Informatik II 4. Oktober 2004

gelesen von Prof. Zitterbart

Durchfallquote:
Nachklausur: 78 %

Zum Bestehen waren 30 von 60 Punkten erforderlich.

Umbruch: Micha Lenk, Timo Weingärtner
Herausgeber: Fachschaft Mathematik/Informatik

Für die Wiedereinführung der echten verfassten Studierendenschaft

Klausur Informatik II vom 4. Oktober 2004 (Prof. Zitterbart)

Aufgabe 1 (10P):
Beantworten Sie die folgenden Fragen kurz aber präzise.

1. Warum wählt man als Hash-Funktion üblicherweise keine injektive Funktion? (2P)
2. Welche der in der Vorlesung besprochenen Formen der Mengen- bzw. Matrix-Darstellung von Graphen erlauben das Einfügen eines Knotens in einen Graphen mit O(1) Aufwand? (3P)
3. Zum Problem des minimalen Spannbaums: Wieso kann ein solcher minimaler spannender Graph keine Zyklen enthalten? (1P)
4. Erklären Sie das EDF-Scheduling-Verfahren. Welche Informationen über Prozesse müssen für dieses Verfahren vorliegen? (2P)
5. Wieso ist der Kontext-Wechsel zwischen Threads effizienter als zwischen Prozessen? Was ist der Nachteil von Threads gegenüber Prozessen? (2P)

Aufgabe 2 (10P):

Sie analysieren das Bestellsystem des Instituts für Telematik: Zu Beginn jeder Anschaffung wird eine Bestellung angelegt. Anschließend wird die Bestellung an einen Lieferanten verschickt. Die Rechnung wird bezahlt, sobald die Lieferung eingetroffen ist. Trifft eine Lieferung 14 Tage nach dem Verschicken der Bestellung nicht ein, so wird der Lieferant gemahnt. Trifft die Lieferung 30 Tage nach dem Verschicken der Bestellung nicht ein, so wird der Lieferant als unzuverlässig markiert und die Bestellung neu angelegt. Formulieren Sie ein UML Zustandsdiagramm für den Bestellvorgang. Verwenden Sie sinnvolle Namen für die einzelnen Aktionen, Bedingungen und Zustände. Verwenden Sie so wenig Zustände wie möglich.

Aufgabe 3 (10P):

Gegeben sei ein Rot-Schwarz-Baum.

a) Welche fünf Eigenschaften gelten für jeden Rot-Schwarz-Baum? (2,5P)

b) Ein Knoten k soll aus einem Rot-Schwarz-Baum entfernt werden. Welche Eigenschaften von Rot-Schwarz-Bäumen werden in Abhängigkeit von Farbe und Position des Knotens k im Baum verletzt? Begründen Sie ihre Lösung kurz. (4,5P)

c) Fügen Sie in einen anfangs leeren Rot-Schwarz-Baum die Zahlen 5, 7, 2, 4, 3 in dieser Reihenfolge ein. Beschriften Sie die Knoten entsprechend ihrer Farben: Ein r neben einem Knoten kennzeichnet einen roten Knoten, ein s kennzeichnet einen schwarzen Knoten. Zeichnen Sie den Baum auf diese Weise nach jedem Einfügeschritt. (3P)

Aufgabe 4 (10P):

Gegeben ist nachfolgende Funktion `int ulam(int n)`, die rekursiv die so genannten Ulam-Zahlen berechnet.

```
int ulam(int n) {
      System.out.println(n);
      if (n==1)
            return 1;
      else {
            if ((n % 2) == 0)
                  return ulam(n/2);
            else
                  return ulam(3*n+1);
      }
}
```

a) Wie lautet die Ausgabe der Funktion ulam, wenn sie mit dem Parameter 3 aufgerufen wird? (2P)

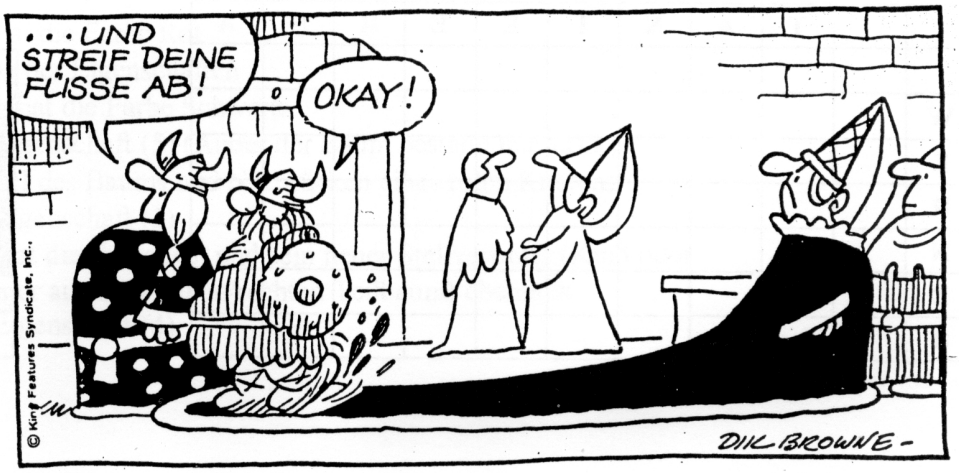
b) Entwerfen Sie eine alternative Funktion `void ulam (int n)`, die iterativ für eine natürliche Zahl n die gleiche Ausgabe wie die rekursive Variante berechnet. (4P)

c) Nachfolgend ist der Java-Quelltext der Funktion partition angegeben, die den Divide-Schritt im Quicksort Algorithmus implementiert. Wie in der Vorlesung teilt sie eine Liste list von start bis end in zwei Teile, einen linken Teil, der alle Elemente kleiner als der Median p aufnimmt, sowie einen rechten Teil, der alle Elemente größer als der Median p aufnimmt. Im Gegensatz zur Vorlesung haben sich hier jedoch einige syntaktische und semantische Fehler eingeschlichen. Finden und korrigieren Sie diese Fehler. Markieren Sie dazu im Lösungs-Quelltext die Fehler deutlich, und schreiben Sie die Korrektur direkt unterhalb die Zeile, in der der Fehler auftritt. (4P)

Hinweis:

In der Vorlesung wurde immer das mittlere Element einer zu sortierenden Teil-Liste als neuer Median ausgewählt.

```
private int partition(int start, int end, Object p) {
     int i == start, j = end;
    while (i <= j) {
          while ((j <= i) && ((Comparable)list.get(i)).compareTo(p) < 0)
                i++;
          while ((i <= j) && ((Comparable)list.get(j)).compareTo(i) > 0)
                i--;
          if (i <= j) {
                list.swap(i, j);
                i++; j++;
          }
    }
    return i;
}
```



Aufgabe 5 (10P):

Gegeben sei das folgende Java-Code-Fragment:

```
double y = 0;
int i = a.length - 1;
while (i>=0) {
      y = a[i] + x * y;
      i--;
}
```

Beweisen Sie, dass der Code das Polynom $P(x) = \sum_{k=0}^{a.length-1} a[k]x^k$ an der Stelle x auswertet.

Im Array double a[] seien die Koeffizienten des Polynoms P(x) gespeichert.

Zu zeigen ist, dass mit der Vorbedingung $P:=\{a\neq null\}$ und der Ausführung des Codes die Nachbedingung $Q:=\{y=P(x)\}$ erreicht wird, y enthält also den Wert des Polynoms an der Stelle x.

a) Beweisen Sie zunächst die partielle Korrektheit des Programms. Verwenden Sie als Schleifen-Invariante $I := \{ y = \sum_{k=i+1}^{a.length-1} a[k]x^{k-i-1} \wedge i \geq -1 \}$ (9P)

Hinweis:

Für beliebige i gilt allgemein $\sum_{i}^{i-1} = 0$

b) Zeigen Sie dass die Schleife terminiert. (1P)

Aufgabe 6 (5P):

Beweisen oder widerlegen Sie: $f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O(2^{g(n)})$

Aufgabe 7 (5P):

Gegeben sei der Text A="**TANTEERNA**", sowie das Muster B="**WERNER**". Führen Sie mit dem Verfahren aus der Vorlesung eine approximative Zeichenkettensuche von B in A durch. Geben Sie schließlich an, wo im Text das komplette Muster mit einer maximalen Editierdistanz von 3 Kosten-Einheiten endet.

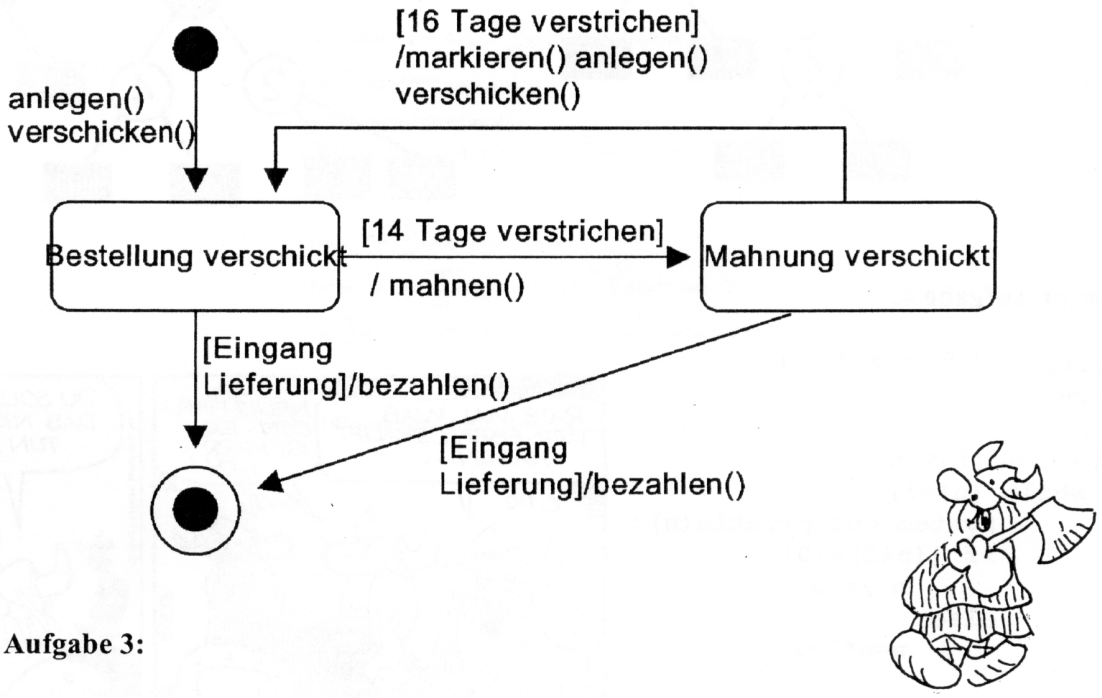
Hinweis:
Folgende Tabelle könnte Ihnen dabei helfen.

		T	A	N	T	E	E	R	N	A
W										
E										
R										
N										
E										
R										

Lösung zu Aufgabe 1:

1. Lösung: Eine injektive Funktion hat einen Wertebereich, der gleich der maximal möglichen Anzahl der einzufügenden Schlüssel ist. Dementsprechend muss die Hash-Tabelle entsprechend groß sein, um diese Anzahl Schlüssel aufzunehmen. Da die tatsächliche Anzahl der Schlüssel meist deutlich geringer ist, werden kleinere Hash-Tabellen mit nicht-injektiven Funktion verwendet.
2. Lösung: Kantenliste, Knotenliste, Adjazenzliste
3. Lösung: Würde ein solcher Graph Zyklen enthalten, dann könnte man die Kanten, die die Zyklen bilden, entfernen, somit Kosten sparen und dennoch alle Knoten erreichen.
4. Lösung: Hier wird der Prozessor immer dem Prozess zugeordnet, dessen Frist als nächstes abläuft. Das Verfahren benötigt die Ablauf-Frist der Prozesse.
5. Lösung: Der Adressraum muss nicht gewechselt werden. Nachteil: Kein eigener Speicherbereich (Schutz), Synchronisations-Mechanismen erforderlich.

Lösung zu Aufgabe 2:



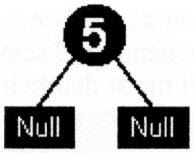
Lösung zu Aufgabe 3:

a) Lösung:
   (1) Jeder Knoten ist entweder rot oder schwarz.
   (2) Die Wurzel ist schwarz.
   (3) Jedes Blatt (Null) ist schwarz.
   (4) Wenn ein Knoten rot ist, so sind beide Kinder schwarz.
   (5) Für jeden Knoten gilt, dass alle Pfade vom Knoten zu einem Blatt dieselbe Anzahl schwarzer Knoten beinhalten.

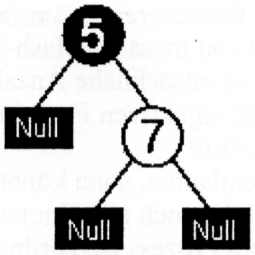
b) Lösung:
   - Entfernter Knoten hat die Farbe Rot
     - Keine Verletzung der Eigenschaften
   - Entfernter Knoten k hat die Farbe Schwarz
     - Verletzung der Eigenschaft (5) (Außer der Baum bestand nur aus k)
   - Entfernen der Wurzel des Baums und Nachrücken eines roten Knotens
     - Verletzung der Eigenschaft (2)
   - Vaterknoten von k hat die Farbe Rot und sein neuer Stellvertreter (Kind oder Vorgänger/Nachfolger aus dem linken/rechten Teilbaum) ebenfalls
     - Verletzung der Eigenschaft (4)

c) Lösung:

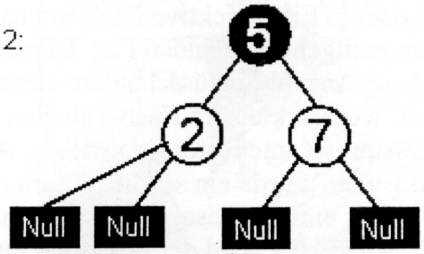
insert 5:



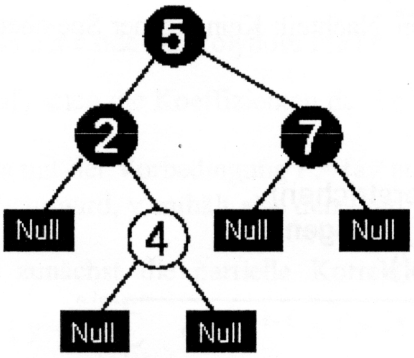
insert 7:



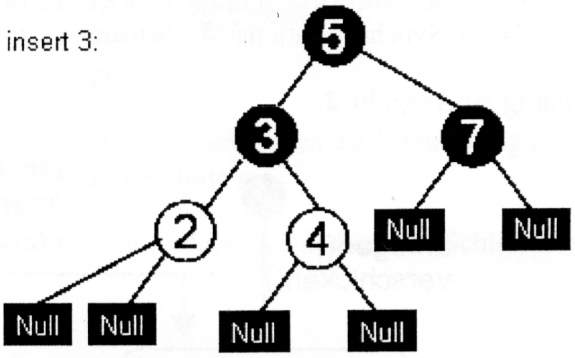
insert 2:



insert 4:



insert 3:



Lösung zu Aufgabe 4:

a) Lösung: 3, 10, 5, 16, 8, 4, 2, 1

b) Lösung:

```
void ulam(int n) {
    while (n!=1) {
          System.out.println(n);
          if ((n%2)==0)
                n /= 2;
          else
                n=n*3+1;
    }
    System.out.println(n);
}
```

c) Lösung:

```
private int partition(int start, int end, Object p) {
    int i = start, j = end;
    while (i <= j) {
          while ((i <= j) && ((Comparable)list.get(i)).compareTo(p) < 0)
                i++;
          while ((i <= j) && ((Comparable)list.get(j)).compareTo(p) > 0)
                j--;
          if (i <= j) {
                list.swap(i, j);
                i++; j--;
          }
    }
    return i;
}
```

5) a) Antwort:

1. I gilt vor der Schleife: P⇒wp(„`double y = 0; int i = a.length - 1;`“, I)

   $= \text{wp}(\text{„}\texttt{double y = 0;}\text{“}, y = \sum_{k=a.length}^{a.length-1} a[k]x^{k-a.length} \wedge \text{a.length-1} \geq -1)$

   $= \{0 = \sum_{k=a.length}^{a.length-1} a[k]x^{k-a.length} \wedge \text{a.length-1} \geq -1\}$

   $= \{\text{TRUE} \wedge \text{a.length} \geq 0\} \Leftarrow \{\text{a} \neq \text{null}\}$

2. Aus I∧B⇒wp(„`y=a[i]+x*y;i--;`“, $y = \sum_{k=i+1}^{a.length-1} a[k]x^{k-i-1} \wedge i \geq -1$)

   $= \text{wp}(\text{„}\texttt{y=a[i]+x*y;}\text{“}, y = \sum_{k=i}^{a.length-1} a[k]x^{k-i} \wedge i \geq 0)$

   $= \{a[i] + x * y = \sum_{k=i}^{a.length-1} a[k]x^{k-i} \wedge i \geq 0\}$

   $= \{x * y = -a[i] + a[i] \sum_{k=i+1}^{a.length-1} a[k]x^{k-i} \wedge i \geq 0\} = \{x * y = \sum_{k=i+1}^{a.length-1} a[k]x^{k-i} \wedge i \geq 0\}$

   $= \{y = \sum_{k=i+1}^{a.length-1} a[k]x^{k-i-1} \wedge i \geq 0\} = \text{I} \wedge \text{B}$ ($x \neq 0$), für x=0 gilt: linke und rechte Seite =0.

5) b) Antwort:
Betrachte i: i fällt streng monoton und ist durch 0 begrenzt.

6) Lösung:

Falsch, Gegenbeispiel:
Betrachte f(n)=n+n, g(n)=n, dann gilt f(n)=O(g(n))
Aus $2^{f(n)}=O(2^{g(n)}) \Rightarrow \exists c \in R, n_0 \in N \rightarrow \forall n \geq n_0 : 0 \leq 2^{n+n} \leq c2^n$ müsste gelten.

Aber es gibt n' für die gilt $2^{n'}2^{n'}>c2^{n'}$, z.B. $n'>\log_2 c$
$2^{\log_2 c}2^{\log_2 c} = c^2 = c2^{\log_2 c}$

7) Lösung:

		T	A	N	T	E	E	R	N	A
	0	0	0	0	0	0	0	0	0	0
W	1	1	1	1	1	1	1	1	1	1
E	2	2	2	2	2	1	1	2	2	2
R	3	3	3	3	3	2	2	1	2	2
N	4	4	4	3	4	3	3	2	1	2
E	5	5	5	4	4	4	3	3	2	2
R	6	6	6	5	5	5	4	**3**	**3**	**3**