Gliederung

1) Einleitung

Aufgabe der Datenkompression ist es, den technischen Aufwand bei der Übertragung oder Speicherung von Informationen möglichst klein zu halten, indem der Informationsfluss der Quellen mit einer möglichst geringen Bitrate codiert wird. Es wird dabei zwischen zwei Konzepten unterschieden:

Unter der Irrelevanz versteht man vom Empfänger der Nachricht nicht benötigte Informationen, wie beispielsweise in der herkömmlichen Telephonie die nicht übertragenen Spektralanteile über 3,4 kHz. Dabei geht streng genommen Information verloren. Man spricht von einer verlustbehafteten Codierung. Das ursprüngliche Sprachsignal kann nicht mehr rekonstruiert werden.

Mit der Redundanz bezeichnet man die im Signal „mehrfach“ vorhandene Information. Sie kann ohne Informationsverlust beseitigt werden, wie beispielsweise durch die Huffmann-Codierung oder auch die Lempel-Ziv-Codierung. Man spricht dann von einer verlustfreien Codierung.

2) Abraham Lempel, Jacob Ziv, Terry Welch

Die zwei israelischen Forscher Jacob Ziv und Abraham Lempel publizierten 1977 und 1978 neue Algorithmen zur verlustfreien Datenkompression. Dabei zeichnete Jacob Ziv für die Idee und die theoretischen Grundlagen und Abraham Lempel für die Programmstruktur verantwortlich.

Ein paar Jahre später, im Jahr 1983, entwickelte Terry Welch bei der Firma SPERRY eine schnellere Variante des Lempel-Ziv-Algorithmus,
die heute so genannte LZW (Lempel-Ziv-Welch)-Komprimierung war damit entstanden.

links: Jacob Ziv, rechts: Abraham Lempel

3) Tabellengestützte Modellierung

Der wichtigste Teil dieser Modelle ist die Tabelle und deren Verwaltung. Es werden verschiedene Strategien verfolgt, von denen die meisten jedoch auf die Arbeiten von Jacob Ziv und Abraham Lempel zurückgehen. Die erste, 1977 veröffentlicht, war bahnbrechend. Es wurde nicht mehr an einer Optimierung eines statistischen Modells gearbeitet, sondern die Idee war, Zeichenfolgen (Phrasen), die mehrmals im Text vorkommen, nur einmal zu speichern. Tritt eine Phrase ein zweites Mal auf, so wird nur noch ein Zeiger auf diese Phrase gespeichert. Ist der Speicherbedarf des Tokens kleiner, als der der Phrase, hat man eine Datenkompression erreicht, anderenfalls bläht man die Datei auf. Dies kann leicht passieren, wenn man Dateien, die schon komprimiert wurden, noch einmal komprimieren will. Um Phrasen zu referenzieren, wird im Allgemeinen eine Tabelle benutzt. In dem Verfahren, das in der 1. Veröffentlichung beschrieben wurde, hatte die Tabelle die Form eines Textfensters, das über die zu codierenden Daten gleitet, d.h. alte Phrasen, die aus dem Fenster bzw. der Tabelle geschoben werden, werden vergessen und können nicht mehr referenziert werden. In dem darauf folgenden Artikel beschrieben sie einen Mechanismus zur Verwaltung einer Tabelle, die unabhängig von einem Fenster ist. In dieser kann theoretisch jede Phrase, die im Text vorkommt, abgelegt sein. Auch die einfache Lauflängen- (Run-Length-) Codierung (RLC) kann man als tabellengestütztes Modell auffassen. In einem Token wird die Anzahl der Zeichen bzw. Wiederholungen und die Tabelle übergeben. Ein Offset ist nicht nötig, da die Tabelle nur ein Zeichen enthält.

4) LZ77

LZ77 ist die Abkürzung für das Komprimierungsverfahren, welches Jacob Ziv und Abraham Lempel 1977 in dem Artikel "A Universal Algorithm for Sequential Data Compression" in "IEEE Transactions on Information Theory“ vorgestellt haben

Der LZ77 Algorithmus war das erste vorgestellte tabellengesteuerte Kompressionsverfahren. Es komprimierte dadurch, dass zuvor eingelesener Text als Tabelle genutzt wird. Phrasen aus dem Eingabetext werden durch Zeiger in der Tabelle ersetzt. Dadurch wird die Komprimierung erreicht. Der Grad der Komprimierung hängt von der Länge der Phrasen, Fenstergröße und Entropie des Ursprungstextes (bezüglich LZ77) ab. Bei gleichen nah beieinander liegenden Textsequenzen kommt es sehr schnell zur Kompression.

Es werden neue Textteile so codiert, dass dazu möglichst lange, bereits vorher codierte Abschnitte in einem fortlaufend aktualisierten Speicher gesucht und als Zöpfe neuer Codewörter benutzt werden. Ein solches Codierverfahren ist universell, also von der Art und der Sprache des Textes unabhängig, und es nähert sich, wie Ziv in den 70er Jahren zeigen konnte, bei gleichförmigen Texten überraschend schnell dem idealen Ziel, nämlich der durch die "Entropie" gegebenen Grenze einer fehlerfreien Datenkompression minimaler Bitzahl. Schließlich lässt sich der ursprüngliche Text auch wieder auf ebenso einfache Weise zurückgewinnen.

Die Datenstruktur besteht aus einem Textfenster und einem nach vorn gerichteten Puffer. Während in dem Textfenster die schon kodierten Symbole stehen, zeigt der Puffer auf die als nächstes zu kodierenden Symbole. Sollte eine Sequenz von Symbolen in dem Puffer und dem Textfenster Übereinstimmen, so wird ein Kode bestehend aus Position und Länge im Textfenster gebildet und abgespeichert. Ansonsten wird der Kode so gespeichert wie er vorlag. Anschließend schieben sich beide Puffer eine Position nach vorn. deshalb wird diese Methode auch die Methode der gleitenden Fenster genannt.

search buffer	look-ahead buffer	Kodierung
	sir_sid_eastman_	(0,0,"s")
s	ir_sid_eastman_e	(0,0,"i")
si	r_sid_eastman_ea	(0,0,"r")
sir	_sid_eastman_eas	(0,0,"_")
sir_	sid_eastman_easi	(4,2,"d")
sir_sid	_eastman_easily_	(4,1,"e")
sir_sid_e	astman_easily_te	(0,0,"a")

Codierung:

- suche in der Tabelle eine möglichst lange Zeichenfolge, die mit den nächsten n zu codierenden Zeichen übereinstimmt
- bilde ein Token und speichere es
- verschiebe das Fenster um (n+1) Zeichen
- wiederhole, bis alle Zeichen codiert sind

Decodierung:

- lese ein Token
- kopiere von der Trefferposition in die 2. Fensterhälfte
- gebe das Zeichen in die 2. Fensterhälfte aus
- verschiebe das Textfenster um (n+1) Zeichen
- wiederhole, bis alle Token decodiert sind

Wird keine passende Phrase gefunden, so wird das Token "0, 0, Zeichen" ausgegeben. Das ist zwar nicht effektiv, aber es kann jede Zeichenfolge codiert werden. Wird nur selten eine Phrase gefunden, so kann die Datei sogar wachsen.

5) LZ78

Im September 1978 beschrieben Jacob Ziv und Abraham Lempel in ihrem Folgeartikel "Compression of Individual Sequences via Variable-Rate Coding" in "*IEEE Transactions on Information Theory*" ein weiteres tabellengestütztes Verfahren. Der LZ78-Algorithmus ist ein adaptives Verfahren. Am Anfang des Algorithmus ist die Phrasentabelle leer. Jedes mal wenn ein Token ausgegeben wird, wird die Phrasentabelle erweitert. Die neue Phrase setzt sich aus der alten plus dem übertragenen Zeichen zusammen. Die Token sind denen des LZ77 ähnlich, ihnen fehlt nur die Längenangabe der Phrase, denn diese ist in der Struktur der Phrasenliste enthalten.

Codierung:

- lese eine Zeichenfolge
- suche diese Zeichenfolge in der Phrasen-Tabelle
- bilde das Token und gebe es aus
- bilde die neue Phrase und füge sie in die Tabelle ein
- wiederhole, bis alle Zeichen codiert sind

Decodierung:

- lese ein Token
- gebe die referenzierte Zeichenfolge aus
- bilde die neue Phrase und füge sie in die Tabelle ein
- wiederhole, bis alle Token decodiert sind

Bsp.: Eingabetext: DAD DADA DADDY DADO

Ausgabephrase	Ausgabezeichen	Codierter String
0	D	D
0	A	A
1		D
1	A	DA
4		DA
4	D	DAD
1	Y	DY
0		
6	O	DADO

6) LZW

LZ78 wurde 1984 von Terry Welch in der Juni-Ausgabe von *IEEE Computer* in dem Artikel "A Technique for High-Performance Data Compression" verbessert, und nach seinem Erfinder LZW genannt. Auf Teile dieses Algorithmuses bekam das Sperry-Forschungszentrum (jetzt Teil von Unisys) ein US Patent.

LZW basiert sehr stark auf dem LZ78-Verfahren von Lempel und Ziv [LZ78]. Die Standardimplementierung ist ein "Wörterbuchverfahren" mit einem Wörterbuch von bis zu 4096 möglichen Einträgen und einer festen Codelänge von 12 bit. Das Wörterbuch wird wie bei allen LZ-Verfahren während des Kompressions- bzw. Dekompressionsvorgangs aufgebaut. Eine Ausnahme bilden die ersten 256 Einträge, die mit allen möglichen Einzel-Zeichen vorbelegt sind. Dadurch wird das übertragen von Einzel-Zeichen vermieden, da sie ja alle schon als Phrase vorliegen. Zur Laufzeit werden aus den Eingabedaten Muster gewonnen, die, sofern sie noch nicht bekannt sind, ins Wörterbuch eingetragen werden. Im anderen Fall wird nicht das Muster sondern sein Index im Wörterbuch ausgegeben.

Kodierung:

Eingabe	Erkanntes Muster	Neuer Wörterbucheintrag
LZWLZ78LZ77LZCLZMWLZAP	L	LZ (=256)
ZWLZ78LZ77LZCLZMWLZAP	Z	ZW (=257)
WLZ78LZ77LZCLZMWLZAP	W	WL (=258)
LZ78LZ77LZCLZMWLZAP	LZ	LZ7 (=259)
78LZ77LZCLZMWLZAP	7	78 (=260)
8LZ77LZCLZMWLZAP	8	8L (=261)
LZ77LZCLZMWLZAP	LZ7	LZ77 (=262)
7LZCLZMWLZAP	7	7L (=263)
LZCLZMWLZAP	LZ	LZC (=264)
CLZMWLZAP	C	CL (=265)
LZMWLZAP	LZ	LZM (=266)
MWLZAP	M	MW (=267)
WLZAP	WL	WLZ (=268)
ZAP	Z	ZA (=269)
AP	A	AP (=270)
P	P	

Ausgabe: L Z W <256> 78 <259> 7 <256> C <256> M <258> Z A P

Erklärung:

Jede Zeile im oberen Beispiel zeigt jeweils den Inhalt des Eingabestrings in jedem Schritt der Kodierung, das in diesem Schritt längste im Wörterbuch gefundene Zeichenmuster und neu hinzugekommene Wörterbucheinträge mit ihren Indexnummern.

Im ersten Schritt wird das längste gefundene Muster zwangsläufig nur ein einziger Buchstabe sein, da beim LZW-Verfahren das Wörterbuch mit allen Einzelzeichen mit den Codewerten von 0 bis 255 vorbelegt ist, aber keine längeren Zeichenketten enthält. In unserem Beispiel findet der Algorithmus also das 'L'. Im gleichen Schritt wird noch ein weiteres Zeichen der Eingabe betrachtet ('Z') und an das 'L' angehängt. Die so erzeugte Zeichenkette ist garantiert noch nicht im Wörterbuch enthalten, wird also unter dem Index 256 neu eingetragen. Im nächsten Schritt wird vom Eingabestrom die im letzen Schritt längste gefundene Zeichenkette (also das 'L') entfernt und ausgegeben. Das 'Z' wird nun zum ersten Zeichen des neuen Eingabestrings. Hier beginnt das gleiche Spiel von vorne. 'Z' ist das längste bekannte Muster. 'ZW' wird unter dem Index 257 ins Wörterbuch aufgenommen. 'Z' wird von der Eingabe entfernt, ausgegeben und ein neuer Durchlauf begonnen. 'W' ist nun längste im Wörterbuch eingetragene Zeichenkette, 'WL' wird mit Indexnummer 258 neu aufgenommen und das 'W' aus der Eingabe gestrichen und zur Ausgabe umgeleitet. Im nächsten Schritt passiert endlich etwas Interessanteres; das längste bekannte Muster ist nun eine Zeichenfolge, die wir selbst ins Wörterbuch eingetragen haben ('LZ' mit Indexnummer 256). Jetzt werden im nachfolgenden Schritt nicht zwei Einzelzeichen ausgegeben, sondern der Index des Musters aus dem Wörterbuch (256).

Dekodierung:

Eingabezeichen	C	Neuer Wörterbucheintrag	p
L			L
Z	Z	LZ (=256)	Z
W	W	ZW (=257)	W
<256>	L	WL (=258)	LZ
7	7	LZ7 (=259)	7
8	8	78 (=260)	8
<259>	L	8L (=261)	LZ7
7	7	LZ77 (=262)	7
<256>	L	7L (=263)	LZ
C	C	LZC (=264)	C
<256>	L	CL (=265)	LZ
M	M	LZM (=266)	M
<258>	W	MW (=267)	WL
Z	Z	WLZ (=268)	Z
A	A	ZA (=269)	A
P	P	AP (=270)	P

Ausgabestring: L Z W LZ 7 8 LZ7 7 LZ C LZ M WL W A P

Erklärung:

Bei der Dekodierung fangen wir ebenfalls mit einem schon vorinitialisierten Wörterbuch an. Wieder sind Einträge von 0 bis 255 vorhanden. In unserem Beispiel ist 'L' die längste bekannte Zeichenkette. Sie wird deshalb auch ausgegeben und in der Variablen P (wie "Präfix") festgehalten. Die nächste Eingabe ist ebenfalls ein bekanntes Zeichen. Dieses wird zunächst in der Variablen C gespeichert. Nachfolgend wird P mit C konkateniert und das Ergebnis ins Wörterbuch unter der Indexnummer 256 aufgenommen. Man beachte, daß 'LZ' sowohl im Dekompressor als auch im Kompressor an der gleichen Stelle im Wörterbuch vorkommt (256). Nachfolgend wird die letzte Eingabe in die Variable P kopiert und P ausgegeben. Der nachfolgende Schritt (mit 'W') funktioniert genauso. Im nächsten Schritt wird nur das erste Zeichen des Wörterbucheintrags nach C übernommen ('L'). Dann wird wieder P und C konkateniert und ein neuer Wörterbucheintrag erzeugt ('WL' mit Index 258). Jetzt wird P der Index 256 zugewiesen und der dazugehörige Wörterbucheintrag ausgegeben.

Problemfall K[omega]K

Bei dieser Eingabekombination kann es bei der Implementierung des Dekoders Probleme geben. Hierbei ist K ein beliebiges Zeichen, gefolgt von einer Zeichenkette [omega], wiederum gefolgt von K. Entscheidend ist hierbei, daß sich K[omega] bereits im Wörterbuch befindet. Die Kodierung selbst bereitet keine Probleme; diese ergeben sich erst im Dekoder. In diesem Fall hinkt sozusagen der Dekoder dem Enkoder hinterher. Der Enkoder schickt hier den Code für seinen letzten Wörterbucheintrag. Dieser Eintrag ist dem Dekoder jedoch zu dieser Zeit noch unbekannt.

Beispiel für den Fall K[omega]K :

APAPAPAPAP

Kodierung:

Eingabe	Erkanntes Muster	Neuer Wörterbucheintrag
APAPAPAPAPAP	A	AP (=256)
PAPAPAPAP	P	PA (=257)
APAPAPAP	AP	APA (=258)
APAPAP	APA	APAP (=259)
PAPAP	PA	PAP (=260)
PAP	PAP	

Ausgabe: A P <256> <258> <257> <260>

Dekodierung:

Eingabezeichen	C	Neuer Wörterbucheintrag	p
A			A
P	P	AP (=256)	P
<256>	A	PA (=257)	AP
<258>	???		

Trick: K[omega]K Fall ist aufgetreten, C ist erstes Zeichen der letzten Ausgabe (also 'A')

	A	APA (=258)	APA
<257>	P	APAP (=259)	PA
<260>	???		

gleiches Problem wie oben, d.h. C muss 'P' sein.

	P	PAP (=260)	PAP

Ausgabe: A P AP APA PA PAP

Terry A. Welch hat LZW mit dem Hintergedanken einer Hardwareimplementierung entwickelt. Für diesen Zweck muss der zugehörige Algorithmus einigen Voraussetzungen genügen. Zum einen muss er für den Benutzer und das Computersystem transparent sein. Zum anderen muss die Kompression und Dekompression sehr schnell sein. Für eine universelle Nutzung der Hardware soll die Komprimierung außerdem ungeachtet der im Datenstrom auftretenden Redundanzarten möglichst optimal sein. LZW erfüllt diese Voraussetzungen aufgrund seiner Struktur weitestgehend. Die Funktionsweise des Verfahrens soll nachfolgend anhand von Beispielen vorgestellt werden.

7) Einsatzgebiete

Die LZW-Codierung findet vorwiegend Verwendung im ARC-, GIF- und ZIP-Format.

LZW sind patentgeschützt und müssen von UNISYS lizensiert werden --> neue Produkte verzichten zunehmend auf LZW wegen der Patentlizenzen

GIF
Bei COMPUSERVE war man 1987 dabei (vor allem Bob Berry), ein komprimiertes Bildformat zu entwickeln, das auf unterschiedlichen Plattformen und damit in Netzwerken verwendbar sein sollte. Man nannte es "Graphics Interchange Format" (Austauschformat für Grafiken) und verwendete die LZW-Komprimierungsmethode. Das Format wurde als GIF87a bekannt und beliebt; es wurde um einige Fähigkeiten erweitert zum GIF89a. Inzwischen waren aber auch ein paar andere Dinge geschehen.

Der LZW-Algorithmus war zum Patent angemeldet worden, und zwar nicht nur vom Entwickler Welch, sondern auch von zwei IBM-Forschern. Als nächstes verschwand die Firma SPERRY in einer neuen Firma namens UNISYS, und das Patent ging mit. Bei COMPUSERVE wußte man angeblich von gar nichts und benutzte LZW für das neue Grafikformat. Sie ahnen, was passierte: es gab eine Reihe von Auseinandersetzungen. Aber UNISYS tolerierte im Allgemeinen die Verwendung von LZW bei Softwareprodukten - bis 1994. Ob mehr aufgrund finanzieller Probleme oder aufgrund der Verdienstmöglichkeiten durch das nun populäre Web, jedenfalls verlangt UNISYS seit 1995 Gebühren für die Verwendung des LZW-Verfahrens bei kommerziellen Software-Produkten. Für jede (käufliche) Software, die GIF-Dateien schreibt, sind seit der Zeit Gebühren zu entrichten. Ihre Bilddateien im GIF-Format dürfen Sie natürlich frei verteilen.

8) Zusätzliche Erläuterungen

ITU-Standard V42.bis:

Die „International Telecommunication Union“ (ITU) wurde am 17.5.1865 in Paris von 20 Staaten gegründet und ist seit dem 15.10.1947 eine Unterorganisation der (UN) mit Sitz in Genf

V-Serie: Datenübertragung über das Telefonnetz; bekannte Standards für Datenschnittstellen

V.42 ist ein Verfahren zur Fehlersicherung

V.42bis ist eine Datenkomprimierung nach Lempel-Ziv-Welch. V.42bis wird nur in Kombination mit dem LAP-M-Modus von V.42 eingesetzt

9) Quellen

Martin Werner: „Information und Codierung“, Vieweg Verlag, 2002

Kay_94 David C. Kay, John R. Levine: "Graphics File Formats", 2-te Ausgabe, Windcrest/McGraw-Hill 1994
S. 25-26

LZW Terry A. Welch: "A Technique for High Performance Data Compression", IEEE Computer vol. 17 no. 6, Juni 1984
S. 8-19

LZ77 J. Ziv, A. Lempel: "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, 1977,
S. 337-343

LZ78 J. Ziv, A. Lempel: "Compression of Individual Sequences Via Variable-Rate Coding", IEEE Transactions on Information Theory, 1978,
S. 530-536

COMP.FAQ "Frequently Asked Questions" aus der Newsgroup comp.compression
YABBA Daniel J. Bernstein, "Y Coding", Draft 4b, March 19, 1991 from the Yabba compression package.