Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

2. Februar 2010

Aufgaben zur Klausur **Algorithmen und Datenstrukturen in C** im WS 2009/10 (BInf 201, BTInf 201, BMInf 201, BWInf 201)

Zeit: 75 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 8 Seiten.

Aufgabe 1:

Gegeben seien die folgenden sieben Hash-Funktionen für Strings:

```
typedef unsigned int Hash;
typedef char *String;

Hash hash1(String s) {
    return *s;
}

Hash hash2(String s) {
    Hash res;
    for (res = 0; *s; res += *s, ++s);
    return res;
}

Hash hash3(String s) {
    return (Hash) s;
}

Hash hash4(String s) {
    Hash res;
    for (res = 0; *s; res = 31 * res + *s, ++s);
    return res;
}

Hash hash5(String s) {
    Hash res;
    for (res = 0; *s; res >>= 5, res += *s, ++s);
    return res;
}

Hash hash6(String s) {
    Hash res;
    for (res = 1; *s; res *= *s, ++s);
    return res;
}

Hash hash7(String s) {
    Hash res;
    for (res = 0; *s; res = (res << 5) + *s - res, ++s);
    return res;
}
```

1. Welche dieser Funktionen erfüllen nicht die funktionalen Anforderungen an eine Hash-Funktion?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

2. Welche dieser Funktionen arbeiten bei der Berechnung mit undefinierten Werten, liefern also zufällige Resultate?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

3. Welche dieser Funktionen sind als Hash-Funktionen ungeeignet, da die Hash-Werte sehr ungleichmäßig häufig getroffen werden?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

4. Welche dieser Funktionen sind als Hash-Funktionen ungeeignet, da sie *ähnliche* Werte auf gleiche Hash-Werte abbilden?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

5. Welche Funktionen sind aus Effizienzgründen als Hash-Funktion ungeeignet?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

6. Welche Funktionen sind für die effizente Implementierung von Hash-Tabellen geeignet?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

Aufgabe 2:

Gegeben sei das folgende C-Programmstück:

```
#include <stdio.h>

typedef char * Element;

typedef struct Node * Tree;
struct Node {
  Element info;
  unsigned int arity;
  Tree * children;
};

typedef void (*ProcessElement)(Element e);

void printElement(Element e) {
  printf("%s\n", e);
}

void printTree(Tree t) {
  printElement(t-> info);
  {
    unsigned int i;
    for (i = 0; i < t->arity; ++i)
      printTree(t->children[i]);
  }
}
```

In diesem Programmteil wird eine Datenstruktur für n-stellige Bäume definiert. Die an den Knoten gespeicherte Information ist in diesem Beispiel ein Text. Die Funktion *printTree* traversiert einen Baum und gibt alle Texte an den besuchten Knoten aus.

Entwickeln Sie eine Funktion *void processTree(Tree t, ...)*, die das Traversieren einen Baumes auf die gleiche Art macht, wie *printTree*, die aber mit der an den Knoten auszuführenden Aktion parametrisiert ist:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Reimplementieren Sie *printTree* mit *processTree*:

. . . . . .

. . . . . .

. . . . . .

Entwicken Sie eine Funktion *unsigned int card(Tree t)*, die die Anzahl der Knoten in einem Baum zählt. Implementieren Sie diese Funktion mit Hilfe von *processTree*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

In dieser Aufgabe geht es darum, die Laufzeit von Operationen für verschiedene Datenstrukturen abzuschätzen. Die Laufzeit von Operationen auf Listen und Bäumen hängt üblicherweise von der Anzahl der Elemente in einer Liste oder in einem Baum ab. In dieser Aufgabe sollen $n, n_1, n_2, ...$ die Anzahl der Elemente in den Listen $l, l_1, l_2, ...$ oder Bäumen $b, b_1, b_2, ...$ bezeichnen. Verwenden Sie bitte die *Groß–O–*Notation.

1. Laufzeitkomplexität für das Einfügen eines Elementes e in eine sortierte, einfach verkettete Liste ohne Duplikate: $insert(e, l)$

   .....

2. Laufzeitkomplexität für das Einfügen eines Elementes e in eine sortierte, einfach verkettete Liste mit Duplikaten: $insert(e, l)$

   .....

3. Laufzeitkomplexität für das Einfügen eines Elementes e in eine unsortierte, einfach verkettete Liste mit Duplikaten: $insert(e, l)$

   .....

4. Laufzeitkomplexität für das Konkatenieren zweier einfach verketteter Listen: $concat(l_1, l_2)$

   .....

5. Laufzeitkomplexität für das Konkatenieren zweier einfach verketteter, als Ring implementierter Listen: $concat(l_1, l_2)$

   .....

6. Laufzeitkomplexität für das Anhängen eines Elementes e am Ende einer einfach verketteten Liste: $append(l, e)$

   .....

7. Laufzeitkomplexität für das Mischen aller Elemente zweier sortierter, einfach verketteter Listen ohne Duplikate: $merge(l_1, l_2)$

   .....

8. Laufzeitkomplexität für das Mischen aller Elemente zweier unsortierter, einfach verketteter Listen ohne Duplikate: $merge(l_1, l_2)$

   .....

9. Laufzeitkomplexität für das Suchen eines Elementes e in einer sortierten, einfach verketteten Liste mit Duplikaten: $isIn(e, l)$

. . . . . .

10. Laufzeitkomplexität im Mittel für das Suchen eines Elementes e in einem Rot-Schwarz-Baum: $isIn(e, b)$

. . . . . .

11. Laufzeitkomplexität im Mittel für das Suchen eines Elementes e in einer binären Halde: $isIn(e, b)$

. . . . . .

12. Laufzeitkomplexität im schlechtesten Fall für das Suchen eines Elementes e in einem binären Suchbaum: $isIn(e, b)$

. . . . . .

13. Laufzeitkomplexität im Mittel für das Einfügen eines Elementes e in einen binären Suchbaum: $insert(e, b)$

. . . . . .

14. Laufzeitkomplexität im schlechtesten Fall für das Einfügen eines Elementes e in einen Rot-Schwar-Baum: $insert(e, b)$

. . . . . .

15. Laufzeitkomplexität im Mittel für das Einfügen eines Elementes e in eine binäre Halde: $insert(e, b)$

. . . . . .

16. Laufzeitkomplexität für das Anhängen eines Elementes e am Ende einer einfach verketteten, als Ring implementierter Liste: $append(l, e)$

. . . . . .

17. Laufzeitkomplexität für das Einfügen eines Elementes e in eine sortierte, einfach verkettete, als Ring implementierte Liste: $insert(e, l)$

. . . . . .