

8.10.2018 16:47:09

AVLTree.java

Page 1/3

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 8 16:47:09 CEST 2018
4  */
5
6  package uebung04.as.aufgabe01;
7
8  import java.util.Collection;
9
10 import uebung02.ml.aufgabe01.BinarySearchTree.Entry;
11
12
13 public class AVLTree <K extends Comparable<? super K>, V> {
14
15     private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImpl<K, V>();
16
17     // Start the GVS-Server first: Double-Click 'GVS_Server_v1.4.jar'
18     //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplGVS<K, V>();
19
20     public V put(K key, V value) {
21         return avlTreeImpl.put(key, value);
22     }
23
24     public V get(K key) {
25         return avlTreeImpl.get(key);
26     }
27
28     public V remove(K key) {
29         return avlTreeImpl.remove(key);
30     }
31
32     public int getHeight() {
33         return avlTreeImpl.getHeight();
34     }
35
36     public int size() {
37         return avlTreeImpl.size();
38     }
39
40     public boolean isEmpty() {
41         return avlTreeImpl.isEmpty();
42     }
43
44     public void clear() {
45         avlTreeImpl.clear();
46     }
47
48     public Collection<Entry<K, V>> inorder() {
49         return avlTreeImpl.inorder();
50     }
51
52     public void printInorder() {
53         avlTreeImpl.printInorder();
54     }
55
56     public void print() {
57         avlTreeImpl.print();
58     }

```

8.10.2018 16:47:09

AVLTree.java

Page 2/3

```

59
60     protected AVLTreeImpl<K, V> getImpl() {
61         return avlTreeImpl;
62     }
63
64     public static void main(String[] args) {
65
66         AVLTree<Integer, String> avlTree = new AVLTree<Integer, String>();
67
68         System.out.println("Inserting 5:");
69         avlTree.put(5, "Str_5");
70         avlTree.print();
71         System.out.println("=====");
72         System.out.println("Inserting 7:");
73         avlTree.put(7, "Str_7");
74         avlTree.print();
75         System.out.println("=====");
76         System.out.println("Inserting 9: Single-Rotation");
77         avlTree.put(9, "Str_9");
78         avlTree.print();
79         System.out.println("=====");
80         System.out.println("Inserting 3:");
81         avlTree.put(3, "Str_3");
82         avlTree.print();
83         System.out.println("=====");
84         System.out.println("Inserting 1: Single-Rotation");
85         avlTree.put(1, "Str_1");
86         avlTree.print();
87         System.out.println("=====");
88         System.out.println("Inserting 4: Double-Rotation");
89         avlTree.put(4, "Str_4");
90         avlTree.print();
91         System.out.println("=====");
92
93         if (avlTree.getImpl() instanceof AVLTreeImplGVS) {
94             ((AVLTreeImplGVS<Integer, String>)avlTree.getImpl()).gvsTree.disconnect();
95         }
96     }
97
98 }
99
100

```

8.10.2018 16:47:09

AVLTree.java

Page 3/3

```

101
102 /* Session-Log:
103
104 Inserting 5:
105 5 - Str_5 : h=0 ROOT
106 =====
107 Inserting 7:
108 5 - Str_5 : h=1 ROOT
109 7 - Str_7 : h=0 \ parent(key)=5
110 =====
111 Inserting 9: Single-Rotation
112 5 - Str_5 : h=0 / parent(key)=7
113 7 - Str_7 : h=1 ROOT
114 9 - Str_9 : h=0 \ parent(key)=7
115 =====
116 Inserting 3:
117 3 - Str_3 : h=0 / parent(key)=5
118 5 - Str_5 : h=1 / parent(key)=7
119 7 - Str_7 : h=2 ROOT
120 9 - Str_9 : h=0 \ parent(key)=7
121 =====
122 Inserting 1: Single-Rotation
123 1 - Str_1 : h=0 / parent(key)=3
124 3 - Str_3 : h=1 / parent(key)=7
125 5 - Str_5 : h=0 \ parent(key)=3
126 7 - Str_7 : h=2 ROOT
127 9 - Str_9 : h=0 \ parent(key)=7
128 =====
129 Inserting 4: Double-Rotation
130 1 - Str_1 : h=0 / parent(key)=3
131 3 - Str_3 : h=1 / parent(key)=5
132 4 - Str_4 : h=0 \ parent(key)=3
133 5 - Str_5 : h=2 ROOT
134 7 - Str_7 : h=1 \ parent(key)=5
135 9 - Str_9 : h=0 \ parent(key)=7
136 =====
137
138 */

```

8.10.2018 16:47:09

AVLTreeImpl.java

Page 1/4

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 8 16:47:09 CEST 2018
4  */
5
6  package uebung04.as.aufgabe01;
7
8  import java.util.Collection;
9  import java.util.LinkedList;
10 import java.util.List;
11
12 import uebung02.ml.aufgabe01.BinarySearchTree;
13
14 class AVLTreeImpl<K extends Comparable<? super K>, V> extends
15     BinarySearchTree<K, V> {
16
17     /**
18      * After a BST-operation, actionNode shall point to where the balance has to
19      * be checked. -> rebalance() will then be called with actionNode.
20      */
21     protected AVLNode actionNode;
22
23
24     protected class AVLNode extends BinarySearchTree<K, V>.Node {
25
26         private int height;
27         private Node parent;
28
29         AVLNode(Entry<K, V> entry) {
30             super(entry);
31         }
32
33         protected AVLNode setParent(AVLNode parent) {
34             AVLNode old = avlNode(this.parent);
35             this.parent = parent;
36             return old;
37         }
38
39         protected AVLNode getParent() {
40             return avlNode(parent);
41         }
42
43         protected int setHeight(int height) {
44             int old = this.height;
45             this.height = height;
46             return old;
47         }
48
49         protected int getHeight() {
50             return height;
51         }
52
53         @Override
54         public AVLNode getLeftChild() {
55             return avlNode(super.getLeftChild());
56         }
57
58         @Override
59         public AVLNode getRightChild() {
60             return avlNode(super.getRightChild());
61         }

```

8.10.2018 16:47:09

AVLTreImpl.java

Page 2/4

```

62
63     @Override
64     public String toString() {
65         String result = String.format("%2d - %-6s : h=%d",
66                                     getEntry().getKey(), getEntry().getValue(), height);
67         if (parent == null) {
68             result += " ROOT";
69         } else {
70             boolean left = (parent.getLeftChild() == this) ? true : false;
71             result += (left ? " / " : " : ") + "parent(key)="
72                     + parent.getEntry().getKey();
73         }
74         return result;
75     }
76 } // End of class AVLNode
77
78
79 protected AVLNode getRoot() {
80     return avlNode(root);
81 }
82
83
84 public V put(K key, V value) {
85     Entry<K, V> entry = find(key);
86     if (entry != null) {
87         // key already exists in the Tree
88         return entry.setValue(value);
89     } else {
90         // key does not exist in the Tree yet
91         super.insert(key, value);
92         rebalance(actionNode);
93         actionNode = null;
94         return null;
95     }
96 }
97
98 public V get(K key) {
99     Entry<K, V> entry = super.find(key);
100    if (entry != null) {
101        return entry.getValue();
102    } else {
103        return null;
104    }
105 }
106
107 @Override
108 protected Node insert(Node node, Entry<K, V> entry) {
109     if (node != null) {
110         actionNode = avlNode(node);
111     }
112     // calling now the BST-insert() which will do the work:
113     AVLNode result = avlNode(super.insert(node, entry));
114     if (node == null) {
115         // In this case: result of super.insert() is the new node!
116         result.setParent(actionNode);
117     }
118     return result;
119 }
120
121 /**
122  * The height of the tree.
123  *
124  * @return The actual height. -1 for an empty tree.
125  */
126 @Override
127 public int getHeight() {
128     return height(avlNode(root));
129 }

```

8.10.2018 16:47:09

AVLTreImpl.java

Page 3/4

```

130
131 /**
132  * Returns the height of this node.
133  *
134  * @param node
135  * @return The height or -1 if null.
136  */
137 protected int height(AVLNode node) {
138     return (node != null) ? node.getHeight() : -1;
139 }
140
141 /**
142  * Restructures the tree with rotations.
143  *
144  * @param xPos
145  *        The X-node.
146  * @return The new root-node of this subtree.
147  */
148 protected AVLNode restructure(AVLNode xPos) {
149     // TODO Implement here...
150     return null;
151 }
152
153 protected AVLNode tallerChild(AVLNode node) {
154     // TODO Implement here...
155     return null;
156 }
157
158 protected AVLNode rotateWithLeftChild(AVLNode k2) {
159     // TODO Implement here...
160     return null;
161 }
162
163 protected AVLNode doubleRotateWithLeftChild(AVLNode k3) {
164     // TODO Implement here...
165     return null;
166 }
167
168 protected AVLNode rotateWithRightChild(AVLNode k1) {
169     // TODO Implement here...
170     return null;
171 }
172
173 protected AVLNode doubleRotateWithRightChild(AVLNode k3) {
174     // TODO Implement here...
175     return null;
176 }
177
178 protected boolean isBalanced(AVLNode node) {
179     // TODO Implement here...
180     return false;
181 }
182
183 /**
184  * Assures the balance of the tree from 'node' up to the root.
185  *
186  * @param node
187  *        The node from where to start.
188  */
189 protected void rebalance(AVLNode node) {
190     // TODO Implement here...
191 }

```

8.10.2018 16:47:09

AVLTreeImpl.java

Page 4/4

```

192
193 /**
194  * Assures the correct height for node.
195  *
196  * @param node
197  *       The node to assure its height.
198  */
199 protected void setHeight(AVLNode node) {
200     if (node == null) {
201         return;
202     }
203     int heightLeftChild = height(node.getLeftChild());
204     int heightRightChild = height(node.getRightChild());
205     node.setHeight(1 + Math.max(heightLeftChild, heightRightChild));
206 }
207
208 /**
209  * Factory-Method. Creates a new node.
210  *
211  * @param entry
212  *       The entry to be inserted in the new node.
213  * @return The new created node.
214  */
215 @Override
216 protected Node newNode(Entry<K, V> entry) {
217     return new AVLNode(entry);
218 }
219
220 public V remove(K key) {
221     // TODO Implement here...
222     return null;
223 }
224
225 /**
226  * Generates an inorder-node-list.
227  *
228  * @param nodeList
229  *       The node-list to fill in inorder.
230  * @param node
231  *       The node to start from.
232  */
233 protected void inorder(Collection<AVLNode> nodeList, AVLNode node) {
234     if (node == null)
235         return;
236     inorder(nodeList, node.getLeftChild());
237     nodeList.add(node);
238     inorder(nodeList, node.getRightChild());
239 }
240
241 @SuppressWarnings("unchecked")
242 protected AVLNode avlNode(Node node) {
243     return (AVLNode)node;
244 }
245
246 public void print() {
247     List<AVLNode> nodeList = new LinkedList<>();
248     inorder(nodeList, avlNode(root));
249     for (AVLNode node: nodeList) {
250         System.out.println(node + " ");
251     }
252 }
253
254 }
255
256

```

8.10.2018 16:47:09

AVLTreeImplGVS.java

Page 1/2

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 8 16:47:09 CEST 2018
4  */
5
6  package uebung04.as.aufgabe01;
7
8  import gvs.tree.GVSBinaryTreeNode;
9  import gvs.tree.GVSTreeWithRoot;
10 import gvs.typ.node.GVSNodeType;
11
12 class AVLTreeImplGVS<K extends Comparable<? super K>, V> extends
13     AVLTreeImpl<K, V> {
14
15     protected GVSTreeWithRoot gvsTree;
16
17     private final int DELAY = 200;
18
19     protected class AVLNodeGVS extends AVLTreeImpl<K, V>.AVLNode implements GVSBinaryTree
20     eNode {
21
22         protected AVLNodeGVS(Entry<K, V> entry) {
23             super(entry);
24         }
25
26         public GVSBinaryTreeNode getGVSLeftChild() {
27             return (GVSBinaryTreeNode) getLeftChild();
28         }
29
30         public GVSBinaryTreeNode getGVSRightChild() {
31             return (GVSBinaryTreeNode) getRightChild();
32         }
33
34         public String getNodeLabel() {
35             Entry<K, V> e = getEntry();
36             return e.getKey() + " " + e.getValue();
37             //return e.getKey().toString();
38         }
39
40         public GVSNodeType getNodeTyp() {
41             return null;
42         }
43     } // class BinaryTreeTestGVS.NodeGVS
44
45
46     AVLTreeImplGVS() {
47         this("AVLTreeGVS");
48     }
49
50     AVLTreeImplGVS(String title) {
51         gvsTree = new GVSTreeWithRoot(title);
52     }
53
54     @Override
55     protected Node newNode(Entry<K, V> entry) {
56         return new AVLNodeGVS(entry);
57     }
58
59     @Override
60     public V put(K key, V value) {
61         V result = super.put(key, value);
62         gvsTree.setRoot((GVSBinaryTreeNode) root);
63         gvsTree.display();
64         try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
65         return result;
66     }

```

8.10.2018 16:47:09

AVLTreeImplGVS.java

Page 2/2

```

67
68  @Override
69  protected AVLNode rotateWithRightChild(AVLNode k1) {
70      gvsTree.setRoot((GVSBinaryTreeNode) root);
71      gvsTree.display();
72      try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
73      AVLNode newRoot = super.rotateWithRightChild(k1);
74      return newRoot;
75  }
76
77  @Override
78  protected AVLNode rotateWithLeftChild(AVLNode k2) {
79      gvsTree.setRoot((GVSBinaryTreeNode) root);
80      gvsTree.display();
81      try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
82      AVLNode newRoot = super.rotateWithLeftChild(k2);
83      return newRoot;
84  }
85
86  @Override
87  public V remove(K key) {
88      V result = super.remove(key);
89      gvsTree.setRoot((GVSBinaryTreeNode) root);
90      gvsTree.display();
91      try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
92      return result;
93  }
94
95  }
96
97
98

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 1/8

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 8 16:47:09 CEST 2018
4  */
5
6  package uebung04.as.aufgabe01;
7
8  import static org.junit.Assert.*;
9
10 import java.util.Collection;
11 import java.util.Hashtable;
12 import java.util.LinkedList;
13 import java.util.Map;
14 import java.util.Random;
15
16 import org.junit.After;
17 import org.junit.Before;
18 import org.junit.FixMethodOrder;
19 import org.junit.Test;
20 import org.junit.runners.MethodSorters;
21
22 import uebung02.ml.aufgabe01.BinarySearchTree.Entry;
23
24
25 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
26 public class AVLTreeJUnitTest {
27
28     AVLTreeImpl<Integer, String> avlTree;
29
30     @Before
31     public void setUp() {
32         //System.setProperty("NoGVS", "true");
33         avlTree = new AVLTree<Integer, String>().getImpl();
34     }
35
36     @After
37     public void tearDown() {
38         if (avlTree instanceof AVLTreeImplGVS) {
39             ((AVLTreeImplGVS<Integer, String>)avlTree).gvsTree.disconnect();
40         }
41     }
42
43     @Test
44     public void test01Put() {
45         int[] keys = { 2, 1, 3 };
46         String[] expected = {
47             " 1 - Str_1 : h=0 / parent(key)=2",
48             " 2 - Str_2 : h=1 ROOT",
49             " 3 - Str_3 : h=0 \\ parent(key)=2",
50         };
51         runTest(keys, expected);
52     }
53
54     @Test
55     public void test02Get() {
56         int[] keys = { 2, 1, 5, 4, 3 };
57         String[] expected = {
58             " 1 - Str_1 : h=0 / parent(key)=2",
59             " 2 - Str_2 : h=2 ROOT",
60             " 3 - Str_3 : h=0 / parent(key)=4",
61             " 4 - Str_4 : h=1 \\ parent(key)=2",
62             " 5 - Str_5 : h=0 \\ parent(key)=4",
63         };
64         runTest(keys, expected);
65         assertEquals("Str_2", avlTree.get(2));
66         assertEquals("Str_5", avlTree.get(5));
67         assertNull(avlTree.get(0));
68         assertNull(avlTree.get(6));
69     }

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 2/8

```

70
71 @Test
72 public void test03SingleRotationLeftInRoot() {
73     int[] keys = { 1, 2, 3 };
74     String[] expected = {
75         " 1 - Str_1 : h=0 / parent(key)=2",
76         " 2 - Str_2 : h=1 ROOT",
77         " 3 - Str_3 : h=0 \\ parent(key)=2",
78     };
79     runTest(keys, expected);
80 }
81
82 @Test
83 public void test04SingleRotationLeftBelowRoot() {
84     int[] keys = { 5, 6, 1, 2, 3 };
85     String[] expected = {
86         " 1 - Str_1 : h=0 / parent(key)=2",
87         " 2 - Str_2 : h=1 / parent(key)=5",
88         " 3 - Str_3 : h=0 \\ parent(key)=2",
89         " 5 - Str_5 : h=2 ROOT",
90         " 6 - Str_6 : h=0 \\ parent(key)=5",
91     };
92     runTest(keys, expected);
93 }
94
95 @Test
96 public void test05SingleRotationRightInRoot() {
97     int[] keys = { 3, 2, 1 };
98     String[] expected = {
99         " 1 - Str_1 : h=0 / parent(key)=2",
100         " 2 - Str_2 : h=1 ROOT",
101         " 3 - Str_3 : h=0 \\ parent(key)=2",
102     };
103     runTest(keys, expected);
104 }
105
106 @Test
107 public void test06SingleRotationRightBelowRoot() {
108     int[] keys = { 2, 1, 5, 4, 3 };
109     String[] expected = {
110         " 1 - Str_1 : h=0 / parent(key)=2",
111         " 2 - Str_2 : h=2 ROOT",
112         " 3 - Str_3 : h=0 / parent(key)=4",
113         " 4 - Str_4 : h=1 \\ parent(key)=2",
114         " 5 - Str_5 : h=0 \\ parent(key)=4",
115     };
116     runTest(keys, expected);
117 }
118
119 @Test
120 public void test07DoubleRotationLeftInRoot() {
121     int[] keys = { 1, 3, 2 };
122     String[] expected = {
123         " 1 - Str_1 : h=0 / parent(key)=2",
124         " 2 - Str_2 : h=1 ROOT",
125         " 3 - Str_3 : h=0 \\ parent(key)=2",
126     };
127     runTest(keys, expected);
128 }

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 3/8

```

129
130 @Test
131 public void test08DoubleRotationLeftBelowRoot() {
132     int[] keys = { 2, 1, 3, 5, 4 };
133     String[] expected = {
134         " 1 - Str_1 : h=0 / parent(key)=2",
135         " 2 - Str_2 : h=2 ROOT",
136         " 3 - Str_3 : h=0 / parent(key)=4",
137         " 4 - Str_4 : h=1 \\ parent(key)=2",
138         " 5 - Str_5 : h=0 \\ parent(key)=4",
139     };
140     runTest(keys, expected);
141 }
142
143 @Test
144 public void test09DoubleRotationRightInRoot() {
145     int[] keys = { 3, 1, 2 };
146     String[] expected = {
147         " 1 - Str_1 : h=0 / parent(key)=2",
148         " 2 - Str_2 : h=1 ROOT",
149         " 3 - Str_3 : h=0 \\ parent(key)=2",
150     };
151     runTest(keys, expected);
152 }
153
154 @Test
155 public void test10DoubleRotationRightBelowRoot() {
156     int[] keys = { 4, 3, 5, 1, 2 };
157     String[] expected = {
158         " 1 - Str_1 : h=0 / parent(key)=2",
159         " 2 - Str_2 : h=1 / parent(key)=4",
160         " 3 - Str_3 : h=0 \\ parent(key)=2",
161         " 4 - Str_4 : h=2 ROOT",
162         " 5 - Str_5 : h=0 \\ parent(key)=4",
163     };
164     runTest(keys, expected);
165 }
166
167 @Test
168 public void test11MultipleSameKeys() {
169     int[] keys = { 3, 1, 2 };
170     String[] expected = {
171         " 1 - Str_1 : h=0 / parent(key)=2",
172         " 2 - Str_2 : h=1 ROOT",
173         " 3 - Str_3 : h=0 \\ parent(key)=2",
174     };
175     runTest(keys, expected);
176     avlTree.put(2, "Str_22");
177     avlTree.put(2, "Str_23");
178     expected = new String[] {
179         " 1 - Str_1 : h=0 / parent(key)=2",
180         " 2 - Str_23 : h=1 ROOT",
181         " 3 - Str_3 : h=0 \\ parent(key)=2",
182     };
183     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
184     avlTree.inorder(nodes, avlTree.getRoot());
185     verify(nodes, expected);
186 }

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 4/8

```

187
188 @Test
189 public void test12RemovingCase1() {
190     // L?schen Fall 1 gem. BST-Folie 12:
191     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
192     int[] keys = { 6, 2, 9, 1, 4, 8 };
193     String[] expected = {
194         " 1 - Str_1 : h=0 / parent(key)=2",
195         " 2 - Str_2 : h=1 / parent(key)=6",
196         " 4 - Str_4 : h=0 \\ parent(key)=2",
197         " 6 - Str_6 : h=2 ROOT",
198         " 8 - Str_8 : h=0 / parent(key)=9",
199         " 9 - Str_9 : h=1 \\ parent(key)=6",
200     };
201     runTest(keys, expected);
202     assertEquals("Str_4", avlTree.remove(4));
203     expected = new String[] {
204         " 1 - Str_1 : h=0 / parent(key)=2",
205         " 2 - Str_2 : h=1 / parent(key)=6",
206         " 6 - Str_6 : h=2 ROOT",
207         " 8 - Str_8 : h=0 / parent(key)=9",
208         " 9 - Str_9 : h=1 \\ parent(key)=6",
209     };
210     avlTree.inorder(nodes, avlTree.getRoot());
211     verify(nodes, expected);
212 }
213
214 @Test
215 public void test13RemovingCase2() {
216     // L?schen Fall 2 gem. BST-Folie 13:
217     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
218     int[] keys = { 6, 2, 9, 1, 4, 8, 5 };
219     String[] expected = {
220         " 1 - Str_1 : h=0 / parent(key)=2",
221         " 2 - Str_2 : h=2 / parent(key)=6",
222         " 4 - Str_4 : h=1 \\ parent(key)=2",
223         " 5 - Str_5 : h=0 \\ parent(key)=4",
224         " 6 - Str_6 : h=3 ROOT",
225         " 8 - Str_8 : h=0 / parent(key)=9",
226         " 9 - Str_9 : h=1 \\ parent(key)=6",
227     };
228     runTest(keys, expected);
229     assertEquals("Str_4", avlTree.remove(4));
230     expected = new String[] {
231         " 1 - Str_1 : h=0 / parent(key)=2",
232         " 2 - Str_2 : h=1 / parent(key)=6",
233         " 5 - Str_5 : h=0 \\ parent(key)=2",
234         " 6 - Str_6 : h=2 ROOT",
235         " 8 - Str_8 : h=0 / parent(key)=9",
236         " 9 - Str_9 : h=1 \\ parent(key)=6",
237     };
238     avlTree.inorder(nodes, avlTree.getRoot());
239     verify(nodes, expected);
240 }

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 5/8

```

241
242 @Test
243 public void test14RemovingCase3() {
244     // L?schen Fall 3 gem. BST-Folie 14:
245     // Hinweis: Baum entsprechend 'aufgef?llt' (wegen AVL!)
246     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
247     int[] keys = { 1, -10, 4, -15, -5, 2, 9, -18, -12, -7, -3, 3, 7, 10, 6 };
248     String[] expected = {
249         "-18 - Str_-18 : h=0 / parent(key)=-15",
250         "-15 - Str_-15 : h=1 / parent(key)=-10",
251         "-12 - Str_-12 : h=0 \\ parent(key)=-15",
252         "-10 - Str_-10 : h=2 / parent(key)=1",
253         "-7 - Str_-7 : h=0 / parent(key)=-5",
254         "-5 - Str_-5 : h=1 \\ parent(key)=-10",
255         "-3 - Str_-3 : h=0 \\ parent(key)=-5",
256         " 1 - Str_1 : h=4 ROOT",
257         " 2 - Str_2 : h=1 / parent(key)=4",
258         " 3 - Str_3 : h=0 \\ parent(key)=2",
259         " 4 - Str_4 : h=3 \\ parent(key)=1",
260         " 6 - Str_6 : h=0 / parent(key)=7",
261         " 7 - Str_7 : h=1 / parent(key)=9",
262         " 9 - Str_9 : h=2 \\ parent(key)=4",
263         "10 - Str_10 : h=0 \\ parent(key)=9",
264     };
265     runTest(keys, expected);
266     assertEquals("Str_4", avlTree.remove(4));
267     expected = new String[] {
268         "-18 - Str_-18 : h=0 / parent(key)=-15",
269         "-15 - Str_-15 : h=1 / parent(key)=-10",
270         "-12 - Str_-12 : h=0 \\ parent(key)=-15",
271         "-10 - Str_-10 : h=2 / parent(key)=1",
272         "-7 - Str_-7 : h=0 / parent(key)=-5",
273         "-5 - Str_-5 : h=1 \\ parent(key)=-10",
274         "-3 - Str_-3 : h=0 \\ parent(key)=-5",
275         " 1 - Str_1 : h=3 ROOT",
276         " 2 - Str_2 : h=1 / parent(key)=6",
277         " 3 - Str_3 : h=0 \\ parent(key)=2",
278         " 6 - Str_6 : h=2 \\ parent(key)=1",
279         " 7 - Str_7 : h=0 / parent(key)=9",
280         " 9 - Str_9 : h=1 \\ parent(key)=6",
281         "10 - Str_10 : h=0 \\ parent(key)=9",
282     };
283     avlTree.inorder(nodes, avlTree.getRoot());
284     verify(nodes, expected);
285 }
286
287 @Test
288 public void test15RemovingAtRoot1() {
289     int[] keys = { 1, 2, 3 };
290     String[] expected = {
291         " 1 - Str_1 : h=0 / parent(key)=2",
292         " 2 - Str_2 : h=1 ROOT",
293         " 3 - Str_3 : h=0 \\ parent(key)=2",
294     };
295     runTest(keys, expected);
296     assertEquals("Str_1", avlTree.remove(1));
297     assertEquals(2, avlTree.size());
298     assertEquals("Str_3", avlTree.remove(3));
299     assertEquals(1, avlTree.size());
300     assertEquals("Str_2", avlTree.remove(2));
301     assertEquals(0, avlTree.size());
302 }

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 6/8

```

303
304 @Test
305 public void test16RemovingAtRoot2() {
306     int[] keys = { 1, 2, 3 };
307     String[] expected = {
308         " 1 - Str_1 : h=0 / parent(key)=2",
309         " 2 - Str_2 : h=1 ROOT",
310         " 3 - Str_3 : h=0 \\ parent(key)=2",
311     };
312     runTest(keys, expected);
313     assertEquals("Str_1", avlTree.remove(1));
314     assertEquals(2, avlTree.size());
315     assertEquals("Str_2", avlTree.remove(2));
316     assertEquals(1, avlTree.size());
317     assertEquals("Str_3", avlTree.remove(3));
318     assertEquals(0, avlTree.size());
319 }
320
321 @Test
322 public void test17RemovingAtRoot3() {
323     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
324     int[] keys = { 1, 2, 3 };
325     String[] expected = {
326         " 1 - Str_1 : h=0 / parent(key)=2",
327         " 2 - Str_2 : h=1 ROOT",
328         " 3 - Str_3 : h=0 \\ parent(key)=2",
329     };
330     runTest(keys, expected);
331     assertEquals("Str_2", avlTree.remove(2));
332     expected = new String[] {
333         " 1 - Str_1 : h=0 / parent(key)=3",
334         " 3 - Str_3 : h=1 ROOT",
335     };
336     avlTree.inorder(nodes, avlTree.getRoot());
337     verify(nodes, expected);
338     assertEquals(2, avlTree.size());
339     assertEquals("Str_3", avlTree.remove(3));
340     assertEquals(1, avlTree.size());
341     assertEquals("Str_1", avlTree.remove(1));
342     assertEquals(0, avlTree.size());
343 }
344
345 @Test
346 public void test18RemovingAtRoot4() {
347     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
348     int[] keys = { 3, 2, 6, 4 };
349     String[] expected = {
350         " 2 - Str_2 : h=0 / parent(key)=3",
351         " 3 - Str_3 : h=2 ROOT",
352         " 4 - Str_4 : h=0 / parent(key)=6",
353         " 6 - Str_6 : h=1 \\ parent(key)=3",
354     };
355     runTest(keys, expected);
356     assertEquals("Str_3", avlTree.remove(3));
357     expected = new String[] {
358         " 2 - Str_2 : h=0 / parent(key)=4",
359         " 4 - Str_4 : h=1 ROOT",
360         " 6 - Str_6 : h=0 \\ parent(key)=4",
361     };
362     avlTree.inorder(nodes, avlTree.getRoot());
363     verify(nodes, expected);
364 }

```

8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 7/8

```

365
366 @Test
367 public void test19RemovingAtRoot5() {
368     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
369     int[] keys = { 3, 2, 6, 1, 4, 7, 5 };
370     String[] expected = {
371         " 1 - Str_1 : h=0 / parent(key)=2",
372         " 2 - Str_2 : h=1 / parent(key)=3",
373         " 3 - Str_3 : h=3 ROOT",
374         " 4 - Str_4 : h=1 / parent(key)=6",
375         " 5 - Str_5 : h=0 \\ parent(key)=4",
376         " 6 - Str_6 : h=2 \\ parent(key)=3",
377         " 7 - Str_7 : h=0 \\ parent(key)=6",
378     };
379     runTest(keys, expected);
380     assertEquals("Str_3", avlTree.remove(3));
381     expected = new String[] {
382         " 1 - Str_1 : h=0 / parent(key)=2",
383         " 2 - Str_2 : h=1 / parent(key)=4",
384         " 4 - Str_4 : h=2 ROOT",
385         " 5 - Str_5 : h=0 / parent(key)=6",
386         " 6 - Str_6 : h=1 \\ parent(key)=4",
387         " 7 - Str_7 : h=0 \\ parent(key)=6",
388     };
389     avlTree.inorder(nodes, avlTree.getRoot());
390     verify(nodes, expected);
391 }
392
393 @Test
394 public void test20RemovingAtRoot6() {
395     int[] keys = { 1 };
396     String[] expected = {
397         " 1 - Str_1 : h=0 ROOT",
398     };
399     runTest(keys, expected);
400     assertEquals(null, avlTree.remove(8888));
401     assertEquals(1, avlTree.size());
402     runTest(keys, expected);
403     assertEquals("Str_1", avlTree.remove(1));
404     assertEquals(0, avlTree.size());
405 }
406
407 @Test
408 public void test21RemovingEntryNotInTree() {
409     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
410     int[] keys = { 1, 2, 3 };
411     String[] expected = {
412         " 1 - Str_1 : h=0 / parent(key)=2",
413         " 2 - Str_2 : h=1 ROOT",
414         " 3 - Str_3 : h=0 \\ parent(key)=2",
415     };
416     runTest(keys, expected);
417     assertNull(avlTree.remove(4));
418     expected = new String[] {
419         " 1 - Str_1 : h=0 / parent(key)=2",
420         " 2 - Str_2 : h=1 ROOT",
421         " 3 - Str_3 : h=0 \\ parent(key)=2",
422     };
423     avlTree.inorder(nodes, avlTree.getRoot());
424     verify(nodes, expected);
425 }

```


8.10.2018 16:47:09

AVLTreeJUnitTest.java

Page 8/8

```

426
427 @Test
428 public void test22StressTest() {
429     final int SIZE = 10000;
430     Random randomGenerator = new Random(1);
431     // a Map to compare:
432     Map<Integer, String> map = new Hashtable<Integer, String>();
433     // key-Counters: count for every key how many time it was generated
434     Map<Integer, Integer> keyCounters = new Hashtable<Integer, Integer>();
435     // fill the Tree
436     for (int i = 0; i < SIZE; i++) {
437         int key = (int) (randomGenerator.nextFloat() * SIZE / 3);
438         Integer numberOfKeys = keyCounters.get(key);
439         if (numberOfKeys == null) {
440             numberOfKeys = 1;
441         } else {
442             numberOfKeys++;
443         }
444         keyCounters.put(key, numberOfKeys);
445         avlTree.put(key, "_" + i);
446         map.put(key, "_" + i);
447         assertEquals(keyCounters.size(), avlTree.size());
448         assertEquals(map.size(), avlTree.size());
449     }
450     verifyInorder();
451     // remove all Keys
452     Integer[] keyArr = new Integer[1];
453     keyArr = map.keySet().toArray(keyArr);
454     for (int key : keyArr) {
455         assertEquals(map.remove(key), avlTree.remove(key));
456         assertEquals(map.size(), avlTree.size());
457         verifyInorder();
458     }
459     assertEquals(0, avlTree.size());
460 }
461
462 private void verifyInorder() {
463     Collection<Entry<Integer, String>> inorderList = avlTree.inorder();
464     int last = Integer.MIN_VALUE;
465     for (Entry<Integer, String> entry: inorderList) {
466         Integer key = entry.getKey();
467         assertTrue(key.compareTo(last) >= 0);
468         last = key;
469     }
470 }
471
472 private void runTest(int[] keys, String[] expected) {
473     for (int key : keys) {
474         avlTree.put(key, "Str_" + key);
475     }
476     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
477     avlTree.inorder(nodes, avlTree.getRoot());
478     assertEquals(expected.length, nodes.size());
479     verify(nodes, expected);
480 }
481
482 private void verify(Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes, String[]
expected) {
483     int i = 0;
484     for (AVLTreeImpl<Integer, String>.AVLNode node: nodes) {
485         String nodeStr = node.toString();
486         String expectedStr = expected[i];
487         assertEquals(expectedStr, nodeStr);
488         i++;
489     }
490 }
491 }
492 }
493

```