

24.9.2018 12:08:35

## BinarySearchTree.java

Page 1/4

```

1  /*
2   * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Mon Sep 24 12:08:35 CEST 2018
4   */
5
6  package uebung02.as.aufgabe01;
7
8  import java.util.Collection;
9  import java.util.LinkedList;
10
11 public class BinarySearchTree<K extends Comparable<? super K>, V> {
12
13     protected Node root;
14
15     public static class Entry<K, V> {
16
17         private K key;
18         private V value;
19
20         public Entry(K key, V value) {
21             this.key = key;
22             this.value = value;
23         }
24
25         protected K setKey(K key) {
26             K oldKey = this.key;
27             this.key = key;
28             return oldKey;
29         }
30
31         public K getKey() {
32             return key;
33         }
34
35         public V setValue(V value) {
36             V oldValue = this.value;
37             this.value = value;
38             return oldValue;
39         }
40
41         public V getValue() {
42             return value;
43         }
44
45         @Override
46         public String toString() {
47             StringBuilder result = new StringBuilder();
48             result.append("[").append(key).append("/").append(value).append("]");
49             return result.toString();
50         }
51     } // End of class Entry
52
53     protected class Node {
54
55         private Entry<K, V> entry;
56         private Node leftChild;
57         private Node rightChild;
58
59         public Node(Entry<K, V> entry) {
60             this.entry = entry;
61         }
62
63         public Node(Entry<K, V> entry, Node leftChild, Node rightChild) {
64             this.entry = entry;
65             this.leftChild = leftChild;
66             this.rightChild = rightChild;
67         }
68     }

```

24.9.2018 12:08:35

## BinarySearchTree.java

Page 2/4

```

69
70     public Entry<K, V> getEntry() {
71         return entry;
72     }
73
74     public Entry<K, V> setEntry(Entry<K, V> entry) {
75         Entry<K, V> oldEntry = entry;
76         this.entry = entry;
77         return oldEntry;
78     }
79
80     public Node getLeftChild() {
81         return leftChild;
82     }
83
84     public void setLeftChild(Node leftChild) {
85         this.leftChild = leftChild;
86     }
87
88     public Node getRightChild() {
89         return rightChild;
90     }
91
92     public void setRightChild(Node rightChild) {
93         this.rightChild = rightChild;
94     }
95
96 } // End of class Node
97
98     public Entry<K, V> insert(K key, V value) {
99         // TODO Implement here...
100         return null;
101     }
102
103     /**
104     * Factory-Method: Creates a new node.
105     *
106     * @param entry
107     *     The entry to be inserted in the new node.
108     * @return The new created node.
109     */
110     protected Node newNode(Entry<K, V> entry) {
111         return new Node(entry);
112     }
113
114     public void clear() {
115         // TODO Implement here...
116     }
117
118     public Entry<K, V> find(K key) {
119         // TODO Implement here...
120         return null;
121     }
122
123     /**
124     * Returns a collection with all entries with key.
125     *
126     * @param key
127     *     The key to be searched.
128     * @return Collection of all entries found. An empty collection is returned if
129     *     no entry with key is found.
130     */
131     public Collection<Entry<K, V>> findAll(K key) {
132         // TODO Implement here...
133         return null;
134     }

```

24.9.2018 12:08:35

## BinarySearchTree.java

Page 3/4

```

135
136 /**
137  * Returns a collection with all entries in inorder.
138  * @return Inorder-Collection of all entries.
139  */
140 public Collection<Entry<K, V>> inorder() {
141     // TODO Implement here...
142     return null;
143 }
144
145 /**
146  * Prints the entries of the tree as a list in inorder to the console.
147  */
148 public void printInorder() {
149     // TODO Implement here...
150 }
151
152 public Entry<K, V> remove(Entry<K, V> entry) {
153     // TODO Implement here...
154     return null;
155 }
156
157 /**
158  * The height of the tree.
159  * @return The actual height. -1 for an empty tree.
160  */
161 public int getHeight() {
162     // TODO Implement here...
163     return -1;
164 }
165
166 public int size() {
167     // TODO Implement here...
168     return -1;
169 }
170
171 public boolean isEmpty() {
172     // TODO Implement here...
173     return true;
174 }
175
176

```

24.9.2018 12:08:35

## BinarySearchTree.java

Page 4/4

```

177
178 public static void main(String[] args) {
179
180     // Example from lecture "L?schen (IV/IV)":
181     BinarySearchTree<Integer, String> bst = new BinarySearchTree<>();
182     //BinarySearchTree<Integer, String> bst = new BinarySearchTreeGVS<>();
183     System.out.println("Inserting:");
184     bst.insert(1, "Str1");
185     bst.printInorder();
186     bst.insert(3, "Str3");
187     bst.printInorder();
188     bst.insert(2, "Str2");
189     bst.printInorder();
190     bst.insert(8, "Str8");
191     bst.printInorder();
192     bst.insert(9, "Str9");
193     bst.insert(6, "Str6");
194     bst.insert(5, "Str5");
195     bst.printInorder();
196
197     System.out.println("Removeing 3:");
198     Entry<Integer, String> entry = bst.find(3);
199     System.out.println(entry);
200     bst.remove(entry);
201     bst.printInorder();
202
203     //if (bst instanceof BinarySearchTreeGVS) {
204     //    ((BinarySearchTreeGVS<Integer, String>)bst).gvsTree.disconnect();
205     //}
206
207 }
208
209 /* Session-Log:
210
211 Inserting:
212 [1/Str1]
213 [1/Str1] [3/Str3]
214 [1/Str1] [2/Str2] [3/Str3]
215 [1/Str1] [2/Str2] [3/Str3] [8/Str8]
216 [1/Str1] [2/Str2] [3/Str3] [5/Str5] [6/Str6] [8/Str8] [9/Str9]
217 Removeing 3:
218 [3/Str3]
219 [1/Str1] [2/Str2] [5/Str5] [6/Str6] [8/Str8] [9/Str9]
220
221 */
222
223
224 } // End of class BinarySearchTree
225

```

24.9.2018 12:08:35

## BinarySearchTreeTest.java

Page 1/2

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Sep 24 12:08:35 CEST 2018
4  */
5
6  package uebung02.as.aufgabe01;
7
8  import java.util.Iterator;
9  import java.util.Random;
10
11  import uebung02.as.aufgabe01.BinarySearchTree.Entry;
12
13  public class BinarySearchTreeTest {
14
15      private static Random randomGenerator = new Random(1);
16
17      private static BinarySearchTree<Integer, String> generateTree(int nodes) {
18          int key;
19          BinarySearchTree<Integer, String> ret = new BinarySearchTree<>();
20          for (int i = 0; i < nodes; i++) {
21              key = randomGenerator.nextInt() * Integer.MAX_VALUE;
22              ret.insert(key, "String_" + i);
23          }
24          return ret;
25      }
26
27      public static void main(String[] args) {
28          System.out.println("BINARY TREE TEST");
29          System.out
30              .println("Please be patient, the following operations may take some time...");
31          final int TESTRUNS = 100;
32          final int BEGINSIZE = 10000;
33          final int VARYSIZE = 10;
34          long startTime = System.currentTimeMillis();
35
36          BinarySearchTree<Integer, String> bst = new BinarySearchTree<>();
37          double avgHeight = 0;
38          double avgEntries = 0;
39          double avgTime = 0;
40          for (int i = 0; i < TESTRUNS; i++) {
41              startTime = System.currentTimeMillis();
42              bst = generateTree(BEGINSIZE + i * VARYSIZE);
43              avgTime += System.currentTimeMillis() - startTime;
44              avgHeight += bst.getHeight();
45              avgEntries += BEGINSIZE + i * VARYSIZE;
46          }
47          avgTime /= TESTRUNS;
48          avgEntries /= TESTRUNS;
49          avgHeight /= TESTRUNS;
50          System.out.println("Test successful, results are as follows:");
51          System.out.println("Average time for generation is: " + avgTime + "ms");
52          System.out.println("Average entries are: " + avgEntries);
53          System.out.println("Average height is: " + avgHeight);
54          System.out.println("In  $h = C \cdot \log_2(n)$ ,  $C = h / \log_2(n) =$ " + avgHeight
55              / (Math.log(avgEntries) / Math.log(2)));
56          System.out.println();

```

24.9.2018 12:08:35

## BinarySearchTreeTest.java

Page 2/2

```

57
58      bst = generateTree(20);
59      int search = 15138431;
60      Entry<Integer, String> searchResult;
61      bst.insert(search, "String_" + search);
62      searchResult = bst.find(search);
63      if (searchResult == null) {
64          System.err.println("Search for node " + search + " failed!");
65      } else {
66          System.out.println("Search for node " + search + " successful!");
67      }
68      System.out.println();
69      bst.insert(search, "String_" + search);
70      bst.insert(search, "String_" + search);
71      bst.insert(search, "String_" + search);
72      Iterator<Entry<Integer, String>> it = bst.findAll(search).iterator();
73      int count = 0;
74      while (it.hasNext()) {
75          count++;
76          it.next();
77          System.out.println("Search for node " + search + " successful!");
78      }
79      System.out.println("Search for node " + search + ": " + count
80          + " nodes found!");
81      System.out.println();
82      it = bst.findAll(search).iterator();
83      count = 0;
84      while (it.hasNext()) {
85          bst.remove(it.next());
86      }
87
88      it = bst.findAll(search).iterator();
89      count = 0;
90      while (it.hasNext()) {
91          count++;
92          it.next();
93          System.out.println("Search for node " + search + " successful!");
94      }
95      System.out.println("Search for node " + search + ": " + count
96          + " nodes found!");
97      }
98  }
99
100  /* Session-Log:
101
102  103  BINARY TREE TEST
104  104  Please be patient, the following operations may take some time...
105  105  Test successful, results are as follows:
106  106  Average time for generation is: 9.07ms
107  107  Average entries are: 10495.0
108  108  Average height is: 30.81
109  109  In  $h = C \cdot \log_2(n)$ ,  $C = h / \log_2(n) = 2.306584099301782$ 
110
111  111  Search for node 15138431 successful!
112
113  113  Search for node 15138431 successful!
114  114  Search for node 15138431 successful!
115  115  Search for node 15138431 successful!
116  116  Search for node 15138431 successful!
117  117  Search for node 15138431: 4 nodes found!
118
119  119  Search for node 15138431: 0 nodes found!
120
121  */
122

```

24.9.2018 12:08:35

## BinarySearchTreeJUnitTest.java

Page 1/4

```

1  /*
2   * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Mon Sep 24 12:08:35 CEST 2018
4   */
5
6  package uebung02.as.aufgabe01;
7
8  import static org.junit.Assert.*;
9
10 import java.util.Collection;
11 import java.util.HashMap;
12 import java.util.LinkedList;
13 import java.util.List;
14 import java.util.Map;
15 import java.util.Random;
16
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.as.aufgabe01.BinarySearchTree.Entry;
23
24 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
25 public class BinarySearchTreeJUnitTest {
26
27     BinarySearchTree<Integer, String> bst;
28
29     @Before
30     public void setUp() {
31         bst = new BinarySearchTree<Integer, String>();
32     }
33
34     @Test
35     public void test01EmptySizeInsertClear() {
36         assertTrue(bst.isEmpty());
37         assertEquals(0, bst.size());
38         bst.insert(1, "String_1");
39         assertEquals(1, bst.size());
40         assertFalse(bst.isEmpty());
41         bst.insert(2, "String_2");
42         assertEquals(2, bst.size());
43         bst.insert(2, "String_2");
44         assertEquals(3, bst.size());
45         bst.clear();
46         assertTrue(bst.isEmpty());
47         assertEquals(0, bst.size());
48     }
49
50     @Test
51     public void test02Find() {
52         Entry<Integer, String> entry;
53         entry = bst.find(1);
54         assertNull(entry);
55         Entry<Integer, String> insertedEntry = bst.insert(1, "String_1");
56         entry = bst.find(1);
57         assertNotNull(entry);
58         assertEquals(Integer.valueOf(1), entry.getKey());
59         assertEquals("String_1", entry.getValue());
60         assertEquals(insertedEntry, entry);
61     }

```

24.9.2018 12:08:35

## BinarySearchTreeJUnitTest.java

Page 2/4

```

62
63     @Test
64     public void test03FindAll() {
65         Collection<Entry<Integer, String>> col;
66         col = bst.findAll(1);
67         assertEquals(0, col.size());
68         bst.insert(1, "String_1");
69         col = bst.findAll(2);
70         assertEquals(0, col.size());
71         bst.insert(2, "String_2");
72         col = bst.findAll(2);
73         assertEquals(1, col.size());
74         bst.insert(2, "String_2");
75         col = bst.findAll(2);
76         assertEquals(2, col.size());
77     }
78
79     @Test
80     public void test04GetHeight() {
81         assertEquals(-1, bst.getHeight());
82         bst.insert(1, "String_1");
83         assertEquals(0, bst.getHeight());
84         bst.insert(2, "String_2");
85         assertEquals(1, bst.getHeight());
86     }
87
88     @Test
89     public void test05Remove() {
90         Entry<Integer, String> entry = new Entry<>(1, "String_1");
91         entry = bst.remove(entry);
92         assertNull(entry);
93         final Entry<Integer, String> entry1 = bst.insert(1, "String_1");
94         entry = bst.remove(entry1);
95         assertEquals(entry, entry1);
96         assertEquals(0, bst.size());
97         final Entry<Integer, String> entry1a = bst.insert(1, "String_1a");
98         final Entry<Integer, String> entry1b = bst.insert(1, "String_1b");
99         assertEquals(2, bst.size());
100        entry = bst.remove(entry1a);
101        assertEquals(entry1a, entry);
102        assertEquals(1, bst.size());
103        entry = bst.remove(entry1b);
104        assertEquals(entry1b, entry);
105        assertEquals(0, bst.size());
106    }

```

24.9.2018 12:08:35

BinarySearchTreeJUnitTest.java

Page 3/4

```

107
108 @Test
109 public void test06RemoveCase3() {
110     bst.insert(1, "String_1");
111     Entry<Integer, String> entryToRemove = bst.insert(3, "String_3");
112     bst.insert(2, "String_2");
113     bst.insert(8, "String_8");
114     bst.insert(6, "String_6");
115     bst.insert(9, "String_9");
116     bst.insert(5, "String_5");
117     assertEquals(7, bst.size());
118     assertEquals(4, bst.getHeight());
119     Entry<Integer, String> removedEntry = bst.remove(entryToRemove);
120     assertEquals(entryToRemove, removedEntry);
121     assertEquals(6, bst.size());
122     assertEquals(3, bst.getHeight());
123     bst.remove(bst.find(6));
124     assertEquals(5, bst.size());
125     assertEquals(3, bst.getHeight());
126     bst.remove(bst.find(9));
127     assertEquals(4, bst.size());
128     assertEquals(2, bst.getHeight());
129 }
130
131 @Test
132 public void test07RemoveCase3Special() {
133     bst.insert(2, "String_2");
134     bst.insert(1, "String_1");
135     bst.insert(3, "String_3.1");
136     bst.insert(3, "String_3.2");
137     Collection<Entry<Integer, String>> col;
138     col = bst.findAll(3);
139     assertEquals(2, col.size());
140     Entry<Integer, String> removedEntry = bst.remove(bst.find(2));
141     assertNotNull(removedEntry);
142     assertEquals("String_2", removedEntry.getValue());
143     col = bst.findAll(3);
144     assertEquals(2, col.size());
145 }
146
147 @Test
148 public void test09StressTest() {
149     final int SIZE = 10000;
150     Random randomGenerator = new Random();
151     List<Entry<Integer, String>> entriesList = new LinkedList<>();
152     // key-Counters: count for every key how many time it was generated
153     Map<Integer, Integer> keyCounters = new HashMap<>();
154     // fill the Tree
155     for (int i = 0; i < SIZE; i++) {
156         int key = (int) (randomGenerator.nextFloat() * SIZE / 3);
157         Integer numberOfKeys = keyCounters.get(key);
158         if (numberOfKeys == null) {
159             numberOfKeys = 1;
160         } else {
161             numberOfKeys++;
162         }
163         keyCounters.put(key, numberOfKeys);
164         Entry<Integer, String> entry = bst.insert(key, "String_" + i);
165         entriesList.add(entry);
166         assertEquals(i + 1, bst.size());
167     }
168     // verify the number of entries per key
169     for (Map.Entry<Integer, Integer> keyEntry : keyCounters.entrySet()) {
170         int key = keyEntry.getKey();
171         int numberOfKeys = keyEntry.getValue();
172         assertEquals(numberOfKeys, bst.findAll(key).size());
173     }

```

24.9.2018 12:08:35

BinarySearchTreeJUnitTest.java

Page 4/4

```

174
175     // remove all entries
176     int size = bst.size();
177     for (Entry<Integer, String> entry : entriesList) {
178         Entry<Integer, String> deletedEntry = bst.remove(entry);
179         assertEquals(entry, deletedEntry);
180         assertEquals(--size, bst.size());
181     }
182 }
183
184 }
185

```