Aufgabenblatt zur 8. Übung

Zeitraum: 05.12. bis 09.12.2011

1. Aufgabe: (AGS 3.22)

Die folgende Typdeklaration für Elemente eines Binärbaumes sei gegeben:

```
typedef struct node *B_Ptr;
typedef struct node {
  int key;
  int weight;
  B_Ptr left, right;
} bnodetype;

B_Ptr Tree;
```

(a) Geben Sie in *C* eine rekursive Funktion `void teilsumme(B_Ptr t)` an, die bei jedem Knoten eines beliebigen Binärbaumes `Tree` des oben aufgeführten Typs in `weight` die Summe aller Schlüsselwerte des Unterbaumes einträgt, der diesen Knoten als Wurzelknoten hat.

(b) Schreiben Sie in *C* eine rekursive Funktion `void gewicht(B_Ptr t)`, die bei jedem Knoten eines beliebigen Binärbaumes `Tree` des oben aufgeführten Typs in `weight` einträgt:
0, falls die Summen der Schlüsselwerte (key) beider Unterbäume gleich sind
1, falls die Summe der Schlüsselwerte im rechten Unterbaum größer ist als die im linken
-1, falls die Summe der Schlüsselwerte im rechten Unterbaum kleiner ist als die im linken

Hinweis: Schreiben Sie gegebenenfalls eine Hilfsfunktion zur Berechnung der Summe der Schlüsselwerte eines Baumes.

2. Aufgabe: (AGS 6.6)

Gegeben sei die Folge: 8 , 5 , 4 , 7 , 3.

Wenden Sie auf diese Folge den Quicksort-Algorithmus an, und dokumentieren Sie den Rechenablauf wie folgt:

- Pivotelement jeweils kennzeichnen
- Stellung der Indizes i, j unmittelbar vor dem Tausch von Elementen
- Stellung der Indizes i, j unmittelbar vor den rekursiven Aufrufen
- Teilfolgen nach den rekursiven Aufrufen

3. Aufgabe: (AGS 6.4)

Geben Sie eine Folge mit 7 selbst gewählten Zahlen an, die bei der Sortierung eine typische Situation entstehen lässt, wo der QuickSort-Algorithmus die worst-case Komplexität besitzt. Führen Sie die Sortierung mit entsprechender Protokollierung durch und geben Sie eine Abschätzung für die auszuführenden Operationen an.

4. Aufgabe: (AGS 6.20)

Gegeben sei die Folge: 3, 5, 15, 8, 13, 18, 12, 7, 6, 17.

Wenden Sie auf diese Folge den HeapSort-Algorithmus an. Dokumentieren Sie dazu in der Phase 1 das schrittweise (knotenweise) Herstellen der heap-Eigenschaft; hier insbesondere die Verndderungen durch die Funktion „sinkenlassen“.
In der Phase 2 brauchen Sie nur zwei Sortierschritte auszufhren.
Dokumentieren Sie jeweils:
- das Abspalten des jeweils letzten Elementes (Blattes) im Wechsel mit der
- Wirkung der Funktion „sinkenlassen“.

Zusatzaufgabe 1: (AGS 3.25*)

Gegeben sei folgende Typdeklaration für bewertete binäre Bäume:

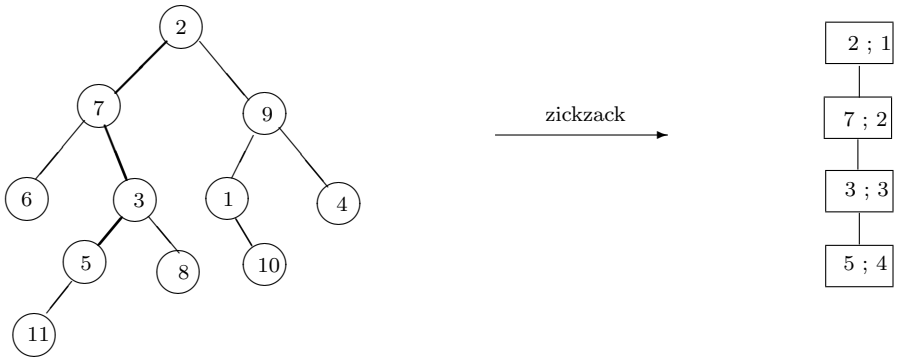
```
typedef struct node *BPtr;
typedef struct node {  int wert;
                         BPtr left,right;
                       } nodetype;
```

Aus einem bewerteten binären Baum t soll nun eine Liste l wie folgt erzeugt werden:

Der Baum t ist, von der Wurzel beginnend, auf einem links–rechts–Pfad zu durchlaufen.
Für jeden auf diesem Weg durchlaufenen Knoten soll
- ein Listenelement erzeugt werden,
- dieses an die bestehende Liste l angehängt werden
- und in das Listenelement die Bewertung des aktuell durchlaufenen Knotens sowie seine Pfadlänge gespeichert werden.

Der Pfad endet in dem Knoten, wo es keinen entsprechenden Nachfolgerknoten mehr gibt, spätestens also in einem Blatt.

Beispiel: bewerteter binärer Baum t Liste l



Hinweise:
links–rechts–Pfad bedeutet, daß der Pfad wechselweise zum linken bzw. rechten Nachfolgerknoten fortgesetzt wird. Der Pfad beginnt am Wurzelknoten des Baumes mit einem linken Nachfolgerknoten.
Pfadlänge zu einem Knoten n : Anzahl der Knoten des Pfades von der Wurzel zu n.
Die Wurzel hat die Pfadlänge 1.

(a) Geben Sie eine geeignete Datenstruktur für die gewünschte Liste l mit dem Bezeichner `list` in C an.

(b) Geben Sie eine `void`–Funktion `zickzack` in C an, die zu jedem bewerteten binären Baum t eine Liste wie oben beschrieben ausgibt. Der Funktionskopf muss mindestens zwei formale Parameter besitzen: nämlich einen Zeiger *baum* vom Typ `BPtr` und einen Zeiger *liste*, der bei Aufruf von `zickzack` auf den Kopf der zu erzeugenden Liste l zeigen soll.

(c) Geben Sie einen Funktionsaufruf von `zickzack` an. Definieren Sie ggf. die benutzten Variablen. Sie können dabei annehmen, dass der übergebene Zeiger vom Typ `BPtr` auf die Wurzel des Baumes zeigt.

Zusatzaufgabe 2: (AGS 6.19*)

Gegeben sei die Folge: 1, 2, 3, 4, 5, 6, 7, 8.

Wenden Sie auf diese Folge den Heapsort-Algorithmus an. Dokumentieren Sie in der Phase 1:
- das Einordnen in den binären Baum,
- das schrittweise (knotenweise) Herstellen der Heap-Eigenschaft; hier insbesondere die Veränderungen durch die Funktion „sinkenlassen“.

In der Phase 2 brauchen Sie nur drei Sortierschritte auszuführen.
Dokumentieren Sie jeweils:
- das Abspalten des jeweils letzten Elementes (Blattes) im Wechsel mit der
- Wirkung der Funktion „sinkenlassen“.