# CHAPTER SEVEN

## *Lossless Compression Algorithms*

# INTRODUCTION

- An important aspect of communication is transfer of data from the creator to the recipient.
- ***Compression*** in computer terms means reducing the physical size of data such that it occupies *less storage space* and *memory*.
- It is done by either ***removing the redundancies*** or in some case ***removing the part of the data*** but ***still maintaining the integrity of the data***.

# PURPOSES OF DATA COMPRESSION

- Data storage and transmission cost money.
- This cost increases with the amount of data available.
- This cost can be reduced by processing the data so that it takes less memory and less transmission time.

# COMPRESSION REQUIREMENTS

- Compressions in multimedia systems are subjected to certain constraints.
  - The quality of the reproduced data should be adequate for applications.
  - The complexity of the technique used should be minimal, to make a cost-effective compression technique.
  - The processing of the algorithm should not take too long.
  - Various data rates should be supported.
  - It should be possible to generate data on one multimedia system and reproduce data on another system.

- As many applications exchange multimedia data using communication networks, the compatibility of compression is required.

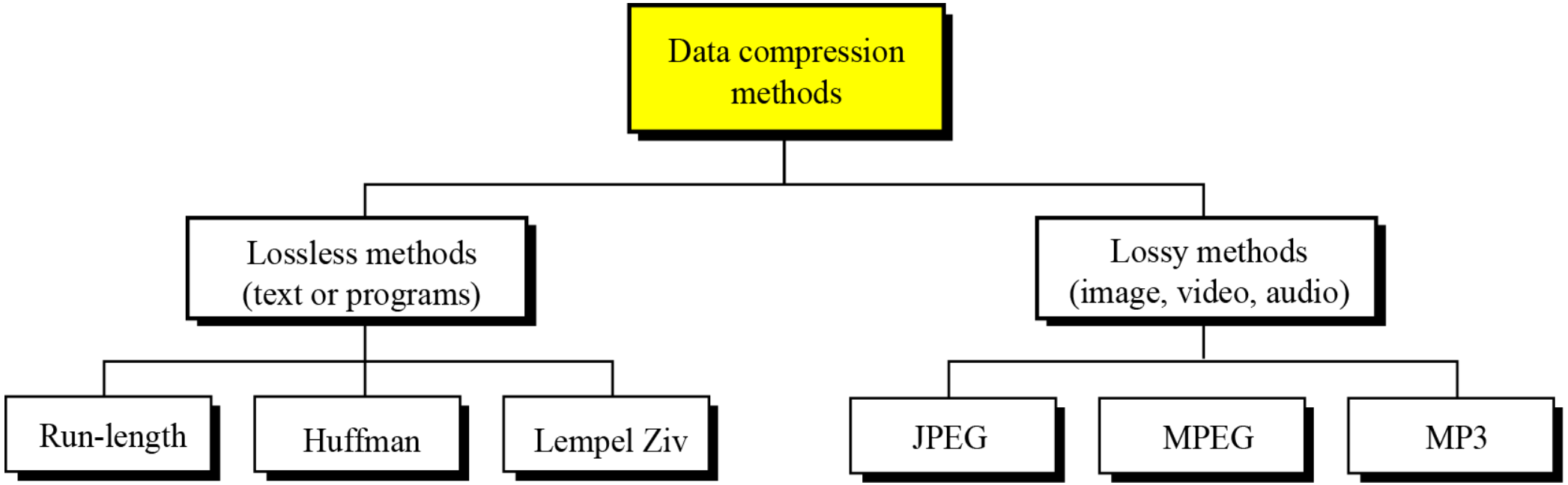
- Standards like CCITT, ISO and MPEG are used to achieve this compatibility.

# COMMON COMPRESSION METHODS

- An array of compression techniques has been set by the **CCITT** (International Consultative Committee for Telephone and Telegraph)Group.
- There are two types of compression:
  - **Lossless Compression**
  - **Lossy Compression**

# COMMON COMPRESSION METHODS CONT…

- Data compression is about storing and sending a smaller number of bits.



# LOSSLESS COMPRESSION

- In ***lossless compression***, data are not altered or lost in the process of *compression* or *decompression*.
- Lossless techniques enable exact reconstruction of the original document from the compressed information.
- This compression technique is used for *text documents, databases,* and *text-related objects*.
- Lossless methods are used when we can't afford to lose any data: legal and medical documents, computer programs.
- E.g. LZW, Run Length Encoding, Huffman.

# LOSSY COMPRESSION

- There is loss of some information.
- The loss of this data is such that the object looks more or less like the original.
- This method is used where absolute data accuracy is not essential.
- Lossy compression is the most commonly used compression type.
- This compression technique is used for image documents, audio, and video objects.
- E.g. JPEG and MPEG

# 1. LOSSLESS COMPRESSION ALGORITHMS

## *1.1. Run-length coding (RLC)*

- RLC is a very simple form of data compression in which *runs of data 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 instance, simple graphic images such as *icons, line drawings,* and *animations*.
- Used for compressing text and black and white image.

# RUN-LENGTH CODING (RLC) CONT…

- Here the consecutive repeated string of characters is replaced by 2 bytes
  - Where the first byte contains the number which represent the no of times the character is repeated &
  - The second byte contains the represented character it self.
- ***Disadvantage***: sometimes more no of bits for representing the compressed data as compared to the original data creates negative compression.!

# 1. LOSSLESS COMPRESSION ALGORITHM CONT..

## ***1.2. Variable-length code***

- In coding theory a variable-length code is a code which *maps source symbols to a variable number of bits*.
- Certain part of the data appear more frequently than others, so it would be more economic to assign fewer bits as their codes.
- The more frequent appearing symbols are coded with fewer bits per symbol and vice-versa.
- Some examples of well-known VLC strategies are Huffman coding,  Lempel–Ziv coding and Arithmetic coding.

# BASICS OF INFORMATION THEORY

- According to the famous scientist Shannon, of Bell Labs, the entropy of an information source with alphabet S={s1,s2,...,sn} is defined as:
- Entropy is the measure of information content in a message

$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

$p_i$ – probability that symbol $s_i$ will occur in $S$.

$\log_2 \frac{1}{p_i}$ – indicates the amount of information ( *self-information* as defined by Shannon) contained in $s_i$, which corresponds to the number of bits needed to encode $s_i$.

$$= -\sum_{i=1}^{n} p_i \log_2 p_i$$

- As can be seen in the above equation the entropy is a weighted-sum of terms ;
- hence it represents ***the average amount of information contained per symbol in the source S***.
- The entropy specifies the lower bound for the average number of bits to code symbol in S.

# Variable-length Coding(VLC)

## ***Shannon–Fano Algorithm***

- The Shannon–Fano algorithm was independently developed by Shannon at Bell Labs and Robert Fano at MIT.
- To illustrate the algorithm, let us suppose the symbols to be coded are the characters in the word “HELLO”.
- The frequency count of the symbols is

| Symbol | H | E | L | L | O |
|---|---|---|---|---|---|
| Count | 1 | 1 | 2 | | |

# Shannon–fano Algorithm Cont..

- A top down approach
  1. *Sort the symbols according to the frequency count of their occurrences.*
  2. *Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.*
- A natural way of implementing is to build a binary tree.
- Lets assign 0 to the left branches and 1 to the right branches.

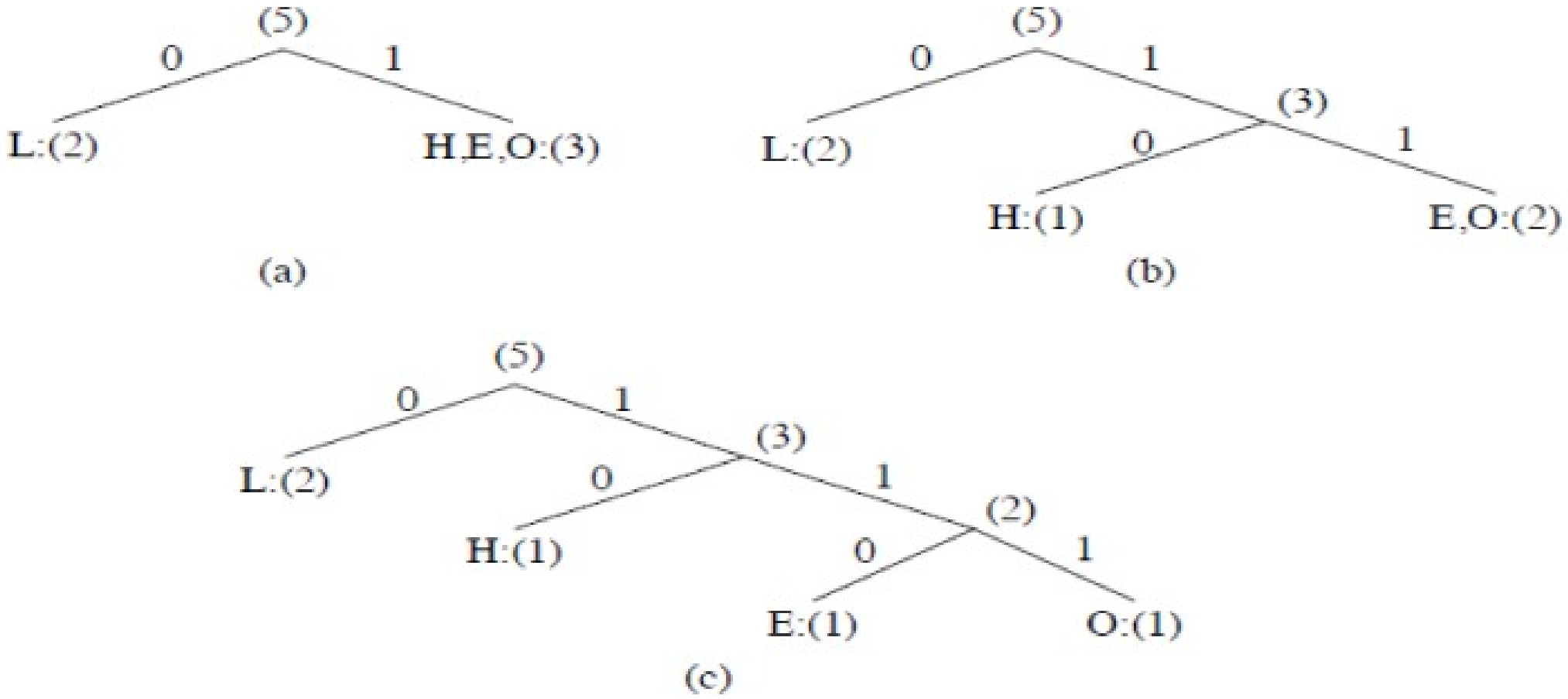



Fig. 7.3: Coding Tree for HELLO by Shannon-Fano.

## Result of Performing Shannon-Fano on HELLO

| Symbol | Count | $\log_2 \frac{1}{p_i}$ | Code | # of bits used |
|---|---|---|---|---|
| L | 2 | 1.32 | 0 | 2 |
| H | 1 | 2.32 | 10 | 2 |
| E | 1 | 2.32 | 110 | 3 |
| O | 1 | 2.32 | 111 | 3 |
| TOTAL number of bits: | | | | 10 |

$$\eta = p_L \cdot \log_2 \frac{1}{p_L} + p_H \cdot \log_2 \frac{1}{p_H} + p_E \cdot \log_2 \frac{1}{p_E} + p_O \cdot \log_2 \frac{1}{p_O}$$
$$= 0.4 \times 1.32 + 0.2 \times 2.32 + 0.2 \times 2.32 + 0.2 \times 2.32 = 1.92$$

- *This suggests the minimum average no of bits to code each character in the word HELLO would be 1.92.*
- *Here S-F algorithm uses an average of 10/5=2 bits to code each symbol which is fairly close to the lower bound of 1.92.*
- *Apparently the result is satisfactory.*

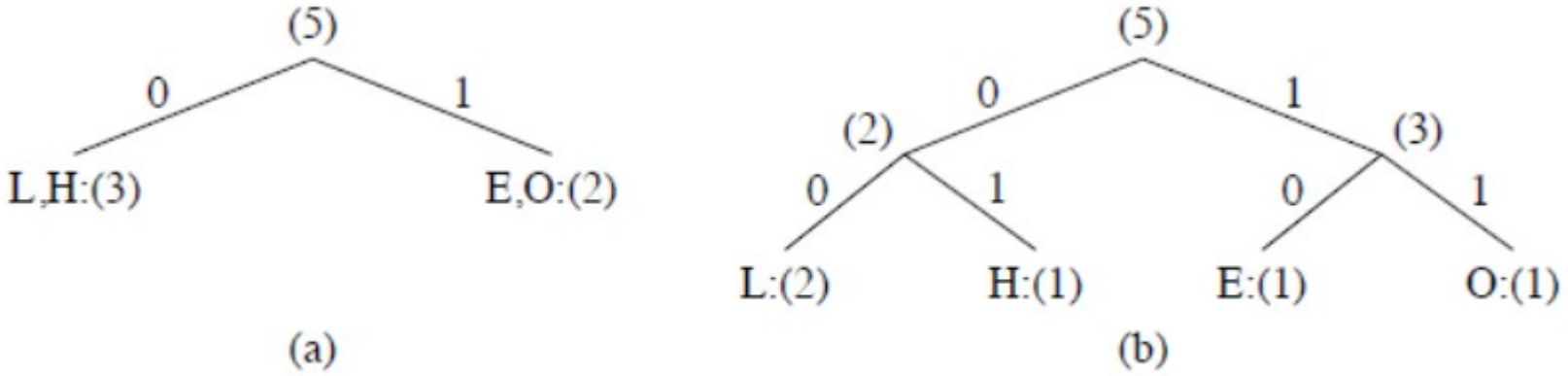



Fig. 7.4 Another coding tree for HELLO by Shannon-Fano.

| Symbol | Count | $\log_2 \frac{1}{p_i}$ | Code | # of bits used |
|---|---|---|---|---|
| L | 2 | 1.32 | 00 | 4 |
| H | 1 | 2.32 | 01 | 2 |
| E | 1 | 2.32 | 10 | 2 |
| O | 1 | 2.32 | 11 | 2 |
| TOTAL number of bits: | | | | 10 |

Table 7.2: Another Result of Performing Shannon-Fano on HELLO (see Fig. 7.4)

# Limpel-Ziv-Welch(LZW) Algorithm

- It is a dictionary based compression technique.
- Unlike VLC it uses fixed length code words to represent symbols/characters that commonly occur together.
- The LZW encoder and decoder build up the same dictionary dynamically while receiving the data.
- LZW proceeds by placing longer and longer repeated entries into a dictionary, then emitting the code for an element rather than the string it self, if the element has already been placed in the dictionary.

# Limpel-Ziv-Welch(LZW) Algorithm cont...

☛*Algorithm*=> **LZW Compression**

```
BEGIN
   s = next input character;
   while not EOF
       { c = next input character;

         if s + c exists in the dictionary
            s = s + c;
         else
           { output the code for s;
             add string s + c to the dictionary with a new code;
             s = c;
           }
       }
   output the code for s;
END
```

# Limpel-Ziv-Welch(LZW) Algorithm cont…

- ***Example***: LZW compression for string "ABABBABCABABBA"
  - Lets start with a very simple dictionary(string table) initially containing only 3 characters, with codes as follows.
  - Now if the input string is "ABABBABC… LZW compression algorithm works as f…

| code | string |
|---|---|
| 1 | A |
| 2 | B |
| 3 | C |

# Limpel-Ziv-Welch(LZW) Algorithm cont…

| s | c | output | code | string |
|---|---|---|---|---|
| | | | 1 | A |
| | | | 2 | B |
| | | | 3 | C |
| A | B | 1 | 4 | AB |
| B | A | 2 | 5 | BA |
| A | B | | | |
| AB | B | 4 | 6 | ABB |
| B | A | | | |
| BA | B | 5 | 7 | BAB |
| B | C | 2 | 8 | BC |
| C | A | 3 | 9 | CA |
| A | B | | | |
| AB | A | 4 | 10 | ABA |
| A | B | | | |
| AB | B | | | |
| ABB | A | 6 | 11 | ABBA |
| A | EOF | 1 | | |

- The output codes are:

  1 2 4 5 2 3 4 6 1

- Therefor instead of sending 14 characters, only 9 codes need to be sent.
- If we assume each character or code is transmitted as a byte, compression ratio=14/9 = 1.56

# Limpel-Ziv-Welch(LZW) Algorithm cont…

☛*Algorithm*=> **LZW Decompression**

```
BEGIN
   s = NIL;
   while not EOF
       {
         k = next input code;
         entry = dictionary entry for k;
         output entry;
         if (s != NIL)
            add string s + entry[0] to dictionary with a new code;
         s = entry;
       }
END
```

# Limpel-Ziv-Welch(LZW) Algorithm cont...

- ***Example***: LZW decompression for string "ABABBABCABABBA"
  - Input codes to the decoder are

    1 2 4 5 2 3 4 6 1 .
  - The initial string table is identical to what is used by the encoder.
  - Apparently, the output string is "ABABBABCABABBA", a truly

| s | k | entry/output | code | string |
|---|---|---|---|---|
| | | | 1 | A |
| | | | 2 | B |
| | | | 3 | C |
| NIL | 1 | A | | |
| A | 2 | B | 4 | AB |
| B | 4 | AB | 5 | BA |
| AB | 5 | BA | 6 | ABB |
| BA | 2 | B | 7 | BAB |
| B | 3 | C | 8 | BC |
| C | 4 | AB | 9 | CA |
| AB | 6 | ABB | 10 | ABA |
| ABB | 1 | A | 11 | ABBA |
| A | EOF | | | |

# Huffman Coding

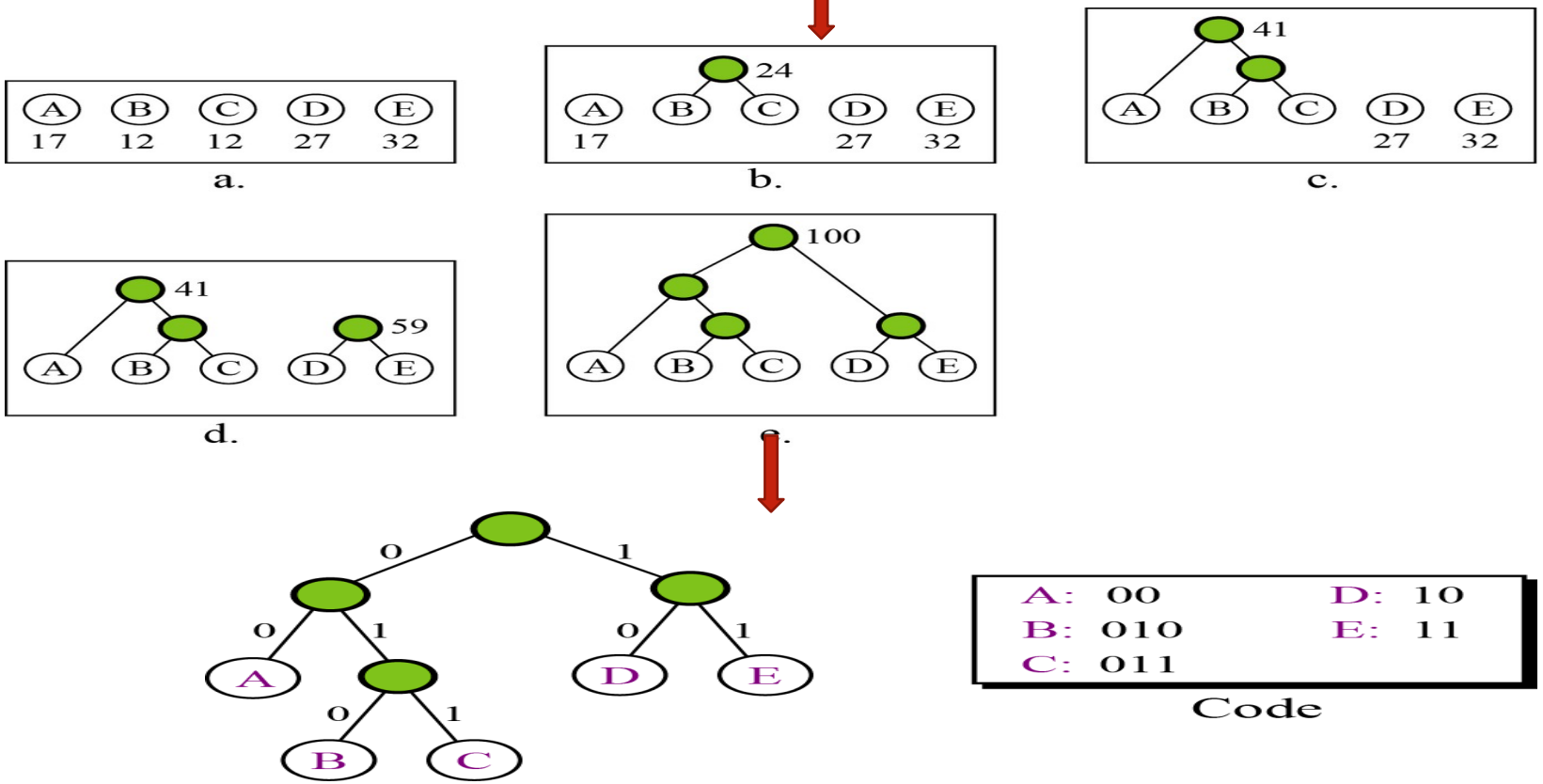
- Assign fewer bits to symbols that occur more frequently and more bits to symbols appear less often.
- There’s no unique Huffman code and every Huffman code has the same average code length.
- ***Algorithm***:
  - Calculate each frequency of the letters.
  - Order in their frequency the lowest frequency come first.
  - Construct its tree structure (tree for binary code) starting from lowest frequency (lowest frequency is the left side).

# Huffman Coding

☛ ***Example***

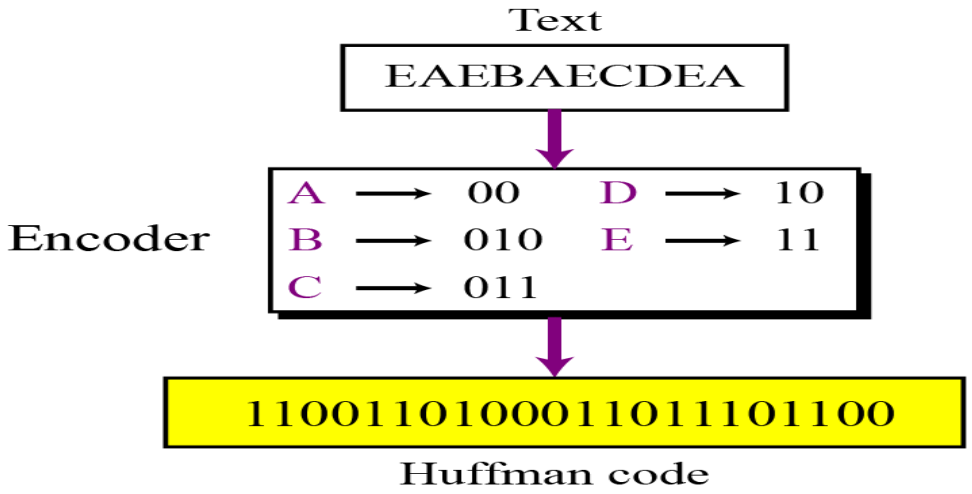
Frequency of characters

| Character | A | B | C | D | E |
|---|---|---|---|---|---|
| Frequency | 17 | 12 | 12 | 27 | 32 |

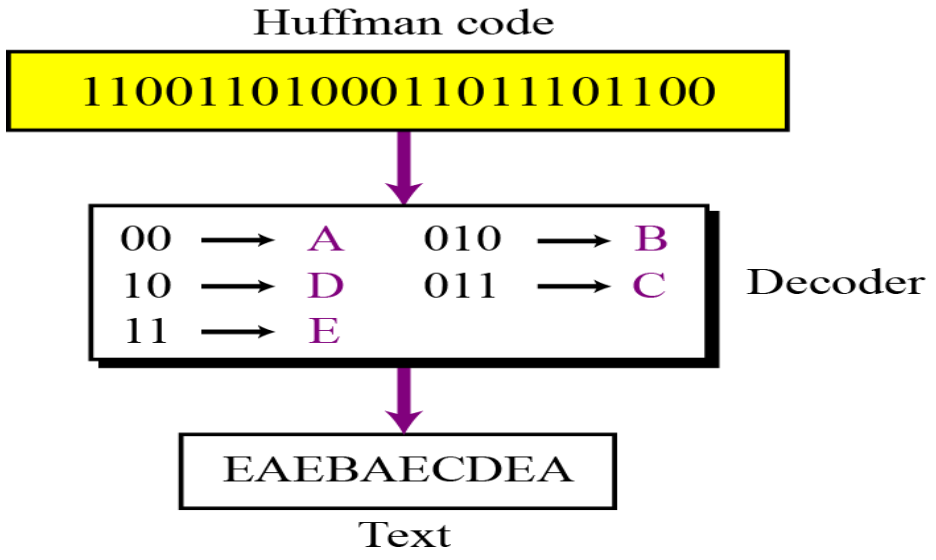


# Huffman Coding

☛Coding




☛Decodin
g



# Class Activity 1

- Compress and decompress the following data using LZW algorithm.
  - **DDAACBDDDAABACDDCC**

- Compress and decompress the following data using Huffman coding method
  - **MISSISSIPPI**