Compression techniques

David Bařina

February 22, 2013

Contents

Introduction

- multimedia data: text, documents, audio, image, animation, video, medical data, geographic data, ...
- need for compression: single frame of HD video $1920 \times 1080 \times 3 \approx 6\,\text{MB}$ (at $60\,\text{FPS} \approx 360\,\text{MB/s}$)
- formats: BMP, PNG, JPEG, MPEG-4, ...

Terminology

Compression

- reduce the amount of data, bitrate
- lossless vs. lossy
- reduce redundancy or irrelevance of data

Redundancy

- reduce redundancy = lossless compression
- original signal can be reconstructed without distortion

Irrelevance

- with regard to human perception
- elimination of irrelevant data = lossy compression
- original signal cannot be exactly reconstructed

Terminology

Coding

- mutually unambiguous assignment of symbols of one alphabet to symbols of other alphabet
- block vs. stream codes
- e.g. symbol 'A' $\rightarrow$ code 0110
- reduce redundancy = compression

Computer program

- compressor/encoder
- decompressor/decoder
- codec
- type of data (e.g. videocodec)

Terminology

Data model

- probabilistic (statistical) model
- context oriented (Markov) model

Compression methods

- symmetric vs. asymmetric (complexity)
- block vs. stream (block of symbols), e.g. bzip2 $\sim$900 kB
- single-pass, two-pass, multi-pass

- static = 1 pass, data model declared in advance
- semi-adaptive = 2 passes, model needs to be transferred
- adaptive (dynamic) = 1 pass, model modified on the fly

Terminology

Variable-length coding

- short codes for more frequently occurring symbols
- e.g. 'A' $\rightarrow$ 1, 'B' $\rightarrow$ 01, ..., 'Z' $\rightarrow$ 000000001
- implemented by entropy coding
- e.g. Huffman coding

Prefix code

- property that no code word is a prefix of another code word
- unambiguous decoding without delimiters
- e.g. 1, 01, 001, 0001, ...

Terminology

Dictionary

- data structure containing fragments of uncompressed file
- e.g. $0 \rightarrow$ "aa", $1 \rightarrow$ "ba", $2 \rightarrow$ "bc"
  $10021 \rightarrow$ "baaaaabcba"

Compression methods

- statistical (statistical data model)
- context (context oriented model)
- dictionary (dictionary)

Terminology

Compression ratio

$$= \frac{\text{output size}}{\text{input size}}$$

($< 1 \Rightarrow$ compression, $> 1 \Rightarrow$ expansion)

Evaluation of compression methods

- efficiency (compression ratio)
- time and memory complexity
- influence of data type on compression ratio
- quality (lossy methods)

Terminology

Compression methods

- predictive (predictor, error)
- transform (e.g. DCT)

Entropy

- quantity indicating the amount of "information"
- information content, necessary number of bits
- number of yes/no questions that can reveal the content of message
- unit is "bit"

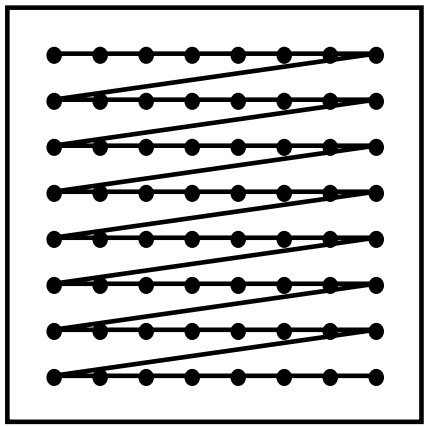
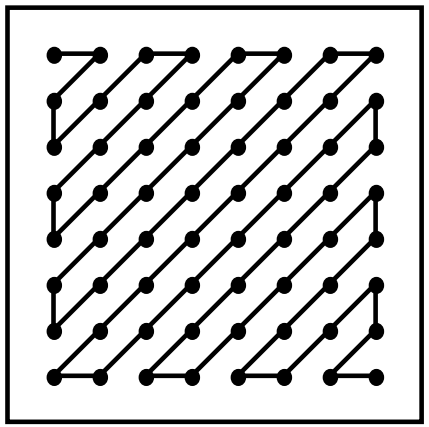
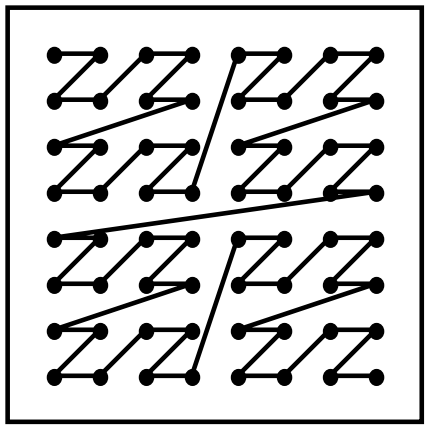
$$H = -\sum_{a \in A} p(a) \cdot log_2 p(a)$$

- entropy encoders compress data
  almost optimally regarding to their entropy

Multidimensional data

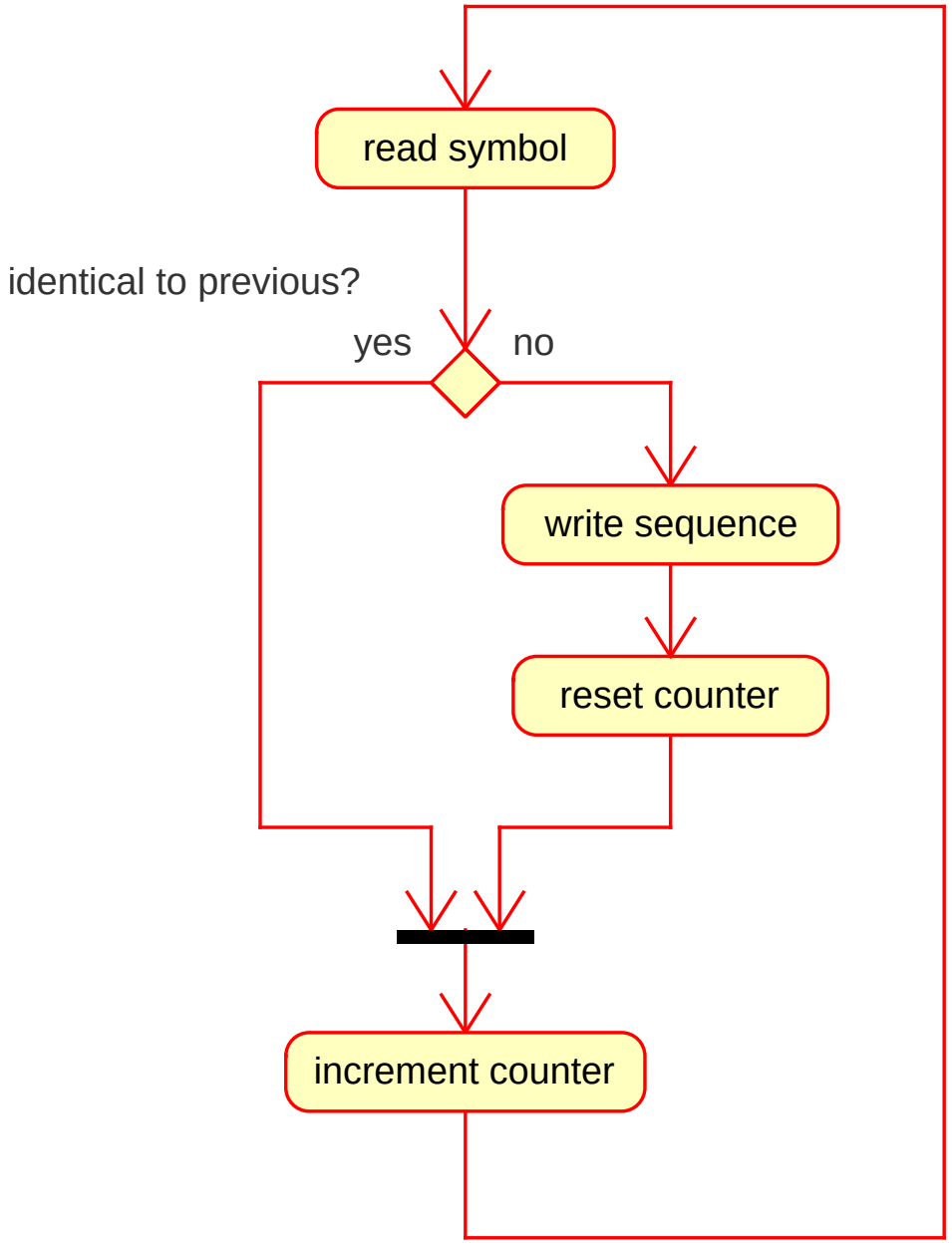
Linearization

- raster order (line by line)
- zig-zag, e.g. DCT in JPEG
- Morton order (Z-curve), e.g. DWT tree

RLE (Run-Length Encoding)

- coding sequences of identical characters
- many modifications
- part of complex methods
- code contains the information about the encoded symbol and the number of repetitions
- e.g. "A A A A B C D" $\rightarrow$ "4$\times$A B C D"
- "escape" symbol, "escape" sequence
- used almost everywhere (BMP, PCX, TIFF, TGA, JPEG, bzip2)



Coding of differences

- relative coding, delta coding
- part of complex methods
- replace input value with difference from the previous value
- trivial predictive method
- error of prediction is futher encoded
- e.g.

  17 16 18 32 35 35 34 28 28 $\rightarrow$ -1 +2 +14 +3 0 -1 -6 0

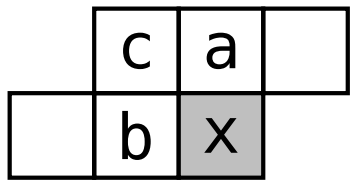
Predictive coding

- the coded symbol is predicted from already encoded symbols
- error of prediction is futher encoded
- order of predictor (1st, 2nd, . . .)
- domains (1-D, 2-D, tree)

Predictors

- linear (delta coding, Lossless JPEG, PNG)
- non-linear (median, MED/LOCO-I, Paeth, GAP)

$$\hat{x} = \begin{cases} \min(a, b) & : c \geq \max(a, b) \\ \max(a, b) & : c \leq \min(a, b) \\ a + b - c \end{cases}$$



Unary coding

- very simple entropy coding
- optimal for $p(n) = 2^{-n}$
- maps non-negative integers N to code words, e.g. $N \rightarrow N \times 1, 0$

  $0 \rightarrow 0$
  $1 \rightarrow 10$
  $2 \rightarrow 110$
  $3 \rightarrow 1110$
  $\ldots$
- 0 and 1 can be exchanged

Golomb-Rice coding

- special case of Golomb coding; fast implementation
- used e.g. in JPEG-LS, FLAC, MPEG-4 ALS
- maps non-negative integers N to code words
- the code is adjustable with parameter $M = 2^C$
- for $M = 1$ the coding is unary coding
- to obtain a code of number N, following variables have to be determined

$$Q = \lfloor N/M \rfloor \quad R = N - Q \cdot M \quad C = \lceil \log_2 M \rceil$$

- Q is futher encoded with unary code, R with binary code with C bits

Golomb-Rice coding

for $M = 4$

N	Q	R	code
0	0	0	1 00
1	0	1	1 01
2	0	2	1 10
3	0	3	1 11
4	1	0	01 00
5	1	1	01 01
6	1	2	01 10
7	1	3	01 11
8	2	0	001 00
9	2	1	001 01
10	2	2	001 10
11	2	3	001 11
12	3	0	0001 00

Shannon–Fano coding

- method creates the code-words according to probabilities of coded symbols (adapts on the data)
- best results are achieved for the power of 2 probabilities
- in practice, Huffman coding has slightly better compresion ratio
- symbols are leaves in binary tree with edges (0 and 1) which represent code-words
- tree construction:
  1. sort symbols according their probabilities
  2. split this set into two sets in such a way that these sets will have equal or very similar sum of probabilities
  3. recursively apply step 2 on both sets (nodes of tree) up to decomposition to single symbols
- used in ZIP/Implode

Shannon–Fano coding

example

probability					code
0.25	1	1			11
0.20	1	0			10
0.15	0	1	1		011
0.15	0	1	0		010
0.10	0	0	1		001
0.10	0	0	0	1	0001
0.05	0	0	0	0	0000

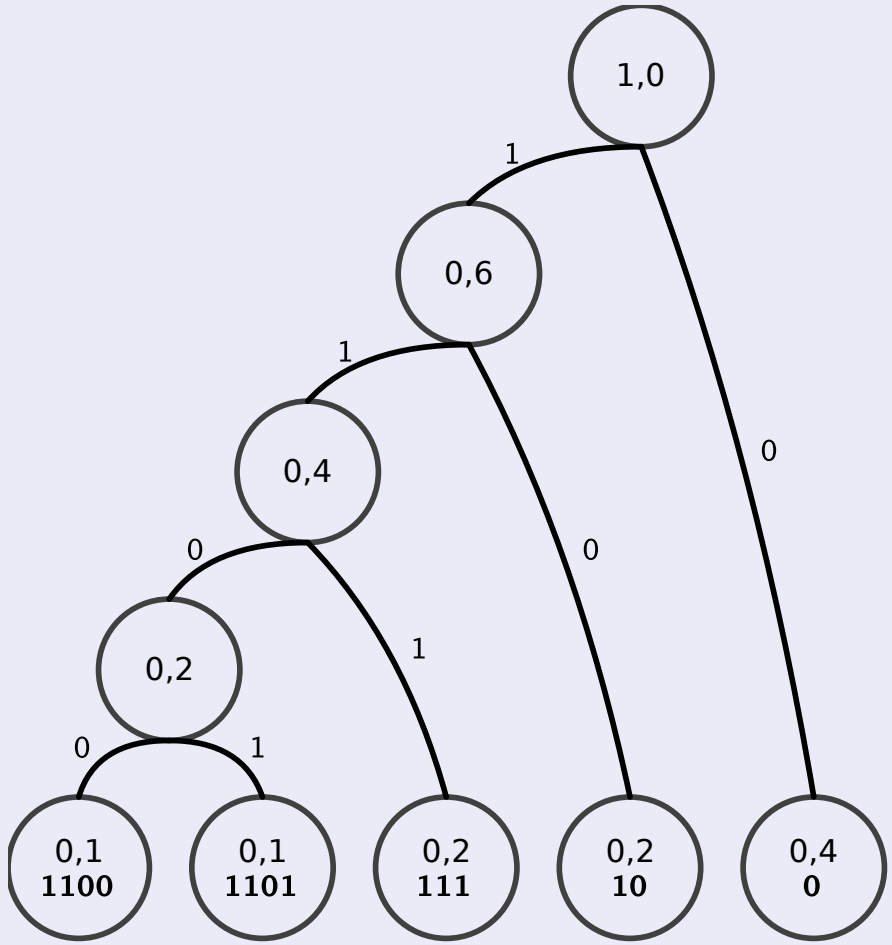
Huffman coding

- popular entropy coding method
- can adapts to data
- best results for the power of 2 probabilities
- symbols are the leaves in a binary tree in which the edges indicate the code word
- used in bzip2, Deflate, JPEG, MP3
- tree construction:
  1. sort symbols according to their probability
  2. take two symbols with lowest probabilities, link them to the new node
  3. continue with step 2 until there are unlinked nodes
- in adaptive variant, the tree has to be corrected on the fly

Huffman coding

example

p					code
0.4				0	0
0.2			0	1	10
0.2		1	1	1	111
0.1	1	0	1	1	1101
0.1	0	0	1	1	1100



Arithmetic coding

- optimal codes for any symbol probability
- method assign one code-word to the entire data, not only single symbol
- method starts with the interval which is narrowed according to the probability of coded symbols
- interval narrowing requires more bits, so the code-word length increases progressively
- the idea: symbol with higher probability narrow interval less in comparison with symbol with lower probability
- practical implementation have to work with integers
- used in: context coders, JPEG, Dirac

Arithmetic coding

example

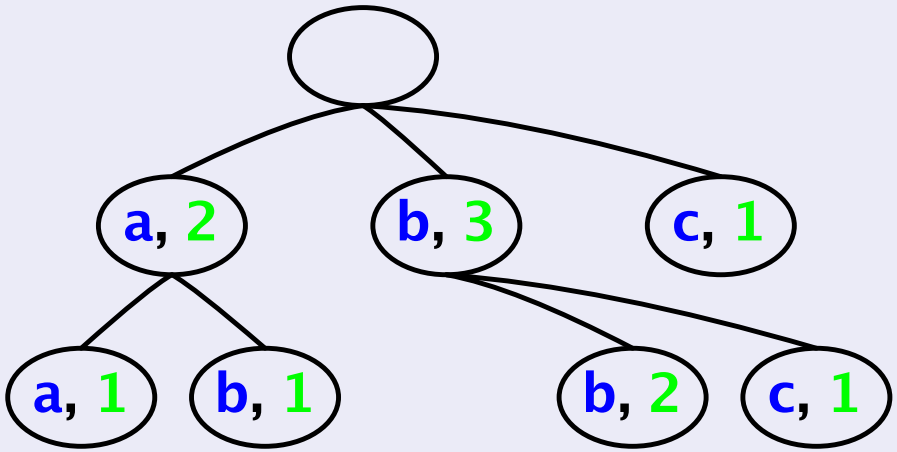
Context-oriented compression

- most commonly using arithmetic coding or its modification
- unlike the arithmetic coding, these methods do not encode the probability of single symbol
- instead, they encode the probability of symbol occurrence in a particular context
- context: several already encoded symbols, pixels, bits or coefficients $\rightarrow$ order of context
- the context is used to predict a next symbol (assign probability to any next symbol)
- escape codes: switch of context (PPMx)
- used in: MPEG-4 (CABAC, CAVLC), JPEG 2000 (EBCOT), JPEG-LS, PPMx

Context-oriented compression

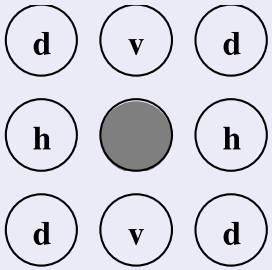
example

- string aabbbc
- context of order of 1
- this model predict e.g. after 'b' symbol occurence of 'b' symbols with probability of 66 % and 'c' symbol with probability 33 %



EBCOT

coder used in JPEG 2000 image compresion standard



LZ77

- published by A. Lempel and J. Ziv in 1977
- many modifications
- used in Deflate
- sliding window
- two parts: dictionary and lookahead buffer
- in practice, thousands vs. tens of symbols
- encoder produces tags `(offset, length, symbol)`

←...`east#easily#t`|`eases`...←

- "eas" found 2× on positions 8 and 13 (match)
- encoder produce tag `(13, 3, 'e')`
- tag elements are encoded with corresponding number of bits $\lceil \log_2 S \rceil, \lceil \log_2 (L-1) \rceil, \lceil \log_2 A \rceil$, where A is a size of used alphabet

LZ77

←...|east#easily#t|rashe|...←

- when fragment is not found in dictionary, encoder produces tag `(0, 0, 'r')`

←...|east#easily#t|ttttt|...←

- match may exceed the search buffer, here `(1, 5, 't')`
- it is the reason for the number of bits $\lceil \log_2(L-1) \rceil$ of length
- LZ77 assumed that fragments occur close together
- there are many enhancements: variable length tags, sophisticated data structures, omit symbol in tag

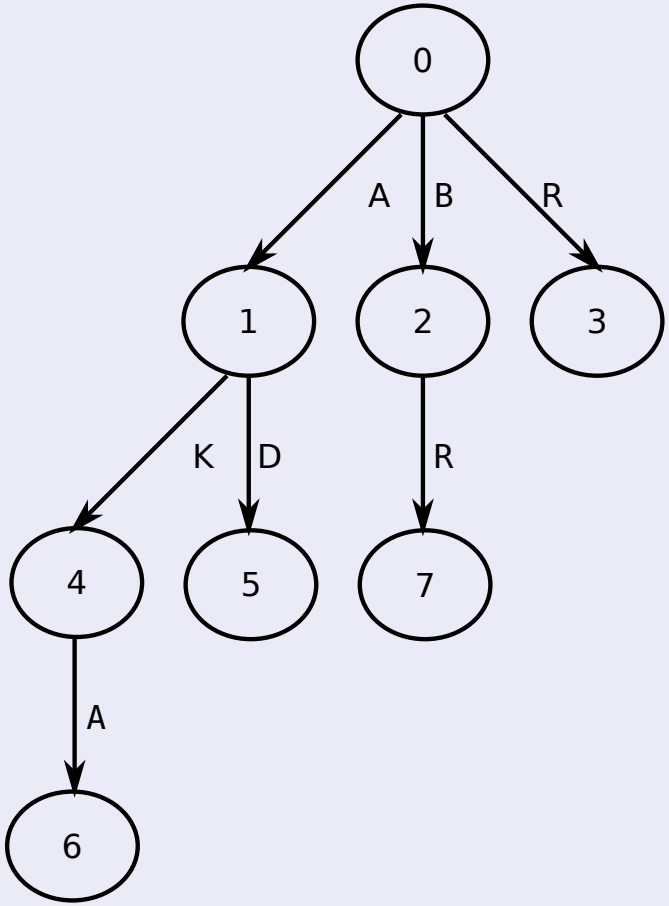
LZ78

- published by A. Lempel and J. Ziv in 1978
- many modifications, e.g. LZW
- no sliding window, only the dictionary
- is memory intensive, can be solved in different ways
- suitable data structure to maintain the dictionary is *trie*
- first item in the dictionary is an empty string
- encoder creates tags `(index, symbol)`
- during compression, longest string stored in dictionary is found in input stream
- now, tag with its index and next symbol is created
- every tag indicates new string, this string is stored into dictionary

LZ78

Example: ABRAKADAKABRA

- at the beginning, the dictionary is empty
- 'A', 'B', 'R' are encoded like
  `(0, 'A'), (0, 'B'), (0, 'R')`
- 'AK' and 'AD' are stored unde node with 'A' and encoded like
  `(1, 'K'), (1, 'D')`
- in the same way, 'AKA', 'BR' and last 'A' are encoded like
  `(4, 'A'), (2, 'R'), (1, EOF)`



LZW

- variant of LZ78 developed by T. Welch in 1984
- used in GIF, TIFF, PDF
- use dictionary initialized with every symbols of alphabet
- tag has only one element: `(index)`
- encoder:
  1. in input stream, looks for longest string I stored in dictionary
  2. thus, next symbol x causes that Ix is not found in dictionary
  3. dictionary index of I is sent to output, Ix is stored into dictionary, I is now x
  4. go to step 1
- decoder:
  1. reads dictionary index of I and puts string I to output
  2. string Ix should be stored into dictionary, however x is not known
  3. reads next index of J and puts corresponding string J to output, first symbol of J is x
  4. now Ix can be stored into dictionary, J is now I
  5. go to step 2.

LZW: example

Example: `sir#sid#` (encoder)

input x='s'	Ix='s' found	I=Ix='s'		
input x='i'	Ix='si' not found	output idx. I='s'	store Ix='si'	I=x='i'
input x='r'	Ix='ir' not found	output idx. I='i'	store Ix='ir'	I=x='r'
input x='#'	Ix='r#' not found	output idx. I='r'	store Ix='r#'	I=x='#'
input x='s'	Ix='#s' not found	output idx. I='#'	store Ix='#s'	I=x='s'
input x='i'	Ix='si' found	I=Ix='si'		
input x='d'	Ix='sid' not found	output idx. I='si'	store Ix='sid'	I=x='d'
input x='#'	Ix='d#' not found	output idx. I='d'	store Ix='d#'	I=x='#'
input x=EOF	Ix='#EOF' not found	output idx. I='#'	end	

Dictionary:

0–255	characters 0–255
256	'si'
257	'ir'
258	'r#'
259	'#s'
260	'sid'
261	'd#'

LZW: example

Example: `sir#sid#` (decoder)

input index J='s'	output J='s'	x=J(1)='s'	store Ix='s'	I=J='s'
input index J='i'	output J='i'	x=J(1)='i'	store Ix='si'	I=J='i'
input index J='r'	output J='r'	x=J(1)='r'	store Ix='ir'	I=J='r'
input index J='#'	output J='#'	x=J(1)='#'	store Ix='r#'	I=J='#'
input index J='si'	output J='si'	x=J(1)='s'	store Ix='#s'	I=J='si'
input index J='d'	output J='d'	x=J(1)='d'	store Ix='sid'	I=J='d'
input index J='#'	output J='#'	x=J(1)='#'	store Ix='d#'	I=J='#'

Dictionary:

0–255	characters 0–255
256	'si'
257	'ir'
258	'r#'
259	'#s'
260	'sid'
261	'd#'

Deflate

- used in ZIP (originaly), zlib/gzip, 7-Zip, PNG, MNG, PDF
- combination of LZ77 and Huffman coding
- in contrast with LZ77, tags have only two elements (`offset`, `length`)
- missing item (`symbol`) is written into output stream separately
- compressed stream consists of three entities: literals/symbols, offsets/distances and lengths
- these entities are encoded using Huffman codes using two tables: literals/lengths and offsets
- lengths are limited up to 258, literals are bytes (0–255); offsets up to buffer size of 32 kB
- data are compressed in blocks of various sizes

Deflate

- defines 3 modes of compression:
  1. uncompressed (max. 65 535 B)
  2. compression with predefined Huffman tables
  3. compression with tables stored in compressed stream
- match selection is delayed (by one symbol)

←...`she#needs#then#there#`|`the#new`...←

- do not selects match (11,3) = "the"
- "t" encodes like literal
- selects "delayed" match (20,5) = "he#ne"

Suitable combinations of methods

- data (uncompressed)
- data $\rightarrow$ RLE (BMP, TGA)
- data $\rightarrow$ prediction $\rightarrow$ EC (JPEG-LS)
- data $\rightarrow$ prediction $\rightarrow$ RLE $\rightarrow$ EC
- data $\rightarrow$ prediction $\rightarrow$ dictionary method (PNG)
- data $\rightarrow$ dictionary method (GIF)
- data $\rightarrow$ transform $\rightarrow$ RLE+EC (JPEG)

Summarization

- terminology (redundancy, data model, types of compression methods, prefix code, dictionary, compression ratio, entropy)
- multidimensional data processing
- basic methods: RLE, delta coding, prediction
- entropy coders (unary, Golomb-Rice, Huffman and arithmetic coders)
- context-oriented compression
- dictionary methods (LZ77, LZ78, LZW and Deflate)
- combinations of these methods

Sources

- David Salomon. Data Compression: The Complete Reference. 4th ed., Springer, 2006.
- `http://www.stringology.org/DataCompression/`
- specifications of formats