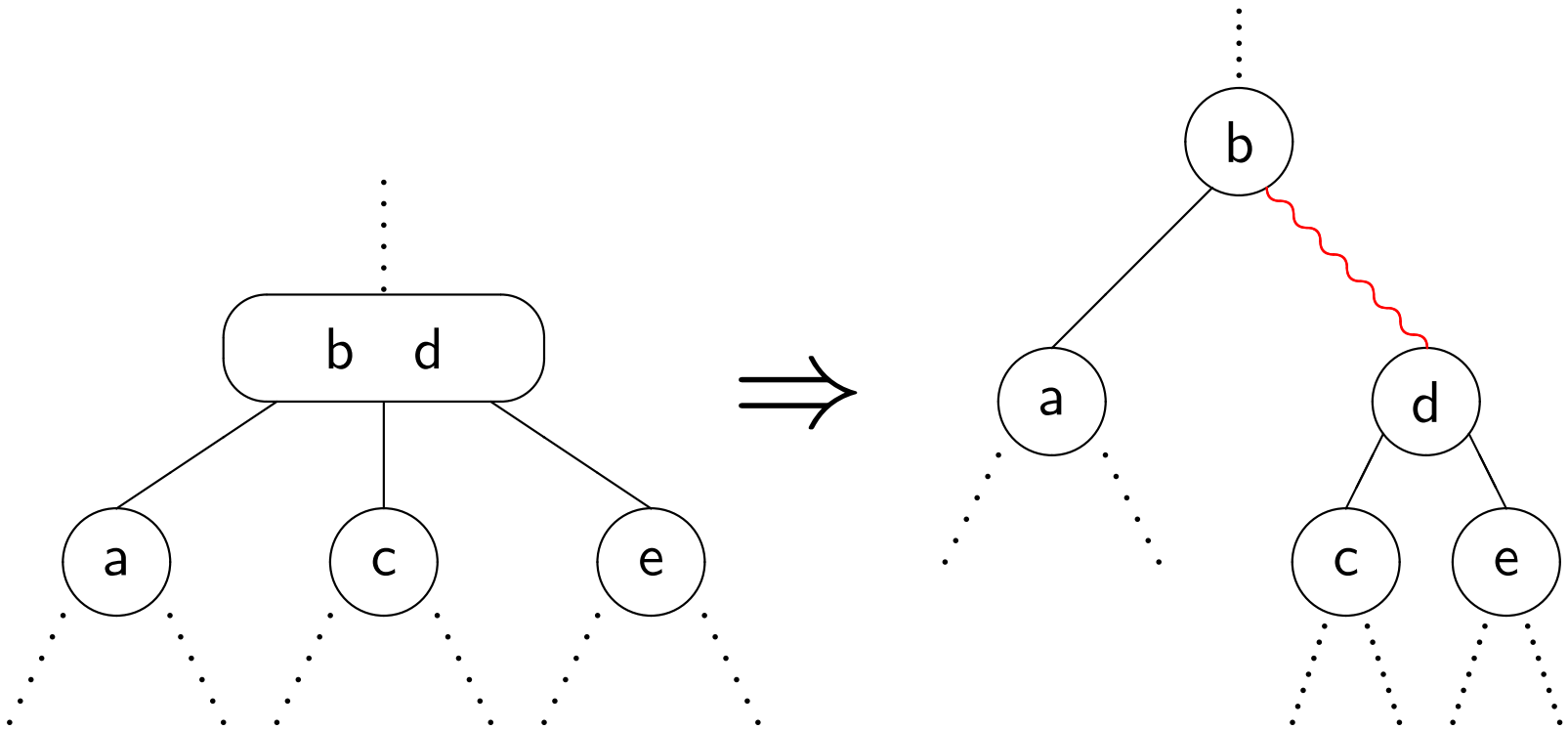
2.1.2 Rot-Schwarz-Bäume

Definition 14
Rot-Schwarz-Bäume sind externe Binärbäume (jeder Knoten hat 0 oder 2 Kinder) mit roten und schwarzen Kanten, so dass gilt:

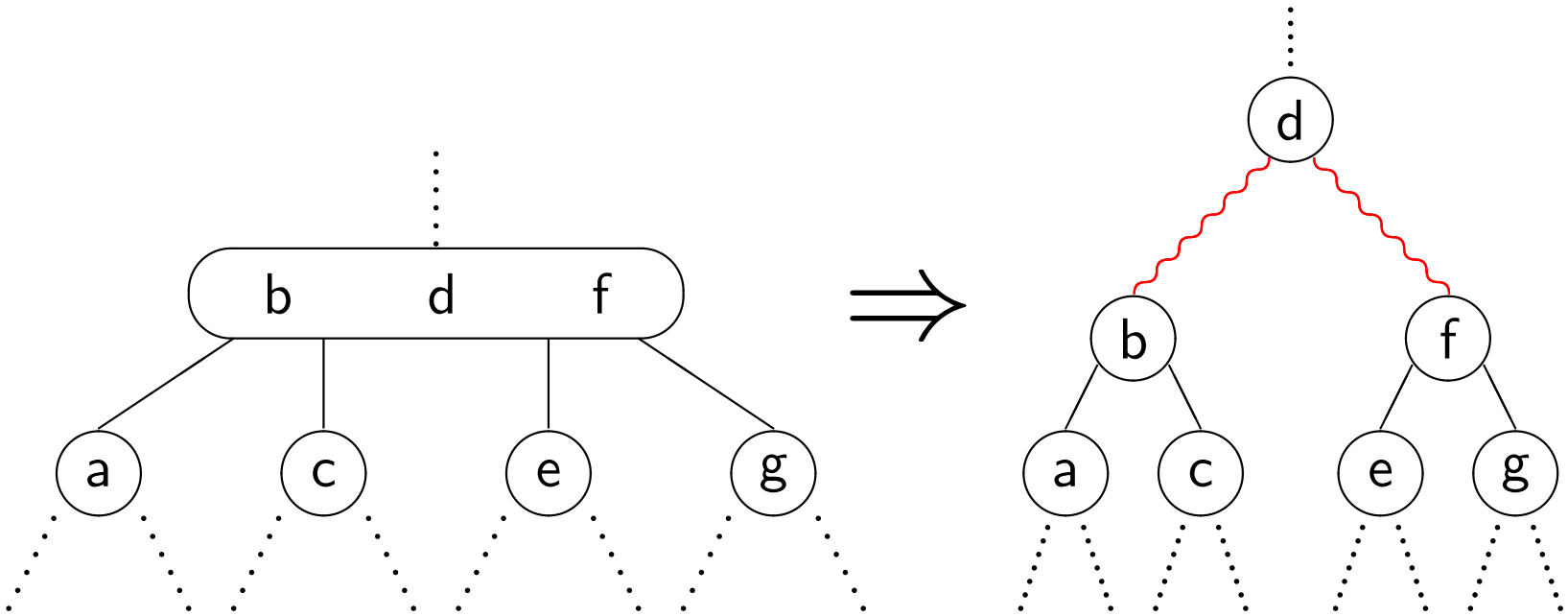
1. alle Blätter hängen an schwarzen Kanten (durchgezogene Linien)
2. alle Blätter haben die gleiche „Schwarztiefe“
3. kein Pfad von der Wurzel zu einem Blatt enthält (zwei oder mehr) aufeinanderfolgende rote Kanten (gewellte Linien).

Dabei ist die „Schwarztiefe“ eines Knoten die Anzahl der schwarzen Kanten auf dem Pfad von der Wurzel zu diesem Knoten.

Rot-Schwarz-Bäume können zur Implementierung von $(2,3)$- oder $(2,4)$-Bäumen dienen, mit dem Vorteil, dass alle internen Knoten Verzweigungsgrad $= 2$ haben!



Implementierung eines 4-Knotens:



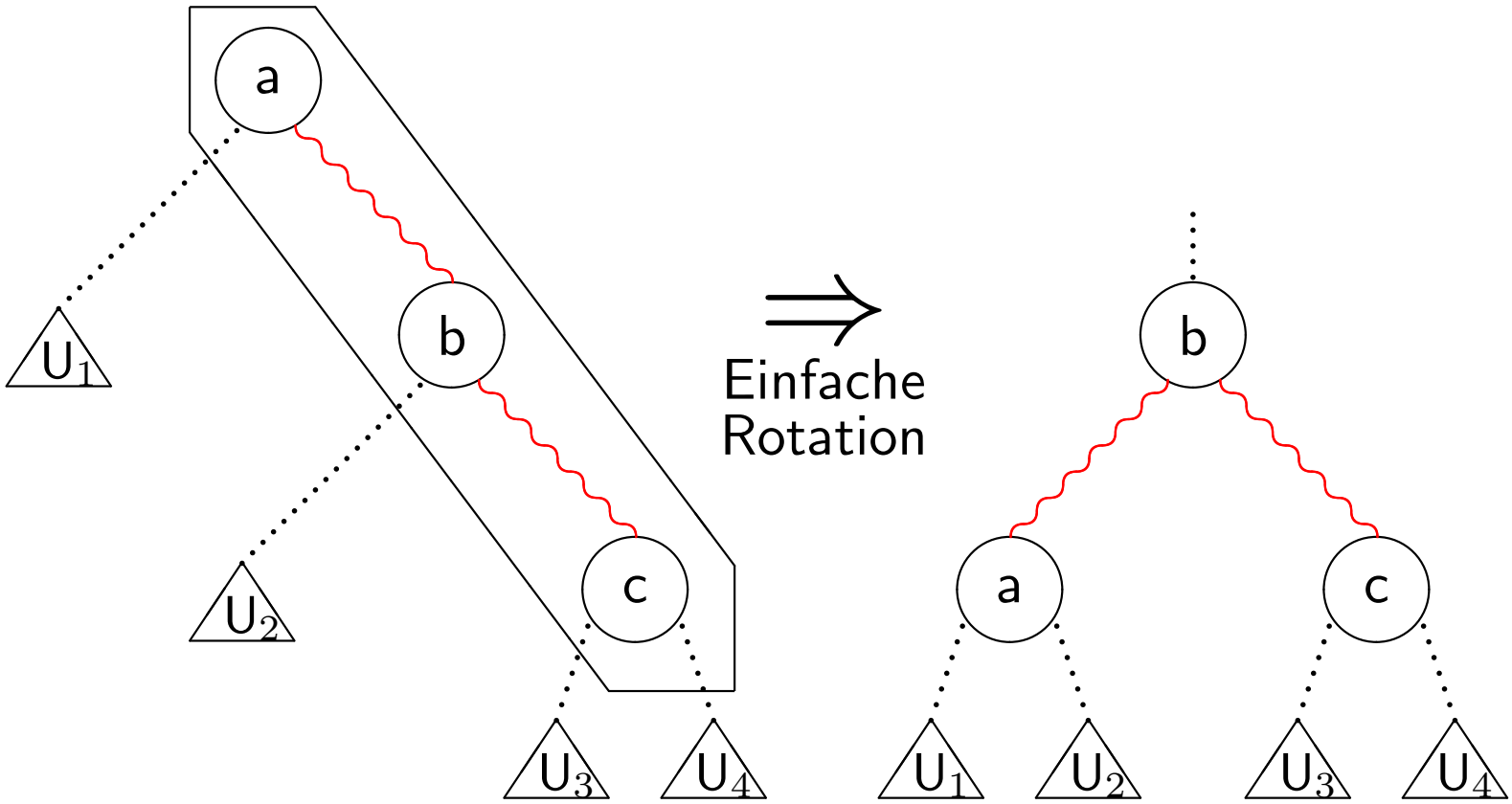
Operationen auf Rot-Schwarz implementierten (a, b)-Bäumen:

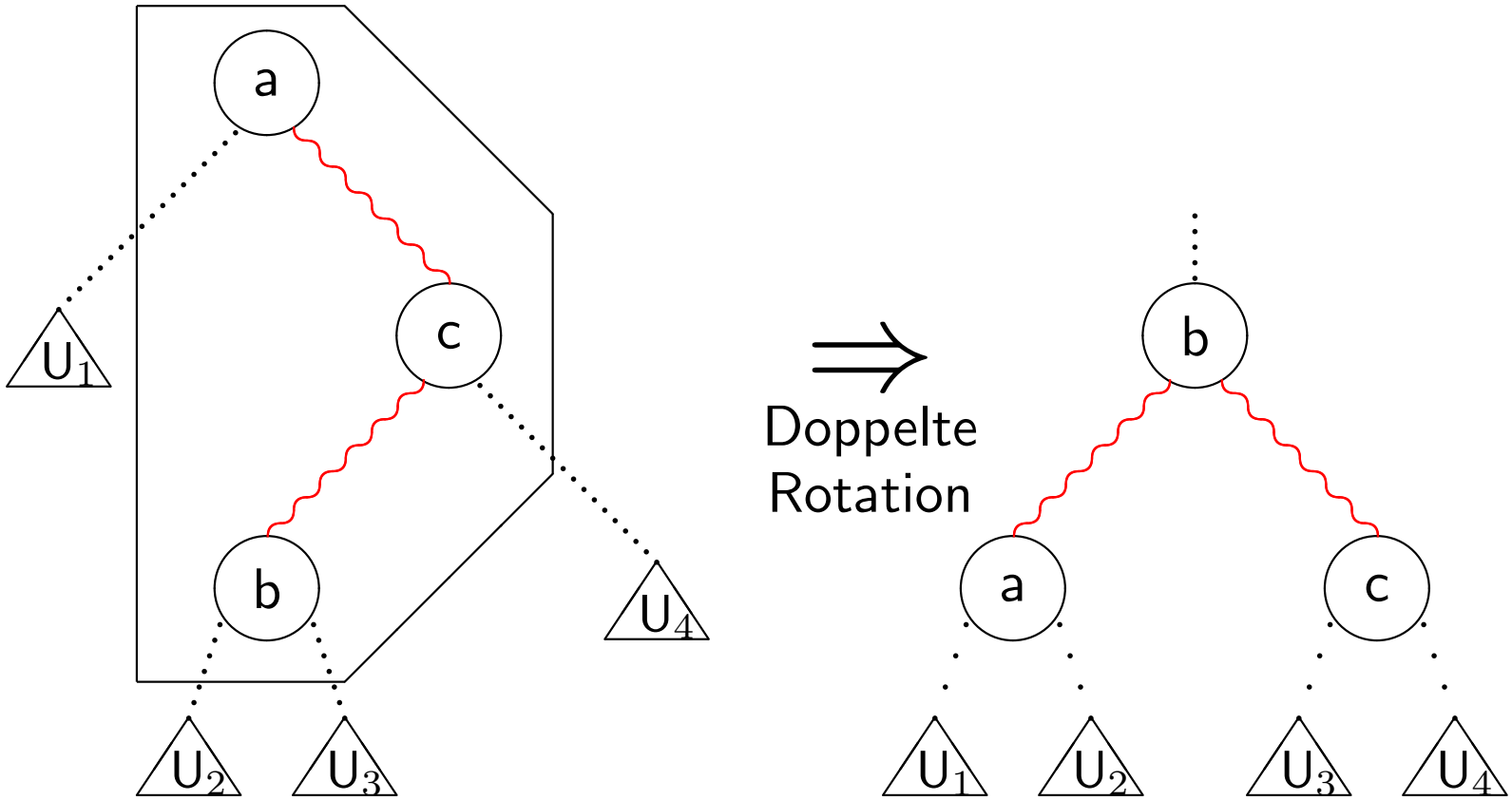
1. *IsElement*(k, T): vgl. (a, b)-Bäume

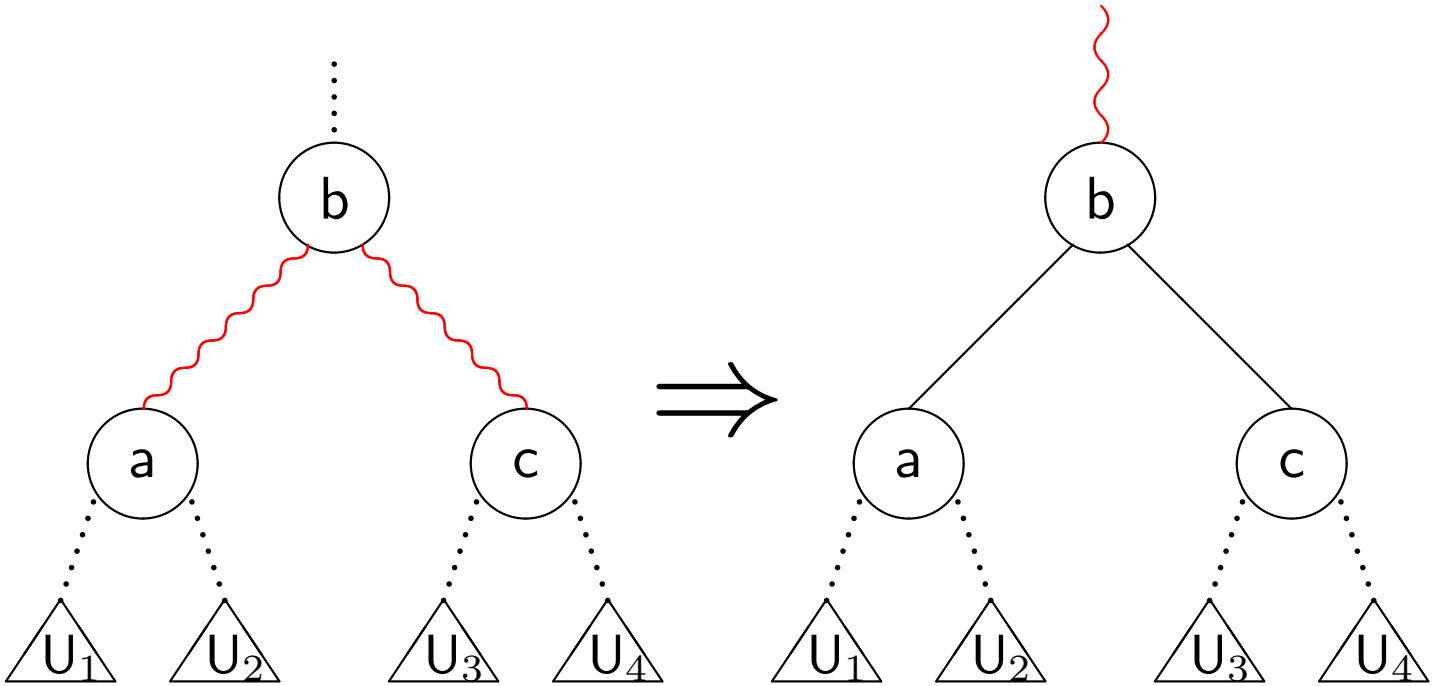
   $$\boxed{\text{Zeit } \mathcal{O}(\log n)}$$

2. *Insert*(k, T): Führe *IsElement*(k, T) aus $\rightsquigarrow$ Blatt w. Dort sei o.B.d.A. nicht das Element v gespeichert. Ersetze w durch neuen internen Knoten w' mit Kindern v, w, wobei v ein neues Blatt mit Schlüssel k ist. w' erhält eine rote Eingangskante.

   Falls nun zwei aufeinanderfolgende rote Kanten an w' vorliegen, führe Rotationen zur Rebalancierung durch.


a
b
c
U_1
U_2
U_3
U_4
Einfache
Rotation
b
a
c
U_1
U_2
U_3
U_4

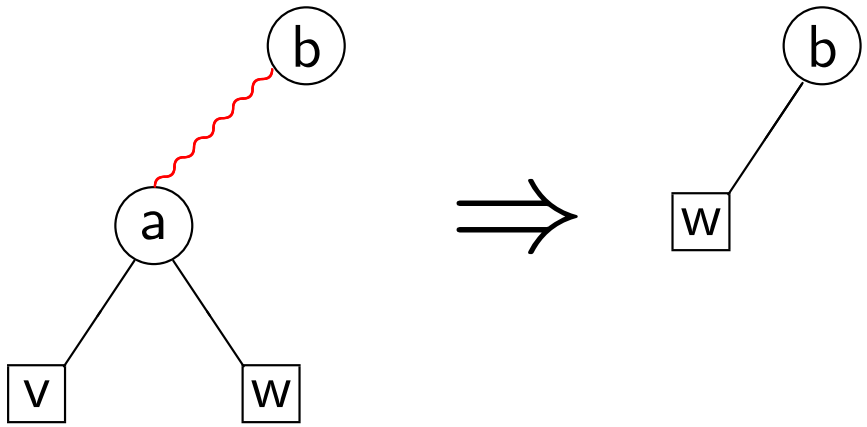

a
c
b
U_1
U_2
U_3
U_4
Doppelte
Rotation
b
a
c
U_1
U_2
U_3
U_4


b
a
c
U1
U2
U3
U4
b
a
c
U1
U2
U3
U4

3. *Delete*(k, T): Das Löschen des Blattes v macht es erforderlich, dass der Vater des Blattes durch den Bruder des Blattes ersetzt wird, damit der Binärbaumstruktur erhalten bleibt. Der Bruder von v ist eindeutig bestimmt.

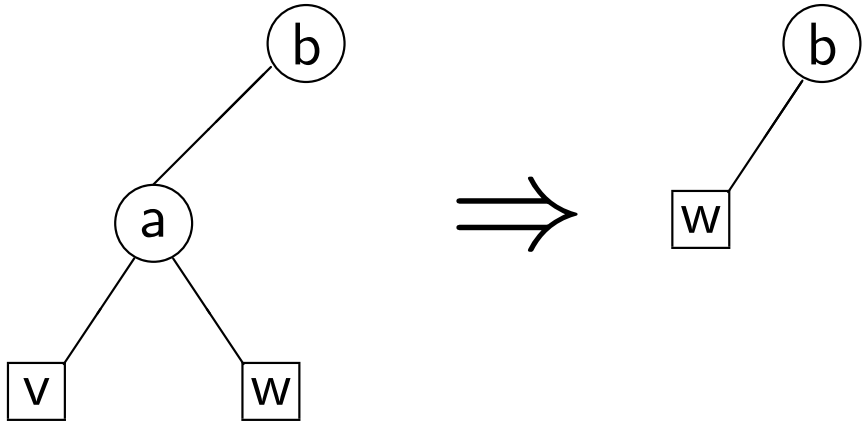
   Dabei treten zwei Fälle auf:

   1. Fall



   Dadurch ändert sich die Schwarztiefe von w nicht, denn die Eingangskante zu w wird (notwendig) zu schwarz umgefärbt. Auch kann es hierdurch nicht zu zwei aufeinanderfolgenden roten Kanten kommen.

2. Fall



Hierdurch verkleinert sich die Schwarztiefe von w um 1. Man kann sie nicht wieder erhöhen, ohne dabei auch die Schwarztiefe anderer Blätter zu vergrößern. Daher muss sie für alle anderen Blätter des Baumes um 1 verkleinert werden. Man betrachte dazu den Pfad von w zur Wurzel als unveränderlich und färbe Kanten so um, dass auf allen anderen Pfaden jeweils genau eine Kante von schwarz zu rot wird. Ferner darf dabei nicht die Situation entstehen, dass zwei aufeinanderfolgende Kanten rot werden. Gegebenenfalls muss gemäß obigem Schema rebalanciert werden.

Satz 15

Die Tiefe eines Rot-Schwarz-Baumes mit n Blättern ist $\mathcal{O}(\log n)$.

Beweis:

Siehe Übungsblatt. □

Korollar 16

Die Wörterbuch-Operationen auf Rot-Schwarz-Bäumen benötigen Zeit $\mathcal{O}(\log n)$.

Beweis:

Siehe Übungsblatt. □

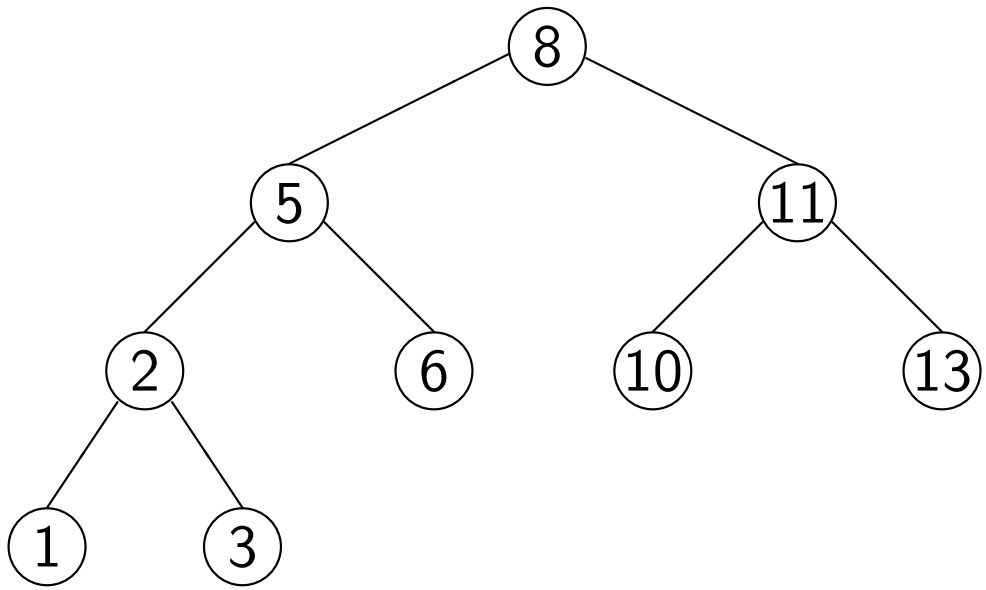
3. Binäre Suchbäume

3.1 Natürliche binäre Suchbäume

Definition 17
Ein natürlicher binärer Suchbaum über einem durch $\leq$ total geordneten Universum U ist ein als interner Suchbaum organisierter Binärbaum (also: Schlüssel an den internen Knoten, Blätter entsprechen leeren Knoten und werden nur wenn nötig angegeben).

Sortierungsbedingung:
Für jeden Knoten v und alle Knoten u im linken und alle Knoten w im rechten Unterbaum von v gilt

$$k(u) < k(v) < k(w)\,.$$



Lemma 18

Die In-Order-Linearisierung eines binären Suchbaumes mit obigen Suchwegbedingungen ergibt die Folge der Schlüssel in (strikt) aufsteigender Folge.

Beweis:

Klar! □

Die Wörterbuch-Operationen:

- *IsElement*(k, T):

$v :=$ Wurzel von T;
while $v \neq$ Blatt **do**
  if $k(v) = k$ **then**
    return $v(v)$
  elif $k(v) > k$ **then**
    $v :=$linkes Kind von v
  else
    $v :=$rechtes Kind von v
  fi
od
return NIL

- *Insert*(k, T):
  Führe *IsElement*(k, T) aus;
  if k nicht in T **then**
  IsElement ergibt Blatt w (NIL-Pointer);
  Füge neuen internen Knoten v mit $k(v) = k$ an Stelle von w ein
  fi

- *Delete*(k, T):
  Führe *IsElement*(k, T) aus;
  if k in T enthalten **then**
  IsElement führt zu Knoten w mit $k = k(w)$
  Falls w keine internen Kinder hat: klar
  Falls w nur ein internes Kind hat: Ersetze w durch dieses
  Ansonsten ersetze w durch seinen In-Order-Vorgänger oder -Nachfolger
  fi

Problem bei natürlichen Suchbäumen:
Bei bestimmten Abfolgen der Wörterbuchoperationen entarten natürliche Suchbäume stark, z.B. bei Einfügen der Schlüssel in monoton aufsteigender bzw. absteigender Folge zu einer linearen Liste der Tiefe n.

Daraus ergibt sich für die Wörterbuch-Operationen *Insert*, *Delete*, *IsElement* eine *worst case*-Komplexität von

$$\Theta(n)\,.$$

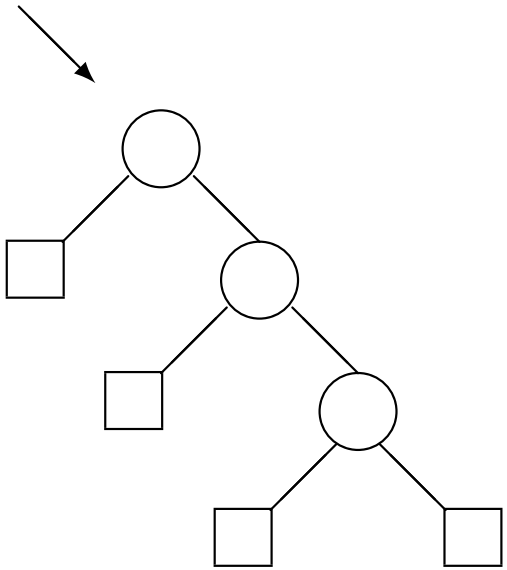
Falls alle Einfügefolgen gleichwahrscheinlich sind, gilt für die Höhe des Suchbaums

$$\mathbb{E}(h) = \mathcal{O}(\log n)\,.$$

Falls alle topologischen (von der Form her gleichaussehenden) Bäume der Größe n gleichwahrscheinlich sind, dann gilt

$$\mathbb{E}(h) = \Theta(\sqrt{n})\,.$$

Ohne Beweis!



3.2 Höhenbalancierte binäre Suchbäume (AVL-Bäume)

Definition 19

AVL-Bäume sind (interne) binäre Suchbäume, die die folgende Höhenbalancierung erfüllen:

Für jeden Knoten v gilt:

$$|\text{Höhe(linker UB}(v)) - \text{Höhe(rechter UB}(v))| \leq 1\,.$$

Bemerkung: AVL-Bäume sind nach ihren Erfindern G. Adelson-Velskii und Y. Landis (1962) benannt.

Satz 20

Ein AVL-Baum der Höhe h enthält mindestens $F_{h+2} - 1$ und höchstens $2^h - 1$ interne Knoten, wobei F_n die n-te Fibonacci-Zahl ($F_0 = 0, F_1 = 1$) und die Höhe die maximale Anzahl von Kanten auf einem Pfad von der Wurzel zu einem (leeren) Blatt ist.

Beweis:

Die obere Schranke ist klar, da ein Binärbaum der Höhe h höchstens

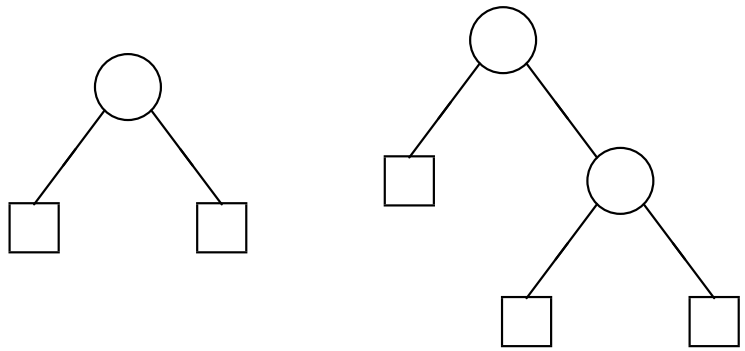
$$\sum_{j=0}^{h-1} 2^j = 2^h - 1$$

interne Knoten enthalten kann.

Beweis:

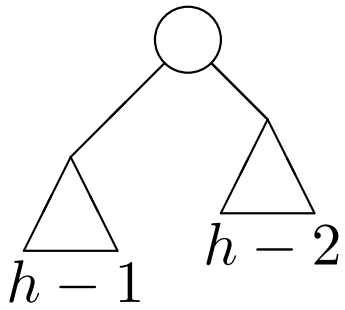
Induktionsanfang:

1. ein AVL-Baum der Höhe $h = 1$ enthält mindestens einen internen Knoten, $1 \geq F_3 - 1 = 2 - 1 = 1$
2. ein AVL-Baum der Höhe $h = 2$ enthält mindestens zwei Knoten, $2 \geq F_4 - 1 = 3 - 1 = 2$



Beweis:

Induktionsschluss: Ein AVL-Baum der Höhe $h \geq 2$ mit minimaler Knotenzahl hat als Unterbäume der Wurzel einen AVL-Baum der Höhe $h-1$ und einen der Höhe $h-2$, jeweils mit minimaler Knotenzahl.



Beweis:

Induktionsschluss: Ein AVL-Baum der Höhe $h \geq 2$ mit minimaler Knotenzahl hat als Unterbäume der Wurzel einen AVL-Baum der Höhe $h-1$ und einen der Höhe $h-2$, jeweils mit minimaler Knotenzahl. Sei

$$f_h := 1 + \text{minimale Knotenanzahl eines AVL-Baums der Höhe } h\,.$$

Dann gilt demgemäß

$$\begin{aligned} f_1 &= 2 && = F_3 \\ f_2 &= 3 && = F_4 \\ f_h - 1 &= 1 + f_{h-1} - 1 + f_{h-2} - 1\,, && \text{also} \\ f_h &= f_{h-1} + f_{h-2} && = F_{h+2} \end{aligned}$$

□

Bemerkung:

Da

$$F(n) \approx \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n ,$$

hat ein AVL-Baum mit n internen Knoten eine Höhe

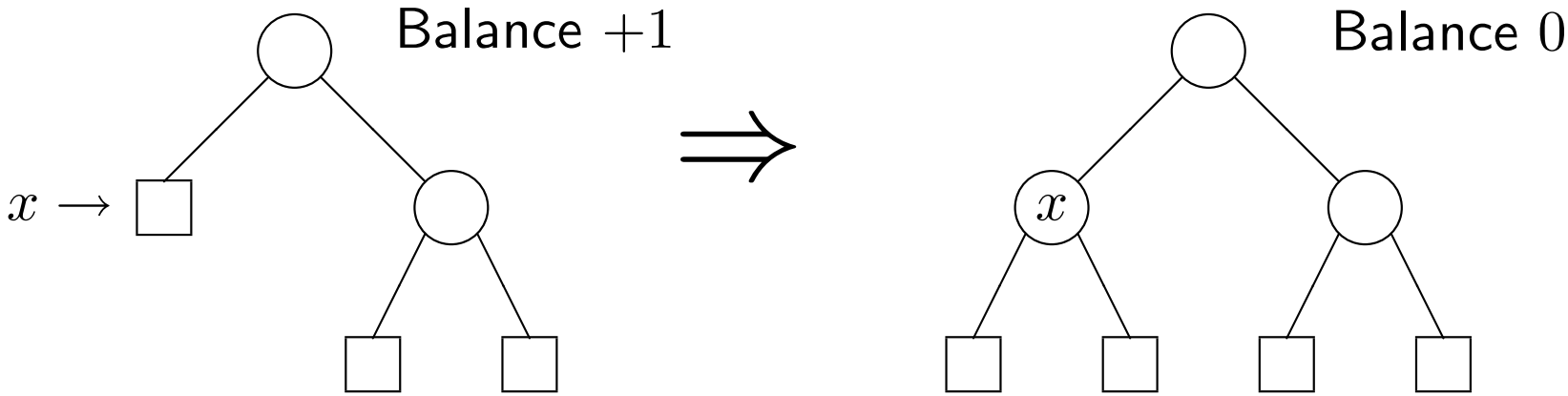
$$\Theta(\log n) .$$

Dies ist ein bedeutender Fortschritt gegenüber der bisweilen bei natürlichen Suchbäumen entstehenden, im *worst-case* linearen Entartung.

Die Operationen auf AVL-Bäumen:

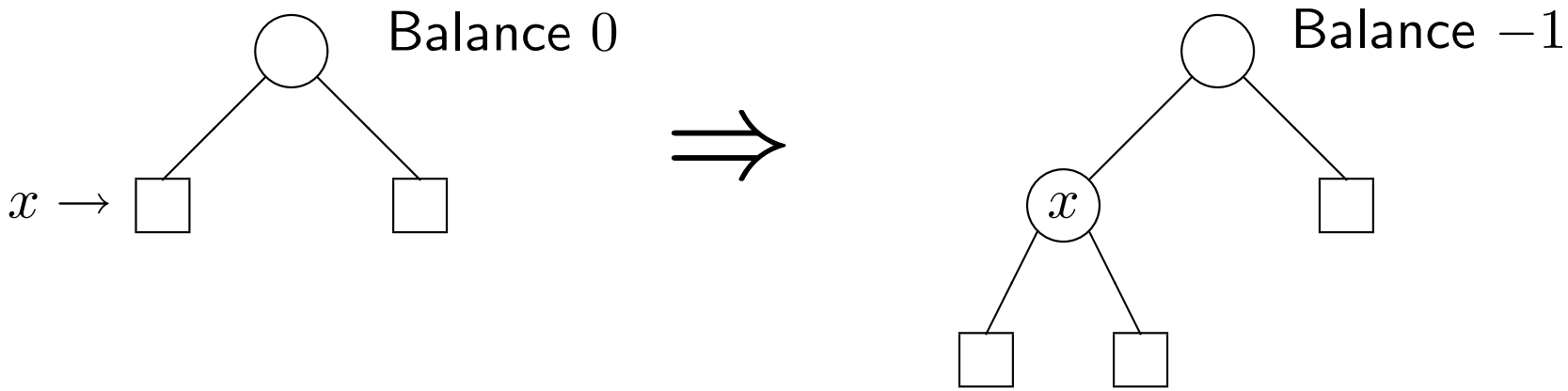
1. *IsElement*: Diese Operation wird wie bei den natürlichen Suchbäumen implementiert. Wie oben ausgeführt, haben aber AVL-Bäume mit n Schlüsseln eine Höhe von $\mathcal{O}(\log n)$, woraus logarithmische Zeit für das Suchen folgt.
2. *Insert*: Zunächst wird eine *IsElement*-Operation ausgeführt, die für den Fall, dass das einzufügende Element nicht bereits enthalten ist, zu einem Blatt führt. Dies ist die Stelle, an der das neue Element einzufügen ist. Dabei ergeben sich 2 Fälle:

1. Fall:



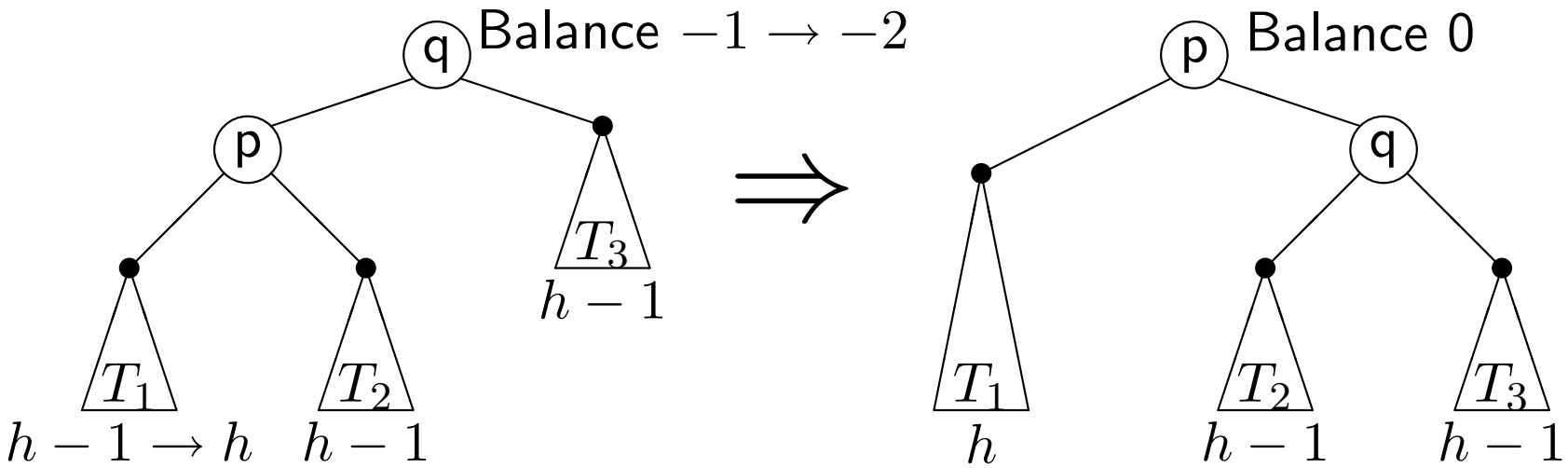
(Balance $-1 \rightarrow 0$: symmetrisch)

2. Fall:

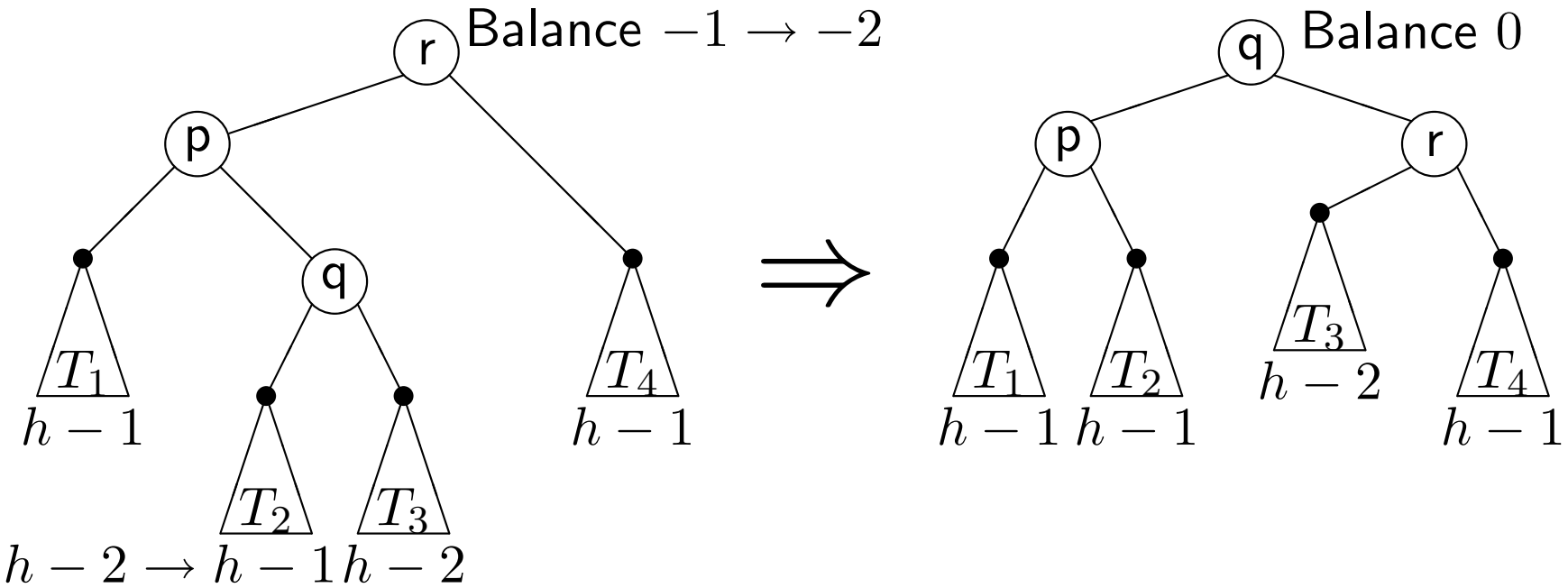


Hier ist eventuell eine Rebalancierung auf dem Pfad zur Wurzel notwendig, denn die Höhe des Unterbaums ändert sich.

Fall 2a:


q
Balance $-1 \rightarrow -2$
p
T_1
$h-1 \rightarrow h$
T_2
$h-1$
T_3
$h-1$
$\Rightarrow$
p
Balance 0
q
T_1
h
T_2
$h-1$
T_3
$h-1$

Fall 2b:


r
Balance $-1 \rightarrow -2$
p
q
T_1
T_2
T_3
T_4
$h-1$
$h-1$
$h-2 \rightarrow h-1$
$h-2$
$\Rightarrow$
q
Balance 0
p
r
T_1
T_2
T_3
T_4
$h-1$
$h-1$
$h-2$
$h-1$

3. *Delete*: Auch diese Operation wird wie bei natürlichen Suchbäumen implementiert. Jedoch hat am Ende ggf. noch eine Rebalancierung auf dem Pfad von dem Blatt, das an die Stelle des zu löschenden Elements geschrieben wird, zur Wurzel zu erfolgen.

Satz 21

Bei AVL-Bäumen sind die Operationen IsElement, Insert, und Delete so implementiert, dass sie die Zeitkomplexität $\mathcal{O}(\log n)$ haben, wobei n die Anzahl der Schlüssel ist.

Beweis:

Klar! □

Im Grundsatz gelten folgende Bemerkungen für AVL-Bäume:

1. Sie haben in der Theorie sehr schöne Eigenschaften, auch zur Laufzeit.
2. Sie sind in der Praxis sehr aufwändig zu implementieren.

Weitere Informationen:


Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman:
The design and analysis of computer algorithms
Addison-Wesley Publishing Company: Reading (MA), 1974


Robert Sedgewick, Philippe Flajolet:
An introduction to the analysis of algorithms
Addison-Wesley Publishing Company, 1996