



MULTIMEDIA

Image Compression

Dr. Darin Shafek Salim

2024-2025

Image compression techniques

□ Image compression

is a type of data compression applied to digital images, to reduce their cost for *storage or transmission*

- The **reduced file** is called the *compressed file* and is used to reconstruct the image
- Resulting in the **decompressed image** is the original image, is called the *uncompressed image file*.

$$\text{Compression Ratio} = \frac{\text{Uncompressed File Size}}{\text{Compressed File Size}} = \frac{\text{size}_U}{\text{size}_C}$$

Image compression techniques



8.9M



68.34K

Example

If you know that original image is **256×256** pixel, single band (**gray scale**), **8-bit per pixel**. After compression the image file is **6,554 byte**.

Calculate the compression ratio

Solution

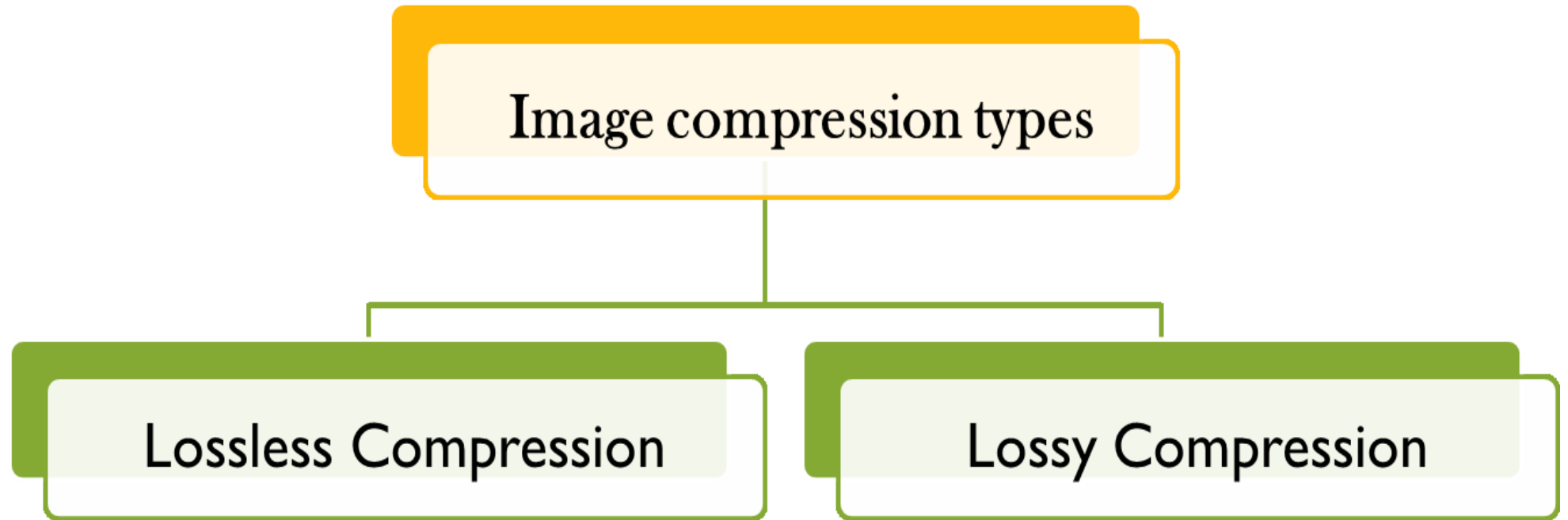
Size U = Size of image = $256 * 256 * 1 = 65,536$ bytes

$$\text{Compression ratio} = \frac{\text{Size U}}{\text{Size C}} = \frac{65536}{6554} = 9.999 = 10$$

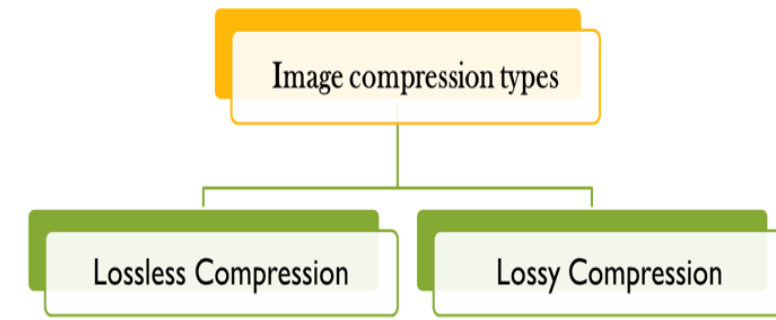
This is called a “10 to 1” compression

Note: repeat the previous question for image with three bands.

Image compression types

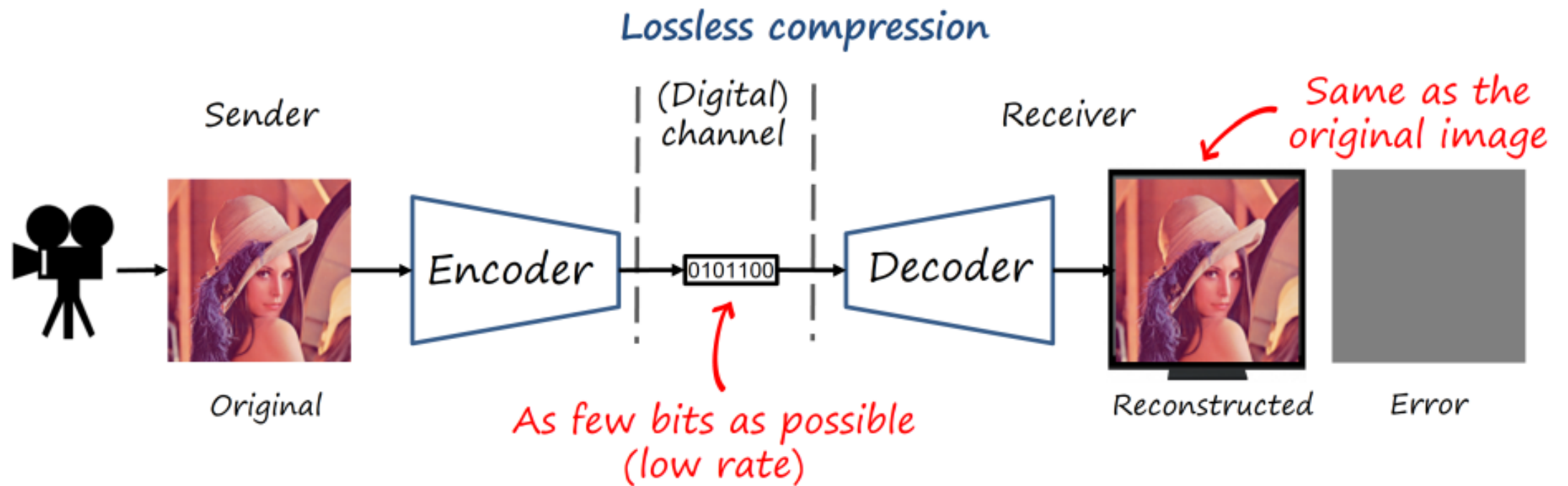


Lossless Compression

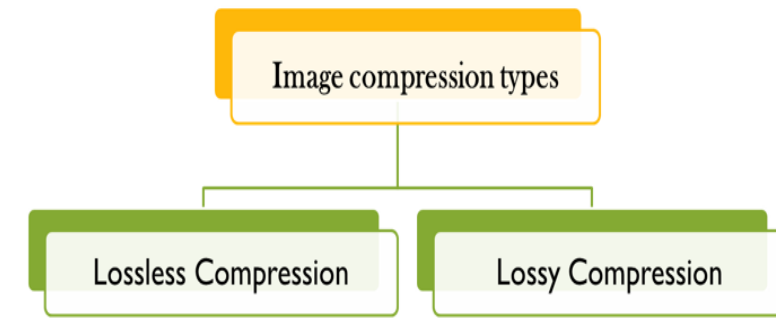


- ❑ This compression is called **lossless** because **no data are lost** (with no loss in image quality)
- ❑ And the original image can **be recreated exactly** from the compressed data
- ❑ Lossless compression is generally used for **text or spreadsheet** files, where losing words or financial data could pose a problem
- ❑ The Graphics Interchange File (**GIF**) is an image format used lossless compression

Lossless Compression



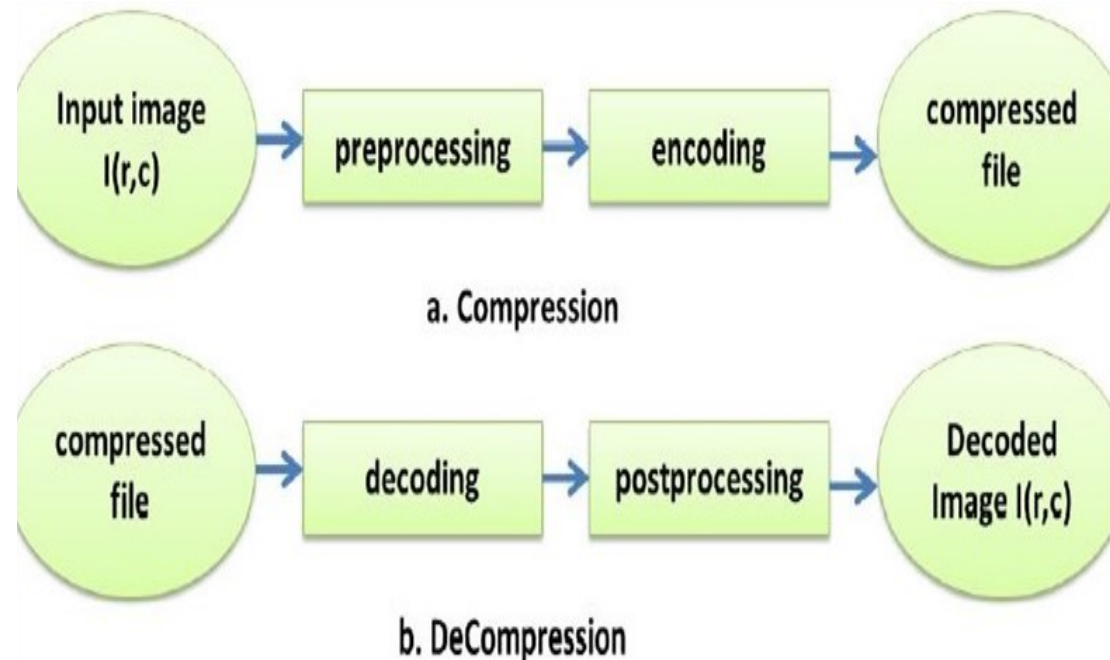
Lossy Compression



- ❑ Lossy Compression allows **a loss** in some of data images.
- ❑ The uncompressed image cannot be same the original image file.
- ❑ Lossy methods can provide high degrees of compression and result in smaller compressed files.

Compression System Model

- The compression system model consists of two parts:
 - The compressor (**Encoding**) and
 - The decompressor (**Decoding**).



Lossless Compression Algorithms

- 1) Run-length Encoding (RLE)
- 2) Simple Repetition Suppression
- 3) Pattern Substitution
- 4) Shanon-Fano Method
- 5) Huffman Code

Run-length Encoding (**RLE**)



RLE is a simple form of data compression where consecutive identical characters or data elements are stored as a single character and its count. This is particularly useful for data with many repeating elements, like images with large areas of the same color.

Example

Use RLE TO code the string "AAABBCDDDD"

Solution

"AAABBCDDDD" would be encoded as "A3B2C1D3".

Run-length Encoding (**RLE**)

1. What is Run-Length Encoding (RLE)?
2. What are the benefits of using Run-Length Encoding?
3. What are the drawbacks of using Run-Length Encoding?
4. In what situations is Run-Length Encoding useful?

Run-length Encoding (**RLE**)



1. What is Run-Length Encoding (RLE)?

Solution

RLE is a straightforward data compression technique that replaces consecutive, identical data elements with a single element and its count.

2. What are the benefits of using Run-Length Encoding?

Solution

- ✓ Simple and easy to implement.
- ✓ Can be very efficient for data with long runs of repeating elements.

Run-length Encoding (**RLE**)



3. What are the drawbacks of using Run-Length Encoding?

Solution

Inefficient for data without many repeating sequences. Can actually increase the data size in some cases.

4. In what situations is Run-Length Encoding useful?

Solution:

RLE is commonly used in image compression (especially for simple images like icons or line drawings) and in certain data storage scenarios..

2

- ## Example

Solution

5

Pattern Substitution

- ✓ This is a simple form of statistical encoding.
- ✓ Here we substitute a frequently repeating pattern(s) with a code.
- ✓ The code is shorter than the pattern giving us compression.
- ✓ The simplest scheme could employ predefined codes: Usually need to have a special flag to denote when the repeated token appears.

Example

Replace all occurrences of pattern of characters 'and' with the predefined code '&'.

Solution

and you and I **becomes:** & you & I

Shanon-Fano Method

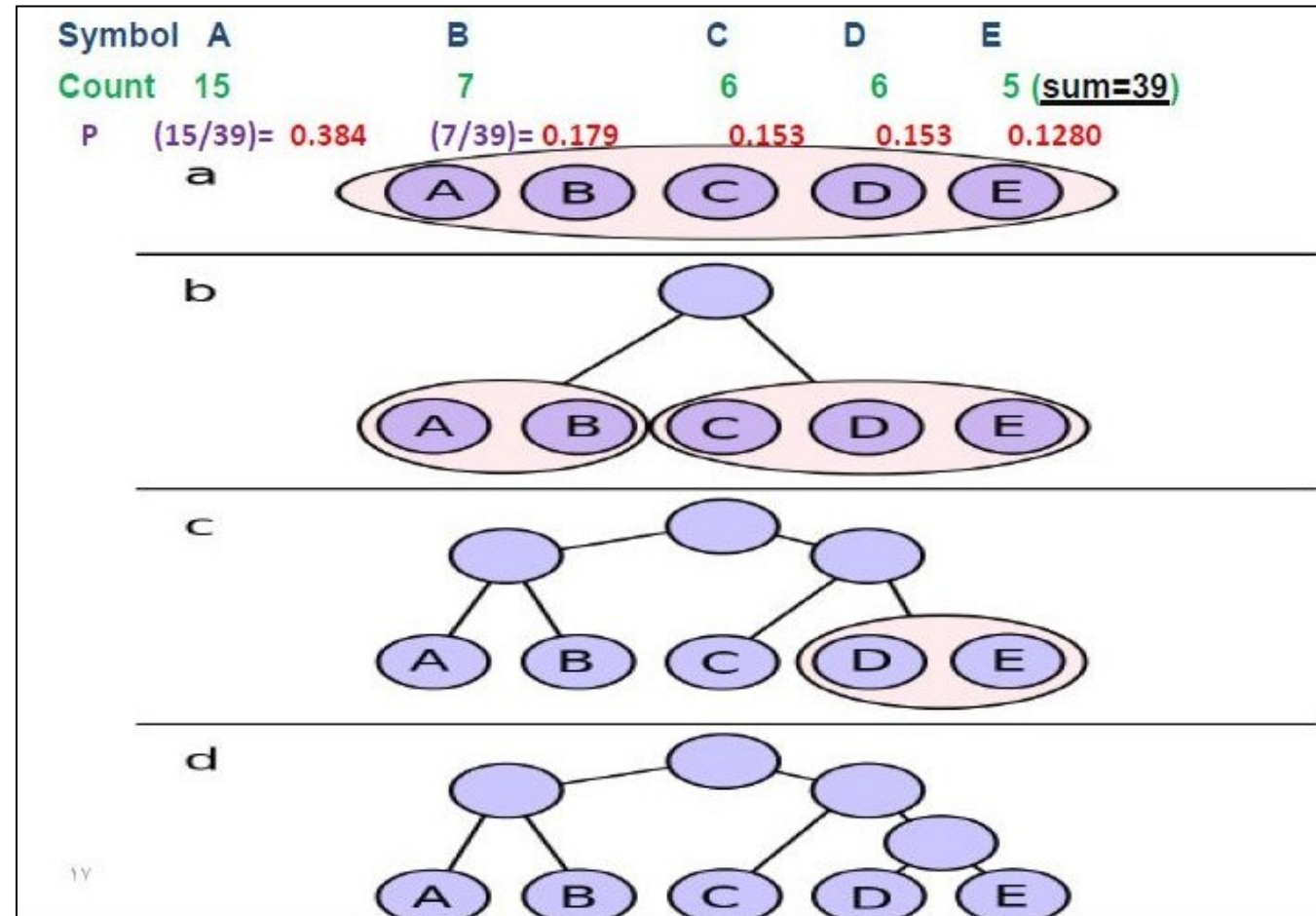
❑ Definition

- Shannon-Fano coding is a variable-length prefix coding technique for lossless data compression. It assigns shorter codes to more frequent symbols and longer codes to less frequent ones.
- The method involves **sorting symbols by frequency**, recursively **dividing** them into **two halves** with roughly equal total probability, and **assigning '0' and '1' bits** to each half. This process creates a binary tree where code words are determined by the path from the root to the leaf representing the symbol.

Shanon-Fano Method

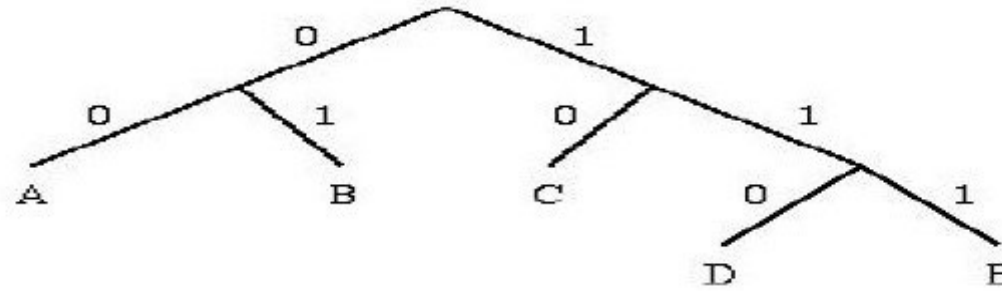
Example

Step 1



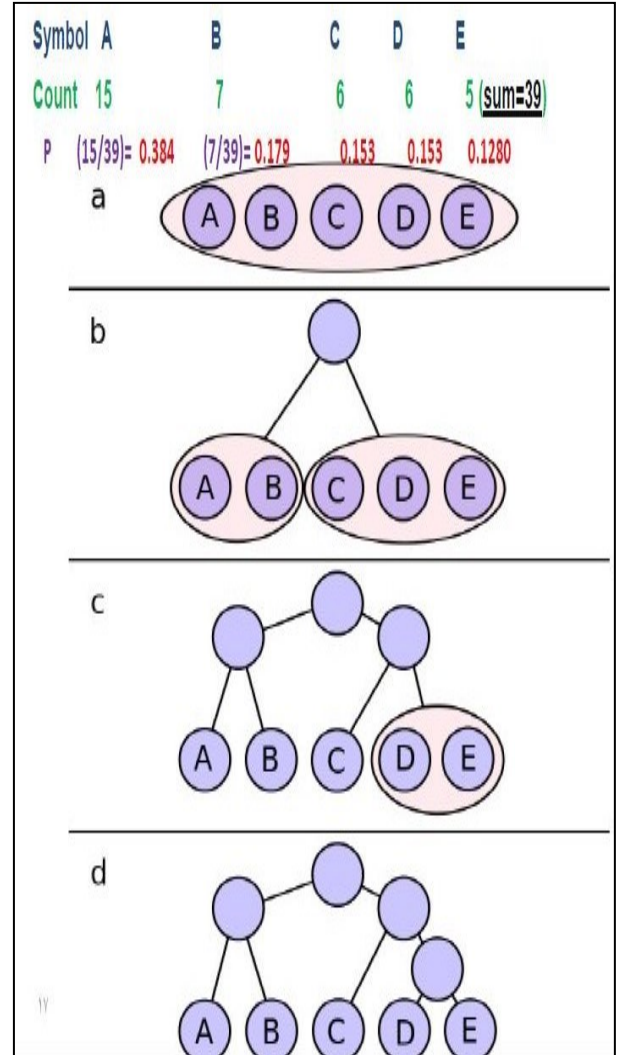
Shanon-Fano Method

Example Step 2



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



❑ Advantages:

- ✓ Simple and fast
- ✓ No need for a pre-built tree
- ✓ Variable-length codes

❑ Disadvantages

- ✓ Not always optimal
- ✓ Sensitive to symbol order
- ✓ Less efficient than Huffman coding

❑ What is the benefit of “No need for a pre-built tree”?

- ✓ Flexibility with dynamic data: This feature is very useful when the data is dynamic (i.e., changes over time). There is no need to rebuild a new tree every time the data changes.
- ✓ Efficiency in resources: Not needing to store a pre-defined tree saves memory space.

Example

Let's assume we have a simple grayscale image with the following pixel values and their corresponding frequencies:

Use the Shannon-Fano method To compress the image

Pixel Value	Frequency
0	50
1	100
2	12
3	25
4	6
5	3
6	1

Huffman Code

□ Definition

The Huffman coding algorithm comprises two steps, **reduction** and **splitting**.

Reduction

- ✓ List the symbols in descending order of probability.
- ✓ Reduce the r least probable symbols to one symbol with a probability equal to their combined probability.
- ✓ Reorder in descending order of probability at each stage.
- ✓ Repeat the reduction step until only two symbols remain.

Splitting

- Assign $r, \dots, 1, 0$ to the r final symbols and work backwards.
- Expand or lengthen the code to cope with each successive split.

Huffman Code **Example:**

Design Huffman codes for $A = \{a_1, a_2, \dots, a_5\}$, having the probabilities $\{0.2, 0.4, 0.2, 0.1, 0.1\}$.

Symbol	Step 1	Step 2	Step 3	Step 4	Codeword
a_2	0.4	0.4	0.4	0.6 0	1
a_1	0.2	0.2	0.4	0.4 1	01
a_3	0.2	0.2	0.2		000
a_4	0.1	0.2			0010
a_5	0.1				0011

Huffman Code **Example:**

Letter	Probability	Codeword
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

The average code word length:

$$L_{avg} = \sum_{i=0}^4 P_i * l_i$$

$$L = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = \mathbf{2.2 \text{ bits/symbol}}$$

Huffman Code

□ Advantages:

- ✓ Huffman coding is a **variable-length** coding, which means that more frequent characters have shorter codes.
- ✓ Huffman coding is an **optimal** coding, which means that it provides the best possible compression for the data.

□ Disadvantages

- ✓ Can be computationally intensive
- ✓ Requires storing the tree for decoding, which increases the data size.
- ✓ May be inefficient for data with high diversity and no repetition

*Thank
you*

