JPEG Compression

Subhadip Basu

Computer Science and Engineering Department
Jadavpur University

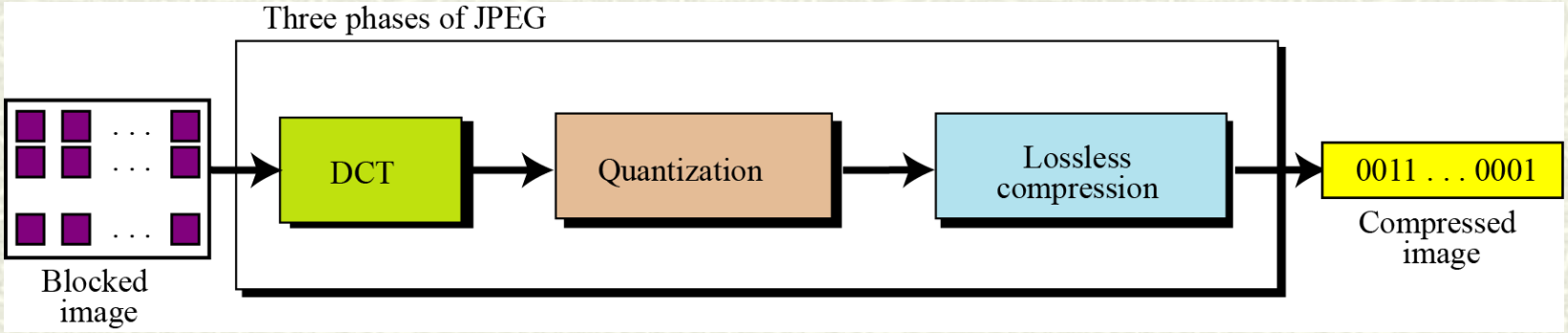
Fact about JPEG Compression

- JPEG stands for Joint Photographic Experts Group
- JPEG compression is used with .jpg and can be embedded in .tiff and .eps files.
- Used on 24-bit color files.
- Works well on photographic images.
- Although it is a lossy compression technique, it yields an excellent quality image with high compression rates.

Steps in JPEG Compression

- 1. (Optionally) If the color is represented in RGB mode, translate it to YUV.
- 2. Divide the file into 8 X 8 blocks.
- 3. Transform the pixel information from the spatial domain to the frequency domain with the Discrete Cosine Transform.
- 4. Quantize the resulting values by dividing each coefficient by an integer value and rounding off to the nearest integer.
- 5. Look at the resulting coefficients in a zigzag order. Do a run-length encoding of the coefficients ordered in this manner. Follow by Huffman coding.

JPEG Compression



Step 1a: Converting RGB to YUV

- YUV color mode stores color in terms of its luminance (brightness) and chrominance (hue).
- The human eye is less sensitive to chrominance than luminance.
- YUV is not required for JPEG compression, but it gives a better compression rate.

RGB vs. YUV

- It's simple arithmetic to convert RGB to YUV. The formula is based on the relative contributions that red, green, and blue make to the luminance and chrominance factors.
- There are several different formulas in use depending on the target monitor. For example:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

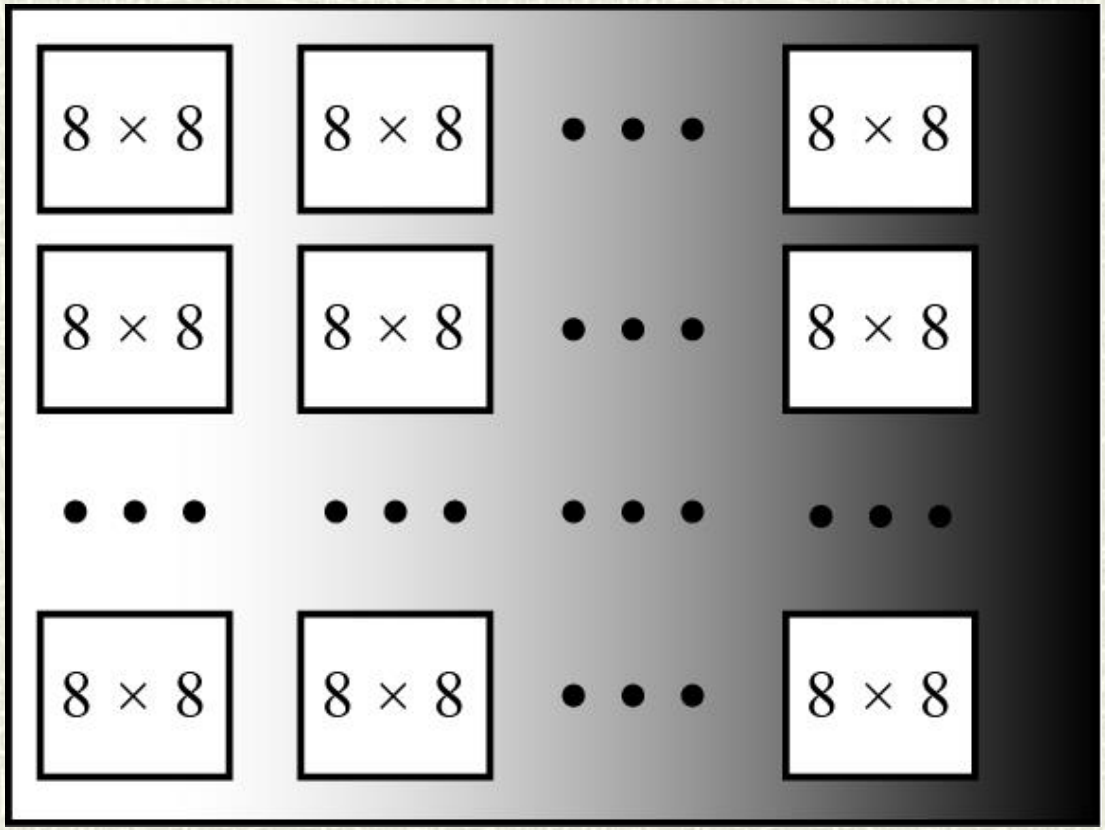
$$U = -0.1687 * R - 0.3313 * G + 0.5 * B + 128$$

$$V = 0.5 * R - 0.4187 * G - 0.813 * B + 128$$

Step 2: Divide into 8 X 8 blocks

- Note that with YUV color, you have 16 pixels of information in each block for the Y component (though only 8 in each direction for the U and V components).
- If the file doesn't divide evenly into 8 X 8 blocks, extra pixels are added to the end and discarded after the compression.
- The values are shifted "left" by subtracting 128.

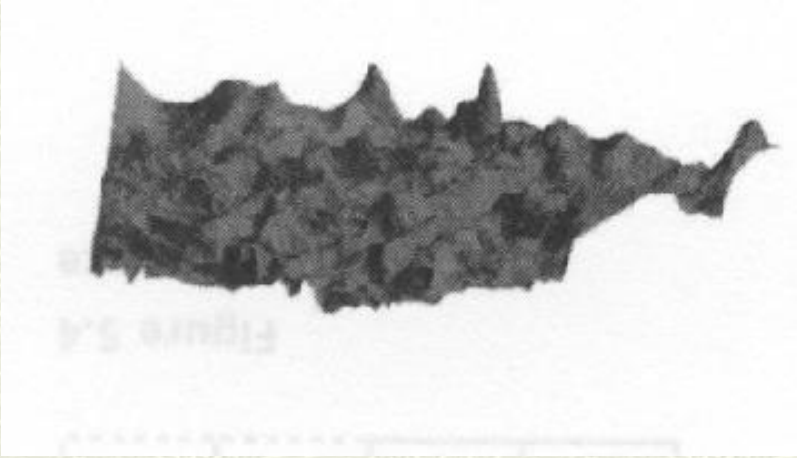
(See JPEG Compression for details.)



JPEG grayscale example, 640 × 480 pixels

Discrete Cosine Transform

- The DCT transforms the data from the spatial domain to the frequency domain.
- The spatial domain shows the amplitude of the color as you move through space
- The frequency domain shows how quickly the amplitude of the color is changing from one pixel to the next in an image file.



Step 3: DCT

- The frequency domain is a better representation for the data because it makes it possible for you to separate out – and throw away – information that isn't very important to human perception.
- The human eye is not very sensitive to high frequency changes – especially in photographic images, so the high frequency data can, to some extent, be discarded.

Step 3: Forward DCT

For an N X N pixel image $[p_{uv}, 0 \le u < N, 0 \le v < N]$

the DCT is an array of coefficients

$[DCT_{uv}, 0 \le u < N, 0 \le v < N]$ where

$$DCT_{uv} = \frac{1}{\sqrt{2N}} C_u C_v \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p_{xy} \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

where

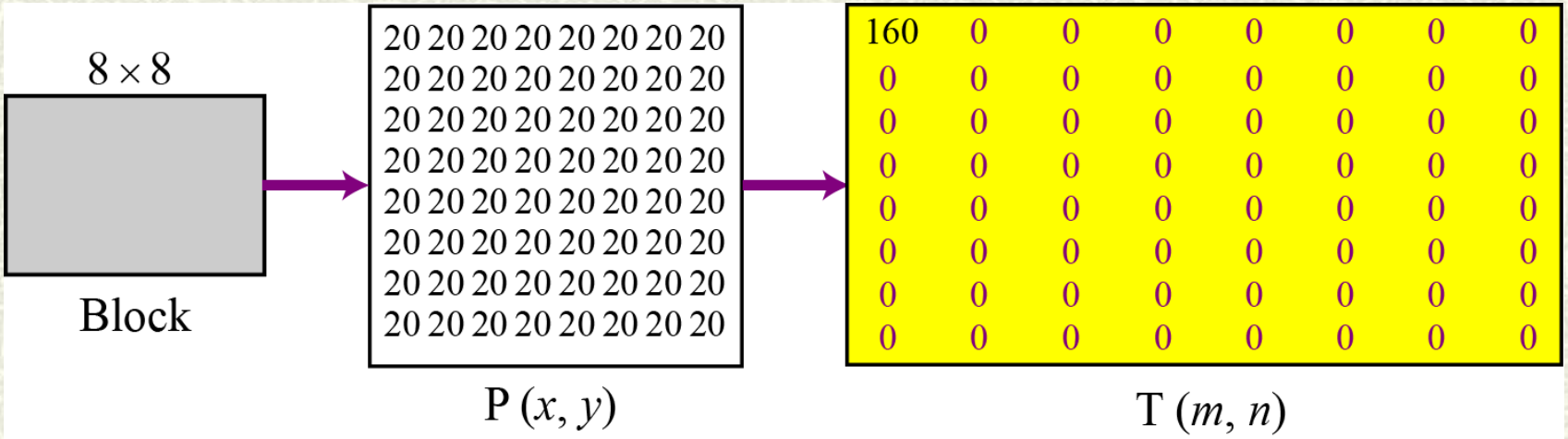
$$C_u C_v = \frac{1}{\sqrt{2}} \; for \; u, v = 0$$

$$C_u C_v = 1 \; otherwise$$

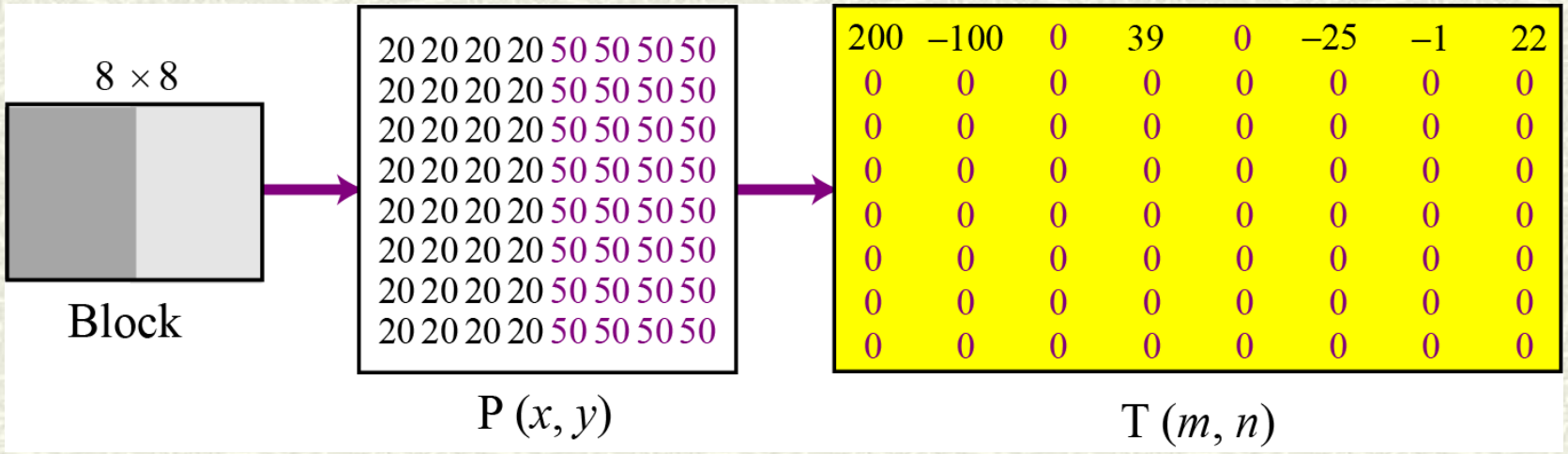
Step 3: DCT

- The color amplitude information can be thought of as a wave (in two dimensions).
- You're decomposing the wave into its component frequencies.
- For the 8 X 8 matrix of color data, you're getting an 8 X 8 matrix of coefficients for the frequency components.

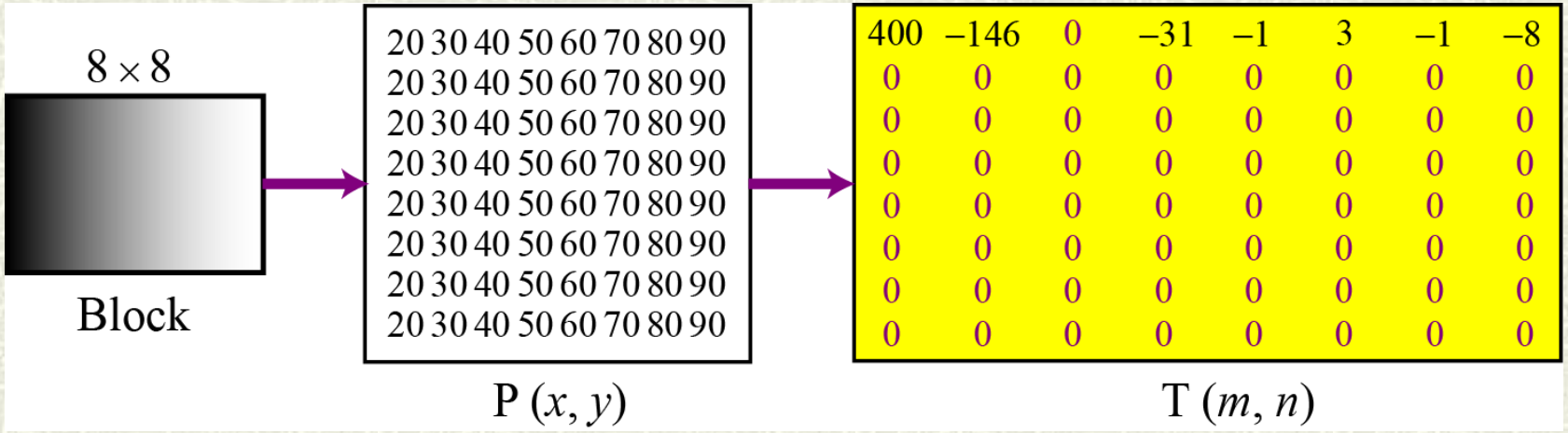
To understand the nature of this transformation, let us see the result of the transformations for three cases



Case 1: uniform grayscale



Case 2: two sections



Case 3: gradient grayscale

Step 4: Quantize the Coefficients Computed by the DCT

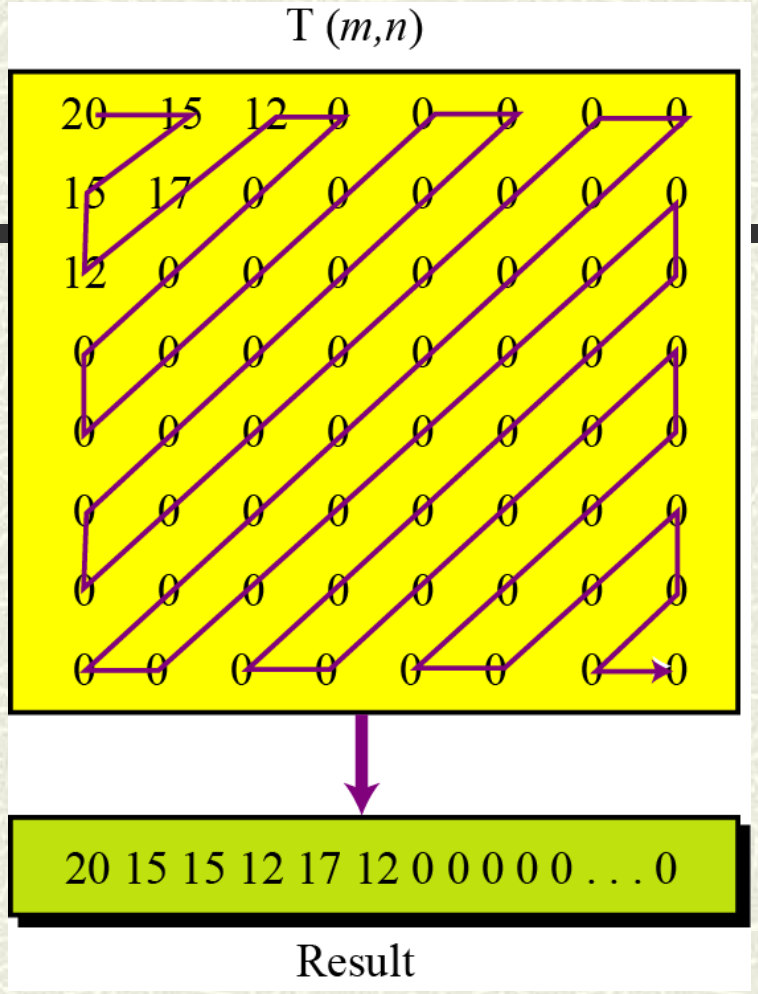
- The DCT is lossless in that the reverse DCT will give you back exactly your initial information (ignoring the rounding error that results from using floating point numbers.)
- The values from the DCT are initially floating-point.
- They are changed to integers by quantization.

Step 4: Quantization

- Quantization involves dividing each coefficient by an integer between 1 and 255 and rounding off.
- The quantization table is chosen to reduce the precision of each coefficient to no more than necessary.
- The quantization table is carried along with the compressed file.

Step 5: Arrange in "zigzag" order

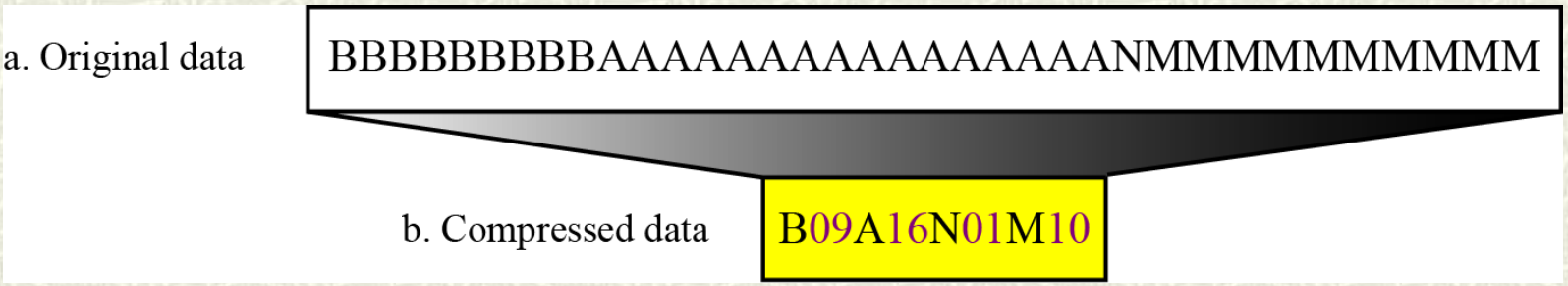
- This is done so that the coefficients are in order of increasing frequency.
- The higher frequency coefficients are more likely to be 0 after quantization.
- This improves the compression of run-length encoding.
- Do run-length encoding and Huffman coding.



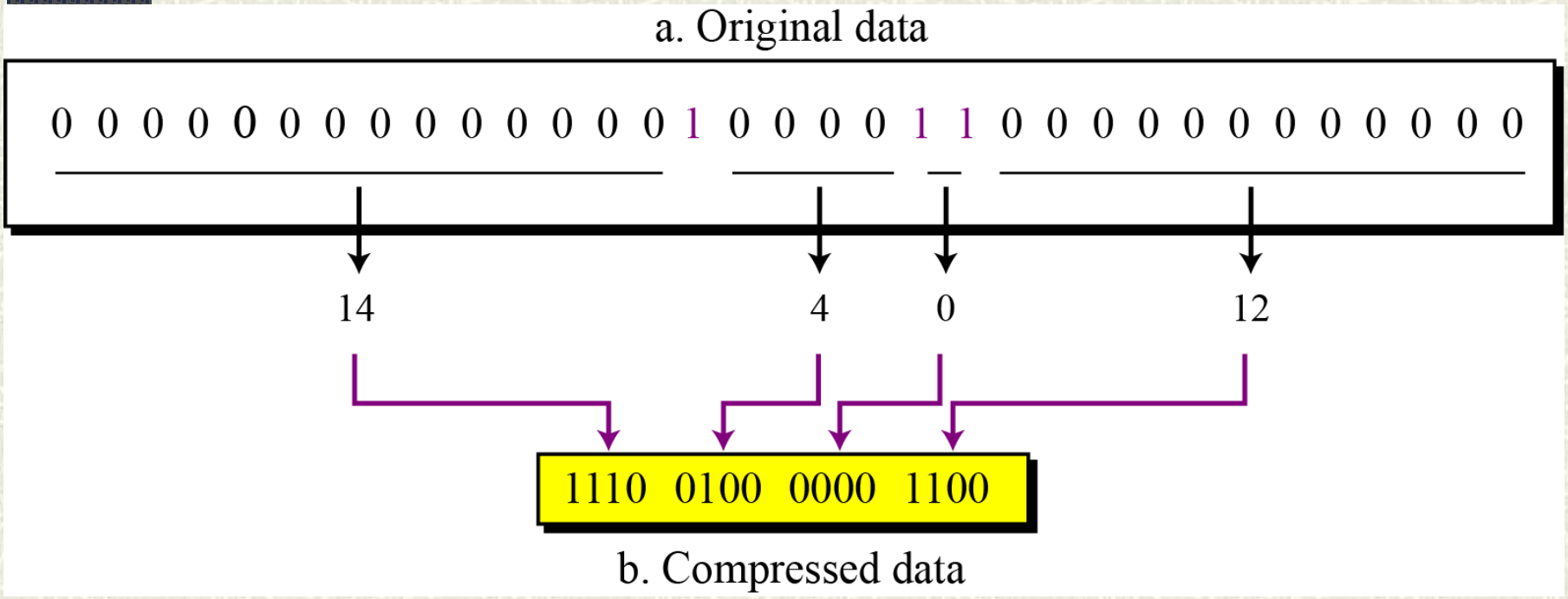
Reading the table

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.



Run-length encoding example



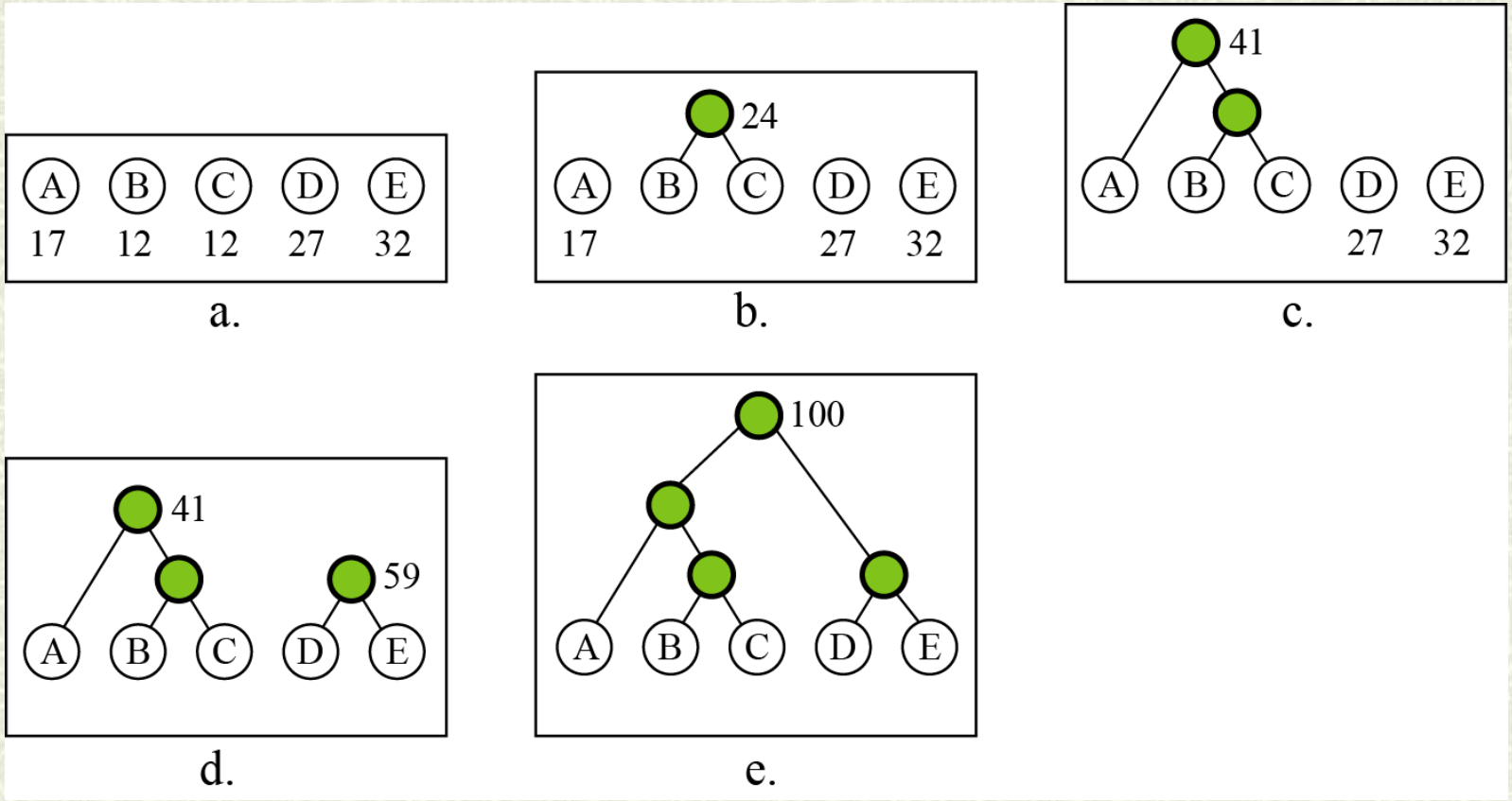
Run-length encoding for two symbols

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



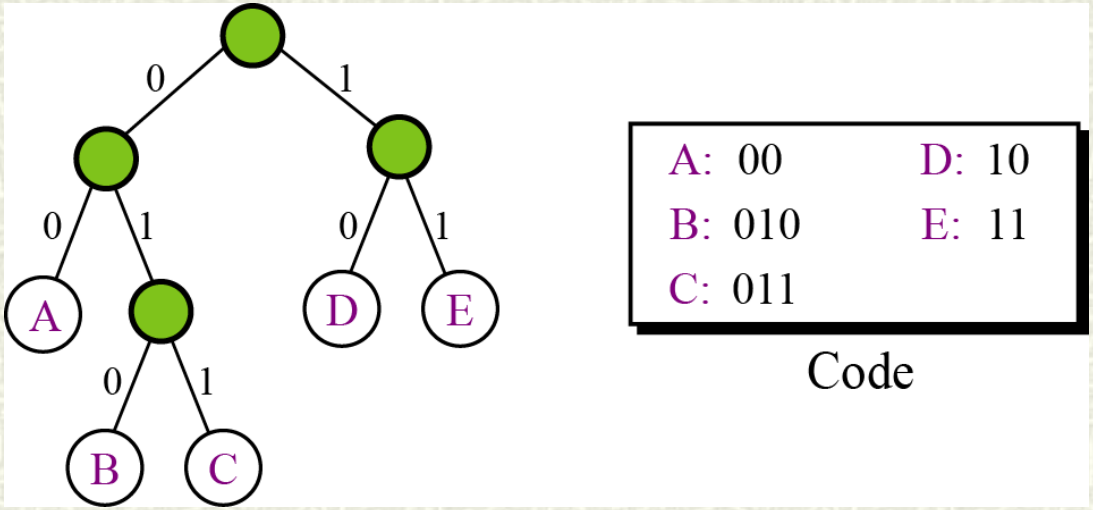
Huffman coding

Construction of the Huffman tree

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority:

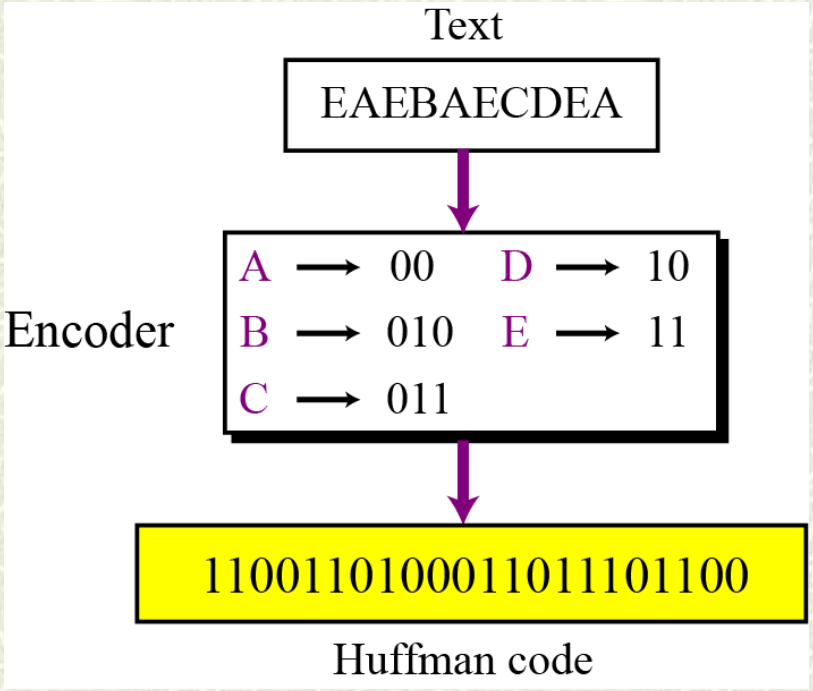
1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
   a. Remove the two nodes of highest priority (lowest probability) from the queue
   b. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
   c. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

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.



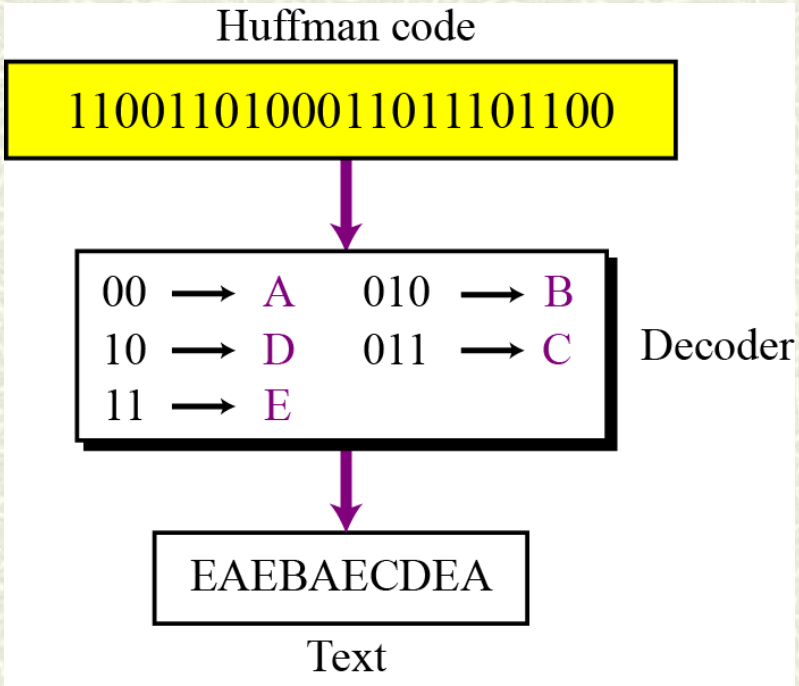
Final tree and code

Encoding



Huffman encoding

Decoding



Huffman decoding