Algorithmen und Datenstrukturen

08. Übungsblatt

Zeitraum: 05. – 09. Dezember 2016

Übung 1 (AGS 3.2.34)

(a) Gegeben sei folgende Typdefinition für einfach verkettete Listen:

```
typedef struct element *list;
typedef struct element {
  int value;
  list next;
} element;
```

Schreiben Sie eine **iterative** Funktion `f(list l)`, die genau dann `1` zurück gibt, wenn in der Liste alle jeweils benachbarten Elemente maximal um `1` voneinander abweichen. Ansonsten soll die Funktion `0` zurück geben.

(b) Gegeben sei die folgende Typdefinition für binäre Bäume:

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

Ein *Blattknoten* ist ein Knoten, dessen linker und rechter Teilbaum jeweils leer ist. Schreiben Sie eine **rekursive** Funktion `defol(tree *p)`, welche alle Blattknoten des übergebenen Baumes entfernt.

Übung 2 (AGS 6.1.10)

Wenden Sie den Quicksort-Algorithmus auf die Folge 4,7,6,2,9 an! Die Zahlen sollen aufsteigend sortiert werden.

Dokumentieren Sie:

- Kennnzeichnung das Pivotelementes
- Stellung der Indizes *i,j* unmittelbar vor dem Tausch von Elementen
- Stellung der Indizes *i,j* unmittelbar vor dem rekursiven Aufruf
- Teilfolgen nach den rekursiven Aufrufen

Übung 3 (AGS 6.2.12)

Gegeben sei die Folge 2, 0, 9, 3, 5, 8, 4, 1, 6, 7. Wenden Sie auf diese Folge den Heapsort-Algorithmus an. In der Phase 2 brauchen Sie nur **zwei** Sortierschritte auszuführen.

Dokumentieren Sie die folgenden Arbeitschritte:

- In der Phase 1:
  - das Einordnen in einen binären Baum
  - das schrittweise (knotenweise) Herstellen der Heap-Eigenschaft; hier insbesondere die Veränderungen durch die Funktion *sinkenlassen*
- in der Phase 2: die notwendige Anzahl an Doppelschritten bestehend aus
  1. dem Austauschschritt

2. der Anwendung der Funktion *sinkenlassen*

Bei der Funktion *sinkenlassen* brauchen Sie keine Einzelschritte dokumentieren.

Zusatzaufgabe 1 (AGS 3.2.21)

Gegeben seien die folgende Typdeklaration für einen Binärbaum:

```
typedef struct node *tree;
struct node {
  int key, summe;
  tree left, right;
};
```

sowie die Typdeklaration für eine Liste:

```
typedef struct element *list;
struct element {
  int value;
  list next;
};
```

(a) Schreiben Sie in C eine Funktion `Summen(...)`, die an jedem Knoten eines beliebigen Binärbaumes des obigen Typs in das Knotenmerkmal `summe` die Summe aller geradzahligen Schlüsselwerte des Teilbaumes, der diesen Knoten als Wurzelknoten hat, einträgt.
Hinweis: Falls Sie dafür Hilfsfunktionen benötigen, müssen Sie diese vollständig angeben!

(b) Schreiben Sie in C eine Funktion `void tree_to_liste(list *l, tree t)`, die eine **fallend** geordnete Liste der Schlüsselwerte, die in dem binären **Suchbaum** (des angegebenen Typs) auftreten, erzeugt.
Geben Sie den Aufruf Ihrer Funktion an!
Hinweis: Sie können eine Funktion `void append(list *l, int n)` entsprechend des Vorlesungsskriptes voraussetzen!

Zusatzaufgabe 2 (AGS 6.2.13 ⋆)

Gegeben sei die Folge 7, 0, 8, 1, 6, 5, 2, 3, 4, 9. Wenden Sie auf diese Folge den Heapsort-Algorithmus an. In der Phase 2 brauchen Sie nur **zwei** Sortierschritte auszuführen.

Zusatzaufgabe 3 (AGS 6.1.1 ⋆)

Gegeben sei die Folge: 9, 5, 4, 2, 3, 8, 1. Wenden Sie auf diese Folge den Quicksort-Algorithmus an!