# Protokoll Programmierbeispiel 2 – Baumprogramm

## Aufgabe 1: Baum einlesen & AVL-Bedingung

***Aufgabenstellung:***

*Implementieren Sie ein Programm das überprüft ob ein binärer Baum ein AVL-Baum ist und statistische Daten zu dem Baum ausgibt. Das Hauptprogramm liest aus einem Textfile (Dateiname wird als Parameter übergeben) Integer-Schlüsselwerte ein und baut mit diesen Werten der Reihe nach einen binären Suchbaum auf. Mehrfach vorhandene Schlüsselwerte werden beim Einfügen verworfen. Weiters sollen rekursive Funktionen entwickelt werden, die für den binären Suchbaum für jeden Knoten den Balance Faktor ausgeben und damit überprüfen ob der gegebene Baum ein AVL-Baum ist. Wird die AVL-Bedingung in einem Knoten verletzt (Balance Faktor >1 oder <-1) so soll dies gesondert ausgegeben werden. Weiters sollen statistische Daten des Baumes (kleinster Schlüsselwert, größter Schlüsselwert)und durchschnittlicher Schlüsselwert (arithmetisches Mittel aller Schlüsselwerte) ausgegeben werden. Diese Daten sollen ebenfalls durch eine Traversierung des Baumes berechnet werden und nicht aus der Eingabedatei bestimmt werden.*

### Definition Datenstruktur „Binärbaum“:

Ein Baum beschreibt eine Datenstruktur, mit der sich hierarchische Strukturen abbilden lassen. In einem Binärbaum gilt die Besonderheit, dass maximal zwei Kinder mit einem Elternknoten verknüpft sein dürfen. In der implementierten Datenstruktur wird im rechten Knoten ein größerer Wert gespeichert als der des Elternknotens. Analog gilt für dies für den linken Knoten. Im Gegensatz zur verketteten Liste ist ein Baum beim Suchen von Daten besonders effektiv mit einer Laufzeit von log(n). Es kann jedoch vorkommen, dass ein Binärbaum, einer verketteten Liste ähnelt und dadurch seine Effizienz verliert.

Beispielsweise wäre eine Zahlenreihe 1,2,3,4,5, die in Reihenfolge in einem Baum gespeichert wird, eine verkettete Liste, wenn im rechten Teilbaum jeweils die größeren Werte gespeichert werden und analog dazu im linken die kleineren Werte.

AVL – Bäume versuchen dieses Ungleichgewicht zu verhindern, um die Effizienz sicherzustellen. Dabei wird ein Balance-Faktor für jeden Knoten des Baumes definiert:

Wobei t ein Teilbaum ist und es jeweils einen linken und rechten Teilbaum gibt. Deren Höhe wird durch Height dargestellt. Die AVL-Bedingung ist erfüllt, wenn für den Balance-Faktor gilt:

Um diese Balance im Falle der Verletzung der Bedingung wiederherzustellen, gibt es sowohl eine Einfachrotation, oder auch die Doppelrotation.

Für diese Aufgabenstellung wurden zwei Datenstrukturen implementiert, eine die einen individuellen Knoten repräsentiert und eine, die den Baum repräsentiert. Die Knotendatenstruktur beinhaltet einen ganzzahligen Wert, einen Zeiger auf einen potenziellen linken Knoten und analog dazu zu einem rechten Knoten. Die Datenstruktur des Baumes speichert jeweils Min/Max Werte sowie alle zur Berechnung vom Durchschnitt relevanten Werte. Es wird ebenfalls gespeichert, ob es der gegebene Baum die AVL-Bedingung erfüllt.

### Operationen - Zeitkomplexität

In einem AVL-Baum gelten folgende Komplexitätslaufzeiten für die geforderten Operationen lt. Angabe:

|  |  |  |
| --- | --- | --- |
| Operation | Average Case (θ) | Worst Case(O) |
| Suchen |  |  |
| Einfügen |  |  |
| Löschen |  |  |
| Min,Max |  |  |

Es wurden folgende zusätzliche Funktionen implementiert:

* Ausgabe des Baums in Form eines Linux-Baumdiagrams
* Erstellung eines Baums in der Kommandozeile

### Überprüfung AVL-Kondition

Für die Überprüfung der AVL Kondition wurde eine rekursive Funktion gewählt, die im Modus reverse Inorder den Baum traversiert und zu jedem Knoten feststellt, ob die AVL-Bedingung verletzt wird oder nicht. Dabei wurde eine Höhenfunktion implementiert, die ebenfalls rekursiv die Höhe vom aktuellen Knoten bis zum Letzt-verknüpften Blattknoten berechnet.

Reverse Inorder überprüft zunächst den rechten Teilbaum, anschließend wird der zugehörige Elternknoten überprüft, gefolgt von der Überprüfung des linken Teilbaumes. Es wird jeweils so lange traversiert, bis ein Nullpointer erreicht wird. Dies signalisiert, dass es an der jeweiligen Stelle keine weiteren Knoten existieren. Daraus lässt sich die AVL-Bedingung überprüfen.

Die Height Funktion gibt jeweils eine ganze Zahl zurück, mit der in der Überprüfungsfunktion gerechnet werden kann, ob die AVL-Bedingung erfüllt ist. Dabei wird für den linken Teilbaum ein Wert zugewiesen, analog dazu zum rechten Teilbaum. In der Überprüfungsfunktion wird die absolute Differenz dieser Werte errechnet. Sollte dabei die Differenz größer als eins sein, ist die AVL-Bedingung verletzt. Bei der Überprüfung werden alle zusätzlich relevanten Daten berechnet, die in der Klasse bereits definiert sind.

Es wurden zwei zusätzliche Funktionen implementiert, die jeweils das Maximum und das Minimum aus dem Baum finden. Dadurch ist eine Laufzeit von log2(n) möglich, anstelle der Laufzeit von O(n), die bei der Prüfung der AVL Bedingung.

### Einfache Suche

Für die einfache Suche wird ein Baum mit einem Knoten generiert. Der Wert dieses Knotens, wird dann mit dem Wert des ersten Knotens im Suchbaum verglichen. Ist dieser gleich so gilt der wert als gefunden. Ist dieser Wert, aber ungleich wird zuerst der Knotenwert in einem Vector zwischengespeichert und überprüft, ob der Suchwert größer als der Knotenwert im Baum ist. Ist dies der Fall, wird kontrolliert, ob ein rechter Knoten existiert. Wenn der rechte Knoten kein Nullpointer ist wird in den rechten teil rekusiert. Falls der Suchwert kleiner als der Kontenwert ist, wird dasselbe mit der linken Seite gemacht. Ist der wert gefunden, wird dies ausgegeben zusammen mit den werten im Vector. Trifft die Suche allerdings auf einen Nullpointer so wird diese abgebrochen und die Meldung ausgegeben das der wert nicht gefunden wurde.

Suche eines Subtrees

Bevor der Subtree gesucht werden kann wird dieser aufgebaut und dabei die Anzahl der Knoten gezählt. Danach wird die Wurzel des Subtrees mit der des Suchbaumes verglichen und ähnlich der einfachen Suche bei Ungleichheit wird die Größe verglichen und die entsprechende Richtung gewählt. Diese suche schlägt ebenfalls fehl, sobald sie auf einen Nullpointer trifft. Sollte aber ein wert übereinstimmen wird überprüft, ob der Subtree links und rechts weitere Knoten hat und mit diesen Rekusiert. Dabei wird bei jeder Übereinstimmung ein Integer wert erhöht der am Ende mit der Anzahl an knoten im Subtree verglichen wird. Stimmen diese überein so war die Suche erfolgreich.