Corso di Biblioteche Digitali

- Vittore Casarosa
  - casarosa@isti.cnr.it
  - Office: 050 621 3115
  - Mobile: 348 397 2168
  - Skype: vittore1201
- “Ricevimento” at the end of the lessons or by appointment
- Final assessment
  - 70% oral examination
  - 30% project (development of a small digital library))
- Reference material:
  - Ian Witten, David Bainbridge, David Nichols, How to build a Digital Library, Morgan Kaufmann, 2010, ISBN 978-0-12-374857-7 (Second edition)
  - Material provided by the teacher
- **http://cloudone.isti.cnr.it/casarosa/BDG/**

Modules

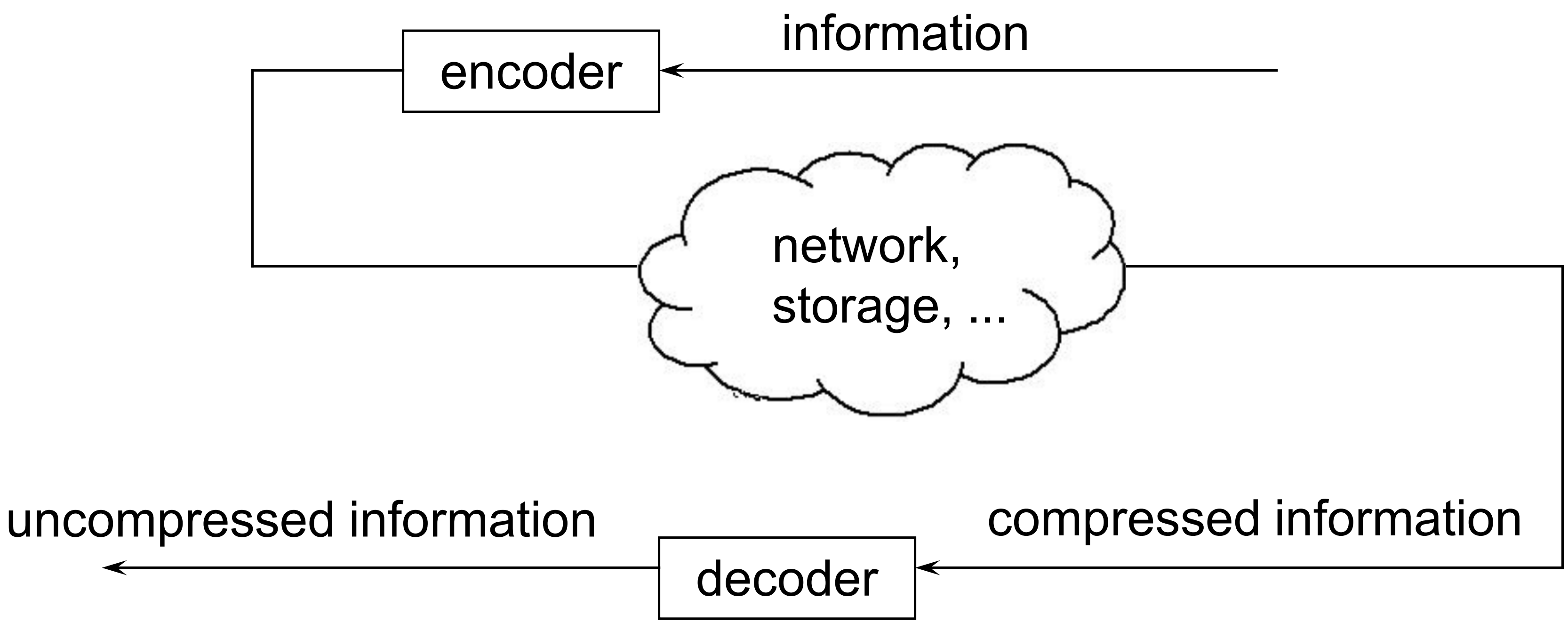
- Computer Fundamentals and Networking
- A conceptual model for Digital Libraries
- Bibliographic records and metadata
- Information Retrieval and Search Engines
- Knowledge representation
- Digital Libraries and the Web
- Hands-on laboratory: the Greenstone system

Refresher

Refresher on Computer Fundamentals and Networking

- History of computers
- Architecture of a computer
- Computer networks and the Internet
- Data representation within a computer

Compression of information



lossless compression: the uncompressed information is identical (bit by bit) to the original information

lossy compression: the uncompressed information contains less “information” than the original information

Lossless data compression

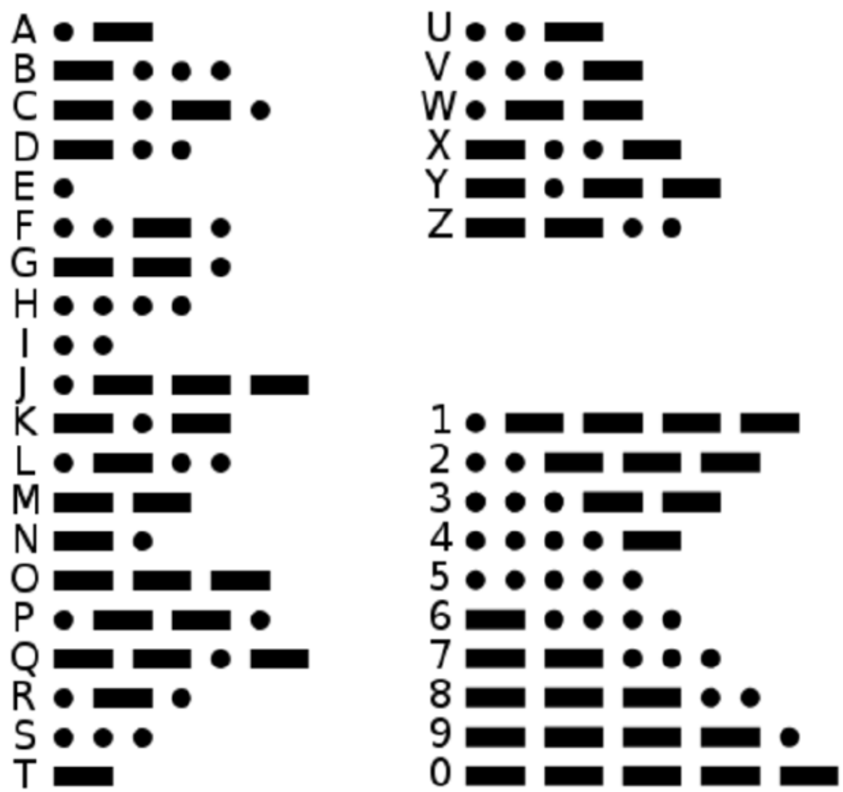
- The idea of text compression, or more generally data compression, is that when the data is not needed for processing (e.g. when in transit over a network or when stored on secondary storage), then it can be represented in a more compact form (with less bits), provided that it can be brought back to the original format when needed, i.e. we want to make a “lossless compression”.
- Given a string of symbols of a given alphabet (e.g. a string of characters out of the 26 letters of the English alphabet, or a string of numbers out of the 10 digits), which is represented in the computer by N bits, the compression process takes this string and represents it in a different way so that after compression the string takes n bits, with $n<N$
- Compression is not usually noticed (which means that it is well done) but it is used in a number of applications, such as transmission of fax, downloading of web pages, transmission of data over a network, storage of data onto secondary storage, zip files, tar files, etc.

Lossless compression techniques

- There are two main classes of lossless data compression methods
  - Symbol-wise encoding
  - Dictionary encoding
- Symbolwise encoding (entropy encoding)
  - The basic idea is that the most frequent symbols can be coded with less bits (short codewords) than the less frequent symbols (long codewords)
  - Symbol coders work by taking one symbol at the time from the input string, and coding it with a codeword whose length depends on the frequency (probability) of the symbol in the given alphabet
  - One of the most common symbol encoders is the Huffman coding
- Dictionary coding
  - The basic idea is to replace a sequence of symbols in the input string with an “index” in a dictionary (list) of “phrases”, i.e. substrings of the input string
  - ZIP (Lempel-Ziv-Welch)

“Symbolwise” Morse code (about 1840)

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.



Frequency distribution of the English letters

A	0.0856
B	0.0139
C	0.0279
D	0.0378
E	0.1304
F	0.0289
G	0.0199
H	0.0528
I	0.0627
J	0.0013
K	0.0042
L	0.0339
M	0.0249
N	0.0707
O	0.0797
P	0.0199
Q	0.0012
R	0.0677
S	0.0607
T	0.1045
U	0.0249
V	0.0092
W	0.0149
X	0.0017
Y	0.0199
Z	0.0008

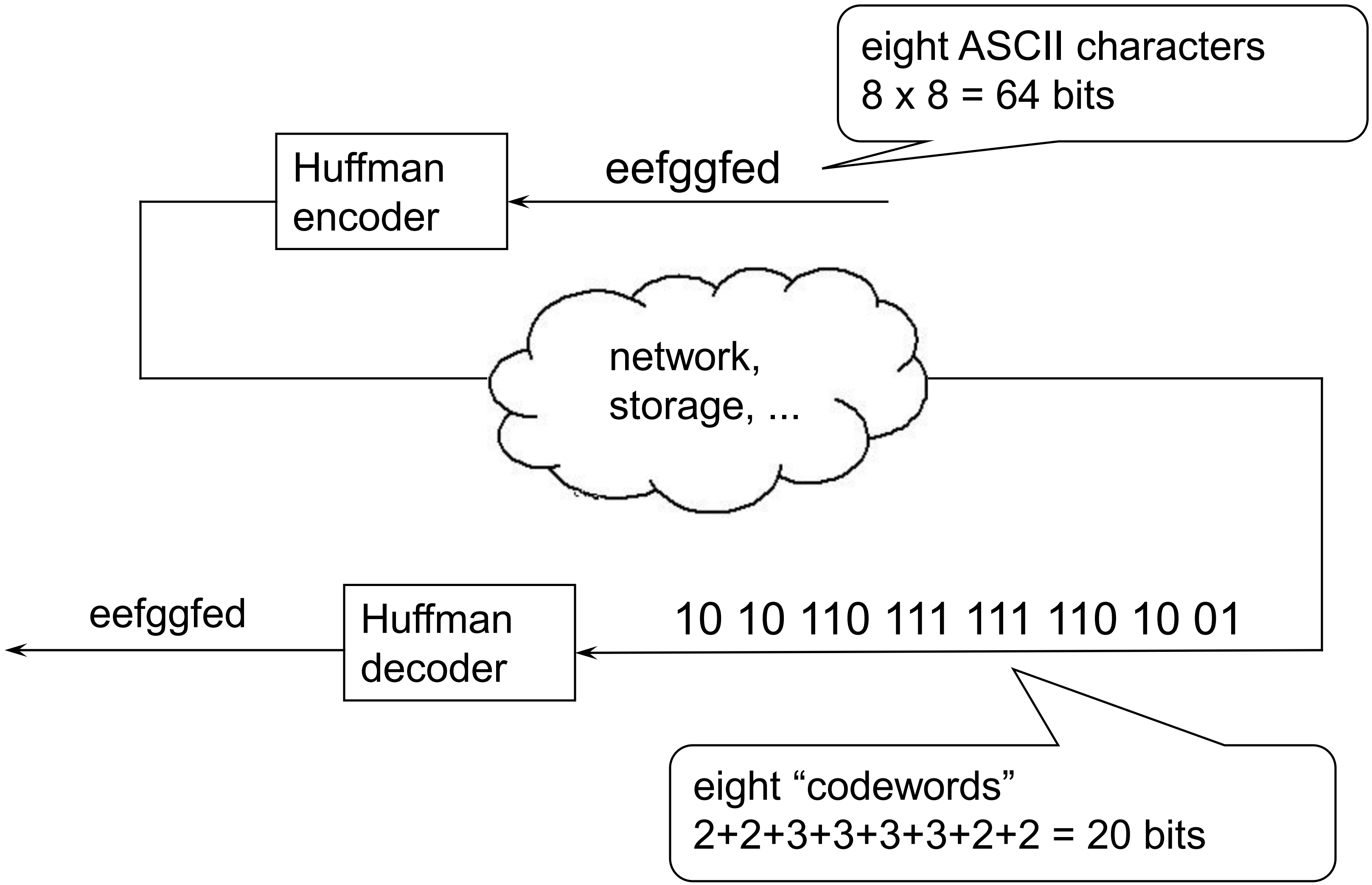
0.1304	E	.
0.1045	T	-
0.0856	A	.-
0.0797	O	---
0.0707	N	-.
0.0677	R	.-.
0.0627	I	..
0.0607	S	...
0.0528	H	....
0.0378	D	-..
0.0339	L	.-..
0.0289	F	..-.
0.0279	C	-.-.
0.0249	M	--
0.0249	U	..-
0.0199	G	--.
0.0199	Y	-.--
0.0199	P	.--.
0.0149	W	.--
0.0139	B	-...
0.0092	V	...-
0.0042	K	-.-
0.0017	X	-..-
0.0013	J	.---
0.0012	Q	--.-
0.0008	Z	--..

Huffman encoder

Alphabet with seven symbol and their probabilities (frequency)

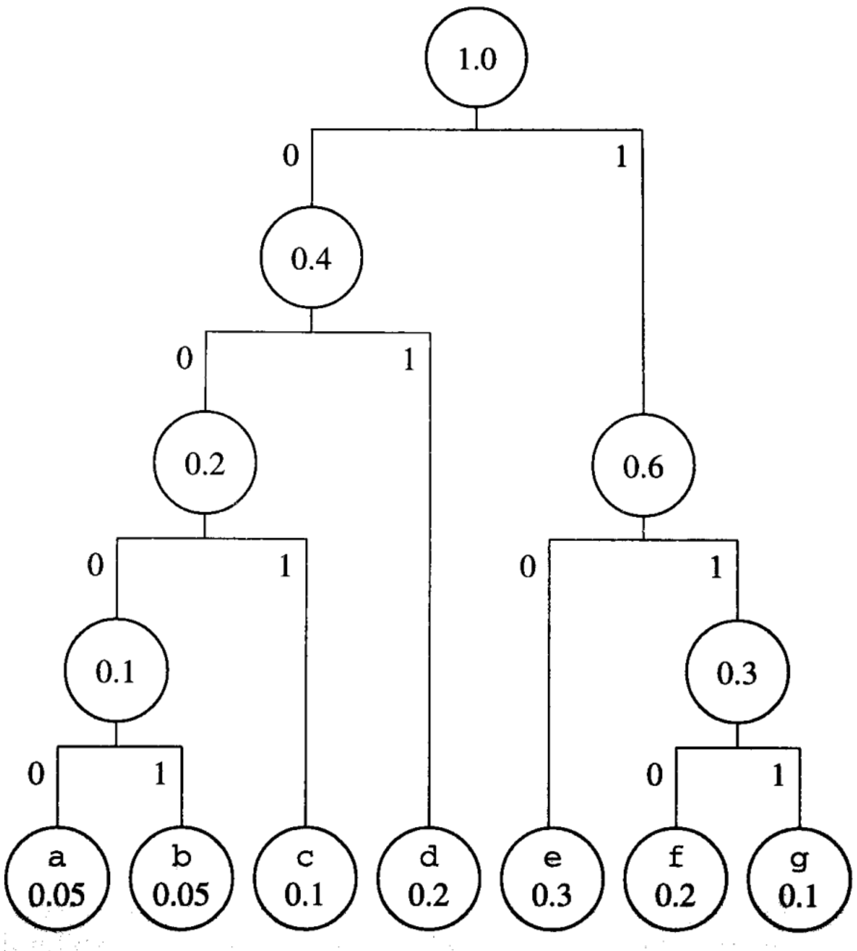
Symbol	Codeword	Probability
a	0000	0.05
b	0001	0.05
c	001	0.1
d	01	0.2
e	10	0.3
f	110	0.2
g	111	0.1

Huffman coding (symbolwise)



Huffman encoding and decoding

fbgced   1100001111001100 1

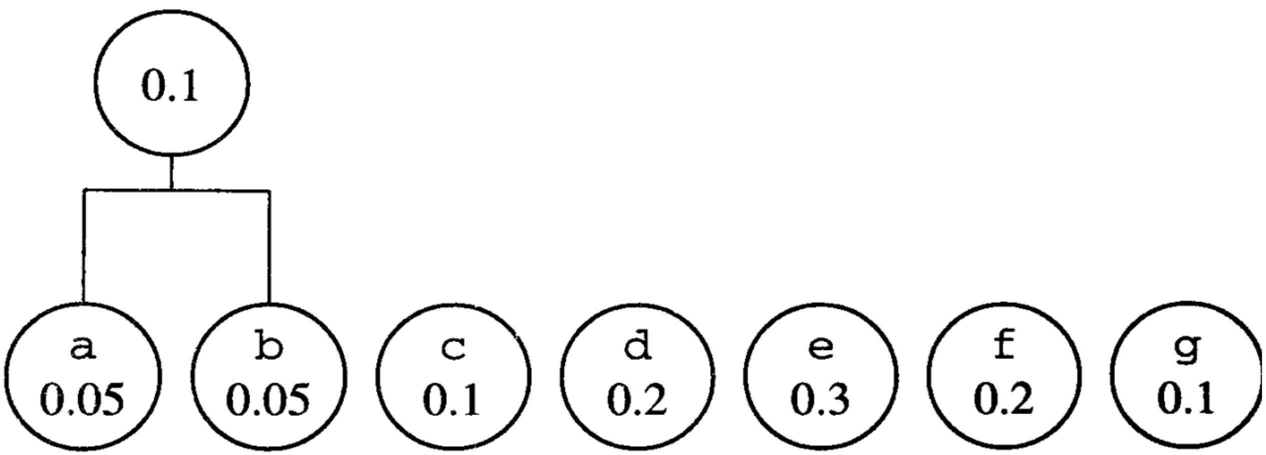


To encode, search the input symbol among the leafs; climb the tree up to the root; the sequence of bits encountered, in reverse order, is the code word.

To decode, take one bit at a time from the string to be decoded; go down the tree according to the value of the bit; when a leaf is reached, that is the value of the symbol.

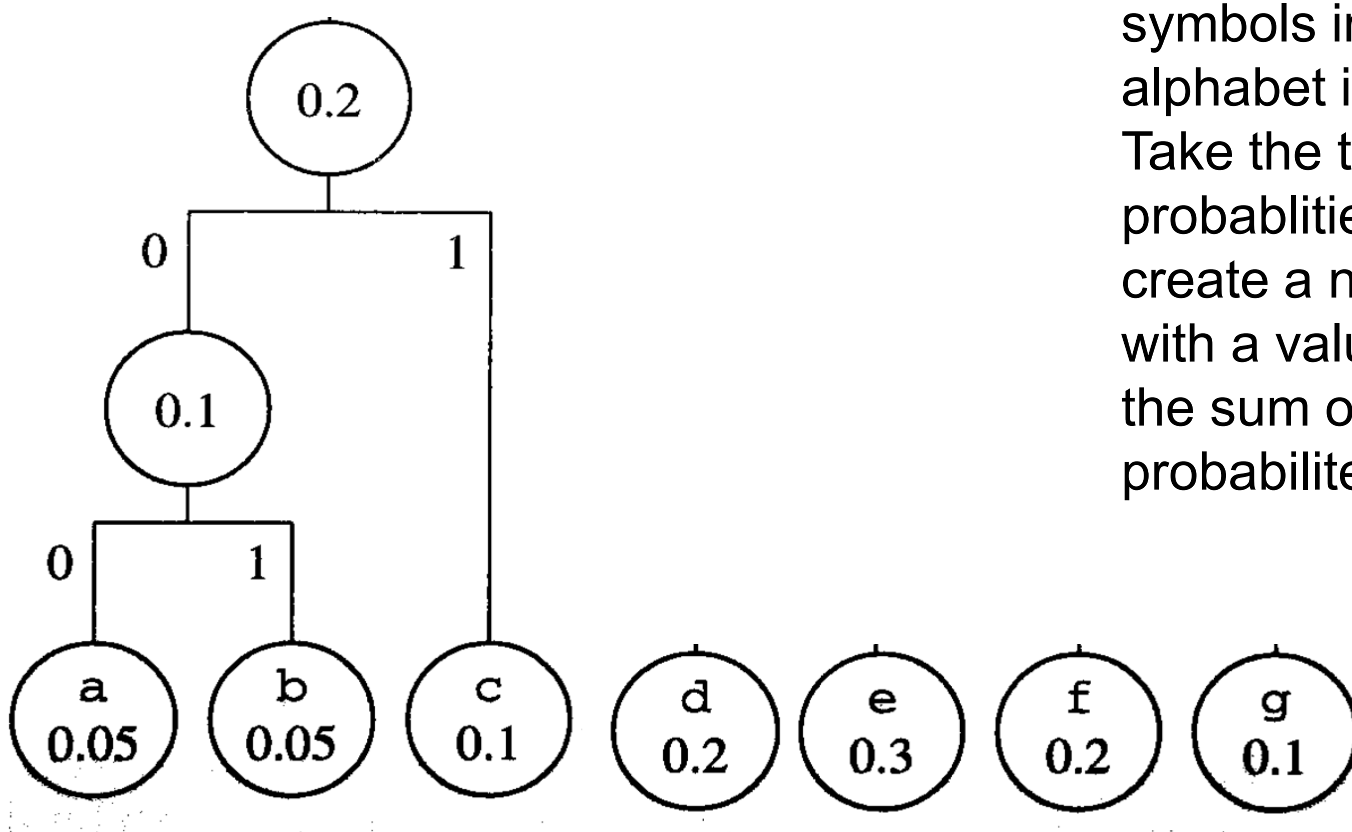
Building the Huffman tree (1)

The probability distribution of the symbols in the alphabet is given;
Take the two lowest probabilities and create a new node, with a value equal to the sum of the probabilities



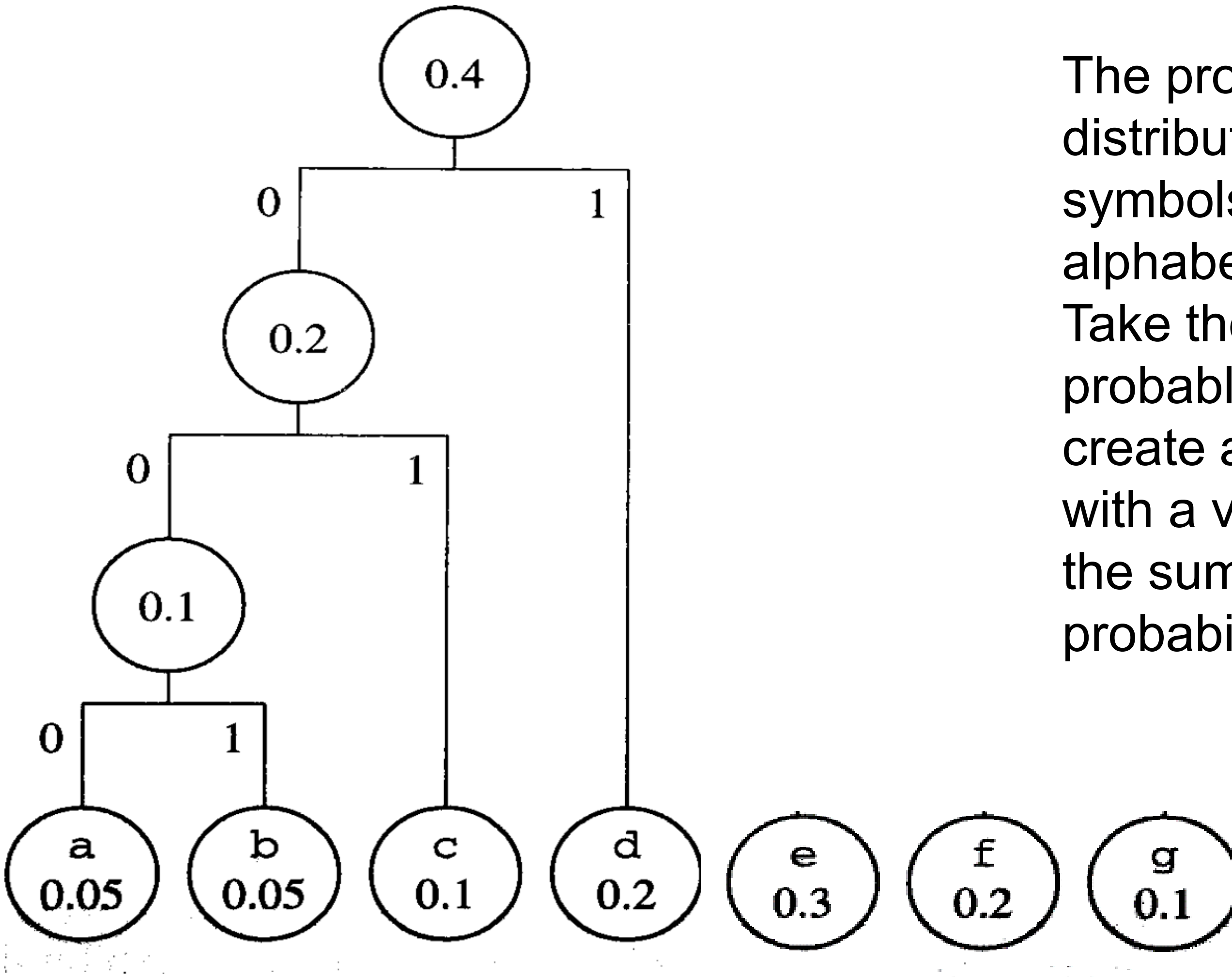
a	0,005
b	0,005
c	0,1
d	0,2
e	0,3
f	0,2
g	0.1

Building the Huffman tree (2)



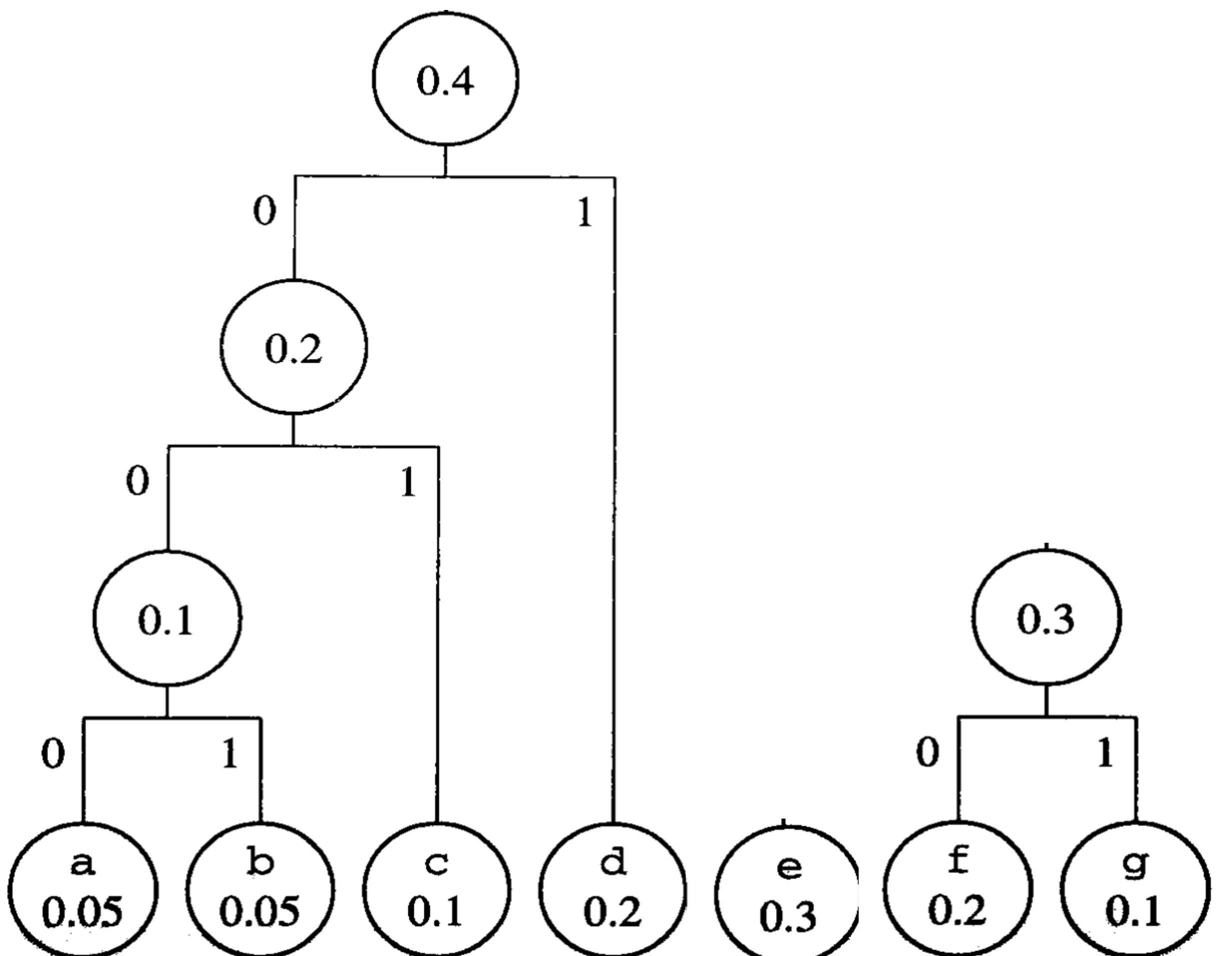
The probability distribution of the symbols in the alphabet is given; Take the two lowest probablities and create a new node, with a value equal to the sum of the probabilites

Building the Huffman tree (3)



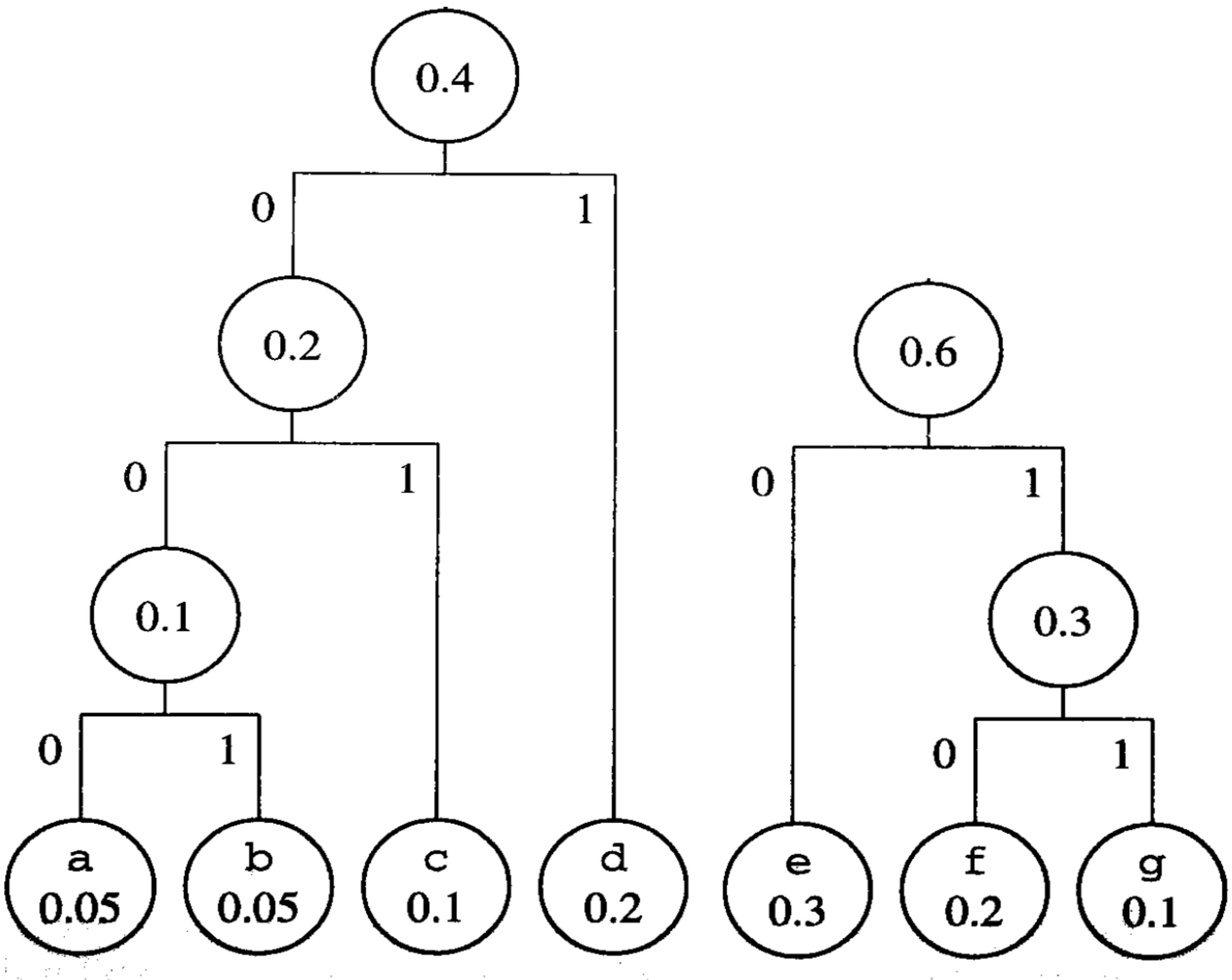
The probability distribution of the symbols in the alphabet is given; Take the two lowest probablities and create a new node, with a value equal to the sum of the probabilites

Building the Huffman tree (4)



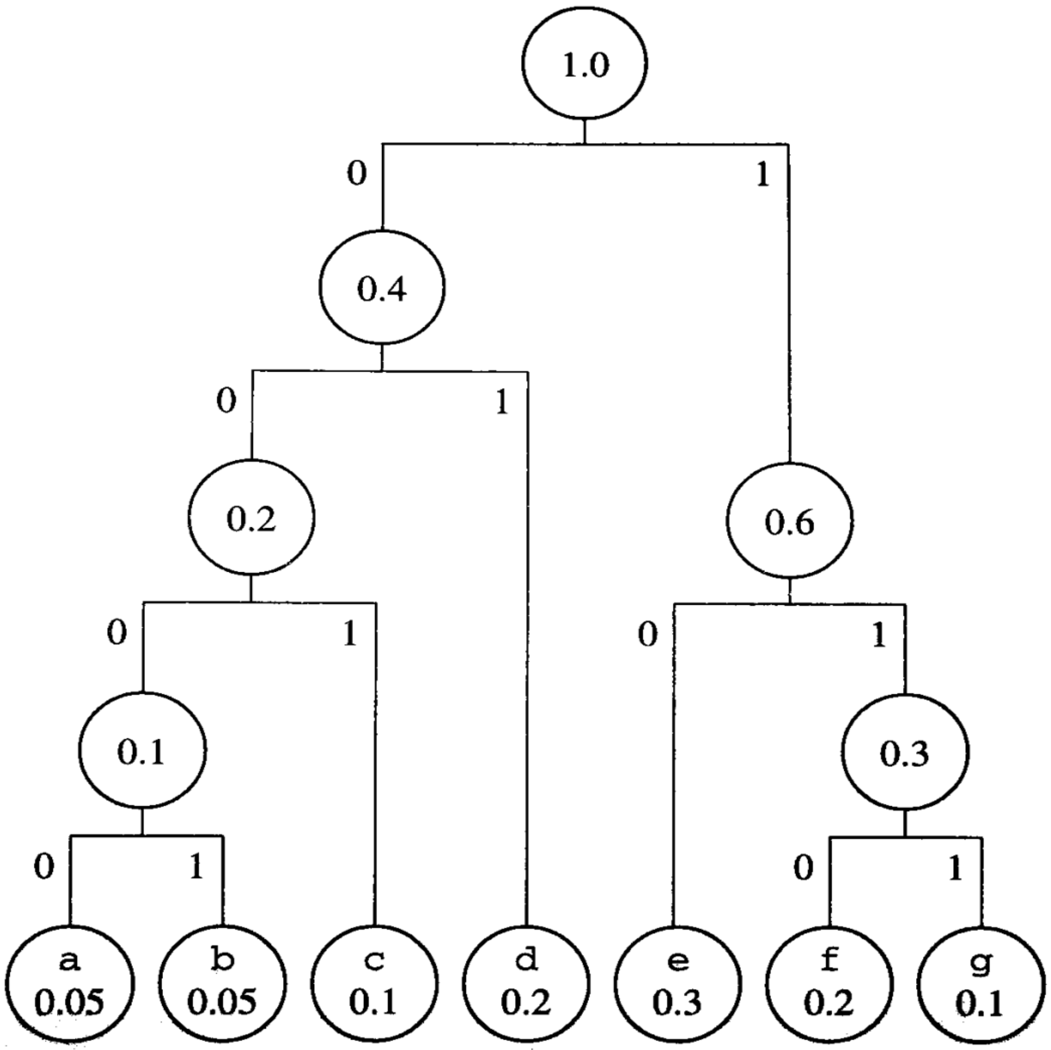
The probability distribution of the symbols in the alphabet is given; Take the two lowest probablities and create a new node, with a value equal to the sum of the probabilites

Building the Huffman tree (5)



The probability distribution of the symbols in the alphabet is given; Take the two lowest probablities and create a new node, with a value equal to the sum of the probabilites

Building the Huffman tree (6)



The probability distribution of the symbols in the alphabet is given; Take the two lowest probablities and create a new node, with a value equal to the sum of the probabilites

Probability distribution in Huffman coding

The last question is: how do we know (build) the probability distribution ?

- Pre-defined, usually based on the context
  - The language (e.g. English, Italian, etc)
  - The type of application (e.g accounting)
  - Main disadvantage: the coder may not perform well in a different context;
- Built ad hoc for each file, with a preliminary scan of the text to be encoded, and a counting the frquency of the symbols of the alphabet
  - Main disadvantages: more processing; need to send the Huffman tree to the decoder

Lossless compression techniques

- There are two main classes of lossless data compression methods
  - Symbol-wise encoding
  - Dictionary encoding
- Symbolwise encoding
  - The basic idea is that the most frequent symbols can be coded with less bits (short codewords) than the less frequent symbols (long codewords)
  - Symbol coders work by taking one symbol at the time from the input string, and coding it with a codeword whose length depends on the frequency (probability) of the symbol in the given alphabet
  - One of the most common symbol encoders is the Huffman coding
- Dictionary coding
  - The basic idea is to replace a sequence of symbols in the input string with an “index” in a dictionary (list) of “phrases”, i.e. substrings of the input string
  - ZIP (Lempel-Ziv-Welch)

LZW: Lempel–Ziv–Welch coding

- The encoder looks in the current dictionary for an entry (a string) matching the initial symbols of the string to be coded
- When found, the codeword for the whole group of symbols is the phrase number in the dictionary (the index)
- A new phrase is added to the dictionary, by concatenating the entry just found with the next input symbol
- Initially, the codewords may be longer than the input symbols (due to few phrases in the dictionary), but as the coding proceeds (new phrases are added to the dictionary), the codewords are representing longer and longer sequences of symbols
- What is needed is an “initial dictionary”
- The dictionary is initialized with all the symbols of the “alphabet”

Starting dictionary (the alphabet) The ASCII table

_phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
....
....

....
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
....
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

new phrases added as the string is being encoded

input string	a b a ab ab ba aba abaa
output string	

Current dictionary

phrase 0 = NUL	....
phrase 1 = SOH	phrase 97= a
phrase 2 = STX	phrase 98 = b
phrase 3 = ETX	phrase 99 = c
phrase 4 = EOT	phrase 100 = d
phrase 5 = ENQ	phrase 101 = e
phrase 6 = ACK	phrase 102 = f
phrase 7 = BEL	....
phrase 8 = BS	phrase 123 = {
phrase 9 = HT	phrase 124 = |
phrase 10 = LF	phrase 125 = }
phrase 11 = VT	phrase 126 = ~
phrase 12 = FF	phrase 127 = DEL
phrase 13 = CR	
....	
....	

....

....

new phrases added as the string is being encoded

input string: a b a ab ab ba aba abaa

output string: 97

Current dictionary

phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
. . . .
. . . .

. . . .
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
. . . .
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

. . . .
phrase 128 = ab

new phrases added as the string is being encoded

input string	b a ab ab ba aba abaa
output string	97 98

Current dictionary

phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
....
....

....
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
....
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

....
phrase 128 = ab
phrase 129 = ba

new phrases added as the string is being encoded

input string	a ab ab ba aba abaa
output string	97 98 97 128 128 129 131 134

Current dictionary

_phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
. . . .
. . . .

. . . .
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
. . . .
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

. . . .
phrase 128 = ab
phrase 129 = ba
phrase 130 = aa

new phrases added as the string is being encoded

input string: ab ab ba aba abaa

output string: 97 98 97 128 128 129 131 134

Current dictionary

phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
. . . .
. . . .

. . . .
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
. . . .
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

. . . .
phrase 128 = ab
phrase 129 = ba
phrase 130 = aa
phrase 131 = aba

new phrases added as the string is being encoded

input string: ab ba aba abaa

output string: 97 98 97 128 128 129 131 134

Current dictionary

phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
. . . .
. . . .

. . . .
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
. . . .
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

. . . .
phrase 128 = ab
phrase 129 = ba
phrase 130 = aa
phrase 131 = aba
phrase 132 = abb

new phrases added as the string is being encoded

input string: ba aba abaa

output string: 97 98 97 128 128 129 131 134

Current dictionary

phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
....
....

....
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
....
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

....
phrase 128 = ab
phrase 129 = ba
phrase 130 = aa
phrase 131 = aba
phrase 132 = abb
phrase 133 = baa

new phrases added as the string is being encoded

input string	aba abaa
output string	97 98 97 128 128 129 131 134

Current dictionary

phrase 0 = NUL
phrase 1 = SOH
phrase 2 = STX
phrase 3 = ETX
phrase 4 = EOT
phrase 5 = ENQ
phrase 6 = ACK
phrase 7 = BEL
phrase 8 = BS
phrase 9 = HT
phrase 10 = LF
phrase 11 = VT
phrase 12 = FF
phrase 13 = CR
. . . .
. . . .

. . . .
phrase 97= a
phrase 98 = b
phrase 99 = c
phrase 100 = d
phrase 101 = e
phrase 102 = f
. . . .
phrase 123 = {
phrase 124 = |
phrase 125 = }
phrase 126 = ~
phrase 127 = DEL

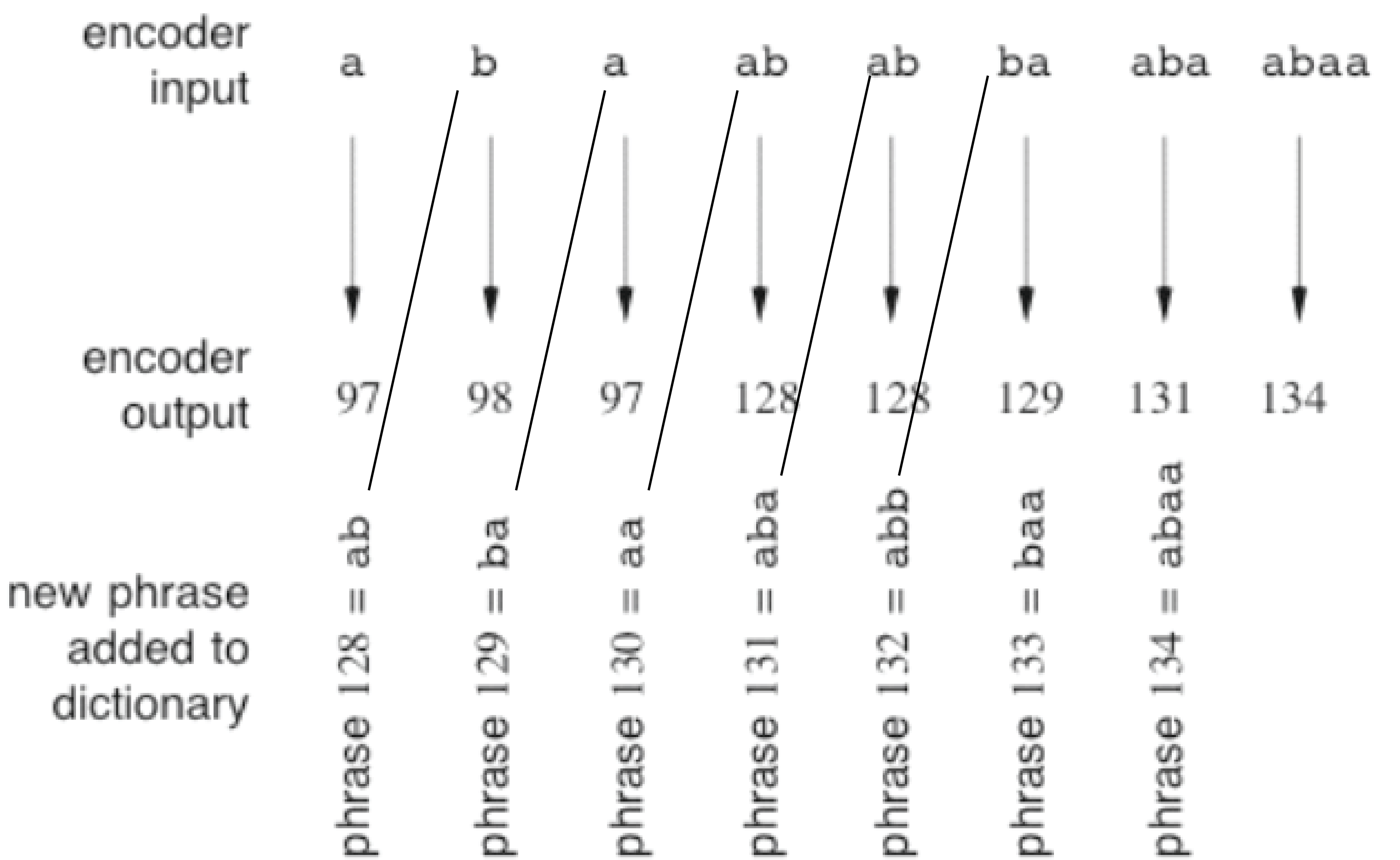
. . . .
phrase 128 = ab
phrase 129 = ba
phrase 130 = aa
phrase 131 = aba
phrase 132 = abb
phrase 133 = baa
phrase 134 = abaa
phrase 135 = abaax
....

new phrases added as the string is being encoded

input string: abaa x

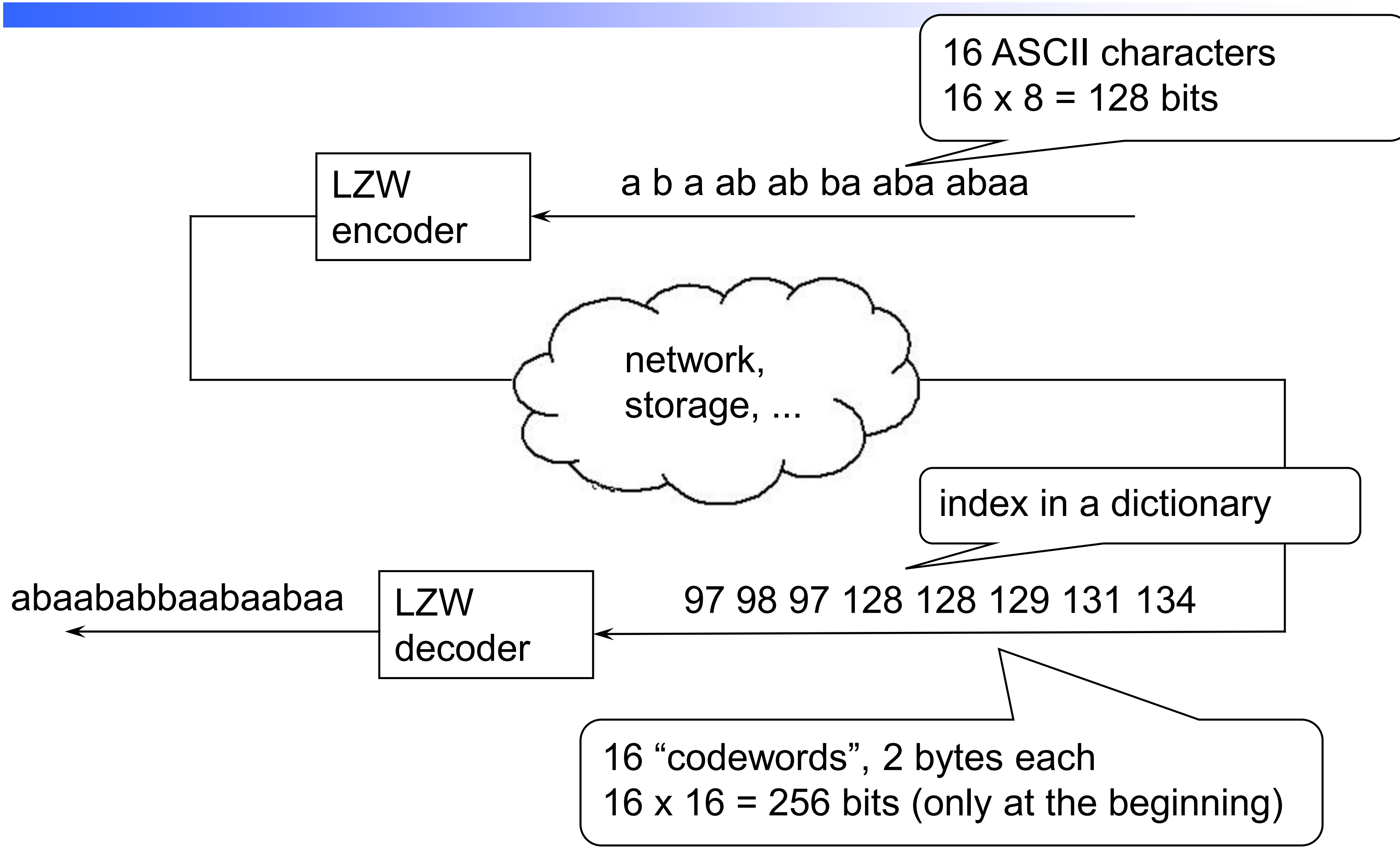
output string: 97 98 97 128 128 129 131 134

Lempel – Ziv – Welch algorithm



Dictionary entries 0-127 are filled with the “alphabet” (in this case ASCII)

LZW coding


16 ASCII characters
16 x 8 = 128 bits
LZW encoder
a b a ab ab ba aba abaa
network, storage, ...
index in a dictionary
abaababbaabaabaa
LZW decoder
97 98 97 128 128 129 131 134
16 “codewords”, 2 bytes each
16 x 16 = 256 bits (only at the beginning)

Families of compressors

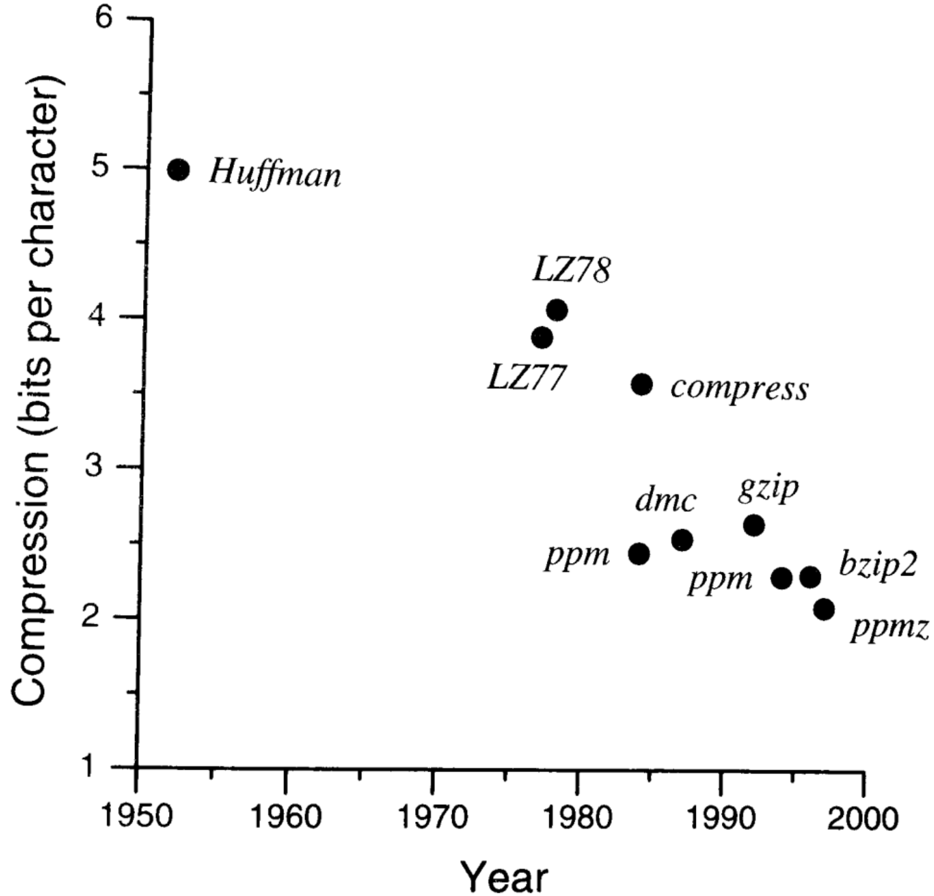
- A number of compressors are available, and many of them are based on the compressors just seen, or their variations
  - compress
  - deflate
  - gzip
  - ...
- Variation mainly introduced to improve the efficiency of coding and decoding (trade-offs between speed and compression rate) and memory occupation (and to overcome problems with patents)
- Common testbeds to compare them

Testbed for compression methods

Table 2.7 The Canterbury corpus of text used to evaluate compression methods.

File	Bytes	Content
text	152,089	The text of Lewis Carroll's *Alice's Adventures in Wonderland*
fax	513,216	A fax bitmap image (CCITT test document 5)
Csrc	11,150	C source code
Excl	1,029,744	Excel spreadsheet
SPRC	38,240	Executable object code for Sun SPARC architecture
tech	426,754	Technical writing (workshop proceedings)
poem	481,861	*Paradise Lost* by John Milton
HTML	24,603	Hypertext Markup Language source
lisp	3,721	A Lisp program
man	4,227	A Unix manual page in the *roff* format
play	125,179	Shakespeare's play, *As You Like It*

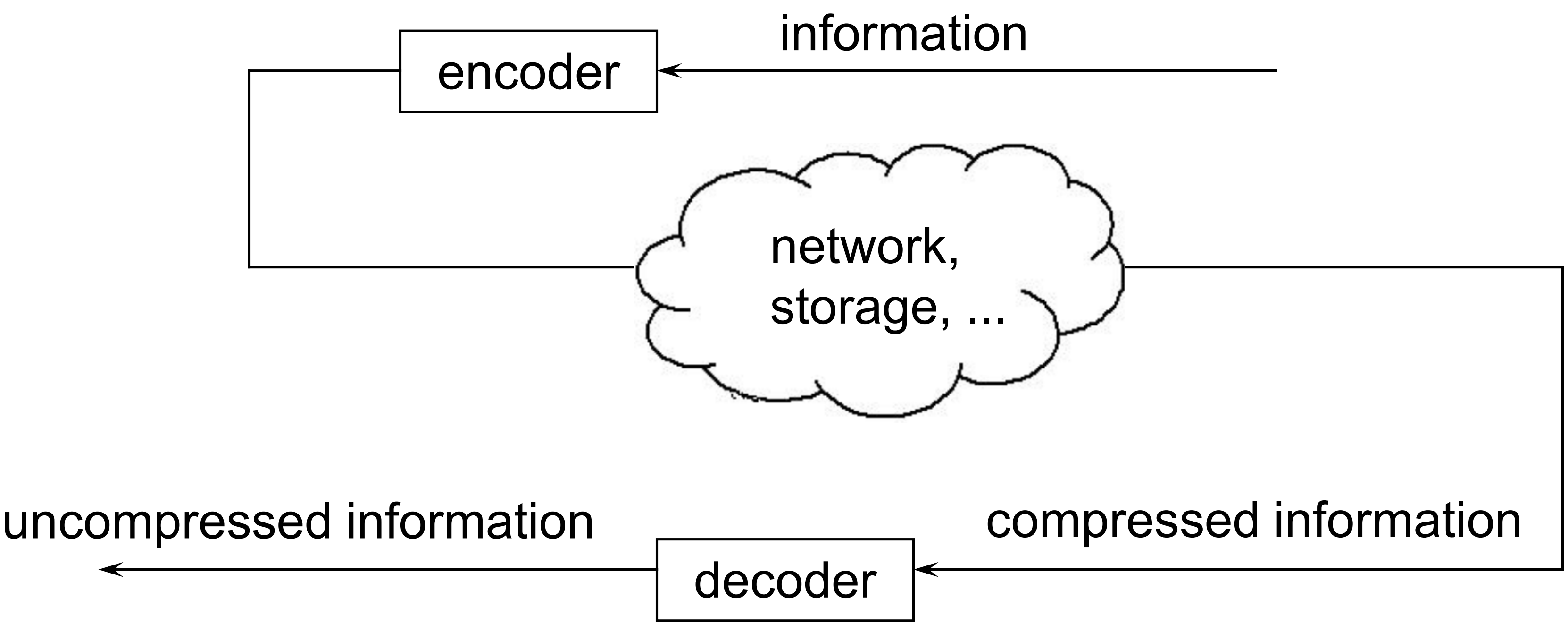
Comparison of compression methods



Comparison of compression methods (bits per char)

Method	Relative speed		Compression	
	Encoding	Decoding	bpc	%
dmc	24.3	24.5	2.40	30.0
ppm	5.3	5.9	2.11	26.4
char	2.9	4.0	4.49	56.1
bzip2	5.5	2.0	2.23	27.9
pack	0.6	0.9	4.53	56.6
huffword	2.2	0.9	2.95	36.9
compress	1.0	0.6	3.31	41.4
lzrw1	0.7	0.4	4.18	52.3
gzip-f	1.1	0.4	2.91	36.4
gzip-b	7.0	0.3	2.53	31.6
null	0.2	0.2	8.00	100.0

Compression of information



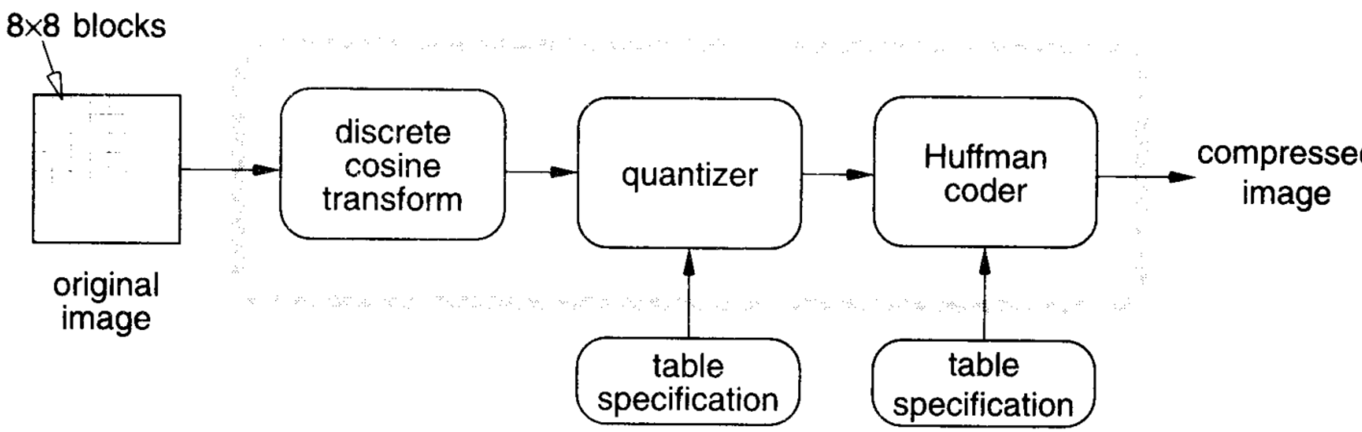
lossless compression: the uncompressed information is identical (bit by bit) to the original information

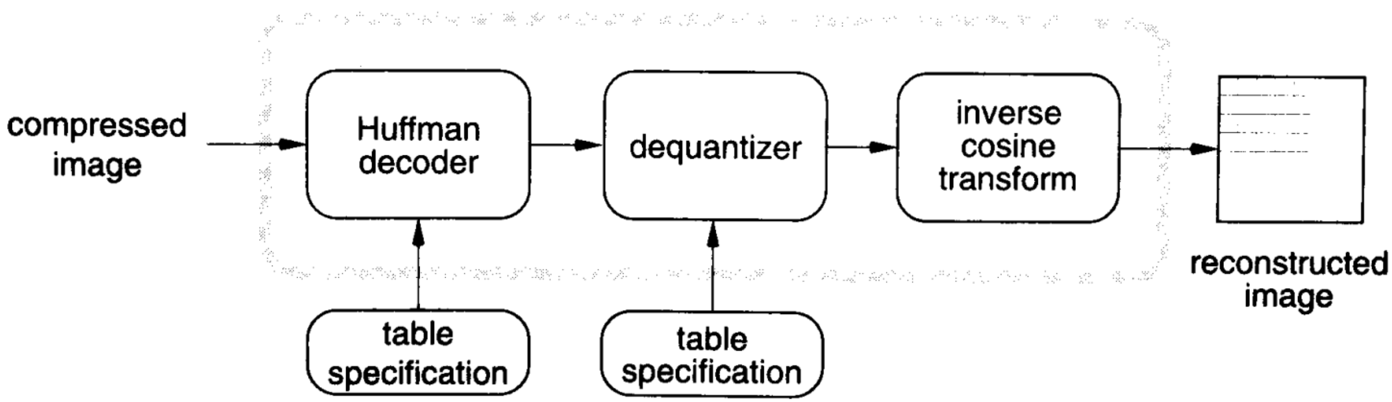
lossy compression: the uncompressed information contains less “information” than the original information

JPEG

- For grayscale and color images, lossless compression still results in “too many bits”
- Lossy compression methods take advantage from the fact that the human eye is less sensitive to small greyscale or color variation in an image
- JPEG - Joint Photographic Experts Group and Joint Binary Image Group, part of CCITT and ISO
- JPEG can compress down to about one bit per pixel (starting with 8-48 bits per pixel) still having excellent image quality
  - Not very good for fax-like images
  - Not very good for sharp edges and sharp changes in color
- The encoding and decoding process is done on an 8x8 block of pixels (separately for each color component)

JPEG encoding and decoding





Discrete Cosine Transform

pixel values

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	110
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

DCT coefficients

162.3	40.6	20.0	72.3	30.3	12.5	–19.7	–11.5
30.5	108.4	10.5	32.3	27.7	–15.5	18.4	–2.0
–94.1	–60.1	12.3	–43.4	–31.3	6.1	–3.3	7.1
–38.6	–83.4	–5.4	–22.2	–13.5	15.5	–1.3	3.5
–31.3	17.9	–5.5	–12.4	14.3	–6.0	11.5	–6.0
–0.9	–11.8	12.8	0.2	28.1	12.6	8.4	2.9
4.6	–2.4	12.2	6.6	–18.7	–12.8	7.7	12.0
–10.0	11.2	7.8	–16.3	21.5	0.0	5.9	10.7

Discrete Cosine Transform

JPEG quantization matrix

$Q_{10} =$

80	60	50	80	120	200	255	255
55	60	70	95	130	255	255	255
70	65	80	120	200	255	255	255
70	85	110	145	255	255	255	255
90	110	185	255	255	255	255	255
120	175	255	255	255	255	255	255
245	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

The values of the quantization matrix are used to divide the DCT coefficients, and the result is rounded to nearest integer.
The quantization matrix determines the amount of “loss” (the higher the values, the higher the loss)

$Q_{90} =$

3	2	2	3	5	8	10	12
2	2	3	4	5	12	12	11
3	3	3	5	8	11	14	11
3	3	4	6	10	17	16	12
4	4	7	11	14	22	21	15
5	7	11	13	16	12	23	18
10	13	16	17	21	24	24	21
14	18	19	20	22	20	20	20

$Q_{50} =$

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

The “lossy step”

162.3	40.6	20.0	72.3	30.3	12.5	–19.7	–11.5
30.5	108.4	10.5	32.3	27.7	–15.5	18.4	–2.0
–94.1	–60.1	12.3	–43.4	–31.3	6.1	–3.3	7.1
–38.6	–83.4	–5.4	–22.2	–13.5	15.5	–1.3	3.5
–31.3	17.9	–5.5	–12.4	14.3	–6.0	11.5	–6.0
–0.9	–11.8	12.8	0.2	28.1	12.6	8.4	2.9
4.6	–2.4	12.2	6.6	–18.7	–12.8	7.7	12.0
–10.0	11.2	7.8	–16.3	21.5	0.0	5.9	10.7

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

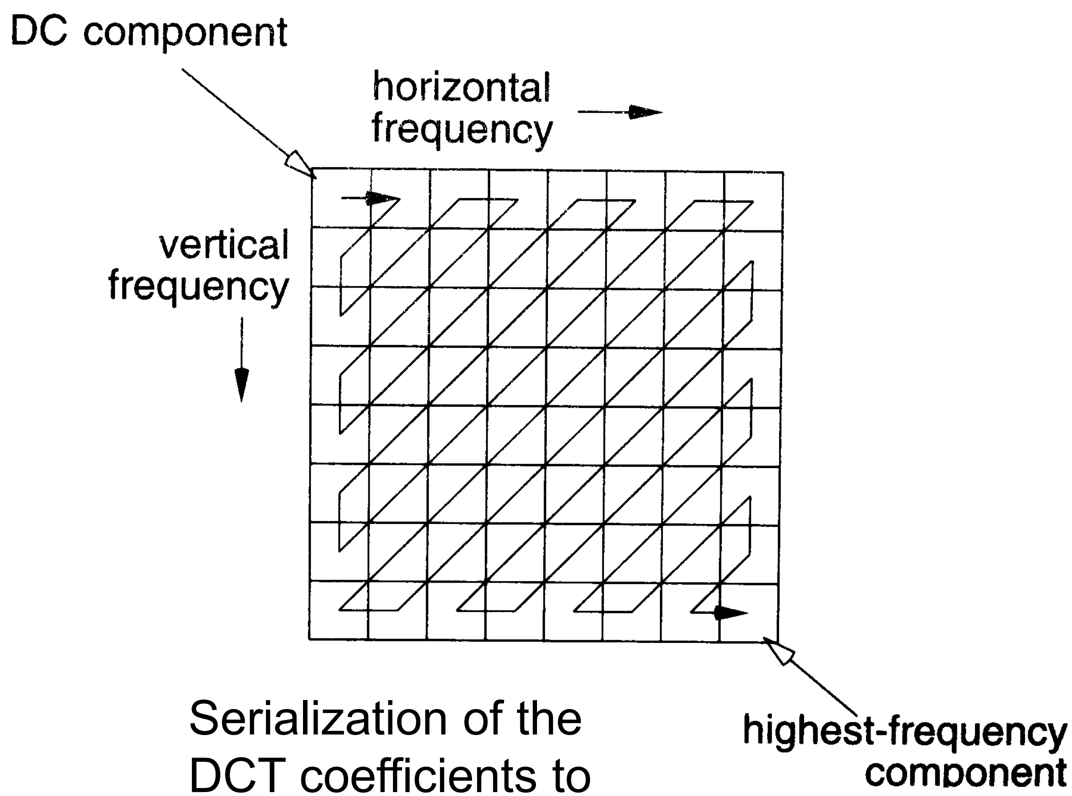
divide DCT coefficients by Q_{50} quantization matrix, round to nearest integer and get this result

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
–7	–5	1	–2	–1	0	0	0
–3	–5	0	–1	0	0	0	0
–2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization and coding

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
–7	–5	1	–2	–1	0	0	0
–3	–5	0	–1	0	0	0	0
–2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

DCT coefficients after quantization Q_{50}
The DCT coefficients have been divided by the quantization matrix and then rounded to nearest integer



Serialization of the DCT coefficients to maximize run-lengths of zeros and therefore take advantage of Huffman coding

JPEG dequantization

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
–7	–5	1	–2	–1	0	0	0
–3	–5	0	–1	0	0	0	0
–2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

160	44	20	80	24	0	0	0
36	108	14	38	26	0	0	0
–98	–65	16	–48	–40	0	0	0
–42	–85	0	–29	0	0	0	0
–36	22	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

DCT coefficients after quantization Q_{50}

In between there is the Huffman coding and decoding

DCT coefficients dequantized

Inverse DCT

160	44	20	80	24	0	0	0
36	108	14	38	26	0	0	0
–98	–65	16	–48	–40	0	0	0
–42	–85	0	–29	0	0	0	0
–36	22	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

149	134	119	116	121	126	127	128
204	168	140	144	155	150	135	125
253	195	155	166	183	165	131	111
245	185	148	166	184	160	124	107
188	149	132	155	172	159	141	136
132	123	125	143	160	166	168	171
109	119	126	128	139	158	168	166
111	127	127	114	118	141	147	135

Inverse of Discrete Cosine Transform

Comparison with original values

149	134	119	116	121	126	127	128
204	168	140	144	155	150	135	125
253	195	155	166	183	165	131	111
245	185	148	166	184	160	124	107
188	149	132	155	172	159	141	136
132	123	125	143	160	166	168	171
109	119	126	128	139	158	168	166
111	127	127	114	118	141	147	135

pixel values after Inverse Cosine Transform

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	110
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

original pixel values

Summary of JPEG compression (1/2)

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	110
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

original pixel values

162.3	40.6	20.0	72.3	30.3	12.5	–19.7	–11.5
30.5	108.4	10.5	32.3	27.7	–15.5	18.4	–2.0
–94.1	–60.1	12.3	–43.4	–31.3	6.1	–3.3	7.1
–38.6	–83.4	–5.4	–22.2	–13.5	15.5	–1.3	3.5
–31.3	17.9	–5.5	–12.4	14.3	–6.0	11.5	–6.0
–0.9	–11.8	12.8	0.2	28.1	12.6	8.4	2.9
4.6	–2.4	12.2	6.6	–18.7	–12.8	7.7	12.0
–10.0	11.2	7.8	–16.3	21.5	0.0	5.9	10.7

DCT coefficients

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
–7	–5	1	–2	–1	0	0	0
–3	–5	0	–1	0	0	0	0
–2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

DCT coefficients
after quantization

linearization and
compression
(Huffman coding)

Summary of JPEG compression (2/2)

de-compression and reconstruction of block

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
–7	–5	1	–2	–1	0	0	0
–3	–5	0	–1	0	0	0	0
–2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

160	44	20	80	24	0	0	0
36	108	14	38	26	0	0	0
–98	–65	16	–48	–40	0	0	0
–42	–85	0	–29	0	0	0	0
–36	22	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

DCT coefficients after de-quantization

149	134	119	116	121	126	127	128
204	168	140	144	155	150	135	125
253	195	155	166	183	165	131	111
245	185	148	166	184	160	124	107
188	149	132	155	172	159	141	136
132	123	125	143	160	166	168	171
109	119	126	128	139	158	168	166
111	127	127	114	118	141	147	135

pixel values after Inverse Cosine Transform

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	110
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

for comparison, original pixel values

JPEG – Final comments

- Arithmetic coding instead of Huffman coding (10% improvement in compression)
- JPEG-2000 - Use of wavelets instead of DCT (20% improvement in compression, better quality for images with sharp edges)
- JPEG-LS – lossless compression
  - For each pixel, what is coded is the difference between the actual pixel value and a prediction of pixel value based on the pixel context
- Compression rates
  - 0.25–0.5 bit/pixel: moderate to good quality, sufficient for some applications
  - 0.5–0.75 bit/pixel: good to very good quality, sufficient for many applications
  - 0.75–1.5 bit/pixel: excellent quality, sufficient for most applications
  - 1.5–2 bits/pixel: usually indistinguishable from the original, sufficient for the most demanding applications