AVL-Bäume

Ziele:

- Logarithmischer Zeitaufwand für die Suche (auch im Worst-Case). Hierzu sind nicht-entartete Bäume notwendig.
- Logarithmischer Zeitaufwand für Änderungen.

AVL-Bäume: Definition

- AVL-Bäume gehen zurück auf die russischen Mathematiker G. M. Adelson-Velsky und J. M. Landis (1962).

Definition 6.6. Ein Binärbaum heißt *AVL-Baum*, wenn für jeden Knoten w das sogenannte *AVL-Kriterium* gilt:

$$|h(T_l(w)) - h(T_r(w))| \leq 1$$

d.h. an jedem Knoten w unterscheiden sich die Höhe des linken Unterbaums $T_l(w)$ und die Höhe des rechten Unterbaums $T_r(w)$ von w höchstens um eins.

AVL-Bäume: Maximale Höhe

Satz 6.3. *Es sei* T *ein AVL-Baum mit* n *Knoten. Dann gilt:*

$$h(T) < 1.5 \log_2(n + 1.5)$$

Beweis:

- Es sei n_h die minimale Anzahl an Knoten in einem AVL-Baum der Höhe h. Dann gilt:
  $n_0 = 0, n_1 = 1, n_2 = 2$
  $n_h = n_{h-1} + n_{h-2} + 1$
- Wir betrachten: $\tilde{n}_h := n_h + 1$. Dann gilt:
  - $\tilde{n}_h = \tilde{n}_{h-1} + \tilde{n}_{h-2}$
  - $\tilde{n}_0 = 1, \tilde{n}_1 = 2$
- F_h sei die h-te Fibonacci-Zahl. Dann gilt: $F_h = \tilde{n}_{h-2}$

- Es gilt:

$$F_h = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^h - \left(\frac{1-\sqrt{5}}{2} \right)^h \right)$$

- Weiterhin gilt für die Fibonacci-Zahlen die folgende Ungleichung:

$$F_h + 1/2 > \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^h$$

- Wir definieren: $\alpha := \frac{1+\sqrt{5}}{2}$. Mit $n_h = \tilde{n}_h - 1 = F_{h+2} - 1$ folgt

$$n_h + \frac{3}{2} \geq \frac{1}{\sqrt{5}} \alpha^{h+2}$$

Daraus folgt:

$$\alpha^{h+2} \leq \sqrt{5} \left(n_h + \frac{3}{2} \right) \quad \Leftrightarrow \quad h + 2 \leq \frac{\log_2(\sqrt{5}(n_h + \frac{3}{2}))}{\log_2 \alpha}$$

$$\Leftrightarrow \quad h \leq \frac{\log_2(n_h + \frac{3}{2})}{\log_2 \alpha} + \frac{\log_2 \sqrt{5}}{\log_2 \alpha} - 2$$

$$\Leftrightarrow \quad h \leq 1.4405 \log_2(n_h + \frac{3}{2}) + c$$

mit

$$1.4405 = 1/\log_2\alpha \text{ und } c = \frac{\log_2 \sqrt{5}}{\log_2 \alpha} - 2$$

Bemerkungen:

- Die Höhe eines AVL-Baums wächst auch im ungünstigsten Fall nur mit dem Logarithmus der Gesamtknotenanzahl, also sehr langsam.
- Ein AVL-Baum mit $n = 10^6$ Einträgen hat auch im ungünstigsten Fall eine Höhe ≤ 30.
- Das langsame Anwachsen der Baumhöhe garantiert, dass die Operationen `contains()` bzw. `get()` stets effizient ausgeführt werden können.
- Die entscheidende Frage ist, ob auch die Operationen `insert()` und `remove()` mit Hilfe eines AVL-Baums effizient ausgeführt werden können.

Einfügen und Rebalancierung (1)

- Wir wollen AVL-Bäume als Suchbäume nutzen.
- Somit können wir ein neues Element in einen AVL-Baum wie in einen gewöhnlichen Suchbaum einfügen.
- Der entstehende Suchbaum braucht aber kein AVL-Baum mehr zu sein.
- Fazit: Nach dem Einfügen eines Elements kann das AVL-Kriterium an verschiedenen Knoten verletzt sein.

Einfügen und Rebalancierung (2)

- ☞ Diejenigen Knoten, für die nach dem Einfügen das AVL-Kriterium verletzt ist, können nur auf dem Pfad liegen, der von der Wurzel zu dem neuen Knoten führt.
- ☞ Die sind höchstens h und somit höchstens $O(\log n)$ viele Knoten.

Rebalancierung:

- Bei den Knoten, an denen das AVL-Kriterium verletzt ist, wird der AVL-Baum durch Rebalancierung so transformiert, dass
  (a) wieder überall das AVL-Kriterium erfüllt ist und
  (b) der Baum weiterhin sortiert ist, also die Bedingungen für einen Suchbaum erfüllt.

Einfügen und Rebalancierung (3)

- Im folgenden wird ein AVL-Unterbaum $\mathfrak{U}$ betrachtet, in den eingefügt wird.
- $\mathfrak{U}$ besteht aus der Wurzel w und den Unterbäumen $\mathfrak{A}$ (links) und $\mathfrak{B}$ (rechts).

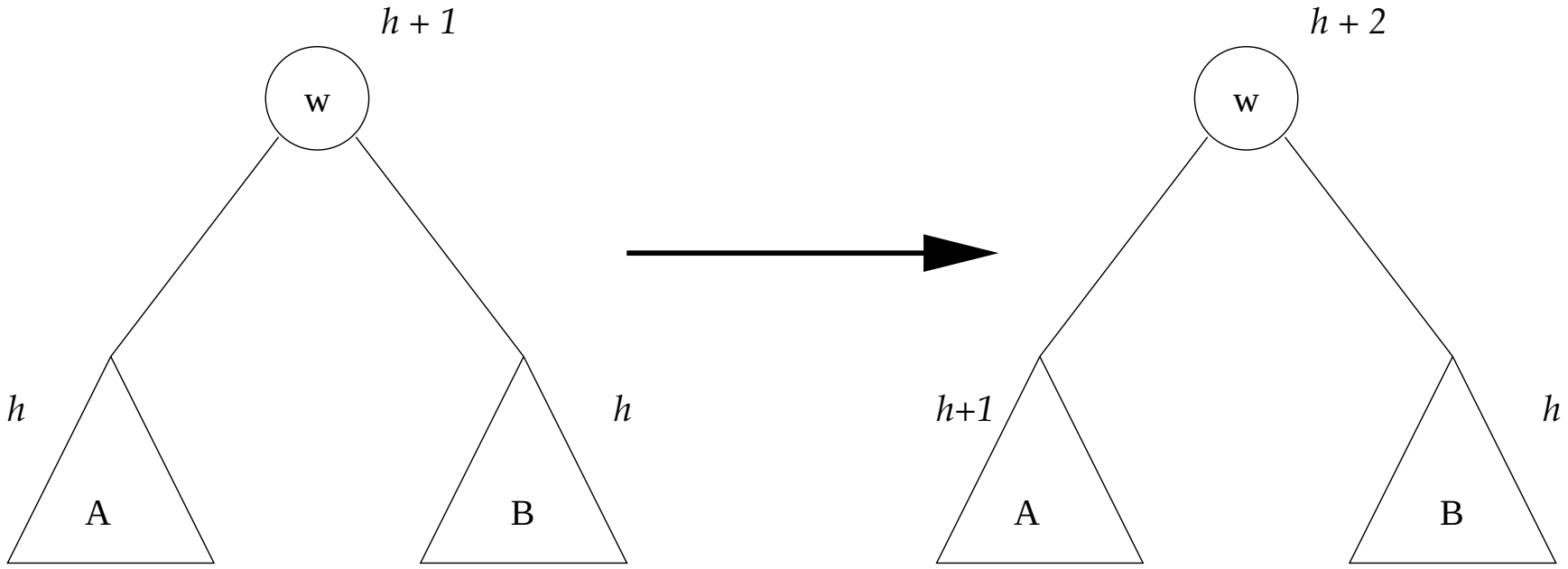
Insgesamt kann man beim Einfügen die folgenden Situationen unterscheiden:

Fall 1:

- $\mathfrak{A}$ und $\mathfrak{B}$ haben die gleiche Höhe $\mathfrak{h}$.
- Durch Einfügen in $\mathfrak{A}$ ändert sich die Höhe von $\mathfrak{A}$.

$\Rightarrow$ Das AVL-Kriterium bleibt am Knoten w erfüllt.

Veranschaulichung:



Bemerkungen:

- Der enstandene Unterbaum ist wieder ein AVL-Baum, d.h. an w ist keine Rebalancierung notwendig.
- Da sich aber die Höhe des Unterbaums $\mathfrak{U}$ geändert hat, kann sich diese Höhenänderung zur Wurzel hin fortpflanzen.
- An einem Vorfahr von w könnte demnach das AVL-Kriterium verletzt sein. Dort liegt dann eine der Situationen von Fall 3 vor (s.u.).

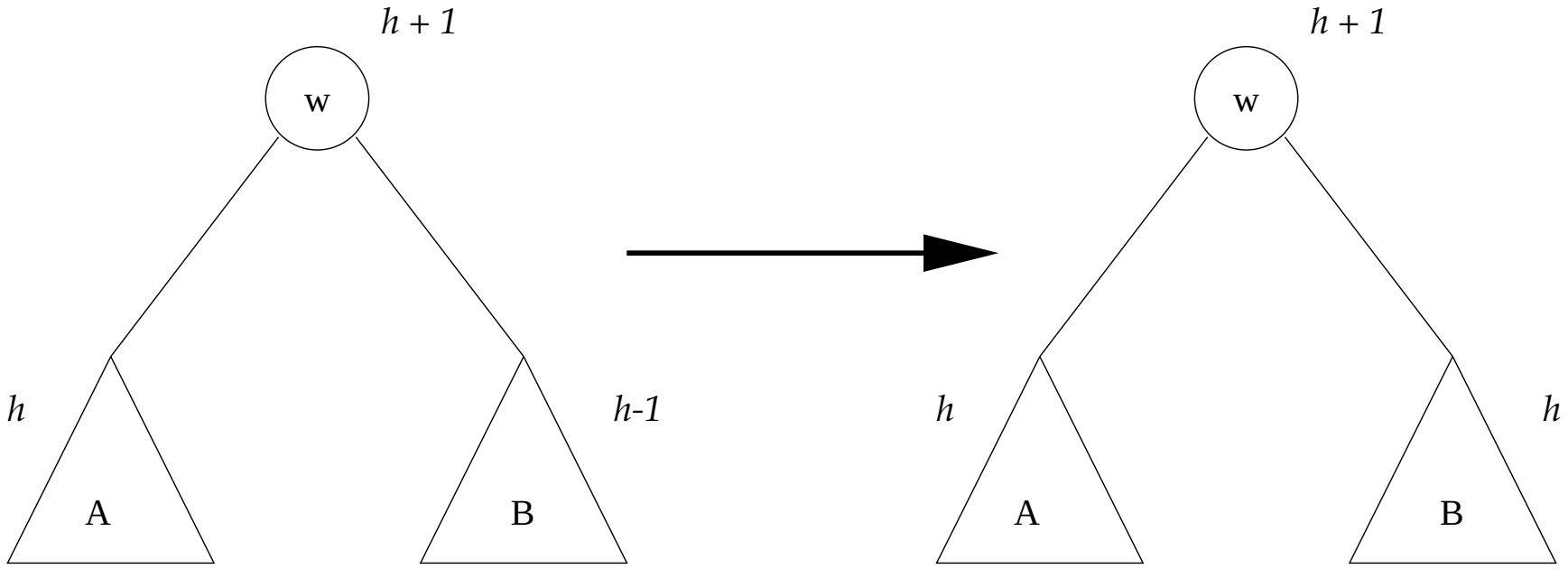
Einfügen und Rebalancierung (4)

Fall 2:

- B ist niedriger als A.
- Durch Einfügen in B ändert sich die Höhe von B.

$\Rightarrow$ Das AVL-Kriterium bleibt am Knoten w erfüllt.

Veranschaulichung:



Bemerkungen:

- Der enstandene Unterbaum ist wieder ein AVL-Baum, d.h. an w ist keine Rebalancierung notwendig.
- Da Für den Unterbaum $\mathfrak{U}$ keine Höhenveränderung stattgefunden hat, wird an w die Änderung absorbiert,
- d.h. an keinem Vorfahr von w wird eine Rebalancierung notwendig sein.

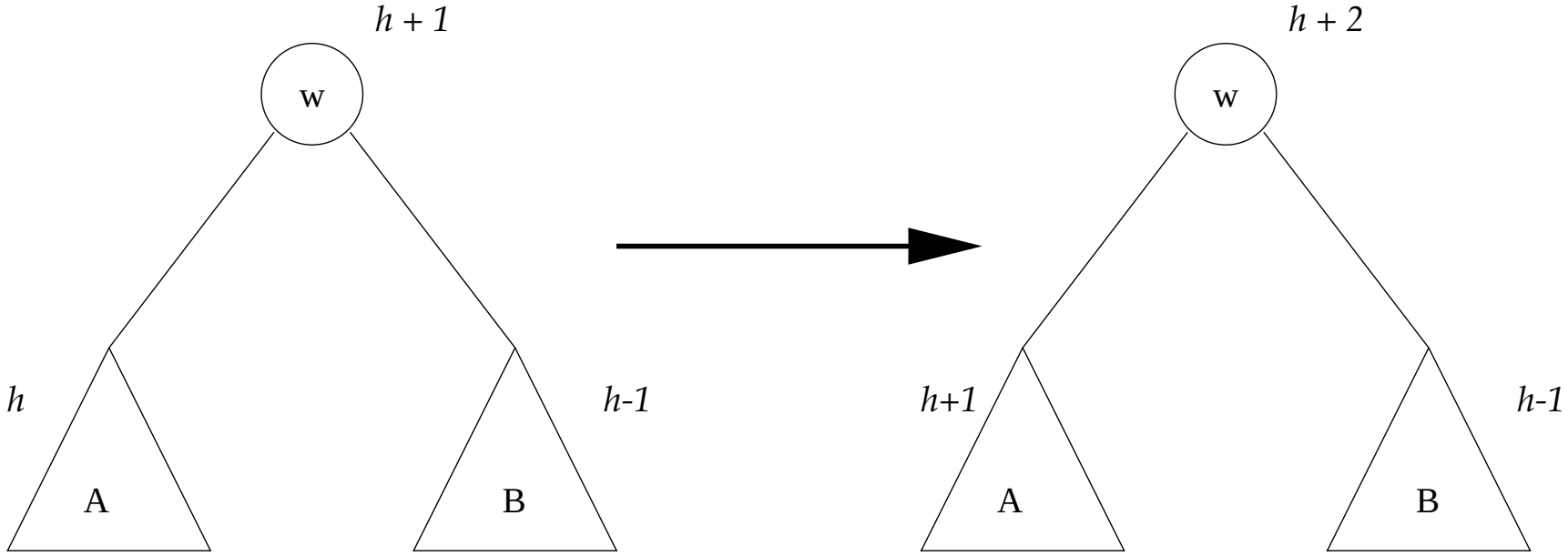
Einfügen und Rebalancierung (5)

Fall 3:

- B ist niedriger als A.
- Durch Einfügen in A ändert sich die Höhe von A.

$\Rightarrow$ Das AVL-Kriterium wird verletzt, d.h. der so entstandene Baum muss rebalanciert werden.

Veranschaulichung:

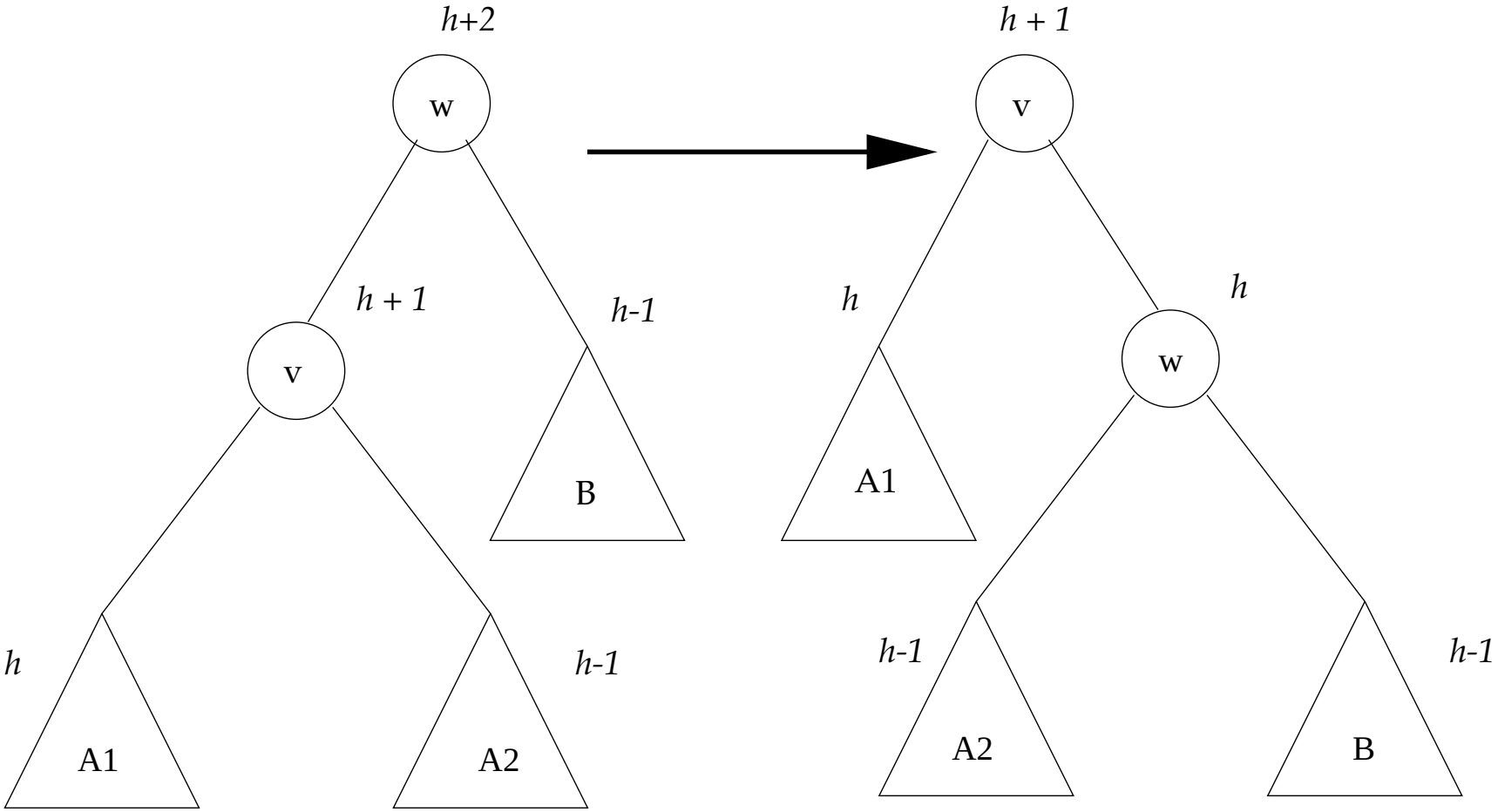


Hier können nun mehrere Unterfälle unterschieden werden.

Fall 3a:

- Im Unterbaum A ist der linke Unterbaum der höhere,
- d.h. die Höhenveränderung enstand durch Einfügen in den linken Unterbaum von A.
- In diesem Fall wird die Rebalancierung durch eine sogenannte *LL-Rotation* vorgenommen.

Veranschaulichung:



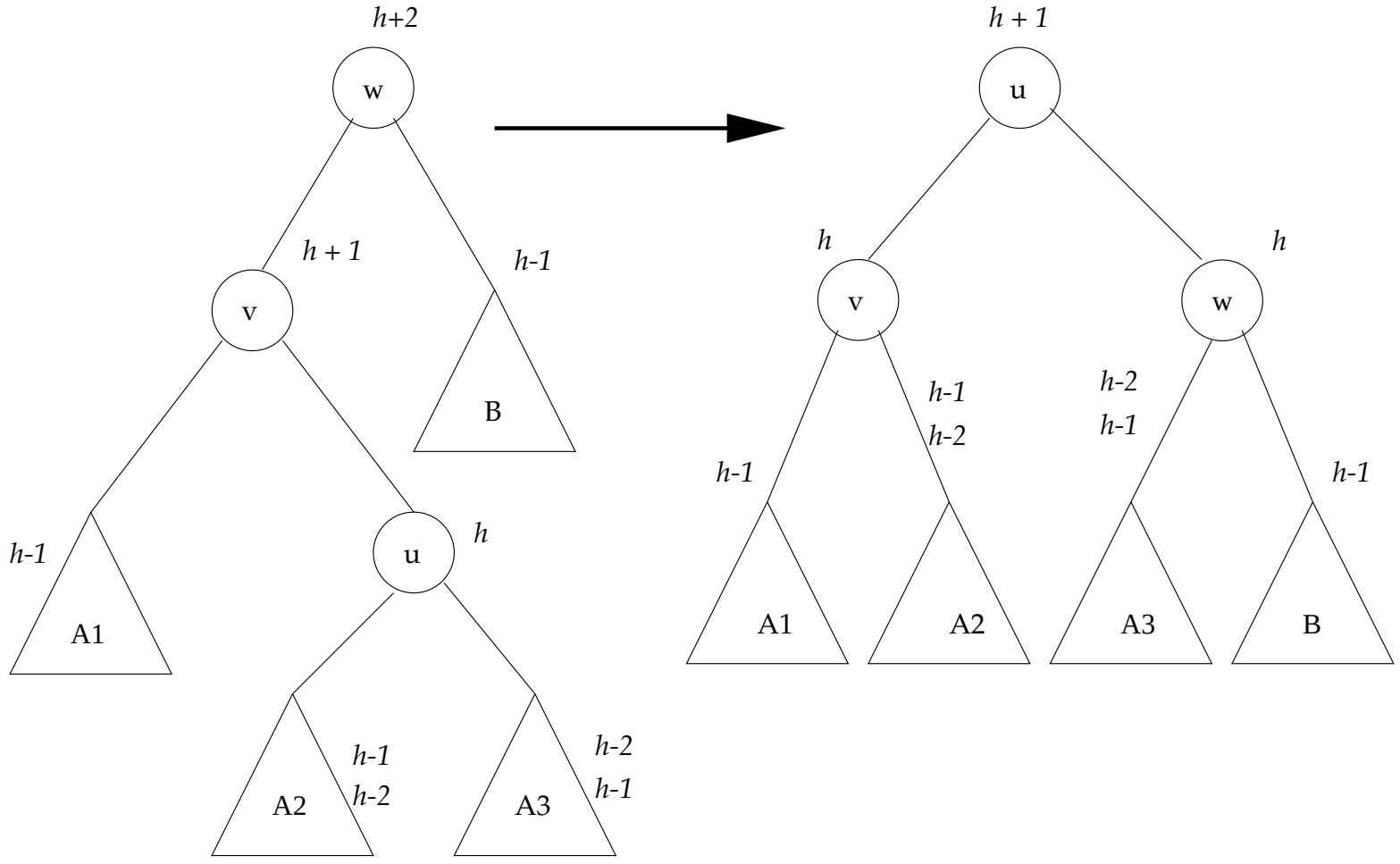
Bemerkungen:

- Nach der LL-Rotation erfüllt der Unterbaum das AVL-Kriterium.
- Die Sortierreihenfolge bleibt bei der LL-Rotation erhalten.
- Der Unterbaum U hat nach der LL-Rotation die gleiche Höhe wie vor der Einfügung.
- Damit wird durch die LL-Rotation die Änderung absorbiert, d.h. auf höherer Ebene ist keine Rebalancierung mehr notwendig.

Fall 3b:

- Im Unterbaum A ist der rechte Unterbaum der höhere,
- d.h. die Höhenveränderung enstand durch Einfügen in den rechten Unterbaum von A.
- In diesem Fall wird die Rebalancierung durch eine sogenannte *LR-Rotation* vorgenommen.

Veranschaulichung:



Bemerkungen:

- Nach der LR-Rotation erfüllt der Unterbaum das AVL-Kriterium.
- Die Sortierreihenfolge bleibt bei der LR-Rotation erhalten.
- Der Unterbaum hat nach der LR-Rotation die gleiche Höhe wie vor der Einfügung.
- Damit wird durch die LR-Rotation die Änderung absorbiert, d.h. auf höherer Ebene ist keine Rebalancierung mehr notwendig.

Einfügen und Rebalancierung: Fazit

- Das AVL-Kriterium kann durch Einfügen zerstört werden, aber durch einfache Umformungen kann es wieder hergestellt werden.
- Der dabei erforderliche Aufwand ist höchstens proportional zur Höhe des Baumes, also $O(\log n)$.
- Es wurden nur die Fälle betrachtet, in denen gilt $h(A) \geq h(B)$. Die anderen Fälle sind zu den behandelten symmetrisch.
- Bei dem zu Fall 3a symmetrischen Fall spricht man von einer *RR-Rotation*.
- Bei dem zu Fall 3b symmetrischen Fall spricht man von einer *RL-Rotation*.
- LL- und RR-Rotation werden auch als *einfache* Rotationen bezeichnet, LR- und RL-Rotation als *doppelte Rotationen*.

Löschen und Rebalancierung (1)

- Zunächst nutzt man den Löschalgorithmus von binären Bäumen.
- Es wird also immer ein Knoten gelöscht, der mindestens einen leeren Unterbaum hat.
- Dadurch kann sich die Höhe eines Unterbaums verringern und somit das AVL-Kriterium verletzt werden.
- Dieses muss anschließend durch Rebalancierung wieder hergestellt werden.
- Hier ergeben sich ähnliche Fälle wie beim Einfügen.

Löschen und Rebalancierung (2)

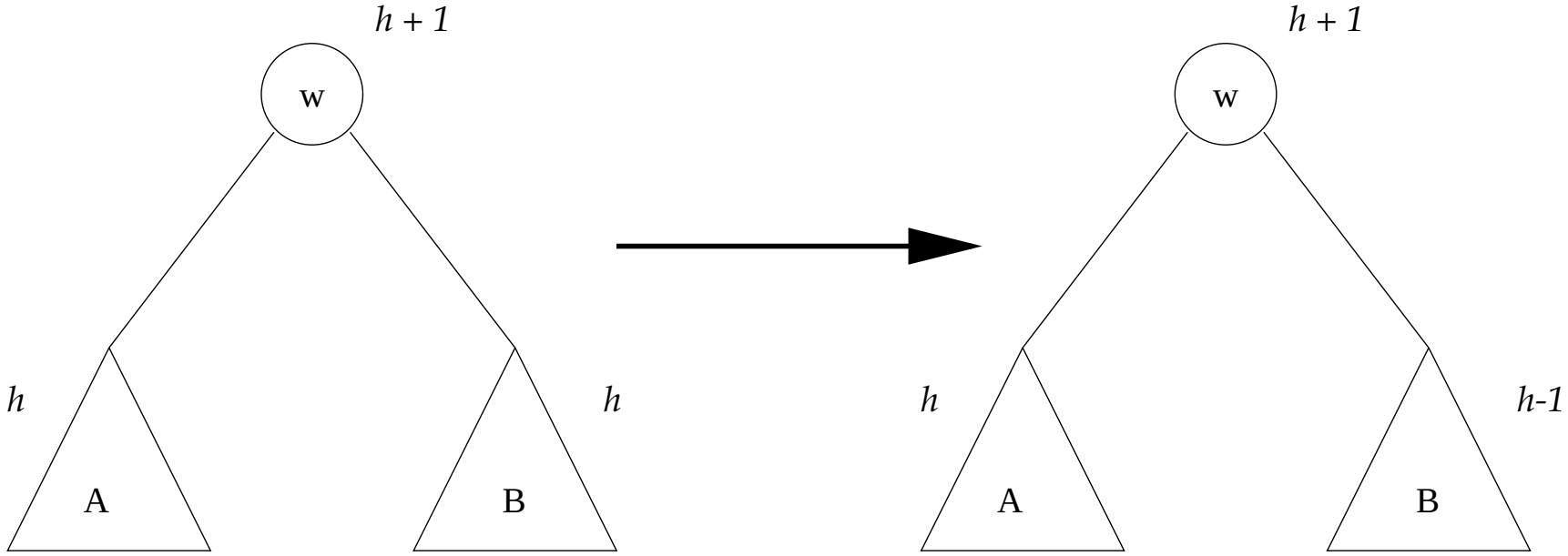
Fall 1:

- A und B haben die gleiche Höhe h.
- Durch Löschen in A (oder B) ändert sich die Höhe von A (bzw. B).

$\Rightarrow$ Das AVL-Kriterium bleibt am Knoten w erfüllt.

- Die Höhenänderung pflanzt sich nicht fort.

Veranschaulichung:



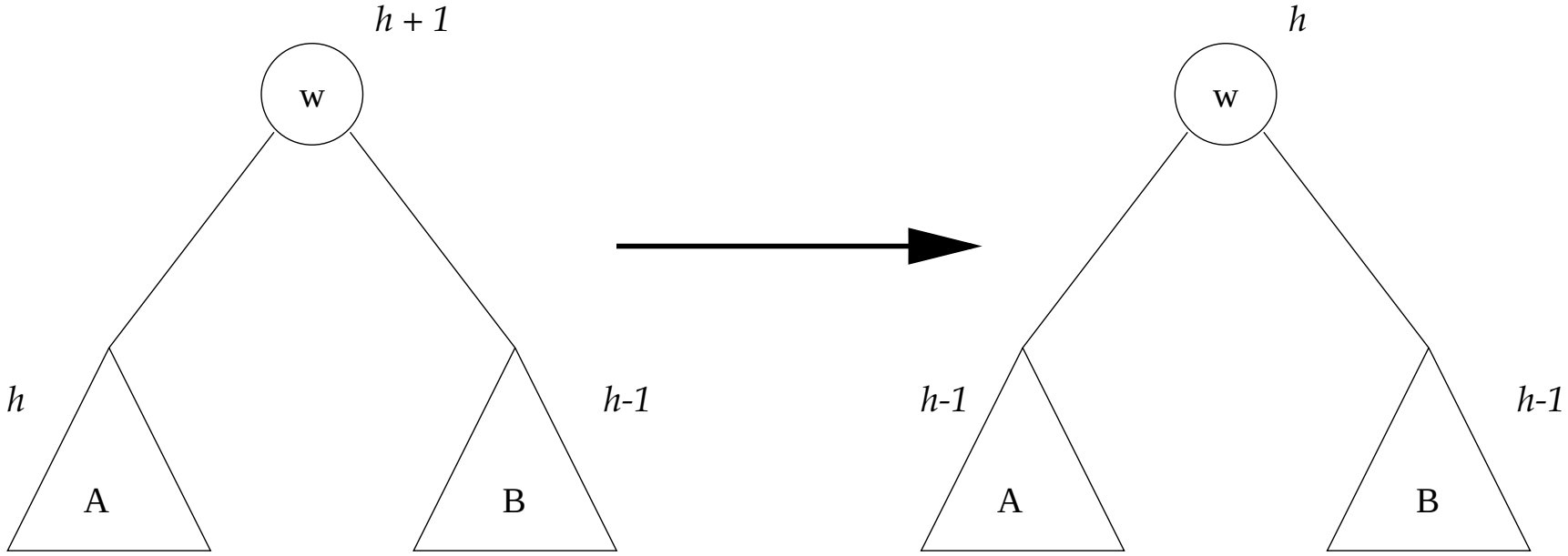
Löschen und Rebalancierung (3)

Fall 2:

- B ist niedriger als A.
- Durch Löschen in A ändert sich die Höhe von A.

$\Rightarrow$ Das AVL-Kriterium bleibt am Knoten w erfüllt.

Veranschaulichung:



- Die Höhe des Unterbaums U hat sich verringert.
- Die Höhenveränderung kann sich somit zur Wurzel hin fortpflanzen.
- An einem der Vorgnger von w kann eine Rebalancierung erforderlich werden.

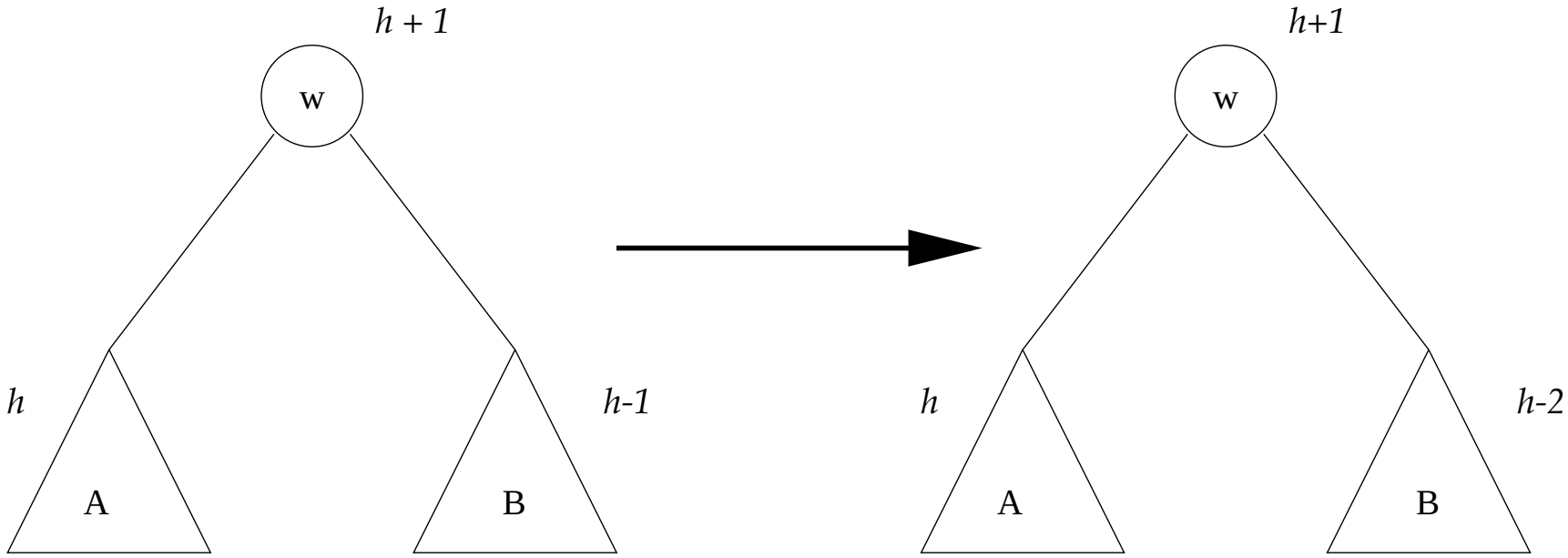
Löschen und Rebalancierung (4)

Fall 3:

- B ist niedriger als A.
- Durch Löschen in B ändert sich die Höhe von B.

$\Rightarrow$ Das AVL-Kriterium ist damit am Knoten w verletzt.

Veranschaulichung:

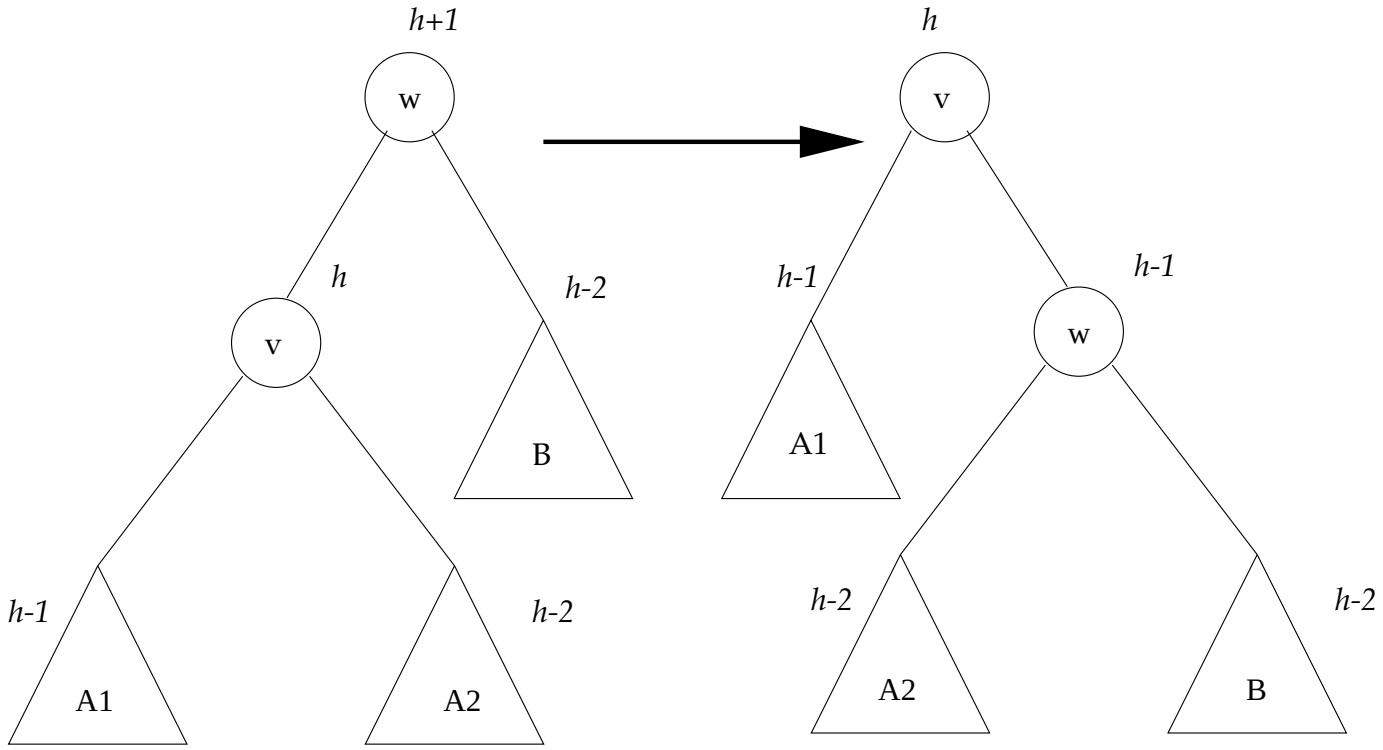


- Analog zum Einfügen können hier verschiedene Unterfälle unterschieden werden.

Fall 3a:

- Im Unterbaum A ist der linke Unterbaum der höhere.
- Ausgleich durch eine LL-Rotation.

Veranschaulichung:



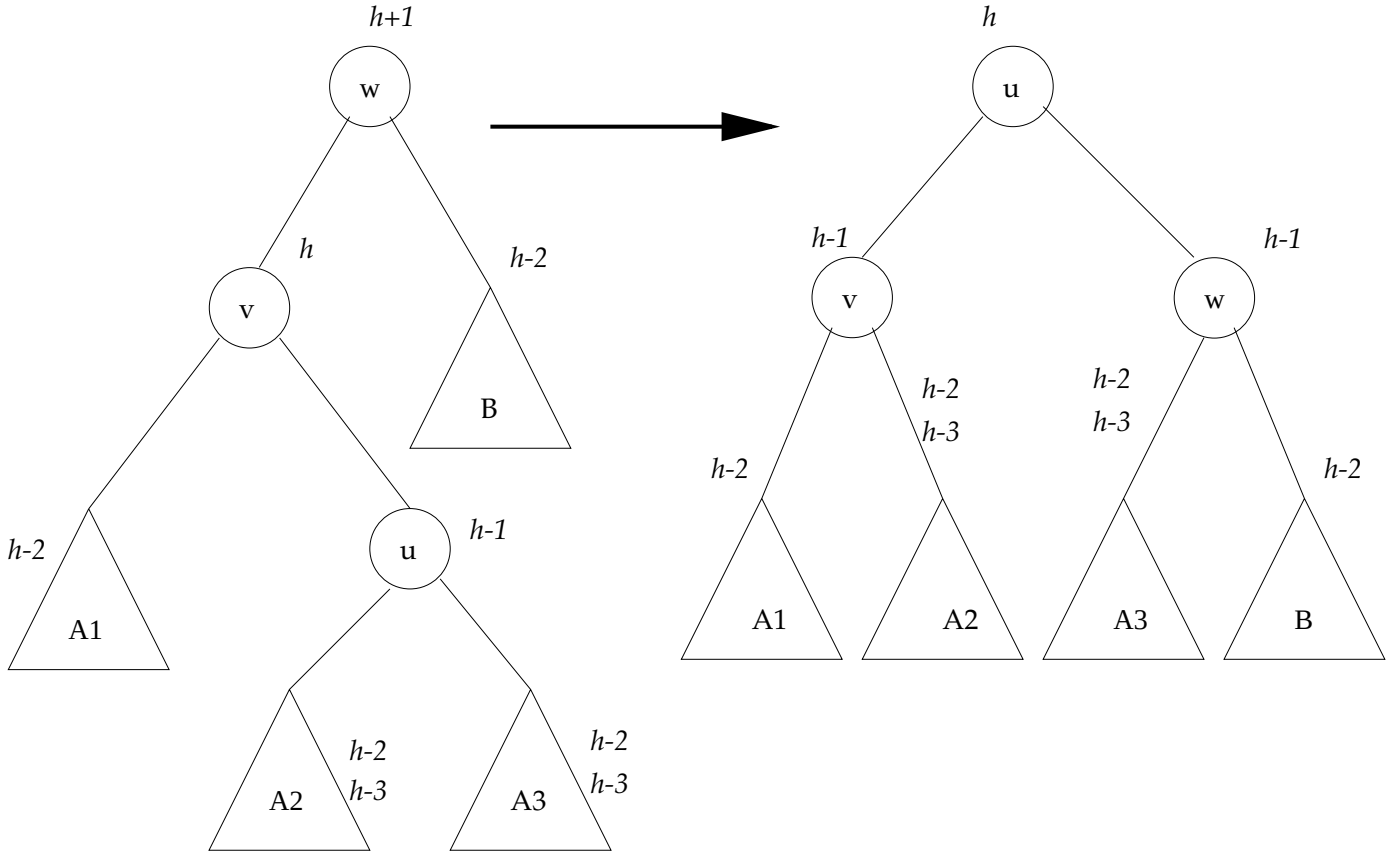
Bemerkungen:

- Durch die LL-Rotation hat sich die Höhe des Unterbaums $\mathfrak{U}$ verringert.
- Die Höhenänderung kann sich zur Wurzel hin fortpflanzen.
- Eventuell sind bei Vorfahren von w weitere Rotationen notwendig.

Fall 3b:

- Im Unterbaum A ist der rechte Unterbaum der höhere.
- Ausgleich durch eine LR-Rotation.

Veranschaulichung:



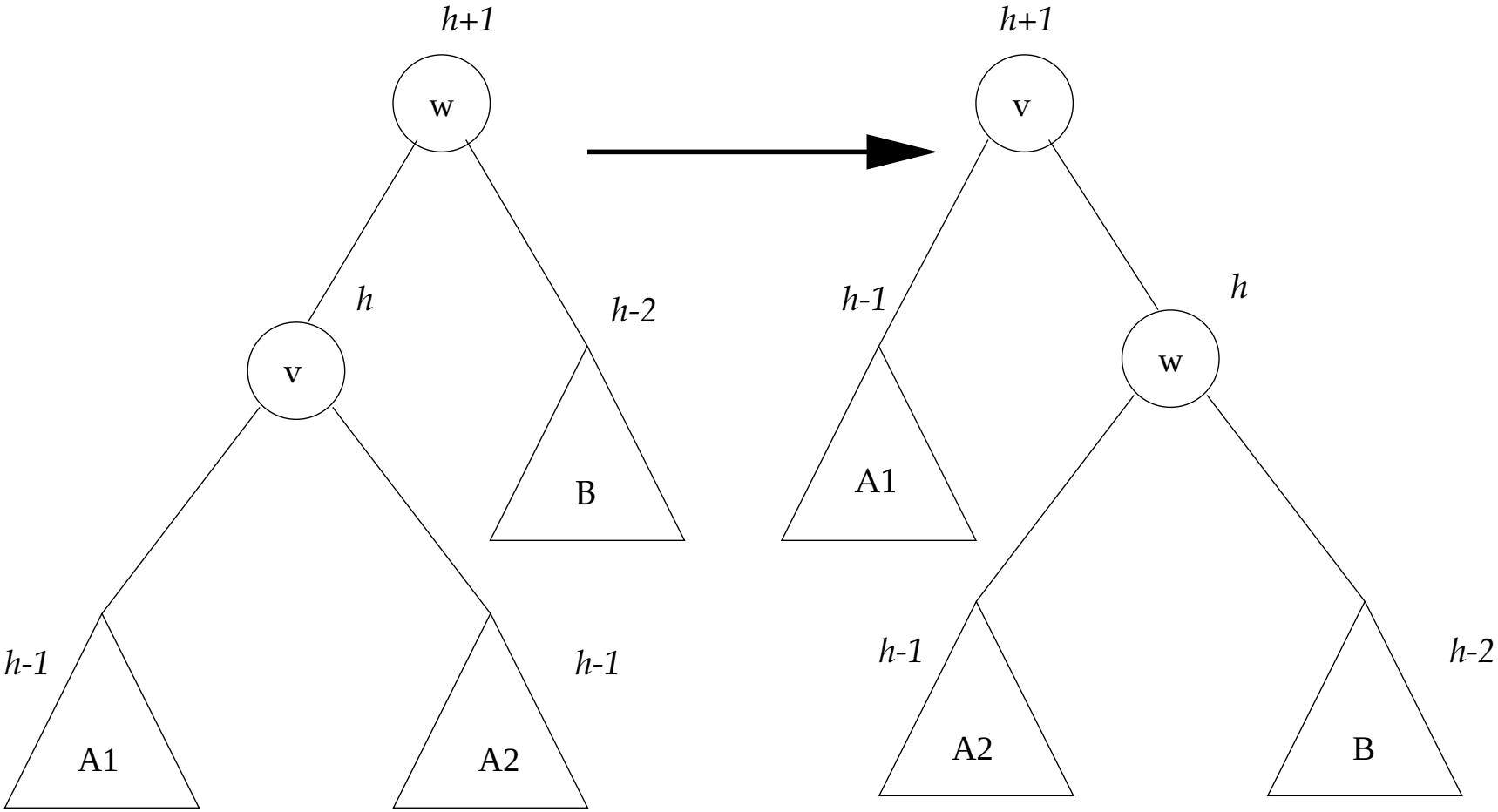
Bemerkungen:

- Durch die LR-Rotation hat sich die Höhe des Unterbaums $\mathfrak{U}$ verringert.
- Die Höhenänderung kann sich zur Wurzel hin fortpflanzen.
- Eventuell sind bei Vorfahren von w weitere Rotationen notwendig.

Fall 3c:

- Im Unterbaum A sind der rechte und der linke Unterbaum gleich hoch.
- Dieser Fall kann nur beim Löschen auftreten.
- Ausgleich durch eine LL-Rotation.

Veranschaulichung:



Bemerkungen:

- Durch die Rotation wird die Höhe des Unterbaums $\mathfrak{U}$ nicht verändert.
- Damit wird durch die LL-Rotation die Änderung absorbiert, d.h. auf höherer Ebene ist keine Rebalancierung mehr notwendig.

Löschen: Fazit

- Es wurden wiederum nur die Fälle mit $h(A) \geq h(B)$ betrachtet.
- Die anderen Fälle sind zu den hier behandelten Fällen symmetrisch.

- Durch Rotationen kann das AVL-Kriterium nach dem Löschen wieder hergestellt werden.
- Aufwand: $O(\log n)$, da
  - jede Rotation in $O(1)$ durchgeführt werden kann und
  - maximal $O(\log n)$ Rotationen notwendig sind.

Fazit für AVL-Bäume

- Mit AVL-Bäumen können geordnete Mengen effizient verwaltet werden.
- Suchen, Einfügen und Löschen ist in Zeit $O(\log n)$ möglich.
- AVL-Bäume haben für die Darstellung von geordneten Mengen im Arbeitsspeicher eine große praktische Bedeutung.
- Für Peripheriespeicher sind sie aber weniger geeignet.