Fakultät IV Elektrotechnik und Informatik
Einführung in die Programmierung WS 2022/2023
Manfred Hauswirth
Damien Foucard / Uwe Kuehn / Aljoscha Meyer

Programmierblatt 10

Ausgabe: *26.01.2022 16:00*
Abgabe: *07.02.2022 08:00*

Thema: *AVL-Bäume*

Abgabemodalitäten

1. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf Deinem Rechner mit dem Befehl `clang -std=c11 -Wall -g` kompilieren.
2. Die Abgabe für den Quellcode erfolgt ausschließlich über unser Git im entsprechenden Branch. Nur wenn ein Ergebnis im ISIS-Kurs angezeigt wird, ist sichergestellt, dass die Abgabe erfolgt ist. Die Abgabe ist bestanden, wenn Du an Deinem Test einen grünen Haken siehst.
3. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben. Lies Dir die Hinweise der Tests genau durch, denn diese helfen Dir Deine Abgabe zu korrigieren.
Bitte beachte, dass ausschließlich die letzte Abgabe gewertet wird.
4. Die Abgabe erfolgt, sofern nicht anders angegeben, in folgendem Branch: `iprg-b<xx>-a<yy>`, wobei `<xx>` durch die zweistellige Nummer des Aufgabenblattes und `<yy>` durch die entsprechende Nummer der Aufgabe zu ersetzen sind.
5. Gib für jede Aufgabe die Quellcodedatei(en) gemäß der Vorgabe ab. Im ISIS-Kurs werden zum Teil Vorgabedateien bereitgestellt. Nutze diese zur Lösung der Aufgaben.
6. Die Abgabefristen werden vom Server überwacht. Versuche Deine Abgabe so früh wie möglich zu bearbeiten. Du minimierst so auch das Risiko, die Abgabefrist auf Grund von „technischen Schwierigkeiten“ zu versäumen. Eine Programmieraufgabe gilt als bestanden, wenn alle bewerteten Teilaufgaben bestanden sind.
7. Sofern die Aufgabenstellenstellung nichts gegenteiliges besagt, dürfen keine weiteren `include` Direktive verwendet werden, d.h., es dürfen keine zusätzlichen Libraryfunktionen verwendet werden. Eigene Funktionen zu implementieren und verwenden ist hingegen legitim und häufig eine gute Idee für besser lesbaren Code.

Definitionen

Bei binären Suchbäumen wurde die Höhe als die Anzahl an Kanten zum Wurzel definiert, d.h., bei einem Baum mit zwei Knoten, ein Elternteil und ein Kind, hat der Elternteil die Höhe 1 und das Kind die Höhe 0. Es wird diese Definition ergänzt, um die Schreibweise weiterhin zu vereinfachen:

Definition: `Die Höhe des leeren Baumes (keine Knoten) im AVL Baum ist -1. Die Höhe eines Blattes (Knoten ohne Kinder) im AVL-Baum ist 0. Andernfalls 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 Balancierungsoperationen an einem Knoten `x` nur angewandt, wenn die Höhen der Kinder (`l[x]`, `r[x]`) sich 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.

Aufgabe 1 Implementieren eines AVL-Baumes (bewertet)

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 2.

Listing 1: `Eingabedatei_avl.txt`

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

Listing 2: Ausgabe des Programms unter der Eingabe von `Eingabedatei_avl.txt`

```
Füge 12 ein...
Füge 2 ein...
Füge 8 ein...
[snip]
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.
```

Nutze zur Lösung der Aufgabe die Vorgaben aus unserem ISIS-Kurs. Füge Deine Lösung als Datei `introprog_avl.c` im entsprechenden Abgabebranch in Dein persönliches Repository ein und übertrage die Lösung an die Abgabeplattform.

Aufgabe 1.1 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 2).

Aufgabe 1.2 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.

Aufgabe 1.3 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.

Aufgabe 1.4 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.

Aufgabe 1.5 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.

Hinweis: Kompiliere dein Programm mit `clang -Wall -std=c11 main_avl.c avl.c introprog_avl.c -lm -g -o introprog_avl`