20SK – Signals and Codes

Lecture 09 – Dictionary coders (2018/11/26)

Topics discussed:

- Dictionary-based compression schemes in general
- RLE
- Lempel-Ziv (LZ77)source coding
- LZW

The relevant literature is [1,2] for LZ77 and [3] for LZW. For more insight into data compression methods see also books by David Salomon [4,5]. Some relevant information can be also found in Wikipedia and other online sources.

Resources

[1] Shor, P.: *Lempel-Zip notes* [online]. MIT, 2005. Available from: http://www-math.mit.edu/~shor/PAM/lempel_ziv_notes.pdf

[2] Ziv, J., Lempel, A.: *A Universal Algorithm for Sequential Data Compression*. IEEE Transactions on Information Theory 23 (3), 1977, pp. 337–343. doi:10.1109/TIT.1977.1055714.

[3] Welch, T.: *A Technique for High-Performance Data Compression*. Computer 17 (6), 1984, pp. 8–19. doi: 10.1109/MC.1984.1659158.

[4] Salomon, D.: *Concise Introduction to Data Compression*. London: Springer Verlag, 2008, 311 pp.

[5] Salomon D., Motta G.: *Handbook of Data Compression*. London: Springer Verlag, 2010, 1360 pp. doi: 10.1007/10.1007/978-1-84882-903-9

Huffman Coding

A .... $p_A = 1/3$
B .... $p_B = 1/3$
C .... $p_C = 1/3$

$H(X) = -\log_2 \frac{1}{3} = 1.585$

Example: Huffman code

A → 0
B → 10
C → 11

ABBCBA → 010101100
⇒ 10 bits, 6 symbols
$E(H(X)) = \frac{10}{6} = 1.67$ bits/symbol

ABCABC → 010110101

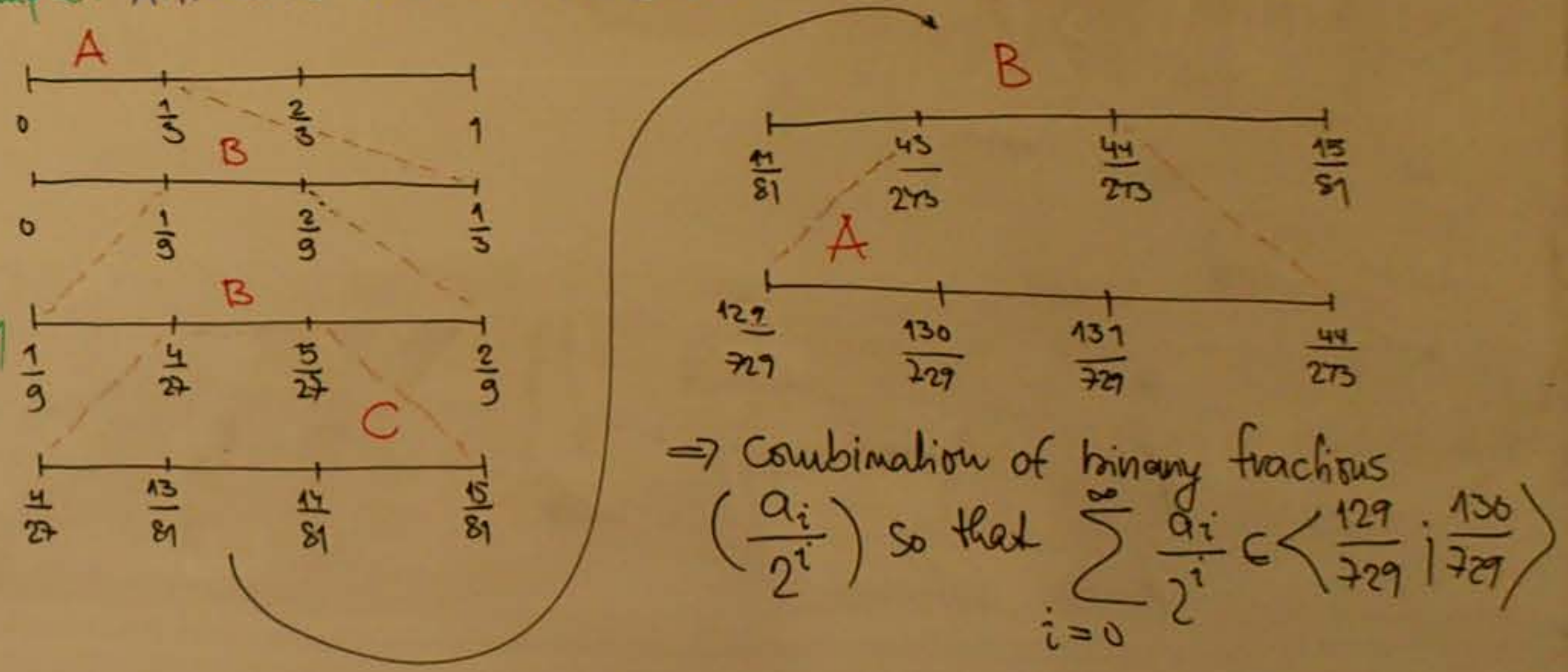
works well only for $p_i = 2^{-k}$
bad for p_i's like 75%, 37,5%, ...
$\frac{3}{4}$ $\frac{3}{8}, \frac{5}{8}, \frac{7}{8}$

ABBCB → 01010110
⇒ 9 bits, 5 symbols
$E(H(X)) = \frac{9}{5} = 1.8$ bits/symbol

Arithmetic Coding

idea: represent the source as real number from (0,1)
by successive subdivision of interval (0,1) according to symbol probabilities

Example: Arithmetic code for ABBCBA



⇒ Combination of binary fractions $\left(\frac{a_i}{2^i}\right)$ so that $\sum_{i=0}^{\infty} \frac{a_i}{2^i} \in \left\langle \frac{129}{729}; \frac{130}{729} \right\rangle$

continued ....

$$\frac{129}{729} < m < \frac{130}{729}$$

$$m \uparrow \frac{a_i}{2^i}$$

$$0.17695 < m < 0.17852$$

$i = 7$ ... no match for m : $\frac{22}{128} < 0.176$ and $\frac{23}{128} > 0.179$

$i = 8$ ... no match for m : $\frac{45}{256} < 0.176$

$i = 9$ .... $m = \frac{91}{512} - \frac{1}{512} + \frac{1}{256} + \frac{1}{64} + \frac{1}{32} + \frac{1}{8}$

$m = 0.001011011$

ABBCBA ⟶ 001011011

⇒ 9 bits, 6 chars

$E(M(x)) = \frac{9}{6} = 1.5$ bits/symbol

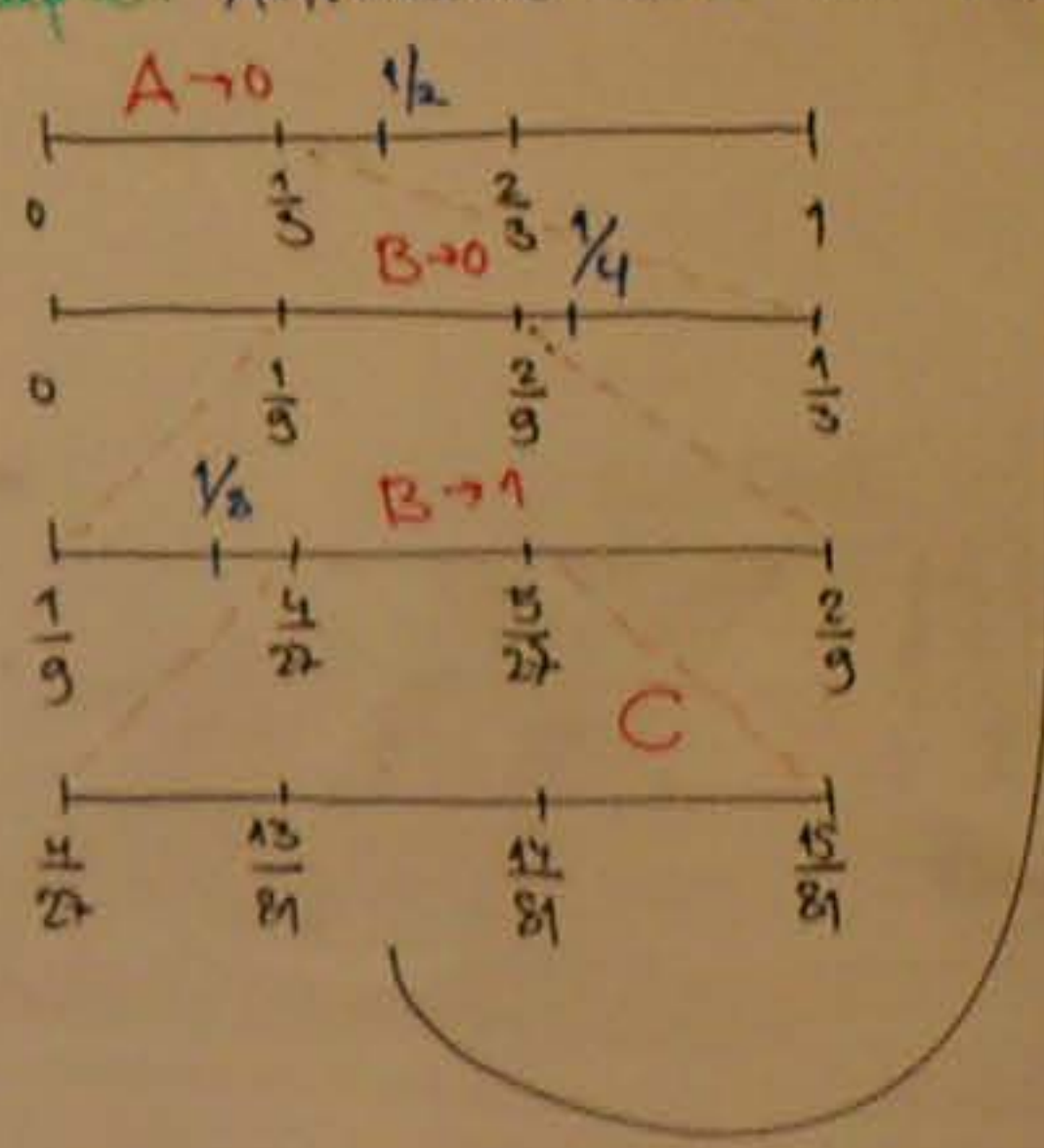
1) arbitrary precision arithmetic
2) needs to read the whole message beforehand

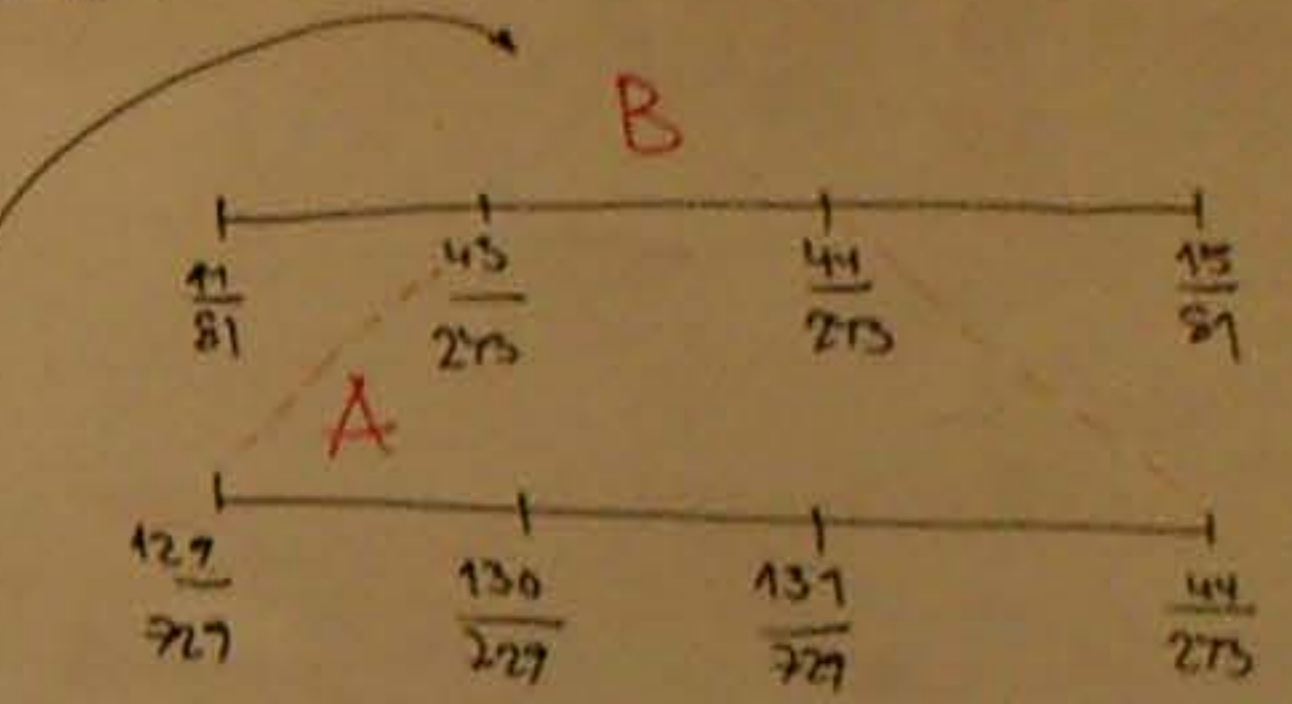
⇒ better algorithms exist for efficient computation of m

Arithmetic Coding

idea: represent the source as real number from (0,1) by successive subdivision of interval (0,1) according to symbol probabilities

Example: Arithmetic code for ABBCBA

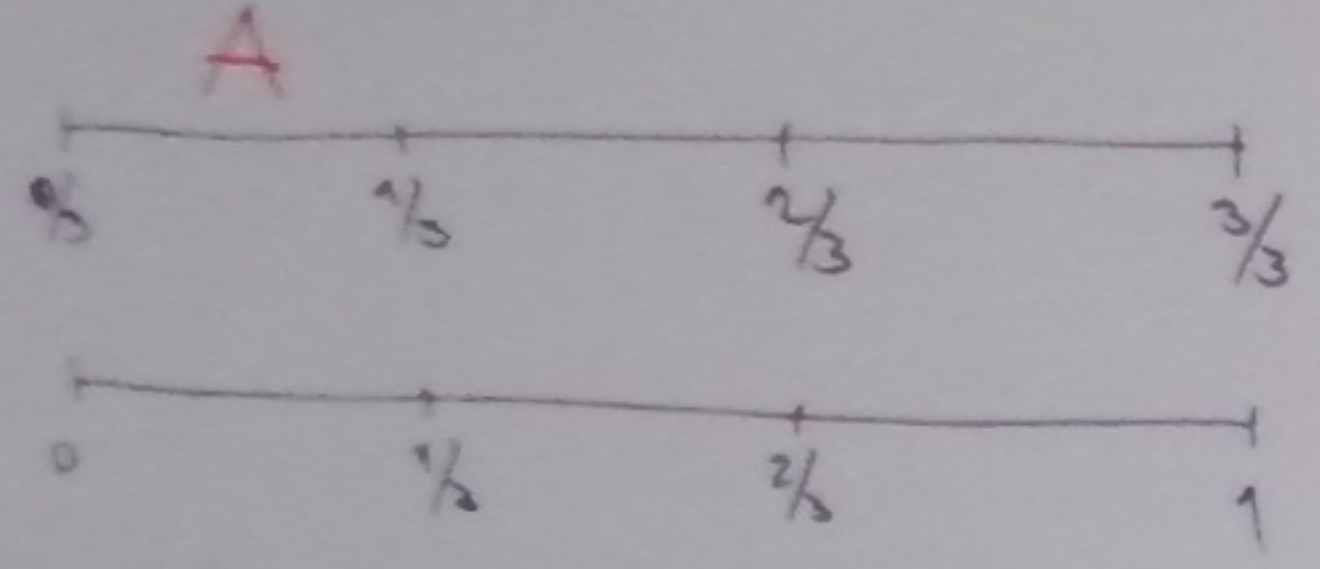




⇒ Combination of binary fractions $\left(\frac{a_i}{2^i}\right)$ so that $\sum_{i=1}^{\infty} \frac{a_i}{2^i} \in \left\langle \frac{129}{729}, \frac{130}{729} \right\rangle$

ARITHMETIC CODING

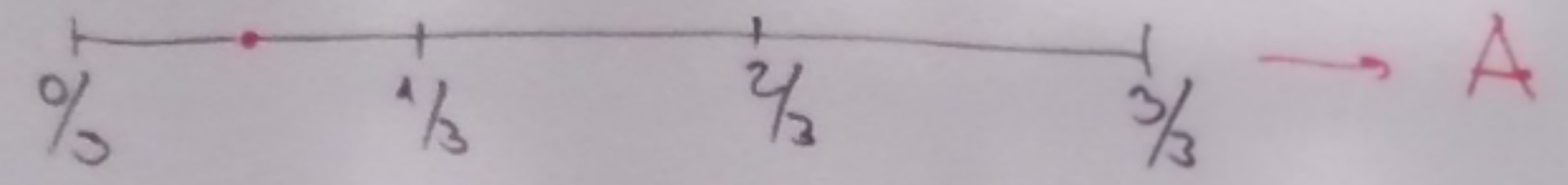
- re-normalisation



we will always use $\langle 0,1 \rangle$ to assign intervals for letters and scale afterwards

DECODING

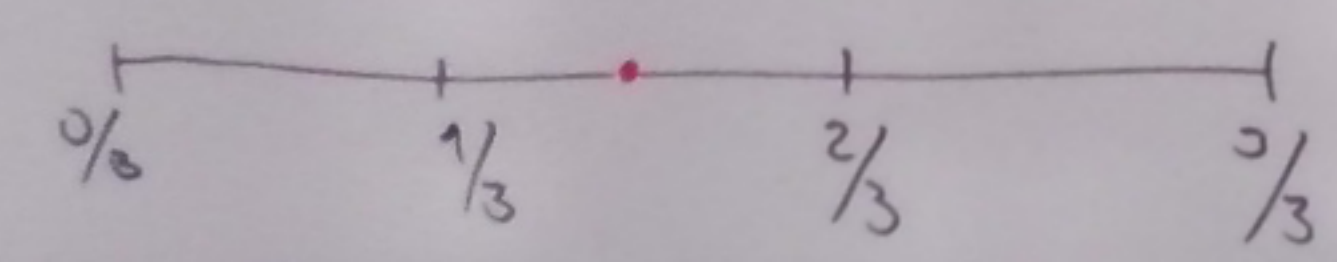
message: 0.001011011 $= \frac{91}{512}$ 0.177...

①  → A

$\left(\frac{91}{512} - \text{lower bound of interval}\right) / \text{length of interval}$

$\Rightarrow$ a number from $\langle 0,1)$ $= \frac{273}{512}$ 0.53...

$\frac{91}{512} / \frac{1}{3} = \frac{3 \cdot 91}{512} = \frac{273}{512}$

②  → B

$\left(\frac{273}{512} - \frac{1}{3}\right) \cdot 3 = \frac{3 \cdot 223 - 512}{512} = \frac{307}{512}$

ABBCBA EOM

ABBCBAAAAA....

A ... 0
B ... 1/3
C ... 2/3

Dictionary methods

- encode „phrases", not single symbols into code words
- output words / symbols are equipotable

Ex: Encoding English text

'␣the␣' → 1
'␣a␣' → 2
'␣is␣' → 3
⋮

- based on „transformation table" → DICTIONARY
- table is dynamic → coding improves with the number of symbols processed (usually!)

Historical remark: RLE (Run-Length Encoding)

PCX (i.i.r.c. max 16 colors)

Ex: binary B&W bitmap

0... black
1... white

1111000000101111
4,1,6,0,1,1,1,0,4,1

× can be handled if binary

8-bit packed

3bits => 8 colours
5bits => max. 32 repetitions

can be implemented as a dictionary, BUT it is computationally ineffect.

Sliding window

- given by its width in 'elements' (source symbols)
- moves through the data set

Ex: sliding window width 8

abbadbabaabbaadbba

position 0
position 10

LZ77 (Lempel-Ziv)

- use a sliding window to search for repetitions of a phrase;
  if found, the current phrase is replaced with a reference

(n, d, c)
- c: next character
- d: length of phrase
- n: the number of chars to go backwards

Kdyby_byly_kdyby_bebyly_by_chyby.

K	d	y	b	y_	byl	yur	ybyyn
(0,0,K)	(0,0,d)	(0,0,y)	(0,0,b)	(2,1,_)	(3,2,l)	(5,2,r)	(10,4,n)

(0,0,K): no previous reference found

l	bylyb	yuc	h	yby.
(0,0,l)	(12,5,b)	(3,2,c)	(0,0,h)	(16,3,.)

32 characters → 13 triplets

× 8bit → 256 bits
× 7bit → 224 bits
(uncompressed, but "optimized" sequence)
→ ASCII

208 bits if 16 bits/triplet

(5bit, 4bit, 7bit) → 16 bits/triplet
- 4bit: max. length of phrase 15 char
- 5bit: length of the window is 32 chars (i.e. the max. backwards step is 32)

LZW (Lempel-Ziv-Welch)

- search in a long sliding window is slow

→ use a tree-based dictionary

a) dictionary is adaptive
(can be discarded after compression and re-created when decompressing)

b) pre-defined basis

Kdyby␣byli␣by␣bylyby␣chyby.

K	d	y	b	y␣	by	l	yu	r	yb	yu	n	e	byl	yu	by
0	1	2	3	2	4	15	5	16	6	14	16	7	8	18	16	15

+ + + +

Kd dy yb by ....

this goes to dictionary

␣	c	h	yby	.	EOF
4	9	10	22	11	

32 characters → 256 bits
224 bits

22 symbols à 6 bits = 132 bits

Dictionary:

k→0	kd→12	byly→26
d→1	dy→13	yub→27
y→2	yb→14	by␣→28
b→3	by→15	␣c→29
␣→4	yu→16	ch→30
l→5	ub→17	by→31
r→6	byl→18	yby.→32
n→7	ly→19	
e→8	yur→20	
c→9	ry→21	
h→10	yby→22	
.→11	yun→23	
(basis)	ne→24	
	eb→25	