Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2014/15
Feldmann / Rost / Streibelt / Lichtblau / Deimel / Lucas

Aufgabenblatt 8

letzte Aktualisierung: 07. Januar, 14:34 Uhr

(4a202062e5d4b4e499d55b4d2cdf73f547f0eb27)

Ausgabe: Mittwoch, 07.01.2015
Abgabe: spätestens Freitag, 16.01.2015, 18:00

Thema: *Löschen aus binären Suchbäumen, Implementierung binärer 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>/Blatt<xx>/submission/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Blatt<xx>/submission/`
- Benutzen Sie 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 ersetzen ist.
  Beispiel: Aufgabe 1.2 wird zu: `introprog_blatt01_aufgabe01_2.c`
  Für jede Unteraufgabe geben Sie maximal eine Quellcodedatei ab, es sei denn, die Aufgabenstellung erfordert explizit die Abgabe mehrerer Dateien pro Aufgabe.

  Alle anderen Abgaben (Pseudocode, Textaufgaben) benennen Sie wie oben beschrieben. Die zugelassenen Abgabeformate sind PDF, ODT und Text (txt). Auch hier verwenden Sie eine Datei pro Aufgabe, nicht jedoch pro Unteraufgabe.

1. Aufgabe: Löschen aus binären Suchbäumen (5 Punkte)

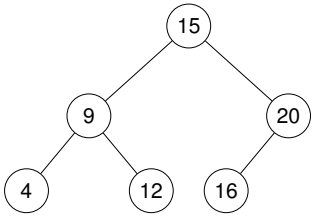

Abbildung 1: Binärbaum

Welcher Baum ergibt sich nachdem Sie die folgenden Elemente aus dem Baum aus der Vorgabe nach dem Verfahren aus der Vorlesung löschen?
(15), (9), (20), (16), (4)
Zeichnen Sie den resultierenden Baum sowie alle Zwischenresultate nach jeder Löschoperation.

Geben Sie Ihre Lösung der Aufgabe in einem Textdokument mit einem der folgenden Namen ab:
`introprog_blatt08_aufgabe01.{txt|odt|pdf}`

2. Aufgabe: Implementierung binärer Suchbäume (35 Punkte)

Schreiben Sie ein C-Programm, das auf einem binären Suchbaum folgende Operationen ausführt: Einfügen, Suchen, Ausgeben ('in-order'), Löschen.

Ihr Programm soll die auszuführenden Operationen zeilenweise aus einer Datei lesen. Das Format ist in Listing 1 verdeutlicht. Ihr Programm erhält den Namen der einzulesenden Datei als einzigen Parameter.

Listing 1: Eingabeformat (`Eingabedatei_Blatt08_02`)

```
+ 10
+ 5
+ 7
+ 9
+ 24
+ 13
+ 11
p -1
- 7
- 13
+ 55
? 9
```

`+ X` Die Zahl **`X`** soll in den binären Suchbaum eingefügt werden.

`- X` Die Zahl **`X`** soll aus dem binären Suchbaum gelöscht werden.

`? X` Es soll ausgegeben werden, ob der Wert **`X`** im binären Suchbaum enthalten ist. Ist der Wert enthalten, soll '`Der Wert X ist im Baum enthalten.\n`' ausgegeben werden. Ist der Wert nicht im Baum enthalten soll '`Der Wert X ist nicht im Baum enthalten.\n`' ausgegeben werden.

`p -1` Es sollen alle Zahlen im binären Suchbaum in 'in-order' Reihenfolge in einer Zeile ausgegeben werden. In obiger Beispieldatei sollte z.B. '` 5 7 9 10 11 13 24\n`' ausgegeben werden.

Benutzen Sie unsere Codevorgabe zur Implementierung (siehe Listing 2) und geben Sie die gesamte Aufgabe als eine Quelltextdatei `introprog_blatt08_aufgabe02.c` ab.

2.1. main-Methode (3 Punkte)

Schreiben Sie die main-Methode, so dass die Eingabedatei richtig geparst wird und die entsprechenden Ausgaben erfolgen bzw. die richtigen Funktionen aufgerufen werden.

2.2. Einfügen in den Binärbaum (6 Punkte)

Implementieren Sie die Funktion `BST_insert_value(BSTree* bst, int value)`, welche den Wert `value` in den Baum `bst` einfügt. Achten Sie darauf, dass insbesondere die `parent` Pointer richtig gesetzt sind und dass die Suchbaumeigenschaft erhalten ist: Im linken Teilbaum sind nur Werte enthalten, die kleiner gleich dem aktuellen Wert sind und der rechte Teilbaum enthält nur Werte, die größer als der aktuelle Wert sind.

2.3. Suchen im binären Suchbaum (2 Punkte)

Implementieren Sie die Funktion `BSTNode* findNode(BSTree* bst, int value)`, welche den Wert `value` in den Baum `bst` sucht und einen entsprechenden Knoten zurückgibt. Sollte der Wert nicht enthalten sein, geben Sie `NULL` zurück.

2.4. Ausgabe von Elementen im binären Suchbaum (4 Punkte)

Implementieren Sie die Funktion `BST_in_order_walk(BSTree* bst)`, welche alle Werte im binären Suchbaum in 'in-order' Reihenfolge auf `stdout` (der Konsole) ausgibt. Beachten Sie, dass bei 'in-order' Reihenfolge die Elemente immer aufsteigend geordnet sind. Die Elemente sollen durch ein einzelnes Leerzeichen getrennt sein; beenden Sie die Ausgabe auf der Kommandozeile durch eine neue Zeile (`\n`).

2.5. Löschen von einzelnen Elementen im binären Suchbaum (14 Punkte)

Implementieren Sie die Funktion `int BST_delete_value(BSTree* bst, int value)`, welche den Wert `value` im binären Suchbaum `bst` löscht. Die Funktion soll 1 zurückgeben, falls das Element im Baum enthalten war und es gelöscht wurde. Ist das Element nicht im Baum enthalten, soll 0 zurückgegeben werden.

Hinweis: Gehen Sie beim Testen der Funktionalität Ihrer Funktion sorgfältig vor und überprüfen Sie nacheinander, ob die verschiedenen Fälle (siehe Vorlesungsfolien) richtig behandelt werden.

2.6. Korrekte Freigabe des Speichers (6 Punkte)

Implementieren Sie die Funktion `void BST_delete_tree(BSTree* bst)`, welche alle Knoten im binären Suchbaum löscht. Implementieren Sie 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. Nutzen Sie diese Funktion in Ihrer `main()`-Methode, damit der gesamte Speicher freigegeben wird.

Listing 2: Codevorgabe `introprog_blatt08_aufgabe02_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>

// Knoten im Binären Suchbaum, wobei ...
// - left und right das linke bzw. rechte Kind spezifiziert.
// - parent auf denjenigen Knoten verweist, dessen Kind dieser Knoten ist.
// - value den ganzzahligen Wert des Suchbaumknotens ist.
struct BSTNode
{
   struct BSTNode* left;
   struct BSTNode* right;
   struct BSTNode* parent;
   int value;
};

// Ein Binärer Suchbaum mit einem Wurzelknoten root und der Anzahl an
// Elementen im Baum.
struct BSTree
{
   struct BSTNode* root;
   int numberOfElements;
};

// typedefs, damit man sich das "struct" sparen kann.
typedef struct BSTNode BSTNode;
typedef struct BSTree  BSTree;

// Fügt den Wert value 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_value(BSTree* bst, int value)
{
   // Hier bitte Ihr Code.
}

// Diese Funktion liefert einen Zeiger auf einen Knoten im Baum
// mit dem Wert value zurück, falls dieser existiert. Ansonsten wird
// NULL zurückgegeben.
BSTNode* findNode(BSTree* bst, int value)
{
   // Hier bitte Ihr Code.
}

// 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)
{
   // Hier bitte Ihr Code.
}


// Diese Funktion löscht einen Knoten im binären Suchbaum, der den Wert
// value hat, stellt die Baumstruktur wieder her und gibt in diesem
// Falle 1 zurück.
// Sollte Knoten den Wert value nicht enthalten, wird 0 zurückgegeben und
   ↪ kein
```

```
57 // Knoten gelöscht.
58 int BST_delete_value(BSTree* bst, int value)
59 {
60    // Hier bitte Ihr Code.
61 }
62
63
64 // Löscht den gesamten Baum bst und gibt den Speicher aller Knoten frei.
65 void BST_delete_tree(BSTree* bst)
66 {
67    // Hier bitte Ihr Code.
68 }
69
70
71 int main(int argc, char** args)
72 {
73    if (argc != 2)
74    {
75       printf("Nutzung:␣%s␣<Dateiname>\n",args[0]);
76       return 1;
77    }
78    // Hier bitte Ihr Code.
79    return 0;
80 }
```