# Basic Compression

Rynson W.H. Lau

CS4185 Multimedia Technologies and Applications

# Need for Compression

- As we have discussed in Sampling, the size of a single 4-minute song is about 40MB and that of a 90-minute HDTV movie is about 782GB.
- This is a lot to store or to transmit.
- Apart from audio and video files, we may also have other file types, such as text and image files.
- We can save memory if we compress these files to smaller sizes before we store or transmit them.

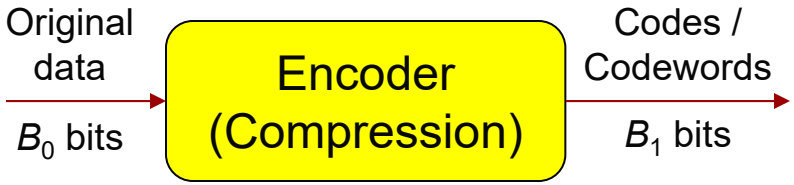
# Data Compression

- ***Data Compression*** is the process of encoding a digital file using less memory.


A general data compression scheme

- A ***Codec*** is an encoder/decoder scheme.

- ***Compression Ratio*** is defined as the ratio between the input data size (in terms of bits) to the output data size (in terms of bits).



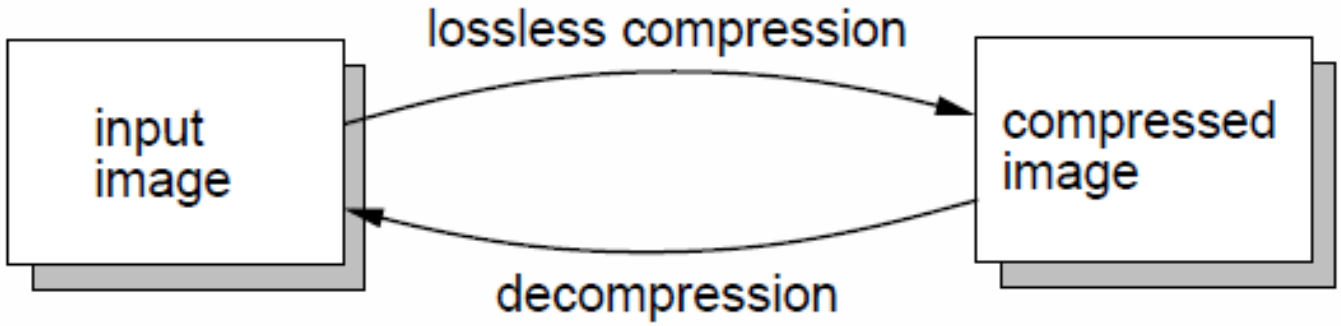

$$\text{Compression Ratio} = \frac{B_0}{B_1}$$

- $B_0$ = Total number of bits required to represent the data *before* compression
- $B_1$ = Total number of bits required to represent the data *after* compression

# Lossless vs. Lossy Compression

- With lossless compression, if a compressed file is uncompressed, the output file will be exactly the same as the file before compression.

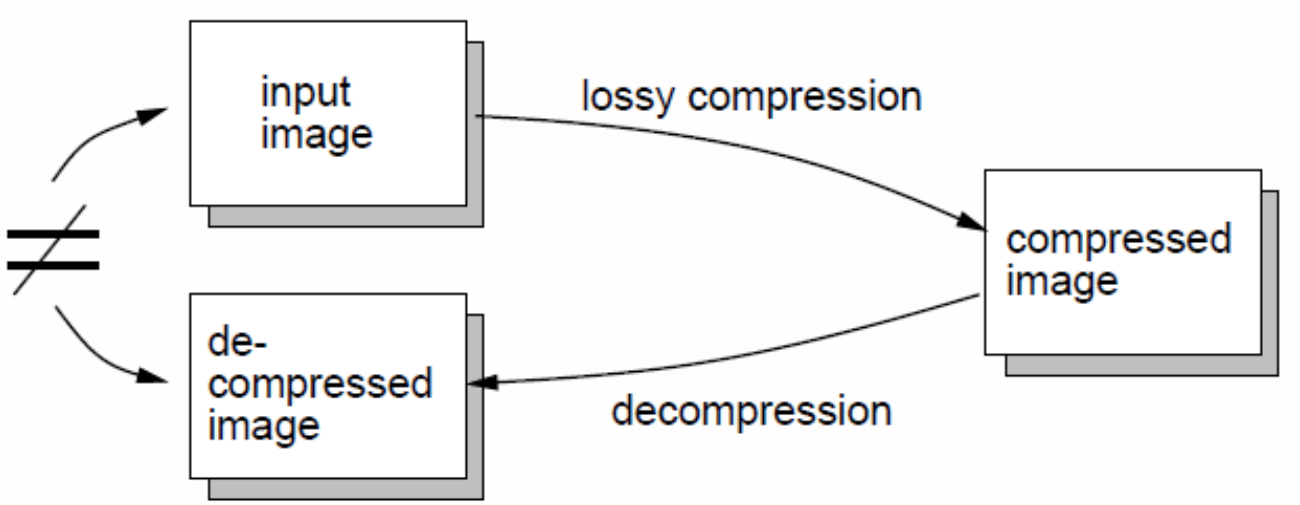
  Examples: **winzip** and **winrar**



- The advantage of lossless compression is that there is no information loss during the compression process.
- The limitation is that the amount of compression is usually small.
- Lossless compression methods are usually used for compressing text and data files, where any information loss is not acceptable.

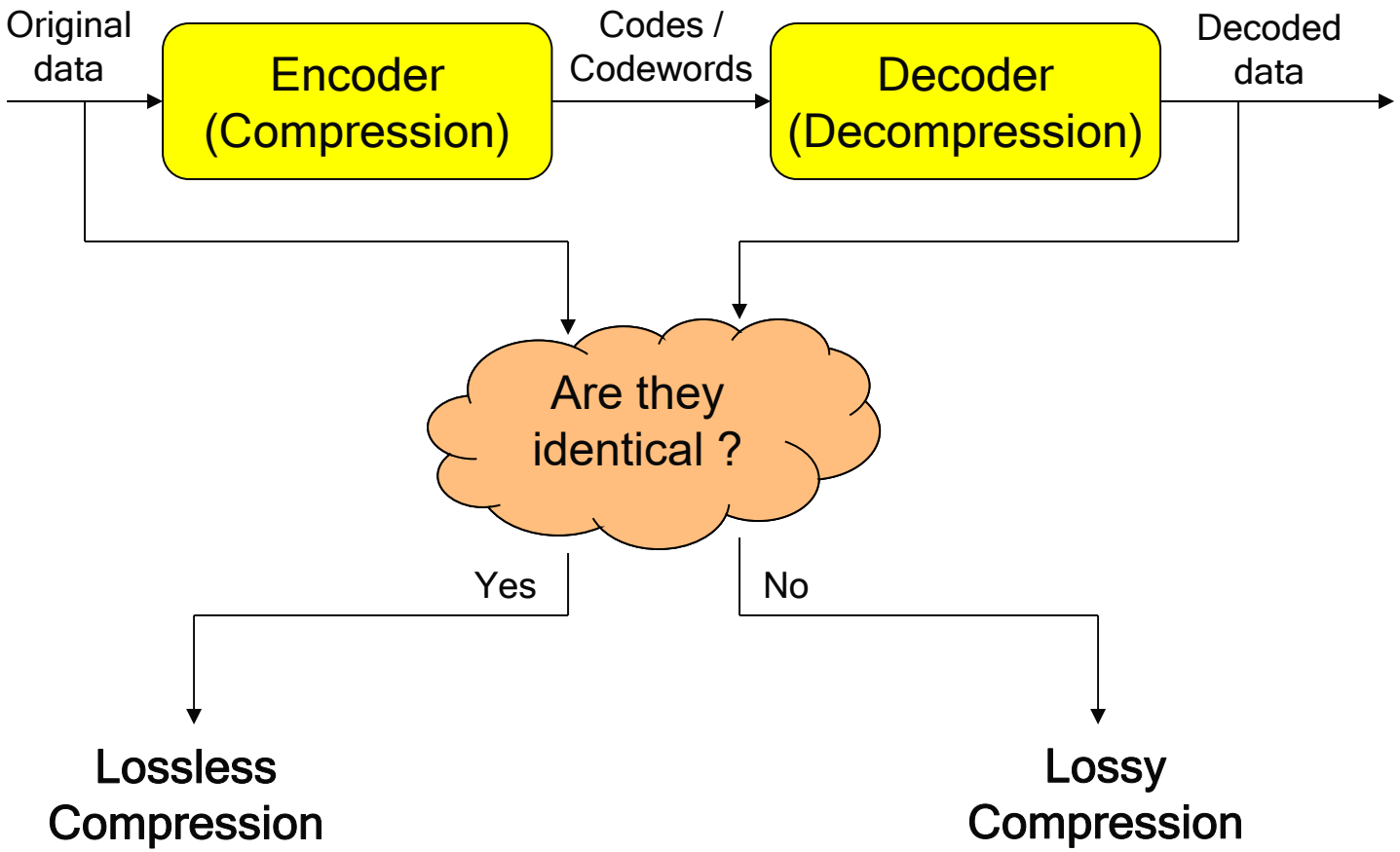
- With lossy compression, redundant and less important information are thrown away in order to obtain a smaller output file size.

  Examples: **JPEG**, **MPEG** and **MP3**



- The advantage of lossy compression is that the compression ratio can be very high. Therefore, the output file size can be very small.
- The disadvantage is that it throws away some information. Hence, information is lost during the compression process, and cannot be recovered.
- Lossy compression methods are usually used for compressing images and videos.

Summary:


Original data
Encoder (Compression)
Codes / Codewords
Decoder (Decompression)
Decoded data
Are they identical ?
Yes
No
Lossless Compression
Lossy Compression

# Basic Information Theory

- In information theory, ***Entropy*** is used to indicate the amount of information available from a given file.
- Entropy is computed as:

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

$$= \sum_{i=1}^{n} \left(- p_i \log_2 p_i\right)$$

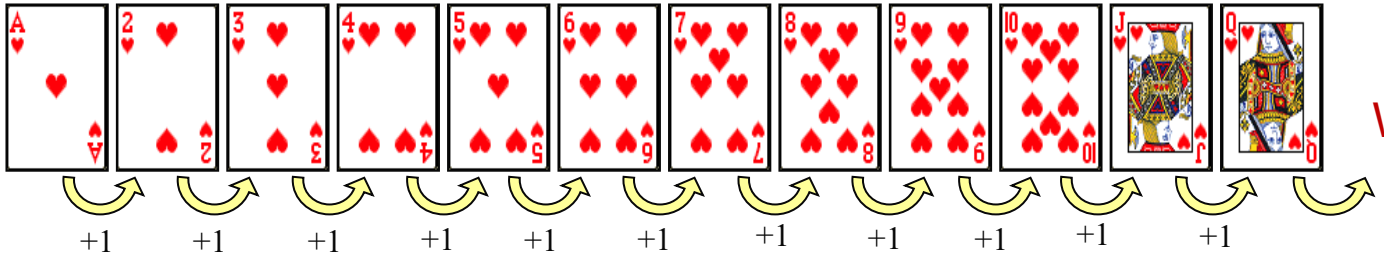
| Alphabet $S$ (Set of symbols) | $s_1$ | $s_2$ | ..... | $s_n$ |
|---|---|---|---|---|
| Probability | $p_1$ | $p_2$ | ..... | $p_n$ |

- The term $p_i \log_2 \frac{1}{p_i}$ indicates the amount of information in $s_i$ (or the number of bits needed to encode $s_i$).

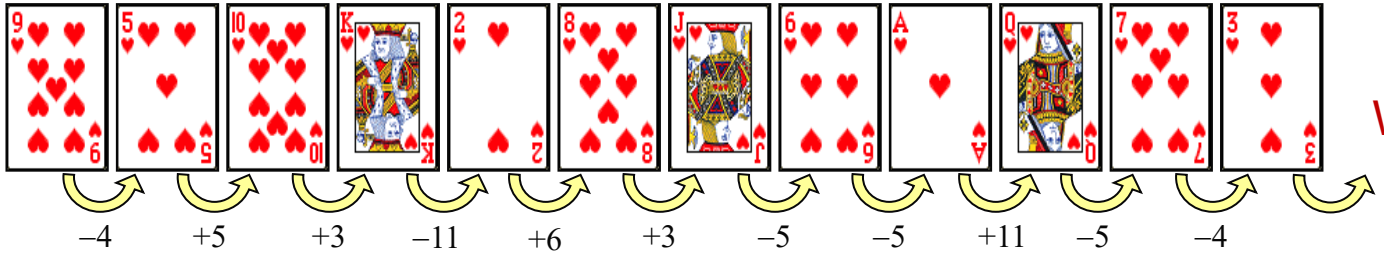

A coin has two sides, i.e., two symbols, $S_T$ and $S_H$

A dice has six sides, i.e., six symbols, $S_1$, $S_2$, $S_3$, $S_4$, $S_5$, and $S_6$

- Entropy is essentially a measure of disorder (or unpredictability).

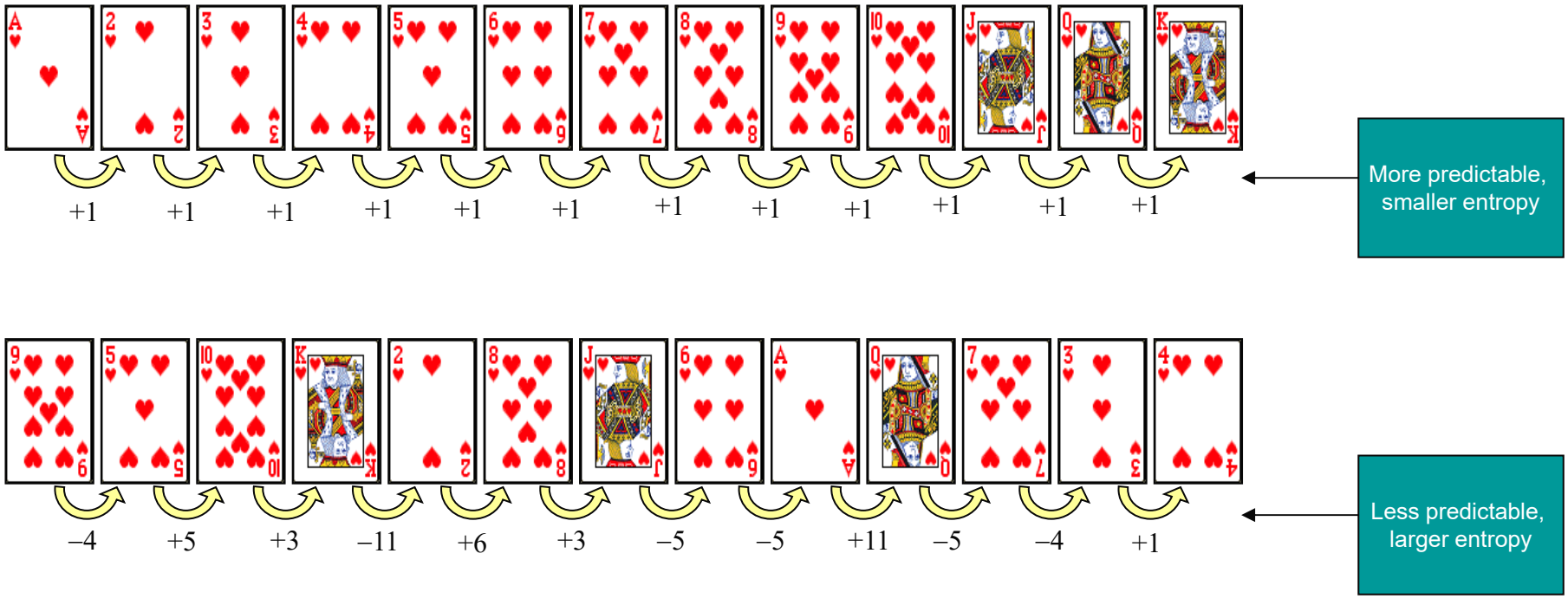



What is next?

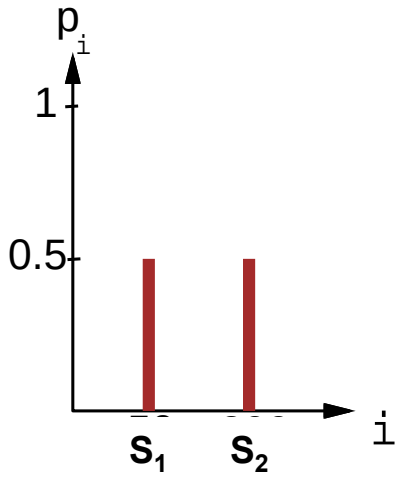



What is next?

- Entropy is essentially a measure of disorder (or unpredictability).



# Example 1

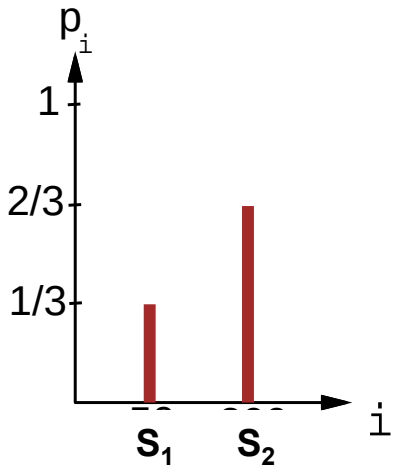



| Symbol | $S_1$ | $S_2$ |
|---|---|---|
| Probability | $\frac{1}{2}$ | $\frac{1}{2}$ |

$$\eta = H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

Note that $\log_2(x) = \log_{10}(x) / \log_{10}(2)$

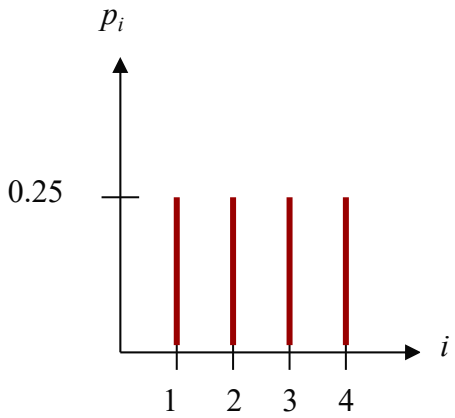
# Example 2




| Symbol | $S_1$ | $S_2$ |
|---|---|---|
| Probability | $\frac{1}{3}$ | $\frac{2}{3}$ |

$$\eta = H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.92$$

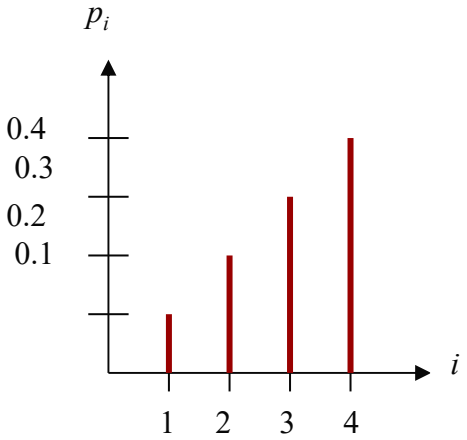
# Example 3




| Symbol | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Probability | $\frac{1}{4}$ | $\frac{1}{4}$ | $\frac{1}{4}$ | $\frac{1}{4}$ |

$$\eta = H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\sum_{i=1}^{4} \frac{1}{4} \log_2 \frac{1}{4} = 2$$

# Example 4




| Symbol | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Probability | 0.1 | 0.2 | 0.3 | 0.4 |

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

$$= -0.1\log_2 0.1 - 0.2\log_2 0.2 - 0.3\log_2 0.3 - 0.4\log_2 0.4 = 1.85$$

- Shannon's Coding Theorem states that the entropy is the lower bound for the average number of bits to encode each symbol in $S$, i.e.,

$$\eta \leq \bar{\ell}$$

where $\bar{\ell}$ is the average length of the codeword (measured in terms of bits) produced by the encoder.

# Lossless Compression
## Run-Length Encoding (RLE)

- Run-length encoding works by detecting repeating symbols in the input file.
- In general, each repeating symbol, called a *run*, can be encoded using two bytes. The first byte indicates the number of occurrences of the symbol in the run and the second byte is the symbol.
- For example, we may encode the following input data as:

aaaaacfffffff → 5a1c7f

Try an exercise

- This method is good at compressing binary images and traditional cartoon images, where large portions of the image have same pixel values.


- RLE encoding is used in most bitmap file formats, such as TIFF and BMP.

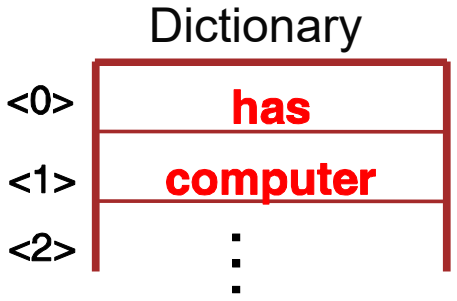
# Lossless Compression
## Lempel-Ziv-Welch (LZW)

- LZW works by detecting patterns. It stores each pattern in a table only once.
- When we encounter the same pattern again, we may replace the pattern with the table index (or a code).
- For example, we may have a Word file that contains the following two sentences:

He has a computer. This computer has a lot of memory.

We may store them as follows:

He <0> a <1>. This <1> <0> a lot of memory.



## *Compression Algorithm:*

```
s = next input character;  /* read first character */
while not EOF {
      c = next input character;  /* read next character */
      if s + c exists in the dictionary
            s = s + c;   /* append character c to end of string s */
      else {
            output the code for s;
            add string s + c to the dictionary with a new code;
            s = c;
      }
}
output the code for s;
```

# ➢ Example: consider encoding “ABABBABC”

***Compression Algorithm:***

```
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;
```

**Input:** ABABBABC

| A | B | A | B | B | A | B | C |
|---|---|---|---|---|---|---|---|

| s | c | o/p |
|---|---|---|
| A | B | |

**Dictionary**

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

standard section

application section

# ➢ Example: consider encoding “ABABBABC”

***Compression Algorithm:***

```
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;
```

**Input:** ABABBABC

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |
| A | B | A | B | B | A | B | C |

| s | c | o/p |
|---|---|---|
| A | B | 1 |
| B | A | 2 |
| A | B | |
| AB | B | 4 |
| B | A | |
| BA | B | 5 |
| B | C | 2 |
| C | EOF | 3 |

**Dictionary**

| code | string | |
|---|---|---|
| 1 | A | standard section |
| 2 | B | |
| 3 | C | |
| 4 | AB | application section |
| 5 | BA | |
| 6 | ABB | |
| 7 | BAB | |
| 8 | BC | |

**Output:** 1 2 4 5 2 3

***Decompression Algorithm:***

```
s = NIL;
while not EOF {
    k = next input code;
    entry = dictionary entry for k;

    /* exception handling */
    if (entry == NULL)
        entry = s + s[0];

    output entry;

    /* reconstruct the dictionary */
    if (s != NIL)
        add string s + entry[0] to
          dictionary with a new code;
    s = entry;
}
```

***Decompression Algorithm:***

```
s = NIL;
while not EOF {
      k = next input code;
      entry = dictionary entry for k;

      /* exception handling */
      if (entry == NULL)
            entry = s + s[0];

       output entry;

      /* reconstruct the dictionary */
      if (s != NIL)
            add string s + entry[0] to
              dictionary with a new code;
      s = entry;
}
```

Input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 4 | 5 | 2 | 3 |
| 1 | 2 | 4 | 5 | 2 | 3 |
| 1 | 2 | 4 | 5 | 2 | 3 |
| 1 | 2 | 4 | 5 | 2 | 3 |
| 1 | 2 | 4 | 5 | 2 | 3 |
| 1 | 2 | 4 | 5 | 2 | 3 |
| 1 | 2 | 4 | 5 | 2 | 3 |

| s | k | entry | 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 | EOF | | | |

Output A B A B B A B C

*What is the compression ratio here?*

Comparing the two dictionaries:

| 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 | EOF | 3 | |

**Compression**

| s | k | entry | 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 | EOF | | | |

**Decompression**

- LZW is most suitable for compressing text files.
- It is used in several image file formats, such as GIF and TIFF, for lossless compression. However, the compression ratio can be low if there are not too many repeating sequences.
- Some compression programs, such as **compress** and **pkzip**, use a method similar to LZW for compression.

Try an exercise

# Lossless Compression
## Huffman Encoding

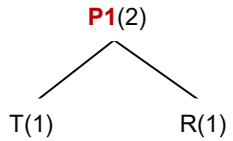
- Unlike the previous methods, Huffman encoding is a variable-length coding method, i.e., symbols are mapped to codes with different code lengths.
- It assigns a code of a certain code length according to the probability of occurrence of the symbol.
- It is a type of entropy coding.

***Algorithm:***

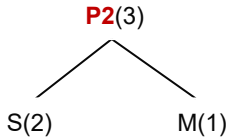
1. Initialization: Put all symbols on a list sorted according to the frequency count (descending order).
2. Repeat the following steps until there is only one symbol left:
   - Remove two symbols with the lowest frequency count from the list.
   - Using these two symbols to form child nodes of a Huffman subtree.
   - Assign the sum of frequency counts of the two symbols to the parent and insert the parent to the list. Keep the list in order.
3. Assign a codeword for each leaf node based on the path from the root.

# Example: To construct codewords for “SEMESTER”.

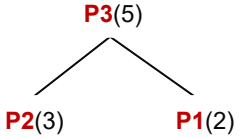
List: E(3), S(2), M(1), T(1), R(1)



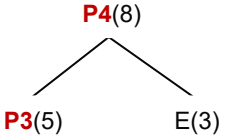

List: E(3), **P1**(2), S(2), M(1)



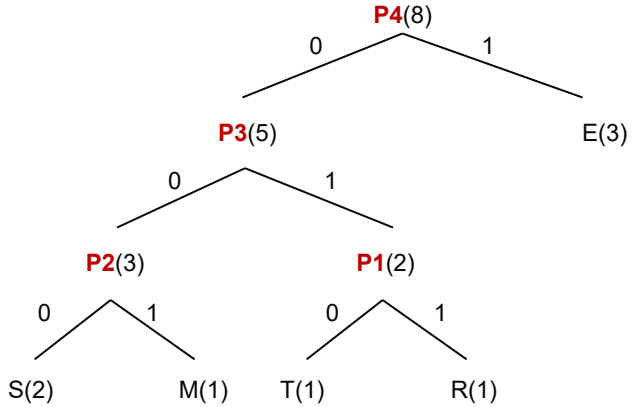

List: E(3), **P2**(3), **P1**(2)




List: **P3**(5), E(3)




List: **P4**(8)




| Symbol | E | S | M | T | R |
|---|---|---|---|---|---|
| Count | 3 | 2 | 1 | 1 | 1 |
| Codeword | 1 | 000 | 001 | 010 | 011 |
| Number of bits | 1 | 3 | 3 | 3 | 3 |

Total number of bits to code “SEMESTER”:
= 3+1+3+1+3+3+1+3 = 18 bits

- Properties of Huffman Coding:

  1. ***Prefix Code***: No Huffman code is a prefix of another Huffman code. This is to prevent any ambiguity in decoding.

     Example 1 (prefix code):

| Symbol | L | H | E | O |
|---|---|---|---|---|
| Codeword | 0 | 10 | 110 | 111 |

     Example 2 (non-prefix code):

| Symbol | L | H | E | O |
|---|---|---|---|---|
| Codeword | 0 | 1 | 01 | 111 |

     If code=01, then is the original message “E” or “LH” ?

     If code=111, then is the original message “O” or “HHH” ?

## 2. ***Optimality***: minimum redundancy code

- This means that one cannot find another set of codes for coding individual symbols that would result in a smaller average code length than that of the Huffman codes.
- The two symbols with lowest frequent counts have the same length for their Huffman codes, differing only at the last bit.
- Symbols that occur more frequently have Huffman codes no longer than symbols that occur less frequently.
- The average code length for a given application, $S$, is strictly less than $\eta$ + 1. Therefore:

$$\eta \leq \bar{\ell} < \eta + 1$$

Using the last example, since we have five symbols, we could use 3 bits to represent different symbols. As “SEMESTER” contains 8 characters, the input file size = 8 * 3 bits = 24 bits.

Hence, the compression ratio = 24 / 18 = 1.333

$\eta = -\frac{3}{8}log_2\frac{3}{8} - \frac{2}{8}log_2\frac{2}{8} - \frac{1}{8}log_2\frac{1}{8} - \frac{1}{8}log_2\frac{1}{8} - \frac{1}{8}log_2\frac{1}{8}$ = 2.156

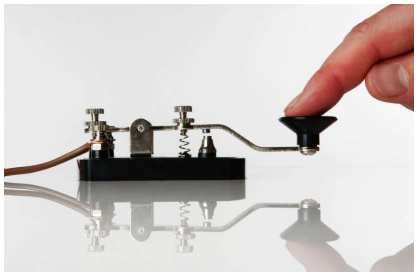
$\bar{\ell}$ = 18 / 8 = 2.25

$\therefore \quad \eta \leq \bar{\ell} < \eta+1$ holds

Try an exercise

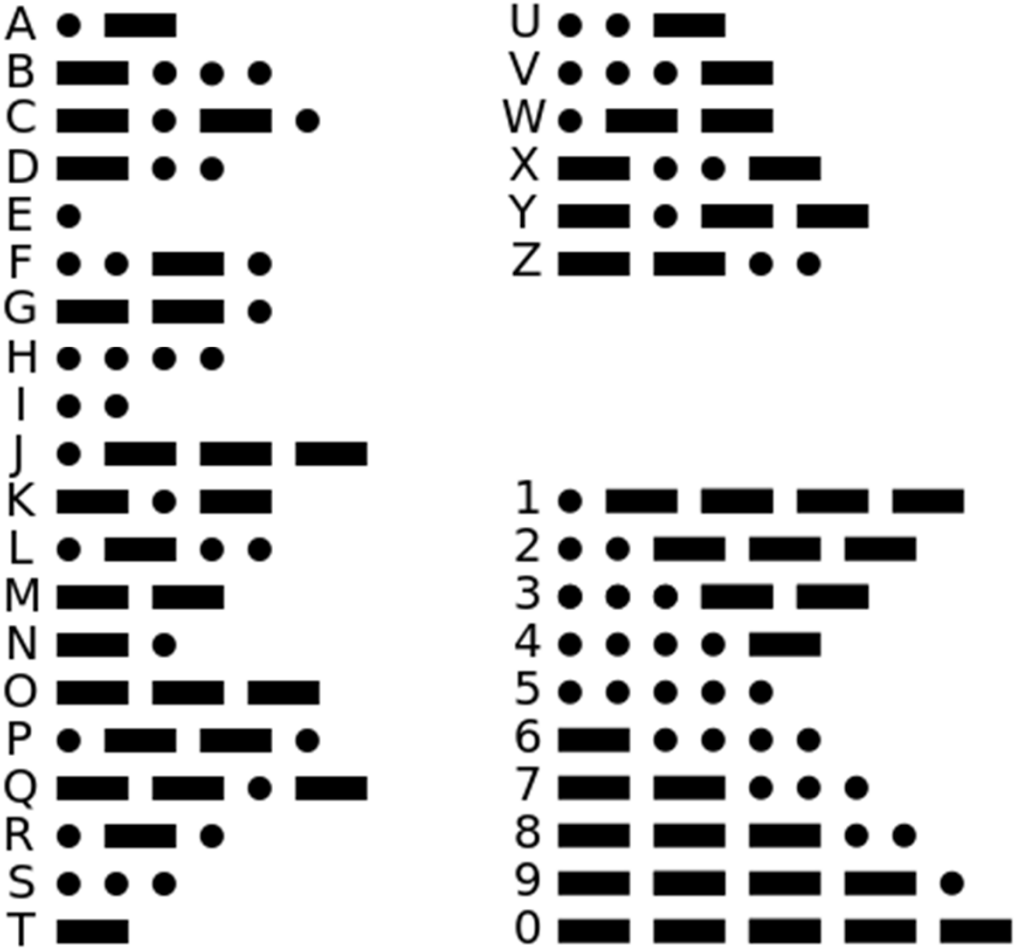
# Lossless Compression
## Morse Code

- This is a method for transmitting text information as a series of on-off tones, lights or clicks that can be directly understood by a skilled listener or observer without special equipment.
- It is also a variable-length coding method, according to the probability of occurrence of each symbol.
- It contains two symbols, dot and dash. A dot is one unit. A dash is three units.

# International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.




| | Morse Code Length | Probability |
|---|---|---|
| A | 2 | 0.08167 |
| B | 4 | 0.01492 |
| C | 4 | 0.02782 |
| D | 3 | 0.04253 |
| E | 1 | 0.12702 |
| F | 4 | 0.02228 |
| G | 3 | 0.02015 |
| H | 4 | 0.06094 |
| I | 2 | 0.06966 |
| J | 4 | 0.00153 |
| K | 3 | 0.00772 |
| L | 4 | 0.04025 |
| M | 2 | 0.02406 |
| N | 2 | 0.06749 |
| O | 3 | 0.07507 |
| P | 4 | 0.01929 |
| Q | 4 | 0.00095 |
| R | 3 | 0.05987 |
| S | 3 | 0.06327 |
| T | 1 | 0.09056 |
| U | 3 | 0.02758 |
| V | 4 | 0.00978 |
| W | 3 | 0.02360 |
| X | 4 | 0.00150 |
| Y | 4 | 0.01974 |
| Z | 4 | 0.00074 |

# Lossless Compression
## CCITT Group 3 1D Compression

- This compression method is primary used in facsimile.
- It is based on Huffman encoding with length of a codeword depending on its probability of occurrence.
- The method considers the fact that a fax page consists of scanlines. Each scanline consists of a white line followed by a black line, then a white line, etc..
- This method replaces a white or black line of a certain number of pixels by a codeword as shown in the following table.

| White Run Length | Code Word | Black Run Length | Code Word |
|---|---|---|---|
| 0 | 00110101 | 0 | 0000110111 |
| 1 | 000111 | 1 | 010 |
| 2 | 0111 | 2 | 11 |
| 3 | 1000 | 3 | 10 |
| 4 | 1011 | 4 | 011 |
| 5 | 1100 | 5 | 0011 |
| 6 | 1110 | 6 | 0010 |
| 7 | 1111 | 7 | 00011 |
| 8 | 10011 | 8 | 000101 |
| 9 | 10100 | 9 | 000100 |
| 10 | 00111 | 10 | 0000100 |
| 11 | 01000 | 11 | 0000101 |
| 12 | 001000 | 12 | 0000111 |
| 13 | 000011 | 13 | 00000100 |
| 14 | 110100 | 14 | 00000111 |
| ⋮ | ⋮ | ⋮ | ⋮ |

Consider a typical black-and-white page that we may fax to a friend:



There are two popular types of DVE architecture, peer-to-peer and client-server [8] [10]. Although peer-to-peer has a relatively lower overall latency due to the direct connections among relevant peers, client-server allows the server to serve as a monitor to enforce security and provide information persistency. Even though it is possible to have a hybrid architecture, allowing any form of direct peer-to-peer communications would make it difficult to control/manage the states of the DVE in practice. This is particularly important in paid applications, such as online games, where change of game states may sometimes involve payments. Hence, most of the paid online games are based on the client-server architecture. With this architecture, more servers may be added as the number of users increases. In this work, we consider a more general multi-server architecture where servers of a DVE system may be distributed around the world. This is important as applications become more globalized.

In a DVE system, whenever a user (referred to as **A**) changes its state, e.g., making a move, **A** needs to send an update message to other users who are near to **A** in the VE. We refer to these users as as relevant users to **A**. A typically approach to determine these relevant users is by defining a circular region around **A**, referred to as **A**'s area of interest (AOI) [26].

In designing a multi-server DVE system that is scalable to the number of users, we have two main challenges. First, it is important for the system to maximize "view consistency" of all users, i.e., if **A** changes its state, all relevant users should receive the corresponding update message in a timely manner so that their views of the VE agree with **A**. When this is not the case, some users may be acting according to incorrect views, which may cause disputes among them. If all relevant users are served by the same server, any update messages sent from the client (i.e., the client machine) of **A** involve only client-server delay. However, if the relevant users are distributed in multiple servers, any update messages sent from **A** involve both client-server and server-server delays. Hence, the view inconsistency problem becomes more significant. Here, if the servers are distributed across the globe, which may help minimize the client-server delay, the server-server delay can become significant. Fig. 7(d) shows that given a multi-server DVE, as the number of users increases, the view inconsistency problem becomes significant if we do not address it (refer to the top two curves). Second, to provide interactive responses to users' actions with low response time, a multi-server DVE should have the ability to efficiently redistribute workloads among servers so as to minimize the number of overloaded servers.

There are two popular types of DVE

Although peer to peer has a relatively low

relevant peers, client-server allows the ser

vide information persistency. Even thoug

any form of direct peer-to-peer communic

states of the DVE in practice. This is parti

games, where change of game states may

online games are based on the client-ser

There are two

~~Although peer to~~

relevant peers, cli

vide information

There are two

~~Although peer to~~

re

v

| White Run Length | Code Word | Black Run Length | Code Word |
|---|---|---|---|
| 0 | 00110101 | 0 | 0000110111 |
| 1 | 000111 | 1 | 010 |
| 2 | 0111 | 2 | 11 |
| 3 | 1000 | 3 | 10 |
| 4 | 1011 | 4 | 011 |
| 5 | 1100 | 5 | 0011 |
| 6 | 1110 | 6 | 0010 |
| 7 | 1111 | 7 | 00011 |
| 8 | 10011 | 8 | 000101 |
| 9 | 10100 | 9 | 000100 |
| 10 | 00111 | 10 | 0000100 |
| 11 | 01000 | 11 | 0000101 |
| 12 | 001000 | 12 | 0000111 |
| 13 | 000011 | 13 | 00000100 |
| 14 | 110100 | 14 | 00000111 |

- Advantages:
  - Simple to implement in both hardware and software
  - Worldwide standard for fax machines. It allows document imaging applications to incorporate fax documents easily.
- Disadvantages:
  - 1D encoding only – does not consider the similarity between consecutive scanlines.
  - No error protection mechanism.

# Lossless Compression
## CCITT Group 3 2D Compression

- This method is a refinement of the last method by combining a 1D coding scheme with a 2D coding scheme.
- The 2D coding scheme works by considering the fact that consecutive scanlines differ very little statistically.
- This method uses a “K” factor where the image is divided into groups of K lines.
- The first line of each group is encoded using the CCITT Group 3 1D method.

- The first line is then used as a reference line for the second line of the group and so on.
- Only the change in positions of the black and white transitions between any two consecutive lines need to be store.
- The major reason for dividing the image into groups is to allow the first line of each group to be the synchronizing line in the event of a transmission error, i.e., for error recovery.

- Advantages:
  - It is also a worldwide fax standard for document imaging applications.
  - It has a higher compression ratio (between 10 to 20) than the CCITT Group 3 1D method.
- Disadvantages:
  - It has a lower compression ratio than the next method.
  - It is slightly more complex to decompress than the CCITT Group 3 1D method.

# Lossless Compression
## CCITT Group 4 Compression

- This method is similar to the CCITT Group 3D 2D method, but without dividing the image into groups of lines.
- An all-white line is used as a reference line to encode the first scanline of the image.
- The first scanline is then used as a reference line to encode the second scanline and so on.
- Each successive scanline is encoded relative to the previous scanline for the complete image.

- Advantages:
  - It has a higher compression ratio than the previous methods.
- Disadvantages:
  - A single error can destroy the rest of the image.
  - It sacrifices the error recovery mechanism of the CCITT Group 3 2D method (in order to achieve a higher compression ratio).