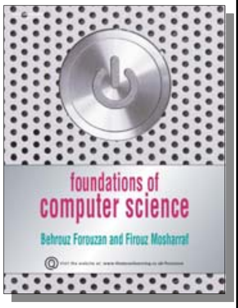


15
Data Compression

Objectives

After studying this chapter, the student should be able to:

- ❑ **Distinguish between lossless and lossy compression.**
- ❑ **Describe run-length encoding and how it achieves compression.**
- ❑ **Describe Huffman coding and how it achieves compression.**
- ❑ **Describe Lempel Ziv encoding and the role of the dictionary in encoding and decoding.**
- ❑ **Describe the main idea behind the JPEG standard for compressing still images.**
- ❑ **Describe the main idea behind the MPEG standard for compressing video and its relation to JPEG.**
- ❑ **Describe the main idea behind the MP3 standard for compressing audio.**

Data compression implies sending or storing a smaller number of bits. Although many methods are used for this purpose, in general these methods can be divided into two broad categories: **lossless** and **lossy** methods.

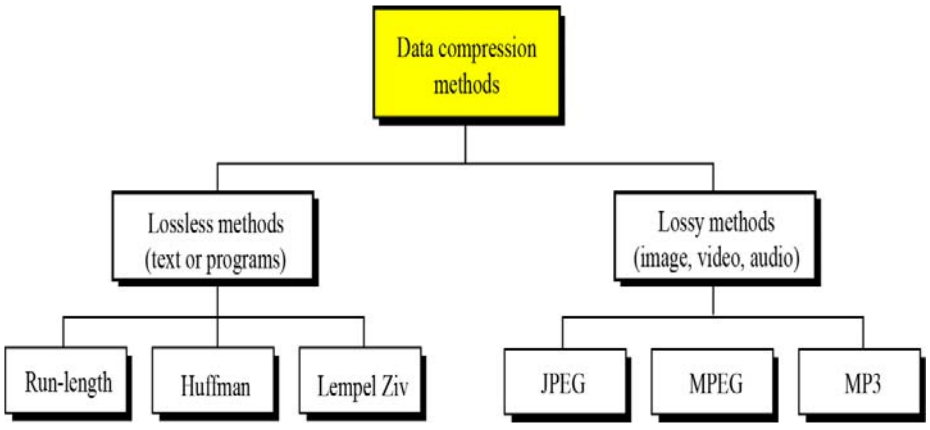

Data compression methods
Lossless methods (text or programs)
Lossy methods (image, video, audio)
Run-length
Huffman
Lempel Ziv
JPEG
MPEG
MP3

Figure 15.1 **Data compression methods**

15.3

15-1 LOSSLESS COMPRESSION

In **lossless** data compression, the integrity of the data is preserved. The original data and the data after compression and decompression are exactly the same because, in these methods, the compression and decompression algorithms are exact inverses of each other: no part of the data is lost in the process. **Redundant data is removed in compression** and **added during decompression**. Lossless compression methods are normally used when we cannot afford to lose any data.

15.4

Run-length encoding

Run-length encoding is probably the simplest method of compression. It can be used to compress data made of any combination of symbols. It does not need to know the frequency of occurrence of symbols and can be very efficient if data is represented as 0s and 1s.

The general idea behind this method **is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences**.

The method can be even **more efficient if the data uses only two symbols (for example 0 and 1)** in its bit pattern and one symbol is more frequent than the other.

Example1 of Run-length encoding

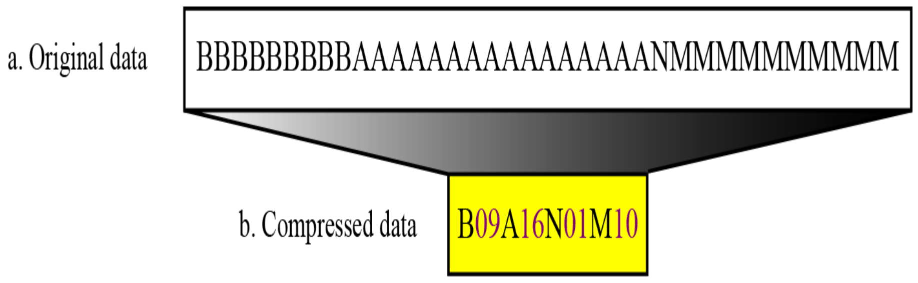
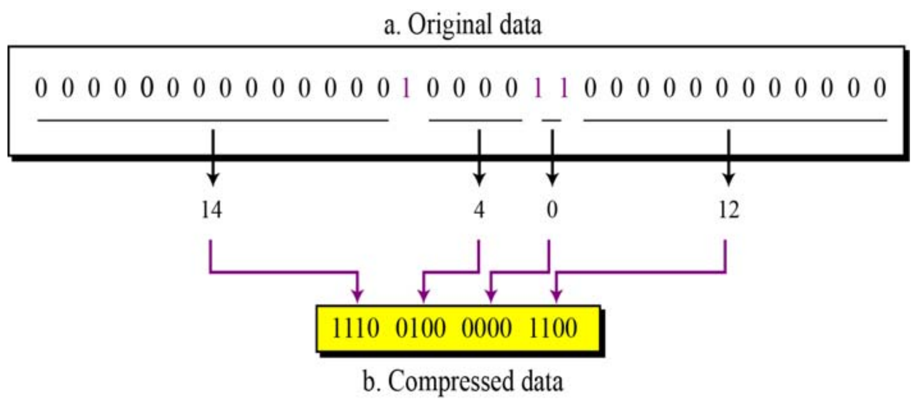


Figure 15.2 **Run-length encoding example**

Example2 of Run-length encoding



There are **number of 0s** before the **first 1**.

Figure 15.3 **Run-length encoding for two symbols**

15.7

Huffman coding

Huffman coding **assigns** shorter codes to symbols that occur more frequently **and** longer codes to those that occur less frequently. For example, imagine we have a text file that uses only five characters (A, B, C, D, E). Before we can assign bit patterns to each character, we assign each character a weight based on its frequency of use. In this example, assume that the frequency of the characters is as shown in Table 15.1.

Table 15.1 Frequency of characters

Character	A	B	C	D	E
Frequency	17	12	12	27	32

15.8

#bits = 3bits*(17+12+12+27+32)=300bits

Example1 of Huffman coding

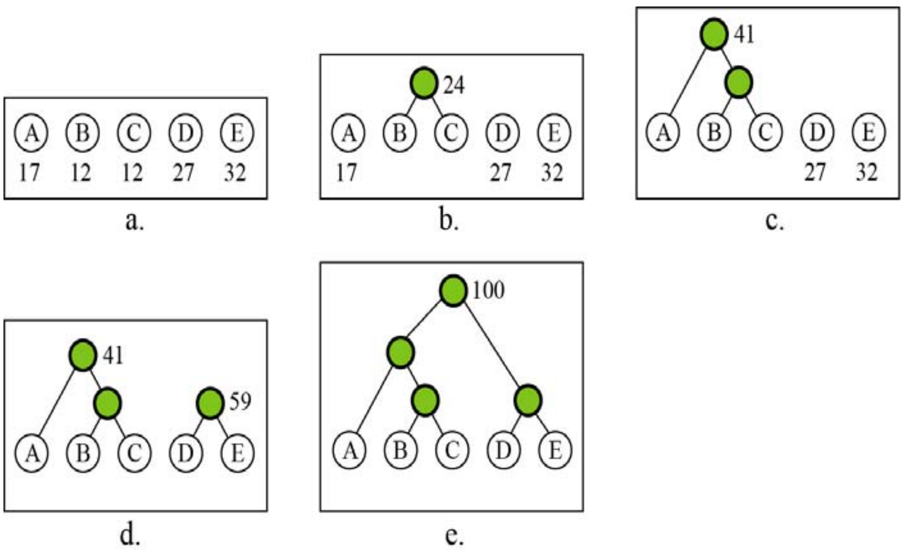
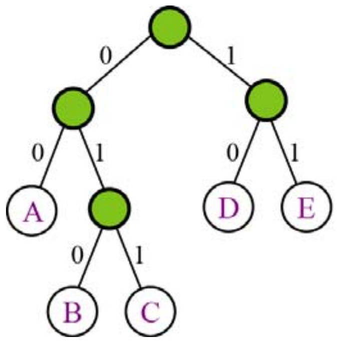


Figure 15.4 **Huffman coding**

15.9

A character's code is found by starting at the root and following the branches that lead to that character. The code itself is the bit value of each branch on the path, taken in sequence.



A: 00	D: 10
B: 010	E: 11
C: 011	

Code

#bits = 2bits*(17+27+32)+3bits*(12+12)=<u>224bits</u>

⇕

Figure 15.5 **Final tree and code** 300bits

15.10

Huffman Encoding

Let us see how to encode text using the code for our five characters. Figure 15.6 shows the original and the encoded text.

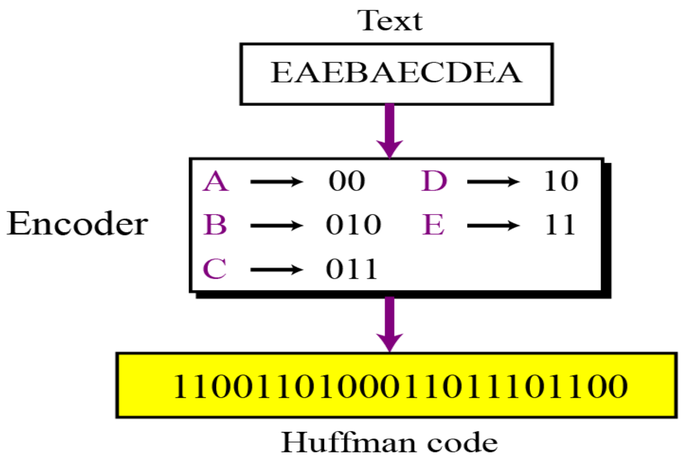


Figure 15.6 **Huffman encoding**

15.11

Decoding

The recipient has a very easy job in decoding the data it receives. Figure 15.7 shows how decoding takes place.

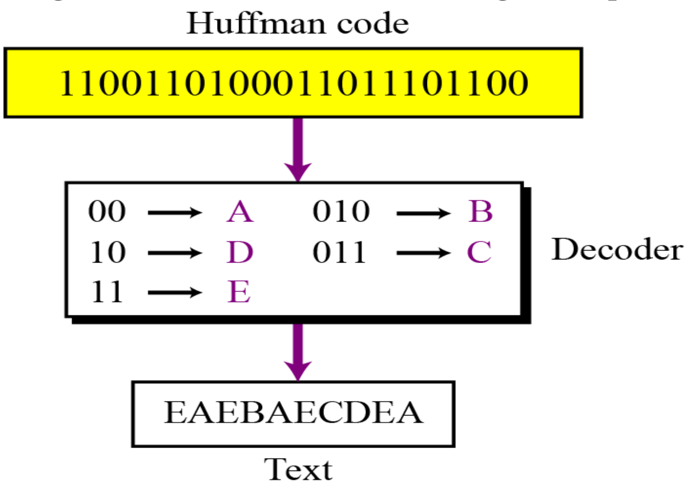


Figure 15.7 **Huffman decoding**

15.12

Lempel Ziv encoding

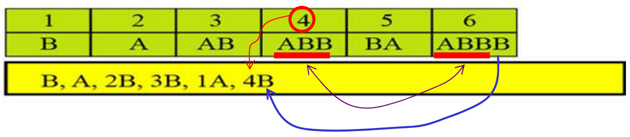
Lempel Ziv (LZ) encoding is an example of a category of algorithms called *dictionary-based* encoding. The idea is to create a dictionary (a table) of strings used during the communication session. If both the sender and the receiver have a copy of the dictionary, then previously-encountered strings can be substituted by their index in the dictionary to reduce the amount of information transmitted.

15.13

Compression

In this phase there are two concurrent events: **building an indexed dictionary** and **compressing a string of symbols**. The algorithm extracts the smallest substring that cannot be found in the dictionary from the remaining uncompressed string. It then stores a copy of this substring in the dictionary as a new entry (**ABBB**) and assigns it an index value (**6**). **Compression occurs** when the substring (**ABB**), except for the last character (**B**), is replaced with the index found in the dictionary. The process then **inserts the index (4) and the last character (B) of the substring into the compressed string**.

Example:



1	2	3	4	5	6
B	A	AB	ABB	BA	ABBB

B, A, 2B, 3B, 1A, 4B

15.14

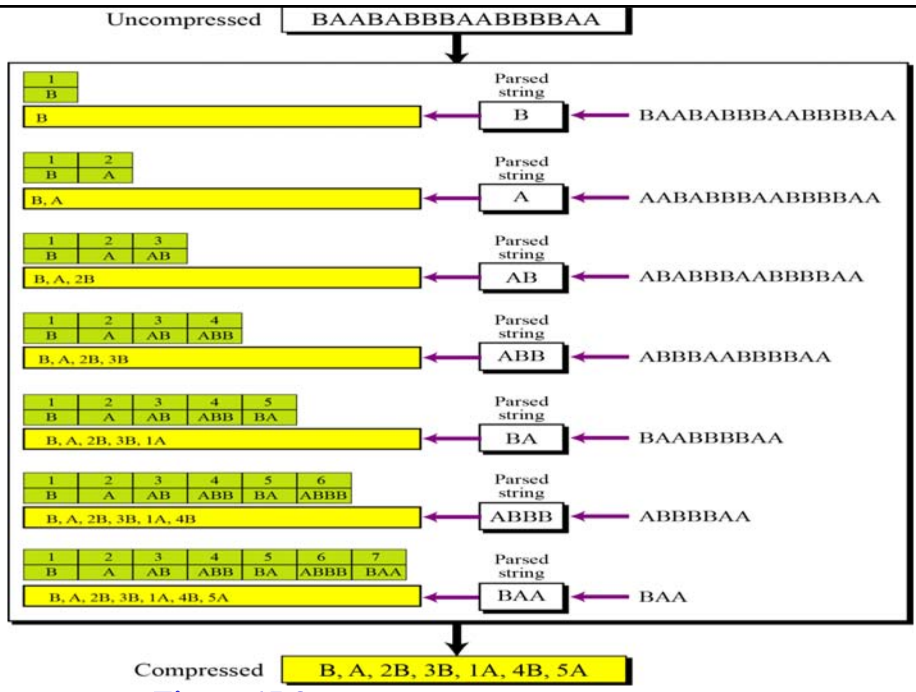


Figure 15.8 **An example of Lempel Ziv encoding**

Decompression

Decompression is the inverse of the compression process. The process extracts the substrings from the compressed string and tries to replace the indexes with the corresponding entry in the dictionary, which is empty at first and built up gradually. The idea is that when an index is received, there is already an entry in the dictionary corresponding to that index.

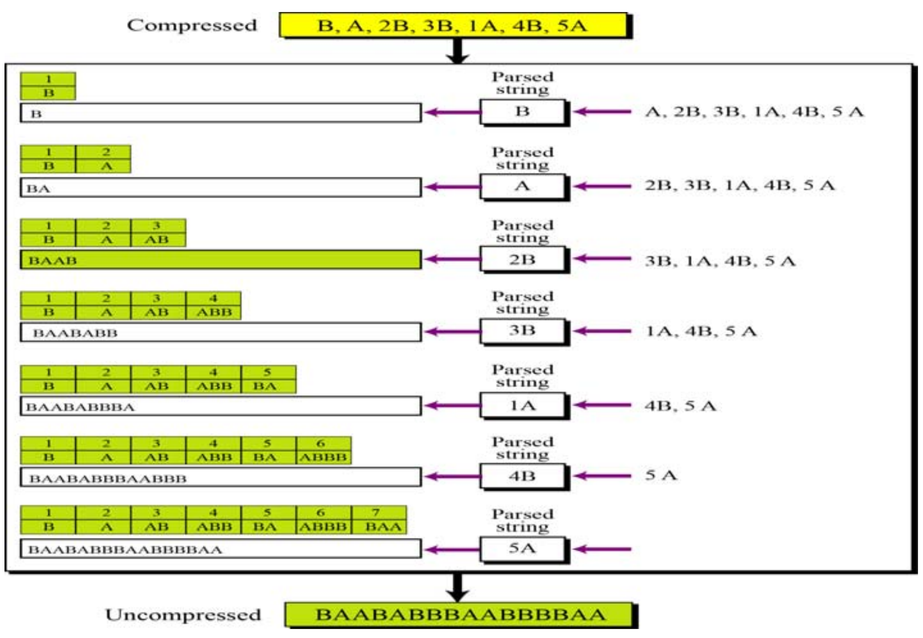


Figure 15.9 **An example of Lempel Ziv decoding**

15.17

15-2 LOSSY COMPRESSION METHODS

Our eyes and ears cannot distinguish subtle changes. In such cases, we can use a lossy data compression method. **These methods are cheaper — they take less time and space** when it comes to sending millions of bits per second for images and video. Several methods have been developed using lossy compression techniques. **JPEG (Joint Photographic Experts Group)** encoding is used to compress pictures and graphics, **MPEG (Moving Picture Experts Group)** encoding is used to compress video, and **MP3 (MPEG audio layer 3)** for audio compression.

15.18

Image compression – JPEG encoding

As discussed in Chapter 2, an image can be represented by a two-dimensional array (table) of picture elements (pixels).
A **grayscale** picture of 307,200 pixels (640*480) is represented by 2,457,600 bits (i.e., 8 bits * 307200), and
a **color** picture is represented by 7,372,800 bits (i.e., 24 bits * 307200).

In JPEG, a grayscale picture is divided into blocks of **8 × 8 pixel blocks** to decrease the number of calculations because, as we will see shortly, **the number of mathematical operations for each picture is the square of the number of units**.
For the entire image of 640 * 480 pixels, required 307200^2 operations can be reduced to be 64^2*80*60 = 19660800 operations

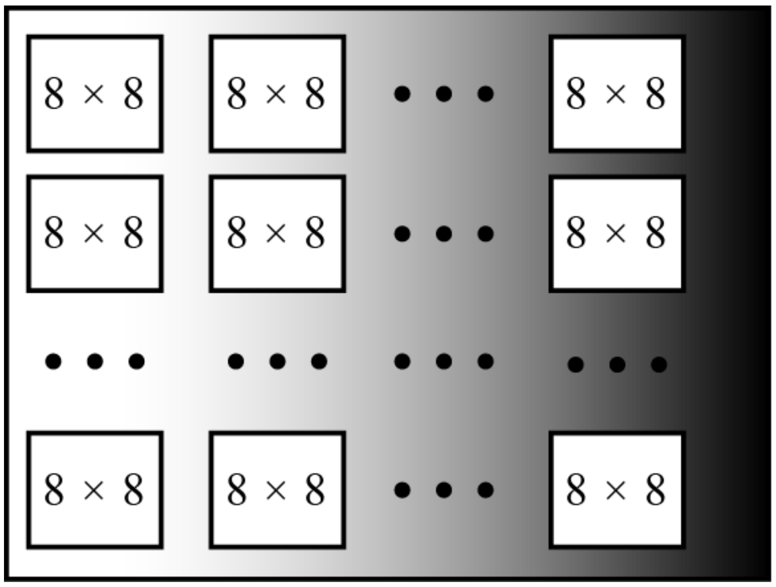
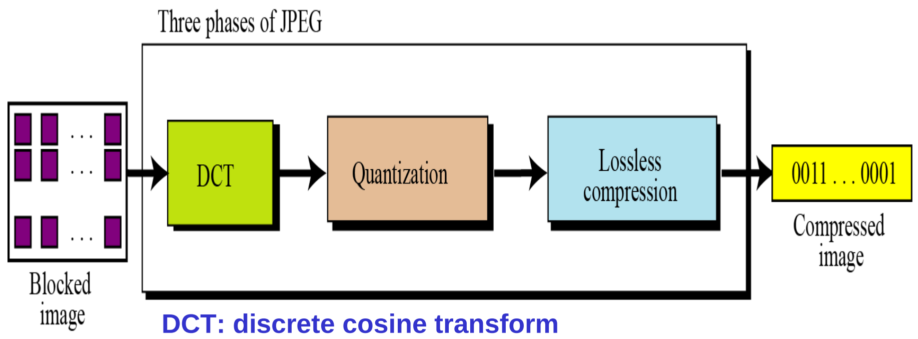


Figure 15.10 JPEG grayscale example, 640 × 480 pixels

15.20

The whole idea of JPEG is to **change the picture into a linear (vector) set of numbers that reveals the redundancies**. The redundancies (lack of changes) can then be removed using one of the lossless compression methods we studied previously. A simplified version of the process is shown in Figure 15.11.



DCT: discrete cosine transform

Figure 15.11 **The JPEG compression process**

15.21

Discrete cosine transform (DCT)

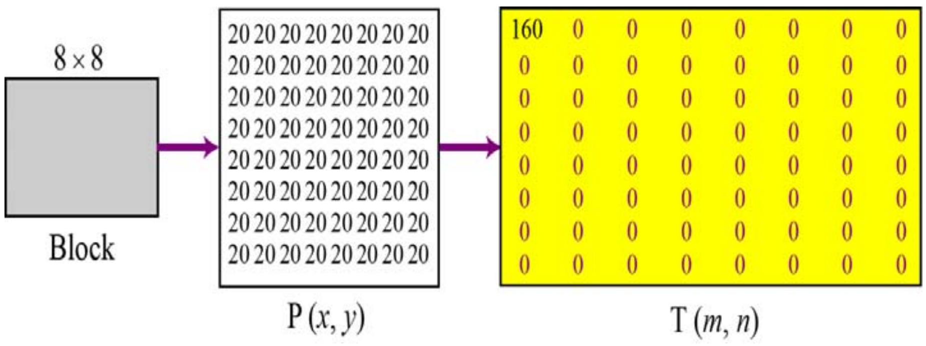
In this step, each block of 64 pixels goes through a transformation called the **discrete cosine transform (DCT).** The transformation **changes the 64 values so that the relative relationships** between pixels are kept but **the redundancies are revealed**.

The formula is given in Appendix G. **P(*x*, *y*) defines one value in the block**, while **T(*m*, *n*) defines the value in the transformed block**.

15.22

To understand the nature of this transformation, let us show the result of the transformations for three cases.

Case 1: Uniform grayscale



P (x, y):

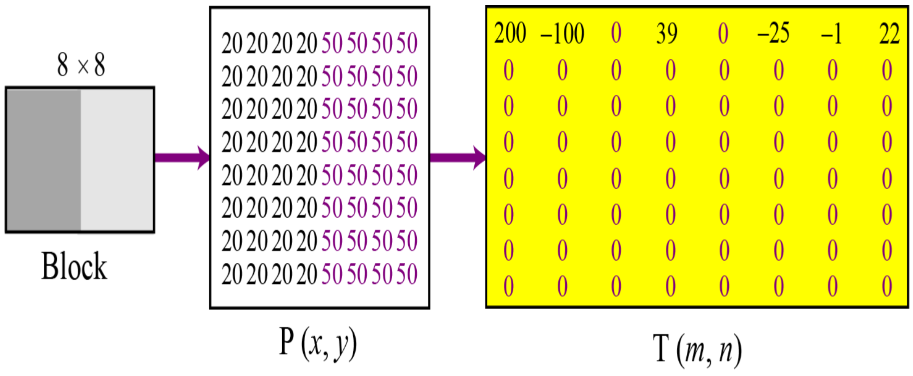
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20

T (m, n):

160	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 15.12 **Case 1: uniform grayscale**

Case 2: Two different uniform grayscale sections



P (x, y):

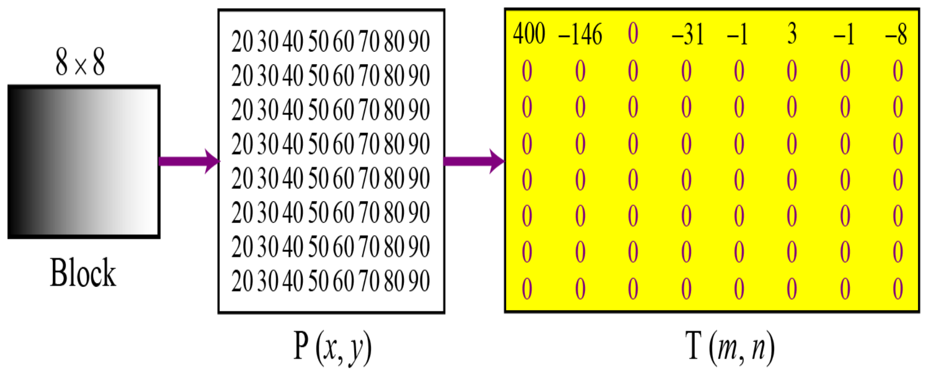
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50
20	20	20	20	50	50	50	50

T (m, n):

200	−100	0	39	0	−25	−1	22
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 15.13 **Case 2: two sections**

Case 3: A block that changes gradually



P (x, y):

20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90
20	30	40	50	60	70	80	90

T (m, n):

400	−146	0	−31	−1	3	−1	−8
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 15.14 **Case 3: gradient grayscale**

Quantization

After the T table is created, the values are quantized to reduce the number of bits needed for encoding. **Quantization divides the number of bits by a constant and then drops the fraction**. This reduces the required number of bits even more. In most implementations, a quantizing table (8 by 8) defines how to quantize each value. The divisor depends on the position of the value in the T table. This is done to optimize the number of bits and the number of 0s for each particular application.

Compression

After quantization the values are read from the table, and redundant 0s are removed. However, **to cluster the 0s together, the process reads the table diagonally in a zigzag fashion rather than row by row or column by column**. The reason is that if the picture does not have fine changes, the bottom right corner of the T table is all 0s.

JPEG usually uses run-length encoding at the compression phase to compress the bit pattern resulting from the zigzag linearization.

15.27

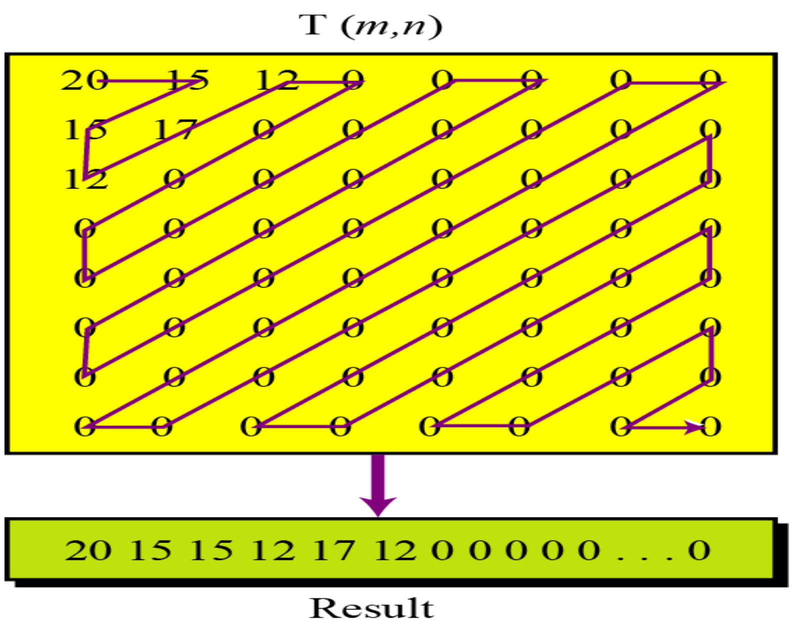


Figure 15.15 Reading the table

15.28

Video compression – MPEG encoding

The **MPEG (Moving Picture Experts Group)** method is used to compress video. In principle, a motion picture is a rapid sequence of a set of frames in which each frame is a picture. In other words, a frame is a spatial combination of pixels, and a video is a temporal combination of frames that are sent one after another. Compressing video, then, means spatially compressing each frame and temporally compressing a set of frames.

15.29

Spatial compression

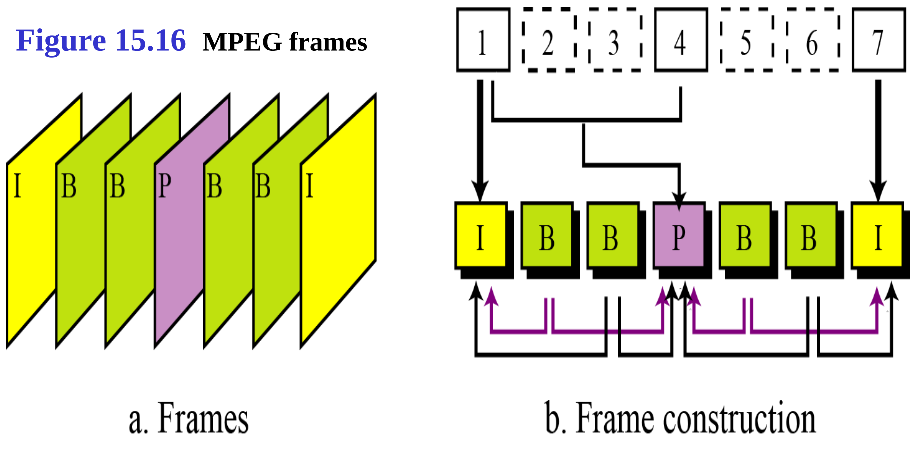
The spatial compression of each frame is done with JPEG, or a modification of it. Each frame is a picture that can be independently compressed.

Temporal compression

In temporal compression, redundant frames are removed. When we watch television, for example, we receive 30 frames per second. However, most of the consecutive frames are almost the same. For example, in a static scene in which someone is talking, most frames are the same except for the segment around the speaker's lips, which changes from one frame to the next.

15.30

Figure 15.16 **MPEG frames**



MPEG divides frames into three categories:
I-frame:* *Intracoded frame
P-frame:* *Predicted frame is related to the preceding I-frame or P-frame.
B-frame:* *Bidirectional frame is relative to the preceding and following I-frame or P-frame.

15.31

Audio compression

Audio compression can be used for speech or music. **For speech we need to compress a 64 kHz digitized signal**, while **for music we need to compress a 1.411 MHz signal**. Two categories of techniques are used for audio compression: **predictive encoding** and **perceptual encoding**.

15.32

Predictive encoding

In predictive encoding, **the differences between samples are encoded instead of encoding all the sampled values**. This type of compression is normally **used for speech**. Several standards have been defined such as GSM (13 kbps), G.729 (8 kbps), and G.723.3 (6.4 or 5.3 kbps). Detailed discussions of these techniques are beyond the scope of this book.

Perceptual encoding: MP3

The most common compression technique **used to create CD-quality audio** is based on the perceptual encoding technique. This type of audio **needs at least 1.411 Mbps**, which cannot be sent over the Internet without compression. MP3 (MPEG audio layer 3) uses this technique.