

18. Februar 2008

# Klausur-Teil 3 zu Informatik I: Algorithmieren und Programmieren

## Musterlösungen

Bitte gut lesbar ausfüllen !

Name: <b>Bachmann</b>	Vorname: <b>Peter</b>
Matrikel: <b>66666666</b>	Studiengang: <b>alles Informatik-nahe</b>
Tutor: <b>ich selber</b>	Raum: <b>EHS 211</b>
Sitzreihe: <b>1</b>	Sitzplatz: <b>1</b>

### Achtung:

- Alle Antworten tragen Sie bitte direkt hinter der Aufgabenstellung in die dafür vorgesehenen Freiräume bzw. Schemata ein.
- Nebenrechnungen machen Sie bitte auf Ihrem eigenen Papier, das aber nicht eingesammelt und deshalb nicht berücksichtigt wird!
- Sie können alle Hilfsmittel in schriftlicher Form benutzen, d.h. z.B. Ihre Aufzeichnungen, das Skript zur Vorlesung, Bücher usw., aber keine Computer und keine Handys.

### Bewertung:

1	2	3	4	5	6	7	8	9	10	Summe
<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>6</b>	<b>60</b>

**1. Induktion:** (6 Punkte)

Gegeben ist die induktive Definition der Funktion  $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  durch

$$f(0, 0) = 0$$

$$f(0, n + 1) = f(n, 0) + 1$$

$$f(m + 1, n) = f(m, n + 1) + 1$$

**1.1** (3 Punkte) Bestimmen Sie:

$$f(0, 2) = f(1, 0) + 1 = f(0, 1) + 1 + 1 = f(0, 0) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3$$

$$f(2, 0) = f(1, 1) + 1 = f(0, 2) + 1 + 1 = 3 + 1 + 1 = 5$$

$$f(2, 2) = f(1, 3) + 1 = f(0, 4) + 1 + 1 = f(3, 0) + 1 + 2 = f(2, 1) + 1 + 3 = f(1, 2) + 1 + 4 = f(0, 3) + 1 + 5 = f(2, 0) + 1 + 6 = 5 + 7 = 12$$

**1.2** (3 Punkte) Beweisen Sie:  $f(m, n) = \frac{1}{2}(m^2 + 3 * m + 2 * m * n + n^2 + n)$

$$f(0, 0) \stackrel{D}{=} 0 = \frac{1}{2}(0^2 + 3 * 0 + 2 * 0 * 0 + 0^2 + 0)$$

$$f(0, n + 1) \stackrel{D}{=} f(n, 0) + 1 \stackrel{IA}{=} \frac{1}{2}(n^2 + 3 * n + 2 * n * 0 + 0^2 + 0) + 1 = \frac{1}{2}(n^2 + 2 * n + 1 + n + 1) \\ = \frac{1}{2}(0^2 + 3 * 0 + 2 * 0 * (n + 1) + (n + 1)^2 + (n + 1))$$

$$f(m + 1, n) \stackrel{D}{=} f(m, n + 1) + 1 \stackrel{IA}{=} \frac{1}{2}(m^2 + 3 * m + 2 * m * (n + 1) + (n + 1)^2 + (n + 1)) + 1 \\ = \frac{1}{2}(m^2 + 5 * m + 2 * m * n + n^2 + 3 * n + 4) \\ = \frac{1}{2}(m^2 + 2 * m + 1 + 3 * m + 3 + 2 * m * n + 2 * n + n^2 + n) \\ = \frac{1}{2}((m + 1)^2 + 3 * (m + 1) + 2 * (m + 1) * n + n^2 + n)$$

**2. Grammatiken** (6 Punkte)

In der folgenden BNF sind die Metasymbole aus der Menge  $\{S, A, B\}$  *kursiv* gesetzt, die Grundsymbole aus der Menge  $\{!, +, *, a, b\}$  in *Schreibmaschinenschrift*:

$$\begin{aligned} S &::= ! \mid A S B \mid B S A \\ A &::= + \mid a B b \\ B &::= * \mid b A a \end{aligned}$$

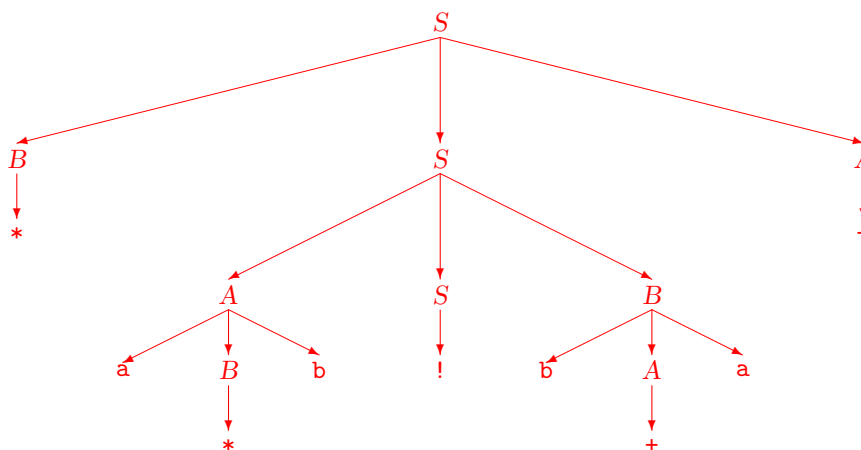
**2.1** (4 Punkte) Geben Sie alle kürzesten Texte an, die durch die BNF aus dem Satzsymbol  $S$  erzeugt werden und die mindestens ein  $a$  enthalten:

$a*b!*$   
 $+!b+a$   
 $b+a!+$   
 $*!a*b$

**2.2** (1 Punkt) Wieviele Texte können aus dem Satzsymbol  $S$  erzeugt werden, in denen mehr Zeichen  $a$  vorkommen, als Zeichen  $b$ ?:

keine

**2.3** (1 Punkt) Geben einen Syntaxbaum für den Text  $*a*b!b+a+$  an, wobei das Satzsymbol wieder  $S$  ist.



**3. Kürzeste Wege (6 Punkte)**

Gegeben ist der gerichtete Graph mit

- der Knotenmenge  
 $N = \{a, b, c, d, e, f, g, h\}$ ,
- der Kantenmenge  
 $E = \{(a, b), (a, d), (b, c), (c, h), (d, c), (d, f), (d, g), (e, b), (e, h), (f, g), (g, e)\}$
- und der Kantenmarkierung  
 $\varphi = \{((a, b), 10), ((a, d), 5), ((b, c), 10), ((c, h), 5), ((d, c), 8), ((d, f), 1), ((d, g), 3), ((e, b), 1), ((e, h), 3), ((f, g), 1), ((g, e), 1)\}$

**3.1 (3 Punkte)** Mit dem Dijkstra-Algorithmus soll der kürzeste Weg vom Knoten  $a$  zum Knoten  $h$  bestimmt werden.

Tragen Sie in die unten angeführten Tabelle die Entwicklung der Umgebung an.  $\chi(v)$  ist dabei die Weglänge vom Knoten  $a$  zum Knoten  $v$ .

**Beachten Sie**, dass nicht notwendigermaßen alle Spalten der Tabelle benötigt werden!

Knoten $v$ in Umgebung:	$a$	$d$	$f$	$g$	$e$	$b$	$h$			
$\chi(v)$ :	0	5	6	7	8	9	11			

**3.2 (3 Punkte)** Diesmal soll mit dem Ford/Moore-Algorithmus der kürzeste Weg vom Knoten  $a$  zu allen anderen Knoten bestimmt werden. Tragen Sie in die unten angeführten Tabelle die notwendigen Iterationen ein. Die Anfangssituation ist dabei bereits vorgegeben.

**Beachten Sie**, dass nicht notwendigermaßen alle Zeilen der Tabelle benötigt werden!

$a$	$b$	$c$	$d$	$e$	$f$	$g$	$h$
0	∞	∞	∞	∞	∞	∞	∞
0	10	∞	5	∞	∞	∞	∞
0	10	13	5	∞	6	8	∞
0	10	13	5	9	6	7	18
0	10	13	5	8	6	7	12
0	9	13	5	8	6	7	11
0	9	13	5	8	6	7	11

#### 4. Quicksort (6 Punkte)

Die folgende C++ Implementierung von Quicksort entspricht exakt dem Skript Seite 89:

```
void exchange(int a[], unsigned i, unsigned j)//tauscht Werte mit Indizes i und j
{ int h=a[i]; a[i]=a[j]; a[j]= h; }//exchange

unsigned med(int a[])//bestimmt den Index des mittleren Wertes
{ return a[0]<a[1]?(a[1]<a[2]?1:(a[0]<a[2]?2:0)):(a[0]<a[2]?0:(a[1]<a[2]?2:1)); }
//med

void qsort(int a[], unsigned l)
{ unsigned lp=1,rp=l-1;
  switch(l)
  { case 0 : case 1: return;
    case 2 : if(a[0]>a[1]) exchange(a,0,1); return;
    case 3 : exchange(a,1,med(a)); if(a[0]>a[2])exchange(a,0,2); return;
    default: exchange(a,0,med(a));
      while(lp<rp)
      { while((a[lp]<=a[0])&&(lp<1))++lp;
        while((a[rp]>a[0]) &&(rp>1))--rp;
        if(lp<rp) exchange(a,lp,rp);
      }
      exchange(a,0,--lp);
      qsort(a,lp); qsort(&a[lp+1],l-lp-1);
      return;
  }
}
} //qsort
```

4.1 (5 Punkte) Das Zahlenfeld in der Kopfzeile der unten angegebenen Tabelle soll mittels dieser Implementierung sortiert werden. Füllen Sie immer dann eine neue Zeile aus, wenn mittels der Funktion `exchange` eine **Veränderung** erfolgte.

**Beachten Sie**, dass nicht notwendigermaßen alle Zeilen der Tabelle benötigt werden!

4	5	6	0	1	3	2	7
5	4	6	0	1	3	2	7
5	4	2	0	1	3	6	7
3	4	2	0	1	5	6	7
3	1	2	0	4	5	6	7
0	1	2	3	4	5	6	7

4.2 (1 Punkt) Wie viele Vertauschungen werden durch `exchange` verursacht, wenn die Zahlenfolge 0 1 2 3 4 5 6 7 (in dieser Reihenfolge) sortiert wird?

Das wurde in der Vorlesung andiskutiert: 6

### 5. Haskell (6 Punkte)

Gegeben sind die folgenden Haskell-Definitionen:

```
li::Int->Int->[Int]
li _ 0 = []
li m (n+1) = m:li (m+2) n

str::Int->Int->[Int]->[Int]
str _ _ [] = []
str x y (a:l)
  | a<x      = a:str x y l
  | a==x     = str (x+y) y l
  | otherwise = str (x+y) y (a:l)

st::Int->[Int]->[Int]
st a l = str a a l

prli::[Int]->[Int]
prli [] = []
prli (p:li) = p : (prli (st p li))

prili::Int->[Int]
prili 0 = []
prili (n+1) = prli (2:(li 3 n))
```

Bestimmen Sie die Resultate für folgende Aufrufe:

li 4 5 = [4,6,8,10,12]

str 2 1 [1,2,3,4,5,6,7,8,9] = [1]

str 2 3 [1,2,3,4,5,6,7,8,9] = [1,3,4,6,7,9]

prli [2,3,4,5,6,7,8] = [2,3,5,7]

prili 1 = [2]

prili 10 = [2,3,5,7,11,13,17,19]

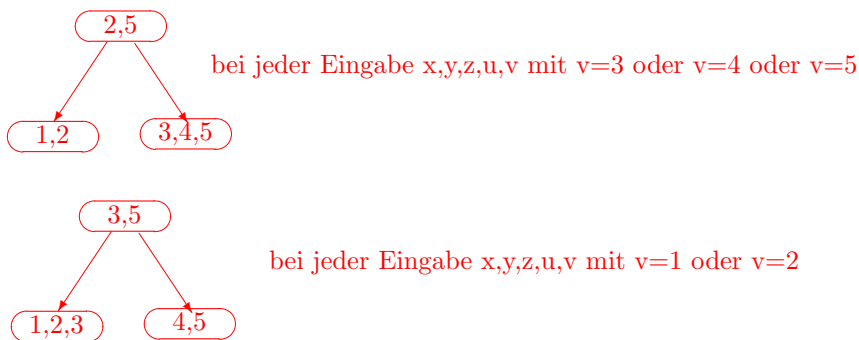
**6. (min,max)-Bäume** (6 Punkte)

**6.1.** (2 Punkte) Wieviele Informationseinheiten können in einen (2,3)-Baum der Höhe 3 (ein Baum der Höhe 1 besteht nur aus der Wurzel) **in den Blättern** eingetragen sein?

a) maximal:  $3^3 = 27$

b) minimal:  $2^3 = 8$

**6.2.** (4 Punkte) In einem (2,3)-Baum seien die Informationen mit den Schlüsseln 1,2,3,4,5 enthalten. Geben Sie alle möglichen Bäume an, die diese Informationen enthalten der entsprechende Baum entsteht.



### 7. C++/Sichtbarkeit (6 Punkte)

Tragen Sie hinter den Zeilenkommentaren die Werte für die Ausgabe ein.

```
{
  int a=1, b=10;
  {
    float a=1.3, c=4e1;
    {
      int i=8;
      a=a+b+c+i;
    }
    cout <<a;           //Ausgabe = 59.3
    {
      int a=5, b=2;
      c=a+b;
    }
    cout <<c;           //Ausgabe = 7
  }
  cout << a;           //Ausgabe = 1
  {
    unsigned u=5, b=9;
    a=b+u;
  }
  cout << a;           //Ausgabe = 14
  {
    int a=6;
    {
      int c=4;
      a=a+c;
    }
    b=a;
  }
  cout << a;           //Ausgabe = 14

  cout << b;           //Ausgabe= 10
}
```

### 8. C++/Parametervermittlung (6 Punkte)

In dem folgenden C++-Programm tragen Sie hinter dem Zeilenkommentaren die Werte für die Ausgabe ein.

```
using namespace std;
#include<iostream>

unsigned a=10,b=2;

unsigned f1(unsigned p)
{ ++p; return 2*p; }
unsigned f2(unsigned *p1, unsigned p2)
{ return (*p1)+--p2; }
unsigned f3(unsigned &p)
{ return (p--)+b; }

main()
{
    cout << f3(a)<<"\n"; //Ausgabe = 12
    cout << a<<"\n"; //Ausgabe = 9
    cout << f2(a,&b)<<"\n"; //Ausgabe = 10
    cout << b<<"\n"; //Ausgabe = 10
    cout << f1(a)<<"\n"; //Ausgabe = 2
    cout << a <<"\n"; //Ausgabe = 22
}
```

### 9. C++/ Felder (6 Punkte)

Es sei  $n \in \mathbb{N}$ , also  $n = \{0, 1, 2, \dots, n-1\}$  eine Menge natürlicher Zahlen und  $\mathcal{N}$  die Zerlegung von  $n$  in eine Menge von Teilmengen. Also gilt:

- Falls  $M \in \mathcal{N}$ , dann  $M \subseteq n$  und  $M \neq \emptyset$   
(jede Menge aus  $\mathcal{N}$  ist Teilmenge von  $n$  und nicht leer),
- $\bigcup \mathcal{N} = n$   
(jede Zahl aus  $n$  ist in einer Menge aus  $\mathcal{N}$ ) und
- Falls  $M, M' \in \mathcal{N}$  und  $M \cap M' \neq \emptyset$ , dann  $M = M'$   
(je zwei verschiedene Mengen aus  $\mathcal{N}$  sind disjunkt).

Eine Funktion  $f : n \rightarrow n$  beschreibt eine solche Zerlegung  $\mathcal{N}$ , falls für jede Zahl  $i \in n$  ein  $k \in n$  existiert, so dass  $f^k(i)$  Fixpunkt von  $f$  ist, das heißt:  $f(f^k(i)) = f^k(i)$ . Es heißt  $f^k(i)$  der Repräsentant von  $i$ . Zwei Zahlen  $i, j$  gehören genau dann beide zu einer Menge  $M \in \mathcal{N}$ , falls beide den gleichen Repräsentanten haben.

Die Funktion  $f$  wird in einer C++ Implementierung als Feld `unsigned f[]` beschrieben, so dass  $f(i) = f[i]$ .

**9.1** (2 Punkte) Beschreiben Sie eine C++ Funktion `unsigned rep(unsigned f[], unsigned i)`, so dass `rep(f, i)` als Ergebnis den Repräsentanten von  $i$  liefert.

```
unsigned rep(unsigned f[], unsigned i)
{
    return (f[i]==i)?i:rep(f,f[i]);
}
```

**9.2** (2 Punkte) Beschreiben Sie eine C++ Funktion `bool inselb(unsigned f[], unsigned i, unsigned j)`, so dass `inselb(f, i, j)` als Ergebnis genau dann `True` liefert, falls  $i$  und  $j$  Elemente der gleichen Menge sind.

```
bool inselb(unsigned f[], unsigned i, unsigned j)
{
    return rep(f,i)==rep(f,j);
}
```

**9.3** (2 Punkte) Beschreiben Sie eine C++ Funktion `void vereinige(unsigned f[], unsigned i, unsigned j)`, so dass `vereinige(f, i, j)` als Ergebnis das Feld `f` so verändert, dass jetzt die Mengen, zu denen  $i$  und  $j$  gehörten, vereinigt sind.

```
void vereinige(unsigned f[], unsigned i, unsigned j)
{
    unsigned ri=rep(f,i), rj=rep(f,j);
    if(i==j) return;
    f[rj]=ri;
}
```

**10. C++ Programmierung / binäre Bäume** (6 Punkte)

Wir betrachten geordnete binäre Bäume, also geordnete Bäume, bei denen jeder Knoten maximal zwei Söhne hat. Jeder Knoten soll mit einer natürlichen Zahl markiert sein (seine Marke).

Jede Adresse in einem geordneten binären Baum ist ein Wort  $\alpha \in \{0, 1\}^*$ , das nur aus Nullen und Einsen besteht. Für einen Baum  $b$  sei  $adr(b)$  die Menge seiner Adressen. Für die Adresse  $\alpha$  eines Knotens sei  $m(\alpha)$  dessen Marke.

Für jedes Wort  $\alpha \in \{0, 1\}^*$  sei  $\bar{\alpha}$  induktiv definiert durch

$$\bar{\varepsilon} = \varepsilon, \quad \overline{0\alpha} = 1\bar{\alpha}, \quad \overline{1\alpha} = 0\bar{\alpha}.$$

Durch Überstreichen wird also aus jeder Eins eine Null und umgekehrt. Wir sagen, durch Überstreichen wird eine Adresse *invertiert*. Für eine Adressenmenge  $A$  sein  $(\bar{A}) = \{\bar{\alpha} \mid \alpha \in A\}$ , also ist  $\bar{A}$  die Menge der invertierten Adressen aus  $A$ .

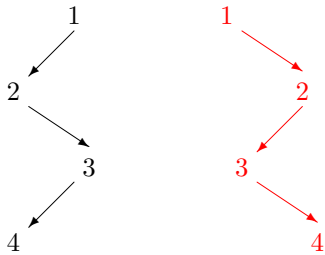
Ein Baum  $b'$  heißt das Spiegelbild eines Baumes  $b$ , falls

- $\overline{adr(b)} = adr(b')$   
(die Adressen von  $b'$  sind genau alle invertierten Adressen von  $b$ ) und
- für alle  $\alpha \in adr(b)$  gilt:  $m(\alpha) = m(\bar{\alpha})$   
(jede Marke an einer Adresse von  $b$  stimmt mit der Marke an der entsprechenden invertierten Adresse in  $b'$  überein).

**10.1** (1 Punkt) Was ist Spiegelbild des leeren Baumes?

Der leere Baum!

**10.2** (1 Punkt) Zeichnen Sie das Spiegelbild des folgenden Baumes.



**Bitte wenden, es folgt eine Teilaufgabe!**

**10.3** (4 Punkte) Geben Sie eine Implementierung der C++-Funktion `spiegel`, die eine 1 zurückgibt, falls ein Baum das Spiegelbild eines anderen Baumes ist, sonst 0. Nutzen Sie dabei die unten vorgegebene Struktur `BB` für einen binären Baum und füllen Sie nur den Körper der Funktion `spiegel` aus.

```
struct BB
{
    unsigned k;
    BB * l;
    BB * r;
};

unsigned spiegel(BB * b1, BB * b2)
{
    if(!b1&&!b2) return 1;
    if(!b1||!b2) return 0;
    if(b1->k!=b2->k) return 0;
    return spiegel(b1->l,b2->r)&&spiegel(b1->r,b2->l);
}
```