ÜBUNG 2 - TREECHECK

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

AUFGABENSTELLUNG

Die Aufgabe ist unterteilt in zwei Teile.

1. Einen Baum einlesen und statistische Informationen ausgeben
2. Suchen im Baum

1. Baum einlesen

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).

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!

2. Suche im Baum

Im ersten Aufgabenteil haben Sie einen AVL-Baum eingelesen. Nun sollen Sie in einem Baum Einträge und Unterbäume (Subtrees) suchen.
Dazu muss Ihr Programm zwei Argumente beim Starten übernehmen. Das erste Argument repräsentiert die Datei mit Schlüsselwerten, die den Suchbaum beschreibt und das zweite Argument repräsentiert die Datei, welche den zu suchenden Subtree enthält.
Für Suchbaum und Subtree gilt das gleiche Fileformat wie in der ersten Aufgabe.
Der Subtree muss mindestens einen Schlüsselwert beinhalten, er kann aber auch mehrere Schlüsselwerte enthalten, die einen Baum repräsentieren.
Somit unterscheiden wir zwei Suchfälle

- Einfache Suche
- Suche eines Subtrees

In beiden Fällen darf nicht in der Eingabeliste gesucht werden, sondern im Suchbaum. Bitte designen Sie Ihr Programm so, dass diese Vorgabe bei dem Codereview klar ersichtlich ist.

Hinweise Programm

Der Programmaufruf sollte wie folgt aussehen:

treecheck *filename-suchbaum filename-subtree*

Hinweise zur Einfachen Suche

Bei der einfachen Suche befindet sich in der Subtree-Datei nur ein Schlüsselwert. Als Ergebnis einer erfolgreichen Suche sollen alle Knoten zwischen Wurzel und dem Schlüsselwert ausgegeben werden. Ist der Eintrag nicht im Baum wird der zu suchende Schlüsselwert und „not found!“ ausgeben.

Beispiele:

- Erfolgreiche Suche nach 7 im Suchbaum 5, 3, 8, 7, 22, 2
  Ausgabe: `7 found 5, 8, 7`
- Nichterfolgreiche Suche nach 1 im Suchbaum 5, 3, 8, 7, 22, 2
  Ausgabe: `1 not found!`

Hinweise zur Suche eines Subtrees

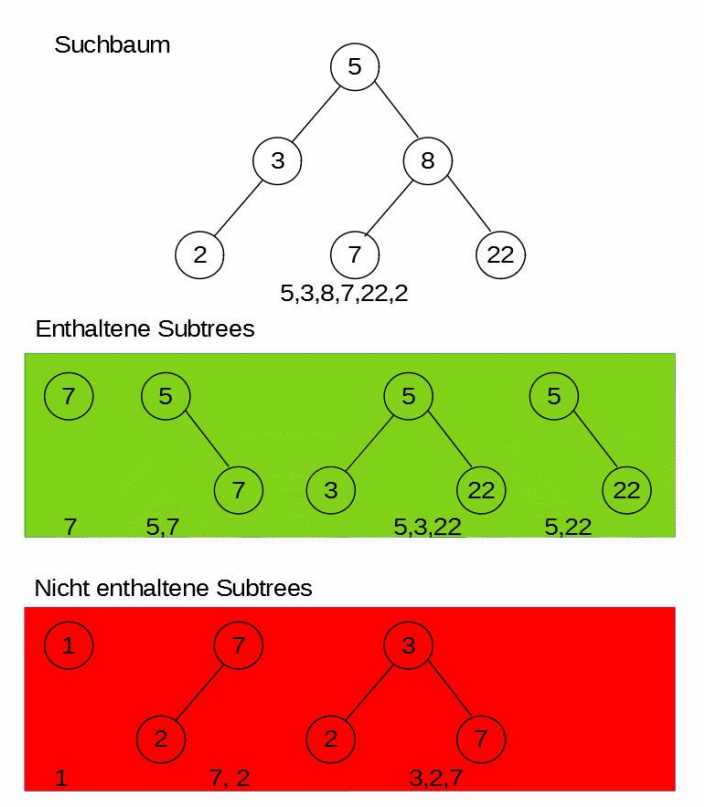
Bei der Suche eines Subtrees soll überprüft werden, ob die Struktur des Subtrees im Baum existiert. Die Struktur ist im Suchbaum enthalten, wenn die Reihenfolge der Schlüsselwerte des Subtrees sich auch in der Reihenfolge der Schlüsselwerten des Suchbaums befindet.
Da diese Lösung trivial ist, ist gefordert den Suchbaum zuerst aufzubauen und **erst dann** die Struktur mittels rekursiven Traversierens durch den Baum zu suchen.
Als Ergebnis einer erfolgreichen Suche soll „Subtree found“ ausgegeben werden und “Subtree not found!“ wenn der Subtree nicht gefunden wurde.

Beispiel:

- Erfolgreiche Suche nach 5, 7 im Suchbaum 5, 3, 8, 7, 22, 2
  Ausgabe: `Subtree found`
- Nichterfolgreiche Suche nach 7, 2 im Suchbaum 5, 3, 8, 7, 22, 2
  Ausgabe: `Subtree not found!`

Hinweis zur Suche

Nachstehende Figure zeigt ein Beispiel mit enthaltenen und nicht enthaltenen Subtrees.



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 18 Punkte erreicht werden (15 Punkte für das Programm und 3 Punkte für das Protokoll).