**Protokoll:**

Rekursive Funktionen:

* insert(node, key)
* height(node)
* balanceFactor(node)
* traverseAndCheckAVL(node, isAVL)
* traverseAndCollectKeys(node)
* printPathToKey(node, key, path)
* checkSubtreeStructure(node, subtreeStructure)

Abbruchbedingung:

* Für alle Funktionen: node == null

Parameter:

* TreeNode node, int key
* TreeNode node
* TreeNode node
* TreeNode node, BooleanWrapper isAVL
* TreeNode node
* TreeNode node, int key, List<Integer> path
* TreeNode node, List<Integer> subtreeStructure

Rückgabewert:

* TreeNode
* int
* int
* void
* List<Integer>
* boolean
* boolean

**Aufwandsabschätzung:**

Die rekursiven Funktionen in dieser Implementierung sind hauptsächlich Baum-Traversierungen. Die Laufzeit dieser Funktionen hängt von der Höhe des Baums ab. Im besten Fall (vollständig ausgewogener AVL-Baum) beträgt die Laufzeit O(log N), im schlimmsten Fall (entarteter Baum) ist sie O(N), wobei N die Anzahl der Knoten im Baum ist.

* insert(node, key) - O(log N) (best case) / O(N) (worst case)
* height(node) - O(log N) (best case) / O(N) (worst case)
* balanceFactor(node) - O(log N) (best case) / O(N) (worst case)
* traverseAndCheckAVL(node, isAVL) - O(N)
* traverseAndCollectKeys(node) - O(N)
* printPathToKey(node, key, path) - O(log N) (best case) / O(N) (worst case)
* checkSubtreeStructure(node, subtreeStructure) - O(N\*M), wobei M die Größe der subtreeStructure ist.