Multimedia Data Compression – Part II

Lecture 08

BIL464 Multimedia Systems

Mustafa Sert
Asst. Prof.
msert@baskent.edu.tr

Department of Computer Engineering, Başkent University
Ankara 06810 TURKEY

Outline

- Motivation
- Transform Encoding
- Compression Standards and Codecs
- LZW Encoding
- JPEG Encoding

Motivation

- The compression algorithms we have examined so far are all lossless
- With these algorithms, a digital image, audio, or video file can be compressed, and the original data can be perfectly reconstructed when it is decompressed (not considering rounding errors that may result from floating point arithmetic)
- The disadvantage of lossless methods is that they do not always give enough compression, especially for large audio and video files.
  - Lossy methods are needed. Fortunately, these methods are designed so that the information lost is relatively unimportant to the perceived quality of the sound or images

Transform Encoding (1/3)

- Lossy methods are often based upon *transform encoding*.
- The idea is that changing the representation of data can sometimes make it possible to extract sounds or visual details that won't be missed because they are beyond the acuity of human perception.
- Thus, this type of compression begins with a *transform* of the data from one way of representing it to another.
- Two of the most commonly used transforms in digital media are the *discrete cosine transform (DCT)* and the *discrete Fourier transform (DFT)*.
- You should note that it is not the transform that is lossy. No information is lost in the DCT or DFT. However, when a transform is used as one step in a compression algorithm, it becomes possible to discard redundant or irrelevant information in later steps, thus reducing the digital file size. This is the lossy part of the process.

Transform Encoding (2/3)

- The discrete cosine transform is applied to digital images to change their representation from the spatial to the frequency domain.
- Transforming image data to the frequency domain with the DCT can be the first step in compression.
  - Once you have separated out the high frequency components of an image, you can remove them. High frequency components correspond to quick fluctuations of color in a short space—changes that aren't easy for the human eye to see.
  - When this information is removed, later steps in the compression (e.g., Huffman encoding) are even more effective. What we have just described is the basis of JPEG compression.

Transform Encoding (3/3)

- The discrete Fourier transform is applied to sound, transforming audio data from the temporal to the frequency domain.
- With the frequency components separated out, it is possible to determine which frequencies mask or block out other ones and then discard the masked frequencies.
- By this method, transform encoding is followed by perceptual encoding, and the result is a smaller audio file

Compression Standards and Codecs (1/3)

- So far, we have looked at types of compression methods and the concepts behind them. In practice, compression algorithms are implemented in a wide variety of ways that are fine-tuned and optimized in their details.
- It is common for methods to be used in combination with each other, as is the case with JPEG and MPEG compression, which combine the DCT, run-length encoding, and Huffman encoding for compression of photographic images.

Compression Standards and Codecs (2/3)

- Some algorithms are standardized by official committees so that the various implementations all produce files in the same format.
  - If a standardized algorithm is patented, then commercial companies must pay a license fee to implement the algorithm and sell it in a commercial product.
  - Two prominent examples of standardized compression algorithms are DV for camcorder-generated digital video and the family of MPEG algorithms.
  - Arithmetic encoding is an example of an image compression algorithm that is covered by patents

Compression Standards and Codecs (3/3)

- Specific implementations of compression algorithms are called codecs—short for compression/decompression.
- The word codec is usually reserved for audio/video compression (as opposed to still images), since real-time decompression is just as important as initial compression with these time-based media.
- Some codecs are offered as shareware or freeware. Most codecs are commercial products.
- Codecs can be embedded in image, audio, or video processing programs, or they can be sold and used separately.
  - Sorenson is an example of a codec that is embedded in other environments (e.g., QuickTime) and also available in a professional-grade version that runs apart from other application programs (Sorenson compressor codec pack includes MPEG and DV compression and the standard Sorenson codec)

LZW Compression (1/8)

- *LZW compression* is a method that is applicable to both text and image compression.
  - A kind of *lossless* compression method.
  - Uses *fixed-length* codewords
- LZW stands for Lempel-Ziv-Welch, the creators of the algorithm. Lempel and Ziv published the first version of the algorithm in 1977 (called LZ77), and the procedure was revised and improved by Welch in 1984.
- The method is commonly applied to GIF and TIFF image files. The patents are relevant to GIF and TIFF software developers but do not hamper an individual's right to own or transmit files in GIF or TIFF format.

LZW Compression (2/8)

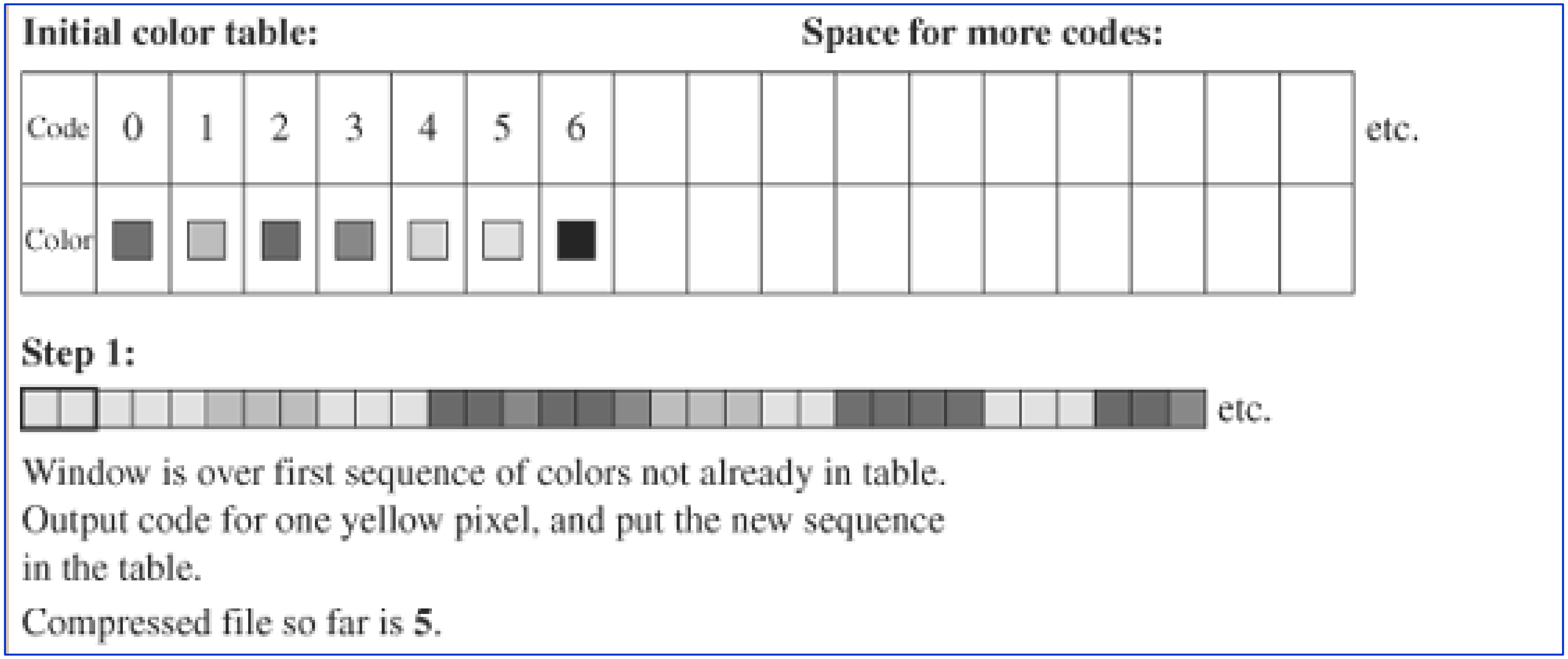
- The LZW algorithm is based on the observation that sequences of color in an image file (or sequences of characters in a text file) are often repeated.
  - Thus, the algorithm uses a sliding expandable window to identify successively longer repeated sequences.
  - These are put into a code table as the file is processed for compression.
  - An ingenious feature of the algorithm is that the full code table does not have to be stored with the compressed file; only the part of it containing the original colors in the image is needed, and then the rest—the codes for the sequences of colors—can be regenerated dynamically during the decoding process. Let's see how this works

LZW Compression (3/8)

- With a first pass over the image file, the code table is initialized to contain all the individual colors that exist in the image file. These colors are encoded in consecutive integers.
- Now, imagine that the pixels in the image file, going left to right and top to bottom, are strung out in one continuous row.
- After initialization, the sliding expandable window moves across these pixels.
  - The window begins with a width of one pixel. (The height is always one pixel.) If the pixel sequence is already in the code table, the window is successively expanded by one pixel until finally a color sequence not in the table is under the window.

LZW Compression (4/8)

- Say that this sequence is *n* pixels long. Then the code for the sequence that is *n* − 1 pixels long is output into the compressed file, and the *n*-pixel-long sequence is put into the code table. This continues until the entire image is compressed.
- A partial trace of LZW compression is illustrated in the following steps:



Initial color table: Space for more codes:

Code	0	1	2	3	4	5	6											
Color	■	□	■	■	□	□	■											

etc.

Step 1:

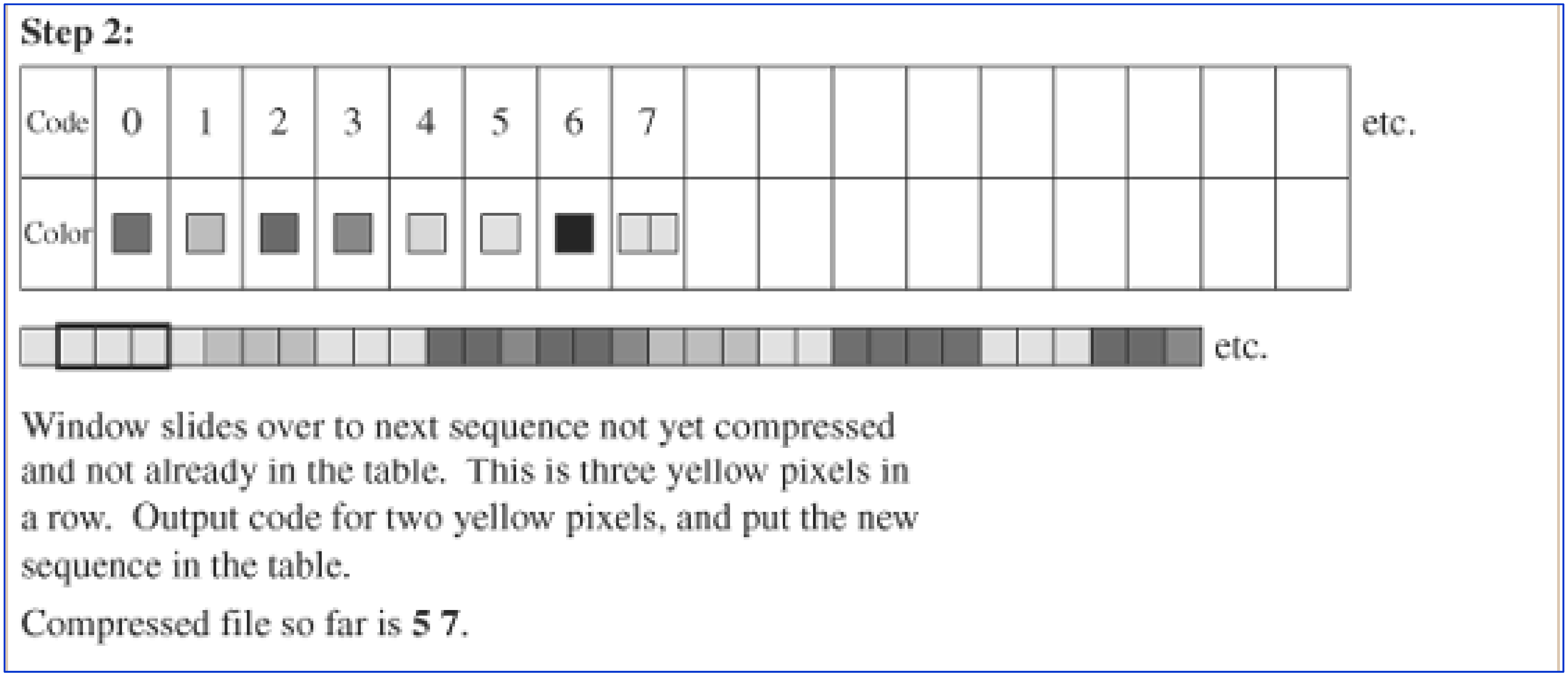
etc.

Window is over first sequence of colors not already in table. Output code for one yellow pixel, and put the new sequence in the table.

Compressed file so far is 5.

LZW Compression (5/8)

- **Step 2:**



Step 2:

Code	0	1	2	3	4	5	6	7										etc.
Color	■	▨	■	▨	□	□	■	□□										

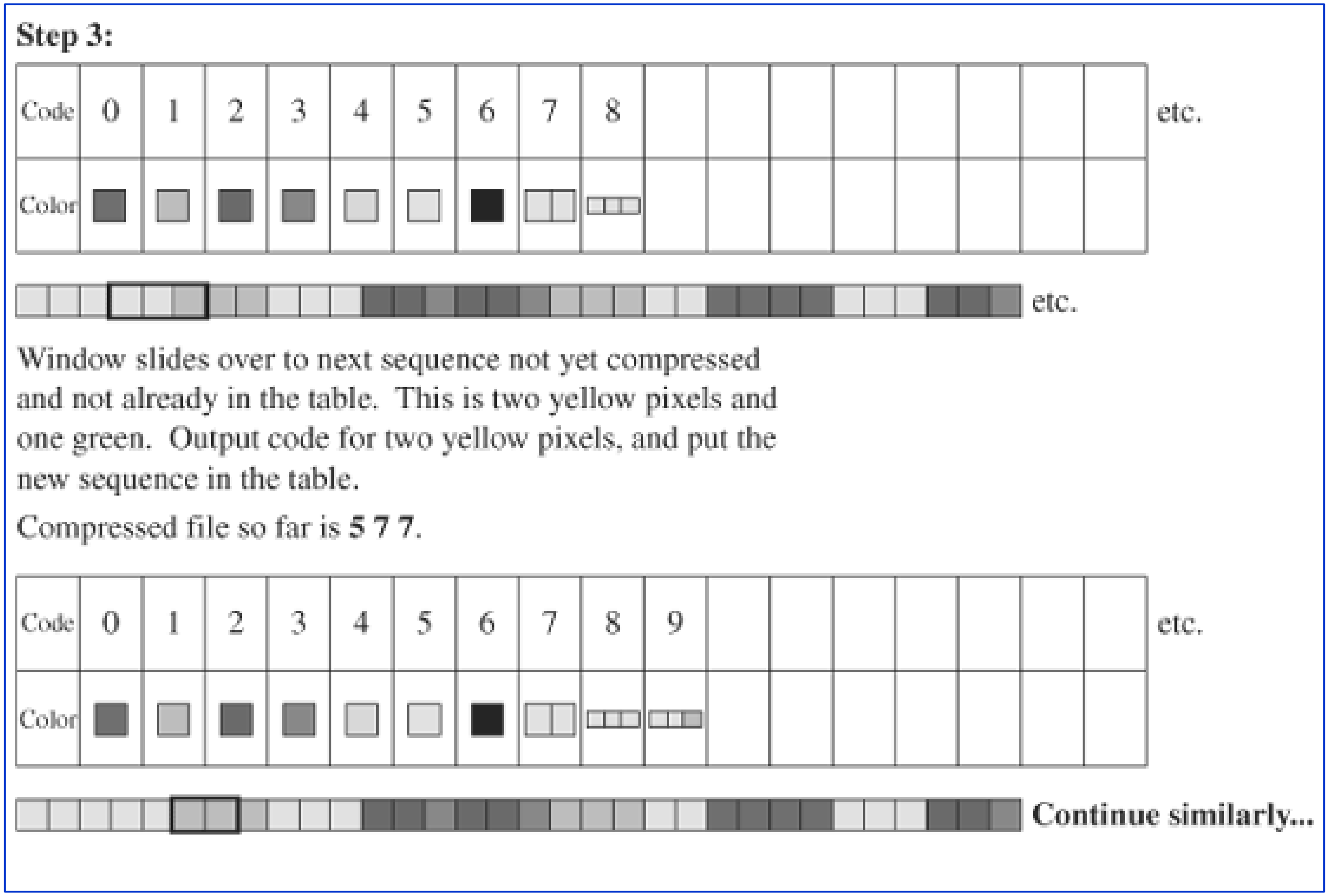
etc.

Window slides over to next sequence not yet compressed and not already in the table. This is three yellow pixels in a row. Output code for two yellow pixels, and put the new sequence in the table.

Compressed file so far is **5 7**.

LZW Compression (6/8)

- **Step 3 & the Compressed file:**



Step 3:

Code	0	1	2	3	4	5	6	7	8								
Color	■	▨	■	■	□	□	■	□□	□□□								

etc.

etc.

Window slides over to next sequence not yet compressed and not already in the table. This is two yellow pixels and one green. Output code for two yellow pixels, and put the new sequence in the table.

Compressed file so far is **5 7 7**.

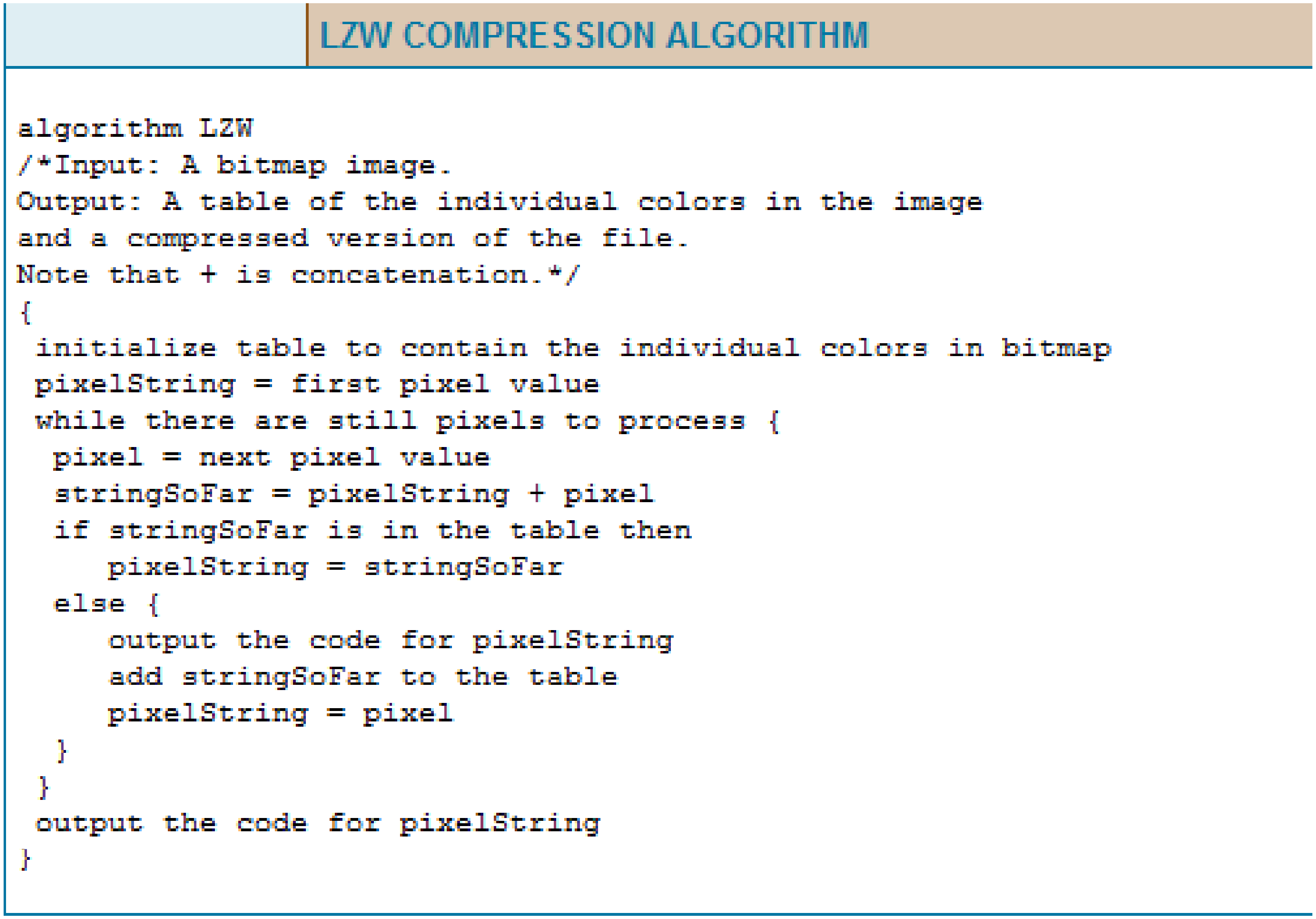
Code	0	1	2	3	4	5	6	7	8	9							
Color	■	▨	■	■	□	□	■	□□	□□□	□□▨							

etc.

Continue similarly...

LZW Compression (7/8)

- The LZW Compression Algorithm:



LZW COMPRESSION ALGORITHM

```
algorithm LZW
/*Input: A bitmap image.
Output: A table of the individual colors in the image
and a compressed version of the file.
Note that + is concatenation.*/
{
 initialize table to contain the individual colors in bitmap
 pixelString = first pixel value
 while there are still pixels to process {
  pixel = next pixel value
  stringSoFar = pixelString + pixel
  if stringSoFar is in the table then
     pixelString = stringSoFar
  else {
     output the code for pixelString
     add stringSoFar to the table
     pixelString = pixel
  }
 }
 output the code for pixelString
}
```

LZW Compression (8/8)

- The decoding of LZW works in a way similar to compression
- The clever thing about the algorithm is that we only have to store the basic colors along with the compressed image
- This consists of the colors that exist in the image and their codes – but not the codes for strings of colors
- As the image is compressed, we will always either find the code we need in the code table, or we will be able to figure out what code we need to insert in the table based on the codes we've already encountered
- Notice that there are a number of variants of the method, that is, this is the basic algorithm
- LZW can work particularly well for images that have long strings of the same color (e.g., cartoon-like figures, or opposed to photographic images)

JPEG Compression (1/30)

- JPEG is an acronym for Joint Photographic Experts Group.
- In general, *JPEG compression* refers to a compression algorithm suitable for reducing the size of photographic images or continuous-tone artwork—pictures where the scene transitions move smoothly from one color to another, as opposed to cartoon-like images where there are sharp divisions between colors.
- If you see a file with the *.jpg* or *.jpeg* suffix, you can assume it has been compressed with the JPEG method, but it is also possible to apply JPEG compression to images saved in TIFF, PICT, and EPS files.

JPEG Compression (2/30)

- Although JPEG is a lossy compression method, it is designed so that the information that is lost is not very important to how the picture looks—that is, the algorithm removes closely spaced changes in color that are not easily perceived by the human eye.
- This is made possible by transforming the image data from the spatial domain to the frequency domain.
- When the image is represented in terms of its frequency components, it is possible to “throw away” the high frequency components, which correspond to barely perceptible details in color change

JPEG Compression (3/30)

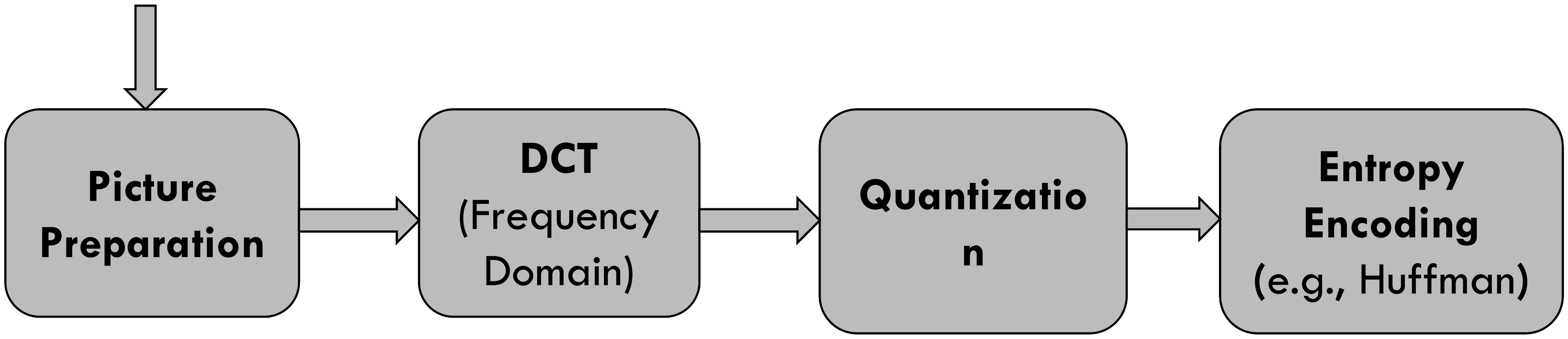
- Another advantage of JPEG compression is that image processing programs allow you to choose the JPEG compression rate, so you can specify how important the image size is versus the image's fidelity to the original subject.
- JPEG compression on a 24-bit color image yields an excellent compression rate. With a rate of about 10 : 1 or 20 : 1, you'll notice hardly any difference from the original to the compressed image. Even compression rates up to 50 : 1 can give acceptable results for some purposes.

JPEG Compression (4/30)

- The main disadvantage to JPEG compression is that it takes longer for the encoding and decoding than other algorithms require, but usually the compression/decompression time is not noticeable and is well justified when compared with the savings in storage space and download time required for the image file.
- Without JPEG compression, most people would not want to spend time it would take to download pictures on web pages

JPEG Compression (5/30)

- JPEG Compression Stages:



JPEG Compression (6/30)

- Step 1: Divide the image into 8 × 8 pixel blocks and convert RGB to a luminance/chrominance color model (YUV, YCbCr)
  - **Motivation:** The image is divided into 8 × 8 pixel blocks to make it computationally more manageable for the next steps. Converting color to a luminance/chrominance model makes it possible to remove some of the chrominance information, to which the human eye is less sensitive, without significant loss of quality in the image
  - **Details:** For efficiency reasons, JPEG compression operates on 8 × 8 pixel blocks on the image file. If the file's length and width are not multiples of eight, the bitmap can be padded and the extra pixels removed later

JPEG Compression (7/30)

- JPEG compression can be performed on 24-bit color or 8-bit grayscale images. If the original pixel data is in RGB color mode, then there is an 8 × 8 pixel block for each of the color channels.
  - The blocks for these color channels are processed separately, but the three blocks are grouped so that the image can be reconstructed upon decoding
- Converting the image file from RGB to a model like YUV/YCbCr makes it possible to achieve an even greater compression rate by means of chrominance subsampling.
  - The first step in JPEG compression is the transformation from the RGB color model to the YUV color model

JPEG Compression (8/30)

- *Chrominance subsampling* (also called *chrominance downsampling*) is a process of throwing away some of the bits used to represent pixels—in particular, some of the color information.
  - For example, with YCbCr color mode, we might choose to save only one Cb value and one Cr value but four Y values for every four pixel values. Three commonly used subsampling rates are depicted below:

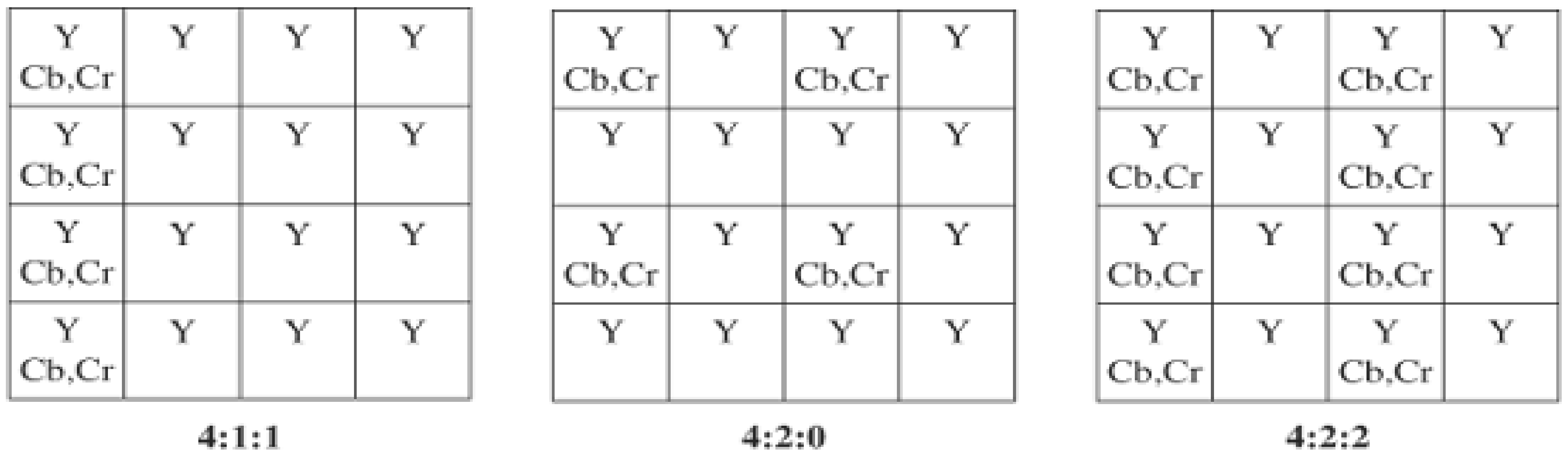

Y Cb,Cr | Y | Y | Y
Y Cb,Cr | Y | Y | Y
Y Cb,Cr | Y | Y | Y
Y Cb,Cr | Y | Y | Y
4:1:1
Y Cb,Cr | Y | Y Cb,Cr | Y
Y | Y | Y | Y
Y Cb,Cr | Y | Y Cb,Cr | Y
Y | Y | Y | Y
4:2:0
Y Cb,Cr | Y | Y Cb,Cr | Y
Y Cb,Cr | Y | Y Cb,Cr | Y
Y Cb,Cr | Y | Y Cb,Cr | Y
Y Cb,Cr | Y | Y Cb,Cr | Y
4:2:2

Figure Chrominance subsampling

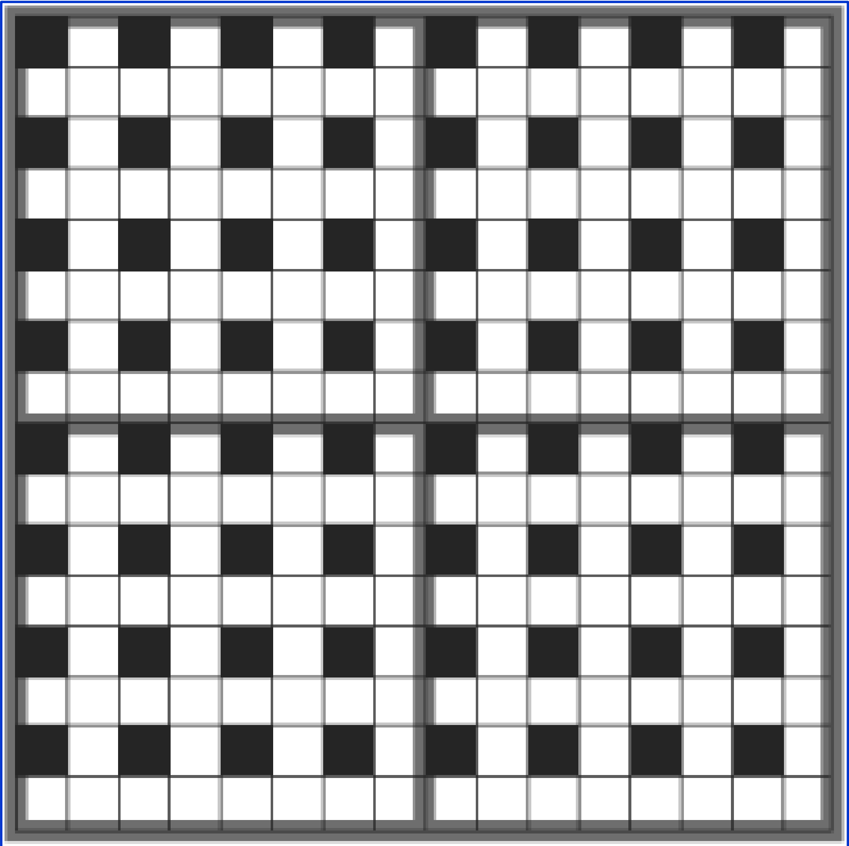
JPEG Compression (9/30)

- The conventional notation for luminance/chrominance subsampling is in the form *a*:*b*:*c*
- Common subsampling rates are 4:1:1, 4:2:0, and 4:2:2
  - To understand what these numbers represent, count the number of samples taken for Y and Cb (or Cr, since they are the same) in each pair of four-pixel-wide rows. *a* is the number of Y samples in both rows. *b* is the number of Cb samples in the first row (and also the number of Cr samples). c is the number of Cb (and Cr samples) in the second row.
  - Note that we have not specified how the Cb and Cr values are derived. Just one of the values in a sub-block could be used, or the values could be averaged. (In fact, MPEG-1 and MPEG-2 video compression use different methods for determining the single chrominance values corresponding to four luminance values in 4 : 2 : 0 downsampling.)

JPEG Compression (10/30)

- With RGB color mode, we can imagine that for each 8×8 pixel section of the image, there 8×8 are actually three blocks to be processed, one for each of the R, G, and B color channels.
- With YCbCr color mode, we have to picture this a little differently.
  - We begin by dividing the image into 16×16 pixel macroblocks. Then, with 4 : 2 : 0 chroma downsampling, we get four 8×8 blocks of Y data for every one 8×8 block of Cb and one 8×8 block of Cr data.
  - This is pictured in the Figure (see next slide). Each of the blocks undergoes the remaining steps of the algorithm, and the resulting compressed data is reconstructed upon decoding

JPEG Compression (11/30)



With 4:2:0 YCbCr chrominance subsampling,
a 16 × 16 macroblock yields:
–four 8 × 8 blocks of Y values
–one 8 × 8 block of Cb values
–one 8 × 8 block of Cr values
Each black location represents one Cb
and one Cr sample taken for a block
of four pixels. Y samples are taken
at all locations.

JPEG Compression (12/30)

- You can see that using a luminance/chrominance color model and then chrominance sub-sampling reduces the image file size even before the rest of the compression steps are performed
- Consider the reduction in file size if 4:2:0 downsampling is used by counting the number of bytes in an area of pixels with a width of four pixels and a height of two pixels—eight pixels total
- Assuming that each component requires one byte, an unsubsampled image requires 8 * 3 = 24 bytes. Subsampling at a rate of 4:2:0 requires 8 + (2 * 2) = 12 bytes (eight for the Y component but only two for each of the Cb and Cr). This is a 2:1 savings

JPEG Compression (13/30)

- Chrominance subsampling is not a required step in JPEG compression, and can be ignored.
- We will trace through an example of JPEG compression performed on an 8 × 8 block of a grayscale image.
  - In grayscale images, the RGB color channels all have the same value, so it's necessary to store only one byte value per pixel
  - Thus, our example will need to show the processing of only one 8 × 8 block for an 8 × 8 pixel area. But keep in mind that if 4 : 2 : 0 chroma subsampled YCbCr color is used, for every 16 × 16 pixel area, there are four 8 × 8 blocks of Y data and one each of Cb and Cr data. If RGB color is used, there are three 8 × 8 blocks—one each for R, G, and B—for every 8 × 8 pixel area

JPEG Compression (14/30)

- Step 2: Shift values by –128 and transform from the spatial to the frequency domain
  - **Motivation:** On an intuitive level, shifting the values by –128 is like looking at the image function as a waveform that cycles through positive and negative values. This step is a preparation for representing the function in terms of its frequency components. Transforming from the spatial to the frequency domain makes it possible to remove high frequency components. High frequency components are present if color values go up and down quickly in a small space. These small changes are barely perceptible in most people's vision, so removing them does not compromise image quality significantly

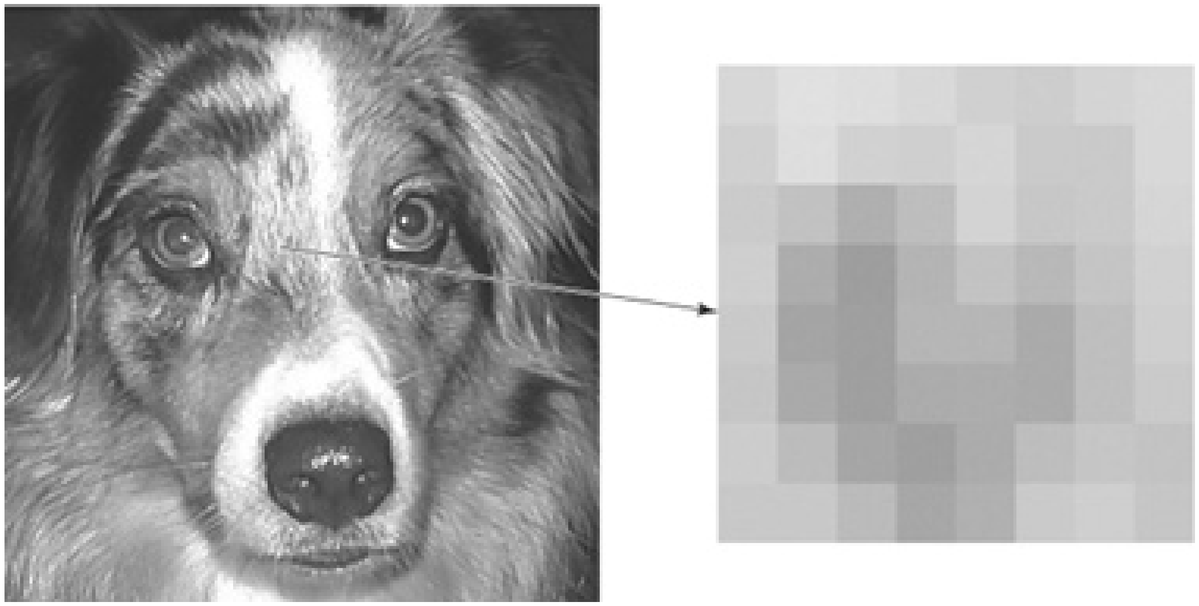
JPEG Compression (15/30)

- The DCT is a mathematical procedure that transforms image data from the spatial to the frequency domain
- It is a lossless procedure, aside from the small unavoidable error introduced through floating point arithmetic.
- The DCT performs the transform in one direction, and the inverse DCT can restore the original data without loss of information
- In JPEG, DCT operates on one block at a time. Because there are 64 elements in an 8x8 block, this is called the 64-element or 64-coefficient DCT. The DCT transform operates on this block in a left-to-right, top-to-bottom manner.
- DCT converts the chroma subsampled image to a frequency-domain representation

JPEG Compression (16/30)

- DCT: Before computing the DCT of the subimage, its gray values are shifted from a positive range to one centered around zero. For an 8-bit image each pixel has 256 possible values: [0,255]. To center around zero it is necessary to subtract by half the number of possible values, or 128.
- Example: Our example is based on an 8×8 pixel area taken from the picture shown in the Figure below:

$$\boxed{\frac{2^{bit}}{2} = \frac{2^8}{2} = 2^7 = 128}$$



JPEG Compression (17/30)

- Pixel values and the values shifted by –128 are given in the following tables:

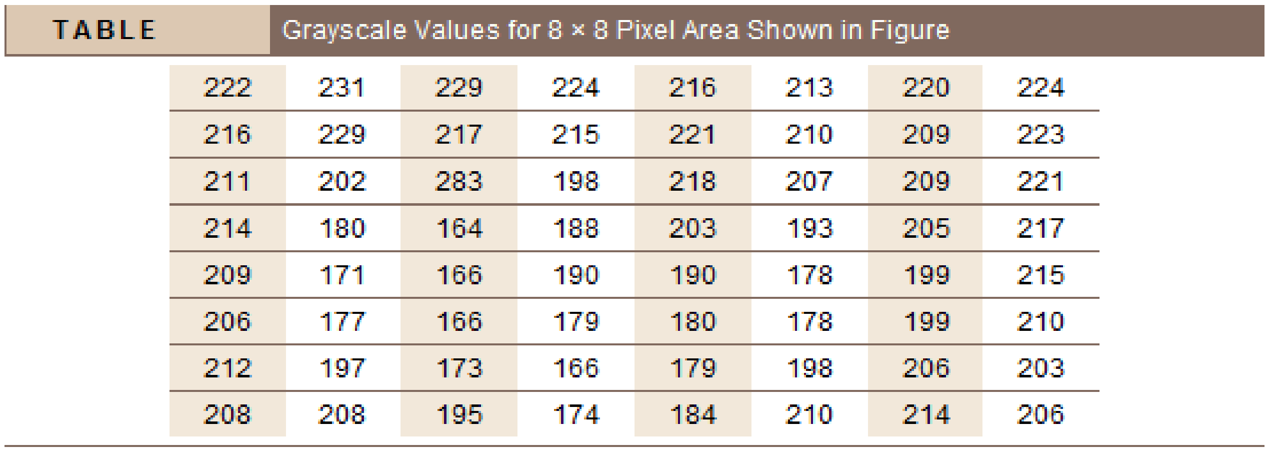


TABLE Grayscale Values for 8 × 8 Pixel Area Shown in Figure

222	231	229	224	216	213	220	224
216	229	217	215	221	210	209	223
211	202	283	198	218	207	209	221
214	180	164	188	203	193	205	217
209	171	166	190	190	178	199	215
206	177	166	179	180	178	199	210
212	197	173	166	179	198	206	203
208	208	195	174	184	210	214	206

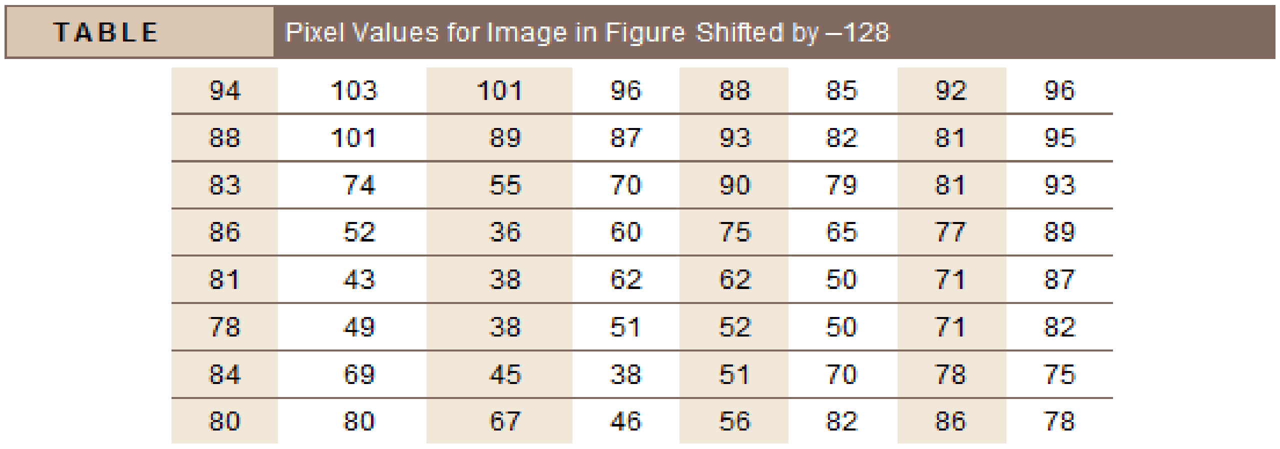


TABLE Pixel Values for Image in Figure Shifted by –128

94	103	101	96	88	85	92	96
88	101	89	87	93	82	81	95
83	74	55	70	90	79	81	93
86	52	36	60	75	65	77	89
81	43	38	62	62	50	71	87
78	49	38	51	52	50	71	82
84	69	45	38	51	70	78	75
80	80	67	46	56	82	86	78

JPEG Compression (18/30)

- The next step is to take the two-dimensional DCT, which is given by:

$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

- Where
  - **u** is the horizontal spatial frequency, for the integers $0 \leq u < 8$
  - **v** is the vertical spatial frequency, for the integers $0 \leq v < 8$
  - The normalizing function: $\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$
  - $\mathbf{g}_{x,y}$ is the pixel value at coordinates (x,y)
  - $\mathbf{G}_{u,v}$ is the DCT coefficient at coordinates (u,v)

JPEG Compression (19/30)

- If we perform this transformation on our matrix above, we get :

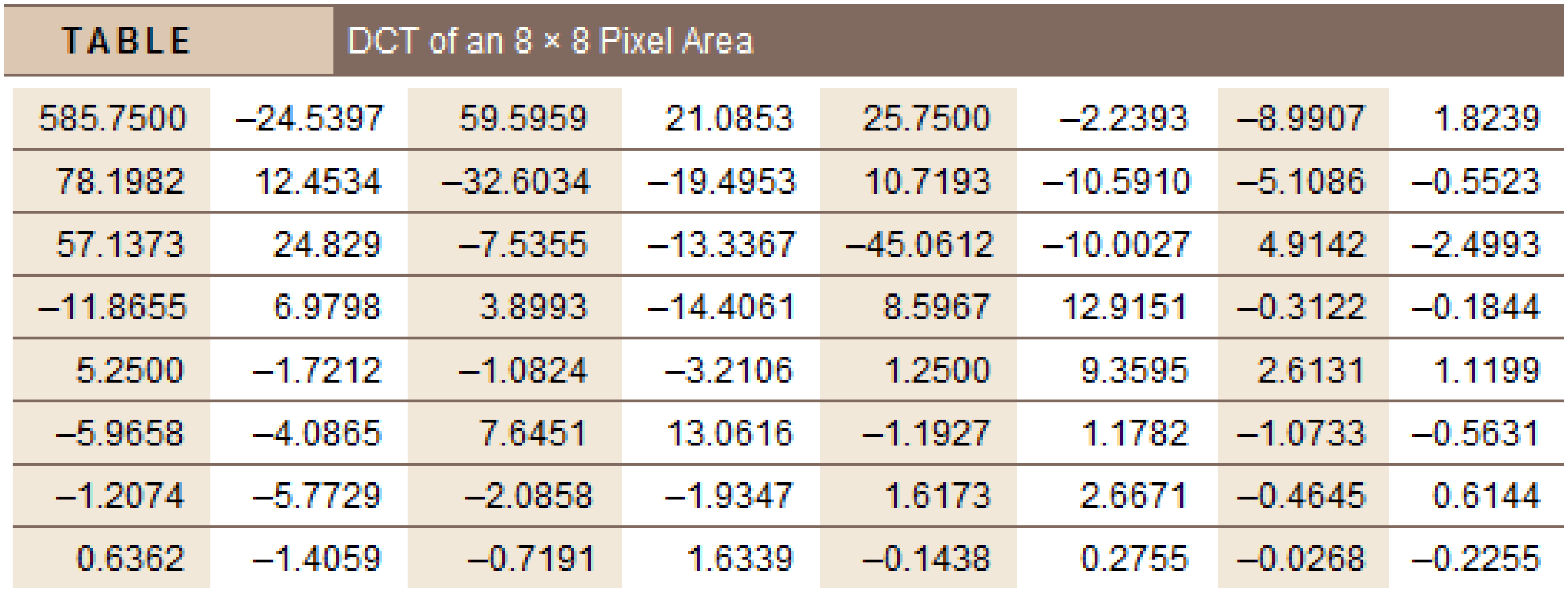


TABLE DCT of an 8 × 8 Pixel Area

585.7500	–24.5397	59.5959	21.0853	25.7500	–2.2393	–8.9907	1.8239
78.1982	12.4534	–32.6034	–19.4953	10.7193	–10.5910	–5.1086	–0.5523
57.1373	24.829	–7.5355	–13.3367	–45.0612	–10.0027	4.9142	–2.4993
–11.8655	6.9798	3.8993	–14.4061	8.5967	12.9151	–0.3122	–0.1844
5.2500	–1.7212	–1.0824	–3.2106	1.2500	9.3595	2.6131	1.1199
–5.9658	–4.0865	7.6451	13.0616	–1.1927	1.1782	–1.0733	–0.5631
–1.2074	–5.7729	–2.0858	–1.9347	1.6173	2.6671	–0.4645	0.6144
0.6362	–1.4059	–0.7191	1.6339	–0.1438	0.2755	–0.0268	–0.2255

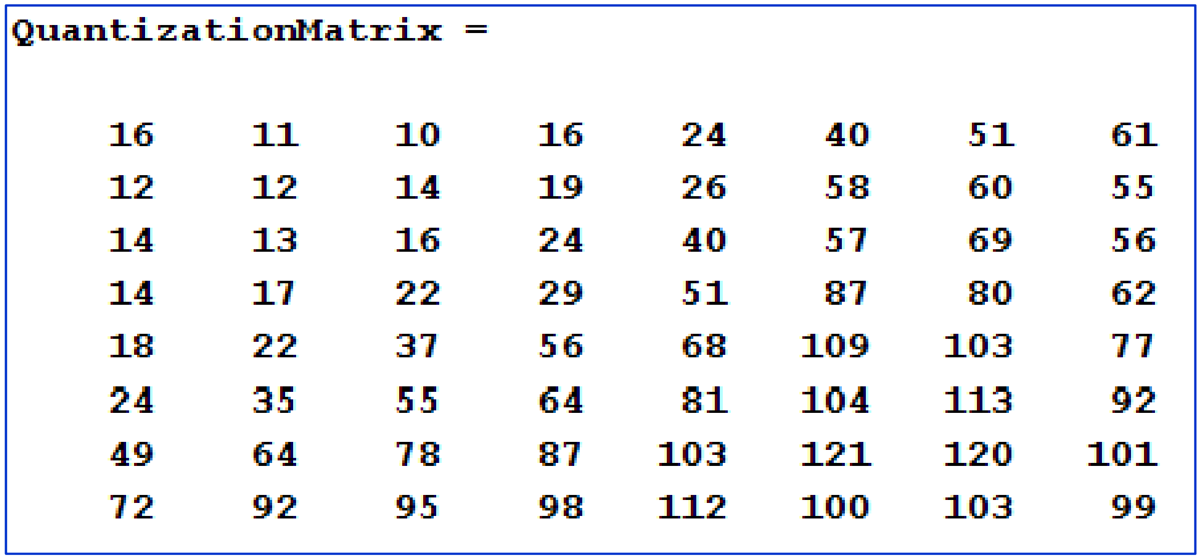
- Note the rather large value of the top-left corner. This is the DC coefficient. The remaining 63 coefficients are called the AC coefficients.

JPEG Compression (20/30)

- Step 3: Quantize the frequency values
  - **Motivation:** Quantization involves dividing each frequency coefficient by an integer and rounding off.
  - **Details:** The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation.
    - The coefficients for high-frequency components are typically small, so they often round down to 0—which means, in effect, that they are thrown away
    - This is the main lossy operation in the whole process

JPEG Compression (21/30)

- A typical quantization matrix, as specified in the original JPEG Standard, is as follows:



```
QuantizationMatrix =

    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
```

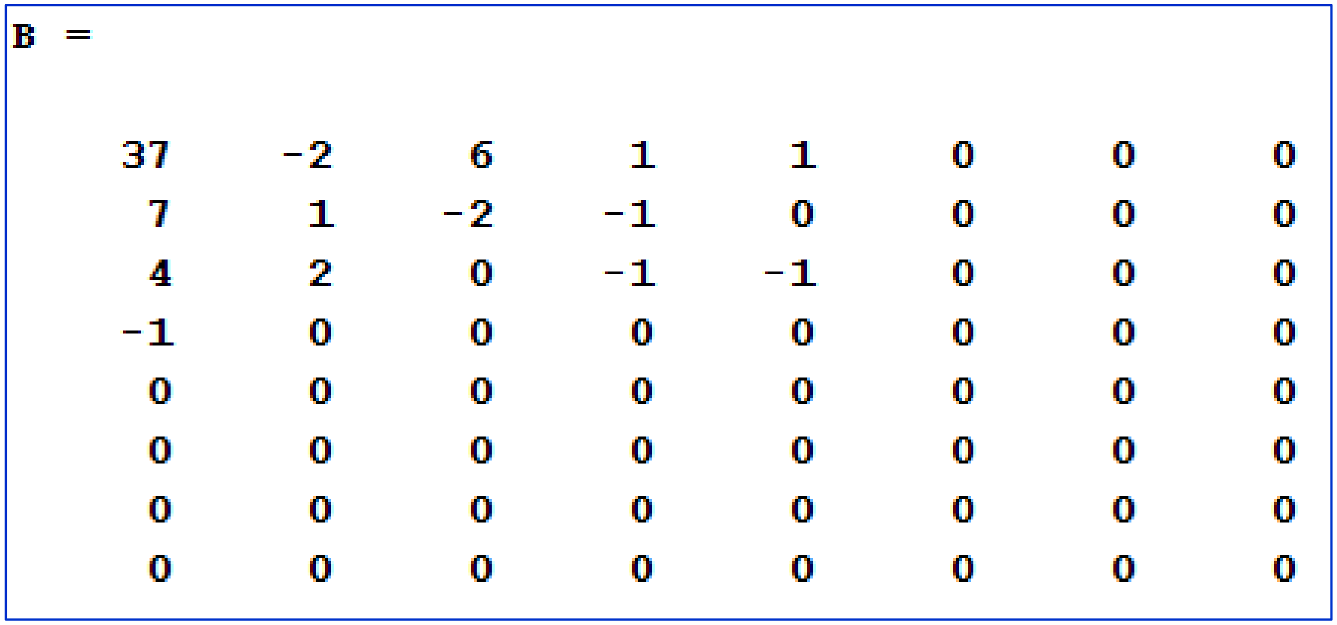
- The quantized DCT coefficients are computed with

$$B_{j,k} = \text{round}\left(\frac{A_{j,k}}{Q_{j,k}}\right) \quad \text{for } j = 0,1,2,\cdots,N_1-1; k = 0,1,2,\cdots,N_2-1$$

  - where *A* is the unquantized DCT coefficients; Q is the quantization matrix above; and *B* is the quantized DCT coefficients

JPEG Compression (22/30)

- Using this quantization matrix with the DCT coefficient matrix from above results in:



```
B =


    37    -2     6     1     1     0     0     0
     7     1    -2    -1     0     0     0     0
     4     2     0    -1    -1     0     0     0
    -1     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
```

- For example, using 585.7500 (the DC coefficient) and rounding to the nearest integer:

$$\boxed{round\left(\frac{585.7500}{16}\right) = round(36.609375) = 37}$$

JPEG Compression (23/30)

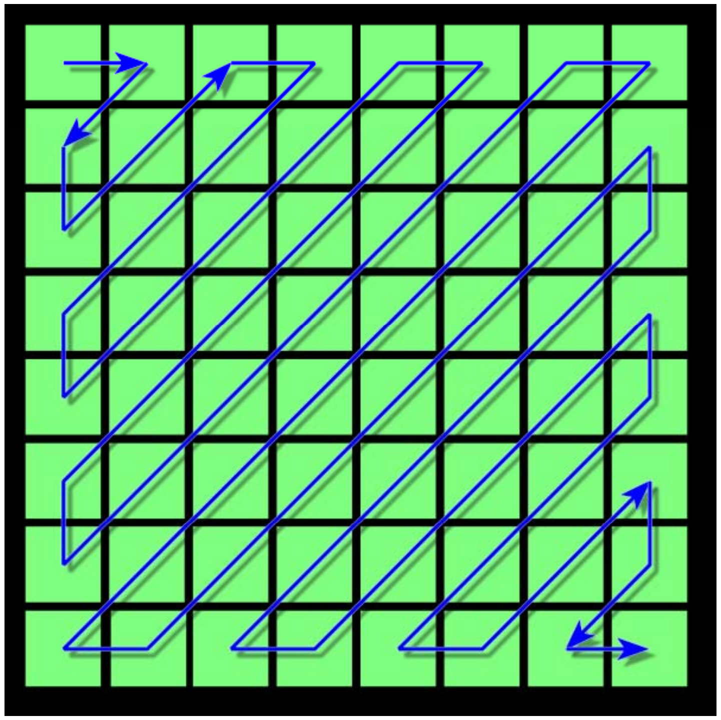
- Typically in JPEG compression, there will be a lot of zeros at the end of the matrix
- Rounding off after dividing by an integer has effectively thrown away many high-frequency components. Then the strings of zeros make a good compression rate possible when run-length encoding is applied.
- Rounding during quantization makes JPEG a lossy compression method, but, depending on the compression rate chosen, the information that is lost usually does not unduly compromise the quality of the image

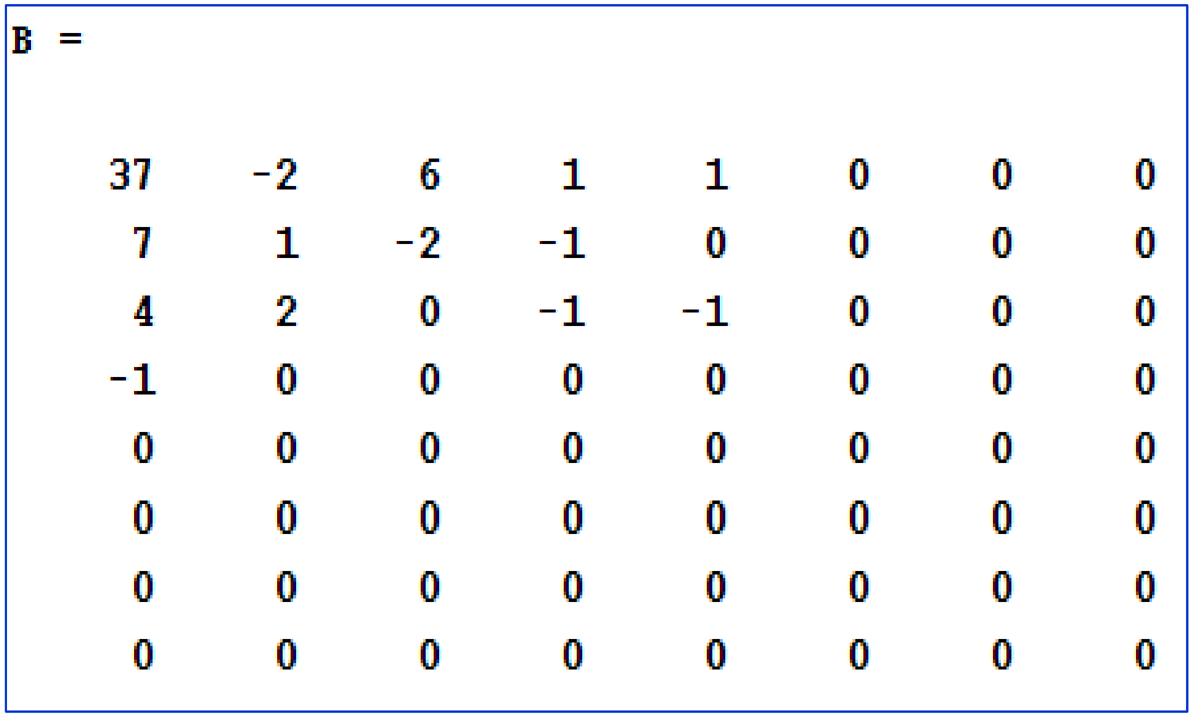
JPEG Compression (24/30)

- Step 4: Entropy Encoding
  - **Motivation:** It involves arranging the image components in a "zigzag" order, employing run-length encoding (RLE) algorithm that groups similar frequencies together, and then using Huffman coding on what is left.
  - The JPEG standard also allows, but does not require, the use of arithmetic coding, which is mathematically superior to Huffman coding. However, this feature is rarely used as it is covered by patents and because it is much slower to encode and decode compared to Huffman coding. Arithmetic coding typically makes files about 5% smaller.

JPEG Compression (25/30)

- The zigzag sequence of the quantized coefficients are obtained as shown in the figure (The order given on the left side is applied to the quantized coefficients given on the right side)



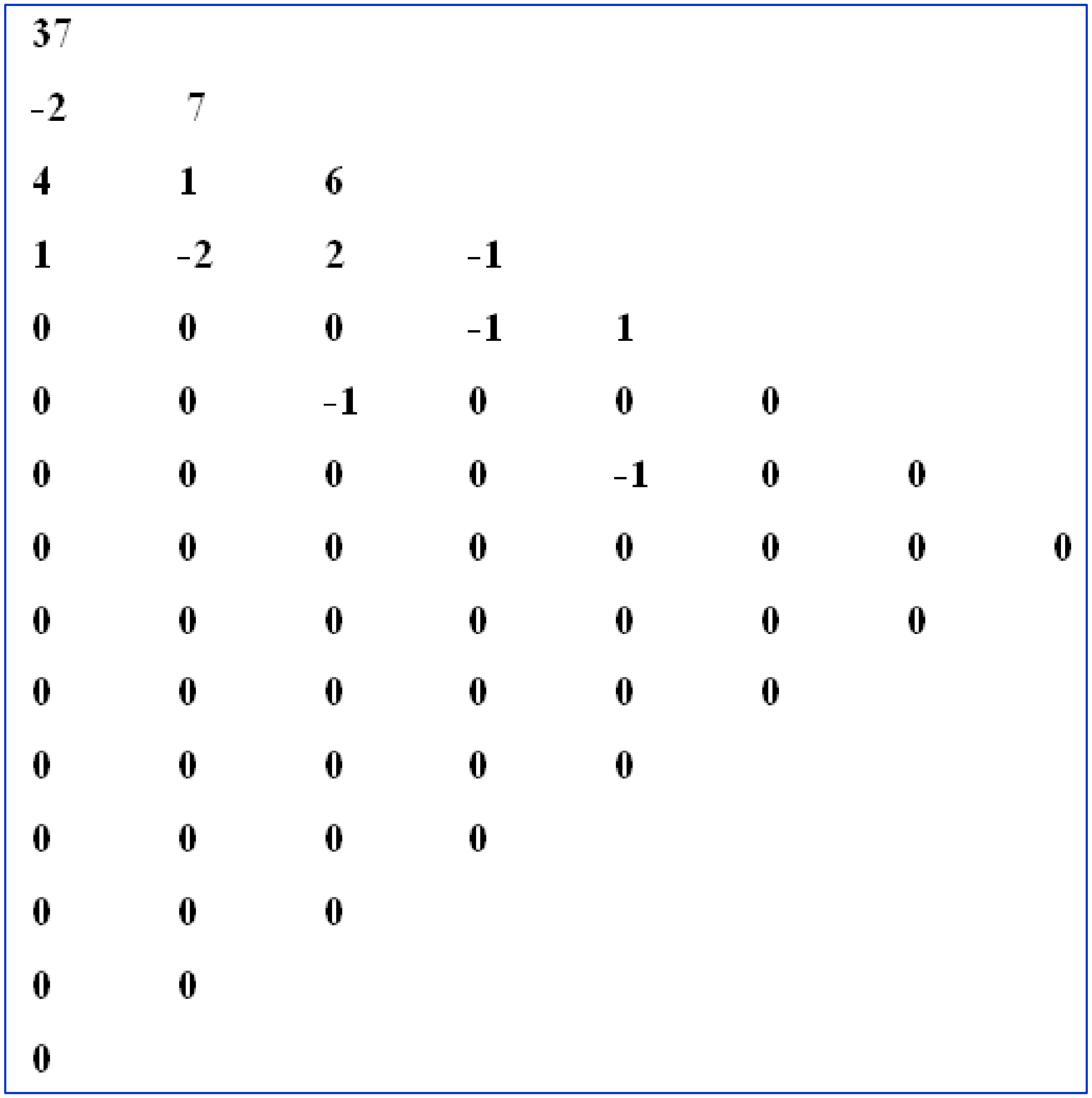


```
B =


    37    -2     6     1     1     0     0     0
     7     1    -2    -1     0     0     0     0
     4     2     0    -1    -1     0     0     0
    -1     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
```

JPEG Compression (26/30)

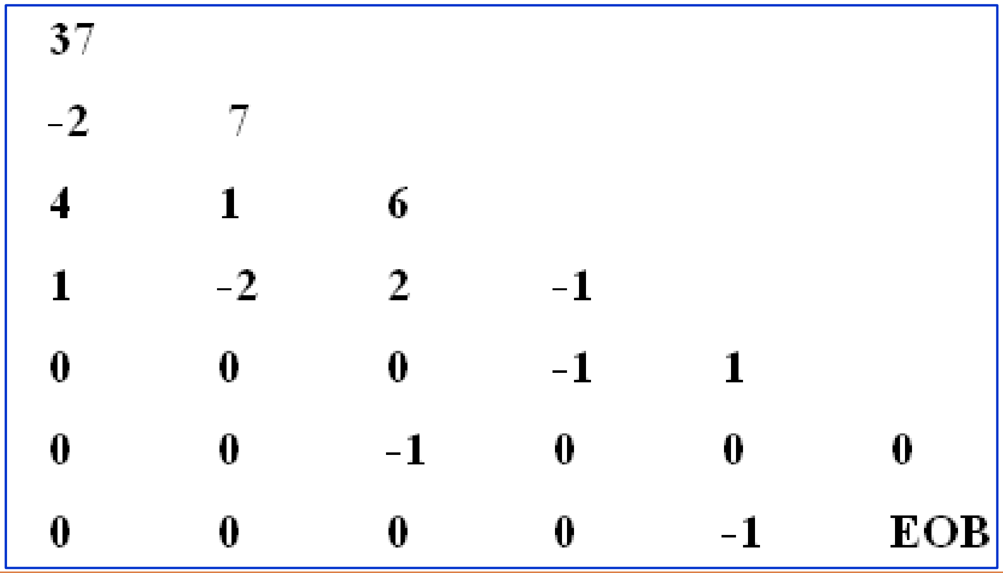
- The order of quantized values is now 37, -2, 7, 4, 1, 6, 1, -2, 2, -1, 0, 0, 0, -1, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, and 36 zeros.
- The zigzag sequence of the quantized coefficients are shown aside. (The format shown is just for ease of understanding/viewing .)



37							
-2	7						
4	1	6					
1	-2	2	-1				
0	0	0	-1	1			
0	0	-1	0	0	0		
0	0	0	0	-1	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							

JPEG Compression (27/30)

- A Standard RLE algorithm is employed to this sequence prior to Huffman method
- Additional compression can be achieved with some kind of entropy encoding – JPEG uses Huffman Encoding
- JPEG has a special Huffman code word for ending the sequence prematurely when the remaining coefficients are zero.
- Using this special code word: "EOB", the sequence becomes:



37					
-2	7				
4	1	6			
1	-2	2	-1		
0	0	0	-1	1	
0	0	-1	0	0	0
0	0	0	0	-1	EOB

JPEG Compression (28/30)

- The JPEG standard also allows, but does not require, the use of arithmetic coding, which is mathematically superior to Huffman coding.
  - However, this feature is rarely used as it is covered by patents and because it is much slower to encode and decode compared to Huffman coding. Arithmetic coding typically makes files about 5% smaller
- The compressed file is put into a standardized format that can be recognized by the decompressor.
  - A header contains global information such as the type of file, the width and height, one or more quantization tables, Huffman code tables, and an indication of any pixel-padding necessary to create properly sized coding units.

JPEG Compression (29/30)

- An alternative JPEG compression method is called *JPEG2000*, noted for its high compression rate and good quality, without some of the blocky artifacts of standard JPEG.
- This method represents digital image data as wavelets as an alternative to the DCT.

JPEG Compression (30/30)

- Decoding: Decoding to display the image consists of doing all the above in reverse
  - Employing decompression (Huffman/RLE)
  - Taking the entry-for-entry product with the quantization matrix from above
  - Taking the inverse DCT (type-III DCT)
  - Adding 128 to each entry
  - Finally, we obtain the uncompressed subimage
    - Uncompressed image can be compared to the original subimage by taking the difference (original – uncompressed). This is called the **error**.