**Beschreibung der Rekursiven Funktionen:  
  
findMin(node):**

Startet bei dem gegebenen Knoten und durchläuft den linken Teilbaum, um den kleinsten Schlüssel zu finden.

**findMax(node):**

Startet bei dem gegebenen Knoten und durchläuft den rechten Teilbaum, um den größten Schlüssel zu finden.

**checkAVL(node):**

Überprüft, ob der Unterbaum des gegebenen Knotens die Eigenschaften eines AVL-Baums erfüllt, also ob der Balancefaktor ≤ 1 ist.

**getHeight(node):**

Berechnet die Höhe des Unterbaums von dem gegebenen Knoten aus.

**getBalance(node):**

Berechnet den Balancefaktor eines Knotens, indem die Höhe des rechten Teilbaums von der Höhe des linken Teilbaums subtrahiert wird.

**findValuePath(node, key, path = []):**

Findet rekursiv den Pfad zu einem Schlüsselwert im Baum und gibt diesen Pfad als Array zurück.

**findSubtree(node, subtreeRoot):**

Überprüft rekursiv, ob ein bestimmter Subtree in der Struktur des Hauptbaums enthalten ist.

**rotateRight(y), rotateLeft(x):**

Führen die entsprechenden Rotationen im AVL-Baum aus, um die Balance des Baums zu erhalten oder wiederherzustellen.

**Aufwandsabschätzung:**

**findMin, findMax:**

Bei einem balancierten Baum durchlaufen beide Funktionen ungefähr die Hälfte des Baums und benötigen daher durchschnittlich O(log n). Wenn der Baum nicht balanciert ist, kann die Laufzeit im schlechtesten Fall O(n) betragen.

**checkAVL:**

Da diese Funktion alle Knoten des Baums durchlaufen muss, hat sie eine Laufzeit von O(n).

**insert:**

Hat durchschnittlich eine Laufzeit von O(log n), da im Regelfall nur ein Teil des Baums durchsucht wird, um den passenden Einfügepunkt zu finden. Das Einfügen selbst benötigt konstante Zeit. Das Aktualisieren der Höhen und das Durchführen von Rotationen (falls notwendig) haben ebenfalls im Durchschnitt eine Laufzeit von O(log n).

**findValuePath:**

Ähnlich wie findMin und findMax, hat diese Methode eine Laufzeit von O(log n), wenn der Baum balanciert ist, da sie einem Pfad folgt.

**findSubtree:**

Die Komplexität dieser Methode hängt von der Größe des Hauptbaums und des Subtrees ab. Im Durchschnitt kann man eine Komplexität von O(n\*m) erwarten, wobei n die Anzahl der Knoten im Hauptbaum und m die Anzahl der Knoten im Subtree sind. Im besten Fall, wenn der Subtree nicht vorhanden ist, kann die Komplexität auf O(n) reduziert werden.

**rotateLeft, rotateRight:**

Die Laufzeitkomplexität dieser Operationen ist O(1), da sie unabhängig von der Größe des Baums eine feste Anzahl von Zeigeraktualisierungen durchführen.

Diese Dokumentation beschreibt die Funktionen der AVLTree-Klasse, die für die Verwaltung und das Durchlaufen des AVL-Baums verwendet werden. Die analytische Bewertung des Laufzeitverhaltens ist essenziell, um die Effizienz der Baumoperationen zu verstehen.