Multimedia Information Retrieval and Technology

Lecture 18
Lossless Compression Algorithms

By : Fangyu Wu

Room: SD555

Xi'an Jiaotong-Liverpool University
西交利物浦大学

1. Introduction
2. Variable-Length Coding (VLC)
   - **Shannon-Fano Algorithm**
   - **Huffman Coding Algorithm**
3. Basics of Information Theory

Introduction

Compression: The process of coding that will effectively reduce the total number of bits needed to represent certain information.

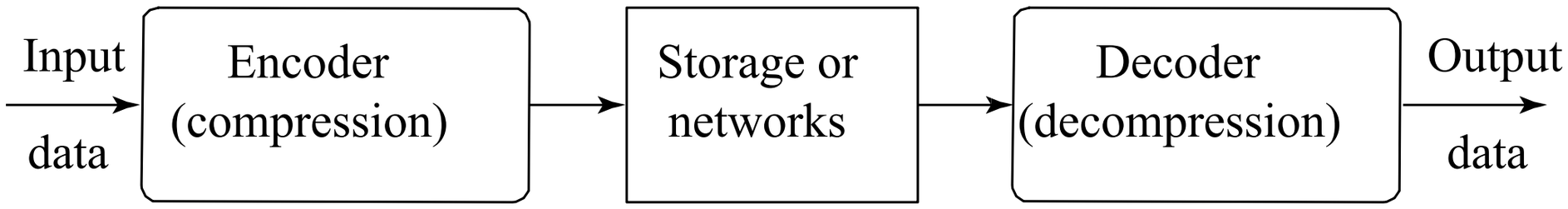


Fig. 7.1: A General Data Compression Scheme.

Introduction (cont'd)

We call the output of the encoder: *codes* or *codewords*.

If the compression and decompression processes induce no information loss, the compression scheme is *lossless;* otherwise, it is *lossy.*

Today we concentrate on lossless compression.

Introduction (cont'd)

- **Compression ratio:**

$$= \frac{B_0}{B_1}$$

B_0 – number of bits before compression

B_1 – number of bits after compression

1. Introduction
2. Variable-Length Coding (VLC)
   - **Shannon-Fano Algorithm**
   - **Huffman Coding Algorithm**
   - **Adaptive Huffman Coding Algorithm**
3. Basics of Information Theory

Variable-Length Coding (VLC)

Variable-length coding (VLC) - the more frequently appearing symbols are coded with fewer bits, and vice versa.

As a result, fewer bits are usually needed to represent the whole collection.

- **Shannon-Fano Algorithm**
- **Huffman Coding Algorithm**
- **Adaptive Huffman Coding Algorithm**

Variable-Length Coding (VLC)

Shannon-Fano Algorithm — a top-down approach

1. Sort the symbols according to the frequency count of their occurrences.
2. Recursively divide the symbols into two parts, each with the same number of counts, until all parts contain only one symbol.

An Example: coding of “HELLO”

Symbol	H	E	L	O
Count	1	1	2	1

Frequency count of the symbols in ”HELLO”.

Shannon-Fano Algorithm

A natural way of implementing the above procedure is to build a binary tree.

As a convention, let's assign bit 0 to its left branches and 1 to the right branches.

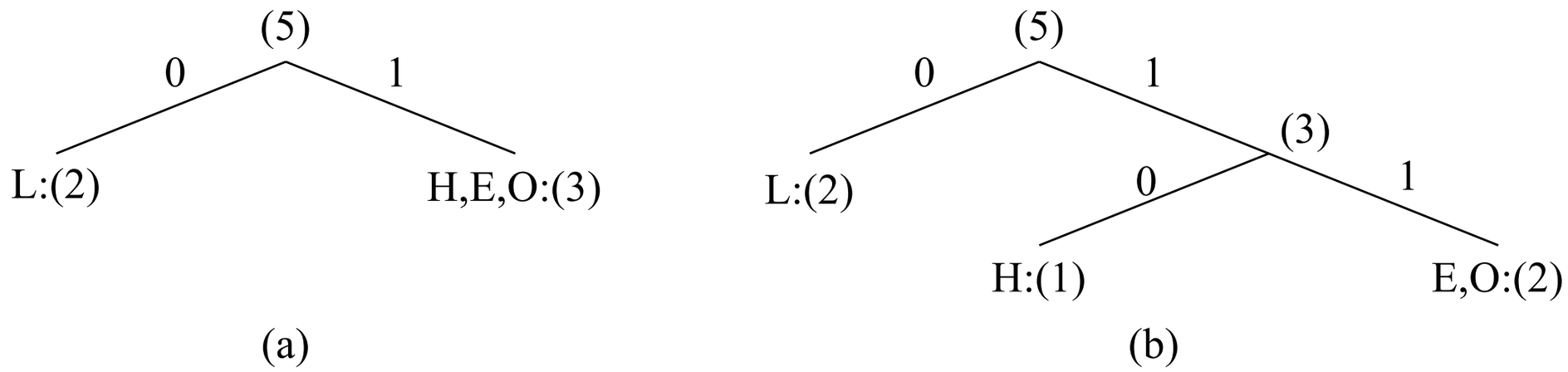


Fig. 7.3: Coding Tree for HELLO by Shannon-Fano.

Shannon-Fano Algorithm

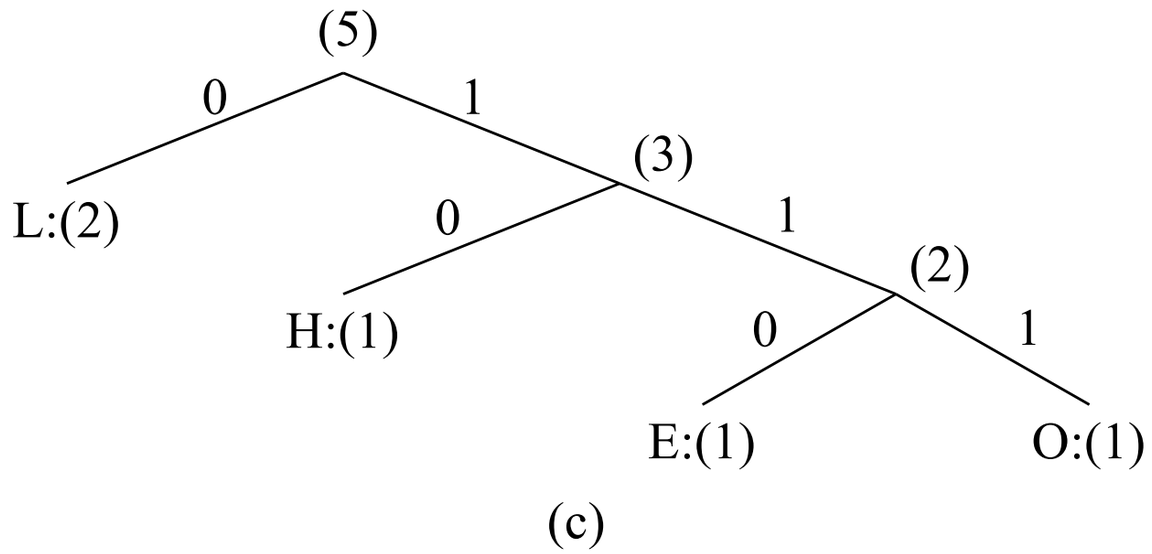


Fig. 7.3: Coding Tree for HELLO by Shannon-Fano.

Shannon-Fano Algorithm

Table 7.1: Result of Performing Shannon-Fano on HELLO (each symbol, its frequency count, information content, resulting codeword, and the number of bits needed to encode HELLO.)

Symbol	Count	ount	Code	# of bits used
L	2	1.32		
H	1	2.32		
E	1	2.32		
O	1	2.32		
TOTAL number of bits:				

Shannon-Fano Algorithm

Table 7.1: Result of Performing Shannon-Fano on HELLO (each symbol, its frequency count, information content, resulting codeword, and the number of bits needed to encode HELLO.)

Symbol	Count	ount	Code	# of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
TOTAL number of bits:				10

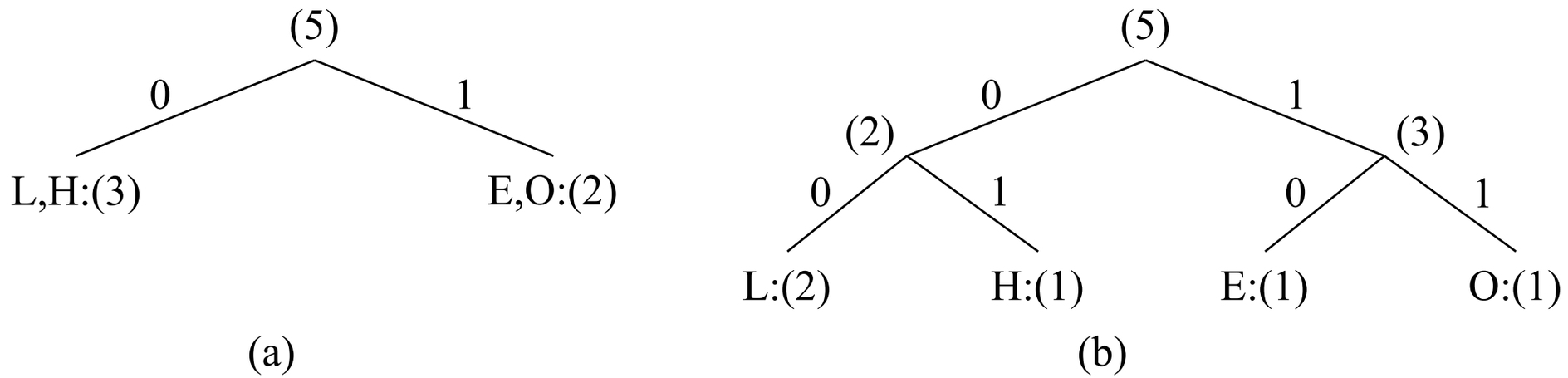


Fig. 7.4 Another coding tree for HELLO by Shannon-Fano.

Table 7.2: Another Result of Performing Shannon-Fano on HELLO

Symbol	Count	ount	Code	# of bits used
L H E O	2	1.32	00	4
	1	2.32	01	2
	1	2.32	10	2
	1	2.32	11	2
TOTAL number of bits:				10

Exercise

Construct a Shannon-Fano tree for the string HELLOLEO.
Determine the Code for each Symbol.

1. Introduction
2. Variable-Length Coding (VLC)
   - **Shannon-Fano Algorithm**
   - **Huffman Coding Algorithm**
   - **Adaptive Huffman Coding Algorithm**
3. Basics of Information Theory

Huffman Coding

First presented by David A. Huffman in a 1952 paper, this method attracted an over whelming amount of research and has been adopted in many commercial applications, such as fax machines, JPEG, and MPEG.

A *bottom-up* approach:

Huffman Coding

ALGORITHM: HUFFMAN CODING

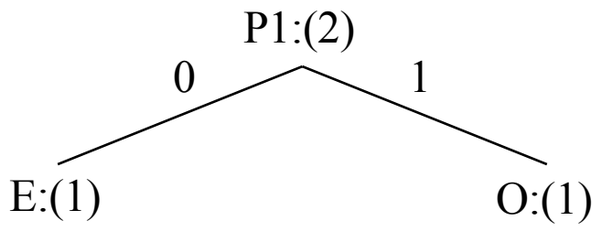
1. Initialization: Put all symbols on a list sorted according to their frequency counts.
2. Repeat until the list has only one symbol left:

   ((1)) Pick **two symbols with the lowest frequency counts**. Form a Huffman subtree that has these two symbols as child nodes and create a parent node.

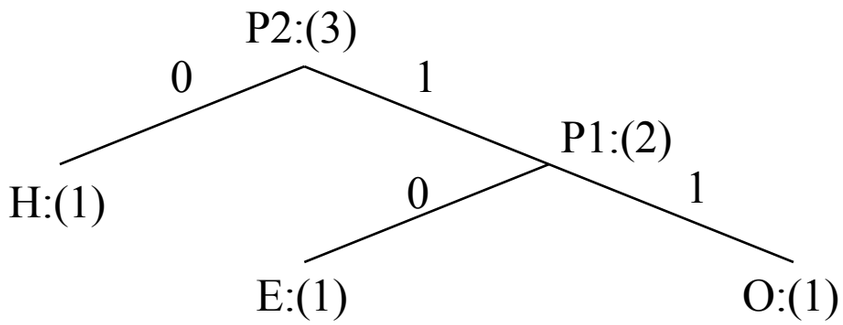
   ((2)) Assign the sum of the children's frequency counts to the parent and insert it into the list such that the order is maintained.

   ((3)) Delete the children from the list.
3. Assign a codeword for each leaf based on the path **from the root.**

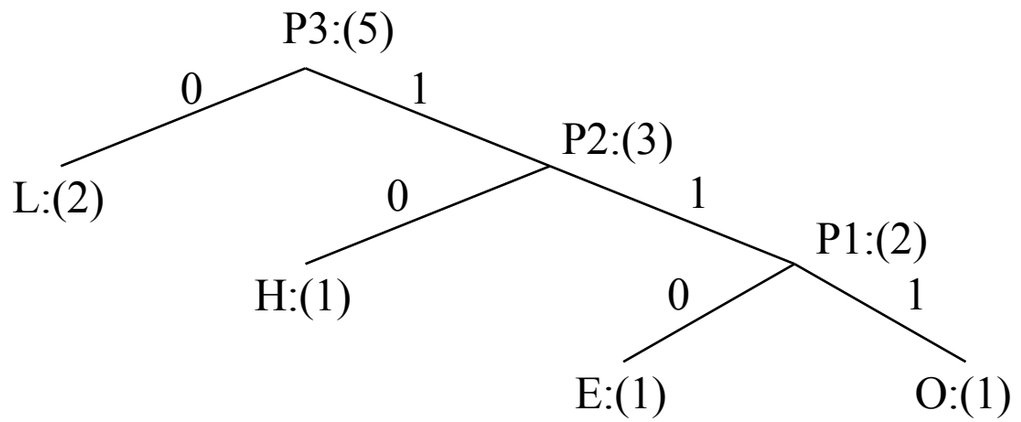
Coding Tree for “HELLO” using the Huffman Algorithm？



(a)



(b)



(c)

Fig. 7.5: Coding Tree for “HELLO” using the Huffman Algorithm.

Huffman Coding (cont'd)

In Fig. 7.5, new symbols P1, P2, P3 are created to refer to the parent nodes in the Huffman coding tree.

The contents in the list are illustrated below:

After initialization: L H E O

After iteration (a): L P1 H

After iteration (b): L P2

After iteration (c): P3

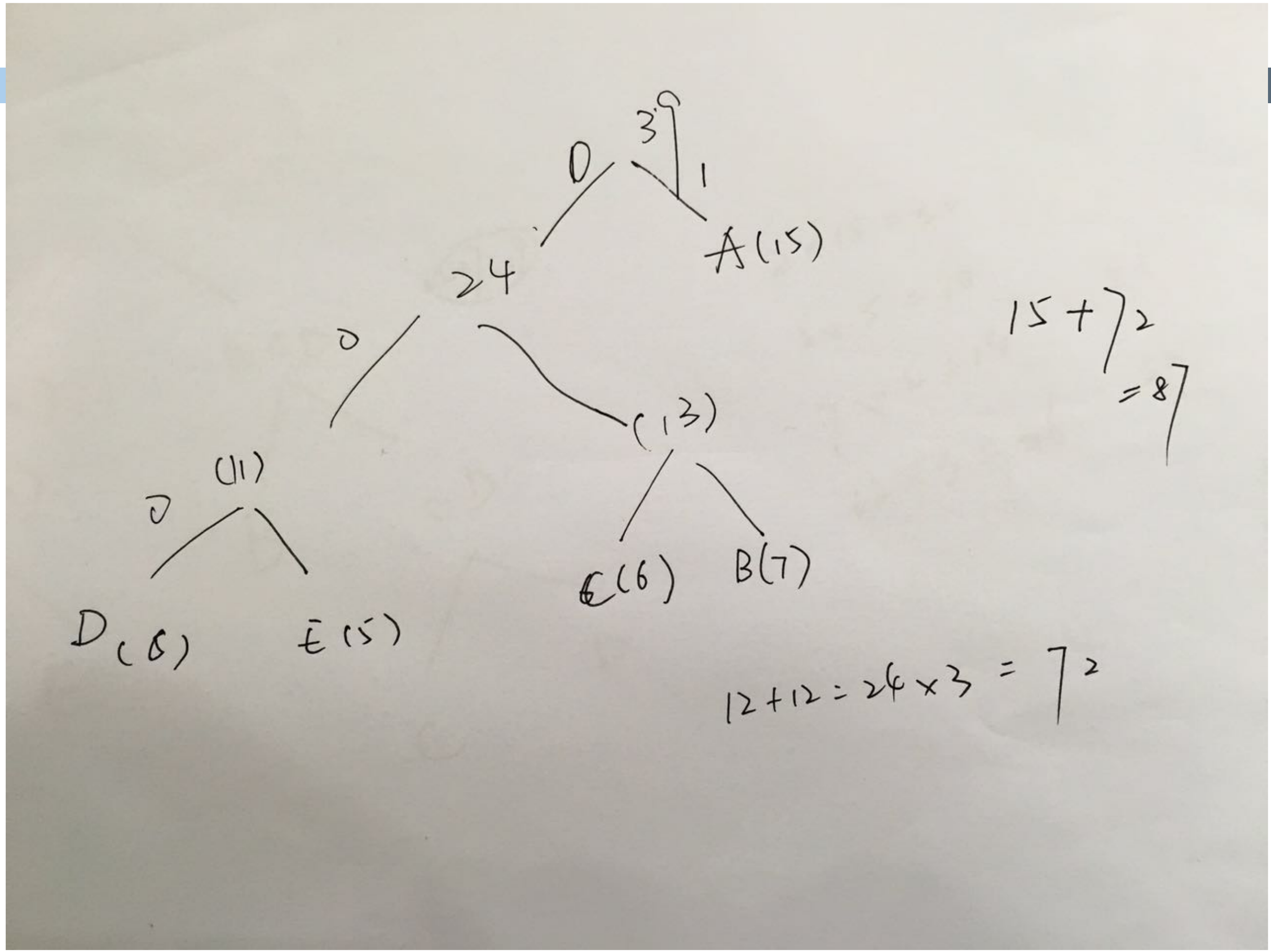
Example

As another simple example, consider a text string containing a set of characters and their frequency counts as follows: A:(l5), B:(7), C:(6), D:(6) and E:(5).

First try the Shannon-Fano algorithm, how many bits do we need to encode? Then try the Huffman algorithm.

Solution

Shannon-Fano algorithm needs a total of 89 bits to encode this string, whereas the Huffman algorithm needs only 87.


0
3
1
A(15)
24
0
(13)
(11)
D
D(8)
E(5)
15+2
=8
C(6)
B(7)
12+12=24×3=72

Properties of Huffman Coding

Unique Prefix Property: No Huffman code is a prefix of any other Huffman code - precludes any ambiguity in decoding.

L: 0

H:10

E:110

O:111

The unique prefix property is guaranteed by the above Huffman algorithm, since it always places all input symbols at the leaf nodes.

Properties of Huffman Coding

The code generated by the Shannon-Fano algorithm is another such example.

This property is essential and also makes for an efficient decoder, since it precludes any ambiguity in decoding.

Properties of Huffman Coding

Optimality: *minimum redundancy* code –

It has been proved that the Huffman code is optimal for a given data model (i.e., a given, accurate, probability distribution):

- The two least frequent symbols will have the same length for their Huffman codes, differing only at the last bit.

- Symbols that occur more frequently will have shorter Huffman codes than symbols that occur less frequently.

Properties of Huffman Coding

The code is instantaneous uniquely decodable.

Instantaneous: each code word in a string of code symbols can be decoded without referencing succeeding symbols.

Uniquely: any string of code symbols can be decoded in only one way.

Properties of Huffman Coding

- **Unique Prefix Property**
- **Optimality**
- **Instantaneous uniquely decodable**

After the code has been created, coding and error-free decoding is accomplished in a simple lookup table.

1. Introduction
2. Variable-Length Coding (VLC)
    - **Shannon-Fano Algorithm**
    - **Huffman Coding Algorithm**
3. Basics of Information Theory

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Basics of Information Theory

- The ***entropy*** η of an information *source* with alphabet $S = \{s_1, s_2, \ldots, s_n\}$ is:

$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

p_i: probability that symbol s_i occurs in S.

$\log_2 \frac{1}{p_i}$: indicates the amount of information contained in s_i, which corresponds to the number of bits needed to encode s_i.

- According to the famous scientist Claude E. Shannon, of Bell Labs

Basics of Information Theory

If the probability of having the character *n* in a manuscript is 1/32,

$$\log_2 \frac{1}{p_n} = 5$$

the amount of information associated with receiving this character is 5 bits.

In other words, a character string *nnn* will require 15 bits to code.

Basics of Information Theory

What is the entropy?

In science, entropy is a measure of the ***disorder*** of a system – the more entropy, the more disorder.

Typically, we add *negative* entropy to a system when we impart more order to it.

Distribution of Gray-Level Intensities

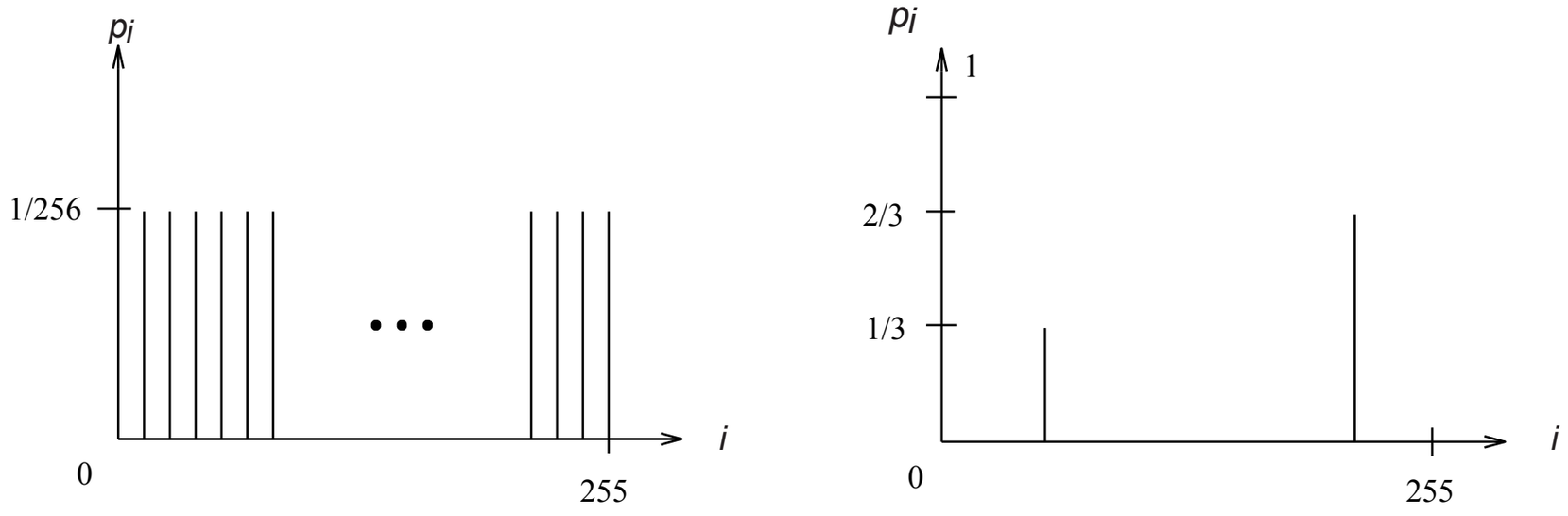


Fig. 7.2 Histograms for Two Gray-level Images.

Fig.7.2(a) shows the histogram of an image with *uniform* distribution of gray-level intensities, i.e., $\forall i\ p_i = 1/256$. Hence, the entropy of this image is?

(7.4)

Distribution of Gray-Level Intensities

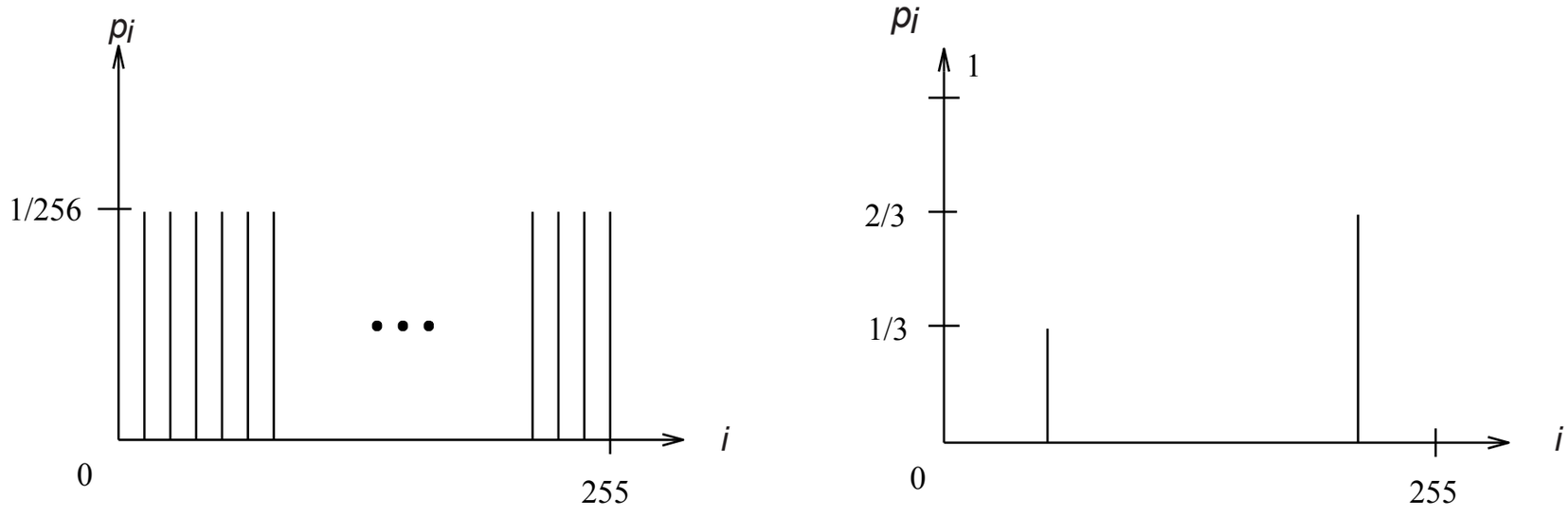


Fig. 7.2 Histograms for Two Gray-level Images.

Fig.7.2(a) shows the histogram of an image with *uniform* distribution of gray-level intensities, i.e., $\forall i\ p_i = 1/256$. Hence, the entropy of this image is: 8

(7.4)

Entropy and Code Length

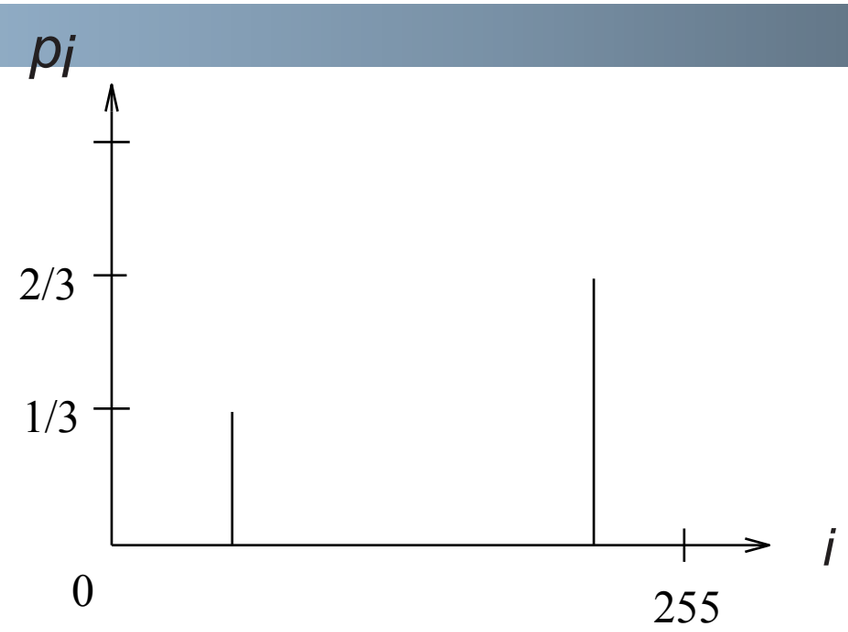


Figure 7.2(b) shows the histogram of another image, in which 1/3 of the pixels are rather dark and 2/3 of them are rather bright. The entropy of this image is

$$
\begin{aligned}
'\eta &= \frac{1}{3} \cdot \log_2 3 + \frac{2}{3} \cdot \log_2 \frac{3}{2} \\
&= 0.33 \times 1.59 + 0.67 \times 0.59 = 0.52 + 0.40 = 0.92
\end{aligned}
$$

Entropy and Code Length

The entropy η is a weighted-sum of terms $\log_2 \frac{1}{p_i}$: hence it represents the average amount of information contained per symbol in the source S.

The entropy η represents the minimum average number of bits required to represent each symbol in S. In other words, it specifies the lower bound for the average number of bits to code each symbol in S, i.e.,

$$\eta \leq \bar{l} \qquad (7.5)$$

$\bar{l}$ the average length (measured in bits) of the codewords produced by the encoder.

RECAP: Shannon-Fano Algorithm

Table 7.1: Result of Performing Shannon-Fano on HELLO (each symbol, its frequency count, information content, resulting codeword, and the number of bits needed to encode HELLO.)

Symbol	Count	ount	Code	# of bits used
L	2		0	
H	1		10	
E	1		110	
O	1		111	
TOTAL number of bits:				

RECAP: Shannon-Fano Algorithm

Table 7.1: Result of Performing Shannon-Fano on HELLO (each symbol, its frequency count, information content, resulting codeword, and the number of bits needed to encode HELLO.) **η=1.92**

Symbol	Count	ount	Code	# of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
TOTAL number of bits:				10

Entropy and Code Length

Coding schemes aim to get as close as possible to this theoretical lower bound.

In the above uniform-distribution example we found that $\eta = 8$: the minimum average number of bits to represent each gray-level intensity is at least 8.

In the context of imaging, this will correspond to the "worst case," where neighboring pixel values have no similarity.

Entropy and Code Length

In general, the entropy is greater when the probability distribution is flat and smaller when it is more peaked.

Exercise I (a)

(a) What is the entropy (η) of the image below, where numbers (0, 20, 50, 99) denote the gray-level intensities?

99	99	99	99	99	99	99	99
20	20	20	20	20	20	20	20
0	0	0	0	0	0	0	0
0	0	50	50	50	50	0	0
0	0	50	50	50	50	0	0
0	0	50	50	50	50	0	0
0	0	50	50	50	50	0	0
0	0	0	0	0	0	0	0

Solution

$$P_{20} = P_{99} = 1/8,\ P_{50} = 1/4,\ P_0 = 1/2.$$

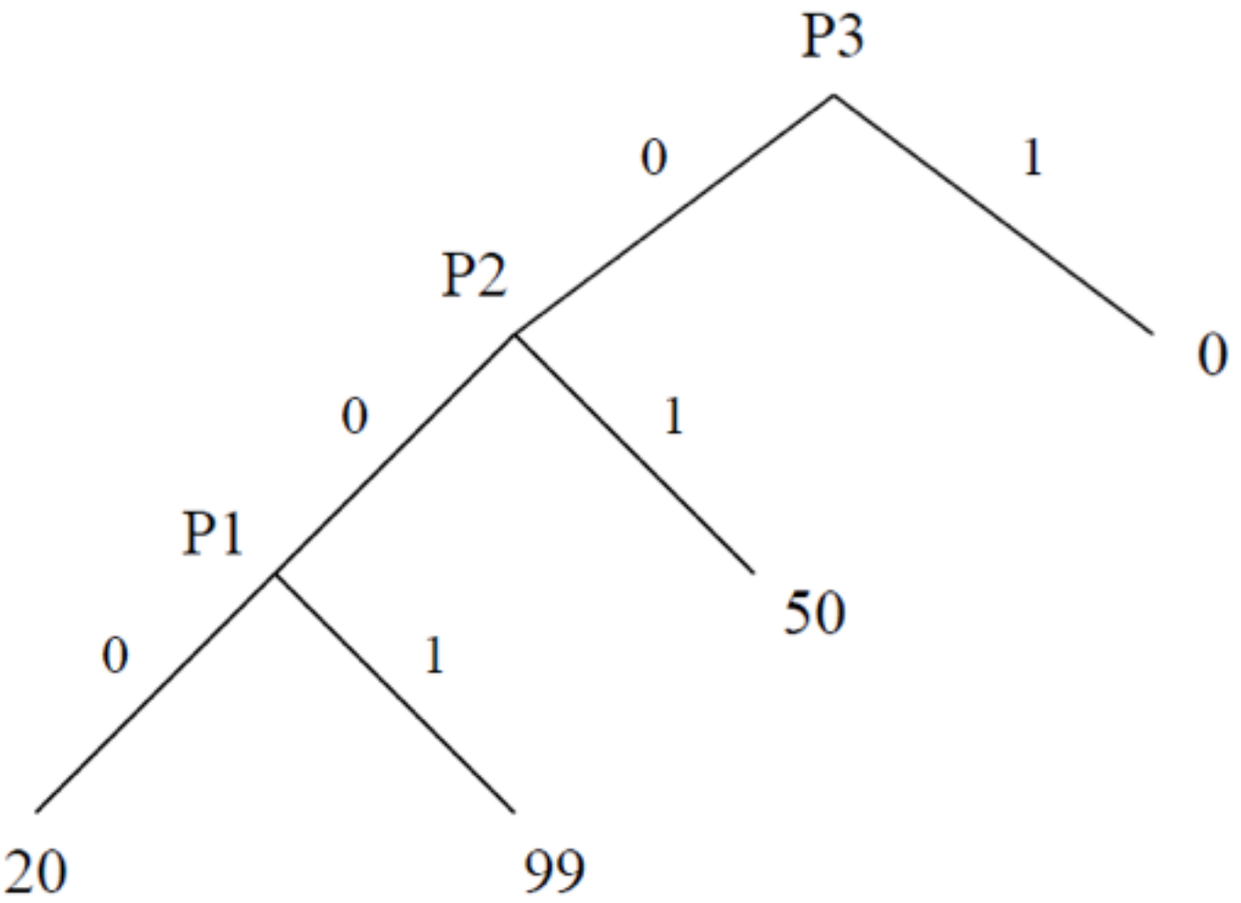
$$\eta = 2 \times \frac{1}{8}\log_2 8 + \frac{1}{4}\log_2 4 + \frac{1}{2}\log_2 2 = \frac{3}{4} + \frac{1}{2} + \frac{1}{2} = 1.75$$

Exercise I (b,c)

(b) Show step by step how to construct the Huffman tree to encode the above four intensity values in this image. Show the resulting code for each intensity value.

(c) What is the average number of bits needed for each pixel, using your Huffman code? How does it compare to η?

(b) Only the final tree is shown below. Resulting code: 0: "1", 50: "01", 20: "000", 99: "001"



(c) Average number of bits = $0.5 \times 1 + 0.25 \times 2 + 2 \times 0.125 \times 3 = 1.75.$

This happens to be identical to η **— it only happens when all probabilities are** 2^{-k} **where** k **is an integer. Otherwise, this number will be larger than** η**.**

Problem 8.9

(a) The entropy of the image is estimated using Eq. (8.1-7) to be

$$
\begin{aligned}
\tilde{H} &= -\sum_{k=0}^{255} p_r(r_k)\log_2 p_r(r_k) \\
&= -\left[\frac{12}{32}\log_2\frac{12}{32}+\frac{4}{32}\log_2\frac{4}{32}+\frac{4}{32}\log_2\frac{4}{32}+\frac{12}{32}\log_2\frac{12}{32}\right] \\
&= -[-0.5306-0.375-0.375-0.5306] \\
&= 1.811 \text{ bits/pixel.}
\end{aligned}
$$

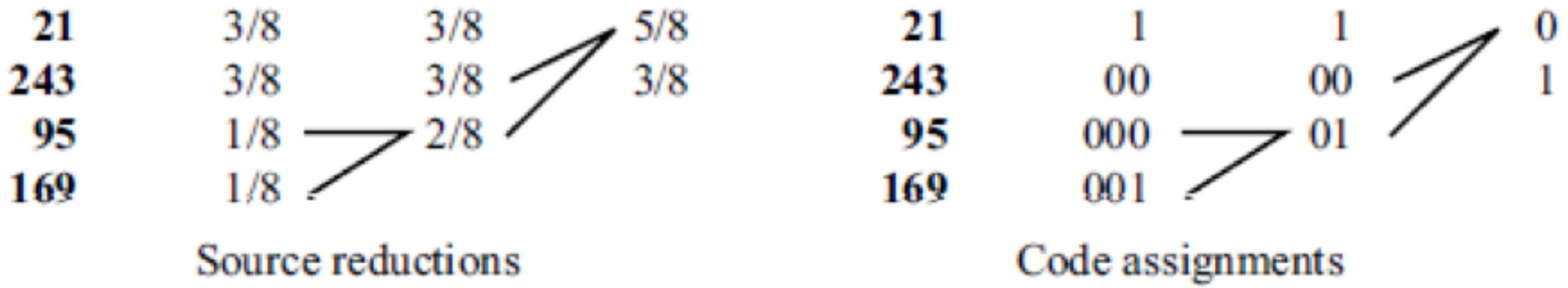


Figure P8.9

Xi'an Jiaotong-Liverpool University
西交利物浦大学

(**c**) Using Eq. (8.1-4), the average number of bits required to represent each pixel in the Huffman coded image (ignoring the storage of the code itself) is

$$L_{avg} = 1\left(\frac{3}{8}\right) + 2\left(\frac{3}{8}\right) + 3\left(\frac{1}{8}\right) + 3\left(\frac{1}{8}\right) = \frac{15}{8} = 1.875 \text{ bits/pixel.}$$

Thus, the compression achieved is

$$C = \frac{8}{1.875} = 4.27.$$

Because the theoretical compression resulting from the elimination of all coding redundancy is $\frac{8}{1.811} = 4.417$, the Huffman coded image achieves $\frac{4.27}{4.417} \times 100$ or 96.67% of the maximum compression possible through the removal of coding redundancy alone.

(**e**) Construct a difference image by replicating the first column of the original image and using the arithmetic difference between adjacent columns for the remaining elements. The difference image is

$$\begin{array}{cccccccc} 21 & 0 & 0 & 74 & 74 & 74 & 0 & 0 \\ 21 & 0 & 0 & 74 & 74 & 74 & 0 & 0 \\ 21 & 0 & 0 & 74 & 74 & 74 & 0 & 0 \\ 21 & 0 & 0 & 74 & 74 & 74 & 0 & 0 \end{array}$$

The probabilities of its various elements are given in Table 8.9-3.

Table P8.9-3

Intensity difference	Count	Probability
21	4	1/8
0	16	1/2
74	12	3/8

The entropy of the difference image is estimated using Eq. (8.1-7) to be

$$\tilde{H} = -\left[\frac{1}{8}\log_2\frac{1}{8} + \frac{1}{2}\log_2\frac{1}{2} + \frac{3}{8}\log_2\frac{3}{8}\right] = 1.41 \text{ bits/pixel.}$$