Improving Run Length Encoding through preprocessing

Sven Fiergolla

14. Januar 2020

Run Length Encoding (RLE)

- Employed in the transmission of analog television signals as far back as 1967 [1]
- Particularly well suited to palette-based bitmap images such as computer icons

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$aaaaabbbbbaaaaaa$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$aaaaabbbbbbaaaaaa$$

$$a^5 b^6 a^6$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^5 b^6 a^6$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^5 b^6 a^6$$

assume $|\Sigma| = 2$ and runs start with a

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^5 b^6 a^6$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 7 ($= 3$ bits per run)

$$566$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^5 b^6 a^6$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 7 ($= 3$ bits per run)

$$\underbrace{5}_{101}66$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^5 b^6 a^6$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 7 ($= 3$ bits per run)

$$\underbrace{5}_{101}\underbrace{6}_{110}6$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^5 b^6 a^6$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 7 ($= 3$ bits per run)

$$\underbrace{5}_{101}\underbrace{6}_{110}\underbrace{6}_{110}$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$\overbrace{a^5 b^6 a^6}^{17 \ \ bit}$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 7 ($= 3$ bits per run)

$$\overbrace{\underbrace{5}_{101}\underbrace{6}_{110}\underbrace{6}_{110}}^{9 \ \ bit}$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$aabaabbabbbababaabb$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$aabaabbabbbababaabb$$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

assume $|\Sigma| = 2$ and runs start with a

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run)

$$212213111122$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run)

$$\underbrace{2}_{10}12213111122$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run)

$$\underbrace{2}_{10}\underbrace{1}_{01}2213111122$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run)

$$\underbrace{2}_{10}\underbrace{1}_{01}\underbrace{2}_{10}213111122$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run)

$$\underbrace{2}_{10}\underbrace{1}_{01}\underbrace{2}_{10}\underbrace{2}_{10}\underbrace{1}_{01}\underbrace{3}_{11}\underbrace{1}_{..}11122$$

Run Length Encoding (RLE) - binary RLE

assume $\Sigma = \{a, b\}$

$$\overbrace{a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2}^{19 \quad bit}$$

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run)

$$\overbrace{\underbrace{2}_{10}\ \underbrace{1}_{01}\ \underbrace{2}_{10}\ \underbrace{2}_{10}\ \underbrace{1}_{01}\ \underbrace{3}_{11}\ \underbrace{1}_{..}\ 11122}^{24 \quad bit}$$

Run Length Encoding (RLE) - binary RLE

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run / per RLE number)

$$b^1 a^5 b^2$$

Run Length Encoding (RLE) - binary RLE

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run / per RLE number)

$$\underbrace{b^1}_{00+01} a^5 b^2$$

Run Length Encoding (RLE) - binary RLE

assume $|\Sigma| = 2$ and runs start with a
set maximum run length to 3 ($= 2$ bits per run / per RLE number)

$$\underbrace{b^1}_{00+01} \quad \underbrace{a^5}_{11+00+10} \quad b^2$$

Run Length Encoding (RLE) - byte wise RLE

Run Length Encoding (RLE) - byte wise RLE

assume $|\Sigma| = 2^8$
set maximum run length to 4 ($= 2$ bits per run / per RLE number)

Run Length Encoding (RLE) - byte wise RLE

$$b^1 a^2 c^3$$

Run Length Encoding (RLE) - byte wise RLE

$$\underbrace{b^1}_{01100010+00} \quad a^2 c^4$$

Run Length Encoding (RLE) - byte wise RLE

$$\underbrace{01100010+00}_{}\quad\underbrace{01100001+01}_{}\quad\underbrace{01100011+11}_{}$$

Run Length Encoding (RLE)

- Used for bitmap images or DNA sequence encoding [2] [3].

Run Length Encoding (RLE)

- Used for bitmap images or DNA sequence encoding [2] [3].
- The International Telecommunication Union describes the standard to encode fax transmissions, known as T.45 [4].
  - RLE is combined with Huffman Encoding into *Modified Huffman Encoding*.

Huffman Encoding

$$\Sigma = \{a, b, c\}$$
$$w = cacbcacc$$

Huffman Encoding

$$\Sigma = \{a, b, c\}$$
$$w = cacbcacc$$
$$l_a(w) = 2 \quad l_b(w) = 1 \quad l_c(w) = 5$$

Huffman Encoding

$$\Sigma = \{a, b, c\}$$
$$w = cacbcacc$$
$$l_a(w) = 2 \quad l_b(w) = 1 \quad l_c(w) = 5$$

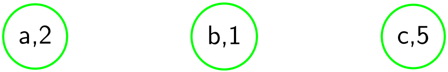


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

$$\Sigma = \{a, b, c\}$$
$$w = cacbcacc$$
$$l_a(w) = 2 \quad l_b(w) = 1 \quad l_c(w) = 5$$

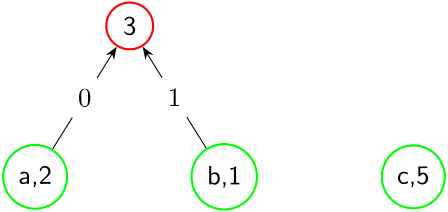


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

$$\Sigma = \{a, b, c\}$$
$$w = cacbcacc$$
$$l_a(w) = 2 \quad l_b(w) = 1 \quad l_c(w) = 5$$

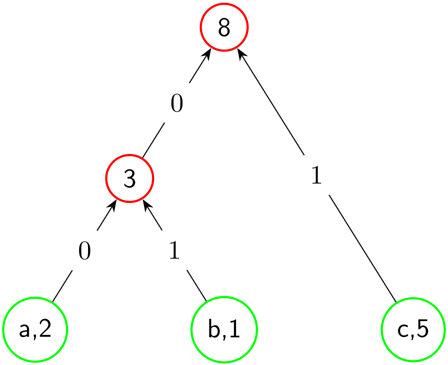


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

$$\Sigma = \{a, b, c\}$$
$$w = cacbcacc$$
$$l_a(w) = 2 \quad l_b(w) = 1 \quad l_c(w) = 5$$

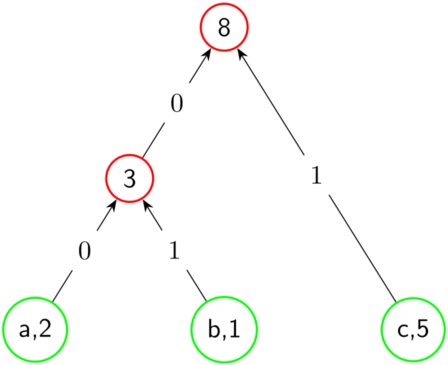


Figure: Example Huffman tree with 3 leaf nodes.

$$a = 00 \quad b = 01 \quad c = 1$$

Modified Huffman Encoding T.45

run length	white run Huffman codes	black run Huffman codes
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
4	1011	011
...		
20	0001000	00001101000
...		
64+	11011	0000001111
128+	10010	000011001000

Table: T4 static Huffman codes.

Modified Huffman Encoding T.45

run length	white run Huffman codes	black run Huffman codes
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
4	1011	011
...		
20	0001000	00001101000
...		
64+	11011	0000001111
128+	10010	000011001000

Table: T4 static Huffman codes.

$$\underbrace{1111}_{2} \rightarrow \text{`11`}$$

Modified Huffman Encoding T.45

run length	white run Huffman codes	black run Huffman codes
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
4	1011	011
...		
20	0001000	00001101000
...		
64+	11011	0000001111
128+	10010	000011001000

Table: T4 static Huffman codes.

$$0^{132} \rightarrow \text{‘}10010\text{‘} + \text{‘}1011\text{‘}$$

Run Length Encoding (RLE)

file	size original	bits per run	size encoded	ratio in %	*bps*
pic	513216	2	350292	68.25	5.46
		3	235067	45.80	3.66
		4	165745	32.29	2.58
		5	126349	24.61	1.96
		6	106773	20.80	1.66
		7	100098	19.50	1.56
		8	101014	19.68	1.57

Table: The bitmap image file *pic* with increasing bits per binary RLE encoded number.

Run Length Encoding (RLE)

file	size original	bits per run	size encoded	ratio in %	*bps*
pic	513216	2	350292	68.25	5.46
		3	235067	45.80	3.66
		4	165745	32.29	2.58
		5	126349	24.61	1.96
		6	106773	20.80	1.66
		7	100098	19.50	1.56
		8	101014	19.68	1.57

Table: The bitmap image file *pic* with increasing bits per binary RLE encoded number.

- Byte-wise RLE achieves 27.2% of its original size using 2.17 *bps* with 6 bits per run.

Calgary Corpus

file	size	description
bib	111261	ASCII text - 725 bibliographic references
book1	768771	unformatted ASCII text
book2	610856	ASCII text in UNIX "troff" format
geo	102400	32 bit numbers in IBM floating point format
news	377109	ASCII text - USENET batch file on a variety of topics
obj1	21504	VAX executable program
obj2	246814	Macintosh executable program
paper1	53161	UNIX "troff" format
paper2	82199	UNIX "troff" format
pic	513216	1728 x 2376 bitmap image
progc	39611	Source code in C
progl	71646	Source code in Lisp
progp	49379	Source code in Pascal
trans	93695	ASCII and control characters

Table: The Calgary Corpus.

RLE - Unmodified compression on the Calgary Corpus

bits per rle number	byte-wise RLE		binary RLE	
	ratio in %	*bps*	ratio in %	*bps*
8	165	13.20	329	26.38
7	154	12.38	288	23.11
6	144	11.57	248	19.87
5	134	10.77	208	16.66
4	125	10.00	168	13.51
3	116	9.29	131	10.50
2	109	8.74	104	8.36

Table: Byte-wise and binary RLE on the Calgary Corpus.

Findings

- Files with long runs work really well with RLE.

Findings

- Files with long runs work really well with RLE.
- Most files other than pallet based images do not contain long runs of identical bit values.

Findings

- Files with long runs work really well with RLE.
- Most files other than pallet based images do not contain long runs of identical bit values.
- Artificially creating runs on arbitrary data will improve the performance of RLE.

Basics of Compression

- Non random data contains redundant information
- Compression is about pattern or structure identification and exploitation

Basics of Compression

- Non random data contains redundant information
- Compression is about pattern or structure identification and exploitation
- No algorithm can compress all possible data of a given length, even by one byte (Kolmogorov Complexity [5])

Basics of Compression - Entropy Encoding

- generating a probability model for the data
- compute variable length codes

Basics of Compression - Entropy Encoding

- generating a probability model for the data
- compute variable length codes
- low speed, high compression strength
- recommended for poorly structured data

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities
- Run Length Encoding (1967)
  - computes runs of identical symbols

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities
- Run Length Encoding (1967)
  - computes runs of identical symbols
- Arithmetic Encoding (1979)
  - encodes a message of symbols in a single rational number in [0,1]

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities
- Run Length Encoding (1967)
  - computes runs of identical symbols
- Arithmetic Encoding (1979)
  - encodes a message of symbols in a single rational number in [0,1]
- Asymmetric Numeral Systems (ANS) Encoding (2014)
  - encodes a message of symbols in a single natural number

Basics of Compression - Dictionary Encoding

- maintain a dictionary of strings for either a *sliding window* or the whole data
  - replace later occurrence with reference position an length

Basics of Compression - Dictionary Encoding

- maintain a dictionary of strings for either a *sliding window* or the whole data
  - replace later occurrence with reference position an length
- High speed, moderate compression strength

Basics of Compression - Dictionary Encoding

- maintain a dictionary of strings for either a *sliding window* or the whole data
  - replace later occurrence with reference position an length
- High speed, moderate compression strength
- Famous Lempel-Ziv methods LZ77 and LZ78 (1977/78)
  - many derivatives, some still used today

State of the art

method	options	size in bytes	compression	*bps*
uncompressed		3,145,718	100.0%	8.00
compress 4.2.4		1,250,382	40.4%	3.24
gzip v1.10	-9	1,021,720	32.4%	2.60
ZIP v3.0	-9	1,019,783	32.4%	2.59
zstandard 1.4.2	–ultra-23 -long=30	887,004	28.1%	2.25
bzip2 v1.0.8	–best	832,443	26.4%	2.11
brotli 1.0.7	-q 11 -w 24	826,638	26.3%	2.10
p7zip 16.02 (deflate)	a -mx10	821,873	26.1%	2.08
p7zip 16.02 (PPMd)	a -mm=ppmd o=32	763,067	24.2%	1.93
ZPAQ v7.15	-m5	659,700	20.9%	1.67
paq8hp*	-	-	-	-
cmix v18	-c -d	554,983	17.6%	1.41

Table: State of the art compression ratios on the Calgary Corpus.

Design - Preprocessing

- Vertical interpretation

Design - Preprocessing

- Vertical interpretation
- Dynamic byte remapping

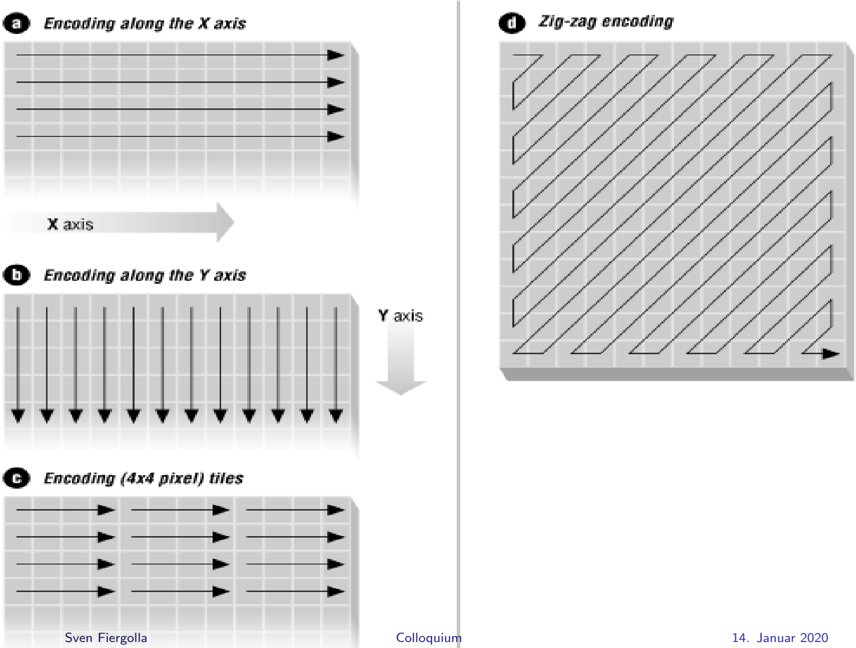
Design - Preprocessing

- Vertical interpretation
- Dynamic byte remapping
- Burrows-Wheeler-Transformation

Design - Preprocessing

- Vertical interpretation
- Dynamic byte remapping
- Burrows-Wheeler-Transformation
- Huffman Encoding of RLE runs

Preprocessing - Vertical interpretation



Preprocessing - Vertical interpretation

$$w = abraca$$

0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1

Preprocessing - Vertical interpretation

$$w = abraca$$

0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1

Preprocessing - Vertical interpretation

$$w = abraca$$

0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1

Preprocessing - Vertical interpretation

$$w = abraca$$

0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w = abraca \\
{\color{red}0}1100001 \\
01100010 \\
01110010 \\
01100001 \\
01100011 \\
01100001
\end{aligned}
\tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w = abraca \\
01100001 \\
{\color{red}0}1100010 \\
01110010 \\
01100001 \\
01100011 \\
01100001
\end{aligned}
\tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w &= abraca \\
& 01100001 \\
& 01100010 \\
& {\color{red}0}1110010 \\
& 01100001 \\
& 01100011 \\
& 01100001
\end{aligned} \tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w = abraca \\
01100001 \\
01100010 \\
01110010 \\
{\color{red}0}1100001 \\
01100011 \\
01100001
\end{aligned}
\tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w &= abraca \\
&01100001 \\
&01100010 \\
&01110010 \\
&01100001 \\
&\textcolor{red}{0}1100011 \\
&01100001
\end{aligned} \tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w &= abraca \\
&01100001 \\
&01100010 \\
&01110010 \\
&01100001 \\
&01100011 \\
&\textcolor{red}{0}1100001
\end{aligned}
\tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w &= abraca \\
& 0{\color{red}1}100001 \\
& 01100010 \\
& 01110010 \\
& 01100001 \\
& 01100011 \\
& 01100001
\end{aligned} \tag{1}
$$

Preprocessing - Vertical interpretation

$$
\begin{aligned}
w &= abraca \\
&01100001 \\
&0\textcolor{red}{1}100010 \\
&01110010 \\
&01100001 \\
&01100011 \\
&01100001
\end{aligned} \tag{1}
$$

Preprocessing - Vertical interpretation

bits per rle number	ratio in %	*bps*
8	255.22	20.41
7	224.45	17.95
6	194.74	15.57
5	167.04	13.36
4	142.58	11.40
3	127.80	10.22
2	139.79	11.18

Table: Binary RLE on vertical interpreted data, fixed run lengths.

Preprocessing - Vertical interpretation

$$w = abraca$$

$$\begin{array}{c}01100001\\01100010\\01110010\\01100001\\01100011\\01100001\end{array}$$

Preprocessing - Vertical interpretation

$$w = abraca$$

$$\text{8 bit per run}\left\{\begin{array}{l}
\left.01100001\right\}\text{2 bits per run}\\
\left.\begin{array}{l}01100010\\01110010\end{array}\right\}\text{2 bits per run}\\
\left.\begin{array}{l}01100001\\01100011\\01100001\end{array}\right\}\text{2 bits per run}
\end{array}\right.$$

Preprocessing - Vertical interpretation

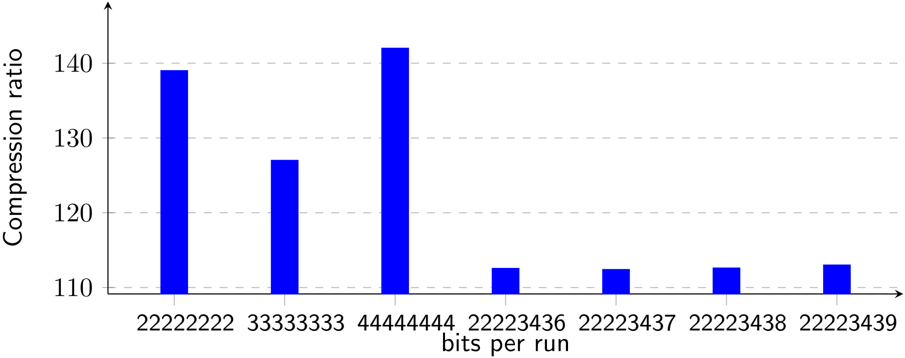


Figure: Byte mapping and varying maximum run lengths.

Preprocessing - Vertical interpretation

file	size original	size encoded	ratio in %	*bps*
bib	111261	129424	116.32	9.31
book1	768771	820463	106.72	8.54
book2	610856	659811	108.01	8.64
geo	102400	162274	158.47	12.68
news	377109	400810	106.28	8.50
obj1	21504	31592	146.91	11.75
obj2	246814	379591	153.80	12.30
paper1	53161	57654	108.45	8.68
paper2	82199	88121	107.20	8.58
pic	513216	533254	103.90	8.31
progc	39611	41360	104.42	8.35
progl	71646	74554	104.06	8.32
progp	49379	53403	108.15	8.65
trans	93695	99818	106.54	8.52
all files	3145718	3536225	112.41	8.99

Table: Calgary Corpus encoded, vertical encoding, using bits per run (2, 2, 2, 2, 3, 4, 3, 7).

Preprocessing

Current preprocessing steps:

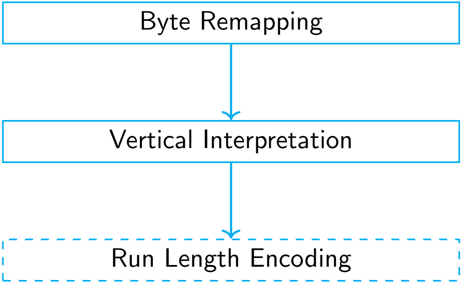
Vertical Interpretation

↓

Run Length Encoding

Preprocessing

Current preprocessing steps:



Preprocessing - Byte Remapping

$w = abracab$

01100001
01100010
01110010
01100001
01100011
01100001
01100010

Preprocessing - Byte Remapping

$$w = abracab$$

$$\begin{array}{l} 01100001 \\ 01100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ 01100001 \\ 01100010 \end{array}$$

$$l_a(w) = 3 \quad l_b(w) = 2 \quad l_c(w) = 1 \quad l_r(w) = 1$$

Preprocessing - Byte Remapping

$$w = abracab$$

01100001
01100010
01110010
01100001
01100011
01100001
01100010

$$l_a(w) = 3 \quad l_b(w) = 2 \quad l_c(w) = 1 \quad l_r(w) = 1$$

Mapping:

$$a = 0x00 \quad b = 0x01 \quad c = 0x02 \quad r = 0x03$$

Preprocessing - Byte Remapping

$$w = abraca$$

00000000
00000001
00000011
00000000
00000010
00000000
00000001

Preprocessing - Byte Remapping

$$
\begin{array}{l}
w = abraca \\
00000000 \\
00000001 \\
00000011 \\
00000000 \\
00000010 \\
00000000 \\
\underbrace{000000}01
\end{array}
$$

- increased average run length on bits of higher significance

Preprocessing - Byte Remapping

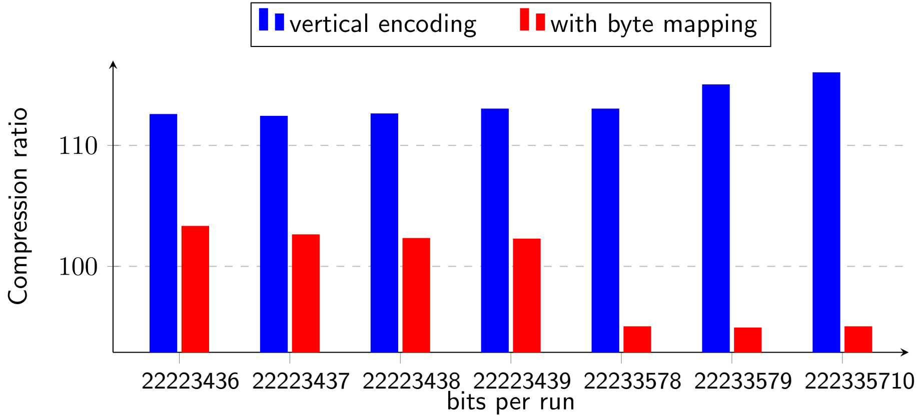


Figure: Byte mapping and varying maximum run lengths.

Preprocessing - Byte Remapping

file	size original	size encoded	ratio in %	*bps*
bib	111261	111579	100.29	8.02
book1	768771	669578	87.10	6.97
book2	610856	551757	90.33	7.23
geo	102400	144974	141.58	11.33
news	377109	363010	96.26	7.70
obj1	21504	30166	140.28	11.22
obj2	246814	340165	137.82	11.03
paper1	53161	50074	94.19	7.54
paper2	82199	71747	87.28	6.98
pic	513216	408136	79.53	6.36
progc	39611	38490	97.17	7.77
progl	71646	63765	89.00	7.12
progp	49379	46093	93.35	7.47
trans	93695	94729	101.10	8.09
all files	3145718	2988359	94.99	7.59

Table: Calgary Corpus encoded with vertical reading, byte remapping, using bits per run (2, 2, 3, 3, 3, 4, 5, 8).

Preprocessing - Burrows-Wheeler-Transformation

$$w = abcabr$$

Preprocessing - Burrows-Wheeler-Transformation

$$w = abcabr$$

a	b	c	a	b	r
r	a	b	c	a	b
b	r	a	b	c	a
a	b	r	a	b	c
c	a	b	r	a	b
b	c	a	b	r	a

Table: Burrows Wheeler Transformation Matrix (all cyclic rotations).

Preprocessing - Burrows-Wheeler-Transformation

$$w = abcabr$$

a	b	c	a	b	r
a	b	r	a	b	c
b	c	a	b	r	a
b	r	a	b	c	a
c	a	b	r	a	b
r	a	b	c	a	b

Table: Burrows Wheeler Transformation Matrix (all cyclic rotations, sorted).

Preprocessing - Burrows-Wheeler-Transformation

$$w = abcabr$$

a	b	c	a	b	r
a	b	r	a	b	c
b	c	a	b	r	a
b	r	a	b	c	a
c	a	b	r	a	b
r	a	b	c	a	b

Table: Burrows Wheeler Transformation Matrix (all cyclic rotations, sorted).

$$L = rcaabb$$

$$i = 1$$

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

word L
r
c
a
a
b
b

Table: Standard permutation generation of the word L.

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

word L	word with position
r	(r,1)
c	(c,2)
a	(a,3)
a	(a,4)
b	(b,5)
b	(b,6)

Table: Standard permutation generation of the word L.

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

word L	word with position	sorted
r	(r,1)	(a,3)
c	(c,2)	(a,4)
a	(a,3)	(b,5)
a	(a,4)	(b,6)
b	(b,5)	(c,2)
b	(b,6)	(r,1)

Table: Standard permutation generation of the word L.

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

word L	word with position	sorted	π_L^t
r	(r,1)	(a,3)	1 3
c	(c,2)	(a,4)	2 4
a	(a,3)	(b,5)	3 5
a	(a,4)	(b,6)	4 6
b	(b,5)	(c,2)	5 2
b	(b,6)	(r,1)	6 1

Table: Standard permutation generation of the word L.

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

This yields a standard permutation of:

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \tag{2}$$

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

This yields a standard permutation of:

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \tag{2}$$

Following $\pi_L^1(1)$ to $\pi_L^6(1)$:

$$3 \xrightarrow{\pi_L} 5 \xrightarrow{\pi_L} 2 \xrightarrow{\pi_L} 4 \xrightarrow{\pi_L} 6 \xrightarrow{\pi_L} 1 \tag{3}$$

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

This yields a standard permutation of:

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \tag{2}$$

Following $\pi_L^1(1)$ to $\pi_L^6(1)$:

$$3 \xrightarrow{\pi_L} 5 \xrightarrow{\pi_L} 2 \xrightarrow{\pi_L} 4 \xrightarrow{\pi_L} 6 \xrightarrow{\pi_L} 1 \tag{3}$$

Applying the sequence to the labeling function of the word L:

$$\lambda_L(3)\,\lambda_L(5)\,\lambda_L(2)\,\lambda_L(4)\,\lambda_L(6)\,\lambda_L(1) = abcabr = w \tag{4}$$

Preprocessing - Burrows-Wheeler-Transformation

Burrows-Wheeler-Transformation

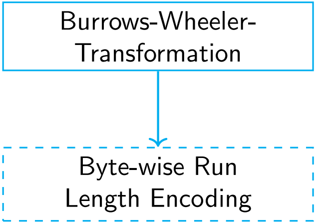
↓

Byte-wise Run Length Encoding

Preprocessing - Burrows-Wheeler-Transformation

bits per rle number	ratio in %	*bps*
3	95.41	7.63
2	91.39	7.31

Table: Initial Burrows-Wheeler-Transformation implementation on byte wise RLE.



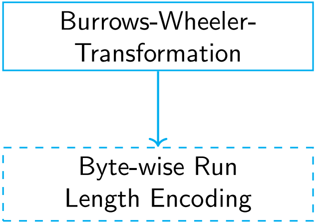
Preprocessing - Burrows-Wheeler-Transformation

bits per rle number	ratio in %	*bps*
3	95.41	7.63
2	91.39	7.31

Table: Initial Burrows-Wheeler-Transformation implementation on byte wise RLE.

bits per rle number	ratio in %	*bps*
3	91.62	7.33
2	89.46	7.15

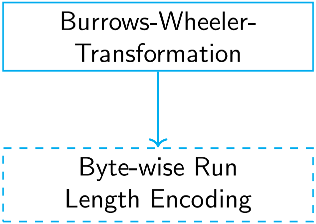
Table: Original Burrows-Wheeler-Transformation on byte wise RLE.



Preprocessing - Burrows-Wheeler-Transformation

bits per rle number	ratio in %	*bps*
8	74.42	5.95
7	69.90	5.59
6	65.58	5.24
5	61.71	4.93
4	58.98	4.71
3	59.18	4.73
2	67.69	5.41

Table: Sophisticated Burrows-Wheeler-Transformation on byte wise RLE.



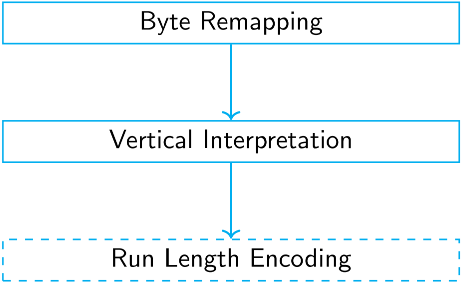
Preprocessing - Burrows-Wheeler-Transformation

file	size original	size encoded	compression	*bps*
bib	111261	59285	53.28	4.26
book1	768771	590879	76.86	6.15
book2	610856	374742	61.35	4.91
geo	102400	101192	98.82	7.91
news	377109	246047	65.25	5.22
obj1	21504	16467	76.58	6.13
obj2	246814	126626	51.30	4.10
paper1	53161	34130	64.20	5.14
paper2	82199	56507	68.74	5.50
pic	513216	136074	26.51	2.12
progc	39611	24312	61.38	4.91
progl	71646	31466	43.92	3.51
progp	49379	20862	42.25	3.38
trans	93695	32835	35.04	2.80
all files	3145718	1855520	58.98	4.71

Table: Calgary Corpus encoded with byte wise RLE after a Burrows-Wheeler-Transformation with 4 bit per run.

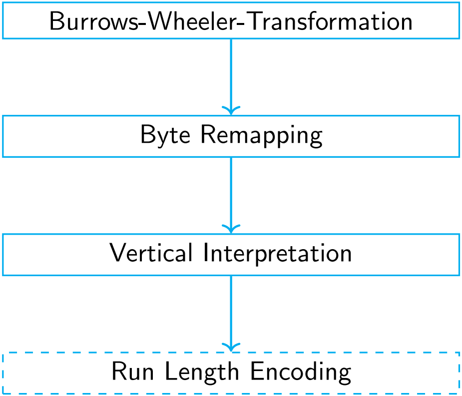
Preprocessing

Current preprocessing steps:



Preprocessing

Current preprocessing steps:



Preprocessing - Burrows-Wheeler-Transformation

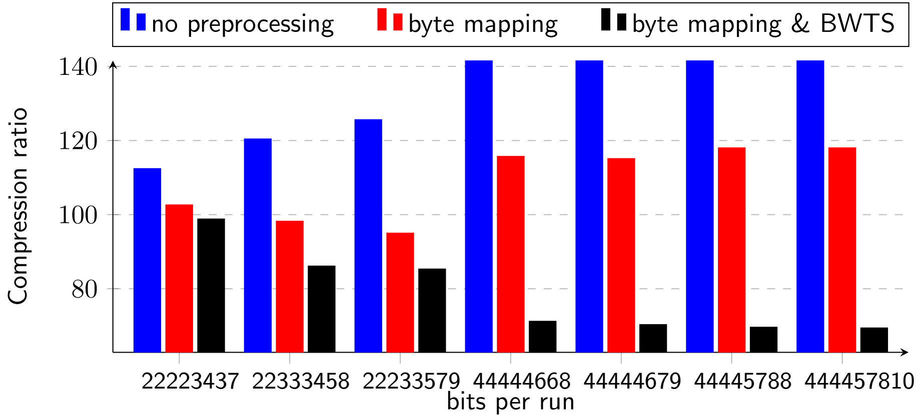


Figure: Byte mapping and varying maximum run lengths, all preprocessing steps.

Preprocessing - Burrows-Wheeler-Transformation

file	size original	size encoded	ratio in %	*bps*
bib	111261	73843	66.37	5.31
book1	768771	570348	74.19	5.94
book2	610856	409639	67.06	5.36
geo	102400	145950	142.53	11.40
news	377109	275396	73.03	5.84
obj1	21504	27023	125.66	10.05
obj2	246814	213392	86.46	6.92
paper1	53161	37344	70.25	5.62
paper2	82199	56490	68.72	5.50
pic	513216	227914	44.41	3.55
progc	39611	28275	71.38	5.71
progl	71646	38144	53.24	4.26
progp	49379	27029	54.74	4.38
trans	93695	49314	52.63	4.21
all files	3145718	2184197	69.43	5.55

Table: Calgary Corpus encoded, byte mapping and a BWTS as preprocessing, using bits per run (4, 4, 4, 4, 5, 7, 8, 10).

Preprocessing - Huffman Encoding RLE runs

Current preprocessing steps:

Burrows-Wheeler-Transformation

↓

Byte Remapping

↓

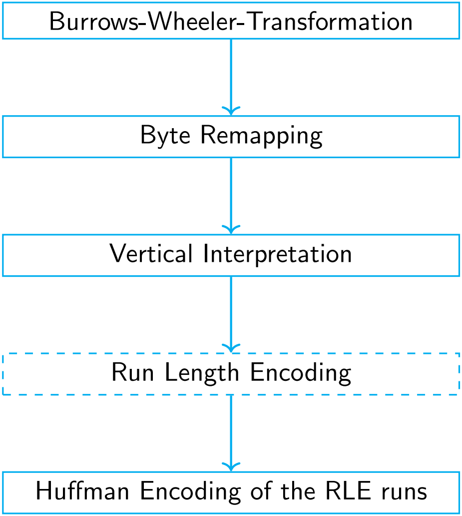
Vertical Interpretation

↓

Run Length Encoding

Preprocessing - Huffman Encoding RLE runs

Current preprocessing steps:



Preprocessing - Huffman Encoding RLE runs

file	size original	size encoded	ratio in %	*bps*
bib	111261	44156	39.69	3.17
book1	768771	340279	44.26	3.54
book2	610856	243092	39.80	3.18
geo	102400	63006	61.53	4.92
news	377109	173207	45.93	3.67
obj1	21504	14405	66.99	5.36
obj2	246814	119957	48.60	3.89
paper1	53161	24917	46.87	3.75
paper2	82199	35939	43.72	3.50
pic	513216	82136	16.00	1.28
progc	39611	18890	47.69	3.82
progl	71646	24649	34.40	2.75
progp	49379	17416	35.27	2.82
trans	93695	31235	33.34	2.67
all files	3145718	1237380	39.33	3.14

Table: Calgary Corpus encoded with vertical reading, byte mapping and a BWTS as preprocessing, using Huffman encoding for all counted runs, 8 bit per run.

Implementation

- written in Kotlin

Implementation

- written in Kotlin
- used libraries:
  - Kotlin Binary Streams (IOStreams for Kotlin)

Implementation

- written in Kotlin
- used libraries:
  - Kotlin Binary Streams (IOStreams for Kotlin)
  - LibDivSufSort (from Kanzi)

Implementation

- written in Kotlin
- used libraries:
  - Kotlin Binary Streams (IOStreams for Kotlin)
  - LibDivSufSort (from Kanzi)
  - miscellaneous
    - google/guava, junit5, staticlog ...

Implementation - Encoding

current bit $= 0$
run $= 1$

$$\begin{matrix} {\color{red}0}1100001 \\ 01100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ 01100001 \end{matrix} \quad (5)$$

Implementation - Encoding

current bit $= 0$
run $= 2$

$$\begin{matrix} 01100001 \\ \color{red}{0}1100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ 01100001 \end{matrix} \tag{5}$$

Implementation - Encoding

current bit $= 0$
run $= 3$

$$\begin{matrix} 01100001 \\ 01100010 \\ {\color{red}0}1110010 \\ 01100001 \\ 01100011 \\ 01100001 \end{matrix} \tag{5}$$

Implementation - Encoding

current bit $= 0$
run $= 4$

$$
\begin{array}{c}
01100001 \\
01100010 \\
01110010 \\
{\color{red}0}1100001 \\
01100011 \\
01100001
\end{array}
\tag{5}
$$

Implementation - Encoding

current bit = 0
run = 5

$$
\begin{array}{c}
01100001 \\
01100010 \\
01110010 \\
01100001 \\
{\color{red}0}1100011 \\
01100001
\end{array}
\tag{5}
$$

Implementation - Encoding

current bit = 0
run = 6

$$
\begin{array}{c}
01100001 \\
01100010 \\
01110010 \\
01100001 \\
01100011 \\
{\color{red}0}1100001
\end{array} \quad (5)
$$

Implementation - Encoding

current bit $= 0$
run $= 0$
result $= \{[6]\}$

$$\begin{matrix} 01100001 \\ 01100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ {\color{red}0}1100001 \end{matrix} \tag{5}$$

Implementation - Encoding

current bit = 0
run = 0
result = $\{[6],[0]\}$

$$\begin{gathered}0\textcolor{red}{1}100001\\01100010\\01110010\\01100001\\01100011\\01100001\end{gathered} \tag{5}$$

Implementation - Encoding

current bit = 1
run = 1
result = $\{[6],[0]\}$

$$\begin{gathered}0{\color{red}1}100001\\01100010\\01110010\\01100001\\01100011\\01100001\end{gathered} \tag{5}$$

Implementation - Encoding

current bit = 1
run = 2
result = {[6],[0]}

$$\begin{matrix} 01100001 \\ 01100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ 01100001 \end{matrix} \quad (5)$$

Implementation - Encoding

current bit = 1
run = 3
result =
{[6],[0,6],[0,6],[2,1,3],[6],[6],[1,2,1,1,1],[0,1,2,3]}

$$\begin{matrix} 01100001 \\ 01100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ 0110000{\color{red}1} \end{matrix} \tag{5}$$

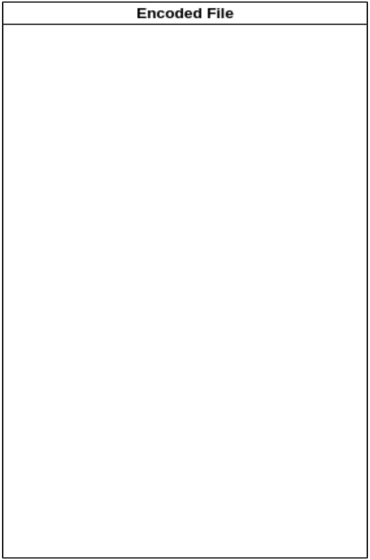
Implementation - Encoding

current bit = 1
run = 3
result =
{[6],[0,6],[0,6],[2,1,3],[6],[6],[1,2,1,1,1],[0,1,2,3]}

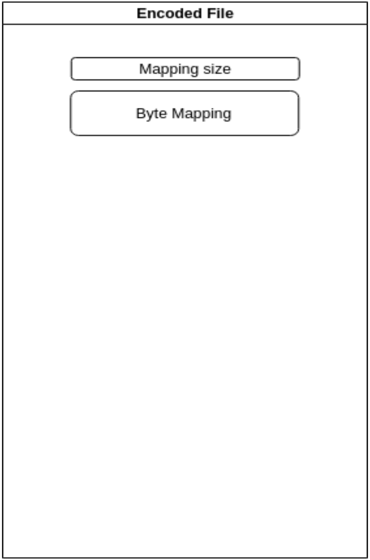
run	amount
6	5
1	4
2	3
0	3
3	2

$$\begin{array}{l} 01100001 \\ 01100010 \\ 01110010 \\ 01100001 \\ 01100011 \\ 0110000{\color{red}1} \end{array} \tag{5}$$

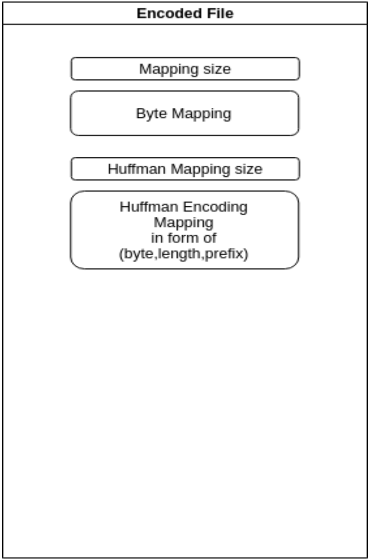
Implementation - Encoding



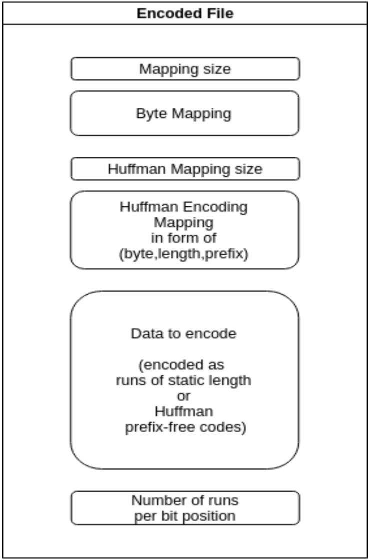
Implementation - Encoding



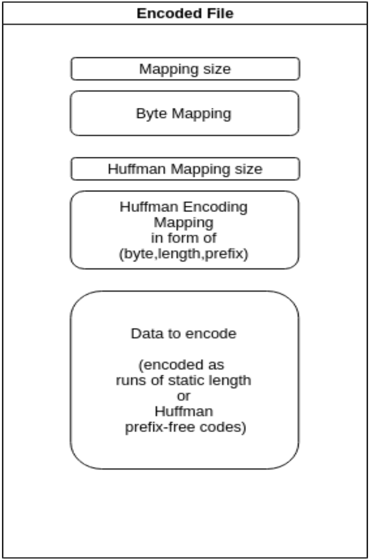
Implementation - Encoding



Implementation - Encoding



Implementation - Encoding



Implementation

DEMO

Evaluation and Discussion

Efficiency with current preprocessing steps:

$\approx 112\%$

Vertical Interpretation

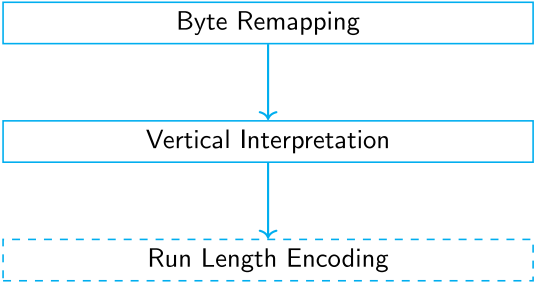
$\downarrow$

Run Length Encoding

Evaluation and Discussion

Efficiency with current preprocessing steps:

$\approx 94\%$



Evaluation and Discussion

Efficiency with current preprocessing steps:

$\approx 69\%$

Burrows-Wheeler-Transformation

↓

Byte Remapping

↓

Vertical Interpretation

↓

Run Length Encoding

Evaluation and Discussion

Efficiency with current preprocessing steps:

$\approx 58\%$

Burrows-Wheeler-Transformation

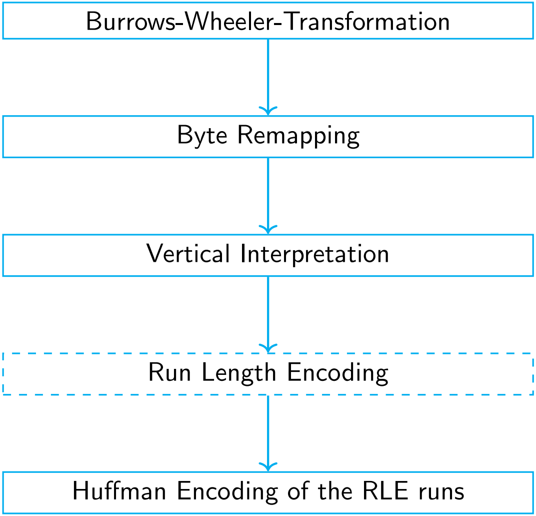
$\downarrow$

Byte-wise Run Length Encoding

Evaluation and Discussion

Efficiency with current preprocessing steps:

$\approx 39\%$



Evaluation and Discussion

file	size original	size encoded	ratio in %	*bps*
alice29.txt	152089	65445	43.03	3.44
asyoulik.txt	125179	59291	47.36	3.79
cp.html	24603	11073	45.01	3.60
fields.c	11150	5183	46.48	3.72
grammar.lsp	3721	1923	51.68	4.13
kennedy.xls	1029744	229823	22.32	1.79
lcet10.txt	426754	170593	39.97	3.20
plrabn12.txt	481861	215628	44.75	3.58
ptt5	513216	82136	16.01	1.28
sum	38240	19616	51.30	4.10
xargs.1	4227	2515	59.50	4.76
all files	2814880	867322	30.81	2.46

Table: Canterbury encoded, all preprocessing steps, using Huffman Encoding for all counted runs.

Evaluation and Discussion

method	size in bytes	compression	*bps*	time	
				encoding	decoding
uncompressed	3,145,718	100.0%	8.00		
compress 4.2.4	1,250,382	40.4%	3.24	0.039s	0.025s
modified vertical RLE	1,237,380	39.3%	3.14	6.840s	15.637s
gzip v1.10	1,021,720	32.4%	2.60	0.232s	0.025s
ZIP v3.0	1,019,783	32.4%	2.59	0.214s	0.022s
zstandard 1.4.2	887,004	28.1%	2.25	0.951s	0.011s
bzip2 v1.0.8	832,443	26.4%	2.11	0.191s	0.088s
brotli 1.0.7	826,638	26.3%	2.10	4.609s	0.015s
p7zip 16.02 (deflate)	794,098	26.1%	2.08	0.431s	0.045s
p7zip 16.02 (PPMd)	763,067	24.2%	1.93	0.345s	0.282s
ZPAQ v7.15	659.700	20.9%	1.67	7.452s	7.735s
paq8hp*	-	-	-	-	-
cmix v18	554,983	17.6%	1.41	$>$3h	$>$2h

Table: Benchmark on the Calgary Corpus.

References I

Robinson, A. H. and Cherry, C.,
Results of a prototype television bandwidth compression scheme, volume 55,
Proceedings of the IEEE 3, 1967.

Lehotay-Kéry, P. and Kiss, A.,
GenPress: A novel dictionary based method to compress DNA data of various species, pages 385–394,
Intelligent Information and Database Systems, Springer, 2019.

Al-Okaily, A., Almarri, B., Yami, S. A., and Huang, C.,
Toward a better compression for DNA sequences using Huffman encoding,
volume 24, page 280–288,
J. Comput. Biol., 2017.

Robinson, A. H. and Hunter, R.,
International digital facsimile coding standards,
Proceedings of the IEEE 68, 1980.

References II

Kolmogorov, A. N.,
On tables of random numbers, page 369–376,
Sankhyā: The Indian Journal of Statistics, Series A, 1963,
Academy of Science USSR.