

## Übungen zu Informatik I

### Aufgabe 12-1                      Textnachrichten auf Mobiltelefonen                      (keine Abgabe)

Teilaufgabe (a) bis (e) in folgendem SML-Kode:

```
(* a *)
datatype ziffer = z2 | z3 | z4 | z5 | z6 | z7 | z8 | z9

(* b *)
(* zunaechst definieren wir eine Abbildung toZiff, welche einem
   Buchstaben die korrespondierende Ziffer zuordnet *)

fun toZiff(x) = if (ord(x) <= ord ("c")) then z2 else
                if (ord(x) <= ord ("f")) then z3 else
                if (ord(x) <= ord ("i")) then z4 else
                if (ord(x) <= ord ("l")) then z5 else
                if (ord(x) <= ord ("o")) then z6 else
                if (ord(x) <= ord ("s")) then z7 else
                if (ord(x) <= ord ("v")) then z8 else z9

(* die elementweise Anwendung wird mit map realisiert; explode
   wandelt Zeichenketten in Listen von Zeichen um *)
val kodiere = (map toZiff) o explode;

(* c *)
(* Zun"achst polymorph in dem Typ der Elemente , die in den Knoten
   gespeichert werden und im Typ ueber den verzweigt wird *)

datatype ('a, 'b) otree = empty | node of 'a * ('b -> ('a, 'b) otree)

(* diesen Typ instantiiieren wir nun f"ur unsere Zwecke *)
type Otree = (string list, ziffer) otree

(* d *)
(* Loesung durch Rekursion mit Einbettung: einfuegen' fuegt die
   Zeichenkette s an dem durch die Liste z von Ziffern bestimmten
   Knoten ein *)

fun einfuegen' (s, nil, empty) = node ([s], fn x => empty)
  | einfuegen' (s, nil, node(l, k)) = node(s::l, k)
  | einfuegen' (s, z::zs, empty) =
      node(nil, fn x => if (x = z) then einfuegen' (s, zs, empty) else empty)
  | einfuegen' (s, z::zs, node(l, k)) =
      node(l, fn x => if (x = z) then einfuegen' (s, zs, k(z)) else k(x))
```

```

(* einfuegenn ist nun das einfuegenn einer Zeichenkette an der durch
ihren Kode bestimmten Stelle *)

fun einfuegen (s, b) = einfuegen' (s, kodiere(s), b);

(* e *)
(* suchen1 *)
fun suchen1 (nil, empty) = nil
  | suchen1 (nil, node (l, k)) = l
  | suchen1 (z::zs, node (l, k)) = suchen1 (zs, k(z))
  | suchen1 (z::zs, empty) = nil;

(* Zwischendurch zum Testen *)
(* Erzeugen eines schoenen Baums, der genau die strings aus einer
gegebenen Liste enthaelt *)
fun baue' (nil, b) = b | baue' (x::xs, b) = baue'(xs, einfuegen(x, b));
fun baue (l) = baue' (l, empty);

(* ein Obaum, der ein paar Woerter enthaelt *)
val lied = baue ["ein", "mops", "kam", "in", "die", "kueche", "und",
                "stahl", "dem", "koch", (* "ein", hammer scho *) "ei"];

(* suchen nach Worten *)
val exhund = suchen1 (kodiere("hund"), lied);
val exmops = suchen1 (kodiere("mops"), lied);
val exein = suchen1 (kodiere("ein"), lied);
val exkoch = suchen1 (kodiere("koch"), lied);
val exgaertner = suchen1 (kodiere("gaertner"), lied);
val exei = suchen1 (kodiere("ei"), lied);
val exstahl = suchen1 (kodiere("stahl"), lied);
val exeisen = suchen1 (kodiere("eisen"), lied);

(* f *)
fun suchen2 (z, nil) = nil
  | suchen2 (z, w::ws) = if (z = kodiere(w)) then (w::suchen2(z, ws))
  else suchen2 (z, ws);

```

Bestimmung der Komplexität: Streng genommen wurden in der Vorlesung Komplexitätsklassen nur für Funktionen einer Variablen definiert; in dieser Aufgabe (und auch in der Realität) interessiert man sich aber auch für Komplexitätsklassen für Funktionen mehrere Veränderlicher. Wir verwenden im folgenden die naheliegende Verallgemeinerung des Komplexitätsbegriffs auf Funktionen mehrerer Veränderlicher.

Wir bezeichnen die Länge der Ziffernfolge, für die passende Wörter gesucht werden sollen, mit  $n$ , die Größe (dh. Anzahl der Wörter) des Wörterbuchs mit  $k$ .

Funktion *extrahiere*: Um alle Zeichenketten des Wörterbuchs, die von einer Ziffernfolge mit Länge  $n$  kodiert werden, zu finden, benötigt man  $n$  Schritte. Daher  $extrahiere \in \mathcal{O}(n)$ .

Funktion *suche*: Um alle Zeichenketten zu finden, die von einer Ziffernfolge kodiert werden,

müssen wir alle Wörter des Wörterbuchs überprüfen. Ein durchschnittliches, in einem Wörterbuch enthaltenes Wort hat Länge  $\log k$ , dh. bei jedem Wort müssen  $\min\{\log k, n\} \in \mathcal{O}(\log k)$  Zeichen verglichen werden. Daher  $extrahiere \in \mathcal{O}(k \cdot \log k)$ .

Die Moral: Typischerweise ist die Anzahl der Einträge in einem Wörterbuch ("k") sehr groß, zumindest verglichen mit der Länge einer Ziffernfolge ("n"), die ein Wort kodiert. Deshalb bevorzugen wir die Funktion *extrahiere*.

Eine Implementierung beider Verfahren in LISP zeigt bei einem Wörterbuch der Größe 45.000, das die Suche im O-Baum im Schnitt ca. 50.000 mal schneller als die Suche in der Liste abläuft. (In SML konnten wir keine Liste dieser Länge erzeugen.)

### Aufgabe 12-2

### Ringpuffer

(8 Punkte)

a) Konstant, linear, konstant.

b) (\* 12-2 (b) \*)

```
exception empty_list;

signature BUFSig =
sig
  type 'a buf
  val new_buffer : 'a list -> 'a buf
  val advance    : 'a buf -> 'a buf
  val value      : 'a buf -> 'a
end;

structure DBUF : BUFSig =
struct
  type 'a buf = 'a list * 'a list
  fun new_buffer xs = (xs, xs)
  fun advance (xs,y1::y2::ys) = (xs,y2::ys)
    | advance (xs,y::nil) = (xs,xs)
    | advance _ = raise empty_list
  fun value (_,x::xs) = x
    | value _ = raise empty_list
end;

use "vector.sml";

structure VBUF : BUFSig =
struct
  type 'a buf = 'a vect * int
  fun new_buffer xs = (linit xs, 1)
  fun advance (v,n) = (v,if n = dim v then 1 else n + 1)
  fun value (v,n) = get(v,n)
end;
```

c) DBUF.new\_buffer [1,3,7,1,2]

### Aufgabe 12-3

### Rekursion

(4 Punkte)

```

fun fromto i j f =
  if i <= j
  then (f i)::(fromto (i+1) j f)
  else nil;;

fun f1 0 = 3
  | f1 1 = 2
  | f1 2 = 1
  | f1 n = f1(n-1) + 2*f1(n-2) + 3*f1(n-3);;

fromto 0 15 f1;;

fun f2 n = let fun iter j a1 a2 a3 =
  if j >= n
  then a3
  else iter (j+1) (a1 + 2*a2 + 3*a3) a1 a2;
in iter 0 1 2 3 end;;

fromto 0 15 f2;;

```