

Binärer Suchbaum

Weiterführende Literatur:

- Wikipedia-Artikel „Binärbaum“
- Wikipedia-Artikel „Binärer Suchbaum“
- Saake und Sattler, *Algorithmen und Datenstrukturen*, Kapitel 14.2.1, Seite 348-358 (PDF 364-374)
- *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5*, Seite 4-14

Visualisierungstools

- <http://btv.melezinek.cz/binary-search-tree.html> Auch für Postorder etc
- <https://visualgo.net/bn/bst>

Ein binärer Suchbaum - häufig abgekürzt als **BST** (von englisch *Binary Search Tree*) - ist ein binärer Baum, bei dem die Knoten „Schlüssel“ tragen, und die Schlüssel des linken Teilbaums eines Knotens nur *kleiner (oder gleich)* und die des rechten Teilbaums nur *größer*¹ als der Schlüssel des Knotens selbst sind.²

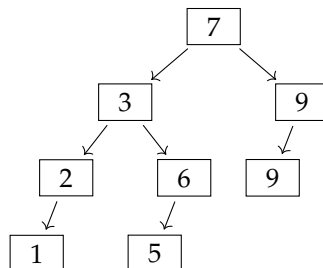
linken Teilbaums
kleiner (oder gleich)
rechten
größer

S	Wert eines Knotens
S_{links}	Wert der Wurzel des linken Teilbaumes
S_{rechts}	Wert der Wurzel des rechten Teilbaumes

Bei einem binären Suchbaum gilt: $S_{\text{links}} \leq S < S_{\text{rechts}}$

Beispiel

Anordnung von 7, 3, 6, 9, 2, 9, 5, 1 in einem int-Baum³



¹Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5, Seite 5.

²Wikipedia-Artikel „Binärer Suchbaum“.

³Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5, Seite 5.

Traversierung eines Baums

Es gibt verschiedene Möglichkeiten einen binären Baum zu traversieren:

- preorder:

besuche die Wurzel, dann den linken Unterbaum, dann den rechten Unterbaum; auch: WLR

```
54  /**
55   * Zuerst wird die Wurzel N betrachtet und anschließend der linke l,
↪ schließlich
56   * der rechte Teilbaum r durchlaufen. (nach Saake Seite 356)
57   *
58   * @param knoten    Der aktuelle Baumknoten, der besucht werden soll.
59   * @param schlüssel Eine Liste, die mit den besuchten Schlüsselwerten
↪ gefüllt
60   *                  wird.
61   */
62  private void besuchePreorder(BaumKnoten knoten, ArrayList<Comparable>
↪ schlüssel) {
63      if (knoten != null) {
64          schlüssel.add((Comparable) knoten.gibSchlüssel());
65          besuchePreorder(knoten.gibLinks(), schlüssel);
66          besuchePreorder(knoten.gibRechts(), schlüssel);
67      }
68  }
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/baum/BinaerBaum.java](https://github.com/orgs/bschlangaul/repositories/tree/main/java/baum/BinaerBaum.java)

- inorder:

besuche den linken Unterbaum, dann die Wurzel, dann den rechten Unterbaum; auch: LWR

```
36  /**
37   * Zuerst wird der linke Teilbaum l durchlaufen, dann die Wurzel N
↪ betrachtet
38   * und schließlich der rechte Teilbaum r durchlaufen. Diese Reihenfolge
39   * entspricht bei binären Suchbäumen der Anordnung der Schlüssel und ist
↪ für die
40   * meisten Anwendungen die gegebene. (nach Saake Seite 356)
41   *
42   * @param knoten    Der aktuelle Baumknoten, der besucht werden soll.
43   * @param schlüssel Eine Liste, die mit den besuchten Schlüsselwerten
↪ gefüllt
44   *                  wird.
45   */
46  private void besucheInorder(BaumKnoten knoten, ArrayList<Comparable>
↪ schlüssel) {
47      if (knoten != null) {
48          besucheInorder(knoten.gibLinks(), schlüssel);
49          schlüssel.add((Comparable) knoten.gibSchlüssel());
50          besucheInorder(knoten.gibRechts(), schlüssel);
51      }
52  }
```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/baum/BinaerBaum.java](https://github.com/orgs/bschlangaul/repositories/tree/main/java/baum/BinaerBaum.java)

- postorder:

besuche den linken Unterbaum, dann den rechten, dann die Wurzel; auch: LRW

```

70  /**
71   * Zuerst wird der linke l, dann der rechte Teilbaum r durchlaufen und
72   * schließlich die Wurzel N betrachtet. (nach Saake Seite 356)
73   *
74   * @param knoten    Der aktuelle Baumknoten, der besucht werden soll.
75   * @param schlüssel Eine Liste, die mit den besuchten Schlüsselwerten
↪   gefüllt
76   *                  wird.
77   */
78  private void besuchePostorder(BaumKnoten knoten, ArrayList<Comparable>
↪   schlüssel) {
79      if (knoten != null) {
80          besuchePostorder(knoten.gibLinks(), schlüssel);
81          besuchePostorder(knoten.gibRechts(), schlüssel);
82          schlüssel.add((Comparable) knoten.gibSchlüssel());
83      }
84  }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/beschlangaul/baum/BinaerBaum.java](https://github.com/beschlangaul/baum/BinaerBaum.java)

- levelorder:

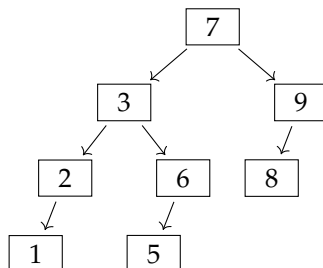
Beginnend bei der Baumwurzel werden die Ebenen von links nach rechts durchlaufen.

```

86  /**
87   * Beginnend bei der Baumwurzel werden die Ebenen von links nach rechts
88   * durchlaufen. (nach Saake Seite 358)
89   *
90   * @param warteschlange Eine Warteschlange aus der die Baumknoten
↪   entnommen
91   *                      werden.
92   * @param schlüssel     Eine Liste, die mit den besuchten Schlüsselwerten
93   *                      gefüllt wird.
94   *
95   * @throws WarteschlangeFehler Ein Warteschlangen-Fehler.
96   */
97  private void besucheLevelorder(Warteschlange warteschlange,
↪   ArrayList<Comparable> schlüssel)
98      throws WarteschlangeFehler {
99      while (!warteschlange.istLeer()) {
100          BaumKnoten knoten = (BaumKnoten) warteschlange.verlasse();
101          if (knoten.gibLinks() != null)
102              warteschlange.betrete(knoten.gibLinks());
103          if (knoten.gibRechts() != null)
104              warteschlange.betrete(knoten.gibRechts());
105          schlüssel.add((Comparable) knoten.gibSchlüssel());
106      }
107  }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/beschlangaul/baum/BinaerBaum.java](https://github.com/beschlangaul/baum/BinaerBaum.java)



preorder	7 3 2 1 6 5 9 8
inorder	1 2 3 5 6 7 8 9
postorder	1 2 5 6 3 8 9 7
levelorder	7 3 9 2 6 8 1 5

Löschen eines Knotens

Die komplizierteste Operation im binären Suchbaum ist das Löschen. Dies liegt daran, dass beim Entfernen eines inneren Knotens einer der beiden Teilbäume des Knotens „hochgezogen“ und gegebenenfalls umgeordnet werden muss.

Grundsätzlich müssen beim Löschen eines Knotens drei Fälle berücksichtigt werden:

Blatt

- (a) Der Knoten ist ein *Blatt*. Dieser Fall ist der einfachste, da hier nur der Elternknoten zu bestimmen ist und dessen Verweis auf den Knoten entfernt werden muss.

einen Kindknoten

- (b) Der Knoten besitzt nur *einen Kindknoten*. In diesem Fall ist der Verweis vom Elternknoten auf den Kindknoten des zu löschenden Knotens umzulenken.

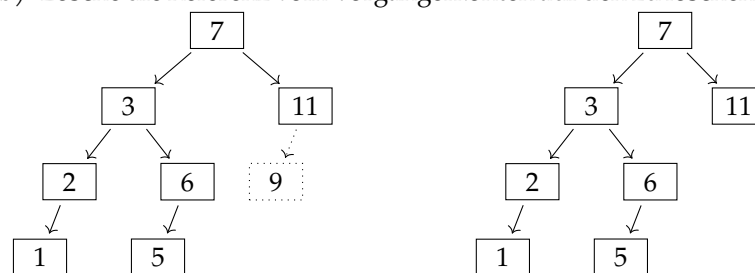
zwei Kindknoten

- (c) Der Knoten ist ein innerer Knoten mit *zwei Kindknoten*. Hierbei muss der Knoten durch den am *weitesten links stehenden Knoten des rechten Teilbaumes* ersetzt werden, da dieser in der Sortierreihenfolge der nächste Knoten ist. Alternativ kann auch der am weitesten rechts stehende Knoten des linken Teilbaumes verwendet werden.⁴

weitesten links stehenden Knoten des rechten Teilbaumes ersetzt

Knoten ohne Nachfolger⁵

- (a) Suche den zu löschenden Knoten
(b) Lösche die Referenz vom Vorgängerknoten auf den zu löschenden Knoten



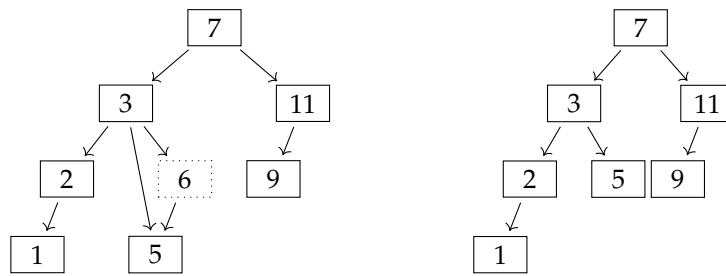
Knoten mit genau einem Nachfolger⁶

- (a) Suche den zu löschenden Knoten
(b) Referenz von Vorgängerknoten auf den (einzigen) Nachfolgerknoten des zu löschenden Knotens

⁴Saake und Sattler, *Algorithmen und Datenstrukturen*, S. 364.

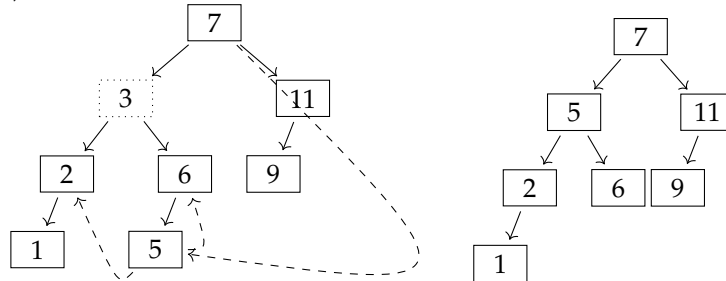
⁵Qualifizierungsmaßnahme Informatik: *Algorithmen und Datenstrukturen 5*, Seite 10.

⁶Qualifizierungsmaßnahme Informatik: *Algorithmen und Datenstrukturen 5*, Seite 11.



Knoten mit zwei Nachfolgern⁷

- Suche den zu löschenden Knoten
- Suche den „kleinsten“ Knoten im rechten Teilbaum
- Ersetze den zu löschenden Knoten durch den „kleinsten“ Knoten



Literatur

- [1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5. Bäume, Hashing.* https://www.studon.fau.de/file2619756_download.html.
- [2] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java.* 2014.
- [3] Wikipedia-Artikel „Binärbaum“. <https://de.wikipedia.org/wiki/Binärbaum>.
- [4] Wikipedia-Artikel „Binärer Suchbaum“. https://de.wikipedia.org/wiki/Binärer_Suchbaum.

⁷Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 5, Seite 12.