
Balancierung: Garantiere, dass ein Baum mit n Knoten stets eine Höhe von $\Theta(\log n)$ hat.

Bayer (1972), Guibas-Sedgewick 1979, Cormen/Leiserson/Rivest/Stein (2001) und Sedgewick (2007, 2008): Rot-Schwarz-Bäume

Perfekt balancierter Suchbaum

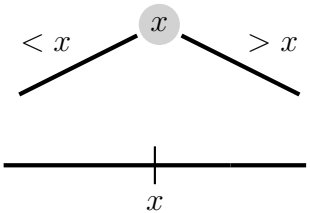


Perfekt balancierter Suchbaum



jeder Pfad von Wurzel zu Blatt hat gleiche Länge $\lceil \log_2(n+1) \rceil$

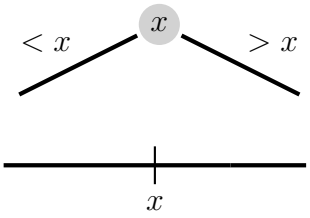
2-Knoten und 3-Knoten



2-Knoten

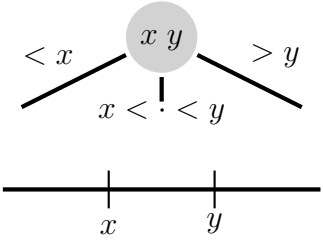
einen Schlüssel x
zwei Kinder

2-Knoten und 3-Knoten



2-Knoten

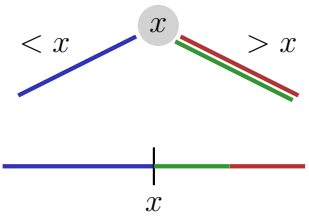
einen Schlüssel x
zwei Kinder



3-Knoten

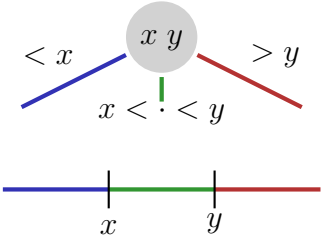
zwei Schlüssel $x < y$
drei Kinder

2-Knoten und 3-Knoten



2-Knoten

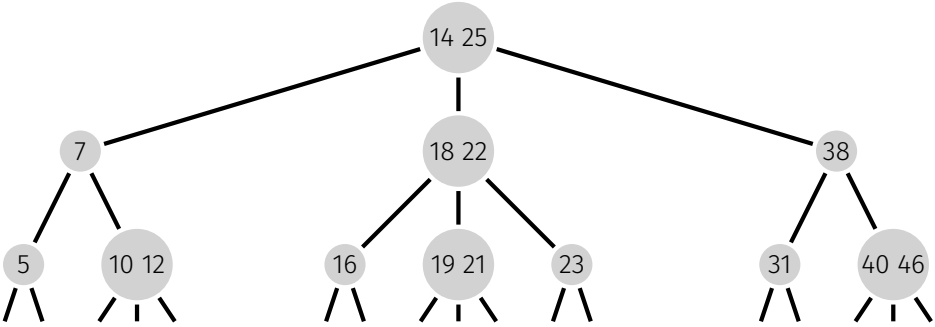
einen Schlüssel x
zwei Kinder



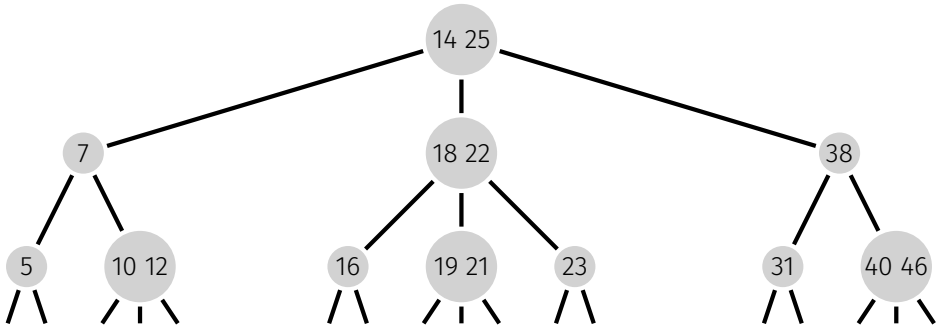
3-Knoten

zwei Schlüssel $x < y$
drei Kinder

2-3-Baum

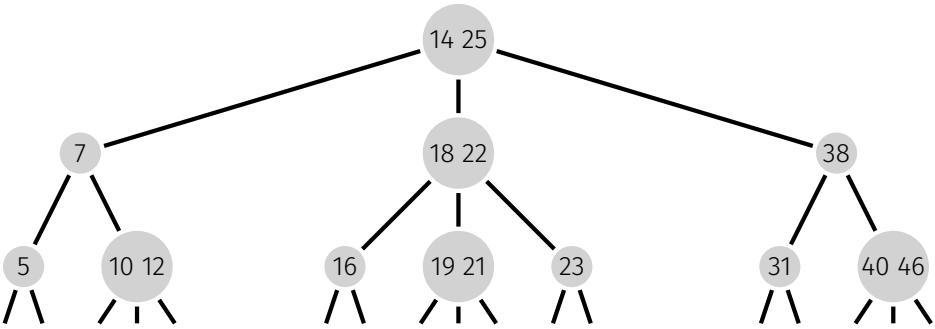


2-3-Baum



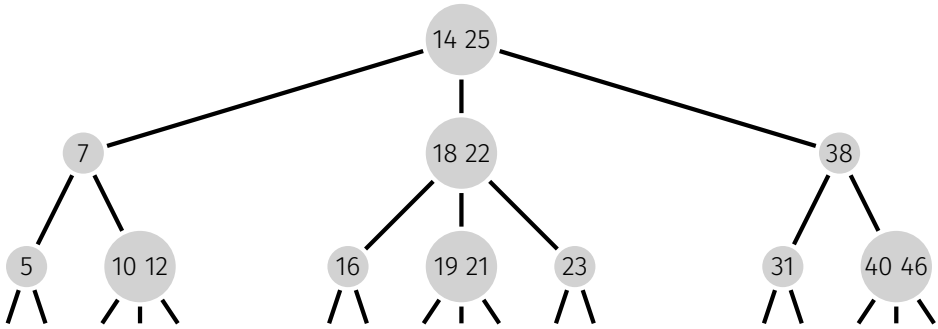
- jeder Knoten ist ein 2-Knoten oder ein 3-Knoten

2-3-Baum



- jeder Knoten ist ein 2-Knoten oder ein 3-Knoten
- der Baum ist perfekt balanciert

2-3-Baum



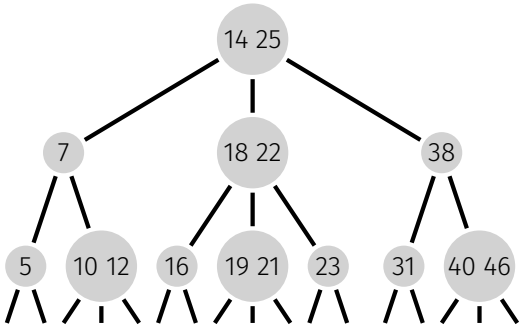
- jeder Knoten ist ein 2-Knoten oder ein 3-Knoten
- der Baum ist perfekt balanciert

$\Rightarrow$ Höhe $\lceil \log_3(n+1) \rceil \leq h \leq \lceil \log_2(n+1) \rceil$

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

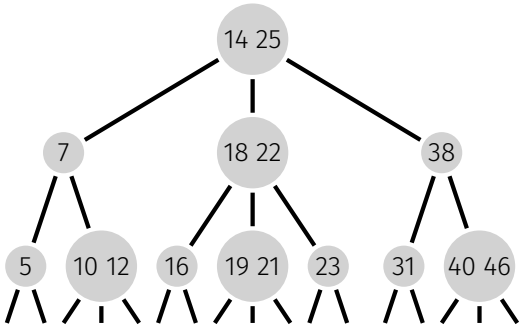
$v \leftarrow r$
while $v \neq$ **null do**
    if $k \in v.\text{keys}$ **then**
        return v
    else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
        $v \leftarrow v.\text{left}$
    else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
        $v \leftarrow v.\text{right}$
    else
        $v \leftarrow v.\text{middle}$
return null



Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null

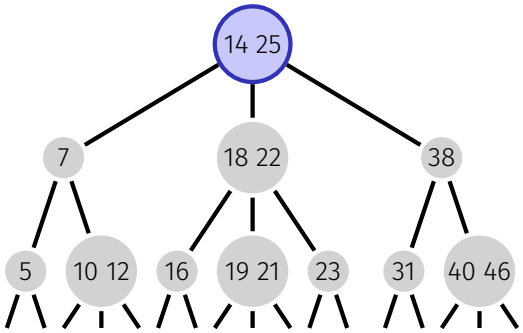


`search(17)`

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null

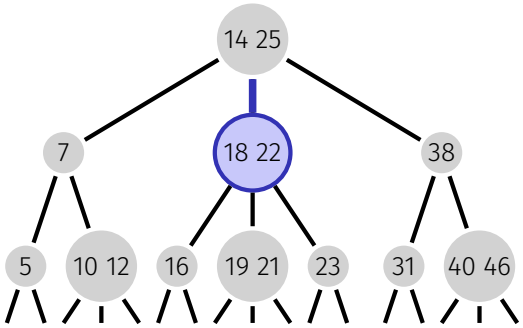


`search(17)`

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null

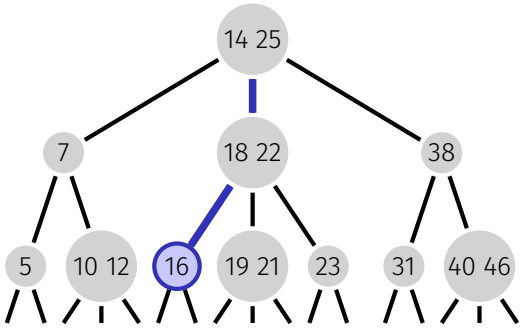


`search(17)`

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null

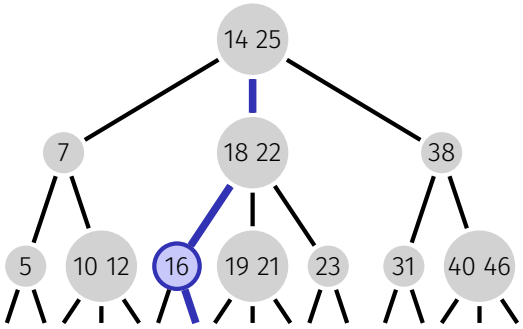


`search(17)`

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null

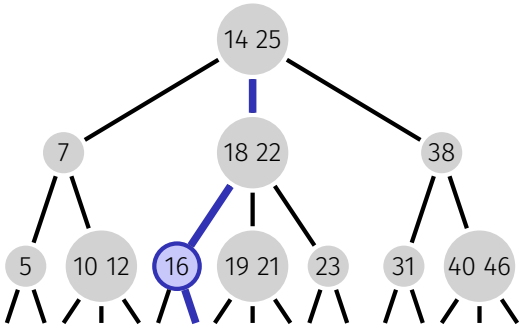


`search(17)`

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null

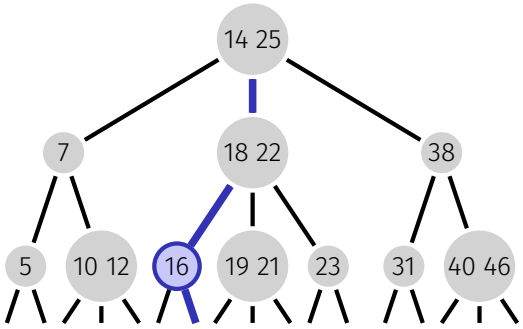


`search`$(17) \rightarrow$ **null**

Suchen

Input: Wurzel r, Schlüssel k
Output: v mit $k \in v.\text{keys}$ oder **null**

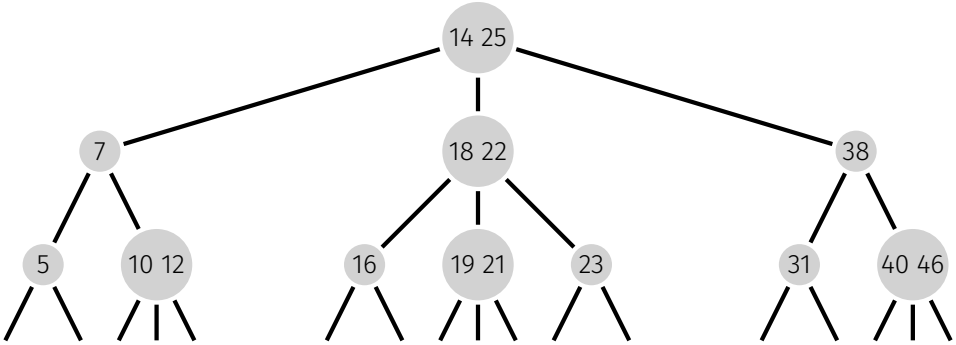
$v \leftarrow r$
while $v \neq$ **null do**
  if $k \in v.\text{keys}$ **then**
    return v
  else if $\forall \text{key} \in v.\text{keys}\colon k < \text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else if $\forall \text{key} \in v.\text{keys}\colon k > \text{key}$ **then**
    $v \leftarrow v.\text{right}$
  else
    $v \leftarrow v.\text{middle}$
return null



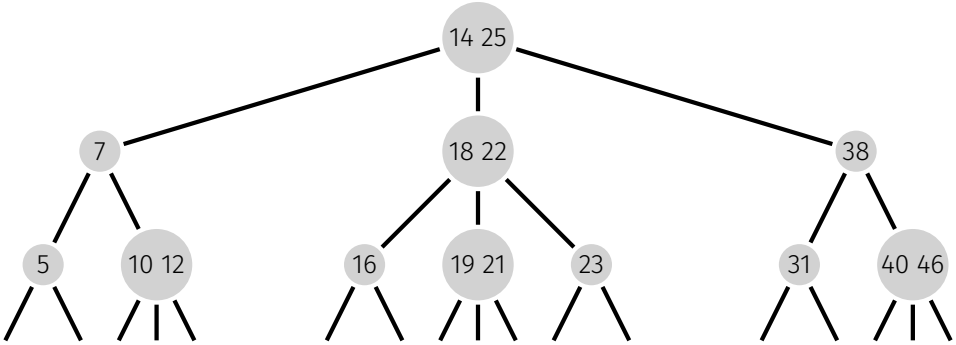
`search`$(17) \rightarrow$ **null**

in $\mathcal{O}(\log n)$

Beispiel: Einfügen in 2-Knoten

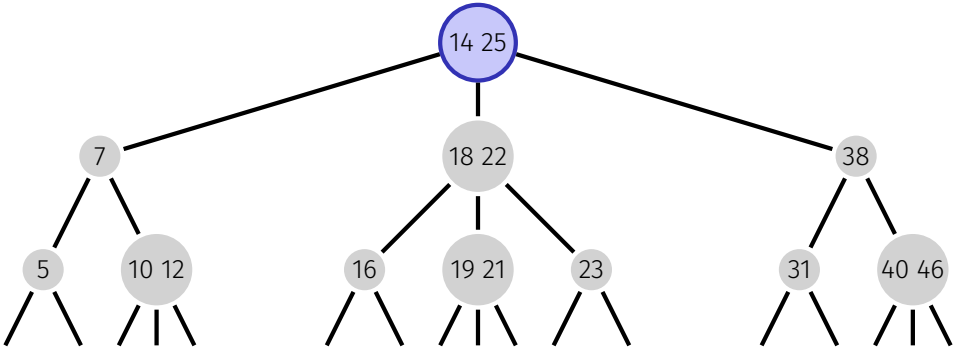


Beispiel: Einfügen in 2-Knoten



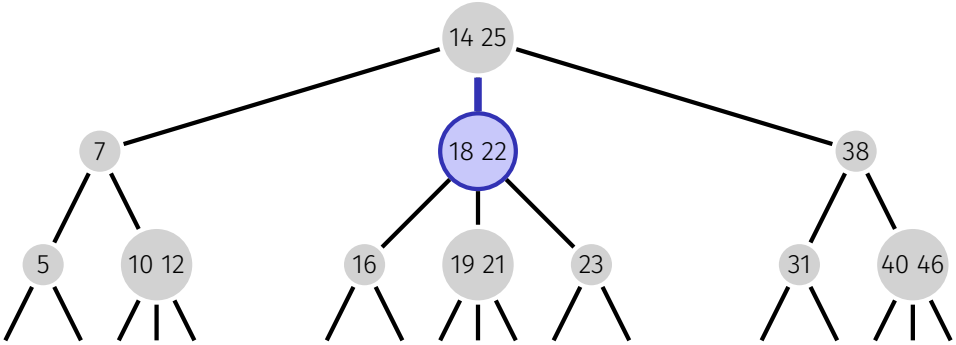
`insert`(17)

Beispiel: Einfügen in 2-Knoten



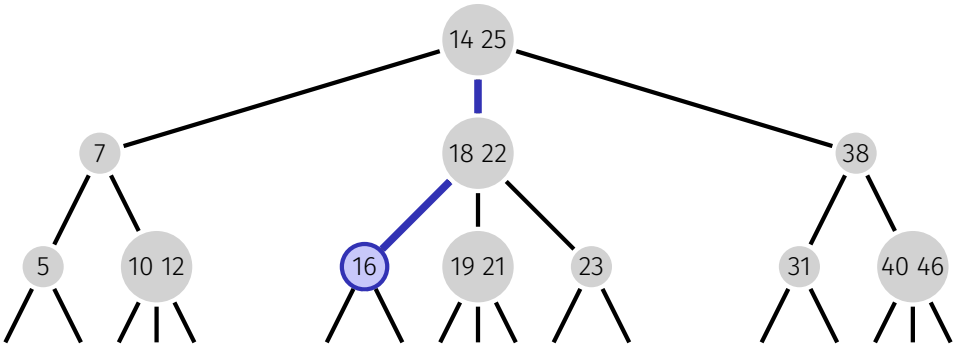
`insert`(17)

Beispiel: Einfügen in 2-Knoten



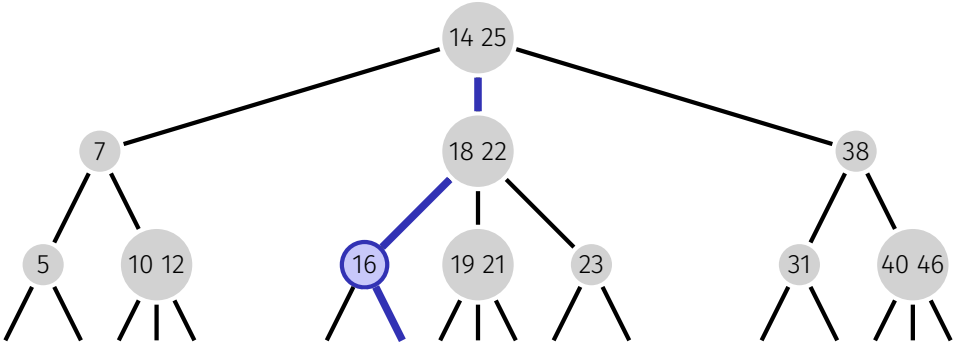
`insert`(17)

Beispiel: Einfügen in 2-Knoten



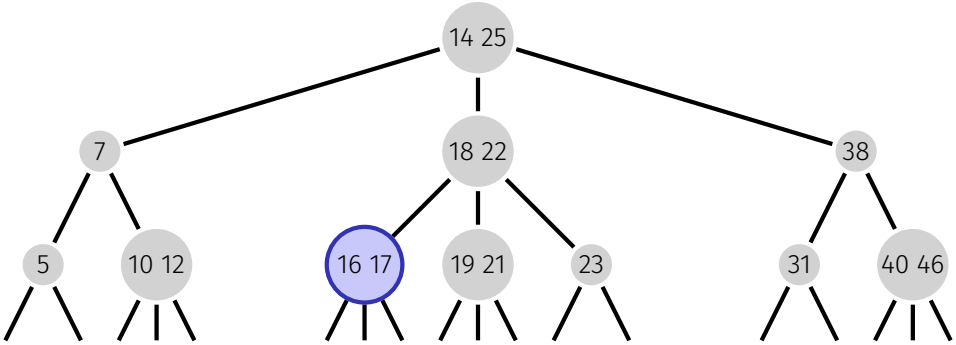
`insert`(17)

Beispiel: Einfügen in 2-Knoten



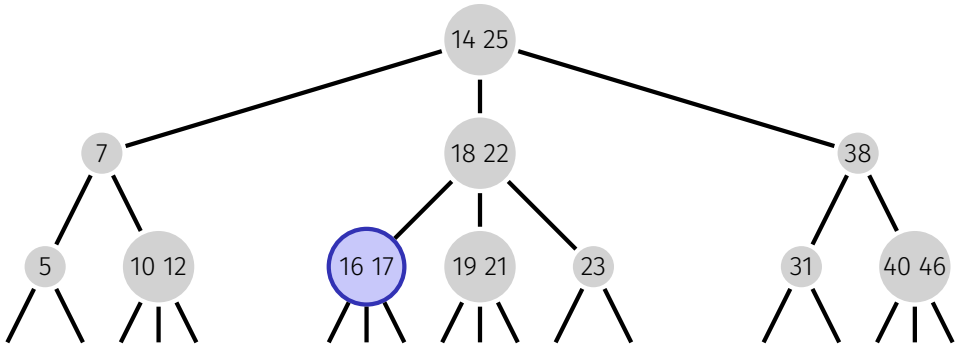
`insert`(17)

Beispiel: Einfügen in 2-Knoten



`insert`(17)

Beispiel: Einfügen in 2-Knoten



`insert`(17)

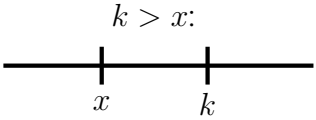
2-Knoten wird zu 3-Knoten (`join`),
alles andere (insbesondere Höhe) bleibt gleich!

Einfügen eines Schlüssels in 2-Knoten

2-Knoten nimmt Schlüssel auf und wird zu 3-Knoten: `join`

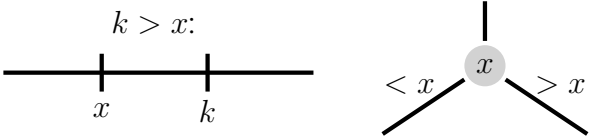
Einfügen eines Schlüssels in 2-Knoten

2-Knoten nimmt Schlüssel auf und wird zu 3-Knoten: `join`



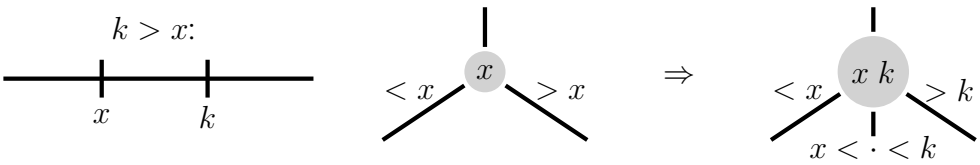
Einfügen eines Schlüssels in 2-Knoten

2-Knoten nimmt Schlüssel auf und wird zu 3-Knoten: `join`



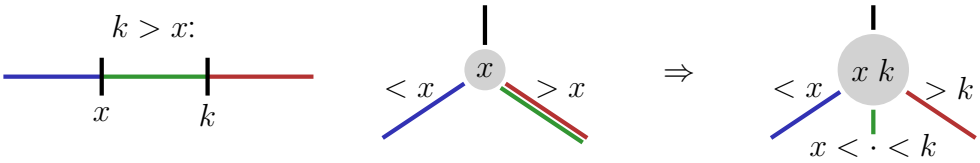
Einfügen eines Schlüssels in 2-Knoten

2-Knoten nimmt Schlüssel auf und wird zu 3-Knoten: `join`



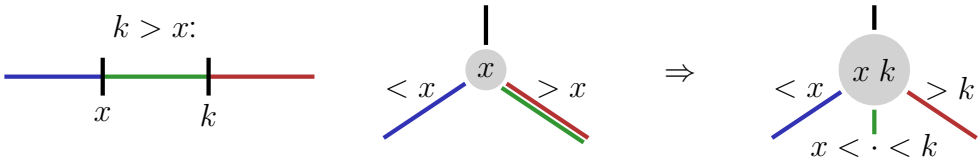
Einfügen eines Schlüssels in 2-Knoten

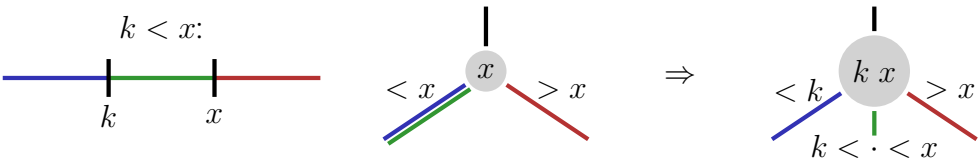
2-Knoten nimmt Schlüssel auf und wird zu 3-Knoten: `join`



Einfügen eines Schlüssels in 2-Knoten

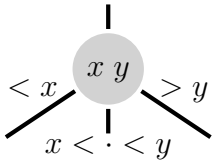
2-Knoten nimmt Schlüssel auf und wird zu 3-Knoten: `join`





Einfügen eines Schlüssels in 3-Knoten

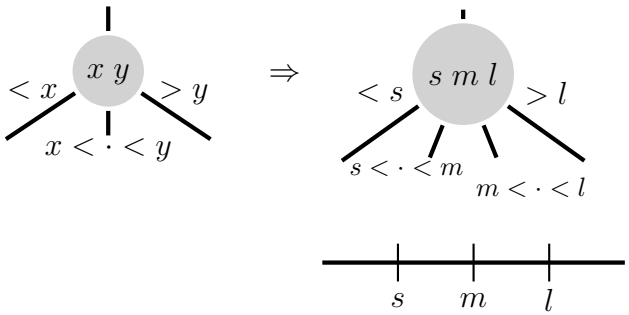
Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)



Einfügen eines Schlüssels in 3-Knoten

Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)

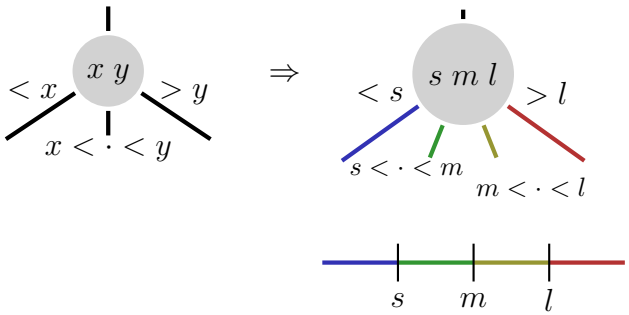
$$s := \min\{x, y, k\}, l := \max\{x, y, k\}, m \in \{x, y, k\} \setminus \{s, l\}$$



Einfügen eines Schlüssels in 3-Knoten

Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)

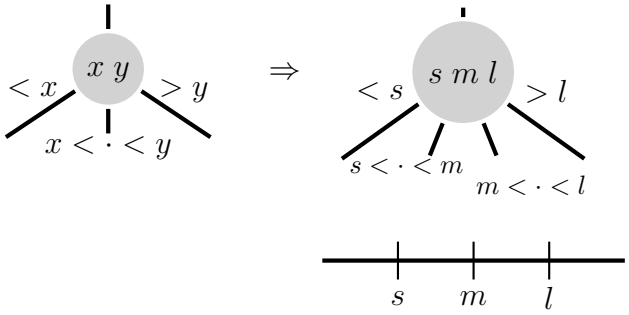
$$s := \min\{x, y, k\}, l := \max\{x, y, k\}, m \in \{x, y, k\} \setminus \{s, l\}$$



Einfügen eines Schlüssels in 3-Knoten

Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)

$$s := \min\{x, y, k\}, l := \max\{x, y, k\}, m \in \{x, y, k\} \setminus \{s, l\}$$

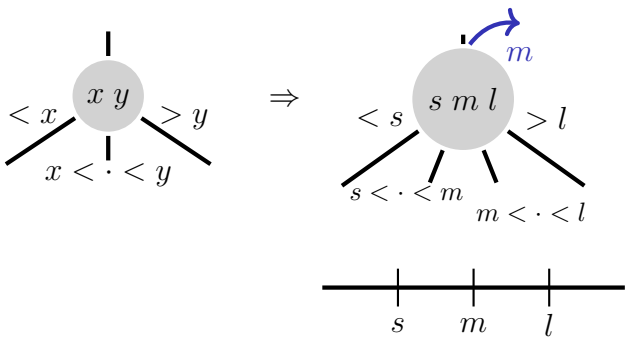


temporärer 4-Knoten

Einfügen eines Schlüssels in 3-Knoten

Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)

$$s := \min\{x, y, k\}, l := \max\{x, y, k\}, m \in \{x, y, k\} \setminus \{s, l\}$$

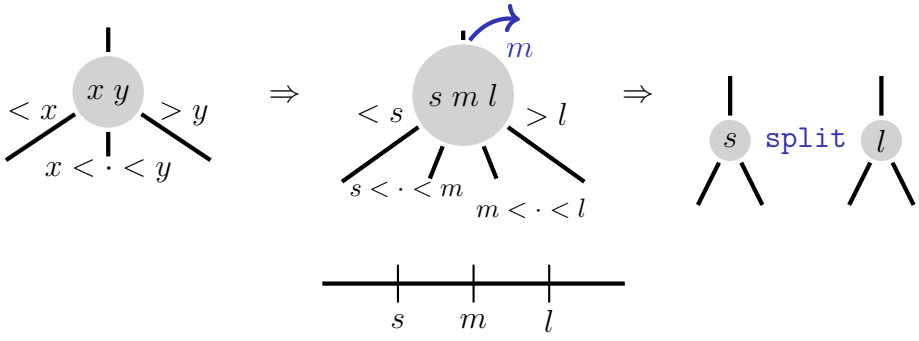


temporärer 4-Knoten

Einfügen eines Schlüssels in 3-Knoten

Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)

$$s := \min\{x, y, k\}, l := \max\{x, y, k\}, m \in \{x, y, k\} \setminus \{s, l\}$$

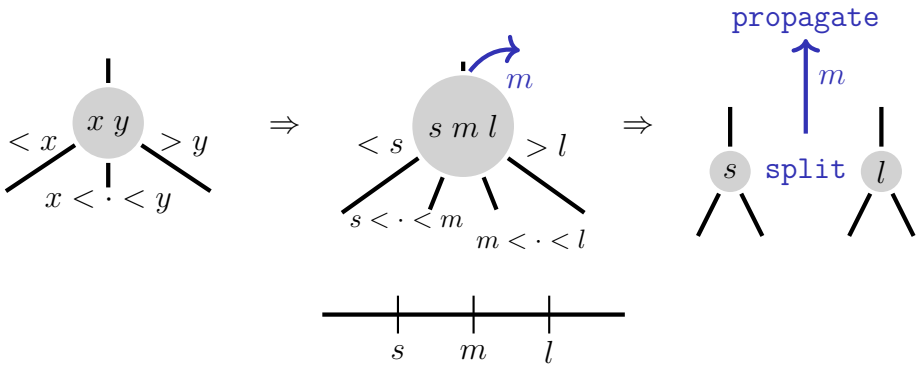


temporärer 4-Knoten

Einfügen eines Schlüssels in 3-Knoten

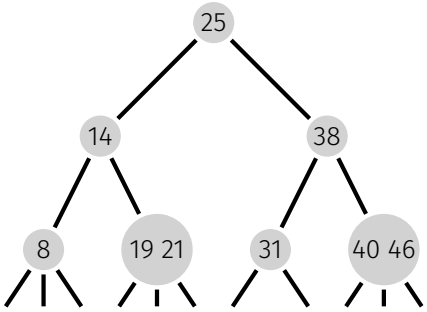
Einfügen in 3-Knoten mit Schlüssel x und y (mit $x < y$)

$$s := \min\{x, y, k\}, l := \max\{x, y, k\}, m \in \{x, y, k\} \setminus \{s, l\}$$

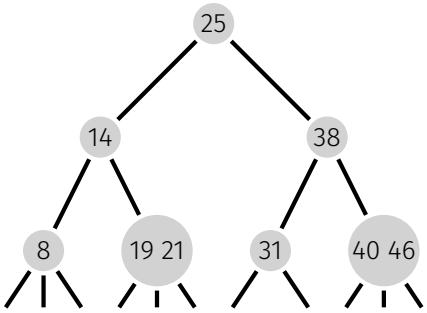


temporärer 4-Knoten

Beispiel: Einfügen in 3-Knoten

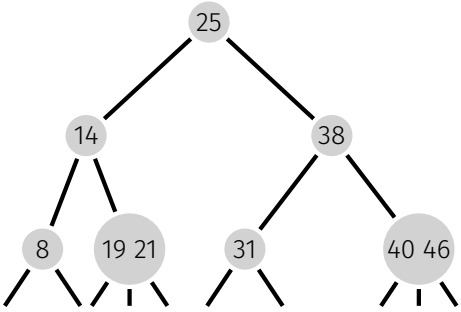


Beispiel: Einfügen in 3-Knoten

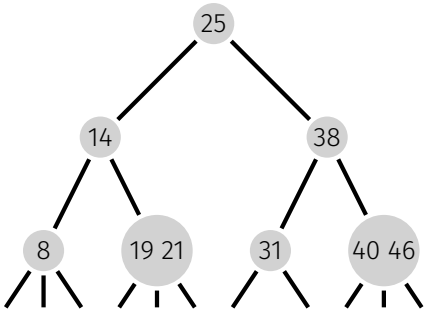


$\texttt{insert}(47)$

$\Rightarrow$

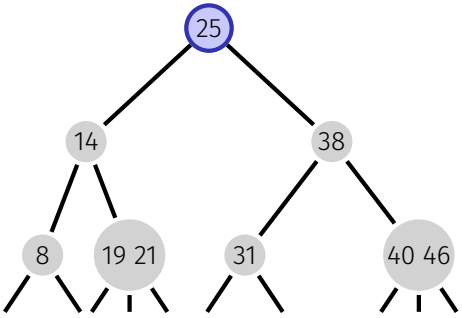


Beispiel: Einfügen in 3-Knoten

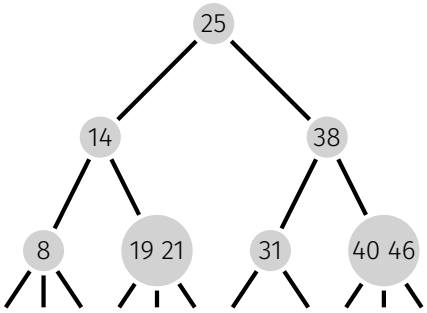


`insert`(47)

$\Rightarrow$

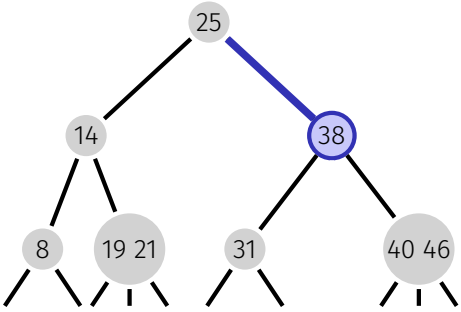


Beispiel: Einfügen in 3-Knoten

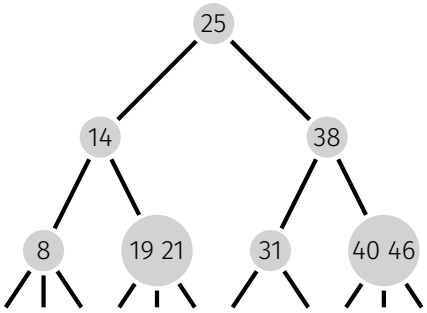


`insert`(47)

$\Rightarrow$

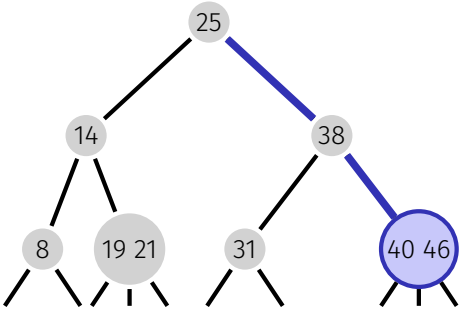


Beispiel: Einfügen in 3-Knoten

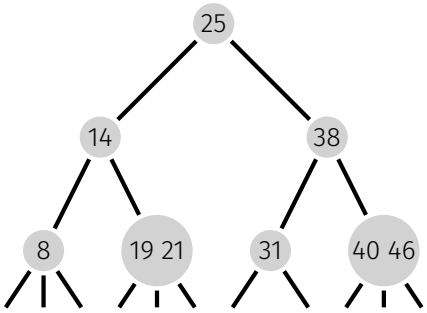


$\texttt{insert}(47)$

$\Rightarrow$

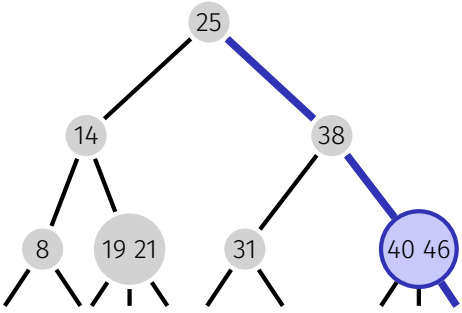


Beispiel: Einfügen in 3-Knoten

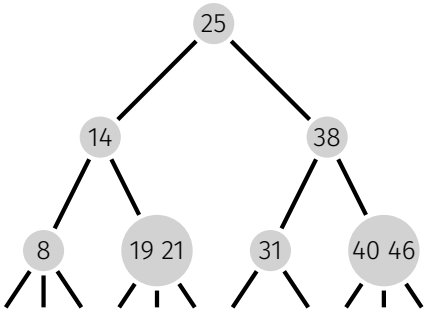


$\texttt{insert}(47)$

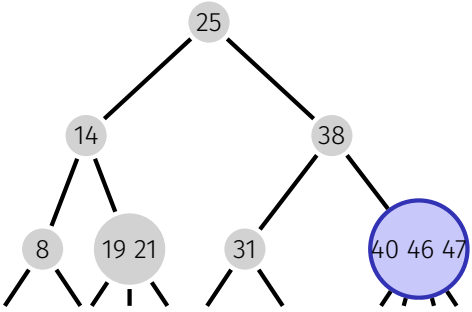
$\Rightarrow$



Beispiel: Einfügen in 3-Knoten

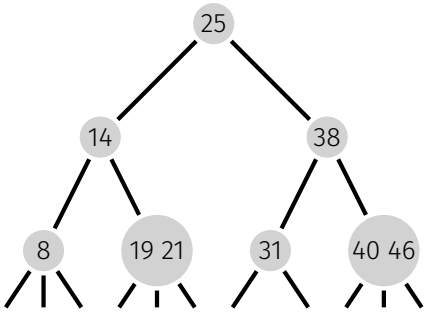


`insert`(47)
$\Rightarrow$

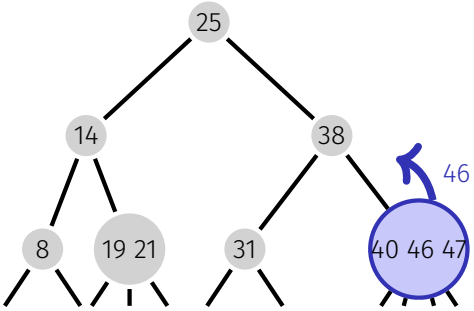


3-Knoten wird zu zwei 2-Knoten (`split` und `propagate`),

Beispiel: Einfügen in 3-Knoten

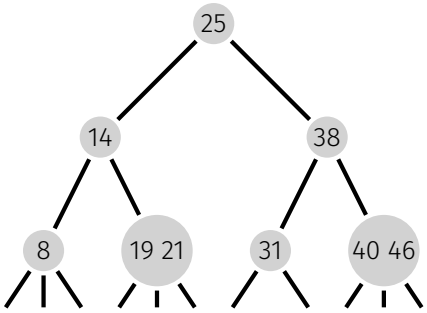


`insert`(47)
⇒

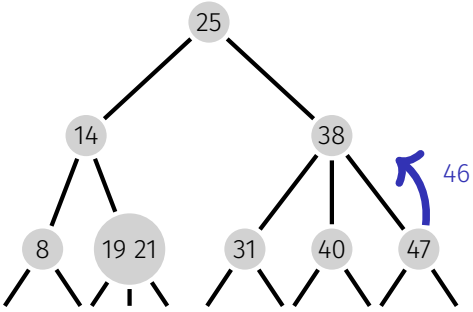


3-Knoten wird zu zwei 2-Knoten (`split` und `propagate`),

Beispiel: Einfügen in 3-Knoten

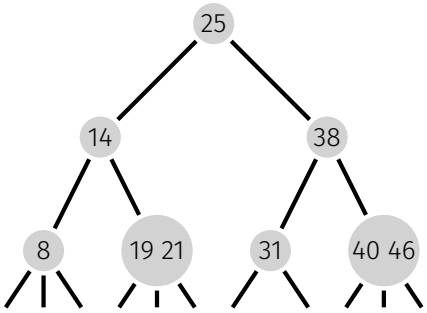


`insert`(47)
⇒

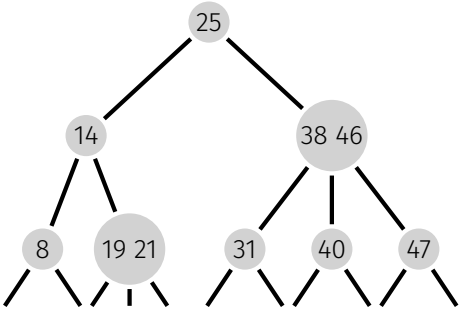


3-Knoten wird zu zwei 2-Knoten (`split` und `propagate`),

Beispiel: Einfügen in 3-Knoten

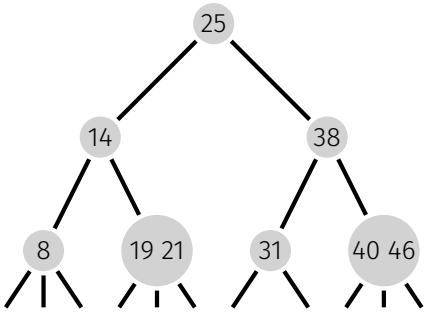


`insert`(47)
⇒

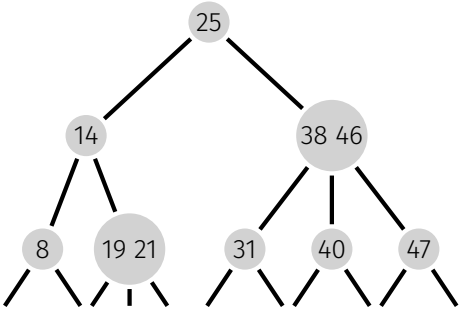


3-Knoten wird zu zwei 2-Knoten (`split` und `propagate`),
Vorgänger wird von 2-Knoten zu 3-Knoten (`join`),

Beispiel: Einfügen in 3-Knoten

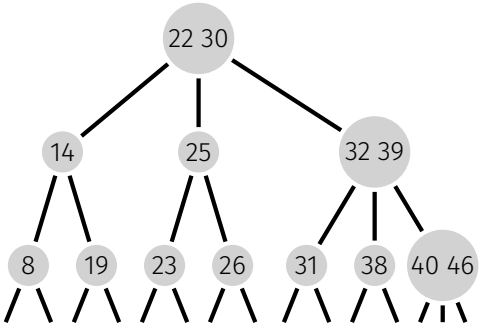


`insert`(47)
$\Rightarrow$

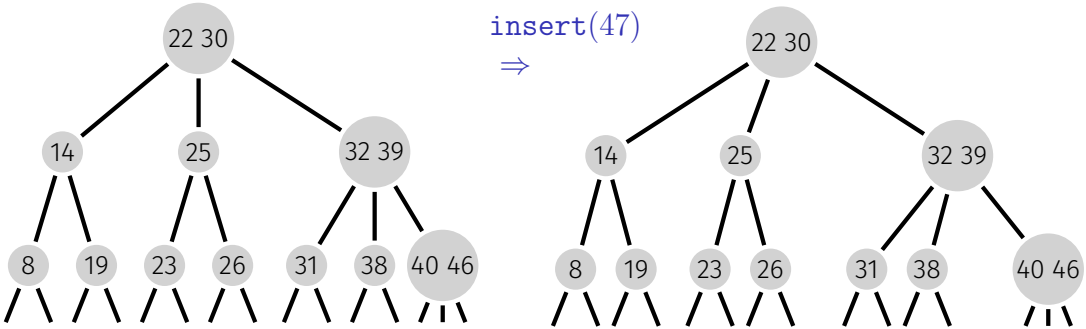


3-Knoten wird zu zwei 2-Knoten (`split` und `propagate`),
Vorgänger wird von 2-Knoten zu 3-Knoten (`join`),
Höhe bleibt gleich

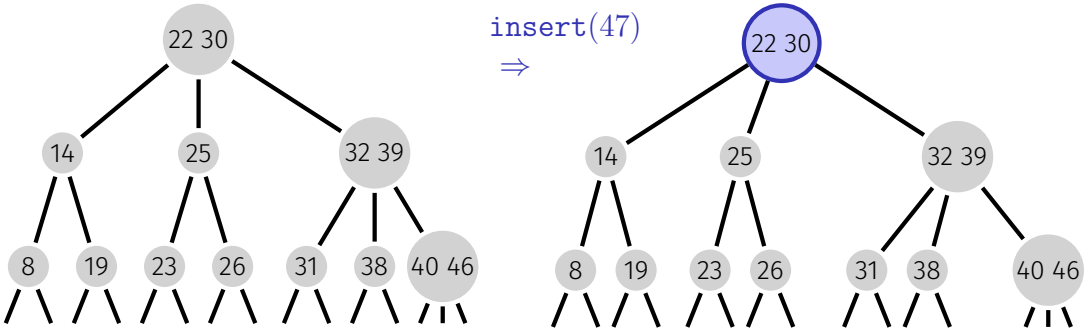
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



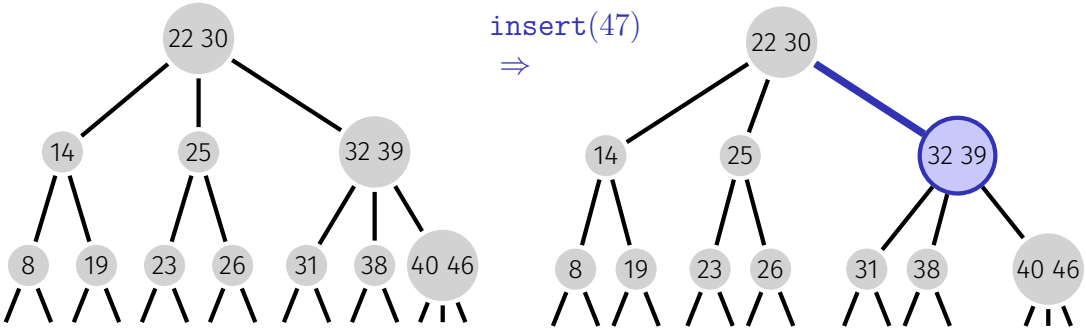
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



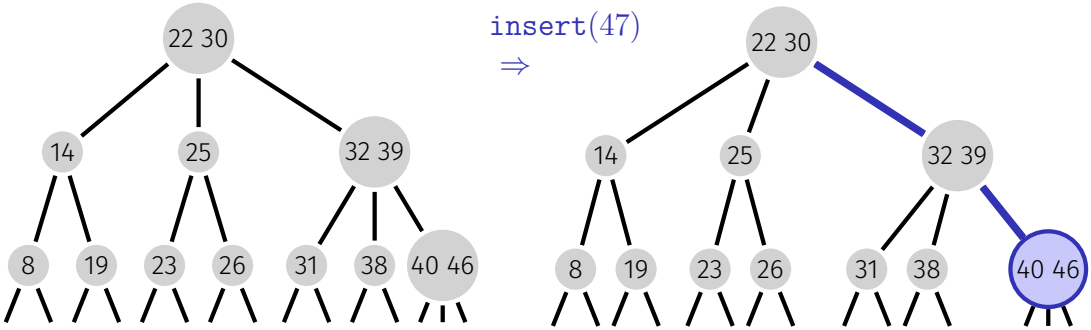
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



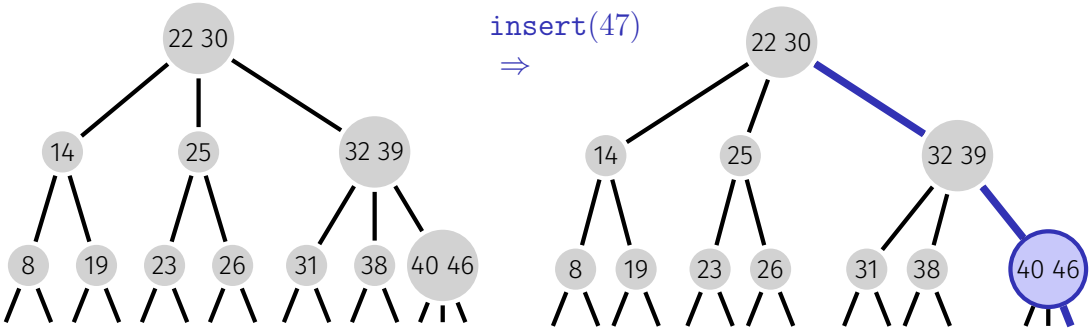
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



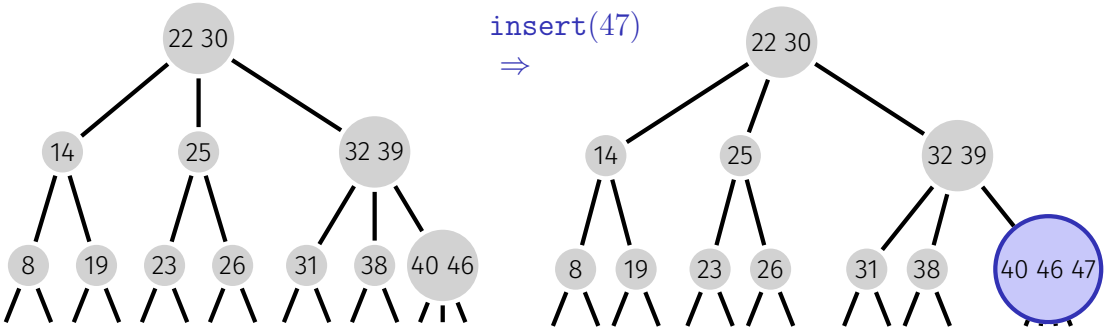
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



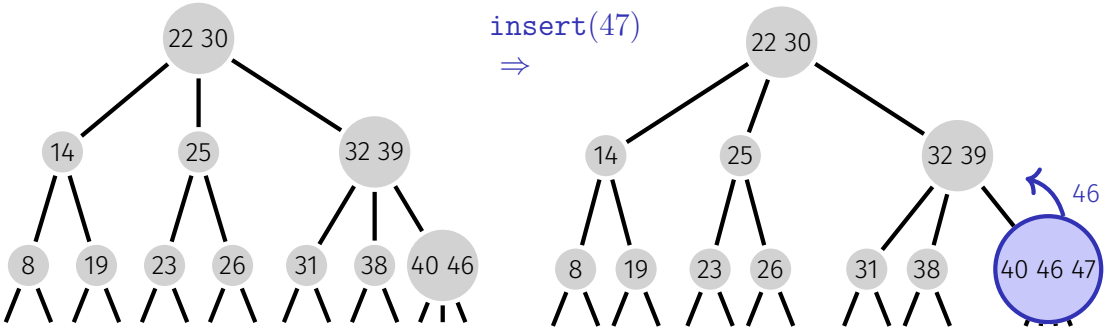
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



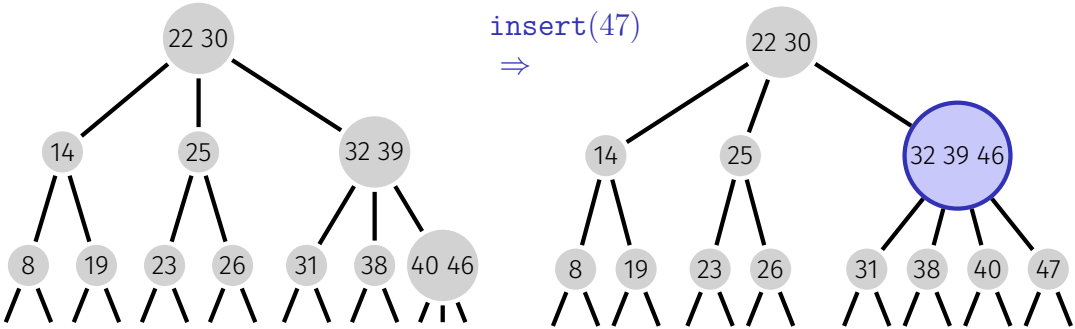
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



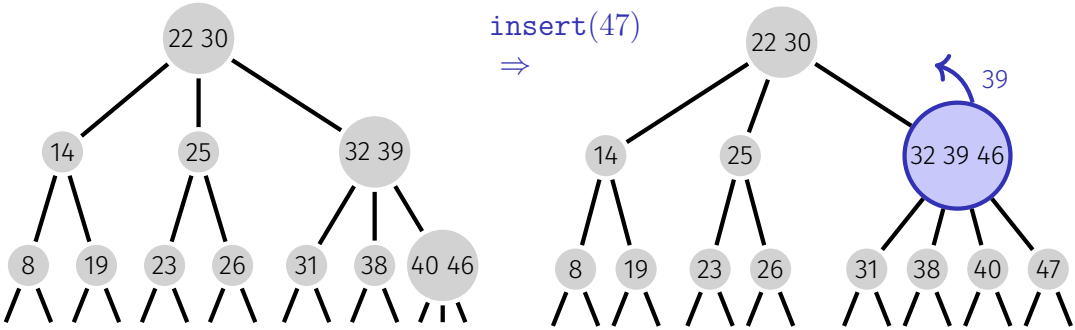
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



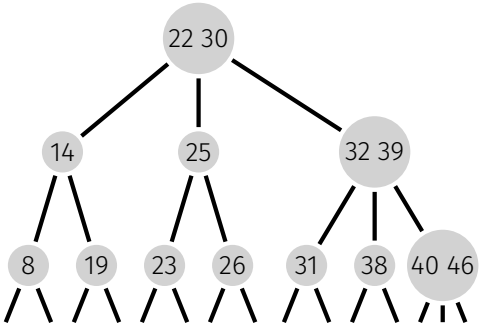
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



Beispiel: Einfügen in 3-Knoten mit Wurzel-Split

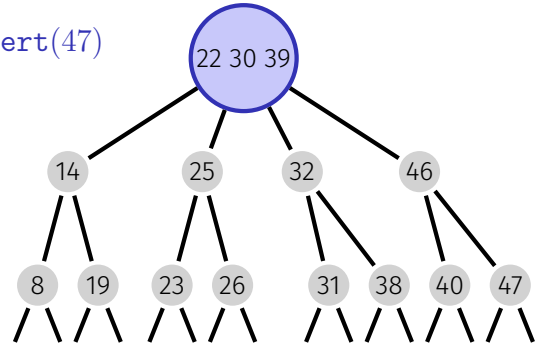


Beispiel: Einfügen in 3-Knoten mit Wurzel-Split

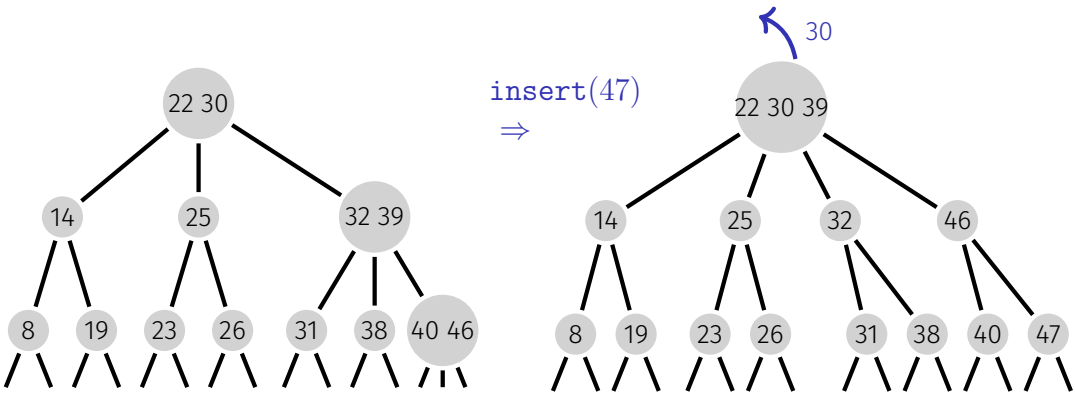


$\texttt{insert}(47)$

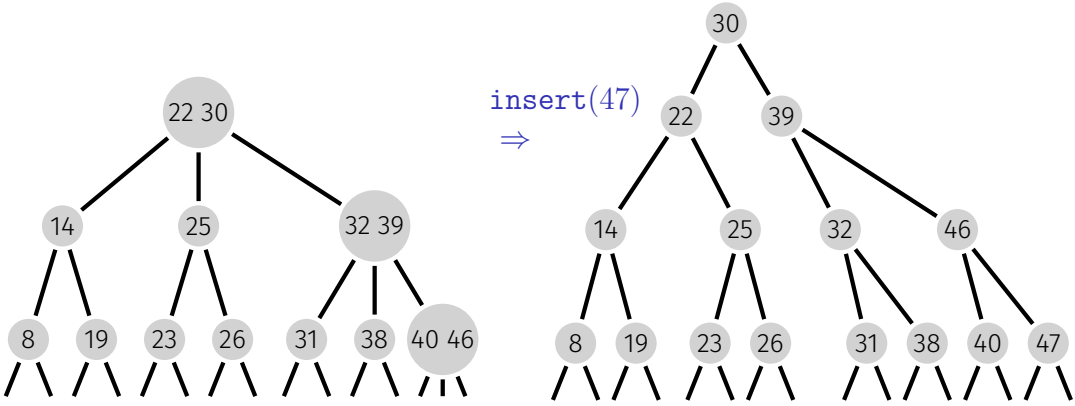
$\Rightarrow$



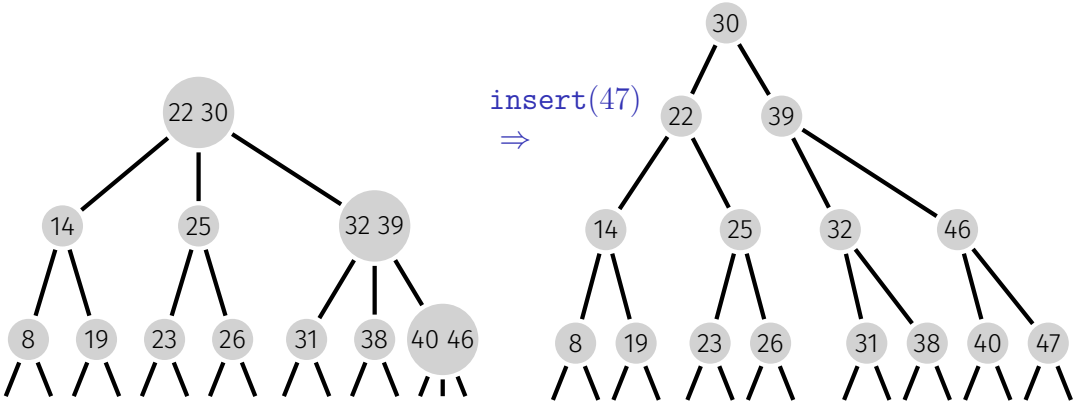
Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



Beispiel: Einfügen in 3-Knoten mit Wurzel-Split

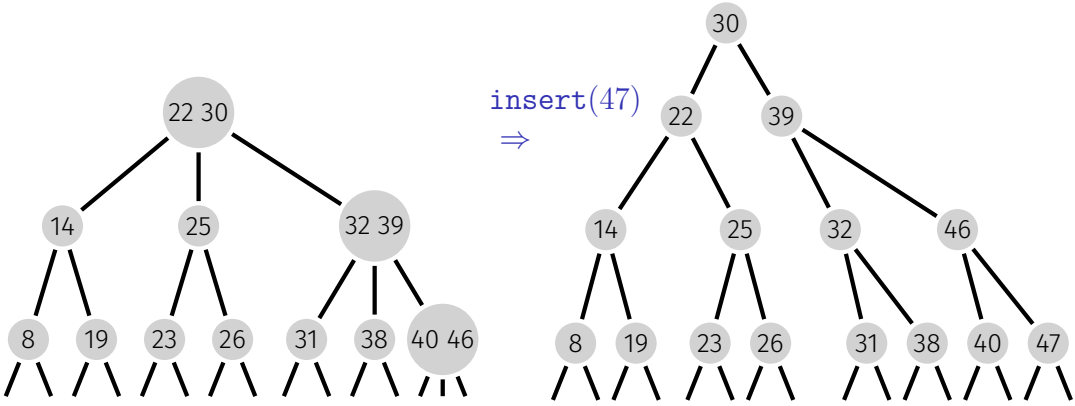


Beispiel: Einfügen in 3-Knoten mit Wurzel-Split



3-Knoten werden zu zwei 2-Knoten (`split` und `propagate`),

Beispiel: Einfügen in 3-Knoten mit Wurzel-Split

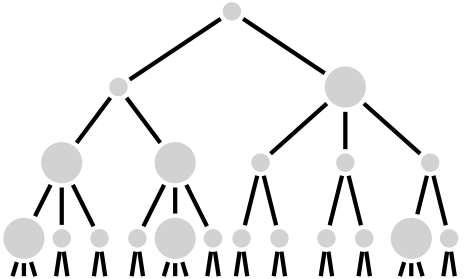


3-Knoten werden zu zwei 2-Knoten (`split` und `propagate`),
neue Wurzel ⇒ Höhe erhöht sich um 1

Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

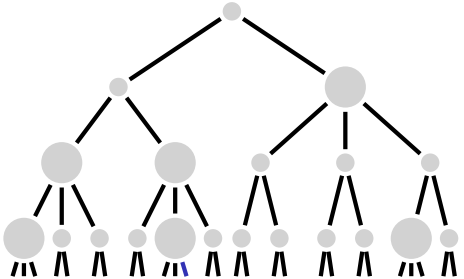
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird.



Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

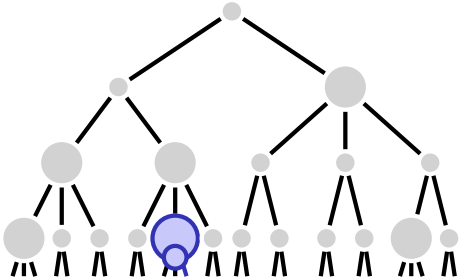
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird.



Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

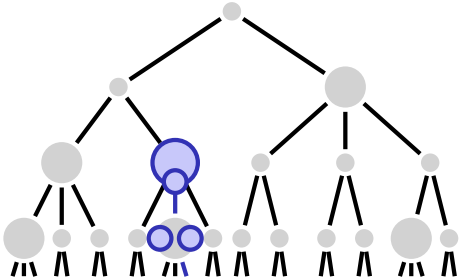
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird.



Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

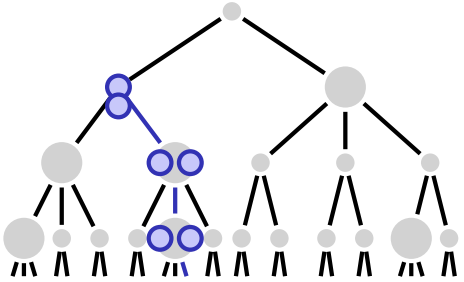
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird.



Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

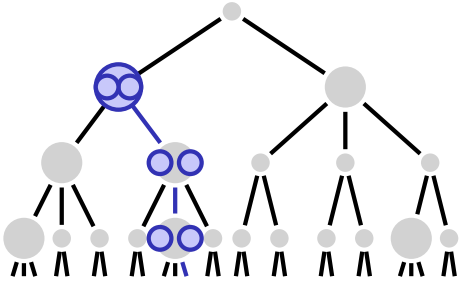
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird.



Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

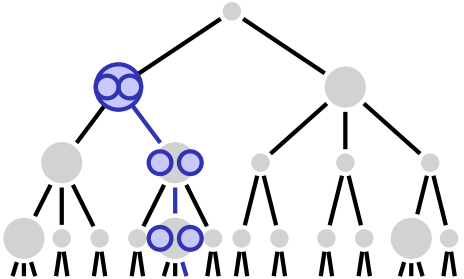
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird.



Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

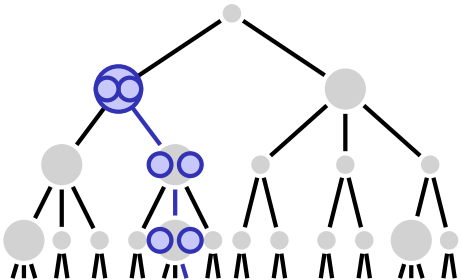
einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird. Dann entsteht ein neuer 3-Knoten mit `join`.

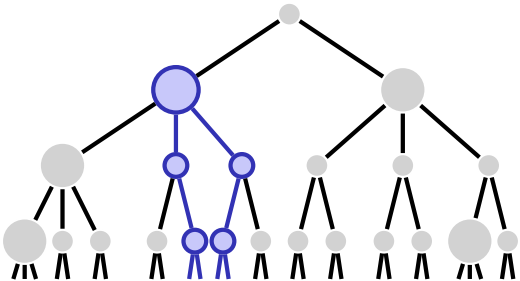


Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird. Dann entsteht ein neuer 3-Knoten mit `join`.

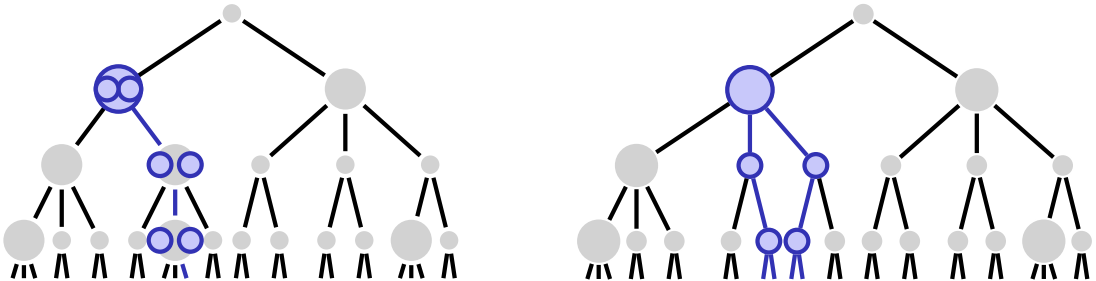




Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird. Dann entsteht ein neuer 3-Knoten mit `join`.

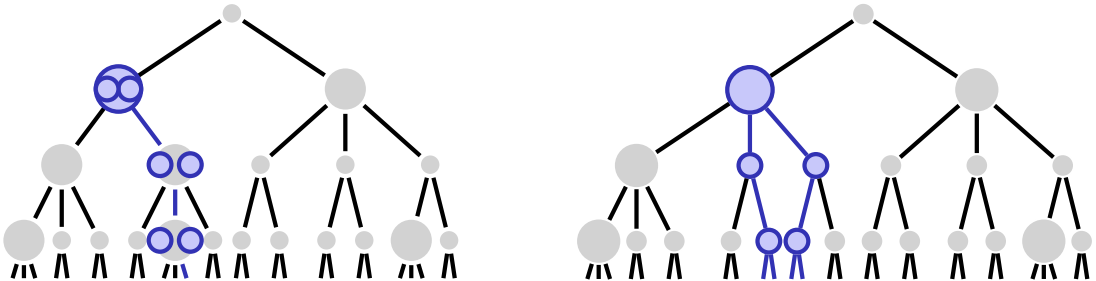


$\mathcal{O}(1)$ Zeit pro `join`- und `split`/`propagate`-Operation

Schlüssel einfügen: Zusammenfassung

Schlüssel k einfügen:

einzufügendes Element (initial k, danach jeweils mittleres Element) wird im Baum nach oben propagiert mit `split` und `propagate`, bis ein 2-Knoten oder die Wurzel erreicht wird. Dann entsteht ein neuer 3-Knoten mit `join`.



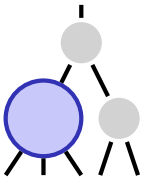
$\mathcal{O}(1)$ Zeit pro `join`- und `split`/`propagate`-Operation

$\mathcal{O}(\log n)$ viele `split`/`propagate`-Operationen, eine `join`-Operation

Schlüssel in Knoten ohne Kinder löschen

Schlüssel in Knoten ohne Kinder löschen

- 3-Knoten ohne Kinder



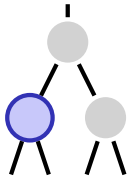
Schlüssel in Knoten ohne Kinder löschen

- 3-Knoten ohne Kinder $\Rightarrow$ wird zu 2-Knoten ohne Kinder

Schlüssel in Knoten ohne Kinder löschen

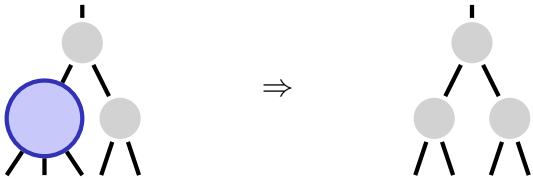
- 3-Knoten ohne Kinder $\Rightarrow$ wird zu 2-Knoten ohne Kinder

- 2-Knoten ohne Kinder



Schlüssel in Knoten ohne Kinder löschen

- 3-Knoten ohne Kinder $\Rightarrow$ wird zu 2-Knoten ohne Kinder



- 2-Knoten ohne Kinder $\Rightarrow$ unbalancierter Baum

Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach

Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach
Können wir sicherstellen, dass k in einem 3-Knoten ohne Kinder ist?

Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach
Können wir sicherstellen, dass k in einem 3-Knoten ohne Kinder ist?

1. Garantie, dass der Knoten mit k keine Kinder hat

Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach
Können wir sicherstellen, dass k in einem 3-Knoten ohne Kinder ist?

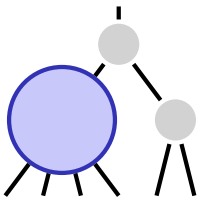
1. Garantie, dass der Knoten mit k keine Kinder hat
2. Garantie, dass k in 3-Knoten oder (temporärem) 4-Knoten ist

Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach
Können wir sicherstellen, dass k in einem 3-Knoten ohne Kinder ist?

1. Garantie, dass der Knoten mit k keine Kinder hat
2. Garantie, dass k in 3-Knoten oder (temporärem) 4-Knoten ist

Löschen von einem 4-Knoten ohne Kinder $\Rightarrow$ wird zu 3-Knoten-Blatt

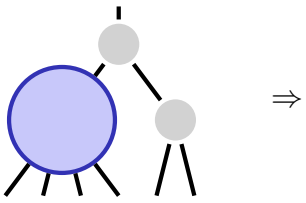


Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach
Können wir sicherstellen, dass k in einem 3-Knoten ohne Kinder ist?

1. Garantie, dass der Knoten mit k keine Kinder hat
2. Garantie, dass k in 3-Knoten oder (temporärem) 4-Knoten ist

Löschen von einem 4-Knoten ohne Kinder $\Rightarrow$ wird zu 3-Knoten-Blatt



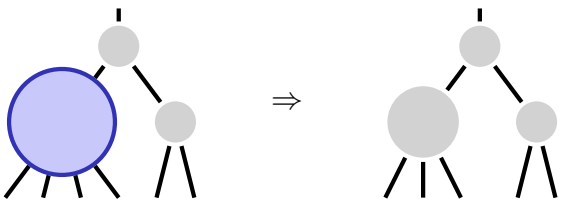
$\Rightarrow$

Löschen: Idee

Beobachtung: Löschen von Schlüssel k in 3-Knoten ohne Kinder ist einfach
Können wir sicherstellen, dass k in einem 3-Knoten ohne Kinder ist?

1. Garantie, dass der Knoten mit k keine Kinder hat
2. Garantie, dass k in 3-Knoten oder (temporärem) 4-Knoten ist

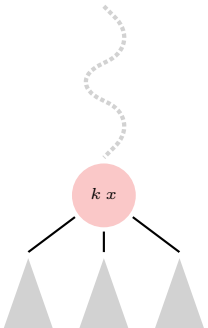
Löschen von einem 4-Knoten ohne Kinder $\Rightarrow$ wird zu 3-Knoten-Blatt



Löschen: Überblick

1. Garantie, dass der Knoten keine Kinder hat

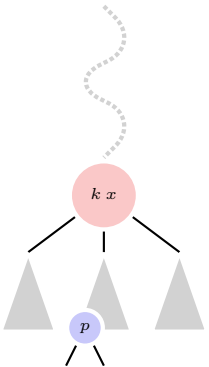
falls nötig, Schlüssel k mit symmetrischem Nachfolger p tauschen



Löschen: Überblick

1. Garantie, dass der Knoten keine Kinder hat

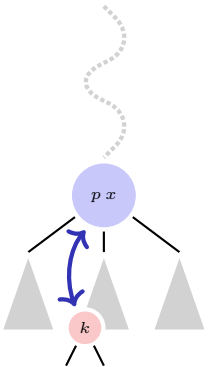
falls nötig, Schlüssel k mit symmetrischem Nachfolger p tauschen



Löschen: Überblick

1. Garantie, dass der Knoten keine Kinder hat

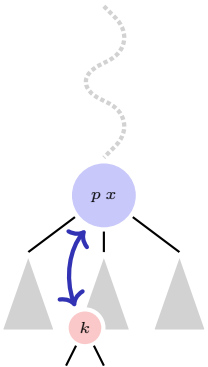
falls nötig, Schlüssel k mit symmetrischem Nachfolger p tauschen



Löschen: Überblick

1. Garantie, dass der Knoten keine Kinder hat

falls nötig, Schlüssel k mit symmetrischem Nachfolger p tauschen



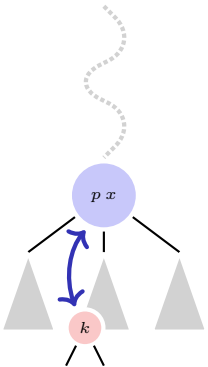
2. Garantie, dass 3- oder 4-Knoten

- k in 3- oder 4-Knoten ohne Kinder

Löschen: Überblick

1. Garantie, dass der Knoten keine Kinder hat

falls nötig, Schlüssel k mit symmetrischem Nachfolger p tauschen



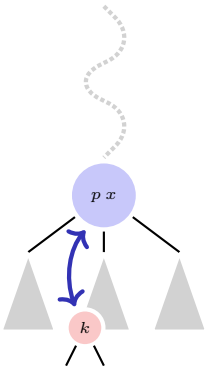
2. Garantie, dass 3- oder 4-Knoten

- k in 3- oder 4-Knoten ohne Kinder

Löschen: Überblick

1. Garantie, dass der Knoten keine Kinder hat

falls nötig, Schlüssel k mit symmetrischem Nachfolger p tauschen



2. Garantie, dass 3- oder 4-Knoten

- k in 3- oder 4-Knoten ohne Kinder
- **4-Knoten** auf Pfad zu Wurzel

Löschen: Überblick

3. Schlüssel löschen

$\Rightarrow$ resultiert in 2- oder 3-Knoten

Löschen: Überblick

3. Schlüssel löschen

$\Rightarrow$ resultiert in 2- oder 3-Knoten

Löschen: Überblick

3. Schlüssel löschen

$\Rightarrow$ resultiert in 2- oder 3-Knoten

4. Aufräumen der 4-Knoten

Löschen: Überblick

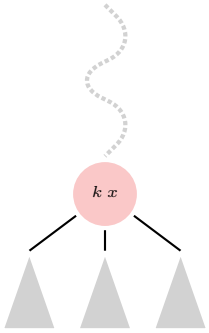
3. Schlüssel löschen

$\Rightarrow$ resultiert in 2- oder 3-Knoten

4. Aufräumen der 4-Knoten

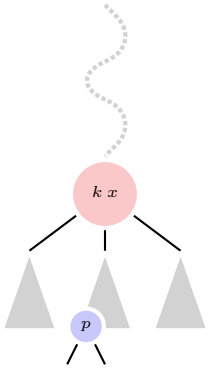
mit `split`/`propagate`

1. Garantie, dass Knoten ohne Kinder



Schlüssel k mit symmetrischem Nachfolger p tauschen

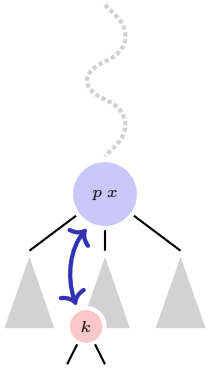
1. Garantie, dass Knoten ohne Kinder



Schlüssel k mit symmetrischem Nachfolger p tauschen

- symmetrischer Nachfolger kann in $\mathcal{O}(\log n)$ Zeit gefunden werden

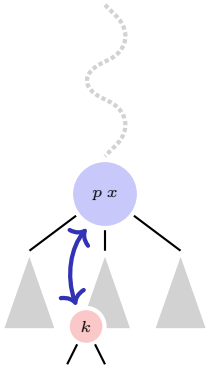
1. Garantie, dass Knoten ohne Kinder



Schlüssel k mit symmetrischem Nachfolger p tauschen

- symmetrischer Nachfolger kann in $\mathcal{O}(\log n)$ Zeit gefunden werden
- symmetrischer Nachfolger ist immer in einem Knoten ohne Kinder

1. Garantie, dass Knoten ohne Kinder

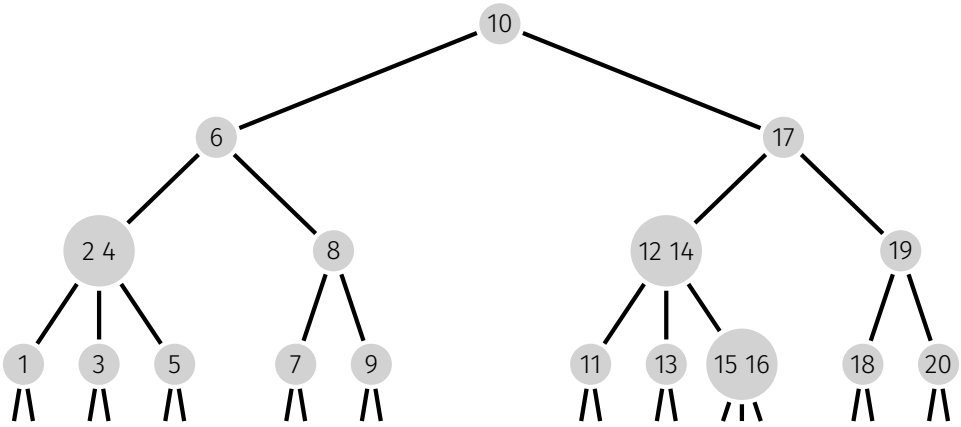


Schlüssel k mit symmetrischem Nachfolger p tauschen

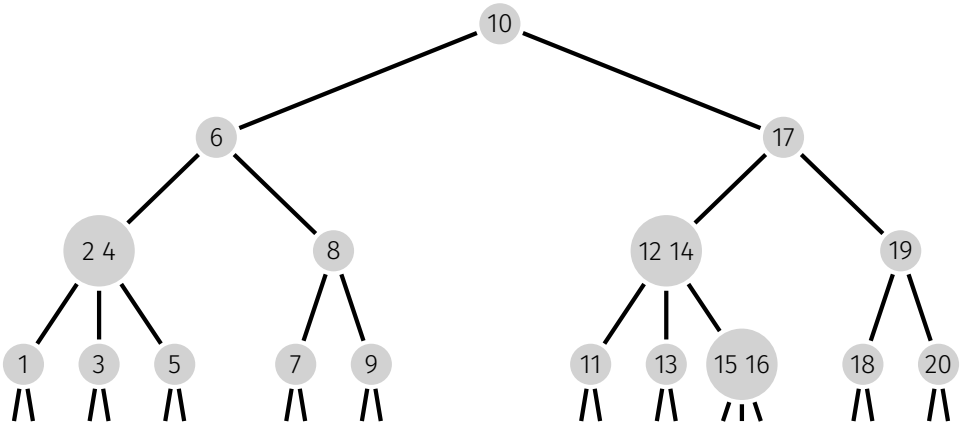
- symmetrischer Nachfolger kann in $\mathcal{O}(\log n)$ Zeit gefunden werden
- symmetrischer Nachfolger ist immer in einem Knoten ohne Kinder

$\Rightarrow$ Schlüssel in Knoten ohne Kinder in $\mathcal{O}(\log n)$ Zeit

Beispiel: Schritt 1

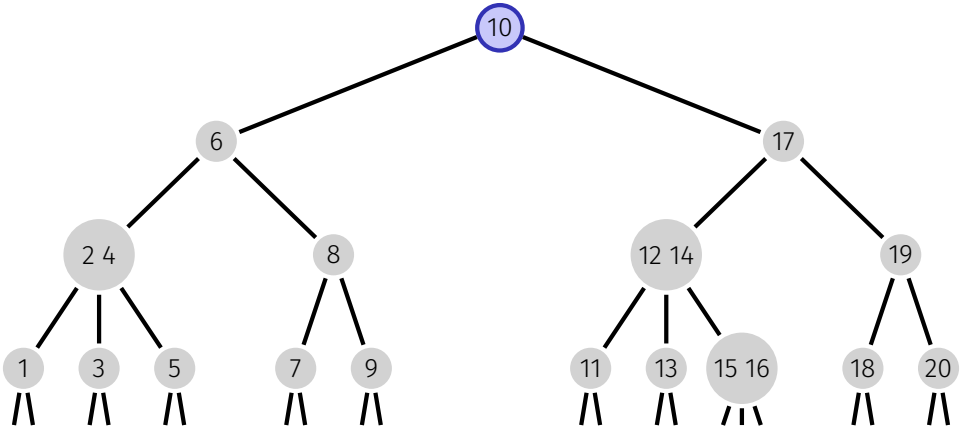


Beispiel: Schritt 1



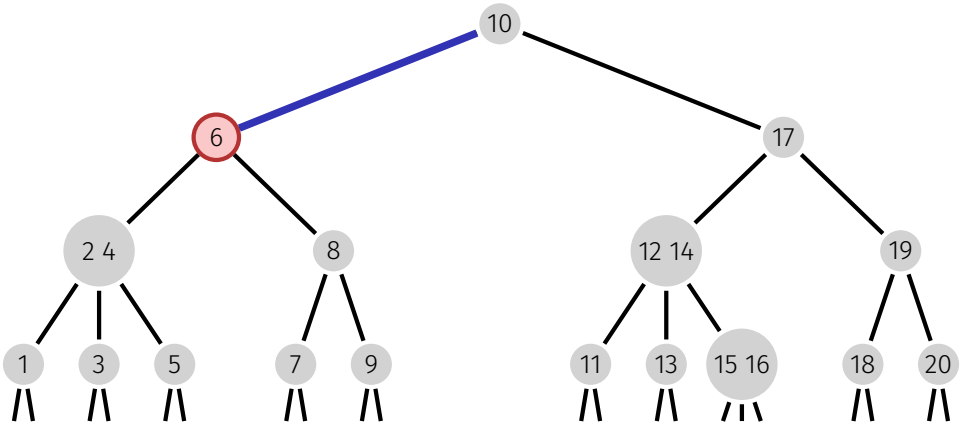
`remove`(6)

Beispiel: Schritt 1



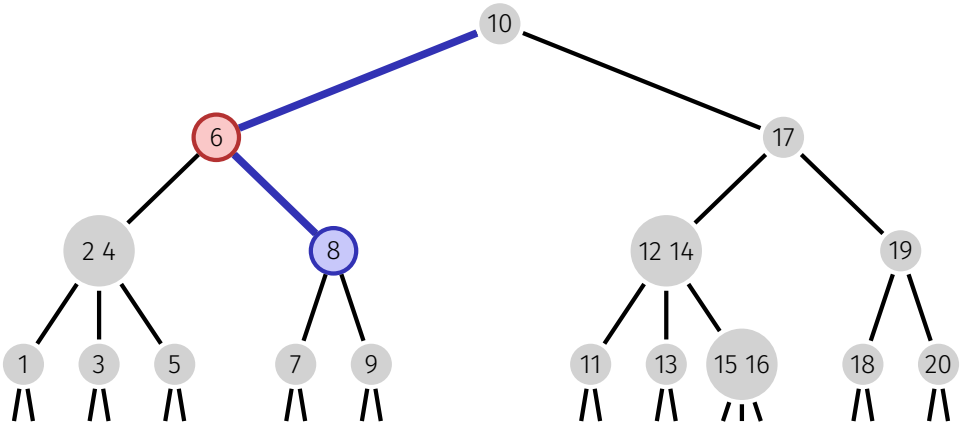
`remove`(6)

Beispiel: Schritt 1



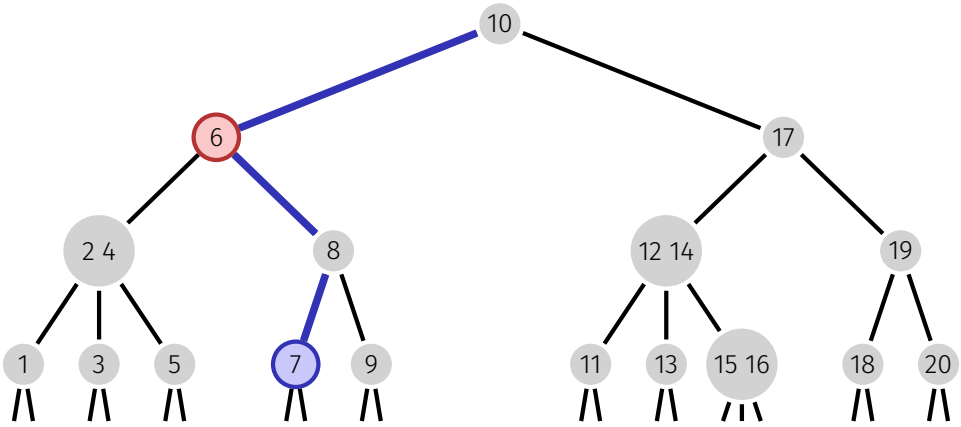
`remove`(6)

Beispiel: Schritt 1



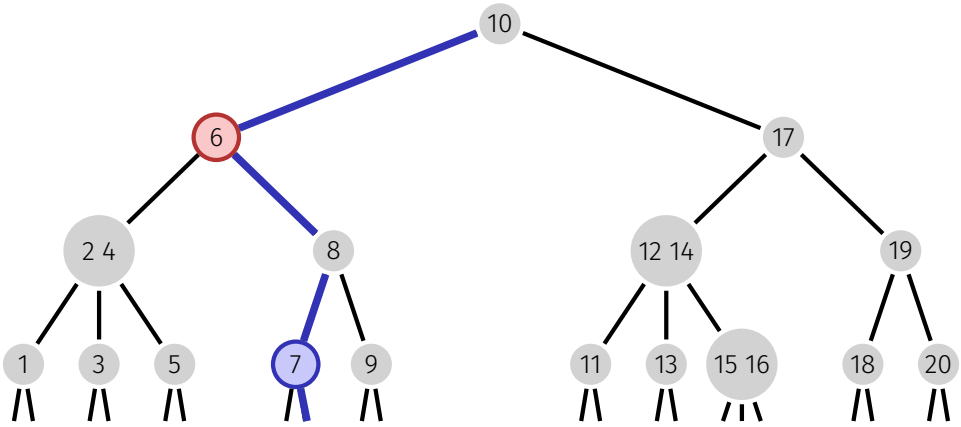
`remove`(6)

Beispiel: Schritt 1



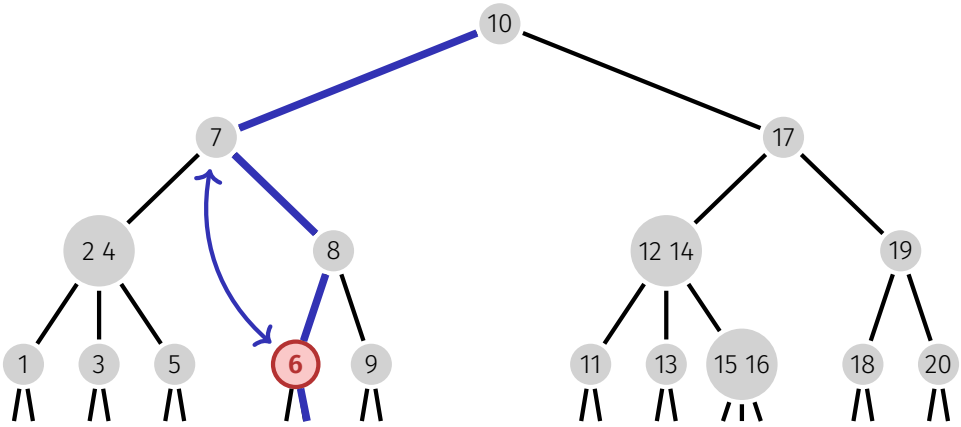
`remove`(6)

Beispiel: Schritt 1



`remove`(6)

Beispiel: Schritt 1



`remove`(6)

2. Knoten in 3- oder 4-Knoten umwandeln

Knoten mit k in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

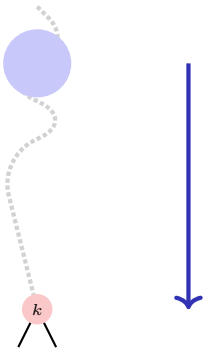
Knoten mit k in 3- oder 4-Knoten umwandeln

- Starte bei unterstem 3- oder 4-Knoten (oder Wurzel) auf dem Weg von der Wurzel zu k

2. Knoten in 3- oder 4-Knoten umwandeln

Knoten mit k in 3- oder 4-Knoten umwandeln

- Starte bei unterstem 3- oder 4-Knoten (oder Wurzel) auf dem Weg von der Wurzel zu k
- bringe zusätzlichen Schlüssel Schritt für Schritt nach unten



2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

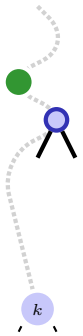
2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln



2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

2. Knoten in 3- oder 4-Knoten umwandeln

Invariante: In jedem Schritt auf dem Weg nach unten:

- der aktuelle Knoten ist ein 3- oder 4-Knoten

2. Knoten in 3- oder 4-Knoten umwandeln

Invariante: In jedem Schritt auf dem Weg nach unten:

- der aktuelle Knoten ist ein 3- oder 4-Knoten
- nur Knoten auf Pfad zur Wurzel sind möglicherweise 4-Knoten

2. Knoten in 3- oder 4-Knoten umwandeln

Invariante: In jedem Schritt auf dem Weg nach unten:

- der aktuelle Knoten ist ein 3- oder 4-Knoten
- nur Knoten auf Pfad zur Wurzel sind möglicherweise 4-Knoten
- Baum bleibt perfekt balanciert

2. Knoten in 3- oder 4-Knoten umwandeln

Wir nehmen im Folgenden an, dass wir bei der Suche immer nach links gehen. Die Fälle Mitte und rechts können analog behandelt werden.

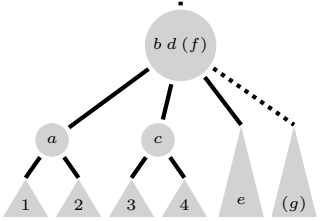
Invariante: In jedem Schritt auf dem Weg nach unten:

- der aktuelle Knoten ist ein 3- oder 4-Knoten
- nur Knoten auf Pfad zur Wurzel sind möglicherweise 4-Knoten
- Baum bleibt perfekt balanciert

2. Iterative Transformation

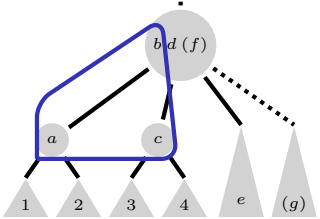
2. Iterative Transformation

- rechtes Kind ist 2-Knoten



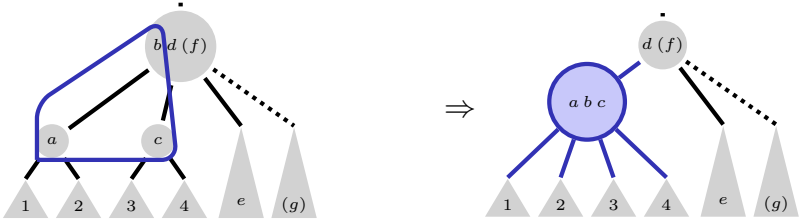
2. Iterative Transformation

- rechtes Kind ist 2-Knoten



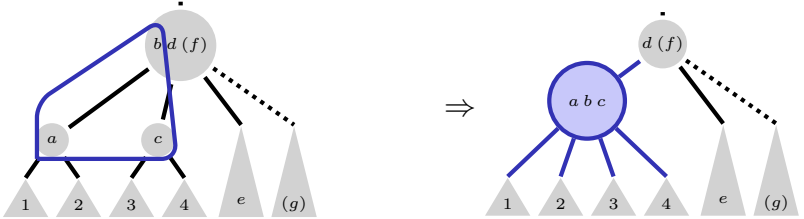
2. Iterative Transformation

- rechtes Kind ist 2-Knoten $\Rightarrow$ `merge` 3 Schlüssel zu 4-Knoten

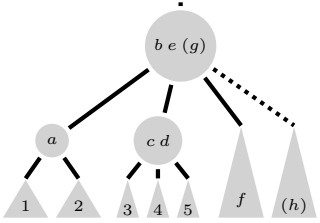


2. Iterative Transformation

- rechtes Kind ist 2-Knoten $\Rightarrow$ `merge` 3 Schlüssel zu 4-Knoten

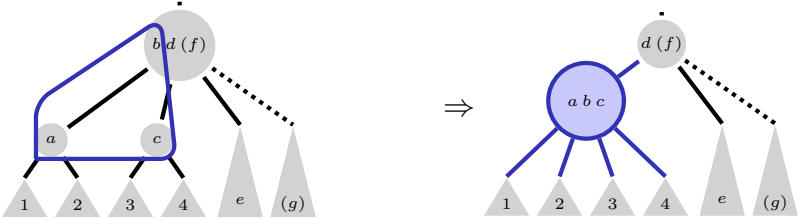


- rechtes Kind ist 3-Knoten

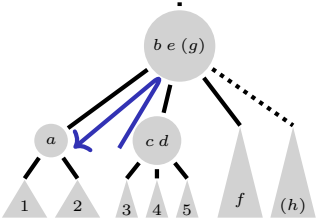


2. Iterative Transformation

- rechtes Kind ist 2-Knoten $\Rightarrow$ `merge` 3 Schlüssel zu 4-Knoten

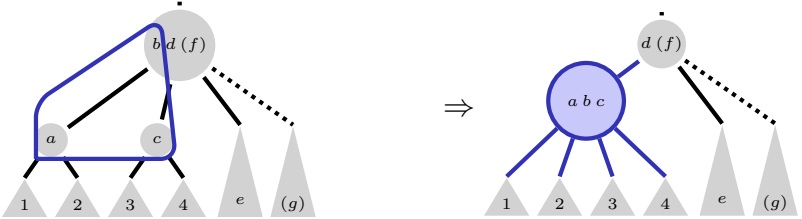


- rechtes Kind ist 3-Knoten

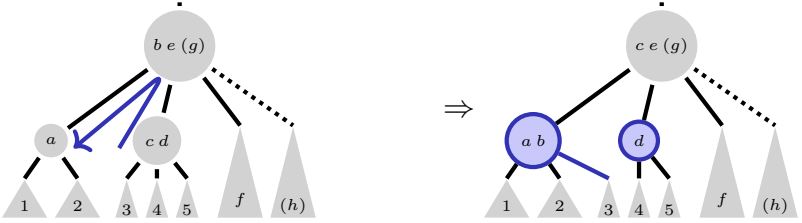


2. Iterative Transformation

- rechtes Kind ist 2-Knoten ⇒ `merge` 3 Schlüssel zu 4-Knoten



- rechtes Kind ist 3-Knoten ⇒ von rechtem Geschwister stehlen



2. Wurzel in 3- oder 4-Knoten umwandeln

2. Wurzel in 3- oder 4-Knoten umwandeln

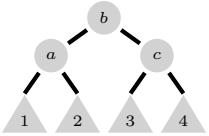
- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig

2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig

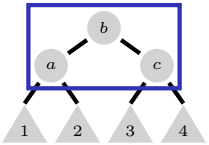
2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- beide Kinder 2-Knoten



2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- beide Kinder 2-Knoten



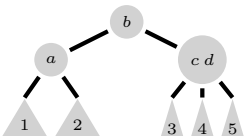
2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- beide Kinder 2-Knoten $\Rightarrow$ `merge` drei Knoten zu einer 4-Knoten-Wurzel

2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- beide Kinder 2-Knoten $\Rightarrow$ `merge` drei Knoten zu einer 4-Knoten-Wurzel

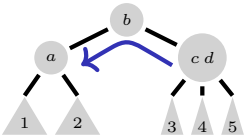
- rechtes Kind ist 3-Knoten



2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- beide Kinder 2-Knoten $\Rightarrow$ `merge` drei Knoten zu einer 4-Knoten-Wurzel

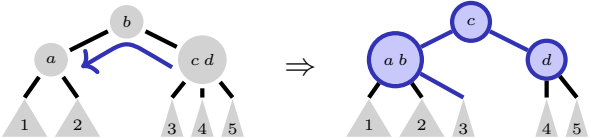
- rechtes Kind ist 3-Knoten



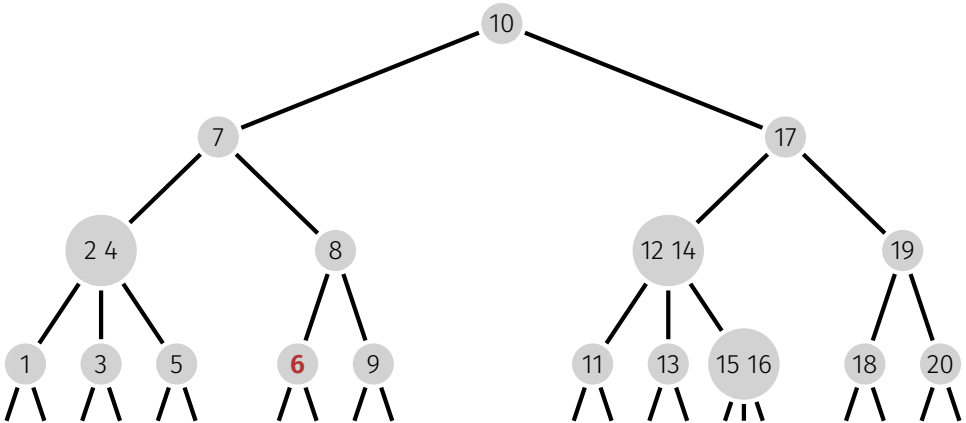
2. Wurzel in 3- oder 4-Knoten umwandeln

- Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- linkes Kind von Wurzel ist ein 3-Knoten $\Rightarrow$ fertig
- beide Kinder 2-Knoten $\Rightarrow$ `merge` drei Knoten zu einer 4-Knoten-Wurzel

- rechtes Kind ist 3-Knoten $\Rightarrow$ von rechtem Geschwister stehlen

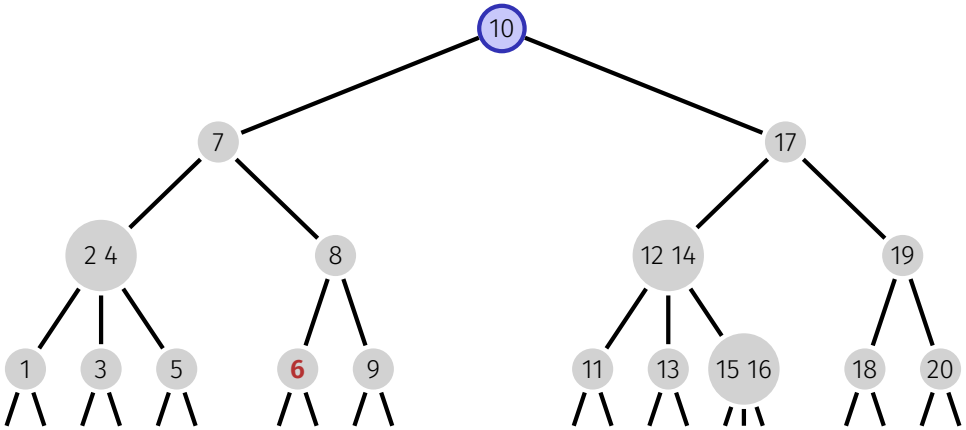


Beispiel: Schritt 2



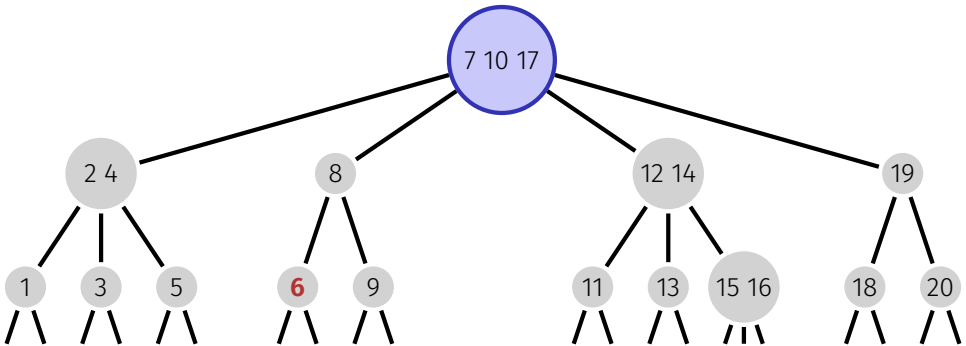
`remove`(6)

Beispiel: Schritt 2



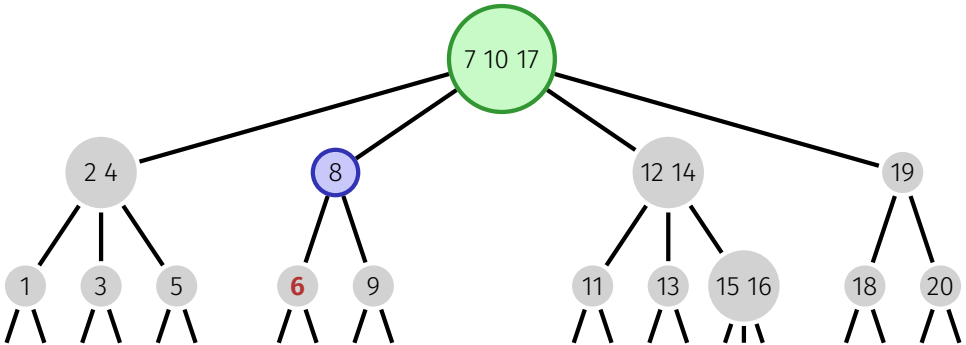
`remove`(6)

Beispiel: Schritt 2



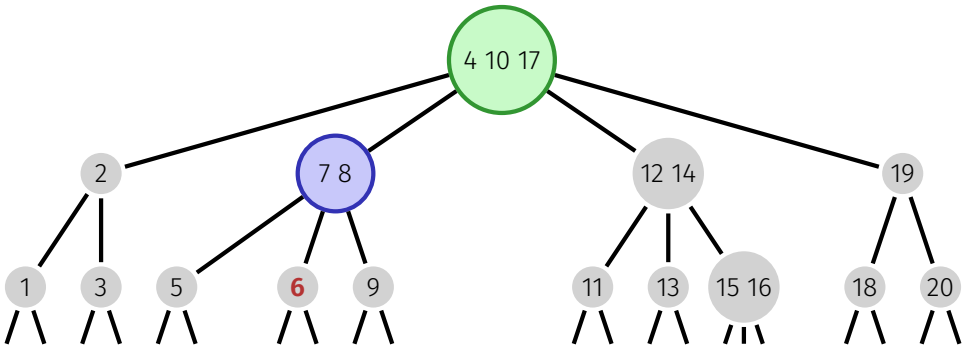
`remove`(6)

Beispiel: Schritt 2



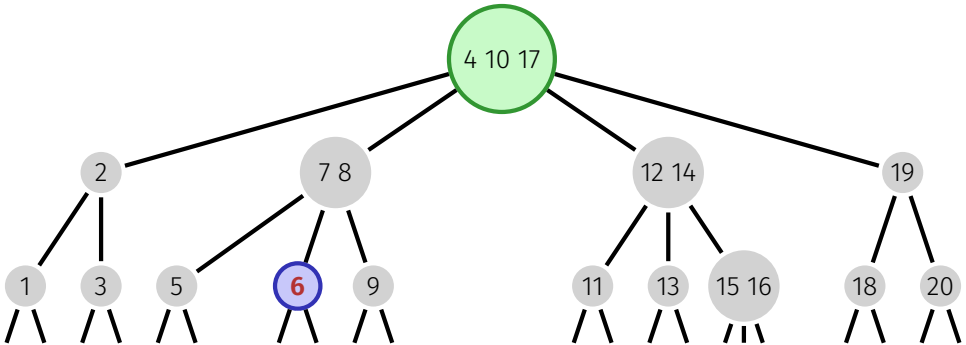
`remove`(6)

Beispiel: Schritt 2



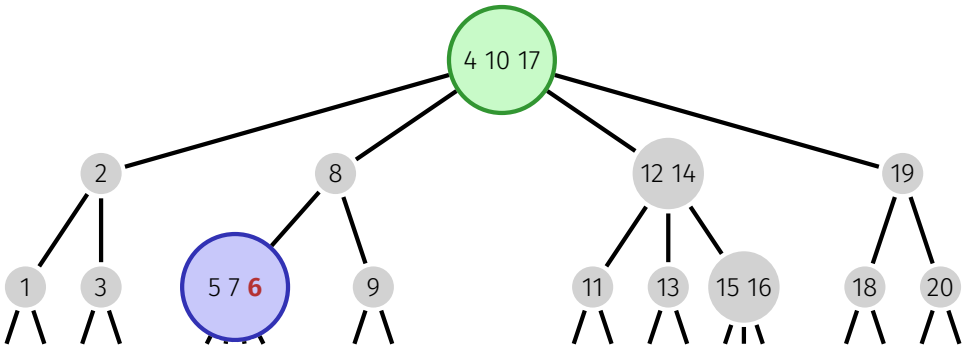
`remove`(6)

Beispiel: Schritt 2



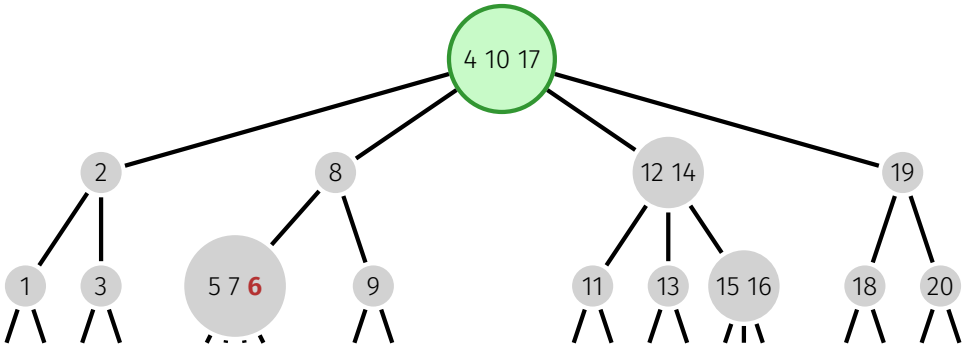
`remove`(6)

Beispiel: Schritt 2



`remove`(6)

Beispiel: Schritt 2



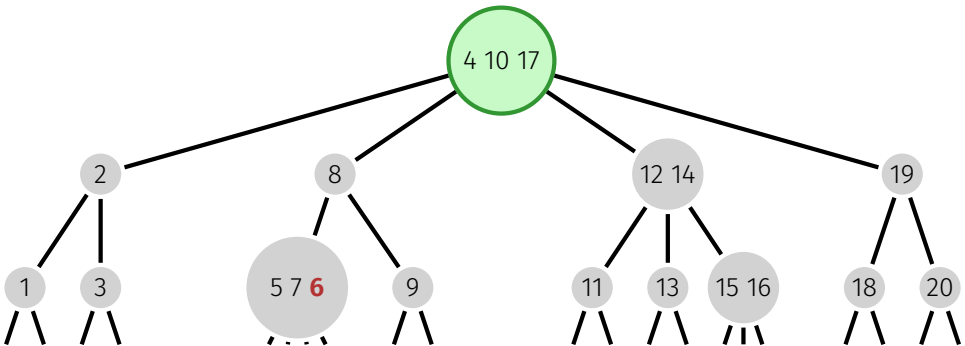
`remove`(6)

3. Schlüssel löschen

einfach, da sich Schlüssel in einem 3- oder 4-Knoten ohne Kinder befindet

3. Schlüssel löschen

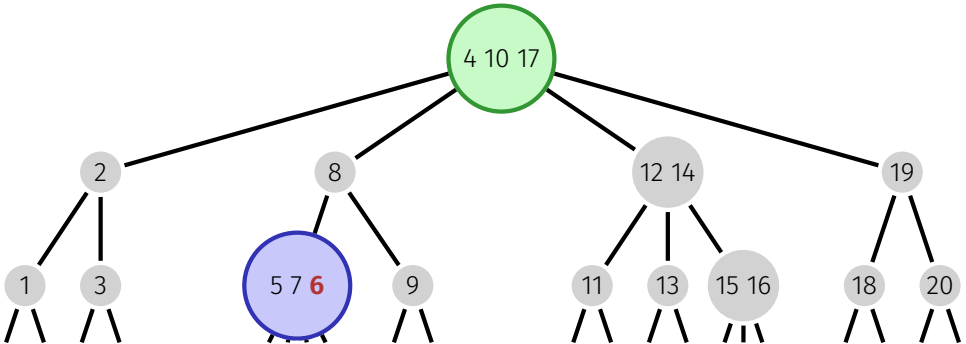
einfach, da sich Schlüssel in einem 3- oder 4-Knoten ohne Kinder befindet



`remove`(6)

3. Schlüssel löschen

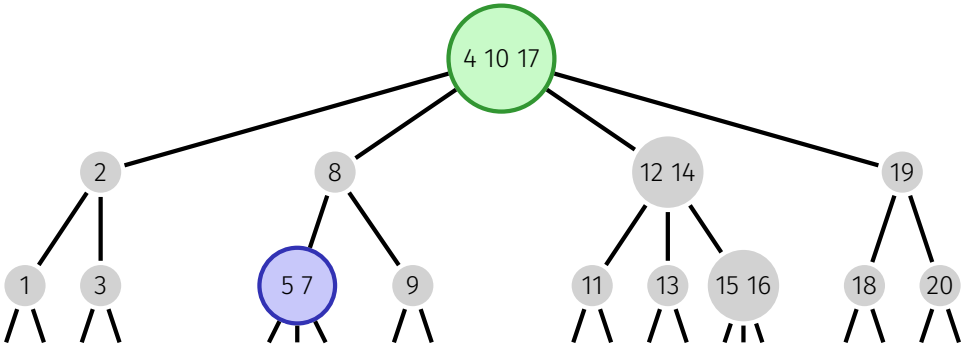
einfach, da sich Schlüssel in einem 3- oder 4-Knoten ohne Kinder befindet



`remove`(6)

3. Schlüssel löschen

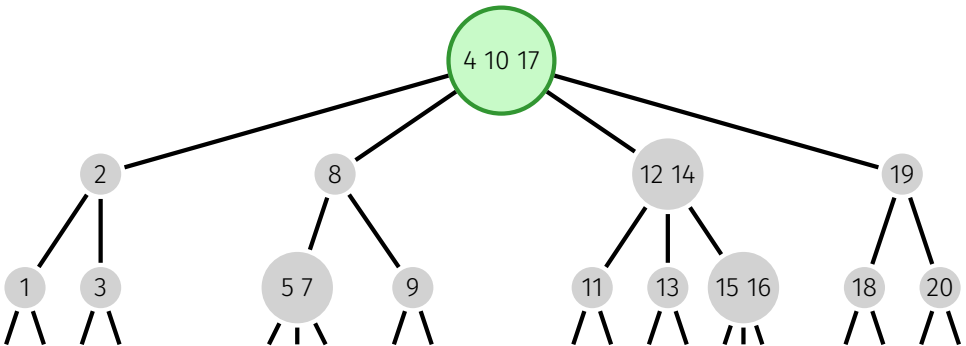
einfach, da sich Schlüssel in einem 3- oder 4-Knoten ohne Kinder befindet



`remove`(6)

3. Schlüssel löschen

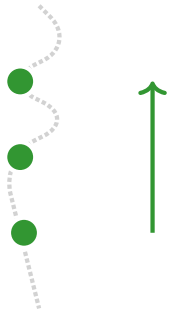
einfach, da sich Schlüssel in einem 3- oder 4-Knoten ohne Kinder befindet



`remove`(6)

4. Aufräumen der 4-Knoten

4. Aufräumen der 4-Knoten

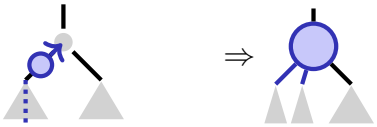


4-Knoten auf Weg nach oben aufräumen mit `split` und `propagate`

4. Aufräumen der 4-Knoten: `split` und `propagate`

4. Aufräumen der 4-Knoten: `split` und `propagate`

- Einfügen in 2-Knoten $\Rightarrow$ wird zu 3-Knoten (`join`)



4. Aufräumen der 4-Knoten: `split` und `propagate`

- Einfügen in 2-Knoten $\Rightarrow$ wird zu 3-Knoten (`join`)

4. Aufräumen der 4-Knoten: `split` und `propagate`

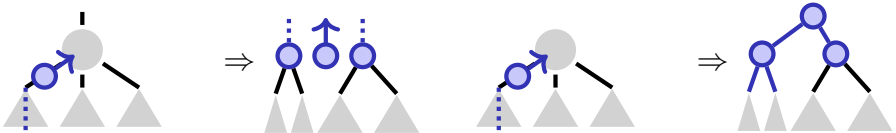
- Einfügen in 2-Knoten $\Rightarrow$ wird zu 3-Knoten (`join`)

- Einfügen in 3-Knoten $\Rightarrow$ Aufteilen in zwei 2-Knoten

4. Aufräumen der 4-Knoten: `split` und `propagate`

- Einfügen in 2-Knoten $\Rightarrow$ wird zu 3-Knoten (`join`)

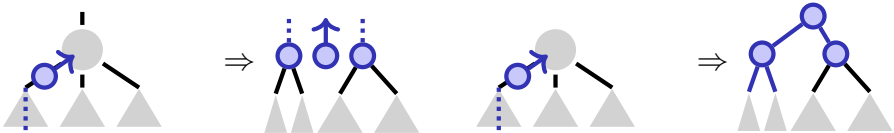
- Einfügen in 3-Knoten $\Rightarrow$ Aufteilen in zwei 2-Knoten



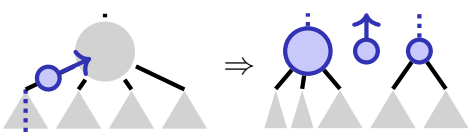
4. Aufräumen der 4-Knoten: `split` und `propagate`

- Einfügen in 2-Knoten $\Rightarrow$ wird zu 3-Knoten (`join`)

- Einfügen in 3-Knoten $\Rightarrow$ Aufteilen in zwei 2-Knoten



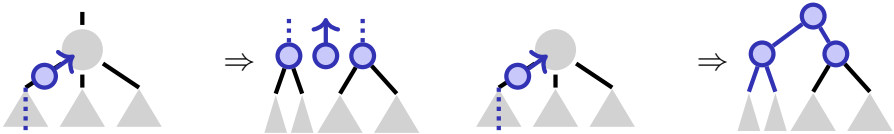
- Einfügen in 4-Knoten $\Rightarrow$ Aufteilen in 2-Knoten und 3-Knoten



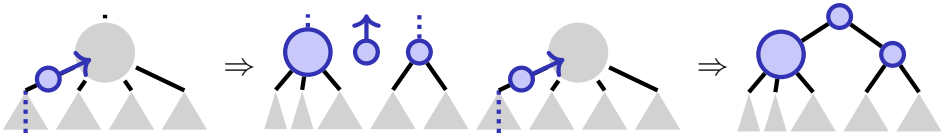
4. Aufräumen der 4-Knoten: `split` und `propagate`

- Einfügen in 2-Knoten $\Rightarrow$ wird zu 3-Knoten (`join`)

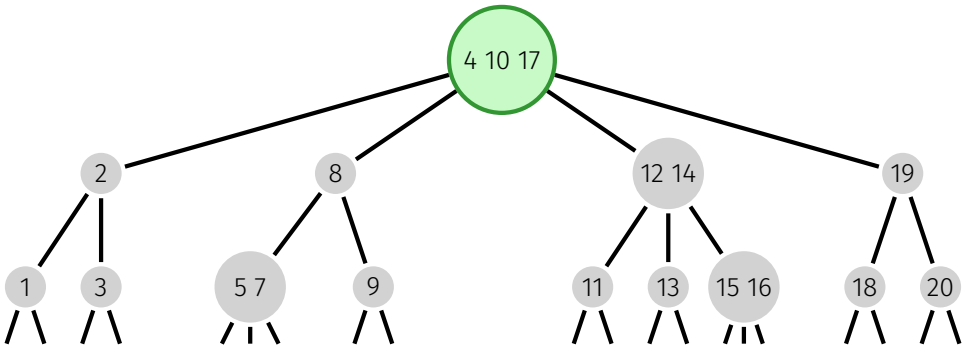
- Einfügen in 3-Knoten $\Rightarrow$ Aufteilen in zwei 2-Knoten



- Einfügen in 4-Knoten $\Rightarrow$ Aufteilen in 2-Knoten und 3-Knoten

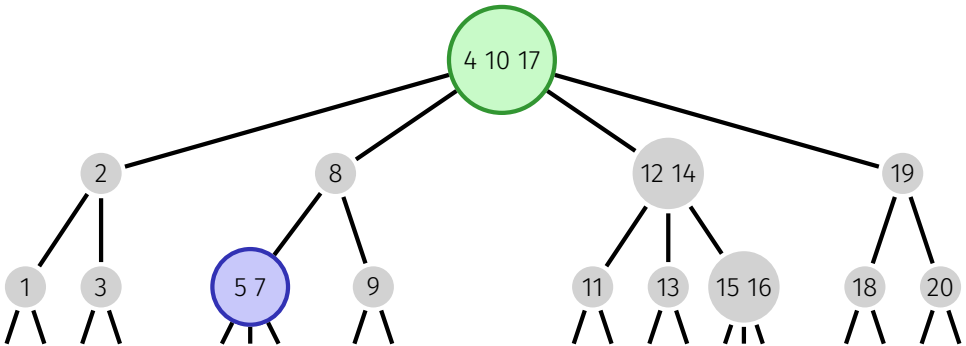


Beispiel: Schritt 4



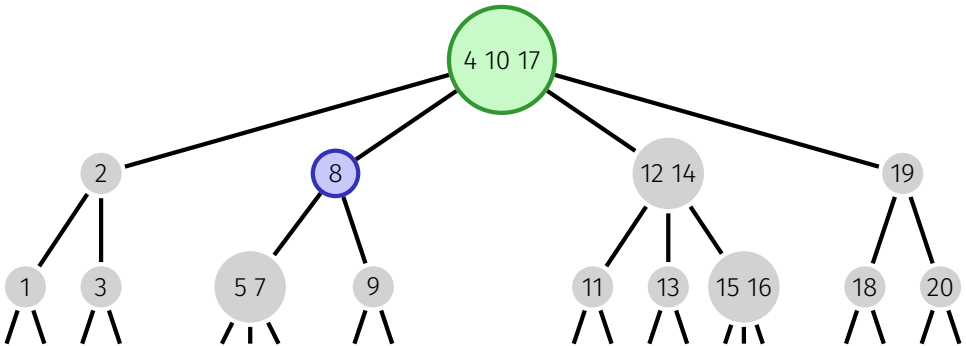
`remove`(6)

Beispiel: Schritt 4



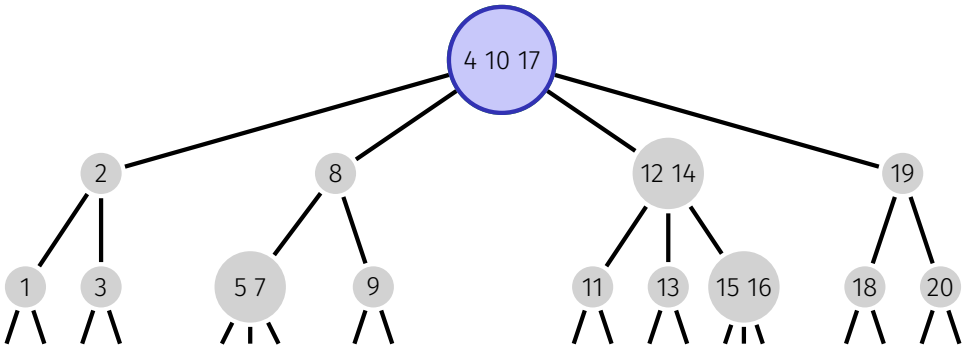
`remove`(6)

Beispiel: Schritt 4



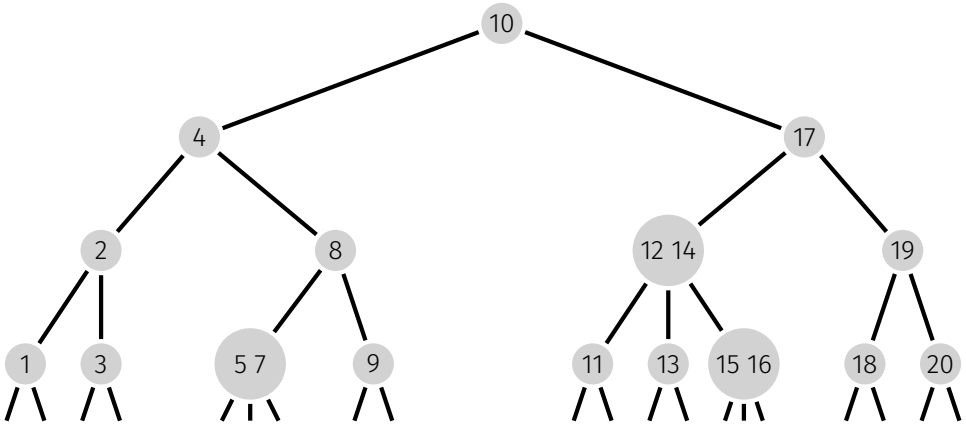
`remove`(6)

Beispiel: Schritt 4



`remove`(6)

Beispiel: Schritt 4



`remove`(6)

2-3-Bäume: Überblick

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

Probleme mit Implementierung:

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

Probleme mit Implementierung:

- drei Typen von Knoten

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

Probleme mit Implementierung:

- drei Typen von Knoten
- viele Fallunterscheidungen

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

Probleme mit Implementierung:

- drei Typen von Knoten
- viele Fallunterscheidungen
- Speichern von Datensätzen bei Knoten schwierig

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

Probleme mit Implementierung:

- drei Typen von Knoten
- viele Fallunterscheidungen
- Speichern von Datensätzen bei Knoten schwierig

$\Rightarrow$ Rot-Schwarz-Bäume als bessere Implementierung von 2-3-Bäumen

2-3-Bäume: Überblick

Operationen:

- Suchen: $\mathcal{O}(\log n)$
- Einfügen: $\mathcal{O}(\log n)$
- Löschen: $\mathcal{O}(\log n)$

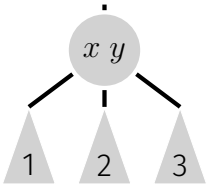
Probleme mit Implementierung:

- drei Typen von Knoten
- viele Fallunterscheidungen
- Speichern von Datensätzen bei Knoten schwierig

$\Rightarrow$ Rot-Schwarz-Bäume als bessere Implementierung von 2-3-Bäumen

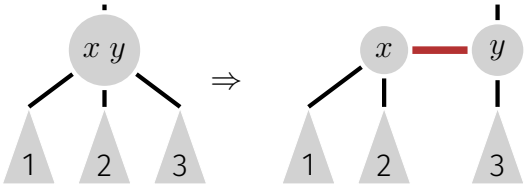
$\Rightarrow$ Ziel: 2-3-Baum als regulären binären Suchbaum darstellen

2-3-Bäume als Rot-Schwarz-Bäume



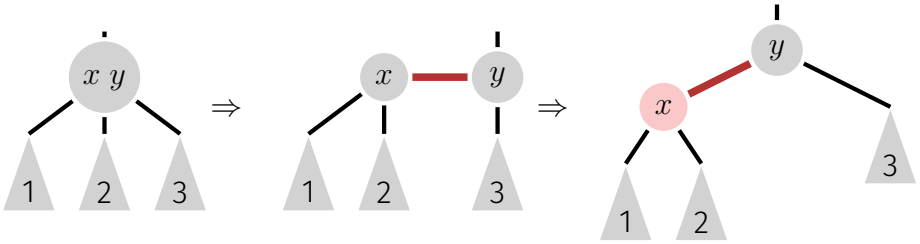
linksgerichteter (left-leaning) Rot-Schwarz-Baum:

2-3-Bäume als Rot-Schwarz-Bäume



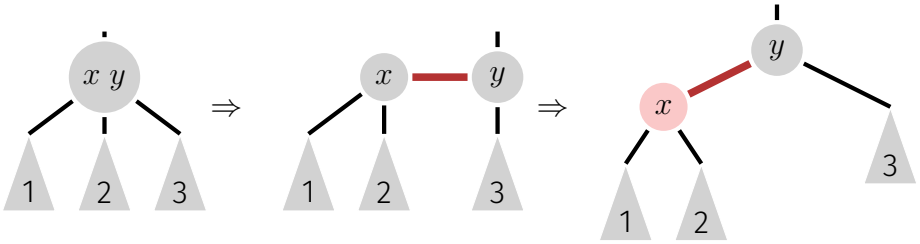
linksgerichteter (left-leaning) Rot-Schwarz-Baum:

2-3-Bäume als Rot-Schwarz-Bäume



linksgerichteter (left-leaning) Rot-Schwarz-Baum:

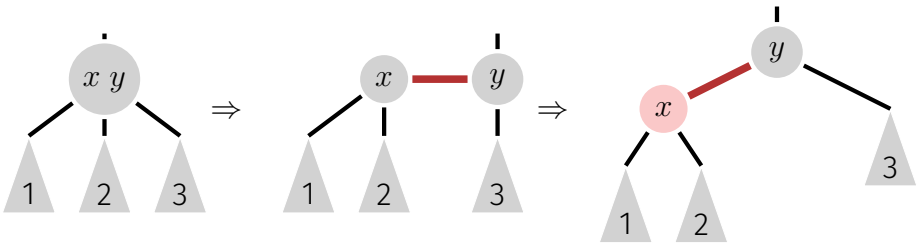
2-3-Bäume als Rot-Schwarz-Bäume



linksgerichteter (left-leaning) Rot-Schwarz-Baum:

- 3-Knoten ⇒ zwei 2-Knoten, kleinerer Schlüssel als linkes Kind

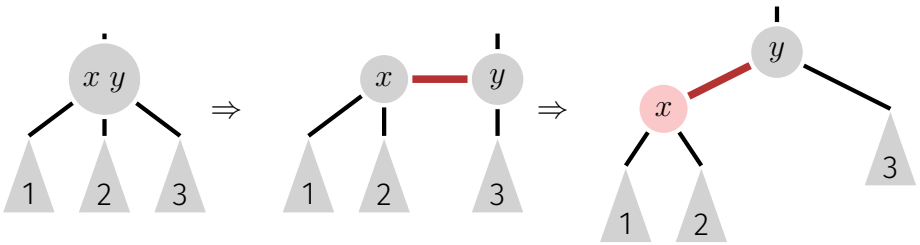
2-3-Bäume als Rot-Schwarz-Bäume



linksgerichteter (left-leaning) Rot-Schwarz-Baum:

- 3-Knoten $\Rightarrow$ zwei 2-Knoten, kleinerer Schlüssel als linkes Kind
- Kante zwischen zwei Knoten (sowie unterer Knoten) rot markiert

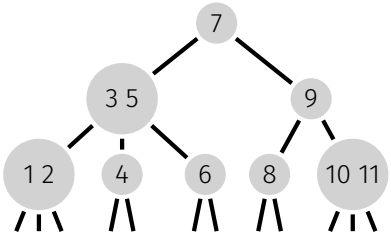
2-3-Bäume als Rot-Schwarz-Bäume



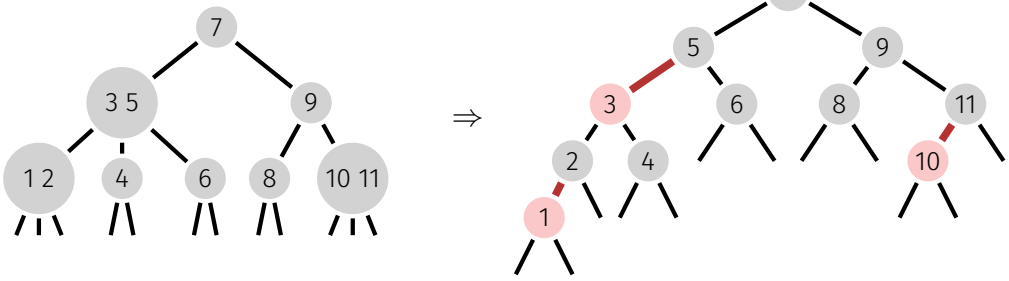
linksgerichteter (left-leaning) Rot-Schwarz-Baum:

- 3-Knoten ⇒ zwei 2-Knoten, kleinerer Schlüssel als linkes Kind
- Kante zwischen zwei Knoten (sowie unterer Knoten) rot markiert
  ⇒ rote Kante ist der Kleber, der die zwei Knoten zusammenhält

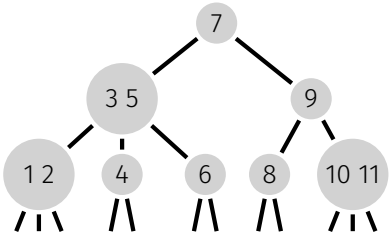
2-3-Bäume als Rot-Schwarz-Bäume

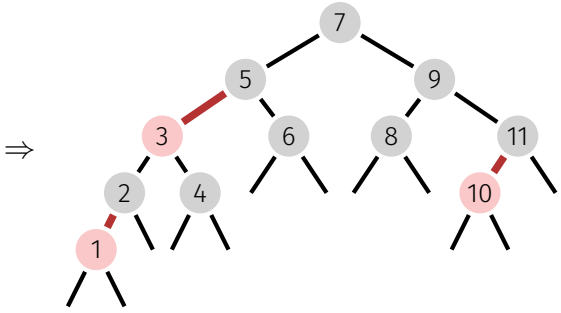


2-3-Bäume als Rot-Schwarz-Bäume



2-3-Bäume als Rot-Schwarz-Bäume

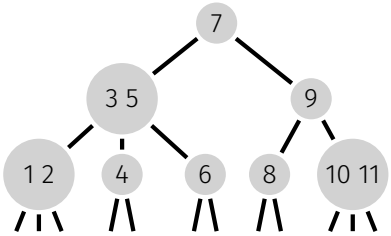


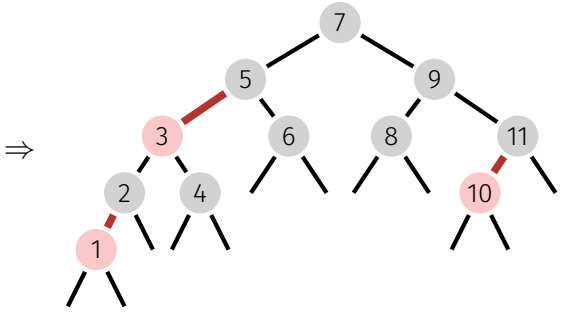


Eigenschaften:

- binärer Suchbaum

2-3-Bäume als Rot-Schwarz-Bäume

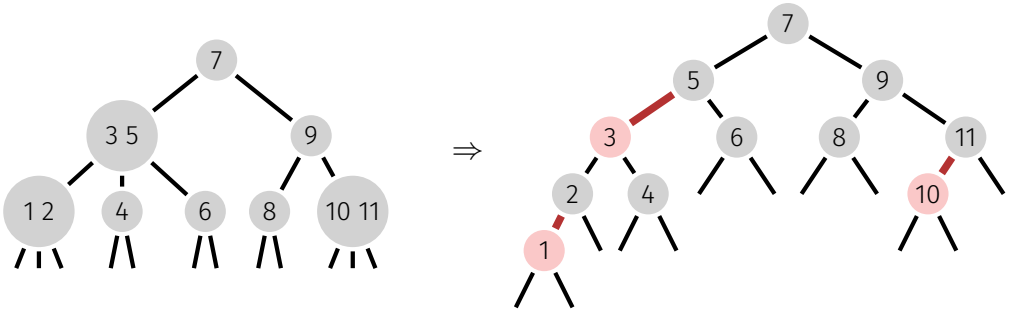




Eigenschaften:

- binärer Suchbaum
- rote Kanten gehen von Knoten zu seinem linken Kind (left-leaning)

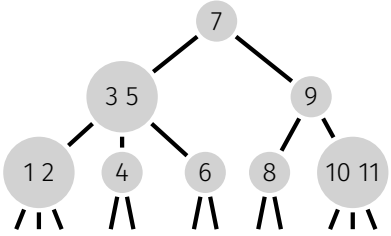
2-3-Bäume als Rot-Schwarz-Bäume



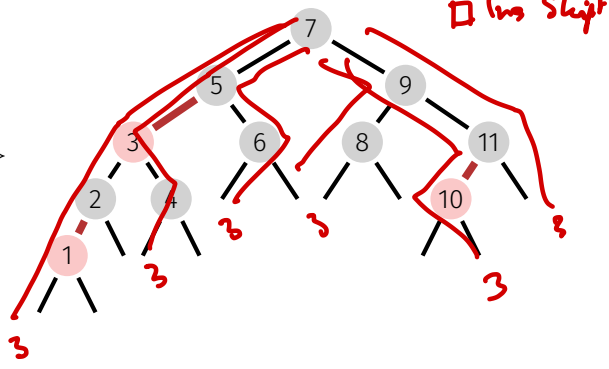
Eigenschaften:

- binärer Suchbaum
- rote Kanten gehen von Knoten zu seinem linken Kind (left-leaning)
- kein Knoten mit zwei roten Kanten (keine 4-Knoten)

2-3-Bäume als Rot-Schwarz-Bäume



⇒



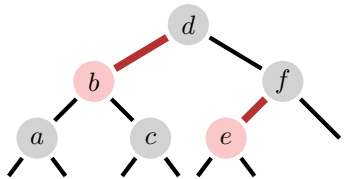
Eigenschaften:

- binärer Suchbaum
- rote Kanten gehen von Knoten zu seinem linken Kind (left-leaning)
- kein Knoten mit zwei roten Kanten (keine 4-Knoten)
- perfekt schwarz balanciert:
  jeder Pfad von Wurzel zu Blatt hat die gleiche Anzahl schwarze Kanten

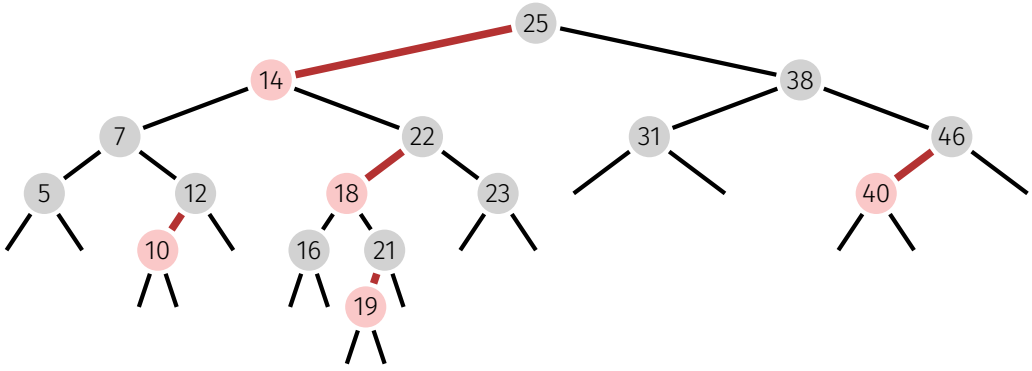
Rot-Schwarz-Bäume: Repräsentation

```
struct Node
{
  Key key;
  Value value;
  Node* left;
  Node* right;
  bool is_red;
};

// make leaves black
bool is_red(Node* node)
{
  return node && node->is_red;
}
```



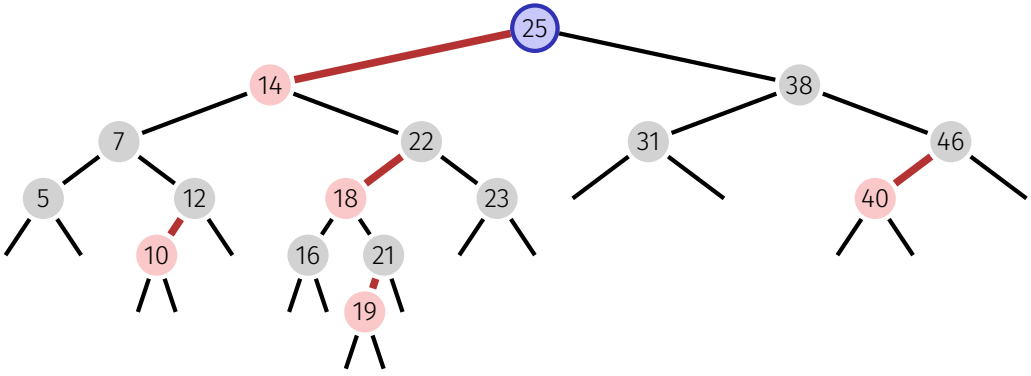
Suche



`search`(17)

normale Suche in binärem Suchbaum

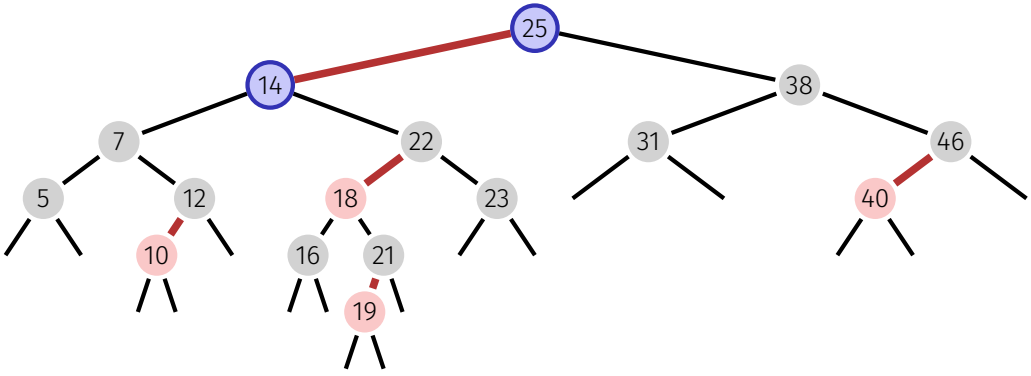
Suche



search(17)

normale Suche in binärem Suchbaum

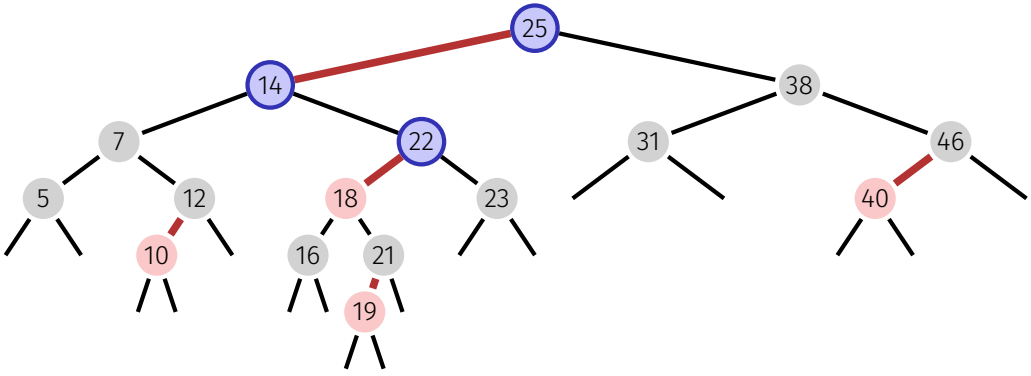
Suche



search(17)

normale Suche in binärem Suchbaum

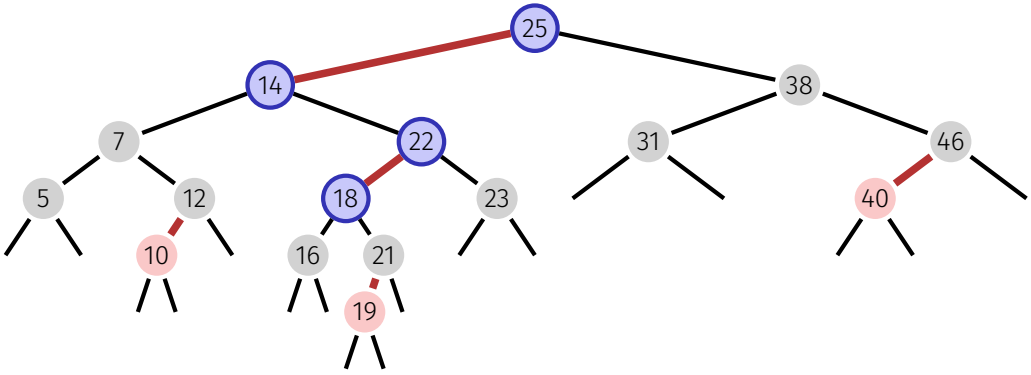
Suche



search(17)

normale Suche in binärem Suchbaum

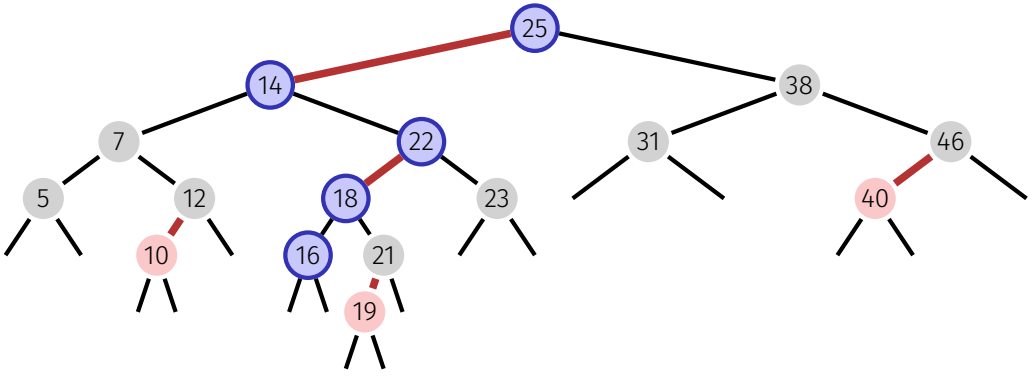
Suche



search(17)

normale Suche in binärem Suchbaum

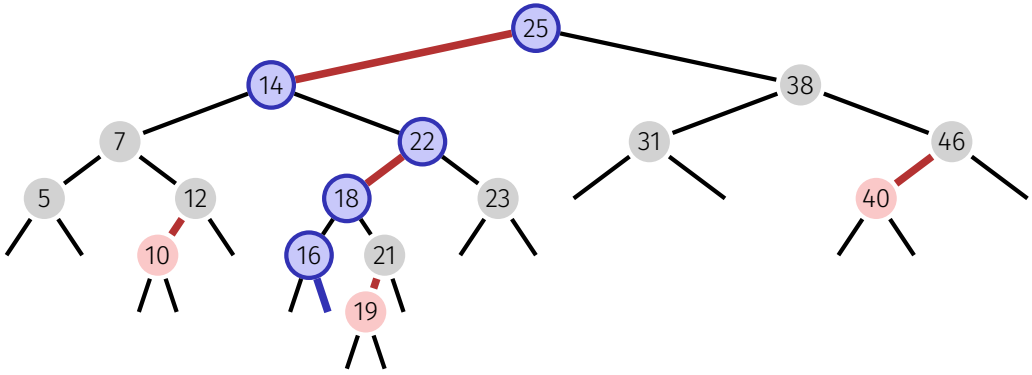
Suche



search(17)

normale Suche in binärem Suchbaum

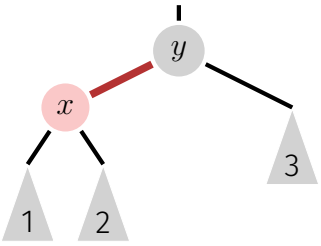
Suche



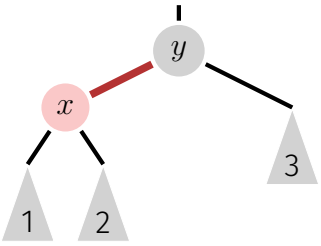
search(17)

- normale Suche in binärem Suchbaum

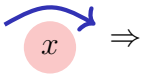
Rotationen `rotate_right` und `rotate_left`



Rotationen `rotate_right` und `rotate_left`

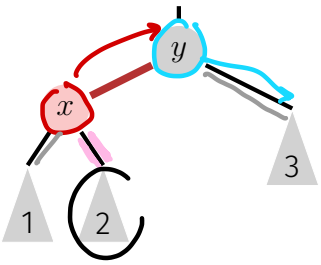


Rechts-Rotation über x

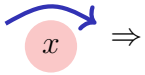


$\Rightarrow$

Rotationen `rotate_right` und `rotate_left`

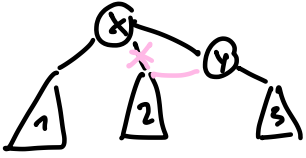


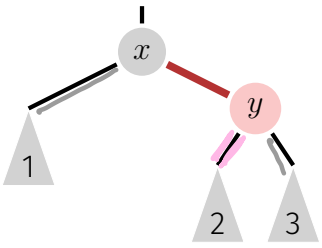
Rechts-Rotation über x



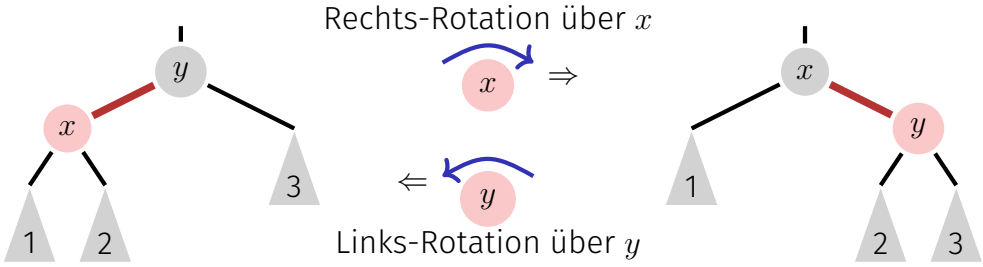
$\Rightarrow$

Auf Farben achten!

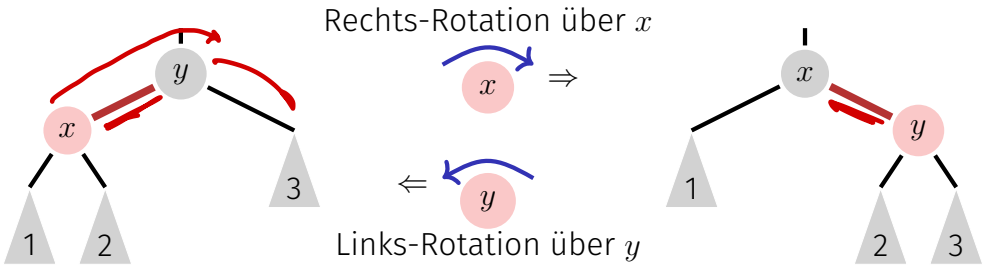




Rotationen `rotate_right` und `rotate_left`

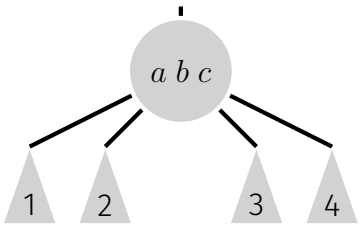


Rotationen `rotate_right` und `rotate_left`

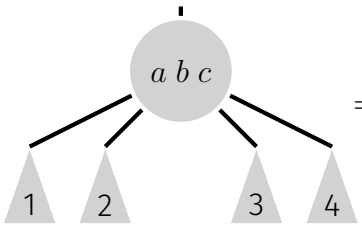


behalten Sortierung und perfekte schwarze Balancierung bei

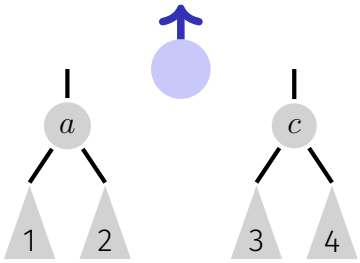
Farbwechsel (Color Flip) `push_up` und `push_down`



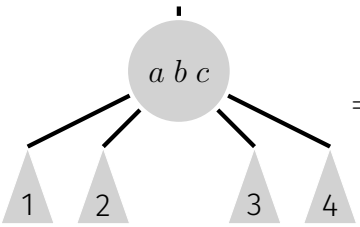
Farbwechsel (Color Flip) `push_up` und `push_down`



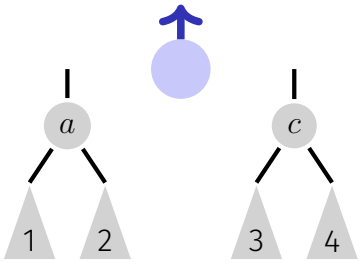
$\Rightarrow$ `split/propagate`

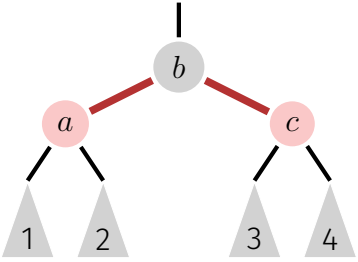


Farbwechsel (Color Flip) `push_up` und `push_down`

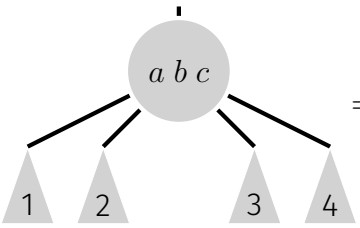


$\Rightarrow$ `split/propagate`

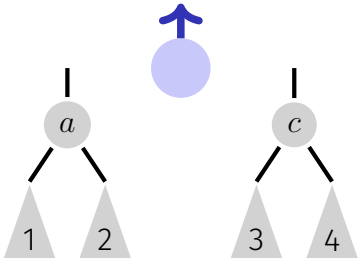


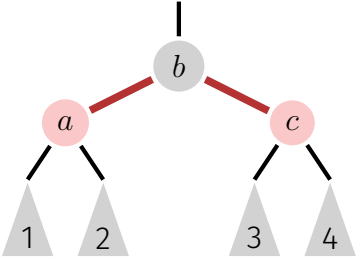


Farbwechsel (Color Flip) `push_up` und `push_down`

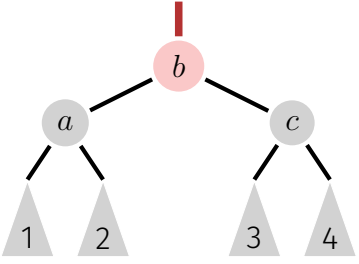


$\Rightarrow$ `split/propagate`

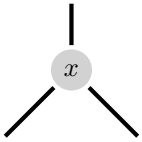




$\Rightarrow$
Farbwechsel

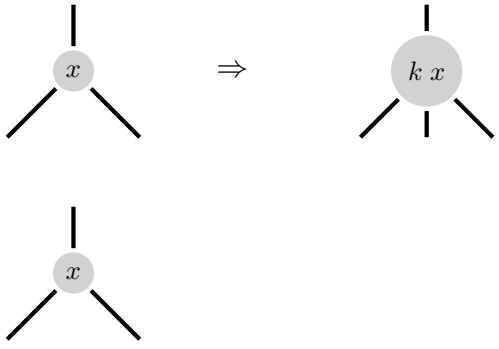


Einfügen eines Schlüssels in 2-Knoten ($k < x$)

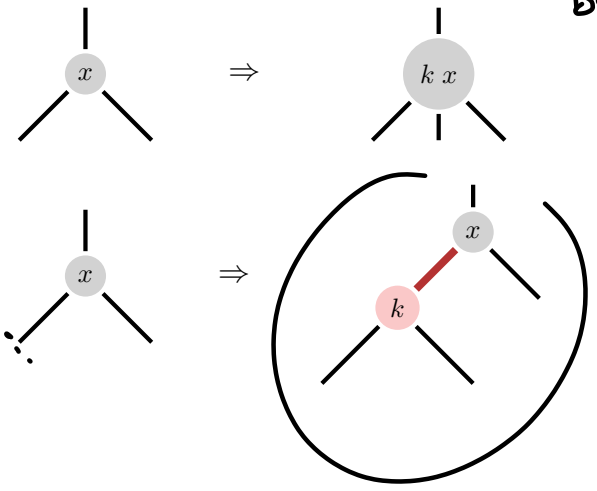


Einfügen eines Schlüssels in 2-Knoten ($k < x$)

Einfügen eines Schlüssels in 2-Knoten ($k < x$)

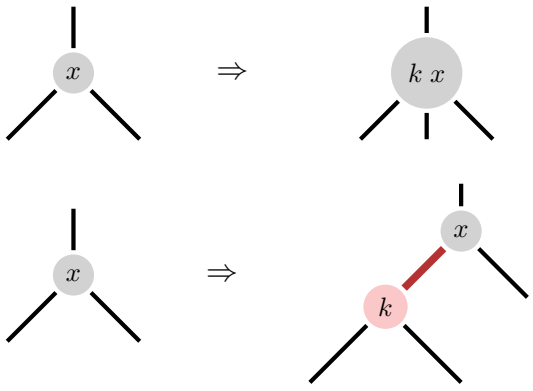


Einfügen eines Schlüssels in 2-Knoten ($k < x$)



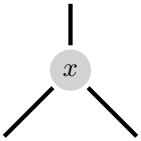
Beim Einfügen:
ROTEN
KNOTEN
verwenden

Einfügen eines Schlüssels in 2-Knoten ($k < x$)



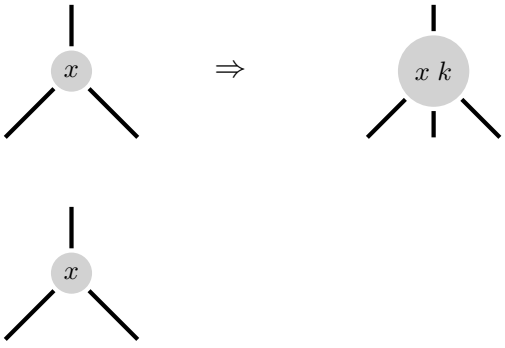
neuen (roten) Knoten mit roter Kante einfügen

Einfügen eines Schlüssels in 2-Knoten ($k > x$)

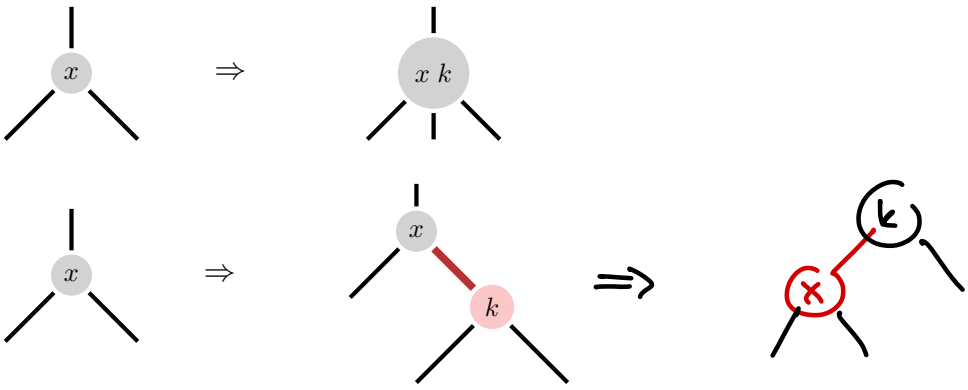


Einfügen eines Schlüssels in 2-Knoten ($k > x$)

Einfügen eines Schlüssels in 2-Knoten ($k > x$)

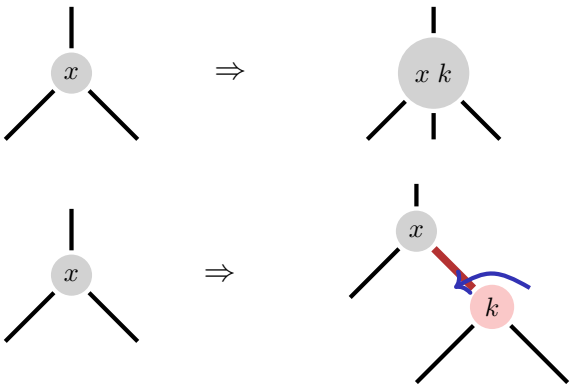


Einfügen eines Schlüssels in 2-Knoten ($k > x$)



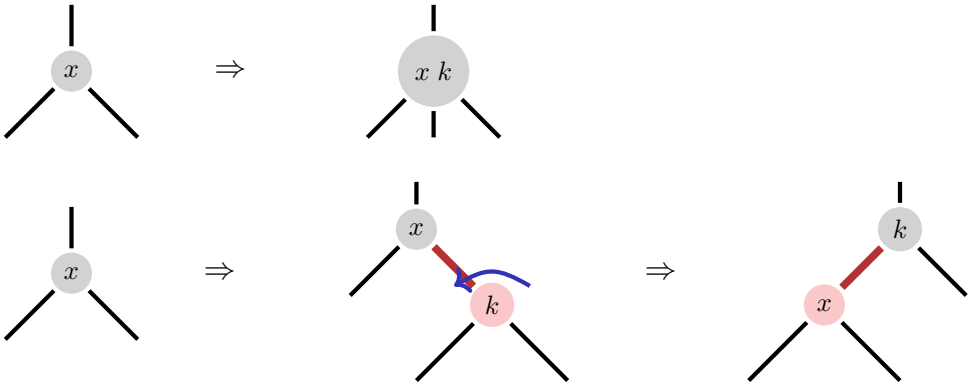
- neuen (roten) Knoten mit roter Kante einfügen

Einfügen eines Schlüssels in 2-Knoten ($k > x$)



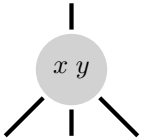
- neuen (roten) Knoten mit roter Kante einfügen

Einfügen eines Schlüssels in 2-Knoten ($k > x$)

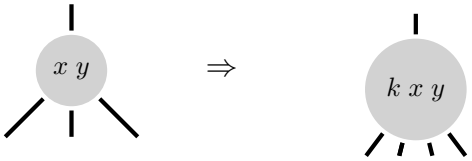


- neuen (roten) Knoten mit roter Kante einfügen
- Links-Rotation, damit left-leaning

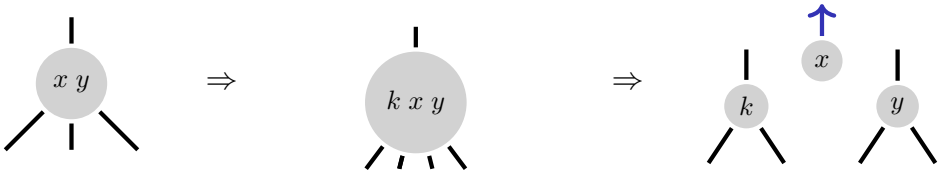
Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)



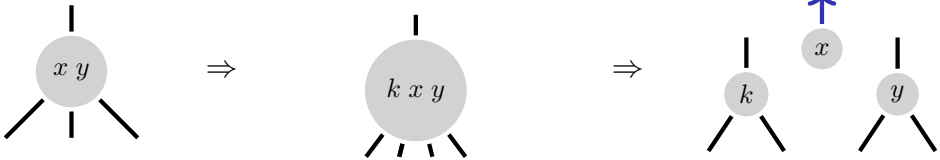
Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)

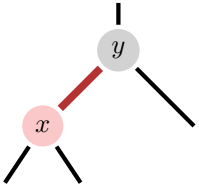


Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)

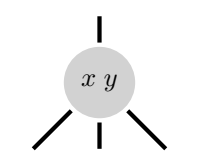


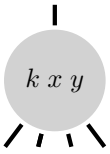
Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)



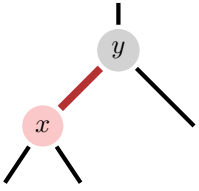


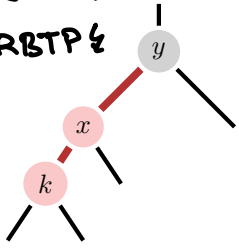
Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)

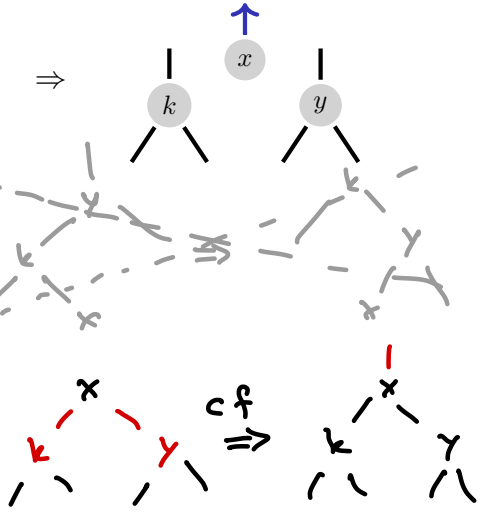




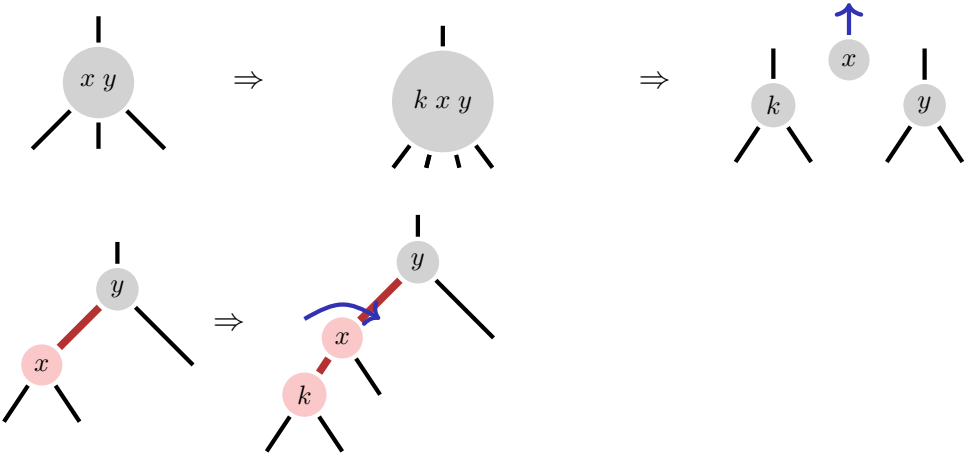
insert(k)
↯ RBTP ↯



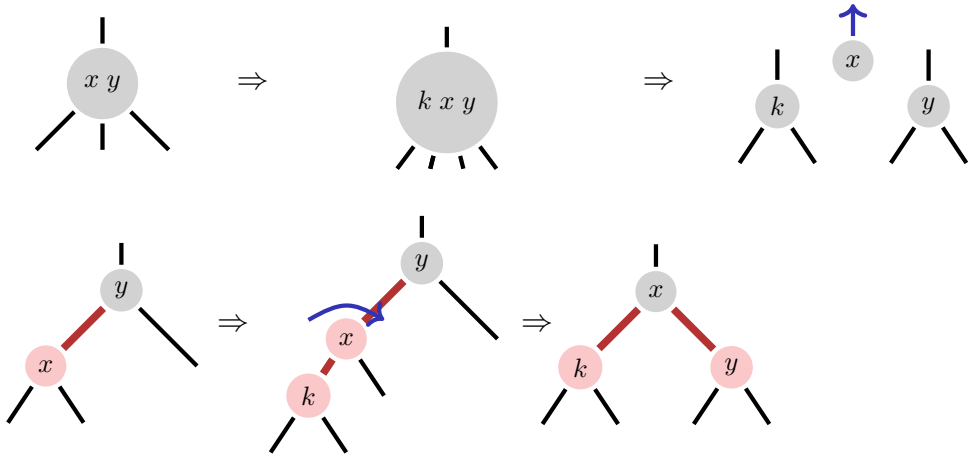




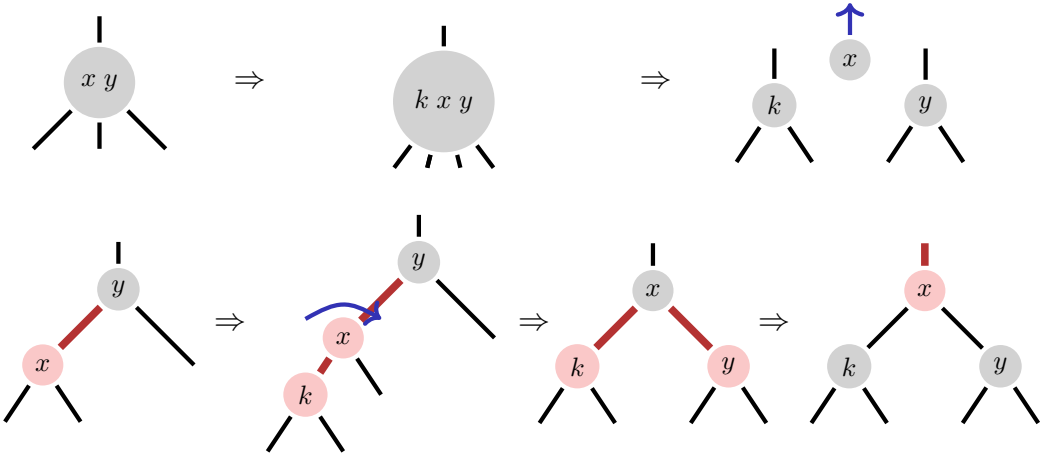
Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)



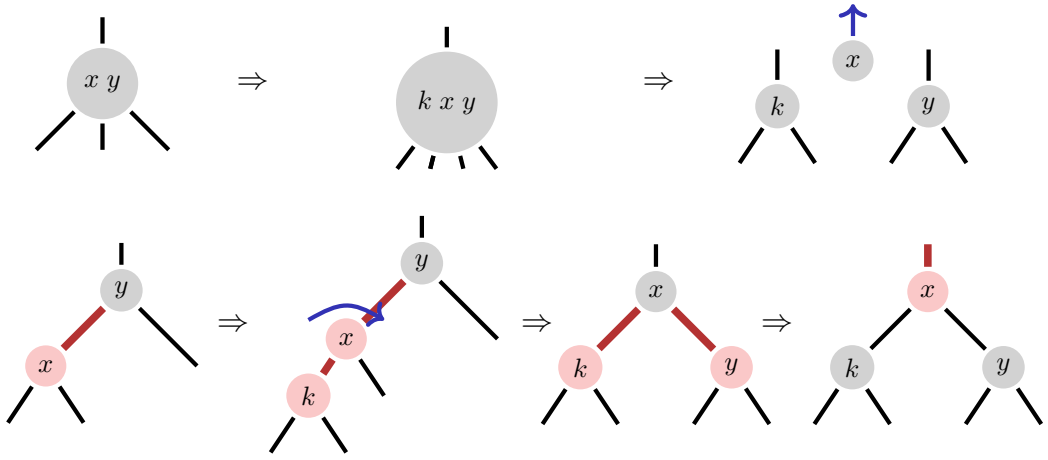
Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)



Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)

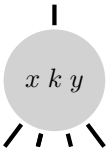


Einfügen eines Schlüssels in 3-Knoten ($k < x < y$)

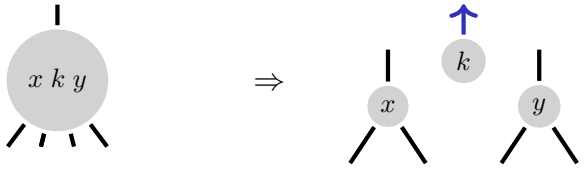


Rechts-Rotation, da zwei rote Kanten nacheinander, und Farbwechsel, da zwei benachbarte rote Kanten

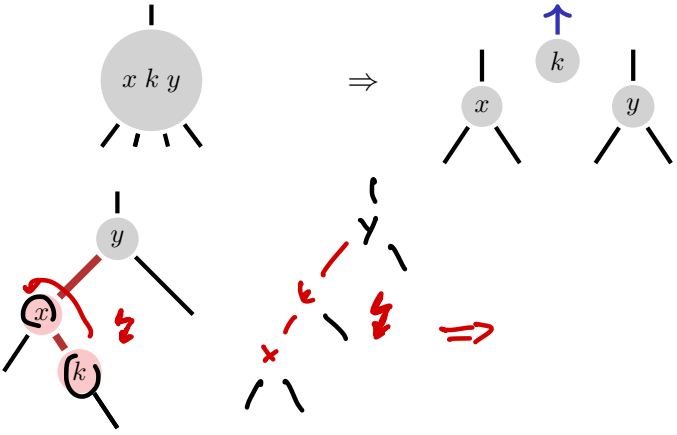
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



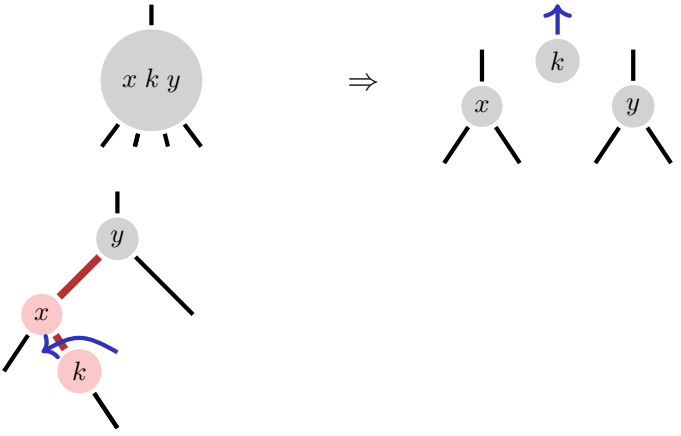
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



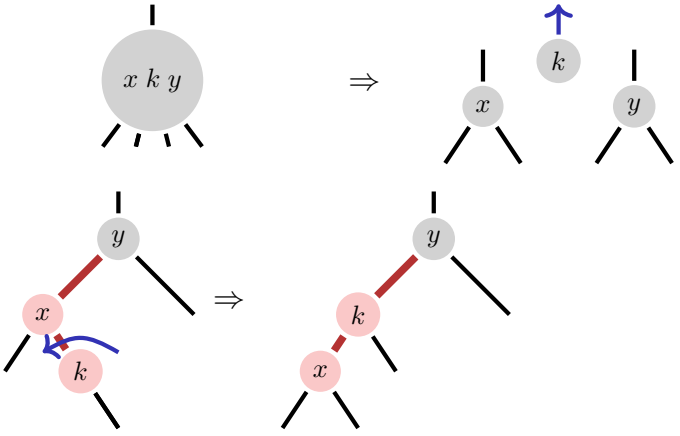
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



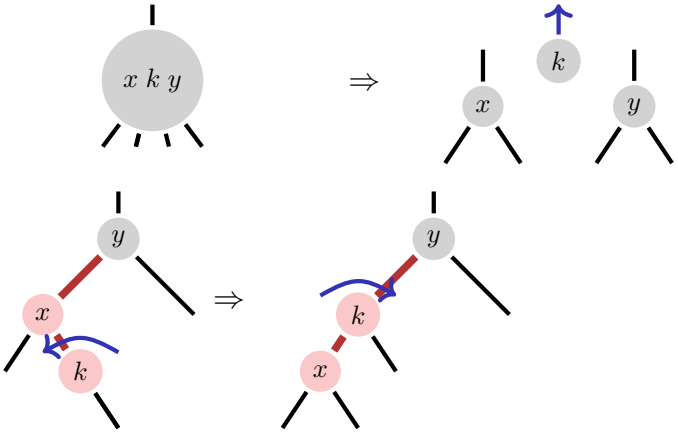
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



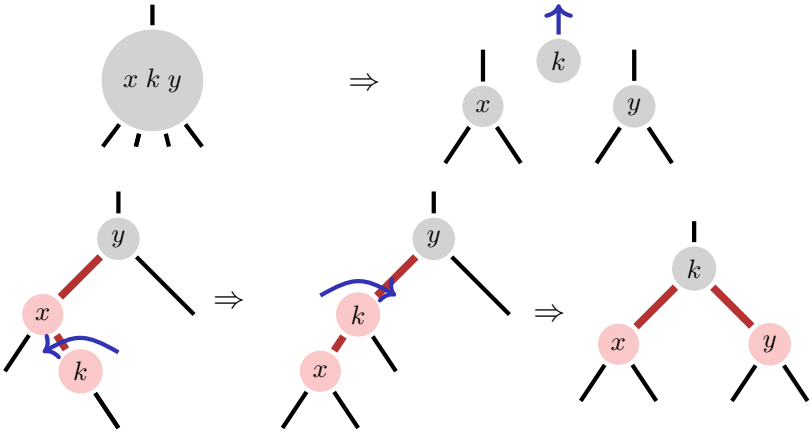
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



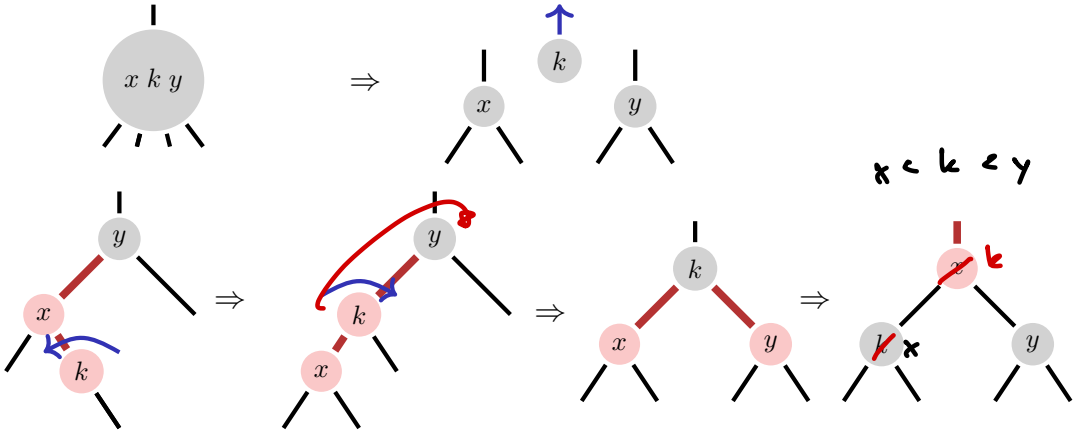
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



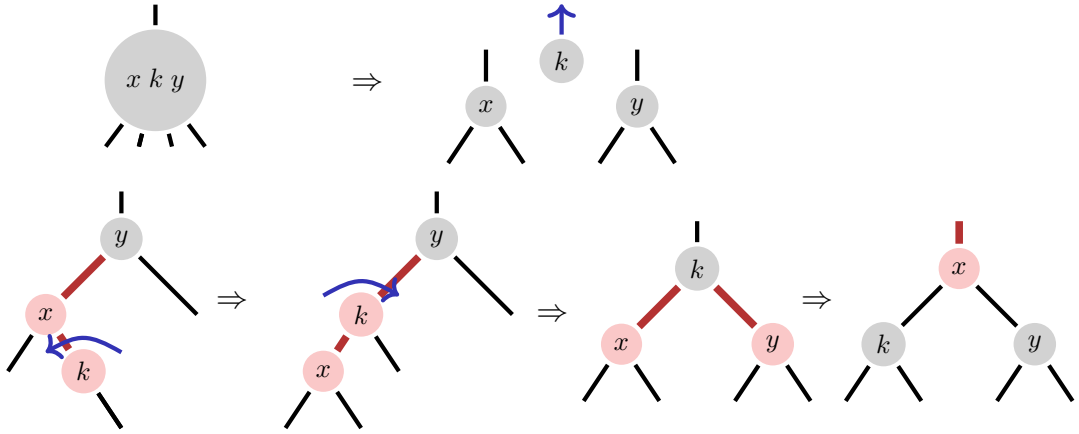
Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)



Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)

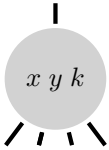


Einfügen eines Schlüssels in 3-Knoten ($x < k < y$)

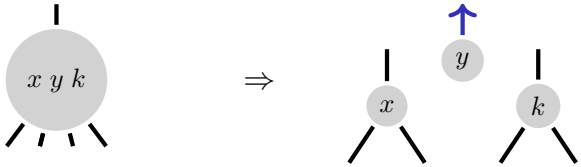


Links-Rotation, damit left-leaning, Rechts-Rotation, um zwei nacheinanderfolgende rote Kanten aufzulösen und Farbwechsel, um zwei benachbarte rote Kanten aufzulösen

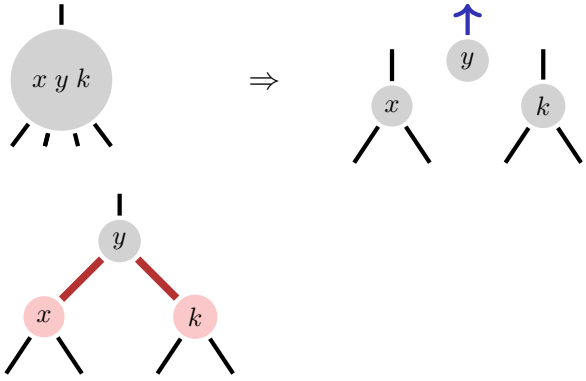
Einfügen eines Schlüssels in 3-Knoten ($k > y > x$)



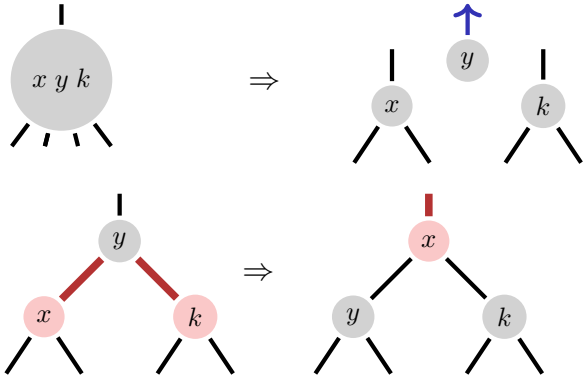
Einfügen eines Schlüssels in 3-Knoten ($k > y > x$)



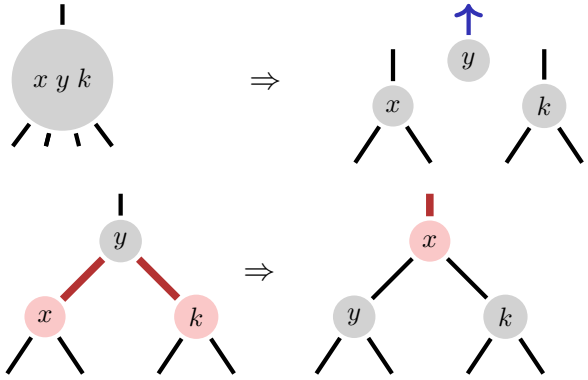
Einfügen eines Schlüssels in 3-Knoten ($k > y > x$)



Einfügen eines Schlüssels in 3-Knoten ($k > y > x$)

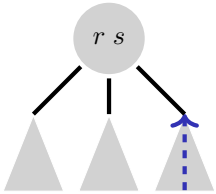


Einfügen eines Schlüssels in 3-Knoten ($k > y > x$)

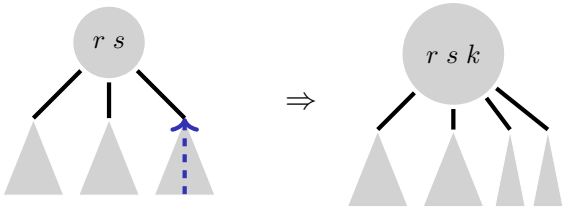


Farbwechsel, um zwei benachbarte rote Kanten aufzulösen

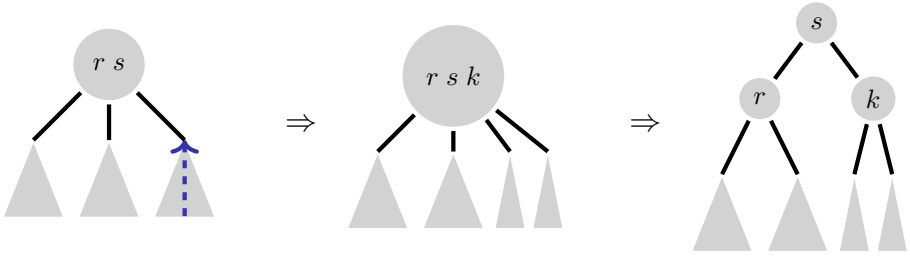
Einfügen eines Schlüssels in Wurzel



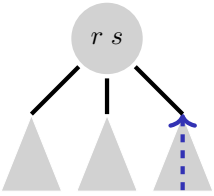
Einfügen eines Schlüssels in Wurzel

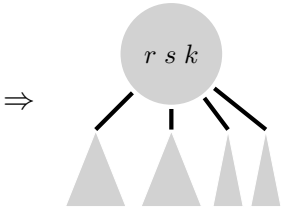


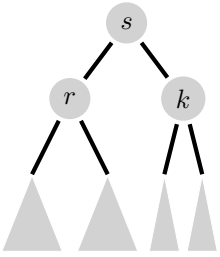
Einfügen eines Schlüssels in Wurzel

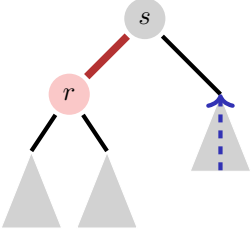


Einfügen eines Schlüssels in Wurzel

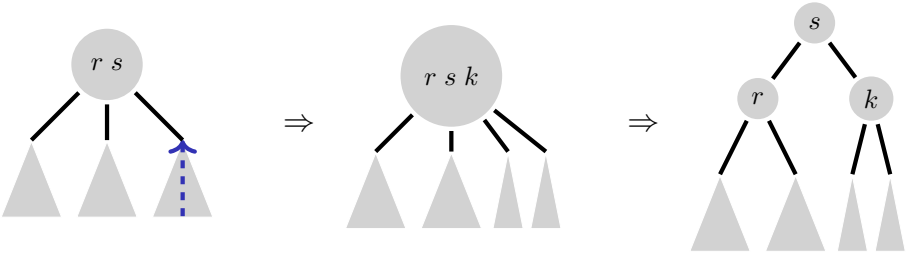


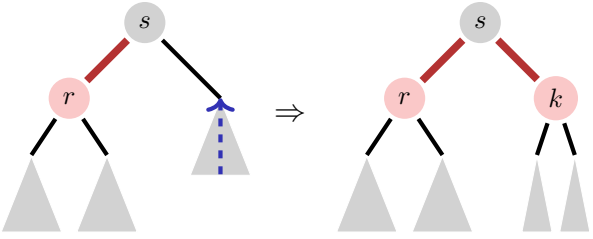




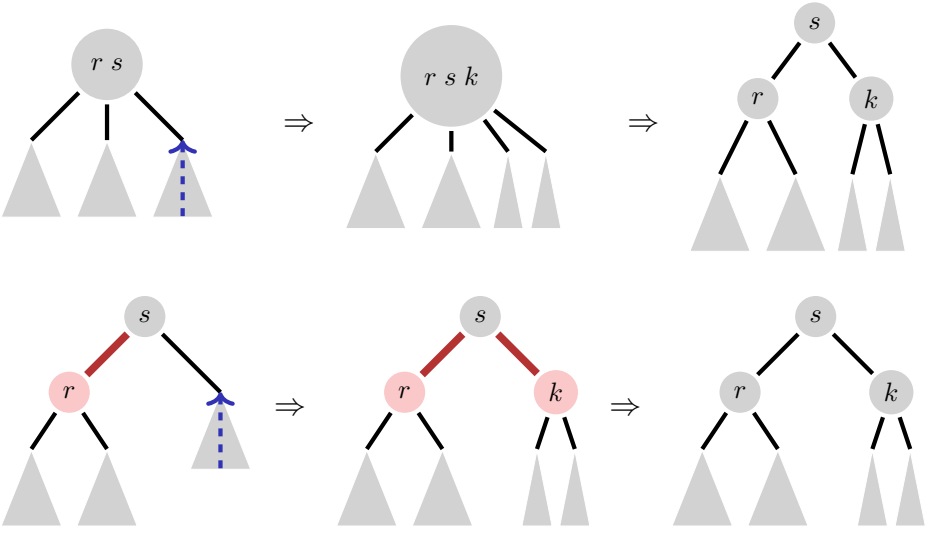


Einfügen eines Schlüssels in Wurzel

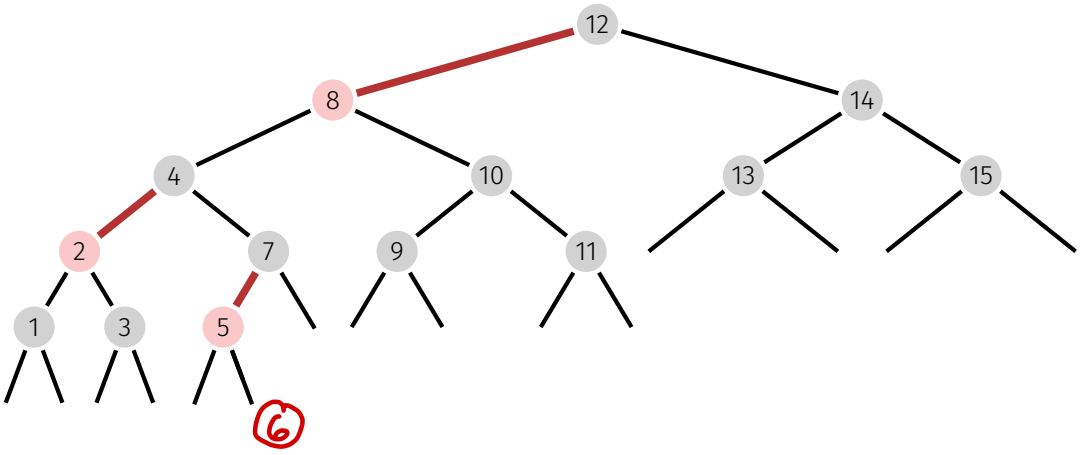




Einfügen eines Schlüssels in Wurzel

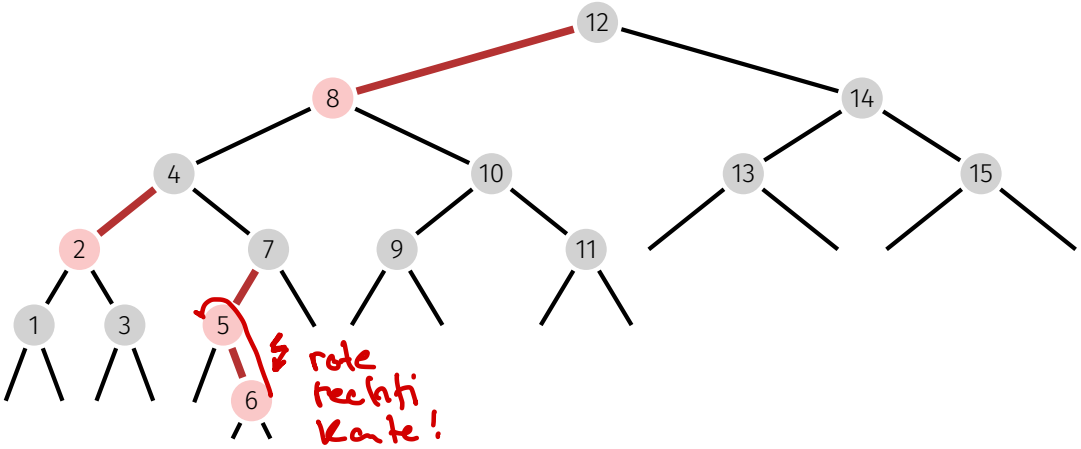


Beispiel: Schlüssel einfügen



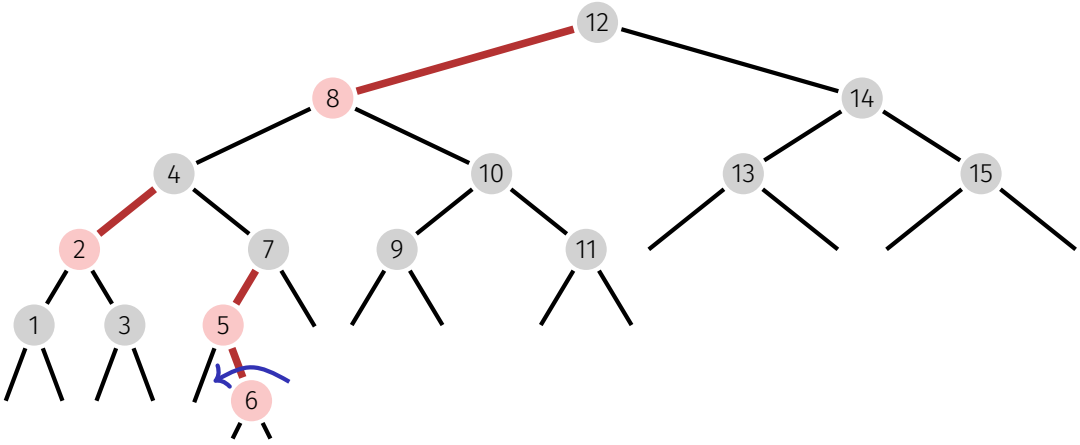
`insert`(6)

Beispiel: Schlüssel einfügen



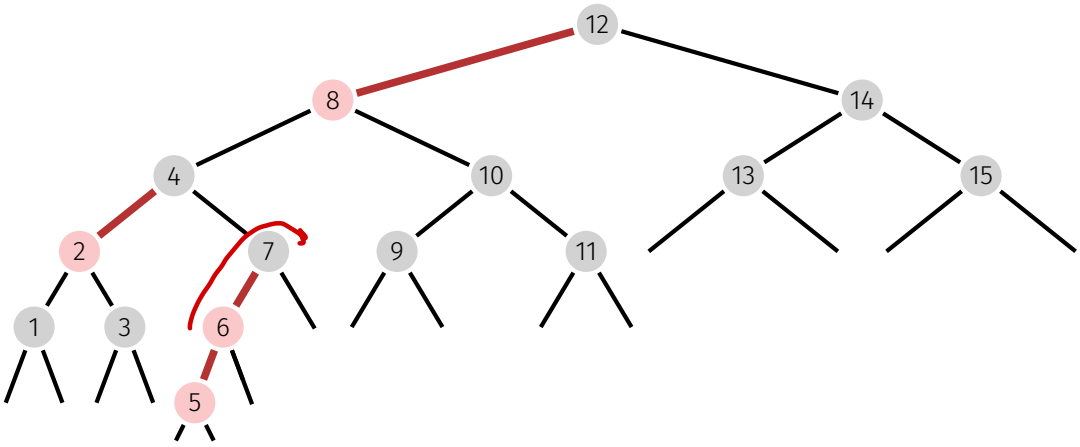
insert(6)

Beispiel: Schlüssel einfügen



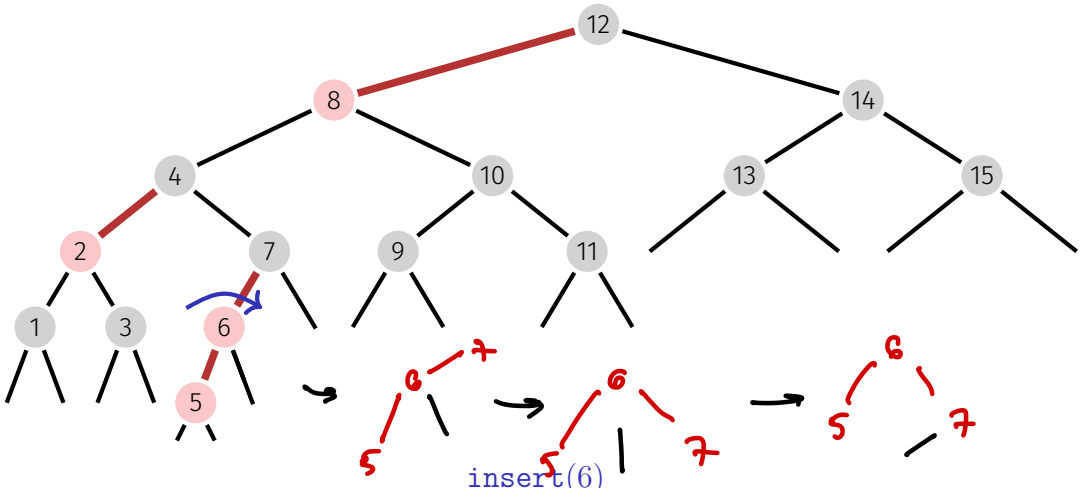
`insert`(6)

Beispiel: Schlüssel einfügen

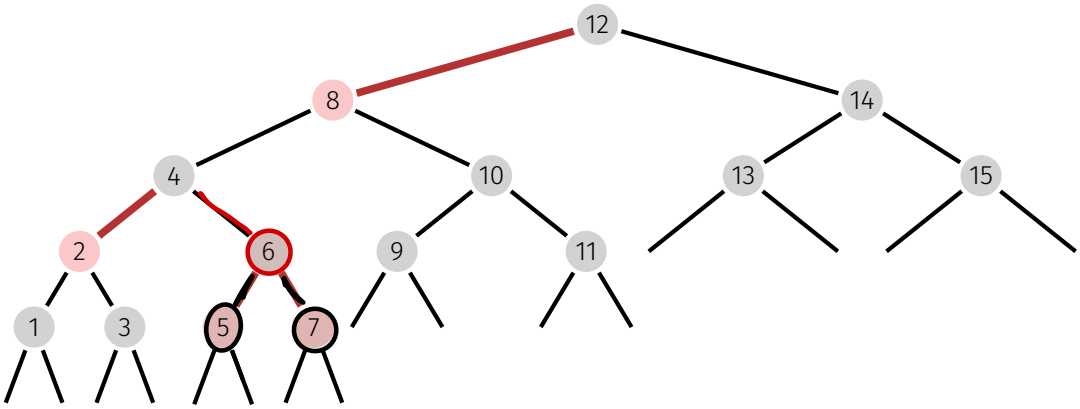


`insert`(6)

Beispiel: Schlüssel einfügen

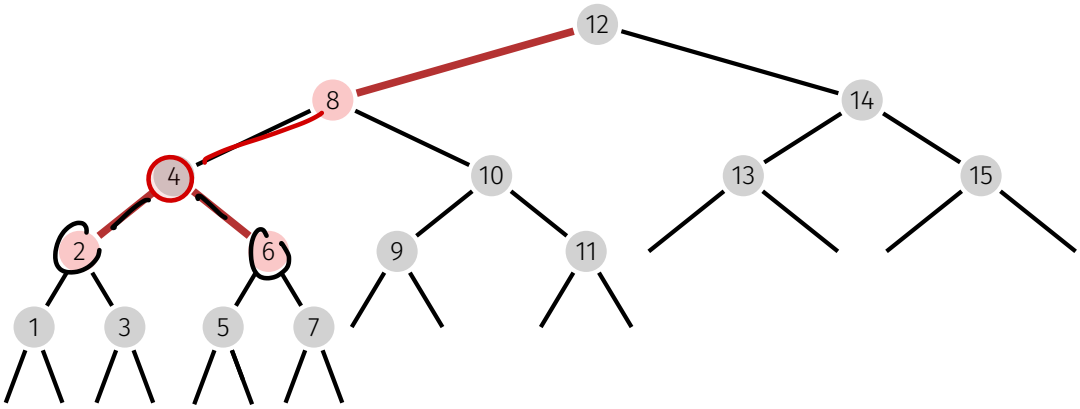


Beispiel: Schlüssel einfügen



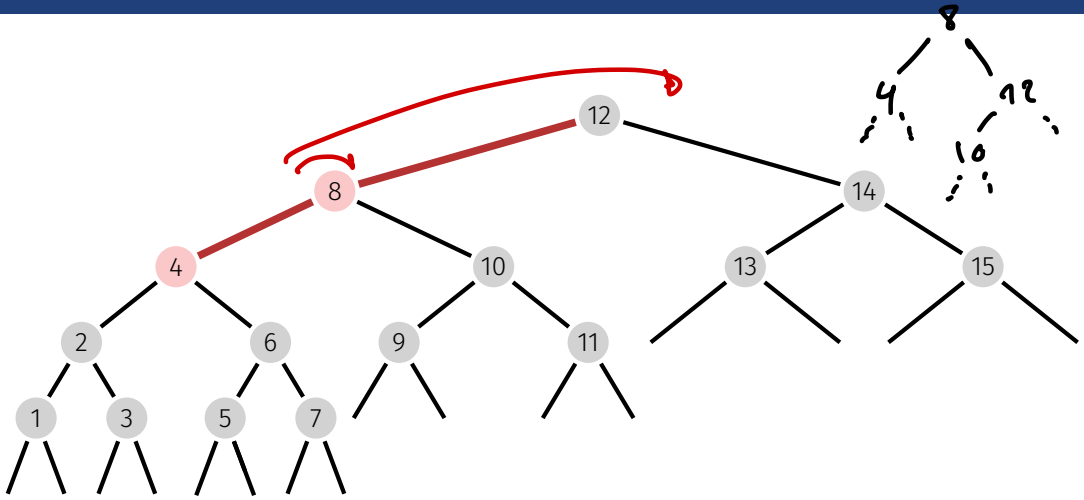
`insert`(6)

Beispiel: Schlüssel einfügen



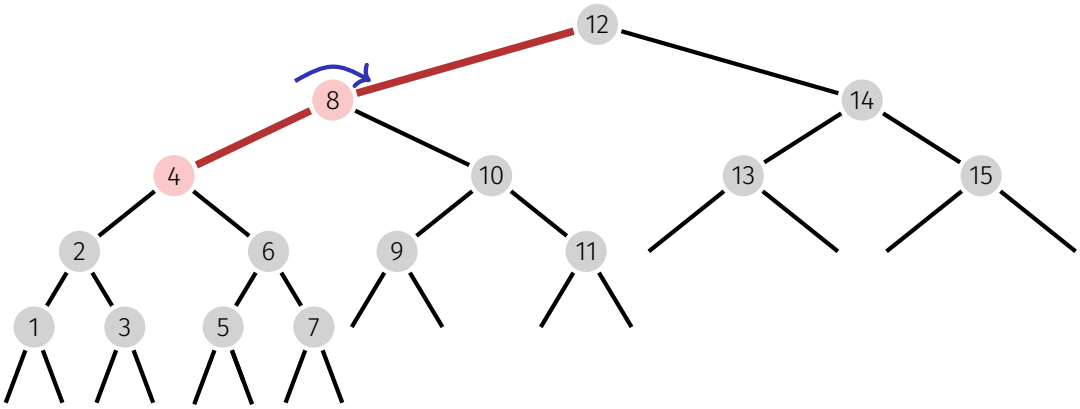
`insert`(6)

Beispiel: Schlüssel einfügen



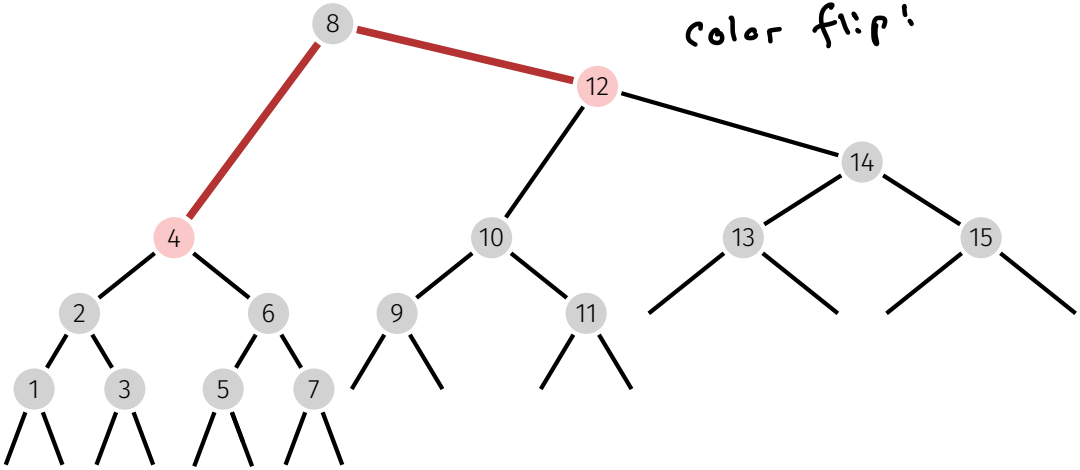
`insert`(6)

Beispiel: Schlüssel einfügen



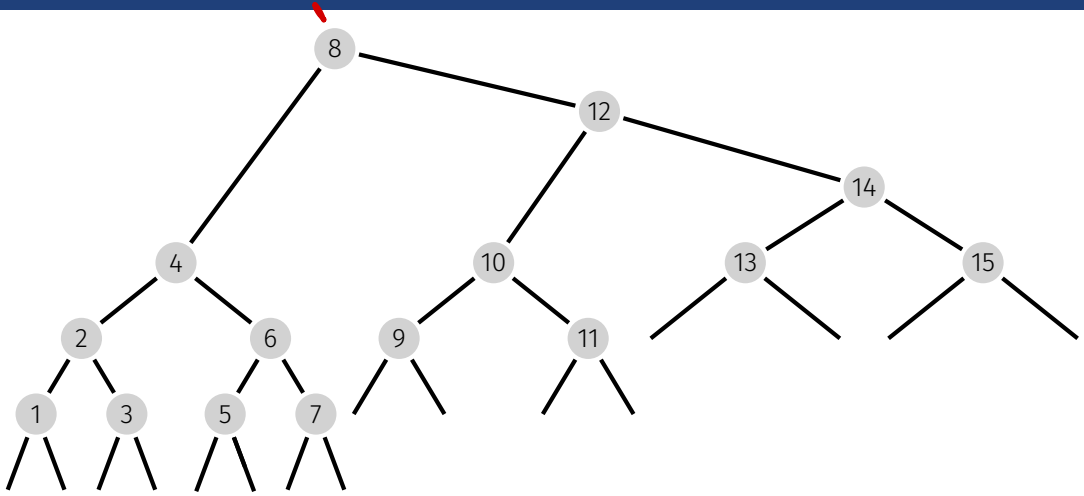
`insert`(6)

Beispiel: Schlüssel einfügen



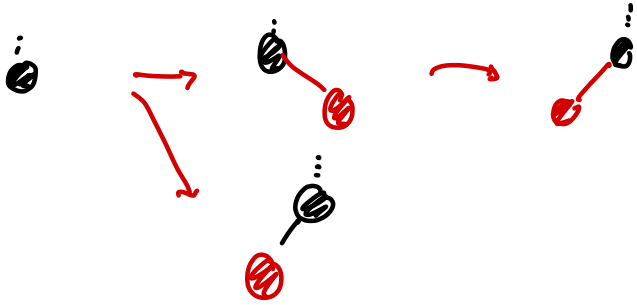
insert(6)

Beispiel: Schlüssel einfügen



`insert`(6)

Einfügen eines Schlüssels: Allgemeiner Fall



Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen

Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen
2. neuen (roten) Knoten k mit roter Kante zu p einfügen

Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen
2. neuen (roten) Knoten k mit roter Kante zu p einfügen
3. falls Kante $p \to k$ right-leaning: Rotation links bei p

Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen
2. neuen (roten) Knoten k mit roter Kante zu p einfügen
3. falls Kante $p \to k$ right-leaning: Rotation links bei p
4. falls zwei nacheinanderfolgende rote Kanten: Rotation rechts beim obersten der drei Knoten

Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen
2. neuen (roten) Knoten k mit roter Kante zu p einfügen
3. falls Kante $p \to k$ right-leaning: Rotation links bei p
4. falls zwei nacheinanderfolgende rote Kanten: Rotation rechts beim obersten der drei Knoten
5. falls zwei rote Kanten nebeneinander: Farbwechsel

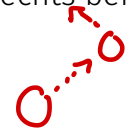
Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen
2. neuen (roten) Knoten k mit roter Kante zu p einfügen
3. falls Kante $p \to k$ right-leaning: Rotation links bei p
4. falls zwei nacheinanderfolgende rote Kanten: Rotation rechts beim obersten der drei Knoten
5. falls zwei rote Kanten nebeneinander: Farbwechsel
6. rekursives Behandeln der neuen roten Kante

Einfügen eines Schlüssels: Allgemeiner Fall

1. Elternknoten p von k suchen
2. neuen (roten) Knoten k mit roter Kante zu p einfügen
3. falls Kante $p \to k$ right-leaning: Rotation links bei p
4. falls zwei nacheinanderfolgende rote Kanten: Rotation rechts beim obersten der drei Knoten
5. falls zwei rote Kanten nebeneinander: Farbwechsel
6. rekursives Behandeln der neuen roten Kante

   Wichtig: neue rote Kante bedeutet, dass zwei Kinder der roten Kante schwarz sind



Schlüssel löschen: Überblick

Schlüssel löschen: Überblick

1. Schlüssel k mit symmetrischem Nachfolger p tauschen

Schlüssel löschen: Überblick

1. Schlüssel k mit symmetrischem Nachfolger p tauschen
2. rote Kante von oben nach unten bringen
   $\Rightarrow$ auf Pfad zu Wurzel hat es potentiell Knoten mit right-leaning Kanten

Schlüssel löschen: Überblick

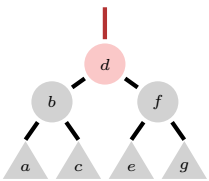
1. Schlüssel k mit symmetrischem Nachfolger p tauschen
2. rote Kante von oben nach unten bringen
   $\Rightarrow$ auf Pfad zu Wurzel hat es potentiell Knoten mit right-leaning Kanten
3. Knoten entfernen

Schlüssel löschen: Überblick

1. Schlüssel k mit symmetrischem Nachfolger p tauschen
2. rote Kante von oben nach unten bringen
   $\Rightarrow$ auf Pfad zu Wurzel hat es potentiell Knoten mit right-leaning Kanten
3. Knoten entfernen
4. right-leaning Kanten korrigieren auf dem Weg zurück nach oben

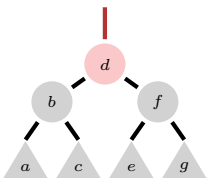
Rote Kante nach unten bewegen: Überblick

Invariante: Kante zu aktuellem Knoten ist rot



Rote Kante nach unten bewegen: Überblick

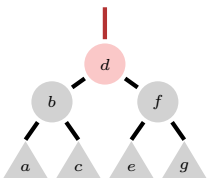
Invariante: Kante zu aktuellem Knoten ist rot



Annahme: Wir gehen zweimal nach rechts, die anderen Fälle sind analog.

Rote Kante nach unten bewegen: Überblick

Invariante: Kante zu aktuellem Knoten ist rot



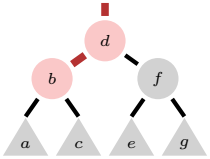
Annahme: Wir gehen zweimal nach rechts, die anderen Fälle sind analog.

zu zeigen: nach konstant vielen Operationen ist entweder rechtes Kind oder rechtes Grosskind rot

Rote Kante nach unten bewegen

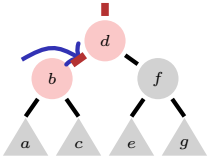
Rote Kante nach unten bewegen

- Geschwister b rot



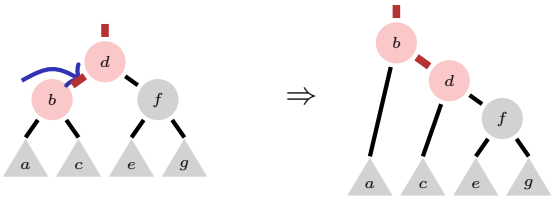
Rote Kante nach unten bewegen

- Geschwister b rot



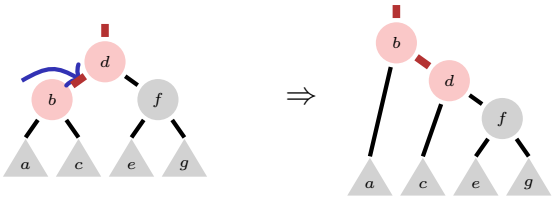
Rote Kante nach unten bewegen

- Geschwister b rot $\Rightarrow$ Rechts-Rotation über b

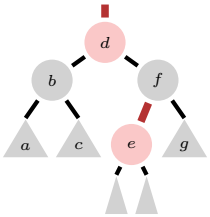


Rote Kante nach unten bewegen

- Geschwister b rot $\Rightarrow$ Rechts-Rotation über b

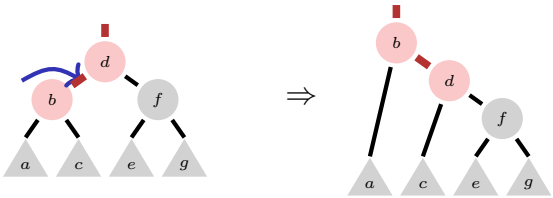


- Gross-Geschwister e rot

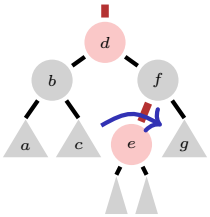


Rote Kante nach unten bewegen

- Geschwister b rot $\Rightarrow$ Rechts-Rotation über b

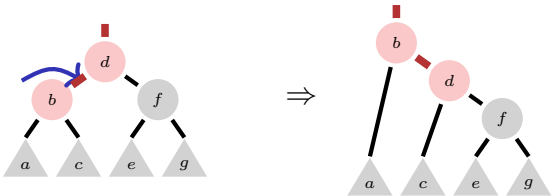


- Gross-Geschwister e rot

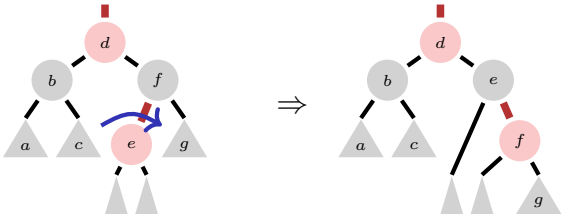


Rote Kante nach unten bewegen

- Geschwister b rot $\Rightarrow$ Rechts-Rotation über b

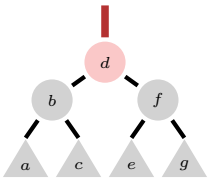


- Gross-Geschwister e rot $\Rightarrow$ Rechts-Rotation über e



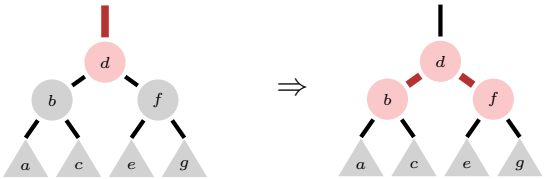
Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz



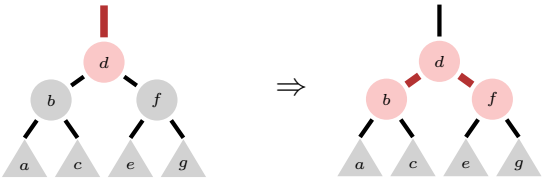
Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`

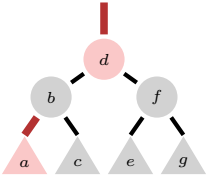


Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`

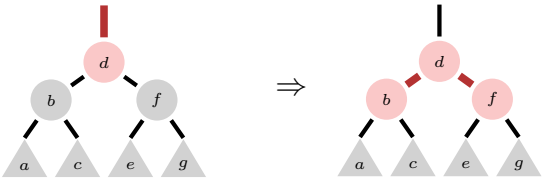


- Geschwister und Grossgeschwister schwarz, (linkes) Kind von b rot

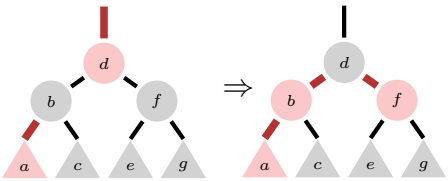


Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`

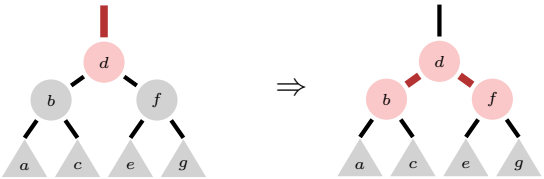


- Geschwister und Grossgeschwister schwarz, (linkes) Kind von b rot

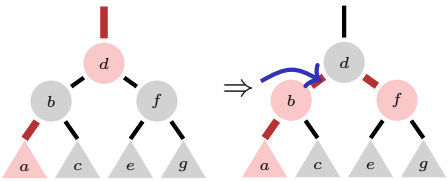


Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`

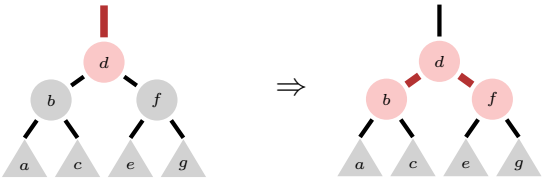


- Geschwister und Grossgeschwister schwarz, (linkes) Kind von b rot

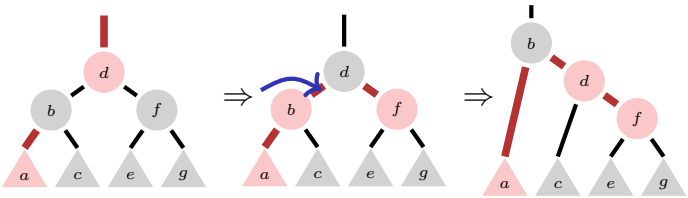


Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`

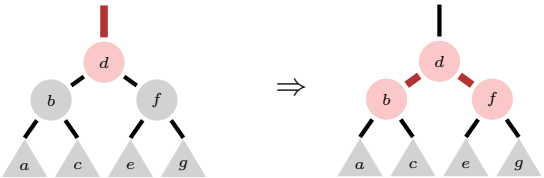


- Geschwister und Grossgeschwister schwarz, (linkes) Kind von b rot

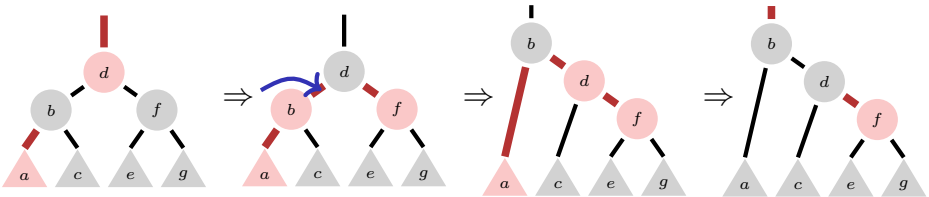


Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`

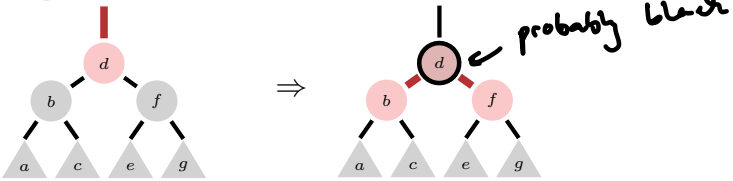


- Geschwister und Grossgeschwister schwarz, (linkes) Kind von b rot

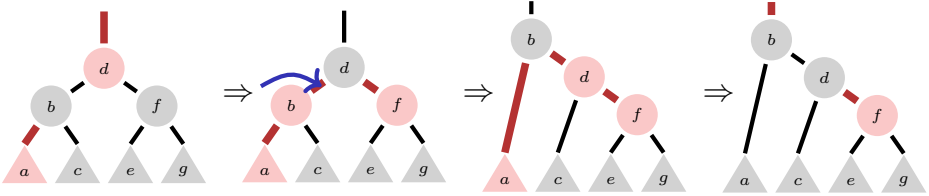


Rote Kante nach unten bewegen

- Geschwister und Grossgeschwister schwarz, Kinder von b schwarz
  $\Rightarrow$ `push_down`



- Geschwister und Grossgeschwister schwarz, (linkes) Kind von b rot
  $\Rightarrow$ `push_down`, `right_rotate`, `push_up`



4. Aufräumen von Right-Leaning Kanten

4. Aufräumen von Right-Leaning Kanten

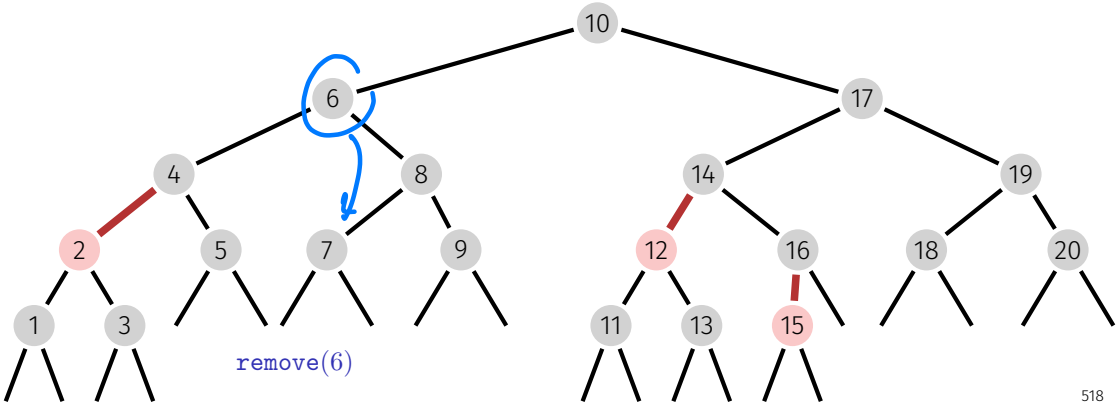
auf dem Weg zurück nach oben zur Wurzel: alle right-leaning Kanten aufräumen

4. Aufräumen von Right-Leaning Kanten

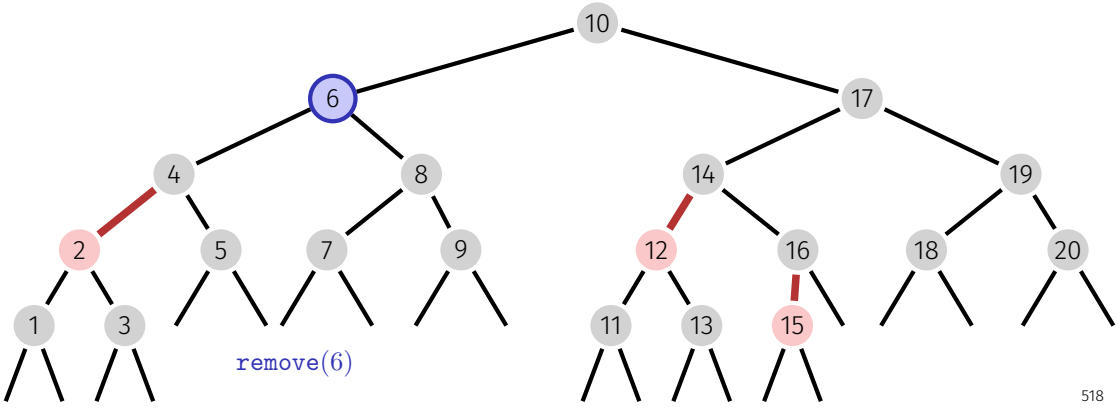
auf dem Weg zurück nach oben zur Wurzel: alle right-leaning Kanten aufräumen

$\Rightarrow$ gleiches Verfahren wie beim Einfügen

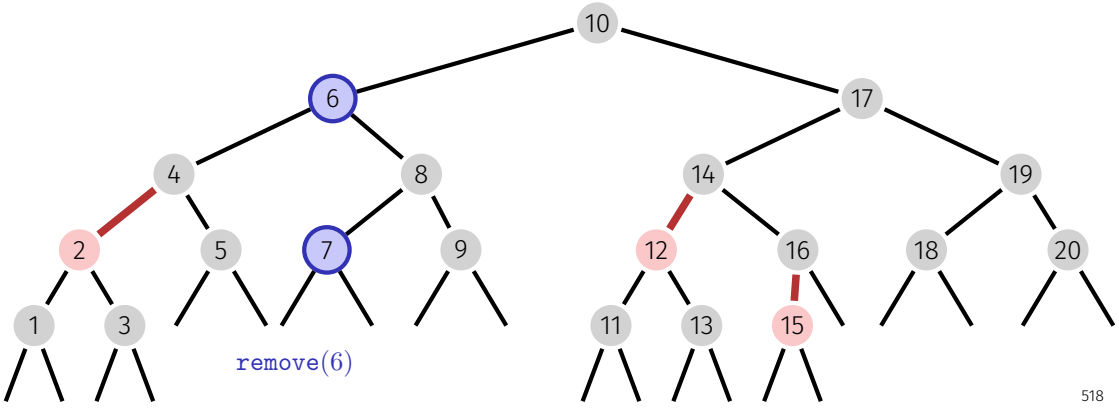
Beispiel: Schlüssel löschen



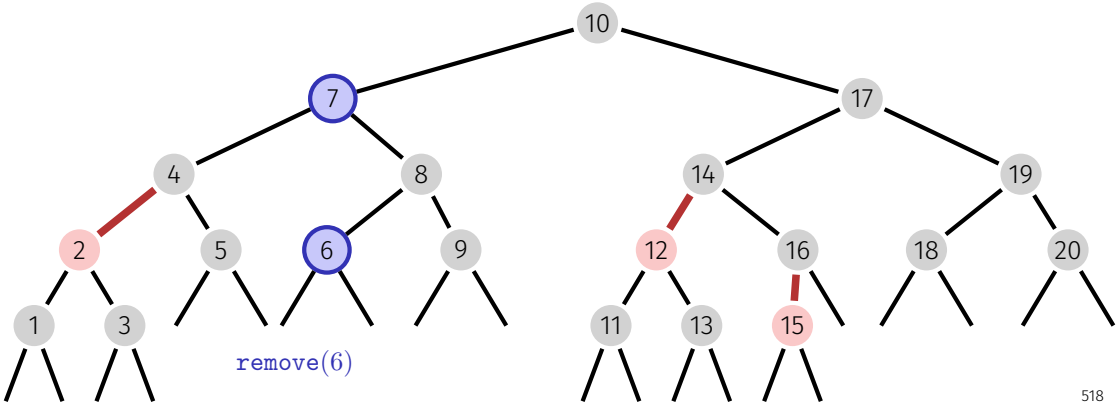
Beispiel: Schlüssel löschen



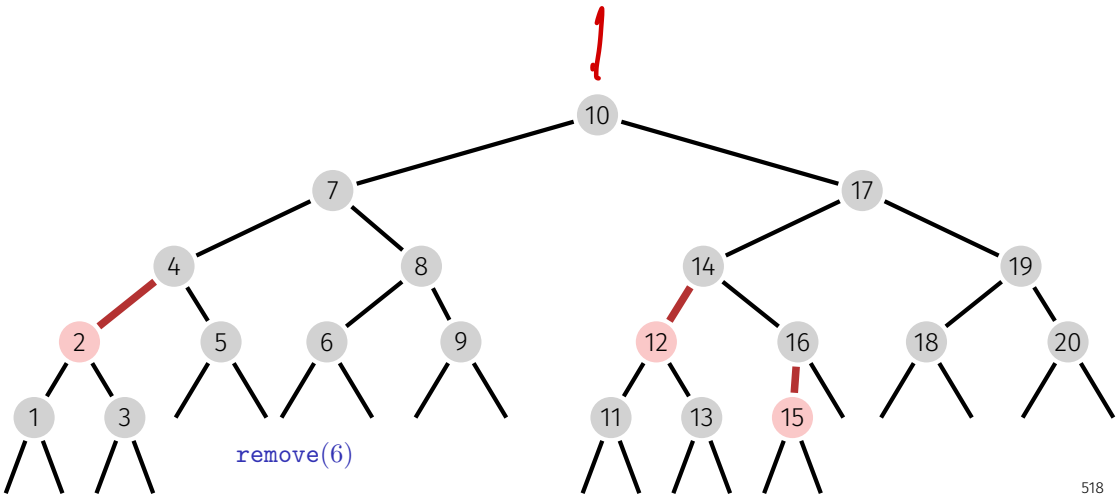
Beispiel: Schlüssel löschen



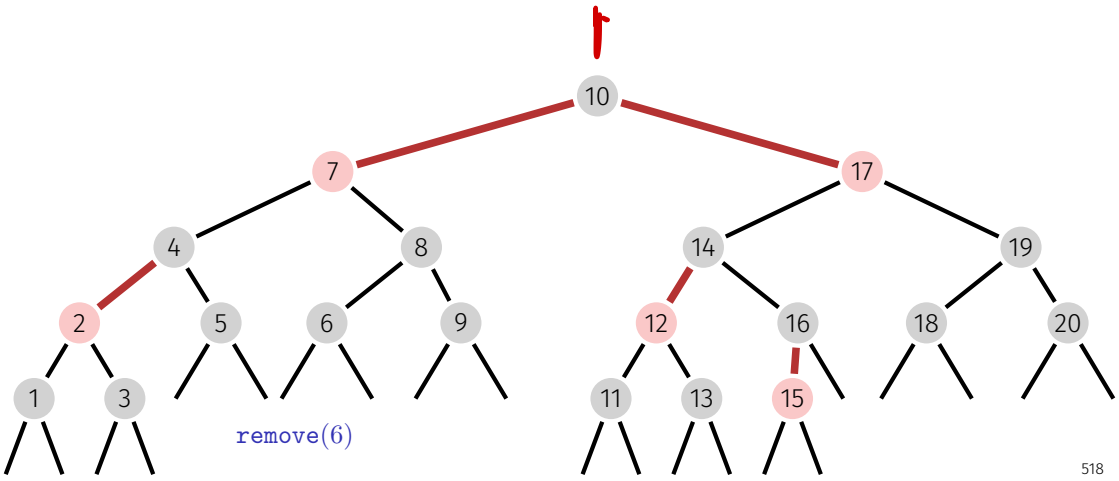
Beispiel: Schlüssel löschen



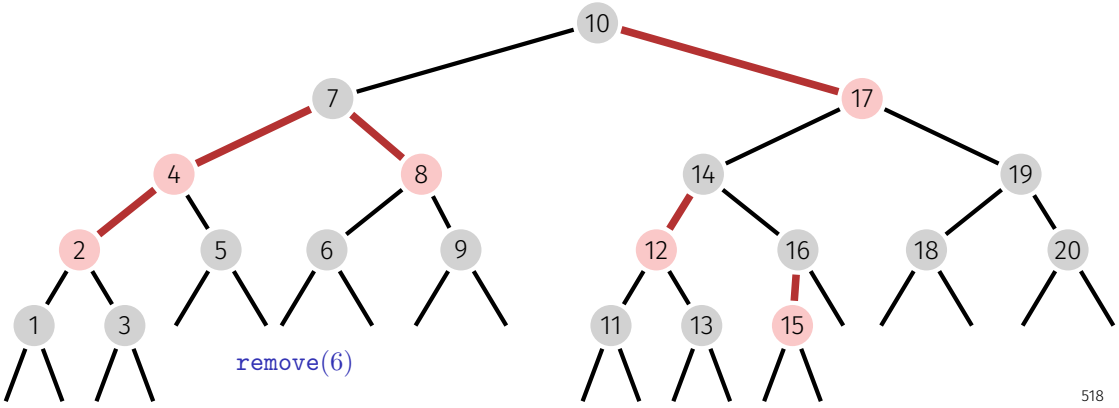
Beispiel: Schlüssel löschen



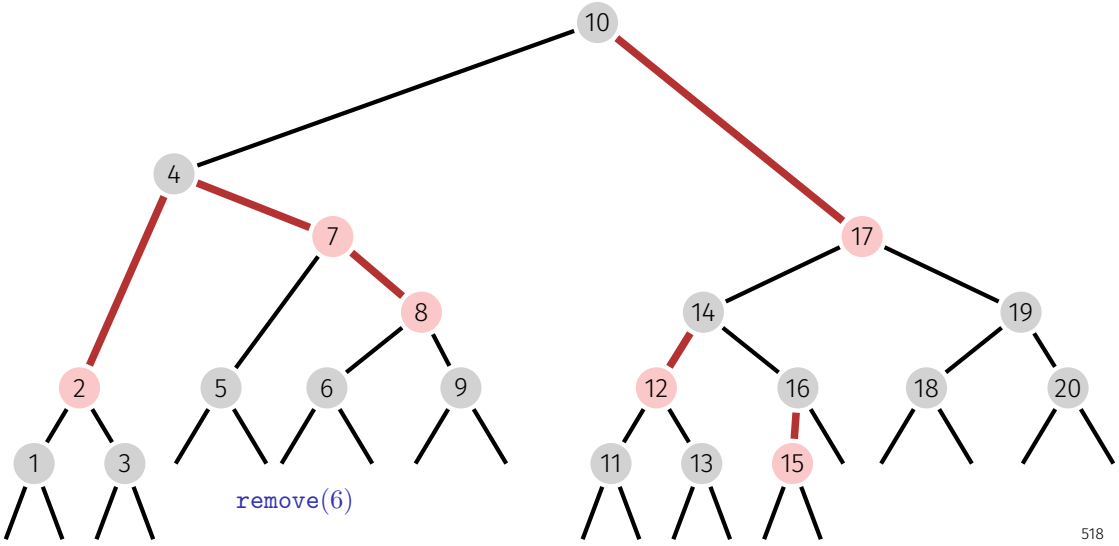
Beispiel: Schlüssel löschen



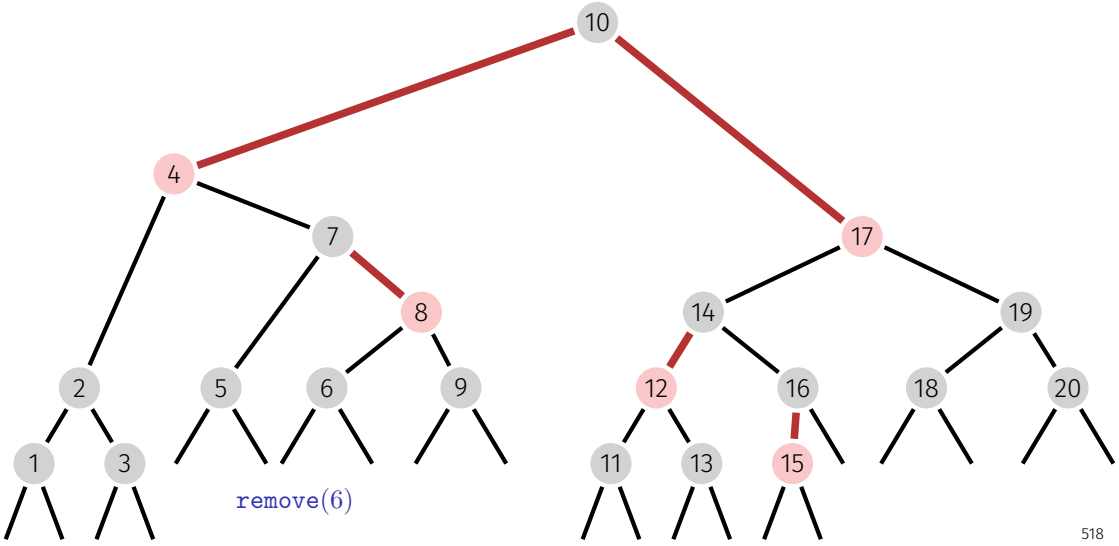
Beispiel: Schlüssel löschen



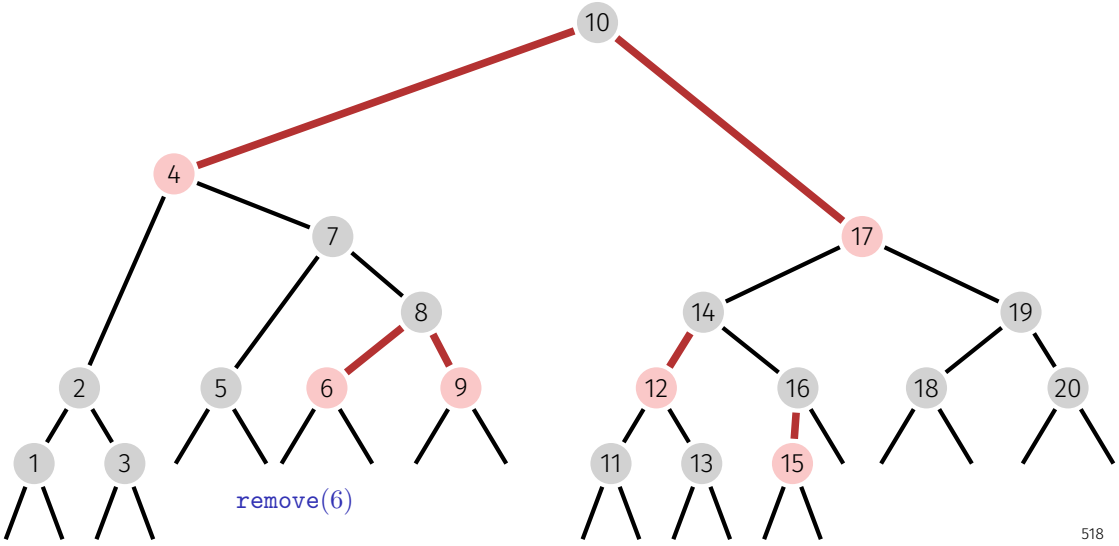
Beispiel: Schlüssel löschen



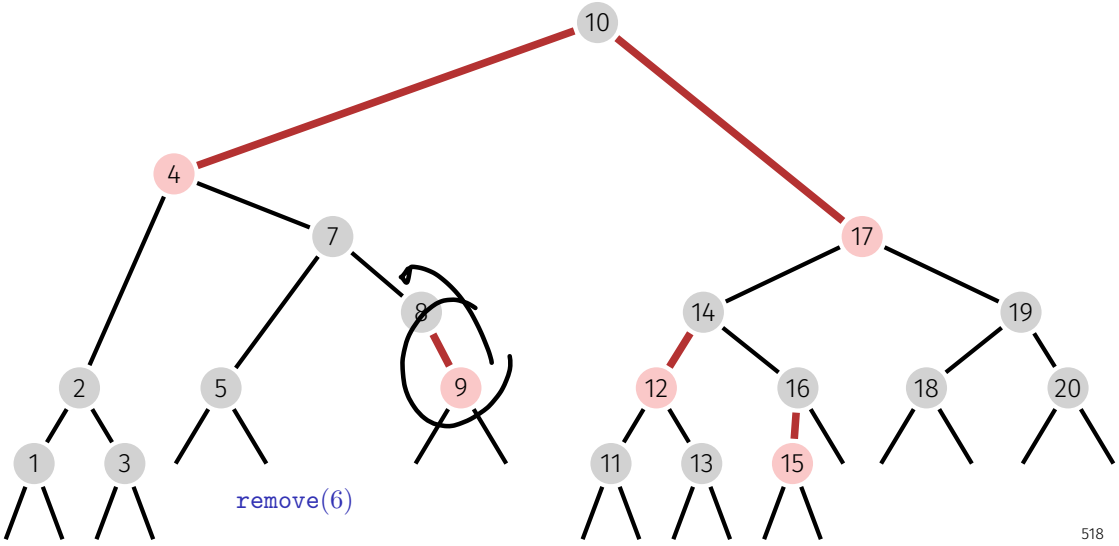
Beispiel: Schlüssel löschen



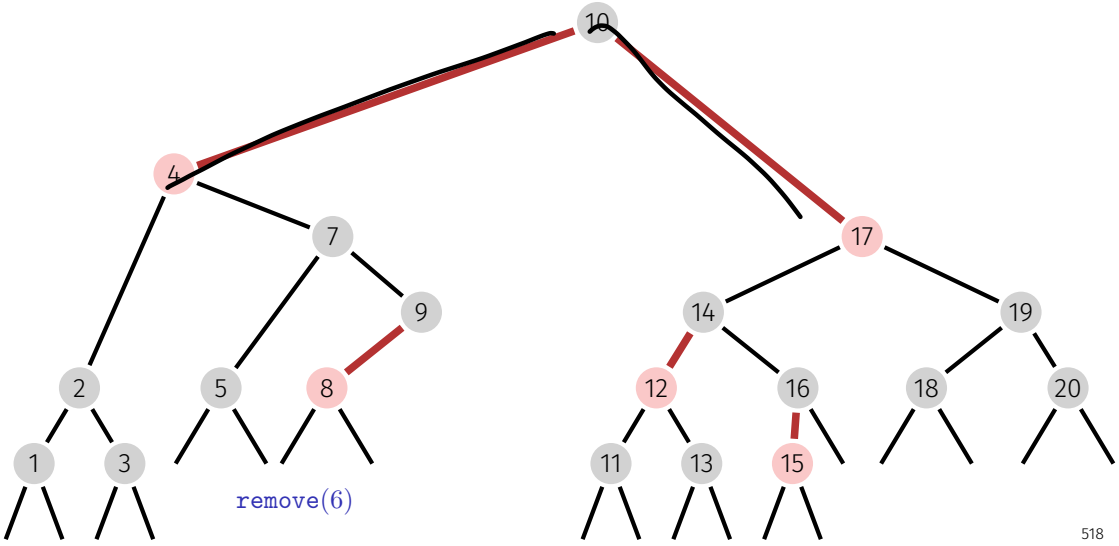
Beispiel: Schlüssel löschen



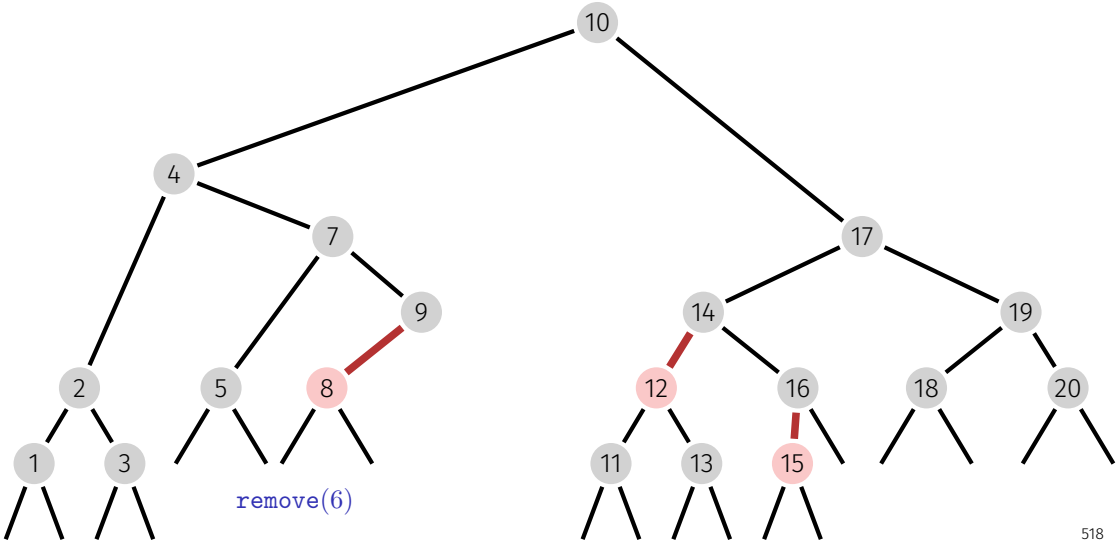
Beispiel: Schlüssel löschen



Beispiel: Schlüssel löschen



Beispiel: Schlüssel löschen



Laufzeit?

Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

$\Rightarrow$ Laufzeit $\mathcal{O}(h)$

Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

$\Rightarrow$ Laufzeit $\mathcal{O}(h)$

Was ist die Höhe des Rot-Schwarz-Baums?

Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

$\Rightarrow$ Laufzeit $\mathcal{O}(h)$

Was ist die Höhe des Rot-Schwarz-Baums?

$R :=$ Anzahl rote Kanten auf Pfad P

$S :=$ Anzahl schwarze Kanten auf Pfad P

Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

$\Rightarrow$ Laufzeit $\mathcal{O}(h)$

Was ist die Höhe des Rot-Schwarz-Baums?
$R :=$ Anzahl rote Kanten auf Pfad P
$S :=$ Anzahl schwarze Kanten auf Pfad P
Länge von Pfad P:

$$l(P) = R + S$$

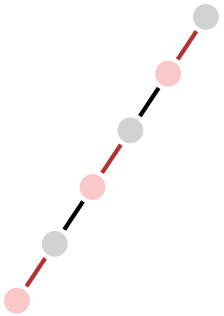
Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

$\Rightarrow$ Laufzeit $\mathcal{O}(h)$

Was ist die Höhe des Rot-Schwarz-Baums?
$R :=$ Anzahl rote Kanten auf Pfad P
$S :=$ Anzahl schwarze Kanten auf Pfad P
Länge von Pfad P:

$$l(P) = R + S \leq (S + 1) + S$$



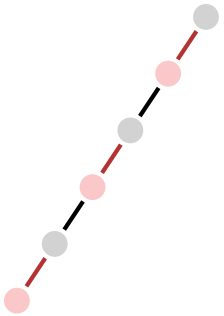
Laufzeit?

Für jede Operation gehen wir höchstens zweimal von Wurzel zu Blatt und zurück

$\Rightarrow$ Laufzeit $\mathcal{O}(h)$

Was ist die Höhe des Rot-Schwarz-Baums?
$R :=$ Anzahl rote Kanten auf Pfad P
$S :=$ Anzahl schwarze Kanten auf Pfad P
Länge von Pfad P:

$$\begin{aligned} l(P) &= R + S \leq (S+1) + S \\ &= 2 \cdot S + 1 \leq 2\lceil \log_2(n+1) \rceil + 1 \end{aligned}$$



[7] in RBT einfügen:

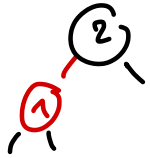
insert(1):

---

insert(2):

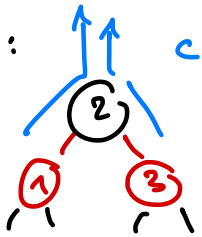
↯ rot-right!

⚠ auf Farben achten



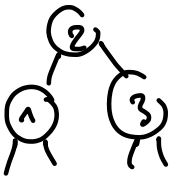
---

insert(3):



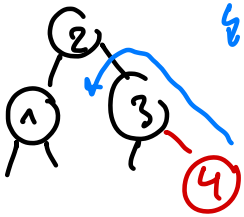
color flip!

↯ zwei rote Nachfahren

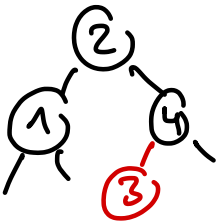


---

insert(4):

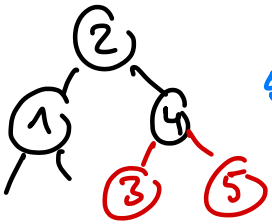


↯ rote rechte Kante!



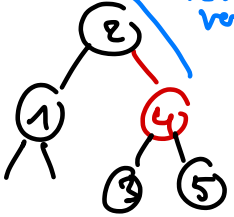
---

insert(5):

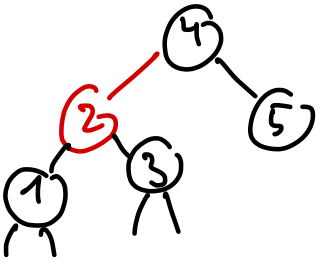


↯ zwei rot

Farbwechsel nicht vergessen

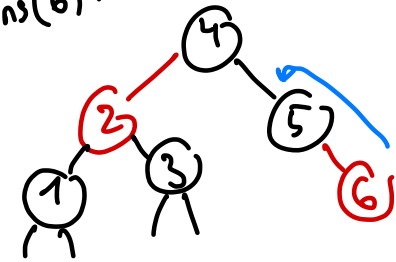


↯ rote Kante auf rechter Seite

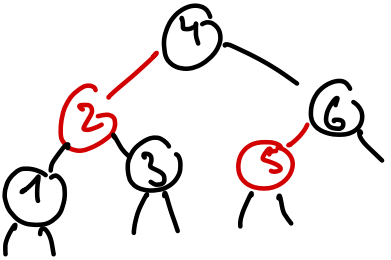


---

ins(6):

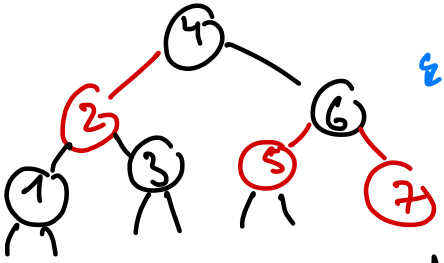


& right rad

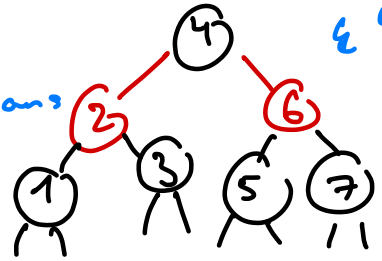


---

insert(7)



& red neighbors



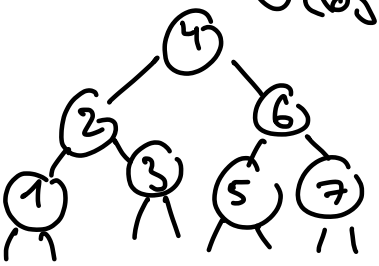
& color!

search

$O(\log(n))$

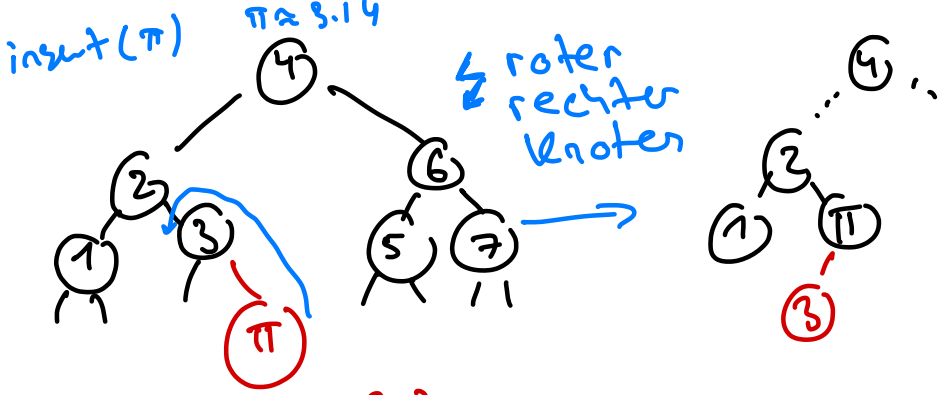
search

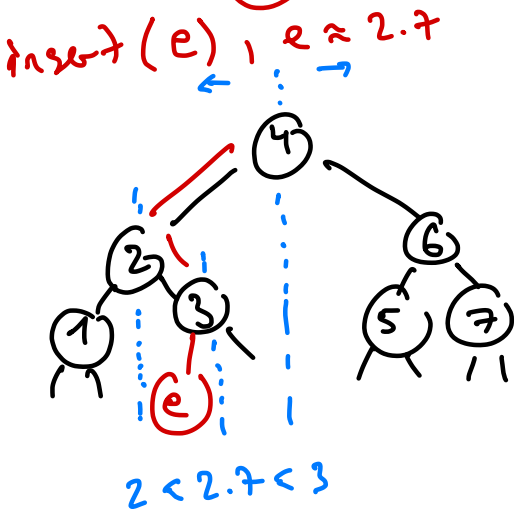
$O(n)$



RB-T

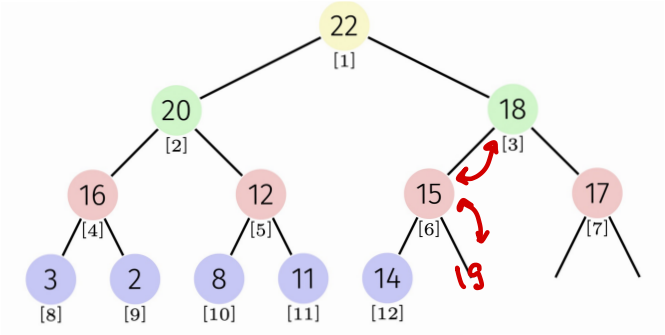
BST

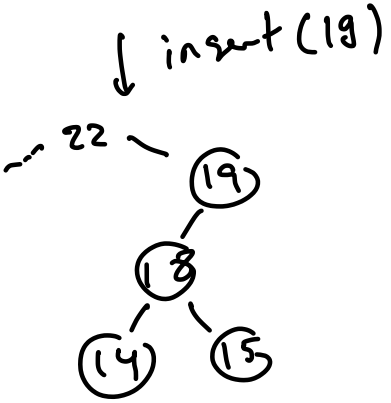




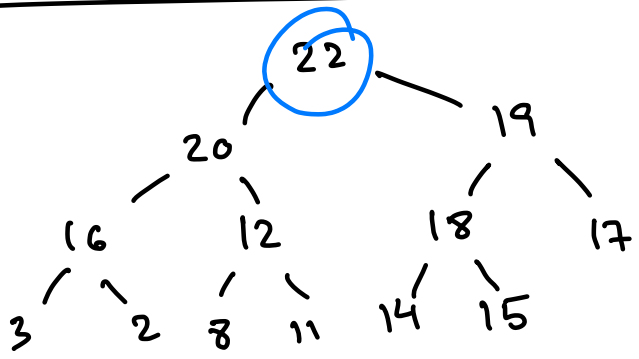
2 < 2.7 < 3

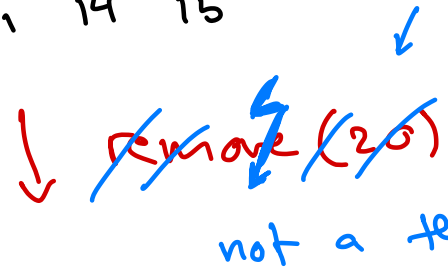
```
Insert(n){
  search(n)
  if(not found){
   - als roter Knoten einf.
      (rote Kante!)
   - rekursiv die
      RBT-P wiederherst.
  }else{
    done!
  }
}
```

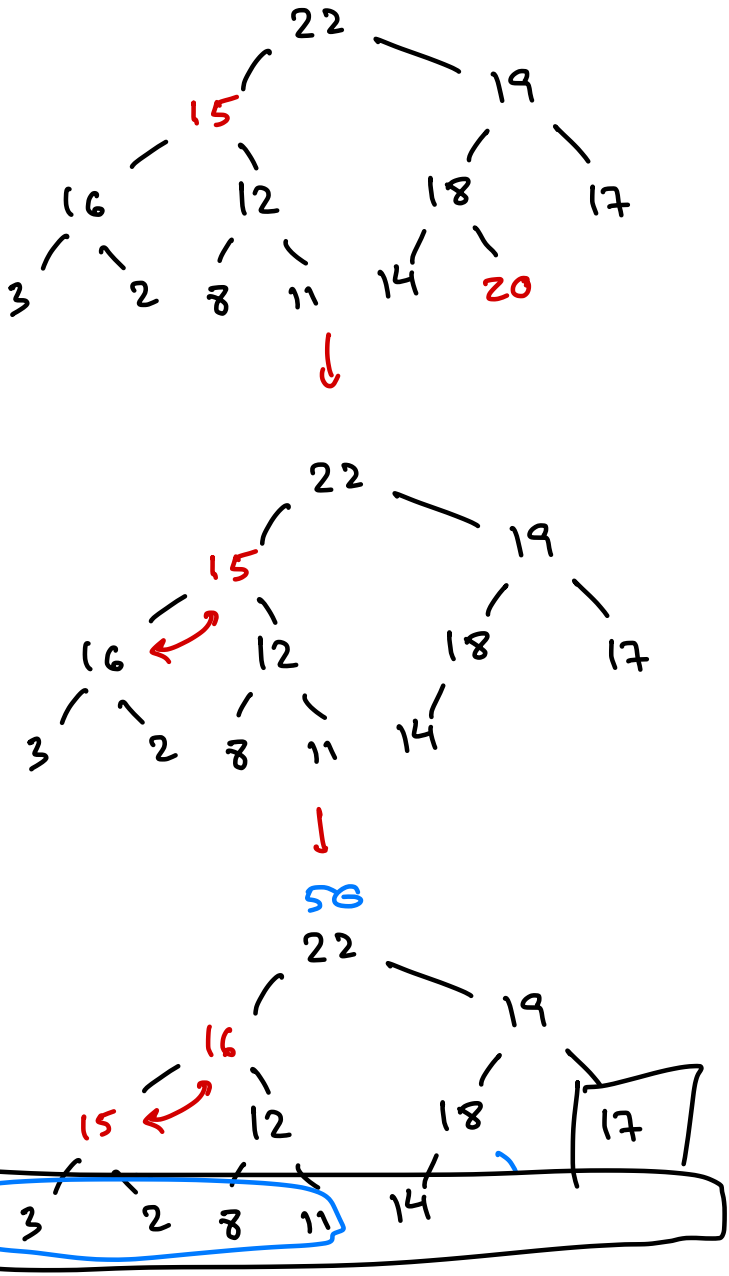




22 20 19 16 12 18 17 3 2 8 11 14 15






22
15
19
16
12
18
17
3
2
8
11
14
20
22
15
19
16
12
18
17
3
2
8
11
14
5G
22
16
19
15
12
18
17
3
2
8
11
14