Übung Algorithmen I
3.6.15

Christoph Striecks
Christoph.Striecks@kit.edu

(Mit Folien von Julian Arz, Timo Bingmann und Sebastian Schlag.)

Roadmap

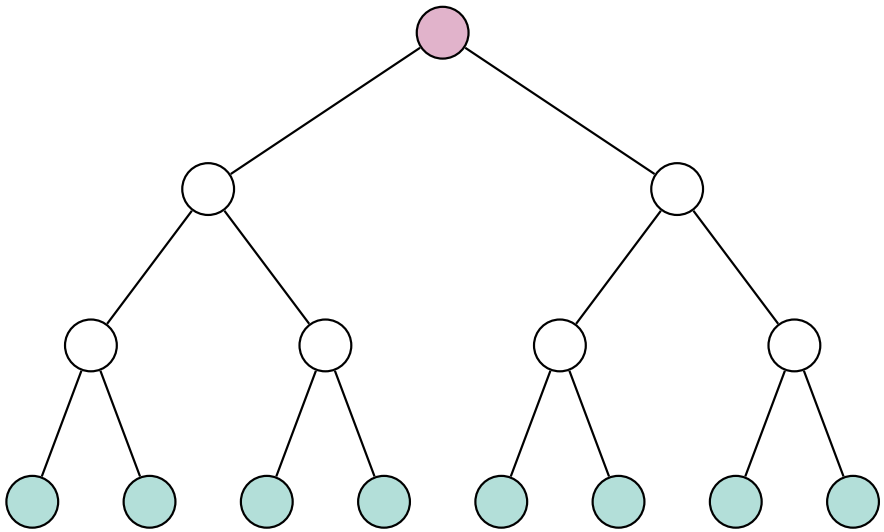
- Hinweise zur Übungsklausur
- (Weitere) Traversierungen von Binärbäumen
- Die Anzahl binärer Suchbäume
- Rot-Schwarz-Bäume

Organisation

- Übungsklausur am Montag, dem 8.6.15, um 15.45, im Audimax
- Freiwillig, keine Prüfungsleistung, *aber* zur Selbstkontrolle
- Deckt den ersten Teil der Vorlesung und Übung (bis 27.5.15) ab
- (Nicht vergessen: Neues Übungsblatt ab heute)

(Weitere) Traversierungen von Binärbäumen

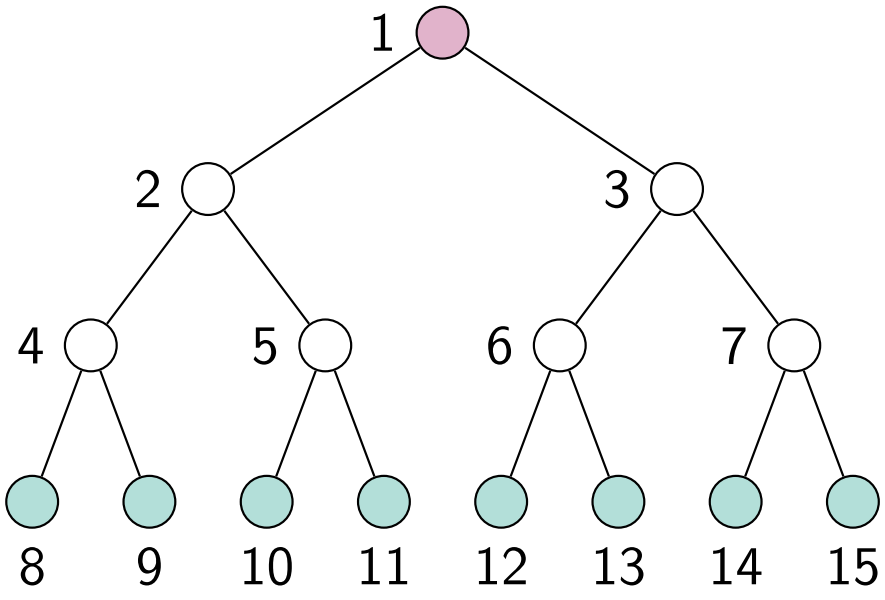
Traversierungen von Binärbäumen



Beispiel:

$n = 15$ Knoten
$b = 8$ Blätter
$c = 7$ innere Knoten
$h = 3$ Höhe

Traversierungen von Binärbäumen

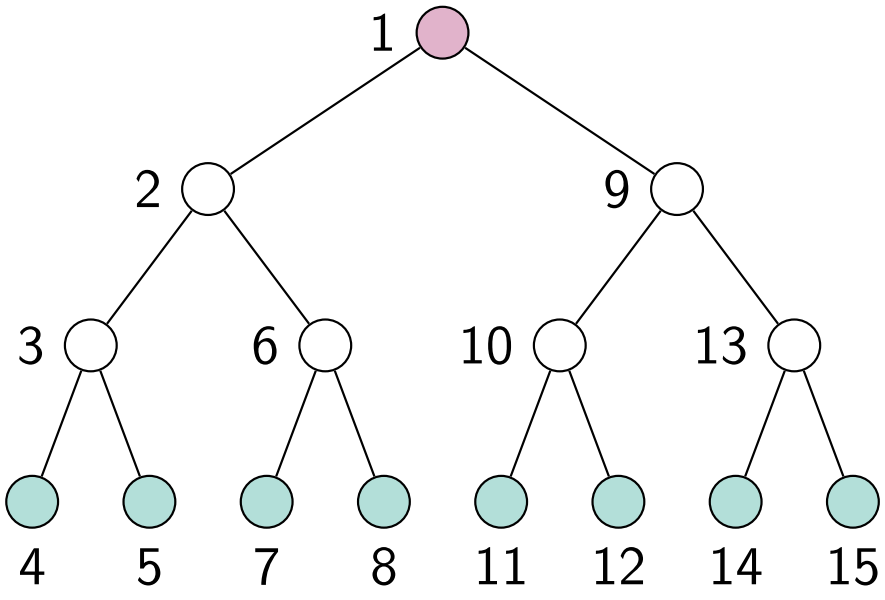


(Level-order-Traversierung.)

Beispiel:

$n = 15$ Knoten
$b = 8$ Blätter
$c = 7$ innere Knoten
$h = 3$ Höhe

Traversierungen von Binärbäumen

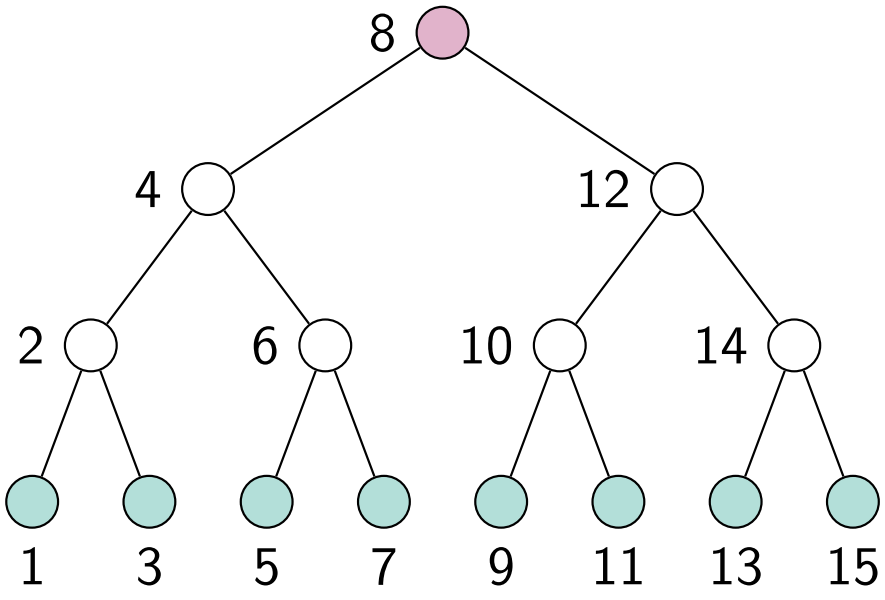


(Pre-order-Traversierung.)

Beispiel:

$n = 15$ Knoten
$b = 8$ Blätter
$c = 7$ innere Knoten
$h = 3$ Höhe

Traversierungen von Binärbäumen

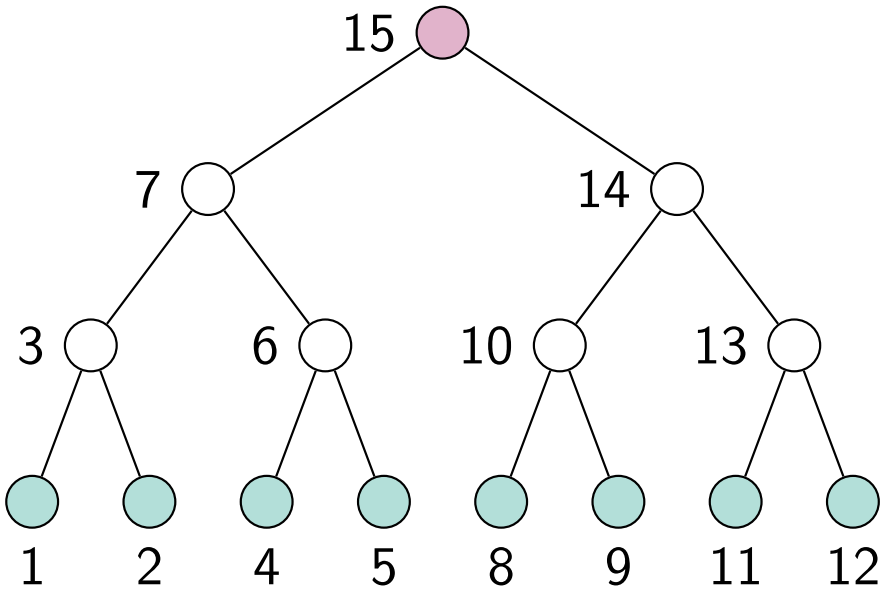


(In-order-Traversierung.)

Beispiel:

$n = 15$ Knoten
$b = 8$ Blätter
$c = 7$ innere Knoten
$h = 3$ Höhe

Traversierungen von Binärbäumen



(Post-order-Traversierung.)

Beispiel:

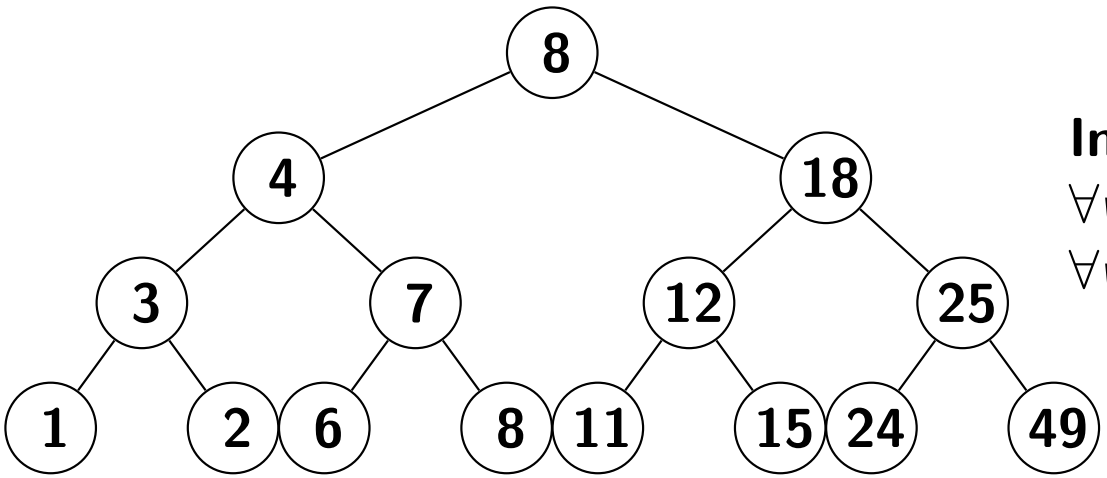
$n = 15$ Knoten
$b = 8$ Blätter
$c = 7$ innere Knoten
$h = 3$ Höhe

Die Anzahl binärer Suchbäume

Binäre Suchbäume

Wiederholung

- In der Vorlesung: Daten nur an Blättern
- Hier in der Übung: Daten in allen Knoten

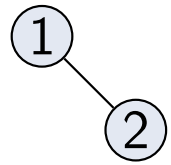


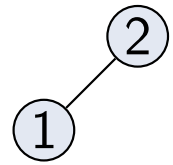
Invariante: Für alle Knoten v:
$\forall w \in \mathit{left}(v) : w \leq v$ und
$\forall w \in \mathit{right}(v) : w > v$

Alle binäre Suchbäume mit $n = 1, 2, 3$:

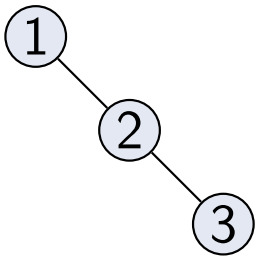
$n = 1$:

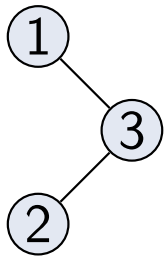
$n = 2$:

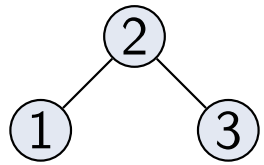


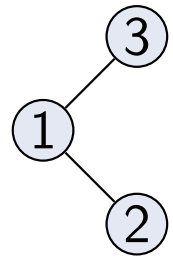


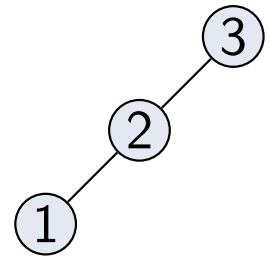
$n = 3$:



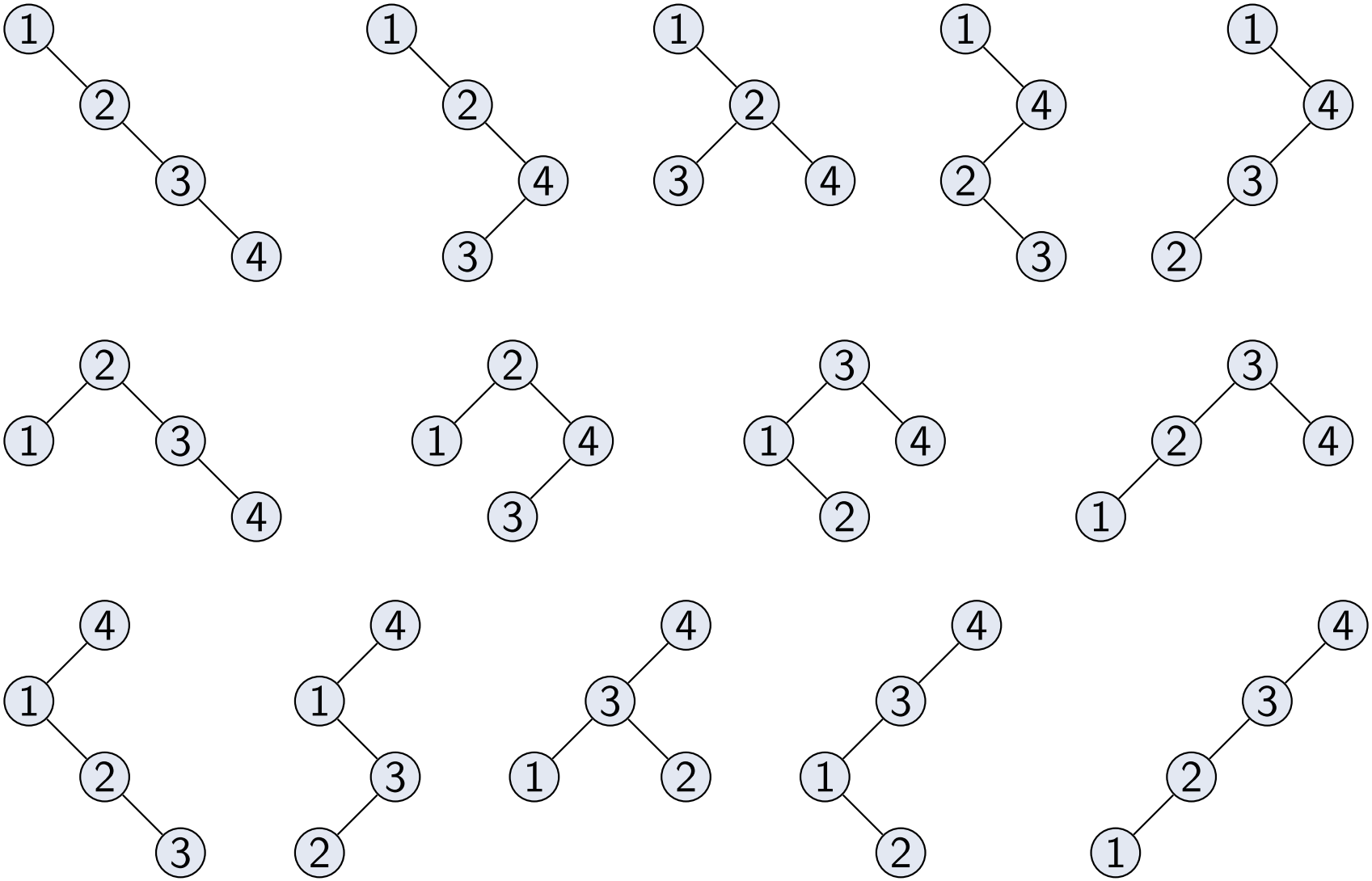




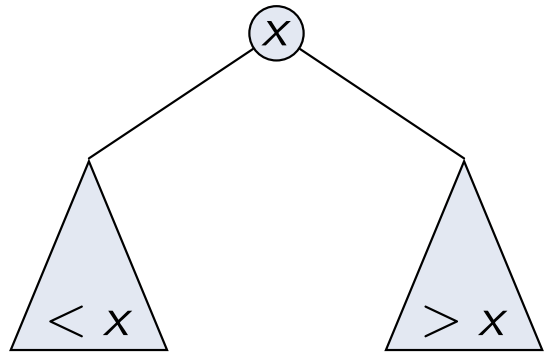




Alle binäre Suchbäume mit $n = 4$:



Die Anzahl binärer Suchbäume C_n



Bekannte Startwerte: $C_0 = 0,\ C_1 = 1,\ C_2 = 2,\ C_3 = 5,\ C_4 = 14.$

Durch Partitionieren der sortieren Folge:

$$C_n = C_0 C_{n-1} + C_1 C_{n-2} + \cdots + C_{n-2} C_1 + C_{n-1} C_0\,, \qquad n > 0.$$

Wende erzeugende Funktionen an:

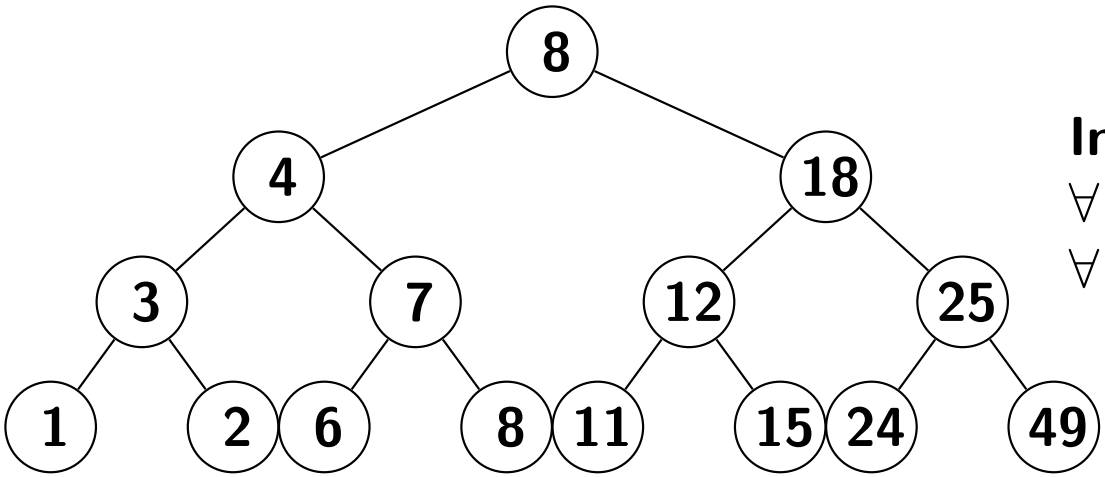
C_n heißen die Catalan-Zahlen.

$$C(z) = \sum_{n=0}^{\infty} C_z z^n = C(z) \cdot zC(z) + 1\,. \quad \Longrightarrow \quad C_n = \frac{1}{n+1}\binom{2n}{n}\,.$$

Balancierte binäre Suchbäume: Rot-Schwarz-Bäume

Binäre Suchbäume

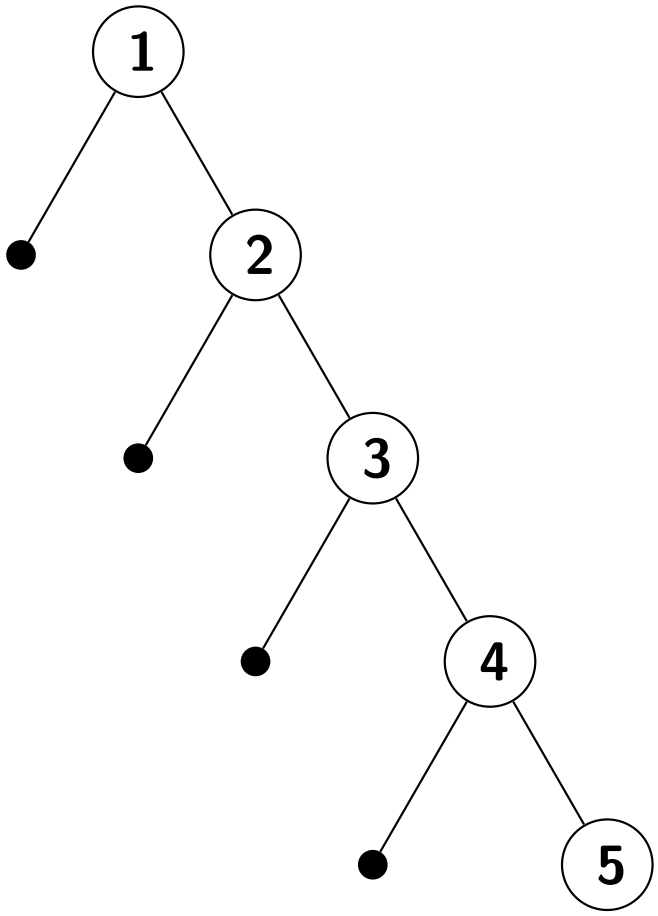
Wiederholung



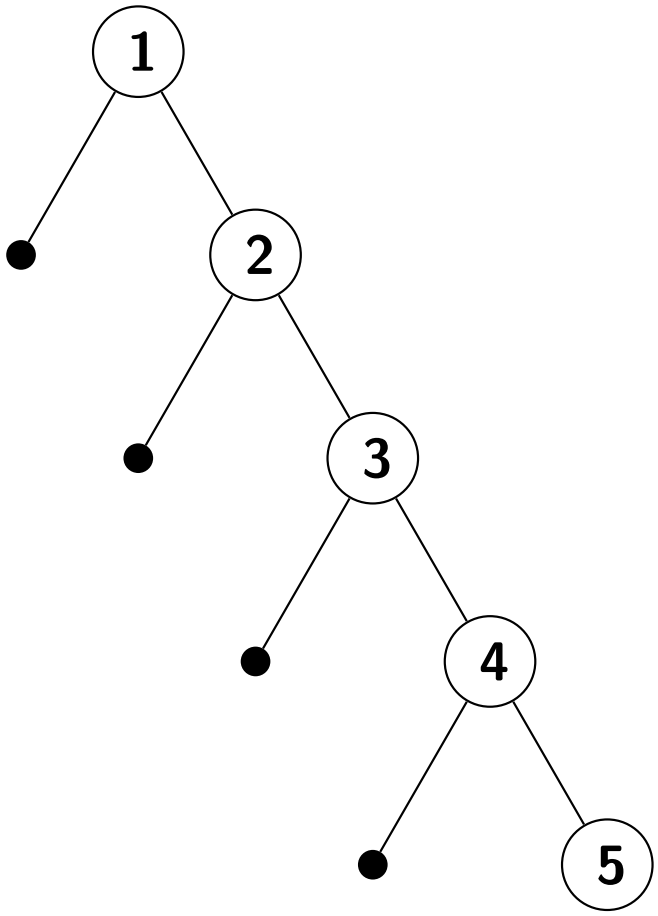
Invariante: Für alle Knoten v:
$\forall w \in \mathit{left}(v) : w \leq v$ und
$\forall w \in \mathit{right}(v) : w > v$

- Problem von binären Suchbäumen: können unbalanciert sein
- Lösung der Vorlesung: (a, b)-Bäume
- Weitere Lösung hier in der Übung: Rot-Schwarz-Bäume

Warum balancierte Bäume?



Warum balancierte Bäume?



- Höhe: $h \in O(n)$
- Suchen: $O(n)$ im Worst Case
- Löschen: $O(n)$ im Worst Case
- Einfügen: $O(n)$ im Worst Case
- Geht das besser?

Ja, Rot-Schwarz-Bäume garantieren Höhe von $O(\log n)$.

Rot-Schwarz-Bäume

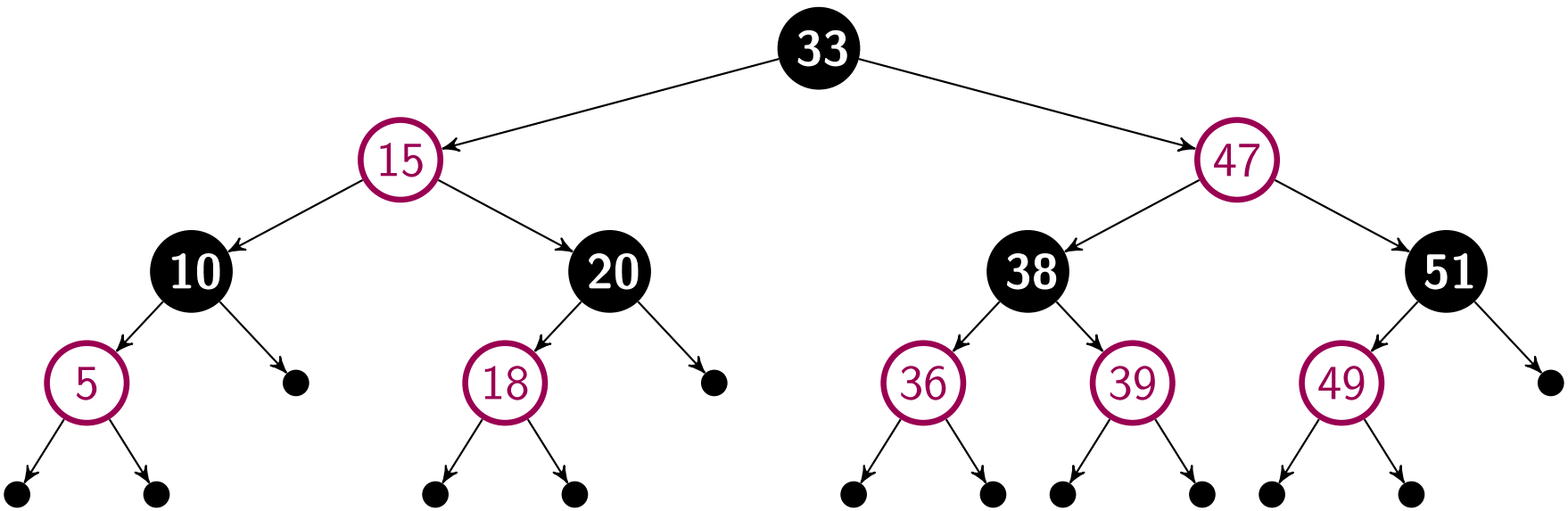
Rot-Schwarz-Bäume sind binäre Suchbäume mit folgenden Eigenschaften:

- Jeder Knoten ist entweder rot oder schwarz. (1 Bit mehr an Information pro Knoten.)
- Die Wurzel ist schwarz.
- Jedes Blatt ist schwarz und enthält keine Werte. (Nur innere Knoten enthalten Werte.)
- Ist ein Knoten rot, so sind beide Kinder schwarz.
- Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Anzahl schwarzer Knoten.

Rot-Schwarz-Bäume

Rot-Schwarz-Bäume sind binäre Suchbäume mit folgenden Eigenschaften:

- Jeder Knoten ist entweder rot oder schwarz. (1 Bit mehr an Information pro Knoten.)
- Die Wurzel ist schwarz.
- Jedes Blatt ist schwarz und enthält "keine Werte". (Nur innere Knoten enthalten Werte.)
- Ist ein Knoten rot, so sind beide Kinder schwarz.
- Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Anzahl schwarzer Knoten.



Höhe von Rot-Schwarz-Bäumen

Satz: Ein Rot-Schwarz-Baum mit n inneren Knoten hat die Höhe $h \leq 2\log_2(n+1)$.
Beweis.

- Betrachte Teilbaum t_x an einem Knoten x
- Sei $sh(x)$ die Anzahl der schwarzen Knoten auf dem Pfad von x zu einem Blatt ohne x selbst (*Schwarzhöhe*)
- **Lemma**: t_x hat mindestens $2^{sh(x)} - 1$ innere Knoten.
- **Beweis.** Induktion über Höhe h von t_x
- **IA**: $h = 0 \Longrightarrow sh(x) = 0$ und $2^0 - 1 = 0$ $\surd$
- **IS**: Aussage gilt für Kinder x_1, x_2 von x mit $h(x) > 0$
- Wir bemerken: $sh(x_i) \geq sh(x) - 1 \Longrightarrow t_{x_i}$ hat mind. $2^{sh(x)-1} - 1$ innere Knoten
- Also hat t_x mindestens $2(2^{sh(x)-1} - 1) + 1 = 2^{sh(x)} - 1$ innere Knoten

Höhe von Rot-Schwarz-Bäumen

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

- Weiter im Beweis ...
- **Zur Erinnerung**:

  4. Ist ein Knoten rot, so sind beide Kinder schwarz.

Lemma. t_x hat mindestens $2^{sh(x)} - 1$ innere Knoten.

- Nach Eigenschaft 4 eines Rot-Schwarz-Baumes sind auf allen Pfaden unterhalb der Wurzel mindestens die Hälfte der Knoten schwarz
- Einsetzen in vorheriges Ergebnis ergibt $n \geq 2^{h/2} - 1$ und damit

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

□

Höhe von Rot-Schwarz-Bäumen

Anders ausgedrückt: Die Höhe eines Rot-Schwarz-Baums ist logarithmisch in der Anzahl der inneren Knoten, also $h \leq 2 \log_2(n+1)$.

- Suchen offensichtlich in $O(\log n)$ möglich.
- Einfügen, Löschen auch in $O(\log n)$, wenn Aufwand nur von der Höhe des Baums abhängt.

Einfügen in Rot-Schwarz-Bäumen

Zur Erinnerung:

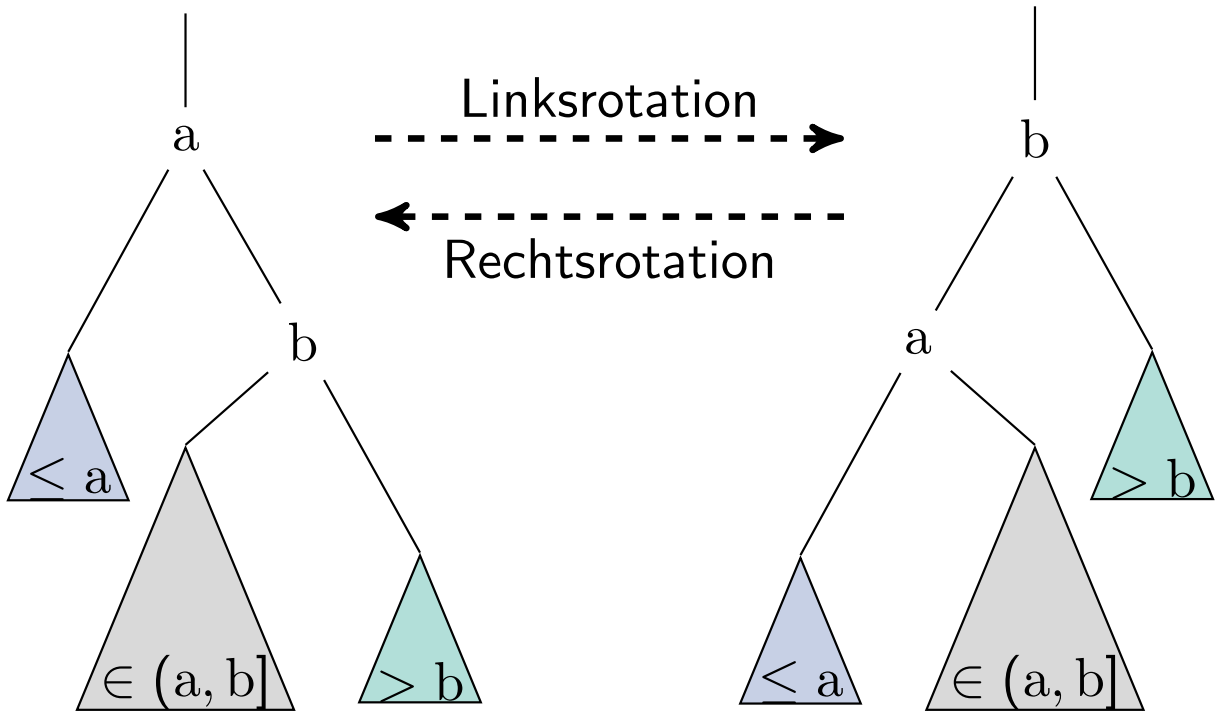
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

Einfügen:

- Einfügen wie in "normalen" binären Suchbäumen
- Einfügen kann Rot-Schwarz-Eigenschaften verletzen.
- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Herstellen der Eigenschaften durch zwei grundlegende Operationen:
  1. Umfärbung von Knoten
  2. Rotation von Teilbäumen

Was ist eine Rotation?

- Lokale Operation, die die binäre Suchbaumeigenschaft erhält
- "Umhängen" einer konstanten Zahl an Pointern pro Teilbaum

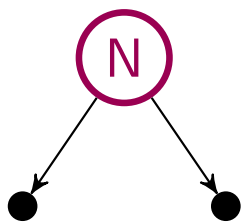


Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

3. Die Wurzel ist schwarz.
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Kein Vorgänger $\longrightarrow$ Neuer Knoten ist Wurzel

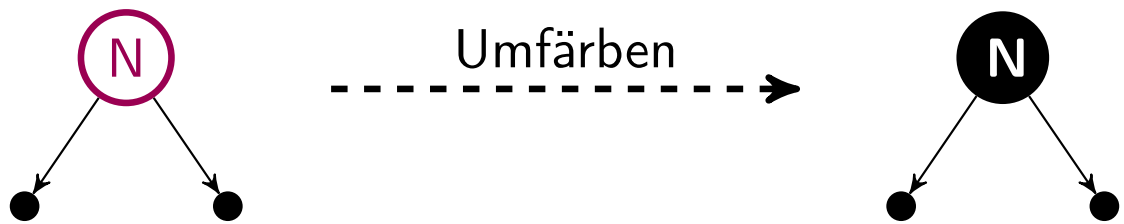


Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

3. Die Wurzel ist schwarz.
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Kein Vorgänger $\longrightarrow$ Neuer Knoten ist Wurzel $\longrightarrow$ Umfärben $\surd$



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

3. Ist ein Knoten rot, so sind beide Kinder schwarz.
4. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

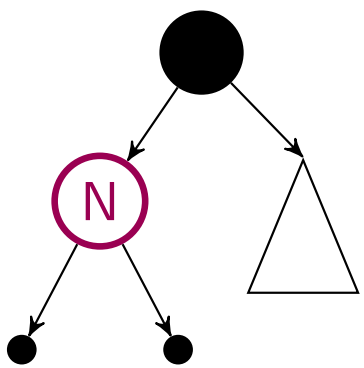
- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Schwarzer Vorgänger

Einfügen in Rot-Schwarz-Bäume

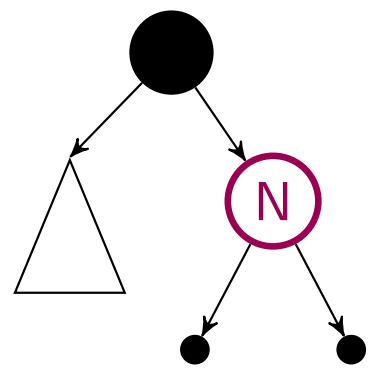
Zur Erinnerung:

3. Ist ein Knoten rot, so sind beide Kinder schwarz.
4. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Schwarzer Vorgänger $\longrightarrow$ Nichts zu tun $\surd$



oder

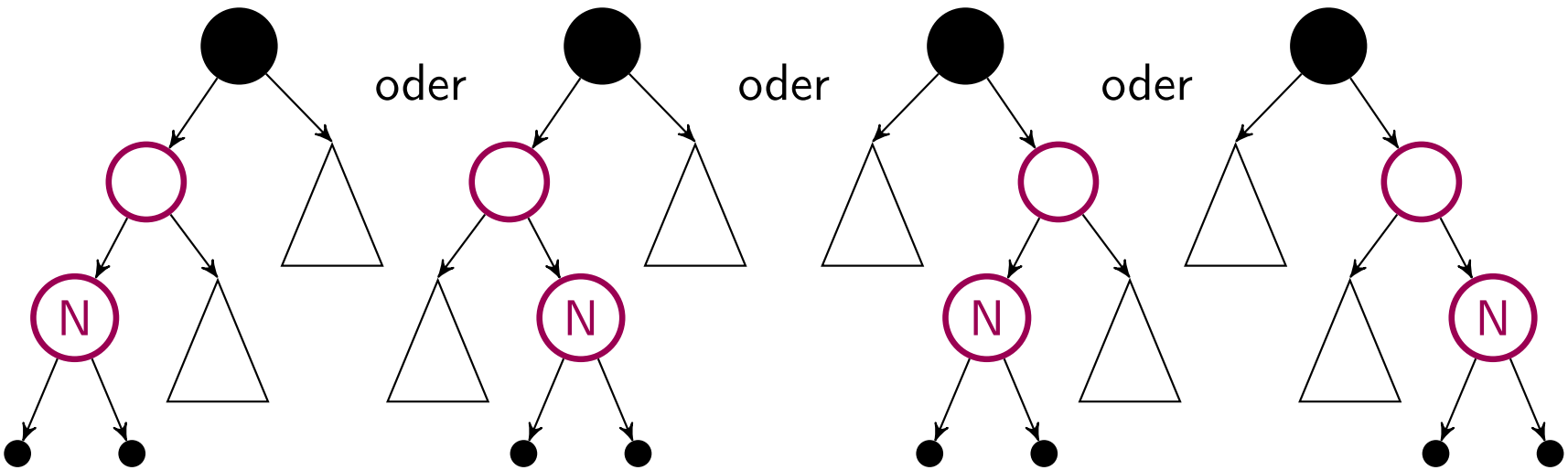


Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

3. Ist ein Knoten rot, so sind beide Kinder schwarz.
4. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Roter Vorgänger

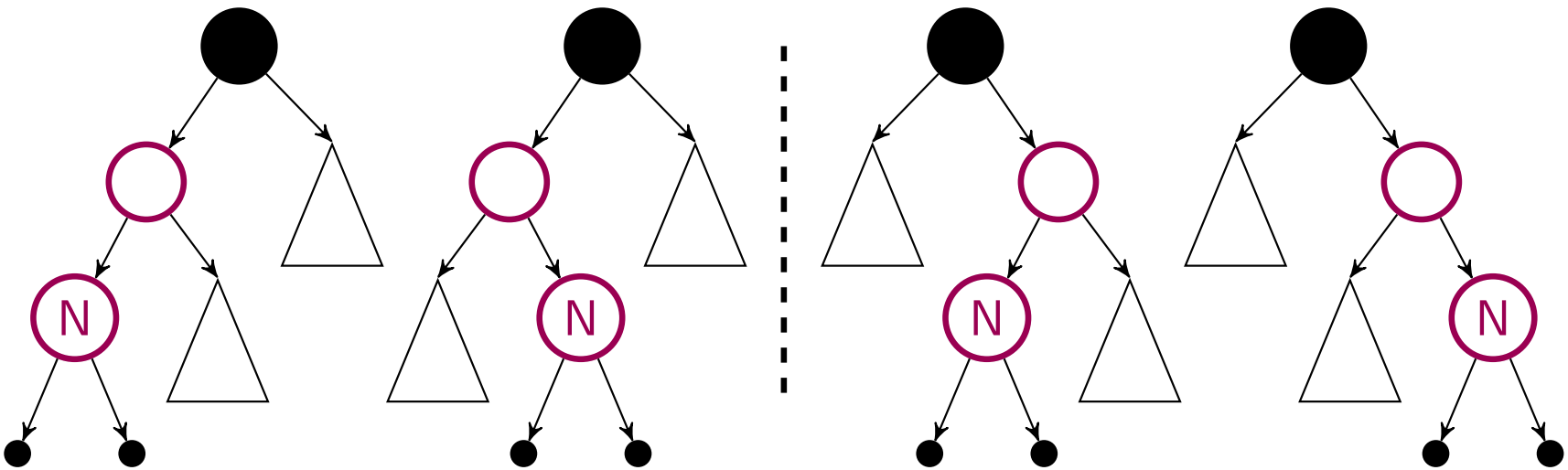


Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

3. Ist ein Knoten rot, so sind beide Kinder schwarz.
4. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Roter Vorgänger $\longrightarrow$ 4 mögliche Strukturen, aber: durch Spiegelung und max. eine Rotation identisch

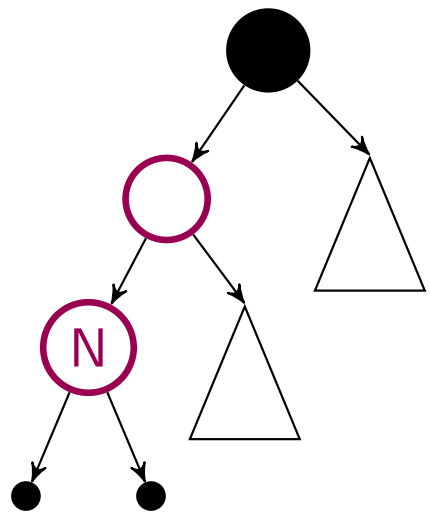


Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

3. Ist ein Knoten rot, so sind beide Kinder schwarz.
4. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.

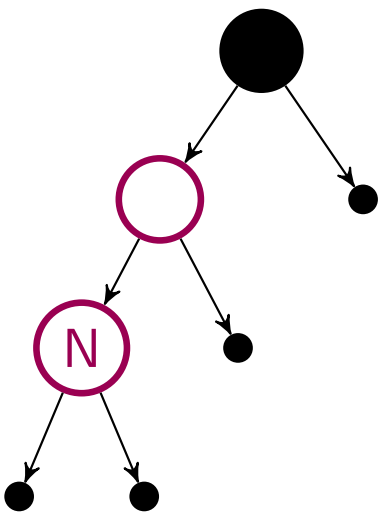
- **Vorgehen**: Roten Knoten in binären Suchbaum einfügen und danach Eigenschaften wieder herstellen
- Roter Vorgänger $\longrightarrow$ 4 mögliche Strukturen, aber: durch Spiegelung und max. eine Rotation identisch



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

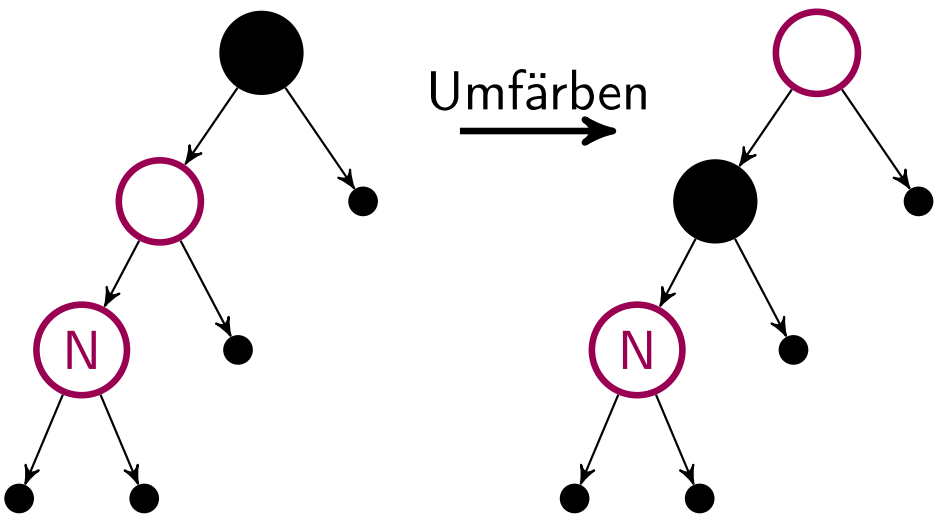
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

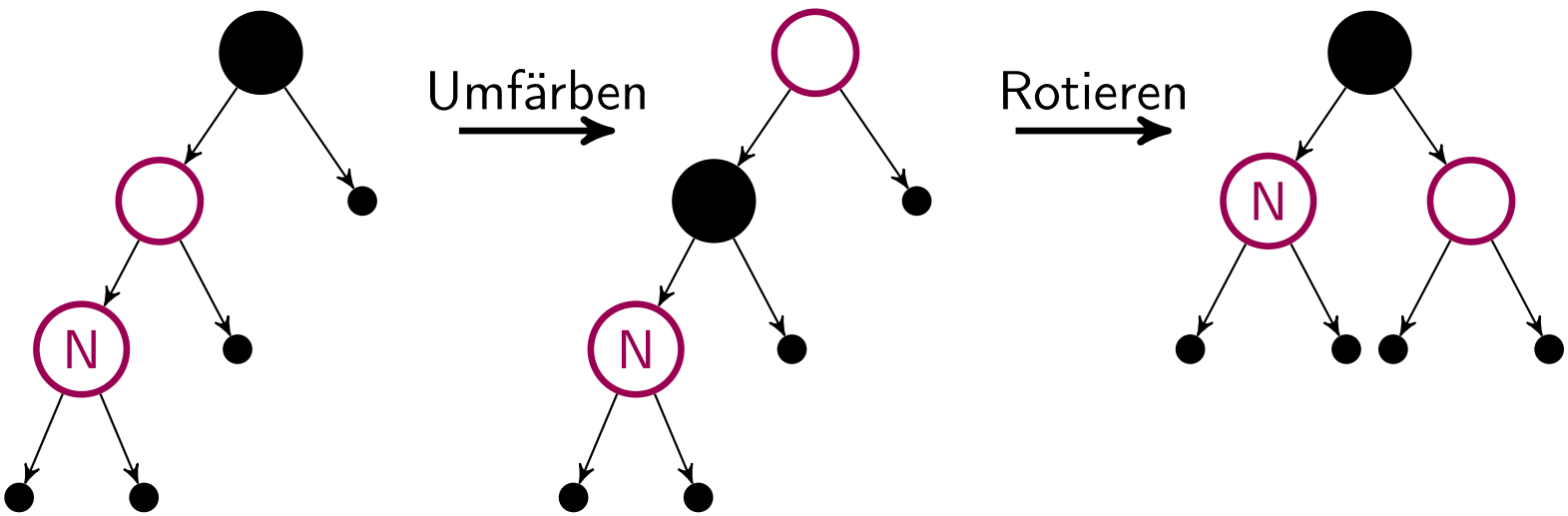
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

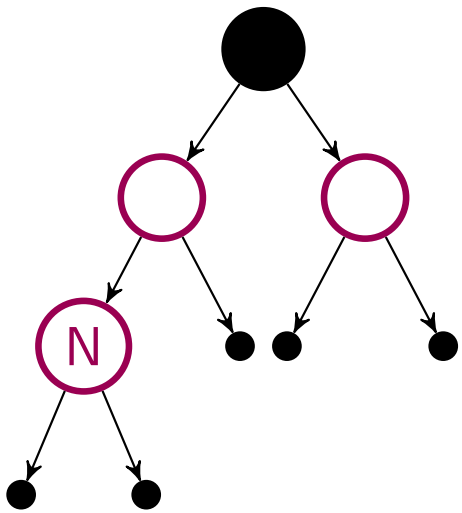
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

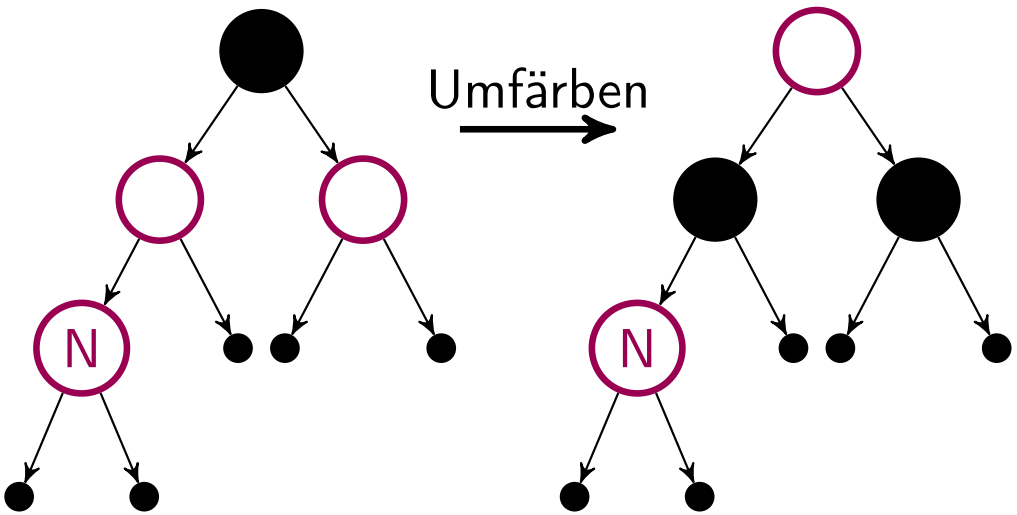
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

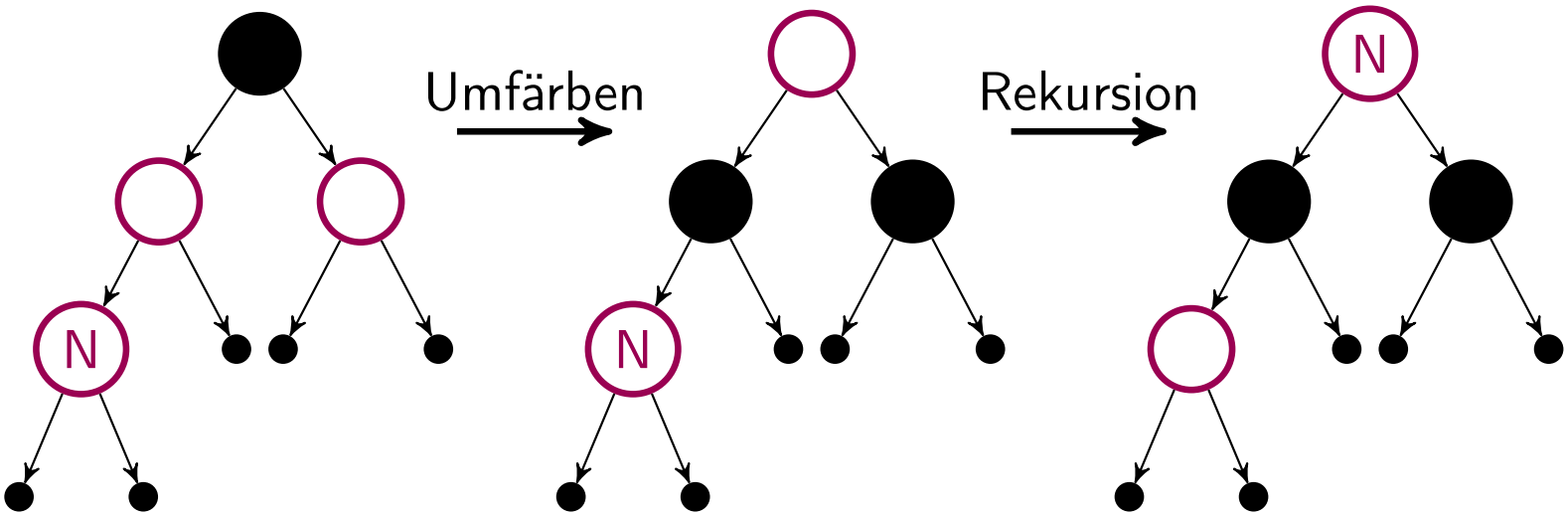
4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.



Einfügen in Rot-Schwarz-Bäume

Zur Erinnerung:

4. Ist ein Knoten rot, so sind beide Kinder schwarz.
5. Für alle Knoten v: Alle Pfade von v zu einem Blatt enthalten die gleiche Zahl schwarzer Knoten.



Rot-Schwarz-Bäume

Aufwandsabschätzung

Speicher:

- 1 Bit pro Knoten für die Farbe
- Parent-Pointer ist nötig

Zeit:

- Konstanter Aufwand pro Knoten (Färben und Rotieren)
- Maximal $O(\log n)$ Rekursionstiefe

Ziel erreicht: $O(n)$ Platz und alle Operationen in $O(\log n)$.

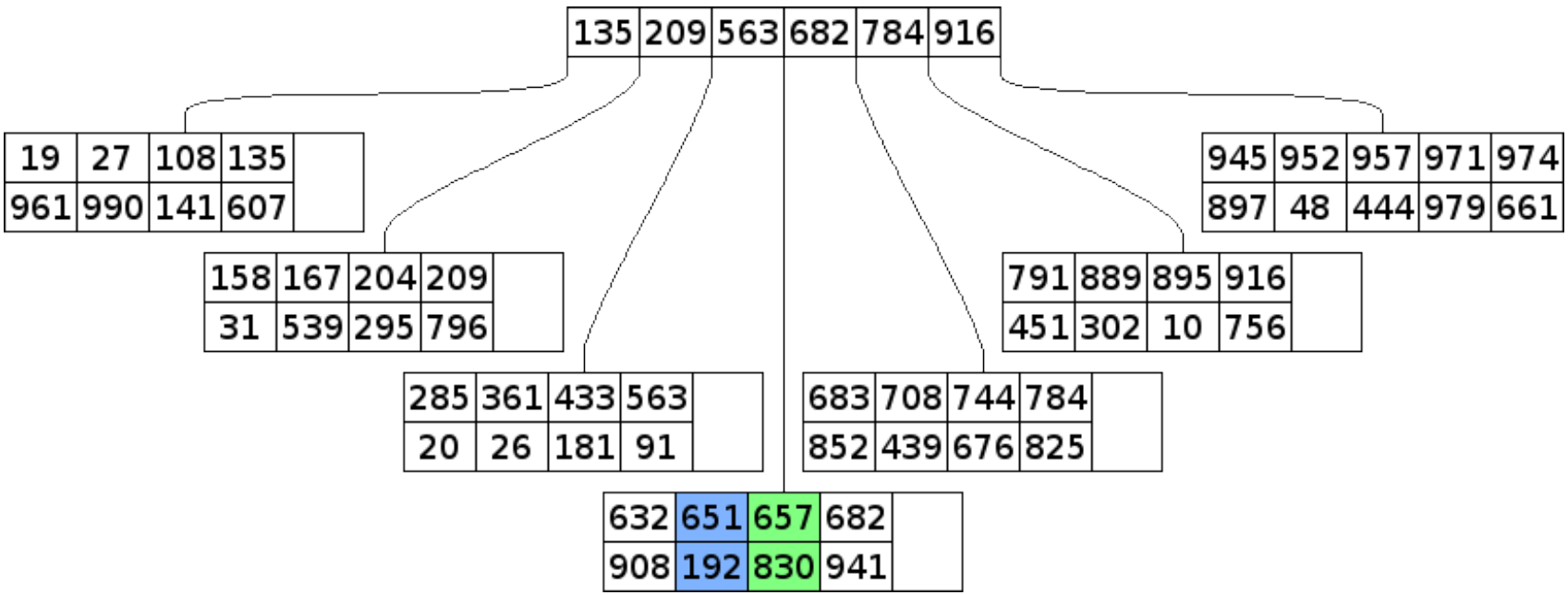
Datenstrukturen in der Wirklichkeit

Datenstrukturen in der Wirklichkeit

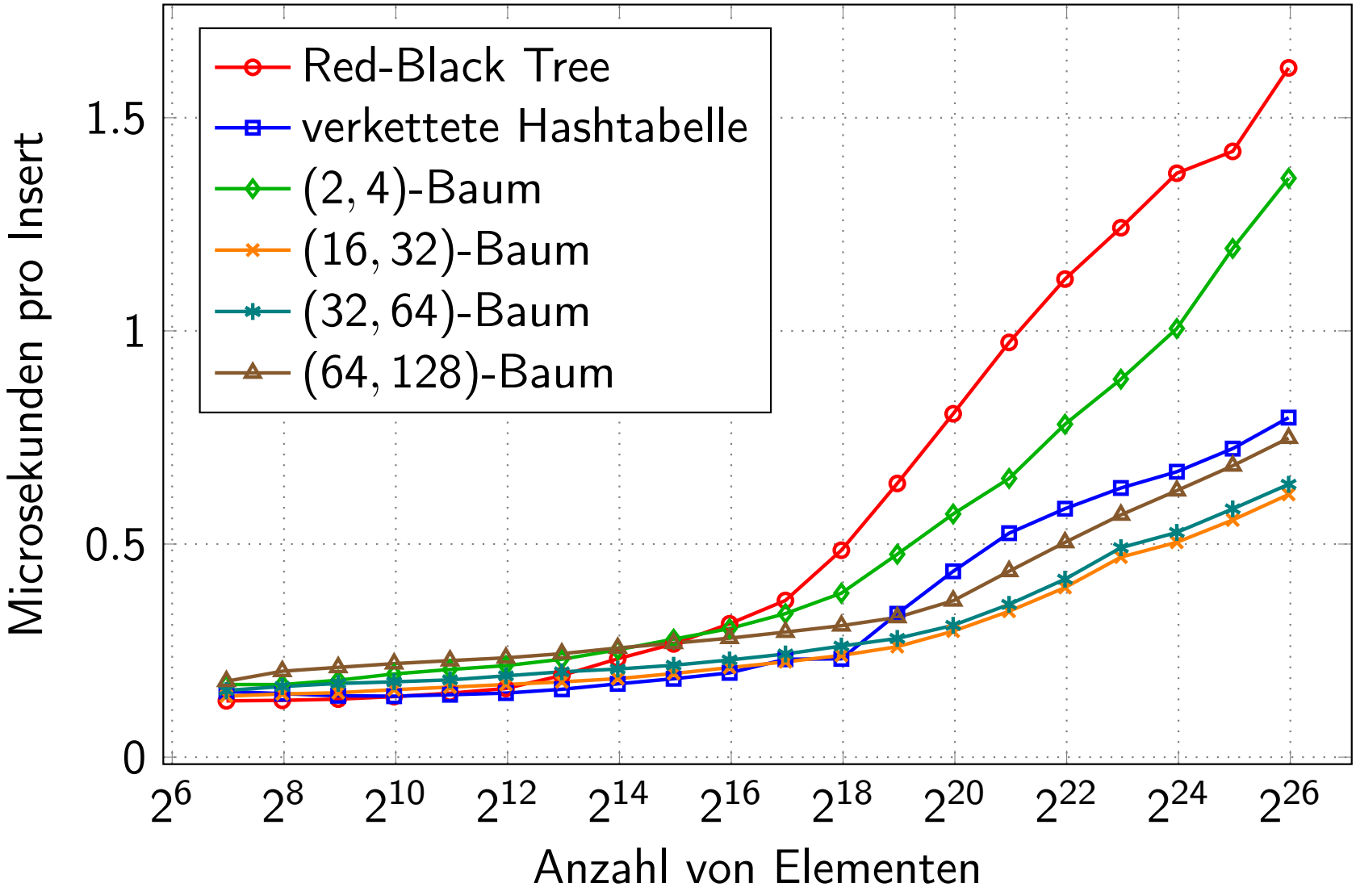
Java Collections	C++ STL
`java.util.LinkedList<T>`	`std::list<T>`
`java.util.ArrayList<T>`	`std::vector<T>`
`java.util.ArrayDeque<T>`	`std::deque<T>`
`java.util.PriorityQueue<T,C>`	`std::priority_queue<T,C>`
`java.util.TreeMap<K,V>`	`std::map<K,V>`
`java.util.TreeSet<K>`	`std::set<K>`
`java.util.HashTable<K,V>`	
`java.util.HashMap<K,V>`	`std::unordered_map<K,V>`
`java.util.HashSet<K>`	`std::unordered_set<K>`

Notizen zu (a, b)-Bäumen

B-Bäume sind fast $(\frac{m}{2}, m)$-Bäume
mit $m = B$ ein Festplatten-Block.



Insert-Geschwindigkeit (Integer, C++)



Locate-Geschwindigkeit (Integer, C++)