AVL-Bäume

Weiterführende Literatur:

- *Wikipedia-Artikel „AVL-Baum“*
- Saake und Sattler, *Algorithmen und Datenstrukturen*, Kapitel 14.4.2, Seite 378-386 (PDF 394-402)
- Halim, *VisuAlgo*, bst

Visualisierungstools

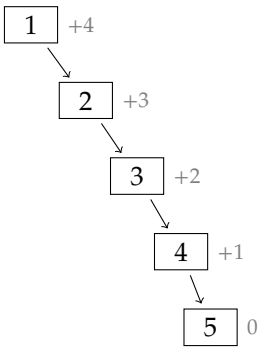
- `https://www.cs.usfca.edu/~galles/visualization/AVLtree.html`
- `https://visualgo.net/bn/bst` (oben in den Tabs umschalten auf AVL)

Ein AVL-Baum ist ein binärer Suchbaum, der *höhenbalanciert* ist, d. h. für jeden Knoten gilt: höhenbalanciert

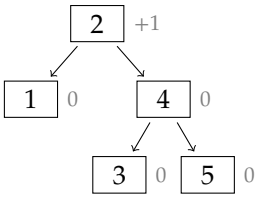
$$|h_{\text{rechterTeilbaum}} - h_{\text{linkerTeilbaum}}| \leq 1$$

Die *„Entartung“* des Baums wird so vermieden. Die Höhe eines AVL-Baums ist $h \in \mathcal{O}(\log n)$. Beim Einfügen und Löschen von Knoten muss die AVL-Eigenschaft durch *Rotationen* wiederhergestellt werden.[1] „Entartung“ Rotationen

Binärer Suchbaum



AVL-Baum



Rotationsregeln

$$b_{\text{Balance-Faktor}} = h_{\text{rechter Teilbaum}} - h_{\text{linker Teilbaum}}$$

Erklärungen

bO Balance-Faktor des „oberen“ Wurzelknotens

bU Balance-Faktor des Kindknoten von *bO*

– Rechtsdrehung

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5*, Seite 22 (PDF 16).

$+$ Linksdrehung

$bU = -1, bO = -2$	Rechtsrotation (Rechtsdrehung um bO)
$bU = +1, bO = +2$	Linksrotation (Linksdrehung um bO)
$bU = +1, bO = -2$	Links-Rechts-Rotation (Linksdrehung um bU, dann Rechtsdrehung um bO)
$bU = -1, bO = +2$	Rechts-Links-Rotation (Rechtsdrehung um bU, dann Linksdrehung um bO)

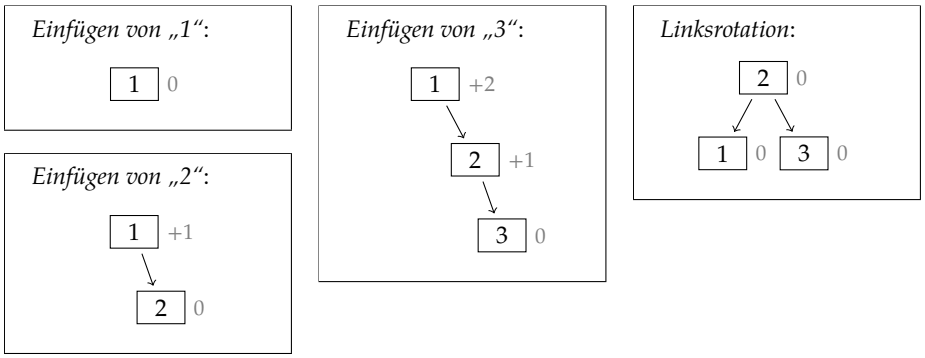
Die Kindknoten, die nach der Drehung „im Weg stehen“, werden „umgeklappt“, also vom linken zum rechten Kind der nächsten Ebene „umgehängt“ und umgekehrt.

Rotation (genaue Darstellung): Linksrotation von k.

	vorher	nachher
Wurzel	k	k.links
linkes Kind der Wurzel	k.links	k.links.links
rechte Kind der Wurzel	k.rechts	k
li. Ki. des li. Ki. der Wurzel	k.links.links	k.links.links.links
re. Ki. des li. Ki. der Wurzel	k.links.rechts	k.links.links.rechts
li. Ki. des re. Ki. der Wurzel	k.rechts.left	k.links.rechts
re. Ki. des re. Ki. der Wurzel	k.rechts.rechts	k.rechts

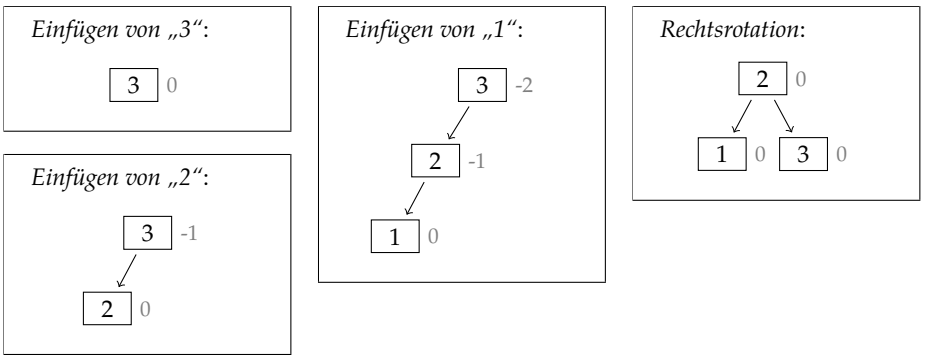
Die „nachher“-Position bezieht sich immer auf den den neuen Wurzelknoten, der Pfad zu dem entsprechenden Knoten beginnt immer beim alten Wurzelknoten k. [2]

Linksrotation 1 2 3

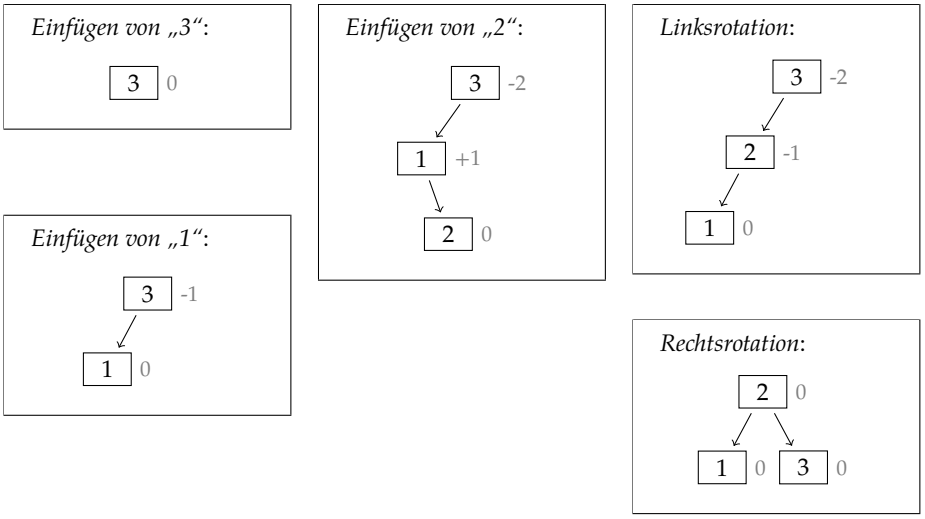


Rechtsrotation 3 2 1

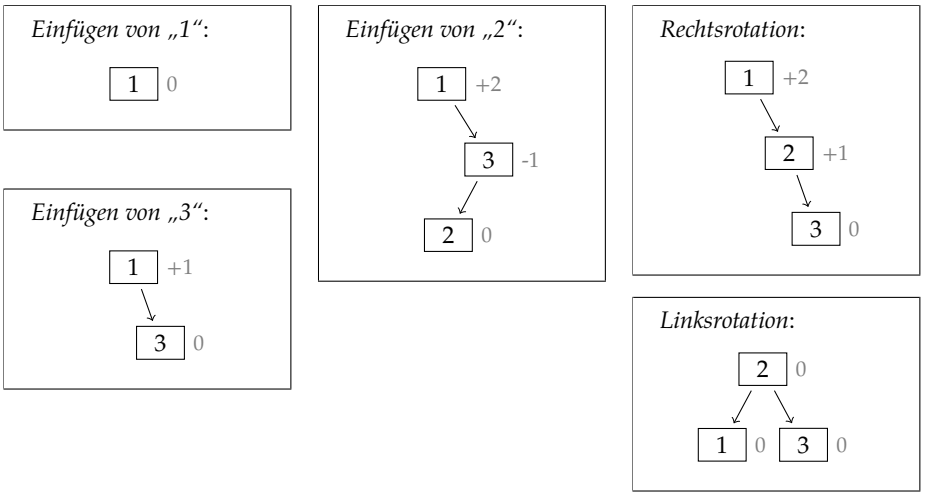
[2] hs-mannheim.de



Links-Rechtsrotation 3 1 2



Rechts-Linksrotation 1 3 2



Löschen

Löschalgorithmus der binären Suchbäume

rebalanciert

Um einen Schlüsselwert in einem AVL-Baum zu löschen, wird der *Löschalgorithmus der binären Suchbäume* angewendet. Wie beim Einfügen in AVL-Bäumen muss der Suchbaum nun gegebenenfalls *rebalanciert* werden. Dazu wird der Weg vom Elternknoten des gelöschten Knotens zur Wurzel zurückgelaufen und die Balance jedes Knotens überprüft. Wenn der gelöschte Schlüsselwert in einem Knoten mit zwei Nachfolgern gespeichert war, muss bei Elternknoten des symmetrischen Nachfolgers begonnen werden. [3]

Laufzeit bzw. Komplexität

	Average-Case	Worst-Case
Suchen	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$

[4]

Rechtsrotation

```
85          reporter.berichteBaum(this, "Nach der Linksrotation", 1);
86          knoten = rotiereLinks(knoten);
87        }
88      } else if (balance < -1) {
89        if (gibHöhe(knoten.links.links) > gibHöhe(knoten.links.rechts)) {
90          reporter.berichteBaum(this, "Nach der Rechtsrotation", 1);
91          knoten = rotiereRechts(knoten);
92        } else {
93          reporter.berichteBaum(this, "Nach der Linksrotation", 1);
94          knoten.links = rotiereLinks(knoten.links);
95          reporter.berichteBaum(this, "Nach der Rechtsrotation", 1);
96          knoten = rotiereRechts(knoten);
97        }
98      }
99      return knoten;
100   }
101
102   /**
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/baum/AVLBaum.java`

Linksrotation

```
104     *
105     * @param knoten Knoten mit einem Balancefaktor von -2.
106     *
107     * @return Der linke Kindknoten des gegebenen Eingangsknoten mit dem
108     *         Balancefaktor -1.
109     */
110   private AVLBaumKnoten rotiereRechts(AVLBaumKnoten knoten) {
111     // Linker Knoten mit Balancefaktor -1
112     AVLBaumKnoten links = knoten.gibLinks();
113     AVLBaumKnoten rechtsVonLinks = links.gibRechts();
```

[3] Universität Würzburg

[4] studyflix.de

```
114       links.rechts = knoten;
115       knoten.links = rechtsVonLinks;
116       aktualisiereHöhe(knoten);
117       aktualisiereHöhe(links);
118       return links;
119     }
120
121     /**
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/baum/AVLBaum.java`

Die Klasse AVLBaum

```
3   /**
4    * Eine Implementation eines AVL-Baums. (Nach
5    * <a href="https://www.baeldung.com/java-avl-trees">baeldung.com</a> bzw.
6    * <a href=
7    * "https://github.com/eugenp/tutorials/blob/master/data-
     ↪  structures/src/main/java/com/baeldung/avltree/AVLTree.java">Repository
8    * auf Gibhub</a>)
9    */
10  @SuppressWarnings({ "rawtypes" })
11  public class AVLBaum extends BinaerBaum {
12
13    private AVLBaumKnoten kopf;
14
15    /**
16     * Um bei der entfernen-Methode einen boolschen Rückgabewert zu haben. Der
17     * binäre Suchbaum gibt auch wahr oder falsch zurück.
18     */
19    private boolean gelöscht;
20
21    private int gibHöhe(AVLBaumKnoten knoten) {
22      return knoten == null ? -1 : knoten.höhe;
23    }
24
25    public int gibHöhe() {
26      return gibKopf() == null ? -1 : kopf.höhe;
27    }
28
29    /**
30     * {@inheritDoc}
31     */
32    public AVLBaumKnoten gibKopf() {
33      return kopf;
34    }
35
36    /**
37     * Gib den Balancefaktor des gegebenen Knotens.
38     *
39     * @param knoten Der Knoten, dessen Balancefaktor ausgegeben werden soll.
40     *
41     * @return Der Balancefaktor des Knotens.
42     */
43    public int gibBalance(AVLBaumKnoten knoten) {
44      return (knoten == null) ? 0 : gibHöhe(knoten.rechts) - gibHöhe(knoten.links);
45    }
46
47    /**
48     * Gib den Balancefaktor des gegebenen Knotens als Text (String). Positive
49     * Balancefaktoren haben ein Plus als Präfix.
```

```
50       *
51       * @param knoten Der Knoten, dessen Balancefaktor ausgegeben werden soll.
52       *
53       * @return Der Balancefaktor des Knotens als Text.
54       */
55      public String gibBalanceText(AVLBaumKnoten knoten) {
56        int balance = gibBalance(knoten);
57        if (balance > 0) {
58          return "+" + balance;
59        } else {
60          return "" + balance;
61        }
62      }
63
64      public AVLBaumKnoten finde(Comparable schlüssel) {
65        AVLBaumKnoten aktuellerKnoten = kopf;
66        while (aktuellerKnoten != null) {
67          if (aktuellerKnoten.schlüssel == schlüssel) {
68            break;
69          }
70          aktuellerKnoten = aktuellerKnoten.vergleiche(schlüssel) < 0 ?
            ↪  aktuellerKnoten.rechts : aktuellerKnoten.links;
71        }
72        return aktuellerKnoten;
73      }
74
75      private AVLBaumKnoten rebalanciere(AVLBaumKnoten knoten) {
76        aktualisiereHöhe(knoten);
77        int balance = gibBalance(knoten);
78        if (balance > 1) {
79          if (gibHöhe(knoten.rechts.rechts) > gibHöhe(knoten.rechts.links)) {
80            reporter.berichteBaum(this, "Nach der Linksrotation", 1);
81            knoten = rotiereLinks(knoten);
82          } else {
83            reporter.berichteBaum(this, "Nach der Rechtsrotation", 1);
84            knoten.rechts = rotiereRechts(knoten.rechts);
85            reporter.berichteBaum(this, "Nach der Linksrotation", 1);
86            knoten = rotiereLinks(knoten);
87          }
88        } else if (balance < -1) {
89          if (gibHöhe(knoten.links.links) > gibHöhe(knoten.links.rechts)) {
90            reporter.berichteBaum(this, "Nach der Rechtsrotation", 1);
91            knoten = rotiereRechts(knoten);
92          } else {
93            reporter.berichteBaum(this, "Nach der Linksrotation", 1);
94            knoten.links = rotiereLinks(knoten.links);
95            reporter.berichteBaum(this, "Nach der Rechtsrotation", 1);
96            knoten = rotiereRechts(knoten);
97          }
98        }
99        return knoten;
100     }
101
102     /**
103      * Führe eine Rechtsrotation durch.
104      *
105      * @param knoten Knoten mit einem Balancefaktor von -2.
106      *
107      * @return Der linke Kindknoten des gegebenen Eingangsknoten mit dem
108      *         Balancefaktor -1.
109      */
110     private AVLBaumKnoten rotiereRechts(AVLBaumKnoten knoten) {
```

```
111       // Linker Knoten mit Balancefaktor -1
112       AVLBaumKnoten links = knoten.gibLinks();
113       AVLBaumKnoten rechtsVonLinks = links.gibRechts();
114       links.rechts = knoten;
115       knoten.links = rechtsVonLinks;
116       aktualisiereHöhe(knoten);
117       aktualisiereHöhe(links);
118       return links;
119     }
120
121     /**
122      * Führe eine Linksrotation durch.
123      *
124      * @param knoten Knoten mit einem Balancefaktor von +2.
125      *
126      * @return Der rechte Kindknoten des gegebenen Eingangsknoten mit dem
127      *         Balancefaktor von +1.
128      */
129     private AVLBaumKnoten rotiereLinks(AVLBaumKnoten knoten) {
130       // Rechter Knoten mit Balancefaktor +1
131       AVLBaumKnoten rechts = knoten.gibRechts();
132       AVLBaumKnoten linksVonRechts = rechts.gibLinks();
133       rechts.links = knoten;
134       knoten.rechts = linksVonRechts;
135       aktualisiereHöhe(knoten);
136       aktualisiereHöhe(rechts);
137       return rechts;
138     }
139
140     private void aktualisiereHöhe(AVLBaumKnoten knoten) {
141       knoten.höhe = 1 + Math.max(gibHöhe(knoten.links), gibHöhe(knoten.rechts));
142     }
143
144     private AVLBaumKnoten fügeEin(AVLBaumKnoten knoten, Comparable schlüssel) {
145       if (knoten == null) {
146         return new AVLBaumKnoten(schlüssel);
147       } else if (knoten.vergleiche(schlüssel) > 0) {
148         knoten.links = fügeEin(knoten.links, schlüssel);
149       } else if (knoten.vergleiche(schlüssel) < 0) {
150         knoten.rechts = fügeEin(knoten.rechts, schlüssel);
151       } else {
152         throw new RuntimeException("duplicate Key!");
153       }
154       return rebalanciere(knoten);
155     }
156
157     /**
158      * {@inheritDoc}
159      */
160     public boolean fügeEin(Comparable schlüssel) {
161       reporter.berichteÜberschrift("Nach Einfügen von „" + schlüssel + """, 0);
162
163       kopf = fügeEin(kopf, schlüssel);
164       reporter.berichteBaum(this, 0);
165       return true;
166     }
167
168     /**
169      * Gib das äußerste Kind eines Knoten.
170      *
171      * @param knoten   Der aktuelle Knoten.
172      * @param richtung „links": der linkesten Knoten des (Teil-)Baums oder
    ↪  „rechts":
```

```
173     *                der rechtesten Knoten des linkesten Knoten des (Teil-)Baums.
174     *
175     * @return Das äußerste Kind eines Knoten.
176     */
177    private AVLBaumKnoten gibÄußerstesKind(AVLBaumKnoten knoten, String richtung) {
178      AVLBaumKnoten aktuellerKnoten = knoten;
179      if (richtung.equals("links")) {
180        while (aktuellerKnoten.links != null) {
181          aktuellerKnoten = aktuellerKnoten.links;
182        }
183      } else {
184        while (aktuellerKnoten.rechts != null) {
185          aktuellerKnoten = aktuellerKnoten.rechts;
186        }
187      }
188      return aktuellerKnoten;
189    }
190
191    /**
192     * Entferne einen Knoten.
193     *
194     * @param knoten    Der aktuelle Knoten.
195     * @param schlüssel Der Schlüssel, der gelöscht werden soll.
196     * @param neuerKopf „links": rechtesten Knoten des linken Kindbaums oder
197     *                  „rechts": den linkesten Knoten des rechten Kindbaums.
198     *
199     * @return Den aktuellen Kopf-Knoten des Baums.
200     */
201    private AVLBaumKnoten entferne(AVLBaumKnoten knoten, Comparable schlüssel,
       ↪  String neuerKopf) {
202      if (knoten == null) {
203        return knoten;
204      } else if (knoten.vergleiche(schlüssel) > 0) {
205        knoten.links = entferne(knoten.links, schlüssel, neuerKopf);
206      } else if (knoten.vergleiche(schlüssel) < 0) {
207        knoten.rechts = entferne(knoten.rechts, schlüssel, neuerKopf);
208      } else {
209        if (knoten.links == null || knoten.rechts == null) {
210          knoten = (knoten.links == null) ? knoten.rechts : knoten.links;
211          gelöscht = true;
212        } else if (neuerKopf.equals("rechts")) {
213          AVLBaumKnoten ganzLinkesKind = gibÄußerstesKind(knoten.rechts, "links");
214          knoten.schlüssel = ganzLinkesKind.schlüssel;
215          knoten.rechts = entferne(knoten.rechts, (Comparable) knoten.schlüssel,
             ↪  "rechts");
216        } else {
217          AVLBaumKnoten ganzRechtesKind = gibÄußerstesKind(knoten.links, "rechts");
218          knoten.schlüssel = ganzRechtesKind.schlüssel;
219          knoten.links = entferne(knoten.links, (Comparable) knoten.schlüssel,
             ↪  "links");
220        }
221      }
222      if (knoten != null) {
223        knoten = rebalanciere(knoten);
224      }
225      return knoten;
226    }
227
228    /**
229     * {@inheritDoc}
230     */
231    public boolean entferne(Comparable schlüssel) {
```

```
232       reporter.berichteÜberschrift("Nach Löschen von „" + schlüssel + """, 0);
233       kopf = entferne(kopf, schlüssel, "rechts");
234       // Wieder auf falsch setzten, damit beim nächsten Löschvorgang der
235       // Wert wieder von neuem gesetzt werden muss.
236       boolean ausgabe = gelöscht;
237       gelöscht = false;
238       reporter.berichteBaum(this, 0);
239       return ausgabe;
240     }
241
242     /**
243      *
244      * @param schlüssel Der Schlüssel, der gelöscht werden soll.
245      * @param neuerKopf „links": rechtesten Knoten des linken Kindbaums oder
246      *                  „rechts": den linkesten Knoten des rechten Kindbaums.
247      */
248     public void entferne(Comparable schlüssel, String neuerKopf) {
249       kopf = entferne(kopf, schlüssel, neuerKopf);
250     }
251
252   }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/baum/AVLBaum.java`

Literatur

[1] Dr. Steven Halim. *VisuAlgo*. `https://visualgo.net/en`. aufgerufen 2020-05-17. National University of Singapore (NUS).

[2] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5. Bäume, Hashing*. `https://www.studon.fau.de/file2619756_download.html`.

[3] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java*. 2014.

[4] *Wikipedia-Artikel „AVL-Baum“*. `https://de.wikipedia.org/wiki/AVL-Baum`.