prediction

Lossless Compression Algorithms

1. Introduction
2. Basics of Information Theory
3. Run-Length Coding
4. Variable-Length Coding (VLC)
5. Dictionary-Based Coding
6. Arithmetic Coding
7. Lossless Image Compression

1. Introduction

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

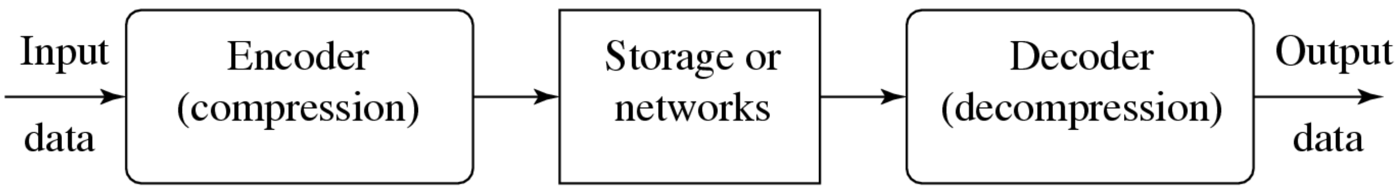


Fig. 7.1: A General Data Compression Scheme.

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

jpeg常见100多

$$compression\ \ ratio = \frac{B_0}{B_1} \tag{7.1}$$

B_0 – number of bits before compression
B_1 – number of bits after compression

- Compression techniques are based on a rich body of literature collectively known as the source coding theory.

- The data sample is treated as a symbol generated by an information source.

- The collection of all possible different symbols is called the alphabet.

- In the source coding theory, entropy and rate-distortion functions are the two most fundamental concepts.

- Entropy provides a measure for information contained in the source data and thereby determines the minimum average bit rate required for perfect reconstruction of the source symbol.

- Rate-distortion function provides a lower bound on the average bit rate for a given distortion in the reconstructed symbols.

2. Basics of Information Theory

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

lower bound guideline

$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \quad (7.2)$$

$$= -\sum_{i=1}^{n} p_i \log_2 p_i \quad (7.3)$$

where p_i is the probability that symbol s_i will occur.

- $\log_2 \frac{1}{p_i}$ indicates the amount of information (self-information as defined by Shannon) contained in s_i, which corresponds to the number of bits needed to encode s_i.

Examples

$$\Sigma\, p_i \log_2 \frac{1}{p_i}$$

$$= \frac{1}{3}\log_2 3 + \frac{2}{3}\log_2 \frac{3}{2}$$

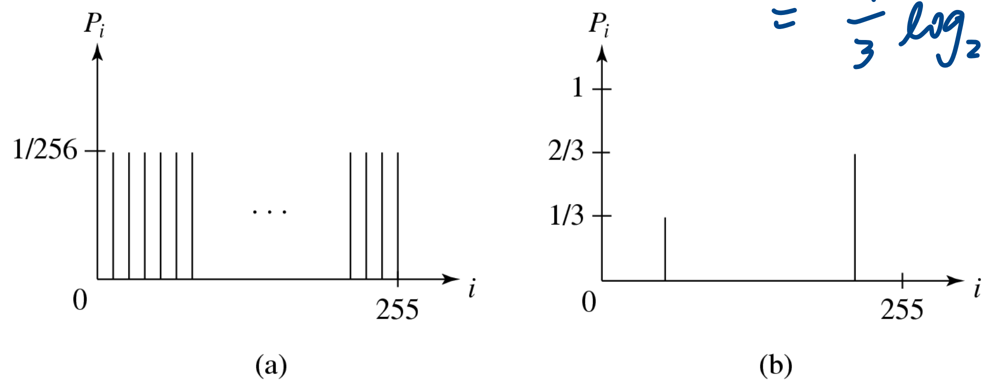


Fig. 7.2 Histograms for two gray-level images.

- 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

$$\log_2 256 = 8 \tag{7.4}$$

- Fig. 7.2(b) shows the histogram of an image with two possible values. Its entropy is

$$\frac{1}{3}\log_2 3 + \frac{2}{3}\log_2 \frac{3}{2} = 0.92$$

Entropy and Code Length

- As can be seen in Eq. (7.3), 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 specifies the lower bound for the average number of bits to code each symbol in S, i.e.,

$$\eta \le \bar{l} \tag{7.5}$$

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

3. Run-Length Coding

- Memoryless Source: an information source that is independently distributed. Namely, the value of the current symbol does not depend on the values of the previously appeared symbols.
- Instead of assuming memoryless source, **Run-Length Coding (RLC) exploits memory** present in the information source.
- Rationale for RLC: if the information source has the property that symbols tend to form continuous groups, then such symbol and the length of the group can be coded.
- Represent the information as a sequence of **(run, run-length)** pairs
- Encode the runs and run-lengths by **FLC** or **VLC**

Definition of Run

-- Dictionary.com --

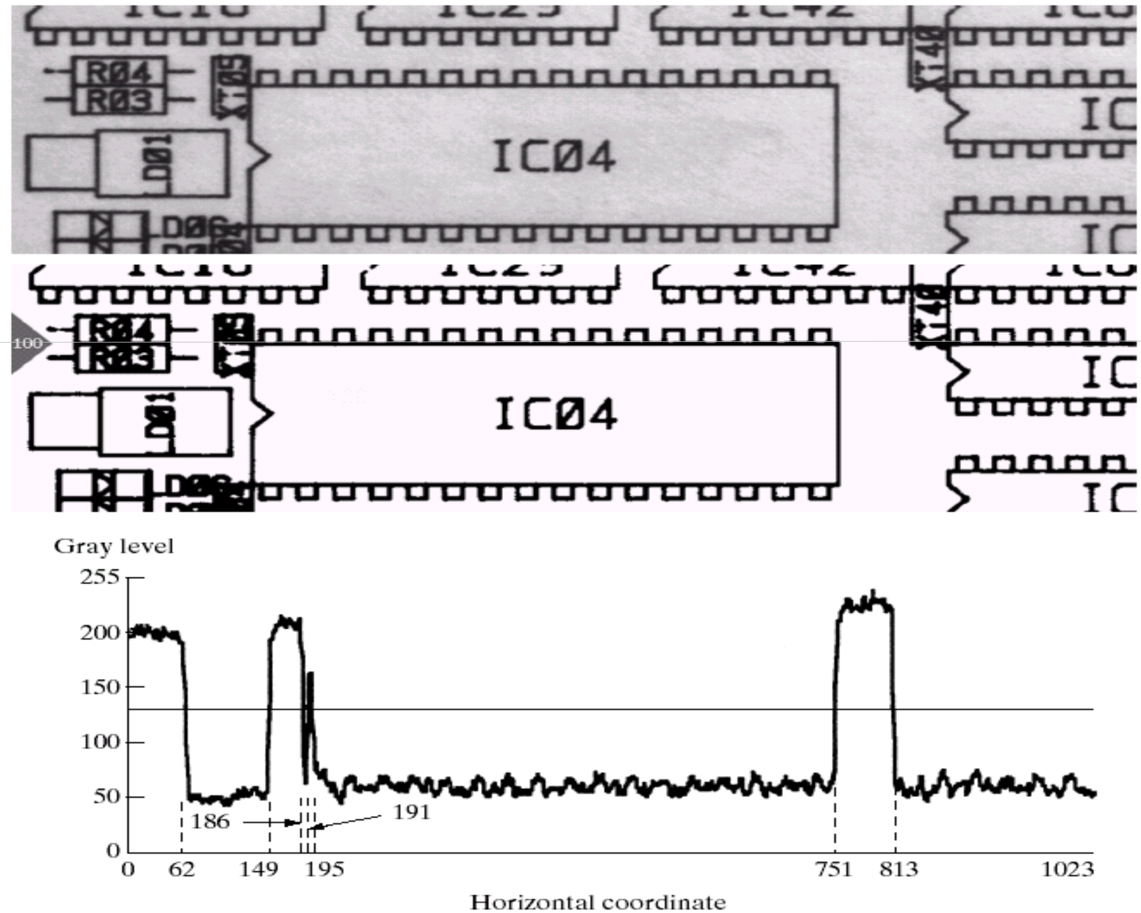
noun

…

121. an uninterrupted course of some state or condition; a spell: a run of good luck; a run of good weather.
122. a continuous extent of something, as a vein of ore.
123. an uninterrupted series or sequence of things, events, etc.: a run of 30 scoreless innings.
124. a sequence of cards in a given suit: a heart run.
125. Cribbage. a sequence of three or more cards in consecutive denominations without regard to suits.

…

Example



a
b
c
d

FIGURE 8.3 Illustration of run-length coding: (a) original image. (b) Binary image with line 100 marked. (c) Line profile and binarization threshold. (d) Run-length code.

Line 100: (1, 63)(0, 87)(1, 37)(0, 5)(1, 4)(0, 556)(1, 62)(0,210) new 1 + 10 = 11 11x8=88

original =

Example of RLC

- Line 100: (1, 63)(0, 87)(1, 37)(0, 5)(1, 4)(0, 556)(1, 62)(0,210)

  88 bits for encoding this line using fixed length code

  - Run: 1 bit
  - Run length: 10 bits (maximum possible length = 1024)
  - 8 runs total
  - 11 x 8 = 88

  Compression ratio

  - 1024/88 = 11.64

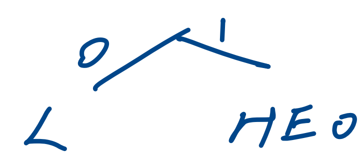
  Additional compression can be obtained by using VLC to encode the run lengths

4. Variable-Length Coding (VLC)

Shannon-Fano Algorithm — a top-down approach

1. Sort the symbols according to the frequency count of their occurrences.

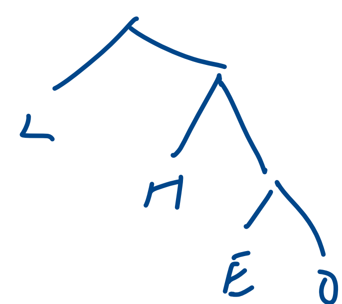
2. Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.



An Example: coding of "HELLO"

Symbol	H	E	L	O
Count	1	1	2	1

Frequency count of the symbols in "HELLO".



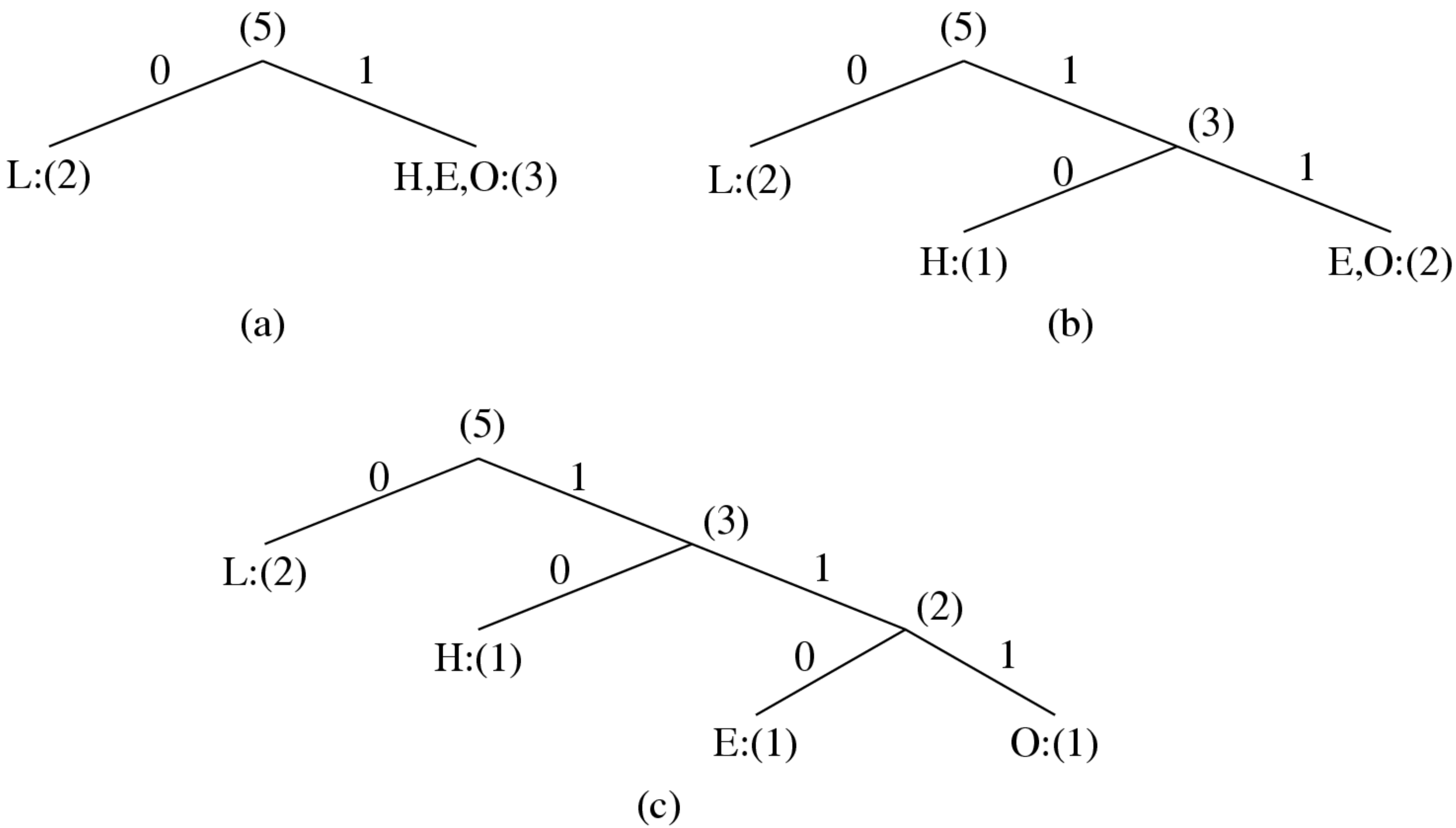


Fig. 7.3: Coding Tree for HELLO by Shannon-Fano.

Table 7.1: Result of Performing Shannon-Fano on HELLO

Symbol	Count	$\text{Log}_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	0	1
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
			TOTAL # of bits:	10

$$\text{Entropy } \eta = p_L \cdot \log_2 \frac{1}{p_L} + p_H \cdot \log_2 \frac{1}{p_H} + p_E \cdot \log_2 \frac{1}{p_E} + p_O \cdot \log_2 \frac{1}{p_O} = 1.92$$

Actual averge number of bits per symbol =10/5=2

$$(0.4 \times 1.32) + (3 \times 0.2 \times 2.32) = 0.528 + 1.392 = 1.92$$

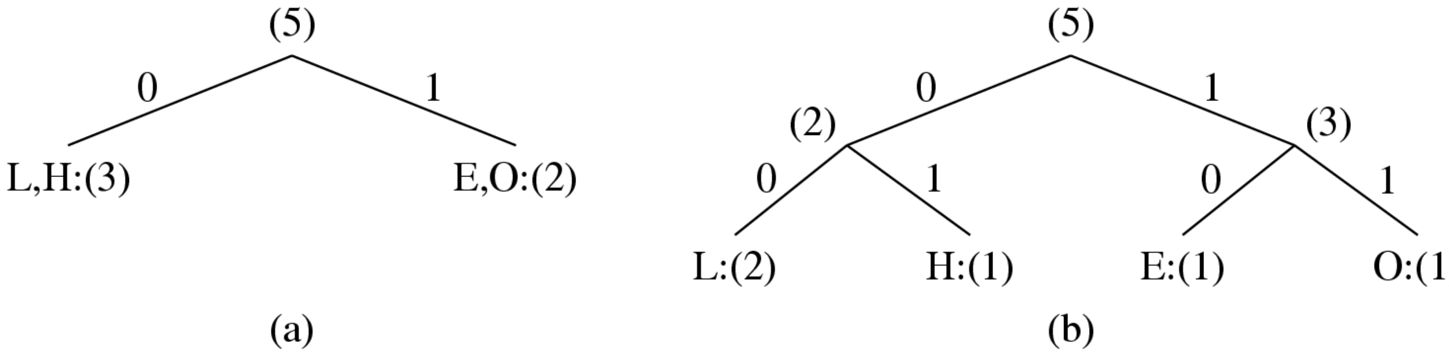


Fig. 7.4 Another coding tree for HELLO by Shannon-Fano.

Table 7.2: Another Result of Performing Shannon-Fano on HELLO (see Fig. 7.4)

Symbol	Count	$\text{Log}_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	00	4
H	1	2.32	01	2
E	1	2.32	10	2
O	1	2.32	11	2
			TOTAL # of bits:	10

Huffman Coding

ALGORITHM 7.1 Huffman Coding Algorithm— a bottom-up approach

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) From the list 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.

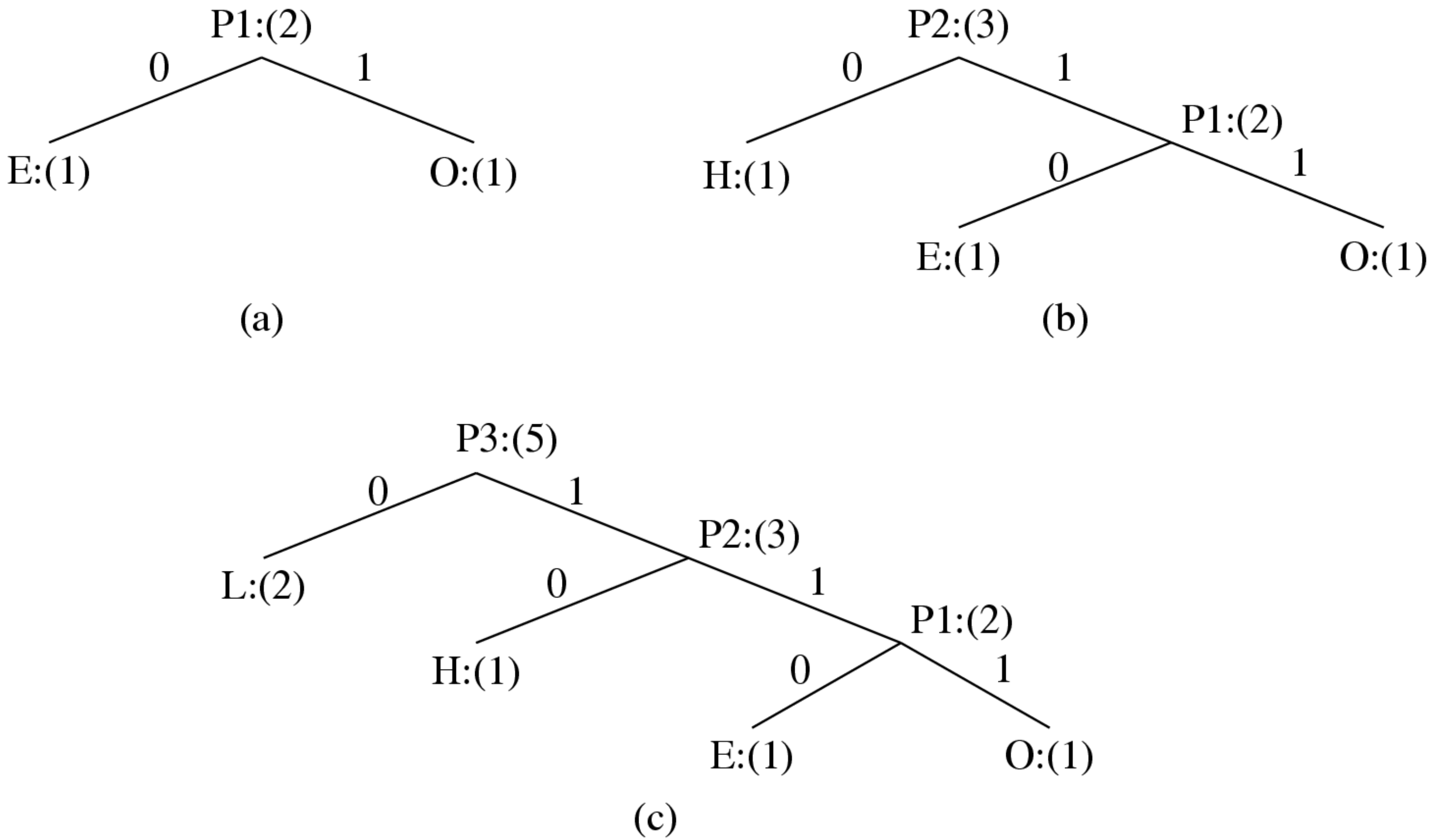


Fig. 7.5: Coding Tree for “HELLO” using the Huffman Algorithm.

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			

Properties of Huffman Coding

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

2. **Optimality**: Huffman code is a *minimum redundancy code.* It 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.

   - The average code length for an information source S is strictly less than η + 1. Combining this with Eq. (7.5), we have:

$$\eta \leq \bar{l} < \eta + 1 \tag{7.6}$$

5. Dictionary-Based Coding

- Another popular data-encoding method that exploits the knowledge of redundancy present in the data.
- This method does not encode single symbols as variable length code. Instead, it encodes several symbols as a single token or word.
- Typically, dictionary-based coding uses fixed-length codewords to represent strings of symbols/characters that commonly occur together, e.g., words in English text.
- The Lempel-Ziv-Welch (LZW) algorithm is a popular dictionary-based coding technique.
- The LZW encoder and decoder build up the same dictionary dynamically while receiving the data. No need to, and hence no overhead for, transmitting the dictionary.
- LZW places longer and longer repeated entries into a dictionary, and then transmits the *code* for an element, rather than the string itself, if the element has already been placed in the dictionary.

6. Arithmetic Coding

- Arithmetic coding is a more modern coding method that usually outperforms Huffman coding.

- Huffman coding assigns each symbol a codeword which has an integral bit length. Arithmetic coding can treat the whole message as one unit, thereby producing fractional bit length per symbol.

- A message is represented by a half-open interval. Initially, the interval is [0, 1). When the message becomes longer, the length of the interval shortens and the number of bits needed to represent the interval increases.

Basic Steps of Arithmetic Coding

- Represent a sequence of symbols by an interval with length equal to its probability
- The interval is specified by its lower boundary (l), upper boundary (u) and length d (=probability)
- The codeword for the sequence is the common bits in binary representations of l and u
- The interval is calculated sequentially starting from the first symbol
- The initial interval is determined by the first symbol
- The next interval is a subinterval of the previous one, determined by the next symbol

$$d_n = d_{n-1} * p_l; \quad l_n = l_{n-1} + d_{n-1} * q_{l-1}; \quad u_n = l_n + d_n.$$

where p_l is the probability of symbol a_l and $q_l = \sum_{k=1}^{l} p_k$ the accumulative probability up to the lth symbol.

CAEE$

ALGORITHM 7.5 Arithmetic Coding Encoder

```
BEGIN
   low = 0.0;      high = 1.0;  range = 1.0;

   while (symbol != terminator)
   {
     get (symbol);
     low = low + range * Range_low(symbol);
     high = low + range * Range_high(symbol);
     range = high - low;
   }
   output a code so that low <= code < high;
END
```

C C 0 1 0.3 0 1 0.5

Example: Encoding in Arithmetic Coding

含0但不含0.2

Symbol	Probability	Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

Fig. 7.8: Arithmetic coding of the symbols “CAEE$”. (a) Probability distribution of symbols. $ is a **special symbol** used to terminate the message

Encode CAEE$

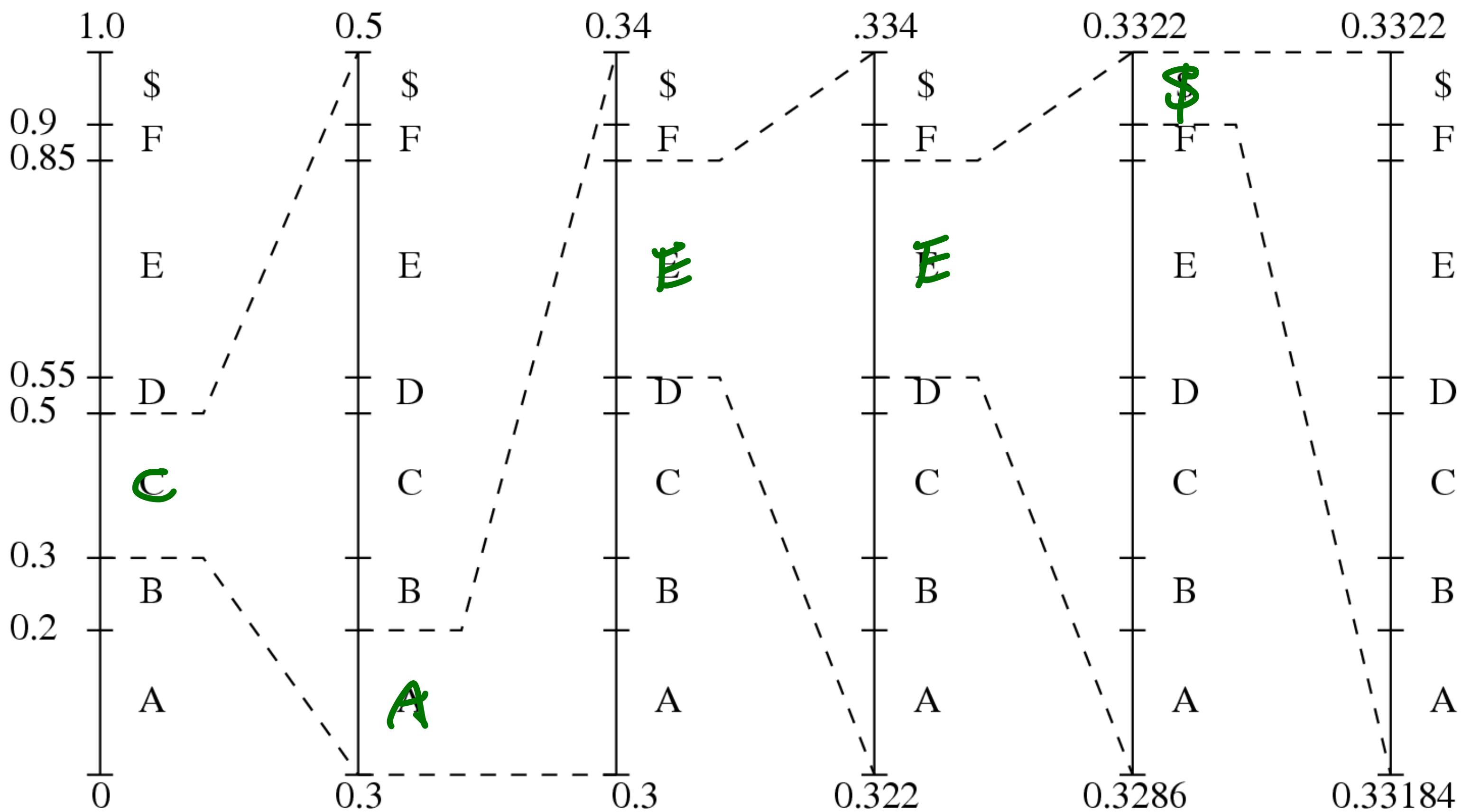


Fig. 7.8(b) Graphical display of shrinking ranges.

Example: Encoding in Arithmetic Coding

Symbol	Low	High	Range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

Fig. 7.8 (c) New *low*, *high*, and *range* generated

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range [*low*, *high*). The following algorithm will ensure that the shortest binary codeword is found.

PROCEDURE 7.2 Generating Codeword for Encoder

```
BEGIN
    code = 0;
    k = 1;
    while (value(code) < low)    /* low is a given number */
    {
       assign 1 to the kth binary fraction bit of the code;
       if (value(code) > high)    /* high is a given number */
          replace the kth bit of the code by 0;
       k = k + 1;
    }
END
```

= 0.01····· = 0.01·····

- For low=0.33184 and high=0.33220, the binary code word generated is 0.01010101=0.33203125

2進位表示

1^{st} fractional bit

最後送出 = 01010101 (前面小數點省略)

ALGORITHM 7.6 Arithmetic Coding Decoder

0.01010101 → 0.33203125

```
BEGIN
    get binary code and convert to decimal value=value(code);
    Do
    {
      find a symbol s so that 找到 C
            Range_low(s) <= value < Range_high(s);
      output s; s=C
      low = Rang_low(s); =0.3
      high = Range_high(s); =0.5
      range = high - low; =0.2
      value = [value - low] / range;
    }         (0.332… − 0.3)/0.2=0.16 ..
    Until symbol s is a terminator;
END
```

再代回去找到A

找下一個 symbol

Table 7.5 Arithmetic coding: decode symbols “CAEE$”

Value	Output Symbol	Low	High	Range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

考試
有可能出一題在AC

Symbol	Probability	Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

7. Lossless Image Compression

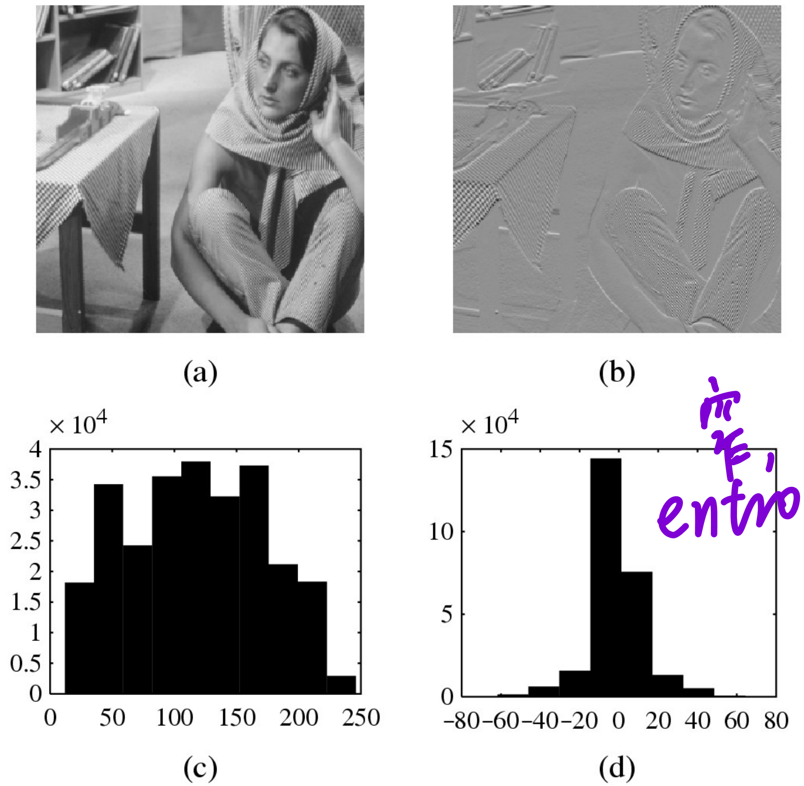
- **Differential Coding** 概念：encode diffence, ∵ diff 的 entropy 比較低 (predict 相鄰 pixel 的 diff 很小)
  - Given an original image $I(x, y)$, using a simple difference operator we can define a difference image $d(x, y)$ as follows:

$$d(x, y) = I(x, y) - I(x - 1, y) \tag{7.9}$$

    or use the discrete version of the 2-D Laplacian operator to define a difference image $d(x, y)$ as

$$d(x, y) = 4\,I(x, y) - I(x, y - 1) - I(x, y + 1) - I(x+1, y) - I(x - 1, y) \tag{7.10}$$

- Due to *spatial redundancy* existing in normal images, the difference image will have a narrower histogram and hence a smaller entropy, as shown in Fig. 7.9.



熵, entropy 低

Fig. 7.9: Distributions for Original versus Derivative Images. (a,b): Original gray-level image and its partial derivative image; (c,d): Histograms for original and derivative images.

(This test image is called “Barb”.)

Lossless JPEG

- **Lossless JPEG**: A special case of the JPEG image compression.

- **A Predictive method**
  1. **Forming a differential prediction**: A predictor combines the values of up to three neighboring pixels as the predicted value for the current pixel, indicated by ‘X’ in Fig. 7.10. The predictor can use any one of the seven schemes listed in Table 7.6.

  2. **Encoding**: The encoder compares the prediction with the actual pixel value at the position ‘X’ and encodes the difference using one of the lossless compression techniques we have discussed, e.g., the Huffman coding scheme.

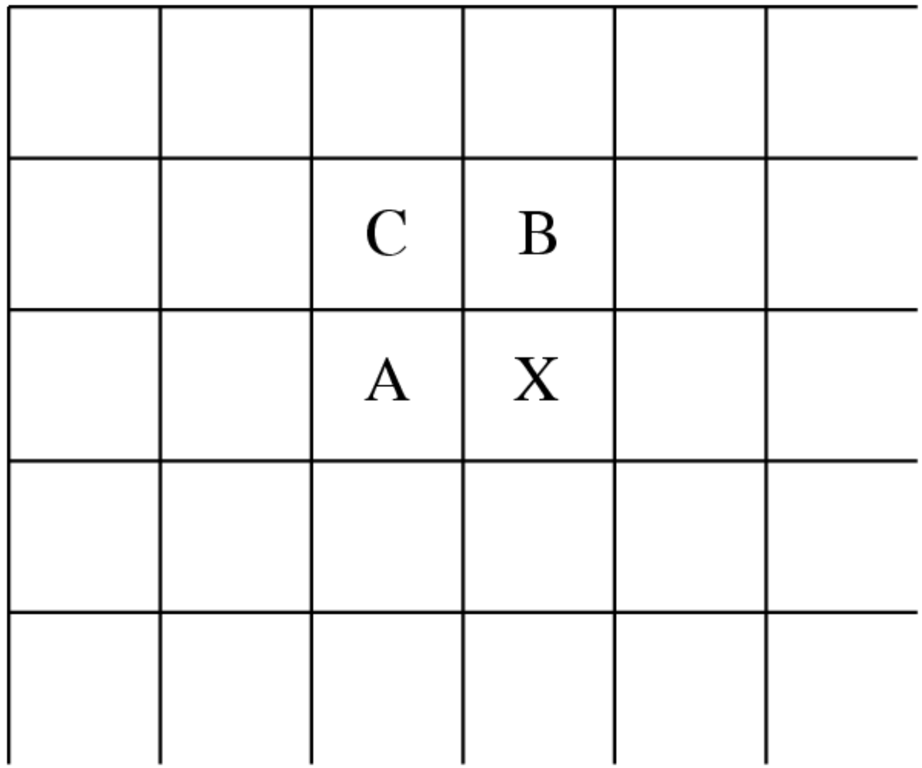


Fig. 7.10: Neighboring Pixels for Predictors in Lossless JPEG.

- **Note**: Any of A, B, or C has already been decoded before it is used in the predictor, on the decoder side of an encode-decode cycle.

Table 7.6: Predictors for Lossless JPEG

Predictor	Prediction
P1	A
P2	B
P3	C
P4	A + B – C
P5	A + (B – C) / 2
P6	B + (A – C) / 2
P7	(A + B) / 2

Table 7.7: Comparison with other lossless compression programs

Compression Program	Compression Ratio			
	Lena	Football	F-18	Flowers
Lossless JPEG	1.45	1.54	2.29	1.26
Optimal Lossless JPEG	1.49	1.67	2.71	1.33
Compress (LZW)	0.86	1.24	2.21	0.87
Gzip (LZ77)	1.08	1.36	3.10	1.05
Gzip -9 (optimal LZ77)	1.08	1.36	3.13	1.05
Pack(Huffman coding)	1.02	1.12	1.19	1.00