Algorithmen und Datenstrukturen 1

Prof. Dr. Carsten Lecon

Rot-Schwarz-Bäume

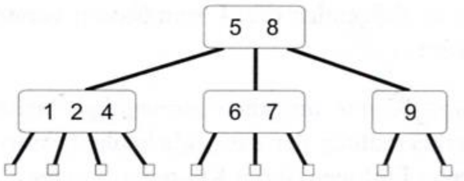
- Motivation
- Einführung
- Operationen auf RST

Rot-Schwarz-Bäume

- Motivation
  - BST sind günstig, falls die Höhe h des Baumes gering ist.
  - Dazu müssen Bäume ausbalanciert sein.
  - Aber: Höhe ist abhängig von der Reihenfolge des Einfügens der Knoten.

Rot-Schwarz-Bäume

- Eine Lösung:
  - 2-3-4-Bäume



- Beispiel:
  - Einfügen von 5-6-2-7-8-9-0-3-10-11

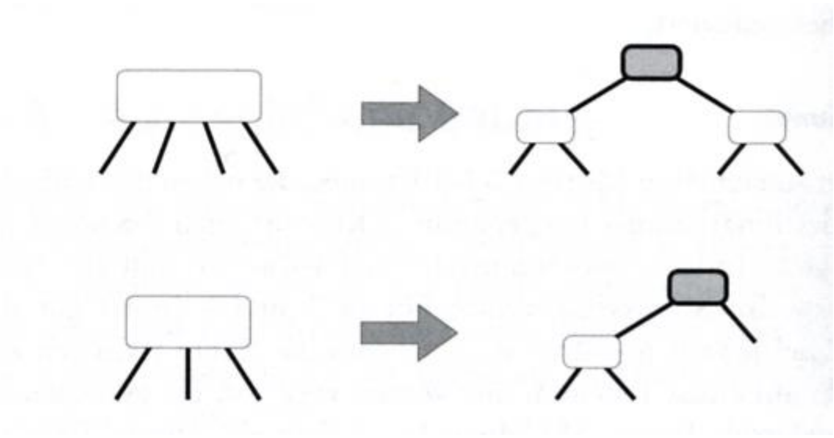
- Problem beim Einfügen:
  - Viererknoten müssen geteilt werden

Quelle: Saake, Sattler: „Algorithmen und Datenstrukturen“

Rot-Schwarz-Bäume

Motivation:

- Teilung → Binärbaumstruktur



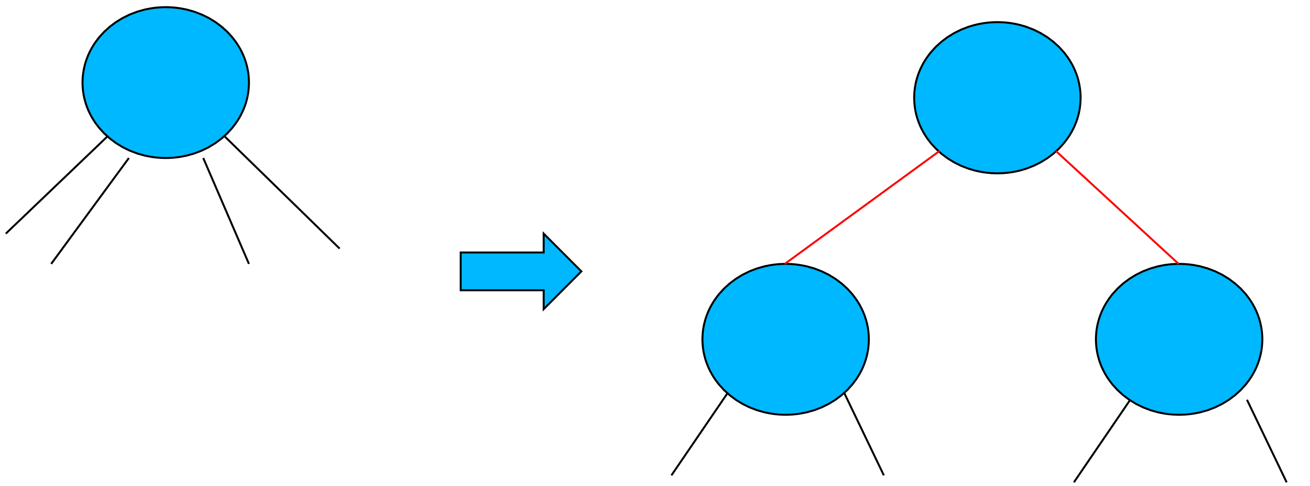
Quelle: Saake, Sattler: „Algorithmen und Datenstrukturen“

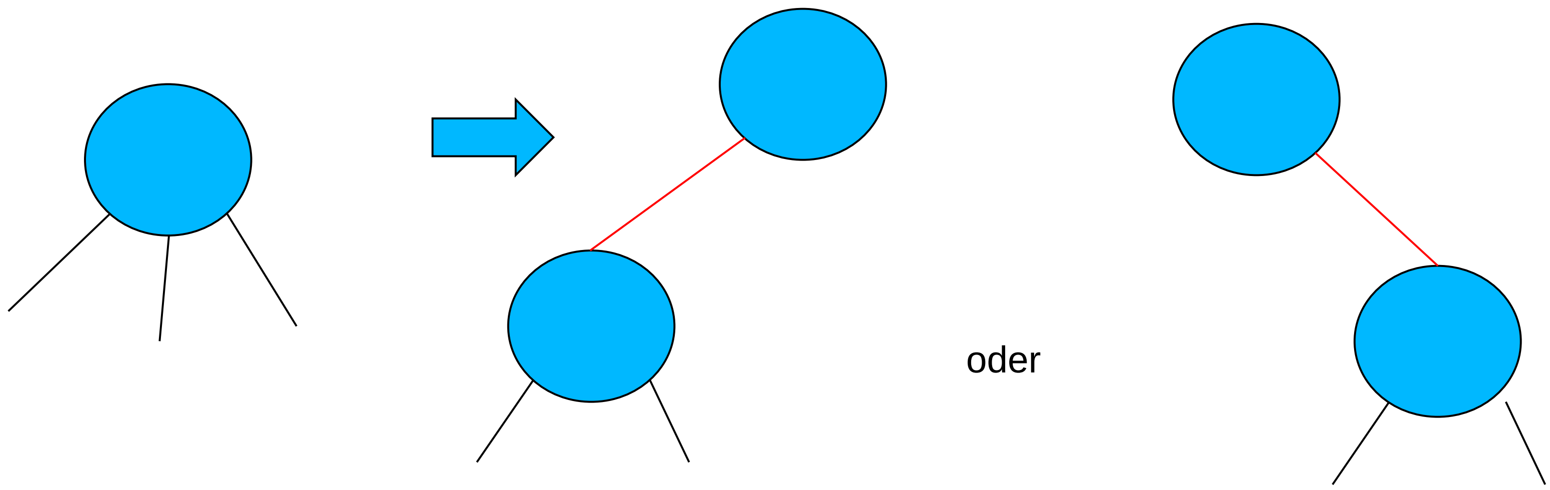
Rot-Schwarz-Bäume

2-3-4-Bäume als Binärbäume:

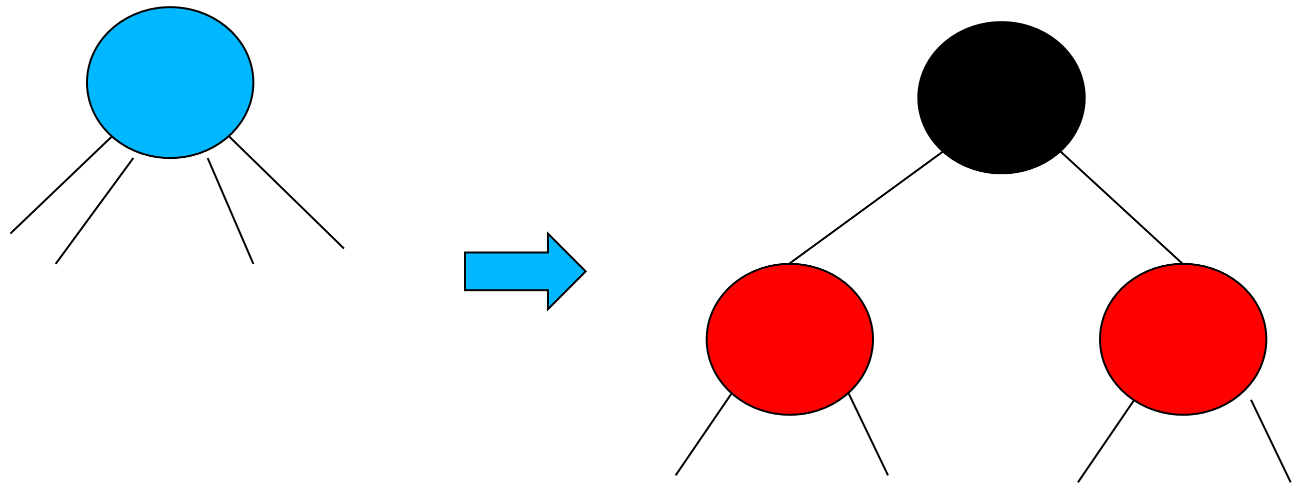
- 2er- und 3er-Knoten als kleine Binärbäume darstellen, die durch „rote“ Verkettungen miteinander verbunden sind.
- Die Vorzüge (Ausgeglichenheit) bleiben erhalten.

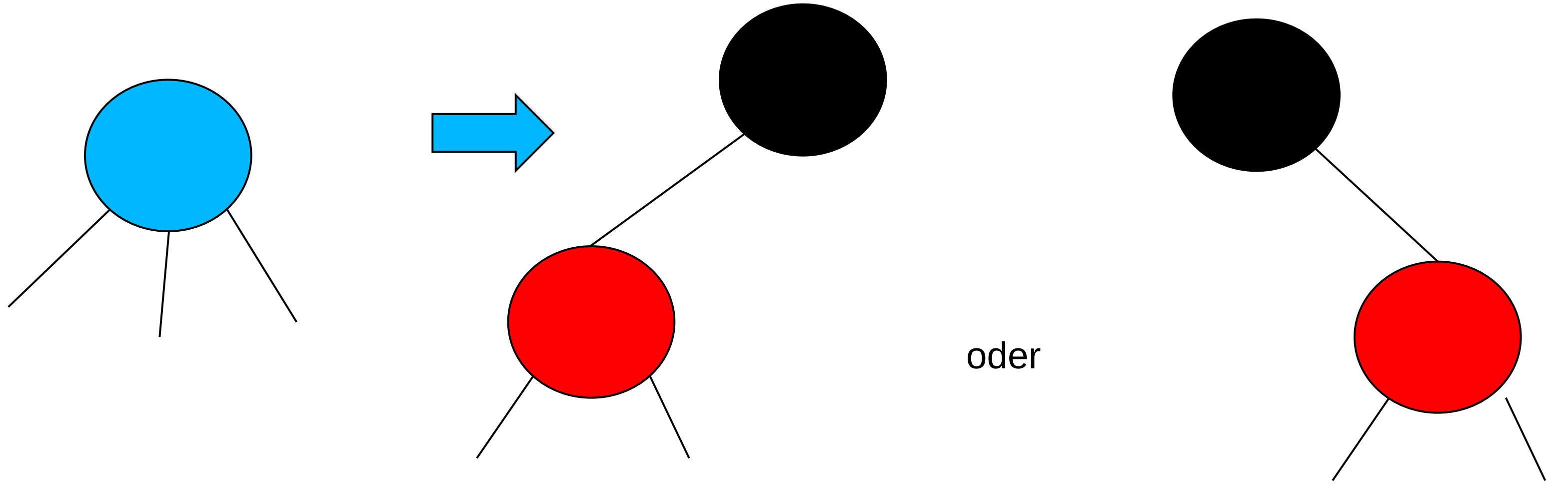
Rot-Schwarz-Bäume



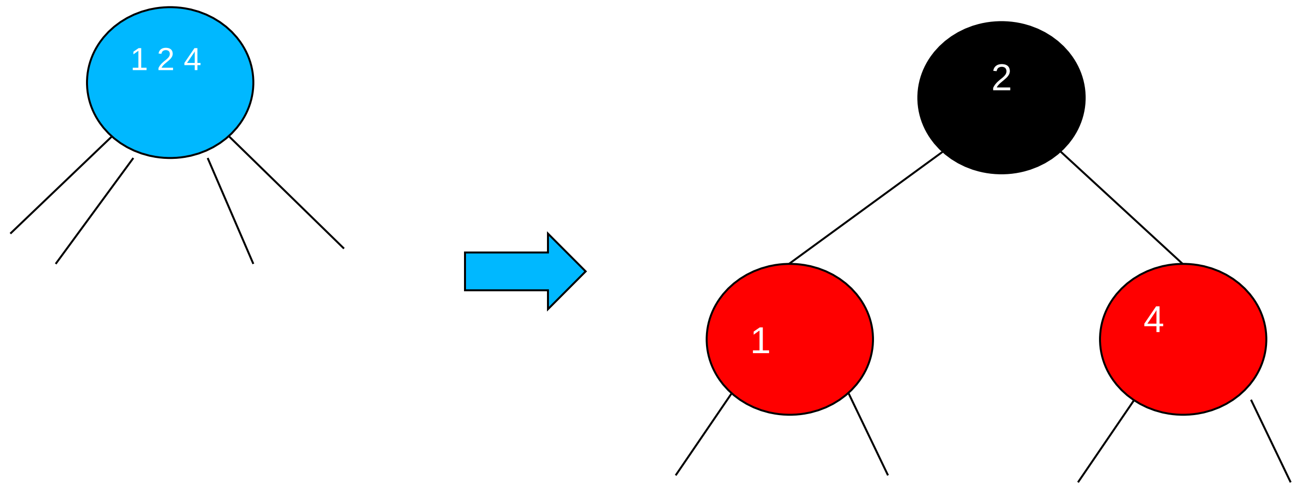


Rot-Schwarz-Bäume (Knotenfarbe)

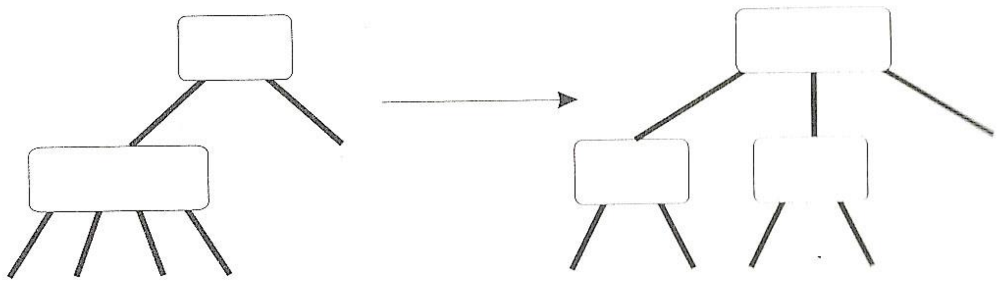


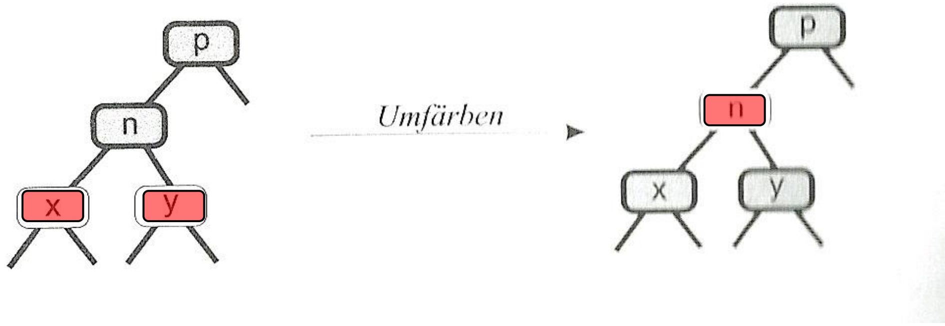


Rot-Schwarz-Bäume (Beispiel)



Splitten 2-3-4 vs. Rot-Schwarz





Entnommen aus: Saake, Sattler: „Algorithmen und Datenstrukturen“

Rot-Schwarz-Bäume

- Eigenschaften:
  - BST als Grundlage plus einige Erweiterungen
  - Knoten erhalten zusätzlich eine Farbe (rot/schwarz) → 1 Bit mehr.
  - Alle Knoten (einschl. Wurzel) werden eingefärbt.
  - Färbung unterliegt Regeln.
  - Anstelle NIL-Verweise: spezielle NIL-Knoten → Alle Knoten mit Schlüsseln sind innere Knoten.

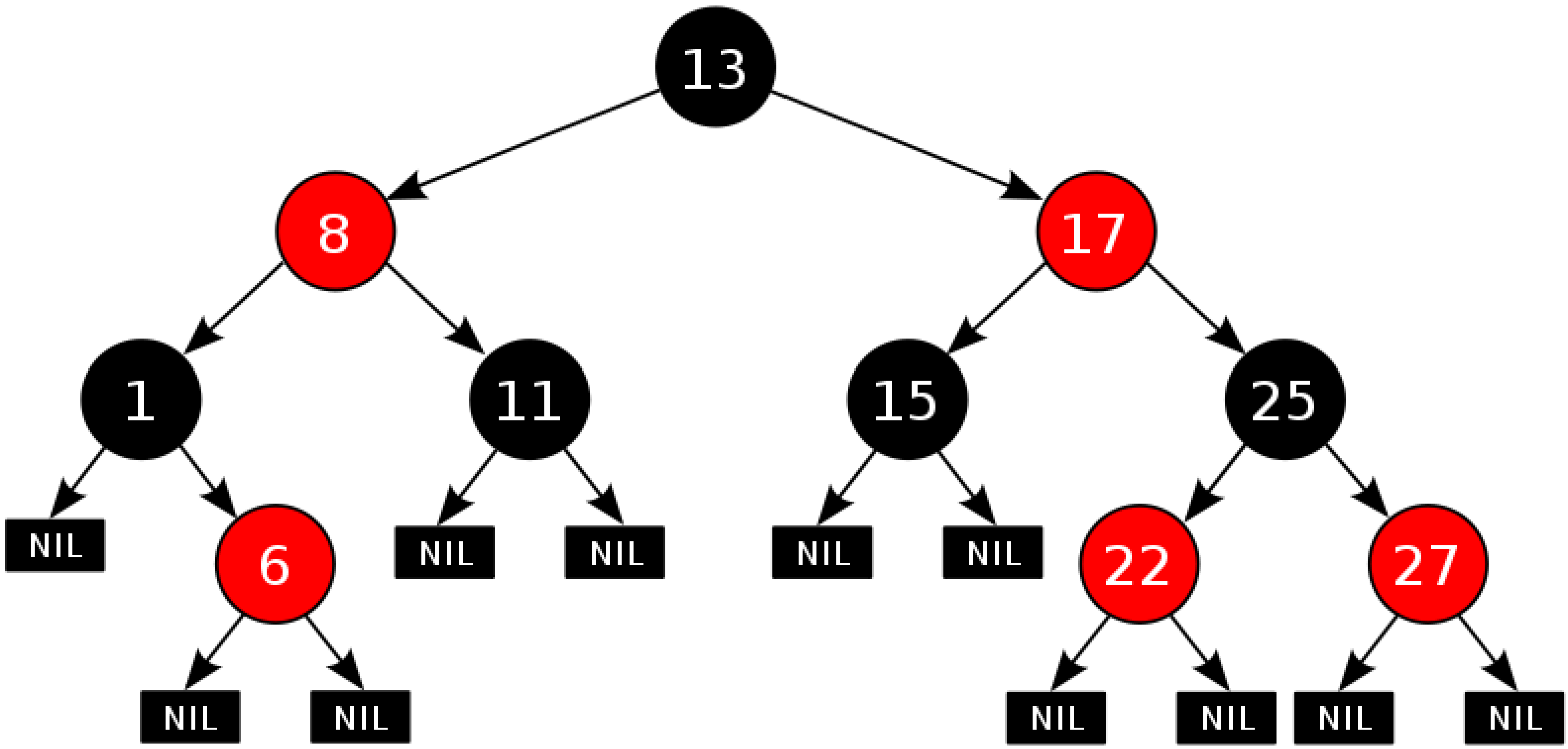
- Laufzeit (Einfügen, Löschen, Suchen) in garantiert O(log n)

Rot-Schwarz-Bäume

- Definition
  - Ein binärer Suchbaum heißt Rot-Schwarz-Baum (*Red-Black-Tree*, *RBT*), wenn folgende Eigenschaften gelten:

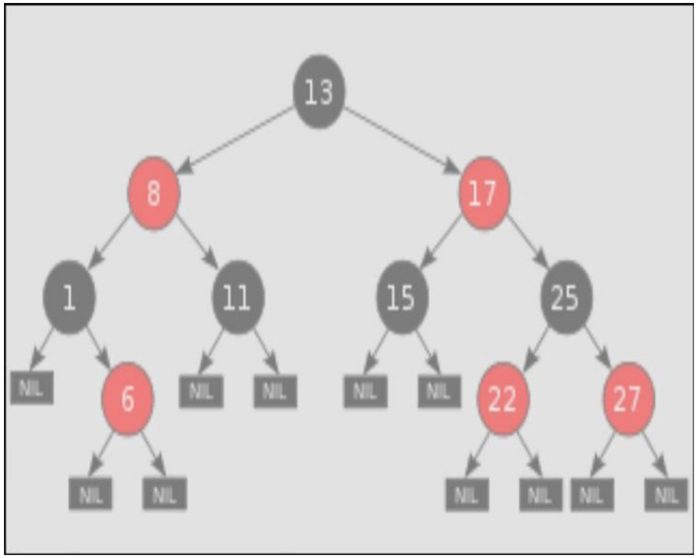
    1. Jeder Knoten ist entweder rot oder schwarz.
    2. Die Wurzel ist schwarz.
    3. Jedes Blatt (NIL-Knoten) ist schwarz.
    4. Wenn ein Knoten rot ist, müssen beide Kinder schwarz sein.
    5. Für jeden Knoten x gilt: Alle Pfade von x zu den Blättern enthalten die gleiche Anzahl an schwarzen Knoten.

Rot-Schwarz-Bäume



Quelle: Wikipedia

Rot-Schwarz-Bäume



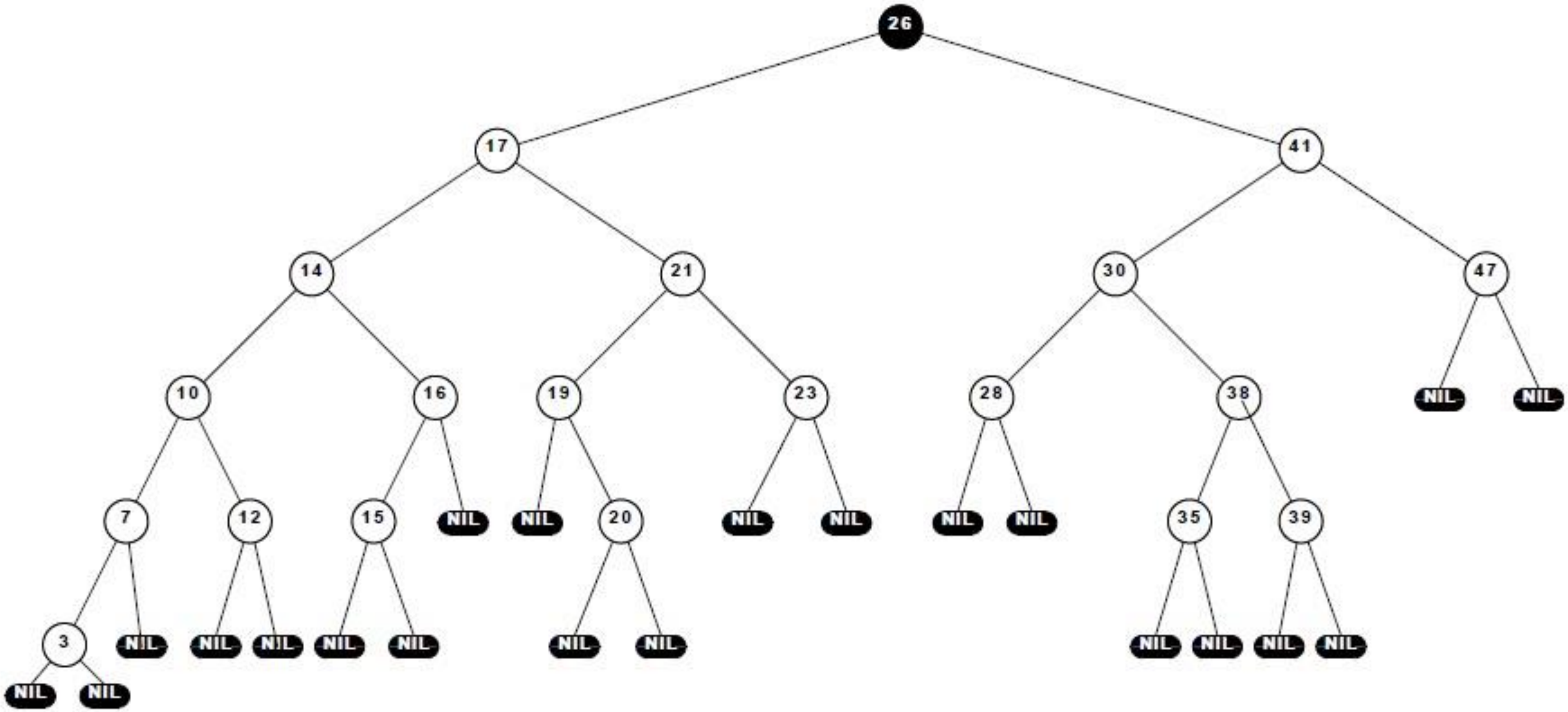
Die Quintessenz von Rot-Schwarz in 44 Sekunden ;-)

Rot-Schwarz-Bäume

- Einfärben:
  - Wurzel schwarz färben
  - Alle NIL-Knoten schwarz färben
  - Längsten Weg suchen und abwechselnd einfärben; dann kann der kürzeste Weg gerade noch gefärbt werden.

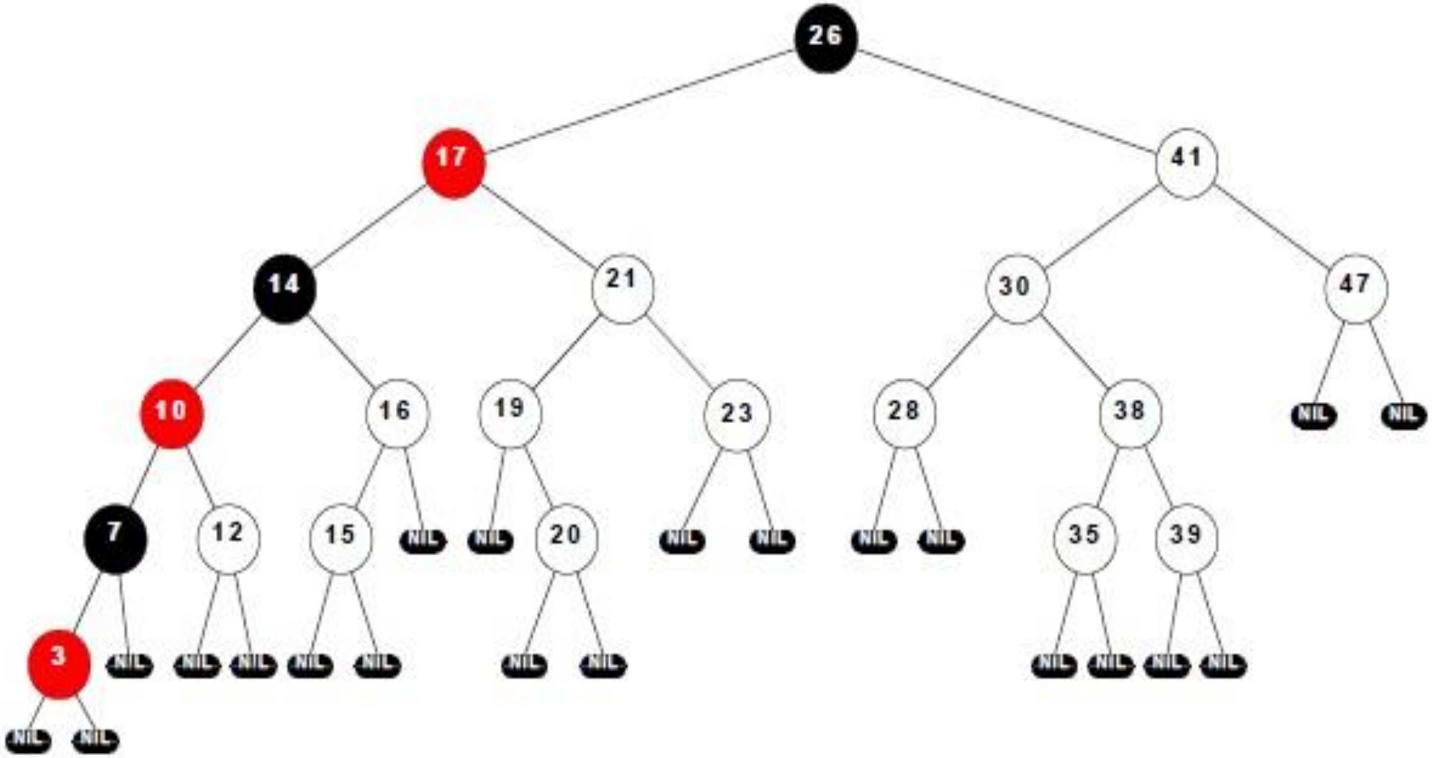
- Die Anzahl der Knoten auf dem längsten Pfad ist nie mehr als doppelt so hoch wie die Anzahl der Knoten des kürzesten Weges von der Wurzel zu einem Blatt.

Rot-Schwarz-Bäume



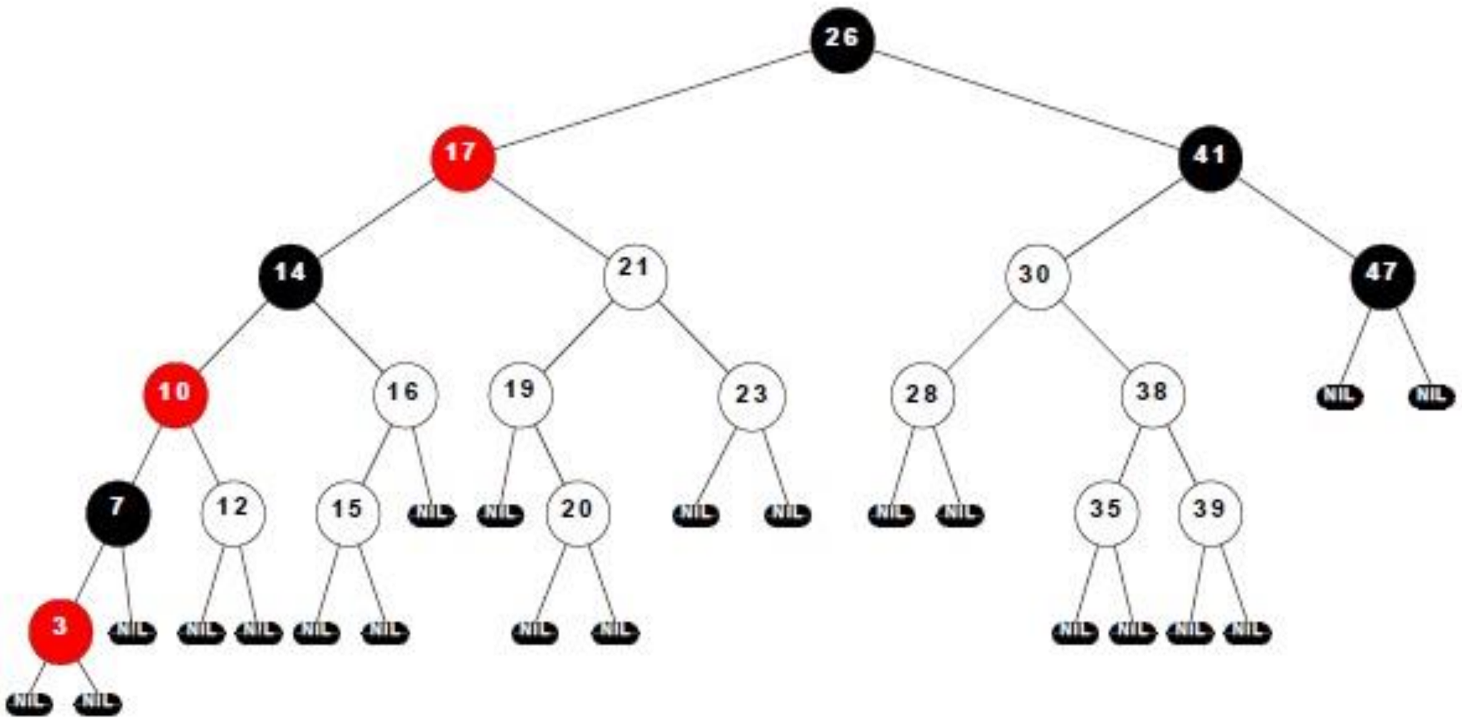
Unmarkierter Baum → Längsten Weg einfärben

Rot-Schwarz-Bäume



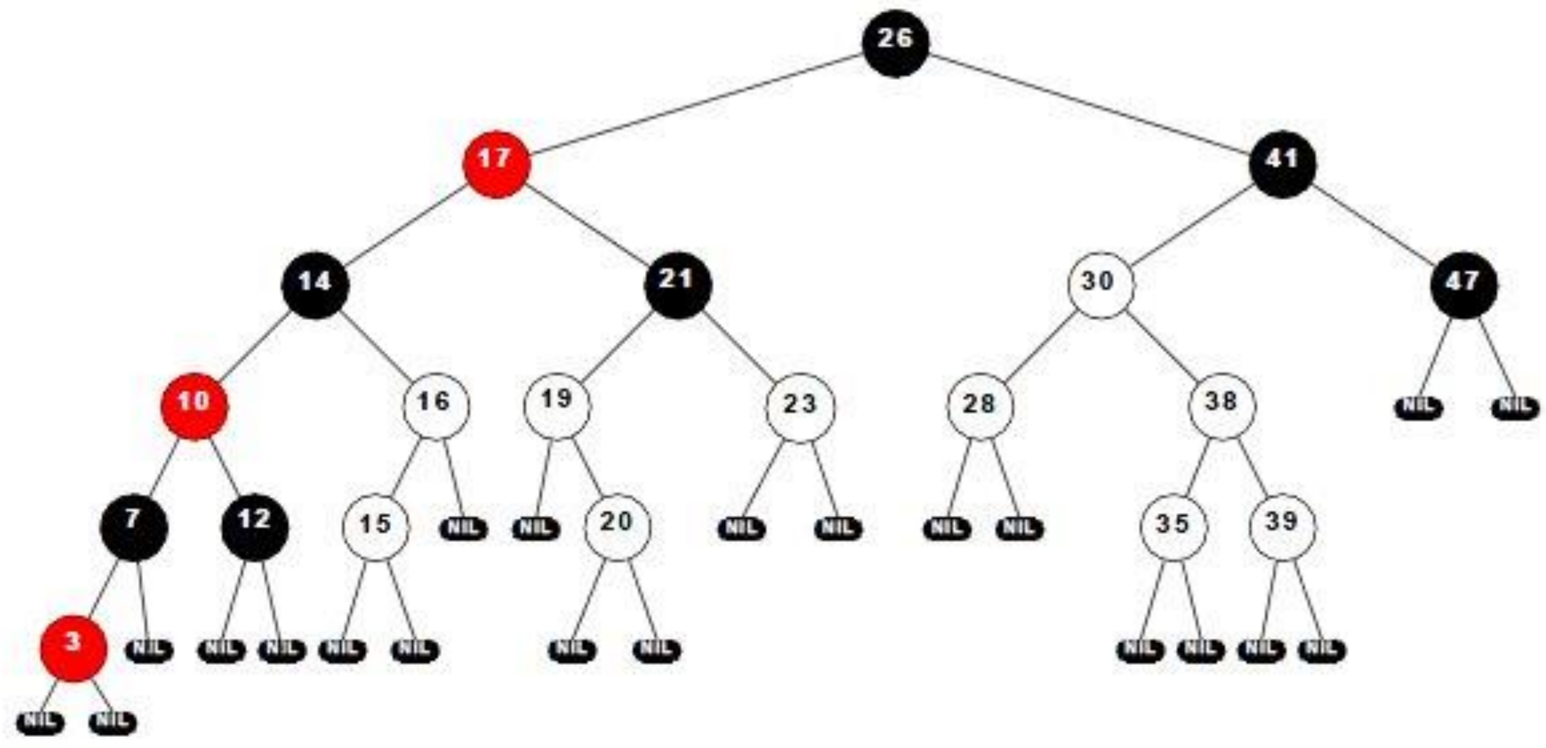
Kürzesten Weg einfärben

Rot-Schwarz-Bäume



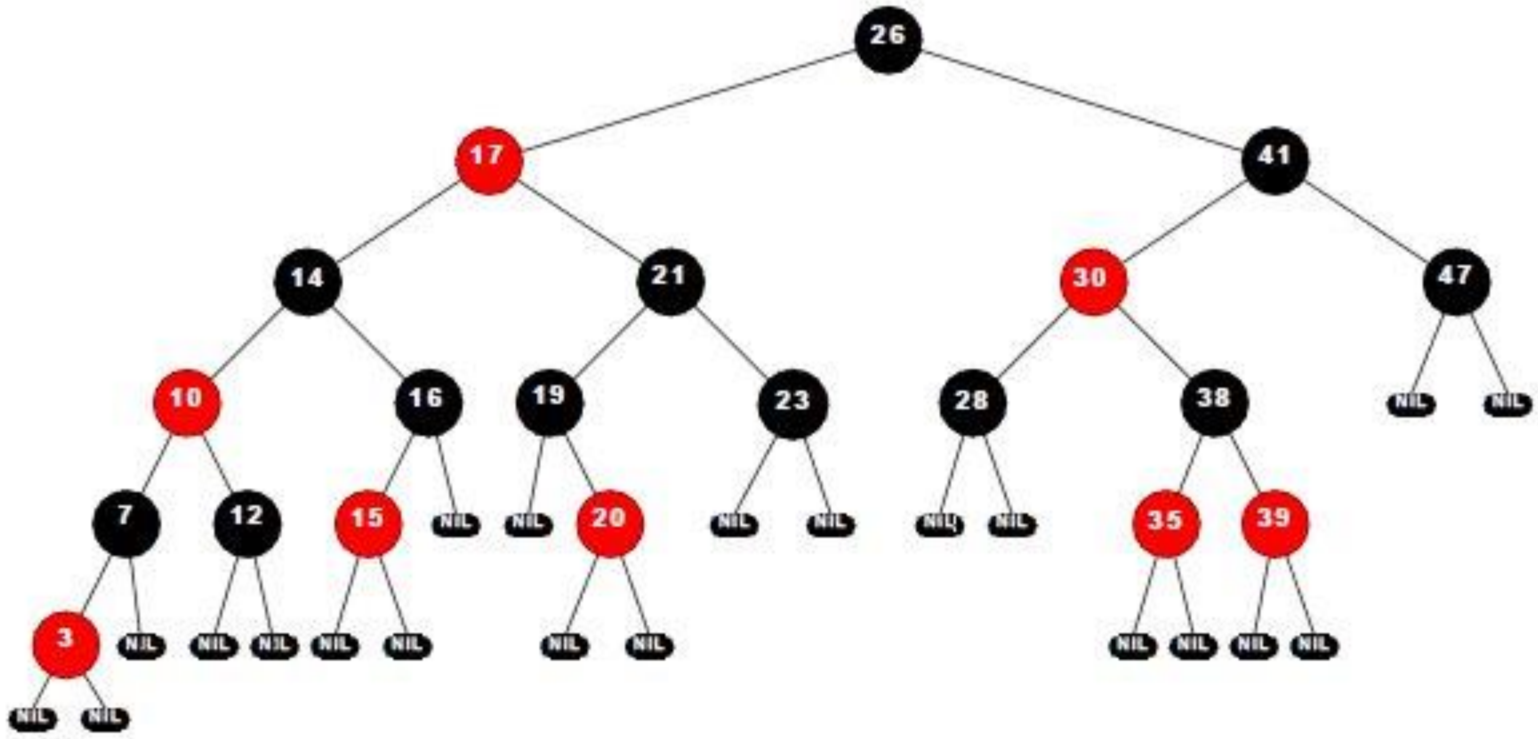
Kinder von roten Knoten schwarz einfärben

Rot-Schwarz-Bäume



Regel (5) berücksichtigen: Alle Pfade von x zu den Blättern enthalten die gleiche Anzahl schwarzer Knoten.

Rot-Schwarz-Bäume



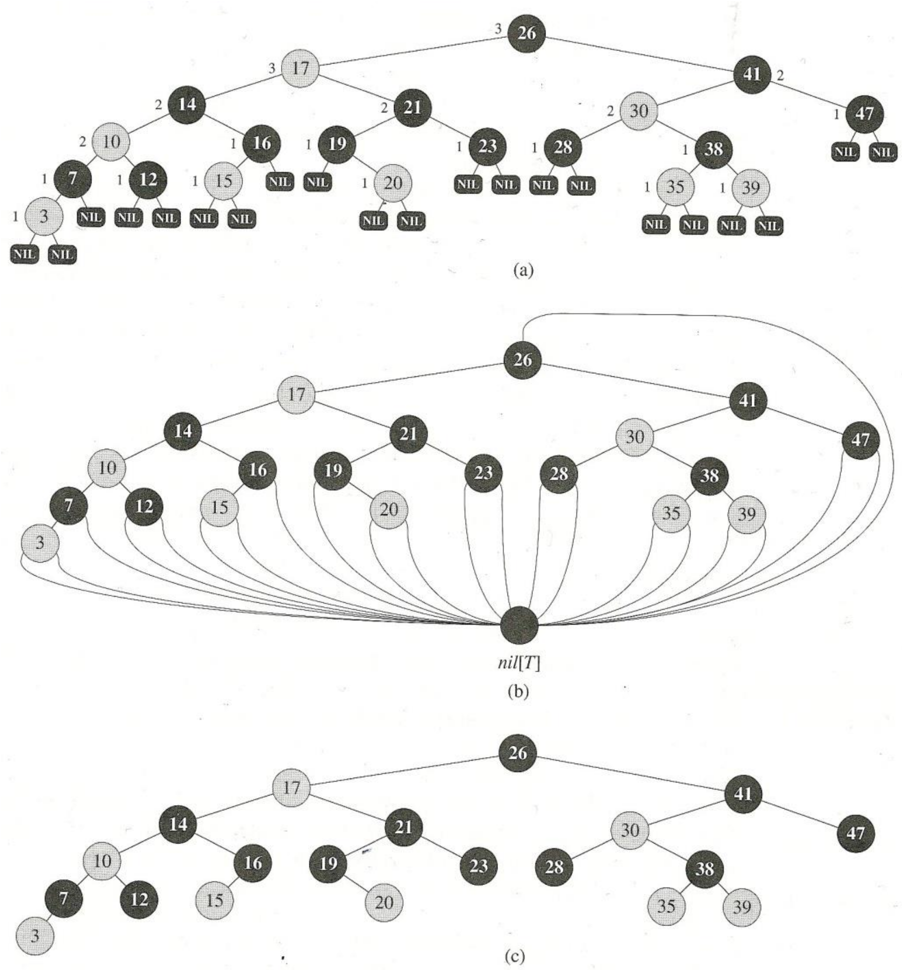
Fertig (NIL-Knoten werden i.d.R. nicht gezeichnet).

Rot-Schwarz-Bäume

- Festlegungen:
  - Im Folgenden nur ein NIL-Knoten für den gesamten Baum
  - Gezeichnet wird ohne NIL-Knoten

- Definition Schwarzhöhe (*bh(x)*)
  - Die **Schwarzhöhe** *bh(x)* eines Knotens x ist die Anzahl der schwarzen Knoten auf dem Weg vom Knoten x zu den Blättern (ohne x selbst).
- Definition Höhe eines Knotens
  - Die **Höhe eines Knotens** die Anzahl der Kanten auf dem längsten, einfachen Weg von dem Knoten zu einem Blatt.

Rot-Schwarz-Bäume



Rot-Schwarz-Bäume

Satz

Ein RS-Baum mit n inneren Knoten hat die Höhe
$h \leq 2\log_2(n+1)$.

Beweis

1. Es wird bewiesen, dass jeder Teilbaum mit Wurzel x mind. $2^{bh(x)}-1$ innere Knoten hat.
2. Diese Zwischenergebnis wird genutzt, um aus der Schwarzhöhe der Wurzel die Anzahl der inneren Knoten auszurechnen.

Rot-Schwarz-Bäume

Beweis Hilfssatz:

- Jeder Teilbaum mit Wurzel x hat mind. $2^{bh(x)}-1$ innere Knoten.
- Induktionsbeweis:
  - 1. Falls Höhe von x=0 (x ist Blatt): $2^{bh(x)}-1 = 2^0-1 = 0$
  - 2. Falls Höhe x>0:
  - Kinder von x haben Schwarzhöhen von
    - •bh(x), falls Kind rot,
    - •bh(x)-1, falls Kind schwarz.
  - x hat genau zwei Kinder
    - •Jedes Kind hat Schwarzhöhe bh(x) (x ist schwarz) oder bh(x)-1 (x ist rot)
    - •Höhe Kinder kleiner als Höhe von x: Jedes Kind hat mind. $2^{bh(x)-1}-1$ Knoten (Induktionsannahme)

      → Teilbaum mit Wurzel x hat mind. so viele innere Knoten: $(2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)+1 = 2^{bh(x)}-1$.

Rot-Schwarz-Bäume

- Hauptbeweis:
  - Sei `h` die Höhe des Baumes.
  - Mind. die Hälfte der Knoten auf dem Weg eines einfachen Pfades von der Wurzel zu einem Blatt sind schwarz (Eigenschaft 4).

  → Schwarzhöhe der Wurzel ist mind. `h/2`.
  - Rechnung:

$$n \geq 2^{h/2}-1$$

$$n+1 \geq 2^{h/2}$$

$$\log_2(n+1) \geq h/2$$

$$h \leq 2\log_2(n+1)$$

Rot-Schwarz-Bäume

Was bedeutet dies?

- Die folgenden Operationen haben alle eine Rechenzeit von O(h) = O(log n).
- Übernahme der Operationen von den binären Suchbäumen.
- Notwendige Änderungen:
  - Ersetzung der NIL-Zeiger durch Verweise auf NIL-Knoten
  - (Teilweise) Neuimplementierung von *insert* und *delete*, um Einhaltung der RS-Eigenschaften zu gewährleisten

Rot-Schwarz-Bäume

- Motivation
- Einführung
- Operationen auf RST

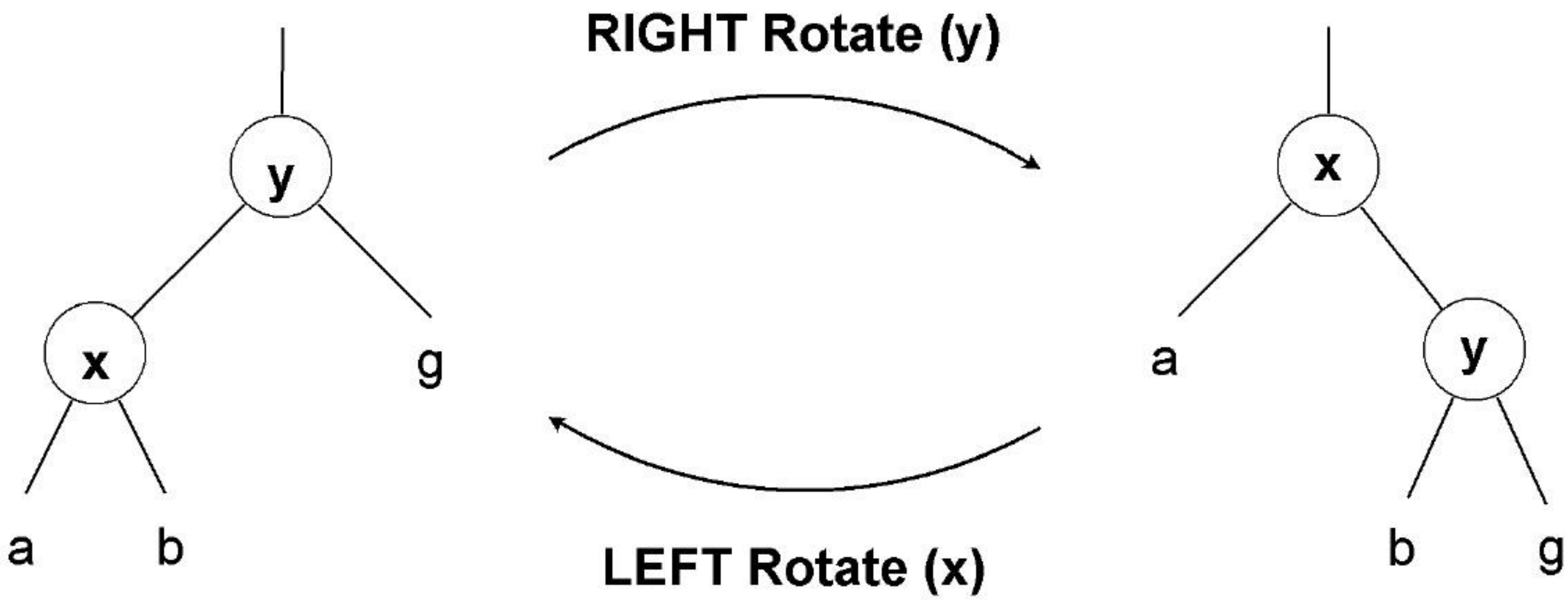
Rot-Schwarz-Bäume

Rotation

- Verletzung der RS-Eigenschaften durch modifizierende Operationen
- Wiederherstellung der RS-Eigenschaften durch Farbänderung (und Zeigeränderung) von Knoten
- Rotation
  – Wichtige Grundoperation
  – Ist eine lokale Operation mit Laufzeit O(1).

Rot-Schwarz-Bäume

Rotation



Rot-Schwarz-Bäume

```
function left-rotate(root, x)
  y = right(x)
  right(x) = left(y)
  parent(left(y)) = x
  parent(y) = parent(x)
  if parent(x) = nilNode then root = y
  else
    if x=left(parent(x)) then left(parent(x)) = y
    else                      right(parent(x)) = y
  end if
  left(y) = x
  parent(x) = y
  return root
end function
```

Rot-Schwarz-Bäume

Rotation:

- Rechtsrotation ist symmetrisch.
- Laufzeit ist $O(1)$.

Rot-Schwarz-Bäume

- Beispiel L-/R-Rotation und Einfärbung

Rot-Schwarz-Bäume

- Einfügen:
  - Verwendung des Algorithmus bei binären Suchbäumen

- Änderungen:
  - NIL-Verweise
  - Initialisierung der Farbe: neuer Knoten rot
  - Falls erforderlich: Farbkorrektur im Baum
  - Falls erforderlich: Umstrukturierung des Baumes

Rot-Schwarz-Bäume

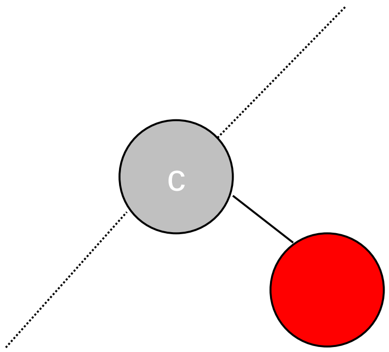
Einfügen:

- Wie bei BST:
  - Finde geeigneten Platz
  - Hänge den neuen Knoten an
- Was kann passieren?
  - Schwarzfärbung des neuen Knotens → ggf. Verletzung der Schwarz-Höhen-Bedingung
  - Rotfärbung des neuen Knotens → ggf. Verletzung der Farbbedingung (Wurzel ist schwarz, rote Knoten haben keine Kinder)
  - → Rotfärbung ist leichter zu korrigieren als Schwarz-Höhen-Verletzung.

Rot-Schwarz-Bäume

Einfügen: Fall 1

- Vater schwarz → kein Problem
- Vater rot → Rot-Rot-Verletzung
  - → Onkel muss betrachtet werden

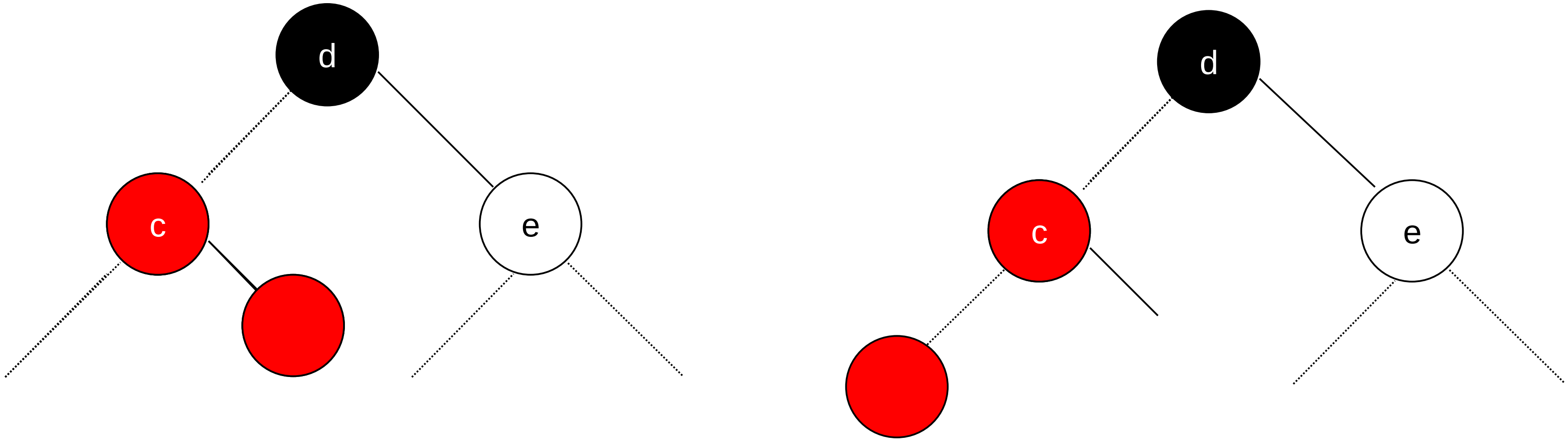


(Neuer Knoten kann auch links angehängt sein, etc.)

Rot-Schwarz-Bäume

Einfügen: Fall 1b

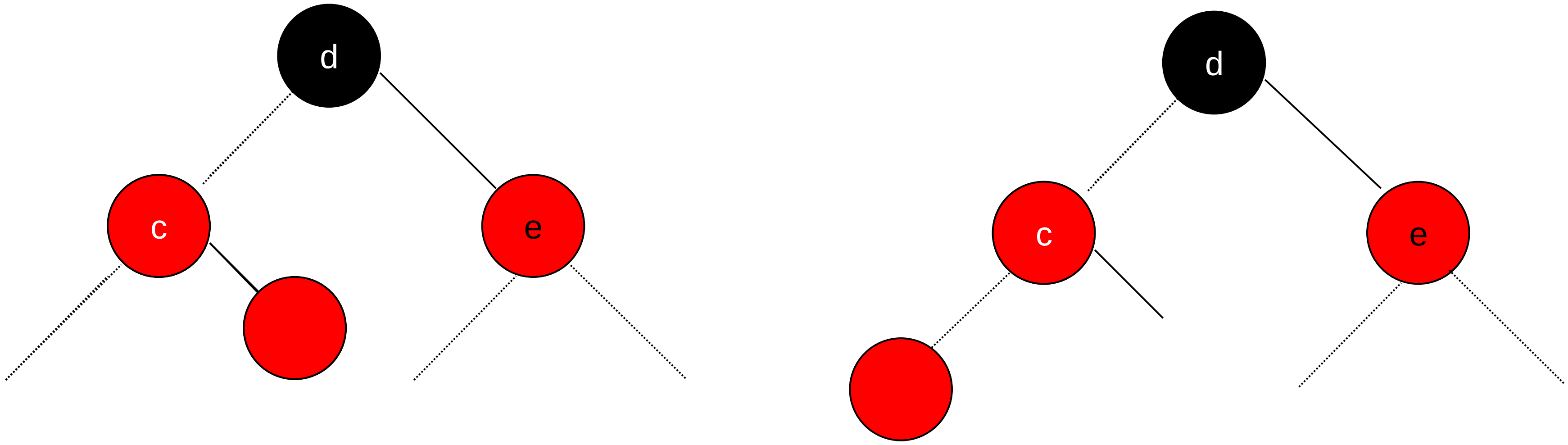
- Vater (c) rot → Onkel (e) betrachten
- Großvater (d) ist schwarz (gemäß RT-Regel)



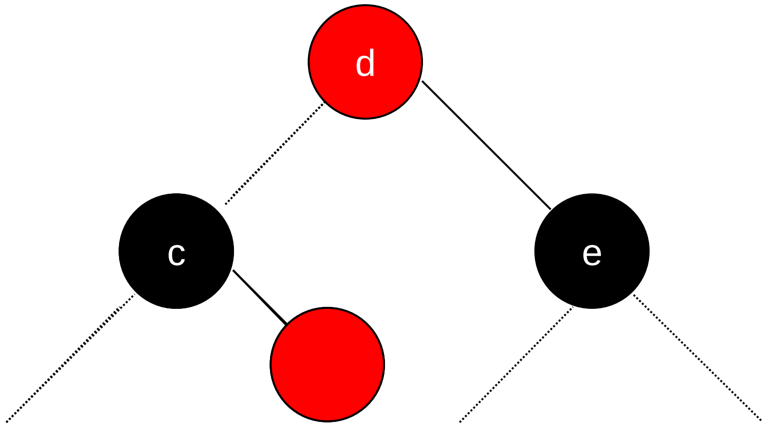
Rot-Schwarz-Bäume

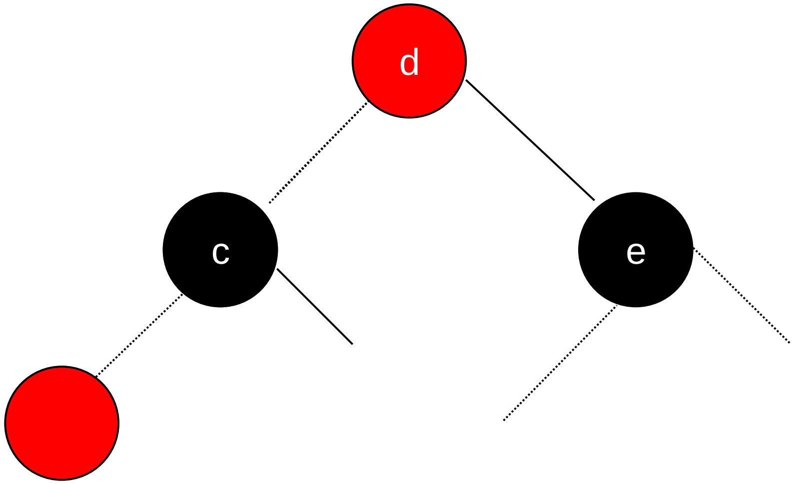
Einfügen: Fall 1b

- Vater (c) rot → Onkel (e) betrachten
- Großvater (d) ist schwarz (gemäß RT-Regel)
- Onkel (e) ist rot:
  - → Umfärben von Eltern und Onkel auf schwarz
  - → Umfärben von Großvater auf rot (wenn Großvater Wurzel: wieder schwarz färben)
  - → Ggf. iterativ fortsetzen



Rot-Schwarz-Bäume

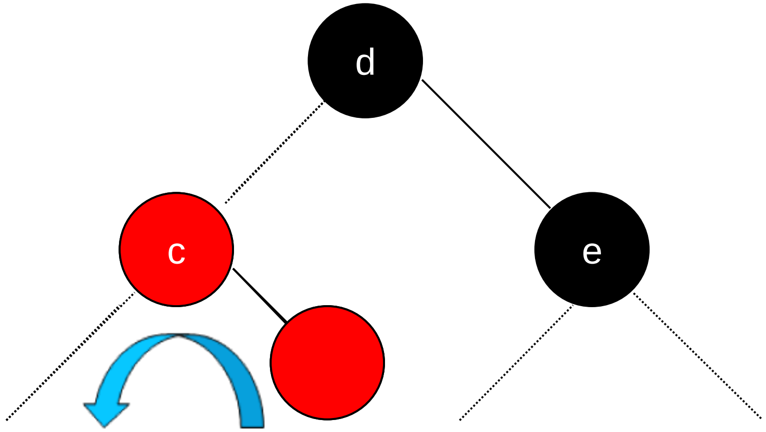




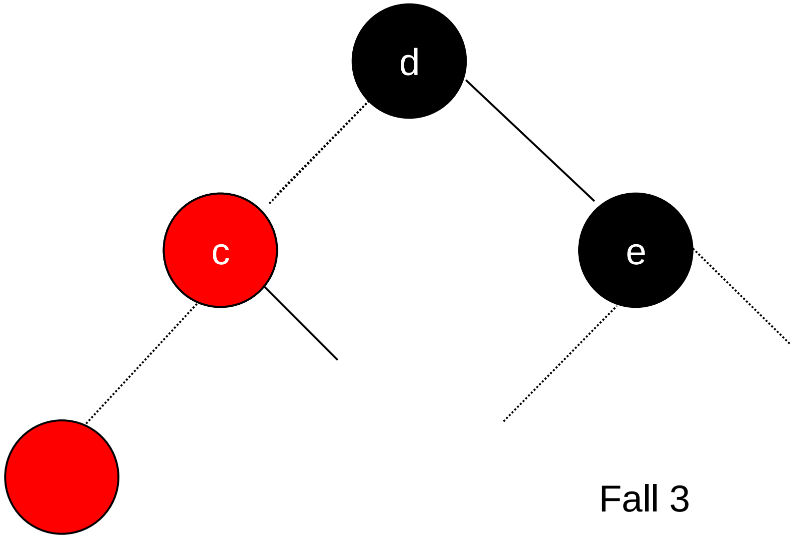
Rot-Schwarz-Bäume

Einfügen: Fall 2 und 3

- Vater (c) rot → Onkel (e) betrachten
- Großvater (d) ist schwarz (gemäß RT-Regel)
- Onkel (e) ist schwarz
- Fall 2 auf Fall 3 reduzierbar (Linksrotation)



Fall 2

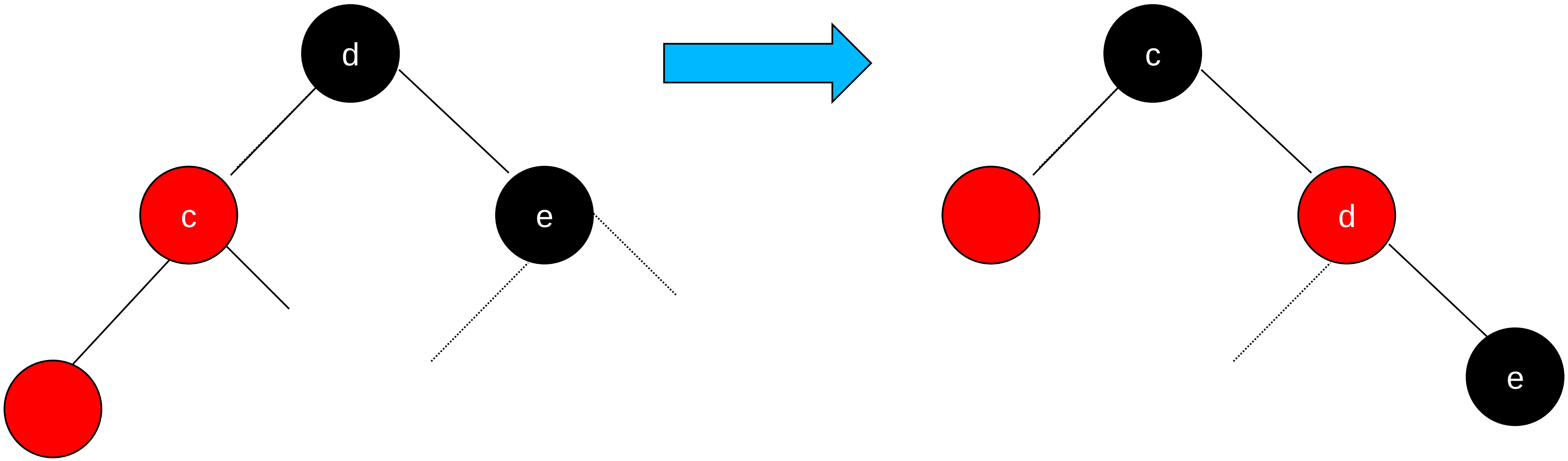


Fall 3

Rot-Schwarz-Bäume

Vorgehen bei Fall 3:

1. Rechtsrotation um d (Großvater)
   (Schwarzhöhen bleiben im Einklang)
2. Knoten d (ehemals Großvater) rot färben
3. Knoten c (ehemals Vater) schwarz färben



Rot-Schwarz-Bäume

```
function rb-insert(root, z)
  parent(z) = left(z) = right(z) = NIL
  if root=nilNode then
    color(z) = BLACK
    return z
  end if
  x = root
  while x≠nilNode do
     y = x
    if key(z) < key(  y) then x=left(y)
    else x=right(y)
    end if
  end while
```

Rot-Schwarz-Bäume

```
parent(z) = y
if key(z) < key(y) then left(y) = z
else right(y) = z
end if

/* Bisher identisch mit Einfügen in BST
(bis auf Einfärbung) */
```

Rot-Schwarz-Bäume

```
  color(z) = RED
  return rb-insert-fixup(root, z)
  /* Rückgabe des korrigierten RBT: */
  end function
```

Rot-Schwarz-Bäume

Die nachfolgende Funktion stellt die RT-Baumeigenschaft wieder her. Rückgabewert ist der korrigierte Baum.

```
function rb-insert-fixup(rt, z)
  while color(parent(z))  = RED do
    /* Elternknoten von z ist also nicht Wurzel */

    /* Schleife: Zwei symmetrische Teile für die Fälle
      a) Elternknoten(z) ist linkes Kind
      b) Elternknoten(z) ist rechtes Kind
    */
```

Rot-Schwarz-Bäume *(Elternknoten ist linkes Kind)*

```
if parent(z) = left(parent(parent(z)) then
  y = right(parent(parent(z))) // Großvater
  if color(y) = RED then // Fall 1 L
    color(parent(z)) = BLACK  // Vater
    color(y) = BLACK
    color(parent(parent(z)) = RED  // Großvater
    z = parent(parent(z)) // Großvater
  else
    if z=right(parent(z)) then // Fall 2 L
      z = parent(z)
      LEFT-ROTATE(rt,z)
    end if
    color(parent(z)) = BLACK
    color(parent(parent(z)) = RED
    rt = RIGHT-ROTATE(rt, parent(parent(z))
  end if
```

Rot-Schwarz-Bäume *(Elternknoten ist linkes Kind)*

```
else   // parent(z) ≠ left(parent(parent(z))
  y = left(parent(parent(z))) // Großvater
  if color(y) = RED then
    color(parent(z)) = BLACK  // Vater
    color(y) = BLACK
    color(parent(parent(z)) = RED  // Großvater
    z = parent(parent(z)) // Großvater
  else
    if z=left(parent(z)) then
      z = parent(z)
      RIGHT-ROTATE(rt,z)
    end if
    color(parent(z)) = BLACK
    color(parent(parent(z)) = RED
    rt = LEFT-ROTATE(rt, parent(parent(z))
  end if
end if
```

Rot-Schwarz-Bäume

```
    end while
    color(rt) = BLACK
    return rt
end function
```

Rot-Schwarz-Bäume

Erläuterung zum Pseudocode:

- Zunächst Einfügen wie in BST, dann Korrektur

Fallunterscheidung:

- Zeilen 11-15: Fall 1 (Onkel ist rot)
- Zeilen 17-19: Fall 2 (Onkel des neuen Knotens ist schwarz und ist rechtes Kind)
- Zeilen 21-23: Fall 3 (Onkel ist schwarz und linkes Kind)

Rot-Schwarz-Bäume

- Beispiel:
  - Einfügen von 11 – 2 – 15 – 7 – 8 – 4 – 3 – 1

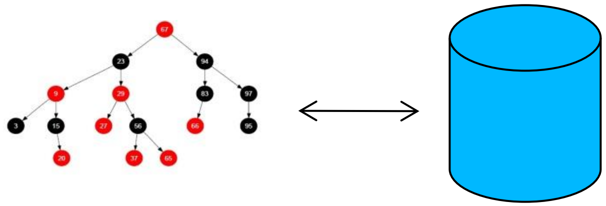
Zusammenfassung Rot-Schwarz-Bäume

- Eigenschaften wie Binäre Suchbäume
- Darüber hinaus Verbesserung
  - Verhinderung einer Entartung
  - Garantierte logarithmische Suchzeit
- Minimaler zusätzlicher Speicherbedarf (1 Bit pro Knoten)
- Maßnahmen zur Ausbalancierung beim Einfügen erforderlich
  - Laufzeit der Maßnahmen zur Ausbalancierung ist O(h)
- Alle Operationen in garantiert O(log n)

Rot-Schwarz-Bäume

Einsatzgebiet z.B.

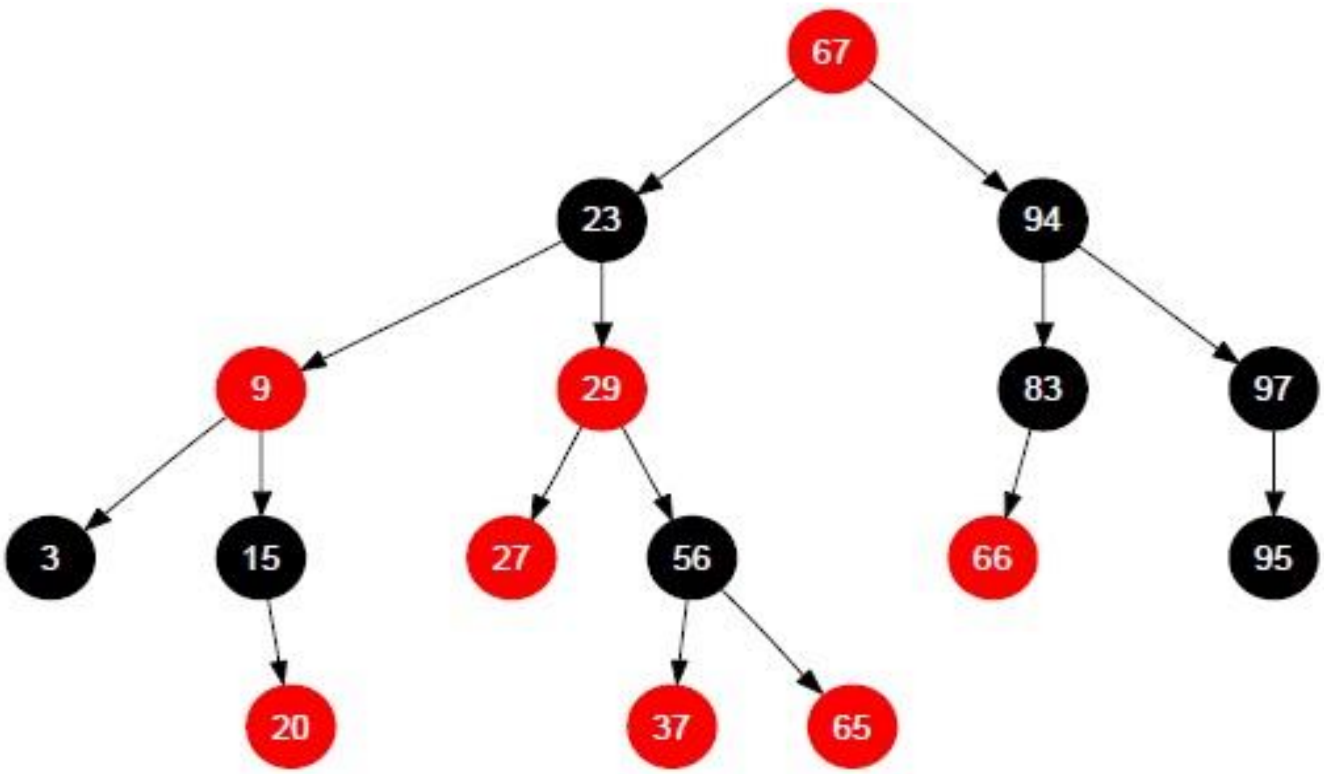
- Datenhaltung der Indexstruktur im Hauptspeicher, während sich die eigentlichen Daten auf einem externen Speicher befinden.
  - Studierendenverwaltung: Matrikelnummer
  - Personalverwaltung: Personalnummer
  - Bestellkatalog: Artikelnummer
  - etc.



Rot-Schwarz-Bäume

Übung

Nennen Sie die Fehler im folgenden RST:



Rot-Schwarz-Bäume

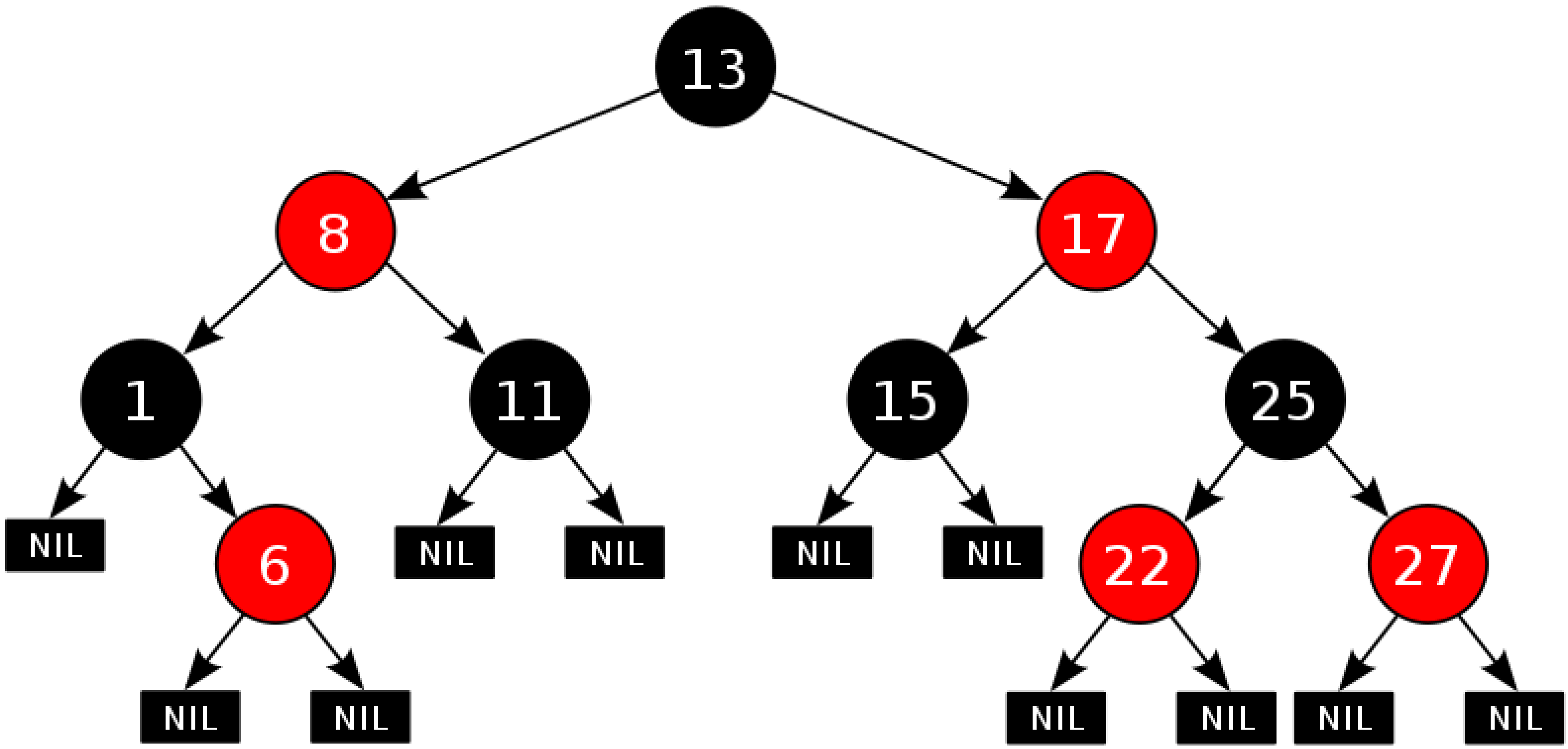
„Live“-Übung

- Fügen Sie die Zahlen von 1 bis 8 (in dieser Reihenfolge) in einen anfangs leeren Rot-Schwarz-Baum ein; skizzieren Sie den Baum nach jedem Einfügevorgang (und ggf. die Zwischenschritte).

Rot-Schwarz-Baum

- Löschen: Reduktion auf Ein-Kind-Knoten (wie bei „normalen“ Binären Suchbäumen)
- Problemfälle:
  - Knoten ist rot
    - kein Problem → Kind rückt auf
  - Knoten ist schwarz, Kind ist rot
    - kein Problem → Kind rückt auf und wird schwarz gefärbt
  - Knoten ist schwarz, Kind ist schwarz
    - → Literatur…

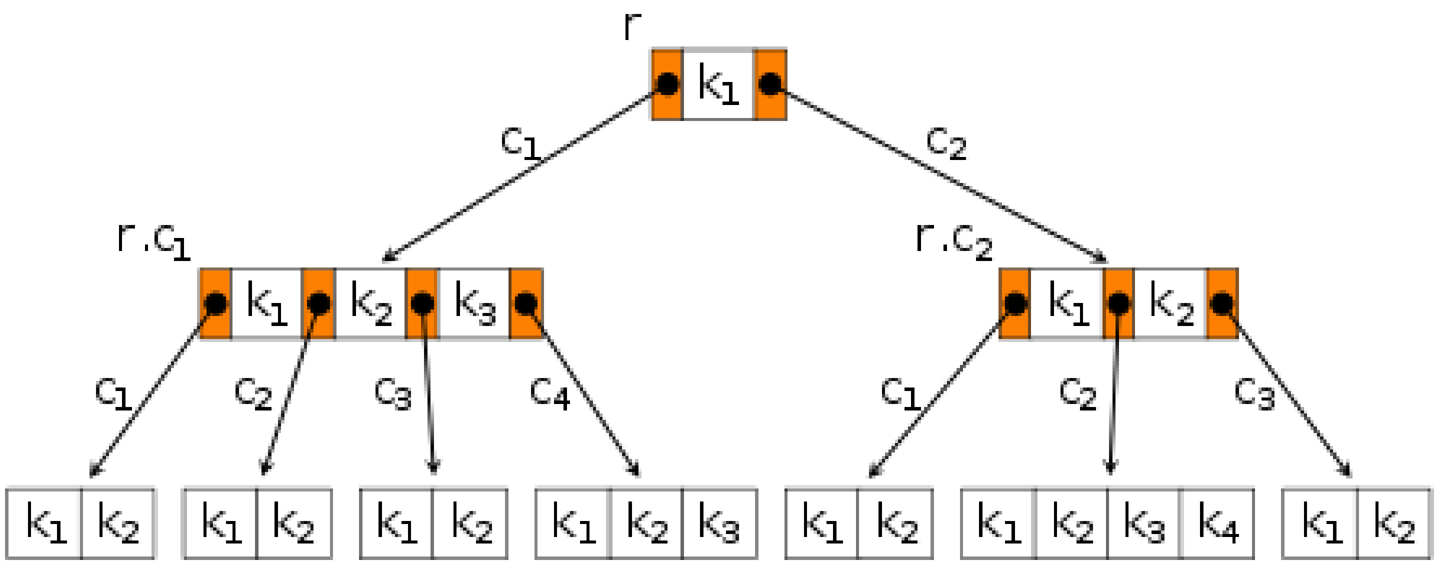
Rot-Schwarz-Baum



B-Baum

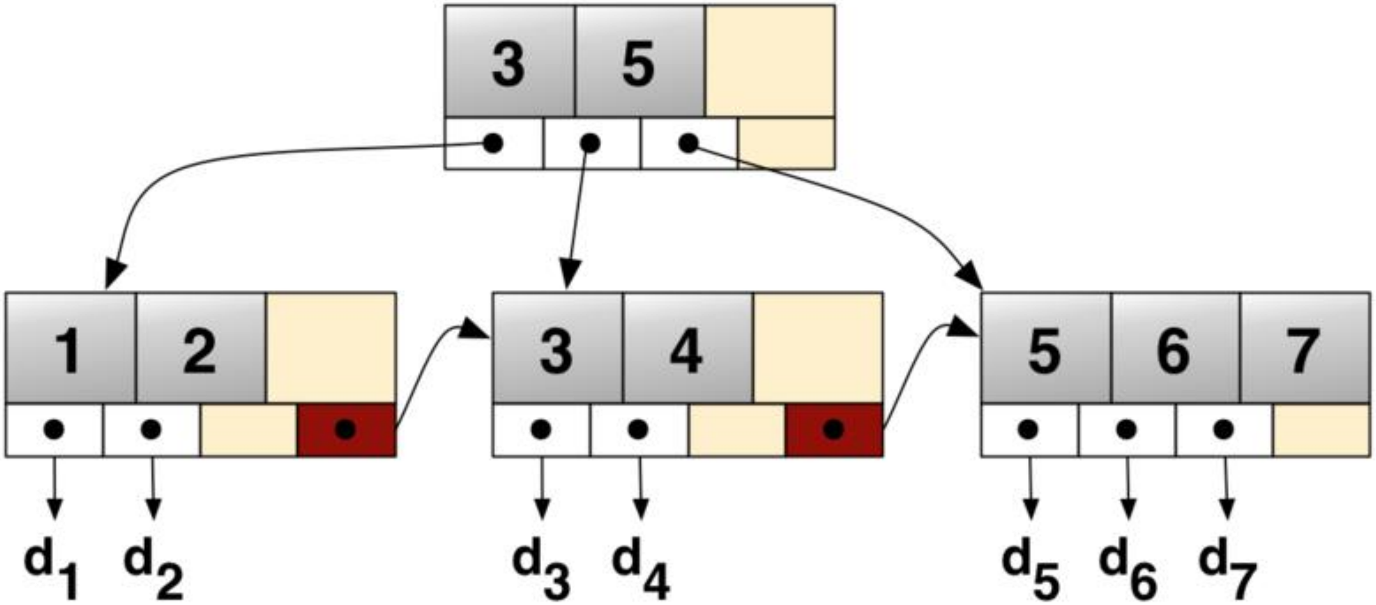
- B-Baum
  - Ähnlich wie Rot-Schwarz-Bäume (ausbalanciert, Aufwand logarithmisch, …)
  - „B“: „balanciert“, „breit“, „buschig“, „Bayer“ (nicht: „binär“!)
  - Knoten enthalten mehrere Schlüssel
  - Mehrere Kind-Knoten
  - Spezialfälle:
    - 2-3-4-Bäume
    - B^+-Bäume (Daten nur in Blättern gespeichert)
    - B^*-Bäume (B^+Bäume, bei denen Kindanzahl gerade sein muss)

B-Baum: Beispiel des Aufbaus



Bildquelle: Wikipdia

B+-Baum: Beispiel des Aufbaus



Bildquelle: Wikipdia

Nachtrag: Löschen

- Löschen:
- Reduktion auf Ein-Kind-Knoten
  - Knoten ist rot

  → Kein Problem: Kind rückt auf
  - Knoten ist schwarz, Kind ist rot

  → Kein Problem: Kind rückt auf und wird schwarz gefärbt
- Knoten ist schwarz, Kind ist schwarz

  → Problem!