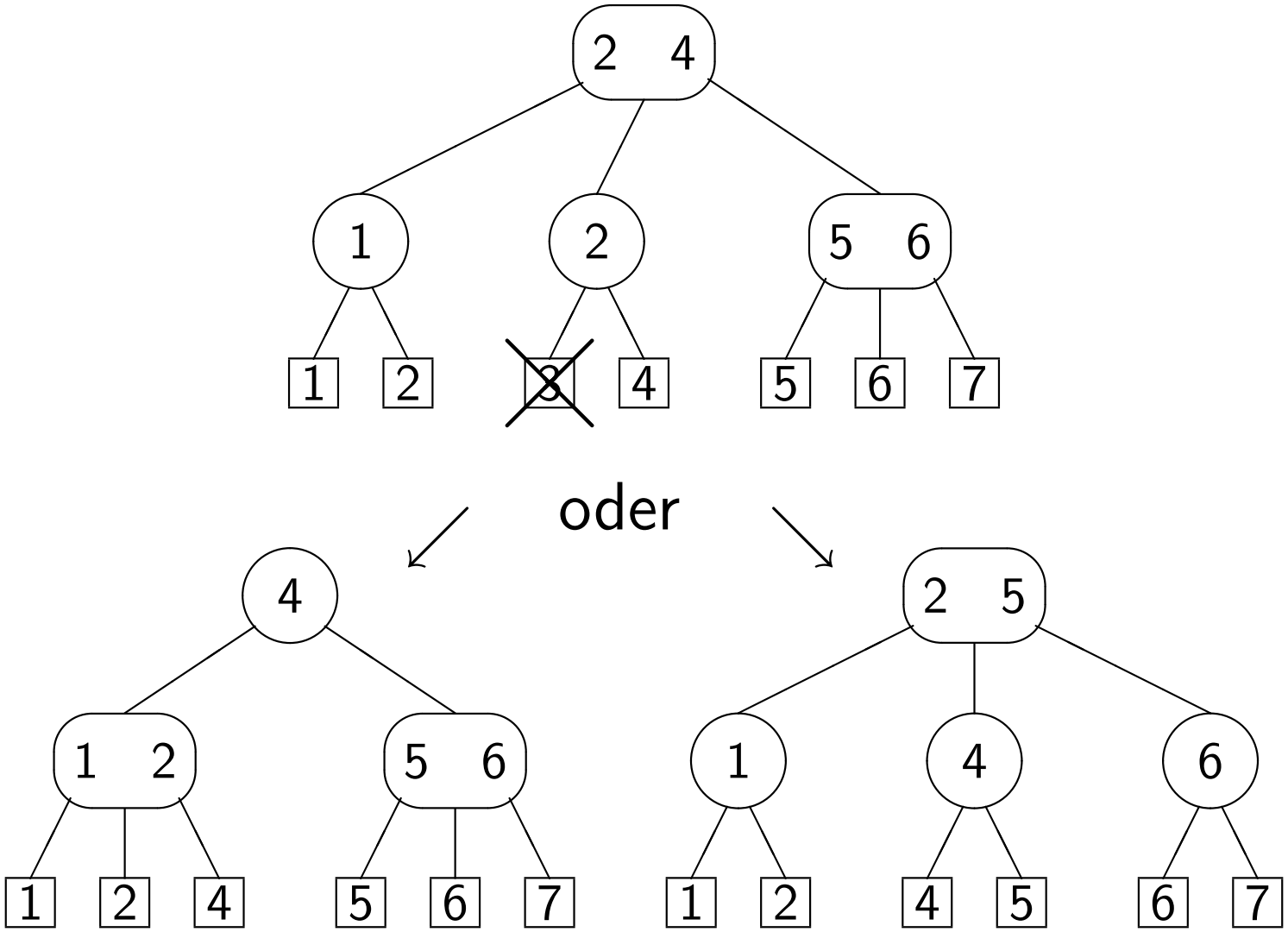
3. *Delete*(k, S):

Führe *IsElement*(k, S) aus. $\rightsquigarrow$ Blatt w. Sei $k(w) = k$;
$v :=$ Vater von w;
Lösche w;
if v hat nunmehr $< a$ Kinder **then**
  führe rekursiv aufsteigend Rebalancierung$'(v)$ durch
fi


2 4
1
2
5 6
1
2
3
4
5
6
7
oder
4
1 2
5 6
1
2
4
5
6
7
2 5
1
4
6
1
2
4
5
6
7

Rebalancierung$'(v)$:

- Falls ein unmittelbarer Nachbar v' von v Grad $> a$ hat, adoptiert v das nächste Kind von v';
- Ansonsten wird v mit einem unmittelbaren Nachbarn verschmolzen; die Gradbedingung für den dadurch entstehenden Knoten ist erfüllt; die Rebalancierung wird rekursiv/iterativ für den Vaterknoten fortgesetzt.

Zeitbedarf: $\mathcal{O}(\log n)$

Spezialfälle von (a, b)-Bäumen:

- $(2, 3)$-Bäume;
- $(2, 4)$-Bäume;
- $(a, 2a - 1)$-Bäume: B-Bäume

Bemerkungen:

1. zur Wahl von a:
   - Daten in RAM $\Rightarrow$ a klein, z.B. $a = 2$ oder $a = 3$.
   - Daten auf Platte $\Rightarrow$ a groß, z.B. $a = 100$.
2. Zur Wahl von b:
   $b \geq 2a$ liefert wesentlich bessere amortisierte Komplexität (ohne Beweis, siehe Mehlhorn).

Top-Down-Rebalancierung: $b \geq 2a$.

Bei der Restrukturierung nach der Top-Down-Strategie folgen wir wie gewohnt dem Pfad von der Wurzel zum gesuchten Blatt. Beim Einfügen stellen wir jetzt jedoch für jeden besuchten Knoten sicher, dass der Knoten weniger als b Kinder hat.
Wenn der gerade betrachtete Knoten ein b-Knoten ist, dann spalten wir ihn sofort auf. Weil der Vater kein b-Knoten ist (das haben wir ja bereits sichergestellt), pflanzt sich der Aufspaltungsprozess nicht nach oben hin fort. Insbesondere kann das neue Element ohne Probleme eingefügt werden, wenn die Suche das Blattniveau erreicht hat.
Damit diese Strategie möglich ist, muss b mindestens $2a$ sein (und nicht nur $2a - 1$), da sonst nach der Spaltung nicht genügend Elemente für die beiden Teile vorhanden sind.

Beim Löschen verfahren wir analog. Für jeden besuchten Knoten, außer der Wurzel des (a, b)-Baumes, stellen wir sicher, dass er mindestens $a + 1$ Kinder hat. Wenn der gerade betrachtete Knoten nur a Kinder hat, so versuchen wir zuerst, ein Element des rechten oder linken Nachbarknoten zu stehlen.
Haben beide Nachbarknoten nur jeweils a Kinder, so verschmelzen wir unseren Knoten mit einem der beiden Nachbarn. Ist der Vater nicht die Wurzel, so hatte er aber vorher mindestens $a + 1$ Kinder, und dieser Verschmelzungsprozess kann sich nicht nach oben fortsetzen. Andernfalls erniedrigt sich der Grad der Wurzel um 1, wenn die Wurzel Grad > 2 hat, oder die alte Wurzel wird gelöscht und der soeben verschmolzene Knoten wird zur neuen Wurzel.

Restrukturierung nach der Top-Down-Strategie sorgt nicht für eine bessere Laufzeit, sondern erleichtert die Synchronisation, wenn mehrere Prozesse gleichzeitig auf einen (a, b)-Baum zugreifen wollen.
Bei herkömmlichen (a, b)-Bäumen schreiten die Such- und die Restrukturierungsoperationen in entgegengesetzter Richtung fort, was dazu führt, dass sich oft Prozesse gegenseitig behindern (der eine will im Baum absteigen, der andere muss auf dem gleichen Pfad aufwärts restrukturieren).
Bei der Top-Down-Strategie gibt es nur Operationsfolgen, die den Baum hinabsteigen. Mehrere Prozesse können so in einer Art Pipeline gleichzeitig einen Pfad mit kurzem Abstand begehen.

2.2 Rot-Schwarz-Bäume

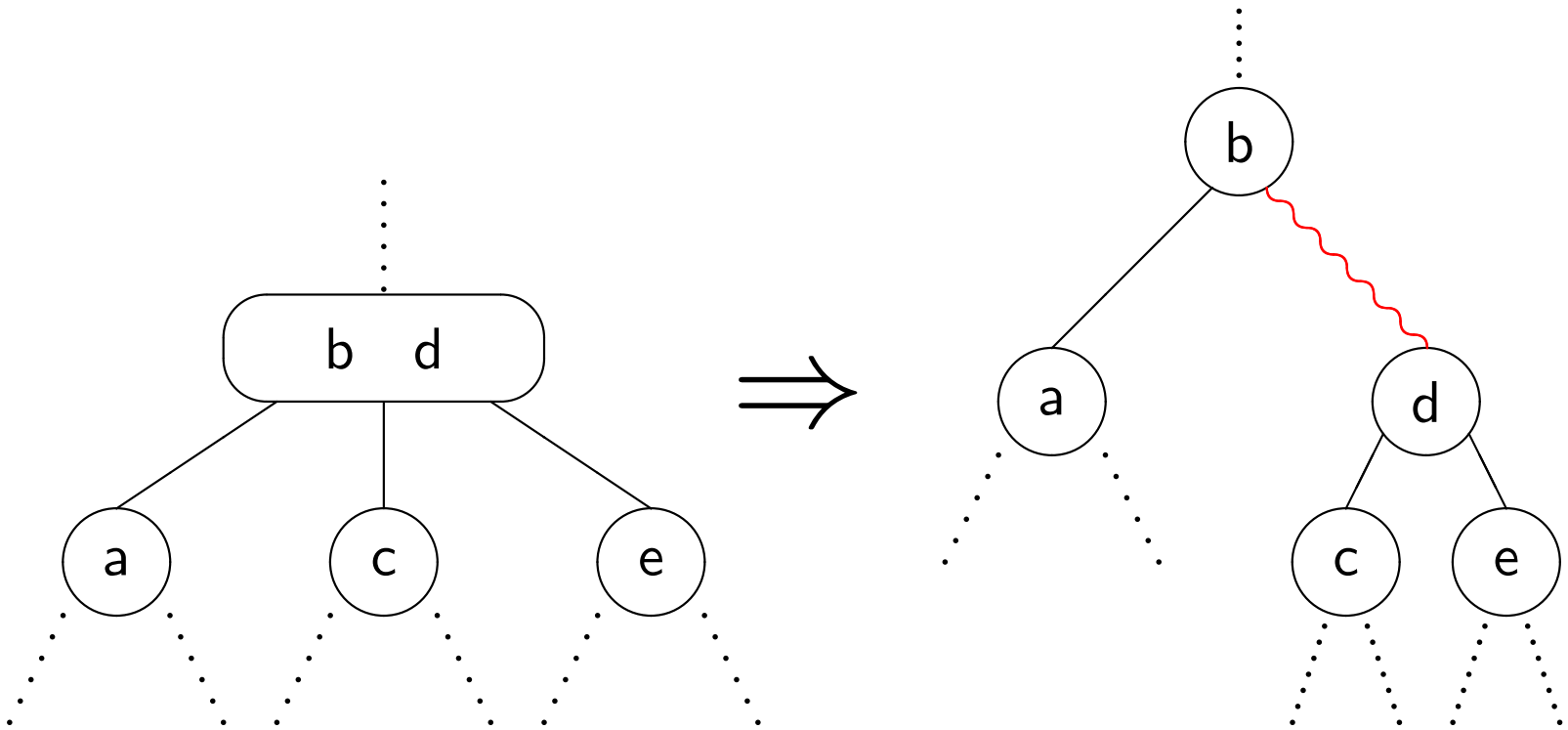
Definition 15

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

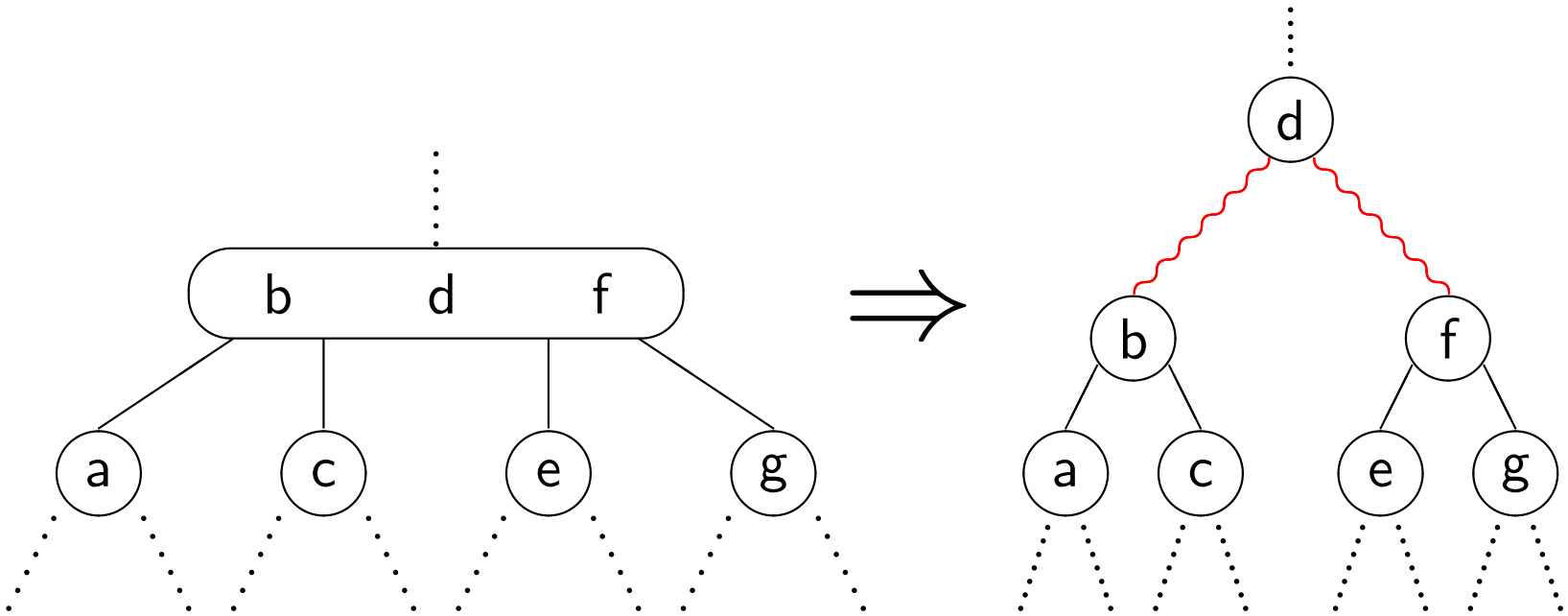
1. alle Blätter hängen an schwarzen Kanten (durchgezogene Linien)
2. alle Blätter haben die gleiche „Schwarztiefe“
3. kein Pfad von der Wurzel zu einem Blatt enthält (zwei oder mehr) aufeinanderfolgende rote Kanten (gewellte Linien).

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:



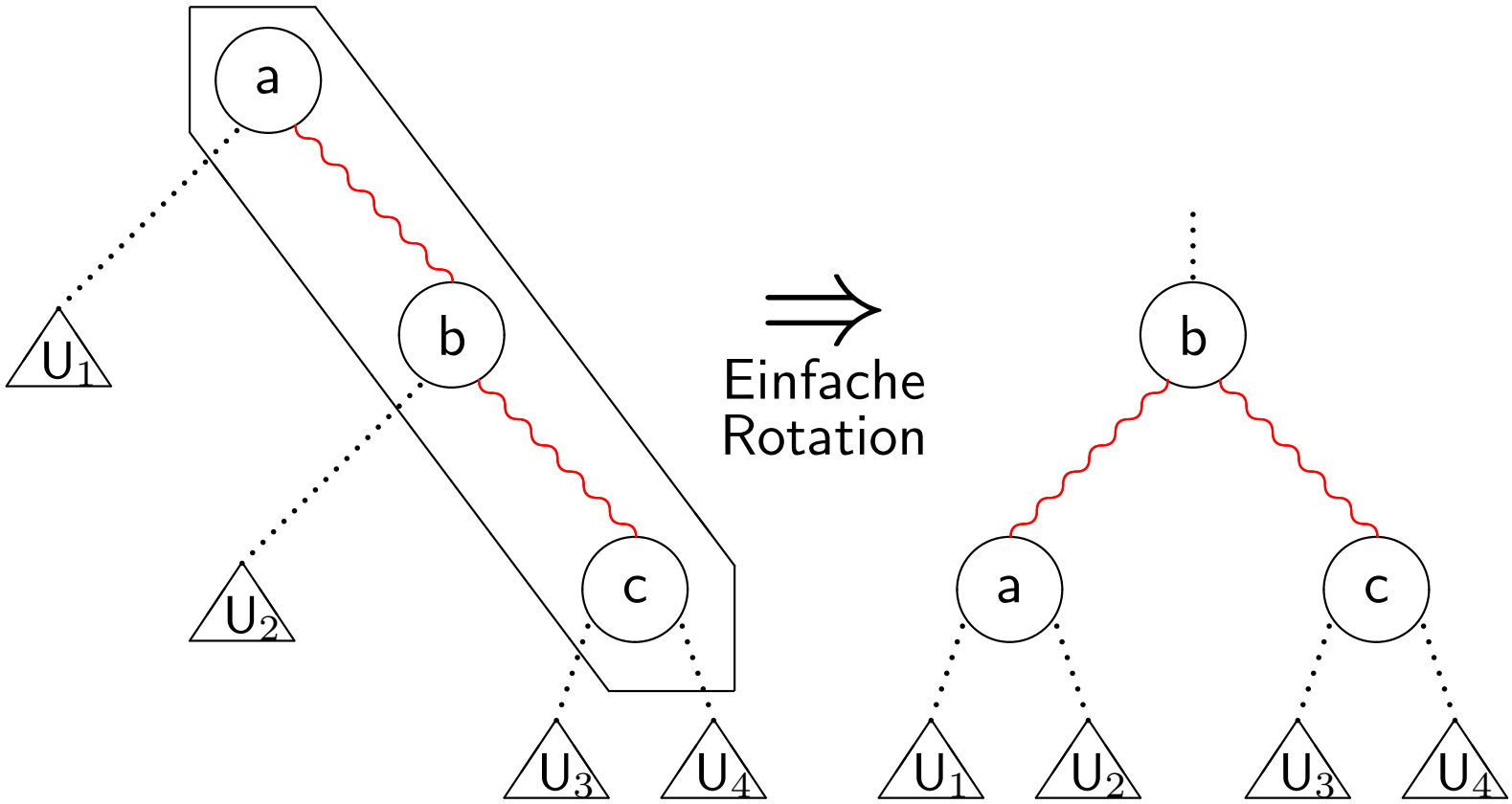
Operationen auf Rot-Schwarz implementierten (a, b)-Bäumen:

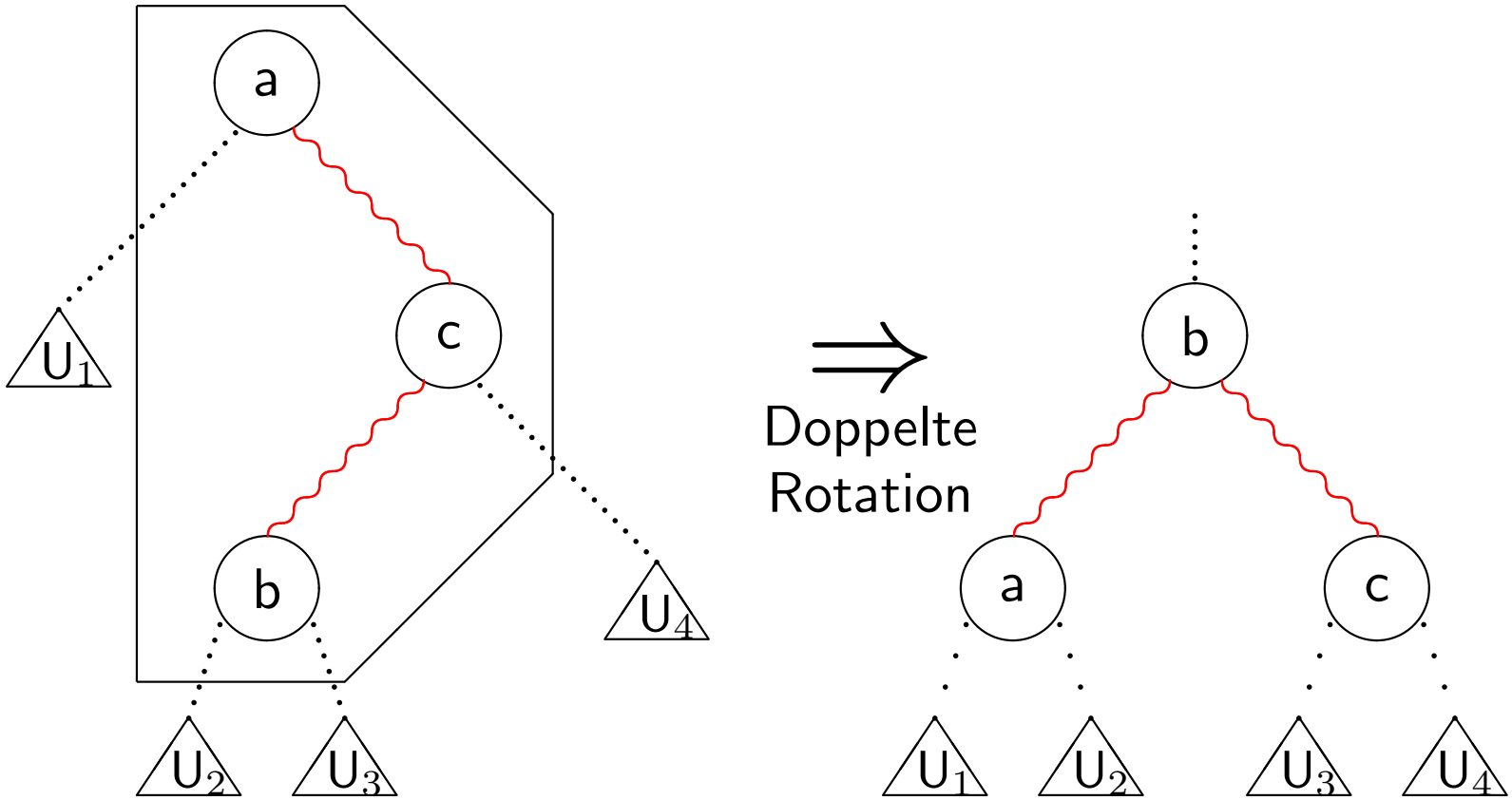
1. *IsElement*(k, T): vgl. (a, b)-Bäume

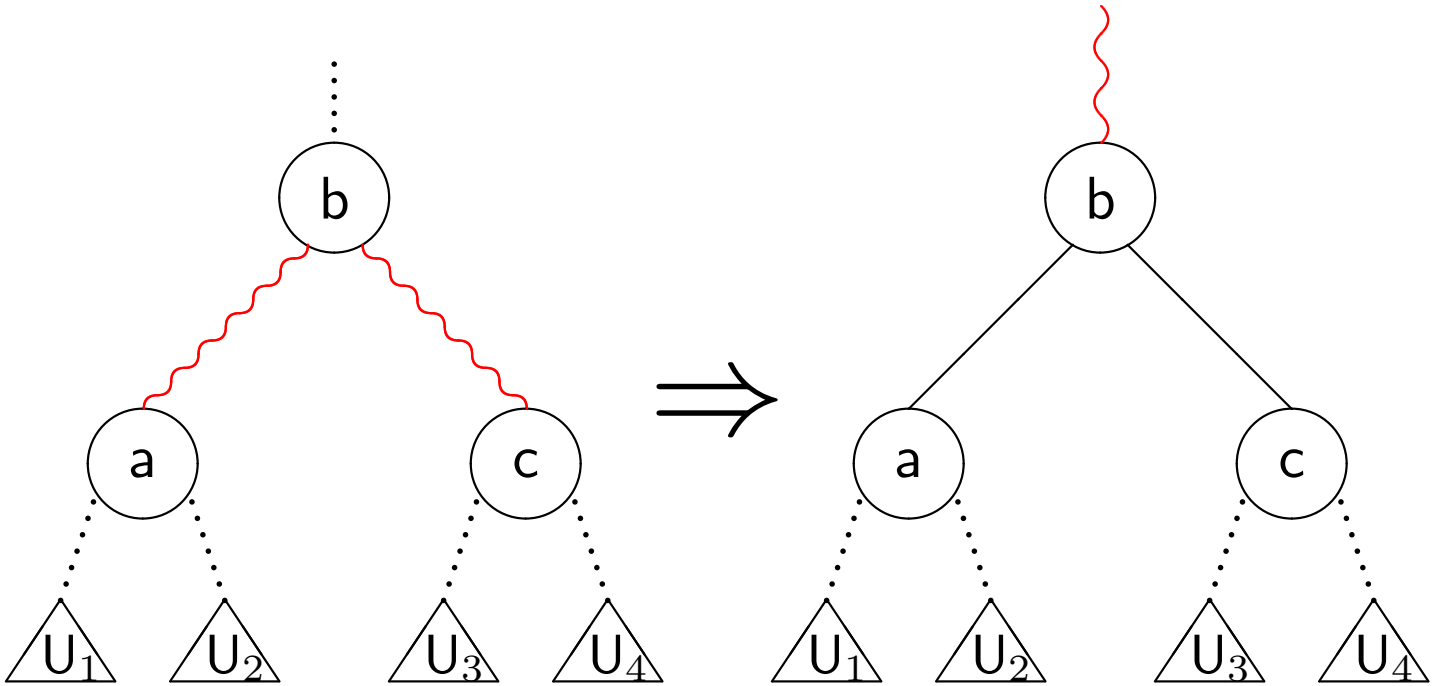
   $$\boxed{\text{Zeit } \mathcal{O}(\log n)}$$

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 (bzw. w) 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

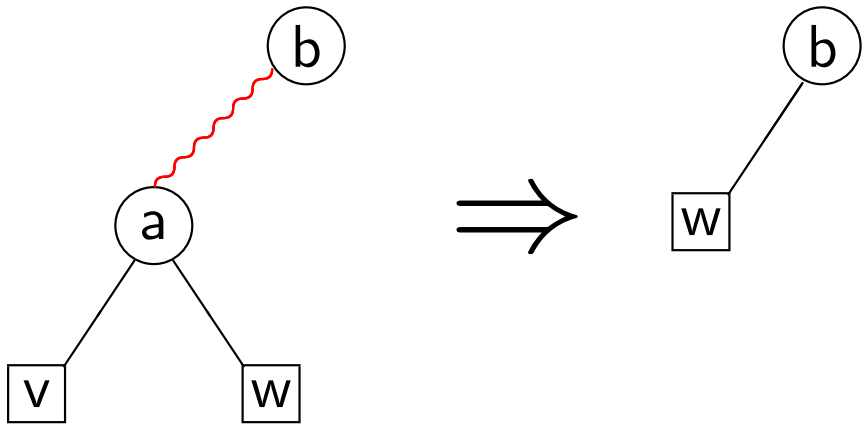

a
c
b
U_1
U_2
U_3
U_4
Doppelte
Rotation
b
a
c
U_1
U_2
U_3
U_4


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.

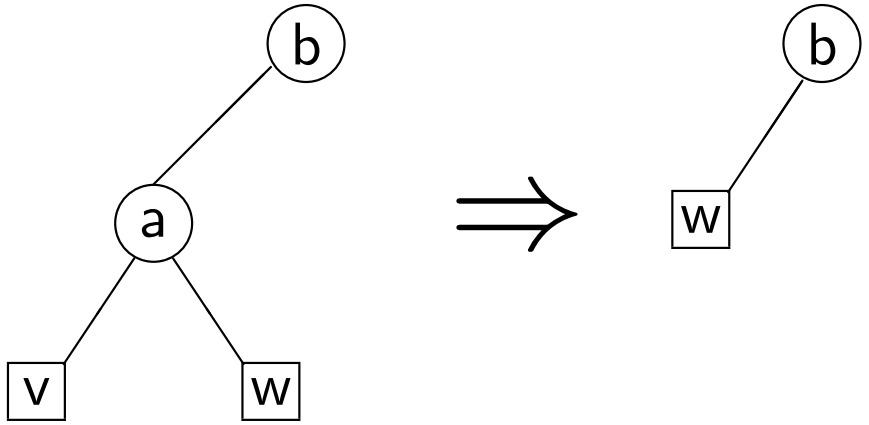
   Dabei treten zwei Fälle auf:

   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. Wir nennen einen Knoten, dessen Schwarztiefe um 1 zu gering ist, „zu hoch“. Falls die Eingangskante eines zu hohen Knoten rot ist, wird sie nach schwarz umgefärbt. Wir unterscheiden nun die folgenden Fälle:

1. Vater von w schwarz, Bruder von w rot: Der Baum wird so rotiert und umgefärbt, dass der Vater rot und der Bruder schwarz ist.
2. Bruder von w schwarz: Die Eingangskante des Bruders wird rot gefärbt und die Prozedur rekursiv mit dem Vater fortgesetzt, der nun (zusammen mit allen Knoten seines Unterbaums) zu hoch ist (Achtung: Falls Vater zuvor rot, Rebalancierung, da dann zwei aufeinander folgende rote Kanten).

Bemerkung: Falls der Bruder von w eine rote Ausgangskante hat, ist im zweiten Fall eine nicht-rekursive Vereinfachung möglich, indem die obersten Ebenen des Unterbaums, dessen Wurzel der Vater von w ist, geeignet umgeordnet werden.

Satz 16

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

Beweis:

Hausaufgabe! □

Korollar 17

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

Beweis:

Hausaufgabe! □

Rot-Schwarz-Bäume wurden von R. Bayer unter der Bezeichnung symmetric binary B-trees erfunden und von Guibas und Sedgewick benannt und weiter erforscht.

R. Bayer:
Symmetric binary B-trees: Data structure and maintenance algorithms,
Acta Inf. **1**, pp. 290–306, 1972

Leo J. Guibas, Robert Sedgewick:
A dichromatic framework for balanced trees,
Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science, pp. 8–21. IEEE Computer Society, 1978

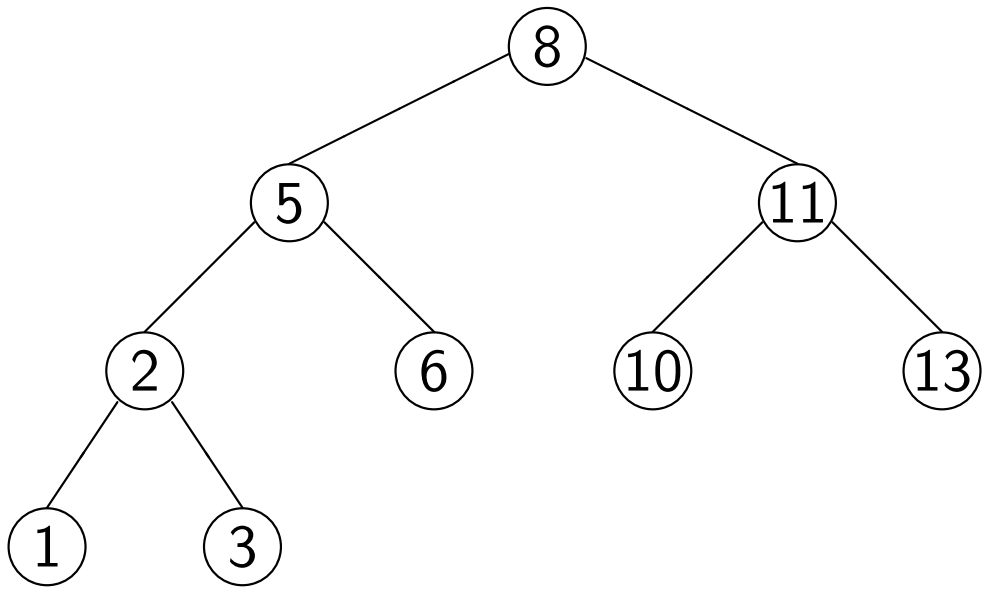
3. Binäre Suchbäume

3.1 Natürliche binäre Suchbäume

Definition 18
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 19

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