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

Aufgabenblatt 5

letzte Aktualisierung: 17. Dezember, 12:27 Uhr

(c7fd4b4f780008ba2451f041be4efcea20a8629f)

Ausgabe: Mittwoch, 2.12.2015
Abgabe: spätestens Freitag, 11.12.2015, 20:00

Thema: *Binäre Suchbäume*

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.

1. Aufgabe: Binärer Suchbaum (1 Punkt)

Hinweis: Der Baum ist so sortiert, dass kleiner gleiche Element immer links und größere Elemente immer rechts platziert werden.

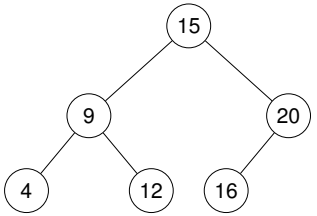

Abbildung 1: Binärbaum

1.1. Löschen aus binären Suchbäumen (0.5 Punkte) Welcher Baum ergibt sich nachdem Du die folgenden Elemente aus dem Baum in Abbildung 1 nach dem Verfahren aus der Vorlesung löscht?

(15), (9), (20), (16), (4)

Zeichne den resultierenden Baum sowie alle Zwischenresultate nach jeder Löschoperation.

Gib Deine Lösung der Teilaufgabe in einem Textdokument mit einem der folgenden Namen ab: `introprog_blatt05_aufgabe01_1.{txt|odt|pdf}`

1.2. Einfügen in Bäume (0.5 Punkte) Welcher Baum ergibt sich nachdem Du die folgenden Elemente in der gegebenen Reihenfolge in den Baum in Abbildung 1 einfügst?

(11), (22), (15), (21)

Zeichnen den resultierenden Baum.

Gib Deine Lösung der Teilaufgabe in einem Textdokument mit einem der folgenden Namen ab: `introprog_blatt05_aufgabe01_2.{txt|odt|pdf}`

2. Aufgabe: Telefonbuch implementieren (2 Punkte)

In dieser Aufgabe soll ein Telefonbuch als binärer Suchbaum eingelesen, durchsucht und ausgegeben werden. Die Telefonnummern sind dabei natürliche Zahlen mit maximal vier Ziffern (also 1 und 9999, aber nicht $-2, 3, 5$ oder 12345) und sollen so angeordnet werden, dass kleinere Zahlen immer links und größere Zahlen immer rechts platziert werden (siehe Abbildung 2). Es dürfen keine Zahlen mehrfach vorkommen. Es gibt eine großzügige Vorgabe, sodass Du Dich ganz auf die Implementierung der folgenden Funktionen beschränken kannst:

- `bst_insert_node(bstree* bst, int phone, char *name)`
  - Platziere einen neuen Knoten in den Baum `bst`.
  - Halte die Ordnung ein wie oben beschrieben.
  - Verändere, wenn nötig, den Pointer auf den Wurzelknoten.
  - Gebe eine Fehlernachricht aus und gehe mit **`return`** aus der Funktion, wenn versucht wird eine Telefonnummer, die schon im Baum existiert, erneut einzugeben. `bst->root`.
- `bst_node* find_node(bstree* bst, int phone)`

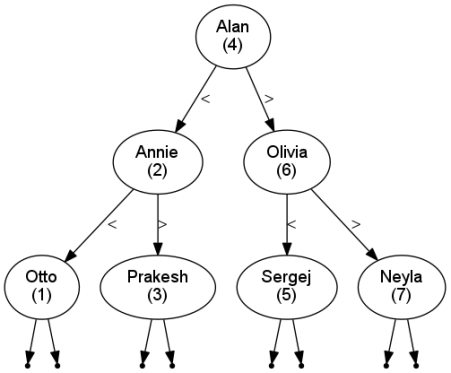

Abbildung 2: Beispiel-Telefonbuch als binärer Suchbaum

  - Finde den Knoten im Baum, der die angegebene Telefonnummer besitzt und gebe ihn zurück.
  - Gebe `NULL` zurück, wenn es keinen entsprechenden Knoten gibt.

- `bst_in_order_walk(bstree* bst)`
  - Gebe den Baum nach Telefonnummern geordnet aus.
  - Benutze dafür die Funktion `print_node`. Sie benötigt als einzigen Parameter einen Pointer auf den gegenwärtigen Knoten (also im Format `bst_node*`).
- `bst_free_tree(bstree* bst)`
  - Gebe den Speicher des binären Suchbaumes vollständig frei.
  - Achte dabei darauf, dass du bei den Blätter beginnst und du zuletzt die Wurzel freigibst.

Um die Aufgabe übersichtlicher zu gestalten, ist der Code auf vier Dateien verteilt:

input_blatt05.c wird wie gehabt verwendet um Eingabefunktionen und kleine Hilfsfunktionen aufzubewahren. Hier solltest Du nichts verändern.

main_blatt05.c kommt dazu, um das Interface von der eigentlichen Logik zu trennen. Hier wird die `main` Funktion implementiert und auch hier solltest Du nichts verändern.

introprog_blatt05_aufgabe02.c In dieser Datei sollst Du die oben genannten Funktionen implementieren. Neu ist dabei, dass sich die `main` Funktion und die **`struct`** in anderen Dateien befinden.

blatt05.h ist die Header Datei, die alle Dateien miteinander verknüpft. In dieser Aufgabe sind an dieser Stelle auch die **`struct`** und **`typedef`**, sowie die `includes` definiert.

Bei der Kompilierung muss neben wie gehabt `input_blatt05.c` auch zusätzlich die Datei `main_blatt05.c` angegeben werden. Das Programm benötigt als Parameter die Angabe des Telefonbuch. Es sind dafür in den Vorgaben vorhanden:

leer.txt Ein leeres Telefonbuch

einfach.txt Ein kleines Telefonbuch-Beispiel

tel.txt Ein großes Telefonbuch-Beispiel

Listing 1: Programmbeispiel

```
> gcc -std=c99 -Wall introprog_blatt05_aufgabe02.c input_blatt05.c \
                     main_blatt05.c -o introprog_blatt05_aufgabe02
> ./introprog_blatt05_aufgabe02 tel.txt
 Fernsprech-Datensatz-System
 ===========================
  Füge in das Telefonbuch ein:  + <Nummer> <Name>
  Gebe das Telefonbuch aus:     p
  Finde den Namen:              ? <Nummer>
  Debugausgabe des Baumes:      d
  Beende das System:            q
```

Um das Debugging zu vereinfachen haben wir eine Debugging Funktionalität eingebaut. Mittels 'd' lässt sich beim Aufruf des Programm der binäre Suchbaum in eine PNG Datei ausgeben. Dafür wird die Software Graphviz benötigt. Auf den IRB Rechnern sollte die schon installiert sein, auf Heimrechnern muss die ggfs. nachinstalliert werden.

Benutze die folgende Codevorgabe:

Listing 2: Vorgabe `introprog_blatt05_aufgabe02_vorgabe.c`

```c
#include "blatt05.h"

// Fügt einen Knoten mit der Telefonnummer phone und dem Namen name in den
// Binären Suchbaum bst ein.  Für den Suchbaum soll die Eigenschaft gelten,
   ↪ dass
// alle linken Kinder einen Wert kleiner gleich (<=) und alle rechten Kinder
// einen Wert größer (>) haben.
void bst_insert_node(bstree* bst, int phone, char *name) {
}

// Diese Funktion liefert einen Zeiger auf einen Knoten im Baum
// mit dem Wert phone zurück, falls dieser existiert. Ansonsten wird
// NULL zurückgegeben.
bst_node* find_node(bstree* bst, int phone) {
}


// Forward-declaration von der Funktion bst_in_order_walk_node
void bst_in_order_walk_node(bst_node* node);

// Gibt den Unterbaum von node in "in-order" Reihenfolge aus
void bst_in_order_walk_node(bst_node* node) {
}

// Gibt den gesamten Baum bst in "in-order" Reihenfolge aus. Die Ausgabe
// dieser Funktion muss aufsteigend soriert sein.
void bst_in_order_walk(bstree* bst) {
    if (bst != NULL) {
        bst_in_order_walk_node(bst->root);
    }
}

// Löscht den Teilbaum unterhalb des Knotens node rekursiv durch
```

```
33 // "post-order" Traversierung, d.h. zurerst wird der linke und dann
34 // der rechte Teilbaum gelöscht. Anschließend wird der übergebene Knoten
35 // gelöscht.
36 void bst_free_subtree(bst_node* node) {
37 }
38
39 // Löscht den gesamten Baum bst und gibt den entsprechenden Speicher frei.
40 void bst_free_tree(bstree* bst) {
41     if(bst != NULL && bst->root != NULL) {
42         bst_free_subtree(bst->root);
43         bst->root = NULL;
44     }
45 }
```