Lossless Compression Algorithms

Lecturer：Jun Xiao
（肖俊）
College of Software and Technology

Content

- Introduction and Basics of Information Theory
- Lossless Coding Algorithms
  - Run-Length Coding
  - Variable-Length Coding (VLC)
  - Dictionary-Based Coding
  - Arithmetic Coding
- Lossless Image Compression

1、Introduction and Basics of Information Theory

- Background
- Data Compression Scheme
- Basics of Information Theory

1.1 Background

- More and more data into digital form
  - Libraries, museums, governments
  - To be stored without any loss
- For example -- To encode 120 million call numbers
  - Each item need a 27-bit number, 2^{27} > 120M
  - Compression to reduce the number of bits needed
- Different data appear at different frequencies
  - To assign fewer bits to present more frequently appeared data
  - VCL – variable-length coding
- Lossless coding
  - Both the compression and decompression processes induce no information loss

1.2 Data Compression Scheme

- Definition: Compression Ratio
  - Compression ratio = B0 / B1
    - The number of bits before compression is B0
    - The number of bits after compression is B1
- Compression ratio must be larger than 1.0;
  - The higher the compression ratio, the better the lossless compression scheme

A general data compression scheme

1.3 Basics of Information Theory

- The entropy of an information source
  - With alphabet S = {s1, s2, …, sn}
  - $\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{n} p_i \log_2 p_i$
  - p_i is the probability that symbol s_i in S will occur

- $\log_2 {}^{1}\!/_{P_i}$ indicate the amount of information contained in characters (Self-information)

1.3 Basics of Information Theory

- For example： the probability of ***n*** in a manuscript is 1/32， so
  - The amount of information is 5 bits
  - A character sting nnn require 15 bits to code

- What is entropy?
  - A measure of the disorder of a system
  - The more entropy, the more disorder

1.3 Basics of Information Theory

- Examples:
  - Suppose a system has 4 states outcome, each outcome has probability 1/4 :

$$4\times\frac{1}{4}\times\log_2 1/\frac{1}{4}=2bits$$

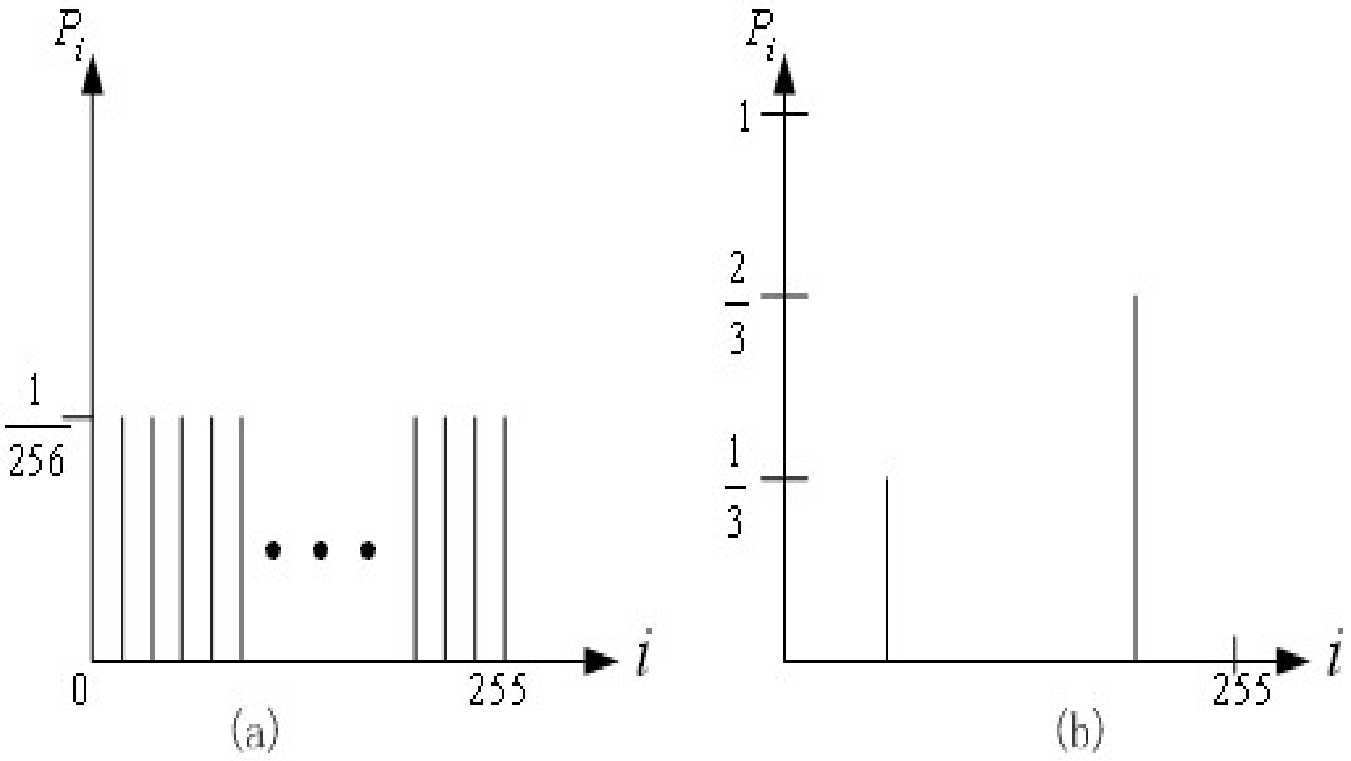
  - If one state had probability ½,the other three had probability 1/6:

$$\frac{1}{2}\times\log_2^2+3\times\frac{1}{6}\times\log_2^6=1.795<2bits$$

  - The most-occurring one means fewer bits to send

- The definition of entropy -- identifying often-occurring symbols as short *codewords*
  - Variable-Length coding

1.3 Basics of Information Theory



Histograms for two gray-level images

1.3 Basics of Information Theory

- The entropy of the two above images
  - The entropy of image a is:
  -
  - $\eta = \sum_{i=0}^{255} \frac{1}{256} \cdot \log_2^{256} = 8$
  - The entropy of image b is:

$$\eta = \frac{1}{3} \cdot \log_2^{\frac{3}{2}} + \frac{2}{3} \cdot \log_2^{\frac{3}{2}}$$

$$= 0.33 \times 1.59 + 0.67 \times 0.59 = 0.92$$

- The entropy is greater when the probability is flat and smaller when it is more peaked.

1.3 Basics of Information Theory

- 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.,

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

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

2、Lossless Coding Algorithms

- Run-Length Coding
- Variable-Length Coding
- Dictionary-Based Coding
- Arithmetic Coding

2.1 Run-Length Coding

- RLC（RUN-LENGTH CODING）
  - One of the simplest forms of data compression
- Basic idea：
  - If symbols of information source tend to form continuous groups, Code one such symbol and the length of the group instead of coding each symbol individually
- Example： a bi-level image can be efficiently coded using RLC
- Two-Dimensional RLC is usually used to code bi-level image.

2.1 Run-Length Coding

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

8x8 gray level image

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

Horizontal RLC

2.1 Run-Length Coding

Input sequence:

0,0,-3,5,1,0,-2,0,0,0,0,2,-4,3,-2,0,0,0,1,0,0,-2

Run-length sequence:

(2,-3)(0,5)(0,1)(1,-2)(4,2)(0,-4)(0,3)(0,-2)(3,1)(2,-2)

- Reduce the number of samples to code
- Implementation is simple

2.2 Variable-Length Coding

- Basic idea
  - Entropy indicates the information content in an information source
- VLC is one of the best-know entropy coding methods
  - Shannon-Fano algorithm
  - Huffman coding
  - Adaptive Huffman coding

2.2 Variable-Length Coding

Shannon-Fano Algorithm

- Developed by Shannon at Bell Labs and Robert Fano at MIT

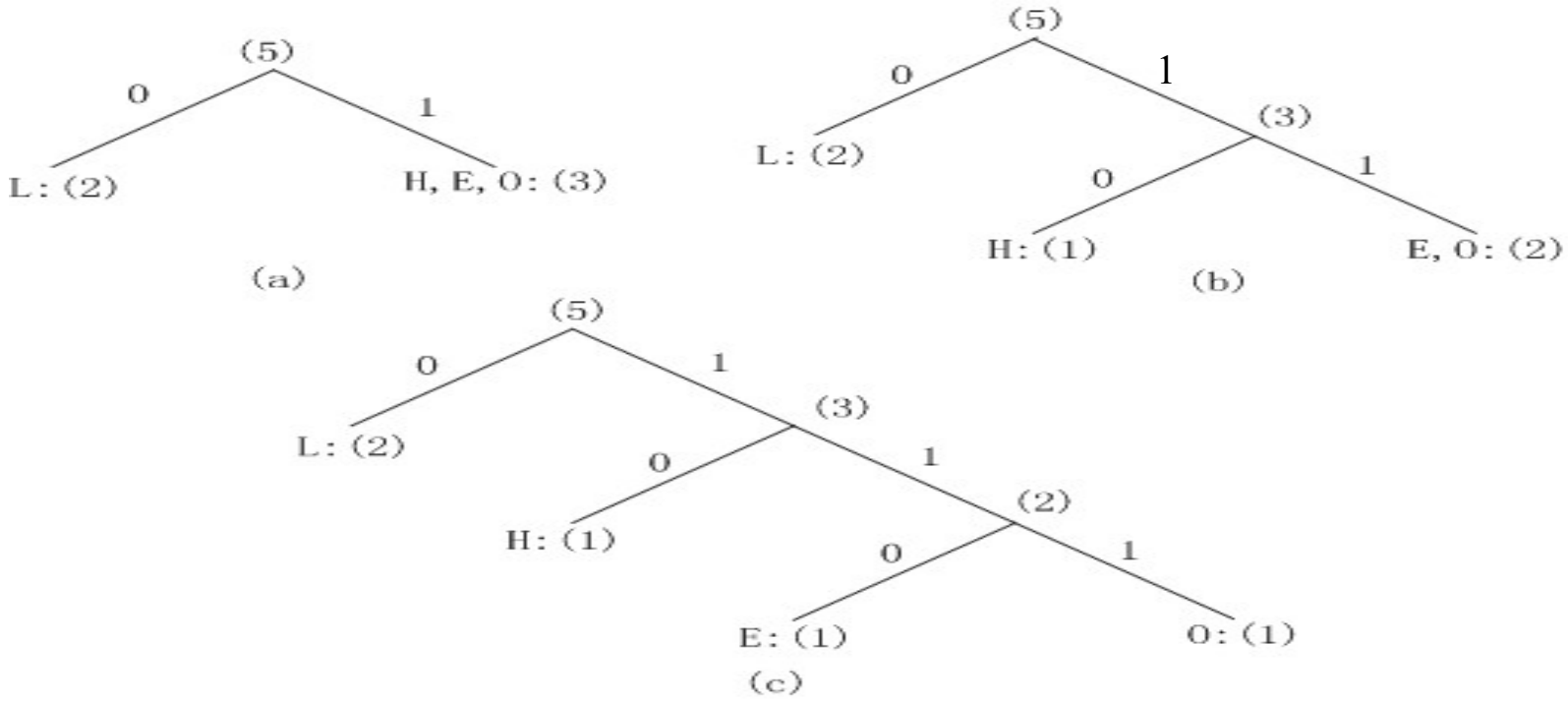
- The Top-Down manner
  - Sort the symbols according to the frequency count of their occurrences
  - Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol

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

2.2 Variable-Length Coding

Example: Hello

Symbol	H	E	L	O
Count	1	1	2	1



2.2 Variable-Length Coding

- The entropy of the example:

$$0.4 \times 1.32 + 0.2 \times 2.32 + 0.2 \times 2.32 + 0.2 \times 2.32$$
$$= 1.92$$

- One result of performing the S-F algorithm on "Hello": average bits 10/5=2

symbol	Count	$\text{Log}_2 P_i^{-1}$	Code	Number 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				

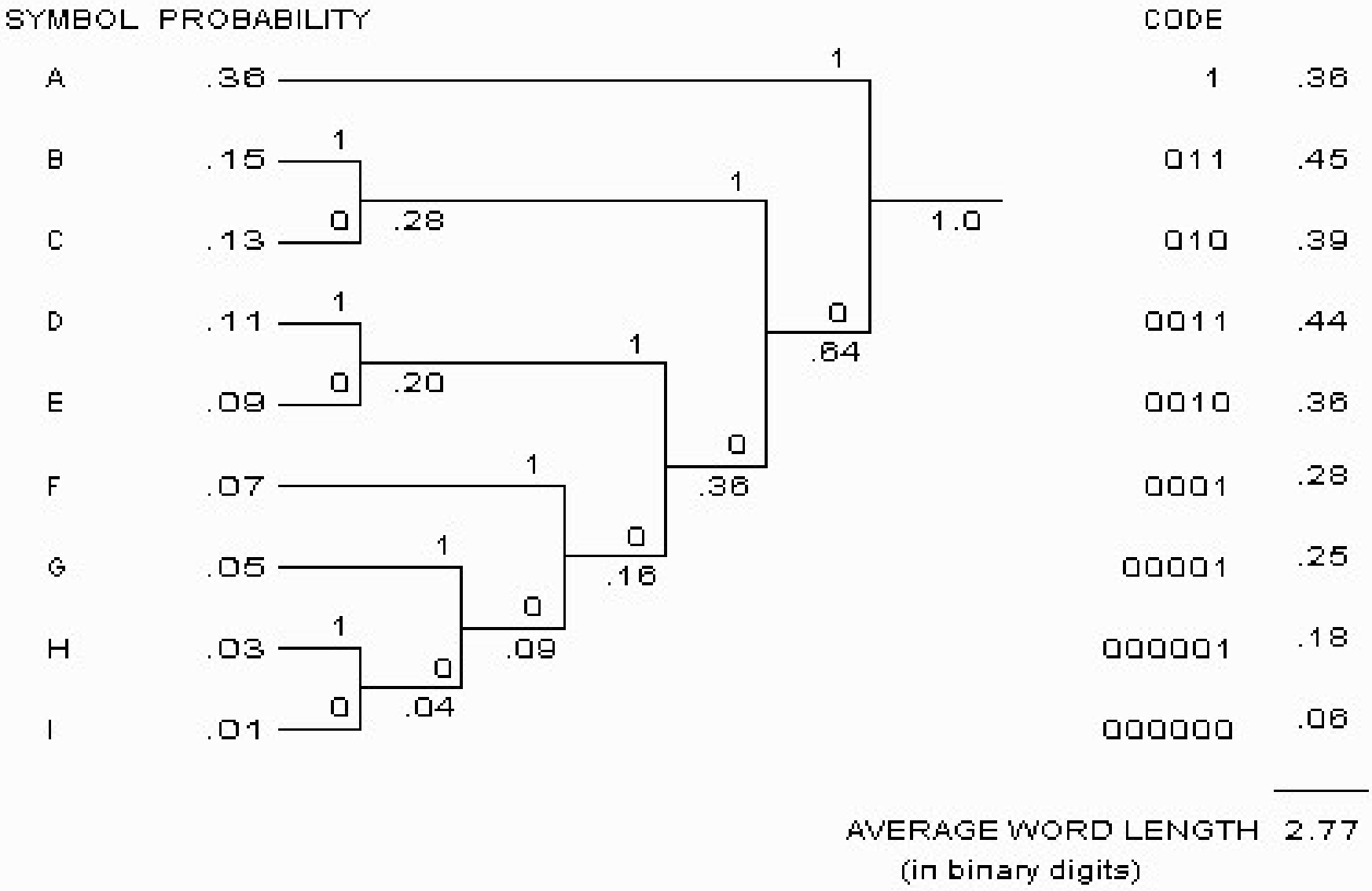
2.2 Variable-Length Coding

Huffman Coding

- First presented by David A. Huffman in 1952
- Adopted in applications, Such as fax、JPEG、MPEG

- A bottom-up manner:
  - Initialization: put all symbols on the list sorted according to their frequency counts
  - Repeat until the list has only one symbol left：
    - From the list, pick two symbols with the lowest frequency counts, form a Huffman sub-tree that has these two symbols as child nodes and create a parent node for them
    - Assign the sum of the children's frequency counts to the parent and insert it into the list, such that the order is maintained.
    - Delete the children from the list
  - Assign a codeword for each leaf based on the path from the root

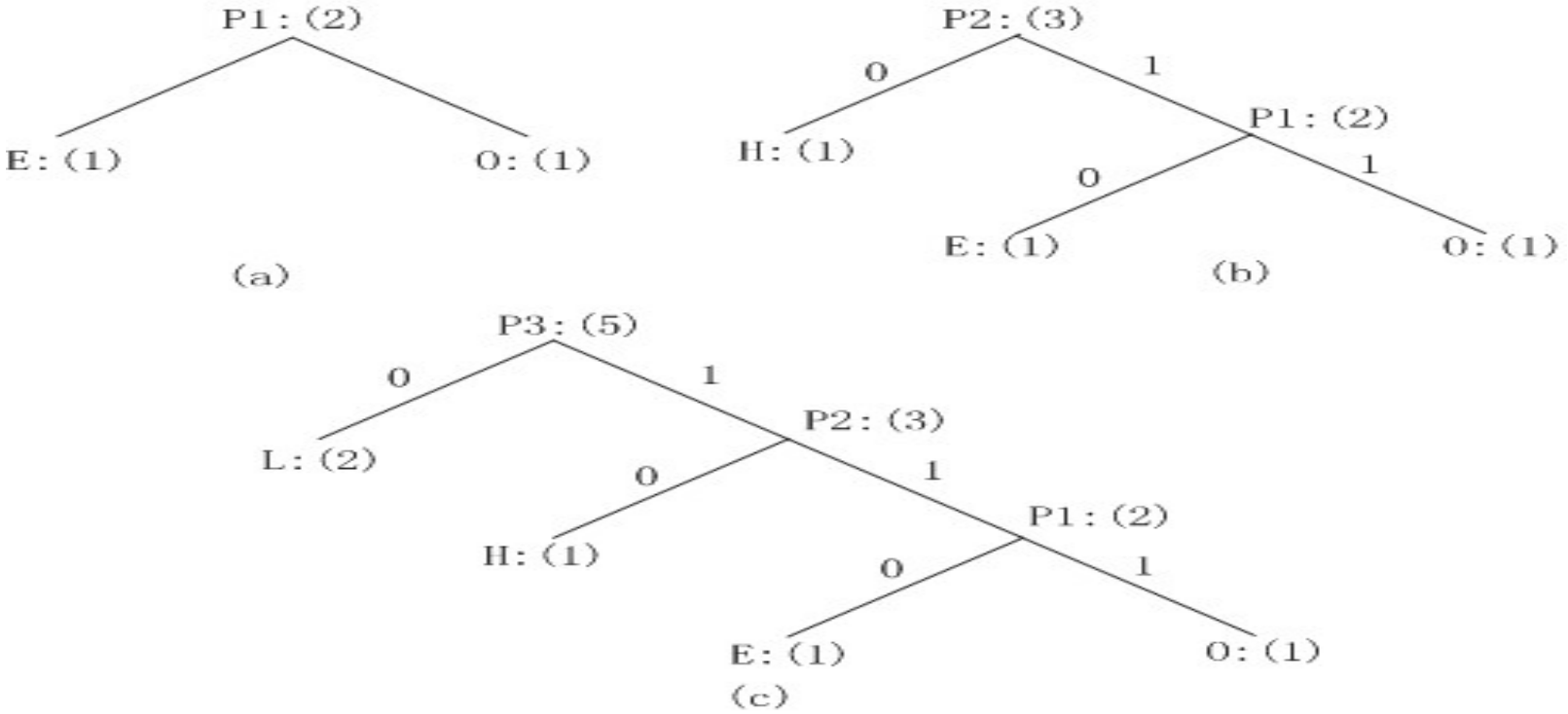
2.2 Variable-Length Coding



2.2 Variable-Length Coding

Example：Huffman Coding “Hello”

Symbol	H	E	L	O
Count	1	1	2	1



2.2 Variable-Length Coding

- For above example, Huffman coding generate the same coding result as Shannon-Fano algorithm

- Another example
  - A:(15), B:(7), C:(6), D:(6) and E:(5)
  - Shannon-Fano needs 89bits；
  - Huffman needs 87 bits

2.2 Variable-Length Coding

Properties of Huffman Coding

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

2. **Optimality**: *minimum redundancy code* - proved 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 $\eta + 1$. Combined with Eq. (7.5), we have:

$$\overline{l} < \eta + 1 \tag{7.6}$$

2.2 Variable-Length Coding

- Extended Huffman Coding
  - Motivation: All codewords in Huffman coding have integer bit lengths. It is wasteful when p_i is very large and hence $\log_2 \frac{1}{p_i}$ is close to 0.
  - Why not group several symbols together and assign a single codeword to the group as a whole?
  - Extended Alphabet: For alphabet S = {s1, s2, . . . , sn}, if k symbols are grouped together, then the extended alphabet is:

$$S^{(k)} = \{\overbrace{s_1 s_1 \ldots s_1}^{k \text{ symbols}}, s_1 s_1 \ldots s_2, \ldots, s_1 s_1 \ldots s_n, s_1 s_1 \ldots s_2 s_1, \ldots, s_n s_n \ldots s_n\}.$$

  — the size of the new alphabet $S(k)$ is n^k.

2.2 Variable-Length Coding

- It can be proven that the average # of bits for each symbol is:

$$\eta \leq \overline{l} < \eta + \frac{1}{k} \qquad (7.7)$$

  An improvement over the original Huffman coding, but not much.

- **Problem**: If k is relatively large (e.g., $k \geq 3$), then for most practical applications where $\mathrm{n} \gg 1$, n^{k} implies a huge symbol table — impractical.

2.2 Variable-Length Coding

Adaptive Huffman Coding

- Huffman Coding requires prior statistical knowledge about the information source, which is often not available
- Even when the statistics are available, the transmission of the symbol table could represent heavy overhead
- Adaptive algorithm, statistics are gathered and updated dynamically as the data-stream arrives
- The probabilities are no longer based on prior knowledge but on actual data received so far

2.2 Variable-Length Coding

- Basic idea of Adaptive Huffman Coding
  - Intial_code: assigns symbols with some initially agreed-upon codes ;
  - Update_Tree: constructing an adaptive Huffman tree
    - Increment the frequency counts for the symbols ;
    - Update the configuration of the tree.
  - The encoder and decoder must use exactly the same intial_code and Update_Tree routines.

2.2 Variable-Length Coding

Adaptive Huffman Coding: statistics are gathered and updated dynamically as the data stream arrives.

```
ENCODER
-------
Initial_code();

while not EOF
 {
       get(c);
       encode(c);
       update_tree(c);
 }
```

```
 DECODER
-------
Initial_code();

while not EOF
{
       decode(c);
       output(c);
       update_tree(c);
}
```

2.2 Variable-Length Coding

Notes on Adaptive Huffman Tree Updating

- Nodes are numbered in order from left to right, bottom to top. The numbers in parentheses indicates the count.
- The tree must always maintain its *sibling* property, i.e., all nodes (internal and leaf) are arranged in the order of increasing counts.

  If the sibling property is about to be violated, a *swap* procedure is invoked to update the tree by rearranging the nodes.
- When a swap is necessary, the farthest node with count N is swapped with the node whose count has just been increased to N +1.

2.2 Variable-Length Coding

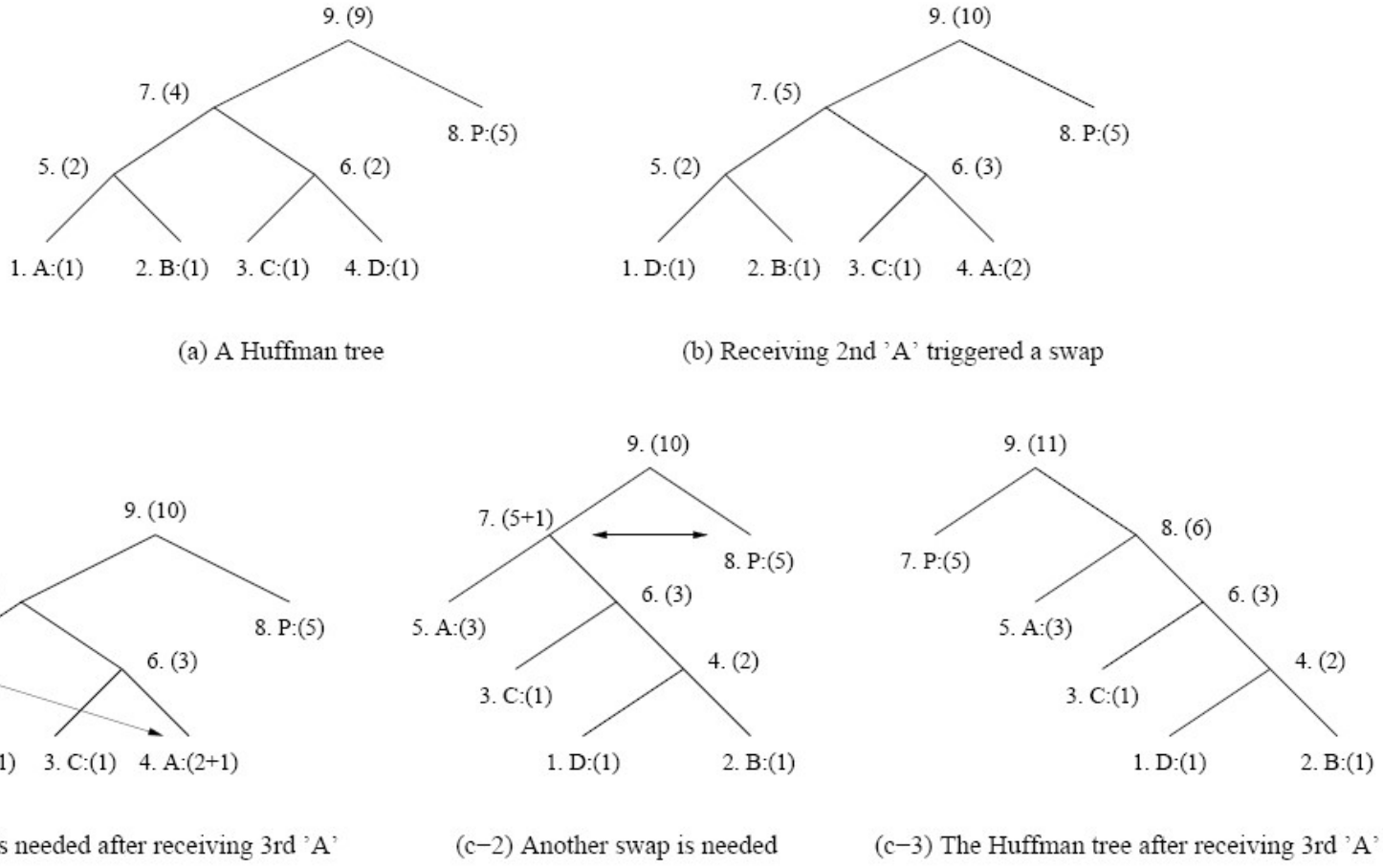
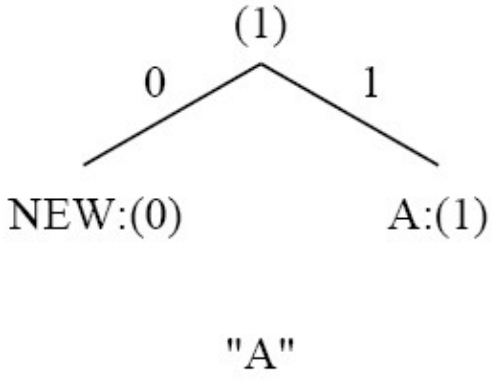


Fig. 7.6: Node Swapping for Updating an Adaptive Huffman Tree

2.2 Variable-Length Coding

Another Example: Adaptive Huffman Coding

- This is to clearly illustrate more implementation details. We show exactly what *bits* are sent, as opposed to simply stating how the tree is updated.
- An additional rule: if any character/symbol is to be sent the first time, it must be preceded by a special symbol, NEW. The initial code for NEW is 0. The *count* for NEW is always kept as 0 (the count is never increased); hence it is always denoted as NEW:(0) in Fig. 7.7.

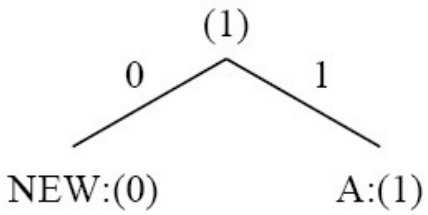


2.2 Variable-Length Coding

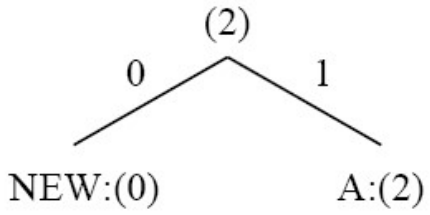
Table 7.3: Initial code assignment for AADCCDD using adaptive Huffman coding.

	Initial Code
NEW:	0
A:	00001
B:	00010
C:	00011
D:	00100
	. .
	. .
	. .

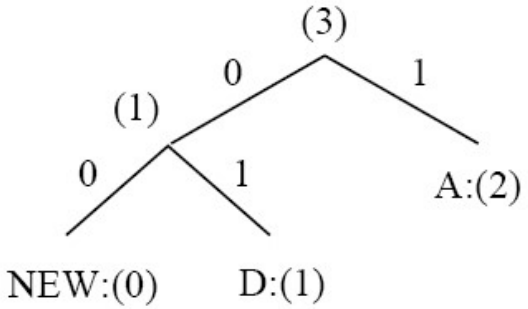
2.2 Variable-Length Coding



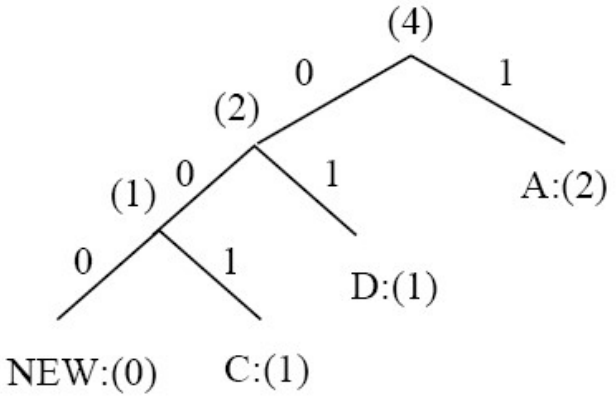
"A"



"AA"

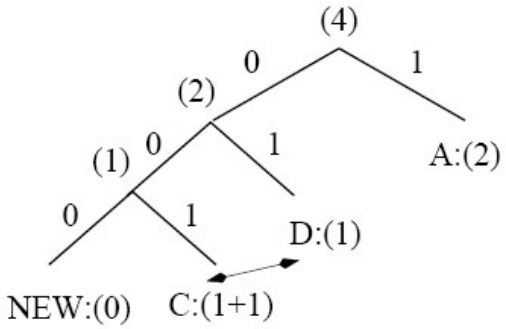


"AAD"

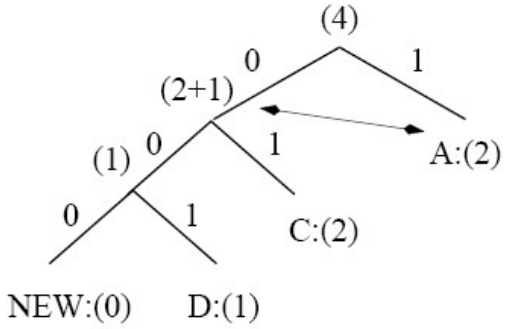


"AADC"

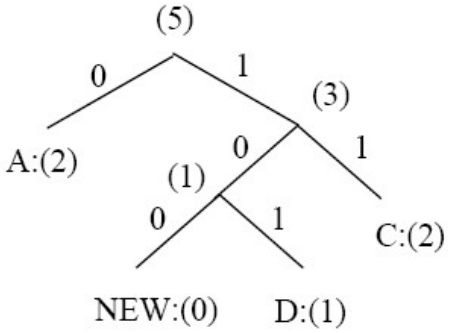
2.2 Variable-Length Coding



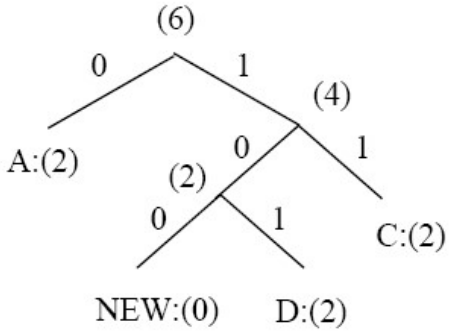
"AADCC" Step 1



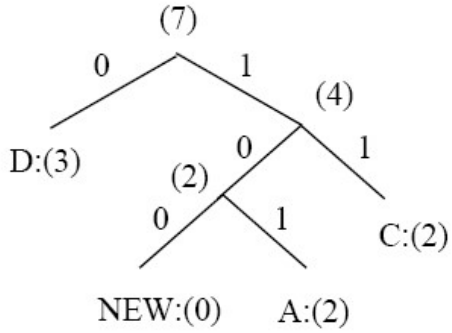
"AADCC" Step 2



"AADCC" Step 3



"AADCCD"



"AADCCDD"

2.2 Variable-Length Coding

Symbol	NEW	A	A	NEW	D	NEW	C	C	D	D
Code	0	00001	1	0	00100	00	00011	001	101	101

- It is important to emphasize that the code for a particular symbol changes during the adaptive Huffman coding process.

For example, after AADCCDD, when the character D overtakes A as the most frequent symbol, its code changes from 101 to 0.

2.3 Dictionary-Based Coding

- First proposed by Ziv and Lempel in 1977 and 1978 respectively
- Terry Welch improved the technique in 1984
- Lempel-Ziv-Welch algorithm (called LZW compression)
- It is used in e.g., UNIX *compress*, GIF, V.42 bis for modems

2.3 Dictionary-Based Coding

- LZW uses fixed-length codewords to represent variable-length strings of symbols/characters that commonly occur together, e.g., words in English text.

- the LZW encoder and decoder build up the same dictionary dynamically while receiving the data.

- LZW places longer and longer repeated entries into a dictionary, and then emits the code for an element, rather than the string itself, if the element has already been placed in the dictionary.

2.3 Dictionary-Based Coding

ALGORITHM 7.2 - LZW Compression

```
BEGIN
   s = next input character;
   while not EOF
   {
        c = next input character;

        if s + c exists in the dictionary
        s = s + c;
        else
        {
        output the code for s;
        add string s + c to the dictionary with a new code;
        s = c;
        }
   }
   output the code for s;
END
```

2.3 Dictionary-Based Coding

Example 7.2 LZW compression for string "ABABBABCABABBA"

- Let's start with a very simple dictionary (also referred to as a "string table"), initially containing only 3 characters, with codes as follows:

Code	String
1	A
2	B
3	C

- Now if the input string is "ABABBABCABABBA", the LZW compression algorithm works as follows:

2.3 Dictionary-Based Coding

S	C	Output	Code	String
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

- The output codes are: 1 2 4 5 2 3 4 6 1. Instead of sending 14 characters, only 9 codes need to be sent (compression ratio = 14/9 = 1.56).

2.3 Dictionary-Based Coding

ALGORITHM 7.3 LZW Decompression (simple version)

```
BEGIN
   s = NIL;
   while not EOF
   {
     k = next input code;
     entry = dictionary entry for k;
     output entry;
     if (s != NIL)
             add string s + entry[0] to dictionary with a new code;
     s = entry;
   }
END
```

Example 7.3: LZW decompression for string "ABABBABCABABBA".

Input codes to the decoder are 1 2 4 5 2 3 4 6 1.

The initial string table is identical to what is used by the encoder.

2.3 Dictionary-Based Coding

The LZW decompression algorithm then works as follows:

S	K	Entry/output	Code	String
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA
ABB	1	A	11	ABBA
A	EOF			

Apparently, the output string is “ABABBABCABABBA”, a truly lossless result!

2.3 Dictionary-Based Coding

ALGORITHM 7.4 LZW Decompression (modified)

```
BEGIN
    s = NIL;
    while not EOF
    {
       k = next input code;
       entry = dictionary entry for k;

       /* exception handler */
       if (entry == NULL)
         entry = s + s[0];

       output entry;
       if (s != NIL)
         add string s + entry[0] to dictionary with a new code;
       s = entry;
    }
END
```

2.3 Dictionary-Based Coding

Example:

Input string:

"^WED^WE

^WEE^WEB

^WET".

w	k	Output	Index	Symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

2.3 Dictionary-Based Coding

Example (continued):

Input string：

"^WED<256>E<260><261><257>B<260>T".

w	k	Output	Index	Symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

2.4 Arithmetic Coding

- Arithmetic coding is a more modern coding method that usually out-performs 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.

- A message is represented by a half-open interval $[a, b)$ where a and b are real numbers between 0 and 1. 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.

2.4 Arithmetic Coding

- Basic idea
  - Instead of present each character as a codeword, Arithmetic Coding represent the whole message by a half-open interval [a,b) contained in [0,1].
  - The length of the interval [a,b) equals the probability of the message. Choose a decimal in [a,b) and transform it into binary form as coding output.
  - Each character can shorten the interval, so the more characters ,the more shorter the interval will be.
  - As the interval become shorter, more bits are needed to present the interval

2.4 Arithmetic Coding

Example: Encoding in Arithmetic Coding

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)

(a) Probability distribution of symbols.

Fig. 7.8: Arithmetic Coding: Encode Symbols “CAEE$”

2.4 Arithmetic Coding

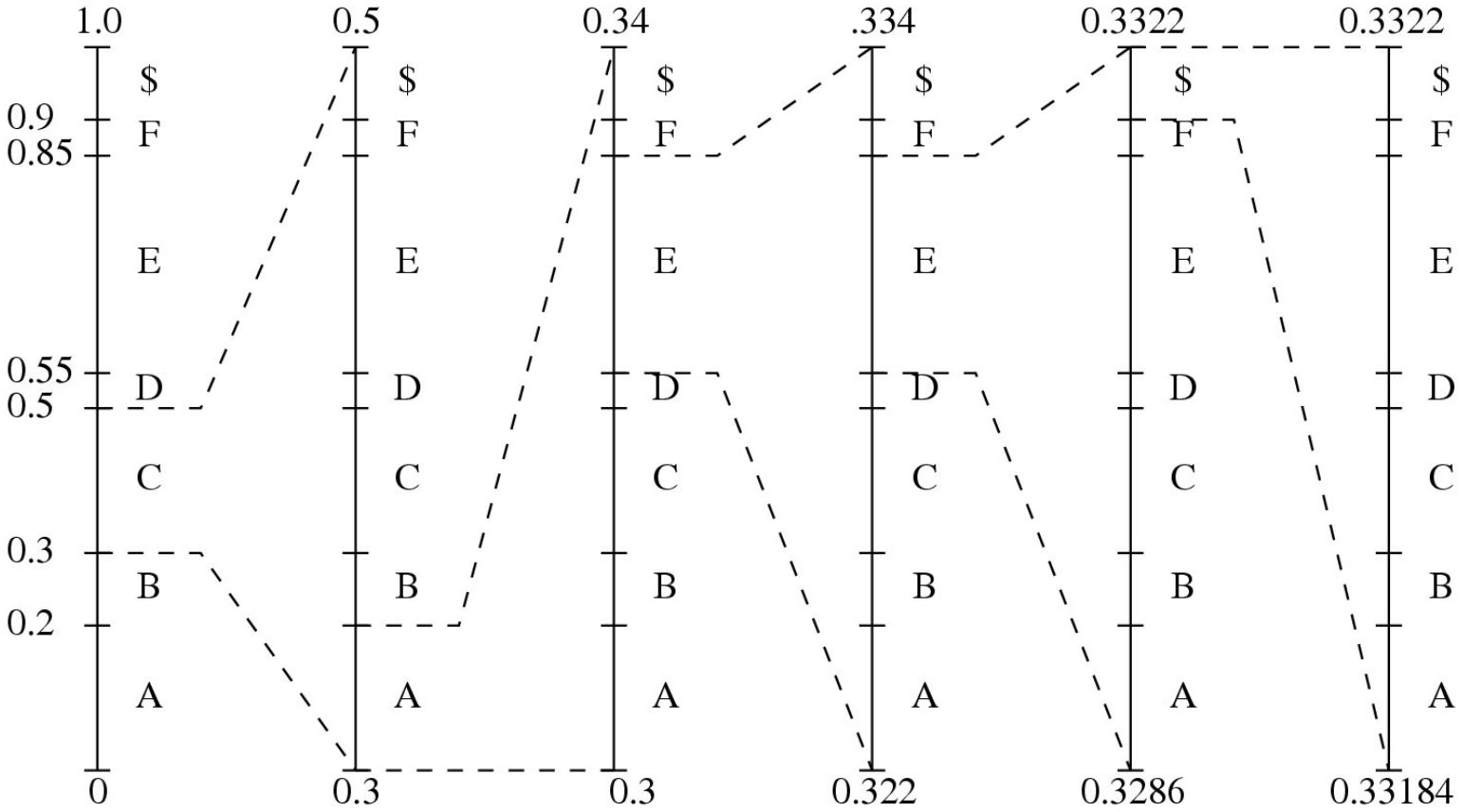


Fig. 7.8(b) Graphical display of shrinking ranges.

2.4 Arithmetic Coding

Example: Encoding in Arithmetic Coding

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)

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

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

Fig. 7.8 (cont'd): Arithmetic Coding: Encode Symbols "CAEE$"

2.4 Arithmetic Coding

PROCEDURE 7.2 Generating Codeword for Encoder

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

For above example, output is: 0.01010101

2.4 Arithmetic Coding

ALGORITHM 7.6 Arithmetic Coding Decoder

```
BEGIN
    get binary code and convert to
    decimal value = value(code);
    Do
    {
      find a symbol s so that
       Range_low(s) <= value < Range_high(s);
      output s;
      low = Rang_low(s);
      high = Range_high(s);
      range = high - low;
      value = [value - low] / range;
    }
    Until symbol s is a terminator
END
```

2.4 Arithmetic Coding

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

3、Lossless Image Compression

- Differential Coding of Images
- Lossless JPEG

3.1 Lossless Image Compression

- Differential coding
  - One of the most commonly used compression techniques in multimedia data compression

- The basic of data reduction in differential coding
  - Existing redundancy in consecutive symbols in a data-stream

3.2 Differential Coding of Images

Given an original image I (x, y), defining a difference image d (x, y)

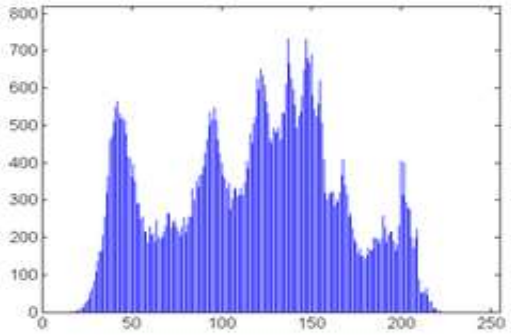
- Using a simple difference operator
  - d(x,y)=I(x,y)-I(x-1,y)
- Discrete 2D Laplacian operator
  - d(x,y) = 4I(x,y)-I(x,y-1)-I(x,y+1)-I(x+1,y)-I(x-1,y)

- Due to *spatial redundancy* existed in normal images I, the difference image d will have a narrower histogram and hence a smaller entropy
  - VLC -- shorter bit-length for the difference image
  - Compression works better on a difference image

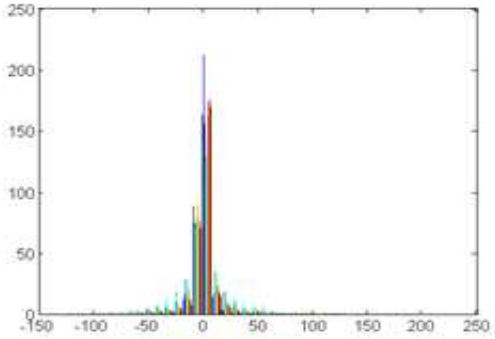
3.2 Differential Coding of Images

(a)

(b)



(c)



(d)

3.3 Lossless JPEG

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

- **The 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.

3.3 Lossless JPEG

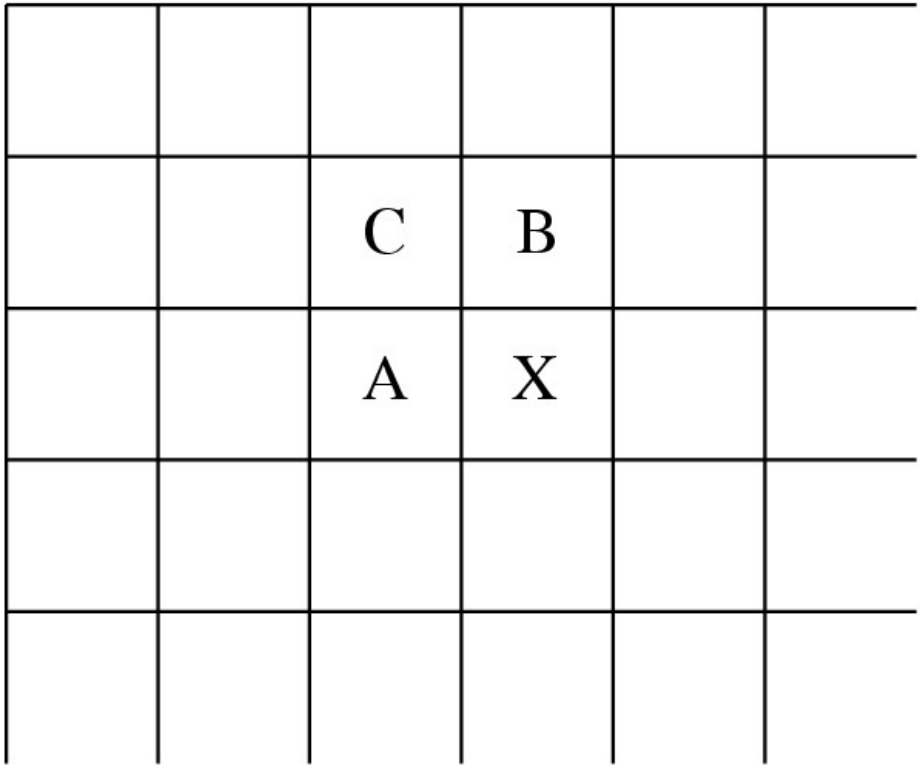


Fig. 7.10: Neighboring Pixels for Predictors in Lossless JPEG.

- **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.

3.3 Lossless JPEG

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

3.3 Lossless JPEG

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

The End

Thanks!

Email: junx@cs.zju.edu.cn