LV „ Algorithmen und Datenstrukturen“
Prof. Dr. H. Vogler / Dipl.-Math. R. Vater

WS 11/12

Aufgabenblatt zur 7. Übung

Zeitraum: 28.11. bis 02.12.2011

1. Aufgabe: (AGS 3.18)

Gegeben sei die folgende Typvereinbarung für binäre Bäume:

```
typedef struct ele *zeiger;
typedef struct ele {
  int schluessel;
  zeiger links, rechts;
} b_ele;
```

(a) Schreiben Sie eine rekursive Funktion `sum`, die die Summe aller Schlüsseleinträge eines Baumes vom oben beschriebenen Typ berechnet und als Wert zurückliefert. Die Funktion `sum` soll nur genau einen Parameter vom Typ `zeiger` haben.

(b) Schreiben Sie eine rekursive Funktion `anzahl`, die für einen beliebigen Baum des oben beschriebenen Typs die Anzahl derjenigen Knoten zählt, die

- keine Blätter sind und
- deren jeweilige Pfadlänge pfl der Bedingung $pfl \,\%\, 2 = 0$ genügt.

Der Baum wird mit Hilfe eines Parameters vom Typ `zeiger` übergeben; die ermittelte Anzahl der Knoten soll als Funktionswert zurückgeliefert werden. Geben Sie abschließend einen Funktionsaufruf an.

Hinweis: Die Pfadlänge pfl ist die Anzahl der Knoten, die man auf dem Weg von der Wurzel zum aktuellen Knoten durchläuft. Der Wurzelknoten selbst habe die Pfadlänge 1.

2. Aufgabe: (AGS 3.19)

Gegeben seien die folgenden Typ- und Variablenvereinbarungen für binäre Bäume:

```
typedef struct ele *node;
typedef struct ele {
  int key;
  node left, right;
} b_ele;

node Tree;
int anz;
```

(a) Schreiben Sie in C eine boolesche Funktion `int blatt(node t)`, die für einen beliebigen Baumknoten des gegebenen Typs ermittelt, ob es ein Blatt ist oder nicht. Liefern Sie als Funktionswert eine `1`, wenn es sich um ein Blatt handelt, sonst den Wert `0`.

(b) Gehen Sie davon aus, dass eine boolesche Funktion `int blatt(node t)` gemäß Aufgabe (a) existiert. Schreiben Sie in C eine rekursive Funktion `int anzahl(node t)`, die von einem gegebenen Baum des angegebenen Typs unter Verwendung der Funktion `blatt` die Anzahl derjenigen Knoten als Ergebnis liefert, welche keine Blätter als (unmittelbare) Nachfolger haben und selbst auch keine Blätter sind. Geben Sie abschließend ein Beispiel für einen Funktionsaufruf an.

3. Aufgabe: (AGS 3.26*)

Ausgehend von einer beliebig langen bewerteten Liste l des Typs:

```
typedef struct l_node *LPtr;
typedef struct l_node {
  int wert;
  LPtr next;
} l_nodetype;
```

soll ein bewerteter vollständiger binärer Baum t mit $t = full(l)$ wie folgt erzeugt werden:

- Die Höhe von t entspricht der Länge von l.
- Knoten von t mit gleicher Pfadlänge $k \geq 1$ erhalten dieselbe Bewertung. Die Bewertung wird dem Listenelement entnommen, welches an Position k von l steht.

Beispiel: Liste l bewerteter vollständiger binärer Baum t

Hinweise: Pfadlänge zu einem Knoten n: Anzahl der Knoten des Weges von der Wurzel zu n. Vollständiger binärer Baum: Alle Knoten bis auf die Blätter haben genau zwei Nachfolgerknoten und alle Pfade zu Blättern haben dieselbe Länge.

Höhe eines Baumes: Längster Pfad von der Wurzel zu einem Blatt.

(a) Geben Sie eine geeignete Datenstruktur in C für binäre Bäume an.

(b) Geben Sie eine Funktion `full` in C an, die im Funktionskopf einen Zeiger auf eine bestehende Liste l als Argument nimmt und einen Zeiger auf die Wurzel des erzeugten bewerteten vollständigen binären Baumes t zurückliefert.

(c) Geben Sie einen Funktionsaufruf von `full` an. Definieren Sie die benutzten Variablen.

4. Aufgabe: (AGS 5.3*)

Ein Feld a folgenden Typs sei gegeben: `typedef int feld[n];`

Dieses Feld soll mit Hilfe eines Programms aufsteigend sortiert werden, das heißt es soll also nach Abschluss der Sortierung gelten: $a_0 \leq a_1 \leq ... \leq a_{n-1}$

(a) Schreiben Sie ein C-Programm, welches folgenden Lösungsgedanken (auch unter dem Namen Bubble–Sort bekannt) realisiert:

Durchlaufe das Feld mehrfach von links nach rechts und vertausche jeweils benachbarte Elemente, falls diese nicht in der gewünschten Ordnung stehen. Das Sortieren ist beendet, wenn in einem Durchlauf keine Vertauschung mehr stattfindet. Machen Sie sich zunächst anhand eines Beispiels mit diesem Algorithmus vertraut.

(b) Ermitteln Sie die Komplexität dieses Algorithmus. Definieren Sie zunächst Ihre Basisoperationen. Unterscheiden Sie in Ihren Betrachtungen in Abhängigkeit von der Eingabefolge $\{a_i\}$, $0 \leq i \leq n-1$, Sortierungen mit geringstem (best case) und mit größtem Aufwand (worst case).

(c) Verbessern Sie den Sortieralgorithmus, so dass eine „Gleichbehandlung“ von kleinsten und größten Elementen erfolgt und bereits sortierte Teilfolgen zur Aufwandssenkung genutzt werden.

Zusatzaufgabe 1: (AGS 3.28)

Gegeben seien die Typdeklaration für einen Knoten eines Binärbaums:

```
typedef struct b_node *BPtr;
typedef struct b_node {
  int key;
  BPtr left, right;
} b_node_type;
```

die Typdeklaration für ein Listenelement sowie die globale Variable `liste`:

```
typedef struct l_ele *LPtr;
typedef struct l_ele {
  int key;
  LPtr next;
} l_ele_type;

LPtr liste = NULL;
```

Eine Funktion `int listtest(LPtr *l, int k)` soll für eine geordnete Liste `l` folgendes leisten:

Falls in `l` bereits ein Element mit dem Schlüsselwert `k` existiert, liefert sie als Funktionswert `1` (`true`), andernfalls fügt sie ein neues Listenelement mit dem Schlüsselwert `k` geordnet in die Liste ein und liefert als Funktionswert `0` (`false`).

(a) Schreiben Sie in *C* die Funktion `int listtest(LPtr *l, int k)`, wobei diese eine Funktion `void listinsert(LPtr *l, int n)` zum geordneten Einfügen eines neuen Elementes mit dem Schlüsselwert `n` benutzen darf.

(b) Gehen Sie davon aus, dass eine Funktion `int listtest(LPtr *l, int k)` entsprechend der oben genannten Spezifikation existiert.
Schreiben Sie in *C* eine rekursive Funktion `int treetest(BPtr t)`, welche für einen beliebigen Binärbaum `t` des oben genannten Typs feststellt, ob mindestens ein Schlüsselwert (des Baumes) mehrfach enthalten ist (dann Rückgabewert `1`) oder nicht (dann Rückgabewert `0`). Nutzen Sie in der Funktion `treetest` die globale Variable `liste`!

Zusatzaufgabe 2: (AGS 3.16*)

Gegeben sei die folgende Deklaration für Elemente einer verketteten Liste:

```
typedef struct list_ele *list;
typedef struct list_ele {
  int key;
  list next;
} l_ele_type;
```

(a) Schreiben Sie in *C* eine Funktion `void delete_n(list *l, int n)`, die aus einer beliebigen Liste mit Elementen des oben aufgeführten Typs *alle* Elemente mit dem Schlüsselwert `n` entfernt.

(b) Geben Sie in *C* eine Funktion `int ordered(list l)` an, welche ermittelt, ob eine solche Liste aufsteigend geordnet ist oder nicht, und die Aussage (`0` für nicht geordnet und `1` für geordnet) als Funktionswert zurückgibt.

Beachten Sie: Schlüsselwerte dürfen hier mehrfach vorkommen; in diesem Fall müssen gleiche Schlüsselwerte nebeneinander stehen.