

6.10.2024 13:58:27

AVLTree.java

Page 1/3

```

1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Sun Oct 6 13:58:27 CEST 2024
4   */
5
6  package ex04.solution.task01;
7
8  import java.util.Collection;
9
10 import ex02.solution.task01.BinarySearchTree.Entry;
11
12
13 public class AVLTree <K extends Comparable<? super K>, V> {
14
15     private AVLTreeImpl<K, V> avlTreeImpl =
16         new AVLTreeImpl<>();
17         //new AVLTreeImplADV<K, V>("AVL-Tree Ü4"); // Show in ADV
18         //new AVLTreeImplADV<K, V>("AVL-Tree Ü4", 3, 2); // Show in ADV: Fix heights
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     }
59
60     protected AVLTreeImpl<K, V> getImpl() {
61         return avlTreeImpl;
62     }

```

6.10.2024 13:58:27

AVLTree.java

Page 2/3

```

63
64     public static void main(String[] args) {
65
66         AVLTree<Integer, String> avlTree = new AVLTree<>();
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     }
94
95 }
96

```

6.10.2024 13:58:27

AVLTree.java

Page 3/3

```

97
98  /* Session-Log:
99
100 Inserting 5:
101   5 - Str_5   : h=0 ROOT
102 =====
103 Inserting 7:
104   5 - Str_5   : h=1 ROOT
105   7 - Str_7   : h=0 \ parent(key)=5
106 =====
107 Inserting 9: Single-Rotation
108   5 - Str_5   : h=0 / parent(key)=7
109   7 - Str_7   : h=1 ROOT
110   9 - Str_9   : h=0 \ parent(key)=7
111 =====
112 Inserting 3:
113   3 - Str_3   : h=0 / parent(key)=5
114   5 - Str_5   : h=1 / parent(key)=7
115   7 - Str_7   : h=2 ROOT
116   9 - Str_9   : h=0 \ parent(key)=7
117 =====
118 Inserting 1: Single-Rotation
119   1 - Str_1   : h=0 / parent(key)=3
120   3 - Str_3   : h=1 / parent(key)=7
121   5 - Str_5   : h=0 \ parent(key)=3
122   7 - Str_7   : h=2 ROOT
123   9 - Str_9   : h=0 \ parent(key)=7
124 =====
125 Inserting 4: Double-Rotation
126   1 - Str_1   : h=0 / parent(key)=3
127   3 - Str_3   : h=1 / parent(key)=5
128   4 - Str_4   : h=0 \ parent(key)=3
129   5 - Str_5   : h=2 ROOT
130   7 - Str_7   : h=1 \ parent(key)=5
131   9 - Str_9   : h=0 \ parent(key)=7
132 =====
133
134 */

```

6.10.2024 13:58:27

AVLTreeImpl.java

Page 1/6

```

1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Sun Oct 6 13:58:27 CEST 2024
4   */
5
6  package ex04.solution.task01;
7
8  import java.lang.reflect.Array;
9  import java.util.Collection;
10 import java.util.LinkedList;
11 import java.util.List;
12
13 import ex02.solution.task01.BinarySearchTree;
14
15
16 class AVLTreeImpl<K extends Comparable<? super K>, V> extends
17     BinarySearchTree<K, V> {
18
19     /**
20      * After a BST-operation, actionNode shall point to where the balance has to
21      * be checked. -> rebalance() will then be called with actionNode.
22      */
23     protected AVLNode actionNode;
24
25
26     protected class AVLNode extends BinarySearchTree<K, V>.Node {
27
28         private int height;
29         private Node parent;
30
31         AVLNode(Entry<K, V> entry) {
32             super(entry);
33         }
34
35         protected AVLNode setParent(AVLNode parent) {
36             AVLNode old = avlNode(this.parent);
37             this.parent = parent;
38             return old;
39         }
40
41         protected AVLNode getParent() {
42             return avlNode(parent);
43         }
44
45         protected int setHeight(int height) {
46             int old = this.height;
47             this.height = height;
48             return old;
49         }
50
51         protected int getHeight() {
52             return height;
53         }
54
55         @SuppressWarnings("unchecked")
56         @Override
57         public void setLeftChild(BinarySearchTree<K, V>.Node leftChild) {
58             super.setLeftChild(leftChild);
59             if (leftChild != null) {
60                 ((AVLNode) leftChild).setParent(this);
61             }
62         }
63
64         @Override
65         public AVLNode getLeftChild() {
66             return avlNode(super.getLeftChild());
67         }
68     }

```

6.10.2024 13:58:27

AVLTreImpl.java

Page 2/6

```

68
69     @SuppressWarnings("unchecked")
70     @Override
71     public void setRightChild(BinarySearchTree<K, V>.Node rightChild) {
72         super.setRightChild(rightChild);
73         if (rightChild != null) {
74             ((AVLNode)rightChild).setParent(this);
75         }
76     }
77
78     @Override
79     public AVLNode getRightChild() {
80         return avlNode(super.getRightChild());
81     }
82
83     @Override
84     public String toString() {
85         String result = String.format("%2d - %-6s : h=%d",
86             getEntry().getKey(), getEntry().getValue(), height);
87         if (parent == null) {
88             result += " ROOT";
89         } else {
90             boolean left = (parent.getLeftChild() == this) ? true : false;
91             result += (left ? " / " : " \\ ") + "parent(key)="
92                 + parent.getEntry().getKey();
93         }
94         return result;
95     }
96
97 } // End of class AVLNode
98
99
100 protected AVLNode getRoot() {
101     return avlNode(root);
102 }
103
104 public V put(K key, V value) {
105     Entry<K, V> entry = find(key);
106     if (entry != null) {
107         // key already exists in the Tree
108         return entry.setValue(value);
109     }
110     // key does not exist in the Tree yet
111     super.insert(key, value);
112     rebalance(actionNode);
113     actionNode = null;
114     return null;
115 }
116
117 public V get(K key) {
118     Entry<K, V> entry = super.find(key);
119     if (entry == null) {
120         return null;
121     }
122     return entry.getValue();
123 }

```

6.10.2024 13:58:27

AVLTreImpl.java

Page 3/6

```

124
125     @Override
126     protected Node insert(Node node, Entry<K, V> entry) {
127         if (node != null) {
128             actionNode = avlNode(node);
129         }
130         // calling now the BST-insert() which will do the work:
131         AVLNode result = avlNode(super.insert(node, entry));
132         if (node == null) {
133             // In this case: result of super.insert() is the new node!
134             result.setParent(actionNode);
135         }
136         return result;
137     }
138
139     /**
140     * The height of the tree.
141     *
142     * @return The current height. -1 for an empty tree.
143     */
144     @Override
145     public int getHeight() {
146         return height(avlNode(root));
147     }
148
149     /**
150     * Returns the height of this node.
151     *
152     * @param node
153     * @return The height or -1 if null.
154     */
155     @SuppressWarnings("static-method")
156     protected int height(AVLNode node) {
157         return (node != null) ? node.getHeight() : -1;
158     }
159
160     /**
161     * Restructures the tree with rotations.
162     *
163     * @param xPos
164     *         The X-node.
165     * @return The new root-node of this subtree.
166     */
167     protected AVLNode restructure(AVLNode xPos) {
168         AVLNode yPos = xPos.getParent();
169         AVLNode zPos = yPos.getParent();
170         AVLNode newSubTreeRoot = null;
171         if (yPos == zPos.getLeftChild()) {
172             if (xPos == yPos.getLeftChild()) {
173                 newSubTreeRoot = rotateWithLeftChild(zPos);
174             } else {
175                 newSubTreeRoot = doubleRotateWithLeftChild(zPos);
176             }
177         } else {
178             if (xPos == yPos.getRightChild()) {
179                 newSubTreeRoot = rotateWithRightChild(zPos);
180             } else {
181                 newSubTreeRoot = doubleRotateWithRightChild(zPos);
182             }
183         }
184         return newSubTreeRoot;
185     }

```

6.10.2024 13:58:27

AVLTreImpl.java

Page 4/6

```

186
187     protected AVLNode tallerChild(AVLNode node) {
188         AVLNode result;
189         if (height(node.getLeftChild()) >= height(node.getRightChild())) {
190             result = node.getLeftChild();
191         } else {
192             result = node.getRightChild();
193         }
194         return result;
195     }
196
197     protected AVLNode rotateWithLeftChild(AVLNode k2) {
198         AVLNode parentSubtree = k2.getParent();
199         AVLNode k1 = k2.getLeftChild();
200         k2.setLeftChild(k1.getRightChild());
201         k1.setRightChild(k2);
202         adjustSubtreeParent(k2, k1, parentSubtree);
203         return k1;
204     }
205
206     protected AVLNode doubleRotateWithLeftChild(AVLNode k3) {
207         //k3.setLeftChild(rotateWithRightChild(k3.getLeftChild()));
208         // -> k3.setLeftChild() is done/ensured in adjustSubtreeParent():
209         rotateWithRightChild(k3.getLeftChild());
210         return rotateWithLeftChild(k3);
211     }
212
213     protected AVLNode rotateWithRightChild(AVLNode k1) {
214         AVLNode parentSubtree = k1.getParent();
215         AVLNode k2 = k1.getRightChild();
216         k1.setRightChild(k2.getLeftChild());
217         k2.setLeftChild(k1);
218         adjustSubtreeParent(k1, k2, parentSubtree);
219         return k2;
220     }
221
222     protected AVLNode doubleRotateWithRightChild(AVLNode k3) {
223         //k3.setRightChild(rotateWithLeftChild(k3.getRightChild()));
224         // -> k3.setRightChild() is done/ensured in adjustSubtreeParent():
225         rotateWithLeftChild(k3.getRightChild());
226         return rotateWithRightChild(k3);
227     }
228
229     /**
230     * Assures the connection between a restructured subtree and the parent of
231     * this subtree.
232     * Used after rotations.
233     *
234     * @param oldSubtreeRoot
235     *     The old root-node of this subtree.
236     * @param newSubtreeRoot
237     *     The new root-node of this subtree.
238     * @param parentSubtree
239     *     The parent-node of this subtree.
240     */
241     protected void adjustSubtreeParent(AVLNode oldSubtreeRoot,
242         AVLNode newSubtreeRoot, AVLNode parentSubtree) {
243         if (parentSubtree != null) {
244             if (parentSubtree.getLeftChild() == oldSubtreeRoot) {
245                 parentSubtree.setLeftChild(newSubtreeRoot);
246             } else {
247                 parentSubtree.setRightChild(newSubtreeRoot);
248             }
249         } else { // newSubtreeRoot is now also the root of the whole tree
250             root = newSubtreeRoot;
251             newSubtreeRoot.setParent(null);
252         }
253     }

```

6.10.2024 13:58:27

AVLTreImpl.java

Page 5/6

```

254
255     protected boolean isBalanced(AVLNode node) {
256         int bf = height(node.getLeftChild()) - height(node.getRightChild());
257         return (-1 <= bf) && (bf <= 1);
258     }
259
260     /**
261     * Assures the balance of the tree from 'node' up to the root.
262     *
263     * @param node
264     *     The node from where to start.
265     */
266     protected void rebalance(AVLNode node) {
267         while (node != null) {
268             setHeight(node);
269             if (!isBalanced(node)) {
270                 AVLNode xPos = tallerChild(tallerChild(node));
271                 node = restructure(xPos);
272                 setHeight(node.getLeftChild());
273                 setHeight(node.getRightChild());
274                 setHeight(node);
275             }
276             node = node.getParent();
277         }
278     }
279
280     /**
281     * Assures the correct height for node.
282     *
283     * @param node
284     *     The node to assure its height.
285     */
286     protected void setHeight(AVLNode node) {
287         if (node == null) {
288             return;
289         }
290         int heightLeftChild = height(node.getLeftChild());
291         int heightRightChild = height(node.getRightChild());
292         node.setHeight(1 + Math.max(heightLeftChild, heightRightChild));
293     }
294
295     /**
296     * Factory-Method. Creates a new node.
297     *
298     * @param entry
299     *     The entry to be inserted in the new node.
300     * @return The new created node.
301     */
302     @Override
303     protected Node newNode(Entry<K, V> entry) {
304         return new AVLNode(entry);
305     }
306
307     public V remove(K key) {
308         Entry<K, V> entry = find(key);
309         if (entry == null) {
310             return null;
311         }
312         // calling now the BST-remove(Entry) which will do the work:
313         super.remove(entry);
314         if (actionNode != null) {
315             assureParentForChilds(actionNode);
316             rebalance(actionNode);
317             actionNode = null;
318         }
319         return entry.getValue();
320     }

```

6.10.2024 13:58:27

AVLTreeImpl.java

Page 6/6

```

321
322 @Override
323 protected RemoveResult remove(Node node, Entry<K, V> entry) {
324     if (node.getEntry() == entry) {
325         actionNode = avlNode(node).getParent();
326     }
327     // calling now the BST-remove(Node, Entry) which will do the work:
328     return super.remove(node, entry);
329 }
330
331 @Override
332 protected Node getParentNext(Node p) {
333     actionNode = avlNode(super.getParentNext(p));
334     return actionNode;
335 }
336
337 @SuppressWarnings("static-method")
338 protected void assureParentForChilds(AVLNode parent) {
339     @SuppressWarnings("unchecked")
340     AVLNode[] childs = (AVLNode[])Array.newInstance(parent.getClass(), 2);
341     childs[0] = parent.getLeftChild();
342     childs[1] = parent.getRightChild();
343     for (AVLNode child : childs) {
344         if (child != null) {
345             child.setParent(parent);
346         }
347     }
348 }
349
350 @Override
351 protected void inorder(Node node, Collection<Node> inorderList) {
352     super.inorder(node, inorderList);
353 }
354
355 // Type-Casting: Node -> AVLNode (Cast-Encapsulation)
356 @SuppressWarnings({ "unchecked", "static-method" })
357 protected AVLNode avlNode(Node node) {
358     return (AVLNode)node;
359 }
360
361 public void print() {
362     List<Node> nodeList = new LinkedList<>();
363     inorder(root, nodeList);
364     for (Node node: nodeList) {
365         System.out.println(node + " ");
366     }
367 }
368
369 }
370
371

```

6.10.2024 13:58:27

AVLTreeJUnitTest.java

Page 1/8

```

1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Sun Oct 6 13:58:27 CEST 2024
4   */
5
6  package ex04.solution.task01;
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 ex02.solution.task01.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<>();
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     }

```

6.10.2024 13:58:27

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 }

```

6.10.2024 13:58:27

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>.Node> nodes = new LinkedList<>();
177     avlTree.inorder(avlTree.getRoot(), nodes);
178     verify(nodes, expected);
179 }

```

6.10.2024 13:58:27

AVLTreeJUnitTest.java

Page 4/8

```

180
181 @Test
182 public void test12RemovingCase1() {
183     // Löschen Fall 1 gem. BST-Folie 12:
184     Collection<AVLTreeImpl<Integer, String>.Node> 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(avlTree.getRoot(), nodes);
204     verify(nodes, expected);
205 }
206
207 @Test
208 public void test13RemovingCase2() {
209     // Löschen Fall 2 gem. BST-Folie 13:
210     Collection<AVLTreeImpl<Integer, String>.Node> 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(avlTree.getRoot(), nodes);
232     verify(nodes, expected);
233 }

```

6.10.2024 13:58:27

AVLTreeJUnitTest.java

Page 5/8

```

234
235 @Test
236 public void test14RemovingCase3() {
237     // Löschen Fall 3 gem. BST-Folie 14:
238     // Hinweis: Baum entsprechend 'aufgefüllt' (wegen AVL!)
239     Collection<AVLTreeImpl<Integer, String>.Node> 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(avlTree.getRoot(), nodes);
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 }

```

6.10.2024 13:58:27

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>.Node> 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(avlTree.getRoot(), nodes);
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>.Node> 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(avlTree.getRoot(), nodes);
356     verify(nodes, expected);
357 }

```

6.10.2024 13:58:27

AVLTreeJUnitTest.java

Page 7/8

```

358
359 @Test
360 public void test19RemovingAtRoot5() {
361     Collection<AVLTreeImpl<Integer, String>.Node> 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(avlTree.getRoot(), nodes);
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>.Node> 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(avlTree.getRoot(), nodes);
417     verify(nodes, expected);
418 }

```


6.10.2024 13:58:27

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<>();
426     // key-Counters: count for every key how many time it was generated
427     Map<Integer, Integer> keyCounters = new Hashtable<>();
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>.Node> nodes = new LinkedList<>();
470     avlTree.inorder(avlTree.getRoot(), nodes);
471     assertEquals(expected.length, nodes.size());
472     verify(nodes, expected);
473 }
474
475 private static void verify(Collection<AVLTreeImpl<Integer, String>.Node> nodes, String[] expected) {
476     int i = 0;
477     for (AVLTreeImpl<Integer, String>.Node node: nodes) {
478         String nodeStr = node.toString();
479         String expectedStr = expected[i];
480         assertEquals(expectedStr, nodeStr);
481         i++;
482     }
483 }
484
485 }
486

```