Die Klasse BinaryTree bietet benutzerfreundliche Methoden zum Einfügen, getMax, getMin und getAvg an, wo der Aufrufende sich nicht um den Root kümmern muss, sondern nur Werte eingibt.

Intern arbeitet die Klasse mit folgenden rekursiven Methoden:

Insert(Node* root, Node* newNode)

Insert bekommt einen fertig erstellen Node. Sie findet rekursiv für ihn den richtigen Platz und passt die Pointer an. Abbruchsbedingung: aktueller Node ist Null. Ist der des neuen Nodes größer als der aktuelle, ruft sich die Funktion rekursiv mit dem rechten Node auf. Analog zum linken Teil und kleineren Werten. Bei Duplikaten wird der newNode verworfen.

Delete(Node* root)

Zuerst nach links, dann nach rechts traversieren, solange root nicht Null ist. Beim jeweiligen Sprung zum vorherigen Aufruf wird der aktuelle Node gelöscht.

Inorder(Node* root)

Zuerst Links traversieren, dann Informationen zum aktuellen Node (Wert, Balance Faktor, AVL ja/nein), dann rechts traversieren.

goRight(Node* root), goLeft(Node* root)

So weit wie möglich rechts bzw. links traversieren, um min/max zu retorunieren.

calcSum(Node* root)

Beim Erreichen einer Null, wird 0 zurückgegeben. Else wird der Wert zum rekursiven Aufruf zum rechten und linken Teil addiert und retourniert.

count(Node* root)

Count deklariert eine locale Variable number mit Initialwert 0. Solange root nicht Null ist, wird number incrementiert und zum rekursiven Aufruf vom rechten und linken Teil addiert und retourniert.

Height(Node* root)

Es wird zu den NullWerten rekursiv traversiert. Die Nullwerte haben die Höhe 0, deswegene wird 0 zurückgegeben. Jede Ebene gibt den vorigen Rückgabewert + 1 zurück. Das Maximum der Werte ist die Baumhöhe. So wird die Höhe von unten nach oben berechnet.

Aufwand:

Der Baum implementiert keinen Selbstbalancierungsmechanismus, daher kann er in worst case zu einer verketteten Liste werden und O(N) beim Einfügen, Löschen und Suchen haben.

Best Case ist dass der Baum balanciert ist, dann besitzt er O(log2 N) bei allen Operationen.