Compression

Outline

Compression

- The goal of image compression is to reduce storage size and transmission time by representing an image using fewer bits than what would be required otherwise

Compression

- Images and videos take up a huge amount of storage space.
- A raw uncompressed image taken by a 5-megapixel camera requires

$5 \times 10^6 \times$ 24 bits per pixel = 120$\times 10^6$ bits = 120/8 megabytes = 15MB

Assume that you take 5 photographs per day then in a year you will accumulate 365 × 5 × 15 = 27375 MB ≈ 27.4 GB

- For a pleasant viewing experience videos normally have about 60 frames/second !

Basics

- **Compressor**: converts a stream of bits to a smaller stream of bits.
- This new stream is then either stored as a file on disk or transmitted across a network, where on the other end a **decompressor** restores the original image.
- The compressor and decompressor are known as a ***coder*** and ***decoder***, respectively.

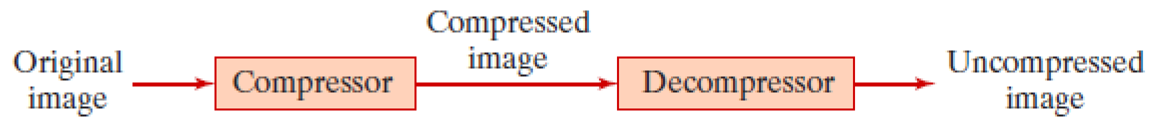


Figure 8.1 In a typical compression / decompression system, an image is first compressed, then the compressed image is either stored or transmitted, after which it is decompressed to yield either the original image (in the case of lossless compression) or an approximation to it (in the case of lossy compression).

Basics

- In **lossless compression**, the restored image is *exactly* the same as the original image, so that no information has been lost.
- Lossless compression is utilized in many common file formats like RAR, ZIP, GIF, PNG
- The image restored by **lossy compression** is only *similar* to the original image.
- Lossy compression is used in JPEG, TIFF, MPEG, MP3

Basics

- An important characteristic of the efficiency of the compression process is the **compression ratio**
  - It is defined as the ratio of the number of bits used to store the original image to the number of bits used to store the compressed image.

$$\text{compression ratio} \equiv \frac{\text{number of bits in uncompressed image}}{\text{number of bits in compressed image}}$$

- Modern compression algorithms typically achieve compression ratios on the order of 5:1 (five to one), 10:1, or even 100:1 !
- Compression algorithms exploit redundancy in data to achieve better compression ratios

Redundancy in images

- Three types of redundancy in images.
- **Coding redundancy:** refers to the fact that not all bit patterns (usually called *symbols*) are equally likely (*e.g.,* In a bright 8-bit image most intensities are closer to 255).

  Intensity values that appear more frequently should be coded with a smaller number of bits.

- **Spatial redundancy:** refers to correlation between pixels.

33	33	34	33
33	62	62	62
62	62	145	145

We can use previous pixel values to encode new intensities. The first row for example can be encoded as 33, 0 , 1, 0

Redundancy in images

- **Psychovisual redundancy**: Not all errors in data are noticeable to a human observer.

Example

$$\begin{matrix} 32 & 33 & 254 & 255 \\ 33 & 32 & 255 & 254 \\ 32 & 33 & 255 & 254 \end{matrix}$$

- When this image is displayed focus will be on the vertical edge not the subtle variations in intensity

Redundancy in images

Lossless compression utilizes coding and spatial redundancy. Lossy compression additional saving advantage comes from utilizing Psychovisual redundancy.

Information Theory

- The branch of mathematics underlying lossless compression is known as **information theory**.
    - Provides the tools necessary to answer questions such as how to determine the theoretical lower limit of the amount of data needed to encode a certain amount of information.

Entropy

- The entropy of *X* is given by the negative of the weighted average of the logarithm of the probabilities

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

- Entropy is often measured in the unit of bits and the used log. is with base 2
- The convention is to define 0 log 0 = 0.

Entropy

- Entropy can be viewed as the amount of new information that can be obtained (surprise).

Example: Find the entropy of

(a) Flipping a fair coin.

$$H(X) = -(½ \log ½ + ½ \log ½) = 1 \text{ bit.}$$

(b) Flipping a coin that has heads on both sides

$$H(X) = - (1 \log 1 + 0) = 0 \text{ bit.}$$

(c) Flipping a fair 6-sided die

$$H(X) = -6 (-1/6 \log 1/6) \approx 2.585 \text{ bits.}$$

Entropy

Example: Compute the entropy of the following 3-bit image

$$\begin{matrix} 7 & 3 & 2 & 0 \\ 0 & 2 & 3 & 1 \\ 7 & 2 & 3 & 3 \\ 2 & 3 & 5 & 4 \end{matrix}$$

Answer:

Entropy=

$-\frac{2}{16}\log_2\frac{2}{16}-\frac{1}{16}\log_2\frac{1}{16}-\frac{4}{16}\log_2\frac{4}{16}-\frac{5}{16}\log_2\frac{5}{16}-\frac{1}{16}\log_2\frac{1}{16}-\frac{1}{16}\log_2\frac{1}{16}-0-\frac{2}{16}\log_2\frac{2}{16}$

≈ 2.52 bits

Source codes and code words

- A **source code** is a function that maps from values to **code words** which are bit strings.

7	3	2	0
0	2	3	1
7	2	3	3
2	3	5	4

- One possible source code for the above image is

$$\varphi(0) = 000, \varphi(1) = 001, \varphi(2) = 010, \varphi(3) = 011,$$
$$\varphi(4) = 100, \varphi(5) = 101, \varphi(6) = 110, \varphi(7) = 111.$$

This code uses a fixed codeword 3 bits per pixel.

Source codes and code words

7	3	2	0
0	2	3	1
7	2	3	3
2	3	5	4

$$\varphi(0) = 000, \varphi(1) = 001, \varphi(2) = 010, \varphi(3) = 011,$$
$$\varphi(4) = 100, \varphi(5) = 101, \varphi(6) = 110, \varphi(7) = 111.$$

- We have 16 pixels, so using the above source code 16 x 3 = 48 bits are needed to store the image:

111 011 010 000 000 010 011 001 111 010 011 011 010 011 101 100

7 3 2 0 0 2 3 1 7 2 3 3 2 3 5 4

Shannon’s source coding theorem

- Given a sequence of symbols assumed as independent and identically distributed (i.i.d) the average codeword length per symbol can be no less than the entropy

Huffman coding

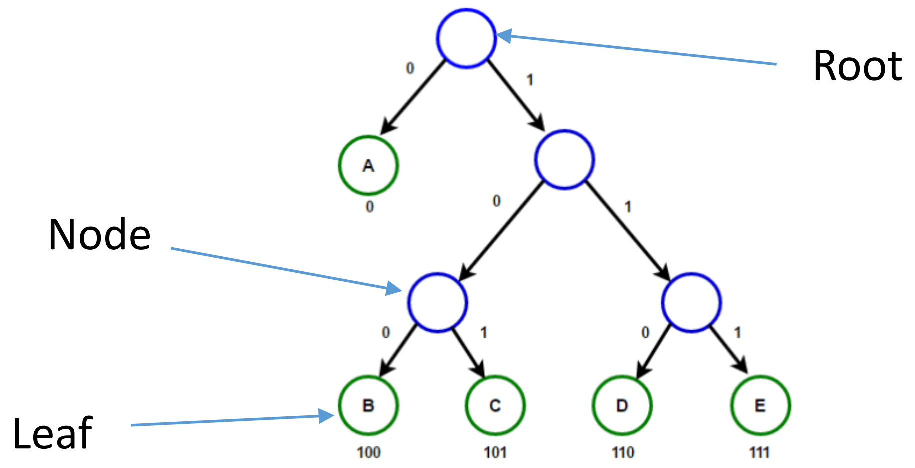
- A popular and simple algorithm that is proven to generate minimum expected codeword length assuming that the symbols are independent from each other.
- Used in many compressed file formats.
- It's effective in dealing with coding redundancy
- Generate the source code for the following image using Huffman coding

7	3	2	0
0	2	3	1
7	2	3	3
2	3	5	4

Huffman coding example

We build the tree from the leafs to the root
Except at the leaf node, each node is a parent of **two nodes**.

Terminology:



Each leaf node represents a symbol we like to encode

Huffman coding example

- Sample image

$$\begin{matrix} 7 & 3 & 2 & 0 \\ 0 & 2 & 3 & 1 \\ 7 & 2 & 3 & 3 \\ 2 & 3 & 5 & 4 \end{matrix}$$

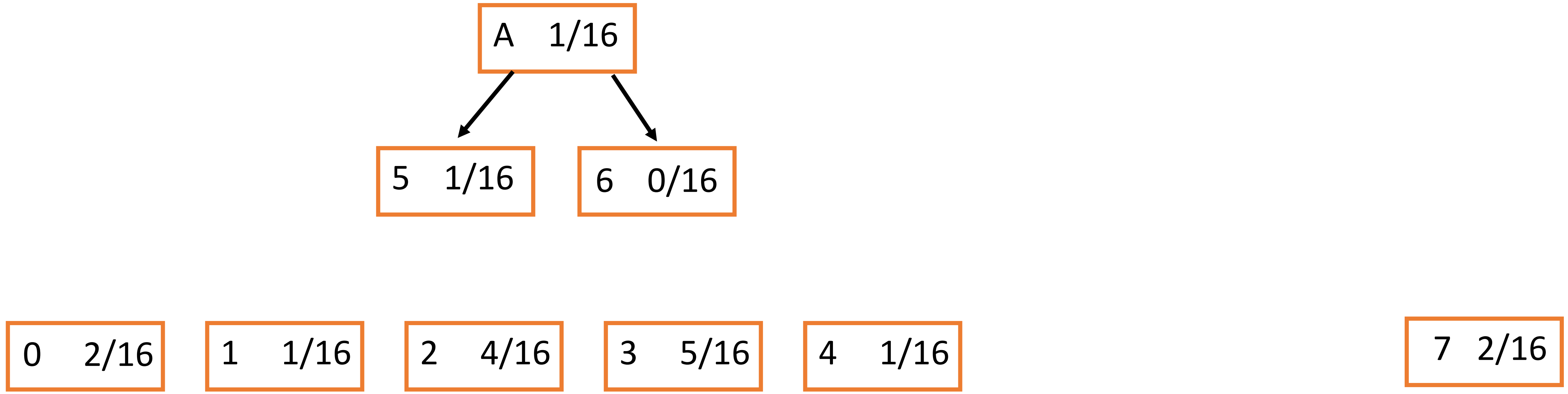
- Leaf nodes

0 2/16	1 1/16	2 4/16	3 5/16	4 1/16	5 1/16	6 0/16	7 2/16

20

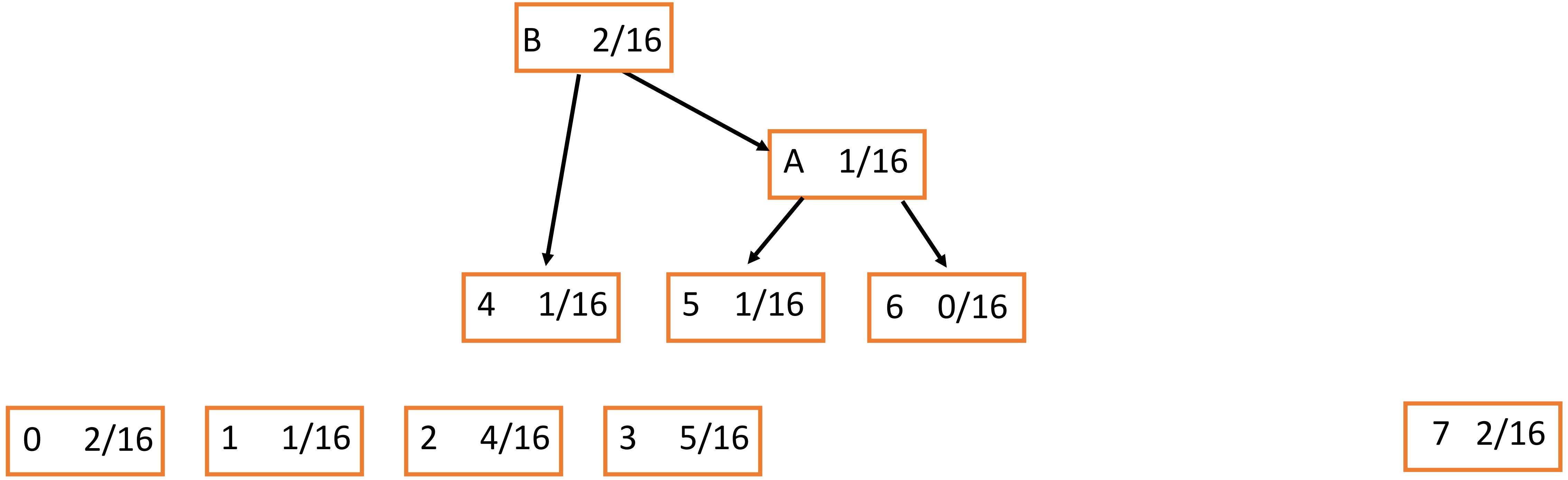
In each step of the algorithm combine the two leaves with the lowest probabilities. Leaf 6 can combine with either 5 or 4 or 1. We will choose 5

Huffman coding example



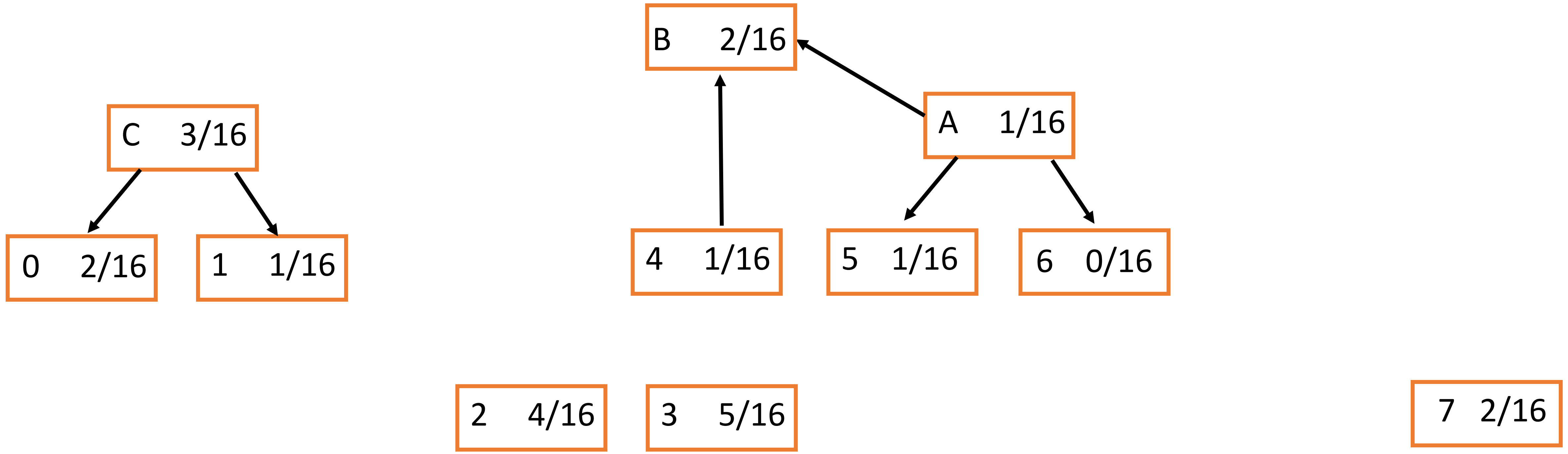
Symbols 4, 1, and A have the lowest probability 1/16. We choose to combine A with 4

Huffman coding example



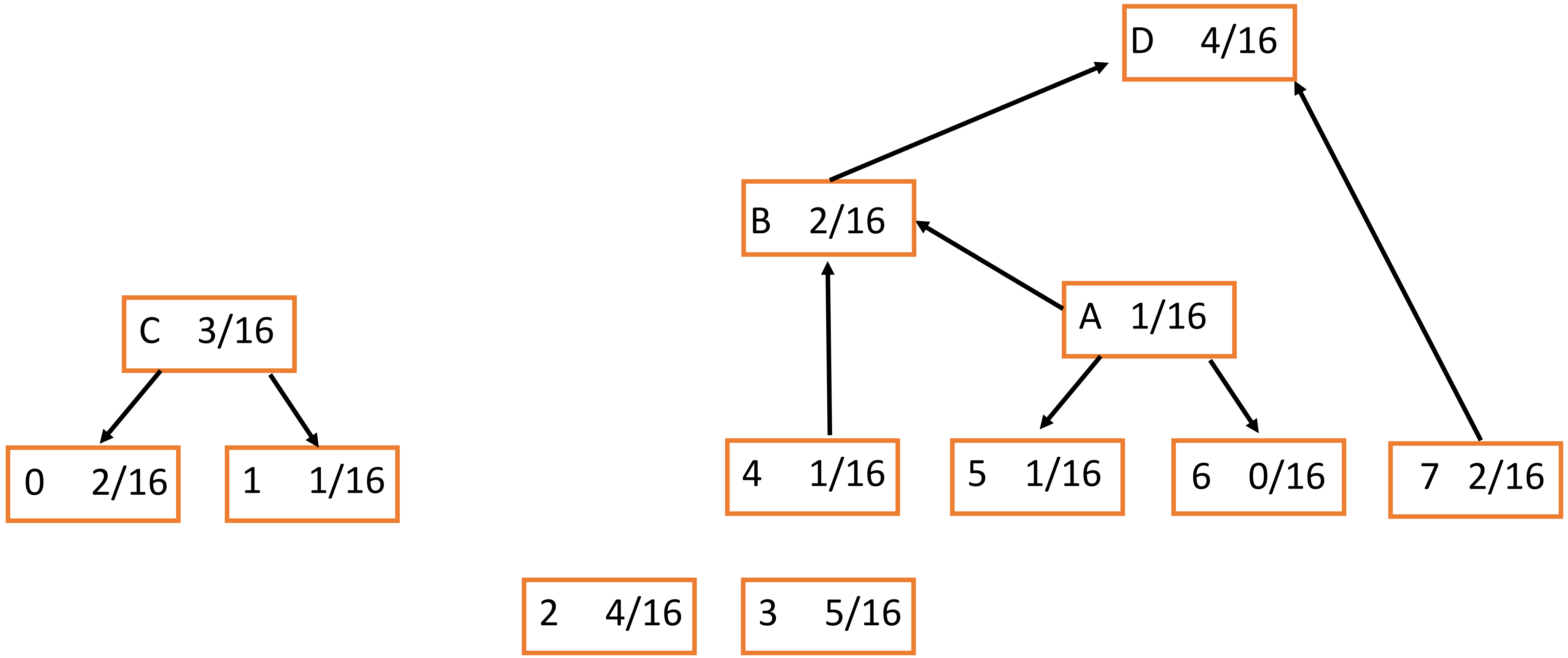
Now the minimum is 1/16 (leaf 1), 2/16 (leaf 0, leaf 7, and leaf B), we combine nodes 1 and 0

Huffman coding example



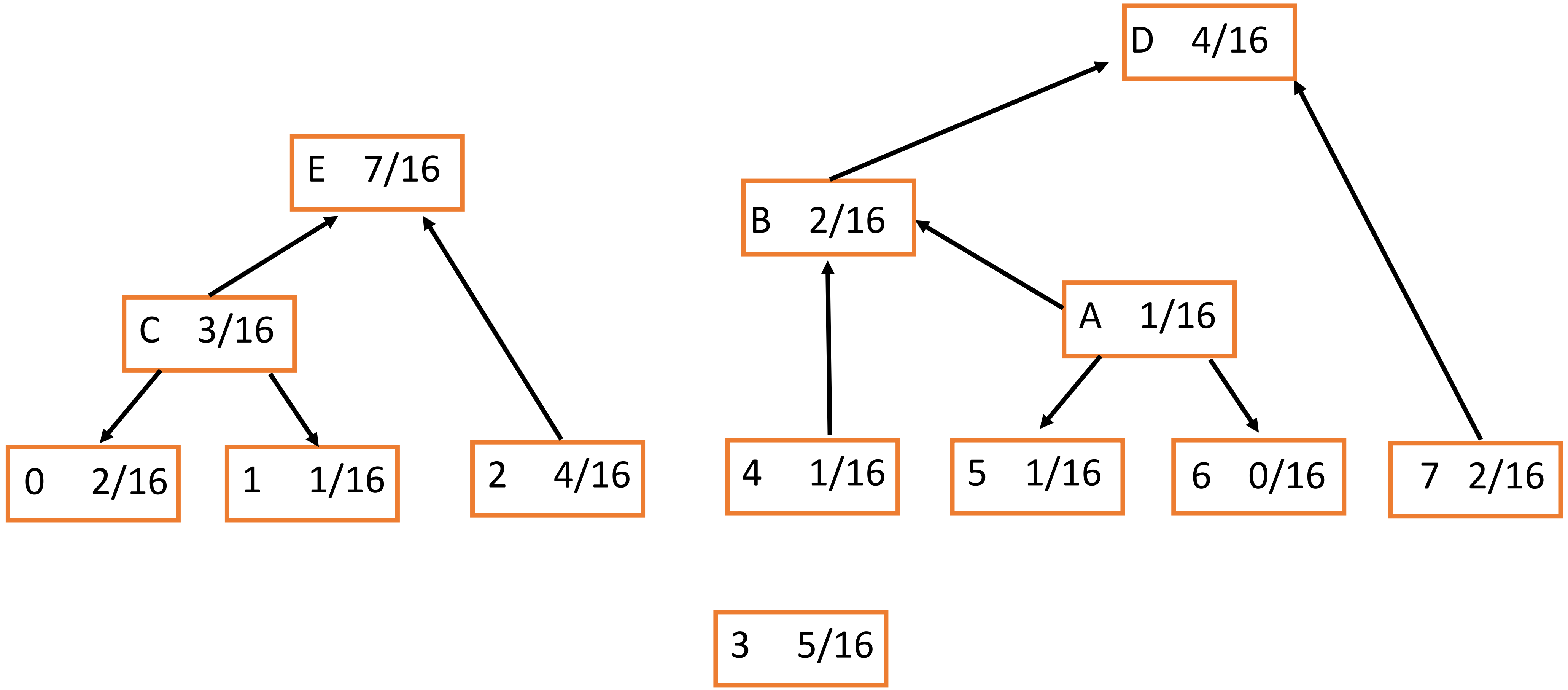
Now the minimum is 2/16 (leaf 7 and node B) and we combine them together.

Huffman coding example



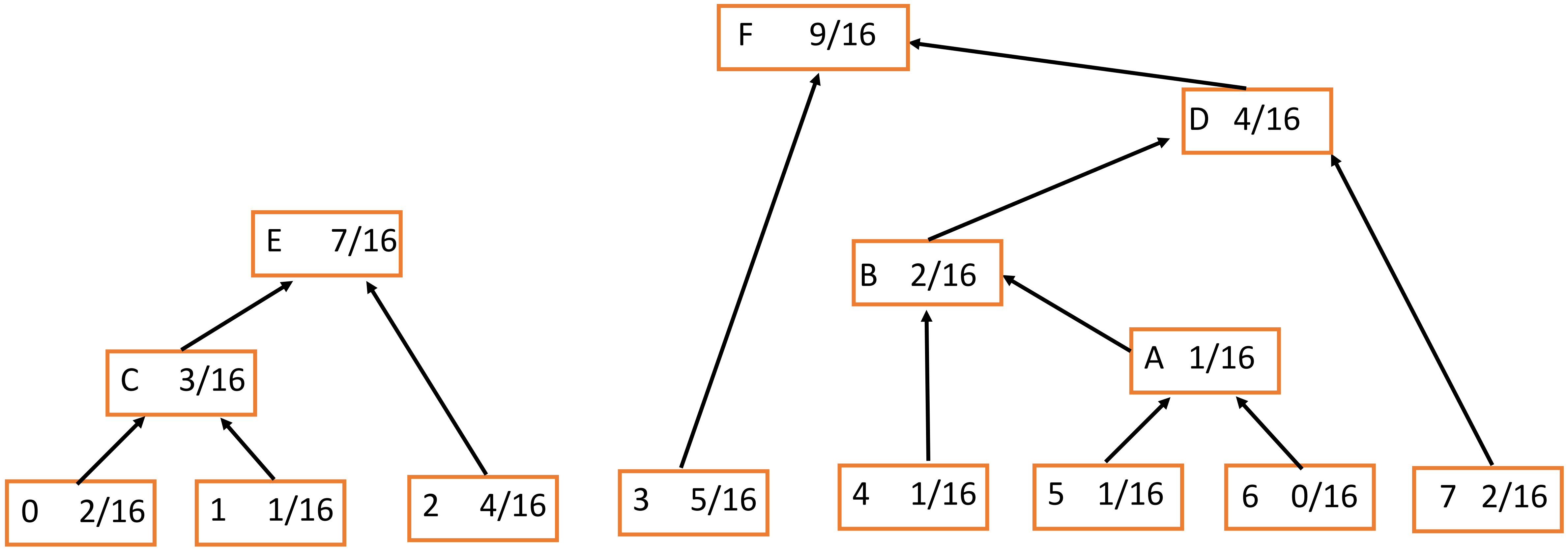
Now the minimum two leaves are 3/16 and 4/16 (node C, leaf 2 and leaf D) and we combine leaf C and leaf 2 together.

Huffman coding example



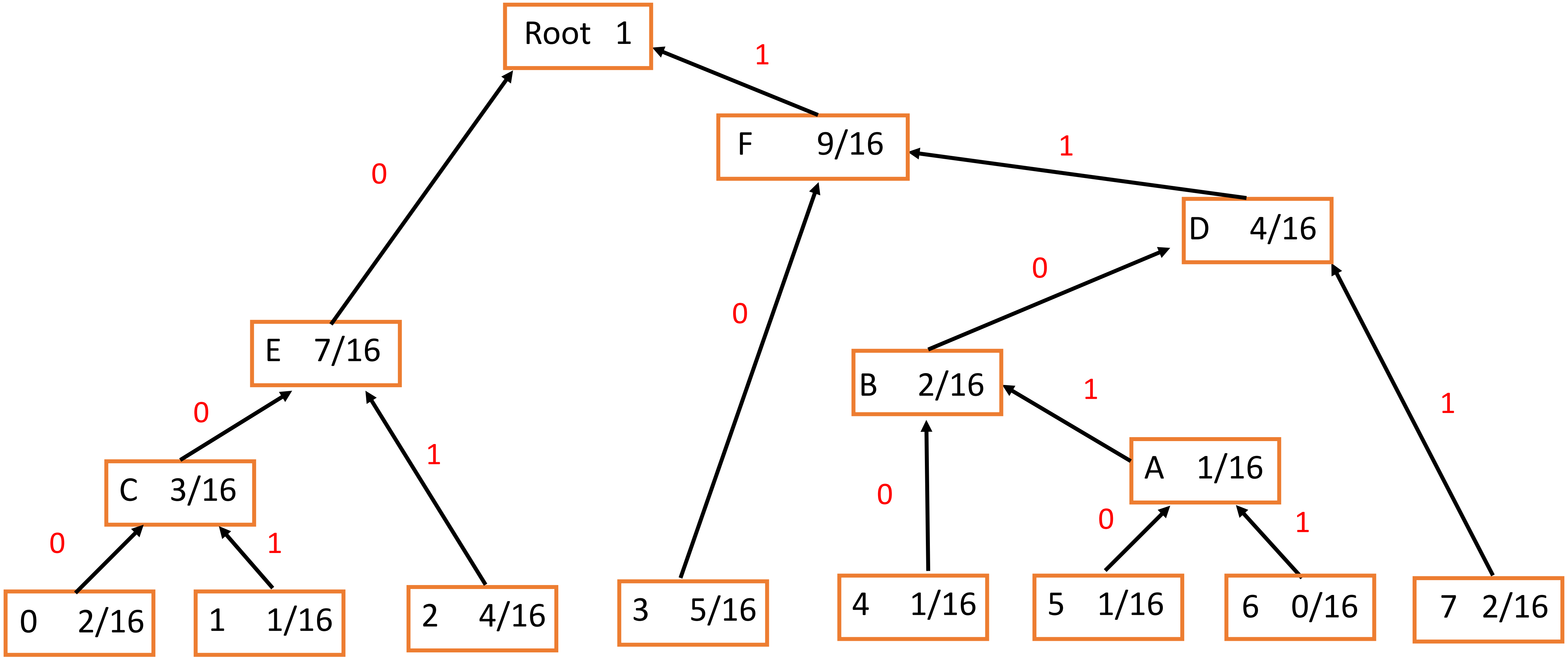
Now the minimum is 4/16 and we combine node D and 3

Huffman coding example



Finally, we combine the remaining nodes E and F

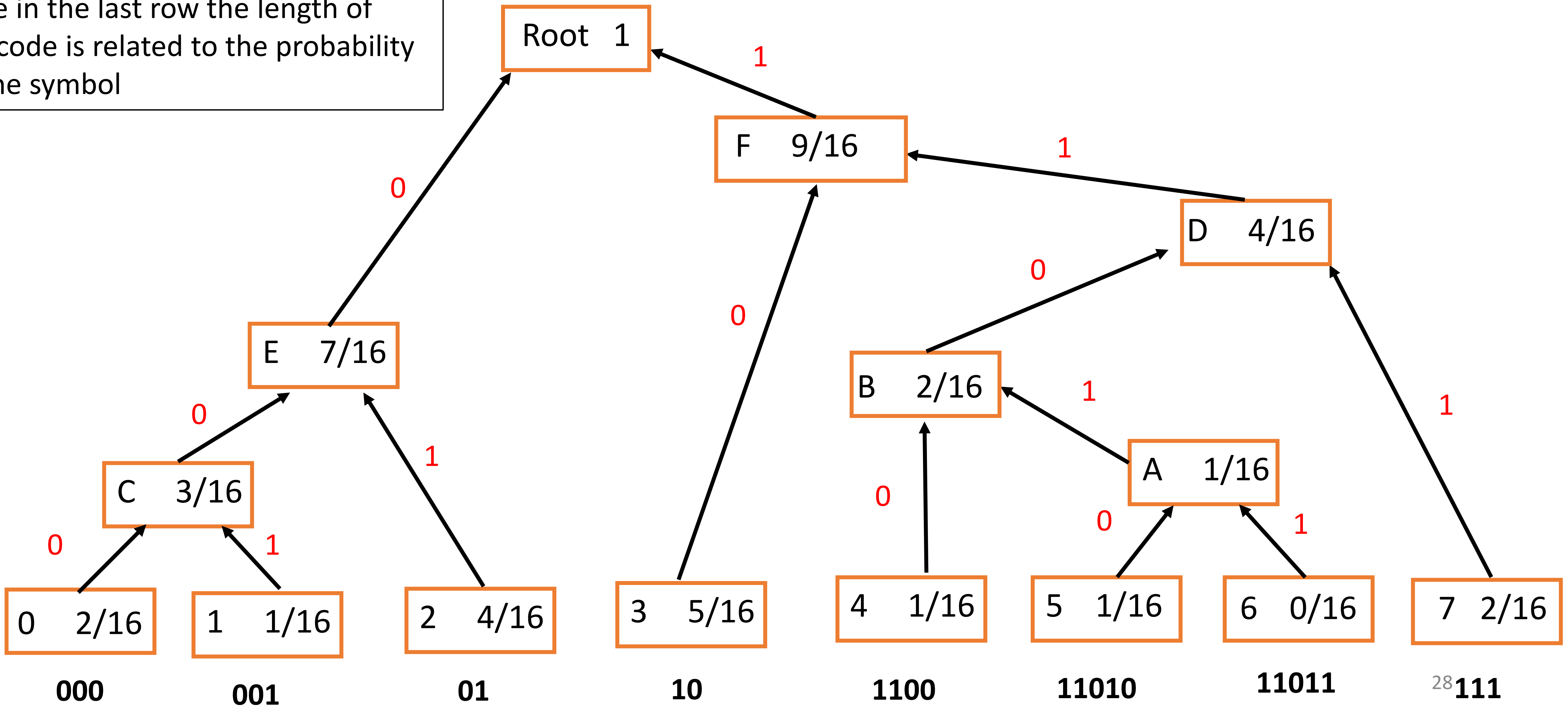
Huffman coding example



The binary nodes 1 and 0 are assigned to the two branches out of parent nodes. Reading from the root to the leafs reveals the source code

Huffman coding example

Note in the last row the length of the code is related to the probability of the symbol



Huffman coding

$$\begin{matrix} 7 & 3 & 2 & 0 \\ 0 & 2 & 3 & 1 \\ 7 & 2 & 3 & 3 \\ 2 & 3 & 5 & 4 \end{matrix}$$

Representing this image using the computed Huffman table we get

111 10 01 000 000 01 10 001 111 01 10 10 01 10 11010 1100

7 3 2 0 0 2 3 1 7 2 3 3 2 3 5 4

Using Huffman code the image can be represented in 42 bits.

Compression ratio : 48/42= 1.14

Huffman coding

- Shannon’s theorem: average code word length can’t be lower than entropy for iid signals

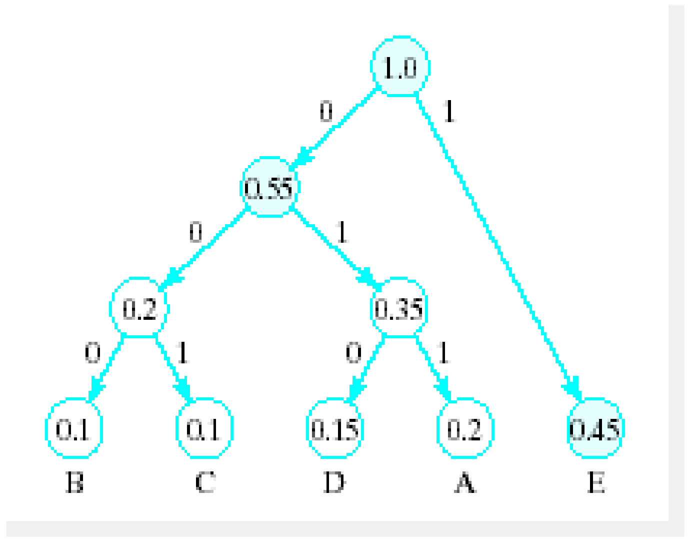
$$\begin{matrix} 7 & 3 & 2 & 0 \\ 0 & 2 & 3 & 1 \\ 7 & 2 & 3 & 3 \\ 2 & 3 & 5 & 4 \end{matrix}$$

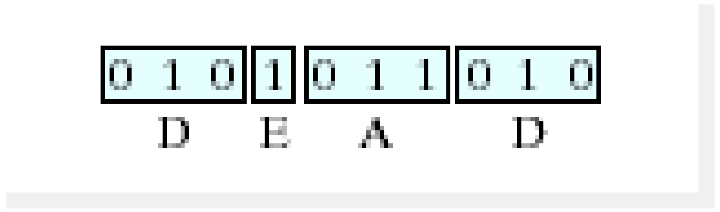
Entropy= 2.52 bits

Average code word length per symbol= 42/16= 2.63

Huffman decoding example

- Decode the following message **`0101011010`** using the following tree





Read the tree from the root until you get to the leaf. Record the letter and repeat the process.

Huffman coding

- A powerful algorithm that is used in several image and data compression formats
- Requires including the coding table or the probabilities of different intensities
- Requires two passes over data. One to compute the probabilities and another one to perform encoding

Lempel-Zif (LZ, LZ-77) encoding

- LZ encoding encodes symbols on the fly as the data are compressed.
- It’s the basis of Portable Network Graphics (PNG) file format.
- One pass over data is sufficient.

Suppose we want to encode the following sequence of bits:

AABABBBABAABABBBABBABB

These can represent a row in a binary image A=0, B=1.

Lempel-Zif (LZ) encoding

AABABBBABAABABBBABBABB

- Start with the first letter A. We have not seen it before, so it forms a new symbol

A|ABABBBABAABABBBABBABB

- The second letter A was seen before, so we skip it and stop at AB

A|AB|ABBBABAABABBBABBABB

- The next phrase we haven't seen is ABB, as we've already seen AB

A|AB|ABB|BABAABABBBABBABB

Lempel-Zif (LZ) encoding

- You keep repeating the process until you get to the end of the data stream

A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB

1 2 3 4 5 6 7 8 9

- The last phrase on the end is a repeated one. That’s OK, as we did reach the end of the sequence.
- The above sequence defines a dictionary with 9 elements. Each phrase is encoded as a number and a symbol to be appended to the number.

Lempel-Zif (LZ) encoding

1	2	3	4	5	6	7	8	9
A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
0A	1B	2B	0B	2A	5B	4B	3A	7

- Symbols in the third row are used to encode the data. They represent sequences by a number called dictionary index and a symbol to be added next.
- Assume that I want to compress symbol 8. I present it as 3 appended with a A.
- As the sequence gets longer, dictionaries will be lengthy reducing the number of stored bits.
- LZ encoding can be viewed as a search algorithm comparing new entries with previously seen entries.
- Different methods are used to encode the resulting sequence of dictionary elements and unrepeated data.

PNG files: Combining Lempel-Zif (LZ) and Huffman coding

- PNG is a popular lossless image compression format.
- A variant of LZ coding is used to search for entries that previously seen. The output of this algorithm is a table of (length, distance) pairs where length is the length of the match and distance is the distance between current location and the found location.
- A Huffman table is used to encode distances and unrepeated segments.
- Another Huffman table is used to encode lengths.
- The use of two tables because of the distribution of distances varies considerably from the distribution of lengths.

Predictive coding

- Huffman coding exploits coding redundancy. Symbols that occur more frequently are assigned shorter codes
- **Spatial redundancy** is common in images. Neighboring pixels often occupy intensities that are close to each other.
- Rung length encoding (RLE) is simple method to reduce spatial redundancy.

Rung length encoding (RLE)

- One of the easiest and simplest encoding method.
- Encode runs of symbols rather than symbols themselves

  bbbaaaddddcfffffffaaaaaddddd

  encoded as 3b3a5d1c7f5a5d
- Especially suitable for binary alphabet

  001111111000000011011111100000

  encoded as 2,7,7,2,1,6,5
- Symbols and the run lengths can be further shortened using Huffman coding.

Rung length encoding (RLE)

- Should RLE encoding be used to encode the following sequences:

(a) AAAABBBBBAAACCCCCCCBBB

Yes

(b) ABCBDAACDADCA

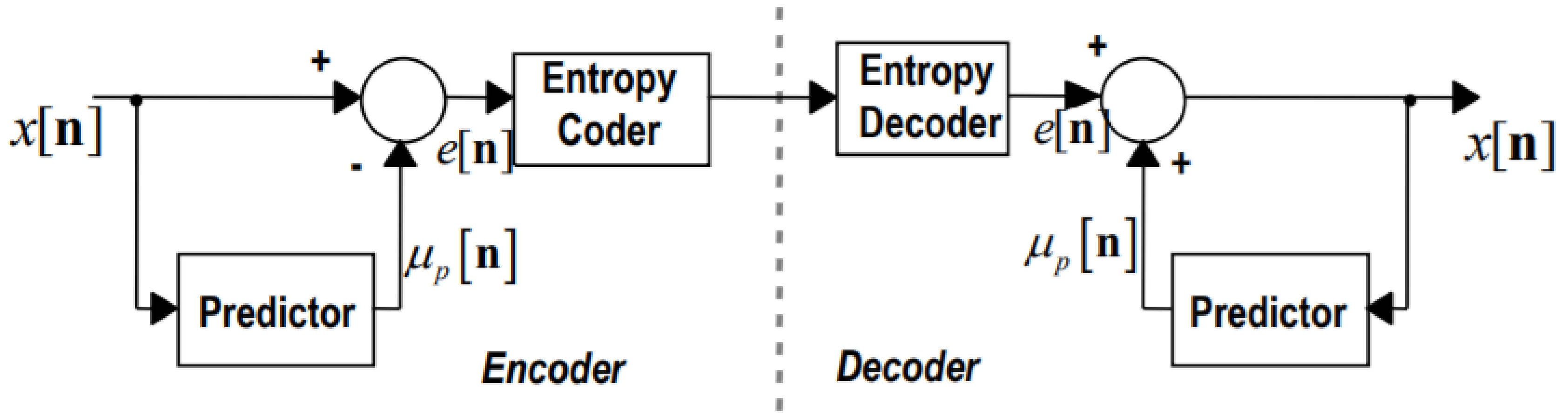
No

Predictive Coding

Predictive coding algorithms utilizes filters to predict pixel alues.

- A simple method to exploit spatial redundancy is to assume that each pixel value is equal to its earlier horizontal neighbor

Predictive Coding



- Prediction is computed from earlier seen samples of x[n].
- Prediction error is computed by subtracting the predictions μ[n] from x[n].
- Signal is reconstructed at the decoder by reversing the encoding process.

Predictive Coding

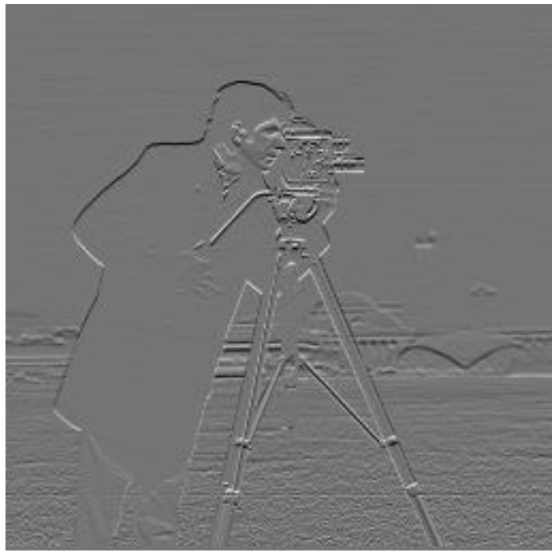
- Matlab example

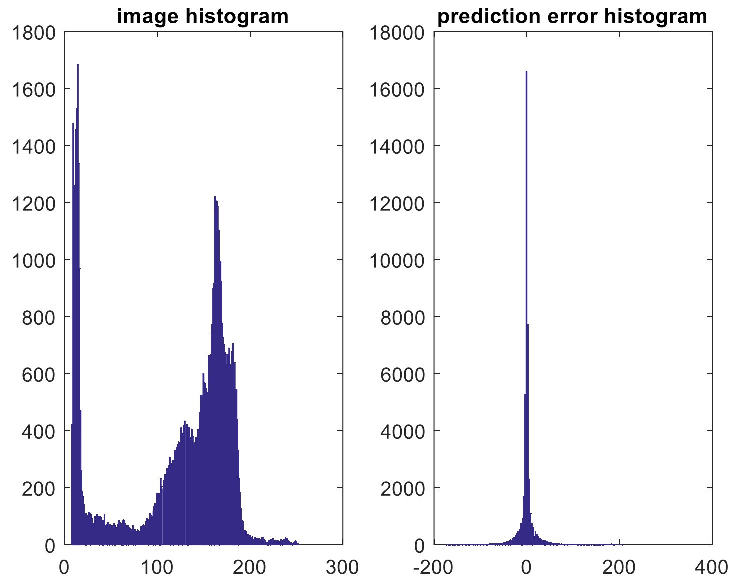
```
close all;
img=imread('cameraman.tif');
img=double(img);
h=[1 0 0]; %A horizontal filter that assigns the previous intensity value to each pixel
% h=[-1 1 0;1 0 0;0 0 0];  %You can try different 2D filters
prediction=imfilter(img,h);
[M,N]=size(img);
error=zeros(M,N);  %The error signal will be compressed and sent to the reciever
for i=1:M
    for j=1:N
        error(i,j)=img(i,j)-prediction(i,j);
    end
end

figure;imshow(error,[])
figure;
subplot(1,2,1);
hist(img(:),256); %img(:) converts the image to a 1D vector to compute the histogram
title('image histogram')
subplot(1,2,2);
hist(error(:),256); %error(:) converts the image to a 1D vector to compute the error histogram
title('error histogram')
```

Predictive Coding

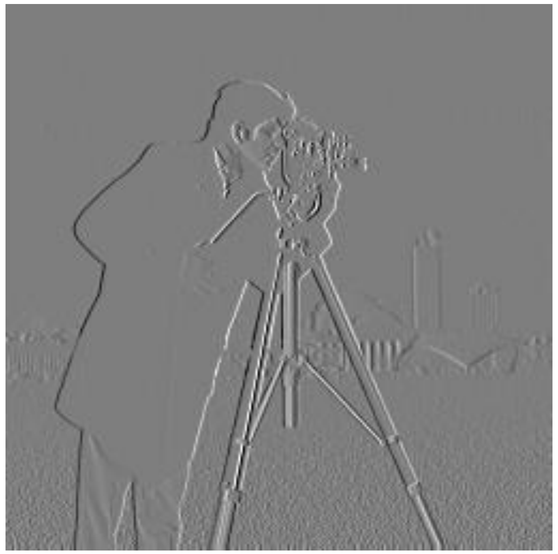
Predictor comparisons: $h = \begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$

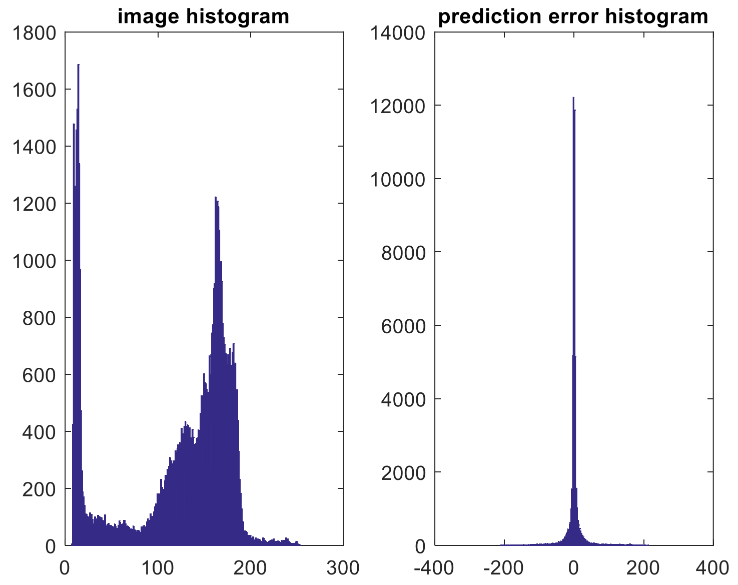




Predictive Coding

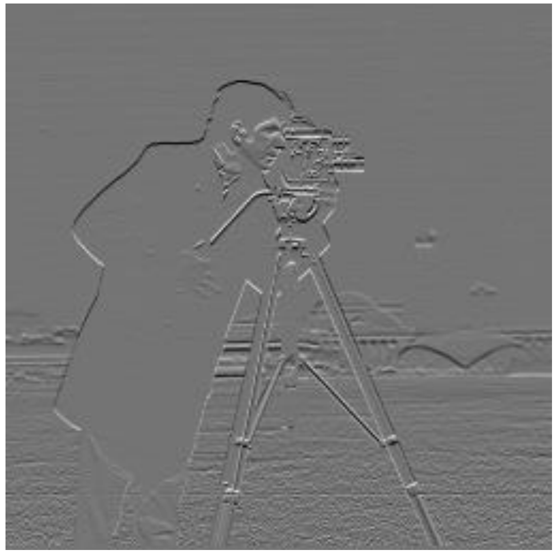
- Predictor comparisons: $h= \begin{matrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$

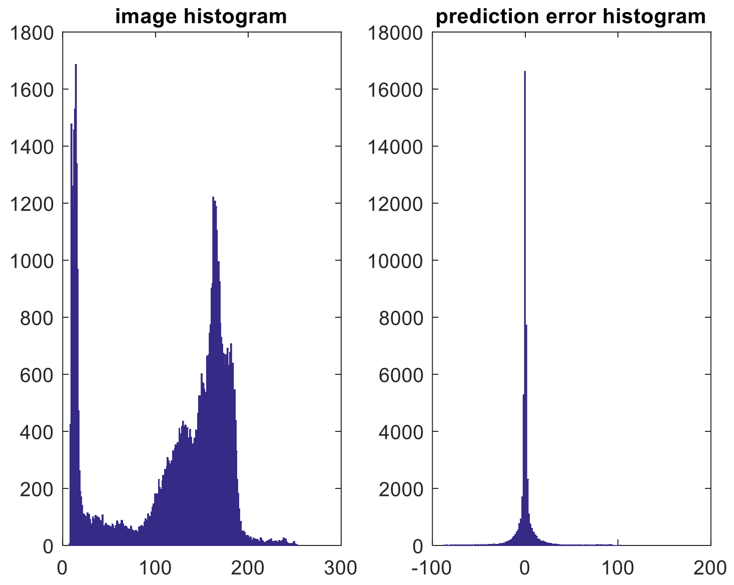




Predictive Coding

- Predictor comparisons: h= $\begin{matrix} 0 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$





Predictive Coding

- Prediction algorithms assume that we are scanning the image from upper left corner. We can’t use values to the right or below center pixel.
- Some of the common predictors used for predictive coding are:
  - The **planar predictor**

$$\mathrm{p} = \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

  - The **LOCO-I** (Low complexity lossless compression for images) predictor

$$h = median(\, l, u, p)$$

    where p is the planar predictor, l is the intensity value at point (x-1,y), u is the intensity value at point (x, y-1)

Predictive Coding

- **Paeth predictor**: The value of the pixel that is closest to the planar predictor among the three neighbors u, l, and d where d is the upper left diagonal pixel (x-1, y-1).

Predictive Coding

- Use the (a) planar predictor and (b) the LOCO-I predictor (c) the Paeth predictor to compute the lower-right pixel in the following image

$$\mathbf{I} = \begin{bmatrix} 25 & 26 \\ 32 & x \end{bmatrix}$$

Answer:

(a) Using the planar predictors x = 26+32-25=33

(b) Using the LOCO-I predictor x = median (26, 32, 33) = 32.

(c) Using the Paeth predictor x = 32 (the value closest to 33).

Predictive Coding

- The best predictive filter depend on image contents.
- Assume that a horizontal row in an image is composed of [1 2 …256].

  Its is optimal to encode this line using a horizontal prediction filter.
- PNG files 5 predictive filters. An algorithm is used to check for the best filter for each row in the image.
- These filters are: 1) the pixel to the left l 2) the pixel above u 3) rounded down average of u and l. 4) Paeth predictor 5) No prediction.
- Prediction error is then sent to the LZ coding algorithm described earlier in the slides.

Predictive Coding

Its desired to compress the following image using the planar predictor as predictive filter. Scan the image by rows from left to right. Upper left pixel is not compressed. Top row should be predicted using the previous pixel value. Values in the first column should be predicted using the previous upper pixel. Compute the error matrix.

$$\boldsymbol{I} = \begin{bmatrix} 25 & 26 & 28 \\ 32 & 30 & 29 \\ 32 & 30 & 22 \end{bmatrix}$$

p=28+30-26

Solution:

First we compute the prediction matrix: $\boldsymbol{I_{predict}} = \begin{bmatrix} 25 & 25 & 26 \\ 25 & 33 & 32 \\ 32 & 30 & 29 \end{bmatrix}$

The error matrix is given by: $\boldsymbol{E} = \boldsymbol{I} - \boldsymbol{I_{predict}} = \begin{bmatrix} 25 & 26 & 28 \\ 32 & 30 & 29 \\ 32 & 30 & 22 \end{bmatrix} - \begin{bmatrix} 25 & 25 & 26 \\ 25 & 33 & 32 \\ 32 & 30 & 29 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 7 & -3 & -3 \\ 0 & 0 & -7 \end{bmatrix}$

Predictive Coding

Use the error matrix you found in the previous problem to recover the original image $\boldsymbol{I}$. Assume that you know that the upper left pixel value is 25.

$$\boldsymbol{E} = \begin{bmatrix} 0 & 1 & 2 \\ 7 & -3 & -3 \\ 0 & 0 & -7 \end{bmatrix}$$

Answer

Previous horiz. value

Error

Reading from top row left to right. Upper left value is known as 25. We repeat the prediction process. Error is added.

$$\boldsymbol{I_{prediction}} = \begin{bmatrix} 25 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 25+1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 26 & 26+2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

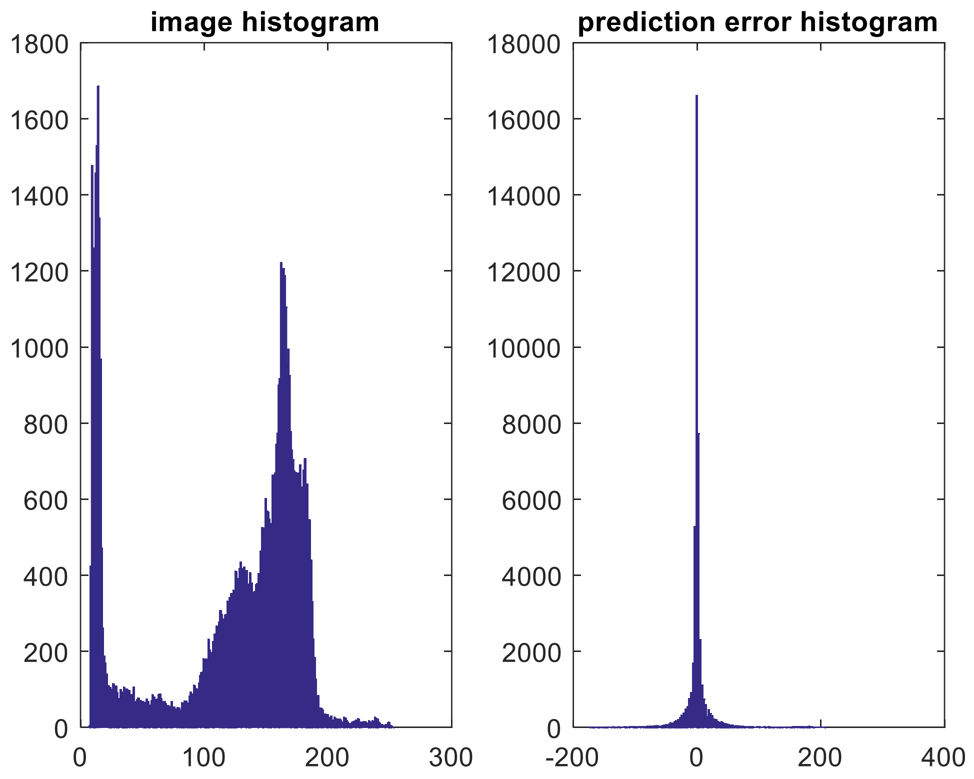
$$\rightarrow \begin{bmatrix} 25 & 26 & 28 \\ 25+7 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 26 & 28 \\ 32 & 33+(-3) & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 26 & 28 \\ 32 & 30 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 26 & 28 \\ 32 & 30 & 32+(-3) \\ 0 & 0 & 0 \end{bmatrix} \rightarrow$$

Planar predicted value

Error

$$\rightarrow \begin{bmatrix} 25 & 26 & 28 \\ 32 & 30 & 29 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \ldots\ldots \quad \rightarrow \begin{bmatrix} 25 & 26 & 28 \\ 32 & 30 & 29 \\ 32 & 30 & 22 \end{bmatrix} = \boldsymbol{I}$$

Encoding errors



Error distribution has a steep decaying curve. Variants of unary coding were found to work better than Huffman In these situations

Unary coding

- The unary code is given by

$$c_0 = \text{"0"} \quad c_1 = \text{"01"} \quad c_2 = \text{"001"} \quad c_3 = \text{"0001"} \quad \dots$$

- Assume that we have symbols that decay in probability in a geometric manner (1/2,1/4 , 1/8, 1/16,…)
- Unary code is mathematically found to be optimal for geometrically decaying sequences.

Unary coding

- Symbols encoded using unary encoding

Symbol	Unary code
0	0
1	01
2	001
3	0001
4	00001
5	000001
6	0000001
…	…

- Normally symbols don’t decay this fast. We need a way to shorten these codes. Unary codes can be very lengthy for frequently occurring symbols

Golomb-Rice coding

- The symbols are encoded using

$$x = mx_q + x_r \text{ with } x_q = \left\lfloor \frac{x}{m} \right\rfloor \text{ and } x_r = x\ mod\ m$$

where m is a multiple of two.

- x_q is encoded using unary coding
- x_r is encoded using binary coding with the following code length

$$k = \log_2 m$$

Golomb-Rice coding m=2

x_q in black

x_r in red

Symbol	Golomb coding m=2
0	00
1	01
2	010
3	011
4	0010
5	0011
6	00010
7	00011
8	000010
9	000011
10	0000010
11	0000011
12	00000010
13	00000011
14	000000010
15	000000011

m=2

Binary coded digit is of fixed length $\log_2 (2) = 1$

Golomb-Rice coding

Most frequent symbol

Image intensities occurring frequently are assigned smaller symbols

Uncommon image intensity values are assigned larger symbols

Least frequent symbol

Symbol	Golomb coding m=4
0	000
1	001
2	010
3	011
4	0100
5	0101
6	0110
7	0111
8	00100
9	00101
10	00110
11	00111
12	000100
13	000101
14	000110
15	000111

m=4

Binary coded digit is of fixed length $\log_2 (4) = 2$

Exercise

Consider the image below

$$\mathbf{I} = \begin{matrix} 5 & 1 & 3 & 2 \\ 4 & 1 & 3 & 4 \\ 0 & 3 & 3 & 6 \\ 7 & 6 & 1 & 7 \end{matrix}$$

a. How many bits are needed to represent this image?
b. Encode the image using Huffman encoding
c. Compute the compression ratio achieved.

Exercise

Lossy compression

- In **lossless compression**, the restored image is *exactly* the same as the original image, so that no information has been lost.
- Lossless compression is utilized in many common file formats like RAR, ZIP, GIF, PNG
- The image restored by **lossy compression** is only *similar* to the original image.
- Lossy compression is used in JPEG, TIFF, MPEG, MP3

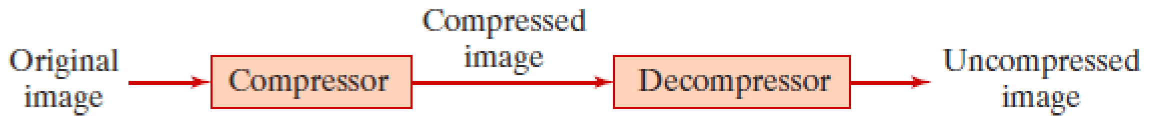


Figure 8.1 In a typical compression / decompression system, an image is first compressed, then the compressed image is either stored or transmitted, after which it is decompressed to yield either the original image (in the case of lossless compression) or an approximation to it (in the case of lossy compression).

Lossy compression

- Lossy compression generates images that are **not** identical to the original data samples. Variations between original data and the compressed data are designed to be not detectable by the casual observer (**Psychovisual redundancy**).
- A quality factor can control the level of degradation accepted.
- Using a low quality factor will produce smaller file sizes at the cost of visual quality.

High quality JPEG
File Size: 77.9 kb

Medium quality JPEG
File Size: 19.11 kb

How to assess compression quality

- Mean Squared Error (MSE)

$$MSE(A,B) = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}(A[m,n] - B[m,n])^2$$

- Root Mean Squared Error (RMSE)

$$RMSE(A,B) = \sqrt{MSE(A,B)}$$

$$RMSE(A,B) = \sqrt{\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}(A[m,n] - B[m,n])^2}$$

How to assess compression quality

- Peak Signal to Noise Ratio (PSNR)

$$PSNR(A,B) = 10\ \log_{10}\frac{MaxI^2}{MSE}$$

- Where *MaxI* is the maximum possible intensity value (e.g. 255 for 8-bit images)

Lossy compression

- Lossy compression methods often represent the image in a frequency domain.
- Using the Fourier transform we have seen that signal energy gets allocated in certain locations. The rest of the frequency domain contains values that are either zero or close to zero
- Encoding data using transforms is known as **transform coding**.

Original image

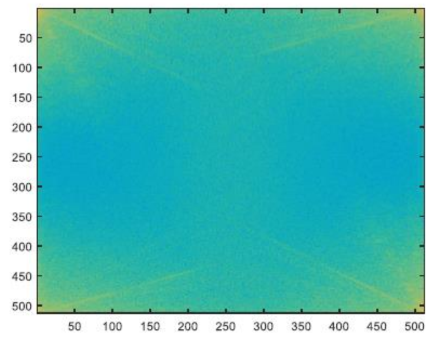


Image B log-magnitude spectrum

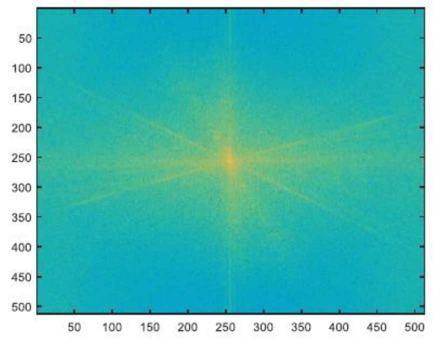


Image C log-magnitude spectrum

Transform coding

- **Transform coding**: the image is transformed to a different domain that better captures the information of the signal as it will be perceived.

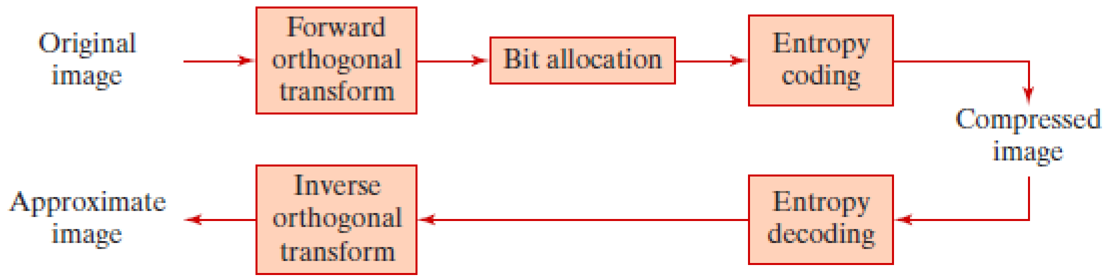


Figure 8.6 Transform coding typically involves three steps to compress an image: an orthogonal transform, bit allocation, and entropy coding. To decompress the image the steps are reversed, but since information is lost during bit allocation, the result is an approximation of the original image.

- **Orthogonal transforms** preserves signal energy. Parseval's theorem applies to these transforms. The use of orthogonal transforms allows one to use the inverse transform to reconstruct the original signal.

Transform coding

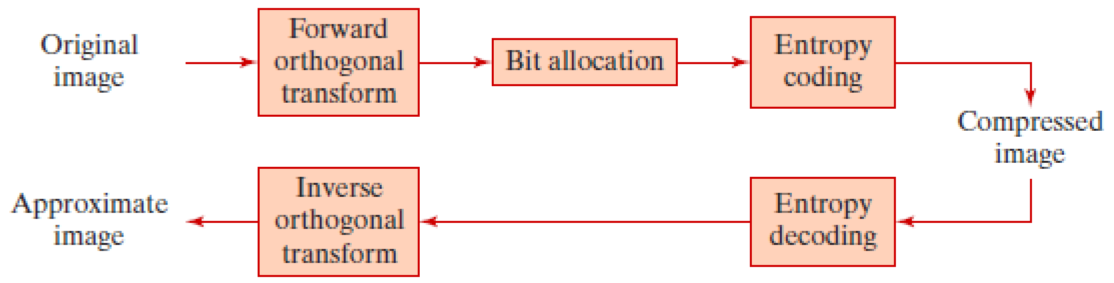


Figure 8.6 Transform coding typically involves three steps to compress an image: an orthogonal transform, bit allocation, and entropy coding. To decompress the image the steps are reversed, but since information is lost during bit allocation, the result is an approximation of the original image.

- The **bit allocation** step is a quantization and thresholding step that can discard fine details in the image. Rounding is an example of a quantization algorithm.
- **Entropy coding**: Huffman coding or a similar technique.

Lossy compression

- Fourier transform has ringing artifacts when thresholding coefficients (Similar to what was observed with ideal low pass filter).
- In addition, Fourier transform generates complex numbers which are not convenient to deal with.
- The discrete cosine transform (DCT) was designed to have properties similar to the Fourier transform but while generating real numbers.
- It’s an essential unit in the jpeg image compression format.

Ringing artifacts (Gibbs phenomenon)

- The allocation step (thresholding/quantization) will generate ringing artifacts if the Fourier transform was used as the orthogonal transform.

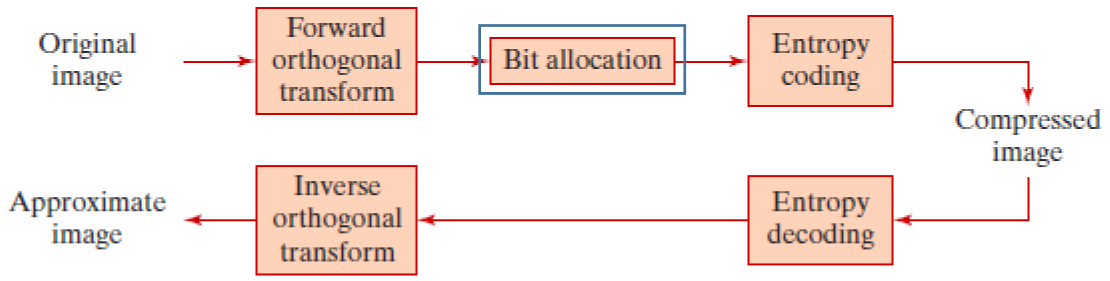
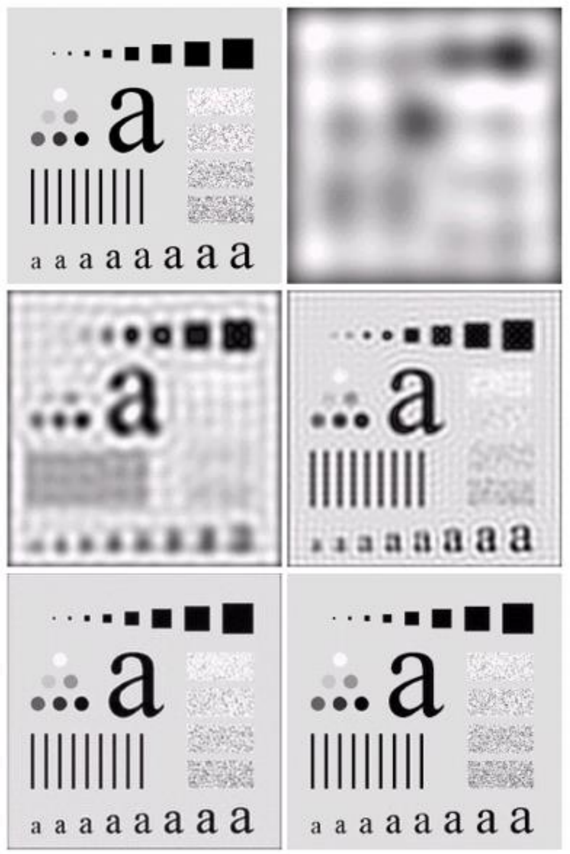


Figure 8.6 Transform coding typically involves three steps to compress an image: an orthogonal transform, bit allocation, and entropy coding. To decompress the image the steps are reversed, but since information is lost during bit allocation, the result is an approximation of the original image.



a b
c d
e f

FIGURE 4.12 (a) Original image. (b)–(f) Results of ideal lowpass filtering with cuto frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Th power removed by these filters was 8, 5.4, 3.6, 2, and 0.5% of the total, respectively.

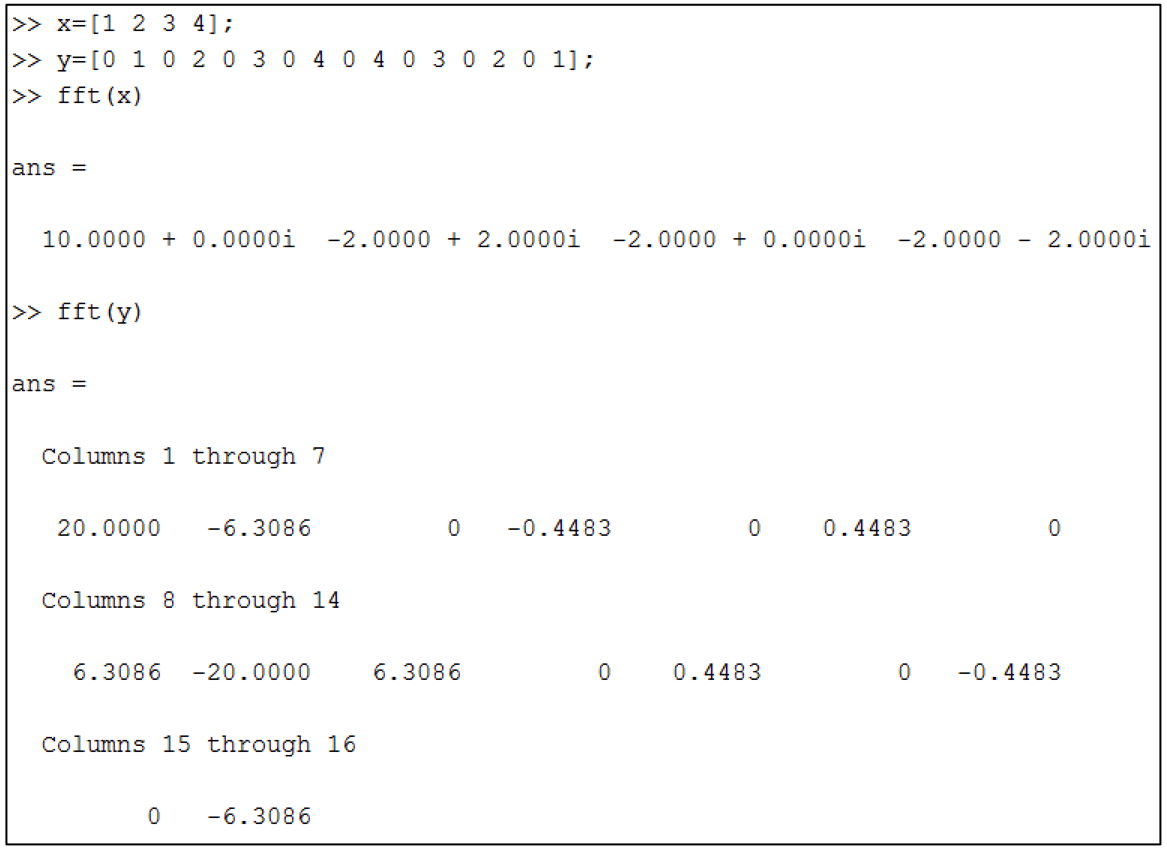
Discrete Cosine Transform (DCT)

- We like to design a frequency transform that generates real coefficients.
- The Fourier transform of a real symmetric signal is real. If we can re-organize the image so that its symmetric, its Fourier transform will be real.
- Different ways are possible to generate this symmetry. These different methods are used in different versions of the DCT.
- The most popular one and the one that is used in compression is DCT type 2. It is often referred to as the DCT.

Discrete Cosine Transform (DCT): the idea

- Assume that we like to generate a symmetric version of the following signal x = [1, 2 ,3 ,4]
- In the DCT-II, y = [0, 1, 0 , 2, 0, 3, 0, 4, 0, 4, 0, 3, 0, 2, 0, 1] signal is $\neq 0$ in odd indicies
- Note that the signal zeros are inserted between elements in the signal and in the first location
- y is 4 times as long as the original signal x
- DFT of y is real.

Discrete Cosine Transform (DCT)



```
>> x=[1 2 3 4];
>> y=[0 1 0 2 0 3 0 4 0 4 0 3 0 2 0 1];
>> fft(x)

ans =

  10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 + 0.0000i  -2.0000 - 2.0000i

>> fft(y)

ans =

  Columns 1 through 7

   20.0000   -6.3086         0   -0.4483         0    0.4483         0

  Columns 8 through 14

    6.3086  -20.0000    6.3086         0    0.4483         0   -0.4483

  Columns 15 through 16

         0   -6.3086
```

But y is now longer. How is this useful for compression?
Answer:
The first four values are sufficient to predict the remaining values. Other values can be inferred from the first four.

Discrete Cosine Transform (DCT)

- Applying the Fourier transform equation to the signal after the insertion of zero's in the manner described earlier, generates the DCT transform formula.

$$F(k) = \sum x(n)\, e^{-i2\pi\, k\, n/N} = \sum_{n=0}^{N-1} x(n) \cos \frac{(2n+1)\pi\, k}{2N} \text{ (DCT formula)}$$

- The sine functions cancel because of the symmetry. The result is a real function.
- Note the cosine function is applied at only odd indices. The even indices cancel as they are equal to zero.
- The summation goes from 0 to N-1 due to symmetry
- In summary, computing the DCT is equivalent to the DFT of the zero-inserted symmetric signal.

Discrete Cosine Transform (DCT)

The 1D DCT

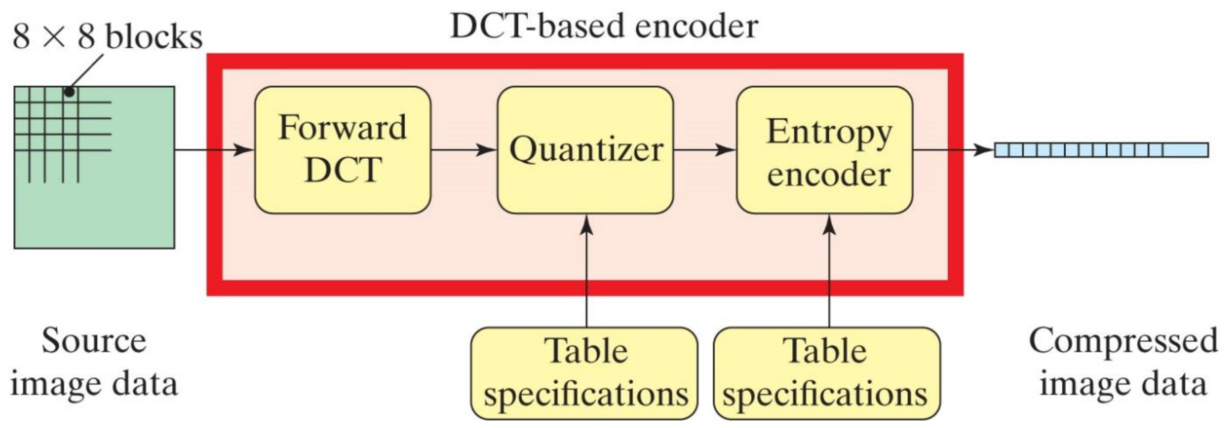
$$X_k = \sum_{n=0}^{N-1} x(n) \cos \frac{(2n+1)\pi k}{2N}$$

The 2D DCT

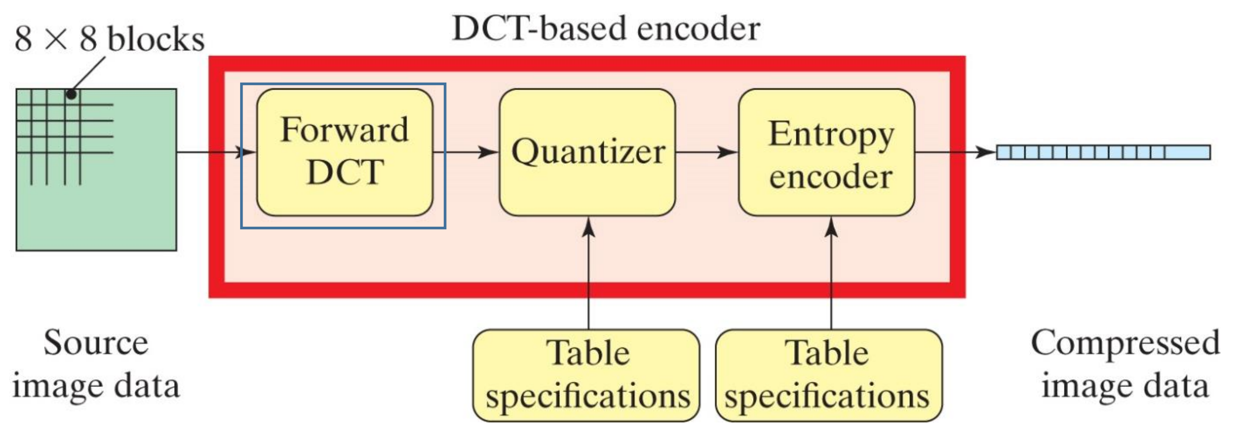
$$X(k_1, k_2) = \sum_{n1=0}^{N-1} \sum_{n2=0}^{N-1} x(n_{1,} n_2) \cos \frac{(2n_1+1)\pi\, k_1}{2N} \cos \frac{(2n_2+1)\pi\, k_2}{2N}$$

JPEG Compression

- A common lossy compression technique.
- It stores quantized DCT values. These values are entropy coded (Huffman coding).
- A quality factor is used to control the quantization process. This quality factor also controls the amount of compression.



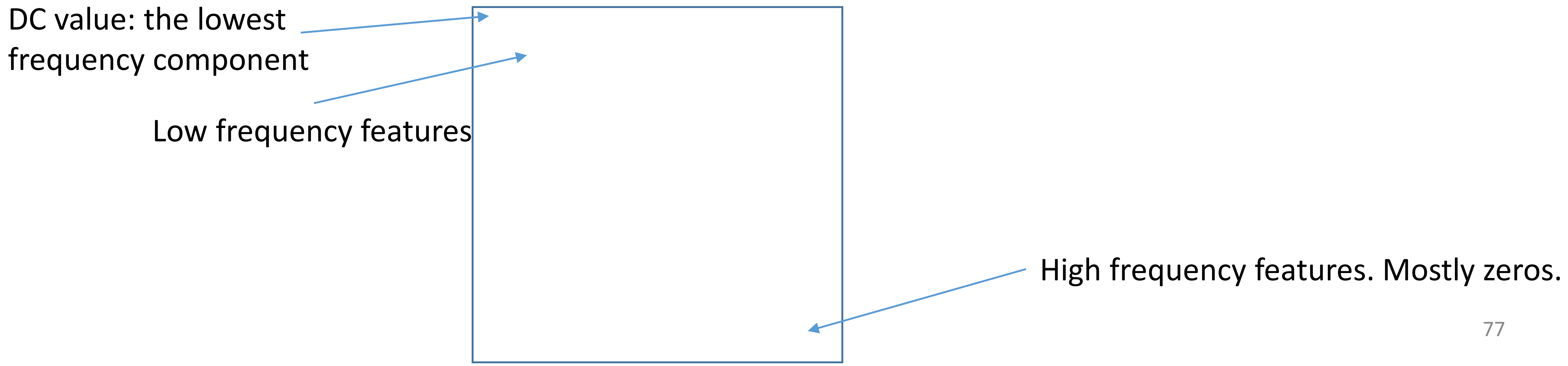
JPEG Compression



- The image is divided into blocks of size 8 x 8.
- Intensities in the image are **level shifted** so that their mid-value is at zero.
- For example if image intensities range from 0 to 255, the value of 256/2=128 is subtracted from each sample.

JPEG Compression

- The forward DCT is applied to each block. The result is 64 element DCT coefficients matrix $\boldsymbol{G_{DCT}}$. Upper left value is called the DC value. It is much higher than the other coefficients.
- Upper left values indicate low frequency components. Higher frequency components are towards the lower right corner.

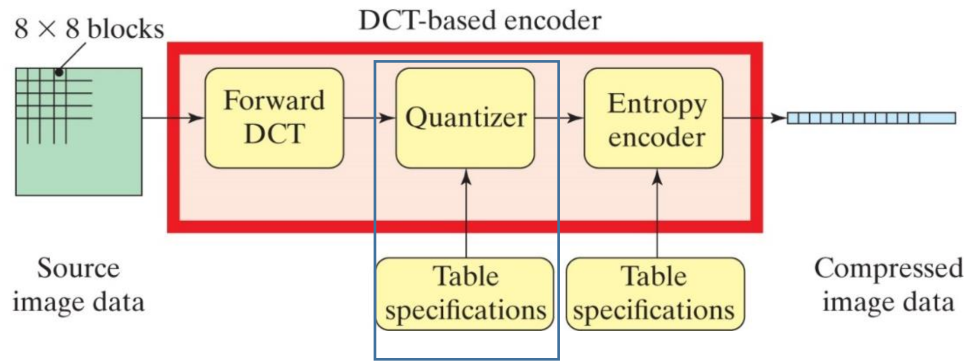

DC value: the lowest frequency component
Low frequency features
High frequency features. Mostly zeros.

JPEG Compression

- Next the block is divided by a quantization table. Results are rounded to the nearest integer.

$$G_{quantized}(u,v) = \left[\frac{G_{DCT}(u,v)}{Q(u,v)}\right]$$

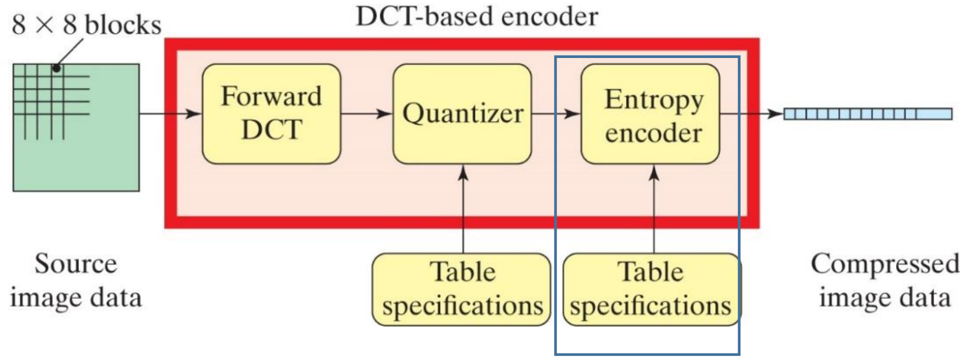
$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$



- Note from the table that high frequency values are assigned larger quantization factors.

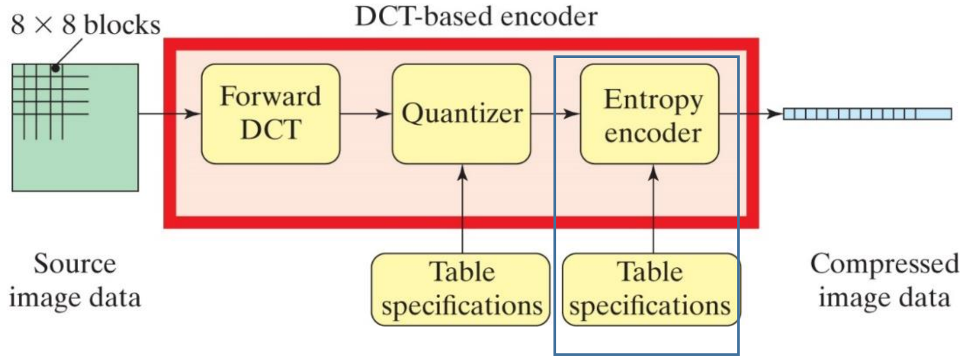
JPEG Compression

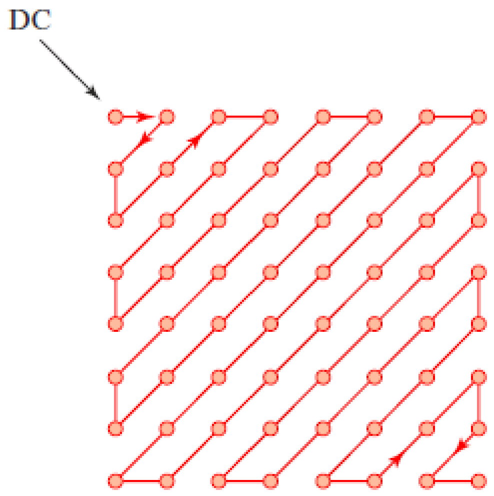
- The first DCT coefficient (DC value) is similar between blocks. The encoder extracts these values in 1D vector. Horizontal predictive coding is used to encode these data. h=[1 0 0]
- Remaining 63 DCT coefficients (AC values). These values contain many zeros and are encoded using Run Length Encoding (RLE)



JPEG Compression

- These AC values are read in a zigzag fashion from low frequency to high frequency.





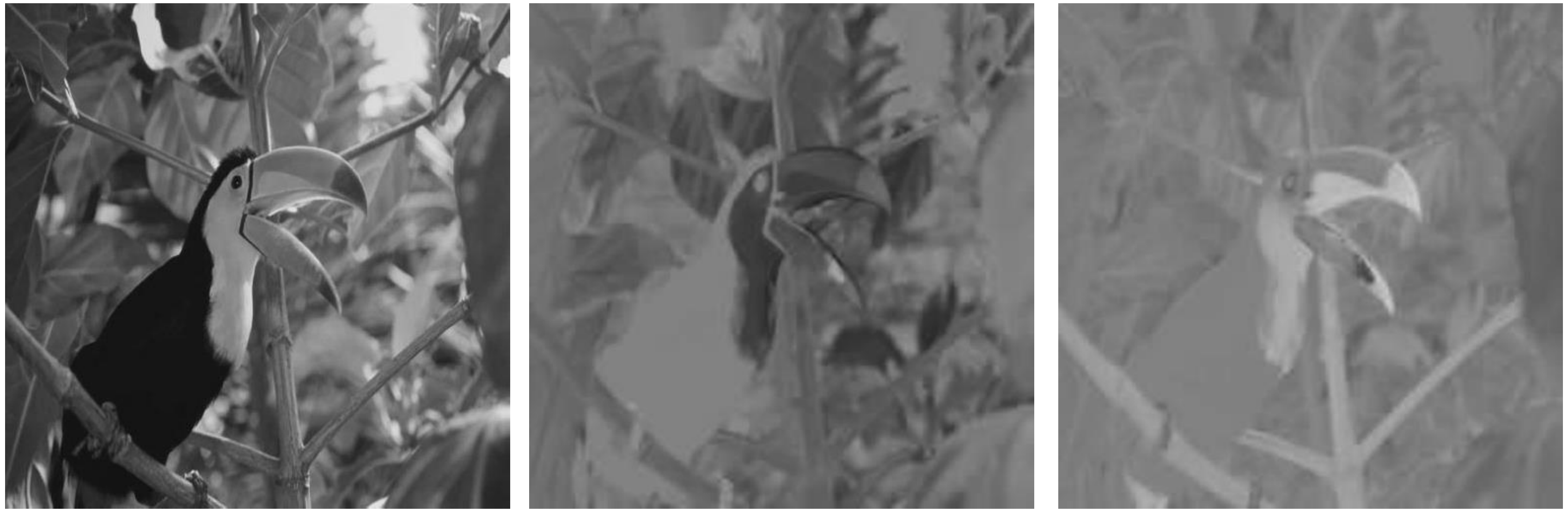
JPEG Compression

- Color representation: Color images are converted from RGB format to a format that separates luminance (i.e. intensity of pixels) from color information.
- This separation reduces redundancy by reducing the correlation of color information between pixel.
- In image compression color information is often presented in the YC_BC_R color space.
- Its is the color space used in digital TV and DVDs.

JPEG Compression

- After decompression, color is converted back to RGB.

Color image


Luminance Y

Chrominance-blue C_B

Chrominance-red C_R

JPEG Compression

- Quantization tables used for the Luminance and Chrominance components.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Luminance table, $Q^{\ell}_{ref}(u,v)$

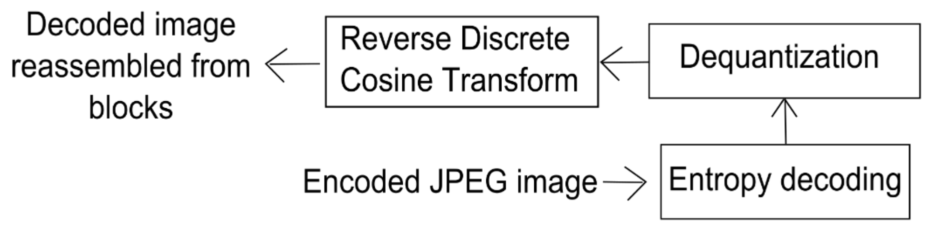
$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

Chrominance table, $Q^{c}_{ref}(u,v)$

- Overall, the chrominance table (which describe color information) is higher in magnitudes. Dividing by chrominance is more likely to generate zeros.
- This factor utilizes the fact that the human eye is less sensitive to spatial variations in chrominance than luminance.

JPEG Decompression

- In the decoding process, operations are reversed


Decoded image reassembled from blocks
Reverse Discrete Cosine Transform
Dequantization
Encoded JPEG image
Entropy decoding

- Reconstructed image is normally not equal to the original image. Losses occurred during the quantization process can't be restored (<u>many values are set to 0 after the division by the quantization tables and these values can't be restored</u>).

JPEG Compression

JPEG uses

- Transform coding (DCT and quantization) for Psychovisual redundancy.
- Horizontal prediction for spatial redundancy in the DCT DC values.
- Run length encoding (RLE) for spatial redundancy in the remaining DCT values.
- Huffman coding for coding redundancy