

Hardwarenahe Programmierung / Angewandte Informatik

Übungsaufgaben – 23. Januar 2017

Aufgabe 1: Ternärer Baum

Der in der Vorlesung vorgestellte *binäre Baum* ist nur ein Spezialfall; im allgemeinen können Bäume auch mehr als zwei Verzweigungen pro Knotenpunkt haben. Dies ist nützlich bei der Konstruktion *balancierter Bäume*, also solcher, die auch im *Worst Case* nicht zu einer linearen Liste entarten, sondern stets eine – möglichst flache – Baumstruktur behalten.

Wir betrachten einen Baum mit bis zu drei Verzweigungen pro Knotenpunkt, einen sog. *ternären Baum*. Jeder Knoten enthält dann nicht nur einen, sondern *zwei* Werte als Inhalt:

```
typedef struct node
{
    int content_left, content_right;
    struct node *left, *middle, *right;
} node;
```

Wir konstruieren nun einen Baum nach folgenden Regeln:

- Innerhalb eines Knotens sind die Werte sortiert: `content_left` muß stets kleiner sein als `content_right`.
- Der Zeiger `left` zeigt auf Knoten, deren enthaltene Werte durchweg kleiner sind als `content_left`.
- Der Zeiger `right` zeigt auf Knoten, deren enthaltene Werte durchweg größer sind als `content_right`.
- Der Zeiger `middle` zeigt auf Knoten, deren enthaltene Werte durchweg größer sind als `content_left`, aber kleiner als `content_right`.
- Ein Knoten muß nicht immer mit zwei Werten voll besetzt sein; er darf auch *nur einen* gültigen Wert enthalten.

Der Einfachheit halber lassen wir in diesem Beispiel nur positive Zahlen als Werte zu. Wenn ein Knoten nur einen Wert enthält, setzen wir `content_right = -1`, und der Zeiger `middle` wird nicht verwendet.

- Wenn wir neue Werte in den Baum einfügen, werden *zuerst* die nicht voll besetzten Knoten aufgefüllt und *danach erst* neue Knoten angelegt und Zeiger gesetzt.
- Beim Auffüllen eines Knotens darf nötigenfalls `content_left` nach `content_right` verschoben werden. Ansonsten werden einmal angelegte Knoten nicht mehr verändert.

(In der Praxis dürfen Knoten gemäß speziellen Regeln nachträglich verändert werden, um Entartungen gar nicht erst entstehen zu lassen – siehe z. B. <https://de.wikipedia.org/wiki/2-3-4-Baum>.)

- (a) Zeichnen Sie ein Schaubild, das veranschaulicht, wie die Zahlen 7, 137, 3, 5, 6, 42, 1, 2 und 12 nacheinander und in dieser Reihenfolge in den oben beschriebenen Baum eingefügt werden – analog zu den Vortragsfolien ([hp-20170123.pdf](#)), Seite 21.
- (b) Dasselbe, aber in der Reihenfolge 2, 7, 42, 12, 1, 137, 5, 6, 3.
- (c) Beschreiben Sie in Worten und/oder als C-Quelltext-Fragment, wie eine Funktion aussehen müßte, um den auf diese Weise entstandenen Baum sortiert auszugeben.

Aufgabe 2: Aufräumen

Im Zuge der Übungsaufgaben von letzter Woche ([hp-uebung-20170116.pdf](#) – *Stack-Operationen, Iterativer Floodfill, Doppelt verkettete Liste*) wurden verschiedene Funktionen zur Manipulation von Arrays und Listen erstellt.

Überarbeiten Sie diese Funktionen derart, daß das Programm auch bei unsinnigen Eingabewerten nicht abstürzt, sondern eine Fehlermeldung ausgibt und sich kontrolliert beendet oder kontrolliert weiterläuft.

Hinweis: Es ist in C leider nicht möglich, bei einem Zeiger zwischen einem sinnvollen und einem zufälligen Wert zu unterscheiden. Wir müssen uns daher bei Zeigern damit begnügen, zwischen sinnvollen Zeigern auf Daten und dem Wert `NULL` zu unterscheiden.

Viel Erfolg bei den Prüfungen!