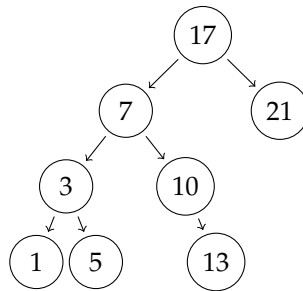


Frühjahr 2014 (46115) - Thema 2, Aufgabe 3

- (a) Fügen Sie die Zahlen 17, 7, 21, 3, 10, 13, 1, 5 nacheinander in der vorgegebenen Reihenfolge in einen binären Suchbaum ein und zeichnen Sie das Ergebnis!



- (b) Implementieren Sie in einer objektorientierten Programmiersprache eine rekursiv festgelegte Datenstruktur, deren Gestaltung sich an folgender Definition eines binären Baumes orientiert!

Ein binärer Baum ist entweder ein leerer Baum oder besteht aus einem Wurzelement, das einen binären Baum als linken und einen als rechten Teilbaum besitzt. Bei dieser Teilaufgabe können Sie auf die Implementierung von Methoden (außer ggf. notwendigen Konstruktoren) verzichten!

Klasse Knoten

```
1 class Knoten {
2     public int wert;
3     public Knoten links;
4     public Knoten rechts;
5     public Knoten elternKnoten;
6
7     Knoten(int wert) {
8         this.wert = wert;
9         links = null;
10        rechts = null;
11        elternKnoten = null;
12    }
13
14    public Knoten findeMiniumRechterTeilbaum() {
15    }
16
17    public void anhängen (Knoten knoten) {
18    }
19 }
20
21 public class BinärerSuchbaum {
22     public Knoten wurzel;
23
24     BinärerSuchbaum(Knoten wurzel) {
25         this.wurzel = wurzel;
26     }
27
28     BinärerSuchbaum() {
```

```

9      this.wurzel = null;
10   }
11
12   public void einfügen(Knoten knoten) {
13   }
14
15   public void einfügen(Knoten knoten, Knoten elternKnoten) {
16   }
17
18   public Knoten suchen(int wert) {
19   }
20
21   public Knoten suchen(int wert, Knoten knoten) {
22   }
23
24   }

```

- (c) Beschreiben Sie durch Implementierung in einer gängigen objektorientierten Programmiersprache, wie bei Verwendung der obigen Datenstruktur die Methode `loescheKnoten(w)` gestaltet sein muss, mit der der Knoten mit dem Eintrag `w` aus dem Baum entfernt werden kann, ohne die Suchbaumeigenschaft zu verletzen!

```

64   public void loescheKnoten(int w) {
65       Knoten knoten = suchen(w);
66       if (knoten == null) return;
67       // Der Knoten hat keine Teilbäume.
68       if (knoten.links == null && knoten.rechts == null) {
69           if (w < knoten.elternKnoten.wert) {
70               knoten.elternKnoten.links = null;
71           } else {
72               knoten.elternKnoten.rechts = null;
73           }
74       }
75
76       // Der Knoten besitzt einen Teilbaum.
77       // links
78       else if (knoten.links != null && knoten.rechts == null) {
79           knoten.elternKnoten.anhängen(knoten.links);
80       }
81       // rechts
82       else if (knoten.links == null) {
83           knoten.elternKnoten.anhängen(knoten.rechts);
84       }
85
86       // Der Knoten besitzt zwei Teilbäume.
87       else {
88           Knoten minimumKnoten = knoten.findeMinimumRechterTeilbaum();
89           minimumKnoten.links = knoten.links;
90           minimumKnoten.rechts = knoten.rechts;
91           knoten.elternKnoten.anhängen(minimumKnoten);
92       }
93   }

```

Klasse „BinärerSuchbaum“:

```

1   package org.bsclangaul.aufgaben.aud.examen_46115_2014_03.suchbaum;
2

```

```

3 public class BinaererSuchbaum {
4     public Knoten wurzel;
5
6     BinaererSuchbaum(Knoten wurzel) {
7         this.wurzel = wurzel;
8     }
9
10    BinaererSuchbaum() {
11        this.wurzel = null;
12    }
13
14    public void einfügen(Knoten knoten) {
15        if (wurzel != null) {
16            einfügen(knoten, wurzel);
17        } else {
18            wurzel = knoten;
19            knoten.elternKnoten = wurzel;
20        }
21    }
22
23    public void einfügen(int wert) {
24        einfügen(new Knoten(wert));
25    }
26
27    public void einfügen(Knoten knoten, Knoten elternKnoten) {
28        if (knoten.wert <= elternKnoten.wert) {
29            if (elternKnoten.links != null) {
30                einfügen(knoten, elternKnoten.links);
31            } else {
32                elternKnoten.links = knoten;
33                knoten.elternKnoten = elternKnoten;
34            }
35        } else {
36            if (elternKnoten.rechts != null) {
37                einfügen(knoten, elternKnoten.rechts);
38            } else {
39                elternKnoten.rechts = knoten;
40                knoten.elternKnoten = elternKnoten;
41            }
42        }
43    }
44
45    public Knoten suchen(int wert) {
46        if (wurzel == null || wurzel.wert == wert) {
47            return wurzel;
48        } else {
49            return suchen(wert, wurzel);
50        }
51    }
52
53    public Knoten suchen(int wert, Knoten knoten) {
54        if (knoten.wert == wert) {
55            return knoten;
56        } else if (wert < knoten.wert && knoten.links != null) {
57            return suchen(wert, knoten.links);
58        } else if (wert > knoten.wert && knoten.rechts != null) {
59            return suchen(wert, knoten.rechts);
60        }
61        return null;
62    }
63

```

```

64 public void loescheKnoten(int w) {
65     Knoten knoten = suchen(w);
66     if (knoten == null) return;
67     // Der Knoten hat keine Teilbäume.
68     if (knoten.links == null && knoten.rechts == null) {
69         if (w < knoten.elternKnoten.wert) {
70             knoten.elternKnoten.links = null;
71         } else {
72             knoten.elternKnoten.rechts = null;
73         }
74     }
75
76     // Der Knoten besitzt einen Teilbaum.
77     // links
78     else if (knoten.links != null && knoten.rechts == null) {
79         knoten.elternKnoten.anhängen(knoten.links);
80     }
81     // rechts
82     else if (knoten.links == null) {
83         knoten.elternKnoten.anhängen(knoten.rechts);
84     }
85
86     // Der Knoten besitzt zwei Teilbäume.
87     else {
88         Knoten minimumKnoten = knoten.findeMinimumRechterTeilbaum();
89         minimumKnoten.links = knoten.links;
90         minimumKnoten.rechts = knoten.rechts;
91         knoten.elternKnoten.anhängen(minimumKnoten);
92     }
93 }
94
95 // Der Baum aus dem Foliensatz
96 public BinaererSuchbaum erzeugeTestBaum() {
97     BinaererSuchbaum binärerSuchbaum = new BinaererSuchbaum();
98     binärerSuchbaum.einfügen(new Knoten(7));
99     binärerSuchbaum.einfügen(new Knoten(3));
100    binärerSuchbaum.einfügen(new Knoten(11));
101    binärerSuchbaum.einfügen(new Knoten(2));
102    binärerSuchbaum.einfügen(new Knoten(6));
103    binärerSuchbaum.einfügen(new Knoten(9));
104    binärerSuchbaum.einfügen(new Knoten(1));
105    binärerSuchbaum.einfügen(new Knoten(5));
106    return binärerSuchbaum;
107 }
108
109 public void ausgebenInOrder() {
110
111 }
112
113 public static void main(String[] args) {
114     BinaererSuchbaum binärerSuchbaum = new BinaererSuchbaum();
115     BinaererSuchbaum testBaum = binärerSuchbaum.erzeugeTestBaum();
116
117     // Teste das Einfügen
118
119     System.out.println(testBaum.wurzel.wert); // 7
120     System.out.println(testBaum.wurzel.links.wert); // 3
121     System.out.println(testBaum.wurzel.links.links.wert); // 2
122     System.out.println(testBaum.wurzel.links.rechts.wert); // 6
123     System.out.println(testBaum.wurzel.rechts.wert); // 11
124

```

```

125 // Teste das Suchen
126
127 System.out.println("Gesucht nach 5 und gefunden: " +
128     ↳ testBaum.suchen(5).wert);
129 System.out.println("Gesucht nach 9 und gefunden: " +
130     ↳ testBaum.suchen(9).wert);
131 System.out.println("Gesucht nach 7 und gefunden: " +
132     ↳ testBaum.suchen(7).wert);
133 System.out.println("Gesucht nach 10 und gefunden: " +
134     ↳ testBaum.suchen(10));
135
136 // Teste das Löschen
137
138 // Der Knoten hat keine Teilbäume.
139 System.out.println("Noch nicht gelöschter Knoten 9: " +
140     ↳ testBaum.suchen(9).wert);
141 testBaum.loescheKnoten(9);
142 System.out.println("Gelöschter Knoten 9: " + testBaum.suchen(9));
143
144 // Der Knoten hat einen Teilbaum.
145 // fristen Testbaum erzeugen.
146 testBaum = binärerSuchbaum.erzeugeTestBaum();
147 Knoten elternKnoten = testBaum.suchen(3);
148 System.out.println("Rechts Kind von 3 vor dem Löschen: " +
149     ↳ elternKnoten.rechts.wert);
150 testBaum.loescheKnoten(6);
151 System.out.println("Rechts Kind von 3 nach dem Löschen: " +
152     ↳ elternKnoten.rechts.wert);
153
154 // Der Knoten hat zwei Teilbäume.
155 // fristen Testbaum erzeugen.
156 testBaum = binärerSuchbaum.erzeugeTestBaum();
157 Knoten wurzel = testBaum.wurzel;
158 System.out.println("Linkes Kind der Wurzel vor dem Löschen: " +
159     ↳ wurzel.links.wert); // 5
160 testBaum.loescheKnoten(3);
161 System.out.println("Linkes Kind der Wurzel nach dem Löschen: " +
162     ↳ wurzel.links.wert); // 3
163 }
164 }

```

Klasse „Knoten“:

```

1 package org.bsclangaul.aufgaben.aud.examen_46115_2014_03.suchbaum;
2
3 class Knoten {
4     public int wert;
5     public Knoten links;
6     public Knoten rechts;
7     public Knoten elternKnoten;
8
9     Knoten(int wert) {
10         this.wert = wert;
11         links = null;
12         rechts = null;
13         elternKnoten = null;
14     }
15
16     public Knoten findeMiniumRechterTeilbaum() {
17         if (rechts != null) {
18             Knoten minimumKnoten = rechts;

```

```
19         while (minimumKnoten.links != null) {
20             minimumKnoten = minimumKnoten.links;
21         }
22         return minimumKnoten;
23     }
24     return null;
25 }
26
27 public void anhängen (Knoten knoten) {
28     if (knoten.wert < wert) {
29         links = knoten;
30     } else {
31         rechts = knoten;
32     }
33 }
34 }
```