CSE4227 Digital Image Processing

Chapter 8 – **Image Compression**

Dr. Kazi A Kalpoma
Professor, Department of CSE

Ahsanullah University of Science & Technology (AUST)

Contact: kalpoma@aust.edu

Google Class code: bux3jc2

Today's Contents

- ❑ Why do we need compression?
- ❑ Compression Ratio and Relative Data Redundancy
- ❑ How can we implement compression?
    - ❑ **Coding redundancy**
    - ❑ **Spatial and temporal redundancy**
    - ❑ **Irrelevant information**
- ❑ Image Entropy
- ❑ Lossy and Loss-free Compression
- ❑ Different Compression Techniques
    - ❖ Huffman coding
    - ❖ Golomb coding
    - ❖ Arithmetic coding
    - ❖ LZW coding
    - ❖ Run length coding

•Chapter 8 from R.C. Gonzalez and R.E. Woods, Digital Image Processing (3rd Edition), Prentice Hall, 2008 [**Section 8.1, 8.2**]

Fundamentals

- The term **Data Compression** refers to the process of reducing the amount of data required to represent a given quantity of information
- **Data** and **Information** are not the same thing in image
- Various amount of data can be used to represent the same information
- Data might contain elements that provide **irrelevant** or **repeated** information : ***Data Redundancy***
- **Data redundancy** is a central issue in image compression.
- It is not an abstract concept but mathematically quantifiable entity

Why do we need compression?

- Data storage
- Data transmission

Applications that require image compression are many and varied such as:

1. Internet,
2. Businesses,
3. Multimedia,
4. Satellite imaging,
5. Medical imaging
6. etc.

Why Images Are Compressed?

- Let a SD TV 2-hour color sequence
  - each of size 720X480
  - frame rate: 30 frames/sec

Why Images Are Compressed?

- Let a SD TV 2-hour color sequence
  - each of size 720X480
  - frame rate: 30 frames/sec
- For a single second, the amount of data to be accessed is

$$30\,frames \times (720 \times 480)\frac{pixels}{frame} \times 3\frac{bytes}{pixel} \approx 31MB$$

Why Images Are Compressed?

- For a 2-hour long SD TV video, the data is

$$31{,}104{,}000\,\frac{bytes}{\sec}\times(60)^2\,\frac{\sec}{hour}\times 2\ \ hours$$

$$\approx 2.24\times 10^{11}\ \ bytes$$

$$= 224\ \mathrm{GB}$$

- Twenty seven 8.5 GB dual-layer DVDs (12 cm disks) are needed to store it.
- To put on a single DVD, must be compressed by a factor of 26.3

Why Images Are Compressed?

- *High Definition* (HD) TV color sequence
  - each of size 1920X1080 pixels

  Compression must be higher

Why Images Are Compressed?

- *High Definition* (HD) TV color sequence
  - each of size 1920X1080 pixels

  Compression must be higher

- Web Images
  - color images of size 128 X 128 through 56Kbps to 12Mbps connections

  Requires 7.0 to 0.03 seconds to download

Why Images Are Compressed?

- *High Definition* (HD) TV color sequence
  - each of size 1920X1080 pixels

  Compression must be higher
- Web Images
  - color images of size 128 X 128 through 56Kbps to 12Mbps connections

  Requires 7.0 to 0.03 seconds to download
- 1 GB Flash memory and 8 Megapixel Digital camera

  Can store at most ~41 uncompressed images

Data Redundancy

- Let n_1 and n_2 denote the number of information carrying units in two data sets that represent the same information

- The **relative redundancy** R_D is define as :

$$R_D = 1 - \frac{1}{C_R}$$

where C_R, commonly called the **compression ratio**, is

$$C_R = \frac{n_1}{n_2}$$

If $n_1 = n_2$, $C_R = 1$ and $R_D = 0$ *no redundancy*

If $n_1 >> n_2$, C_R □ ∞ and R_D □ *1 high redundancy*

If $n_1 << n_2$, C_R □ 0 and R_D □ ∞ *undesirable*

- *A* compression ration of 10 (10:1) means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second (compressed) data set.

How can we implement compression?

- **Coding redundancy**
  Most 2-D intensity arrays contain more bits than are needed to represent the intensities

- **Spatial and temporal redundancy**
  Pixels of most 2-D intensity arrays are correlated spatially and video sequences are temporally correlated

- **Irrelevant information**
  Most 2-D intensity arrays contain information that is ignored by the human visual system

Coding Redundancy

- ❑ Code is system of symbols to represent info
- ❑ Codeword or sequence of symbols represents a piece of info
- ❑ The number of symbols required is its length
- ❑ Try to minimize codeword length
- Variable length Codeword representations for a piece of info

Optimal information Coding

- Recall from the histogram calculations

$$p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n}$$

where *$p(r_k)$* is the probability of a pixel to have a certain value *r_k*

- If the number of bits used to represent r_k is ***$l(r_k)$***, then

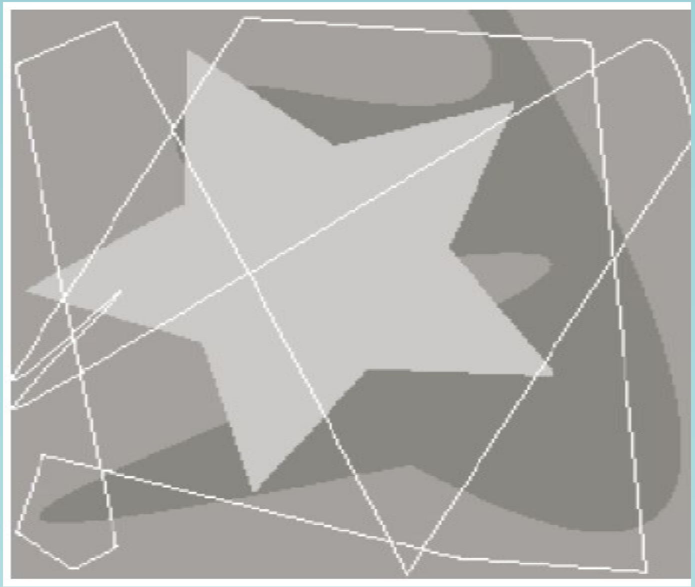
average number of bits required to represent each pixel is

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

Total bit requirment is $MNL_{avg} = MN \sum_{k=0}^{k=L-1} l(r_k)p(r_k)$

Coding Redundancy: Example

Computer generated 256x256x8 bit image with coding redundancy



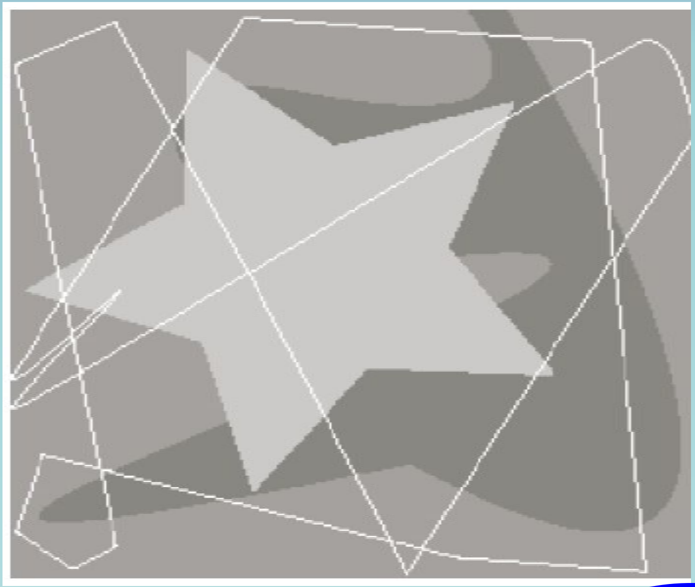
Total Bit required is

=256X256X8

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$
$r_{87} = 87$	0.25	01010111	8
$r_{128} = 128$	0.47	10000000	8
$r_{186} = 186$	0.25	11000100	8
$r_{255} = 255$	0.03	11111111	8
r_k for $k \neq 87, 128, 186, 255$	0	—	8

Coding Redundancy: Example

Computer generated 256x256x8 bit image with coding redundancy



Total =256X256X1.81 bits

C =8/1.81=~4.42

R=1-1/C=.774

77.4 %

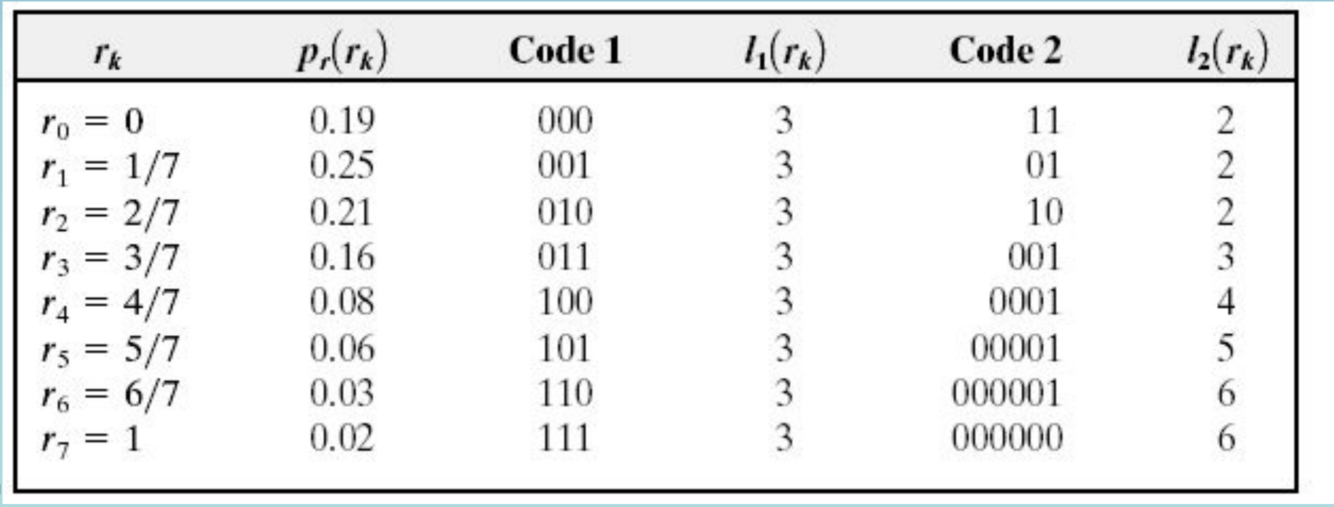
L_{avg} =.25X2+.47X1+.25X3+.03X3=1.81 bits

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

Coding Redundancy

Variable-Length Coding

2nd Example:



r_k	$p_r(r_k)$	**Code 1**	$l_1(r_k)$	**Code 2**	$l_2(r_k)$
$r_0 = 0$	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

$$L_{av} = \sum_{k=0}^{7} l(r_k)(p(r_k)$$

$$= 2(019) + 2(0.25) + 3(0.16) + \ldots + 6(0.02)$$

$$= 2.7\ bits$$

$$C_R = \frac{3}{2} = 1.11$$

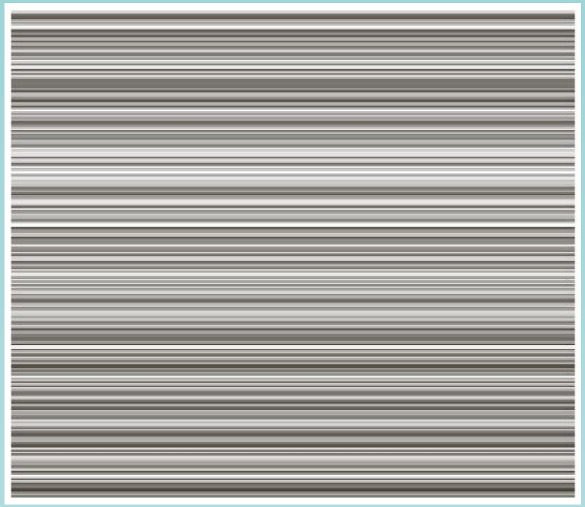
$$R_D = 1 - \frac{1}{1.11} = 0.099$$

9.9% data redundant.

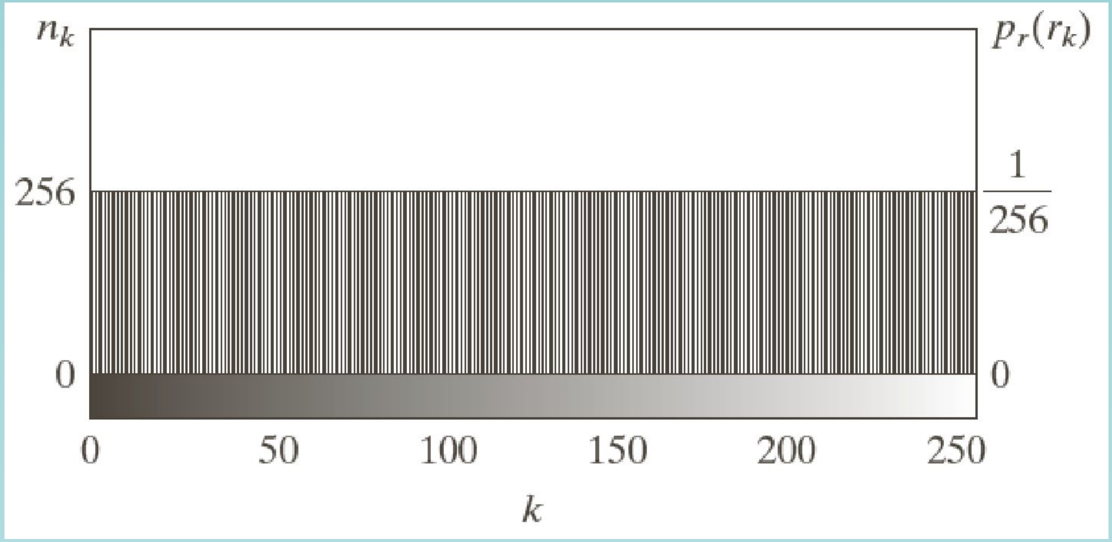
Spatial Redundancy

- Intensities of each pixel may correlate to its neighbors
- Many info is *unnecessarily* replicated
- Try to **minimize unnecessary data**

Spatial and Temporal Redundancy Example



Computer generated 256x256x8 bit image with spatial redundancy



Histogram

- Histogram is equiprobable
- Vertically, pixels are independent
- Horizontally, they are maximally correlated
- The image cannot be compressed using variable length coding

Spatial and Temporal Redundancy Example

Computer generated 256x256x8 bit image with spatial redundancy

Original Bit required

=256X256X8

In Run length

=(256+256)X8

C =(256X256X8)/(256+256)X8

=128:1

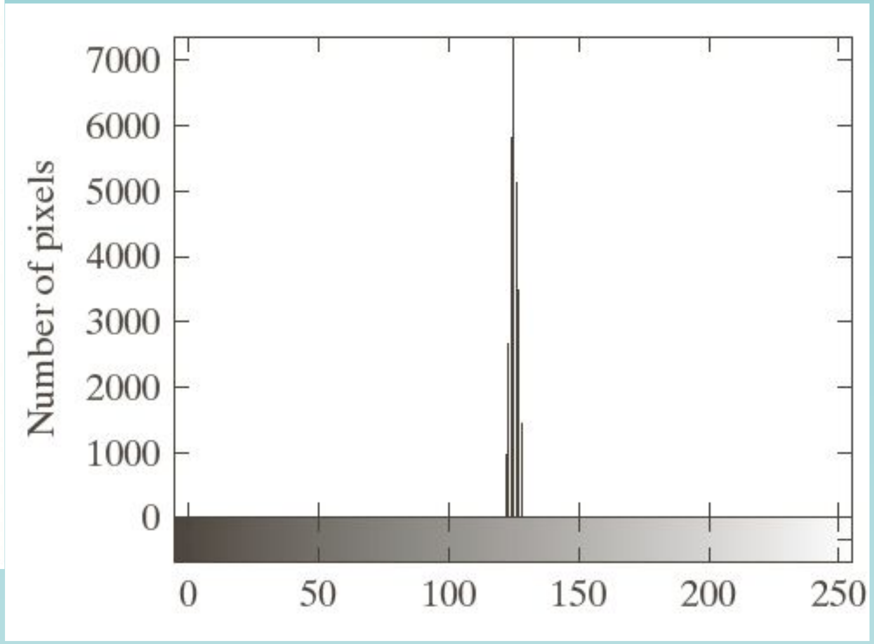
- The image can be represented as a sequence of run length pair : *intensity value* and *num of pixels with that intensity*

Irrelevant Information

- Many images has info that are ignored by the *Human Visual System* (HVS)
- Try to **remove these extraneous data**

Irrelevant Information

Computer generated 256x256x8 bit image with irrelevant information



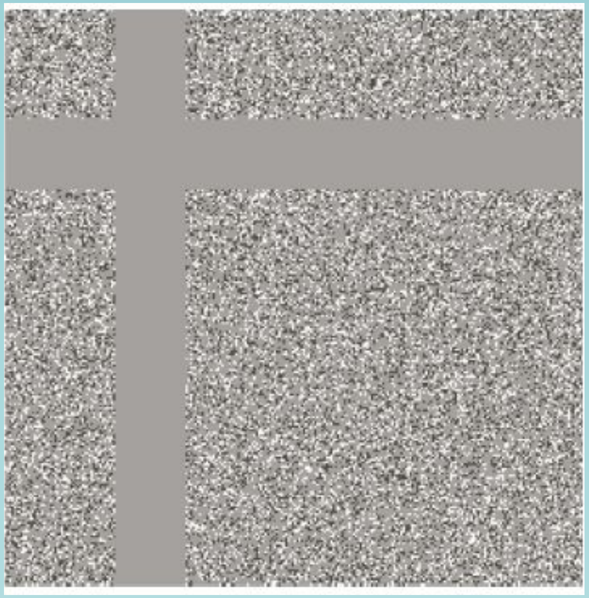
- Can be represented using a single average gray level.

C =256X256X8:8

=65536:1

Irrelevant Information

Computer generated 256x256x8 bit image with irrelevant information



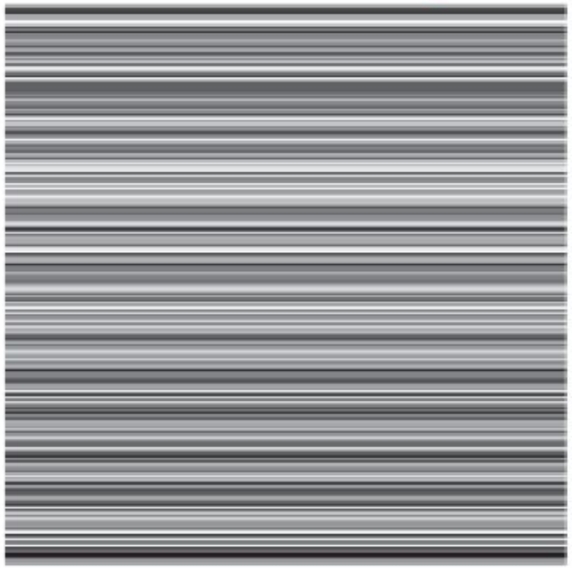
- However, this coarse quantization may remove useful *invisible* information

Data redundancy in images

Coding redundancy

Does not need all 8 bits

Spatial redundancy



Information is unnecessarily replicated

Irrelevant information

Information is not useful

Redundancies We Tried to Remove

- *Coding redundancy*
- *Spatial redundancy*
- *Irrelevant information*

How many bits do we really need to represent image-information?

Information theory: does it have any answer?

Measuring Image Information
Information Theory Review

A random event E with probability $P(E)$
carries $I(E)$ units of information,

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

- *Higher the uncertainty, higher the information content*

Information theory

If we have a source of random events from a discrete set of events $\{a_1, a_2, a_3, \ldots, a_J\}$ with probabilities, $P(a_1), P(a_2), P(a_3), \ldots, P(a_J)$ then the average information per event or the entropy of the source,

$$H = \sum_{j=1}^{J} -P(a_j)\log P(a_j)$$

Calculating Image Entropy

If we consider the pixel intensities as random events, then the intensity histogram is the approximation of the probabilities

$$\widetilde{H} = \sum_{k=1}^{L-1} - P_r(r_k) \log P(r_k)$$

✔*Entropy* is the measurement of the average information in an image

Image information

- Entropy

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2(p_r(r_k))$$

where

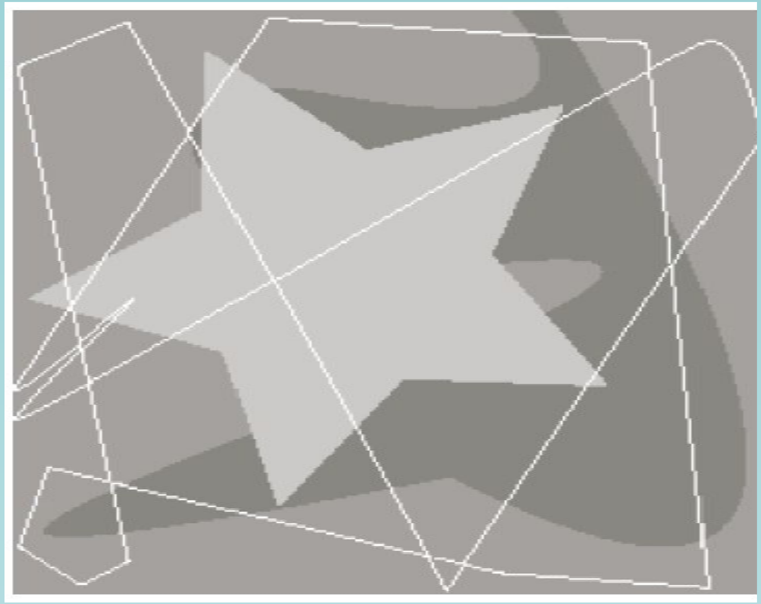
L is the number of intensity or gray levels
r_k is input image intensity or gray level value k
$p_r(r_k)$ is normalized histogram of input image

  - It is not possible to encode input image with fewer than $\tilde{H}$ bits/pixel

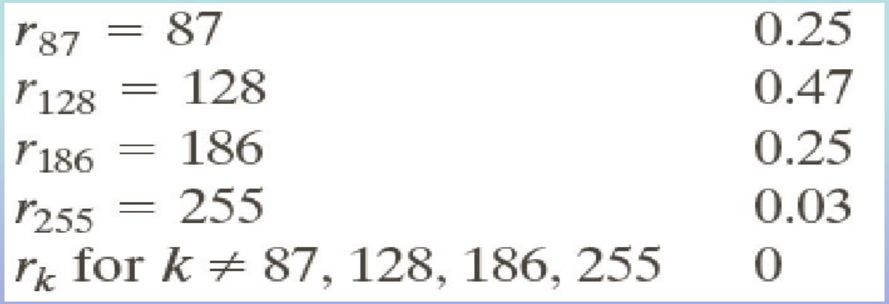
Image Entropy

- Image entropy is a quantity which is used to describe the amount of information which must be coded for by a compression algorithm.
- Low entropy images, such as those containing a lot of black sky, have very little contrast and large runs of pixels with the same or similar DN values.
- An image that is perfectly flat will have an entropy of zero. Consequently, they can be compressed to a relatively small size.
- On the other hand, high entropy images such as an image of heavily cratered areas on the moon have a great deal of contrast from one pixel to the next and consequently cannot be compressed as much as low entropy images.

Image Entropy: Example

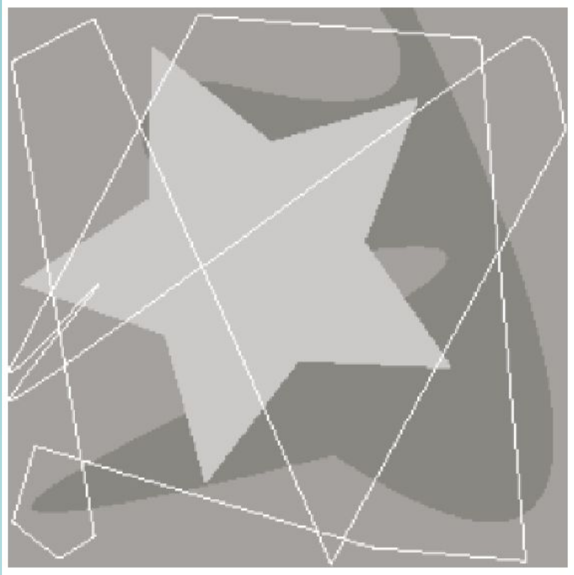


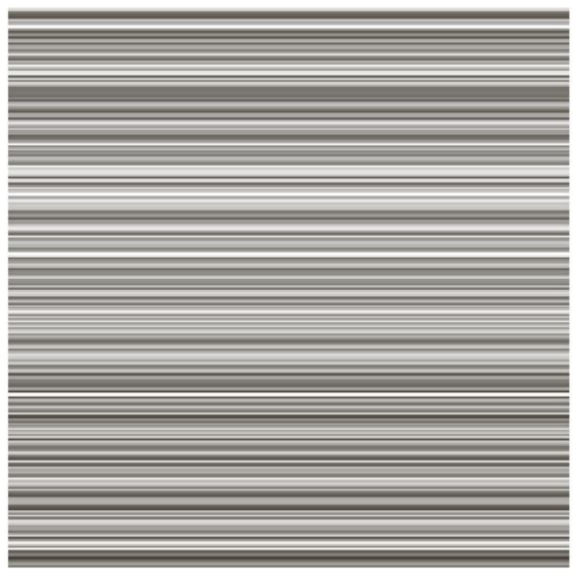
$$
\begin{aligned}
\widetilde{H} &= \sum_{k=1}^{L-1} - P_r(r_k) \log P(r_k) \\
&= -[0.25 \log_2 0.25 + 0.47 \log_2 0.47 \\
&\quad + 0.25 \log_2 0.25 + 0.03 \log_2 0.03] \\
&\approx -[0.25(-2) + 0.47(-1.09) \\
&\quad + 0.25(-2) + 0.03(-5.06)] \\
&\approx 1.6614 \text{ bits/pixel}
\end{aligned}
$$

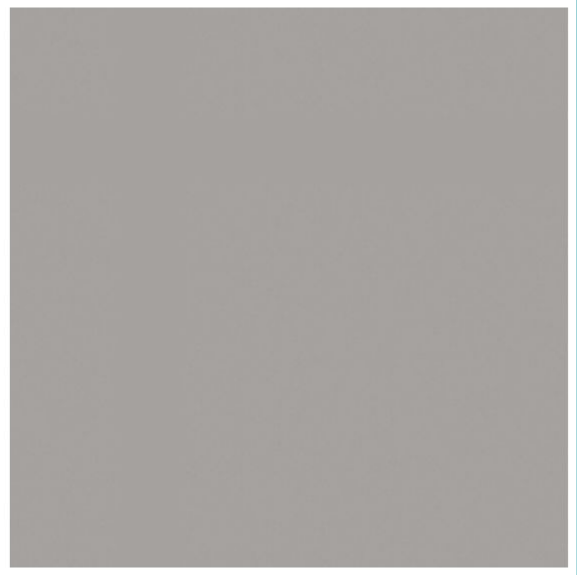


r_k	$P_r(r_k)$
$r_{87} = 87$	0.25
$r_{128} = 128$	0.47
$r_{186} = 186$	0.25
$r_{255} = 255$	0.03
r_k for $k \neq 87, 128, 186, 255$	0

Image Entropy: Example

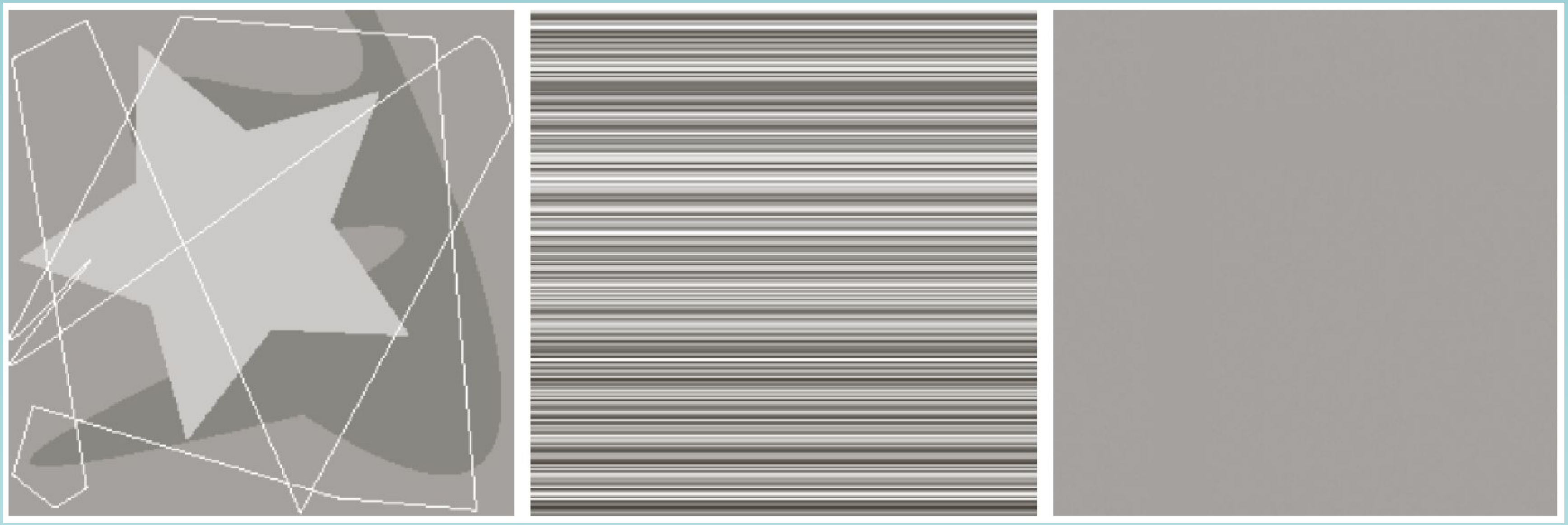

H=1.6614


H=8


H=1.566

✔*The amount of Entropy* and thus information in an image is far from intuitive.

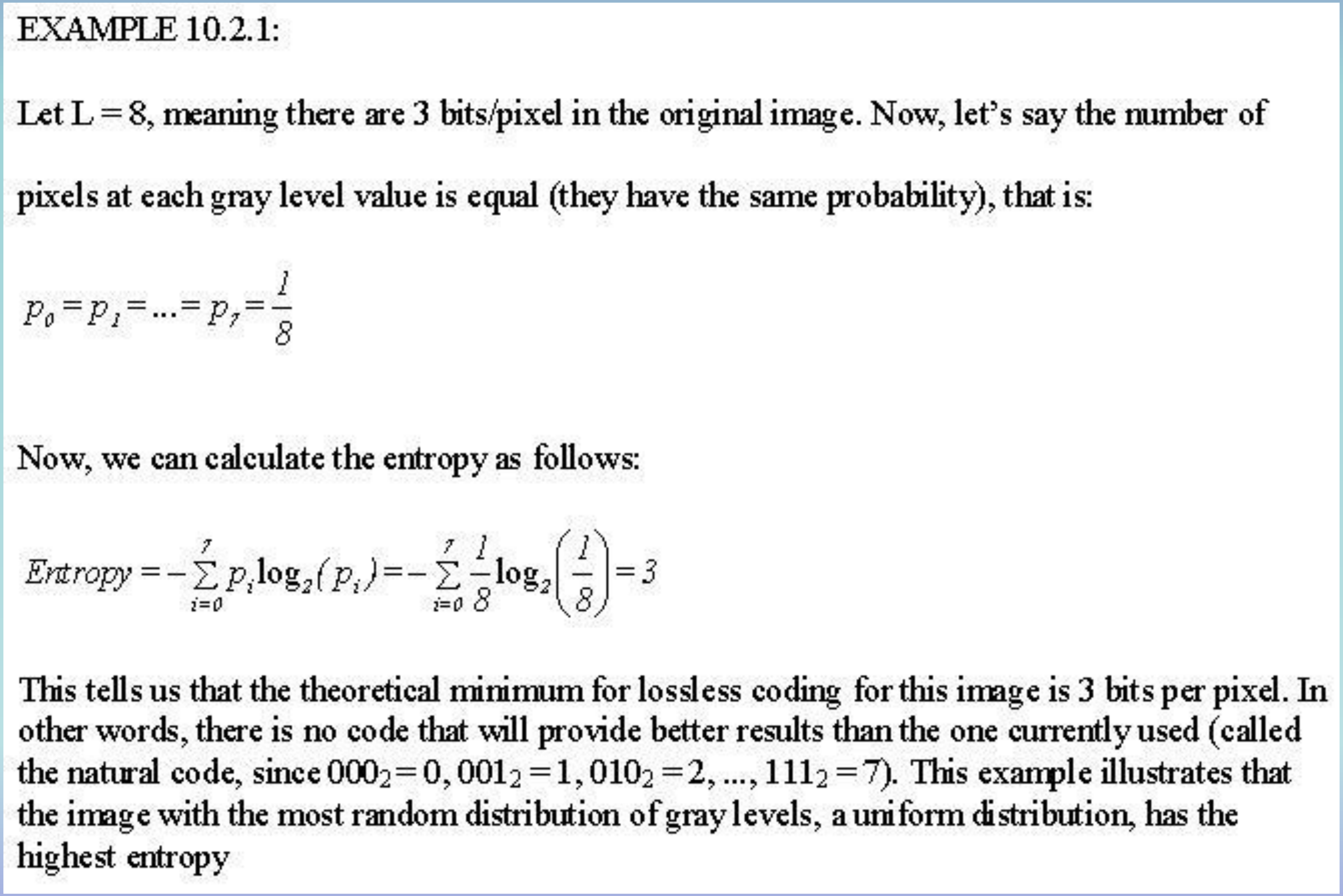
Image Entropy: Example



H=1.6614 *H*=8 *H*=1.566

- Visual perception and entropy (information) are far from compliance

Entropy - Example



EXAMPLE 10.2.1:

Let L = 8, meaning there are 3 bits/pixel in the original image. Now, let's say the number of pixels at each gray level value is equal (they have the same probability), that is:

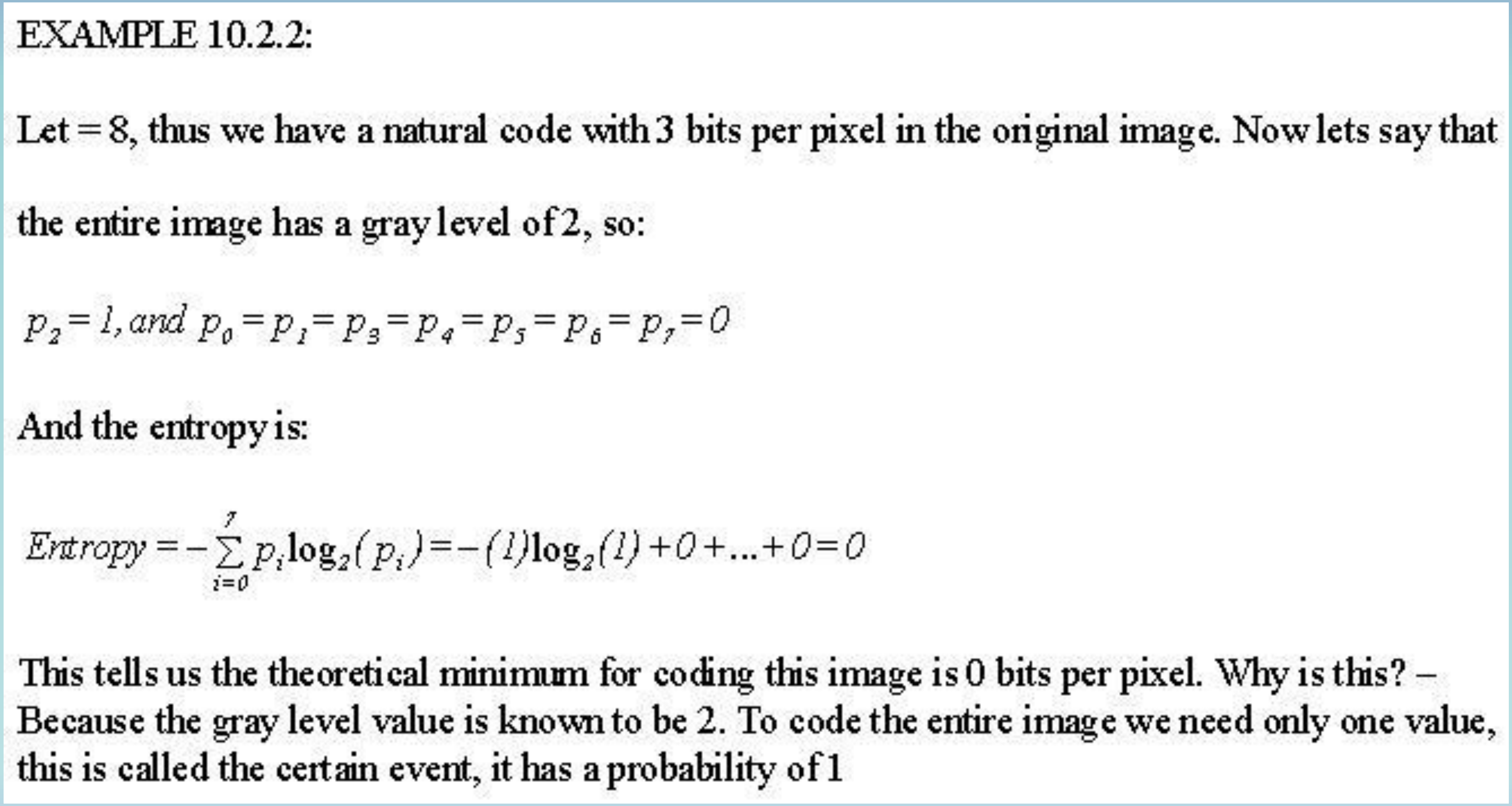
$$p_0 = p_1 = \ldots = p_7 = \frac{1}{8}$$

Now, we can calculate the entropy as follows:

$$Entropy = -\sum_{i=0}^{7} p_i \log_2(p_i) = -\sum_{i=0}^{7} \frac{1}{8} \log_2\left(\frac{1}{8}\right) = 3$$

This tells us that the theoretical minimum for lossless coding for this image is 3 bits per pixel. In other words, there is no code that will provide better results than the one currently used (called the natural code, since $000_2 = 0, 001_2 = 1, 010_2 = 2, \ldots, 111_2 = 7$). This example illustrates that the image with the most random distribution of gray levels, a uniform distribution, has the highest entropy

Entropy - Example



EXAMPLE 10.2.2:

Let = 8, thus we have a natural code with 3 bits per pixel in the original image. Now lets say that the entire image has a gray level of 2, so:

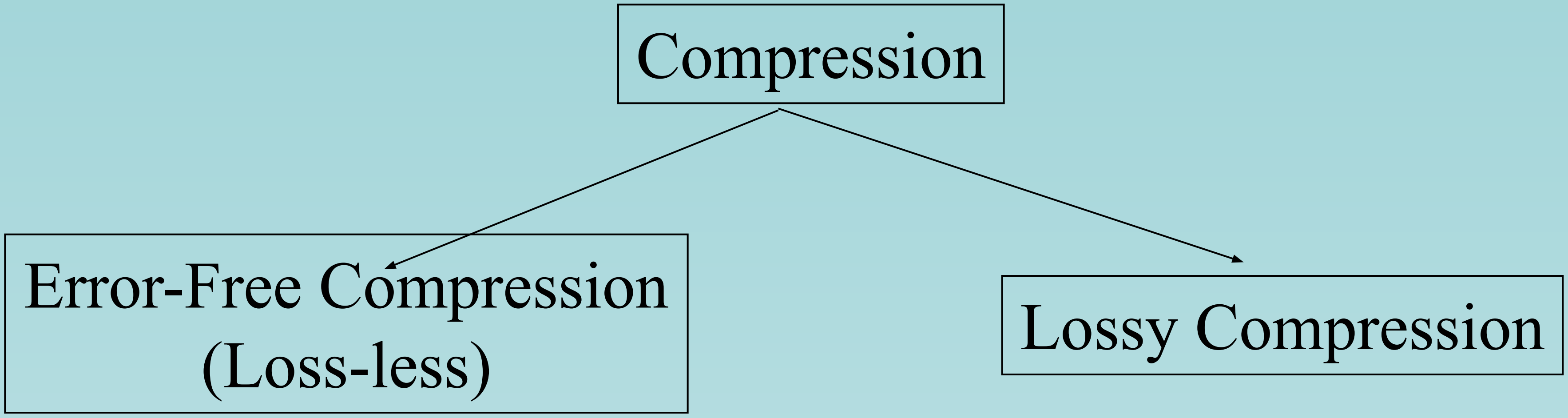
$$p_2 = 1, and\ p_0 = p_1 = p_3 = p_4 = p_5 = p_6 = p_7 = 0$$

And the entropy is:

$$Entropy = -\sum_{i=0}^{7} p_i \log_2(p_i) = -(1)\log_2(1) + 0 + \ldots + 0 = 0$$

This tells us the theoretical minimum for coding this image is 0 bits per pixel. Why is this? – Because the gray level value is known to be 2. To code the entire image we need only one value, this is called the certain event, it has a probability of 1

Compression Types



Different Error Free Compression Techniques

- Variable length coding
  - Huffman coding
  - Golomb coding
  - Arithmetic coding

- LZW coding
- Bit plane coding
  - Constant area coding
  - Run length coding

Huffman Coding

- The most popular technique for removing coding redundancy is due to Huffman (1952)
- Huffman Coding yields the smallest number of code symbols per source symbol
- The resulting code is *optimal*

Steps for Huffman Algorithm

- The Huffman algorithm can be described in five steps:

1. Find the gray level probabilities for the image by finding the histogram
2. Order the input probabilities (histogram magnitudes) from largest to smallest
3. Combine the smallest two by addition
4. GOTO step 2, until only two probabilities are left
5. By working backward along the tree, generate code by alternating assignment of 0 and 1

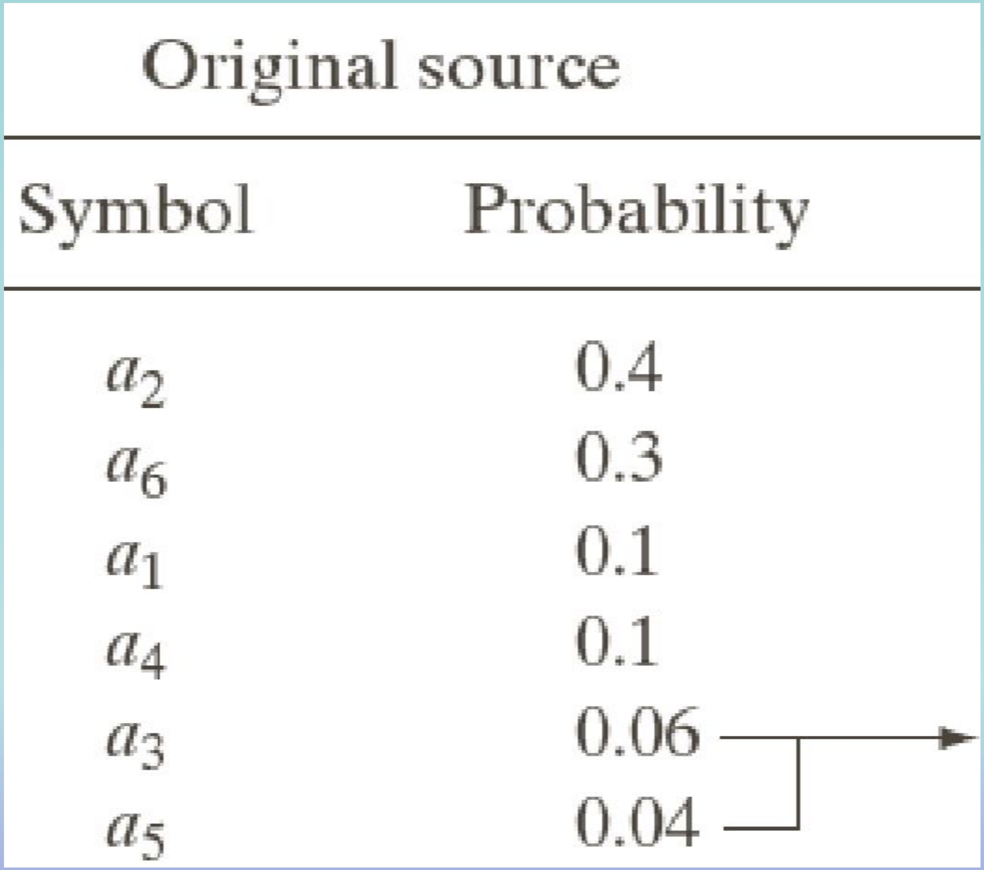
Huffman Coding: Example

Consider a 3 bit image with the following probabilities:

Source Symbol	Probability
a_1	0.1
a_2	0.4
a_3	0.06
a_4	0.1
a_5	0.04
a_6	0.3

Huffman Coding: Example

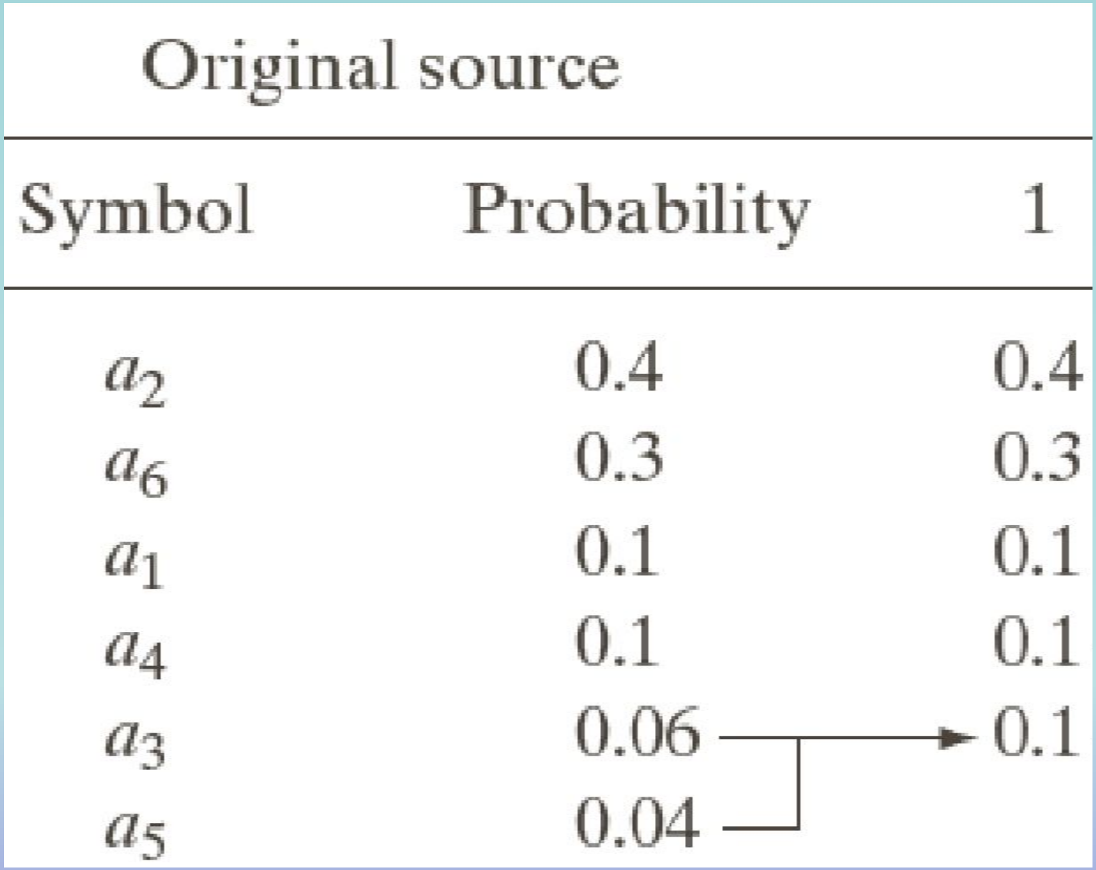
Consider a 3 bit image with the following probabilities:



Original source	
Symbol	Probability
a_2	0.4
a_6	0.3
a_1	0.1
a_4	0.1
a_3	0.06
a_5	0.04

Huffman Coding: Example

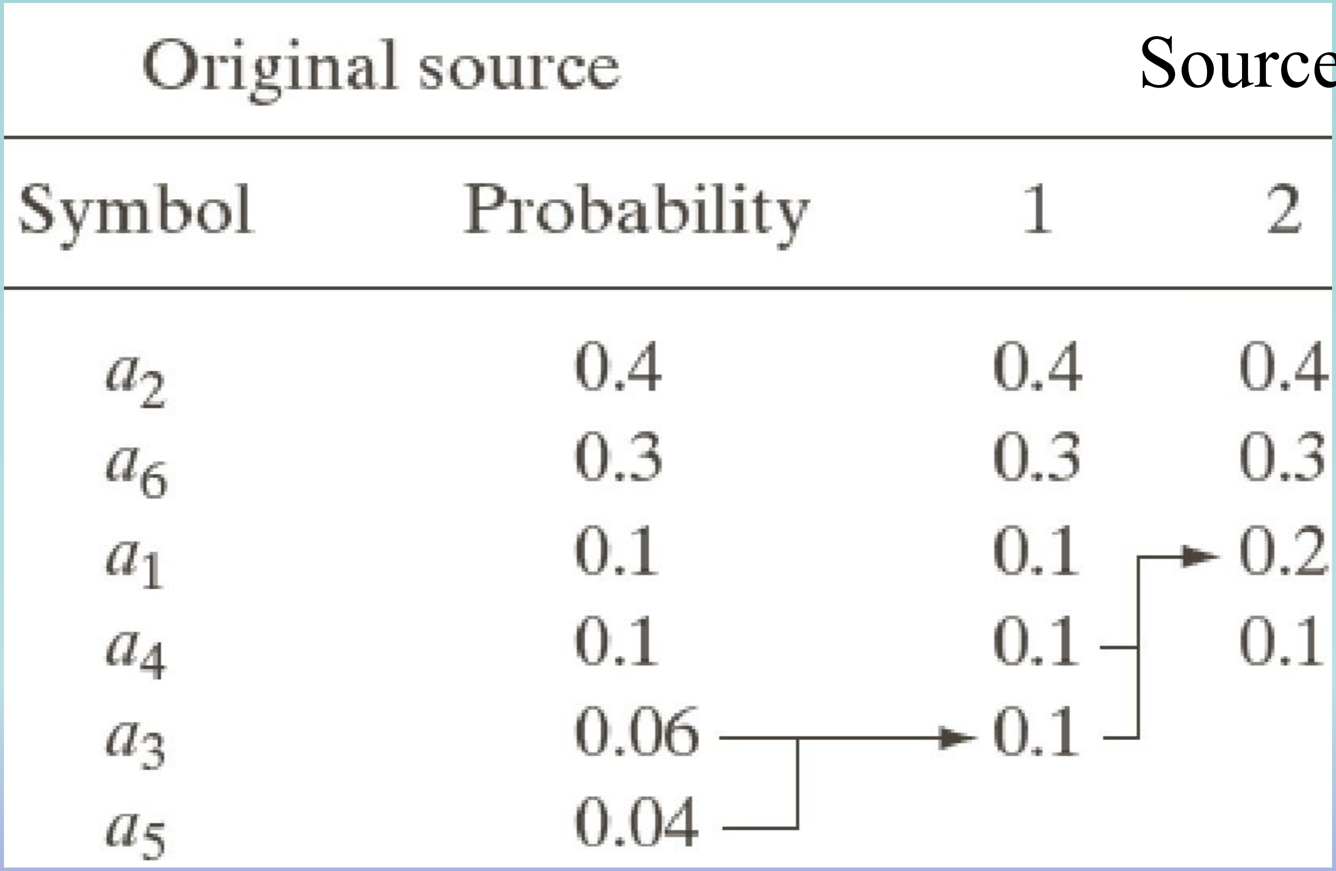
Consider a 3 bit image with the following probabilities:



Original source		
Symbol	Probability	1
a_2	0.4	0.4
a_6	0.3	0.3
a_1	0.1	0.1
a_4	0.1	0.1
a_3	0.06	0.1
a_5	0.04	

Huffman Coding: Example

Consider a 3 bit image with the following probabilities:

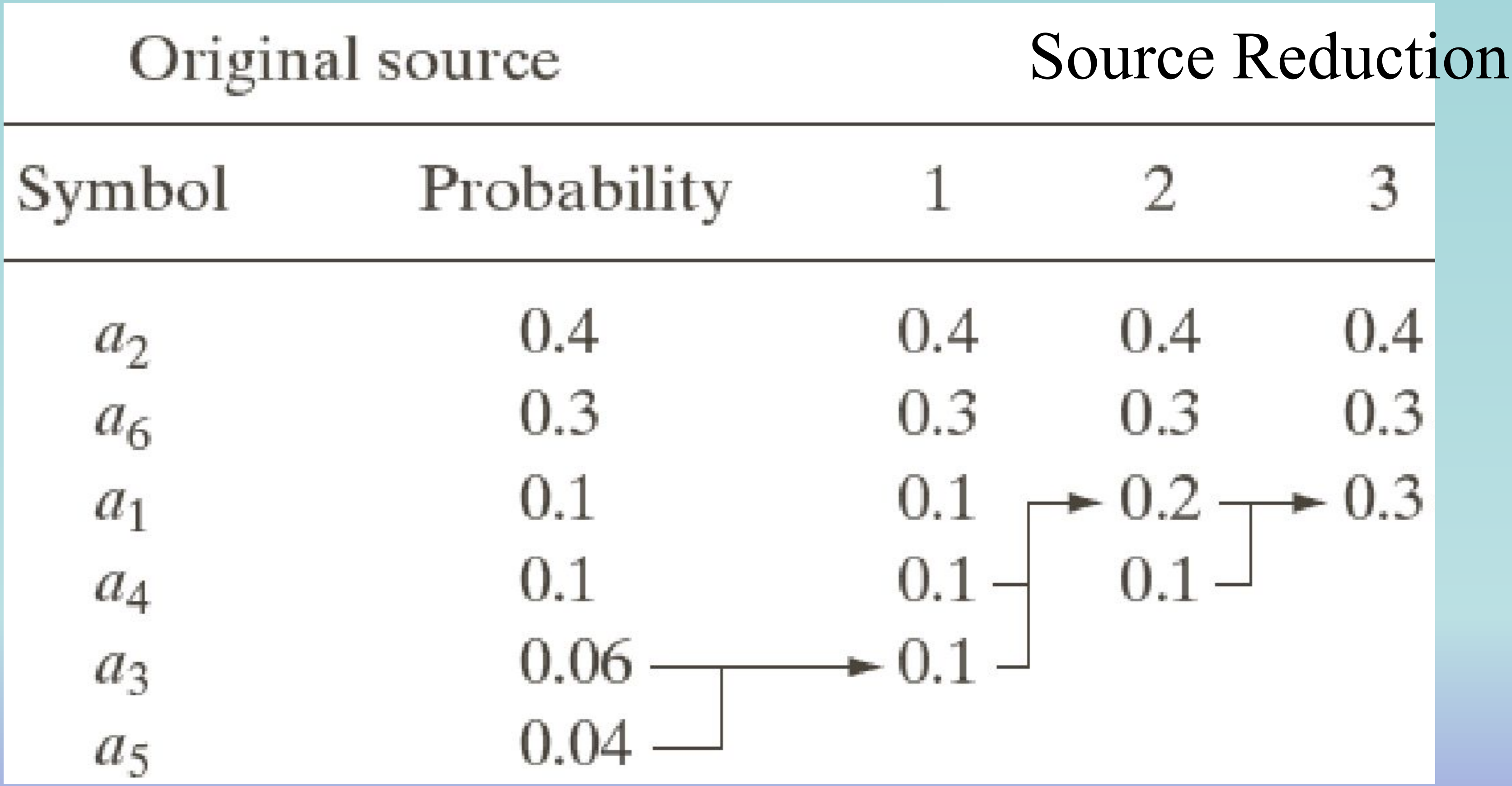


Source Reduction

Original source			
Symbol	Probability	1	2
a_2	0.4	0.4	0.4
a_6	0.3	0.3	0.3
a_1	0.1	0.1	0.2
a_4	0.1	0.1	0.1
a_3	0.06	0.1	
a_5	0.04		

Huffman Coding: Example

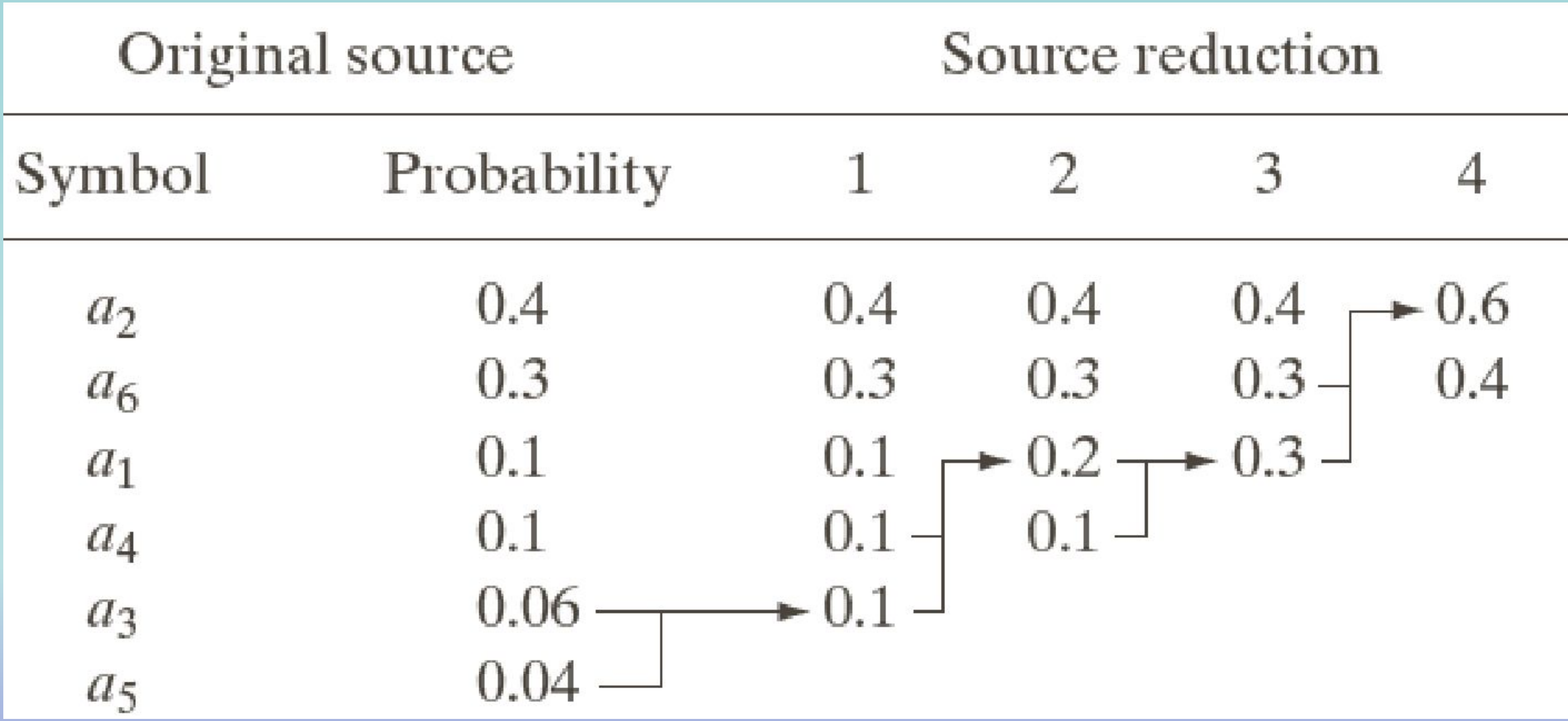
Consider a 3 bit image with the following probabilities:



Original source		Source Reduction		
Symbol	Probability	1	2	3
a_2	0.4	0.4	0.4	0.4
a_6	0.3	0.3	0.3	0.3
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			

Huffman Coding: Example

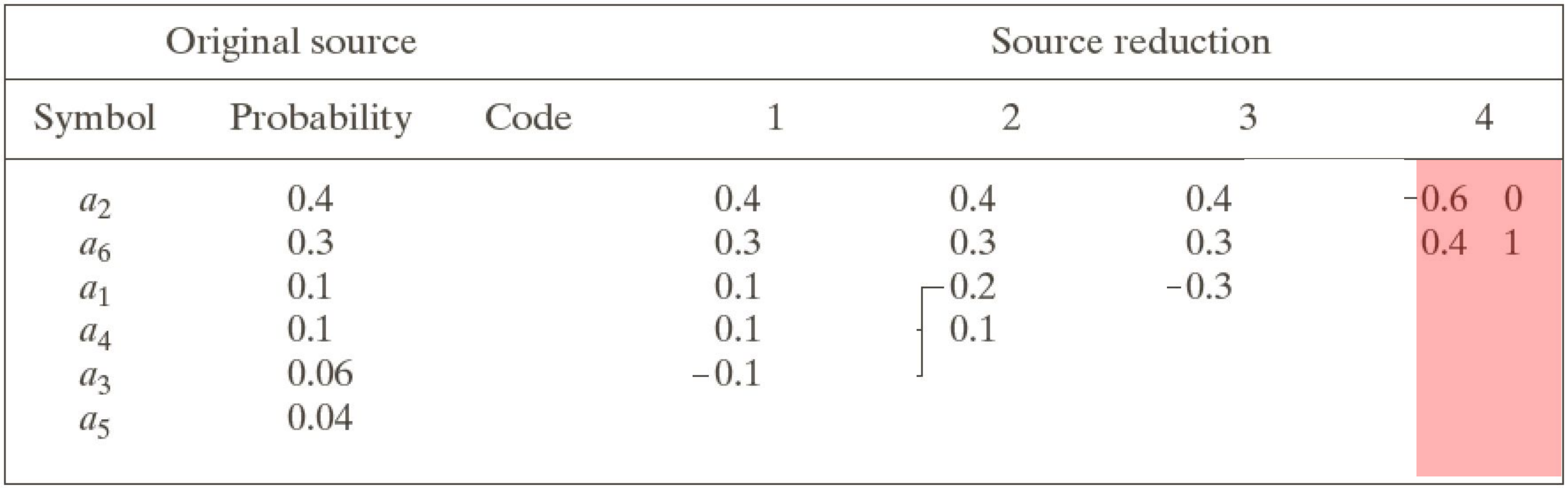
Consider a 3 bit image with the following probabilities:



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				

Huffman Coding: Example

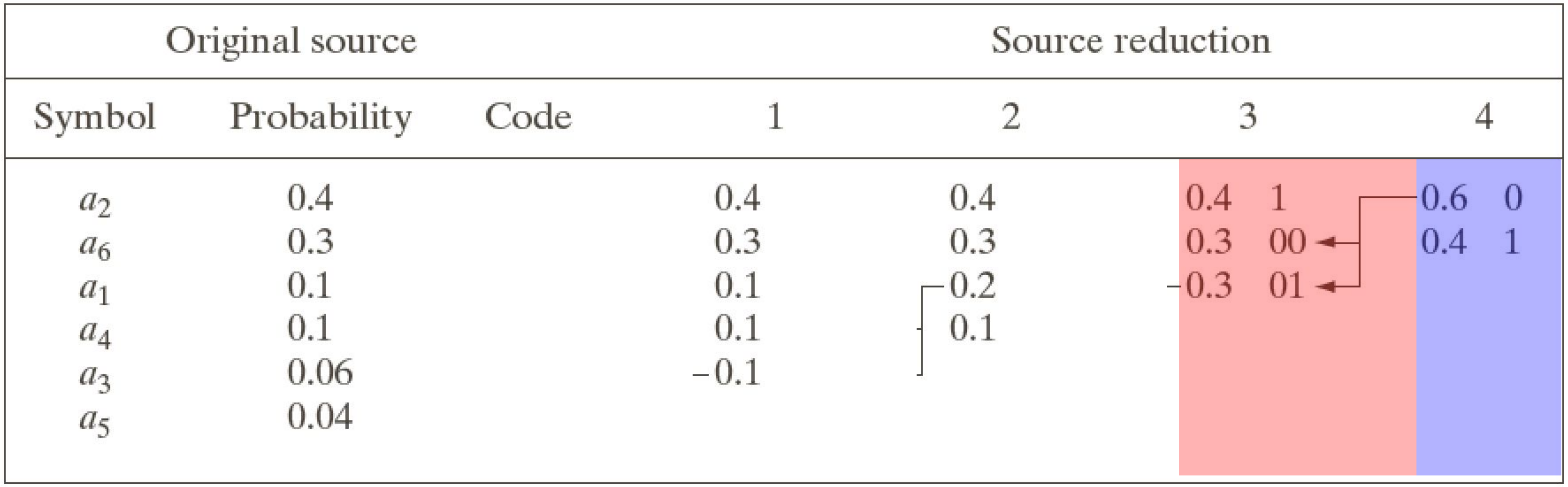
Huffman Code assignment
with 0 and 1:



Original source			Source reduction			
Symbol	Probability	Code	1	2	3	4
a_2	0.4		0.4	0.4	0.4	0.6 0
a_6	0.3		0.3	0.3	0.3	0.4 1
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					

Huffman Coding

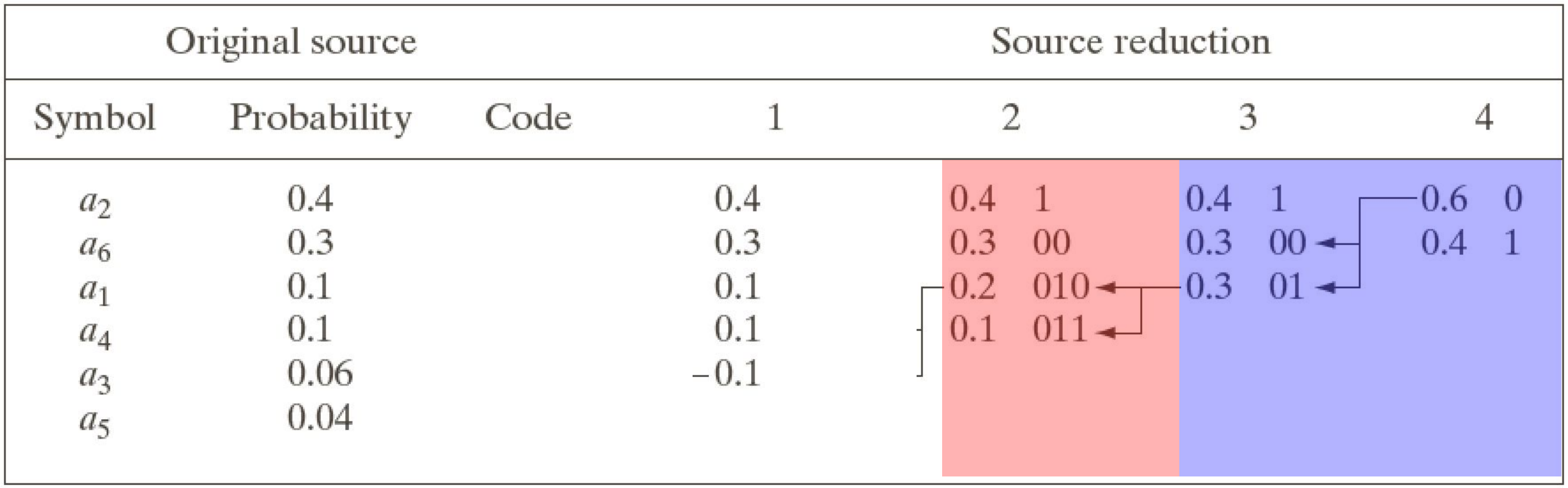
Huffman Code assignment:



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

Huffman Coding

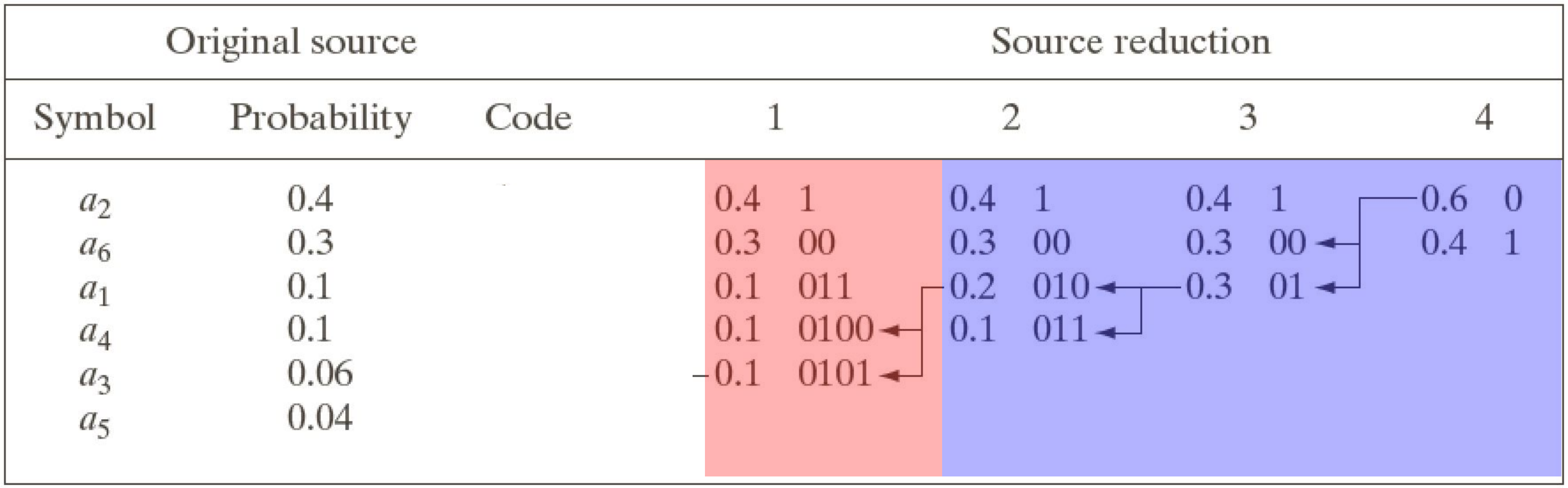
Huffman Code assignment:



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

Huffman Coding

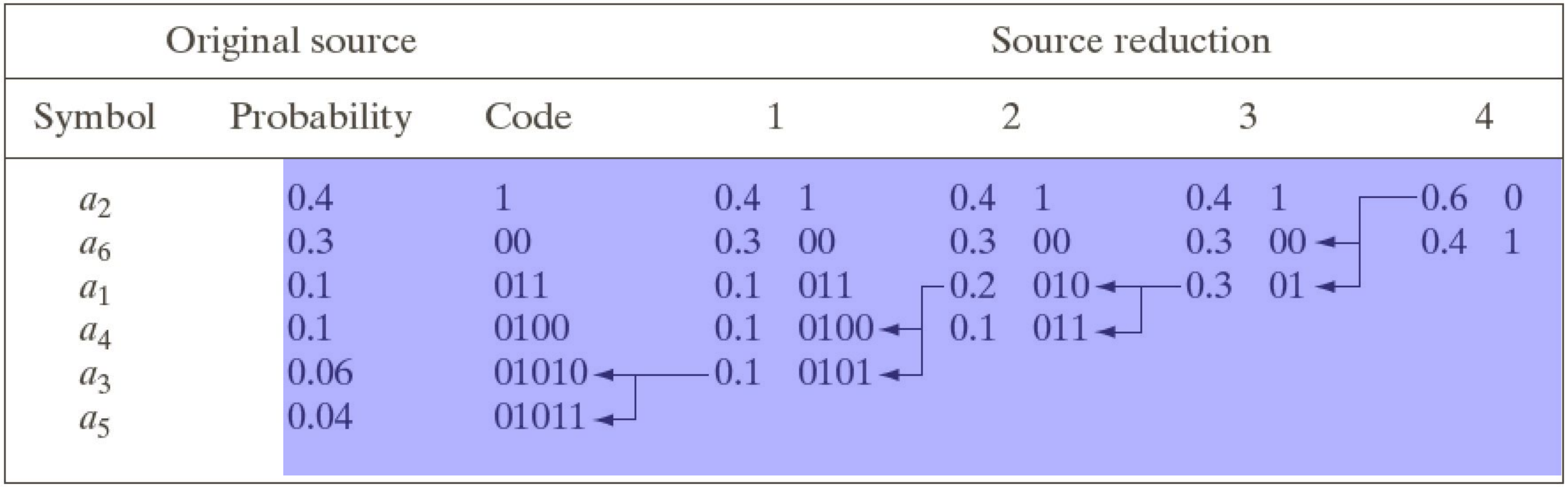
Huffman Code assignment:



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

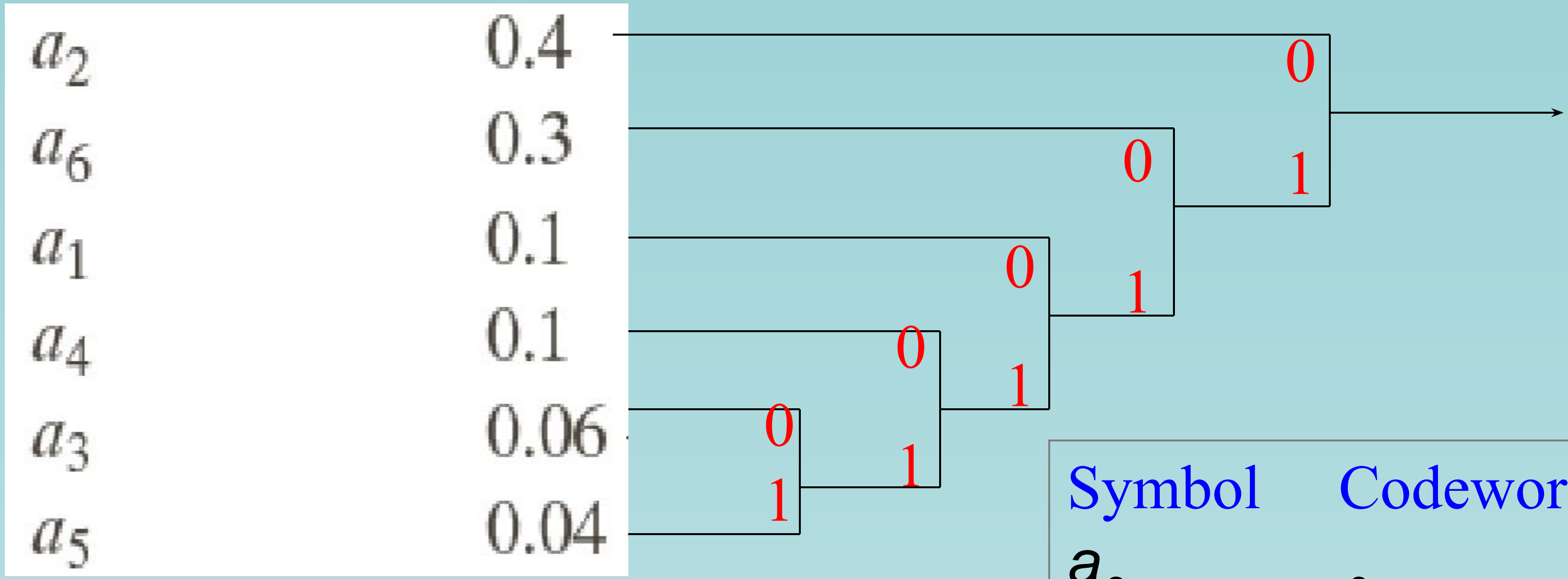
Huffman Coding

Huffman Code assignment:



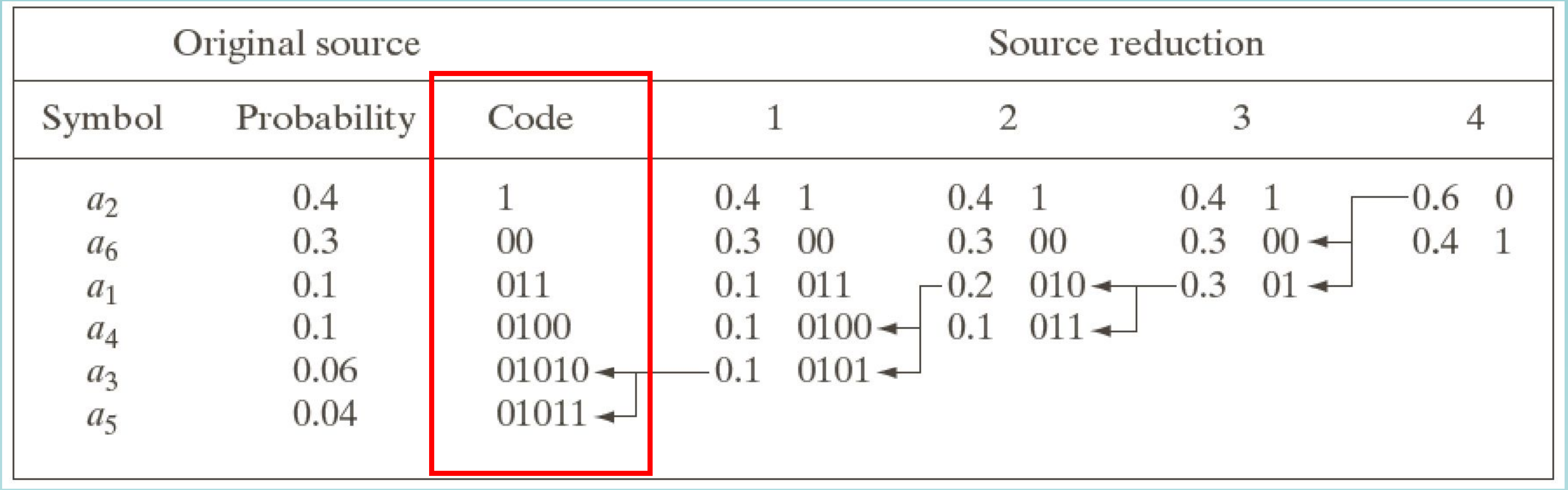
Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Huffman Coding



Symbol	Codeword
a_2	0
a_6	10
a_1	110
a_4	1110
a_3	11110
a_5	11111

Huffman Coding Results



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								

Avg. bit requirement/symbol:

$$L_{\text{avg}} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5$$
$$= 2.2 \text{ bits/symbol}$$

$H = -[(0.4)\log_2(0.4)+(0.3)\log_2(0.3)+(0.1)\log_2(0.1)$
$+(0.1)\log_2(0.1)+(0.06)\log_2(0.06)+(0.04)\log_2(0.04)$
$= 0.53+0.52+0.33+0.33+0.24+0.19$
= 2.14 bits/pixel

$\boxed{\text{Log}_2(x) = \log_{10}(x) \times 3.322}$

$C = 3/2.2 = 1.364$

$R_D = (1 - 1/1.364) = 0.267$

Huffman Coding Example 2

Original Gray Level (Natural Code)	Probability	Huffman code
g_0: 00_2	0.2	010
g_1: 01_2	0.3	00
g_2: 10_2	0.1	011
g_3: 11_2	0.4	1

- More frequently occurring $\rightarrow$ fewer bits for the code

Huffman Coding Example – Entropy vs. Ave bpp

Example 2:

$$Entropy = -\sum_{i=0}^{3} p_i \log_2(p_i)$$

$$= -[(0.2)\log_2(0.2) + (0.3)\log_2(0.3) + (0.1)\log_2(0.1) + (0.4)\log_2(0.4)$$

$$\approx 1.846 \text{ bits/pixel}$$

(Note : $\log_2(x)$ can be found by taking $\log_{10}(x)$ and multiplying by 3.322)

$$L_{ave} = \sum_{i=0}^{L-1} l_i \, p_i$$

$$= 3(0.2) + 2(0.3) + 3(0.1) + 1(0.4)$$

$$= 1.9 \text{ bits/pixel} \quad \text{(Average length with Huffman code)}$$

Huffman Coding Example – Results

- In the example, we observe a 2.0 : 1.9 compression, which is about a 1.05 compression ratio, providing about 5% redundant data compression.

- From the examples, we can see that the Huffman code is highly dependent on the histogram, so any preprocessing to simplify the histogram will help improve the compression ratio

Huffman Coding

Properties :

- *codes one symbol at a time*
- *Block code*
  - each symbol is mapped to a block of code
- *Instantaneously decodable*
  - does not need to foresee the future codes while decoding
- *Uniquely decodable*
  - any string of code symbol can be decoded in only one way

Golomb Coding

The Golomb code of **n** with respect to **m**, denoted $\mathbf{G_m(n)}$, is a combination of the unary code of quotient, **floor**[n/m] and the binary representation of reminder (n **mod** m).

- *Can only be used to represent **nonnegative** **integers** inputs*
- *with **exponentially decaying probability distributions***
- *can be **optimally** encoded*
- *using a family of codes.*

- *Computationally **simpler** than **Huffman codes**.*

Some Basic Compression Methods: Golomb Coding

Given a nonnegative integer n and a positive integer divisor $m > 0$, the Golomb code of n with respect to m, denoted $G_m(n)$, constructed as follows:

Step 1. Form the unary code of quotient $\lfloor n/m \rfloor$

(The unary code of integer q is defined as q 1s followed by a 0)

Step2. Let k=$\lceil \log_2 m \rceil$, $c = 2^k - m$, $r = n \bmod m$, and compute truncated remainder r' such that

$$r' = \begin{cases} r \text{ truncated to } k\text{-}1 \text{ bits} & 0 \le r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases}$$

Step 3. Concatenate the results of steps 1 and 2.

Some Basic Compression Methods: Golomb Coding

Step 1. Form the unary code of quotient $\lfloor n/m \rfloor$

(The unary code of integer q is defined as

q 1s followed by a 0)

$G_4(9)$:

Step2. Let k=$\lceil \log_2 m \rceil, c = 2^k - m, r = n \bmod m,$

and compute truncated remainder r' such that

$$r' = \begin{cases} r \text{ truncated to } k\text{-}1 \text{ bits} & 0 \le r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases}$$

Step 3. Concatenate the results of steps 1 and 2.

Some Basic Compression Methods: Golomb Coding

Step 1. Form the unary code of quotient $\lfloor n/m \rfloor$

(The unary code of integer q is defined as

q 1s followed by a 0)

Step2. Let k=$\lceil \log_2 m \rceil, c = 2^k - m, r = n \bmod m,$

and compute truncated remainder r' such that

$G_4(7)$?

$$r' = \begin{cases} r \text{ truncated to } k\text{-1 bits} & 0 \le r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases}$$

Step 3. Concatenate the results of steps 1 and 2.

Golomb Coding

<u>Fill the chart at home</u>

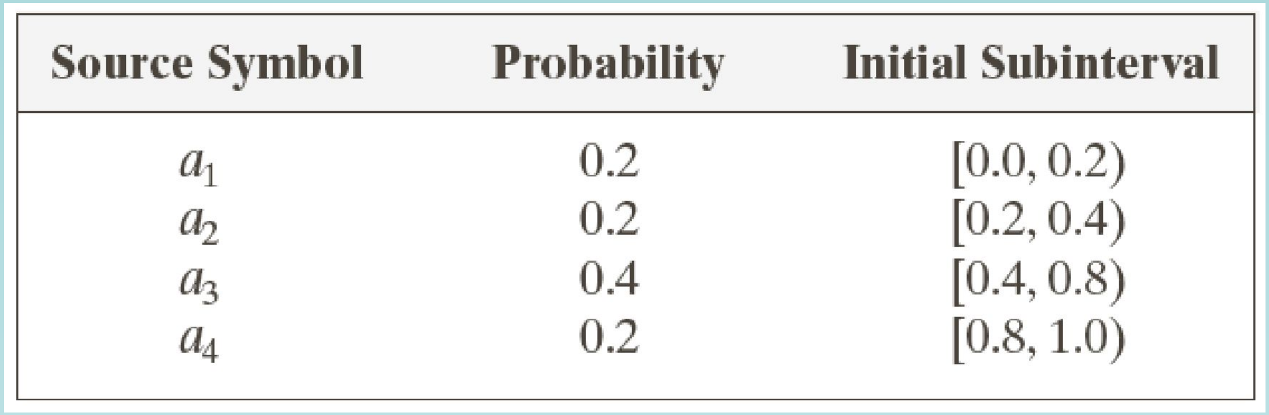
n	$G_1(n)$	$G_2(n)$	$G_4(n)$
0			
1			
2			
3			
4			
5			
6			
7	11111110	11101	1011

Loss Less Compression: **Arithmetic Coding:**

- ***Variable length code***
- ***Error-free compression technique***
- ***Non block coding***
  - one to one correspondence between symbol and code **does not exist**
  - An entire sequence of source symbols (string of symbol) is mapped to a **single arithmetic number (code word)**
  - The code word itself defines an interval of real numbers between 0 and 1.
- **This coding can achieve theoretically higher compression rates than Huffman codes**

Arithmetic Coding

- *Maintains **an interval** between* [0 1] based on the probabilities of the symbol



Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

<u>**A four symbol source**</u>

Arithmetic Coding

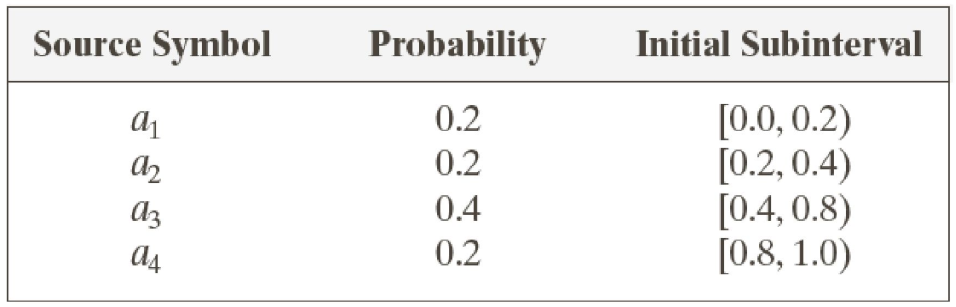
- Let we have a **symbol sequence** or message of a four symbol source

$$a_1 a_2 a_3 a_3 a_4$$

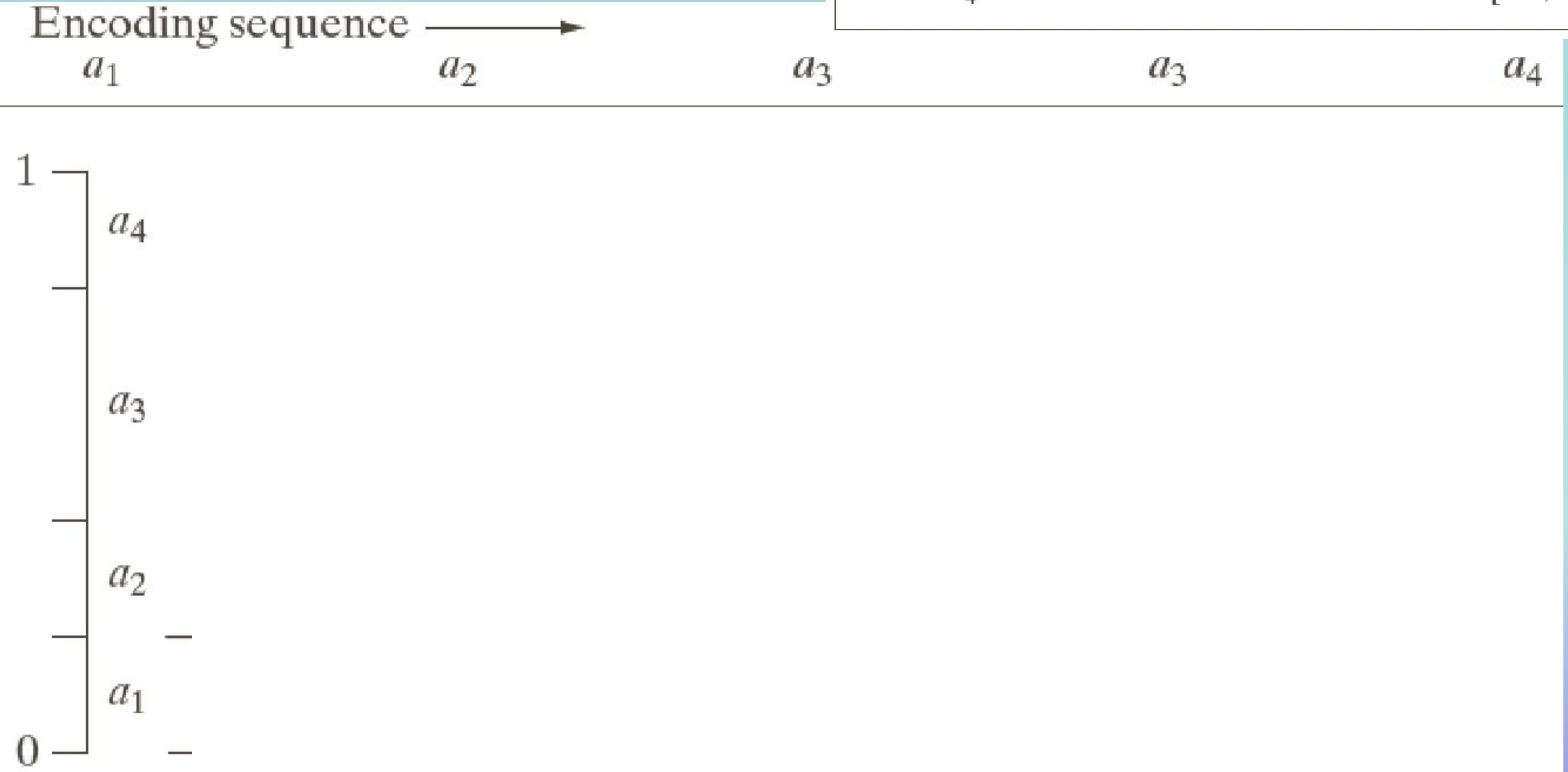
- need to **code the sequence** with Arithmetic coding.

Arithmetic Coding

At the start, interval **[0,1)** is subdivided initially into four regions based on the probabilities

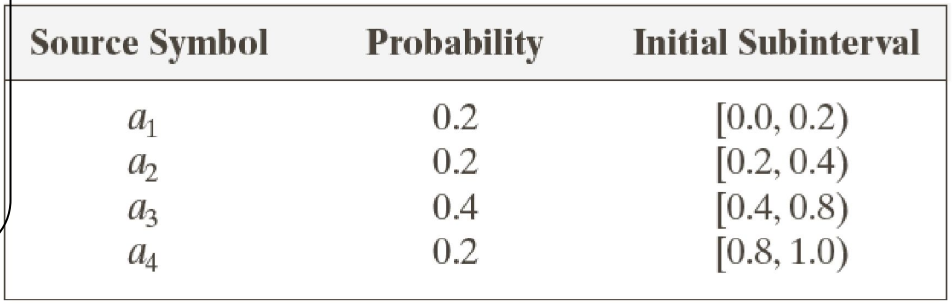


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

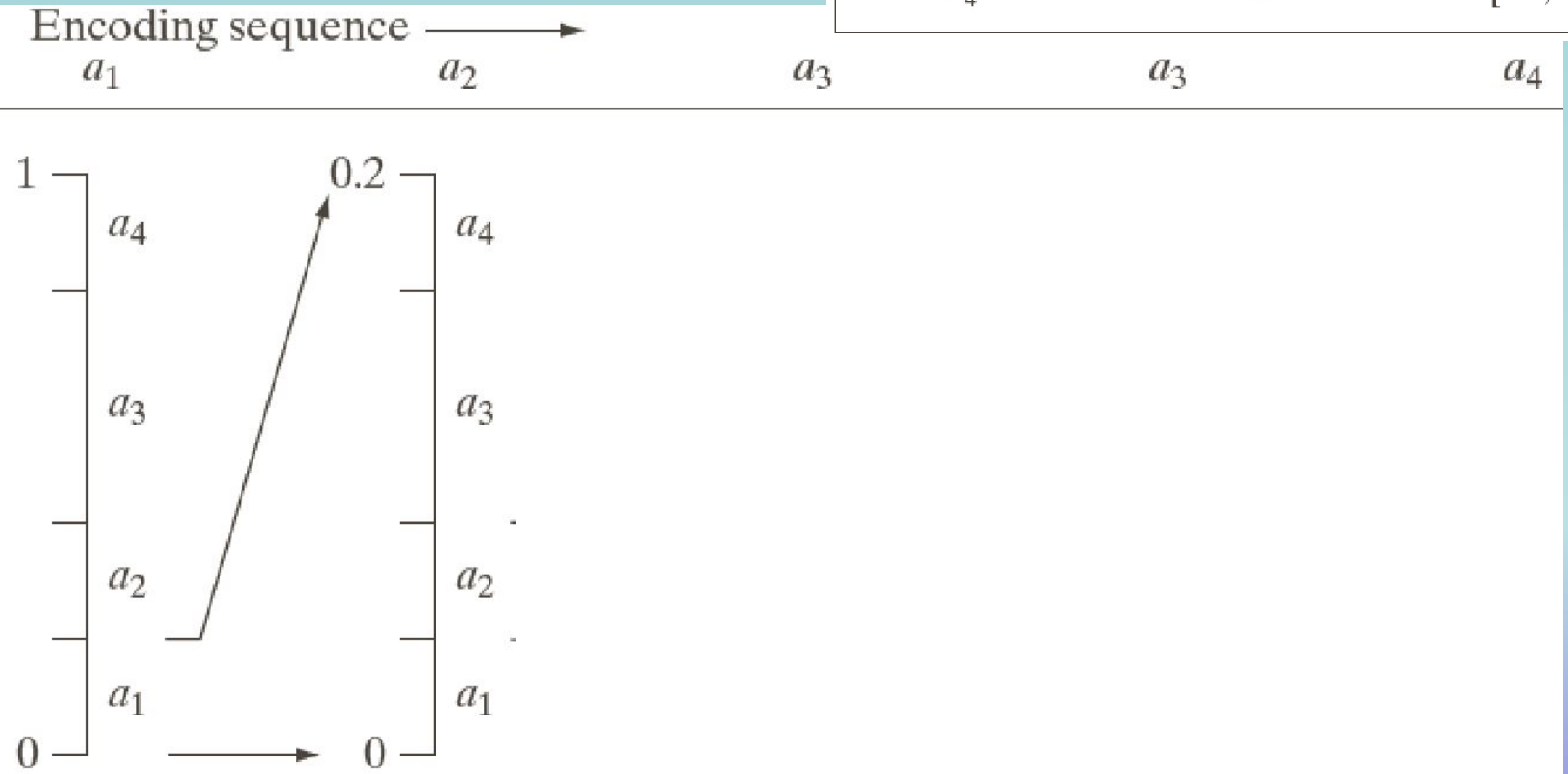


Arithmetic Coding $a_1 a_2 a_3 a_3 a_4$

Symbol ***a***$_1$, associated with subinterval **[0,0.2)** is the first message being coded and expanded the full height in narrow range

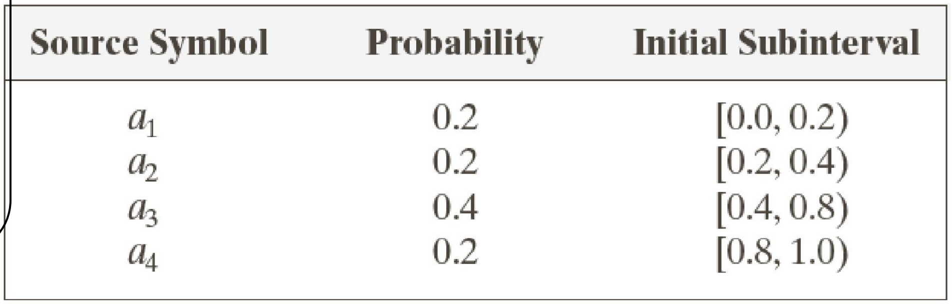


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

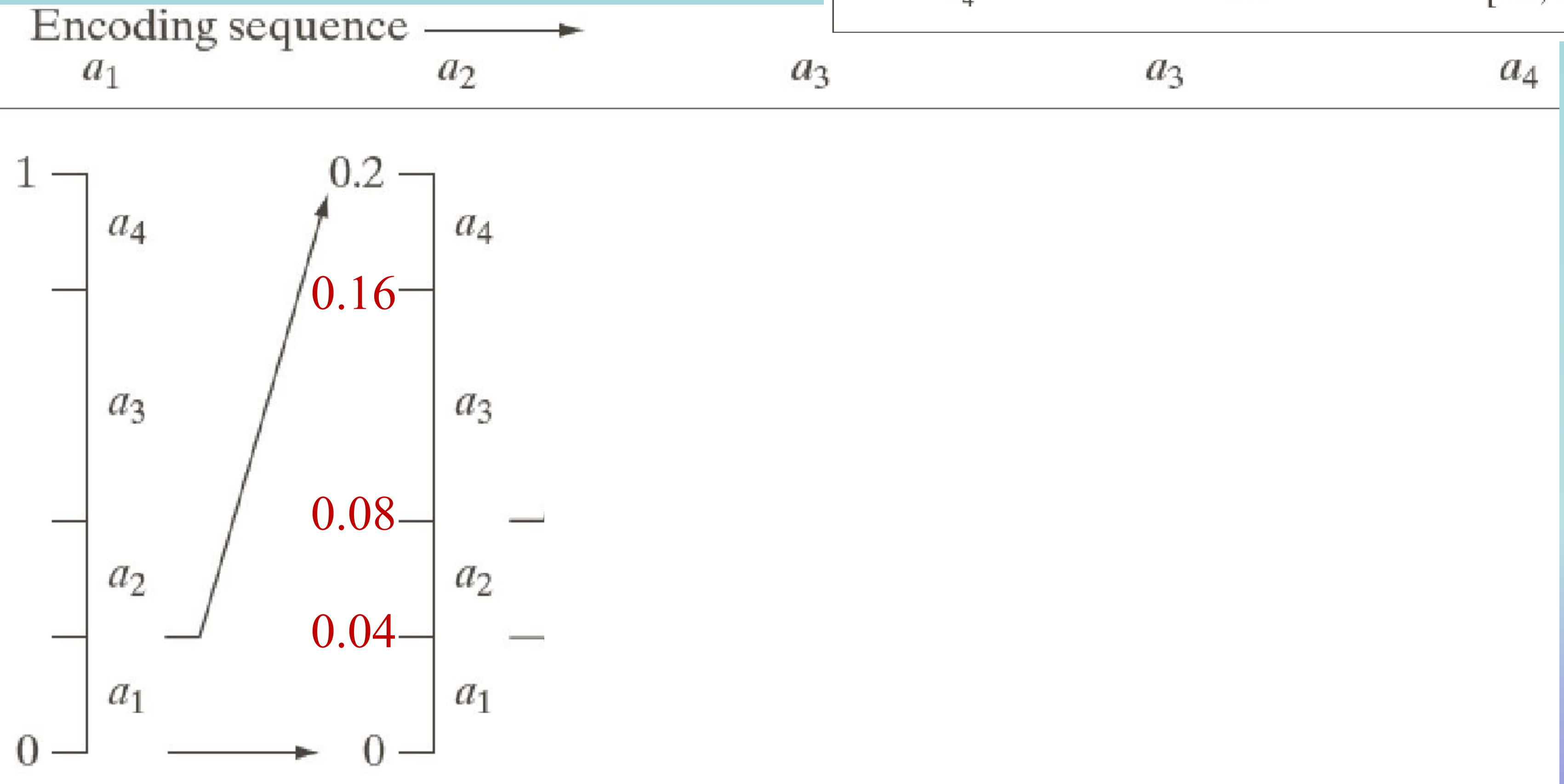


Arithmetic Coding

Symbol a_1, associated with subinterval **[0,0.2)** is the first message being coded and expanded the full height in narrow range



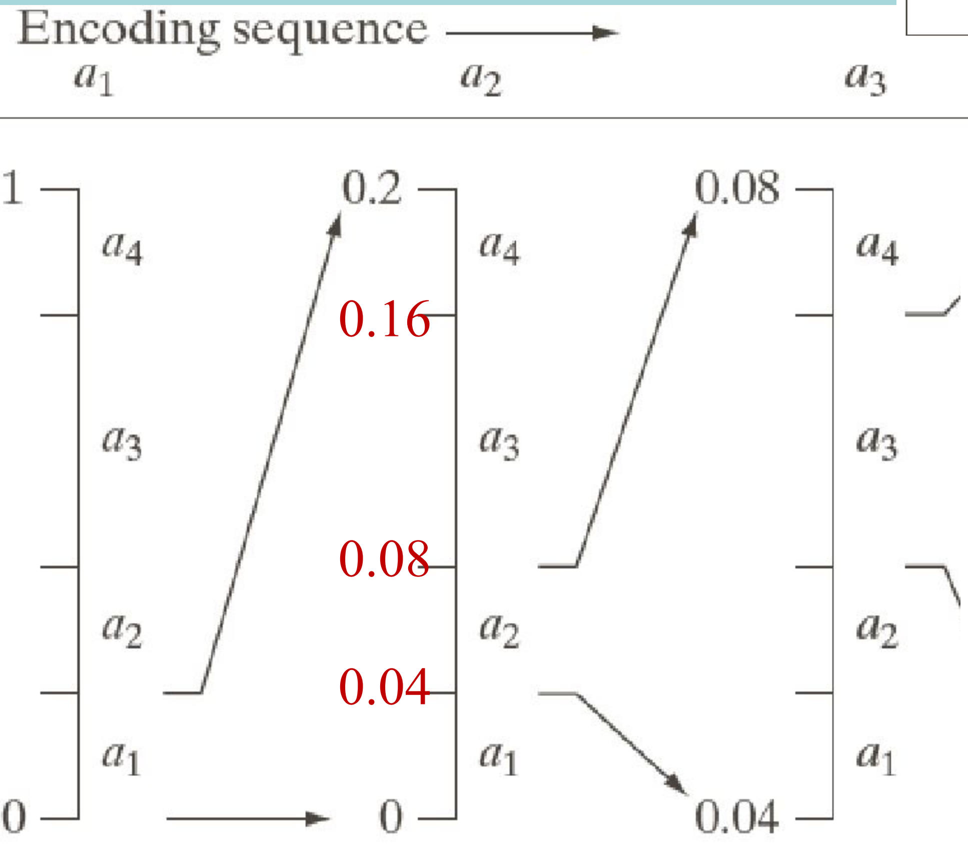
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



Arithmetic Coding

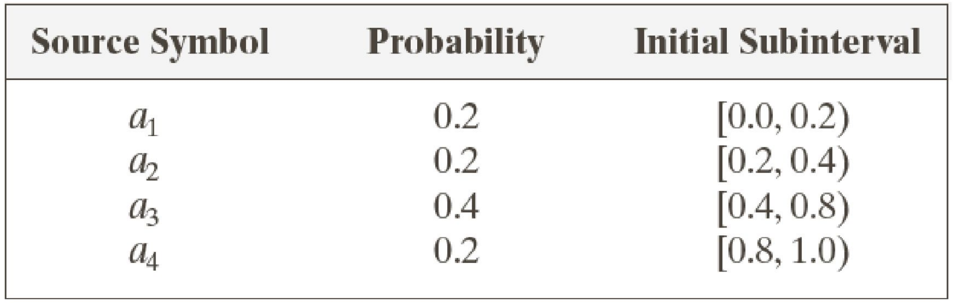
Symbol ***a*₂**, associated with subinterval **[0.04, 0.08)** is expanded the full height in narrow range again

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

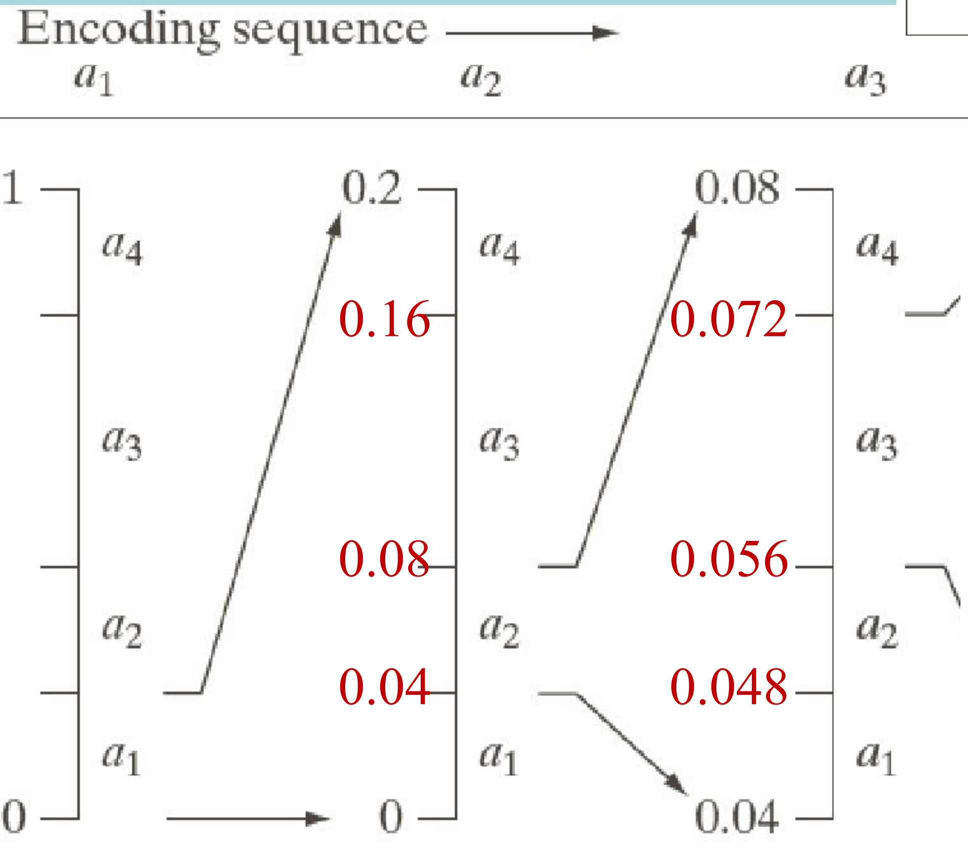


Arithmetic Coding

Symbol a_2, associated with subinterval **[0.04, 0.08)** is expanded the full height in narrow range again

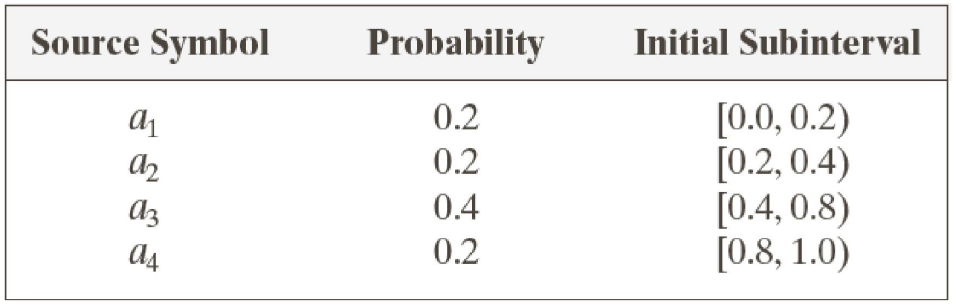


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

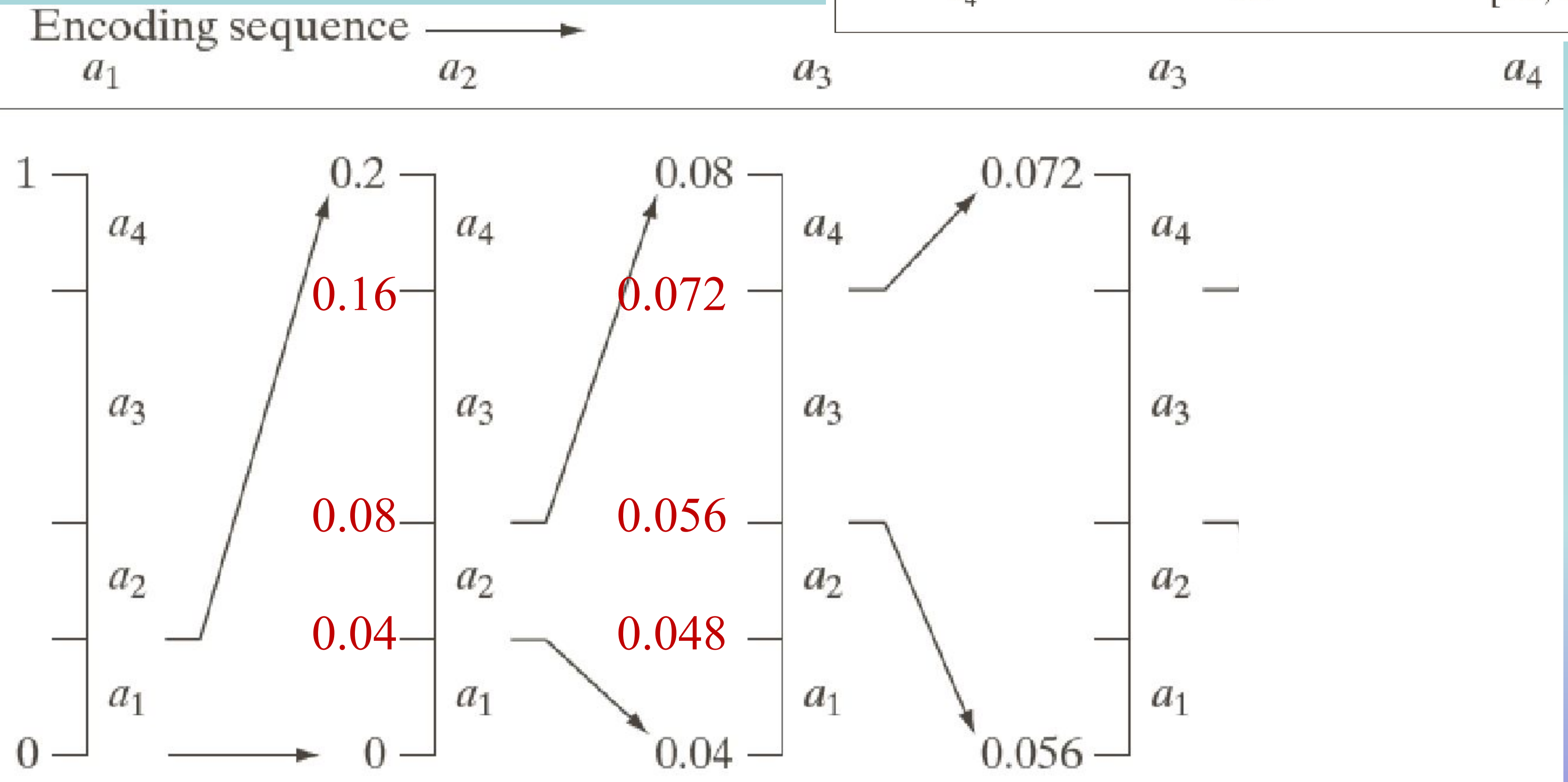


Arithmetic Coding

Symbol ***a*₃**, associated with subinterval **[0.056, 0.072)** is expanded the full height in narrow range again

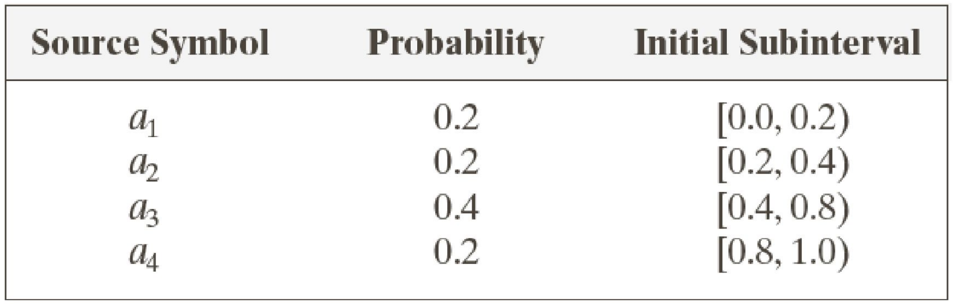


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

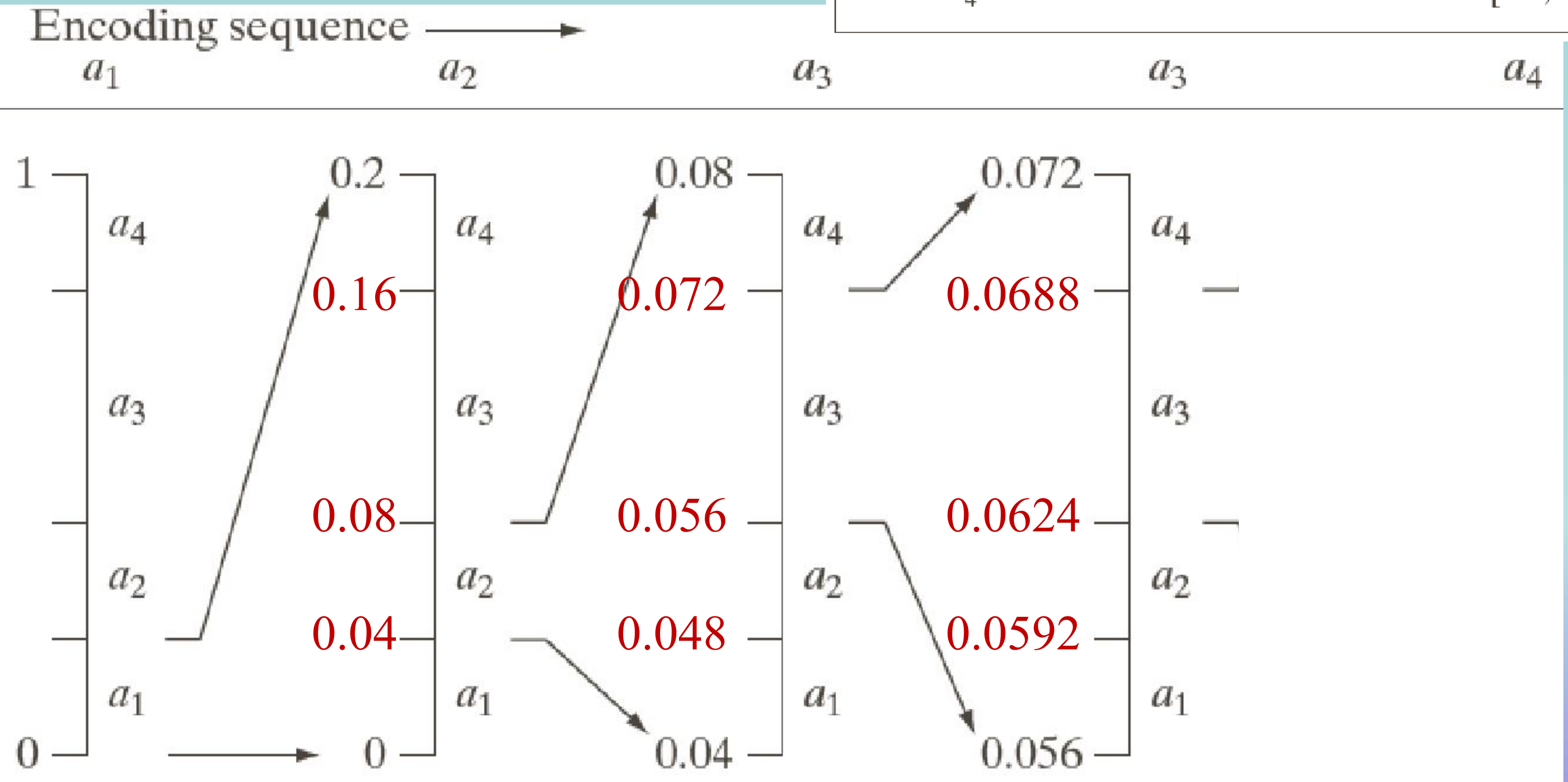


Arithmetic Coding

Symbol a_3, associated with subinterval **[0.056, 0.072)** is expanded the full height in narrow range again

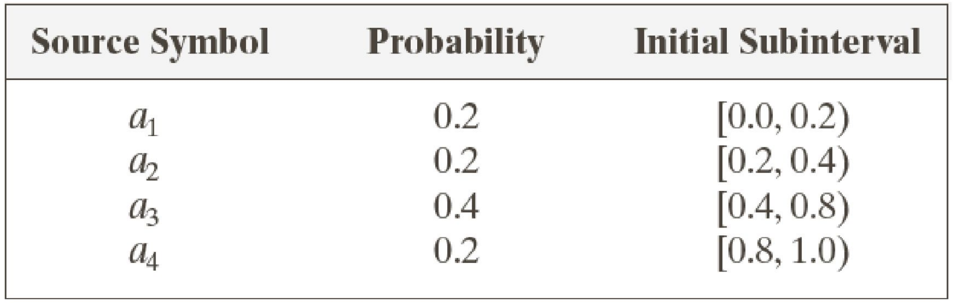


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

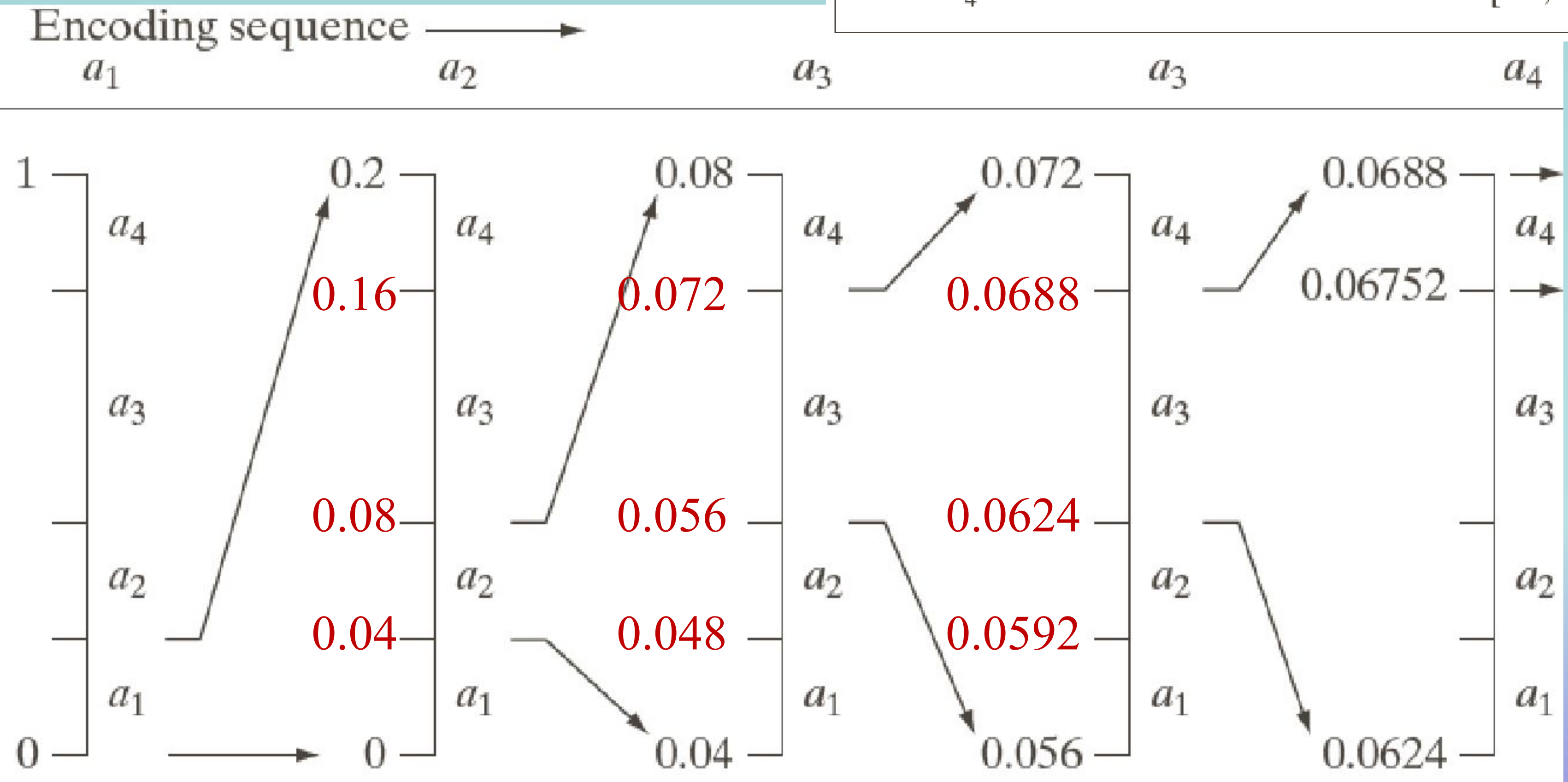


Arithmetic Coding

Now symbol a_3, associated with subinterval **[0.064, 0.0688)** is expanded the full height in narrow range again

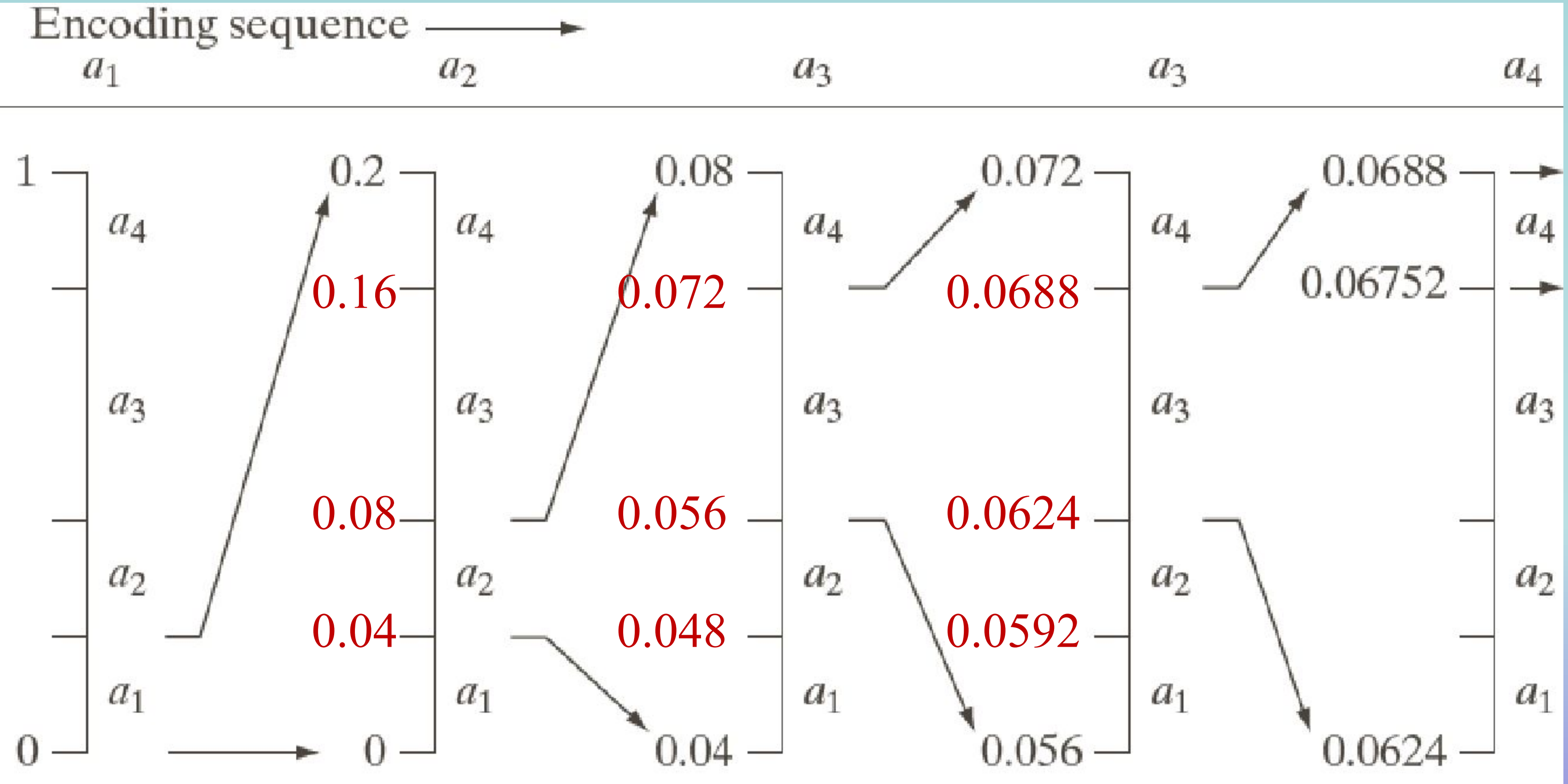


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



Arithmetic Coding

Now symbol ***a***$_4$, associated with subinterval **[0.06752, 0.0688)** is expanded the full height in narrow **final** range. Any number within this range can be use as message. **Example: 0.068**



Arithmetic Coding

The arithmetic code for the sequence

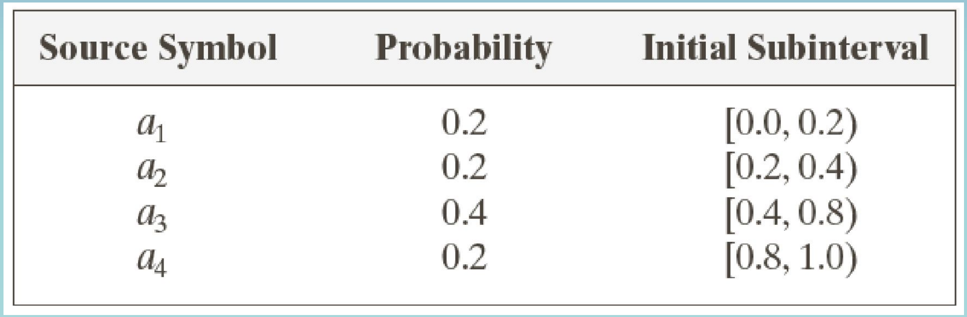
$$a_1 a_2 a_3 a_3 a_4$$

is **0.068**

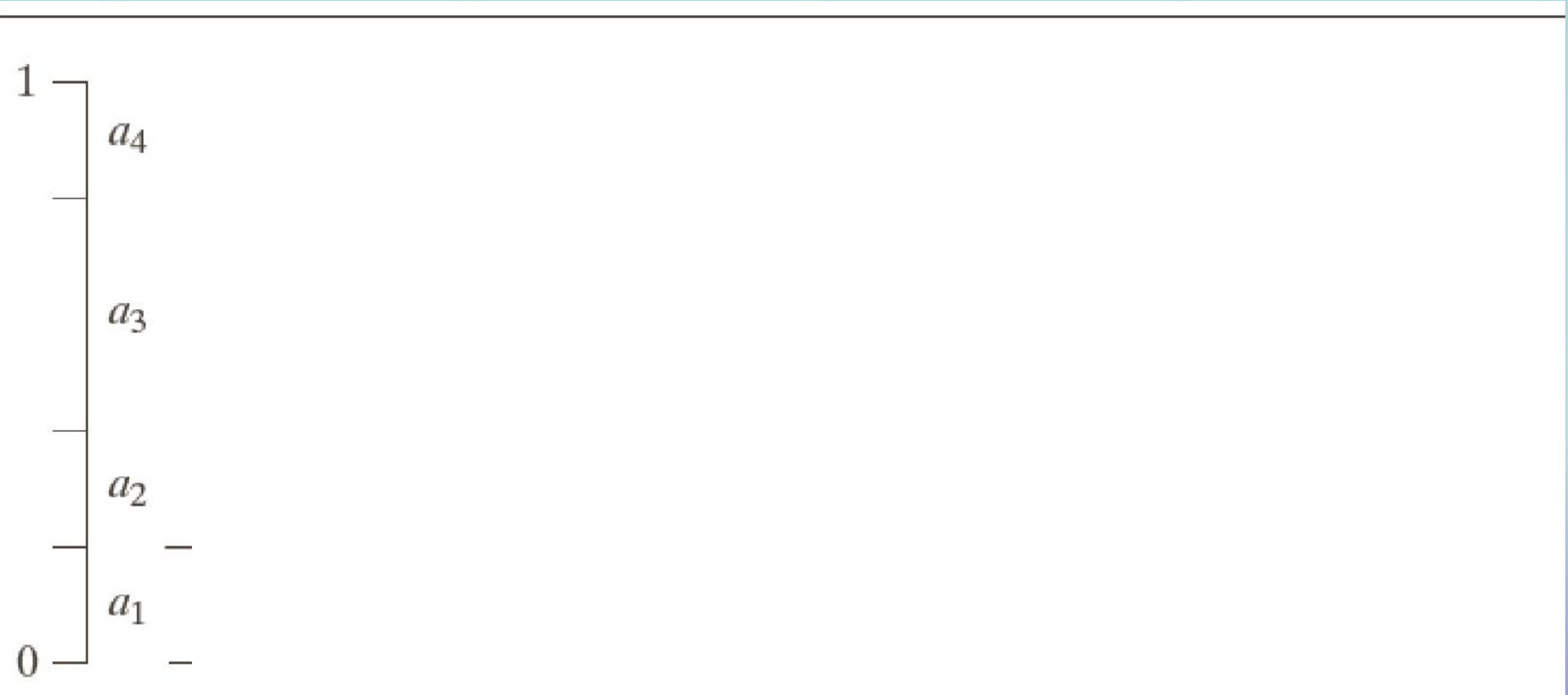
How to Decode it?

Decoding

0.068



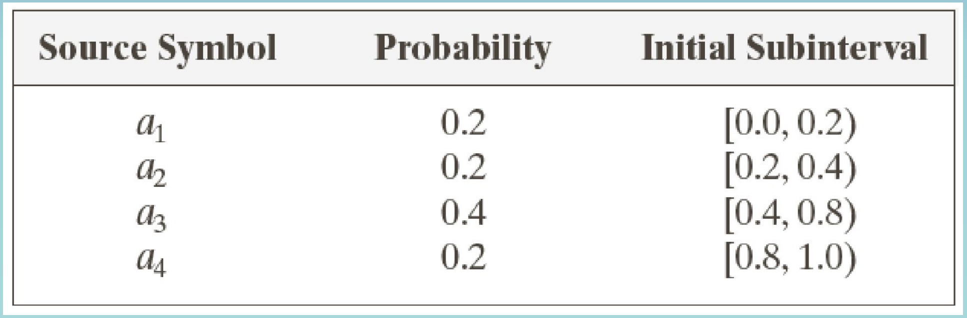
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



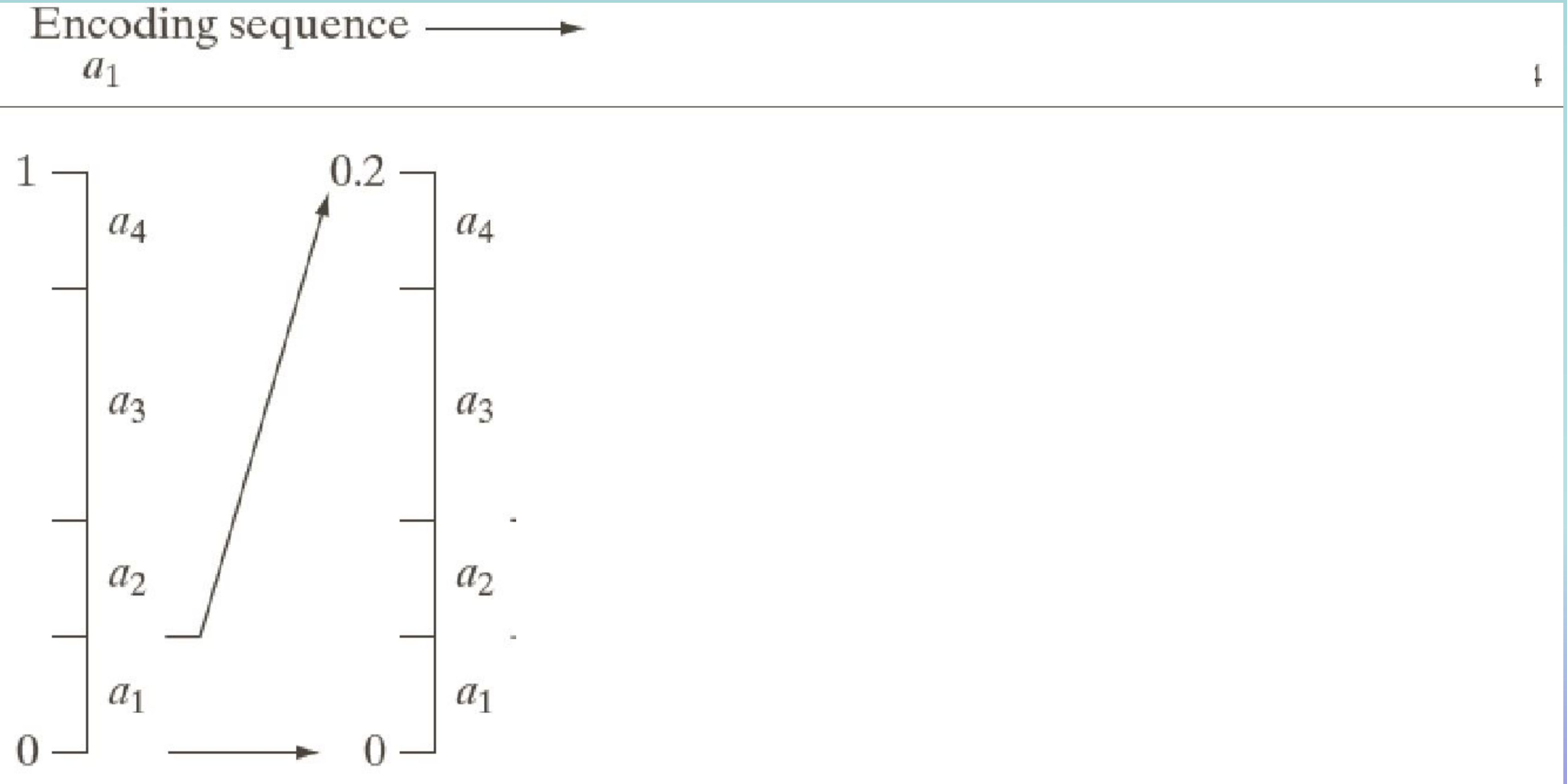
Decoding

$a_1 a_2 a_3 a_3 a_4$

0.068



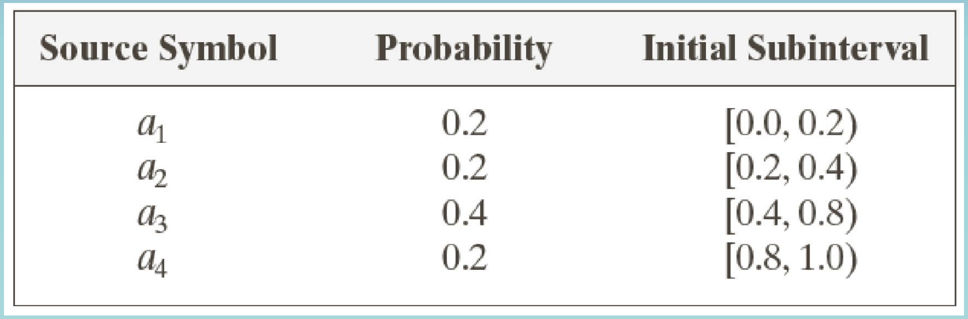
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



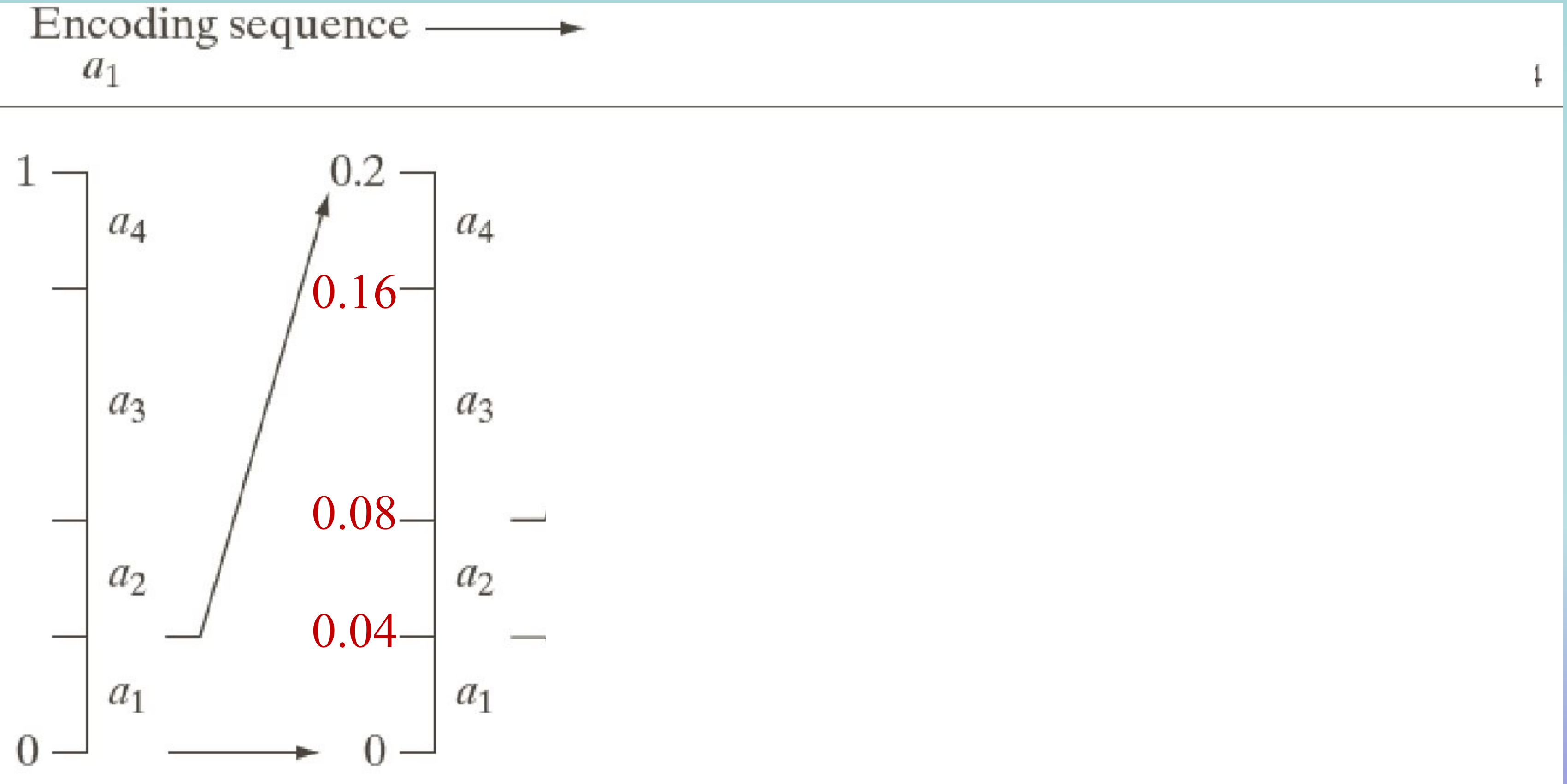
Decoding

$a_1 a_2 a_3 a_3 a_4$

0.068



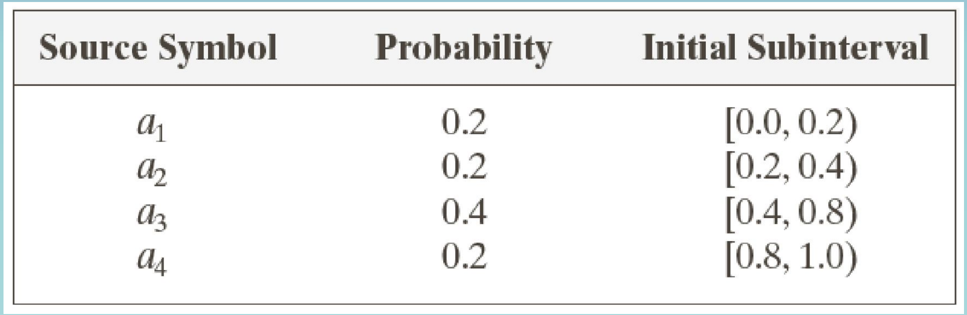
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



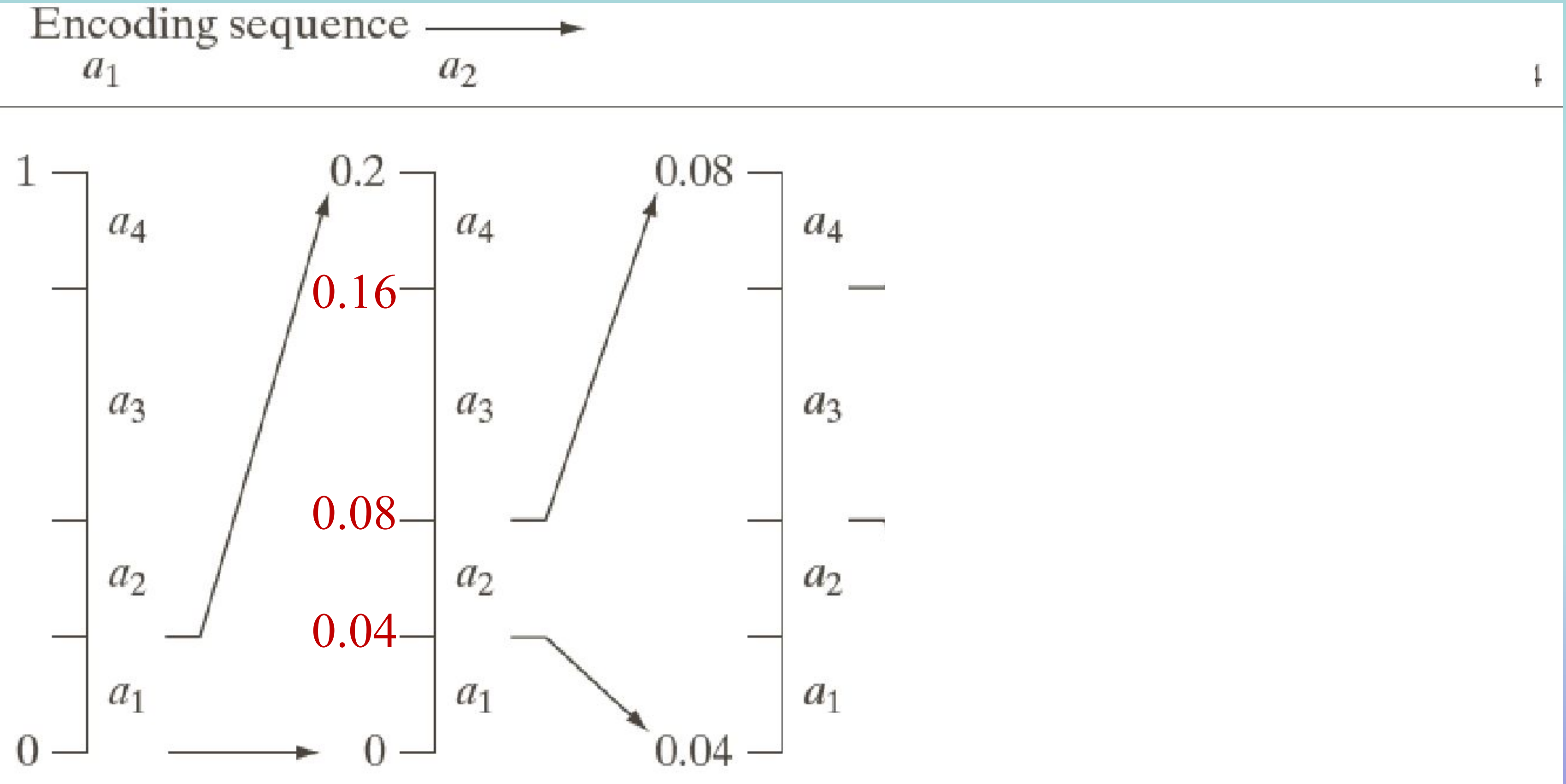
Decoding

$a_1 a_2 a_3 a_3 a_4$

0.068



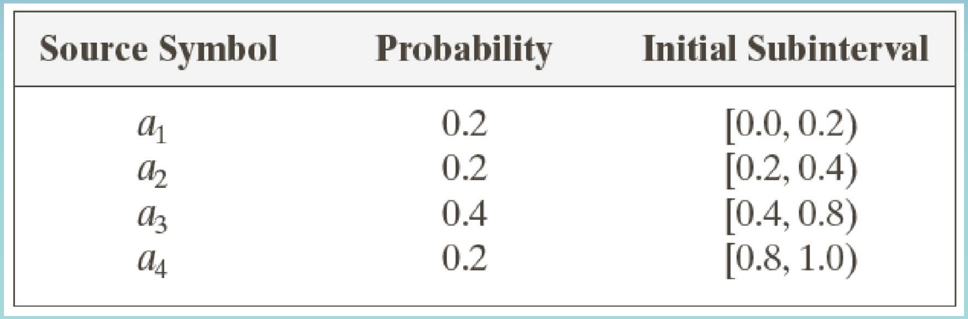
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



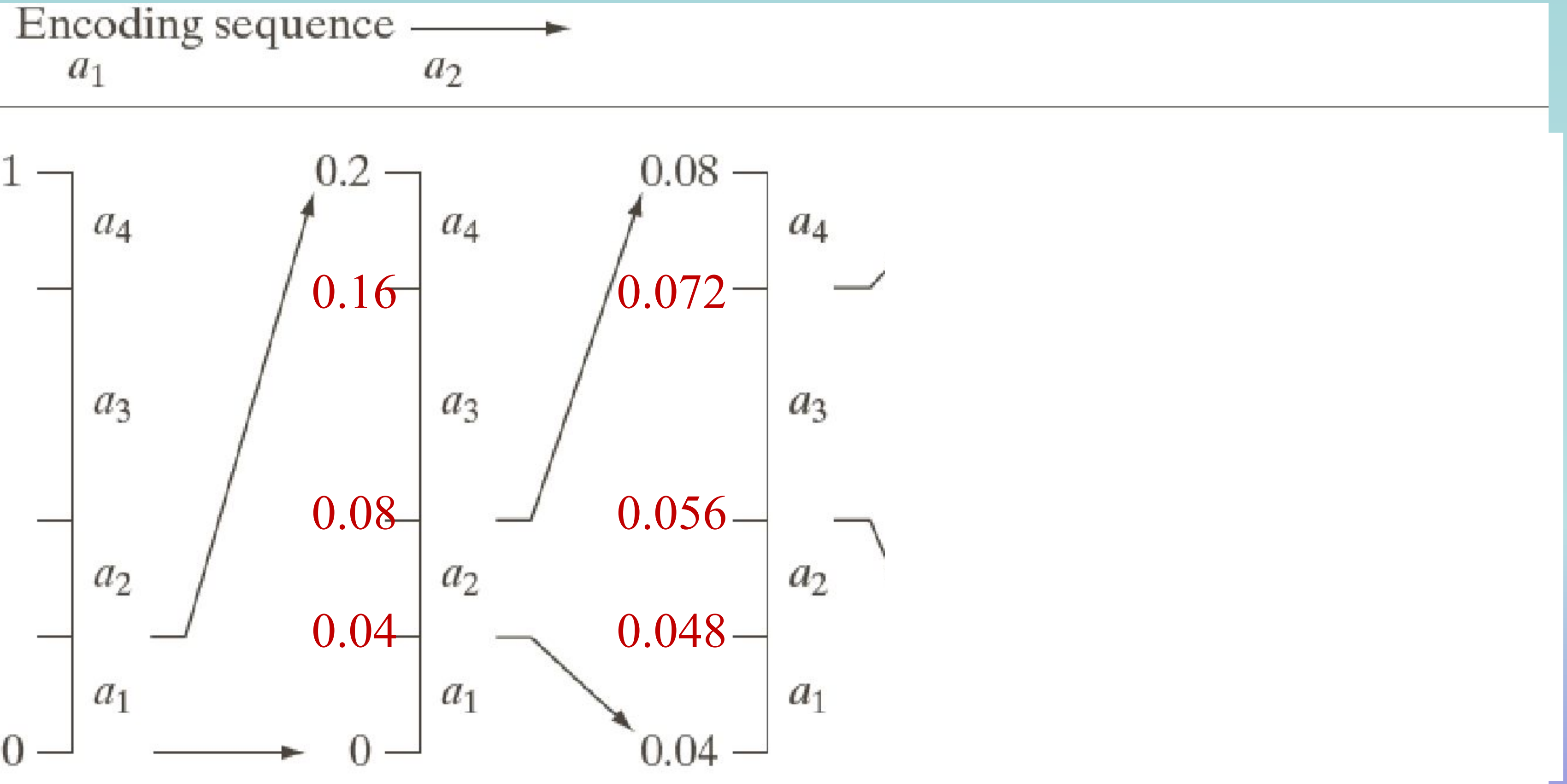
Decoding

$a_1a_2a_3a_3a_4$

0.068



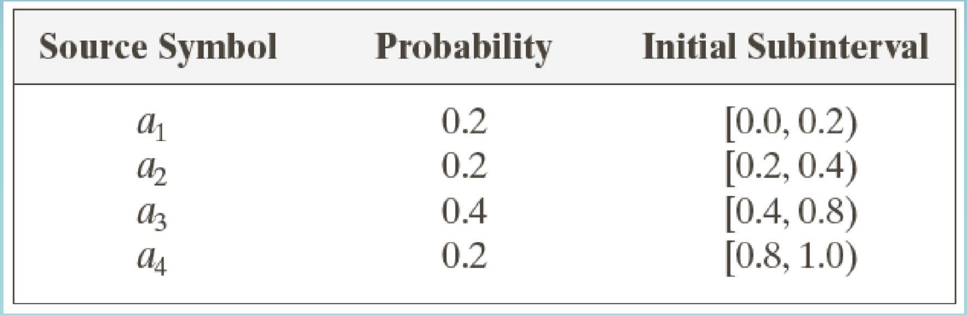
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



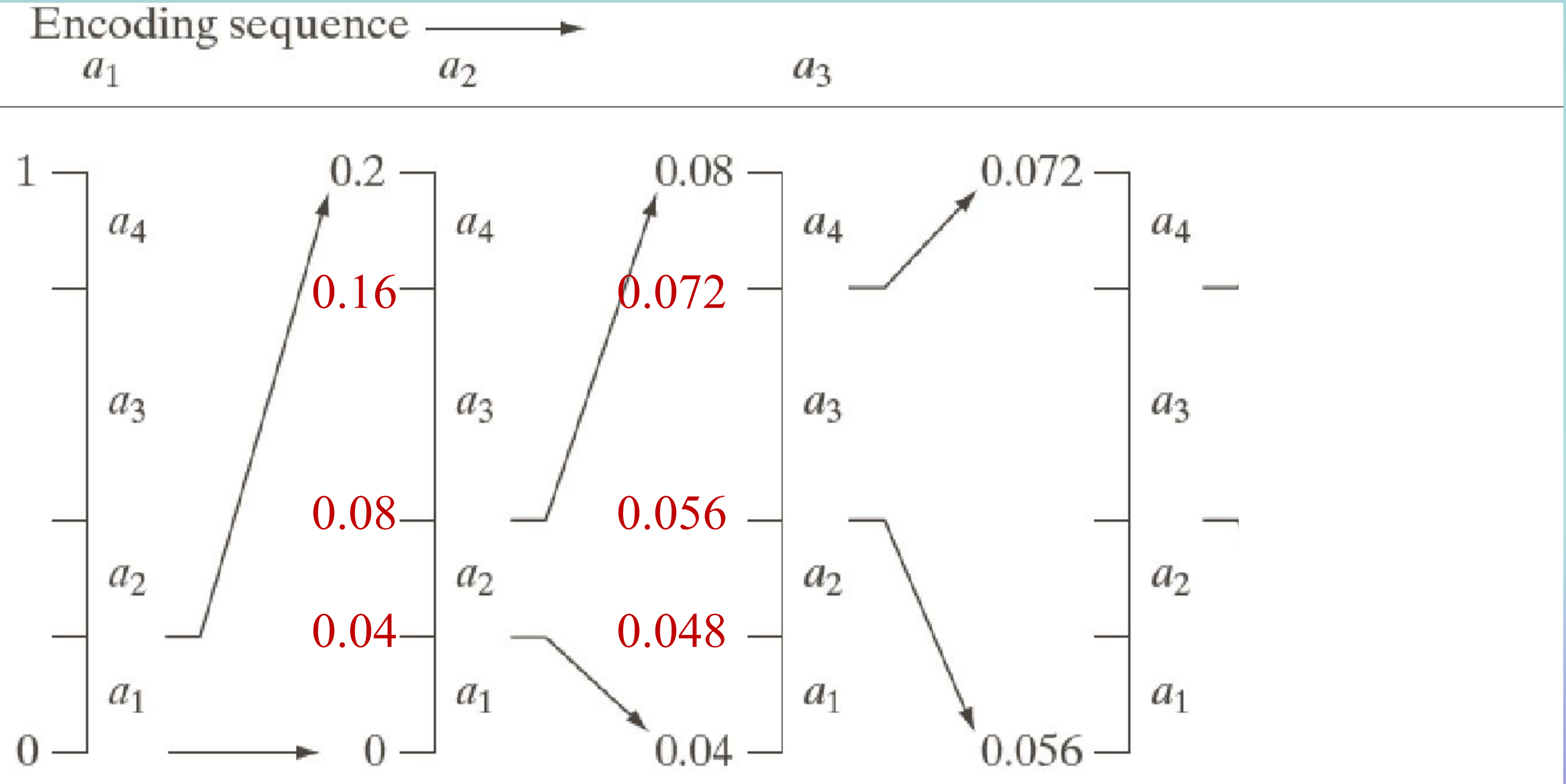
Decoding

$a_1 a_2 a_3 a_3 a_4$

0.068



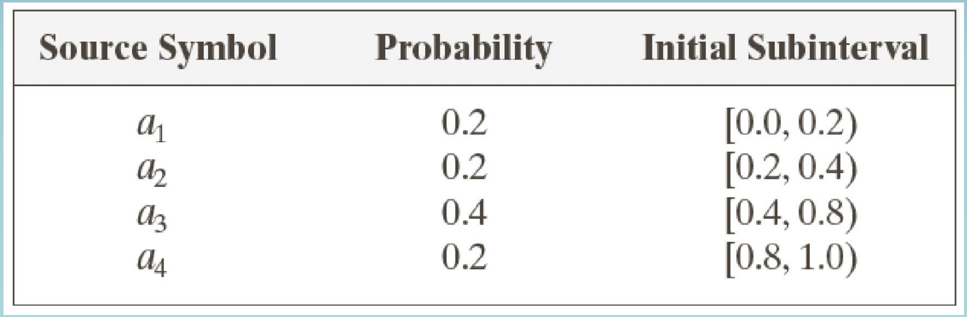
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



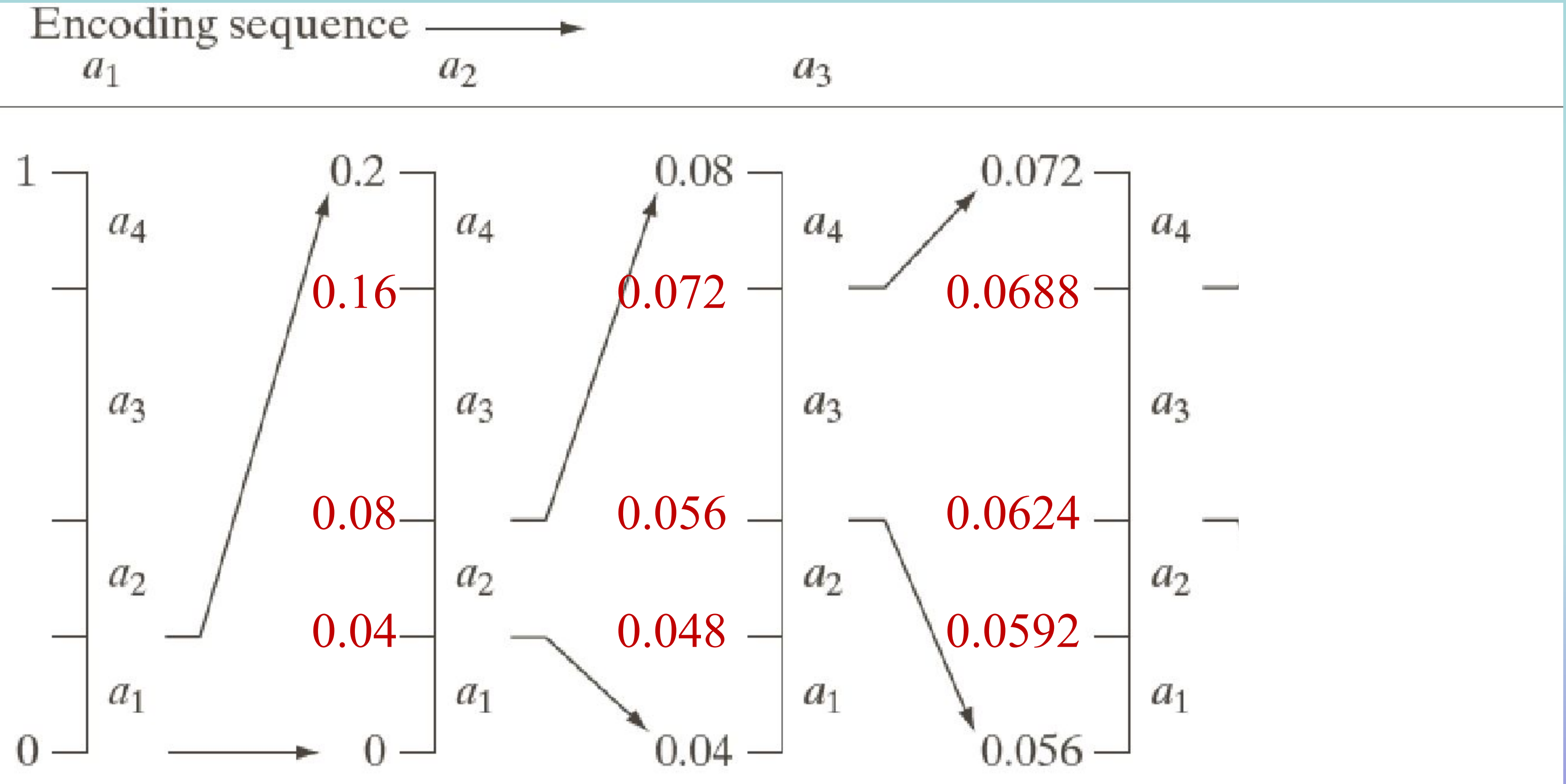
Decoding

$a_1a_2a_3a_3a_4$

0.068



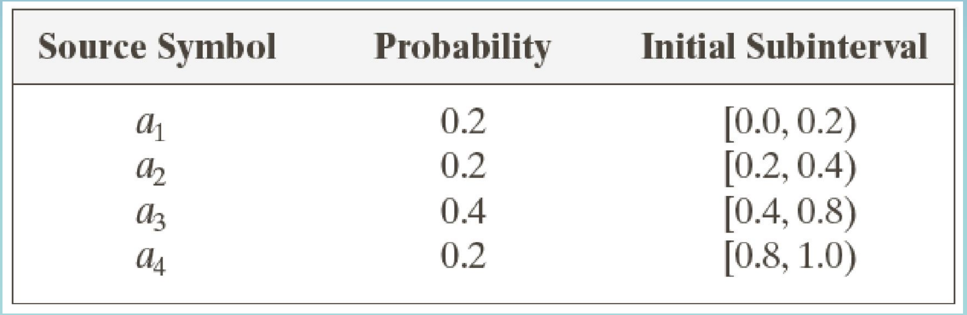
Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



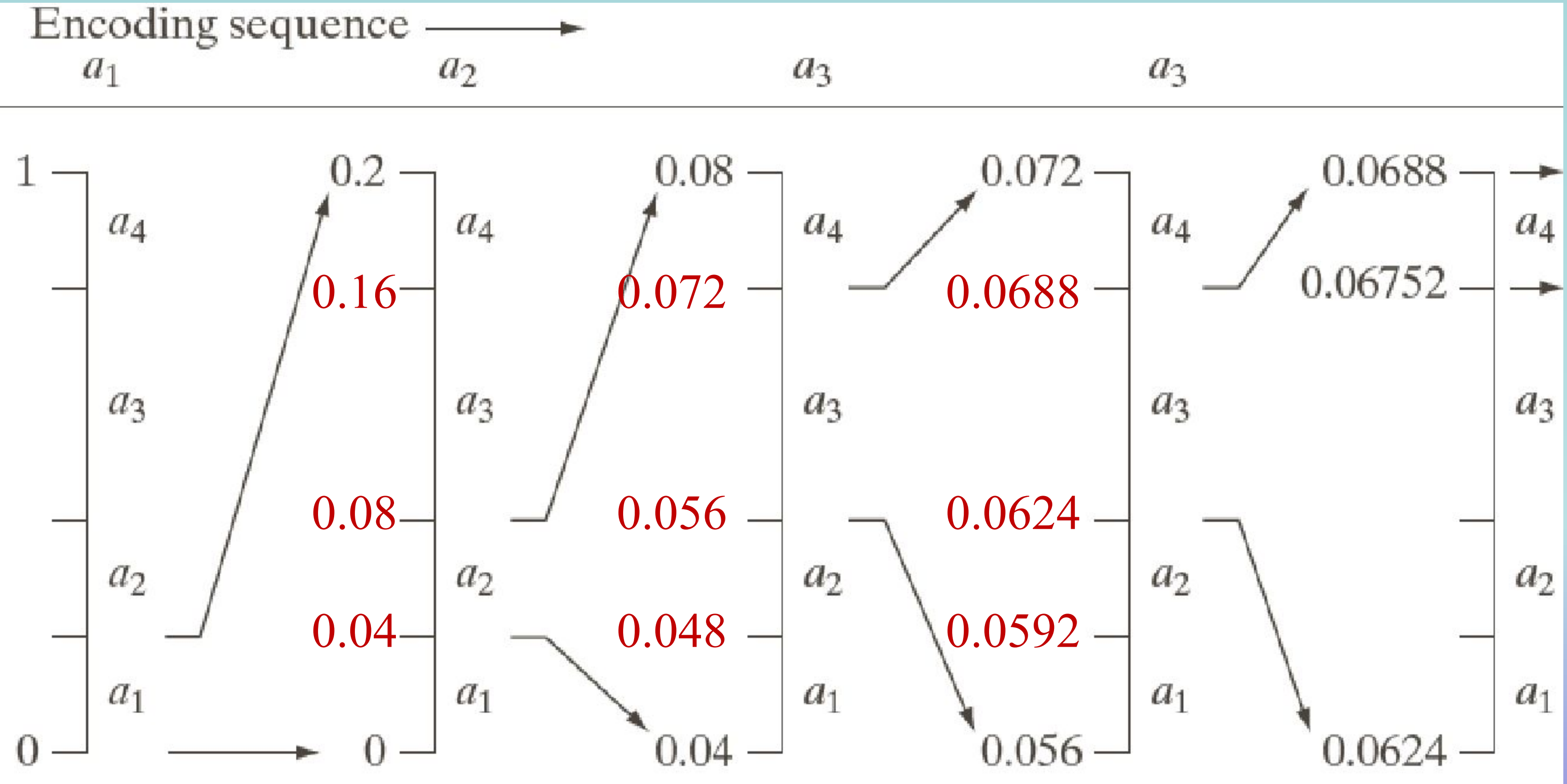
Decoding

$a_1 a_2 a_3 a_3 a_4$

0.068

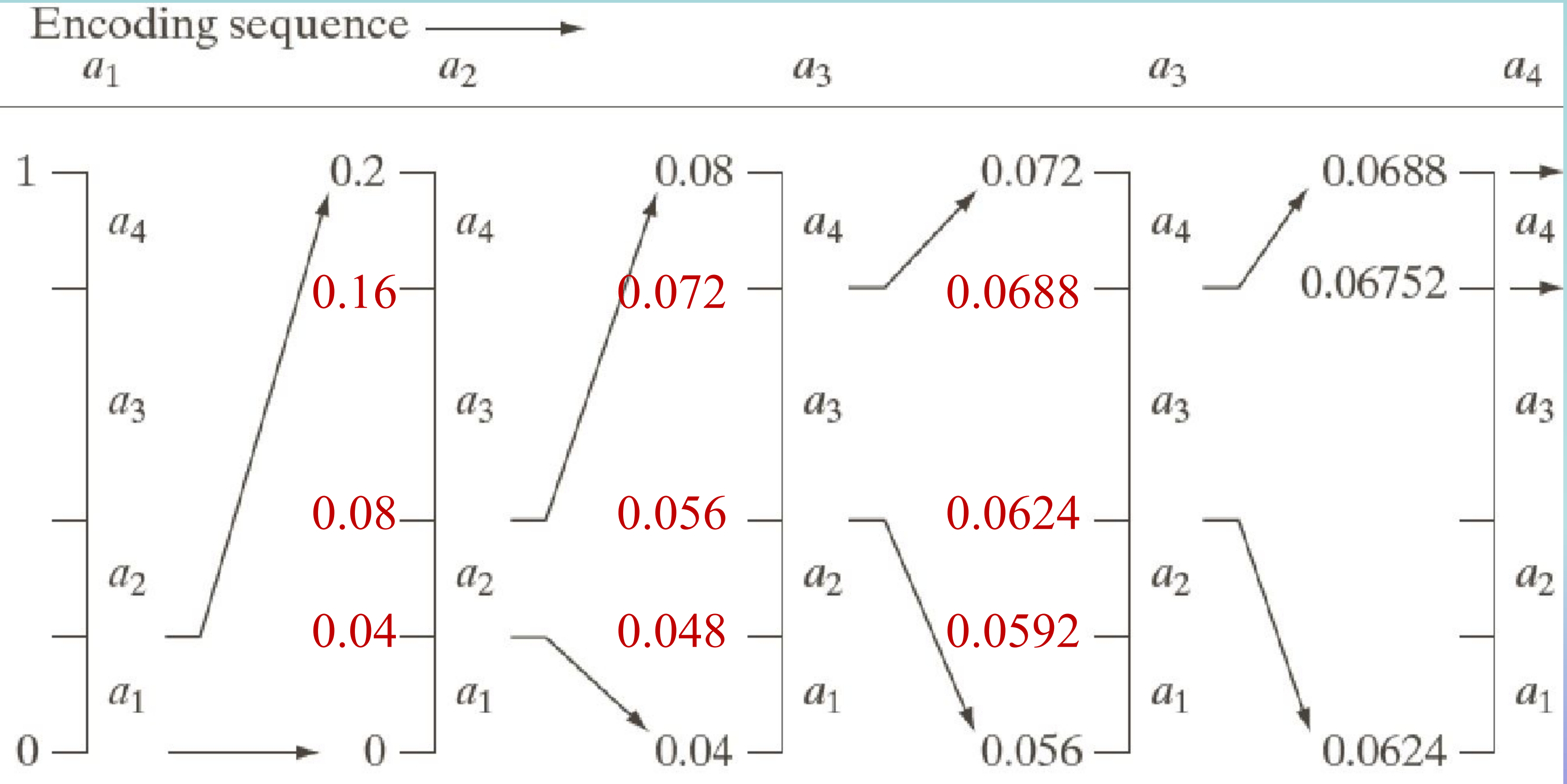


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



Decoding

0.068 $a_1a_2a_3a_3a_4$

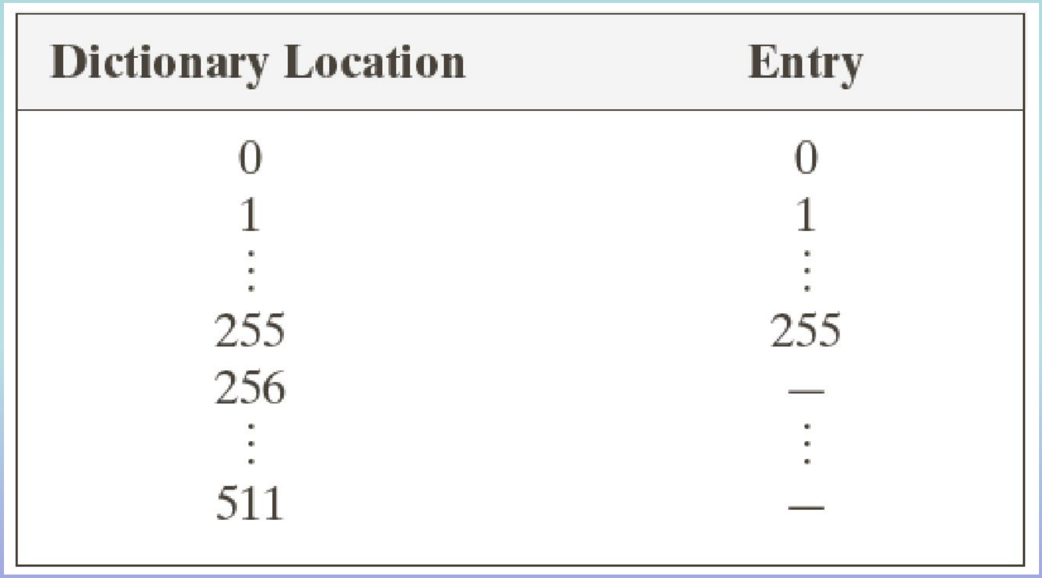


Lempel-Ziv-Welch (LZW) Coding

- An error-free compression technique
- Removes spatial redundancy
- Assign fixed-length code words to variable length sequences of source symbols
- It does not require any knowledge of probability of occurrence
- LZW coding is used in the GIF, TIFF and PDF formats

LZW Encoding – Images

- Images are scanned from left to right and from top to bottom
- A codebook or dictionary containing the source symbols to be coded is constructed on the fly
- For 8-bit monochrome images, first 256 words of the dictionary are assigned to intensities 0,1,2,3,…,255.



Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

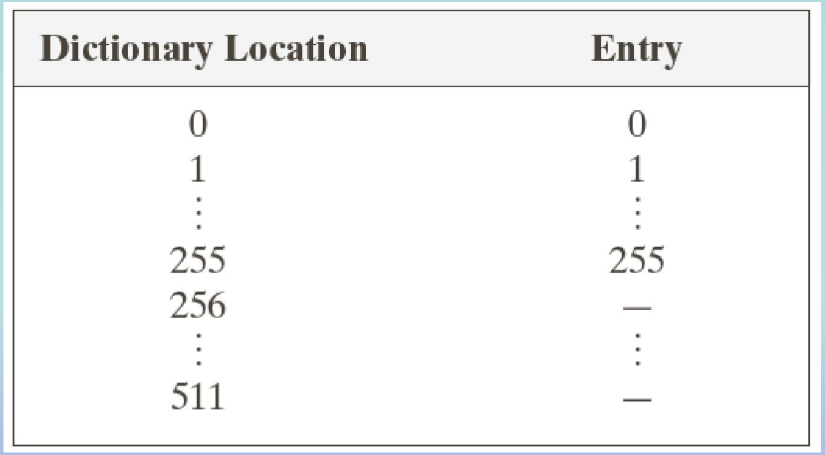
Lempel-Ziv-Welch (LZW) Coding

<u>Coding Technique</u>

- Generate a codebook/dictionary
  - first 256 entries are assigned to gray levels 0,1,2,..,255.
  - New gray level sequences not already in the dictionary are assigned to a new entry.
  - Example: for sequence “255-255” can be assigned to entry# 256, the address following the locations reserved for gray levels 0 to 255.

LZW Encoding – Images

- The next time that two consecutive white pixels are encountered, codeword 256 (the address of the location containing 255-255) is used to represent them
- If 9-bit, 512-word dictionary is employed then two pixels (16 bits) can be represented by 9 bits only
-



Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Lempel-Ziv-Welch (LZW) Coding

- Example

Consider the following 4 x 4 8 bit image

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	**Entry**
0	0
1	1
.	.
255	255
256	-
511	-

Initial Dictionary

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	-
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	-
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39				

Sequence= Concatenate(‘ ’, ‘39’) = 39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	-
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39			

Sequence= Concatenate('39', '39') = 39-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39

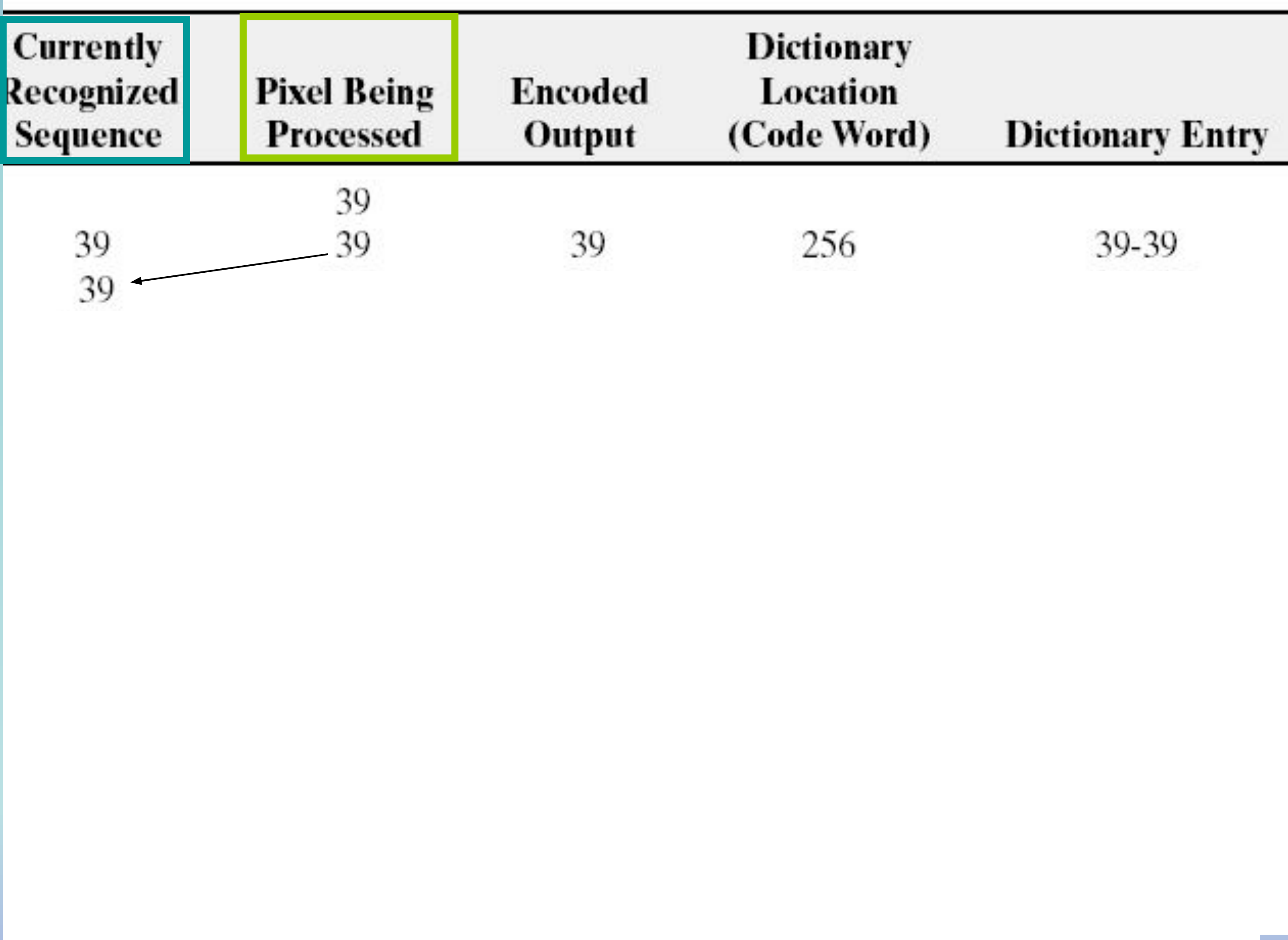
Address of '39'

Sequence= Concatenate('39', '39') = 39-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39				

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126			

Sequence= Concatenate('39', '126') = 39-126

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126

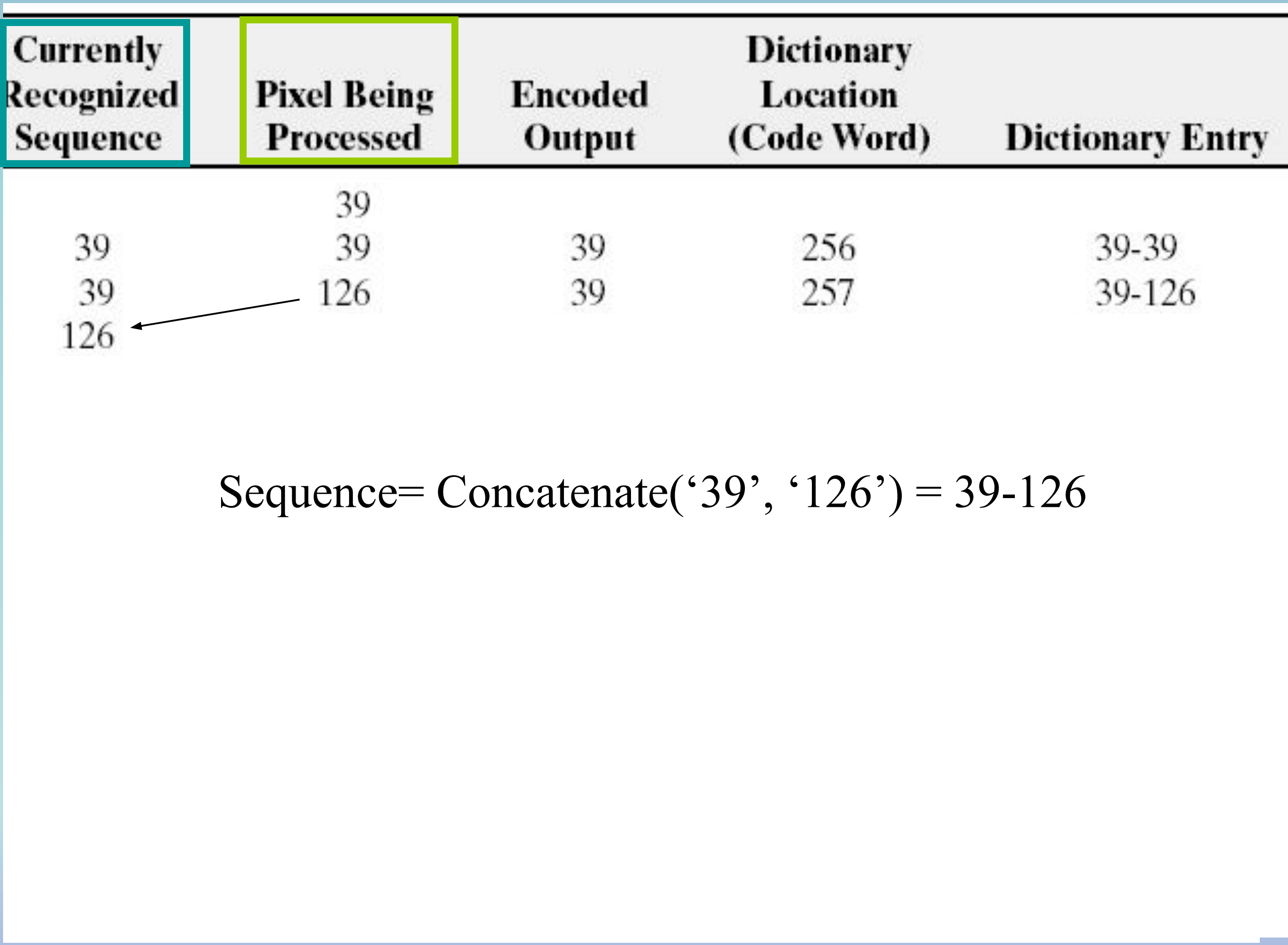
Address of ‘39’

Sequence= Concatenate(‘39’, ‘126’) = 39-126

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



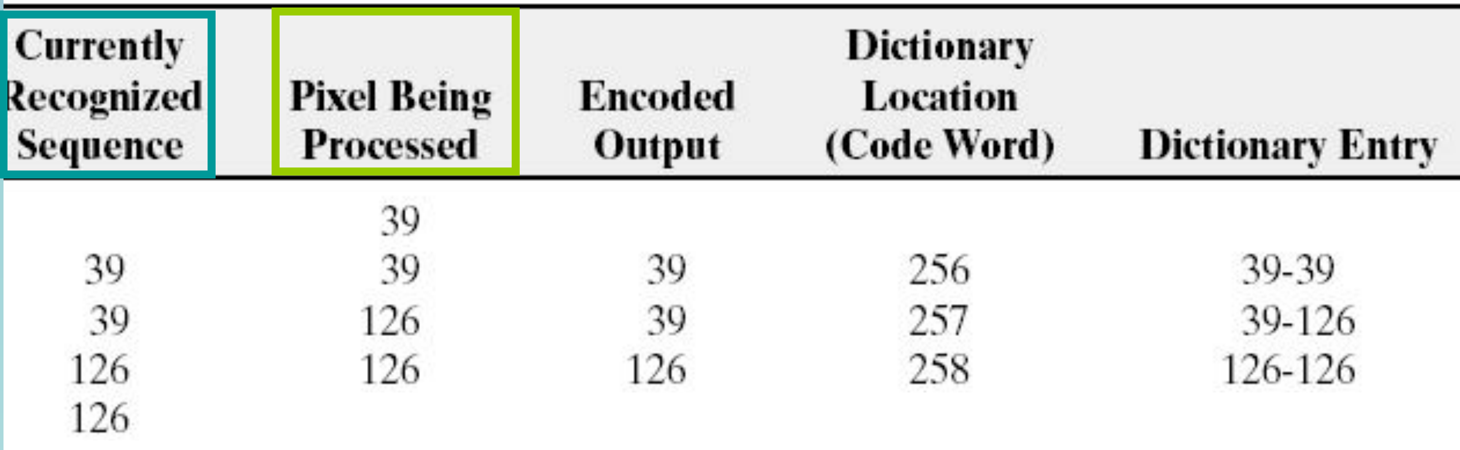
Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126				

Sequence= Concatenate(‘39’, ‘126’) = 39-126

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126				

Sequence= Concatenate(‘126’, ‘126’) = 126-126

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

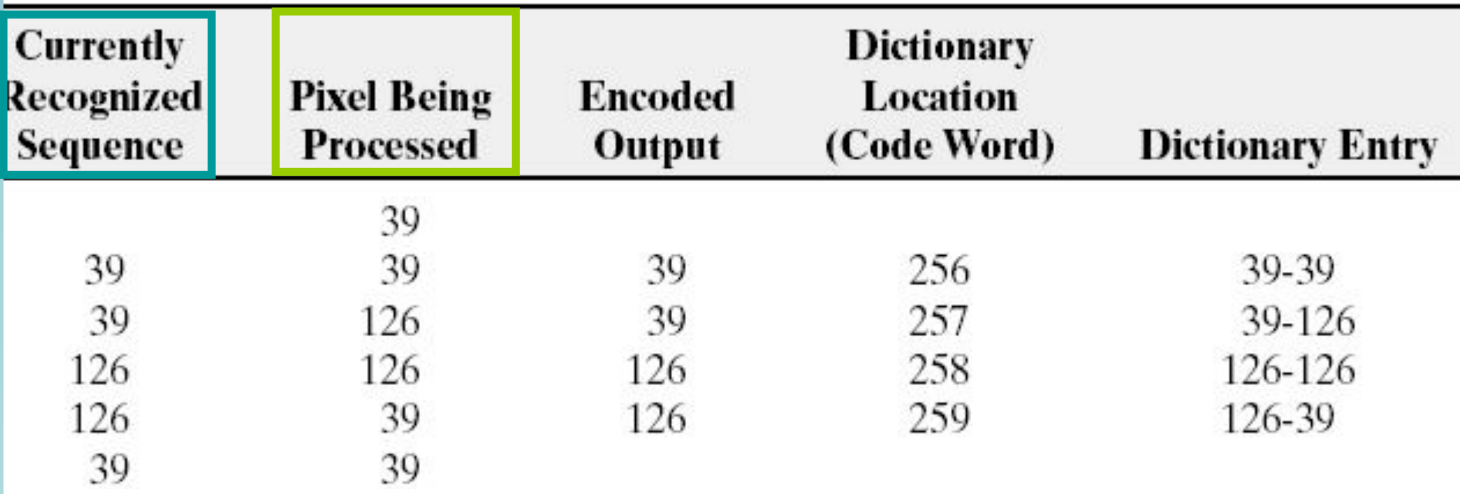
Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39				

Sequence= Concatenate('126', '39') = 126-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			

Sequence= Concatenate('39', '39') = 39-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			

Sequence= Concatenate('39', '39') = 39-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39				

Sequence= Concatenate('39', '39') = 39-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

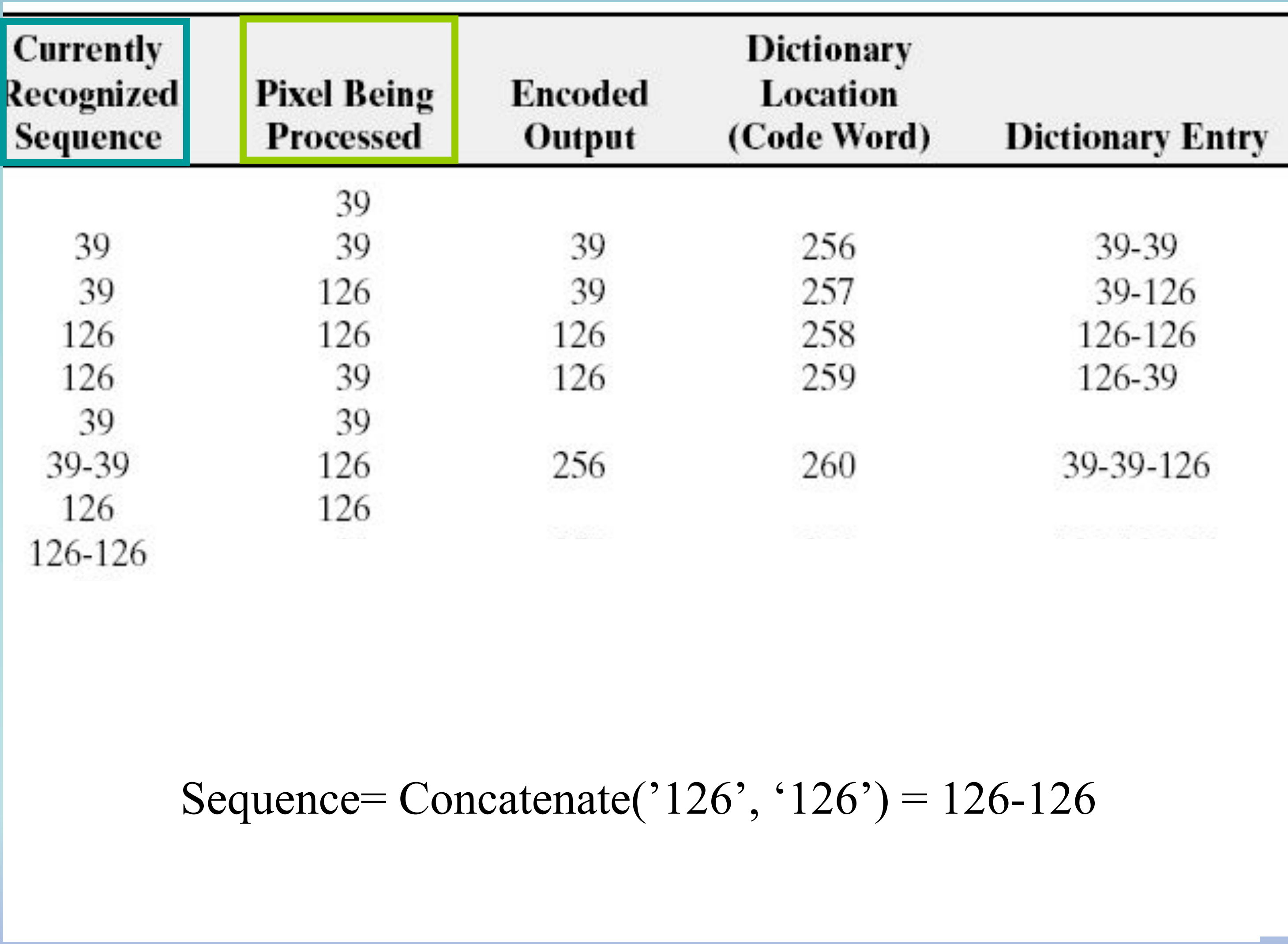
Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126				

Sequence= Concatenate('39-39', '126') = 39-39-126

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



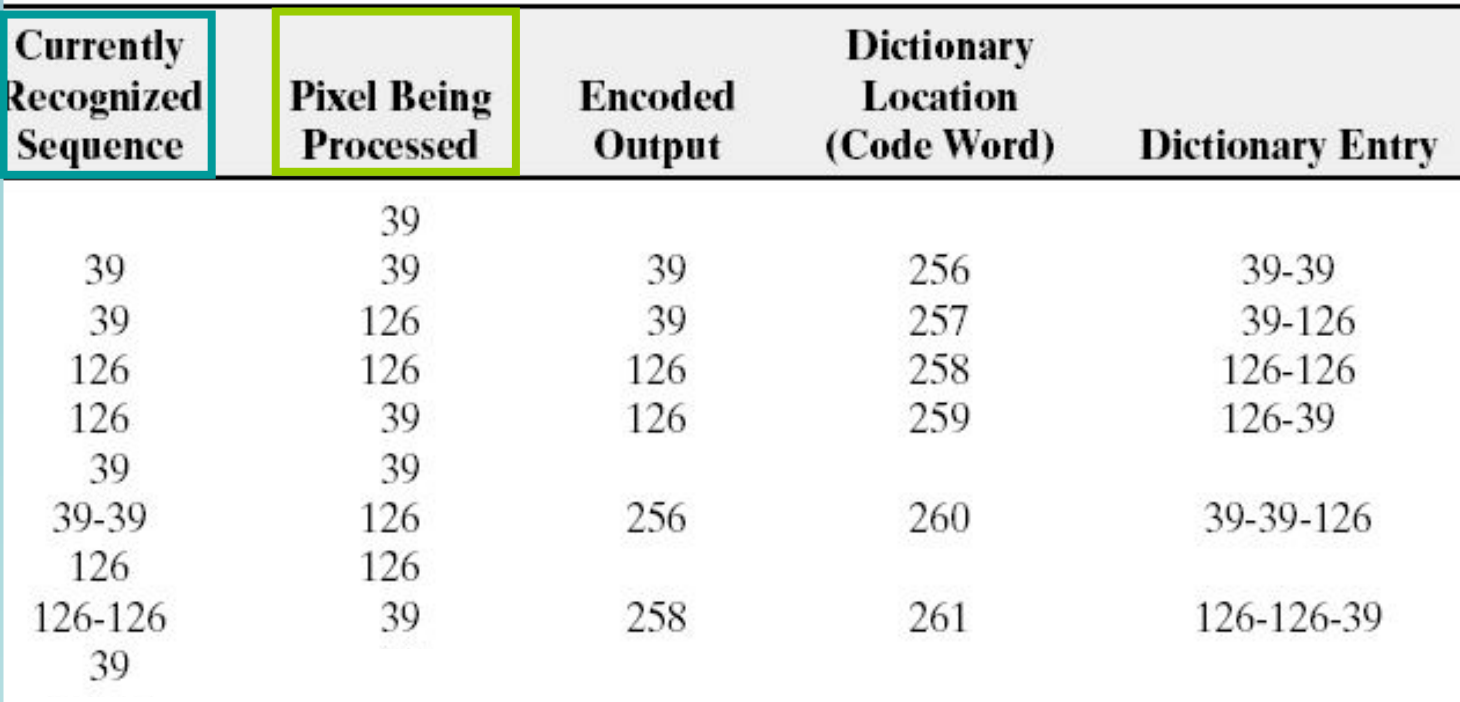
Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126				

Sequence= Concatenate('126', '126') = 126-126

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



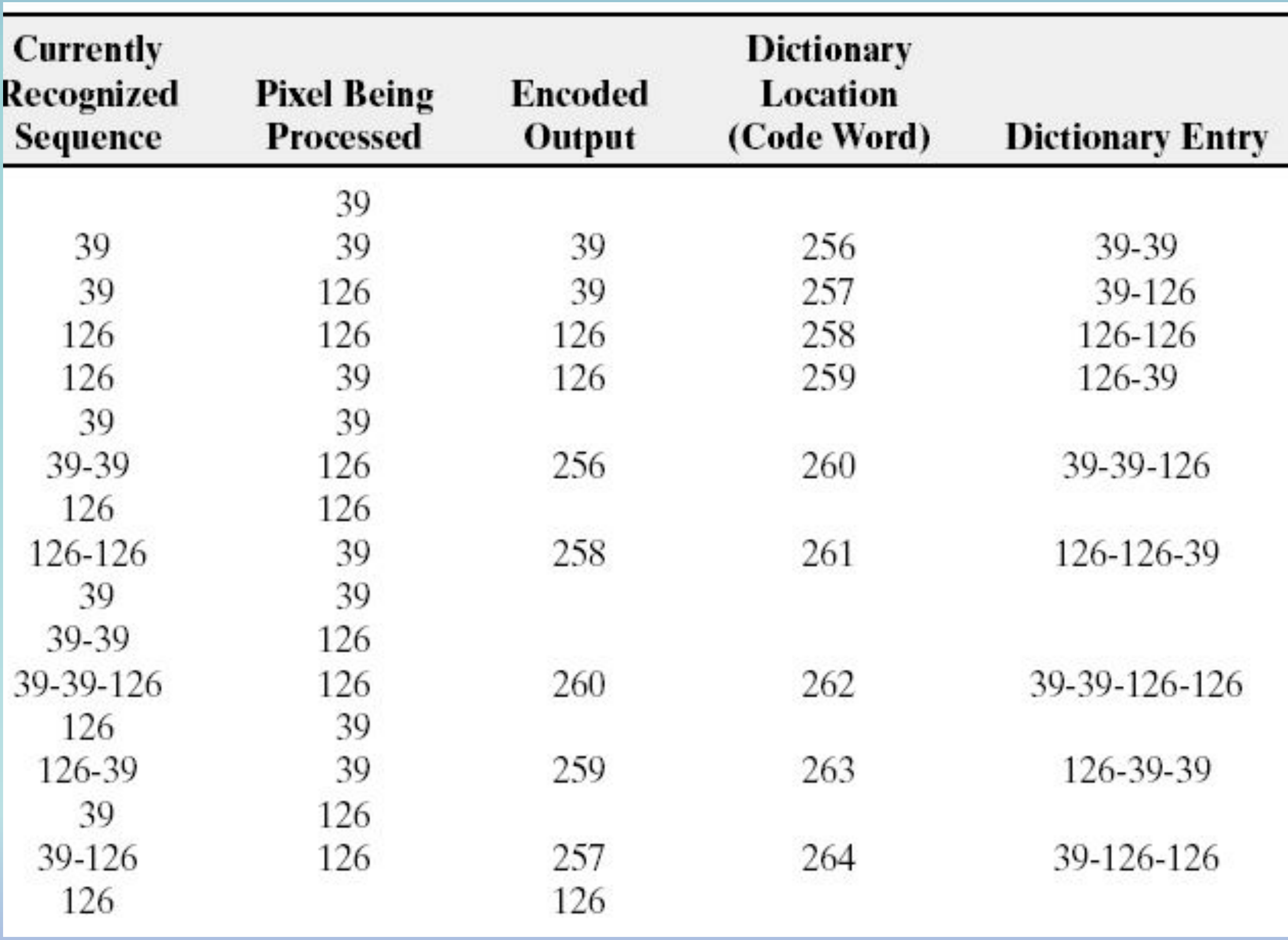
Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39				

Sequence= Concatenate('126-126', '39') = 126-126-39

Lempel-Ziv-Welch (LZW) Coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-



Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

Lempel-Ziv-Welch (LZW) Coding

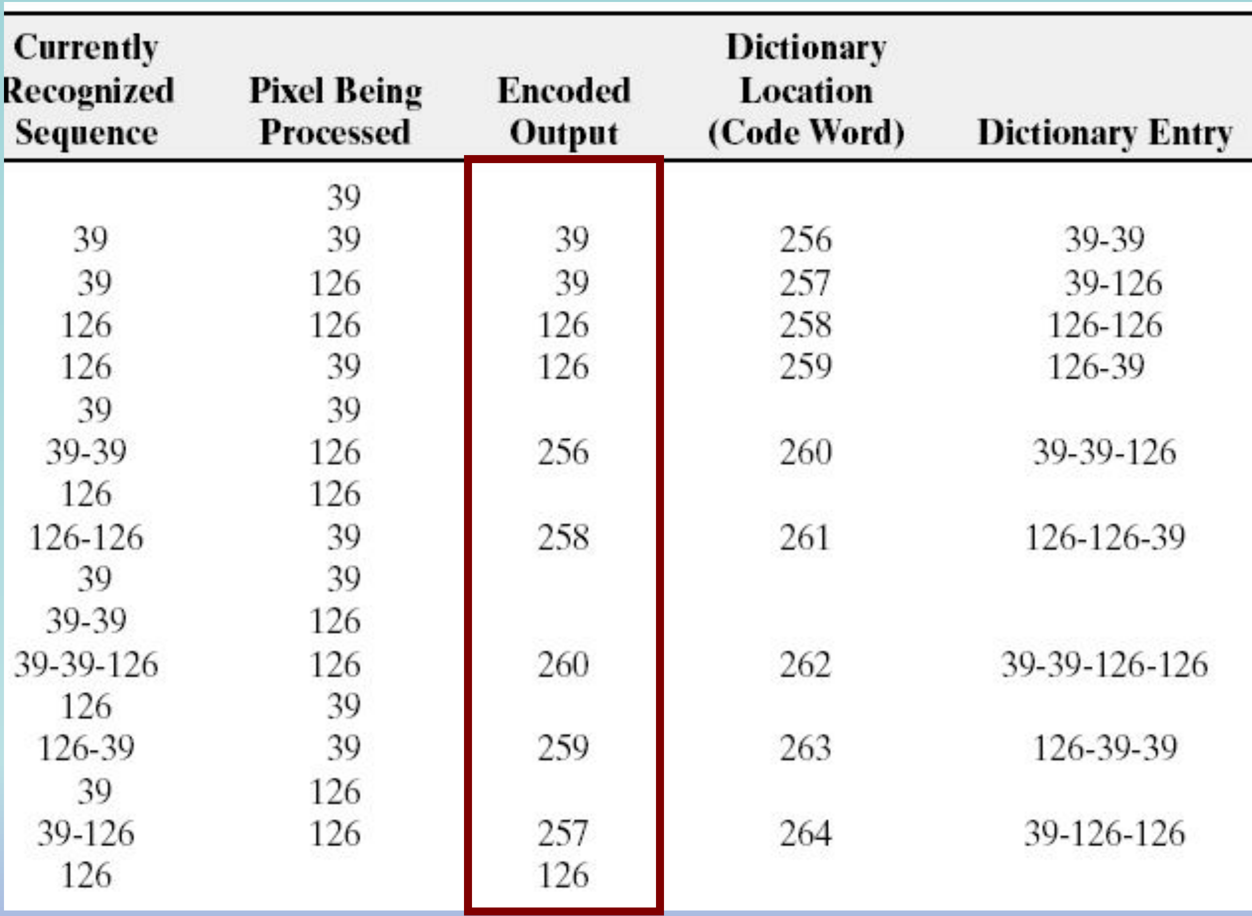
Encoding of

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

is

39 39 126 126
256 258 260 259 257 126

How to decode?



Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

LZW Decoding

Decoding of

39 39 126 126 256 258 260 259 257 126

is

39

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39~~ 39 126 126 256 258 260 259 257 126

is

39 39

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39~~ ~~39~~ 126 126 256 258 260 259 257 126

is

39 39 126

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39~~ ~~39~~ ~~126~~ 126 256 258 260 259 257 126

is

39 39 126 126

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39 39 126 126~~ 256 258 260 259 257 126

is

3939 126 126

39 39

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39 39 126 126 256~~ 258 260 259 257 126

is

3939 126 126

39 39 126 126

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39 39 126 126 256 258~~ 260 259 257 126

is

3939 126 126

39 39 126 126

39 39 126

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39 39 126 126 256 258 260~~ 259 257 126

is

3939 126 126

39 39 126 126

39 39 126 126

39

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39 39 126 126 256 258 260 259~~ 257 126

is

3939 126 126

39 39 126 126

39 39 126 126

39 39 126

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

LZW Decoding

Decoding of

~~39 39 126 126 256 258 260 259 257~~ 126

is

3939 126 126

39 39 126 126

39 39 126 126

39 39 126 126

Dictionary Code word	Dictionary entry
0	0
1	1
|	|
39	39
|	|
126	126
|	|
255	255
256	39-39
257	39-126
258	126-126
259	126-39
260	39-39-126
261	126-126-39
262	39-39-126-126
263	126-39-39
264	39-126-126

Compression Ratio using LZW Coding

- Suppose sequence **39-39-126** is replaced by **260**
- In other words **3X8=24** bits are replaced by **9 bits**

- Original image = 16X8 bits = **128 bits**
- Compressed Image = 10X9 bits = **90 bits**
- Compression Ratio = **128:90** = 1.42:1

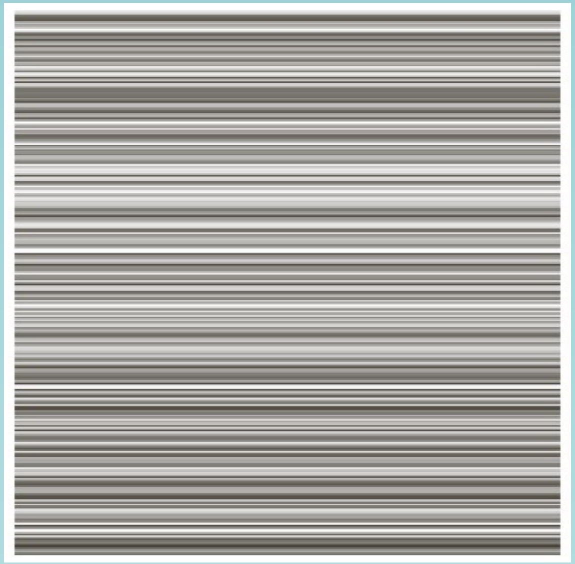
Recall

Run Length Encoding

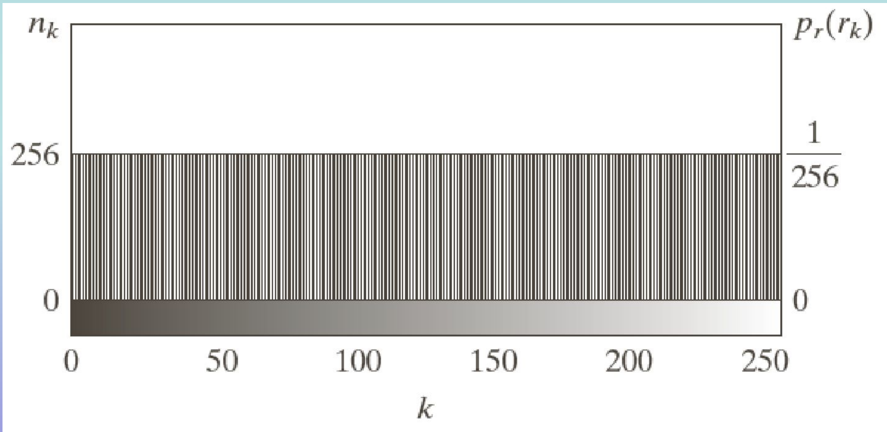
- Image features
  - All 256 gray levels are equally probable uniform histogram

    (variable length coding can not be applied)
  - The gray levels of each line are selected randomly so pixels are independent of one another in vertical direction
  - Pixels along each line are identical, they are completely dependent on one another in horizontal direction

Spatial redundancy



A computer generated (synthetic) 8-bit image M = N = 256

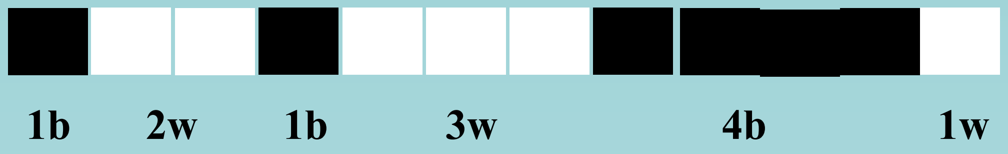


Recall

Run Length Encoding

- The spatial redundancy can be eliminated by using *run-length pairs (a mapping scheme)*
- **Run length pairs** has two parts
  - Start of new intensity
  - Number of consecutive pixels having that intensity
- Example (consider the image shown in previous slide)
  - Each 256 pixel line of the original image is replaced by a single 8-bit intensity value
  - Length of consecutive pixels having the same intensity = 256
  - Compression Ratio = $\boxed{\dfrac{256 \text{ x } 256 \text{ x } 8}{[256+256] \text{ x } 8} = 128}$

Run Length Encoding (RLC)



- Uses run length pairs
  - (0, 1), (1, 2), (0, 1), (1, 3), …
- Eliminates small **spatial redundancies**
- However, *small runs results in expansion instead of compression*

EXAMPLE 10.2.9:

Given the following 8x8, 4-bit image:

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 0 & 0 & 0 & 10 & 10 & 10 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 5 & 10 & 10 & 9 & 9 & 10 \\ 5 & 5 & 5 & 4 & 4 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

EXAMPLE 10.2.9 (contd):

The corresponding gray levels pairs are as follows:

First row: 10,8

Second row: 10,5 12,3

Third row: 10,5 12,3

Fourth row: 0,3 10,3 0,2

Fifth row: 5,3 0,5

Sixth row: 5,3 10,2 9,2 10,1

Seventh row: 5,3 4,3 0,2

Eighth row: 0,8

These numbers are then stored in the RLC compressed file as:

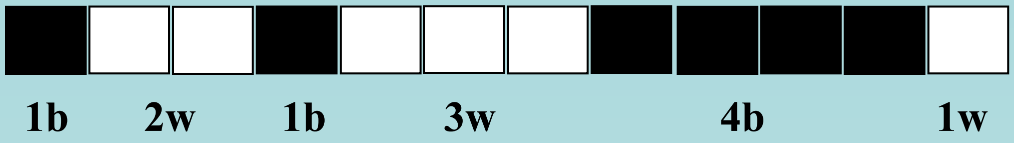
10,8,10,5,12,3,10,5,12,3,0,3,10,3,0,2,5,3,0,5,5,3,10,2,9,2,10,1,5,3,4,3,0,2,0,8

Run Length Encoding (RLC)

- ❑ Original image = 64X4 bits = **256 bits**
- ❑ Compressed Image = 36X4 bits = **144 bits**
- ❑ Compression Ratio = **64:36** = 16:9

Run Length Encoding (RLE) in Binary Image

- Run-length Encoding is also effective in case of **binary images**
- can be represented as a **sequence of runs only**
  - Example:



  - Its representation: 121341
- Fix a way to determine the run values:
  - Specify the value of first run
  - Assume each row begins with a *white* run!!

RLE – Binary Images

- Adjacent pixels in binary images are more likely to be identical
  - *Scan an image row from left to right and code each contiguous group (i.e. run) of 0s or 1s according to its length*
  - *Establish a convention to determine the value of the run*
- Common conventions are
  - *To specify the value of the first run of each row*
  - *To assume that each row begins with a white run, whose run length may in fact be zero*

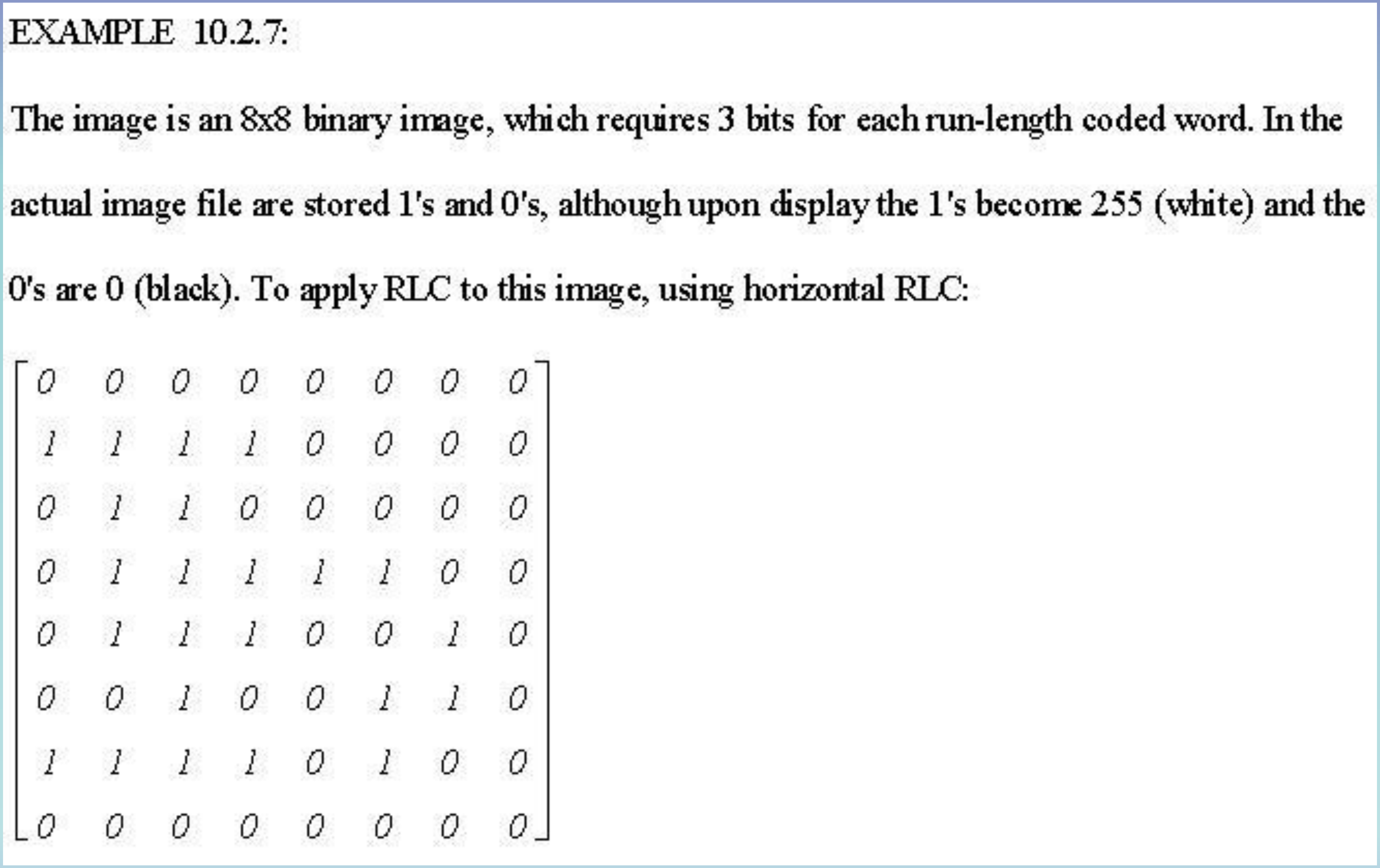
RLE – Binary Images

- Example:

  000001100000100000000101100000 (30 bits in a row)

- Specify the value of the first run of each row:
  - *0 5 2 5 1 8 1 1 2 5: gray level of the first run is ‘0’ (black)*

- Assume each row begins with a white run (bit ‘1’):
  - *0 5 2 5 1 8 1 1 2 5: length of the first (white) run is 0*



EXAMPLE 10.2.7:

The image is an 8x8 binary image, which requires 3 bits for each run-length coded word. In the actual image file are stored 1's and 0's, although upon display the 1's become 255 (white) and the 0's are 0 (black). To apply RLC to this image, using horizontal RLC:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Use the convention that the first number corresponds to the number of zeros (black) in a run

EXAMPLE 10.2.7 (contd):

The RLC numbers are:

First row: 8

Second row: 0, 4, 4

Third row: 1, 2, 5

Fourth row: 1, 5, 2

Fifth row: 1, 3, 2, 1, 1

Sixth row: 2, 1, 2, 2, 1

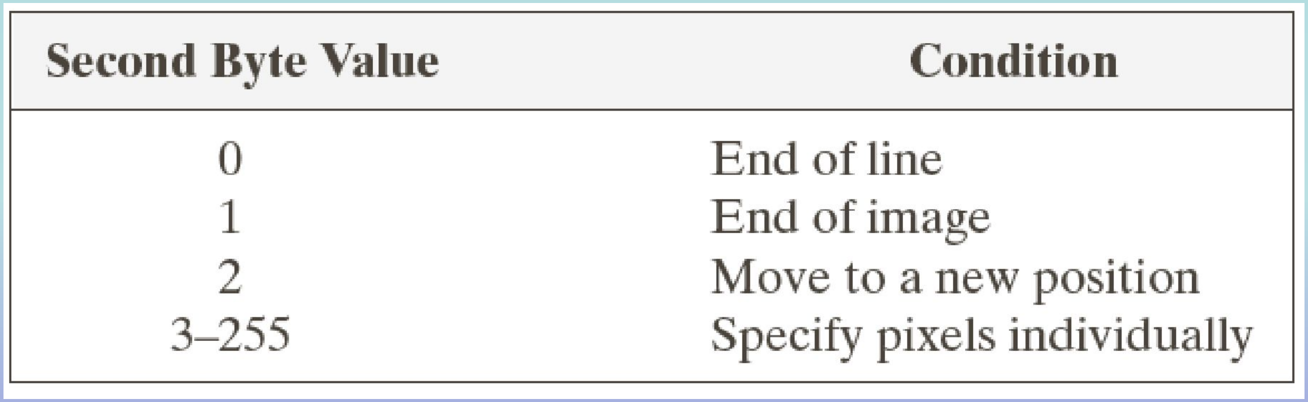
Seventh row: 0, 4, 1, 1, 2

Eighth row: 8

Note that in the second and seventh rows, the first RLC number is 0, since we are using the convention that the first number corresponds to the number of zeros in a run

Run Length Encoding in BMP

- Uses a combination of ***encoded*** and ***absolute*** mode
- Either mode can appear anywhere in the image
- **Encoded mode**:
  - 2 byte RLC
  - First byte: run length --Second byte: gray/color index
- **Absolute mode**:
  - 2 byte RLC
  - First byte: 0 --Second byte: as follows



Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3–255	Specify pixels individually