

Klausur Grundgebiete der Informatik 1

WS 08/09

(BPO07/DPO04)

Datum: 17.02.2009

Keine offizielle Lösung!

Name:
<i>(in Druckschrift)</i>	
Matr.Nr.:
Unterschrift:

Keinerlei Gewähr auf Richtigkeit!

Aufgabe	Max. Punkte	Korrektur		Einsicht	
		Punkte	Kürzel	Punkte	Kürzel
1	13				
2	5				
3	6				
4	8				
5	19				
6	9				
7	24				
8	16				
Σ	100				

Aufgabe 1 – Wissensfragen

Bitte beachten Sie, dass bei den folgenden Teilaufgaben - sofern in der Aufgabenstellung nicht anders verlangt - eine oder mehrere Antworten richtig sein können. Kreuzen Sie bei den folgenden Teilaufgaben jeweils die richtige(n) Aussage(n) an. Das Ankreuzen falscher Antworten führt nicht zu Punktabzügen. Für eine Teilaufgabe wird volle Punktzahl gegeben, wenn alle Kreuze richtig gesetzt sind. Fehlende Kreuze oder falsch gesetzte Kreuze führen dazu, dass die entsprechende Teilaufgabe mit 0 Punkten bewertet wird.

(a) Welche der folgenden Aussagen sind richtig?

- Lokale Variablen können dieselben Namen wie globale Variablen besitzen.
- Auf globale Variablen kann nur innerhalb der main-Funktion zugegriffen werden.
- Globale Variablen können in Funktionen verwendet werden.

(b) Auf den Anfang welches Strings zeigt `ptr` nach Ausführung des folgenden C-Codes?

```
char *ptr;  
char myString[] = "abcdefg";  
ptr = myString;  
ptr += 5;
```

- fg
- efg
- defg
- cdefg
- keiner der obigen

(c) Welche der folgenden Aussagen sind richtig?

- $3n^2 + 2n - 1 \in O(n^2)$
- $3n^2 + 2n - 1 \in O(n)$
- $3n^2 + 2n - 1 \in O(n^3)$

(d) Welche der folgenden Aussagen treffen zu?

- Bubblesort arbeitet nach dem divide-and-conquer Prinzip.
- Bubblesort ist stabil.
- QuickSort und Bubblesort haben im Worst-Case unterschiedliche Zeitkomplexität.

- (e) Gibt es einen (nicht unbedingt vollständigen) binären Baum mit insgesamt 10 Knoten und der angegebenen Anzahl an Blättern? (Die Blätter zählen auch als Knoten.)

0 Blätter

1 Blatt

6 Blätter

- (f) Die Fakultätsfunktion $n!$ berechnet

$$\prod_{i=1}^n i$$

Welche der nachfolgenden Schleifen entspricht der rekursiven Implementierung

```
int fakultaet(int i){  
    if (i == 1) return i;  
    else return i*fakultaet(i-1);  
}
```

`b=1; for(i=0;i<n ;i++) b=b*i;`

`b=1; for(i=0;i<n+1;i++) b=b*i;`

`b=1; for(i=1;i<n ;i++) b=b*i;`

`b=1; for(i=1;i<n+1;i++) b=b*i;`

Aufgabe 2 – Lokale und globale Variablen

Gegeben ist der folgende C-Code:

```
1  #include <stdio.h>
2
3  int count = 0;
4
5  int counter1 (int i) {
6      int count = 0;
7      count = count + i;      /* deklariert in Zeile: 6. */
8      return count;
9  }
10
11 int counter2 (int i) {
12     count = count + i;      /* deklariert in Zeile: 3. */
13     return count;
14 }
15
16 int main() {
17     int i, j1, j2;
18     for (i=0; i<=5; i++) {
19         j1 = counter1(i);
20         j2 = counter2(i);
21     }
22     printf("j1=%d\t j2=%d\t", j1, j2);
23     return 0;
24 }
```

- (a) Geben Sie an den vorgegebenen Stellen im Kommentar die Nummern der Zeilen an, in denen die dort verwendete Variable count deklariert wurde.
- (b) Welche Ausgabe erzeugt das Programm?

5 15

Aufgabe 3 – Call-by-value / Call-by-reference

Gegeben ist der folgende C-Code:

```
#include <stdio.h>

int next_fib (int *i, int *j);

int main() {
    int a = 0, b = 1;
    int i;

    printf("%d, \t%d, \t", a, b);
    for (i=0; i<9; i++) {
        printf("%d, \t", next_fib(&a, &b));
    }
    return 0;
}
```

Das Programm soll die Folge der Fibonacci-Zahlen für $n = 0, \dots, 10$ ausgeben.

Zur Berechnung der nachfolgenden Fibonacci-Zahl Fib_n aus der Summe der beiden Vorgänger $Fib_{n-1} + Fib_{n-2}$ wird die Funktion `int next_fib (int *i, int *j)` verwendet.

Eingaben sind die beiden Vorgängerzahlen $*i = Fib_{n-1}$ und $*j = Fib_{n-2}$. Rückgabewert ist die nächste Fibonacci-Zahl Fib_n . Die beiden Übergabeparameter sollen so aktualisiert werden, dass sie die beiden Vorgänger für die nächste zu berechnende Zahl zurückliefern.

Implementieren Sie die Funktion `next_fib` mit linearem Aufwand (d.h. $O(n)$).

```
int next_fib (int *i, int *j){
    sum = *i + *j;
    *i = *j;
    *j = sum;
    return sum;
}
```

g

Aufgabe 4 – Arrays und Strukturen

Gegeben seien Codeworte der Länge 10 bestehend aus Nullen und Einsen, z.B. „0110101001“. Für ein solches Codewort kann ein Parity-Bit nach gerader Parität bestimmt werden, indem die Anzahl der Einsen im Codewort zu einer geraden Anzahl ergänzt wird. Im Beispiel „0110101001“ wäre das Parity-Bit gleich 1, da das Codewort fünf Einsen enthält.

Der Datentyp zur Darstellung der Codeworte und des Parity-Bits ist durch folgende C-Struktur gegeben. Das Codewort wird als String dargestellt, das Parity-Bit als einzelnes Zeichen.

```
typedef struct
{ char codewort[11];
  char parity; } parity_struct;
```

Implementieren Sie eine Funktion `char eval_parity(parity_struct str)`, die als Eingabe eine Variable vom Typ `parity_struct` übergeben bekommt und für das darin enthaltene `codewort` das Parity-Bit berechnet und als Return-Value zurückliefert. Enthält das Codewort eine gerade Anzahl Einsen, liefert die Funktion das Zeichen '0' zurück, enthält es eine ungerade Anzahl Einsen, liefert die Funktion das Zeichen '1' zurück.

```
char eval_parity(parity_struct str){
    int i, z=0;
    for (i=0; i<10; i++)
        if (str.codewort[i] == '1') z++;
        if (str.codewort[i] == '1') z++;
    return (z%2) ? '1' : '0';
    return (z%2) ? '1' : '0';
}
```

A

Aufgabe 5 – O-Notation

(a) Berechnen Sie mit Hilfe der folgenden Rechenregeln der O-Notation

$$(1) O(c) = O(1)$$

$$(2) O(c * f(n)) = O(f(n))$$

$$(3) O(f(n)) + O(f(n)) = O(f(n))$$

$$(4) O(O(f(n))) = O(f(n))$$

$$(5) g(n) = a_k * n^k + a_{k-1} * n^{k-1} + \dots + a_0 \Rightarrow O(g(n)) = O(n^k)$$

$$(6) O(f(n)) * O(g(n)) = O(f(n) * g(n))$$

$$(7) O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$$

das Laufzeitverhalten der folgenden Funktionen. Geben Sie dabei für jeden Schritt die verwendete Rechenregel an.

$$(i) 4n^2 + \frac{n^3}{4}$$

$$O\left(4n^2 + \frac{n^3}{4}\right) \stackrel{(5)}{=} O(n^3)$$

$$(ii) 5n^4 + 7 \cdot 2^n$$

$$O(5n^4 + 7 \cdot 2^n) \stackrel{(7)}{=} O(7 \cdot 2^n) \stackrel{(2)}{=} O(2^n)$$

(b) Gegeben ist der folgende Algorithmus für ein integer-Array a der Länge n :

```
int i, j;
for (i=0; i<n-1; i++) {
    for (j=i+1; j<n; j++) {
        if(a[i] == a[j]) return 1;
    }
}
return 0;
```

(i) Was macht der Algorithmus?

überprüfen, ob 2 gleiche Elemente in dem Array $a \in \mathbb{N}$ vorhanden sind.

(ii) Analysieren Sie die Anzahl der durchgeführten Schritte im besten Fall. Begründen Sie Ihre Antwort.

1, da bei Gleichheit der ersten beiden Elemente der Algorithmus schon beim ersten Vergleich abgeschlossen ist.

(iii) Analysieren Sie die Anzahl der durchgeführten Schritte im schlechtesten Fall. Begründen Sie Ihre Antwort.

$\frac{1}{2}n^2$, da die äußere Schleife n -mal durchläuft und die innere im Mittel $\frac{n}{2} \Rightarrow n \cdot \frac{n}{2} = \frac{1}{2}n^2$

(iv) Nehmen Sie an, das Array a sei aufsteigend sortiert, d.h. es gilt $a[i] \leq a[i+1] \forall 0 \leq i < n-1$. Geben Sie einen besseren Algorithmus für dasselbe Problem an.

```
int i;
for (i=0; i<n-1; i++) {
    if (a[i] == a[i+1]) return 1;
}
return 0;
```

(v) Welche Komplexität in Form der O-Notation besitzt der verbesserte Algorithmus im schlechtesten Fall? Begründen Sie Ihre Antwort.

$O(n)$

er enthält nur noch eine Schleife, die n -mal aufgerufen wird.

Aufgabe 6 – Sortieralgorithmen

- (a) Welche Zeitkomplexität besitzt der in der Vorlesung vorgestellte Bubblesort im Worst-Case? Es ist keine Begründung erforderlich.

$$O(n^2)$$

- (b) Wenden Sie den Algorithmus auf das gegebene Array an. Sortieren Sie aufsteigend nach dem Wert der Zahlen. Stellen Sie alle Veränderungen im Datenfeld dar. Interne Zustandsvariablen des Algorithmus müssen nicht dargestellt werden. Umkreisen Sie dabei jeweils die beiden Elemente, die miteinander vertauscht wurden.

1	7	3	0	5	4	9	2	6
1	7	3	0	5	4	(2)	(9)	6
1	7	3	0	5	(2)	(4)	9	6
1	7	3	0	(2)	(5)	4	9	6
1	7	(0)	(3)	2	5	4	9	6
1	(0)	(7)	3	2	5	4	9	6
(0)	(1)	7	3	2	5	4	9	6
0	1	7	3	2	5	4	(6)	(9)
0	1	7	3	2	(4)	(5)	6	9
0	1	7	(2)	(3)	4	5	6	9
0	1	(2)	(7)	3	4	5	6	9
0	1	2	(3)	(7)	4	5	6	9
0	1	2	3	(4)	(7)	5	6	9
0	1	2	3	4	(5)	(7)	6	9
0	1	2	3	4	5	(6)	(7)	9

Aufgabe 7 – Lineare Datenstrukturen

Gegeben ist die Datenstruktur einer einfach verketteten Liste mit Daten vom Typ `int`:

```
typedef struct node {
    int data;
    struct node* next; } Node, *nodeptr;
typedef struct list {
    nodeptr first, last; } List, *listptr;
```

Weiterhin sind die folgenden Funktionen gegeben:

```
/* Erzeugung einer neuen leeren Liste */
void Init(listptr);

/* Prüft, ob das Element in der Liste enthalten ist.
   Liefert 1 zurück, wenn das Element in der Liste enthalten ist, sonst 0 */
int IsIn(nodeptr, listptr);

/* Einfügen eines Elements an das Ende der Liste */
void AppendLast(nodeptr, listptr);
```

- (a) Implementieren Sie eine Funktion `listptr Union(listptr p1, listptr p2)`. Die Funktion gibt eine **neue** Liste zurück, die der Vereinigung der beiden Listen entspricht. Mehrfachvorkommen von Elementen sollen dabei eliminiert werden. Gehen Sie davon aus, dass beide Listen nicht leer sind. Gehen Sie weiterhin davon aus, dass innerhalb von Liste 1 keine Elemente mehrfach enthalten sind, dies gilt ebenso für Liste 2. Die oben deklarierten Funktionen sollen für die Implementierung verwendet werden.

Beginnen Sie mit Ihrer Lösung auf der nächsten Seite!

```

nodeptr newnode (int datum) {
    nodeptr np;
    np = (nodeptr) malloc(sizeof(Node));
    np->data = datum;
    np->next = NULL;
    return np;
}

listptr Union (listptr p1, listptr p2) {
    nodeptr nodeptr np;
    List united;
    Listptr United = &united;
    Init (United);
    np = p1->first;
    while (np != NULL) {
        if (!IsIn (np, p2)) {
            AppendLast (newnode (np->data), United);
            np = np->next;
        }
    }
    np = p2->first;
    while (np != NULL) {
        AppendLast (newnode (np->data), United);
        np = np->next;
    }
    return United;
}

```

- (b) Wie hoch ist die Laufzeitkomplexität der Funktion in Form der O-Notation? Begründen Sie Ihre Antwort.
 Hinweis: Gehen Sie bei Ihrer Analyse davon aus, dass beide Listen die Länge n besitzen. Achten Sie darauf, dass die Funktion IsIn für eine Liste der Länge n die Komplexität $O(n)$ besitzt.

1. while-Schleife n Durchläufe mit jeweils n ($IsIn$) Schritten

2. while-Schleife n Durchläufe mit jeweils 1 Schritt.

Alles andere $O(1)$

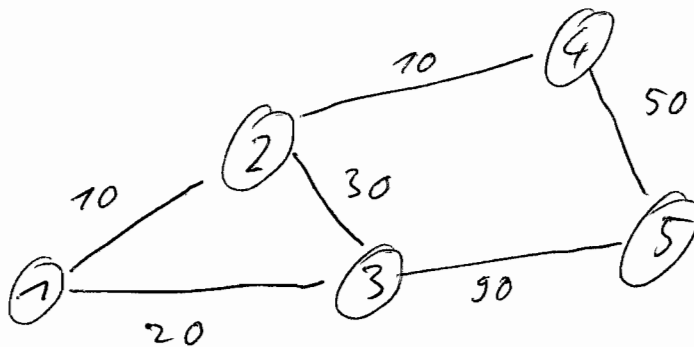
$$\Rightarrow O(n^2 + n) = O(n^2)$$

Aufgabe 8 – Graphen

Gegeben ist ein ungerichteter, kantengewichteter Graph $G = (V, E, w)$, der durch die Adjazenzmatrix In des folgenden C-Codes repräsentiert wird. Einträge ungleich Null stellen die Kantengewichte dar. Ein 0 in der Adjazenzmatrix bedeutet, dass keine Kante zwischen den beiden Knoten vorhanden ist. Die Knotenmenge V besteht aus den Knoten 1, 2, 3, 4, 5.

```
int In[5][5] =  
{  
    0, 10, 20, 0, 0,  
    10, 0, 30, 10, 0,  
    20, 30, 0, 0, 90,  
    0, 10, 0, 0, 50,  
    0, 0, 90, 50, 0  
};
```

(a) Zeichnen Sie den durch die Adjazenzmatrix repräsentierten Graphen.



- (b) Führen Sie für den obigen Graphen den Algorithmus von Kruskal zur Berechnung des **minimalen** Spannbaums durch. Geben Sie dafür die sortierte Kantenliste L und die Liste T der Kanten des Spannbaums an. Nutzen Sie die folgende Tabelle, um jede Änderung von T oder L in einer neuen Tabellenzeile darzustellen. Geben Sie die Kanten in der Form (A, B) an für eine Kante von Knoten A nach Knoten B. Skizzieren Sie zuletzt den Spannbaum.

step	T	L
init	$\{\}$	$\{(1,2), (2,4), (1,3), (2,3), (4,5), (3,5)\}$
1	$\{(1,2)\}$	$\{(2,4), (1,3), (2,3), (4,5), (3,5)\}$
2	$\{(1,2), (2,4)\}$	$\{(1,3), (2,3), (4,5), (3,5)\}$
3	$\{(1,2), (2,4), (1,3)\}$	$\{(2,3), (4,5), (3,5)\}$
4	$\{(1,2), (2,4), (1,3), (4,5)\}$	$\{(2,3), (3,5)\}$ $\{(3,5)\}$

