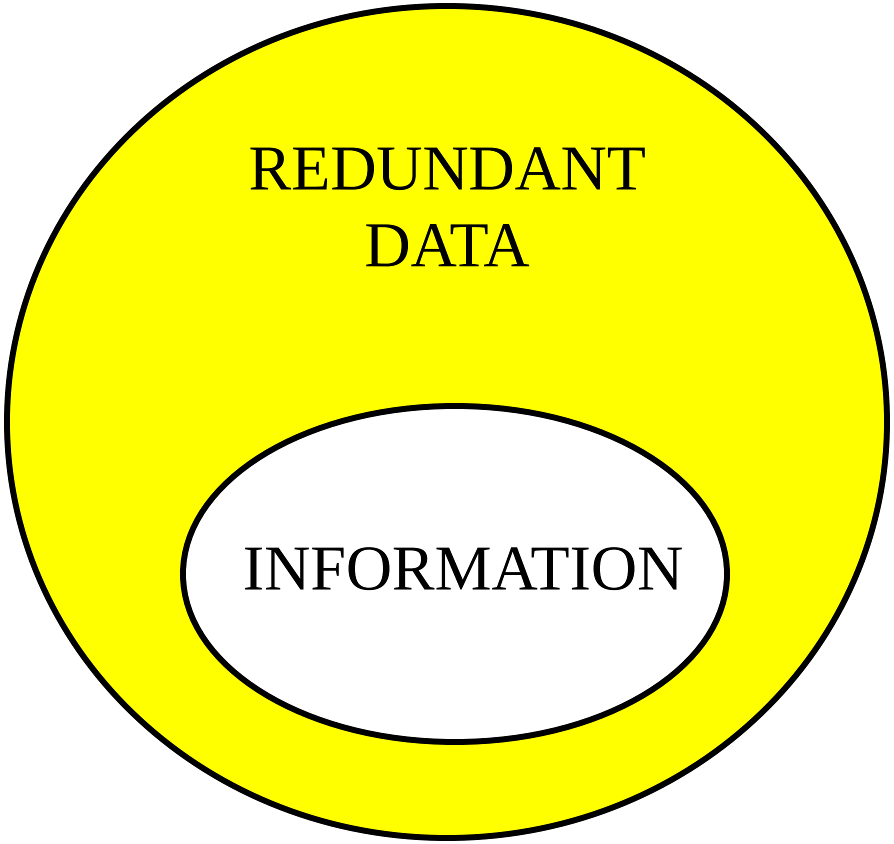
Lecture 11
Compression

IMAGE COMPRESSION

IMAGE COMPRESSION

- Addresses the problem of reducing the amount of data required to represent a digital image
- The underlying basis of the reduction process is the removal of redundant data

Information vs Data



REDUNDANT DATA

INFORMATION

DATA = INFORMATION + REDUNDANT DATA

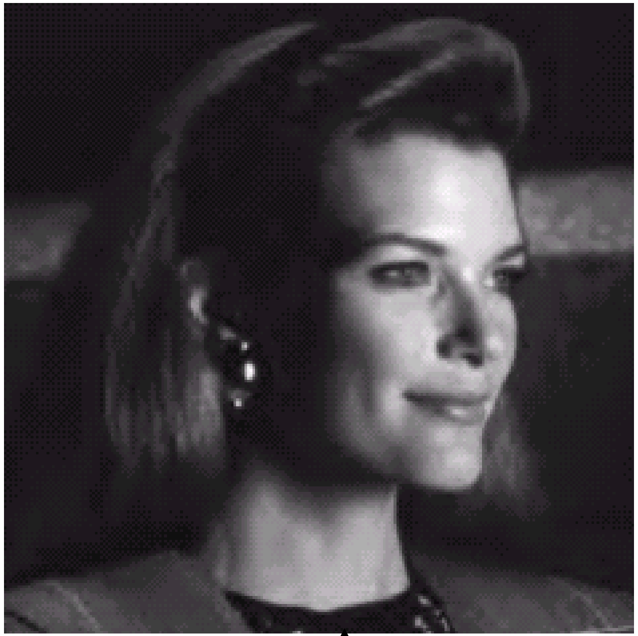
IMAGE COMPRESSION: CODING AND DECODING

original image
262144 Bytes

image encoder

compressed bit stream
00111000001001101…
(2428 Bytes)

image decoder



compression ratio (CR) = 108:1

LOSSY VS LOSSLESS COMPRESSION

- Lossless (Information preserving)
  - Images can be compressed and restored without any loss of information.
  - Application: Medical images
- Lossy
  - Perfect recovery is not possible but provides a large data compression.
  - Example: TV signals, teleconferencing

FUNDAMENTALS

- Raw image: A set of n1 bits
- Compressed image: A set of n2 bits.
- Compression ratio: $$C_R = \frac{n_1}{n_2}$$
- Relative Data Redundancy of first set: $$R_D = 1 - \frac{1}{C_R}$$
- Example: *n1*= 100KB and *n2* = 10Kb, then *CR* = 10 , and *RD* = 90%
- Special cases:
  - n1>>n2 → CR ≈ ∞, RD ≈ 1
  - n1 ≈ n2 → CR ≈ 1, RD ≈ 0

IMAGE COMPRESSION MODEL

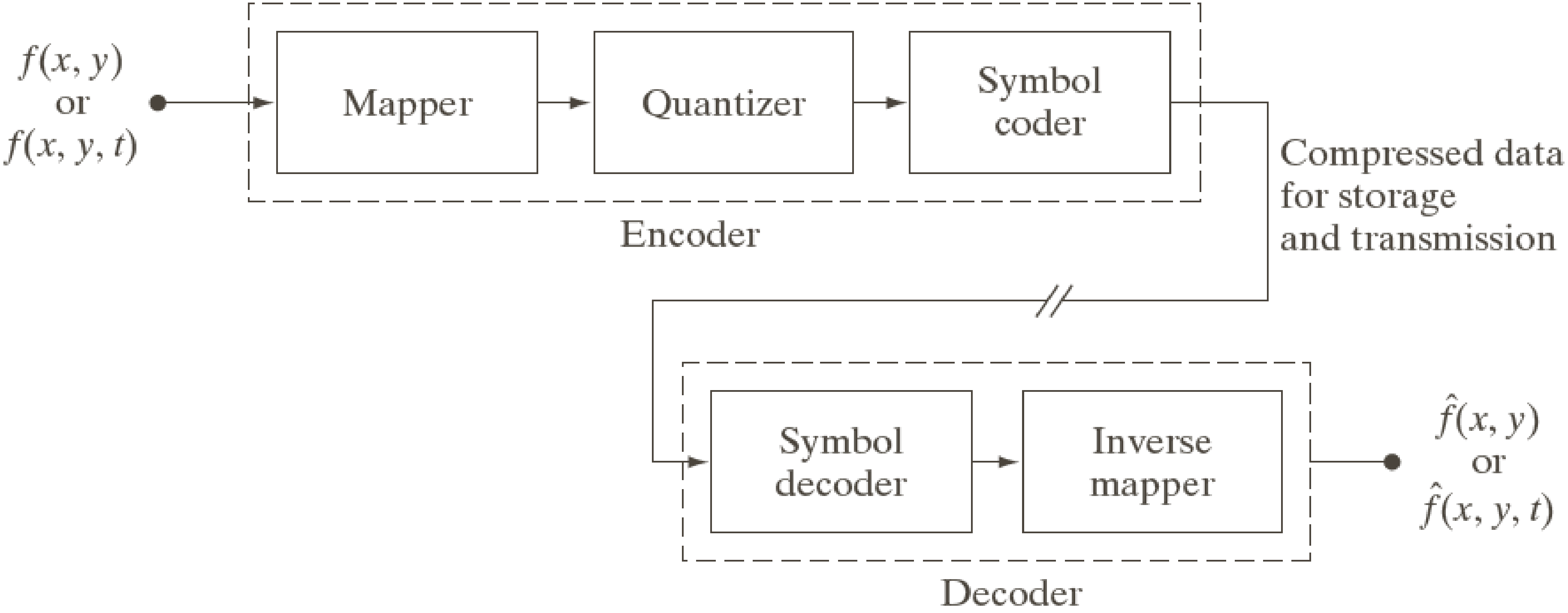


FIGURE 8.5 Functional block diagram of a general image compression system.

DATA REDUNDANCY

- Three basic data redundancies:
  - Coding redundancy
  - Spatial and Temporal redundancy
  - Irrelevant Information

FIGURE 8.1 Computer generated 256 × 256 × 8 bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

CODING REDUNDANCY

- Type of coding (# of bits for each gray level)
- Image histogram:
  - *rk*: Represents the gray levels of an image
  - *pr*(*rk*): Probability of occurrence of *rk*

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L-1$$

  - *l(rk)*: Number of bits used to represent each *rk* (after compression)
  - Lavg: Average # of bits required to represent each pixel:

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

CODING REDUNDANCY

- It makes sense to assign fewer bits to those rk for which pr(rk) are large in order to reduce the sum.
- This achieves data compression and results in a variable length code.
- More probable gray levels will have fewer # of bits.
- Basic type is variable length coding

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)\, p_r(r_k)$$

VARIABLE LENGTH CODING

r_k	$p_r(r_k)$	code1	$l_1(r_k)$	code2	$l_2(r_k)$
$r_0=0$	0.19	000	3	11	2
$r_1=1/7$	0.25	001	3	01	2
$r_2=2/7$	0.21	010	3	10	2
$r_3=3/7$	0.16	011	3	001	3
$r_4=4/7$	0.08	100	3	0001	4
$r_5=5/7$	0.06	101	3	00001	5
$r_6=6/7$	0.03	110	3	000001	6
$r_7=1$	0.02	111	3	000000	6

VARIABLE LENGTH CODING

- Computing L_{avg}

$$L_{avg} = \sum_{k=0}^{7} l_2(r_k)\, p_r(r_k)$$

=2(0.19)+2(0.05)+2(0.21)+3(0.16)+4(0.08)+5(0.06)+6(0.03)+6(0.02)

=2.7 bits

- C_R=3/2.7=1.11
- R_D=1-1/1.11=0.099=9.9%

LOSSLESS GRAY-SCALE IMAGE COMPRESSION

VARIABLE WORD LENGTH CODING: EXAMPLE

- **A 4x4 4bits/pixel original image is given by**

Default Code Book

0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
7: 0111
8: 1000
9: 1001
10: 1010
11: 1011
12: 1100
13: 1101
14: 1110
15: 1111

2	8	6	6
6	8	8	8
8	8	10	10
9	10	10	14

encode

0010	1000	0110	0110
0110	1000	1000	1000
1000	1000	1010	1010
1001	1010	1010	1110

Bit rate = 4bits/pixel

Total # of bits used to represent the image:

4x16 = 64 bits

VARIABLE WORD LENGTH CODING: EXAMPLE

- **Encode the original image with a CODE BOOK given left**

Huffman Code Book

0: 0000000
1: 0000001
2: 0001
3: 0000010
4: 0000011
5: 0000100
6: 01
7: 0000101
8: 10
9: 00100
10: 11
11: 0000110
12: 0000111
13: 001010
14: 0011
15: 001011

2	8	6	6
6	8	8	8
8	8	10	10
9	10	10	14

encode

0001	10	01	01
01	10	10	10
10	10	11	11
00100	11	11	0011

Total # of bits used to represent the image:

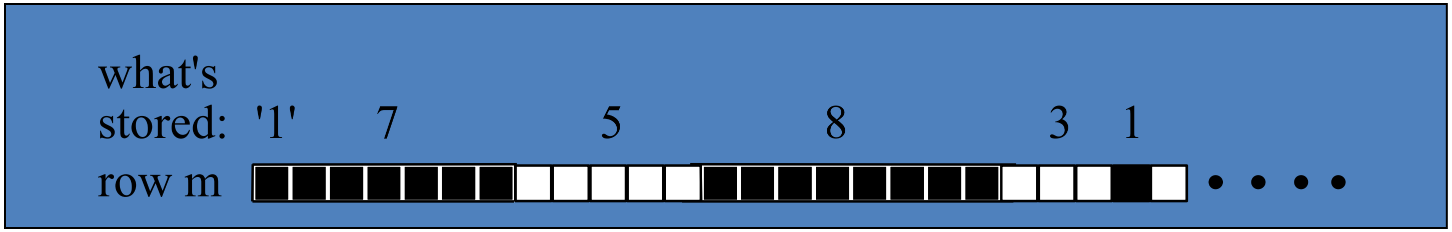
4+2+2+2+2+2+2+2+2+2+
2+2+5+2+2+4
= 39 bits

Bit rate = 39/16
= 2.4375 bits/pixel

CR = 64/39 = 1.6410

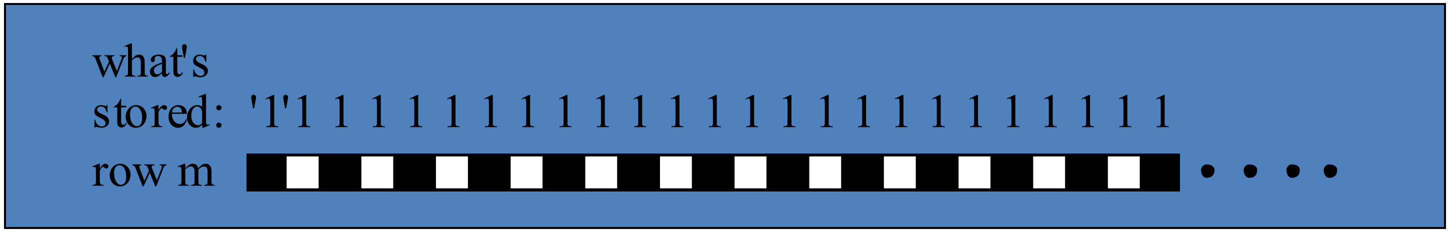
RUN LENGTH CODING

- Run Length
  - **The length of consecutively identical symbols**
- Run length Coding Example



what's
stored: '1' 7 5 8 3 1
row m • • • •

- When Does it Work?
  - **Images containing many runs of 1's and 0's**
- When Does it Not Work?



what's
stored: '1'1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
row m • • • •

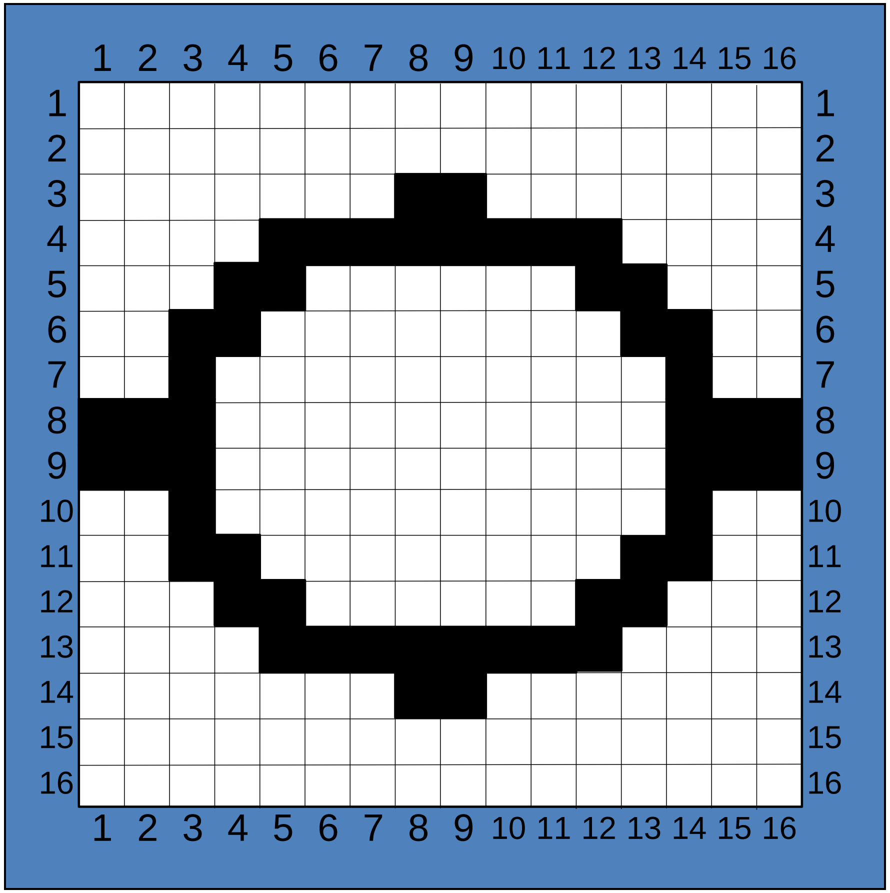
RUN LENGTH CODING

- Decoding Example

A binary image is encoded using run length code row by row, where “0” represents white, and “1” represents black. The code is given by

Row 1: “0”, 16
Row 2: “0”, 16
Row 3: “0”, 7, 2, 7
Row 4: “0”, 4, 8, 4
Row 5: “0”, 3, 2, 6, 3, 2
Row 6: “0”, 2, 2, 8, 2, 2
Row 7: “0”, 2, 1, 10, 1, 2
Row 8: “1”, 3, 10, 3
Row 9: “1”, 3, 10, 3
Row 10: “0”, 2, 1, 10, 1, 2
Row 11: “0”, 2, 2, 8, 2, 2
Row 12: “0”, 3, 2, 6, 3, 2
Row 13: “0”, 4, 8, 4
Row 14: “0”, 7, 2, 7
Row 15: “0”, 16
Row 16: “0”, 16

decode →



Entropy

Entropy calculation for a two symbol alphabet.

Example 1: A $p_A=0.5$

B $p_B=0.5$

$$\mathrm{H}(\mathrm{A},\mathrm{B}) = -\mathrm{p_A}\log_2 \mathrm{p_A} - \mathrm{p_B}\log_2 \mathrm{p_B} =$$
$$= -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$$

It requires one bit per symbol on the average to represent the data.

Example 2: A $p_A=0.8$

B $p_B=0.2$

$$H(A,B) = -p_A\log_2 p_A - p_B\log_2 p_B =$$
$$= -0.8\log_2 0.8 - 0.2\log_2 0.2 \cong 0.7219$$

It requires less than one bit pe symbol on the average to represent the data.

How can we code this ?

HUFFMAN CODING

- Uses frequencies (Probability) of symbols in a string to build a variable rate prefix code.
- Each symbol is mapped to a binary string.
- More frequent symbols have shorter codes.
- No code is a prefix of another. (Uniquely decodable)

HUFFMAN CODING

- Each symbol is assigned a variable-length code, depending on its frequency. The higher its frequency, the shorter the codeword
- Number of bits for each codeword is an integral number
- A prefix code
- A variable-length code
- Huffman code is the **optimal** prefix and variable-length code, **given** the symbols' probabilities of occurrence
- Codewords are generated by building a Huffman Tree

HUFFMAN CODING

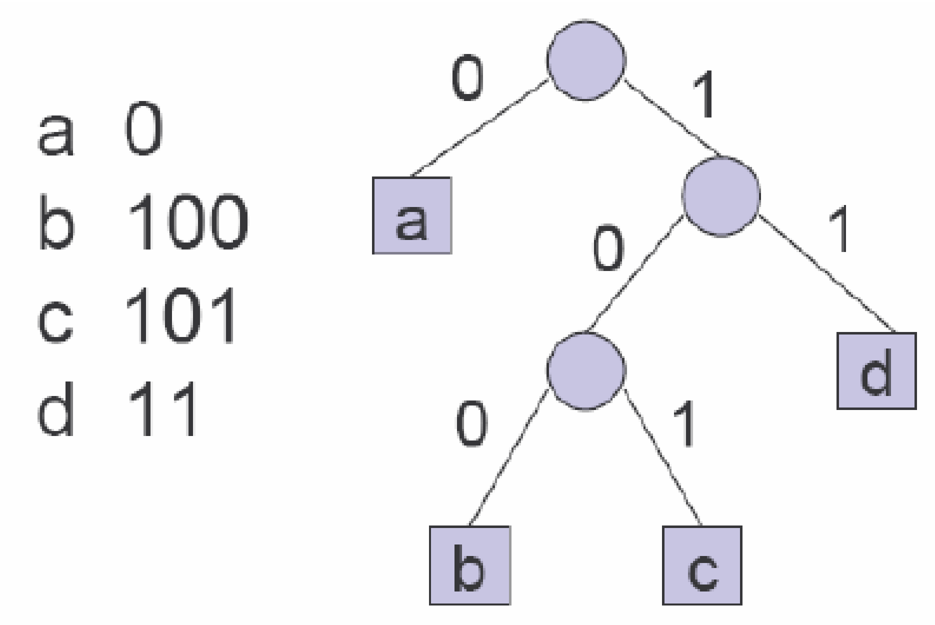
- **Prefix code** - No codeword is a prefix of any other codeword.

  A = 0;   B = 10;   C = 110; D = 111

- **Uniquely decodable code** - Has only one possible source string producing it.
  - Unambigously decoded
  - Examples:
    - Prefix code - the end of a codeword is immediately recognized without ambiguity: 010011001110 → 0 | 10 | 0 | 110 | 0 | 111 | 0
    - Fixed-length code

HUFFMAN CODING

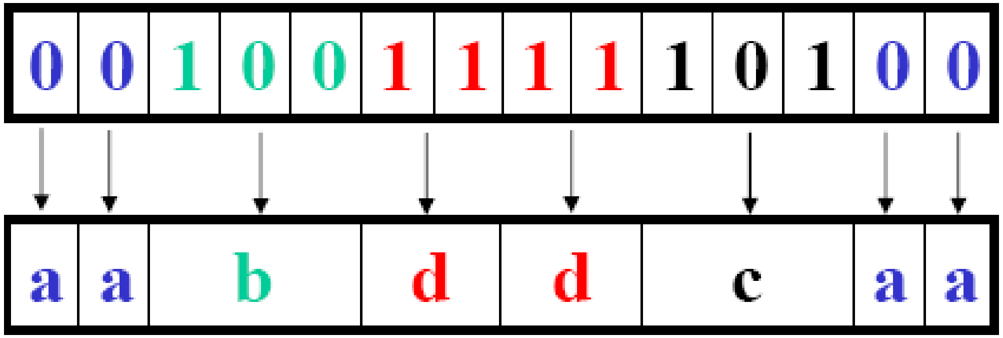
- Example:
  - We have four symbols a, b, c, and d.



a	0
b	100
c	101
d	11

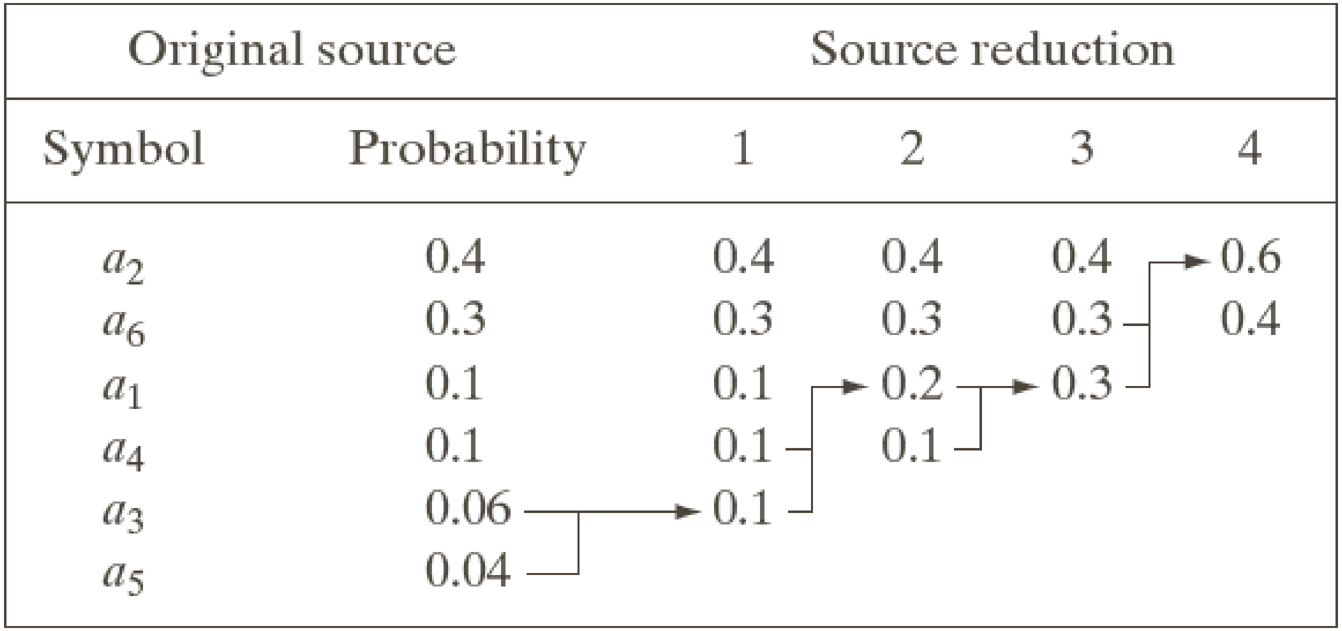
HUFFMAN CODING

- A source string: **aabddcaa**
  - Fixed Length Coding: 16 bits (ordinary coding)
    - **00 00 01 11 11 10 00 00**
- Variable length coding: 14 bits (Huffman coding)
  - **0 0 100 11 11 101 0 0**
- Uniquely Decodable:



HUFFMAN CODING

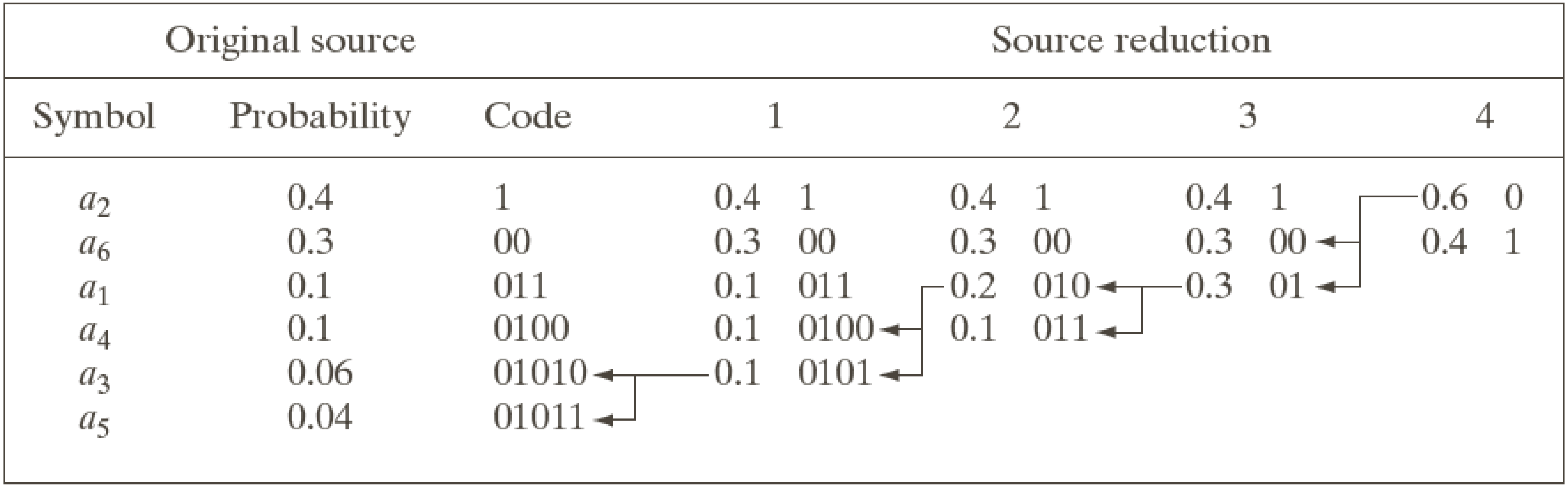
- Step-1
  - Arrange probability in decreasing order and consider them as tree leaves
- Step-2
  - Merge two nodes with smallest prob. to a new node and sum up prob.
  - Arbitrarily assign 1 and 0 to each pair of merging branch
- Step-3
  - Repeat until no more than one node left.
  - Read out codeword sequentially from root to leaf



Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

FIGURE 8.7
Huffman source reductions.

HUFFMAN CODING



Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

FIGURE 8.8
Huffman code assignment procedure.

Readings from Book (3^{rd} Edn.)

- Image Compression (Chapter-8)

Acknowledgements

Material in these slides has been taken from, the following resources

- Digital Image Processing”, Rafael C. Gonzalez & Richard E. Woods, Addison-Wesley, 2002