Image Compression

Image compression involves reducing the size of image data file, while is retaining necessary information, the reduced file is called the compressed file and is used to reconstruct the image, resulting in the decompressed image. The original image, before any compression is performed, is called the uncompressed image file. The ratio of the original, uncompressed image file and the compressed file is referred to as the compression ratio.

$$\textbf{Compression Ratio} = \frac{\text{Uncompressed File Size ...... Size U}}{\text{Compressed File Size ....Size C}}$$

It is often written as SizeU: SizeC

Examples

Example: **the original image is 256×256 pixel, single band (gray scale), 8-bit per pixel. This file is 65,536 bytes (64K). After compression the image file is 6,554 byte.**

The compression ratio is:

$\text{Size}_U : \text{Size}_{C\,=}$ this can be written as 10:1

This is called a “10 to 1” compression or a “10 times compression”, or it can be stated as “compressing the image to 1/10 original size.

Another way to state the compression is to use the terminology of bits per pixel. For an N×N image

$$\textit{Bit per Pixel} = \frac{\text{Number of Bites}}{\text{Number of Pixels}} = \frac{8(\text{Number of Bytes})}{N * N}$$

Examples Cont.

Example: using preceding example, with a compression ratio of 65536/6554 bytes. We want to express this as bits per pixel.

This is done by first finding the number of pixels in the image.

256×256 = 65,536

We then find the number of bits in the compressed image file

6554× (8 bit/byte) =52432 bits.

Now, we can find the bits per pixel by taking the ratio

52432/65536=0.8 bit/ Pixel.

The reduction in file size is necessary to meet

1- The bandwidth requirements for many (كمية البيانات) transmission systems.

2-The storage requirements in computer data base.

The amount of data required for digital images is enormous. For example, a single 512×512, 8-bit image requires 2,097,152 bits for storage. If we wanted to transmission this image over the World Wide Web, it would probably take minutes for transmission –too long for most people to wait.

512 ×512×8= 2,097,152.

<u>Example</u>

To transmit a digitized color scanned at 3,000×2,000 pixels, and 24 bits, at 28.8(kilobits/second), it would take about

$$\frac{(3000 \times 2000 \text{ pixels})(24 \text{ bits/pixels})}{(28.8 \times 1024 \text{ bits / second})} \cong 4883 \text{ Second}$$

$$= 81 \text{ minutes.}$$

Couple this result with transmitting multiple image or motion images, and the necessity of image compression can be appreciated.

The key to a successful compression schema comes with the second part of the definition –retaining necessary information. To understand this we must differentiate between data and information.

For digital images, data refer to pixel gray-level values the correspond to the brightness of a pixel at a point in space. Information is interpretation of the data in a meaningful way. Data are used to convey information, much like the way the alphabet is used to convey information via words. Information is an elusive concept, it can be application specific. For example, in a binary image that contains text only, the necessary information may only involve the text being readable, whereas for a medical image the necessary information may be every minute detail in the original image.

There are two primary types of images compression methods and they are:

Lossless Compression

Lossy Compression.

Lossless Compression

This compression is called lossless because no data are lost, and the original image can be recreated exactly from the compressed data. For simple image such as text-only images.

Lossy Compression.

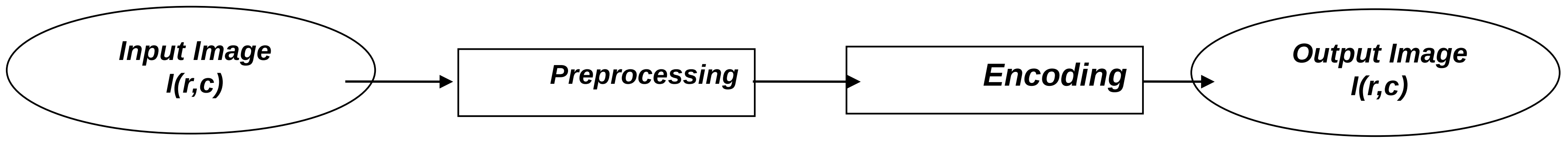
These compression methods are called Lossy because they allow a loss because they allow a loss in actual image data, so original uncompressed image can not be created exactly from the compressed file. For complex images these techniques can achieve compression ratios of 100 0r 200 and still retain in high – quality visual information. For simple image or lower-quality results compression ratios as high as 100 to 200 can be attained.

Compression System Model

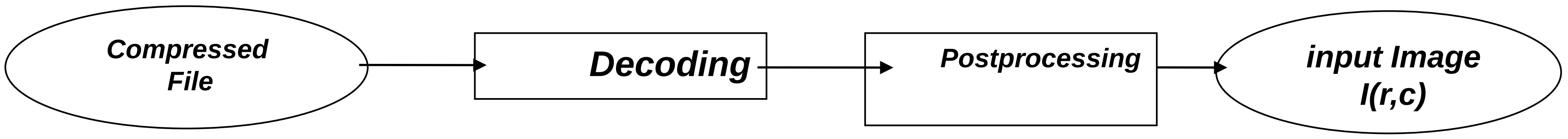
The compression system model consists of two parts: Compressor and Decompressor.

Compressor: consists of preprocessing stage and encoding stage.

Decompressor: consists of decoding stage followed by a post processing stage, as following figure



a. Compression



b. Decompression

Lossles Compression Method

Lossless compression methods are necessary in some imaging applications. For example with medical image, the law requires that any archived medical images are stored without any data loss.

However lossless compression techniques may be used for both preprocessing and postprocessing in image compression algorithms. Additionally, for simple image the lossless techniques can provide perfect compression.

An important concepts here is the ides of measuring the average information in an image, referred to as entropy.

The entropy for N×N image can be calculated by this equation.

Why Data Compression?

Graphical images in bitmap format take a lot of memory

– e.g. For 3 Byte image the size (1024 x 768)pixels x 24 bits-per-pixel = 18,874,368 bits, or 2.4Mbytes.

- But 120 Gbyte disks are now readily available!!
- So why do we need to compress?
- How long did that graphics-intensive web page take to download over your 56kbps modem?

- How many images could your digital camera hold?

– Picture messaging?

- CD (650Mb) can only hold less than 10 seconds of uncompressed video (and DVD only a few minutes)
- We need to make graphical image data as small as

possible for many applications

Types of Compression

- Pixel packing
- RLE (Run-length Encoding)
- Dictionary-based methods
- JPEG compression
- Fractal Image Compression

Some factors to look out for:

- *Lossy* or *lossless* compression?
- What sort of data is a method good at compressing?
- What is its compression ratio?

2. Run-length encoding (RLE)

is a very simple form of data compression in which *runs* of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. ***This is most useful on data that contains many such runs: for example, simple graphic images such as icons and line drawings***.

- For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white [pixels](#) in the blank space, and many short runs of black pixels within the text. Let us take a hypothetical single [scan line](#), with B representing a black pixel and W representing white:
  - **WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW**

- ***If we apply the run-length encoding (RLE) data compression algorithm to the above hypothetical scan line, we get the following: 12WB12W3B24WB14W***
- ***Interpret this as twelve W's, one B, twelve W's, three B's, etc.***

Another example is this:

AAAAAAAAAAAAAAA would encode as 15A

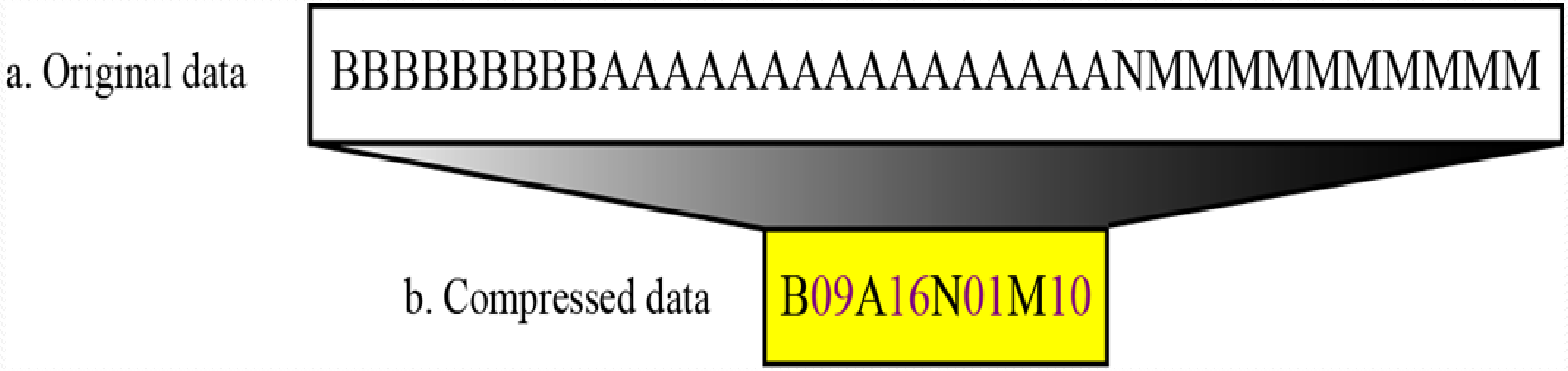
AAAAAAbbbXXXXXt would encode as 6A3b5X1t

So this compression method is good for compressing large expanses of the same color.

Run-length Encoding

Simplest method of compression.

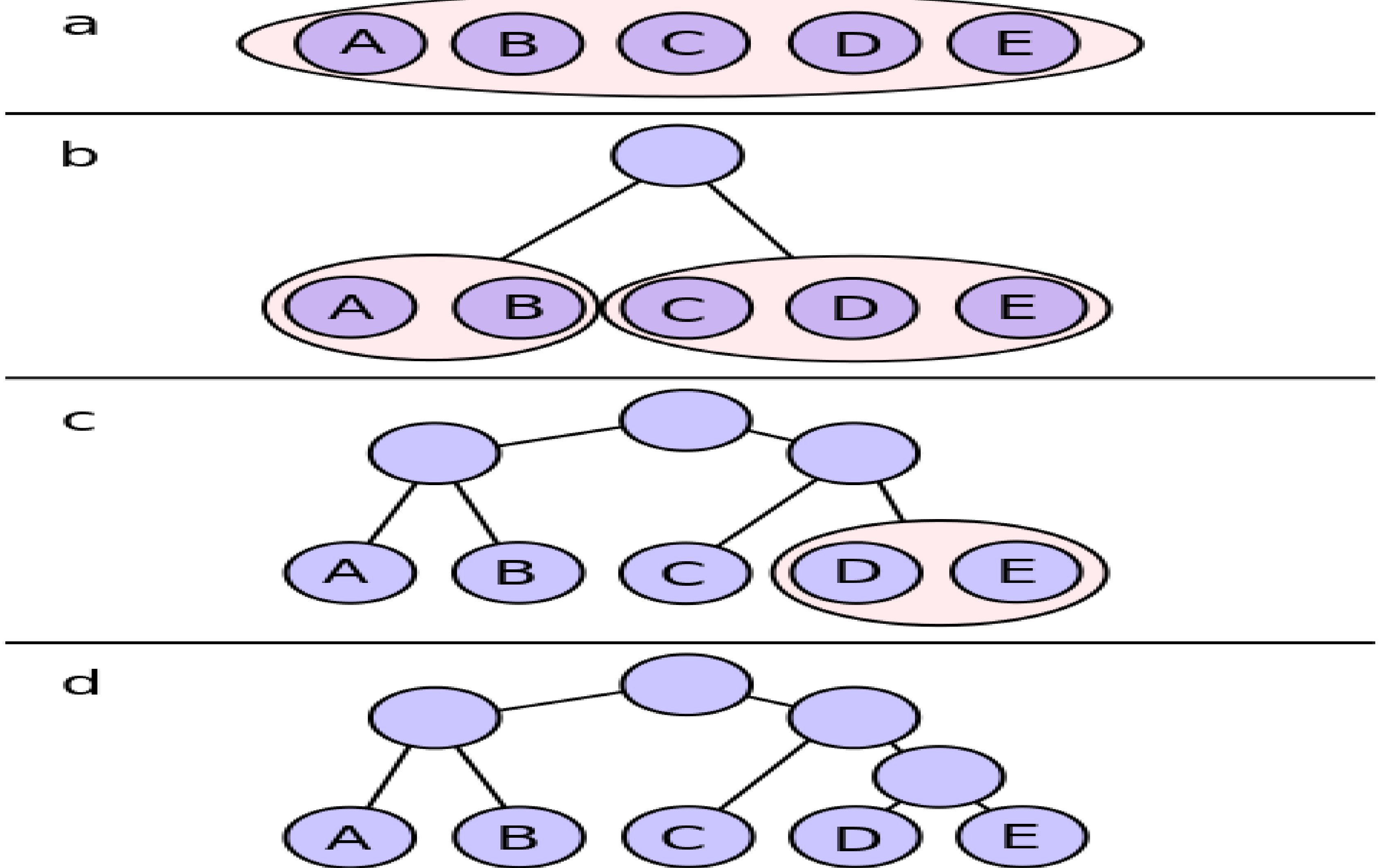
How: replace consecutive repeating occurrences of a symbol by 1 occurrence of the symbol itself, then followed by the number of occurrences.



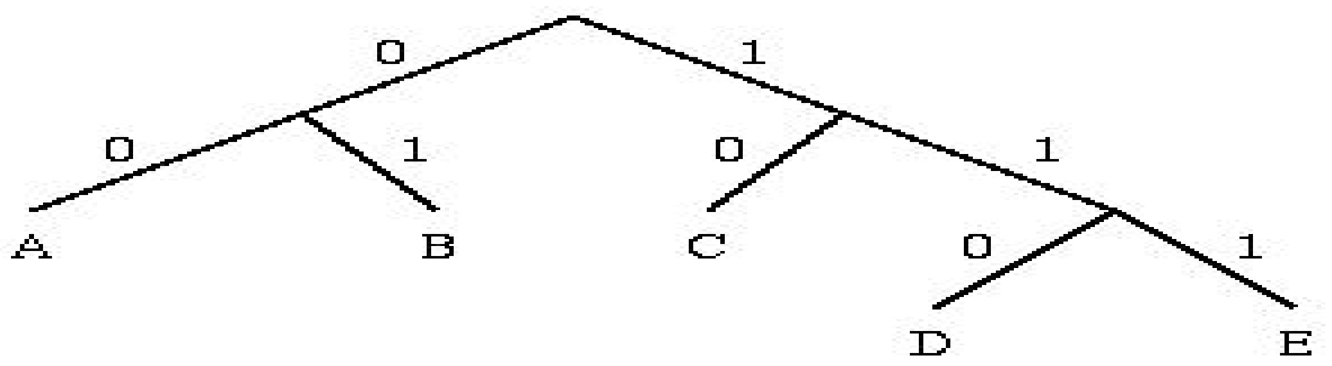
The Shannon-Fano Algorithm

This is a basic information theoretic algorithm. A simple example will be used , to illustrate the algorithm:

Symbol	**A**	**B**	**C**	**D**	**E**
Count	**15**	**7**	**6**	**6**	**5** (**sum=39**)
P	**(15/39)= 0.384**	**(7/39)= 0.179**	**0.153**	**0.153**	**0.1280**



The Shannon-Fano Algorithm cont.



A	15	0.384	00	30
B	7	0.179	01	14
C	6	0.153	10	12
D	6	0.153	110	18
E	5	0.128	111	15

TOTAL (# of bits): 89

Huffman Coding

A bottom-up approach

1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times

(e.g., ABCDE).

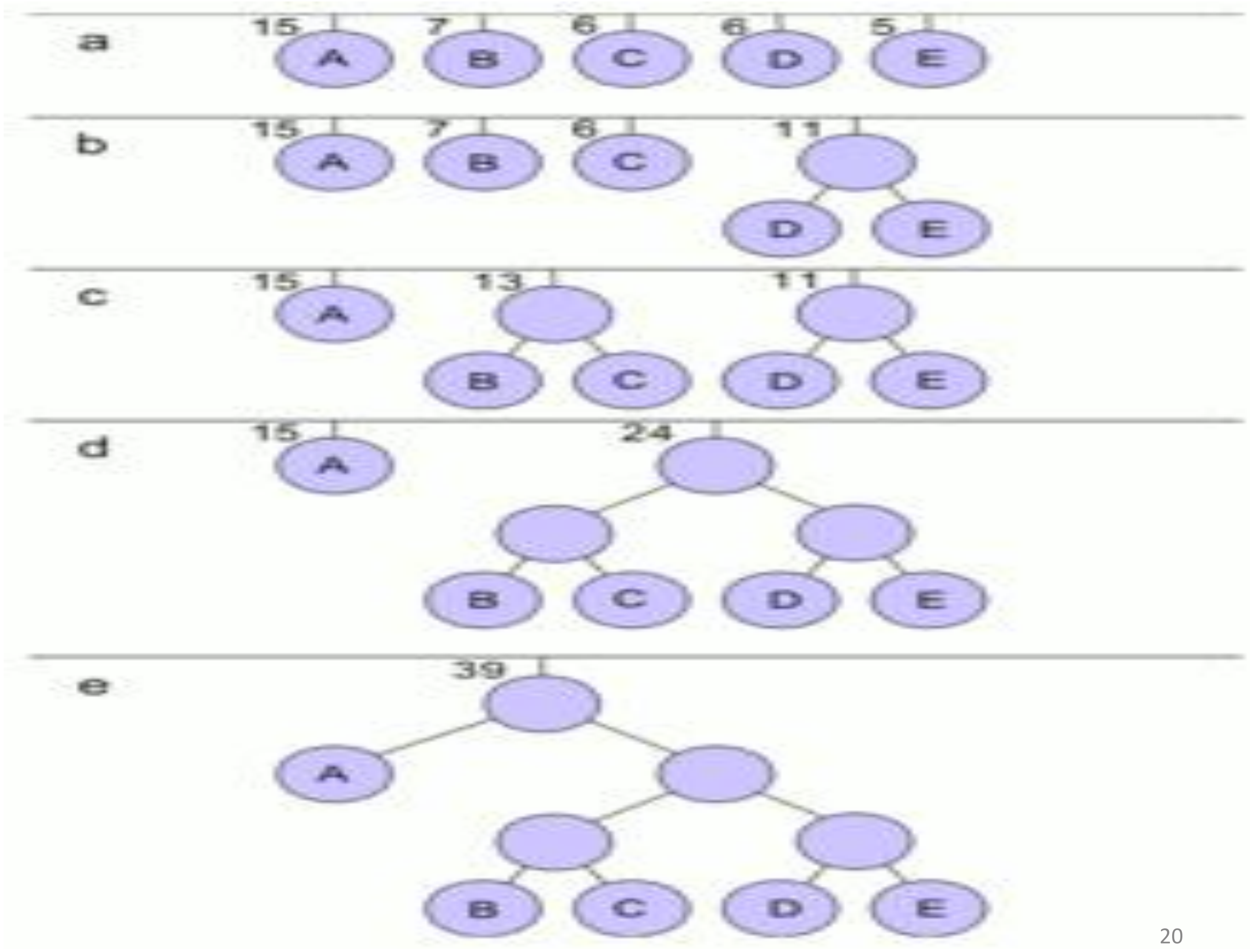
2. Repeat until the OPEN list has only one node left:

(a) From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.

(b) Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.

(c) Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.

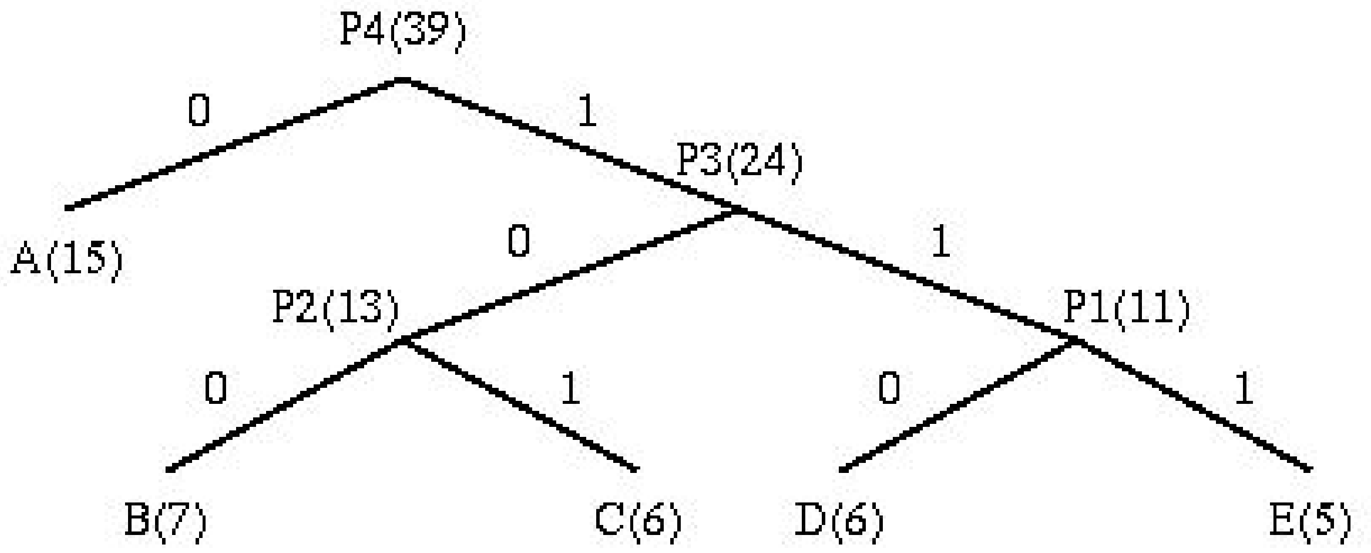
Huffman cont.



Huffman Coding cont.

Symbol	Count	Code	Subtotal (# of bits)
A	15	0	
B	7	100	
C	6	101	
D	6	110	
E	5	111	

TOTAL (# of bits): 87


P4(39)
0
1
P3(24)
A(15)
0
1
P2(13)
P1(11)
0
1
0
1
B(7)
C(6)
D(6)
E(5)