Die LZ-Familie

Seminar Effiziente Programmierung

Hendrik Pfennig

10.01.2019

Inhaltsverzeichnis

Einführung Komprimierung

- Komprimierungsalgorithmen entweder verlustbehaftet oder verlustfrei
  - Behaftet: Bild-, Video-, Audiodaten
  - Frei: Ausführbare Programme, Text

- Gemessen wird:
  - Kompressionsrate
  - (De)Kompressionsgeschwindigkeit
  - Entropie
  - Kolmogorov-Komplexität

Entropie

- Maß für mittleren Informationsgehalt einer Quelle
- Definiert als $-\sum_i p_i * \log(p_i)$
- Entropiekodierung weist häufig auftretenden Zeichen kürzere Kodierungen zu
- Beispiele
  - Huffman Kodierung
  - Shannon-Fano Kodierung
  - Bereichskodierung (LZMA)

Kolmogorov-Komplexität

- Maß für Komprimierbarkeit einer Zeichenkette S
- Nicht berechenbar
- Kleinstes S erzeugendes Programm
- Kleiner oder gleich |S|
- Unkomprimierbar/zufällig gdw. |S| == Kolmogorov(S)
- Beispiel: Erzeugung von abababababababababab durch ab*10

Verlustbehaftete Komprimierung

- Algorithmus „entscheidet“ welche Teile des Datensatzes vernachlässigt werden können
- Komprimierte Datei oft für Menschen verständlich
- Wiederherstellung der Originaldatei nicht eindeutig möglich

- Beispiel Mp3
  - Für den Menschen nicht hörbare Signalanteile werden entfernt
  - Kaum merklicher Qualitätsverlust
  - Hohe Kompressionsrate (etwa bis 11:1)

Verlustbehaftete Komprimierung

Beispiel JPEG mit Kompressionsrate 13:1

Verlustfreie Komprimierung

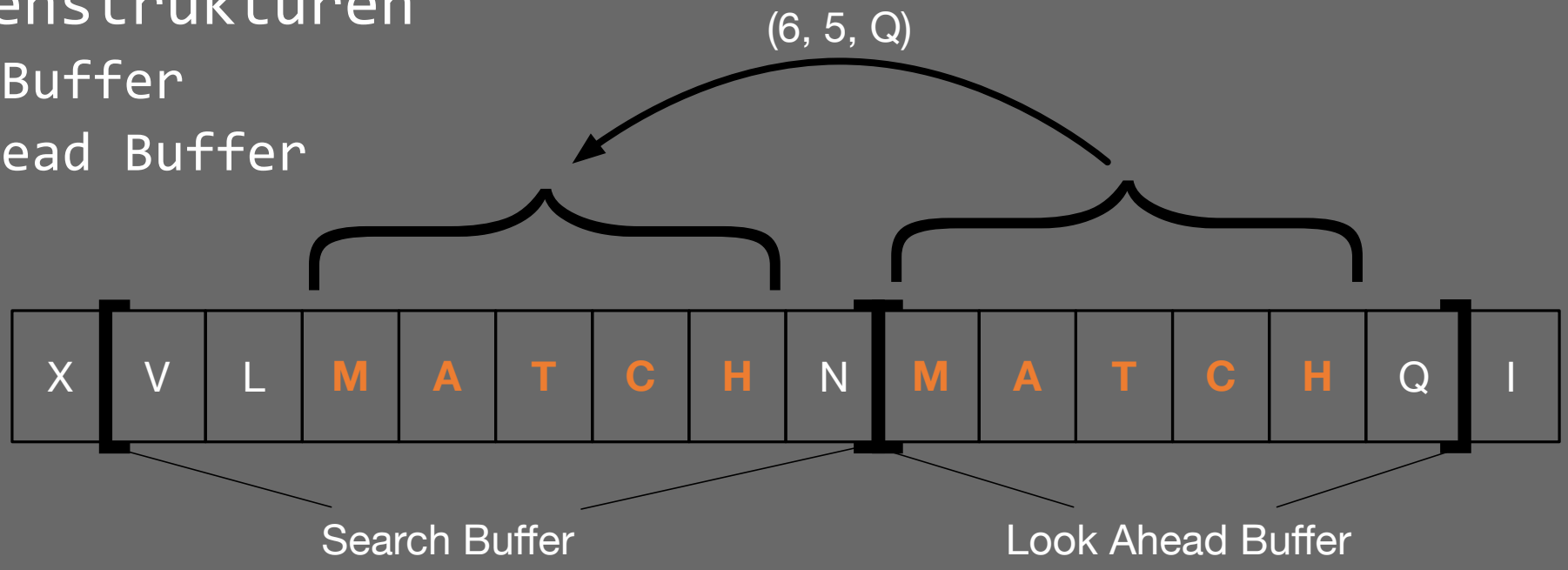
- Algorithmus findet und entfernt Redundanzen
- Komprimierte Datei für Menschen nicht lesbar
- Exakte Wiederherstellung der Originaldatei durch Dekomprimierung

- Bekannte Beispiele:
  - PNG
  - Deflate
  - LZ-Familie

Inhaltsverzeichnis

Lempel Ziv 1977 (LZ77)

- Abraham Lempel, Jacob Ziv
- Ausgabe von Zeigern bei Redundanz
- Zwei Datenstrukturen
  - Search Buffer
  - Look Ahead Buffer



Inhaltsverzeichnis

Lempel Ziv 1978 (LZ78)

- Datenstruktur: On-the-fly Wörterbuch
- Wörterbuch ist zu Beginn leer
- Redundanzen im Input-Stream werden durch Index ersetzt
- Ausgabe jeweils der Form (Index, Literal)
  - Bei Dekompression: Wörterbuch[Index] + Literal
- Neuer Eintrag (X, Wörterbuch[Index] + Literal)

Inhaltsverzeichnis

Lempel Ziv Welch (LZW)

- Umsetzung des LZ78 Konzepts durch Terry Welch
- Wörterbuch wird zunächst mit allen 1-Byte Wörtern initialisiert
- Keine Ausgabe von Literalen; nur Indizes
- Optionaler Clear Code bei vollem Wörterbuch
- Verwendung bei GIF, compress

```
1   input = "Inputstream"
2   d = initDict()
3   #d enthält alle Einzelzeichen des inputs
4   p = ""
5   while input != "":
6       z = nextChar(input)
7       if p+z in d.values():
8           p += z
9       else:
10          printKey(d, p)
11          d[len(d)] = p+z
12          p = z
13  printKey(d, p)
```

Inhaltsverzeichnis

Bereichskodierung

- Partitionierung von $[x, y] \in \mathbb{Z}$ in Teile
- Es gilt $|\text{Teile}| == |\Sigma|$
- Größe entspricht relativer Anzahl der Zeichen im Wort
- Für jedes Bit aus Eingabewort Bereich rekursiv auf Unterbereich abbilden
- Zur Dekomprimierung
  - Entweder len(input) oder EoM
  - Wahrscheinlichkeitsverteilung P
- Output: höherwertige Stellen eines $n \in [x_{neu}, y_{neu}]$

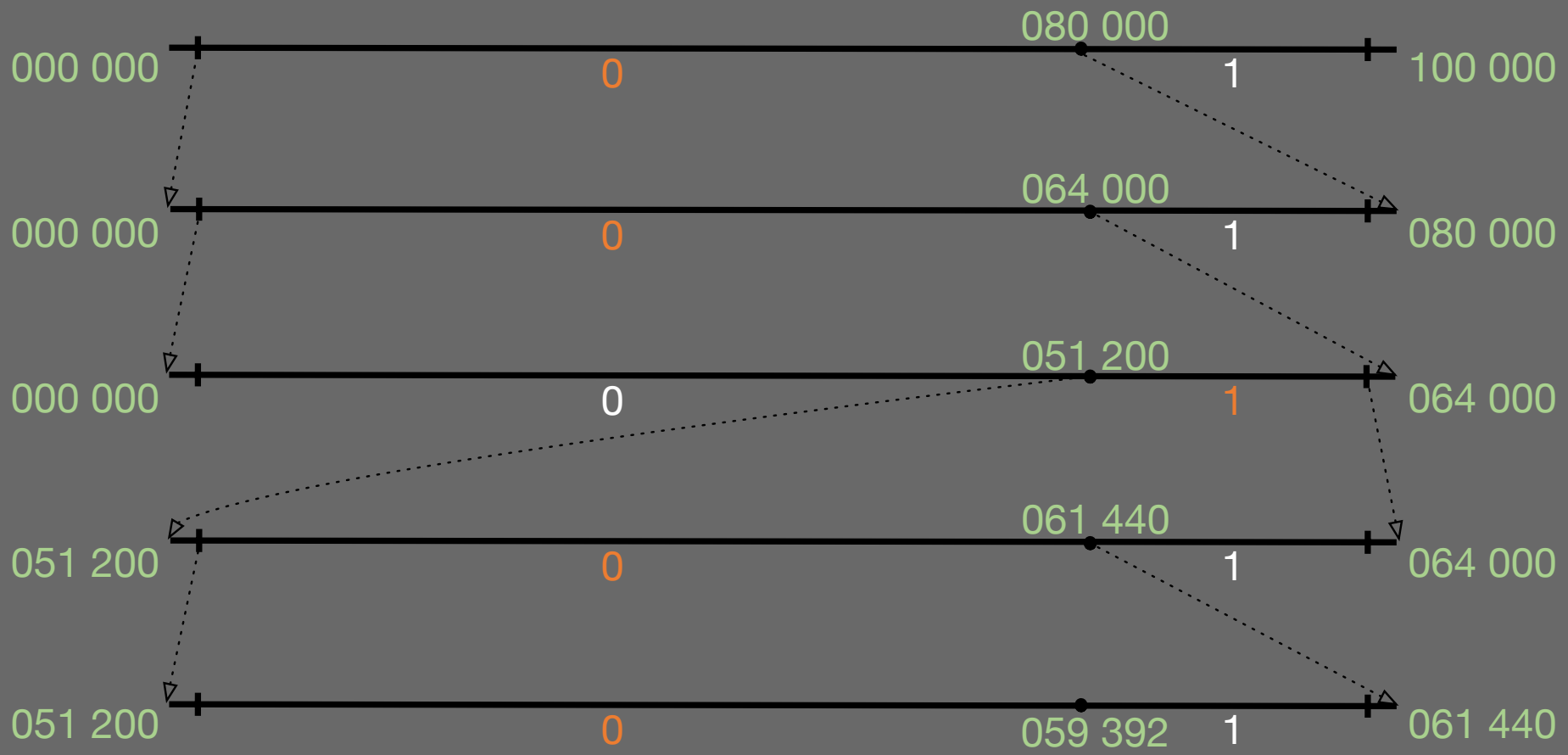
Bereichskodierung

```
range = [0, 100000]
probZero = 0.8
input = "00100"

while input != "":
    c = nextChar(input)
    x = range[0] + probZero * (range[1] - range[0])
    if c == "0":
        range = [range[0], x]
    else:
        range = [x, range[1]]
```

Bereichskodierung

Wort = „00100“
P_0 = 0.8
P_1 = 0.2



Output:
052

Inhaltsverzeichnis

LZ Markov Chain Algorithm (LZMA)

- Lempel-Ziv-Markov chain algorithm (von Igor Pavlov)
- Anwendung bspw. bei 7zip, tar
- LZ77 Konzept + Binärer Bereichskodierer
- Deutlich größeres Wörterbuch (bis zu 4 GB)
- Erfordert geeignete Suchstrukturen

LZ Markov Chain Algorithm (LZMA)

- Alle Bytes im Search Buffer werden durchlaufen
- Hashfunktion 2^{8w}, w = n Bytes, festes n ∈ {2,3,4}
- Hash bildet ab auf
  - Verkettete Liste
  - Binärbaum
- Kürzest zu kodierendes Match wird gewählt

LZ Markov Chain Algorithm (LZMA)

Output in sieben verschiedenen Formen

- Literal LIT = 0 + bytecode
- LZ77-Sequenzen MATCH = 10 + len + dist
- Schnellzugriff auf letztes Match SHORTREP = 1100
- Schnellzugriff auf letzte vier LZ77 Distanzen:
  - LONGREP[0]: 1101 + len
  - LONGREP[1]: 1110 + len
  - LONGREP[2]: 11110 + len
  - LONGREP[3]: 11111 + len

Inhaltsverzeichnis

Lempel Ziv 4

- Fast Compression vs High Compression
- Verwendet Hashtabellen zur Treffersuche
- Bei FC wird erster Treffer gesucht, bei HC bester
- Variante für beschränkten Speicher vorhanden

Inhaltsverzeichnis

Zusammenfassung

Algorithmus	Kompressions-geschwindigkeit	Dekompressions-geschwindigkeit	Kompressionsrate
LZ77	0,13 mb/s	0,5 mb/s	1,21:1
LZ78	0,006 mb/s	0,5 mb/s	1,88:1
LZW	2,4 mb/s	2,2 mb/s	1,42:1
LZMA	17 mb/s	45 mb/s	2,38:1
LZ4 (HC)	22 mb/s	2392 mb/s	1,79:1
LZ4 (FC)	449 mb/s	2176 mb/s	1,54:1

Quellen

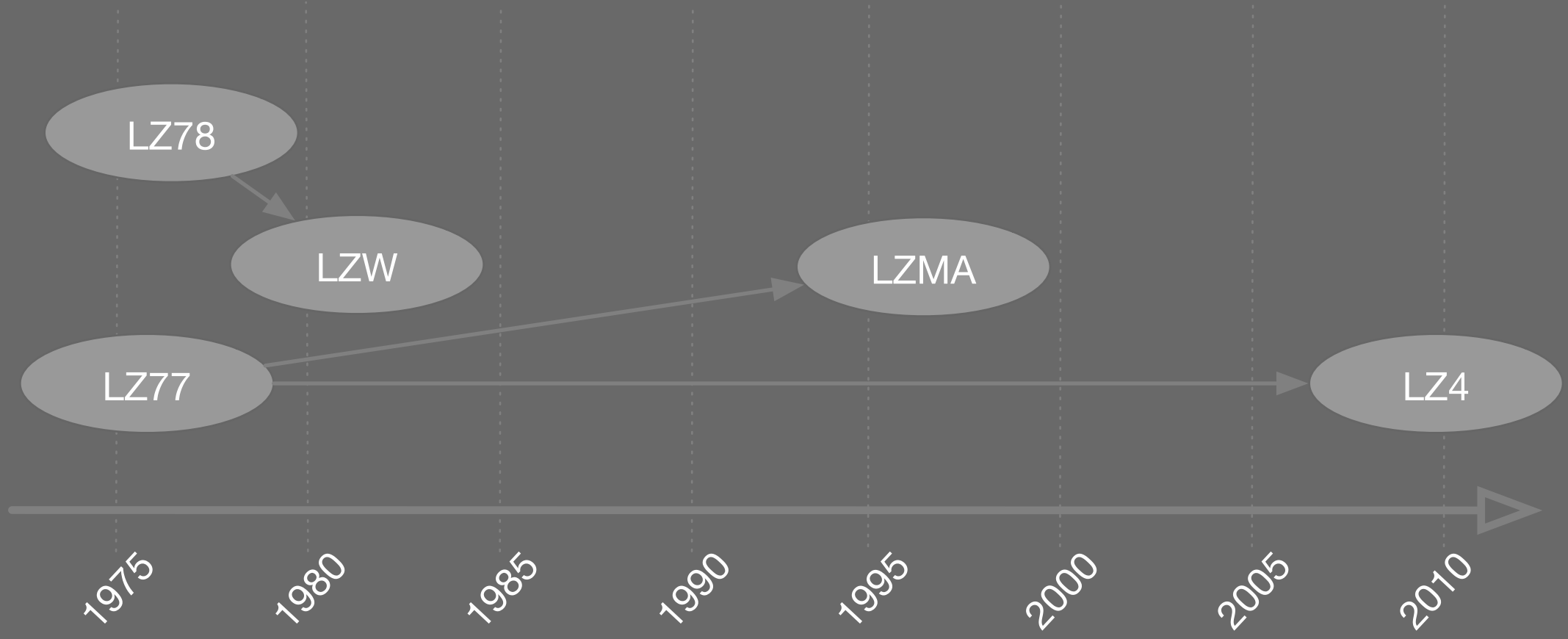
- Lz77 - https://www2.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf
- Lz78 - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.2892&rep=rep1&type=pdf
- Lzw - https://pi4.informatik.uni-mannheim.de/pi4.data/content/animations/losslesscompression/studarbeit.pdf
- Lzma - http://www.7-zip.de/sdk.html
- Range Encoding - https://docplayer.net/9798601-G-n-n-martin-presented-in-march-1979-to-the-video-data-recording-conference-ibm-uk-scientific-center-held-in-southampton-july-24-27-1979.html
- Lz4
  - https://ticki.github.io/blog/how-lz4-works/
  - http://fastcompression.blogspot.com/2011/05/lz4-explained.html
  - http://lz4.org
- Testimplementationen
  - https://github.com/inikep/lzbench
  - https://github.com/efutch/lzw

LZ Markov Chain Algorithm (LZMA)

Outputlängen:

- LIT: 0 + bytecode = 9 bit
- MATCH: 10 + len + dist = range(38, 44) bit
- SHORTREP: 1100 = 4 bit
- LONGREP:
  - LONGREP[0]: 1101 + len = range(8, 14) bit
  - LONGREP[1]: 1110 + len = range(8, 14) bit
  - LONGREP[2]: 11110 + len = range(9, 15) bit
  - LONGREP[3]: 11111 + len = range(9, 15) bit

Zeitstrahl


LZ78
LZW
LZMA
LZ77
LZ4
1975
1980
1985
1990
1995
2000
2005
2010

Bereichskodierer ganz

```
range = [0, 100000]
probZero = 0.8
input = "00100"

while input != "":
    c = nextChar(input)
    x = range[0] + 0.8 * (range[1] - range[0])
    if c == "0":
        range = [range[0], x]
    else:
        range = [x, range[1]]
```

```
output_Num = ""
a = str(range[0])
b = str(range[1])
for i in range(0, len(a))
    if a[i] == b[i]:
        output_Num += a[i]
        continue
    if int(a[i])+1 < int(b[i]):
        output_Num += str(int(a[i])+1)
    elif i+1 < len(a):
        output_Num += a[i]+ a[i+1]
    else:
        output_Num += a[i]

```