ÜBUNG 2 - TREECHECK

Arbeiten Sie in 2er Gruppen an folgendem Programmierbeispiel. Die Wahl der Programmiersprache ist Ihnen überlassen (C,C++,C#,Java). Die Datenstruktur muss jedoch selbst implementiert werden!

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.

Überlegen Sie sich vor der Implementierung, wie die rekursiven Funktionen aufgebaut werden müssen (Abbruchbedingung, Parameter, Rückgabewert, …) und protokollieren Sie ihre Überlegungen. Schätzen Sie weiters den Aufwand der Funktionen mittels O-Notation in Abhängigkeit der Anzahl N der Integer-Werte in der Eingabedatei.

Hinweise Programm

Der Programmaufruf sollte wie folgt aussehen: **treecheck** *filename*

Hinweise Fileformat

Das Inputfile ist ein Textfile, das einen Key pro Zeile enthält. Alle Keys sind Integer-Werte und es sind beliebig viele davon erlaubt.

Beispiel:

```
5
3
17
9
23
```

```
54
11
79
30
12
```

Hinweise Datenstruktur

Definieren Sie eine geeignete Datenstruktur für einen Knoten des binären Suchbaums. In C könnte die Struktur etwa wie folgt aussehen:

```
struct tnode{
   int key;
   struct tnode *left;
   struct tnode *right;
};
```

Der Balance Faktor eines Knotens bal(k) ist definiert als

bal(k) = h(rechter Teilbaum) - h(linker Teilbaum)

(siehe Vorlesung).

Hinweise Ausgabe

Pro Knoten wird der Balancefaktor in folgendem Format ausgegeben:

```
bal(key) = x
```

Bei Verletzung der AVL-Bedingung wird dies durch folgende Ausgabe angezeigt:

```
bal(key) = x (AVL violation!)
```

Danach wird noch ausgegeben ob es sich bei dem Baum um einen AVL-Baum handelt (d.h. ob alle Knoten die AVL-Bedingung erfüllen) oder nicht:
Ausgabe von `AVL: yes` wenn es sich um einen AVL-Baum handelt, bzw. `AVL: no` wenn es sich um keinen AVL-Baum handelt.

Am Ende sollen dann noch statistische Daten ausgegeben werden
`min: x, max: y, avg: z`

Referenzausgabe für die oben angeführten Testdaten:

```
bal(79) = 0
bal(30) = 0
bal(54) = 0
bal(23) = 2 (AVL violation!)
bal(12) = 0
```

```
bal(11) = 1
bal(9) = 2 (AVL violation!)
bal(17) = 0
bal(3) = 0
bal(5) = 3 (AVL violation!)
AVL: no
min: 3, max: 79, avg: 24.3
```

Die von Ihrem Programm erzeugte Ausgabe muss diesem Format entsprechen!

Abgabe

Im Abgabesystem ist ein .zip oder .tgz File abzugeben. Dieses soll beinhalten:

- Alle Sourcen inkl. Code Kommentaren!
- ausführbares Programm
- Protokoll mit Beschreibung der rekursiven Funktionen und Aufwandsabschätzung
- Testfiles mit Schlüsselwerten

Die Abgabe muss beim zweiten Code Review präsentiert werden, es können für diese Übung maximal 13 Punkte erreicht werden.

Punkteschlüssel

- Einlesen und Aufbau des Baums 3
- Rekursive Berechnung AVL 5 (ohne Rekursion keine Punkte!)
    - Achten Sie auf die Effizienz Ihrer Berechnung!
- min/max/avg 2
- Protokoll 3

Hinweis: Unsaubere Implementierung und daraus folgende Crashes in der zentralen Funktionalität führen zu Punkteabzügen!