Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
Übungsblatt 3
2. Juni 2006

Übungen zu Einführung in die Informatik II

Aufgabe 8 Eigenschaften der Entropie

Beweis der Implikation $H(S) = \text{ld}\, q \Leftarrow p_i = \frac{1}{q} \quad \forall i$ folgt durch Einsetzen (vgl. Musterlsg.).

zu zeigen: $H(S) = \text{ld}\, q \Rightarrow p_i = \frac{1}{q} \quad \forall i$

Nach Aufgabe 1c gilt: $H(S) \leq \text{ld}\, q$, insbesondere $H(S)$ maximal bei ld q

daher verbleibt nur zu zeigen, dass falls $H(S)$ maximal, dann $p_i = \frac{1}{q}$.

Es gilt: $\log x < x - 1$ (vgl. Anmerkung Aufgabe 1c)

Damit: $\log(\frac{1}{qp_i}) \leq \frac{1}{qp_i} - 1$

$\Leftrightarrow -p_i(\log q + \log p_i) \leq \frac{1}{q} - p_i$ mit p_i durchmultiplizieren und Logarithmus aufteilen

$\Leftrightarrow -\sum_i p_i \log q - \sum_i p_i \log p_i \leq \sum_i \frac{1}{q} - \sum_i p_i$ über i summieren

$\Leftrightarrow -\sum_i p_i \log q + H(S) \leq 0$ mit Entropie ersetzen und Wkt.summe ausnutzen

$\Leftrightarrow H(S) \leq H([p_i = \frac{1}{q} \quad \forall i])$

Die Entropie ist damit für alle Verteilungen kleiner als für die Gleichverteilung, daher gilt

$H(S) = \text{ld}\, q \Rightarrow H(S)$ maximal $\Rightarrow p_i = \frac{1}{q} \quad \forall i$

Aufgabe 9 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:

$$L(c) = \sum_{i=1}^{q} p_i |c(z_i)|$$

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

Gemäß der Induktionsannahme ist $L(c^*)$ minimal. Da p^* ebenfalls minimal ist, ist auch $L(c)$ minimal. Wir haben somit gezeigt, dass der Huffmanalgorithmus einen optimalen Präfixcode erzeugt. Zusätzlich lässt sich beweisen, dass man eine optimale Datenkompression, die auf Zeichencodierung basiert, stets durch die Verwendung eines Präfixcodes erreichen kann. Der oben durchgeführte Beweis gilt daher allgemein. D.h. es kann keine bessere / kürzere Codierung geben, als die vom Huffmanalgorithmus erzeugte. Allerdings gibt es neben dieser Lösung evtl. noch andere genauso gute Lösungen (indem man z.B. gleichlange Codierungen verschiedener Zeichen vertauscht).

Aufgabe 10 Prüfpolynome und CRC-Berechnung (Lösungsvorschlag)

a) Gegeben sei das Generator Polynom $G_1(x) = x^{16} + x^{12} + x^5 + 1$.
Bestimmen Sie die Prüfsumme der Nachricht 100010000011111 mittels $G_1(x)$ nach dem CRC-Verfahren.

```
10001000001111100000000000000000
10001000000100001
----------------
00000000001011110100000000000
          10001000000100001
          -----------------
            11001000010000100
            10001000000100001
            -----------------
             10000000101001010
             10001000000100001
             -----------------
                 10001011010110
                 ==============
```

b) Ein Empfänger erhält eine mit $G_2(x) = x^3 + x + 1$ gesicherte CRC-Nachricht 1101110111010. Überprüfen Sie, ob ein Übertragungsfehler eingetreten ist.

Ist kein Übertragungsfehler aufgetreten, so müsste restfreie Division der gesicherten Nachricht mit dem Generatorpolynom möglich sein. Dies ist hier jedoch nicht der Fall, so dass offensichtlich Fehler aufgetreten sind:

```
1101110111010
1011
----
 1101
 1011
 ----
  1101
  1011
  ----
   1100
   1011
   ----
    1111
    1011
    ----
     1001
     1011
     ----
       1010
       1011
       ----
          110
          ===
```

c)

```
let append_zeroes data n =
  let rec append_zeroes_emb n = match n with
    | 0 -> []
    | _ -> 0 :: (append_zeroes_emb (n - 1))
  in data @ (append_zeroes_emb n);;
```

d)

```
let rec xor_begin l1 l2 = match (l1,l2) with
    | (_,[]) -> l1
    | (hd1::tl1,hd2::tl2) when (hd1 == hd2) -> 0:: (
       xor_begin tl1 tl2)
    | (hd1::tl1,hd2::tl2) -> 1:: (xor_begin tl1 tl2)
    | _ -> failwith "error";;
```

e)

```
let crc data polynomial=
  let rec crc_emb word = match word with
    | _ when ((List.length word) < (List.length polynomial))
       -> word
    | 0::tl -> crc_emb tl
    | _ -> crc_emb (xor_begin word polynomial )
  in data @ (crc_emb (append_zeroes data (List.length
     polynomial - 1)));;
```

```
let frame = [1;1;0;1;0;1;1;0;1;1];;
let polynomial = [1;0;0;1;1];;
crc frame polynomial;;
- : int list = [1; 1; 0; 1; 0; 1; 1; 0; 1; 1; 1; 1; 1; 0]

let frame2 = [1;0;0;0;1;0;0;0;0;0;1;1;1;1;1];;
let polynomial2 = [1;0;0;0;1;0;0;0;0;0;0;1;0;0;0;0;1];;
crc frame2 polynomial2;;
- : int list = [1; 0; 0; 0; 1; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1;
    0; 0; 1; 0; 0; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0]
```

Aufgabe 11 LZW-Kompressionsalgorithmus

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

a) Gehen Sie die Schritte durch, die bei der Komprimierung der Zeichenkette `ABRAKADABRA-KADABRAKADABRA...` 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 $\{1 = A, 2 = B, \ldots, 26 = Z\}$.

prefix	*next-char*	*string-table*	*code-output*
""	'A'		
"A"	'B'	$27 = "AB"$	1
"B"	'R'	$28 = "BR"$	2
"R"	'A'	$29 = "RA"$	18
"A"	'K'	$30 = "AK"$	1
"K"	'A'	$31 = "KA"$	11
"A"	'D'	$32 = "AD"$	1
"D"	'A'	$33 = "DA"$	4
"A"	'B'		
"AB"	'R'	$34 = "ABR"$	27
"R"	'A'		
"RA"	'K'	$35 = "RAK"$	29
"K"	'A'		
"KA"	'D'	$36 = "KAD"$	31
"D"	'A'		
"DA"	'B'	$37 = "DAB"$	33
"B"	'R'		
"BR"	'A'	$38 = "BRA"$	28
"A"	'K'		
⋮	⋮	⋮	⋮

b) Gehen Sie die Schritte durch, die bei der Dekomprimierung von `[8.15.11.21.19.16.28.30.27.29.31.33.19.35.30.32.36.]` 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> = <code>;
         )
         (no: [...] <- translation for <old>;
            K <- first character of [...];
            output [...]^K to charstream and add it to string table;
            <old> = <code>
         )
end
```

Die Auswertung für `[8.15.11.21.19.16.28.30.27.29.31.33.19.35.30.32.36.]` läuft dann wie folgt ab:
Initialisierung des *string-table* mit $\{1 = A, 2 = B, \ldots, 26 = Z\}$.

next-code	*output*	*string-table*
8	"H"	
15	"O"	27 = "HO"
11	"K"	28 = "OK"
21	"U"	29 = "KU"
19	"S"	30 = "US"
16	"P"	31 = "SP"
28	"OK"	32 = "PO"
30	"US"	33 = "OKU"
27	"HO"	34 = "USH"
29	"KU"	35 = "HOK"
31	"SP"	36 = "KUS"
33	"OKU"	37 = "SPO"
19	"S"	38 = "OKUS"
35	"HOK"	39 = "SH"
30	"US"	40 = "HOKU"
32	"PO"	41 = "USP"
36	"KUS"	42 = "POK"

c) **Implementierung des LZW Algorithmus in Java**

```
import java.lang.String;
import java.util.Hashtable;

class Lzw
{
  public static void main(String args[]) {
    if (args.length <= 1) {
      System.out.println("Aufruf: 'Lzw -c|-u Zeichenkette'");
      return;
    }
    if (args[0].compareTo("-c") == 0) {
      compress(args[1]);
    }
    if (args[0].compareTo("-u") == 0) {
      uncompress(args[1]);
    }
  }


  private static void compress(String comp) {
    Hashtable patternList = new Hashtable();
    initCodeHash(patternList);
    int hashCount = 27;

    String checkString = new String();
    String prefix = new String();
    String outString = new String();
    for (int i = 0; i < comp.length(); i++) {
      checkString = prefix.concat(comp.substring(i, i+1));
      System.out.print("\"" + prefix + "\"     '" + comp.substring(i, i+1) + "'");
```

```
      if (patternList.get(checkString) != null) {
        System.out.println();
        prefix = checkString;
      }
      else {
        System.out.println("      " + hashCount + "=\"" + checkString
          + "\"     " + patternList.get(prefix));
        patternList.put(checkString, new Integer(hashCount));
        hashCount++;
        outString = outString.concat(patternList.get(prefix) + ".");
        prefix = comp.substring(i, i+1);
      }
    }
    if (prefix.length() != 0)
      outString = outString.concat(patternList.get(prefix) + ".");
    System.out.println();
    System.out.println("Komprimierte Sequenz:");
    System.out.println(outString);
  }

  private static void uncompress(String comp) {
    Hashtable patternList = new Hashtable();
    initStringHash(patternList);
    int hashCount = 27;
    int loopCount = 0;
    int old = -1;
    String outString = new String();

    while(comp.length() > 0) {
      int index = 0;
      int code = -1;
      boolean found = false;
      while (!found) {
        if (comp.charAt(index) == '.') {
          code = Integer.parseInt(comp.substring(0, index));
          comp = comp.substring(index+1, comp.length());
          found = true;
        }
        index++;
      }
      if (loopCount == 0) {
        outString = outString.concat((String)(patternList.get(new Integer(code))));
        old = code;
        System.out.println(code + "     \""
          + (String)(patternList.get(new Integer(code))) + "\"");
        loopCount++;
      }
      else {
        if (patternList.get(new Integer(code)) != null) {
          String translation = (String)(patternList.get(new Integer(code)));
          outString = outString.concat(translation);
          patternList.put(new Integer(hashCount),
((String)(patternList.get(new Integer(old)))).concat(translation.substring(0, 1)));
          System.out.println(code + "     \"" + translation + "\"       "
```

```
            + hashCount + "=\"" +
((String)(patternList.get(new Integer(old)))).concat(translation.substring(0, 1)) + "\"");
          }
          else {
            String translation = (String)(patternList.get(new Integer(old)));
            translation = translation.concat(translation.substring(0, 1));
            outString = outString.concat(translation);
            patternList.put(new Integer(hashCount), translation);
            System.out.println(code + "     \"" + translation + "\"       "
              + hashCount + "=\"" + translation + "\"");
          }
          hashCount++;
          old = code;
        }
      }
      System.out.println(outString);
    }

    private static void initCodeHash(Hashtable pl) {
        pl.put("A", new Integer(1));
        pl.put("B", new Integer(2));
        pl.put("C", new Integer(3));
        pl.put("D", new Integer(4));
        pl.put("E", new Integer(5));
        pl.put("F", new Integer(6));
        pl.put("G", new Integer(7));
        pl.put("H", new Integer(8));
        pl.put("I", new Integer(9));
        pl.put("J", new Integer(10));
        pl.put("K", new Integer(11));
        pl.put("L", new Integer(12));
        pl.put("M", new Integer(13));
        pl.put("N", new Integer(14));
        pl.put("O", new Integer(15));
        pl.put("P", new Integer(16));
        pl.put("Q", new Integer(17));
        pl.put("R", new Integer(18));
        pl.put("S", new Integer(19));
        pl.put("T", new Integer(20));
        pl.put("U", new Integer(21));
        pl.put("V", new Integer(22));
        pl.put("W", new Integer(23));
        pl.put("X", new Integer(24));
        pl.put("Y", new Integer(25));
        pl.put("Z", new Integer(26));
    }

    private static void initStringHash(Hashtable pl) {
        pl.put(new Integer(1), "A");
        pl.put(new Integer(2), "B");
        pl.put(new Integer(3), "C");
        pl.put(new Integer(4), "D");
        pl.put(new Integer(5), "E");
        pl.put(new Integer(6), "F");
```

```
        pl.put(new Integer(7), "G");
        pl.put(new Integer(8), "H");
        pl.put(new Integer(9), "I");
        pl.put(new Integer(10), "J");
        pl.put(new Integer(11), "K");
        pl.put(new Integer(12), "L");
        pl.put(new Integer(13), "M");
        pl.put(new Integer(14), "N");
        pl.put(new Integer(15), "O");
        pl.put(new Integer(16), "P");
        pl.put(new Integer(17), "Q");
        pl.put(new Integer(18), "R");
        pl.put(new Integer(19), "S");
        pl.put(new Integer(20), "T");
        pl.put(new Integer(21), "U");
        pl.put(new Integer(22), "V");
        pl.put(new Integer(23), "W");
        pl.put(new Integer(24), "X");
        pl.put(new Integer(25), "Y");
        pl.put(new Integer(26), "Z");
    }
}
```