

6.10.2019 19:25:51

AVLTree.java

Page 1/3

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sun Oct 6 19:25:51 CEST 2019
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     // Show in ADV:
18     //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplADV<K, V>("AVL-Tree ?4");
19     //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplADV<K, V>("AVL-Tree ?4", 3,
20     2);
21     // Be aware of NodeFixationException!
22
23     public V put(K key, V value) {
24         return avlTreeImpl.put(key, value);
25     }
26
27     public V get(K key) {
28         return avlTreeImpl.get(key);
29     }
30
31     public V remove(K key) {
32         return avlTreeImpl.remove(key);
33     }
34
35     public int getHeight() {
36         return avlTreeImpl.getHeight();
37     }
38
39     public int size() {
40         return avlTreeImpl.size();
41     }
42
43     public boolean isEmpty() {
44         return avlTreeImpl.isEmpty();
45     }
46
47     public void clear() {
48         avlTreeImpl.clear();
49     }
50
51     public Collection<Entry<K, V>> inorder() {
52         return avlTreeImpl.inorder();
53     }
54
55     public void printInorder() {
56         avlTreeImpl.printInorder();
57     }
58
59     public void print() {
60         avlTreeImpl.print();
61     }
62
63     protected AVLTreeImpl<K, V> getImpl() {
64         return avlTreeImpl;
65     }

```

6.10.2019 19:25:51

AVLTree.java

Page 2/3

```

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

```

6.10.2019 19:25:51

AVLTree.java

Page 3/3

```

99
100 /* Session-Log:
101
102 Inserting 5:
103 5 - Str_5 : h=0 ROOT
104 =====
105 Inserting 7:
106 5 - Str_5 : h=1 ROOT
107 7 - Str_7 : h=0 \ parent(key)=5
108 =====
109 Inserting 9: Single-Rotation
110 5 - Str_5 : h=0 / parent(key)=7
111 7 - Str_7 : h=1 ROOT
112 9 - Str_9 : h=0 \ parent(key)=7
113 =====
114 Inserting 3:
115 3 - Str_3 : h=0 / parent(key)=5
116 5 - Str_5 : h=1 / parent(key)=7
117 7 - Str_7 : h=2 ROOT
118 9 - Str_9 : h=0 \ parent(key)=7
119 =====
120 Inserting 1: Single-Rotation
121 1 - Str_1 : h=0 / parent(key)=3
122 3 - Str_3 : h=1 / parent(key)=7
123 5 - Str_5 : h=0 \ parent(key)=3
124 7 - Str_7 : h=2 ROOT
125 9 - Str_9 : h=0 \ parent(key)=7
126 =====
127 Inserting 4: Double-Rotation
128 1 - Str_1 : h=0 / parent(key)=3
129 3 - Str_3 : h=1 / parent(key)=5
130 4 - Str_4 : h=0 \ parent(key)=3
131 5 - Str_5 : h=2 ROOT
132 7 - Str_7 : h=1 \ parent(key)=5
133 9 - Str_9 : h=0 \ parent(key)=7
134 =====
135
136 */

```

6.10.2019 19:25:51

AVLTreeImpl.java

Page 1/4

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sun Oct 6 19:25:51 CEST 2019
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         }

```

6.10.2019 19:25:51

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 }

```

6.10.2019 19:25:51

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 }

```

6.10.2019 19:25:51

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 // Type-Casting: Node -> AVLNode (Cast-Encapsulation)
242 @SuppressWarnings("unchecked")
243 protected AVLNode avlNode(Node node) {
244     return (AVLNode)node;
245 }
246
247 public void print() {
248     List<AVLNode> nodeList = new LinkedList<>();
249     inorder(nodeList, avlNode(root));
250     for (AVLNode node: nodeList) {
251         System.out.println(node + " ");
252     }
253 }
254
255 }
256
257

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 1/8

```

1  /*
2   * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Sun Oct 6 19:25:51 CEST 2019
4   */
5
6  package uebung04.as.aufgabe01;
7
8  import static org.junit.Assert.assertEquals;
9  import static org.junit.Assert.assertNull;
10 import static org.junit.Assert.assertTrue;
11
12 import java.util.Collection;
13 import java.util.Hashtable;
14 import java.util.LinkedList;
15 import java.util.Map;
16 import java.util.Random;
17
18 import org.junit.Before;
19 import org.junit.FixMethodOrder;
20 import org.junit.Test;
21 import org.junit.runners.MethodSorters;
22
23 import uebung02.ml.aufgabe01.BinarySearchTree.Entry;
24
25
26 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
27 public class AVLTreeJUnitTest {
28
29     AVLTreeImpl<Integer, String> avlTree;
30
31     @Before
32     public void setUp() {
33         avlTree = new AVLTreeImpl<Integer, String>();
34     }
35
36     @Test
37     public void test01Put() {
38         int[] keys = { 2, 1, 3 };
39         String[] expected = {
40             " 1 - Str_1 : h=0 / parent(key)=2",
41             " 2 - Str_2 : h=1 ROOT",
42             " 3 - Str_3 : h=0 \\ parent(key)=2",
43         };
44         runTest(keys, expected);
45     }
46
47     @Test
48     public void test02Get() {
49         int[] keys = { 2, 1, 5, 4, 3 };
50         String[] expected = {
51             " 1 - Str_1 : h=0 / parent(key)=2",
52             " 2 - Str_2 : h=2 ROOT",
53             " 3 - Str_3 : h=0 / parent(key)=4",
54             " 4 - Str_4 : h=1 \\ parent(key)=2",
55             " 5 - Str_5 : h=0 \\ parent(key)=4",
56         };
57         runTest(keys, expected);
58         assertEquals("Str_2", avlTree.get(2));
59         assertEquals("Str_5", avlTree.get(5));
60         assertNull(avlTree.get(0));
61         assertNull(avlTree.get(6));
62     }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 2/8

```

63
64 @Test
65 public void test03SingleRotationLeftInRoot() {
66     int[] keys = { 1, 2, 3 };
67     String[] expected = {
68         " 1 - Str_1 : h=0 / parent(key)=2",
69         " 2 - Str_2 : h=1 ROOT",
70         " 3 - Str_3 : h=0 \\ parent(key)=2",
71     };
72     runTest(keys, expected);
73 }
74
75 @Test
76 public void test04SingleRotationLeftBelowRoot() {
77     int[] keys = { 5, 6, 1, 2, 3 };
78     String[] expected = {
79         " 1 - Str_1 : h=0 / parent(key)=2",
80         " 2 - Str_2 : h=1 / parent(key)=5",
81         " 3 - Str_3 : h=0 \\ parent(key)=2",
82         " 5 - Str_5 : h=2 ROOT",
83         " 6 - Str_6 : h=0 \\ parent(key)=5",
84     };
85     runTest(keys, expected);
86 }
87
88 @Test
89 public void test05SingleRotationRightInRoot() {
90     int[] keys = { 3, 2, 1 };
91     String[] expected = {
92         " 1 - Str_1 : h=0 / parent(key)=2",
93         " 2 - Str_2 : h=1 ROOT",
94         " 3 - Str_3 : h=0 \\ parent(key)=2",
95     };
96     runTest(keys, expected);
97 }
98
99 @Test
100 public void test06SingleRotationRightBelowRoot() {
101     int[] keys = { 2, 1, 5, 4, 3 };
102     String[] expected = {
103         " 1 - Str_1 : h=0 / parent(key)=2",
104         " 2 - Str_2 : h=2 ROOT",
105         " 3 - Str_3 : h=0 / parent(key)=4",
106         " 4 - Str_4 : h=1 \\ parent(key)=2",
107         " 5 - Str_5 : h=0 \\ parent(key)=4",
108     };
109     runTest(keys, expected);
110 }
111
112 @Test
113 public void test07DoubleRotationLeftInRoot() {
114     int[] keys = { 1, 3, 2 };
115     String[] expected = {
116         " 1 - Str_1 : h=0 / parent(key)=2",
117         " 2 - Str_2 : h=1 ROOT",
118         " 3 - Str_3 : h=0 \\ parent(key)=2",
119     };
120     runTest(keys, expected);
121 }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 3/8

```

122
123 @Test
124 public void test08DoubleRotationLeftBelowRoot() {
125     int[] keys = { 2, 1, 3, 5, 4 };
126     String[] expected = {
127         " 1 - Str_1 : h=0 / parent(key)=2",
128         " 2 - Str_2 : h=2 ROOT",
129         " 3 - Str_3 : h=0 / parent(key)=4",
130         " 4 - Str_4 : h=1 \\ parent(key)=2",
131         " 5 - Str_5 : h=0 \\ parent(key)=4",
132     };
133     runTest(keys, expected);
134 }
135
136 @Test
137 public void test09DoubleRotationRightInRoot() {
138     int[] keys = { 3, 1, 2 };
139     String[] expected = {
140         " 1 - Str_1 : h=0 / parent(key)=2",
141         " 2 - Str_2 : h=1 ROOT",
142         " 3 - Str_3 : h=0 \\ parent(key)=2",
143     };
144     runTest(keys, expected);
145 }
146
147 @Test
148 public void test10DoubleRotationRightBelowRoot() {
149     int[] keys = { 4, 3, 5, 1, 2 };
150     String[] expected = {
151         " 1 - Str_1 : h=0 / parent(key)=2",
152         " 2 - Str_2 : h=1 / parent(key)=4",
153         " 3 - Str_3 : h=0 \\ parent(key)=2",
154         " 4 - Str_4 : h=2 ROOT",
155         " 5 - Str_5 : h=0 \\ parent(key)=4",
156     };
157     runTest(keys, expected);
158 }
159
160 @Test
161 public void test11MultipleSameKeys() {
162     int[] keys = { 3, 1, 2 };
163     String[] expected = {
164         " 1 - Str_1 : h=0 / parent(key)=2",
165         " 2 - Str_2 : h=1 ROOT",
166         " 3 - Str_3 : h=0 \\ parent(key)=2",
167     };
168     runTest(keys, expected);
169     avlTree.put(2, "Str_22");
170     avlTree.put(2, "Str_23");
171     expected = new String[] {
172         " 1 - Str_1 : h=0 / parent(key)=2",
173         " 2 - Str_23 : h=1 ROOT",
174         " 3 - Str_3 : h=0 \\ parent(key)=2",
175     };
176     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
177     avlTree.inorder(nodes, avlTree.getRoot());
178     verify(nodes, expected);
179 }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 4/8

```

180
181 @Test
182 public void test12RemovingCase1() {
183     // L?schen Fall 1 gem. BST-Folie 12:
184     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
185     int[] keys = { 6, 2, 9, 1, 4, 8 };
186     String[] expected = {
187         " 1 - Str_1 : h=0 / parent(key)=2",
188         " 2 - Str_2 : h=1 / parent(key)=6",
189         " 4 - Str_4 : h=0 \\ parent(key)=2",
190         " 6 - Str_6 : h=2 ROOT",
191         " 8 - Str_8 : h=0 / parent(key)=9",
192         " 9 - Str_9 : h=1 \\ parent(key)=6",
193     };
194     runTest(keys, expected);
195     assertEquals("Str_4", avlTree.remove(4));
196     expected = new String[] {
197         " 1 - Str_1 : h=0 / parent(key)=2",
198         " 2 - Str_2 : h=1 / parent(key)=6",
199         " 6 - Str_6 : h=2 ROOT",
200         " 8 - Str_8 : h=0 / parent(key)=9",
201         " 9 - Str_9 : h=1 \\ parent(key)=6",
202     };
203     avlTree.inorder(nodes, avlTree.getRoot());
204     verify(nodes, expected);
205 }
206
207 @Test
208 public void test13RemovingCase2() {
209     // L?schen Fall 2 gem. BST-Folie 13:
210     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
211     int[] keys = { 6, 2, 9, 1, 4, 8, 5 };
212     String[] expected = {
213         " 1 - Str_1 : h=0 / parent(key)=2",
214         " 2 - Str_2 : h=2 / parent(key)=6",
215         " 4 - Str_4 : h=1 \\ parent(key)=2",
216         " 5 - Str_5 : h=0 \\ parent(key)=4",
217         " 6 - Str_6 : h=3 ROOT",
218         " 8 - Str_8 : h=0 / parent(key)=9",
219         " 9 - Str_9 : h=1 \\ parent(key)=6",
220     };
221     runTest(keys, expected);
222     assertEquals("Str_4", avlTree.remove(4));
223     expected = new String[] {
224         " 1 - Str_1 : h=0 / parent(key)=2",
225         " 2 - Str_2 : h=1 / parent(key)=6",
226         " 5 - Str_5 : h=0 \\ parent(key)=2",
227         " 6 - Str_6 : h=2 ROOT",
228         " 8 - Str_8 : h=0 / parent(key)=9",
229         " 9 - Str_9 : h=1 \\ parent(key)=6",
230     };
231     avlTree.inorder(nodes, avlTree.getRoot());
232     verify(nodes, expected);
233 }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 5/8

```

234
235 @Test
236 public void test14RemovingCase3() {
237     // L?schen Fall 3 gem. BST-Folie 14:
238     // Hinweis: Baum entsprechend 'aufgef?llt' (wegen AVL!)
239     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
240     int[] keys = { 1, -10, 4, -15, -5, 2, 9, -18, -12, -7, -3, 3, 7, 10, 6 };
241     String[] expected = {
242         "-18 - Str_-18 : h=0 / parent(key)=-15",
243         "-15 - Str_-15 : h=1 / parent(key)=-10",
244         "-12 - Str_-12 : h=0 \\ parent(key)=-15",
245         "-10 - Str_-10 : h=2 / parent(key)=1",
246         "-7 - Str_-7 : h=0 / parent(key)=-5",
247         "-5 - Str_-5 : h=1 \\ parent(key)=-10",
248         "-3 - Str_-3 : h=0 \\ parent(key)=-5",
249         " 1 - Str_1 : h=4 ROOT",
250         " 2 - Str_2 : h=1 / parent(key)=4",
251         " 3 - Str_3 : h=0 \\ parent(key)=2",
252         " 4 - Str_4 : h=3 \\ parent(key)=1",
253         " 6 - Str_6 : h=0 / parent(key)=7",
254         " 7 - Str_7 : h=1 / parent(key)=9",
255         " 9 - Str_9 : h=2 \\ parent(key)=4",
256         "10 - Str_10 : h=0 \\ parent(key)=9",
257     };
258     runTest(keys, expected);
259     assertEquals("Str_4", avlTree.remove(4));
260     expected = new String[] {
261         "-18 - Str_-18 : h=0 / parent(key)=-15",
262         "-15 - Str_-15 : h=1 / parent(key)=-10",
263         "-12 - Str_-12 : h=0 \\ parent(key)=-15",
264         "-10 - Str_-10 : h=2 / parent(key)=1",
265         "-7 - Str_-7 : h=0 / parent(key)=-5",
266         "-5 - Str_-5 : h=1 \\ parent(key)=-10",
267         "-3 - Str_-3 : h=0 \\ parent(key)=-5",
268         " 1 - Str_1 : h=3 ROOT",
269         " 2 - Str_2 : h=1 / parent(key)=6",
270         " 3 - Str_3 : h=0 \\ parent(key)=2",
271         " 6 - Str_6 : h=2 \\ parent(key)=1",
272         " 7 - Str_7 : h=0 / parent(key)=9",
273         " 9 - Str_9 : h=1 \\ parent(key)=6",
274         "10 - Str_10 : h=0 \\ parent(key)=9",
275     };
276     avlTree.inorder(nodes, avlTree.getRoot());
277     verify(nodes, expected);
278 }
279
280 @Test
281 public void test15RemovingAtRoot1() {
282     int[] keys = { 1, 2, 3 };
283     String[] expected = {
284         " 1 - Str_1 : h=0 / parent(key)=2",
285         " 2 - Str_2 : h=1 ROOT",
286         " 3 - Str_3 : h=0 \\ parent(key)=2",
287     };
288     runTest(keys, expected);
289     assertEquals("Str_1", avlTree.remove(1));
290     assertEquals(2, avlTree.size());
291     assertEquals("Str_3", avlTree.remove(3));
292     assertEquals(1, avlTree.size());
293     assertEquals("Str_2", avlTree.remove(2));
294     assertEquals(0, avlTree.size());
295 }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 6/8

```

296
297 @Test
298 public void test16RemovingAtRoot2() {
299     int[] keys = { 1, 2, 3 };
300     String[] expected = {
301         " 1 - Str_1 : h=0 / parent(key)=2",
302         " 2 - Str_2 : h=1 ROOT",
303         " 3 - Str_3 : h=0 \\ parent(key)=2",
304     };
305     runTest(keys, expected);
306     assertEquals("Str_1", avlTree.remove(1));
307     assertEquals(2, avlTree.size());
308     assertEquals("Str_2", avlTree.remove(2));
309     assertEquals(1, avlTree.size());
310     assertEquals("Str_3", avlTree.remove(3));
311     assertEquals(0, avlTree.size());
312 }
313
314 @Test
315 public void test17RemovingAtRoot3() {
316     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
317     int[] keys = { 1, 2, 3 };
318     String[] expected = {
319         " 1 - Str_1 : h=0 / parent(key)=2",
320         " 2 - Str_2 : h=1 ROOT",
321         " 3 - Str_3 : h=0 \\ parent(key)=2",
322     };
323     runTest(keys, expected);
324     assertEquals("Str_2", avlTree.remove(2));
325     expected = new String[] {
326         " 1 - Str_1 : h=0 / parent(key)=3",
327         " 3 - Str_3 : h=1 ROOT",
328     };
329     avlTree.inorder(nodes, avlTree.getRoot());
330     verify(nodes, expected);
331     assertEquals(2, avlTree.size());
332     assertEquals("Str_3", avlTree.remove(3));
333     assertEquals(1, avlTree.size());
334     assertEquals("Str_1", avlTree.remove(1));
335     assertEquals(0, avlTree.size());
336 }
337
338 @Test
339 public void test18RemovingAtRoot4() {
340     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
341     int[] keys = { 3, 2, 6, 4 };
342     String[] expected = {
343         " 2 - Str_2 : h=0 / parent(key)=3",
344         " 3 - Str_3 : h=2 ROOT",
345         " 4 - Str_4 : h=0 / parent(key)=6",
346         " 6 - Str_6 : h=1 \\ parent(key)=3",
347     };
348     runTest(keys, expected);
349     assertEquals("Str_3", avlTree.remove(3));
350     expected = new String[] {
351         " 2 - Str_2 : h=0 / parent(key)=4",
352         " 4 - Str_4 : h=1 ROOT",
353         " 6 - Str_6 : h=0 \\ parent(key)=4",
354     };
355     avlTree.inorder(nodes, avlTree.getRoot());
356     verify(nodes, expected);
357 }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 7/8

```

358
359 @Test
360 public void test19RemovingAtRoot5() {
361     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
362     int[] keys = { 3, 2, 6, 1, 4, 7, 5 };
363     String[] expected = {
364         " 1 - Str_1 : h=0 / parent(key)=2",
365         " 2 - Str_2 : h=1 / parent(key)=3",
366         " 3 - Str_3 : h=3 ROOT",
367         " 4 - Str_4 : h=1 / parent(key)=6",
368         " 5 - Str_5 : h=0 \\ parent(key)=4",
369         " 6 - Str_6 : h=2 \\ parent(key)=3",
370         " 7 - Str_7 : h=0 \\ parent(key)=6",
371     };
372     runTest(keys, expected);
373     assertEquals("Str_3", avlTree.remove(3));
374     expected = new String[] {
375         " 1 - Str_1 : h=0 / parent(key)=2",
376         " 2 - Str_2 : h=1 / parent(key)=4",
377         " 4 - Str_4 : h=2 ROOT",
378         " 5 - Str_5 : h=0 / parent(key)=6",
379         " 6 - Str_6 : h=1 \\ parent(key)=4",
380         " 7 - Str_7 : h=0 \\ parent(key)=6",
381     };
382     avlTree.inorder(nodes, avlTree.getRoot());
383     verify(nodes, expected);
384 }
385
386 @Test
387 public void test20RemovingAtRoot6() {
388     int[] keys = { 1 };
389     String[] expected = {
390         " 1 - Str_1 : h=0 ROOT",
391     };
392     runTest(keys, expected);
393     assertEquals(null, avlTree.remove(8888));
394     assertEquals(1, avlTree.size());
395     runTest(keys, expected);
396     assertEquals("Str_1", avlTree.remove(1));
397     assertEquals(0, avlTree.size());
398 }
399
400 @Test
401 public void test21RemovingEntryNotInTree() {
402     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
403     int[] keys = { 1, 2, 3 };
404     String[] expected = {
405         " 1 - Str_1 : h=0 / parent(key)=2",
406         " 2 - Str_2 : h=1 ROOT",
407         " 3 - Str_3 : h=0 \\ parent(key)=2",
408     };
409     runTest(keys, expected);
410     assertNull(avlTree.remove(4));
411     expected = new String[] {
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     avlTree.inorder(nodes, avlTree.getRoot());
417     verify(nodes, expected);
418 }

```

6.10.2019 19:25:51

AVLTreeJUnitTest.java

Page 8/8

```

419
420 @Test
421 public void test22StressTest() {
422     final int SIZE = 10000;
423     Random randomGenerator = new Random(1);
424     // a Map to compare:
425     Map<Integer, String> map = new Hashtable<Integer, String>();
426     // key-Counters: count for every key how many time it was generated
427     Map<Integer, Integer> keyCounters = new Hashtable<Integer, Integer>();
428     // fill the Tree
429     for (int i = 0; i < SIZE; i++) {
430         int key = (int) (randomGenerator.nextFloat() * SIZE / 3);
431         Integer numberOfKeys = keyCounters.get(key);
432         if (numberOfKeys == null) {
433             numberOfKeys = 1;
434         } else {
435             numberOfKeys++;
436         }
437         keyCounters.put(key, numberOfKeys);
438         avlTree.put(key, "_" + i);
439         map.put(key, "_" + i);
440         assertEquals(keyCounters.size(), avlTree.size());
441         assertEquals(map.size(), avlTree.size());
442     }
443     verifyInorder();
444     // remove all Keys
445     Integer[] keyArr = new Integer[1];
446     keyArr = map.keySet().toArray(keyArr);
447     for (int key : keyArr) {
448         assertEquals(map.remove(key), avlTree.remove(key));
449         assertEquals(map.size(), avlTree.size());
450         verifyInorder();
451     }
452     assertEquals(0, avlTree.size());
453 }
454
455 private void verifyInorder() {
456     Collection<Entry<Integer, String>> inorderList = avlTree.inorder();
457     int last = Integer.MIN_VALUE;
458     for (Entry<Integer, String> entry: inorderList) {
459         Integer key = entry.getKey();
460         assertTrue(key.compareTo(last) >= 0);
461         last = key;
462     }
463 }
464
465 private void runTest(int[] keys, String[] expected) {
466     for (int key : keys) {
467         avlTree.put(key, "Str_" + key);
468     }
469     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
470     avlTree.inorder(nodes, avlTree.getRoot());
471     assertEquals(expected.length, nodes.size());
472     verify(nodes, expected);
473 }
474
475 private void verify(Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes, String[]
expected) {
476     int i = 0;
477     for (AVLTreeImpl<Integer, String>.AVLNode node: nodes) {
478         String nodeStr = node.toString();
479         String expectedStr = expected[i];
480         assertEquals(expectedStr, nodeStr);
481         i++;
482     }
483 }
484
485 }
486

```