Information Theory. 4th Chapter Slides

Boris Kudryashov

ITMO University

December 26, 2016

Agenda

1. Monotone codes
2. Interval coding
3. Sliding dictionary method (LZ-77)
4. LZW Algorithm (LZ-78)
5. Prediction by Partial Matching
6. Archivers characteristics

Monotone codes

- for Gallagher-Van Voorhees code

$$\alpha^T + \alpha^{T+1} \leq 1 < \alpha^{T-1} + \alpha^T. \qquad (1)$$

- for monotone code

$$\mathrm{mon}(i) = \left(\mathrm{unar}\left(\left|\mathrm{bin}'(i)\right| + 1\right), \mathrm{bin}'(i)\right). \qquad (2)$$

- Levenshtein codeword

$$\mathrm{lev}(21) = (1110)(0)(00)(0101) = 11100000101.$$

- Elias code

$$\mathrm{elias}(i) = (\mathrm{unar}(\ |\mathrm{bin}'(|\mathrm{bin}'(i)|)|+2\ ),\ \mathrm{bin}'(|\mathrm{bin}'(i)|),\ \mathrm{bin}'(i)\ ) \quad (4.1.4)$$

- $$\text{elias}(21) = (1110)(00)(0101) = 1110000101.$$
- Elias codeword length for arbitrary i

$$l_i = \begin{cases} 1, & i = 1, \\ \lfloor \log i \rfloor + 2 \lfloor \log \lfloor \log i \rfloor \rfloor + 2, & i > 1, \end{cases} \tag{3}$$

- satisfies

$$l_i \leq \log i + 2 \log(1 + \log i) + 2, \quad i = 1, 2, \ldots \quad . \tag{4}$$

Table: Monotone codes table

i	Monotone code		Levenshtein code		Elias code	
	$\mathrm{mon}(i)$	l_i	$\mathrm{lev}(i)$	l_i	$\mathrm{elias}(i)$	l_i
1	0	1	0	1	0	1
2	100	3	100	3	100	3
3	101	3	101	3	101	3
4	11000	5	110000	6	110000	6
...	...	5	...	6	...	6
7	11011	5	110011	6	110011	6
8	1110000	7	1101000	7	1101000	7
...	...	7	...	7	...	7
2^{1023}	$1^{1023}00^{1023}$	2047	$1^4010011^90^{1023}$	1041	$1^{10}01^910^{1023}$	1043

Theorem

Let random variable i take values from natural numbers and it's probability distribution satisfies: $p_i \leq p_j$ if $i > j$. Then, average length of Elias codewords $\bar{l}$ satisfies

$$\bar{l} \leq H(1 + o(H)),$$

where H is entropy of i, and $o(H) \to 0$ when $H \to \infty$.

Proof.

- $$p_i \leq \frac{1}{i} \quad , \quad i \leq \frac{1}{p_i}, \quad i = 1, 2, \ldots \quad . \tag{5}$$
- $$\begin{aligned} \bar{l} &= \sum_{i=1}^{\infty} l_i p_i \leq \\ &\leq \sum_{i=1}^{\infty} p_i (\log i + 2 \log \log i + 2) \leq \\ &\leq \sum_{i=1}^{\infty} p_i \log \frac{1}{p_i} + 2 \sum_{i=1}^{\infty} p_i \log \left(1 + \log \frac{1}{p_i}\right) + 2 \leq \\ &\leq H + 2 \log(1 + H) + 2. \end{aligned} \tag{6}$$

Monotone codes

$$IF\_WE\_CANNOT\_DO\_AS\_WE\_WOULD\_WE\_SHOULD$$

$$\_DO\_AS\_WE\_CAN$$

$$(74, 72, 35, 88, 73, 3, 72, 71, 80, 1, 81, 86, 6,$$

76, 4, 3, 6, 86, 3, 10, 10, 3, 3, 6, 87, 83, 9, 6, 6, 7, 3, 8, 81,
9, 9, 9, 9, 7, 2, 5, 3, 10, 8, 3, 10, 10, 3, 13, 6, 13)

Theorem

Let $H(X)$ be the entropy of one-dimensional distribution of discrete stationary source. Average code rate for interval coding in composition with Elias code satisfies:

$$\bar{R} \leq H(X)(1 + o(H(X)),$$

where $o(H) \to 0$ when $H \to \infty$.

Proof.

- $$l_i(a) \leq \log r_i(a) + 2\log(1 + \log r_i(a)) + 2. \qquad (7)$$
- $$\bar{\bar{l}}(a) \leq \log \bar{r}(a) + 2\log(1 + \log \bar{r}(a)) + 2. \qquad (8)$$
- $$\bar{r}(a) \leq \frac{1}{p(a)}. \qquad (9)$$
- $$\bar{R} = \sum_a p(a)\bar{l}(a) \leq H(X) + 2\log(1 + H(X)) + 2.$$

Lemma

Let random variable i be taking values from natural number $\{1, 2, ...\}$ according to distribution $\{p_i, i = 1, 2, ...\}$. Then holds

$$\mathrm{M}[i] = \sum_{j=1}^{\infty} \mathrm{P}(i \geq j). \tag{10}$$

- $$r = \min\{k \geq 1 : x_0 = x_k = a\}. \qquad (11)$$

- $$q_a(r) = \mathrm{P}(x_1, ..., x_{k-1} \neq a, x_k = a | x_0 = a), \qquad (12)$$

- $$\bar{r}_a = \sum_{r=1}^{\infty} r q_a(r) \quad . \qquad (13)$$

Lemma

(Katz Lemma). For discrete stationary source such that $p(a) > 0$ *holds:*

$$\bar{r}_a p(a) = \mathrm{P}(x_n = a \quad \textit{at least one} \quad n, \quad n = 0, 1, ...). \tag{14}$$

for an ergodic source:

$$\bar{r}(a) = \frac{1}{p(a)}. \tag{15}$$

- $$r_a p(a) = p(a) \sum_{t=1}^{\infty} r q_a(r) = p(a) \sum_{t=1}^{\infty} Q_a(t), \tag{16}$$
- $$Q_a(t) = \sum_{j=t}^{\infty} q_a(j) = \mathrm{P}(r \geq t).$$
- $$Q_a(t) = \mathrm{P}(x_1, \ldots, x_t \neq a | x_0 = a).$$

- $$r_a p(a) = \mathrm{P}(x_0 = a) \sum_{t=1}^{\infty} \mathrm{P}(x_1, \ldots x_t \neq a | x_0 = a) =$$
$$= \sum_{t=1}^{\infty} \mathrm{P}(x_0 = a, x_1, \ldots x_t \neq a).$$

- $$r_a p_a(t) = \sum_{t=1}^{\infty} \tilde{\mathrm{P}}(x_0, \ldots, x_{t-1} \neq a, x_t = a) =$$
$$= \tilde{\mathrm{P}}(x_t = a \quad \text{for at least one} \quad t, \quad t = 1, 2, \ldots).$$

Sliding dictionary method (LZ-77)

Alphabet: $X = \{0, 1, ..., M-1\}$, "window" length W;
Input: sequence length n, source sequence $\boldsymbol{x} = \boldsymbol{x}_1^n$;;
Output: Codeword $\boldsymbol{c}$ for $\boldsymbol{x}$;

`Init:` $N = 0$, $\boldsymbol{c}$ `is ''empty'' codeword;;`

while $N < n$ **do**
find max l such that $\boldsymbol{x}_{N+1}^{N+l} = \boldsymbol{x}_{N-d+1}^{N-d+l}$;
for some $d \in \{1, ..., W\}$; ;

if $l > 0$ **then** ;
(sequence is found) ;

- Concatenate to a codeword $\boldsymbol{c}$:;
  - flag 1;;
  - distance to d as a binary sequence of length $\lceil \log W \rceil$; ;
  - match length l as a non-uniform prefix code;
- $N \leftarrow N + l$;;

else ;
(sequence not found) ;

- Concatenate to a codeword $\boldsymbol{c}$:;
  - flag 0; ;
  - new letter x_{N+1} as binary sequence of length $\lceil \log M \rceil$;
- $N \leftarrow N + 1$;;

end

Figure: LZ-77 Algorithm

Sliding dictionary method (LZ-77)

Table: Sliding dictionary method (LZ-77)

Step	Flag	Letter sequence	d	l	Code sequence	Bits
0	0	I	-	0	0 bin(I)=0	9
1	0	F	-	0	0 bin(F)	9
2	0	_	-	0	0 bin(_)	9
3	0	W	-	0	0 bin(W)	9
4	0	E	-	0	0 bin(E)	9
5	1	_	2(5)	1	1 010 0	5
6	0	C	-	0	0 bin(C)	9
7	0	A	-	0	0 bin(A)	9
8	0	N	-	0	0 bin(N)	9
9	1	N	0(9)	1	1 0000 0	6
10	0	O	-	0	0 bin(O)	9
11	0	T	-	0	0 bin(T)	9

LZFG Algorithm

Alphabet, $X = \{0, 1, ..., M-1\}$, window length W;
Input: Sequence length n;;
Source sequence $\boldsymbol{x} = \boldsymbol{x}_1^n$;;
Output: codeword $\boldsymbol{c}$ для $\boldsymbol{x}$;

`Init:` ;
$N = 0$, $\boldsymbol{c}$ - ‘‘`empty`’’ `codeword;`;

while $N < n$ **do**

Find max l in range $\{3, ...17\}$ such that $\boldsymbol{x}_{N+1}^{N+l} = \boldsymbol{x}_{N-d+1}^{N-d+l}$ for some $d \in \{1, ..., W\}$; ;

if $l > 2$ **then**

- Write number l to a codeword $\boldsymbol{c}$ as a binary sequence of length 4 $([3, \ldots 17] <-> [0001, \ldots 1111]$ distance to sample d as a binary sequence pf length $\lceil \log W \rceil$; ;
- $N \leftarrow N + l$;;

;
else ;
$(l \leq 2)$;

- To a codeword $\boldsymbol{c}$ concatenate:;
  - sequence 0000; ;
  - number of letters $L \in \{1...16\}$, at input(without coding); $[1, ..., 16] <-> [0000, \ldots, 1111]$;
  - L letters of source without coding as a binary sequence of length $\lceil \log M \rceil$;
- $N \leftarrow N + L$;;

end

Figure: Algorithm LZFG

Sliding dictionary method (LZ-77)

Table: Example of LZFG

Step	Length of match	Distance to pattern	Number of "new" letters	Code characters	Transmitted letters	Costs (bit)
1	0	–	16	0000 1111 bin(I..._)	IF_WE_ CANNOT _DO_	8+8×16 =136
2	1(<3)	–	2	0000 0001 bin(AS)	AS	8+8×2 =24
3	4	15(18)	–	0010 01111	_WE_	4+5=9
4	1(<3)	–	5	0000 0100 bin(WOULD)	WOULD	8+5×8 =48
5	4	8(27)	–	0010 01000	_WE_	4+5 =9
6	– –	– –	2	0000 0001 bin(SH)	SH	8+2×8 =24
7	5	9(33)	–	0011 001001	OULD_	4+6 =10
8	9	24(38)	– –	0111 011000	DO_AS _WE_	4+6=10
9	3	40(47)	–	0001 101000	CAN	4+6=10
In total						280

Sliding dictionary method (LZ-77)

Source alpabet $X = \{0, 1, ..., M-1\}$; ;
Input: Sequence length n;;
Source sequence $\boldsymbol{x} = \boldsymbol{x}_1^n$;;
Output: codeword $\boldsymbol{c}$ для $\boldsymbol{x}$;

`Init:` $N = 0$; $\boldsymbol{c}$ is "empty" word; ;
The dictionary consists of M words of length 1, i.e. of letters of X. $c = M$. ;

while $N < n$ **do**

- find max l such that $\boldsymbol{x}_{N+1}^{N+l}$ matches j-th word in dictionary for some $j < c$;
  (at first step $j = c$ is allowed).
- Number of word j is concatenated to a word as a binary sequence;
  of length $\lceil \log(c-1) \rceil$ (For first step sequence of length $\lceil \log c \rceil = \lceil \log M \rceil$ is allowed).
- New dictionary of length $l+1$ is added to a dictionary ;
  like $\boldsymbol{x}_{N+1}^{N+l+1} = (\boldsymbol{x}_{N+1}^{N+l}, x_{N+l+1})$.
- $N \leftarrow N + l$;;
- $c \leftarrow c + 1$;;

end

Figure: LZW Algorithm

Sliding dictionary method (LZ-77)

Table: LZW usage example

Step	Dictionary	Word number	Code characters	Costs (bit)
0	esc	–	–	–
1	I	0	bin(I)	0+8=8
2	F	0	bin(F)	log(1)+8=8
3	_	0	0bin(_)	$\lceil \log(2) \rceil + 8 = 9$
4	W	0	00bin(W)	$\lceil \log(3) \rceil + 8 = 10$
5	E	0	00bin(E)	$\lceil \log(4) \rceil + 8 = 10$
6	_C	3	011	3
7	C	0	000bin(C)	3+8=11
8	A	0	000bin(A)	3+8=11
9	N	0	000bin(N)	3+8=11
10	NO	0	000bin(N)	4+8=12
11	O	0	0000bin(O)	4+8=12
12	T	0	0000bin(T)	4+8=12
13	_D	3	0011	4
14	D	0	0000bin(D)	4+8=12
15	O_	11	1011	4=4

Sliding dictionary method (LZ-77)

Table: LZW usage example

Step	Dictionary	Word number	Code characters	Costs (bit)
0	esc	–	–	–
16	_A	3	0011	4
17	AS	8	1000	4
18	S	0	00000bin(S)	5+8
19	_W	3	00011	5
20	WE	4	00100	5
21	E_	5	00101	5
22	_WO	19	10011	5
23	OU	11	01011	5
24	U	0	00000bin(U)	5+8=13
25	L	0	00000bin(L)	5+8=13
26	D_	14	01110	5
27	_WE	19	10011	5
28	E_S	21	10101	5
29	SH	18	10010	5

Sliding dictionary method (LZ-77)

Table: LZW usage example

Step	Dictionary	Word number	Code characters	Costs (bit)
30	H	0	00000bin(H)	5+8=13
31	OUL	23	10111	5
32	LD	25	11001	5
33	D_D	26	11010	5
4	DO	14	001110	6
35	O_A	15	001111	6
36	AS_	17	010001	6
37	_WE_	27	011011	6
38	_CA	6	000110	6
39	AN	8	001000	6
40	N	9	001001	6
Total				291

Prediction by Partial Matching

Alphabet $X = \{0, 1, ..., M-1\}$, max length of condext D ;

Input: Sequence length n, source sequence $\boldsymbol{x} = \boldsymbol{x}_1^n$;;

Output: Codeword $\boldsymbol{c}$ for $\boldsymbol{x}$;

`Init:` $t = 0$; $\boldsymbol{c}$ – "empty" word; ;

while $t < n$ **do**

find max d such that $\hat{p}_t(\boldsymbol{x}_{t-d+1}^t) > 0$.;
Choose context $\boldsymbol{s} = \boldsymbol{x}_{t-d+1}^t$.;
if $\hat{p}_t(x_{t+1}|\boldsymbol{s}) > 0$ **then** ;
;
encode x_{t+1} according to c $\hat{p}_t(x|\boldsymbol{s})$. ;
else ;
;
while $\hat{p}_t(x_{t+1}|\boldsymbol{s}) = 0$ **do**
Encode *esc* according to $\hat{p}_t(esc|\boldsymbol{s})$.;
`Decrease condets length:` ;
$d \leftarrow d - 1$, $\boldsymbol{s} = \boldsymbol{x}_{t-d+1}^t$.;
if $d > 0$ **then** ;
;
Encode x_{t+1} according to $\hat{p}_t(x|\boldsymbol{s})$.;

Prediction by Partial Matching

- $$\hat{p}_t(a|\boldsymbol{s}) = \frac{\tau_t(\boldsymbol{s}, a)}{\tau_t(\boldsymbol{s}) + 1}, \quad \tau_t(\boldsymbol{s}, a) > 0, \tag{17}$$
$$\hat{p}_t(esc|\boldsymbol{s}) = \frac{1}{\tau_t(\boldsymbol{s}) + 1}, \tag{18}$$

  where $\tau_t(\boldsymbol{s})$ and $\tau_t(\boldsymbol{s}, a)$ denote number of sequences $\boldsymbol{s}$ and $(\boldsymbol{s}, a)$ respectively, which contain in sequence of length t.

- $$\hat{p}_t(a|\boldsymbol{s}) = \frac{\tau_t(\boldsymbol{s}, a) - 1/2}{\tau_t(\boldsymbol{s})}, \quad \tau_t(\boldsymbol{s}, a) > 0, \tag{19}$$
$$\hat{p}_t(esc|\boldsymbol{s}) = \frac{M_t(\boldsymbol{s})}{2\tau_t(\boldsymbol{s})}, \tag{20}$$

  where $M_t(\boldsymbol{s})$ is number of different letters, which appear in sequence of length followed by the context $\boldsymbol{s}$.

Table: PPMA usage example

Шаг	Letter	Context s	$\tau_t(s)$	PPMA	
				$\hat{p}_t(esc\|s)$	$\hat{p}_t(a\|s)$
1	I	#	0	1	1/256
2	F	#	1	1/2	1/255
3	_	#	2	1/3	1/254
4	W	#	3	1/4	1/253
5	E	#	4	1/5	1/252
6	_	#	5		1/6
7	C	_	1,6	1/2×1/6'	1/251
8	A	#	7	1/8	1/250
9	N	#	8	1/9	1/249
10	N	#	9		1/10
11	O	N	1,10	1/2×1/9'	1/248
12	T	#	11	1/12	1/247
13	_	#	12		2/13
14	D	_	2,13	1/3×1/12'	1/246
15	O	#	14		1/15
16	_	O	1,15	1/2	3/15'
17	A	_	3,16	1/4	1/14'
18	S	A	1,17	1/2×1/16'	1/245
19	_	#	18		4/19
20	W	_	4		1/5
21	E	_W	1		1/2
22	_	_WE	1		1/2
23	W	_WE_	1,1,1,5	1/2×1'×1'	2/5'
24	O	_W	2,2,23	1/3×1'	2/22'
25	U	O	2,24	1/3×1/18'	1/244
26	L	#	25	1/26	1/243
27	D	#	26		1/27
28	_	D	1,27	1/2	6/25'
29	W	_	6		3/7
30	E	_W	3		2/4
31	_	_WE	2		2/3
32	S	_WE_	2,2,2,7,31	1/3×1'×1'×1/3'	1/23'
33	H	S	1,32	1/2×1/25'	1/242
34	O	#	33		3/34
35	U	O	3		1/4
36	L	OU	1		1/2
37	D	OUL	1		1/2
38	_	OULD	1		1/2
39	D	OULD	1,1,1,8	1/2×1'×1'×1'	1/4'

Prediction by Partial Matching

Table: PPMD usage example

LLlar	Letter	Context s	$\tau_t(s)$	$M_t(s)$	PPMD	
					$\hat{p}_t(esc\|s)$	$\hat{p}_t(a\|s)$
1	I	#	0	0	1	1/256
2	F	#	1	1	1/2	1/255
3	_	#	2	2	2/4	1/254
4	W	#	3	3	3/6	1/253
5	E	#	4	4	4/8	1/252
6	_	#	5	5		1/10
7	C	_	1,6	1,5	1/2×4/8'	1/251
8	A	#	7	6	6/14	1/250
9	N	#	8	7	7/16	1/249
10	N	#	9	8		1/18
11	O	N	1,10	1,8	1/2×8/18'	1/248
12	T	#	11	9	9/22	1/247
13	_	#	12	10		3/24
14	D	_	2,13	2,10	2/4×8/22'	1/246
15	O	#	14	11		1/28
16	_	O	1,15	1,11	1/2	5/28'
17	A	_	3,16	3,11	3/6	1/26'
18	S	A	1,17	1,11	1/2×10/30'	1/245
19	_	#	18	12		7/36
20	W	_	4	4		1/8
21	E	_W	1	1		1/2
22	_	_WE	1	1		1/2
23	W	_WE_	1,1,1,5	1,1,1,4	1/2×1'×1'	3/8'
24	O	_W	2,2,23	1,1,12	1/4×1'	3/42'
25	U	O	2,24	2,12	2/4×10/34'	1/244
26	L	#	25	13	13/50	1/243
27	D	#	26	14		1/52
28	_	D	1,27	1,14	1/2	11/52'
29	W	_	6	4		5/12
30	E	_W	3	2		3/6
31	_	_WE	2	1		3/4
32	S	_WE_	2,2,2,7,31	2,2,2,4,14	2/4×1'×1'×2/4'	1/48'
33	H	S	1,32	1,14	1/2×13/50	1/242
34	O	#	33	15		5/66
35	U	O	3	3		1/6
36	L	OU	1	1		1/2
37	D	OUL	1	1		1/2
38	_	OULD	1	1		1/2
39	D	OULD	1,1,1,1,8	1,1,1,1,5	1/2×1'×1'×1'	1/8

Prediction by Partial Matching

0	A	NIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_	C
1	A	NNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_	C
2	A	S_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO	_
3	A	S_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO	_
4	C	ANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE	_
5	C	ANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE	_
6	D	O_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD	_
7	D	O_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT	_
8	D	_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOU	L
9	D	_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOU	L
10	E	_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_	W
11	E	_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_	W
12	E	_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_	W
13	E	_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_	W
14	F	_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CAN	I*
15	H	OULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_	S
16	I	F_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CA	N
17	L	D_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHO	U
18	L	D_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WO	U
19	N	IF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_C	A
20	N	NOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_C	A
21	N	OT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CA	N
22	O	T_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CAN	N
23	O	ULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_S	H
24	O	ULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_	W
25	O	_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_	D
26	O	_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_	D
27	S	HOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE	_
28	S	_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_	A
29	S	_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_	A
30	T	_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANN	O
31	U	LD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SH	O
32	U	LD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_W	O
33	W	E_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS	_
34	W	E_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF	_
35	W	E_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD	_
36	W	E_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS	_
37	W	OULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE	_
38	_	AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_D	O
38	_	AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_D	O

Prediction by Partial Matching

Letter	number	bits
esc	15	log(50)
0	23	log(50–15+1)=log(36)
1	1	log(50–15–23+1)=log(13)
2	4	log(12)
3	1	log(8)
4	2	log(7)
5	1	log(5)
6	1	log(4)
7	0	log(3)
8	0	log(3)
9	1	log(3)
10	1	log(2)
Total $\lceil 33, 98 \rceil + 1$=35 bit		

Prediction by Partial Matching

i	Sequence	$y_i(y_{i,\max})$
0	Ĉ?_?????L?W???ISNU?A???H?D????O?????????E??T??F???	0
1	CC≙?????L?W???ISNU?A???H?D?Ĥ??O?????????E??T??F???	11(29)
2	CC______L̂?W???ISNU?A???H?D?_??O?????????E??T??F???	0
3	CC______LLŴ???ISNU?A???H?D?_??O?????????E??T??F???	5(24)
4	CC______LLWWWWÎSNU?A???HWD?_??O?????????E??T??F???	0
5	CC______LLWWWWIŜNU?A???HWD?_??O?????????E??T??F???	20(23)
6	CC______LLWWWWISÑU?A??HWD?_??O?????????E??T?SF???	3(22)
7	CC______LLWWWWISNÛ?A?N?HWD?_??O?????????E??T?SF???	0
8	CC______LLWWWWISNUUÂ?N?HWD?_?̂?O?????????E??T?SF???	3(19)
9	CC______LLWWWWISNUUAAÑ?HWD?_A?O?????????E??T?SF???	0
10	CC______LLWWWWISNUUAANNĤWD?_A?O?????????E??T?SF???	0
11	CC______LLWWWWISNUUAANNHŴD?_A?O?????????E??T?SF???	0
12	CC______LLWWWWISNUUAANNHWD̂?_A?O?????????E?Ĥ?T?SF???	12(16)
13	CC______LLWWWWISNUUAANNHWDD≙A?O???̂??????E?DT?SF???	4(15)
14	CC______LLWWWWISNUUAANNHWDD_Â?O??_??????E?DT?SF???	0
15	CC______LLWWWWISNUUAANNHWDD_AAÔ??_?????̂?E?DT?SF???	5(11)
16	CC______LLWWWWISNUUAANNHWDD_AAOOO≙????O?E?DT?SF???	0
17	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OÔE?DT?SF???	0
18	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OOÊ?DT?̂SF???	1(4)
19	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OOEEĐTESF̂???	1(3)
20	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OOEEDT̂ESFD??	0
21	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OOEEDTÊSFD?̂	2(2)
22	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OOEEDTEŜFD?E	1(1)
23	CC______LLWWWWISNUUAANNHWDD_AAOOO_____OOEEDTESFDSE	

Prediction by Partial Matching

- $$p_i(a) = \begin{cases} \frac{1}{2}, & a = 0, \\ \frac{1}{2}\frac{1}{y_{i,\max}}, & a \neq 0 \end{cases}$$
- $$22 + \left\lceil -\log\left(\frac{1}{29 \cdot 24 \cdot 23 \cdot 22 \cdot 19 \cdot 16 \cdot 15 \cdot 11 \cdot 4 \cdot 3 \cdot 2 \cdot 1}\right)\right\rceil =$$
- $$I = 6 + 120 + 42 + 61 + 6 = 235 \quad \text{bit},$$

Archivers characteristics

Algorithm	Section	Costs (bits)
Transmitting without coding		400
Two-pass coding, Huffman coding	??	302
Enumerative coding	??	283
Adaptive arithmetic coding (A algorithm)	??	293
Adaptive arithmetic coding (D algorithm)	??	283
LZ77 algorithm	??	280
LZFG algorithm	??	299
LZW algorithm	??	291
PPMA algorithm	??	250
PPMD algorithm	??	**232**
BW Transform + book stack	??	255
BW transform + distance coding	??	**235**

Archivers characteristics

Table: Archivers comparison on Calgary Corpus

Archiver (author)	How it works	Rate (bit/byte)
RK 1.04.01 (M. Taylor)	LZ, PPMZ	1.8226
SBC 0.968 (S. J. Mäkinen)	Burrows-Wheeler transform	1.8846
RAR(Win32)3.00b5 (E.Rashol)	LZ77 variant + Huffman	1.9244
ACB 2.00c (G.Buyanovsky)	Associative Coding Algorithm	1.9546
PPMD vE (D. Shkarin)	PPM	2.0153 2.0153
PKZIP 2.6.02 Win95 (PKWARE)	LZH, LZW	2.6062