AVL-Bäume*

Hans Jürgen Ohlbach

18. April 2018

Keywords: Suchbäume, AVL-Bäume

Empfohlene Vorkenntnisse: Bäume, siehe Baeume.pdf

Inhaltsverzeichnis

*Dieser Text ist Teil einer Sammlung von Miniskripten zur Einführung in die Informatik. Er ist in erster Linien für Nichtinformatiker gedacht, kann aber natürlich auch als erste Einführung für Informatiker nützlich sein.

1 Definition von AVL-Bäumen

AVL-Bäume sind nach deren Erfindern, Georgi Maximowitsch Adelson-Velski und Jewgeni Michailowitsch Landis, benannte ***binäre Suchbäume***. Sie genügen dem ***AVL-Kriterium***, welches besagt:

Die Pfadlängen bis zu den Blattknoten dürfen sich nur um maximal 1 unterscheiden.

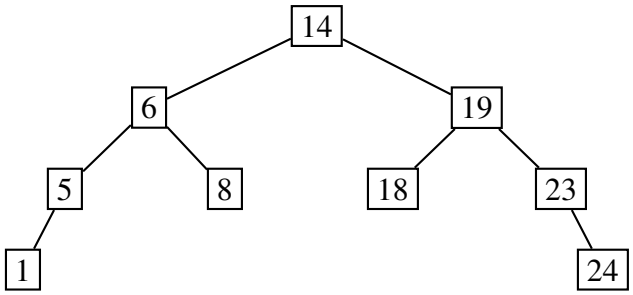
Das bedeutet im Einzelnen:

Binärbäume: Jeder Knoten hat maximal zwei Kindknoten.

Suchbäume:
- Als Suchbäume sind die Knotenlabel (Schlüssel) aus einer Menge mit einer Totalordnung.
- Damit binäre Suche funktioniert, muss für einen Knoten mit Schlüssel n,
  die *linken* Unterknoten kleinere Schlüssel haben als n,
  und die *rechten* Unterknoten größere Schlüssel als n haben.

Balanciertheit: Wegen der AVL-Bedingung sind die Bäume *balanciert*,
d.h. alle inneren Knoten bis auf die in der untersten Ebene haben zwei Kindknoten.
Das bedeutet dann auch insbesondere, dass sie bei n Knoten eine Tiefe von $\mathcal{O}(\log(n))$ haben.

Als Beispiel zeigen wir einen AVL-Baum für die Zahlen 23,1,6,19,14,18,8,24,15.
Der Baum sieht so aus:



AVL-Baum

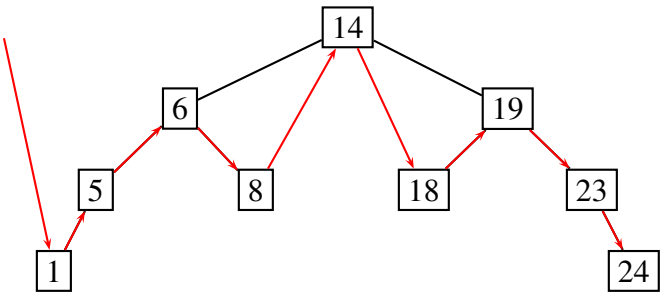
Links sind immer die kleineren Zahlen, und rechts die größeren.

Der große Vorteil von AVL-Bäumen gegenüber binärer Suche in sortierten Arrays ist, dass man mit wenig, d.h. $\mathcal{O}(\log(n))$ Aufwand Schlüssel einfügen und löschen kann. Wenn Einfügen und Löschen sehr oft vorkommt, sind AVL-Bäume den Arrays vorzuziehen.

2 Traversieren eines AVL-Baums

Ein binärer Suchbaum dient ja in erster Linie dazu, eine sortierte Reihenfolge von Schlüsseln zu verwalten. Daher muss es auch möglich sein, diese sortierte Reihenfolge durchzulaufen.

Die Reihenfolge **Depth-First In-Order** (links-Knoten-rechts) erzeugt dabei genau die *sortierte* Folge der Schlüssel.



Reihenfolge: 1,5,6,8,14,18,19,23,24

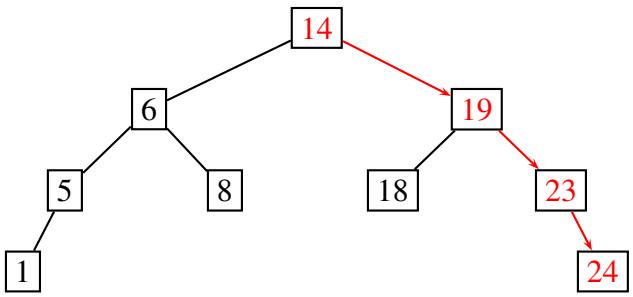
Traversieren des AVL-Baums

3 Suche im binären Suchbaum

Die Suche nach einem bestimmten Schlüssel funktioniert in allen binären Suchbäumen gleich:

- Man testet zunächst, ob der Wurzelknoten den gesuchten Wert als Schlüssel hat,
- Falls nicht, vergleicht man den Wert mit dem Schlüssel des Wurzelknotens.
  Falls der Wert kleiner ist, sucht man im linken Teilbaum weiter, ansonsten im rechten Teilbaum.
- Das wiederholt man rekursiv mit den Wurzelknoten des entsprechenden Teilbaums,
  bis man entweder den Schlüssel gefunden hat, oder bis man am Blattknoten angekommen ist.

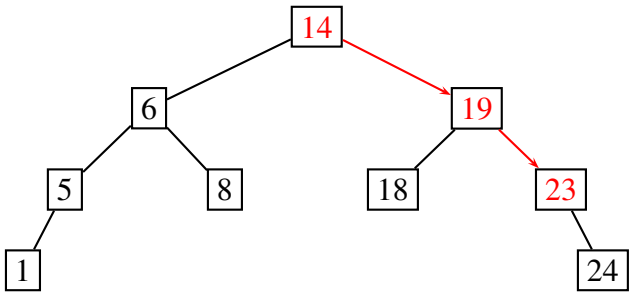
Als Beispiel zeigen wir die Suche nach dem Schlüssel 24. Der rote Pfad zeigt den direkten Weg zum richtigen Knoten, der ohne Umwege gefunden wird.



Erfolgreiche binäre Suche nach 24

Da die Baumtiefe für n Knoten $\mathcal{O}(\log(n))$ ist, ist die Suche nach $\mathcal{O}(\log(n))$ Zeit fertig.

Als nächstes suchen wir nach dem Schlüssel 20. In diesem Fall stoppt die Suche bei 23 weil es dort keinen passenden Kindknoten mehr gibt.



Erfolglose binäre Suche nach 20

Sucht man stattdessen nach dem Schlüssel 25, dann stoppt die Suche am Knoten 24 weil es dort keinen Kindknoten mehr gibt.

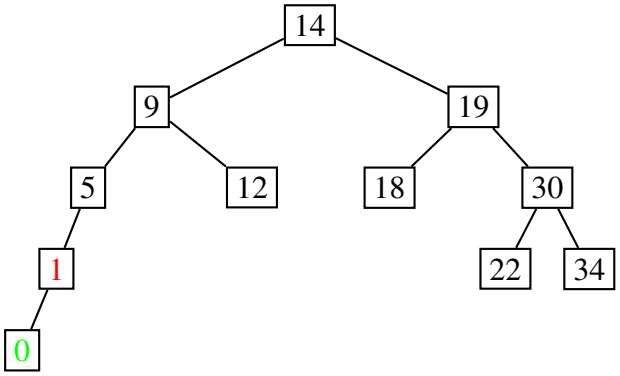
4 Einfügen eines neuen Schlüssels

Das Einfügen eines neuen Schlüssels in einen AVL-Baum ist deutlich komplizierter als z.B. bei einem Heap. Es besteht aus drei Schritten:

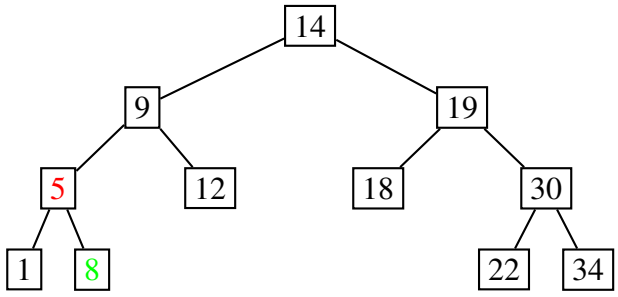
1. den Einfügepunkt finden,
2. den neuen Schlüssel einhängen,
3. den Baum ausbalancieren (rebalancing), falls die AVL-Bedingung verletzt ist.

Die ersten beiden Schritte sind leicht zu realisieren. Man versuchte, mit dem binären Suchverfahren den Knoten mit dem passenden Schlüssel zu finden. Wenn es den Schlüssel noch nicht gibt, endet das Suchverfahren an dem Knoten, wo es nicht mehr weitergeht. Dies ist der *Einfügepunkt*. Jetzt wird der neue Schlüssel als neuer Blattknoten unter den Einfügepunkt gehängt.

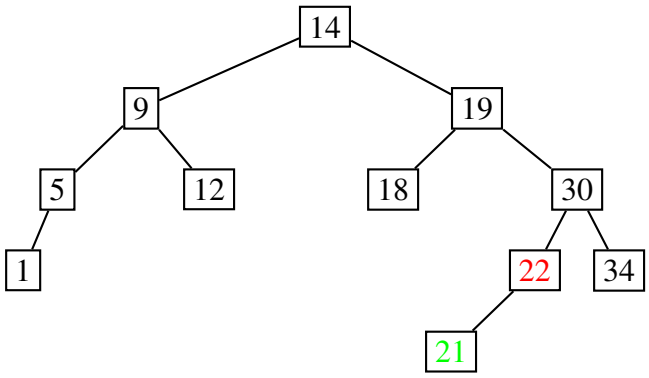
Die folgenden Beispiele illustrieren das Verfahren.
Rot ist der Einfügepunkt, und grün ist der neue Blattknoten.



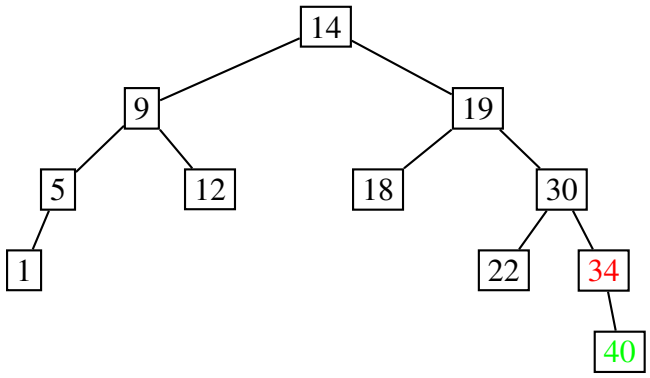
Einfügen von 0



Einfügen von 8



Einfügen von 21



Einfügen von 40

Dummerweise kann jetzt die AVL-Bedingung verletzt werden, dass sich nämlich die Pfadlängen zu den Blattknoten nur um 1 unterscheiden dürfen. Beim Einfügen von 0, 21 und 40 in den Beispielen oben ist das passiert.

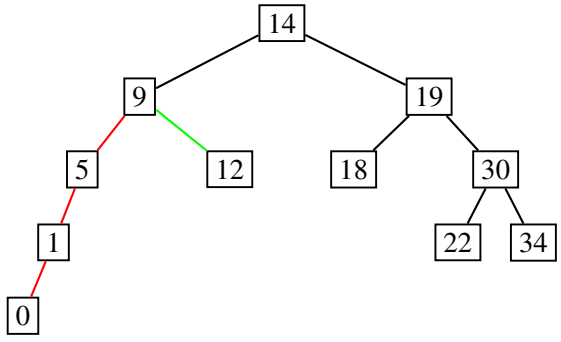
Jetzt muss der Baum umstrukturiert werden, damit die AVL-Bedingung wieder gilt.

5 Rebalancing

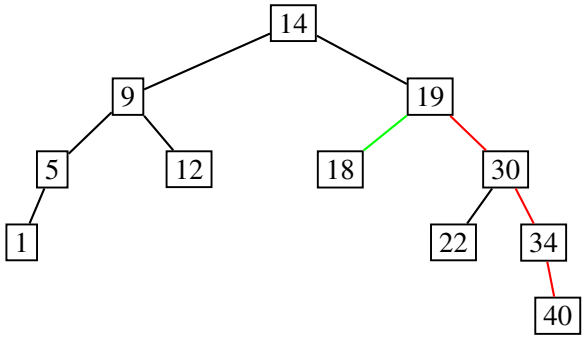
Nach dem Einfügen eines Knoten können sich die Pfadlängen um höchstens 2 unterscheiden. Es gibt dabei zwei verschiedene Situationen, die unterschiedliche Vorgehensweisen erfordern.

5.1 Einfachrotation

Diese Operation wird angewendet, wenn *ein äußerer Pfad relativ zu einem inneren Pfad* zu lang ist, wie in den Bäumen unten:

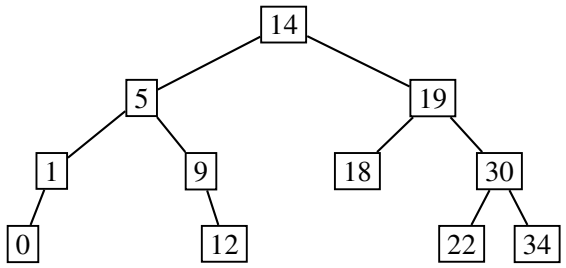


linker Pfad zu lang

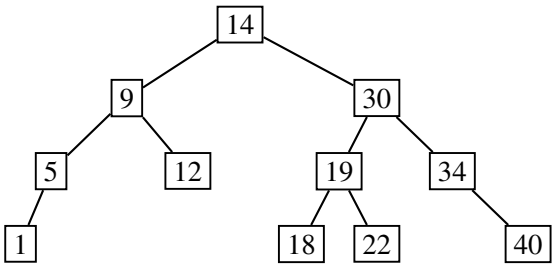


rechter Pfad zu lang

In diesem Fall muss man den langen Pfad verkürzen, indem man den langen Pfad zu dem kürzeren hin *rotiert*. Man bezeichnet es als *Links-Rotation* bzw. *Rechts-Rotation*. Bei den beiden Bäumen oben wirkt sich das so aus:



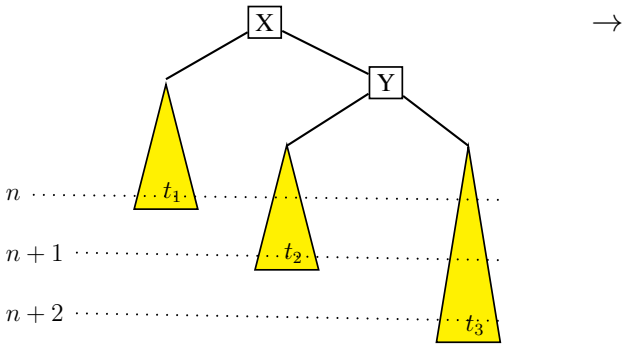
Rechts-Rotation



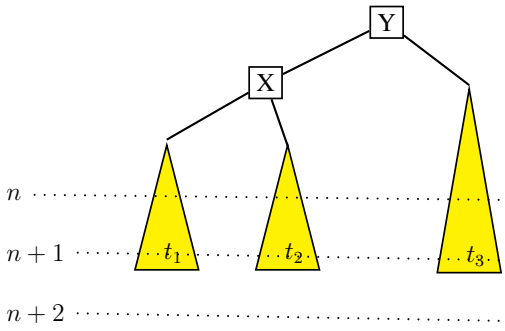
Links-Rotation

Bei der Rechts-Rotation ist die 5 hoch gewandert, und die 9 rechts herunter gewandert.
Bei der Links-Rotation ist die 30 hoch gewandert, und die 19 links herunter gewandert.

Das allgemeine Schema für eine **Links-Rotation** sieht so aus:
X rutscht nach unten, Y nach oben



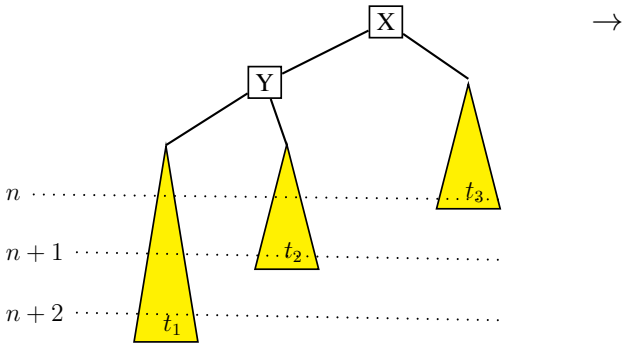
$\rightarrow$



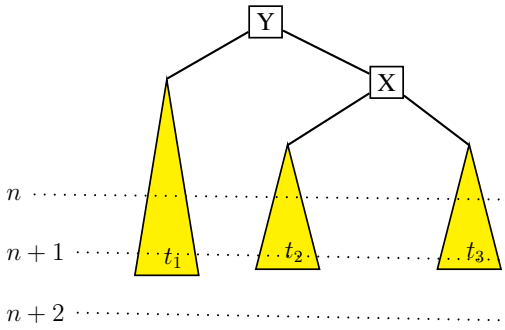
t_2 darf dabei auch bis zur Ebene $n+2$ reichen.

Die Rechts-Rotation ist analog, nur gespiegelt.

Das allgemeine Schema für eine **Rechts-Rotation** sieht so aus:
X rutscht nach unten, Y nach oben

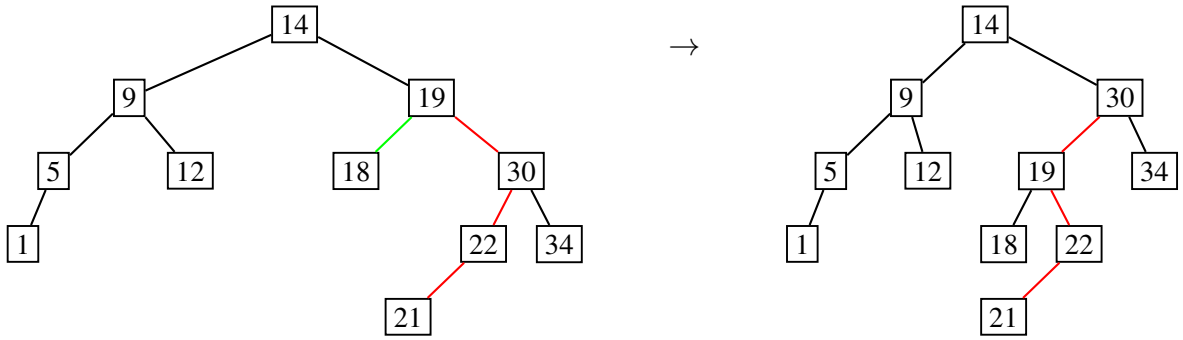


$\rightarrow$



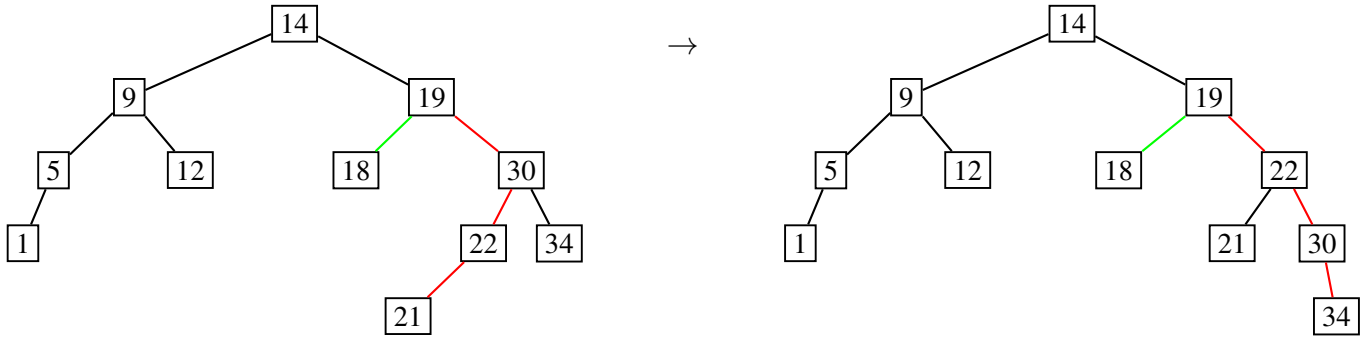
5.2 Doppelrotation

Leider funktioniert die Einfachrotation nicht wenn der *innere Pfad* zu lang ist Wenn man Einfachrotation versucht, bleibt der Pfad immer noch zu lang.



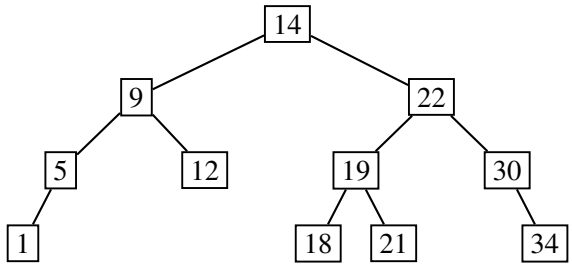
innerer Pfad zu lang — Einfachrotation, Pfad noch zu lang

Stattdessen verschiebt man den zu langen Pfad nach außen, um dann erst die Einfachrotation anzuwenden.



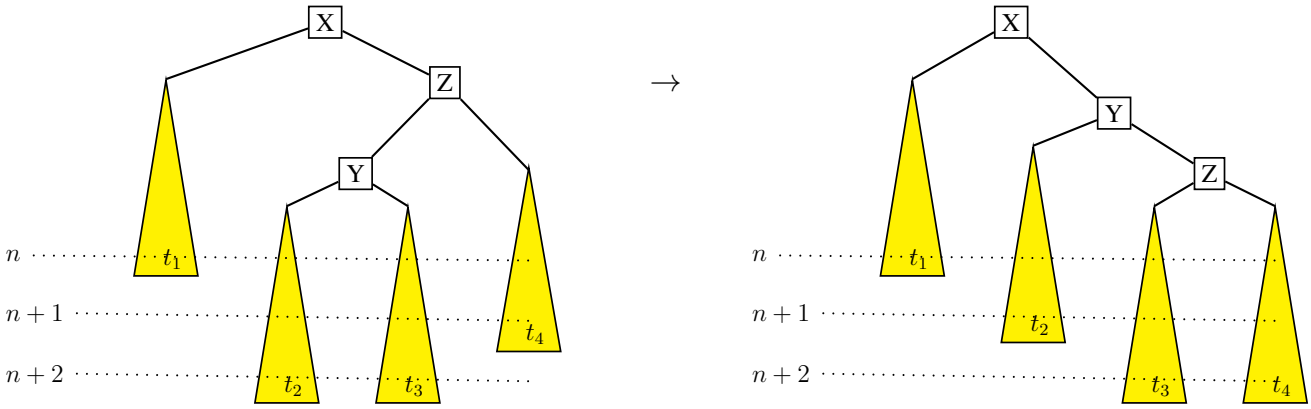
innerer Pfad zu lang — Vorbereitet für Einfachrotation

Jetzt hat man die Struktur, für die die Einfachrotation passt:

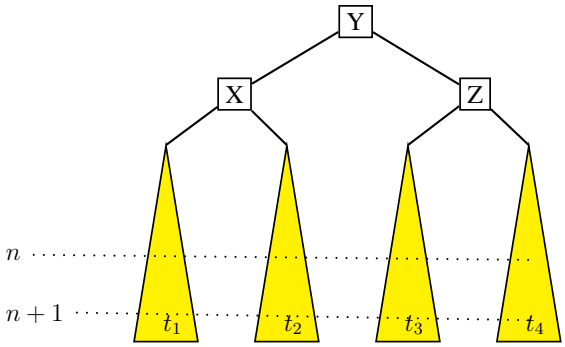


ausbalanciert

Das allgemeine Schema für die Doppelrotation (von rechts nach links) sieht dann so aus:



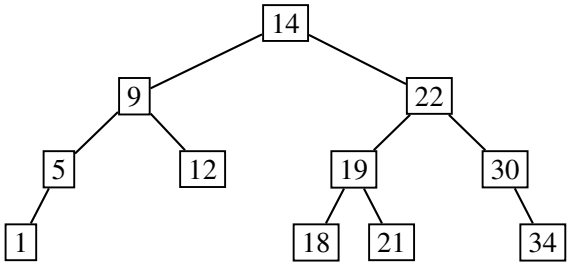
Die nachfolgende Einfachrotation balanciert jetzt den Baum aus:



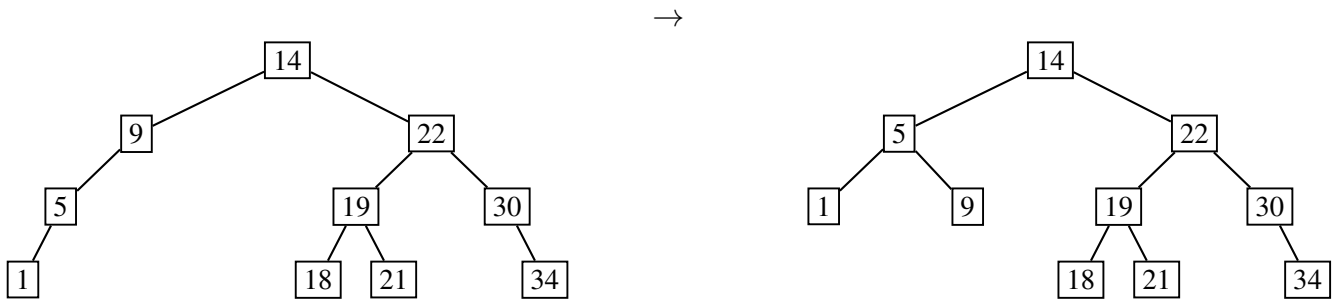
Ganz analog macht man die Doppelrotation von links nach rechts.

6 Löschen eines Schlüssels

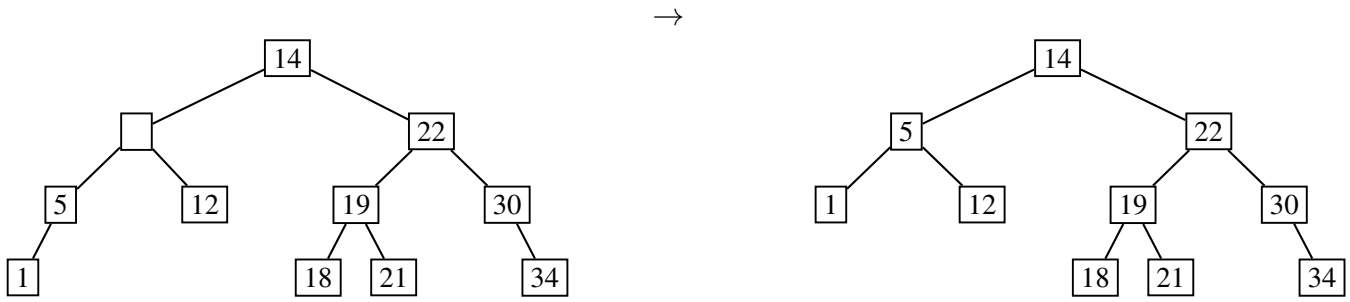
Bei Löschen eines Schlüssels sind verschiedene Fälle zu unterscheiden. Wir illustrieren typische Fälle an folgendem Beispiel:



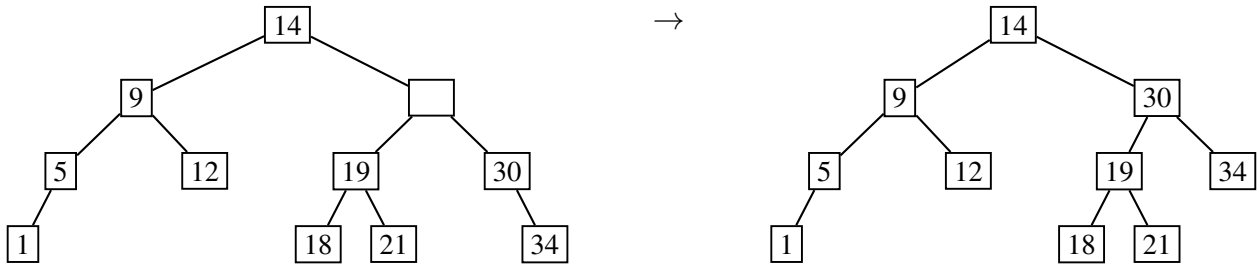
Untere Blattknoten wie 1, 18, 21 und 34 können einfach entfernt werden.
Die Entfernung eines höherer Blattknotens wie die 12 erfordert, den Baum zu rebalancieren:



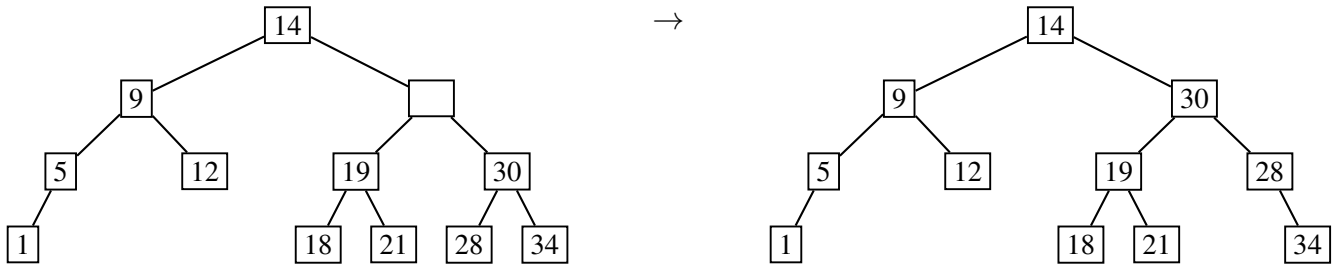
Innere Knoten zu entfernen, ist schon etwas trickreicher. Dann müssen untere Knoten nach oben rücken. Wenn in unserem Beispielbaum z.B. die 9 entfernt wird, dann muss die 5 nachrücken:



Wird die 22 entfernt, dann kann die 30 nachrücken:



Das war einfach, weil die 30 nur einen rechten Kindknoten hatte. Wenn aber beide Kindknoten des zu entfernenden Knotens selbst zwei Kindknoten haben, dann muss einer davon ausgewählt werden, und dessen Kindknoten müssen umstrukturiert werden.



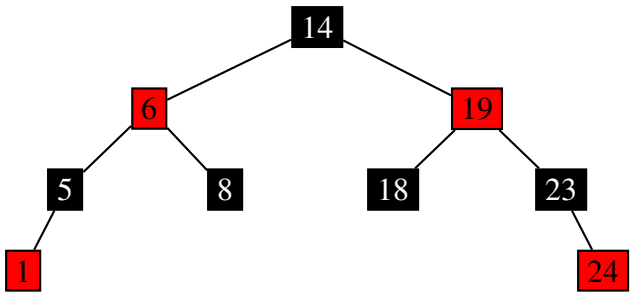
Die Beispiele sollen nur einen ungefähren Eindruck geben, wie die Löschverfahren funktionieren. Exakte Beschreibungen findet man in der Fachliteratur.

7 Zusammenfassung und Ausblick

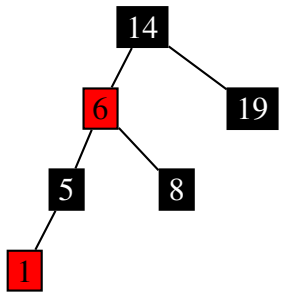
AVL-Bäume haben eine Reihe von Vorteilen:

- Suchen, Einfügen und Löschen eines Knotens gehen mit $\mathcal{O}(\log(n))$ Aufwand.
- Auch den kleinsten und größten Schlüssel kann man mit $\mathcal{O}(\log(n))$ Aufwand bestimmen. Man läuft immer den linkesten/rechtesten Pfad entlang.
- Das Traversieren des Baums, um die sortierte Reihenfolge abzulaufen, geht mit einem einfachen rekursiven Programm zur Tiefensuche.
- Selbst Operationen auf ganzen AVL-Bäumen, wie das Verketten zweier Bäume und das Spalten eines Baumes geht mit $\mathcal{O}(\log(n))$ Aufwand (in diesem Miniskript nicht thematisiert).

Eine Alternative zu den AVL-Bäumen sind die ***Rot-Schwarz-Bäume***. Sie sind ebenfalls binäre Suchbäume, wobei die Balanciertheit etwas anders hergestellt wird als bei AVL-Bäumen. Sie heißen deswegen Rot-Schwarz-Bäume, weil die Knoten unterschiedlich klassifiziert sind, was man z.B. als Farben rot und schwarz ausdrücken kann, wie unten gezeigt:



Rot-Schwarz-Baum



Rot-Schwarz-Baum

Jeder AVL-Baumm kann als Rot-Schwarz-Baum dargestellt werden, aber nicht jeder Rot-Schwarz-Baum entspricht einem AVL-Baum. Der rechte Baum ist ein Beispiel dafür. Laufzeitmessungen haben gezeigt, dass beide Versionen ähnliche Performanz haben, wobei die AVL-Bäume leicht besser waren. Daher wird für die Details von Rot-Schwarz-Bäumen auf die Fachliteratur verwiesen.

Eine etwas andere Art von Suchbäumen sind die B-Bäume, die insbesondere für Datenbankanwendungen genutzt werden. Bei diesen tragen die inneren Knoten mehr als einen Schlüssel, und haben auch mehr als zwei Kindknoten. Sie werden im Miniskript BBaeume.pdf beschrieben.