Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering

Kurs Information und Codierung

Datenkompression: Lempel-Ziv (LZ) Codierung

Studiengang IT
21.08.2017

https://olat.zhaw.ch/...

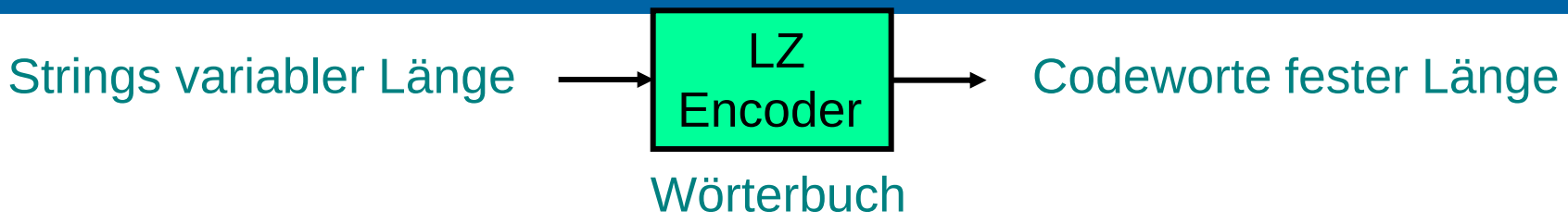
Autoren: Prof. Dr. Marcel Rupf, Kurt Hauser

Dozenten: Dr. Jürg Stettbacher, Kurt Hauser

Lernziele

- **Die Studierenden kennen die grundsätzliche Funktionsweise der Datenkompression nach Lempel-Ziv**
- **Sie kennen das Sliding-Window-Verfahren nach LZ77 und können hierfür Beispiele lösen**
- **Sie kennen die Wörterbuchverfahren nach LZ78 und LZW und können hierfür Beispiele lösen**
- **Sie können für das Verfahren nach LZ78 den Wörterbuch-Baum erstellen.**

Lempel-Ziv-Codierung



- **Parser unterteilt die Symbolfolge eindeutig in Strings variabler Länge, die sich nur in einem Bit unterscheiden**
- **Encoding eines String: [Position des Präfix-Strings, neues Bit]**

neuer String => [0000 0] **neuer String => [0011 0]**

- **Beispiel: 0‘ 1‘ 00‘ 001‘10‘ 000‘ 101‘ 0000‘ 01‘ 010‘ …**

String gleich wie 3. String

Vorteile und Nachteile

\+ universell bzw. nicht von der Quellenstatistik abhängig
\+ asymptotisch optimal, d.h. R -> H(X) (von oben)
\- Anzahl Strings bzw. Grösse des Wörterbuchs ist beschränkt

Datenkompression mit Wörterbuch

Referenz

D. Salomon, „*Data Compression: The Complete Reference*“,
3rd Edition, Springer-Verlag, 2004

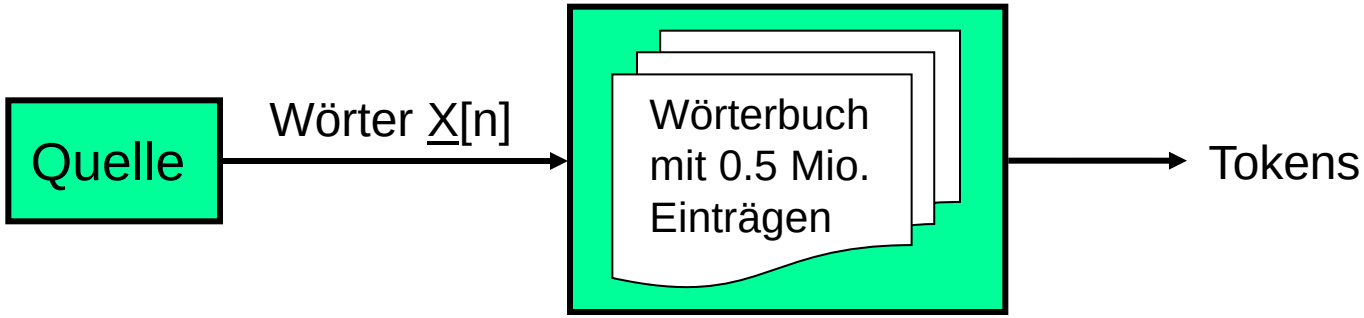
Statistische Kompressionsmethoden

basieren auf statistischem Datenmodell (siehe z.B. Huffman)
Qualität hängt davon ab, wie gut das Modell ist

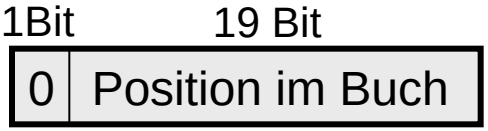
Wörterbuch-basierende Kompressionsmethoden

encodieren Symbolstrings mit Wörterbuch-Referenzen fester Länge
Wörterbuch ist statisch oder dynamisch (zu bevorzugen)
sind Entropie-optimal, wenn grosse Files komprimiert werden
im Prinzip bessere Kompression als mit statistischen Methoden
normalerweise sind Dekoder einfacher als Encoder
J. Ziv und A. Lempel entwickelten LZ77 und LZ78 Methoden
=> grosse Variantenvielfalt, breit eingesetzt, vor allem auch LZW

Beispiel mit statischem Wörterbuch



Wort gefunden



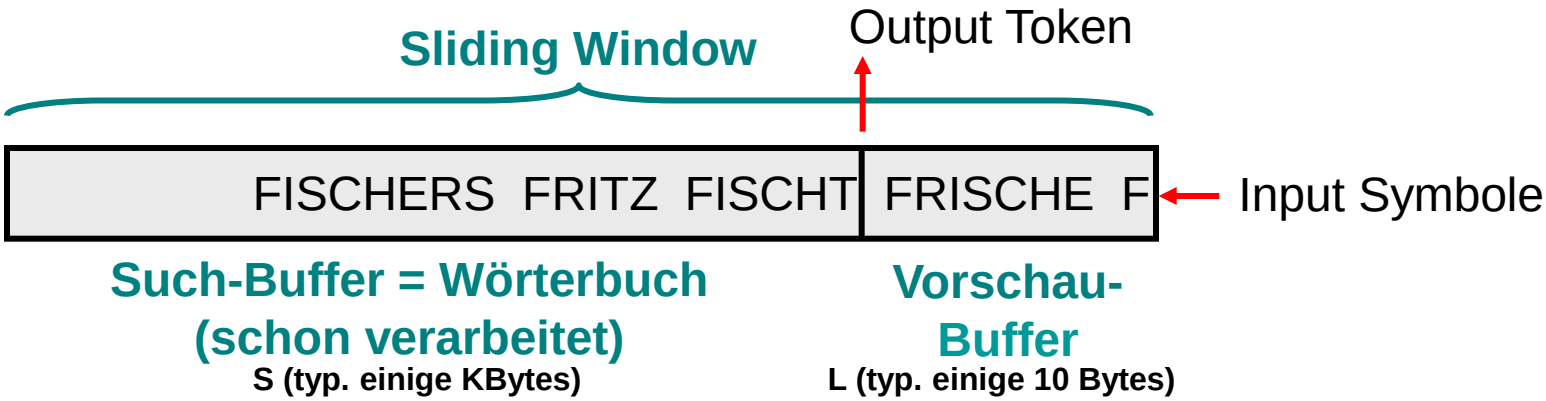
Wort nicht gefunden

Frage: Wie oft muss ein Wort im Wörterbuch sein, damit Rate R<1?

- P sei die Wahrscheinlichkeit, dass ein Wort im Buch gefunden wird
- Mittlere Wortgrösse: 5 Bytes bzw. 40 Bits

Input: N Wörter => Output: $N \cdot P \cdot 20$ Bits + $N \cdot (1-P) \cdot 48$ Bits,
=> Rate $R = (48-28 \cdot P)/40 < 1$ wenn $P > 0.29$, z.B. $R=0.57$ wenn $P=0.9$

LZ77 (Sliding Window)



1. Erstes Symbol des Vorschau-Buffers im Such-Buffer suchen
   rückwärts von rechts nach links, hier: Leerschlag
2. Token der längsten (letzten) Übereinstimmung ausgeben
   Token = (**Offset**, **Länge**, **nächstes Symbol**), hier: (13,4,“S“)
   Token-Länge: $[\log_2(S+1)] + [\log_2(L)] + 8$, typisch: 11 + 5 + 8 = 24 Bit
   wenn keine Übereinstimmung: (0,0,nächstes Symbol)
3. Fenster um Länge+1 nach rechts verschieben

FISCHERS FRITZ FISCHT FRISCHE FISCHE

LZ77 (mit Ausdehnung des Vergleichs)

Beispiel

	ANAN	=> (0,0,“A“)
A	NANA	=> (0,0,“N“)
AN	ANAS	=> (2,3,“S“) Vergleich auf Vorschau-Buffer ausgedehnt
ANANAS		=> fertig, da Vorschau-Buffer leer

Decoder ist viel einfacher als Encoder

- Buffer gleicher Grösse wie im Encoder erforderlich
- findet Übereinstimmung mit Offset und Länge (kein Suchen!)

LZ77 vergleicht Vorschau-Buffer mit benachbartem Input-Text

- Daten mit nahe beieinander liegenden „Mustern“ komprimieren gut
- Daten mit weit auseinander liegenden „Mustern“ komprimieren schlecht

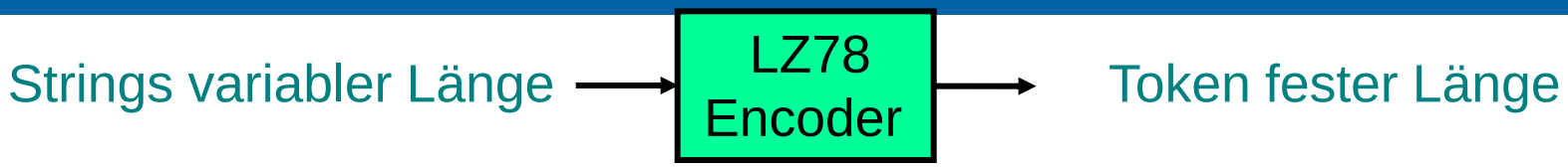
Bessere Kompression mit grösseren Buffern

- Vorschau-Buffer muss aber klein gehalten werden (Anzahl Vergleiche!)
- Such-Buffer darf auch nicht allzu gross sein (Suchzeit!)

Verwendung in embedded systems

- mit Huffman kombiniert im oft benutzten **Deflate**-Algorithmus
  - Zip, Gzip => HTTP, PPP, PNG, MNG, PDF
  - Kompressionsfaktoren 1/R = 2.5 … 3 für Text

LZ78



Wörterbuch, kein Buffer !

Wörterbuch enthält Symbol-Strings des bearbeiteten Inputs

ist am Anfang leer, Grösse eigentlich durch Memory beschränkt.
Parser unterteilt Symbolfolge in unterschiedliche Strings variabler Länge, die sich von Vorgängern nur in einem Symbol unterscheiden.

Output besteht aus 2-Feld-Token

(Pointer auf einen String im Wörterbuch, „neues“ Symbol)

Beispiel: F‘I‘S‘C‘H‘E‘R‘S ‘FR‘IT‘Z‘ FISCHT FRISCHE FISCHE

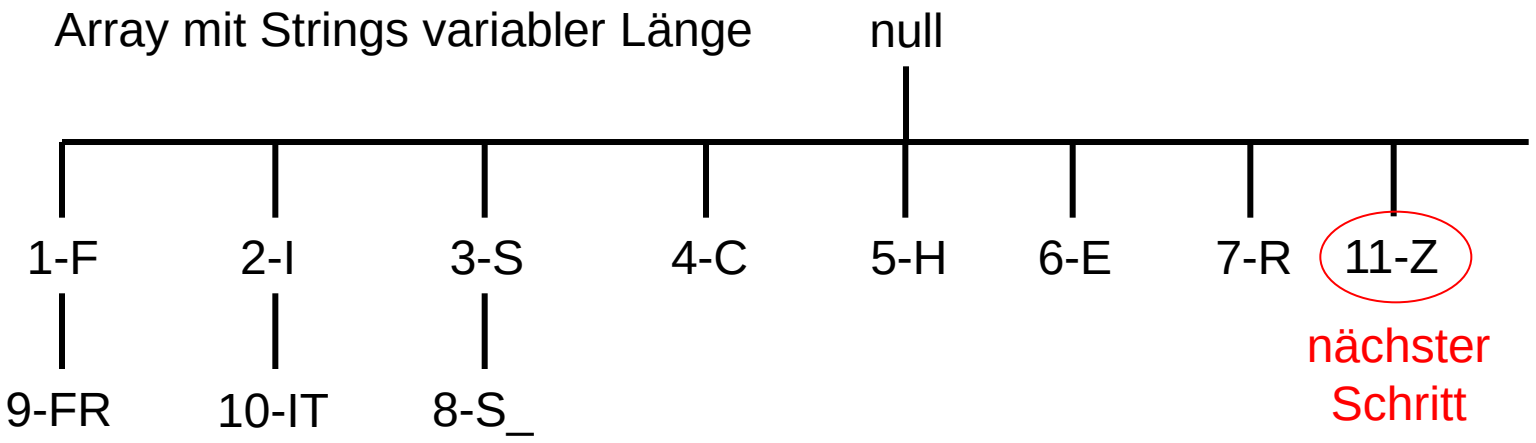
Wörterbuch		**Token**	**Wörterbuch**		**Token**
0	null		6	E	(0,“E“)
1	“F“	(0,“F“)	7	R	(0,“R“)
2	“I“	(0,“I“)	8	S_	(3,“_“)
3	“S“	(0,“S“)	9	FR	(1,“R“)
4	“C“	(0,“C“)	10	IT	(2,“T“)
5	“H“	(0,“H“)	…		

LZ78

grosses Wörterbuch erlaubt längere Übereinstimmungen

auf Kosten längerer Pointers und langsamerer Suche

Gute Datenstruktur für das Wörterbuch ist ein *Baum*



Massnahmen bei vollem Wörterbuch

einfrieren (Wörterbuch wird statisch)

löschen und neu anfangen, usw.

Dekoder

muss Wörterbuch bilden wie der Encoder, komplexer als LZ77-Dekoder

LZW (Lempel-Zip-Welch

Populäre Variante von LZ78 von T. Welch (1984)

Main Feature: Eliminierung des Symbol-Felds im Token

=> Wörterbuch mit Alphabet initialisieren, z.B. mit 256 8-Bit-ASCII-Zeichen

Algorithmus

0. Initialisierung I=[]
1. neues Symbol x zu String I hinzufügen => I = I x setzen
2. Ix im Wörterbuch verzeichnet? Wenn ja, dann zu step 1. sonst zu step 3.
3. a) Output = Wörterbuch-Pointer von I
   b) Neuer Wörterbucheintrag mit Phrase Ix
   c) I = “x“ setzen

Beispiel: ANANAS

I	verzeichnet	WB-Eintrag	Output
A	ja		
AN	nein	256: AN	65 (A)
N	ja		
NA	nein	257: NA	78 (N)
A	ja		
AN	ja		
ANA	nein	258: ANA	256 (AN)
A	ja		
AS	nein	259: AS	65 (A)
S	ja		
S,eof	nein		83 (S)

Wörterbuch-Baum

Liste bzw. Array mit Knoten mit je 2 Feldern (Symbol, Pointer auf Eltern)

Decoder

0. Wörterbuch initialisieren (normalerweise mit 256 Symbolen)
1. Pointer lesen und String I ausgeben
2. Pointer lesen, String J ausgeben und erstes Symbol x von J isolieren
3. Ix im Wörterbuch eintragen, und I=J setzen
4. falls es noch Pointers am Eingang gibt, dann step 2. sonst: Ende

Beispiel:

Input/Pointer	I	J = 'x...'	Wörterbucheintrag Ix
65	**A**		
78	A	**N**	256: AN
256	N	**AN**	257: NA
65	AN	**A**	258: ANA
83	A	**S**	259: AS

Anwendungen (lizenzpflichtig)

UNIX *compress*: Wörterbuch wächst langsam, zuerst 9 Bit Pointers, ...,
Grafik File Formate *GIF 89a* und *TIFF* verwenden LZW-Varianten.