Technische Universität München
Fakultät für Informatik
Dr. H. Ehler, Dr. M. Pizka,
W. Prenninger, Dr. M. Schneider

WS 2001
Lösungsvorschläge zu Blatt 10
20. Dezember 2001

Übungen zu Einführung in die Informatik I

Aufgabe 40 Suchalgorithmen (Lösungsvorschlag)

a) Folgende Tabelle zeigt als Beispiel eine absteigende sortierte Sequenz s ganzer Zahlen und zwei unterschiedliche Auffassungen vom Begriff "n-größtes Element".

Sequenz s	20	20	18	17	17	17	16	13	...
n-größtes Element	1	2	3	4	5	6	7	8	...
n-größtes Element	1	1	2	3	3	3	4	5	....

Der entscheidende Unterscheid ist: wie geht man mit Duplikaten in der Sequenz um? Obiges Beispiel legt zwei unterschiedliche Definitionen des Begriffs "n-größtes Element" einer Sequenzen ganzer Zahlen nahe:

(i) Alle Duplikate werden bei der Bestimmung des n-größten Elements berücksichtigt. Definition: Sei s eine Sequenz ganzer Zahlen und $sort(s)$ die zu s zugeordnete absteigend sortierte Sequenz. Dann ist das n-größte Element von s das n-te Element von $sort(s)$.

(ii) Duplikate werden bei der Bestimmung des n-größten Elements nicht berücksichtigt. Definition: Sei s und $sort(s)$ wie bei (i) definiert. Dann ist das n-größte Element von s das n-te Element von $sort(reduziert(s))$, wobei $reduziert(s)$ die zu s zugeordnete Sequenz ist, die alle paarweise verschiedenen Elemente von s enthält.

b) Die Implementierung der Funktion `zerlege` ist unabhängig von der in a) gewählten Definition:

```
zerlege :: Int -> [Int] -> ([Int], [Int], [Int])
zerlege p [] = ( [], [], [] )
zerlege p (x:xs) | x>p       = (x:grp,   glp,   klp)
                 | x==p      = (  grp, x:glp,   klp)
                 | otherwise = (  grp,   glp, x:klp)
                   where (grp, glp ,klp) = zerlege p xs
```

c) Implementierung gemäß Definition (i):

```
quickmax :: Int -> [Int] -> Int
quickmax n s | n <= anz_grp = quickmax n grp
             | n <= (anz_grp + anz_glp) = head glp
             | otherwise = quickmax (n - (anz_grp + anz_glp)) klp
               where (grp, glp ,klp) = zerlege (head s) s;
                              anz_grp = length grp;
                              anz_glp = length glp;
                              anz_klp = length klp
```

Implementierung gemäß Definition (ii):

```
-- quickmax2 stuetzt sich auf reduziert und quickmax
quickmax2 :: Int -> [Int] -> Int
quickmax2 n s = quickmax n (reduziert s)

-- Implementierung von reduziert mit Abstuetzung auf eleminiereN
reduziert :: [Int] -> [Int]
reduziert [] = []
reduziert (x:xs) = x:(reduziert (eliminiereN x xs))

eliminiereN :: Int -> [Int] -> [Int]
eliminiereN n [] = []
eliminiereN n (x:xs) | x==n = eliminiereN n xs
                     | otherwise = x:(eliminiereN n xs)
```

Aufgabe 41 Effizienz rekursiver Algorithmen (Lösungsvorschlag)

a) **fct** croot = (**nat** n) **nat** :
    if n $\stackrel{?}{=}$ 0 **then** 0
        else **nat** x = croot(n-1)+1;
            if x*x*x $\leq$ n **then** x
                else x-1
            fi
    fi

b) **fct** bicroot = (**nat** n) **nat** :
    if n $\stackrel{?}{=}$ 0 **then** 0
        else **nat** x = 2*bicroot(n $\div$ 8)+1;
            if x*x*x $\leq$ n **then** x
                else x-1
            fi
    fi

c) Im Falle von croot(n) sind n rekursive Aufrufe erforderlich. Im Falle von bicroot(n) sind lediglich etwa $log_8(n)$ rekursive Aufrufe erforderlich. Für größere n ergeben sich hieraus erheblich Effizienzunterschiede.

   croot(n) $\approx O(n)$; bicroot(n) $\approx O(\log_8 n)$

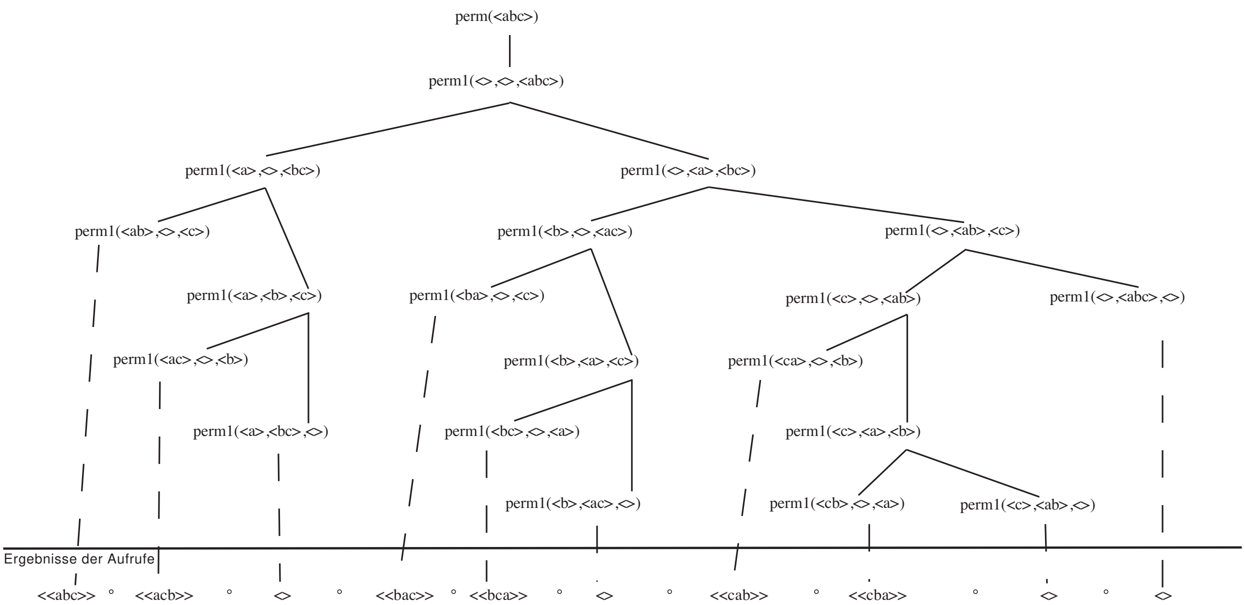
Aufgabe 42 Permutationen auf Sequenzen (Lösungsvorschlag)

a) Die Technik der Einbettung besteht darin, eine Aufgabe als Spezialfall einer allgemeineren Aufgabe zu lösen. Oft ist nämlich eine allgemeinere Aufgabe einfacher als der Spezialfall. In der Hilfestellung ist eine Einbettung von *perm* auf eine Rechenvorschrift *perm1* bereits vorgegeben.

fct perm = (**seq m** s) **seq m** :
  perm1(<>,<>,s),
fct perm1 = (**seq m** t, **seq m** l, **seq m** r) **seq m** :
  if r $\stackrel{?}{=}$<> **then** <>
    elif (rest(r) $\stackrel{?}{=}$<>) ∧ (l $\stackrel{?}{=}$<>) **then** <t ∘ <first(r)> >
    else perm1(t ∘ <first(r)>, <>, l ∘ rest(r)) ∘ perm1(t, l ∘ <first(r)>, rest(r))
    fi
  fi

b) Der Rekursionstyp von *perm1* ist kaskadenartig, da im else-Zweig der Fallunterscheidung zwei rekursive Aufrufe von perm1 erfolgen.

c) Der Aufrufbaum für *perm(<abc>)*:



Aufgabe 43 (Lösungsvorschlag)

a) In Pseudocode lautet die gesuchte Funktion:

fct A_{fib} = (**nat** n :¬(n = 0)) **nat**:
  if n $\stackrel{?}{=}$1 or n $\stackrel{?}{=}$2 **then** 1
    else $1+A_{fib}(n-1)+A_{fib}(n-2)$
  fi

b) Zu zeigen ist die Gültigkeit der Aussage $P(n) = (A_{fib}(n) \stackrel{?}{=} 2 \cdot fib(n) - 1)$. Dies geschieht mittels vollständige Induktion über n.

Induktionsanfang: Wir haben hier den Fall einer Induktion mit zwei Induktionsanfängen und es ist die Gültigkeit von $P(n_0)$, mit $n_0 = 1$ und $P(n_1)$, mit $n_1 = 2$ zu zeigen. Gemäß Definition ist $A_{fib}(1) = 1$ und $A_{fib}(2) = 1$; andererseits ist, ebenfalls per Definition, $fib(1) = 1$ und $fib(2) = 1$, woraus $2 \cdot fib(1) - 1 = 1$ bzw. $2 \cdot fib(2) - 1 = 1$ folgt; somit gilt die Aussage $P(n_0)$ und $P(n_1)$.

Induktionsschluss: Es gelte nun $P(n)$ und $P(n+1)$ (**Induktionshypothese**). Zu zeigen ist, dass daraus die Gültigkeit von $P(n+2)$ folgt.
Wir betrachten $A_{fib}(n+2)$. Dann gilt:

$$\begin{aligned}
A_{fib}(n+2) &= 1+A_{fib}(n+1)+A_{fib}(n) && \text{(Definition von } A_{fib}\text{)}\\
&= 1+(2\cdot fib(n+1)-1)+(2\cdot fib(n)-1) && \text{(Induktionsannahmen)}\\
&= 2\cdot(fib(n+1)+fib(n))-1 &&\\
&= 2\cdot fib(n+2)-1 && \text{(Definition von fib(n))}
\end{aligned}$$

Es gilt also $A_{fib}(n+2) = 2\cdot fib(n+2)-1$.

c) Wir zeigen die Gültigkeit von

$$f(n,1,1,1) = f(n,k,fib(k+1),fib(k)),\ \text{ für } 1 \leq k \leq n. \quad (*)$$

durch Induktion über k.

Induktionsanfang: Sei $k = 1$. Dann gilt per Definition der Fibonacci-Funktion

$$f(n,1,fib(2),fib(1)) = f(n,1,1,1),$$

und der Induktionsanfang somit bewiesen.

Induktionsschluss: Es gelte die Behauptung (*) für ein $1 \leq k < n$. Zu zeigen ist dann die Gültigkeit von $f(n,1,1,1) = f(n,k+1,fib(k+2),fib(k+1))$:

$$\begin{aligned}
f(n,1,1,1) &= f(n,k,fib(k+1),fib(k)) && \text{(Induktionsannahme)}\\
&= f(n,k+1,fib(k+1)+fib(k),fib(k+1)) && \text{(Definition von f)}\\
&= f(n,k+1,fib(k+2),fib(k+1)) && \text{(Definition von fib)}
\end{aligned}$$

Die Aussage (*) ist also bewiesen.

Insbesondere gilt nun die Aussage (*) auch für $k = n$:

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

Ferner gilt nach Definition von f: $f(n,n,fib(n+1),fib(n)) = fib(n)$. Die Funktion $fib(n)$ kann also durch den Aufruf $f(n,1,1,1)$ eingebettet werden.

d) Die Funktion f ist linear rekursiv. Für die Anzahl der rekursiven Aufrufe A_f zur Berechnung von $f(n)$ gilt: $A_f(n) = n$. Die Funktion fib dagegen ist nichtlinear rekursiv. Es ergibt sich folgende Tabelle (Die Berechnung von $A_{fib}(n)$ erfolgt beispielsweise mit Hilfe einer geeigneten Goferfunktion):

n	$A_f(n)$	$A_{fib}(n)$
1	1	1
5	5	9
10	10	109
20	20	13 529
30	30	1 664 079