Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2015/16
Feldmann / Semmler / Lichtblau / Streibelt / Pujol / Rost

Aufgabenblatt 11

letzte Aktualisierung: 27. Januar, 06:06 Uhr

(4bc945439c59b0260c428445b170dd660ed5a7b6)

Ausgabe: Mittwoch, 27.01.2016
Abgabe: spätestens Freitag, 05.02.2016, 20:00

Thema: *AVL-Bäume*

Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des tubIT/IRB mittels `gcc -std=c99 -Wall` kompilieren.
- Abgaben erfolgen prinzipiell immer in Gruppen à 2 Personen, welche in den Tutorien festgelegt wurden. Einzelabgaben sind explizit als solche gekennzeichnet.
- Die Abgabe erfolgt ausschließlich über SVN. Die finale Abgabe
  - für Gruppenabgaben erfolgt im Unterordner `Tutorien/t<xx>/Gruppen/g<xx>/Abgaben/Blatt<xx>/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Abgaben/Blatt<xx>/`

  wobei die Ordner von uns erstellt werden.
- Benutze für alle Abgaben von Programmcode das folgende Namensschema: `introprog_blatt0X_aufgabe0Y_Z.c`, wobei X durch die Blattnummer, Y durch die Aufgabe und Z durch die Unteraufgabe zu ersetzen ist.
  Beispiel: Aufgabe 1.2 wird zu: `introprog_blatt01_aufgabe01_2.c`
  Gib für jede Unteraufgabe maximal eine Quellcodedatei ab, es sei denn, die Aufgabenstellung erfordert explizit die Abgabe mehrerer Dateien pro Aufgabe.

  Benenne alle anderen Abgaben (Pseudocode, Textaufgaben) wie oben beschrieben. Die zugelassenen Abgabeformate sind PDF, ODT und Text (txt). Verwende auch hier eine Datei pro Aufgabe, nicht jedoch pro Unteraufgabe.

Definitionen

Abweichend von der Definition bei binären Suchbäumen, verwenden wir zur Bestimmung der Höhe im Kontext der AVL-Bäume folgende Definition:

Definition: `Die Höhe eines Blattes (d.h. der Knoten hat keine Kinder) im AVL-Baum ist 1. Anderenfalls ist die Höhe eines (inneren) Knotens x (d.h. x hat mindestens ein Kind) definiert als:`

```
Höhe(x) = 1 + max(Höhe(l[x]), Höhe(r[x] ))
```

In AVL-Bäumen werden Balancierungs-Operationen an einem Knoten `x` nur angewandt, wenn die Höhen der Kinder (`l[x]`, `r[x]`) um mehr als 1 unterscheiden. Wir benutzen auch folgende Definition:

Definition: `Der Balance-Wert eines Knoten ist die Baumhöhe des linken Kindes minus der Baumhöhe des rechten Kindes.`

Ein AVL-Baum ist somit balanciert, sofern als Balance-Werte nur -1, 0, und 1 vorkommen.

1. Aufgabe: Handsimulation AVL-Baum (1 Punkt)

In dieser Aufgabe soll die Einfügen-Operation auf AVL Bäumen geübt werden.
Füge die Elemente 100, 50, 25, 10, 37, 32, 200 nacheinander in einen initial leeren AVL-Baum ein. Zeichne den Baum jeweils nach dem Einfügen des Elements sowie nach jeder benötigten Rotation. Notiere für jeden Baum (auch jeden Zwischenzustand) den entsprechenden Balance-Wert an jedem Knoten und benenne jeweils die durchgeführten Rotationen analog zu dem Beispiel in Abbildung 1.
Gib deine Lösung der Aufgabe in einem Textdokument mit einem der folgenden Namen ab:
`introprog_blatt11_aufgabe01.{txt|odt|pdf}`

Hinweis: Mit einer korrekten Lösung der Aufgabe 2 kannst du diese Aufgabe sehr einfach lösen. Andererseits soll die "Handsimulation" dazu dienen, den Code der in Aufgabe 2 geschrieben werden muss, zu verstehen.

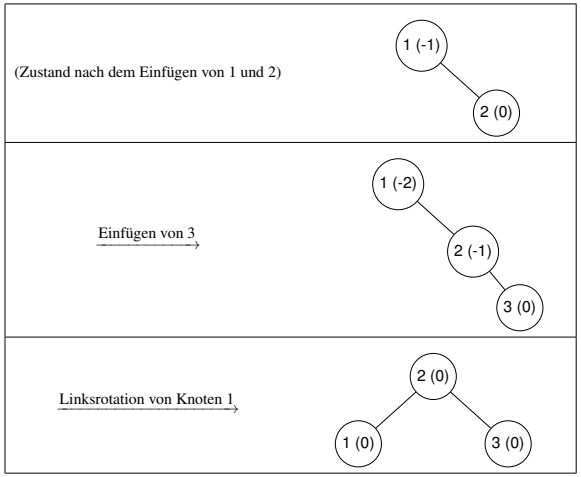

Abbildung 1: Beispiel zur Notation von Bäumen und den Balance-Werten.

2. Aufgabe: Implementieren eines AVL-Baumes (3 Punkte)

In dieser Aufgabe soll ein AVL-Baum implementiert werden. Das resultierende Programm soll dabei die Eingabe entweder von der Konsole oder aus einer Datei einlesen. Folgende Befehle sollen zur Verfügung stehen:

<Z> Die Zahl **<Z>** soll in den Suchbaum eingefügt werden.

p Gibt den gesamten AVL-Baum aus.

w Gibt alle Knoten in "in-order" Reihenfolge aus.

c Gibt die Anzahl an enthaltenen Knoten aus.

q Beendet das Programm.

Die Eingabedatei aus Listing 1 führt zur Ausgabe in Listing

Listing 1: `Eingabedatei_Blatt11.txt`

```
12
2
8
3
21
9
4
7
p
w
c
q
```

Listing 2: Ausgabe des Programms unter der Eingabe der Datei `Eingabedatei_Blatt11.txt`

```
Füge 12 ein...
Füge 2 ein...
Füge 8 ein...
Füge 3 ein...
Füge 21 ein...
Füge 9 ein...
Füge 4 ein...
Füge 7 ein...
                          8
              /-----------|-----------\
              3                       12
       /------|------\               /------|------\
       2              4             9             21
                      |---\
                          7

 2 3 4 7 8 9 12 21
Im AVL-Baum sind 8 Elemente enthalten.
```

Während wiederum eine `main`-Funktion sowie das Parsen der Eingabe von uns bereitgestellt werden, müssen die Funktionen aus Listing 3 implementiert werden.

Listing 3: `introprog_blatt11_aufgabe02_vorgabe.c`

```c
#include <stdlib.h>
#include <stdio.h>      //Ein- / Ausgabe
#include <math.h>    //Für die Berechnungen der Ausgabe
#include "blatt11.h"

// Gibt den gesamten AVL Baum in "in-order" Reihenfolge aus.
void AVL_in_order_walk(AVLTree* avlt)
{
   //Hier Code implementieren!
}

// Diese Funktion führt eine Linksrotation auf dem angegebenen Knoten aus.
// Beachtet: Die Höhen der entsprechenden Teilbäume müssen (lokal) angepast
   ↪ werden.
void AVL_rotate_left(AVLTree* avlt, AVLNode* x)
{
   //Hier Code implementieren!
}

// Diese Funktion führt eine Rechtsrotation auf dem angegebenen Knoten aus.
// Beachtet: Die Höhen der entsprechenden Teilbäume müssen (lokal) angepast
   ↪ werden.
void AVL_rotate_right(AVLTree* avlt, AVLNode* y)
{
   //Hier Code implementieren!
}

//Balanciere den Teilbaum unterhalb von node.
void AVL_balance(AVLTree* avlt, AVLNode* node)
{
   //Hier Code implementieren!
}


// Fügt der Wert value in den Baum ein.
// Die Implementierung muss sicherstellen, dass der Baum nach dem Einfügen
// immer noch balanciert ist!
void AVL_insert_value(AVLTree* avlt, int value)
{
   //Hier Code implementieren!
}


// Löscht den gesamten AVL-Baum und gibt den Speicher aller Knoten frei.
void AVL_remove_all_elements(AVLTree* avlt)
{
   //Hier Code implementieren!
}
```

2.1. Ausgabe von Elementen im AVL-Baum

Implementiere die Funktion `void AVL_in_order_walk(AVLTree* avlt)`, welche sämtliche Werte im binären Suchbaum in "in-order" Reihenfolge auf `stdout` (der Konsole) ausgibt. Beachte, dass bei der "in-order" Reihenfolge, die Elemente immer aufsteigend geordnet sind. Die Elemente sollen durch ein einzelnes Leerzeichen getrennt sein. Nach der Ausgabe aller Elemente muss ein einfacher Zeilensprung (`\n`) folgen (siehe auch Listing 3).

2.2. Links- und Rechtsrotation

Implementiere die Funktionen `void AVL_rotate_left(AVLTree* avlt, AVLNode* x)` und `void AVL_rotate_right(AVLTree* avlt, AVLNode* y)`, welche jeweils die AVL Links- und Rechtsrotation auf dem Knoten `x` bzw. `y` ausführen.

Hinweis: Achte darauf, dass die Pointer richtig vertauscht werden, und dass nach der jeweiligen Rotation in allen Knoten die gespeicherten Baumhöhen stimmen. Beachte weiterhin, dass eventuell der Wurzelknoten des gesamten Baumes angepasst werden muss. Diese Funktionen müssen jeweils eine konstante Laufzeit haben.

2.3. Wiederherstellen der Balance eines AVL-Baumes

Implementiere die Funktionen `void AVL_balance(AVLTree* avlt, AVLNode* node)`, welche den AVL-Baum am Knoten `node` (gegebenenfalls) balanciert.

Hinweis: Diese Funktion muss eine konstante Laufzeit haben. Es soll also wirklich nur der übergebene Knoten balanciert werden.

2.4. Einfügen in den AVL-Baum

Implementiere unter Verwendung der vorherigen Funktionen die Funktion `void AVL_insert_value(AVLTree* avlt, int value)`, welche den Wert `value` in den Baum `avlt` einfügt.

Hinweis: Achte darauf, dass insbesonders die `parent` Pointer richtig gesetzt sind, dass die Suchbaum-Eigenschaft erhalten bleibt und dass der Baum nach dem Einfügen überall balanciert ist. Diese Funktion muss eine Laufzeit von $\mathcal{O}(\log n)$ haben.

2.5. Korrekte Freigabe des Speichers

Implementiere abschließend die Funktion `void AVL_remove_all_elements(AVLTree* avlt)`, welche alle Knoten im Baum löscht. Implementiere die Funktion dabei so, dass der Suchbaum nur einmal durchlaufen wird: Gegeben einen Knoten im Baum wird zuerst der rechte Unterbaum, dann der linke Unterbaum und anschließend der aktuelle Knoten gelöscht. Diese Art der Traversierung wird auch als "post-order" bezeichnet.

Gib den Quelltext in einer Datei mit folgendem Namen ab:
`introprog_blatt11_aufgabe02.c`

Hinweis: Kompiliere dein Programm mit `gcc -Wall -std=c99 main_blatt11.c blatt11.c introprog_blatt11_aufgabe02.c -lm -g -o introprog_blatt11`