Grundkurs Codierung

Lösungsvorschläge zu den Fragen in den Unterkapiteln „Was blieb?“
Stand 08.05.2007

Unterkapitel 6.1.5, Seite 336

Zu Frage 1:

Das RLE-Verfahren ist ausschließlich für die Kompression von Datenströmen geeignet, bei denen oft Folgen gleicher Daten (z. B. gleicher Bytes) auftreten. Bei Texten der natürlichen Sprachen liegen solche Verhältnisse praktisch nicht vor, wohl aber z. B. bei Bilddaten mit ihrer Beschreibung gleichartiger Flächen (z. B. rote Fläche = gleiche Byte-Zuordnung), oder auch anderen, nicht sprachlichen Datenmengen.

Manchmal lassen sich durch systematische Anordnung der Daten Folgen gleicher Bytes erzwingen, wie etwa beim Sägezahnverfahren (= circle zag) auf das Ergebnis einer zweidimensionalen DCT, siehe Unterkapitel 6.2.3, Seite 360.

Bei “Zippern” wird vor Einsatz der eigentlichen Kompressionsverfahren zunächst das Datenmaterial nach verschiedenen Gesichtspunkten analysiert und anhand des Ergebnisses ein optimales Verfahren oder eine Folge mehrerer Verfahren eingesetzt.

Zu Frage 2:

Da das Huffman-Verfahren die Struktur eines Binärbaums verwendet, ist das Zuordnungsergebnis der Zeichen zu Bit-Folgen dann optimal, wenn die Zeichenhäufigkeiten Potenzen von ½ sind.

Zu Frage 3:

Falls die Häufigkeiten der Zeichen- oder Bitmuster-Verteilung keine Potenzen von ½ darstellen, liefert die arithmetische Codierung optimale Ergebnisse. Allerdings ist der Aufwand höher und man muss entscheiden, ob das schnellere Huffman-Verfahren mit seinem suboptimalen Ergebnis für den vorgesehenen Zweck nicht bereits ausreicht.

Zu Frage 4:

Liegt eine Menge von Zeichen x_i vor (z. B. Schriftzeichen, Farbcodes, Audio-Files von Silben einer natürlichen Sprache, Pixel-Werte eines Bildes, oder auch ganz abstrakte Bitmuster), so kann nach C. E. Shannon mit der Auftrittswahrscheinlichkeit $p(x_i)$ deren Informationsgehalt als Entropie $H(x_i)$ angegeben werden:

$$H(x_i) = - p(x_i)\ \mathrm{ld}\ p(x_i),$$

siehe Unterkapitel 1.3, Seite 13 und 14. Die Entropie ist einfach die Anzahl der Bits, mit der sich jedes Zeichen binär darstellen lässt und liefert damit zugleich eine normierte Basis für beliebige physikalische Erscheinungsbilder der Zeichen. Für deren Analyse und Weiterbehandlung hat dies fundamentale Bedeutung.

So kann man z. B. als **mittlere Entropie**

$$\bar{H} = \frac{-\sum_{i=1}^{n} p(x_i) \cdot \mathrm{ld}\, p(x_i)}{n}$$

die mittlere Anzahl der für ein Zeichen erforderlichen Bits bestimmen, wenn man die Summe aller Zeichen-

Entropien bildet und durch die Anzahl n teilt. Dies dient als Ausgangspunkt für die Kernfrage der Datenkompression, nämlich ob der Zeichensatz durch einen gleichwertigen anderen mit kleinerer mittlerer Entropie ersetzbar ist.

Da die Zeichen eines Zeichensatzes häufig nicht unabhängig von einander auftreten, also statistisch abhängig sein können, reicht die ausschließliche Betrachtung einzelner Zeichen zur vollständigen Bestimmung der mittleren Entropie allerdings nicht aus. Hier müssen nach dem gleichen Verfahren, wie oben angegeben, die Entropien von **Zeichenfolgen** statt von Einzelzeichen ermittelt werden. Der Aufwand steigt exponentiell zu n, das Ergebnis gestattet aber genauere Aussagen.

Man kann zunächst die mittlere Entropie aller n^2 Doppelzeichen betrachten, dann diejenige aller n^3 Dreifachzeichen usw. und hört auf, wenn sich der Wert nicht mehr nennenswert verringert (warum verringert?).

Eine weitere, für Aussagen zu Verfahren der Fehlerkorrektur nützliche und notwendige Variante ist die der **bedingten Entropie** $H_w(x)$, siehe Unterkapitel 1.3, Seite 14. Sie spielt dann eine Rolle, wenn von einer Quelle die Zeichen x_i gesendet werden, der Empfänger aber die von einem zufälligen Rauschen r_i überlagerte Folge $w_i = x_i + r_i$ erhält. Die bedingte Entropie $H_w(x)$ gibt dann die Ungewissheit darüber an, ob ein Empfangsbit w_i das richtige oder falsche Sendebit x_i darstellt.

Ein Beispiel hierzu nennt C. E. Shannon selbst: Eine Sender überträgt pro Sekunde 1000 Bits x_i , im Mittel wird 1 Bit pro 100 Bits durch die Störungen r_i gekippt, die Fehlerwahrscheinlichkeit beträgt also $p_r = 0.01$. Die Ungewissheit, ob richtig oder falsch, ist

$$H_w(x) = -p_r \text{ ld } p_r - (1-p_r) \text{ ld } (1-p_r) = 0.0664 + 0.0144 = 0.082.$$

Was steckt hinter dieser Zahl? Wenn der Sender 918 Info-Bits u_i pro Sekunde fehlerfrei übertragen will, muss er bei einer Geschwindigkeit von 1000 Bits pro Sekunde aus den 918 Info-Bits 82 geeignete Prüf-Bits y_i berechnen (die im Übrigen nichts zur Entropie von u beitragen) und zusammen mit den Info-Bits senden. Der Empfänger kann dann prinzipiell aus den 1000 fehlerhaften Empfangs-Bits

$$w_i = x_i + r_i = u_i + y_i, + r_i$$

die 918 Info-Bits vollständig rekonstruieren (Aussage des berühmten **Kanalkapazitätstheorems**, siehe Unterkapitel 3.5.6, Seite 82 ff). Allerdings gibt es keinen Hinweis darauf, wie man dies systematisch durchführen muss.

Etwas allgemeiner lässt sich dies ausdrücken, wenn man mit H(x) die mittlere Entropie der Sendefolge bezeichnet, wobei

$$H(x) = H(u) = H(u + y)$$

gilt, da die Prüf-Bits y_i keine über den Informationsgehalt der Nachricht u, aus der sie bestimmt werden, hinausgehende Information besitzen ***)**:

$$H(x) = H(w) - H_w(x).$$

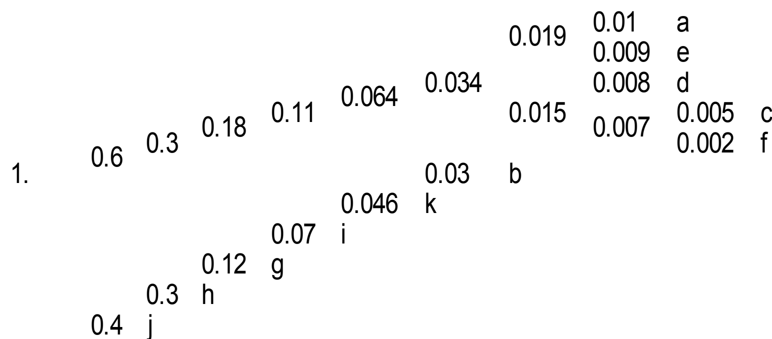
H(w) ist die mittlere Entropie der Empfangsfolge w_i, die gegenüber der Entropie der Sendefolge x_i durch die zufälligen Fehler r_i gerade um den Betrag der bedingten Entropie $H_w(x)$ vergrößerte wurde, siehe Unterkapitel 3.5.6, Seite 83.

***)** *Wer's nicht glaubt:* Durch die Prüfbits wird zwar die Anzahl der Bits pro Wort vergrößert, nicht aber die Häufigkeitsverteilung der Codewörter v gegenüber den Info-Wörtern. Daher wirkt sich das auch nicht auf die Entropie aus.

Zu Frage 5:

Zunächst ein **Fehlerhinweis!** Die Häufigkeit von e ist **0.009** (nicht, wie angegeben, 0.09), sonst wäre auch die Summe der Häufigkeiten nicht 1.0.

Der Binärbaum, von rechts nach links "aufsteigend" dargestellt, sieht so aus (das etwas ungewöhnliche Erscheinungsbild ist durch das erzeugende Programm bedingt):



Die beiden Zeichen mit der geringsten Wahrscheinlichkeit sind "f" (0.005) und "c" (0.002), die Summe ist 0.007. Da das nächste Zeichen "d" mit 0.008 einen größeren Wert als der Summenknoten von "f" und "c" hat, kommt "d" in die nächsthöhere (= rechte) Ebene und bildet zusammen mit 0.007 den übernächsten Knoten 0.015. "e" und "a" weisen kleinere Häufigkeitswerte als 0.015 auf und bleiben daher in der Ebene von "d" usw., siehe Unterkapitel 6.1.2, Seite 325 ff

Der Knoten in der ganz linken Ebene muss den Wert 1.0 zeigen. Der Binärbaum für die 11 Zeichen erzeugt den Huffman-Code

j =0
h =10
g =110
i =1110
k =11110
b =111110
e =11111111
a =11111111
d =11111101
c =111111001
f =111111000

Würden die Zeichen mit gleicher Bit-Zahl dargestellt, so benötigte man ld 11 = 3.45 Bits/Zeichen und müsste auf 4 Bits/Zeichen aufrunden. Der Huffman-Code beansprucht im Mittel 2.297 Bits, was sich aus

$$\text{mittlere Anzahl Bits/Zeichen} = \sum_{i=1}^{11} p\left(\text{Zeichen } i\right)\cdot\left(\text{Anzahl Bits des Huffman-Zeichens } i\right)$$

ergibt. Der Kompressionsgewinn beträgt also theoretisch 1.0 – (2.297/3.45) = 1.0 – 0.664 = 0.336, praktisch (wegen der notwendigen Aufrundung auf 4 Bits) 1.0 – (2.297/ 4.0) = 1.0 – 0.574 = 0.426, oder **42.6%**.

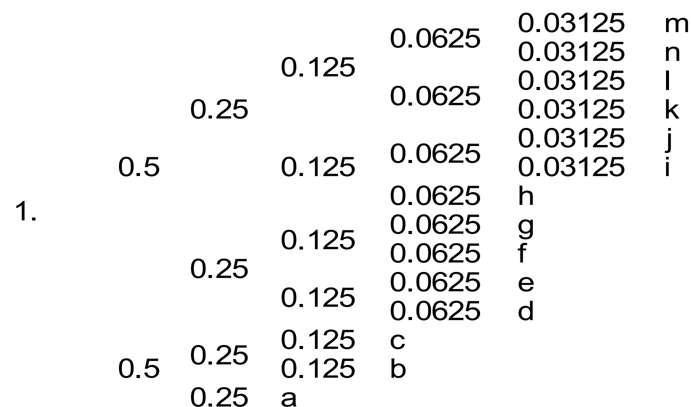
Frage 5 enthält auch noch die Sicht auf mögliche Varianten bei der Zuordnung der Zeichen zu den Ebenen des Binärbaums, bei der sich eventuell ein veränderter Huffman-Code ergeben kann → Bitte selbst überlegen oder ausprobieren, ob sich der Kompressionsgewinn dadurch steigern liesse. Ein optimales Ergebnis liefert der Huffman-Code wegen der nicht aus Potenzen von 0.5 bestehenden Zeichenhäufigkeiten aber

ohnehin nicht, dies wäre erst bei Einsatz des arithmetischen Codes gegeben, siehe Unterkapitel 6.1.4, Seite 333 ff.

Optimal ist das Ergebnis jedoch, wenn die Zeichenhäufigkeiten ausschließlich Potenzen von 0.5 darstellen, wie etwa bei

p(a)	= 0.25
p(b)	= 0.125
p(c)	= 0.125
p(d)	= 0.0625
p(e)	= 0.0625
p(f)	= 0.0625
p(g)	= 0.0625
p(h)	= 0.0625
p(i)	= 0.03125
p(j)	= 0.03125
p(k)	= 0.03125
p(l)	= 0.03125
p(m)	= 0.03125
p(n)	= 0.03125

Der Binärbaum hat hier die Form



und ergibt den Huffman-Code

a	=00
b	=010
c	=011
d	=1000
e	=1001
f	=1010
g	=1011
h	=1100
i	=11010
j	=11011
k	=11110
l	=11100
n	=11110
m	=11111

Der Kompressionsgewinn beträgt 1.0 – (2.795/5) = 1.0 – 0.559 = 0.441, also **44,1%**.

Zu Frage 6:

Bitte selbst durchführen, z. B. mit einem C-Programm. Hier als Starthilfe zwei Beispiele mit einem Textauszug aus **Walter Moers** “Die 13 ½ Leben des Käpt'n Blaubär” (köstlich, wärmstens zu empfehlen). Dieser Textauszug ist zur Vereinfachung - aber wenig elegant - direkt in die Programme eingebaut. Überhaupt lassen sich diese wesentlich verbessern, hier geht es aber zunächst nur um Aussagen über die Zeichenhäufigkeiten in einem Text. *Man kann eine Kopie der Beispiele unmittelbar als Quelltext in den Gnu-Compiler übernehmen* (“sollte laufen”).

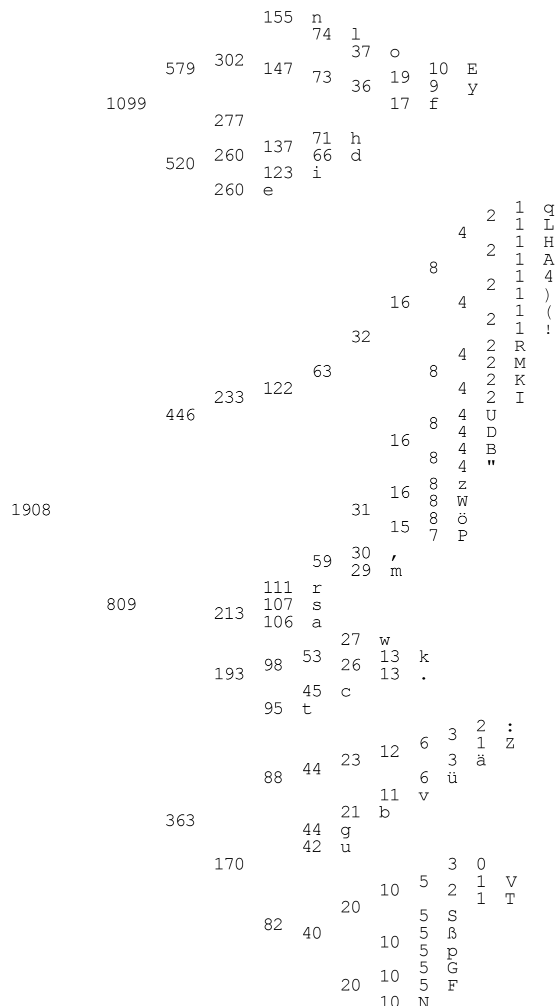
Programmbeispiel 1, Bestimmung der Häufigkeiten von Einzel-Zeichen:

```
// C-Programm zur Bestimmung der Häufigkeit von Zeichen im Text txt
// Entwicklungsumgebung Dev-C++  Version 4.9.9.2, frei ladbare GNU-Software
// Nur für Experimentierzwecke!
// Der Text txt kann durch beliebige andere Zeichenketten ersetzt werden.
// 08.05.2007
// Textbeispiel: // Walter Moers, "Die 13 1/2 Leben des Käpt'n Blaubär", Mein Leben in den Finsterbergen
// Hinweis: Die Funktionen strstr und strcat vertragen beim verwendeten Compiler
// als zweite Parameter keine Einzel-Character (diese haben nicht die geforderte Endekennung mit '\0').
// Daher muss der zweite Parameter als String cha[2] und der Vorbelegung cha[1] = '\0' deklariert sein,
// sonst "hängt" sich das System beim Compilieren auf.
// Im Programmbeispiel 2 mit mit den Doppel-Zeichen tritt das Problem nicht auf,
// da diese ohnehin als String deklariert sind.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main (void)
{
  int ia,i,ih, iha, ihb, ie, j, ja,je, k,ka, ke, sum, ntxt, nchc, sum_zeichen[500], f_hilf[500];

  char ch, cha[2]={' ','\0'}, chc[500], chc_hilf[500], *txt;

  unsigned char l1u;

txt="\"Wissen\", brüllte Professor Nachtigaller in den Klassenraum und riß dabei seine Augen auf, bis sie so groß wie Untertassen waren, \"Wissen ist Nacht!\" Das war ein Lehrsatz der Eydeetischen Philophysik, ein Fach, das nur an der Nachtschule gelehrt wurde. Professor Nachtigaller sagte öfter solche Sachen, wahrscheinlich um uns aus der Fassung zu bringen. Es steckte Methode in diesen scheinbar sinnlosen Behauptungen. Bevor man dahinterkam, dass sie völlig blöde waren, hatte man in alle möglichen Richtungen gedacht. Und das war genau das, was Professor Nachtigaller wollte: Wir sollten denken lernen, und zwar in möglichst viele verschiedenen Richtungen. In diesem Fall steckte allerdings eine gewisse Wahrheit in dem, was er gesagt hatte, denn Professor Nachtigaller war ein Eydeet. Eydeete sind die intelligentesten Wesen Zamoniens (und vermutlich der ganzen Welt, wenn nicht sogar des Universums): Bei normaler Beleuchtung haben sie einen Intelligenzquotienten von 4000, aber wenn es dunkel wird, steigert er sich ins Unvorstellbare. Daher halten sich Eydeete geren in möglichst finsteren Verhältnissen auf, und deshalb war Nachtigallers Nachtakademie in einem düsteren Höhlensystem in den Finsterbergen untergebracht. Professor Nachtigaller arbeitete in seiner Freizeit an einem System, Dunkelheit noch dunkler zu machen. Er hatte sich dafür eigens eine Dunkelkammer eingerichtet, die niemand außer ihm betreten durfte. Wir legten auch keinen großen Wert darauf, denn die Geräusche. die wir hörten, wenn wir gelegentlich an der Tür horchten, waren alles andere als einladend. Ein normaler Eydeet hat drei Gehirne, ein begabter vier, ein Eydeet mit Geniestatus fünf, Professor Nachtigaller hatte sieben. Eins davon befand sich im Kopf, vier wuchsen ihm aus der Schädelplatte, eins saß da, wo normalerweise die Milz ist, und wo das siebte Gehirn war, blieb ewiger Gegenstand der Spekulation seiner Schüler.";

  ntxt=strlen(txt);
```

```
 printf("\n%s\t\t\t%d\n","Anzahl der Zeichen im Text:",ntxt);

//Liste der verschiedenen Zeichen, Ergebnis ist die Zeichenkette chc
 strcpy(chc,"");
 for(i=0;i<ntxt;i++)
  {
  ch=txt[i];
  cha[0]=ch;
  if (!strstr(chc,cha)) strcat(chc,cha);
   }
 nchc=strlen(chc);
 printf("\n%s\n\n","Alle verschiedenen Zeichen im Text:");
 // Ausgabe-Anpassung der Umlaute usw.:
 for (i=0;i<nchc;i++)
 {
    l1u= chc[i];
    if(l1u==228) l1u=132; //ä
    if(l1u==246) l1u=148; //ö
    if(l1u==252) l1u=129; //ü
    if(l1u==196) l1u=142; //Ä
    if(l1u==214) l1u=153; //Ö
    if(l1u==220) l1u=154; //Ü
    if(l1u==223) l1u=225; //ß
    printf("%c", l1u);
 }
   printf("\n\n%s\t%d\n\n%s","Anzahl aller verschiedenen Zeichen im Text:",nchc,"Zur Ausgabe der un-
sortierten Zeichen: ");

 // Haltepunkt für Bildschirmausgabe:
 system("Pause");

 //Zählung der Häufigkeiten der Einzelzeichen:

 for (i=0;i<nchc;i++)
 {
 ch=chc[i];
 sum=0;
 for(j=0;j<ntxt;j++) if (ch==txt[j]) sum++;
 sum_zeichen[i]=sum;
 }
 //Ausgabe der Zeichen und ihre Häufigkeiten
 printf("\n%s\n","Unsortierte Liste der Häufigkeiten und Wahrscheinlichkeiten:");

 sum=0;
 for (i=0;i<nchc;i++)
 {
    l1u= chc[i];
    if(l1u==228) l1u=132; //ä
    if(l1u==246) l1u=148; //ö
    if(l1u==252) l1u=129; //ü
    if(l1u==196) l1u=142; //Ä
    if(l1u==214) l1u=153; //Ö
    if(l1u==220) l1u=154; //Ü
    if(l1u==223) l1u=225; //ß

    printf("\n%d\t%c\t%d\t%2.6f",i,l1u,sum_zeichen[i], ((float) sum_zeichen[i]/(float) ntxt));
    sum=sum+sum_zeichen[i];
```

```
}

printf("\n\n%s","Zur Kontrollausgabe der Zeichenanzahlen: ");

  system("Pause");

// Kontrollausgabe der Zeichenanzahl im Vergleich:

printf("\n\%s%d\%s%d\n","Anzahl der Textzeichen und Summe der Häufigkeiten: ",ntxt, " und ",sum);

printf("\n\n%s","Zur Ausgabe der sortierten Einzel-Zeichen: ");
system("Pause");

// Ausgabe der sortierten Liste der Einzel-Zeichen mit Häufigkeiten und Wahrscheinlichkeiten

 printf("\n\n%s\n","Sortierte Liste der Einzel-Zeichen nach fallenden Häufigkeiten:");

//Kopieren in Hilfsfelder:
for (i=0;i<nchc;i++) chc_hilf[i]= chc[i];
for (i=0;i<nchc;i++) f_hilf[i]= sum_zeichen[i];

// Sortieren

ia=0;
while (ia < nchc)
{
   iha=0;
   ih=f_hilf[ia];
   ch=chc_hilf[ia];
   for (j=ia+1;j<nchc;j++) if (ih<f_hilf[j])  iha=j;
   if (iha != 0)
   {
   for (k=ia;k<iha;k++) {f_hilf[k]=f_hilf[k+1]; chc_hilf[k]=chc_hilf[k+1];}
   f_hilf[iha]=ih;
   chc_hilf[iha]=ch;
   }  else ia=ia+1;
}

for (i=0;i<nchc;i++)
{
   l1u= chc_hilf[i];
   if(l1u==228) l1u=132; //ä
   if(l1u==246) l1u=148; //ö
   if(l1u==252) l1u=129; //ü
   if(l1u==196) l1u=142; //Ä
   if(l1u==214) l1u=153; //Ö
   if(l1u==220) l1u=154; //Ü
   if(l1u==223) l1u=225; //ß
   printf("\n%d\t%c\t%d\t%2.6f",i,l1u,f_hilf[i],((float) f_hilf[i]/(float) ntxt));
}

// Das Ergebnis kann z. B. zum Aufbau eines Huffman-Binärbaumes dienen

 printf("\n\n%s", "Zum Beenden: ");

system("PAUSE");
}
```

Als Huffman-Binärbaum erhält man:



Bei ASCII-Darstellung würde man 8 Bit pro Zeichen investieren. Huffman-komprimiert werden entsprechend obigem Baum im Mittel 4.45 Bit/Zeichen benötigt. Der Kompressionsgewinn beträgt 44.4 %.

Für den speziellen Umfang des Beispieltextes von 58 Zeichen braucht man bei gleicher Bitlänge allerdings nur 6 Bit/Zeichen (Vorsicht: Dies ist eine recht künstliche Annahme, da man mit den Partnern für die Übertragung diese Vorgaben vereinbaren muss, was den praktischen Gebrauch aufwändig machen würde). Der Kompressionsgewinn wäre hier 26%.

Programmbeispiel 2, Bestimmung der Häufigkeiten von Zeichenpaaren:

```
// C-Programm zur Bestimmung der Häufigkeit von Zeichen-Paaren (Doppel-Zeichen) im Text txt
// Entwicklungsumgebung Dev-C++  Version 4.9.9.2, frei ladbare GNU-Software
// Nur für Experimentierzwecke!
// Der Text txt kann durch beliebige andere Zeichenketten ersetzt werden.
// 08.05.2007
// Textbeispiel: // Walter Moers, "Die 13 1/2 Leben des Käpt'n Blaubär", Mein Leben in den Finsterbergen

#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

void main (void)
{
  int a, ia,i,ih, iha, j, k, sum, ntxt, nchc, nchcb, sum_zeichen[500], sum_zeichenb[500],f_hilf[500],fb_hilf[500];

  char b[2], ch, chcb[500], chc_hilf[500], chcb_hilf[500], *txt;

  unsigned char l1u, l2u;

txt="\"Wissen\", brüllte Professor Nachtigaller in den Klassenraum und riß dabei seine Augen auf, bis sie so groß wie Untertassen waren, \"Wissen ist Nacht!\" Das war ein Lehrsatz der Eydeetischen Philophysik, ein Fach, das nur an der Nachtschule gelehrt wurde. Professor Nachtigaller sagte öfter solche Sachen, wahrscheinlich um uns aus der Fassung zu bringen. Es steckte Methode in diesen scheinbar sinnlosen Behauptungen. Bevor man dahinterkam, dass sie völlig blöde waren, hatte man in alle möglichen Richtungen gedacht. Und das war genau das, was Professor Nachtigaller wollte: Wir sollten denken lernen, und zwar in möglichst viele verschiedenen Richtungen. In diesem Fall steckte allerdings eine gewisse Wahrheit in dem, was er gesagt hatte, denn Professor Nachtigaller war ein Eydeet. Eydeete sind die intelligentesten Wesen Zamoniens (und vermutlich der ganzen Welt, wenn nicht sogar des Universums): Bei normaler Beleuchtung haben sie einen Intelligenzquotienten von 4000, aber wenn es dunkel wird, steigert er sich ins Unvorstellbare. Daher halten sich Eydeete geren in möglichst finsteren Verhältnissen auf, und deshalb war Nachtigallers Nachtakademie in einem düsteren Höhlensystem in den Finsterbergen untergebracht. Professor Nachtigaller arbeitete in seiner Freizeit an einem System, Dunkelheit noch dunkler zu machen. Er hatte sich dafür eigens eine Dunkelkammer eingerichtet, die niemand außer ihm betreten durfte. Wir legten auch keinen großen Wert darauf, denn die Geräusche, die wir hörten, wenn wir gelegentlich an der Tür horchten, waren alles andere als einladend. Ein normaler Eydeet hat drei Gehirne, ein begabter vier, ein Eydeet mit Geniestatus fünf, Professor Nachtigaller hatte sieben. Eins davon befand sich im Kopf, vier wuchsen ihm aus der Schädelplatte, eins saß da, wo normalerweise die Milz ist, und wo das siebte Gehirn war, blieb ewiger Gegenstand der Spekulation seiner Schüler.";

  ntxt=strlen(txt);
  printf("\n%s\t\t%d\n","Anzahl der Zeichen im Text:",ntxt);

  //Zeichenkette chcb der verschiedenen Doppel-Zeichen

  chcb[0]=txt[0];
  chcb[1]=txt[1];

   for(i=0;i<ntxt/2;i++)
   {
   b[0]= txt[2*i];
   b[1]= txt[2*i+1];

   nchcb=strlen(chcb);
   a=0;
   for(j=0;j<nchcb/2;j++)
   if ((b[0]== chcb[2*j]) && (b[1]== chcb[2*j+1])) a=1;
   if (a==0) strcat(chcb,b);
    };
   nchcb=strlen(chcb)/2;

  printf("\n%s\n\n","Alle Doppelzeichen im Text:");
  // Ausgabe-Anpassung der Umlaute usw.:
  for (i=0;i<(2*nchcb);i++)
  {
     l1u= chcb[i];
```

```
    if(l1u==228) l1u=132; //ä
    if(l1u==246) l1u=148; //ö
    if(l1u==252) l1u=129; //ü
    if(l1u==196) l1u=142; //Ä
    if(l1u==214) l1u=153; //Ö
    if(l1u==220) l1u=154; //Ü
    if(l1u==223) l1u=225; //ß
    printf("%c", l1u);
  }

  printf("\n\n%s\t%d\n\n%s","Anzahl aller Doppelzeichen im Text:",nchcb,"Zur Ausgabe der unsortierten Doppel-Zeichen: ");

  // Haltepunkt für Bildschirmausgabe:
  system("Pause");

  // Zählung der Doppel-Zeichen:

  for (i=0;i<nchcb;i++)
  {
    sum=0;
    for(j=0;j<ntxt/2;j++) if ((chcb[2*i]==txt[2*j]) && (chcb[2*i+1]==txt[2*j+1])) sum++;
    sum_zeichenb[i]=sum;
  }
  sum=0;

  printf("\n%s\n","Unsortierte Liste der Häufigkeiten und Wahrscheinlichkeiten:");

  for (i=0;i<nchcb;i++)
  {
    l1u= chcb[2*i]; l2u=chcb[2*i+1];
    if(l1u==228) l1u=132; //ä
    if(l2u==228) l2u=132; //ä
    if(l1u==246) l1u=148; //ö
    if(l2u==246) l2u=148; //ö
    if(l1u==252) l1u=129; //ü
    if(l2u==252) l2u=129; //ü
    if(l1u==196) l1u=142; //Ä
    if(l2u==196) l2u=142; //Ä
    if(l1u==214) l1u=153; //Ö
    if(l2u==214) l2u=153; //Ö
    if(l1u==220) l1u=154; //Ü
    if(l2u==220) l2u=154; //Ü
    if(l1u==223) l1u=225; //ß
    if(l2u==223) l2u=225; //ß
    printf("\n%d\t%c%c\t%d\t%2.6f",i,l1u,l2u,sum_zeichenb[i],((float) sum_zeichenb[i]/(float) (ntxt/2)));
    sum=sum+2*sum_zeichenb[i];
  }

  printf("\n\n%s","Zur Kontrollausgabe der Zeichenanzahlen: ");
  // Kontrollpunkt für Bildschirmausgabe
  system("Pause");

  // Kontrollausgabe der Zeichenanzahl im Vergleich:

  printf("\n\%s%d\%s%d\n","Anzahl der Textzeichen und Summe der Häufigkeiten * 2: ",ntxt, " und ",sum);
```

```
 // Kopieren in Hilfsfelder:

 for (i=0;i<nchcb;i++) {chcb_hilf[2*i]= chcb[2*i]; chcb_hilf[2*i+1]= chcb[2*i+1];}
 for (i=0;i<nchcb;i++) {fb_hilf[2*i]= sum_zeichenb[2*i]; fb_hilf[2*i+1]= sum_zeichenb[2*i+1];}

 // Sortieren nach fallenden Häufigkeiten

 ia=0;
 while (ia < nchcb)
 {
    iha=0;
    ih=fb_hilf[ia];
    b[0]=chcb_hilf[2*ia]; b[1]=chcb_hilf[2*ia+1];
    for (j=ia+1;j<nchcb;j++) if (ih<fb_hilf[j])  iha=j;
    if (iha != 0)
    {
          for (k=ia;k<iha;k++) {fb_hilf[k]=fb_hilf[k+1]; chcb_hilf[2*k]=chcb_hilf[2*k+2];
chcb_hilf[2*k+1]=chcb_hilf[2*k+3];}
       fb_hilf[iha]=ih;
       chcb_hilf[2*iha]=b[0];
       chcb_hilf[2*iha+1]=b[1];
    }
    else ia=ia+1;
 }

 printf("\n\n%s","Zur Ausgabe der sortierten Doppelzeichen: ");
 system("Pause");
 // Ausgabe der sortierten Liste der Doppel-Zeichen mit Häufikeiten und Wahrscheinlichkeiten

 printf("\n\n%s\n","Sortierte Liste der Doppelzeichen nach fallenden Häufigkeiten:");

 for (i=0;i<nchcb;i++)
 {
    l1u= chcb_hilf[2*i]; l2u=chcb_hilf[2*i+1];
    if(l1u==228) l1u=132; //ä
    if(l2u==228) l2u=132; //ä
    if(l1u==246) l1u=148; //ö
    if(l2u==246) l2u=148; //ö
    if(l1u==252) l1u=129; //ü
    if(l2u==252) l2u=129; //ü
    if(l1u==196) l1u=142; //Ä
    if(l2u==196) l2u=142; //Ä
    if(l1u==214) l1u=153; //Ö
    if(l2u==214) l2u=153; //Ö
    if(l1u==220) l1u=154; //Ü
    if(l2u==220) l2u=154; //Ü
    if(l1u==223) l1u=225; //ß
    if(l2u==223) l2u=225; //ß
    printf("\n%d\t%c%c\t%d\t%2.6f",i,l1u,l2u,fb_hilf[i],((float) fb_hilf[i]/(float) (ntxt/2)));
 }

 // Das Ergebnis kann z. B. zum Aufbau eines Huffman-Binärbaumes dienen

 printf("\n\n%s", "Zum Beenden: ");
 system("PAUSE");
}
```

Der Huffman-Binärbaum ist hier schon so umfangreich, dass er nicht mehr übersichtlich dargestellt werden kann. Für die Huffman-komprimierten Doppelzeichen benötigt man 7.28 Bit. Der Kompressionsgewinn beträgt gegenüber der 16-Bit ASCII-Darstellung für Doppelzeichen 54.5%. Gegenüber der Einfach-Zeichen-Kompression wurde das Ergebnis also um zusätzliche 10% verbessert.

Dies lässt sich mit Dreifach-, Vierfach-, Fünffach-Zeichen usw. weiter steigern, wobei man folgende Entwicklung beobachtet:

- Wegen der exponentiell mit ihrer Länge wachsenden Zahl der Mehrfach-Zeichen steigt der Rechenaufwand beträchtlich.
- Um den Entropie-Vorteil wirklich ausschöpfen zu können, muss die Zahl der beteiligten Zeichen groß genug sein. Im obigen Beispiel hat der Text 1908 Zeichen. Hierin sind 58 verschiedene Einfach-Zeichen und 262 verschiedene Doppel-Zeichen (bei dann insgesamt 1908/2 = 954) enthalten. Wertet man die Dreifach-Zeichen aus, so sind es von insgesamt 1908/3 = 636 "nur" noch 421 Verschiedene, der Entropievorteil beträgt mit 52.8 % bereits weniger als bei Doppelzeichen. Eine Verbesserung wäre erst wieder mit längeren Texten zu erreichen.
- Der "Löwenanteil" ergibt sich bei geringstem Aufwand bereits durch Auswertung der Einfach-Zeichen.

Zu Frage 7:

Die mittleren Entropien des Originaltextes berechnen sich aus den Auftrittswahrscheinlichkeiten der Zeichen bzw. der Zeichenpaare. Da die mittleren Entropien die mittlere Zahl von Bits pro Zeichen angeben, ist die Frage bereits mit den Ergebnissen von Frage 6 beantwortet.

Zu Frage 8:

Das LZW-Verfahren findet prinzipiell alle sich wiederholenden Zeichenfolgen, jedoch bei Verarbeitung jedes weiteren Zeichens immer nur einmal pro Schritt. Bevor also z. B. eine Dreier-Folge erkannt werden kann, muss zunächst die führende Zweierfolge registriert worden sein.

Copyright 2006