2. Digitale Codierung und Übertragung

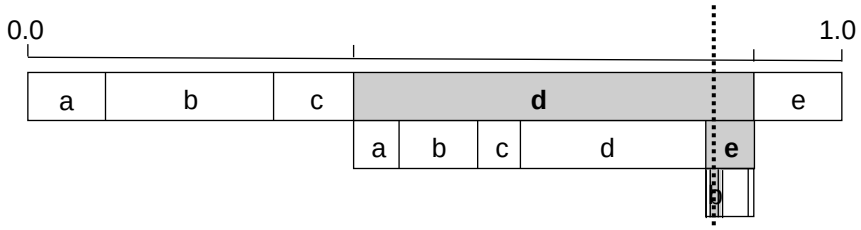
2.1 Informationstheoretische Grundlagen
2.2 Verlustfreie universelle Kompression
2.3 Digitalisierung, Digitale Medien

Kompressionsverfahren: Übersicht

- Klassifikationen:
  - Universell vs. speziell (für bestimmte Informationstypen)
  - Verlustfrei vs. verlustbehaftet
  - In diesem Kapitel: nur universelle & verlustfreie Verfahren
- Im folgenden vorgestellte Verfahren:
  - Statistische Verfahren:
    - » Huffman-Codierung
    - » Arithmetische Codierung
  - Zeichenorientierte Verfahren:
    - » Lauflängencodierung (RLE Run Length Encoding)
    - » LZW-Codierung

Arithmetische Codierung (1)

- Gegeben: Zeichenvorrat und Häufigkeitsverteilung
- Ziel: Bessere Eignung für Häufigkeiten, die keine Kehrwerte von Zweierpotenzen sind
- Patentiertes Verfahren; nur mit Lizenz verwendbar
- Grundidee:
  - Code = Gleitkommazahl berechnet aus den Zeichenhäufigkeiten
  - Jedes Eingabezeichen bestimmt ein Teilintervall



Arithmetische Codierung (2)

- Beispiel:

Zeichenindex i	1=Leerz.	2=I	3=M	4=S	5=W
Häufigkeit p_i	0.1	0.2	0.1	0.5	0.1
linker Rand L_i	0.0	0.1	0.3	0.4	0.9
rechter Rand R_i	0.1	0.3	0.4	0.9	1.0

Allgemein: $L_i = \sum_{j=1}^{i-1} p_j \quad R_i = \sum_{j=1}^{i} p_j$

- Algorithmus:

  real L = 0.0; **real** R = 1.0;
  Solange Zeichen vorhanden **wiederhole**
  Lies Zeichen und bestimme Zeichenindex i;
  real B = (R–L);
  R = L + B*R_i;
  L = L + B*L_i;
  Ende Wiederholung;
  Code des Textes ist Zahl im Intervall [L, R]

Algorithmus in "Pseudocode":

"**real**" Datentyp (Gleitkommazahl)

"=" Zuweisung an Variable

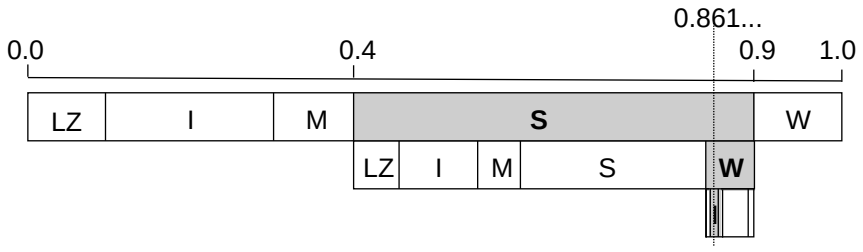
Arithmetische Codierung (3)

- Beispieltext-Codierung ("SWISS_MISS"):

Zeichen	L	R
S	0,4	0,9
W	0,85	0,9
I	0,855	0,865
S	0,859	0,864
S	0,861	0,8635
Leerz.	0,861	0,86125
M	0,861075	0,86110
I	0,8610775	0,8610782
S	0,86107778	0,86107813
S	0,86107792	0,861078095

Arithmetische Kodierung (4)

- Problem Gleitkomma-Arithmetik:
  - Konversion in Ganzzahl-Bereich durch "Skalieren"
- Welcher Binärcode:
  - Ober- und Untergrenze binär codieren
  - Code = Oberer Wert, abgebrochen nach der ersten Stelle, die verschieden vom unteren Wert ist
- Veranschaulichung:



Kompressionsverfahren: Übersicht

- Klassifikationen:
  - Universell vs. speziell (für bestimmte Informationstypen)
  - Verlustfrei vs. verlustbehaftet
  - In diesem Kapitel: nur universelle & verlustfreie Verfahren

- Im folgenden vorgestellte Verfahren:
  - Statistische Verfahren:
    - » Huffman-Codierung
    - » Arithmetische Codierung
  - Zeichenorientierte Verfahren:
    - » Lauflängencodierung (RLE Run Length Encoding)
    - » LZW-Codierung

Lauflängencodierung

- Unkomprimierte Repräsentationen von Information enthalten häufig Wiederholungen desselben Zeichens
  (z.B. lange Folgen von x00- oder xFF-Bytes)
- Idee: Ersetzen einer Folge gleicher Zeichen durch 1 Zeichen + Zähler
- Eingesetzt z.B. in Fax-Standards

- Beispiel:

  aaaabcdeeefgggghiabtttiikkkddde

  ersetzt durch

  #a4bcd#e3f#g4hiab#t3#i2#k3#d3e

- Probleme:
  - Bei geringer Häufigkeit von Wiederholungen ineffektiv (verschlechternd)
  - Syntaktische Trennung von Wiederholungsindikatoren und unverändertem Code

Kompressionsverfahren: Übersicht

- Klassifikationen:
  - Universell vs. speziell (für bestimmte Informationstypen)
  - Verlustfrei vs. verlustbehaftet
  - In diesem Kapitel: nur universelle & verlustfreie Verfahren

- Im folgenden vorgestellte Verfahren:
  - Statistische Verfahren:
    - » Huffman-Codierung
    - » Arithmetische Codierung
  - Zeichenorientierte Verfahren:
    - » Lauflängencodierung (RLE Run Length Encoding)
    - » LZW-Codierung 

Wörterbuch-Kompressionen

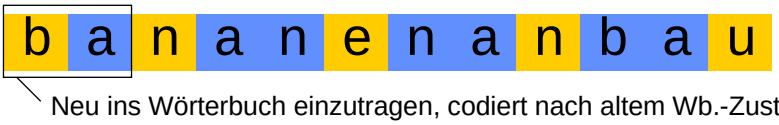
- Grundidee:
  - Suche nach dem "Vokabular" des Dokuments, d.h. nach sich wiederholenden Teilsequenzen
  - Erstelle Tabelle: Index --> Teilsequenz ("Wort")
  - Tabelle wird dynamisch während der Kodierung aufgebaut
  - Codiere Original als Folge von Indizes

- Praktische Algorithmen:
  - Abraham Lempel, Jacob Ziv (Israel), Ende 70er-Jahre
    - » LZ77- und LZ78-Algorithmen
  - Verbessert 1984 von A. Welch = "LZW"-Algorithmus (Lempel/Ziv/Welch)
  - Basis vieler semantikunabhängiger Kompressionsverfahren (z.B. UNIX "compress", Zip, gzip, V42.bis)
  - Verwendet in vielen Multimedia-Datenformaten (z.B. GIF)

Prinzip der LZW-Codierung

- Nicht alle Teilworte ins Wörterbuch, sondern nur eine "Kette" von Teilworten, die sich um je ein Zeichen überschneiden.
- Sequentieller Aufbau:
  Neu einzutragendes Teilwort = Kürzestes ("erstes") noch nicht eingetragenes Teilwort
- Beispiel:

- Codierung:



Neu ins Wörterbuch einzutragen, codiert nach altem Wb.-Zustand

LZW-Codierung (1)

- Tabelle mit Abbildung Zeichenreihe -> Indizes
  - Vorbesetzung der Tabelle mit fest vereinbarten Codes für Einzelzeichen (muß nicht explizit gespeichert und übertragen werden)
- Prinzipieller Ablauf:

SeqChar *p* = < NächstesEingabezeichen >;
Char *k* = NächstesEingabezeichen;
Wiederhole:
  Falls *p* & < *k* > in Tabelle enthalten
    dann *p* = *p* & < *k* >
    sonst trage *p* & <*k*> neu in Tabelle ein
      (und erzeuge neuen Index dafür);
      Schreibe Tabellenindex von *p* auf Ausgabe;
      p = < *k* >;
  Ende Fallunterscheidung;
  k = NächstesEingabezeichen;
solange bis Eingabeende
Schreibe Tabellenindex von *p* auf Ausgabe;

Algorithmus-Beschreibung (“Pseudo-Code”)

- Variablen (ähnlich zu C/Java-Syntax):
  - Datentyp fett geschrieben, gefolgt vom Namen der Variablen
  - Zuweisung an Variable mit “=“
- Datentypen:
  - **int**: Ganze Zahlen
  - **Char**: Zeichen (Buchstaben, Zahlen, Sonderzeichen)
  - **SeqChar**: Zeichenreihen (Sequenzen von Zeichen)
    - » Einelementige Zeichenreihe aus einem Zeichen: < x >
    - » Aneinanderreihung (Konkatenation) mit &
- NächstesEingabezeichen:
  - Liefert nächstes Zeichen der Eingabe und schaltet Leseposition im Eingabepuffer um ein Zeichen weiter

LZW-Codierung (2)

- Vorbesetzte Tabelle (z.B. mit ASCII-Codes):
  [(<a>, 97), (<b>, 98), (<c>, 99), (<d>, 100), (<e>, 101), (<f>, 102), (<g>, 103), (<h>, 104), (<i>, 105), (<j>, 106), (<k>, 107), (<l>, 108), (<m>, 109), (<n>, 110), (<o>, 111), (<p>, 112), (<q>, 113), (<r>, 114), (<s>, 115), (<t>, 116), (<u>, 117), (<v>, 118), (<w>, 119), (<x>, 120), (<y>, 121), (<z>, 122)]
- Für neue Einträge z.B. Nummern von 256 aufwärts verwendet.

LZW-Codierung (3)

- Beispieltext: “bananenanbau"
- Ablauf:

Lesen (k)	Codetabelle schreiben (p & <k>)	Ausgabe	Puffer füllen (p)
b			<b>
a	(<ba>, 256)	98	<a>
n	(<an>, 257)	97	<n>
a	(<na>, 258)	110	<a>
n			<an>
e	(<ane>, 259)	257	<e>
n	(<en>, 260)	101	<n>
a			<na>
n	(<nan>, 261)	258	<n>
b	(<nb>, 262)	110	<b>
a			<ba>
u	(<bau>, 263)	256	<u>
EOF		117	

Kompression durch LZW

- Am Beispiel:
  - 9 (16-Bit-)Worte statt 12 (16-Bit-)Worte, d.h. 25%
- In realen Situationen werden oft ca. 50% erreicht.
- Verfeinerungen des Algorithmus (z.B. Unix "compress"):
  - Obergrenze für Tabellengröße, dann statisch
  - Laufendes Beobachten der Kompressionsrate und ggf. Neustart

LZW-Decodierung bei bekannter Tabelle

Wiederhole solange Eingabe nicht leer:

k = NächsteEingabezahl;

Schreibe Zeichenreihe mit Tabellenindex *k* auf Ausgabe;

Ende Wiederholung;

LZW-Decodierung (1)

- Grundidee („symmetrische Codierung“):
  - Das aufgebaute Wörterbuch muß *nicht* zum Empfänger übertragen werden.
  - Das Wörterbuch wird nach dem gleichen Prinzip wie bei der Codierung bei der Decodierung dynamisch aufgebaut.
  - Das funktioniert, weil bei der Codierung immer *zuerst* der neue Eintrag für das Wörterbuch nach bekannten Regeln aus dem schon gelesenen Text aufgebaut wird, bevor der neue Eintrag in der Ausgabe verwendet wird.
- Algorithmusidee:
  - Neu einzutragendes Teilwort = letztes Teilwort plus erstes Zeichen des aktuellen Teilworts

LZW-Decodierung (2)

- Prinzipieller Algorithmus:

  SeqChar *p* := <>;
  int *k* = NächsteEingabezahl;
  Schreibe Zeichenreihe mit Tabellenindex *k* auf Ausgabe;
  int *old* = *k*;
  Wiederhole solange Eingabe nicht leer:
  k = NächsteEingabezahl;
  SeqChar *akt* = Zeichenreihe mit Tabellenindex *k*;
  Schreibe Zeichenreihe *akt* auf Ausgabe;
  p = Zeichenreihe mit Tabellenindex *old* (letztes Teilwort);
  Char *q* = erstes Zeichen von *akt*;
  Trage *p* & <*q*> in Tabelle ein
  (und erzeuge neuen Index dafür);
  old = *k*;
  Ende Wiederholung;

LZW-Decodierung (3)

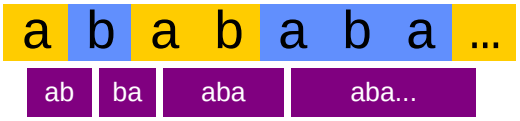
- Beispielzeichenreihe: “98-97-110-257-101-258-110-256-117"
- Ablauf:

Lesen (k)	Ausgabe (q ist jeweils unterstrichen)	Puffer füllen (p)	Codetabelle schreiben (p & <q>)	Merken (old)
98	b			98
97	**a**	b	(<ba>, 256)	97
110	**n**	a	(<an>, 257)	110
257	**a**n	n	(<na>, 258)	257
101	**e**	an	(<ane>, 259)	101
258	**n**a	e	(<en>, 260)	258
110	**n**	na	(<nan>, 261)	110
256	**b**a	n	(<nb>, 262)	256
117	**u**	ba	(<bau>, 263)	117
EOF				

LZW-Decodierung (4)

- Beispielzeichenreihe: “abababa...“, Beispielcode: “97-98-256-258“
- Ablauf:

Lesen (k)	Ausgabe (q ist jeweils unterstrichen)	Puffer füllen (p)	Codetabelle schreiben (p & <q>)	Merken (old)
97	a			97
98	**b**	a	(<ab>, 256)	98
256	**a**b	b	(<ba>, 257)	256
258	**???**			



Decodierung ist so noch nicht korrekt!

LZW-Decodierung, vollständige Fassung

SeqChar *p* := <>;
int *k* = NächsteEingabezahl;
Schreibe Zeichenreihe mit Tabellenindex *k* auf Ausgabe;
int *old* = *k*;
Wiederhole solange Eingabe nicht leer:
    k = NächsteEingabezahl;
    SeqChar *akt* = Zeichenreihe mit Tabellenindex *k*;
    p = Zeichenreihe mit Tabellenindex *old* (letztes Teilwort);
    Falls Index *k* in Tabelle enthalten
        dann **Char** *q* = erstes Zeichen von *akt*;
            Schreibe Zeichenreihe *akt* auf Ausgabe;
        sonst **Char** *q* = erstes Zeichen von *p*;
            Schreibe Zeichenreihe *p* & <*q*> auf Ausgabe;
    Ende Fallunterscheidung;
    Trage *p* & <*q*> in Tabelle ein
    (und erzeuge neuen Index dafür);
    old = *k*;
Ende Wiederholung;