Technische Universität München
Fakultät für Informatik
M. Bauer, D. Golias, S. Riesner,
Dr. M. Schneider

WS 2002/2003
Lösungsvorschläge zu Blatt 6
21. November 2002

Übungen zu Einführung in die Informatik I

Aufgabe 20 Optimalität der Huffman-Codierung

Um zu zeigen, dass die Huffmancodierung c der Quelle S optimal ist, müßen wir zeigen, dass die mittlere Länge der Codierung eines Zeichens $L(c)$ minimal ist. Dies beweisen wir durch Induktion über die Anzahl der Zeichen.

Induktionsanfang: Wenn $q = 1$ ist, $c = \{\varepsilon\}$ mit $L(c) = 0$ und damit die Codierung optimal.

Induktionsschluß: Induktionsannahme: $q > 1$; Huffmancodierung ist optimal für alle Quellen S^* mit $q-1$ Zeichen.

Bezeichne Z^* den Zeichensatz, der sich durch Reduktion gemäß dem Huffmanalgorithmus (vgl. Aufgabe 15) ergibt, c^* die entsprechende Codierungsfunktion, z^* das Zeichen, das die beiden Zeichen mit minimaler Häufigkeit z_{q-1} und z_q ersetzt und p^* die Summe der beiden minimalen Häufigkeiten. Dann gilt: $|Z^*| = q-1$. Darüber hinaus ergibt sich:

$$
\begin{aligned}
L(c) &= \sum_{i=1}^{q} p_i|c(z_i)| \\
&= \sum_{i=1}^{q-2} (p_i|c(z_i)|) + p_{q-1}|c(z_{q-1})| + p_q|c(z_q)| \\
&= \sum_{i=1}^{q-2} (p_i|c(z_i)|) + p_{q-1}|c^*(z^*)\circ < O > | + p_q|c^*(z^*)\circ < L > | \\
&= \sum_{i=1}^{q-2} (p_i|c(z_i)|) + (p_{q-1} + p_q)(|c^*(z^*)| + 1) \\
&= \sum_{i=1}^{q-2} (p_i|c(z_i)|) + p^*|c^*(z^*)| + p^* \\
&= \sum_{i=1}^{q-1} (p_i|c^*(z_i)|) + p^* \\
&= L(c^*) + p^*.
\end{aligned}
$$

Gemäß der Induktionsannahme ist $L(c^*)$ minimal. Da p^* ebenfalls minimal ist, ist auch $L(c)$ minimal.

Aufgabe 21 Eigenschaften der Entropie

a) Die Entropie ist bekanntlich durch

$$H(s) = -\sum_{i=1}^{q} p_i \text{ld } p_i$$

gegeben; außerdem ist $0 \leq p_i \leq 1 \Rightarrow \text{ld } p_i \leq 0$. Somit besteht die Summe nicht aus negativen Summanden und kann insgesamt nicht negativ sein.

b)

$$\begin{aligned} H(S) = 0 &\Leftrightarrow p_i \text{ld} p_i = 0, \forall i \in \{1,..,q\} \\ &\Leftrightarrow p_i = 0 \vee (\text{ld } p_i = 0), \forall i \in \{1,..,q\} \\ &\Leftrightarrow p_i = 0 \vee p_i = 1, \forall i \in \{1,..,q\} \end{aligned}$$

Da aber $\sum_i p_i = 1$ sein muß, muß genau ein $p_i = 1$ sein und das Prädikat $\forall j : j \neq i \Rightarrow p_j = 0$ gelten . Die Entropie erreicht in diesem Fall also ihr Minimum.

c) In dem angegebenen Satz wählen wir in dem angegebenen Hilfssatz $x_i = p_i$ und $y_i = 1/q$. Dann folgt:

$$\begin{aligned} H(S) &= \sum_{i=1}^{q} p_i \text{ld } \frac{1}{p_i} \\ &\leq \sum_{i=1}^{q} p_i \text{ld } q \\ &= \text{ld } q \sum_{i=1}^{q} p_i \\ &= \text{ld } q \end{aligned}$$

Bemerkung: Der angegebene Hilfsatz läßt sich mit der bekannten Abschätzung $\ln x \leq x - 1$, für $x > 0$, leicht beweisen.

d) Einsetzen der Gleichverteilung $p_i = 1/q$ in die Definition der Entropie liefert: $H(S) = \text{ld } q$ als maximale Entropie.

Aufgabe 22 LZW-Kompressionsalgorithmus

In dieser Aufgabe soll die Funktionsweise des LZW-Algorithmus aus der Vorlesung anhand eines Beispiels von Hand nachvollzogen werden.

Eine gute Erklärung der Funktionsweise des LZW-Algorithmus findet sich auch unter

`http://www.la.unm.edu/~mbrettin/algorithms/lzexplained.html`

a) Gehen Sie die Schritte durch, die bei der Komprimierung der Zeichenkette `abrakadabra-kadabrakadabrakadabra...` ablaufen. Wie groß ist die Einsparung durch die Codierung in diesem Fall?

In Pseudo-Code sieht der Algorithmus wie folgt aus:

```
Initialize string table;
prefix  <- empty;
while (K <- next character in charstream) do
    Is prefix^K in string table?
        yes: prefix <- prefix^K;
        no:  add prefix^K to the string table;
             output the code for prefix to the codestream;
             prefix <- K;
end
output the code for prefix to the codestream
```

Die Auswertung für den String “abrakadabrakadabra...” läuft dann wie folgt ab:
Initialisierung des *string-table* mit $\{0 = a, 1 = b, \ldots, 25 = z\}$.

prefix	*next-char*	*string-table*	*code-output*
""	'a'		
"a"	'b'	$26 =$ "ab"	0
"b"	'r'	$27 =$ "br"	1
"r"	'a'	$28 =$ "ra"	17
"a"	'k'	$29 =$ "ak"	0
"k"	'a'	$30 =$ "ka"	10
"a"	'd'	$31 =$ "ad"	0
"d"	'a'	$32 =$ "da"	3
"a"	'b'		
"ab"	'r'	$33 =$ "abr"	26
"r"	'a'		
"ra"	'k'	$34 =$ "rak"	28
"k"	'a'		
"ka"	'd'	$35 =$ "kad"	30
"d"	'a'		
"da"	'b'	$36 =$ "dab"	32
"b"	'r'		
"br"	'a'	$37 =$ "bra"	27
"a"	'k'		
⋮	⋮	⋮	⋮

b) Gehen Sie die Schritte durch, die bei der Dekomprimierung von `[7; 14; 10; 20; 18; 15; 27; 29; 26; 28; 30; 32; 18; 34; 29; 31; 35]` ablaufen.
Die Dekompression sieht in Pseudo-Code so aus:

```
Initialize string table;
get first code: <code>
output the string for <code> to the charstream;
<old> = <code>
while ( <code> <- next code in codestream) do
    does <code> exist in the string table?
        (yes: output the string for <code> to the charstream;
            [...] <- translation for <old>
            K <- first character of translation for <code>;
            add [...]^K to the string table;
            <old> <- &ltcode>;
         )
         (no: [...] <- translation for <old>;
            K <- first character of [...];
            output [...]^K to charstream and add it to string table;
            <old> <- <code>
         )
end
```

Die Auswertung für `[7; 14; 10; 20; 18; 15; 27; 29; 26; 28; 30; 32; 18; 34; 29; 31; 35]` läuft dann wie folgt ab:
Initialisierung des *string-table* mit $\{0 = a, 1 = b, \ldots, 25 = z\}$.

next-code	*output*	*string-table*
7	"h"	
14	"o"	26 = "ho"
10	"k"	27 = "ok"
20	"u"	28 = "ku"
18	"s"	29 = "us"
15	"p"	30 = "sp"
27	"ok"	31 = "po"
29	"us"	32 = "oku"
26	"ho"	33 = "ush"
28	"ku"	34 = "hok"
30	"sp"	35 = "kus"
32	"oku"	36 = "spo"
18	"s"	37 = "okus"
34	"hok"	38 = "sh"
29	"us"	39 = "hoku"
31	"po"	40 = "usp"
35	"kus"	41 = "pok"

c) **(H) Implementierung des LZW Algorithmus in OCaml**

Diese Implementierung ist ineffizient, aber sehr elegant.

Zuerst benötigen wir eine Reihe von Hilfsfunktionen:

```
let string_of_char a = Char.escaped a;;
let eq a b = (a=b);;
let initCodeTable = ["a";"b";"c";"d";"e";"f";"g";"h";"i";"j";"k";"l";"m";
                     "n";"o";"p";"q";"r";"s";"t";"u";"v";"w";"x";"y";"z"];;

(* position eines elements in der liste *)
let rec indexInList_step li a k = match (li, k) with
   | ([], _) -> failwith "element not found in list"
   | (x::tl, k) when (x=a) -> k
   | (x::tl, k)            -> indexInList_step tl a k+1;;

let indexInList li a = match li with
   | [] -> failwith "list is empty"
   | li -> indexInList_step li a 0;;
```

(i) Schreiben Sie eine Funktion *compress*: *string* → *int list* die eine Zeichenkette nimmt und als Liste von Zahlen entsprechend dem LZW-Algorithmus zurückgibt.

```
let rec compress_step prefix char_list string_table code_list =
  match char_list with
   | []    -> code_list @ [indexInList string_table prefix]
   | x::tl when List.exists (eq (prefix^string_of_char(x))) string_table
           -> compress_step (
   (* haenge x an prefix an *)
                 prefix^string_of_char(x))
   (* weiter mit dem rest *)
                 tl
```

```
  (* tabelle unveraendert *)
              string_table
  (* keine ausgabe *)
              code_list
  | x::tl -> compress_step (
  (* neues prefix *)
              string_of_char(x))
  (* weiter mit dem rest *)
              tl
   (* fuege prefix^"x" in tabelle ein *)
             (string_table @ [prefix^string_of_char(x)])
  (* ausgabe: code von prefix *)
             (code_list @ [indexInList string_table prefix])
  ;;

let compress char_list = match char_list with
  | [] -> []
  | x::tl -> compress_step (string_of_char(x)) tl initCodeTable [];;
    (* einbettung mit x als prefix, ["a";"b";...;"z"] als tabelle
       und leere liste als ausgabe *)

compress (list_of_string("abrakadabrakadabrakadabrakadabrakadabra"));;
```

(ii) Schreiben Sie eine Funktion *uncompress*: *int list* → *string* die eine so kodierte Liste wieder in eine Zeichenkette umwandelt.

```
let rec uncompress_step lastcode code_list output_list string_table =
  match code_list with
  | [] -> output_list
  | x::tl when ((List.length ( string_table )) > x)
         -> uncompress_step x
               tl
  (* ausgabe: decode von x *)
               (output_list @ [List.nth string_table x])
  (* [...] <- translation for <old> *)
  (* K <- first character of translation for <code> *)
  (* add [...]^K to the string table; *)
               (string_table @ [(List.nth string_table lastcode)
                  ^(string_of_char(String.get (List.nth string_table x) 0))
   | x::tl -> uncompress_step x
                             tl
   (* K <- first character of [...]; output [...]^K *)
          (output_list @
            [(List.nth string_table lastcode)
             ^(string_of_char(String.get (List.nth string_table lastcode) 0))
   (* and add it to string table *)
          (string_table @
            [(List.nth string_table lastcode)
             ^(string_of_char(String.get (List.nth string_table lastcode) 0))
  ;;
```

```
let uncompress code_list = match code_list with
 | [] -> []
 | x::tl -> uncompress_step x
      tl
      [(List.nth initCodeTable x)] (* erstes zeichen gleich ausgeben *)
      initCodeTable;;

String.concat "" (uncompress [7; 14; 10; 20; 18; 15; 27; 29; 26;
      28; 30; 32; 18; 34; 29; 31; 35]);;
```