Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

WS 2001
Übungsblatt 10
20. Dezember 2001

Übungen zu Einführung in die Informatik I

Aufgabe 40 Suchalgorithmen

In dieser Aufgabe soll ein Algorithmus entwickelt werden, der für jede Sequenz von ganzen Zahlen das *n*-größte Element ausgibt.

a) Diskutieren Sie den Begriff "*n*-größtes Element" einer Sequenz von ganzen Zahlen und geben Sie eine oder gegebenenfalls mehrere Definitionen an.

Nach geeigneter Begriffsdefinition wollen wir nun eine Problemlösung entwickeln, die auf der Idee des *Quicksort*-Algorithmus beruht. Das Prinzip des Quicksort-Algorithmus zum Sortieren einer Sequenz *s* ganzer Zahlen lautet:

1. Wähle ein beliebiges Element *p* aus *s*.
2. Zerlege *s* in drei Sequenzen:
   groesserp, die alle Elemente aus *S* enthält, die größer sind als *p*,
   gleichp, die alle Elemente aus *s* enthält, die gleich sind zu *p*, und
   kleinerp, die alle Elemente aus *s* enthält, die kleiner sind als *p*.
3. Wende die Schritte 1 und 2 rekursiv auf die Sequenzen kleinerp und groesserp an, bis die Sequenzen Länge 1 erreicht haben (diese sind trivialerweise sortiert) und füge die sortierten Teilsequenzen zu der schließlich sortierten Gesamtsequenz zusammen.[1]

Bei der Suche nach dem *n*-größten Element einer Sequenz kann man nach demselben Prinzip vorgehen. Allerdings genügt es, das entsprechende Maximum nur in einer der Sequenzen groesserp, gleichp und kleinerp zu suchen (Warum?). Zur Suche des *n*-größten Element ist also das Sortieren der Sequenz nicht notwendig!

b) Entwickeln Sie eine Goferfunktion `zerlege p s`, die eine Sequenz *s* in die drei Teilsequenzen groesserp, gleichp und kleinerp zerlegt und als Tripel `(groesserp, gleichp, kleinerp)` ausgibt.

c) Entwickeln Sie gemäß Ihrer Begriffsdefinition aus Aufgabe (a) eine Goferfunktion `quickmax n s`, die sich auf `zerlege` stützt und das *n*-größte Element der Sequenz *s* ausgibt.

[1] Die algorithmische Methode, die bei Quicksort angewendet wird, nennt man "divide and conquer"-Verfahren.

Aufgabe 41 Effizienz rekursiver Algorithmen

Die Funktion croot(n) sei wie folgt spezifiziert:

fct croot = (**nat** n) **nat**:
some nat y: $y^3 \leq n < (y+1)^3$

a) Geben Sie in Vorlesungsnotation eine einfache Rechenvorschrift für die Berechnung der Funktion croot(n) an.

b) Analog zu der in Vorlesung und Buch angegeben Berechnung der Quadratwurzel mittels Binarisierung kann auch die dritte Wurzel erheblich effizienter durch Binarisierung berechnet werden. Geben Sie eine entsprechende Rechenvorschrift bicroot an.

Hinweis: Wenn man eine Zahl n durch 8 teilt, dann ist die dritte ganzzahlige Wurzel von n÷8 wg. $2^3 = 8$ etwa die Hälfte der dritten ganzzahligen Wurzel von n.

c) Vergleichen Sie die Effizienz der beiden Varianten, indem Sie die Aufruftiefe abschätzen.

Aufgabe 42 Permutationen auf Sequenzen

Permutationen sind bijektive Abbildungen $\pi : [1,n] \to [1,n]$. Die Darstellung erfolgt meist in Kurzschreibweise $(\pi(1),\ldots,\pi(n))$. Eine Permutationen π induziert eine Abbildung $\hat{\pi}$ auf Zeichenreihen der Länge n über einem Alphabet T wie folgt

$$\hat{\pi} : (T^*)^\perp \to (T^*)^\perp, \quad \hat{\pi}(< s_1 \ldots s_n >) = < s_{\pi(1)} \ldots s_{\pi(n)} >$$

Die Sequenz $\hat{\pi}(s)$ wird auch Permutation von s genannt.

a) Schreiben Sie eine Rechenvorschrift *perm*, die zu einer Seuqenz s die Sequenz aller Permutationen von s liefert. Zum Beispiel gilt also

$$perm(< abc >) = << abc >< acb >< bac >< bca >< cab >< cba >>$$

Hilfestellung: Verwenden Sie die Technik der Einbettung. Schreiben Sie dazu eine rekursive Rechenvorschrift *perm1* mit Funktionalität

fct perm1 = (**seq m** t, **seq m** l, **seq m** r) **seq m**

die die Sequenz aller Zeichenreihen $t \circ p$ liefert, wobei p eine Permutation von $l \circ r$ ist, die aber mit einem Zeichen aus r beginnt. $perm(s)$ läßt sich dann berechnen als $perm1(<>,<>,s)$.

b) Welchen Rekursionstyp hat *perm1*?

c) Zeichnen Sie den Aufrufbaum für $perm(< abc >)$.

Aufgabe 43 (H) Vergleich verschiedener Implementierungen der Fibonaccifunktion

Aus der Vorlesung ist Ihnen die Fibonaccifunktion

```
fct fib = (nat n: ¬(n ?= 0)) nat:
  if n ?= 1 or n ?= 2 then 1
                      else fib(n-1) + fib(n-2)
  fi
```

bekannt.

a) Formulieren Sie (in Pseudocode) eine Funktion A_{fib}(n), die die Anzahl der bei der Berechnung von `fib(n)` auftretenden Funktionsaufrufe ermittelt.

b) Zeigen Sie: $A_{fib}(n) = 2 \cdot \text{fib(n)} - 1$

c) Es sei die Funktion

```
fct f = (nat n, nat k, nat a, nat b) nat:
    if k =?n then b
            else f(n, k+1, a+b, a)
    fi
```

gegeben.

Man zeige, dass *fib(n)* in f durch den Aufruf *f(n, 1, 1, 1)* eingebettet werden kann, d.h. für alle *n* gilt: $fib(n) = f(n,1,1,1)$.

Hinweis: Man zeige

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

d) Vergleichen Sie die Rekursionstypen von *fib* und f! Geben Sie die Anzahl der rekursiven Aufrufe von *fib* und f für $n \in \{1,5,10,20,30\}$ an!

Wir wünschen Allen ein frohes Weihnachtsfest und einen guten Rutsch ins neue Jahr!