Multimedia System

Lecture 8: Overview of compression

Dr. Reham Reda Mostafa

Mansoura University
Faculty of Computers and Information
Dept. of Information System

dr.reham2206@yahoo.com

26. November 2019

Overview of compression

- The amount of digital media data that is produced in the form of text, video, audio, 3D graphics, and combinations of these media types is extraordinarily large, and the rate of creation increases every day.
- This growing mass of data needs to be stored, accessed, and delivered to a multitude of clients over digital networks, which have varying bandwidths.
  - Fiber-optic networks can provide a bandwidth upwards of 1Gbits/second
  - Cell phone networks, are limited to a range of 20–300 kbps.
  - The third generation (3G) cell phone standard provides transmission bandwidths up to 5 Mbps
- The existence of voluminous data, from creation to storage and delivery, motivates the need for compression.

Outlines

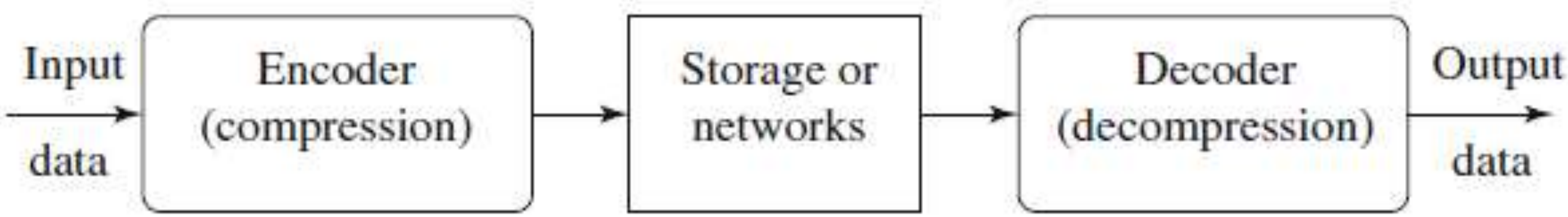
- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

The need for compression

- The need to compress media data is motivated by both storage requirements and transmission requirements.
- In earlier days of the information age, disk space was limited and expensive, and bandwidth was limited, so compression was a must.
- Today, inexpensive storage in a variety of forms is readily available, and high-bandwidth connections (DSL, fiber optics and T-1) are becoming more popular.
- However, the amount and type of information we consume has also increased many fold, with the use of images, audio, video, and graphical animations.

The need for compression (cont'd)

- **Compression:** the process of coding that will effectively reduce the total number of bits needed to represent certain information.
- Following Figure depicts a general data compression scheme, in which compression is performed by an encoder and decompression is performed by a decoder.



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

— The intermediate medium could either be data storage or a communication/computer network.

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

The need for compression (cont'd)

- Compression of media is a necessity to make many applications work with respect to storage and networked delivery.
- To compress multimedia data, we look at data as containing *information*.
- Compression, then, can be considered as a science that reduces the amount of data used to convey the same information.
- It relies on the fact that information, such as an image or video, is not a random collection of pixels but exhibits order and patterns. If this coherence can be understood or even modeled, the information can often be represented and stored/transmitted with a lesser amount of data.

Outlines

- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

BASICS OF INFORMATION THEORY

- When transmitting information from a source to a destination, information theory concerns itself with the *efficiency* and *reliability* of the transmission.
  - ***Efficient transmission***—How efficiently can a source transmit information to a receiver over a given channel and, thereby, communicate the information in the least amount of bits?
  - ***Reliable transmission***—How reliably can a source transmit information to a receiver over a given channel, knowing that the channel is noisy and can corrupt the information during transmission.
- The first point relates to compression.

Entropy and Code Length

- An important question to answer is “How many bits are necessary to represent the information in an image?”
- or alternatively, “What is the minimum amount of data that is sufficient to describe completely an image without loss of information?”
- The information theory provides the mathematical framework to answer theses questions.
- How do we measure the information content of a message/image?

Entropy and Code Length (cont'd)

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

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

  - p_i is the probability that symbol s_i will occur 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.

- 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 specifies the lower bound for the average number of bits to code each symbol in S, i.e.,

$$\eta \leq \ell$$

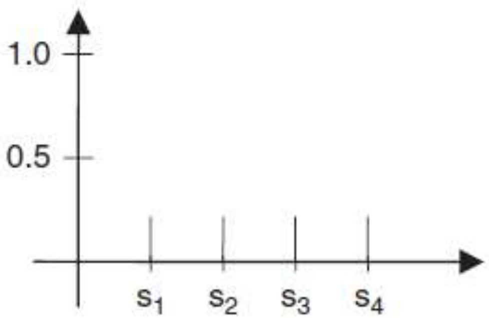
ℓ - the average length (measured in bits) of the codewords produced by the encoder.

Entropy and Code Length (cont’d)

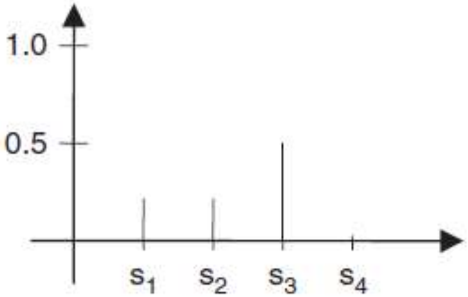
- It is impossible to compress the data such that the average number of bits per symbol is less than the Shannon entropy of the source (in noiseless channel)
- The entropy depends only on the probabilities of symbols and, hence, on the source.
- It is highest when the source produces symbols that are equally probable (all p_i have the same value) and reduces as some symbols become more likely to appear than the other.

Entropy and Code Length

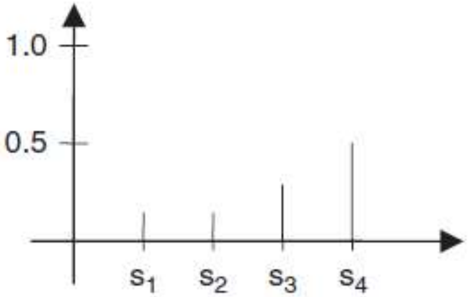
$$P_i = \{0.25,\ 0.25,\ 0.25,\ 0.25\}$$
$$H = -(4 \times 0.25 \times \log_2 0.25)$$
$$H = 2$$



$$P_i = \{0.25\ \ 0.25\ \ 0.5\ \ 0.0\}$$
$$H = -(2 \times 0.25 \times \log_2 0.25 + 0.5 \times \log_2 0.5)$$
$$H = 1.5$$



$$P_i = \{0.125\ \ 0.125\ \ 0.25\ \ 0.5\}$$
$$H = -(2 \times 0.125 \times \log_2 0.125 + 0.25 \times \log_2 0.25 + 0.5 \times \log_2 0.5)$$
$$H = 1.75$$



Entropy and Code Length (cont’d)

- Consider the two messages shown in Figure. Each message is a black-and-white image made up of symbols (s_1 , s_2), which are either black or white.
- The first message is a completely black image. Here, P_1 is 1 and P_2 is 0.
  - — H in this case evaluates to zero, suggesting that there is no information in the image.
- The second picture has equal distribution of black and white pixels. Here, P_1 and P_2 are both roughly 0.5.
  - — H in this case evaluates to 1, which is exactly the minimum number of bits needed to represent the two symbols ($s_1 = 0, s_2 = 1$).

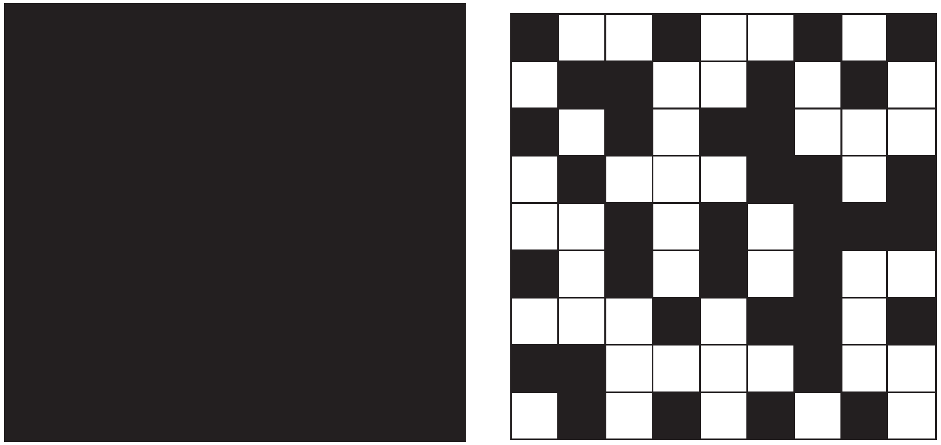


Figure 6-4 Two images are shown.

Efficiency

- Given an alphabet of symbols, the goal of any coding system is to transmit the information contained in a string composed of these symbols in the most efficient manner, that is, in the least number of bits.
- The coding system needs to evaluate the frequency of occurrence of each symbol in the string and then decide how many bits to code each symbol so that the overall number of bits used for that string is the least.
- Optimal average symbol length is given by the entropy.
- Thus, any coding system's goal is to assign binary codes to each symbol in a manner so as to minimize the average symbol length.
- In such cases, *efficiency* is an important metric to evaluate the coding system's ability of compression and is defined as

$$Efficiency = \frac{Entropy}{Average\ Symbol\ length}$$

Outlines

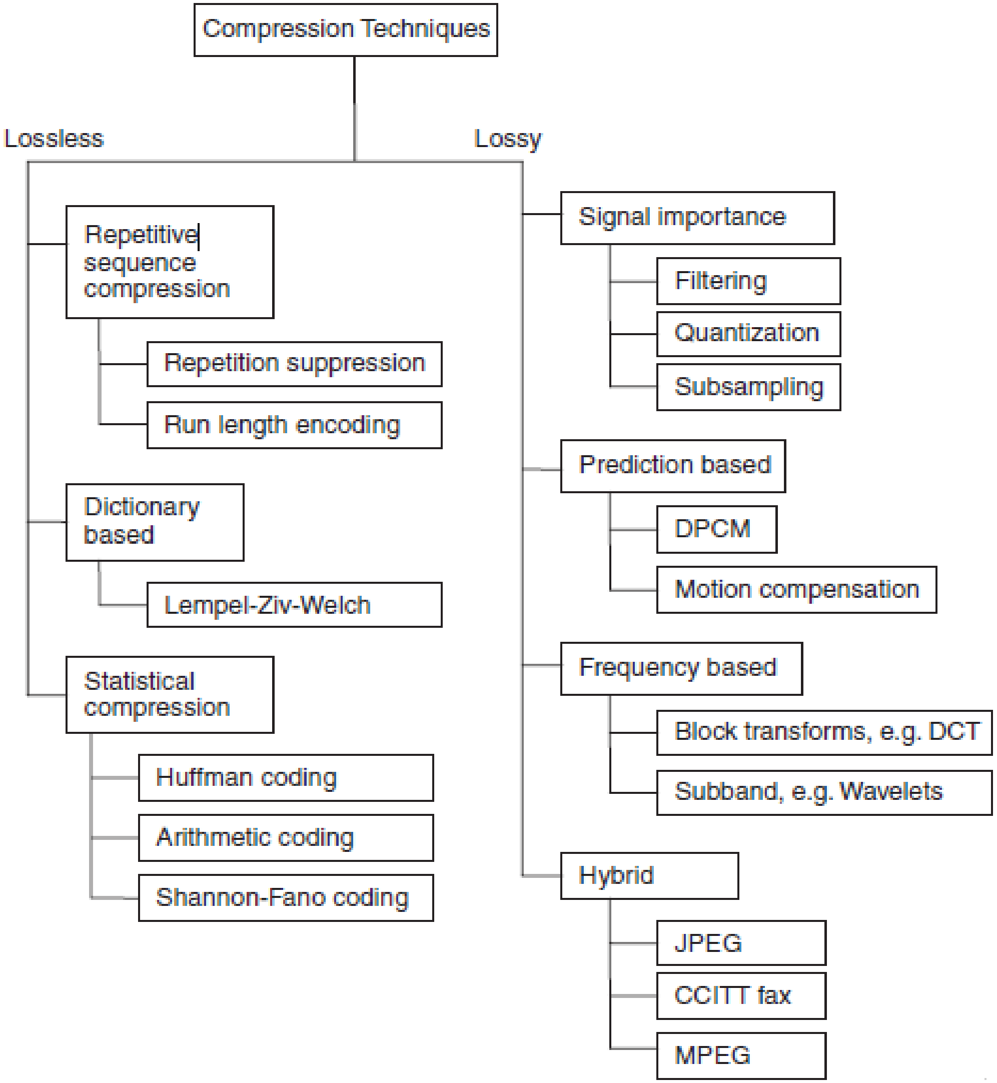
- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

A taxonomy of compression

- A variety of data compression techniques compress multimedia data. They fall into either the lossless or lossy categories.
- **Lossless compression**, results in a compressed signal, which produces the exact original signal when decompressed.
- In **lossy compression**, the compressed signal when decompressed does not correspond to the original signal.
  - Lossy compression produces distortions between the original and the decompressed signal.

A taxonomy of compression (cont'd)

- Figure shows a chart that categorizes lossless and lossy compression techniques.
- Both techniques can be used separately, or in a combined manner for various applications and have resulted in a variety of standards used to compress text, images, video, audio, and so on.



Compression matrices

- Compression ratio is defined as the ratio of the size (or rate) of the original data to the size (or rate) of the compressed data.
- Compression ratio:

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

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

- The relative data redundancy, R can be determined as:

$$R = 1 - \frac{1}{C}$$

- For example, if an audio signal is originally represented as 16 bits per sample and is compressed down to 4 bits per sample, the compression ratio is 4-to-1.

Distortion rate

- In lossy compression, compression corrupts the information; therefore, the reconstructed signal is not the same as the original signal.
- An important metric of such lossy techniques is the amount of distortion caused by compression.
- To measure distortion, a distortion criterion has to be established. These measure the error relative to the signal
  - signal-to-noise ratio (SNR)
  - peak signal-to-noise ratio (PSNR)

signal-to-noise ratio (SNR)

- Signal to noise ratio (SNR) is a measure of the quality of the signal and it is the ratio of the power of the correct signal to the noise.
- The SNR is measured in decibels (**dB**), where 1 dB is a tenth of a **bel**.

$$SNR = 10\ log_{10} \frac{V_{signal}^{2}}{V_{noise}^{2}} = 20\ log_{10} \frac{V_{signal}}{V_{noise}}$$

- **Example:** if the signal voltage $\boldsymbol{V_{signal}}$ is 10 times the noise, then SNR = 20 log (10) = 20 dB.

Outlines

- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

Lossless compression

- Lossless compression techniques achieve compression by removing the redundancy in the signal.
- This is normally achieved by assigning new codes to the symbols based on the frequency of occurrence of the symbols in the message.
- The more frequent symbols are assigned shorter codes and vice versa.
- The commonly used lossless compression processes:
  - Run-length Coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding

Run-length coding

- Run length encoding is the simplest form of redundancy removal.
- It removes redundancy by relying on the fact that a string contains repeated sequences or "runs" of the same symbol.
- The runs of the same symbol are encoded using two entities—a count suggesting the number of repeated symbols and the symbol itself.
- For instance, a run of five symbols *aaaaa* will be replaced by the two symbols 5*a*.
- A sample string of symbols is as follows:

  BBBBEEEEEEEECCCCDAAAAA

- It can be represented as

  4*B*8*E*4*C*1*D*5*A*

- Here, *BBBB* is represented as 4*B*, EEEEEEEE is represented as 8*E*, and so forth.
- The original 22-byte message (assuming a byte for each symbol) has been reduced to 10 bytes, giving a compression ratio of 2.2.

Run-length coding (cont’d)

- Run length encoding performs best in the presence of repetitions or redundancy of symbols in the information.
- Run length encoding is used in a variety of tools involving text, audio, images, and video.
- Run length encoding has also become a part of compression standards—it is supported by most bitmap file formats such as TIFF, BMP, and PCX and a variant of it is even adopted in the JPEG compression pipeline

Run-length coding (cont’d)

BBBBHHDDXXXXKKKKWWZZZZ → 4B2H2D4X4K2W4Z

```
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000111111111111111111110000000000
0000000000100000000000000000010000000000
0000000000100000000000000000010000000000
0000000000100000000000000000010000000000
0000000000111111111111111111110000000000
0000000000000000000000000000000000000000
```

Image of a rectangle

→

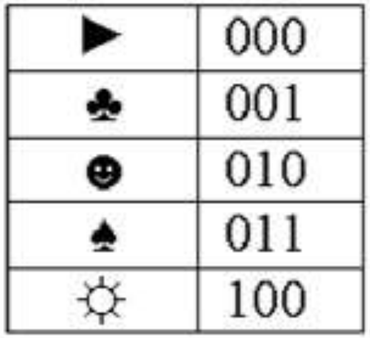
```
0,40
0,40
0,10 1,20 0,10
0,10 1,1 0,18 1,1 0,10
0,10 1,1 0,18 1,1 0,10
0,10 1,1 0,18 1,1 0,10
0,10 1,20 0,10
0,40
```

Outlines

- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

Fixed Length Coding (FLC)

- A simple example
  - The message to code: ►♣♣♠☻►♣☼►☻
- 5 different symbols at least 3 bits
- Message length: 10 symbols
- Codeword table:

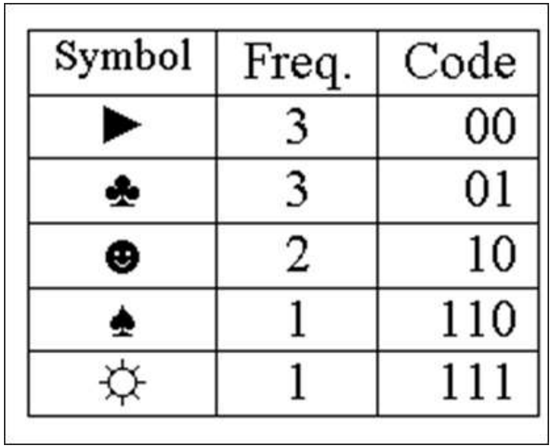


►	000
♣	001
☻	010
♠	011
☼	100

- The Total bits required to code: 10*3 = 30 bits

Variable Length Coding (VLC)

- Intuition: Those symbols that are more frequent should have smaller codes, yet since their length is not the same, there must be a way of distinguishing each code.
- The message to code: ▶♣♣♠☻▶♣☼▶☻
- Codeword table



Symbol	Freq.	Code
▶	3	00
♣	3	01
☻	2	10
♠	1	110
☼	1	111

- Total bits required to code: 3*2 +3*2+2*2+3*1+3*1 = 24 bits
- How to find the optimal codeword table?

Variable Length Coding (VLC)

- Assume that a discrete random variable r_k in the interval $[0, L-1]$ is used to represent the intensities of an $M \times N$ image and that probability of occurrence of each r_k is $p_r(r_k)$. Then

$$p_r(r_k) = n_k/n$$

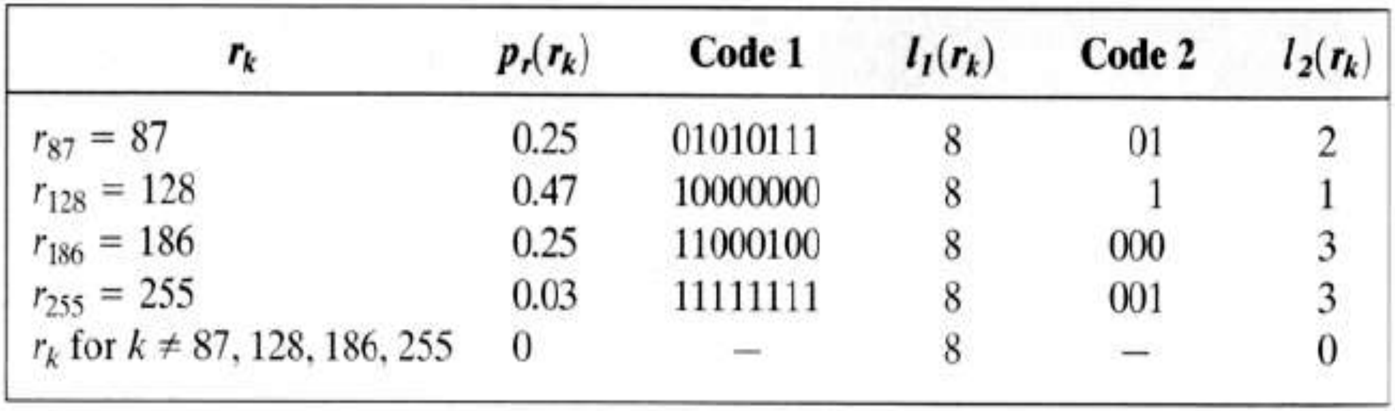
- Here L is the number of intensity values and n_k is the number of times that the k^{th} intensity appears in the image. If **the number of bits needed to represent each value of $\boldsymbol{r_k}$ is $\boldsymbol{l(r_k)}$**, the average number of bits required to represent each pixel is

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

- The total number of bits required to represent an MxN image is MNL_{avg}.

Variable Length Coding (VLC)

Example 1: Image [256 × 256] have the intensity values, 87, 128, 186, and 255. Each intensity have the following probability value:



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

- The “start” image has the intensity distributions shown above. “code 1” – natural 8-bit code: $l_1(r_k) = 8$, bits for all r_k.
- On the other hand, if the “code 2” is used, the average number of bits of the encoded pixels is

$$L_{avg} = 0.25 * 2 + 0.47 * 1 + 0.25 * 3 + 0.03 * 3 = 1.81 \ \ bits$$

Variable Length Coding (VLC)

Example 1: Image [256 × 256] have the intensity values, 87, 128, 186, and 255. Each intensity have the following probability value:

- The total number of bits needed to represent the entire image is

$$MNL_{avg} = 256 * 256 * 1.81 = 118.621 \ \ bits$$

- As compared to the number of bits needed if 8-bit “code1” was used:

$$MNL_{avg} = 256 * 256 * 8 = 524.288 \ \ bits$$

- Therefore, the resulting compression is

$$C = \frac{524.288}{118.621} = \frac{8}{1.81} = 4.42$$

- And the data redundancy is

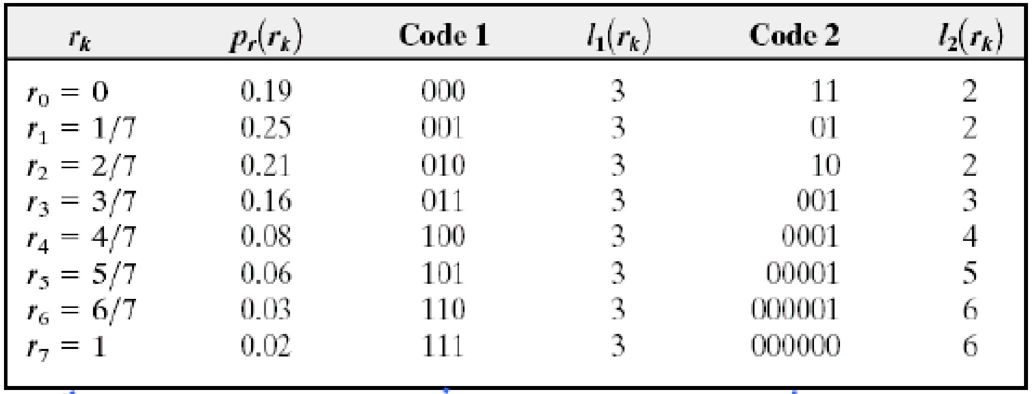
$$R = 1 - \frac{1}{C} = 1 - \frac{1}{4.42} = 0.774$$

suggesting that 77.4% of the data in the original 8-bit image is redundant.

- The compression achieved by “code 2” results from assigning fewer bits to the more probable intensity values and more bits to the less probable ones. The result is a ***variable-length code***.

Variable Length Coding (VLC)

Example 2: An 8 gray level image has the following gray level distribution



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

TABLE 8.1
Example of variable-length coding.

8 gray levels

Fixed 3-bit code

Variable length code

- ➢ The average number of bit used for fixed 3-bit code:

$$L_{avg} = \sum_{k=0}^{7} l_1(r_k) p_r(r_k) = 3 \sum_{k=0}^{7} p_r(r_k) = 3.1 = 3 \ bits$$

- ➢ The average number of bits used for variable-length code in this particular example:

$$L_{avg} = \sum_{k=0}^{7} l_1(r_k) p_r(r_k) = 2(0.19) + 2(0.25) + 2(0.21) +$$

$$3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02)$$

Variable Length Coding (VLC)

Example 2: An 8 gray level image has the following gray level distribution

- The Compression ratio

$$C_R = \frac{3}{2.7} = 1.11$$

- The relative data redundancies:

$$R_D = 1 - \frac{1}{1.11} = 0.099 \Rightarrow \sim \%10$$

- **Data compression is achieved by**
  - The shortest code are assigned to the most frequent (high probability) gray levels
  - The longest code are assigned to the least frequent (low probability) gray levels

Variable Length Coding (VLC)

- Since the entropy indicates the information content in an information source *S*, it leads to a family of coding methods commonly known as *entropy coding* methods.
- As described earlier, *variable-length coding* (VLC) is one of the best-known such methods.
- Here, we will study the Huffman coding, and arithmetic coding.

Huffman Coding

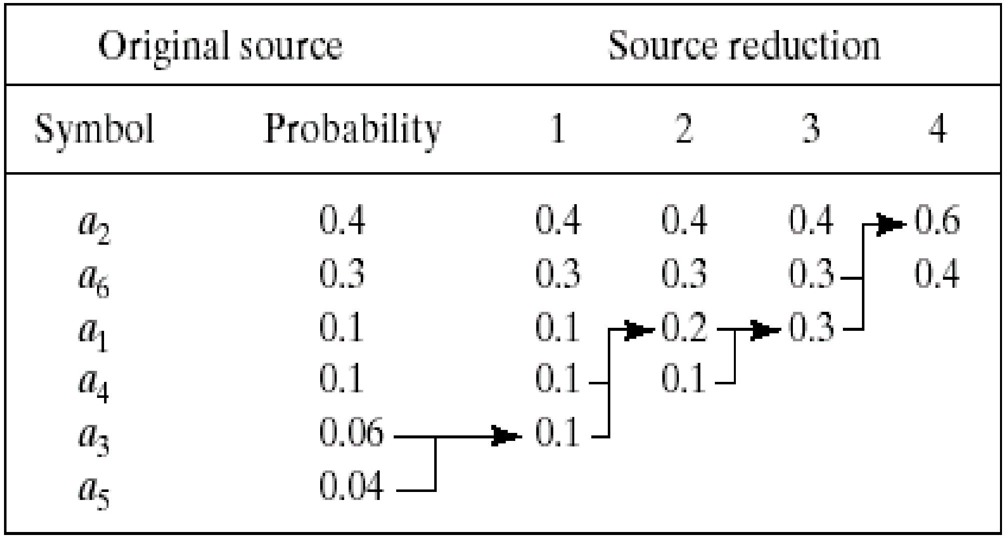
- First presented by Huffman in a 1952 paper, this method has been adopted in many important and/or commercial applications, such as fax machines, JPEG, and MPEG.
- When data is initially sampled, each sample is assigned the same number of bits. This length is $log_2 n$, where n is the number of distinct samples or quantization intervals.
- This is true of audio data (8 bits or 16 bits per sample), images (24 bits per pixel), text ASCII data, and so on.
- Assigning an equal number of bits to all samples might not be the most efficient coding scheme because some symbols occur more commonly than others.
- The goal, as mentioned earlier, is to have more frequent symbols that have smaller code lengths and less frequent symbols that have longer code lengths.
- This variable-length representation is normally computed by performing statistical analysis on the frequency of occurrence of each symbol.
- Huffman coding provides a way to efficiently assign new codes to each symbol, depending on the frequency of occurrence of the symbol.

Huffman Coding (cont'd)

- **Algorithm** (Huffman Coding).
  1. Initialization: put all symbols on the list sorted according to their frequency counts.
  2. Repeat until the list has only one symbol left.
     a) 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 for them.
     b) Assign the sum of the children's frequency counts to the parent and insert it into the list, such that the order is maintained.
     c) Delete the children from the list.
  3. Assign a codeword for each leaf based on the path from the root.

Huffman Coding (cont’d)

- Example 1:
  - The first step is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction.



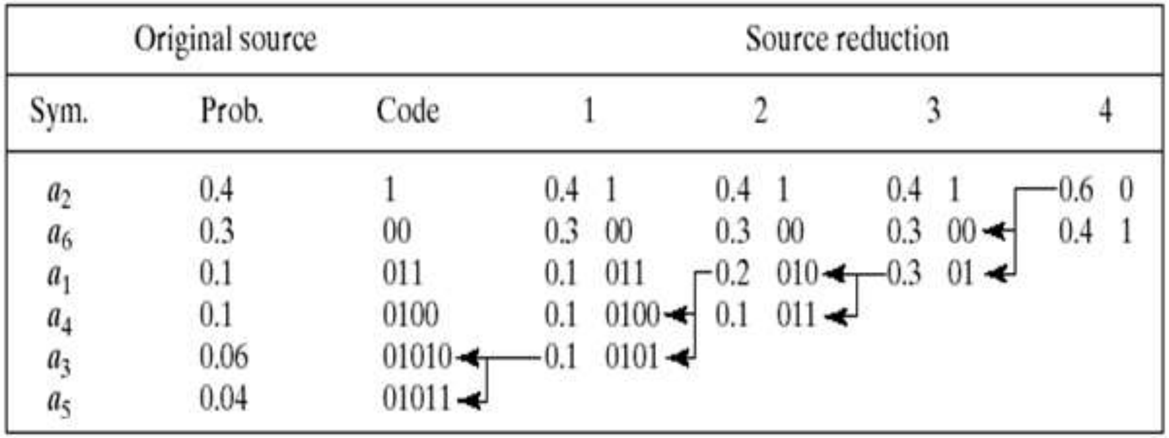
Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

FIGURE 8.11
Huffman source reductions.

Huffman Coding (cont’d)

- Example1 (cont’d):
  - The second step is to code each reduced source, starting with the smallest source and working back to the original source.

FIGURE 8.12 Huffman code assignment procedure.



Original source			Source reduction							
Sym.	Prob.	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								

- The average code length is:

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

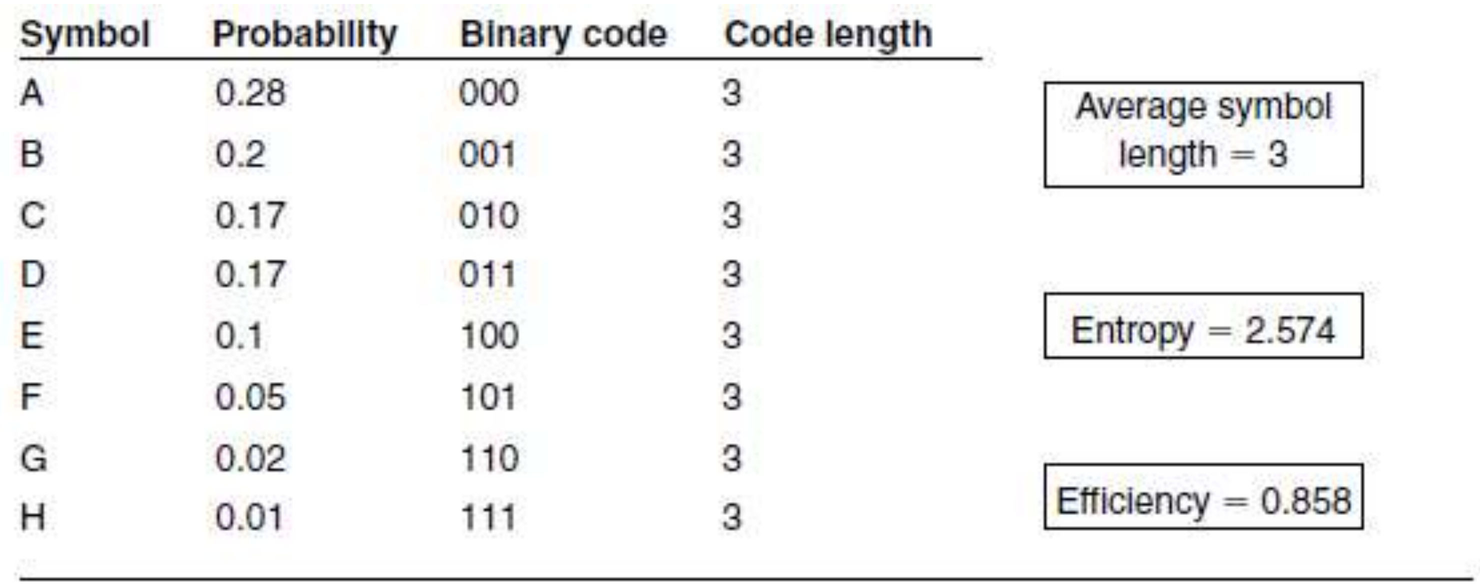
- The entropy of the source is 2.14 bits/symbol.

$$H = -\sum_{j=1}^{J} P(a_j) \log P(a_j) = 2.14 \; bits \, / \, symbol.$$

- The resulting Huffman coding efficiency is %97.3 (2.14/2.2).
- Note that Huffman Coding is not optimal and many more efficient versions of it as well as other variable-length coding methods can be used.

Huffman Coding (cont’d)

- Example 2:



Symbol	Probability	Binary code	Code length
A	0.28	000	3
B	0.2	001	3
C	0.17	010	3
D	0.17	011	3
E	0.1	100	3
F	0.05	101	3
G	0.02	110	3
H	0.01	111	3

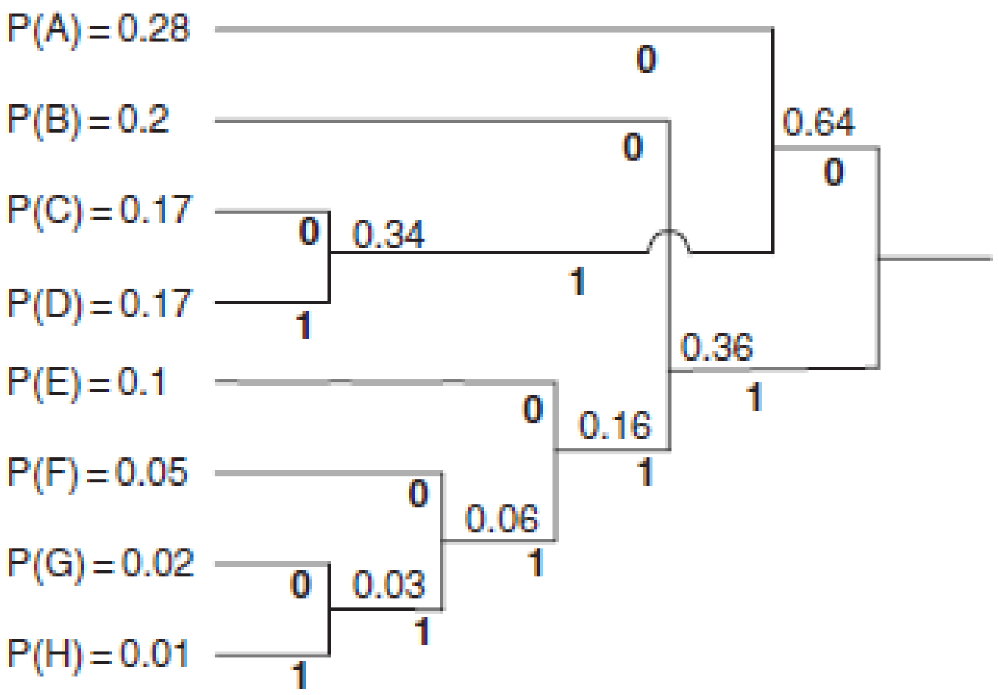
Average symbol length = 3

Entropy = 2.574

Efficiency = 0.858

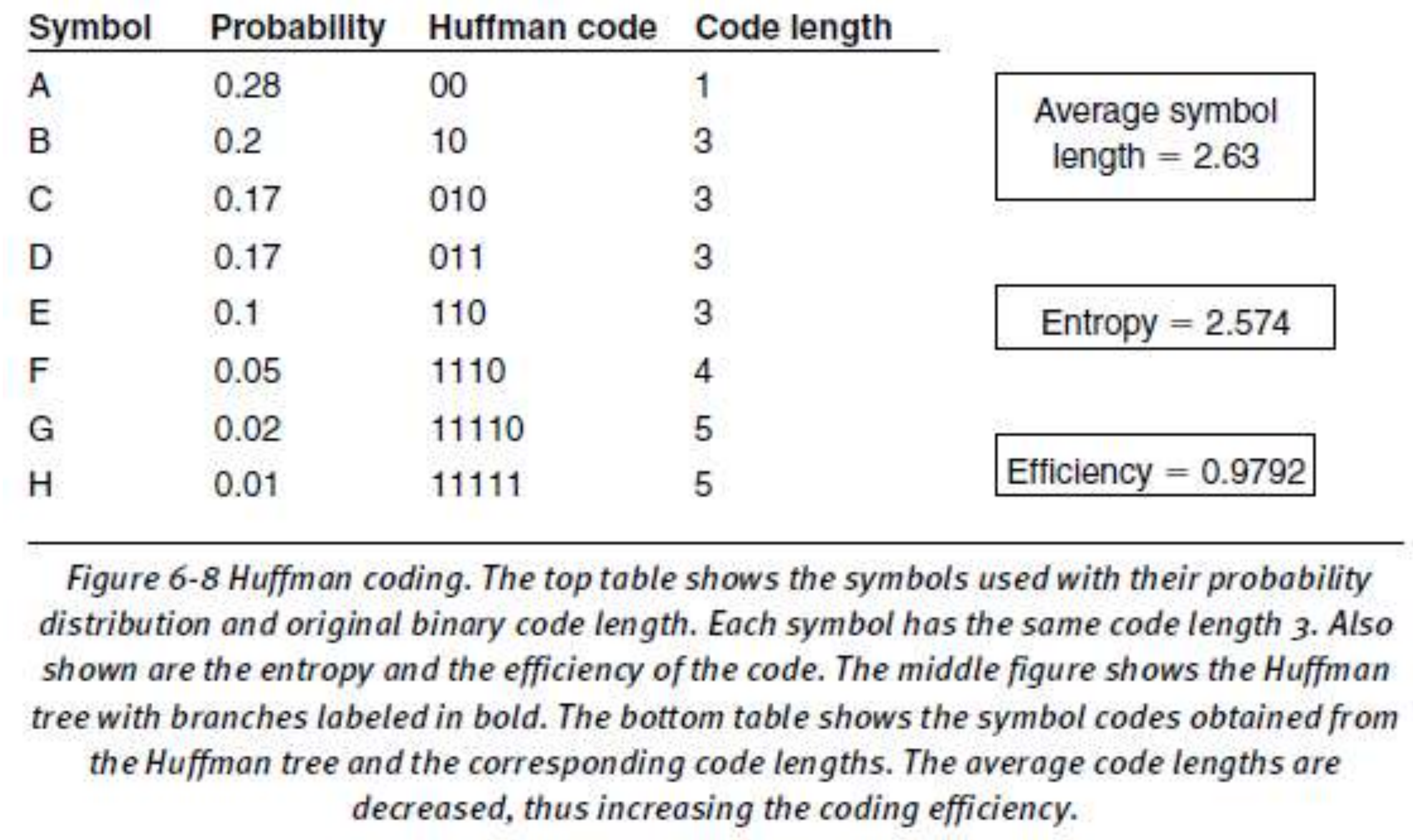
Huffman Coding (cont'd)

- Example 2 (cont'd):



Huffman Coding (cont’d)

- Example 2 (cont’d):



Symbol	Probability	Huffman code	Code length
A	0.28	00	1
B	0.2	10	3
C	0.17	010	3
D	0.17	011	3
E	0.1	110	3
F	0.05	1110	4
G	0.02	11110	5
H	0.01	11111	5

Average symbol length = 2.63

Entropy = 2.574

Efficiency = 0.9792

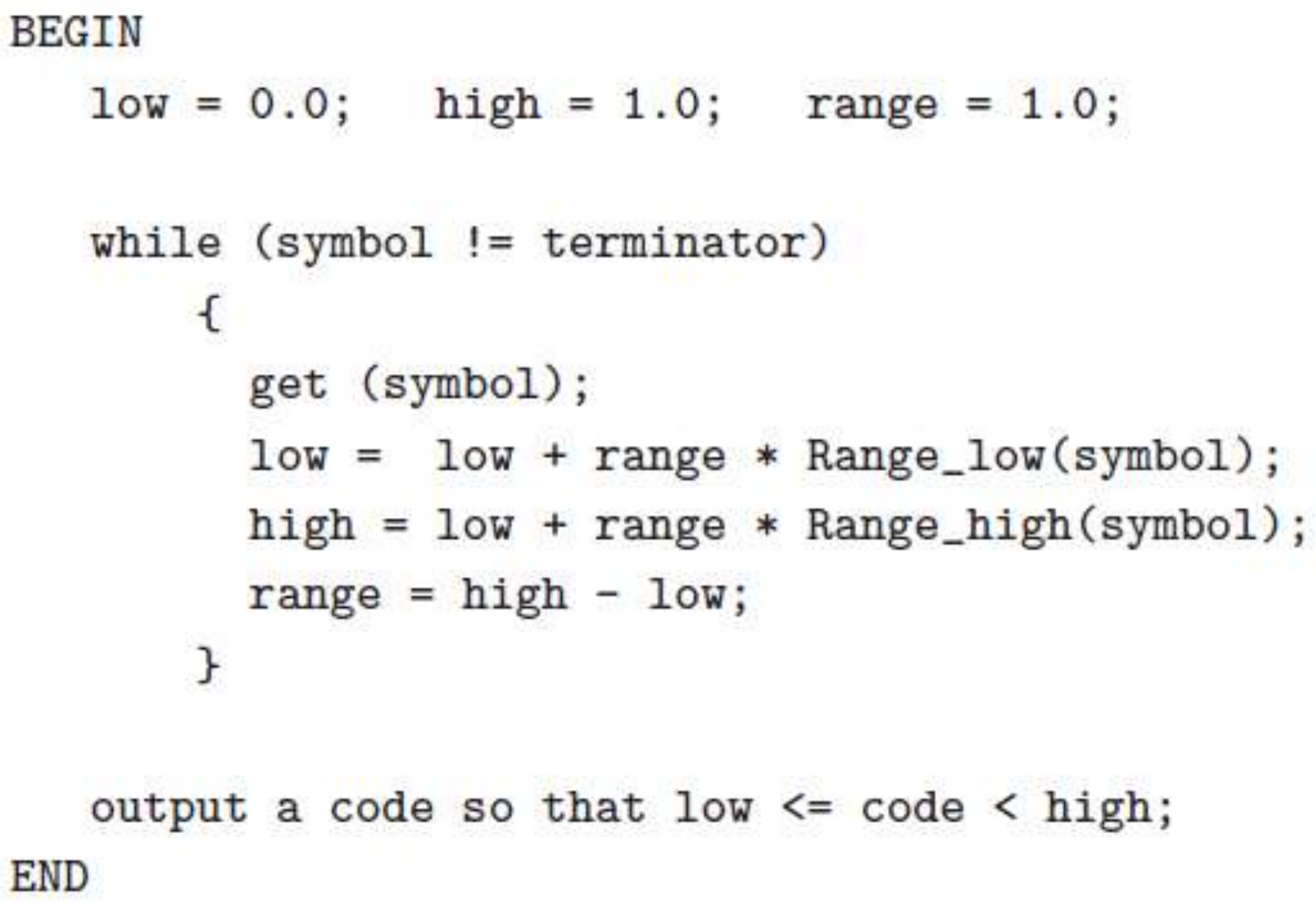
Figure 6-8 Huffman coding. The top table shows the symbols used with their probability distribution and original binary code length. Each symbol has the same code length 3. Also shown are the entropy and the efficiency of the code. The middle figure shows the Huffman tree with branches labeled in bold. The bottom table shows the symbol codes obtained from the Huffman tree and the corresponding code lengths. The average code lengths are decreased, thus increasing the coding efficiency.

Arithmetic Coding

- Arithmetic coding is a more modern coding method that usually out-performs Human coding.
- Human 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.
- Arithmetic vs. Huffman
  - Arithmetic encoding does not encode each symbol separately; Huffman encoding does.
  - Compression ratios of both are similar.

Arithmetic Coding (cont’d)

➢ Arithmetic Coding Encoder ALGORITHM



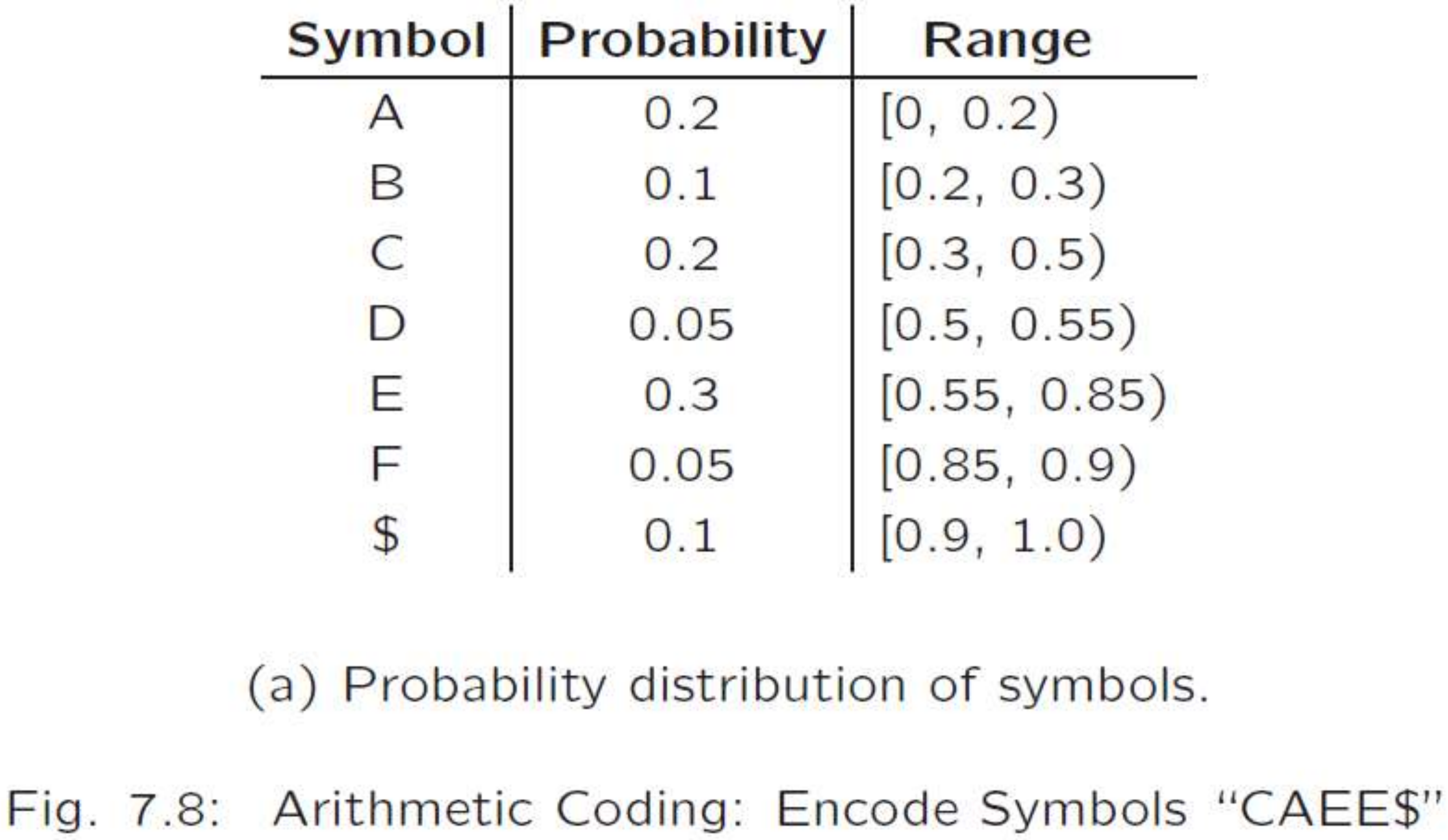
```
BEGIN
    low = 0.0;    high = 1.0;    range = 1.0;

    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
```

Arithmetic Coding (cont’d)

➢ 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$”

Arithmetic Coding (cont'd)

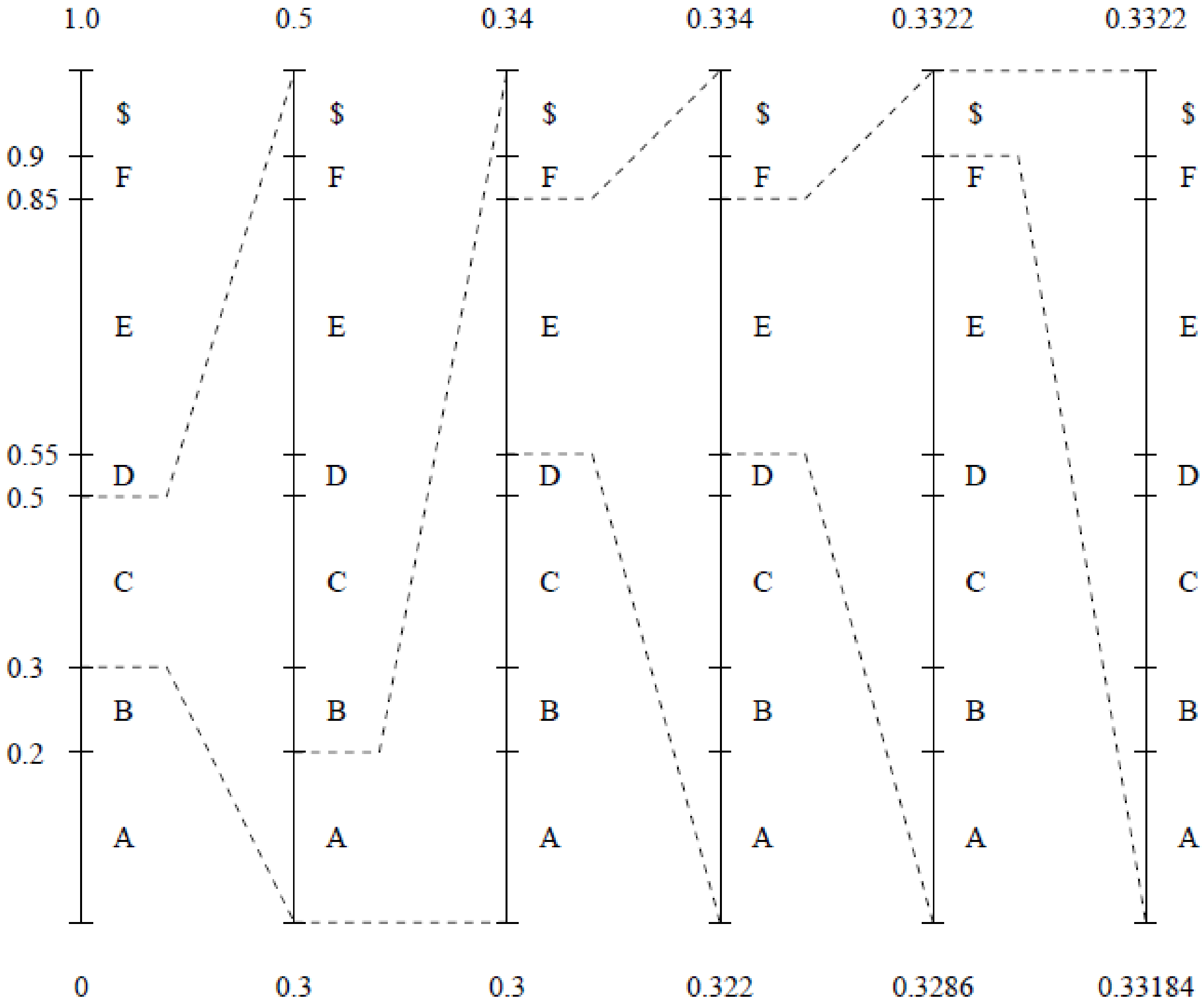
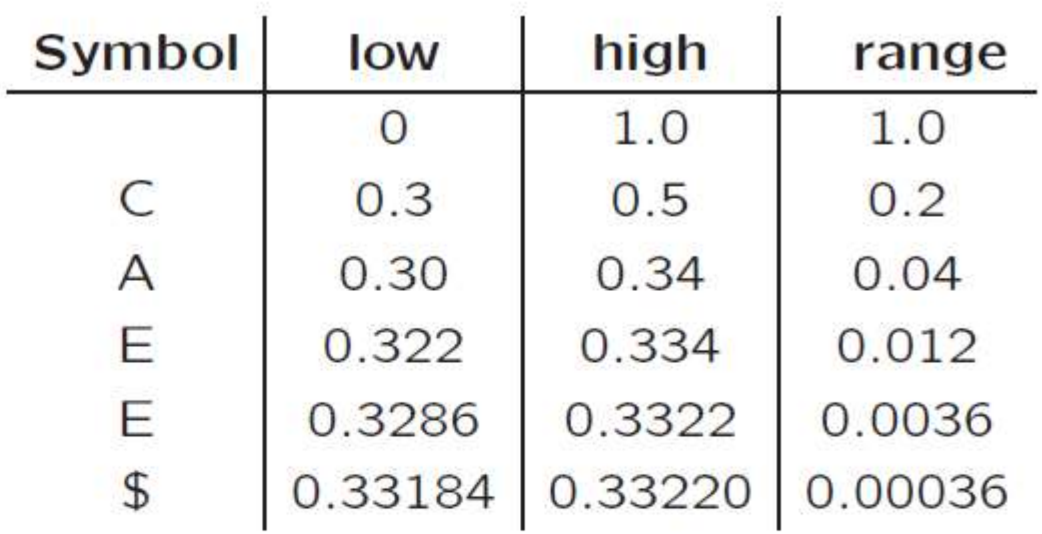


Fig. 7.8(b) Graphical display of shrinking ranges.

Arithmetic Coding (cont’d)



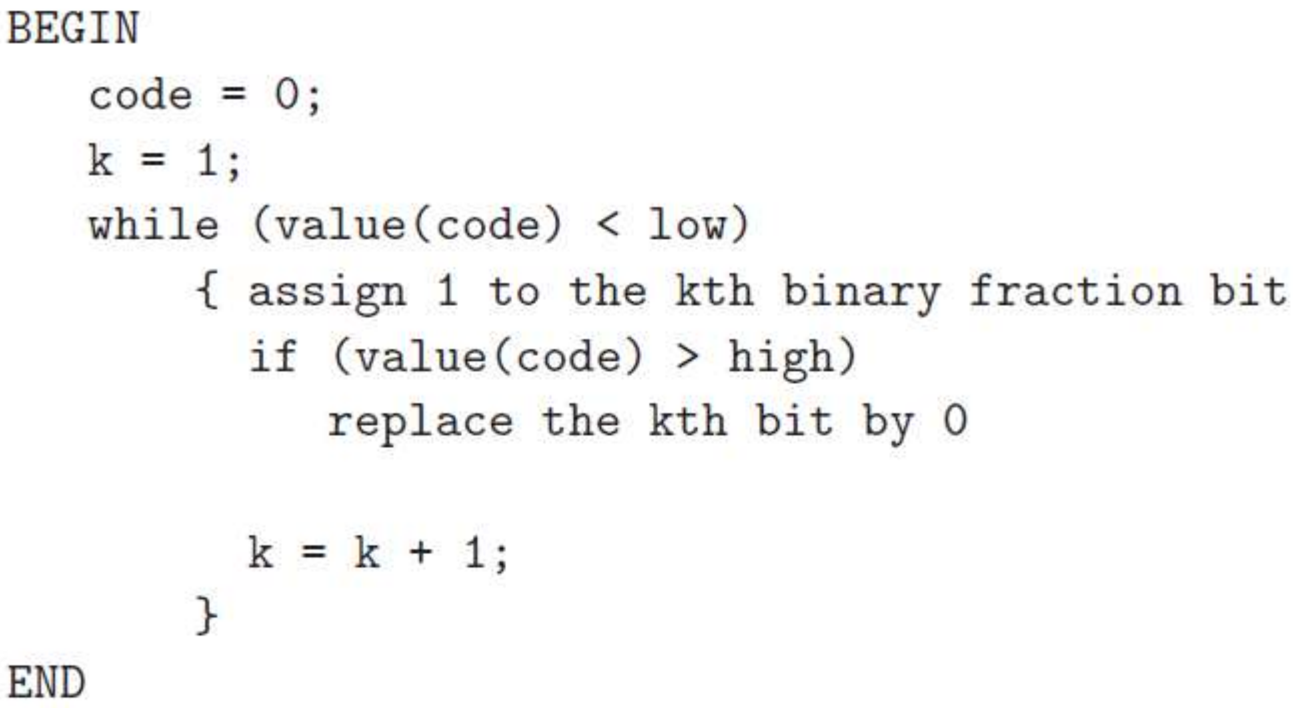
Symbol	low	high	range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

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

Fig. 7.8 (cont’d): Arithmetic Coding: Encode Symbols “CAEE$”

Arithmetic Coding (cont’d)

➢ PROCEDURE Generating Codeword for Encoder



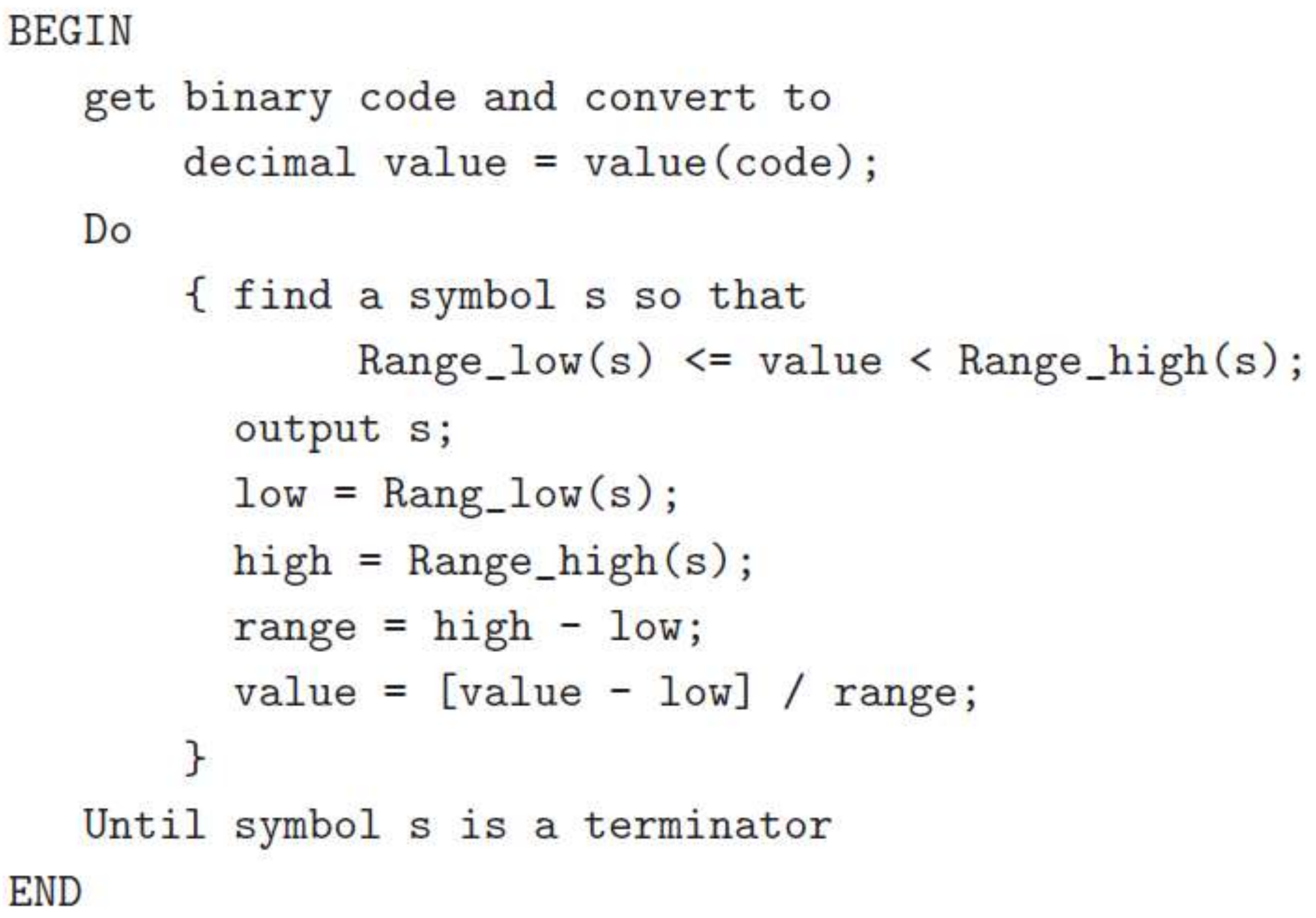
```
BEGIN
   code = 0;
   k = 1;
   while (value(code) < low)
       { assign 1 to the kth binary fraction bit
         if (value(code) > high)
            replace the kth bit by 0

         k = k + 1;
       }
END
```

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

Arithmetic Coding (cont’d)

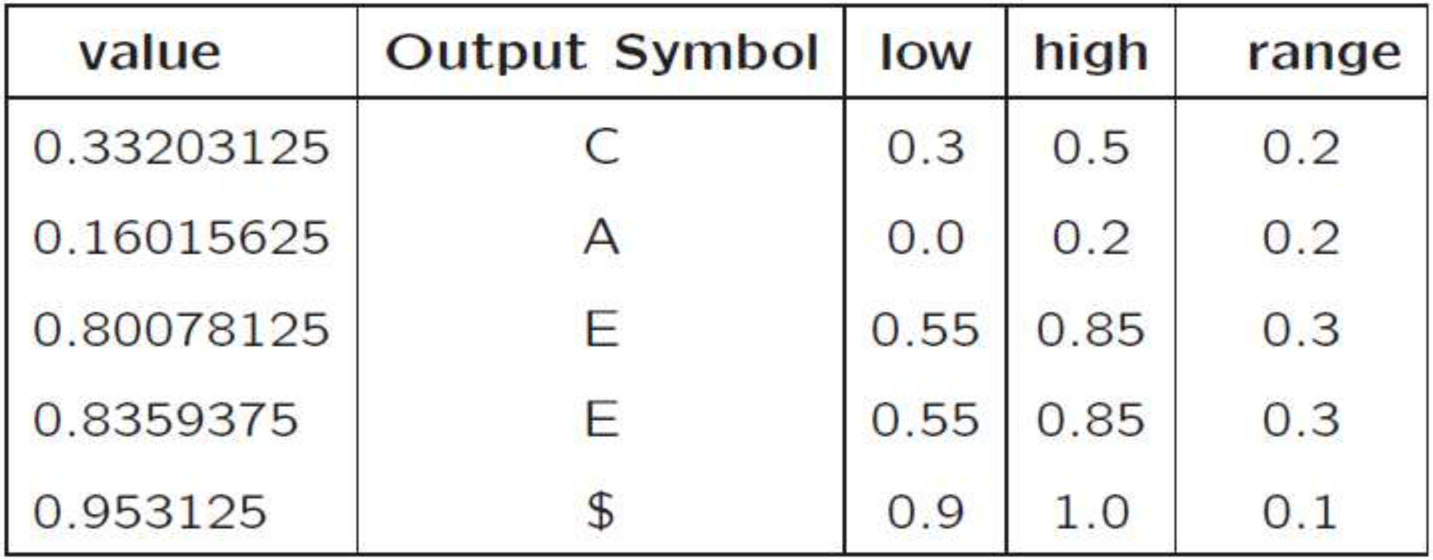
➢ Arithmetic Coding Decoder ALGORITHM



```
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
```

Arithmetic Coding (cont’d)

- **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

Outlines

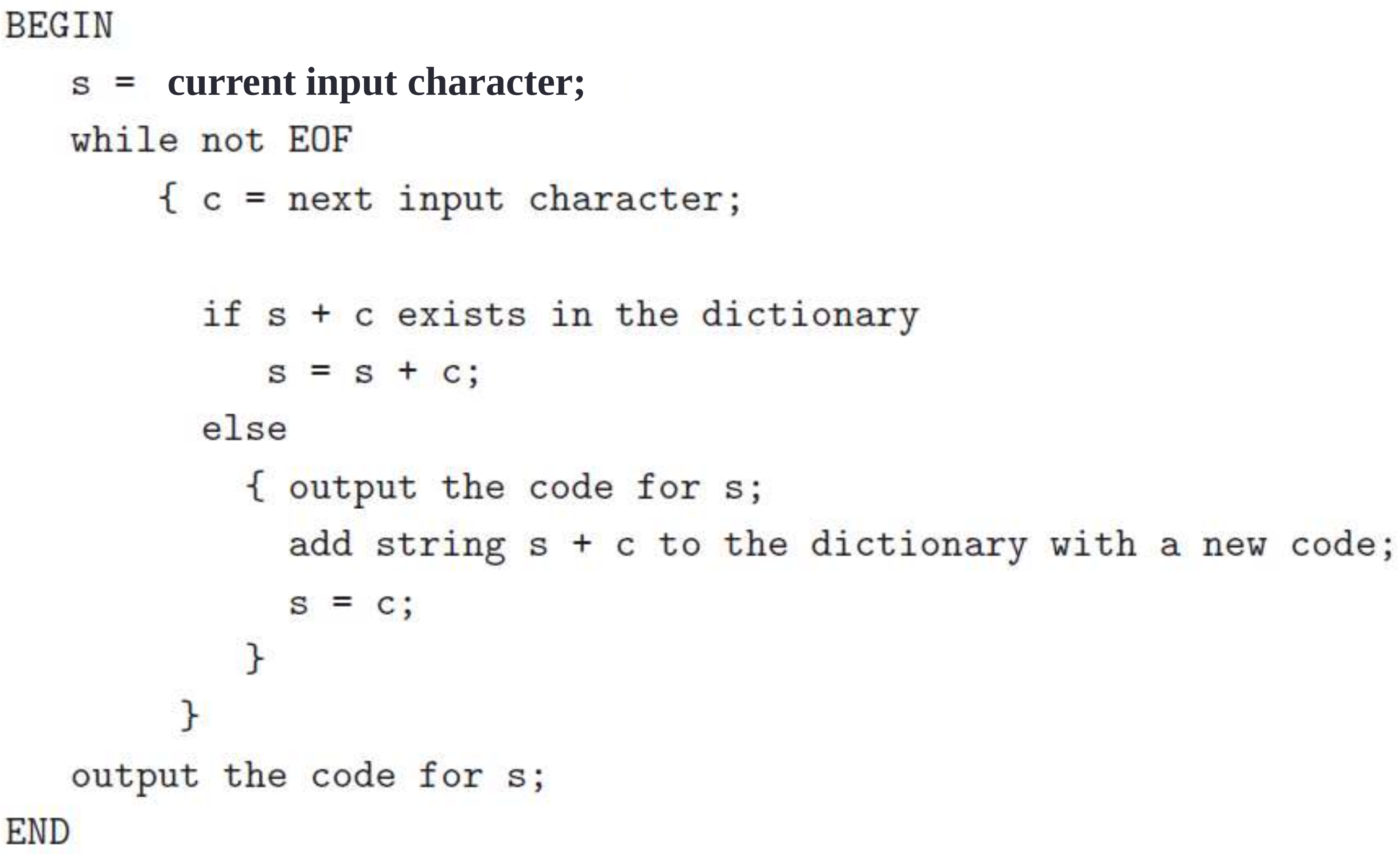
- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

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.

Dictionary-based Coding (cont'd)

➢ LZW Compression ALGORITHM

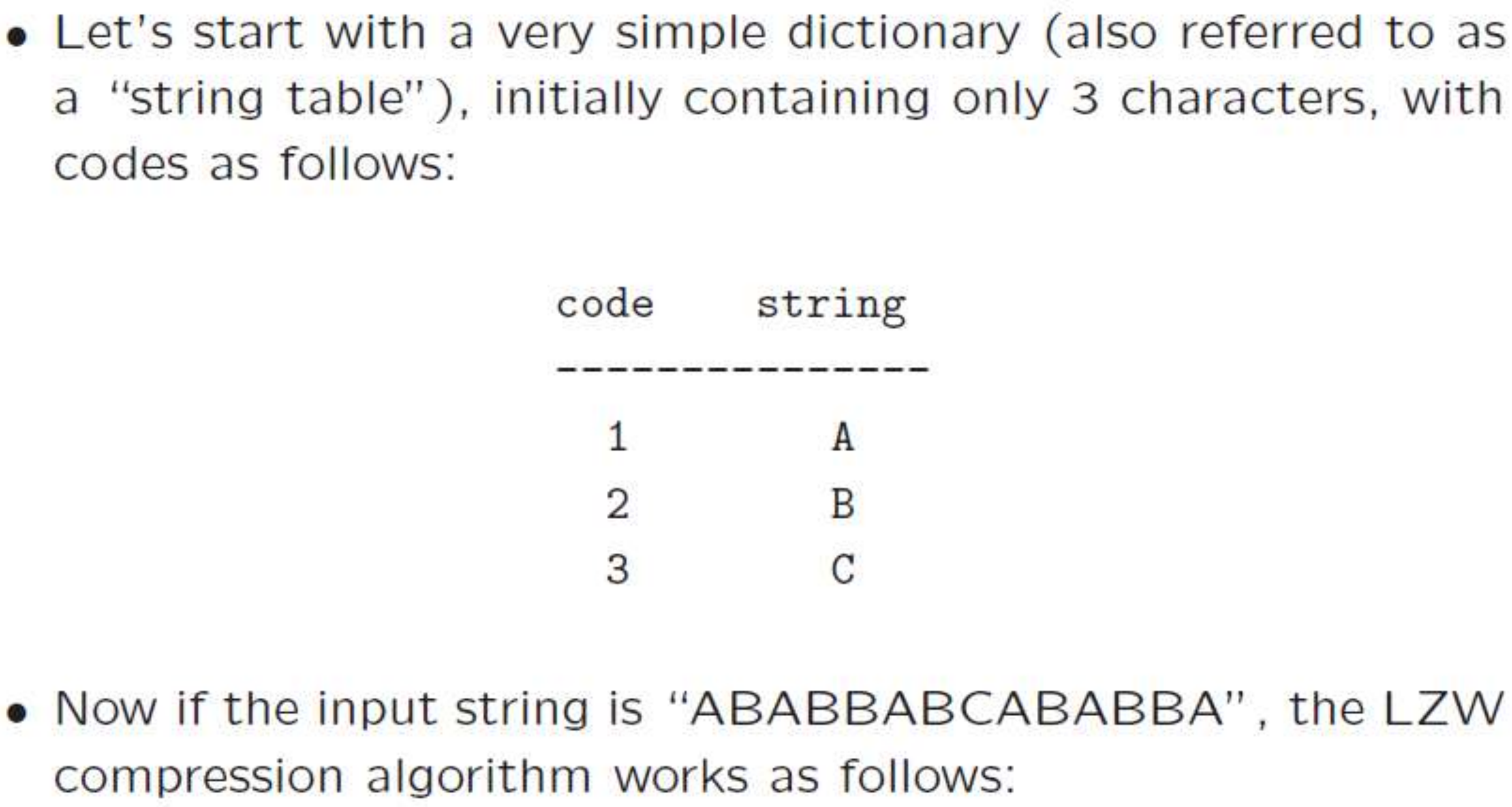


```
BEGIN
    s = current input character;
    while not EOF
        { c = next input character;

          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
```

Dictionary-based Coding (cont'd)

- **Example: LZW compression for string "ABABBABCABABBA"**

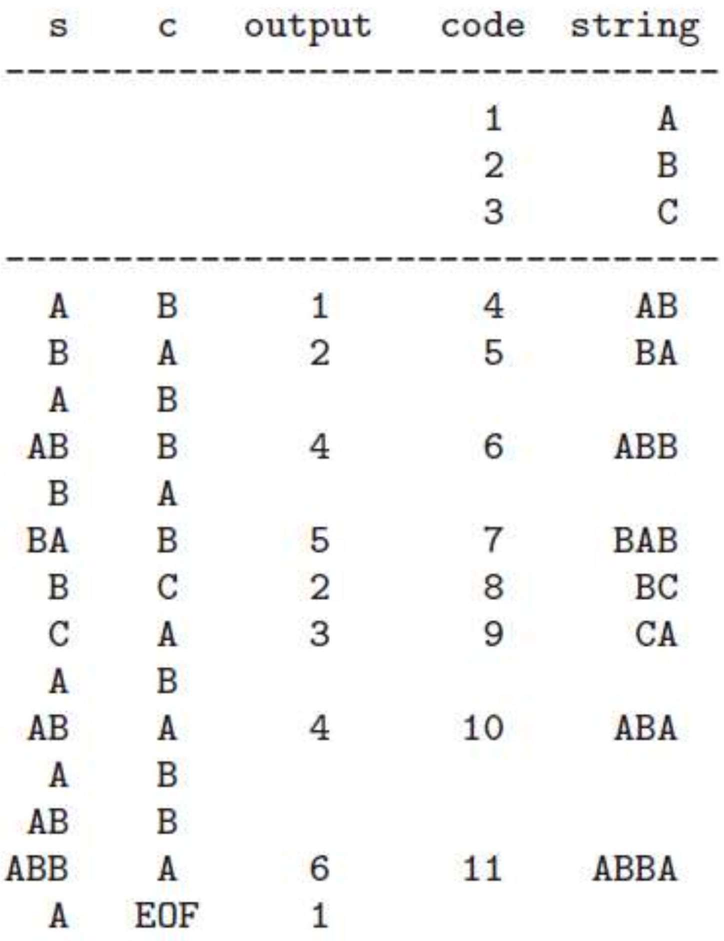


- Let's start with a very simple dictionary (also referred to as a "string table"), initially containing only 3 characters, with codes as follows:

code	string
1	A
2	B
3	C

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

Dictionary-based Coding (cont'd)

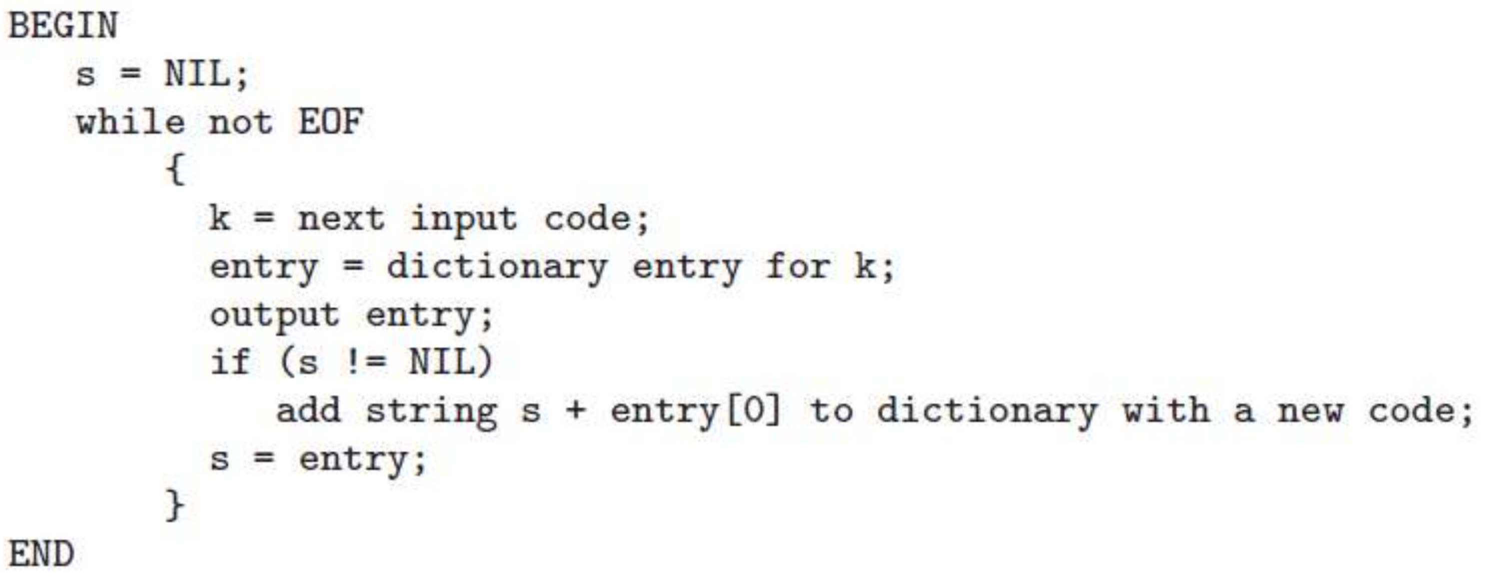


s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

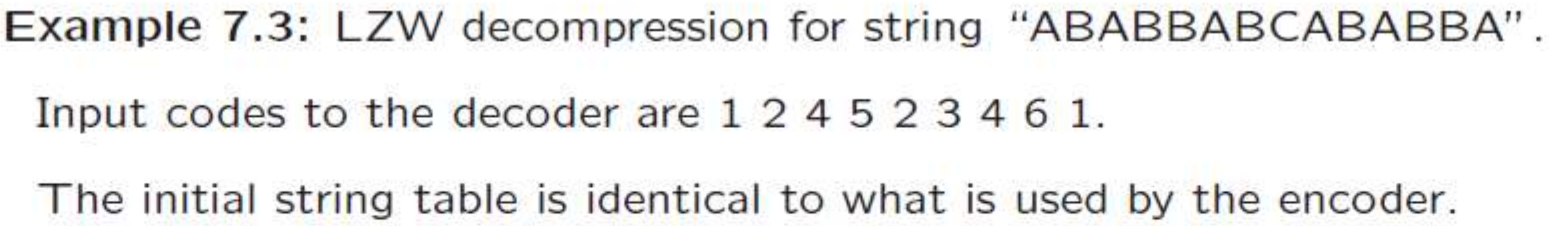
- The output codes are: 1 2 4 5 2 3 4 6 1. Instead of sending 14 characters, only 9 codes need to be sent (compression ratio = 14/9 = 1.56).

Dictionary-based Coding (cont'd)

➢ LZW Decompression ALGORITHM (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
```



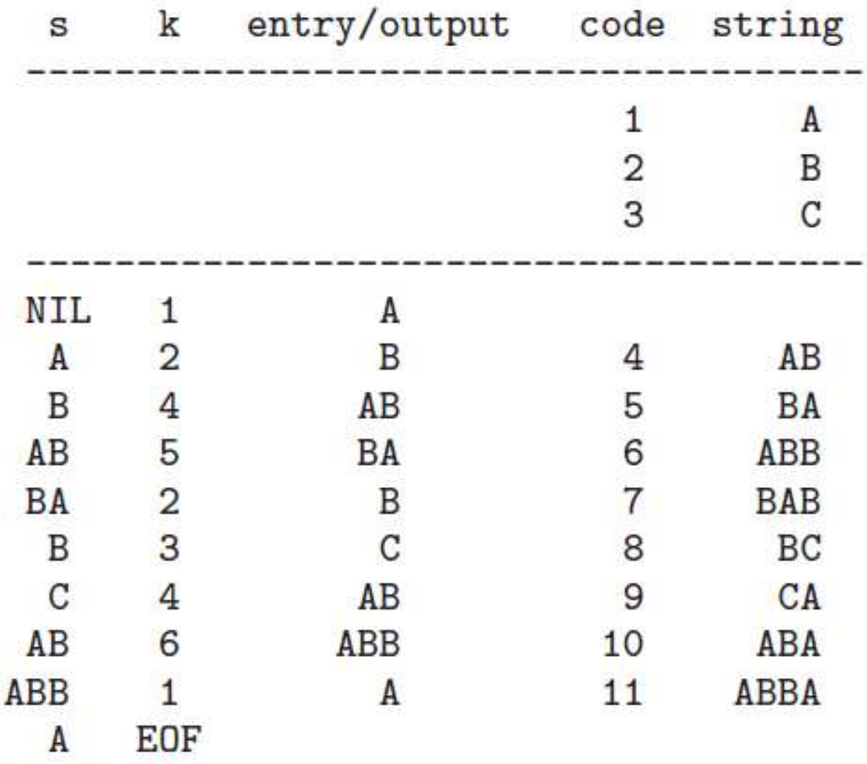
Example 7.3: LZW decompression for string "ABABBABCABABBA".

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

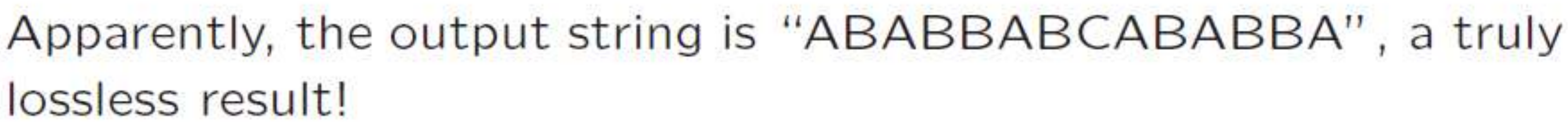
The initial string table is identical to what is used by the encoder.

Dictionary-based Coding (cont’d)

- 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!

Outlines

- The need for compression
- Basics of Information Theory
  - Entropy
  - Efficiency
- A taxonomy of compression
  - Compression metrics
  - Rate distortion
- Lossless compression
  - Run Length coding
  - Variable-length Coding
    - Huffman coding
    - Arithmetic coding
  - Dictionary-based coding
- Lossy compression

Lossy Compression

- Next week!!

Thank You