Aufgaben mit dem TreeEditor – Lösungen

2. Aufgaben für Binäre Suchbäume

a) Mit *inorder*-Traversierung, damit die Wurzel zwischen den Teilbäumen steht:

```
private String toString(Node<K, E> n) {
 if (n != null)
  return "[" + toString(n.left) + n.key + toString(n.right) + "]";
 else return "";
}
```

b)
```
public void remove(K key) { root = remove(root, key); }

private Node<K, E> remove(Node<K, E> node, K key) {
 if (node != null) {
  int c = key.compareTo(node.getKey());
  if (c < 0) node.left = remove(node.getLeft(), key);
  else if (c > 0) node.right = remove(node.getRight(), key);
  else if (node.getRight() == null) { node = node.getLeft(); --nodeCount; }
  else if (node.getLeft() == null) { node = node.getRight(); --nodeCount; }
  else {
   Node<K, E> n = node.getRight(), p = null;
   while (n.getLeft() != null) { p = n; n = n.getLeft(); }
   if (p != null) {
    p.left = n.getRight(); n.left = node.getLeft(); n.right = node.getRight();
   } else {
    n.left = node.getLeft();
   }
   node = n;
   --nodeCount;
  }
 }
 return node;
}
```

3. Aufgaben für AVL-Bäume

a)
```
private Node<K, E> rotateR(Node<K, E> n) {
 Node<K, E> n1 = n.left;
 n.left = n1.right;
 n1.right = n;
 return n1;
}

private Node<K, E> rotateL(Node<K, E> n) {
 Node<K, E> n1 = n.right;
 n.right = n1.left;
 n1.left = n;
 return n1;
}
```

b)
```
private Node<K, E> rotateLR(Node<K, E> n) {
 n.left = rotateL(n.left);
 return rotateR(n);
}

public String toString() {
 if (root != null) return toString(root);
 else return "[]";
}
```

c)
```
private Node<K, E> insert(Node<K, E> p, K key, E element) {
 if (p == null) {
  nodeCount++;
  return new Node<K, E>(key, element);
 } else {
  int c = key.compareTo(p.key);
  if (c < 0) {
   p.left = insert(p.left, key, element);
   if (height(p.right) - height(p.left) == -2) {
    if (height(p.left.left) < height(p.left.right)) p = rotateLR(p);
    else p = rotateR(p);
   }
  } else if (c > 0) {
   p.right = insert(p.right, key, element);
   if (height(p.right) - height(p.left) == 2) {
    if (height(p.right.right) < height(p.right.left)) p = rotateRL(p);
    else p = rotateL(p);
   }
  } else p.element = element;
  return p;
 }
}
```

d)
```
private Node<K, E> remove(Node<K, E> node, K key) {
 if (node != null) {
  int c = key.compareTo(node.key);
  if (c < 0) node.left = remove(node.left, key);
  else if (c > 0) node.right = remove(node.right, key);
  else if (node.right == null) { node = node.left; nodeCount--; }
  else if (node.left == null) { node = node.right; nodeCount--; }
  else {
   Node<K, E> n1 = node.right;
   while (n1.left != null) {
    n1 = n1.left;
   }
   n1.right = remove(node.right, n1.key);
   n1.left = node.left;
   node = n1;
  }
 }
 if (node != null) {
  if (height(node.right) - height(node.left) == -2) {
   if (height(node.left.left) < height(node.left.right))
    node = rotateLR(node);
   else
    node = rotateR(node);
  }
  if (height(node.right) - height(node.left) == 2) {
   if (height(node.right.right) < height(node.right.left))
    node = rotateRL(node);
   else
    node = rotateL(node);
  }
 }
 return node;
}
```