Faculty of Computers & Informatics

Fundamentals of Multimedia

Lecture 4
Lossless Data Compression
Fixed Length Coding

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Outcomes of Lecture 3

- *Physical and perceptual aspects of color*
  - Human Vision
- *Color models in image*
  - RGB
  - CMYK
  - HSB
- *Gamma Correction*
- *Color models in video*
  - YUV
  - YCbCr

Outline

- *Basics of Information Theory*
  - Data entropy
- *Fixed Length Coding*
  - Run Length Coding (RLC)
  - Dictionary-based Coding
    - *Lempel-Ziv-Welch (LZW) algorithm*

Outline

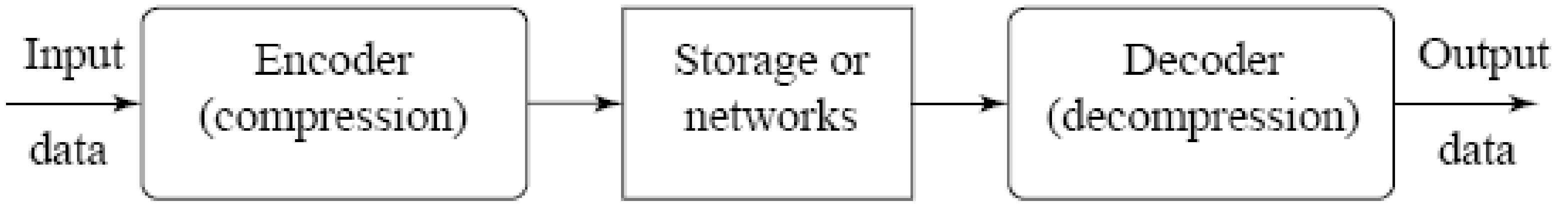
- ***Basics of Information Theory***
  - Data entropy
- *Fixed Length Coding*
  - Run Length Coding (RLC)
  - Dictionary-based Coding
    - *Lempel-Ziv-Welch (LZW) algorithm*

Data Compression

- *What is Compression?*
  - The process of coding
  - Reduce the total number of bits needed to represent certain information.
- *Why?*
  - Huge volume of multimedia data
  - More efficient data storage, processing and transmission
- *Compression Ratio*
  - Compression ratio= B_0 / B_1
  - B_0 : number of bits before compression
  - B_1 : number of bits after compression

Compression Schemes

A General Data Compression Scheme.



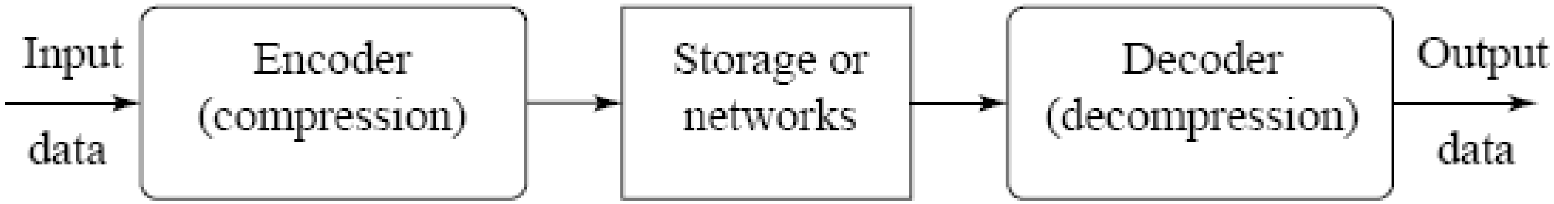
- *Lossy Compression*
  - The compression and decompression processes induce ***information loss***.
- *Lossless Compression*
  - The compression and decompression processes induce ***no information loss***.

Example of Compression Schemes

- *Transmit the data {250, 251, 251, 252, 253, 253, 254, 255} by the network*
  - Rewrite the data sequence using binary: {11111010, 11111011, 11111011, 11111100, 11111101, 11111101, 11111110, 11111111}
  - Totaly require 8*8 = 64 bits for transmission
- *The available bandwidth is limited*
  - Only 16 bits available.
  - Compression is necessary.

Example of Lossy Compression

- *Encode:* *Drop the least significant bits*
- *Encode data:* *8*2 bit = 16 bits*



250	11/111010	11	11/000000	192
251	11/111011	11	11/000000	192
251	11/111011	11	11/000000	192
252	11/111100	11	11/000000	192
253	11/111101	11	11/000000	192
253	11/111101	11	11/000000	192
254	11/111110	11	11/000000	192
255	11/111111	11	11/000000	192

Induce Information Loss

Example of Lossless Compression

- *Encode:* *Encode the difference*
- *Encode data:* *8-bit + 7* 1-bit = 15 bits*

250	250	11111010	250	250
251	1	1	+1	251
251	0	0	+0	251
252	1	1	+1	252
253	1	1	+1	253
253	0	0	+0	253
254	1	1	+1	254
255	1	1	+1	255

No Information Loss

Bound of Lossless Compression

- *The user expects*
  - Compression ratio as much as it can be
  - Without influence the recovery of the original file.
- *But! Compression ration can't be infinite.*
- *Entropy defines the bound of lossless compression*
  - The number of bits should be used to represent the information source on average
- *It can be interpreted as the **average shortest message length**, in bits, that can be sent to communicate the true value to a recipient.*

Definition of Entropy

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

- *Alphabet:* $S = \{s_1, s_2, \ldots s_n\}$
  - Possible values of the information source
- *Probability:* $P = \{p_1, p_2, \ldots p_n\}$
  - Relevant probability that the s_i occurs.
- *Self-information:* $log_2 \frac{1}{p_i}$
  - The amount of information contained in s_i
  - A value that occurs with very high probability carries little "surprise" or very little information.

Example of Entropy Calculation

- *Message: {abcdabaa}*
- *Alphabet={a, b, c, d} with probability {4/8, 2/8, 1/8, 1/8}*

a => 00

b => 01

c => 10

d => 11

- *Message: {abcdabaa} => {00 01 10 11 00 01 00 00}*
- *Average lenght=16 bits / 8 chars = 2*

Example of Entropy Calculation

- *Alphabet={a, b, c, d} with probability {4/8, 2/8, 1/8, 1/8}*
- $\eta = 4/8 * \log_2 2 + 2/8 * \log_2 4 + 1/8 * \log_2 8 + 1/8 * \log_2 8$
- $\eta = 1/2 + 1/2 + 3/8 + 3/8 = 1.75$ *average length*
- *a => 0 b => 10 c => 110 d => 111*
- *Message: {abcdabaa} => {0 10 110 111 0 10 0 0}*
- *average length = 14 bits / 8 chars = 1.75*

Outline

- ***Basics of Information Theory***
  - Data entropy
- ***Fixed Length Coding***
  - Run Length Coding (RLC)
  - Dictionary-based Coding
    - *Lempel-Ziv-Welch (LZW) algorithm*

Run-Length Coding

- ***Rationale for RLC:*** *if the information source has the property that symbols tend to form continuous groups, then such symbol and the length of the group can be coded.*
- ***Memoryless Source:*** *Namely, the value of the current symbol does not depend on the values of the previously appeared symbols.*
- *Instead of assuming memoryless source, Run-Length Coding (RLC) exploits* ***memory present*** *in the information source.*

Run-Length Coding

- *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.*

WWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWW

↓

6W1B12W3B14W

- *Compression Ratio 36/10= 3.6*

Run-Length Coding

- *Extreme Cases:*
  - Best Case: AAAAAAAA → 8A
    - *Compression Ratio: 8/2=4*
  - Worst case: ABABABAB → 1A1B1A1B1A1B1A1B
    - *Compression Ratio: 8/16=0.5*
    - ***Negative compression:*** *the resulting compressed file is larger than the original one.*

Dictionary-based Coding

- *Use **fixed-length** codeword*
  - Represent variable-length strings of possible values (symbols or characters) that commonly occur together, such as words in English text.
- *Limpel-Ziv-Welch (LZW) is an adaptive, dictionary-based technique*
  - Unix compress, GIF files.
  - The LZW encoder and decoder build up the same dictionary dynamically while receiving the data

LZW Compression for String

- *Input data*
  - ABABBABCABABBA
- *Initial simple dictionary only includes the possible values of the alphabet*

code	string
1	A
2	B
3	C

- *Then, apply the following algorithm*

LZW Compression Algorithm

```
BEGIN
s = first 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
```

LZW Compression Algorithm

"ABABBABCABABBA"

s=next char

c=next char

s	c	output	code	string
			1	A
			2	B
			3	C
A	B			

The output codes are:

LZW Compression Algorithm

“ABABBABCABABBA”

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B				

Check s+c

s+c (AB) is not in the Dic.

--- output the code for s
A => 1

--- insert AB in Dic.

--- s=c

The output codes are: 1

LZW Compression Algorithm

“ABABBABCABABBA”

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A			

The output codes are: 1

LZW Compression Algorithm

“ABABBABCABABBA”

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A				

Check s+c

s+c (BA) is not in the Dic.

--- output the code for s
B => 2

--- insert BA in Dic.

--- s=c

The output codes are: 1

LZW Compression Algorithm

“ABABBABCABABBA”

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			

The output codes are: 1 2

LZW Compression Algorithm

"ABABBABCABABBA"

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB				

Check s+c

s+c (AB) is in the Dic.

--- s=s+c

The output codes are: 1 2

LZW Compression Algorithm

"ABABBABCABABBA"

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		

- output codes are: 1 2 4 5 2 3 4 6 1
- From 14 characters, only 9 codes are sent
- compression ratio = 14/9 = 1.56

LZW Decompression

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

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1			

S=nil
K=1

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		

Entry = A
Output = A

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		

if (s != NIL)
    add string s + entry[0]
    to dictionary with a new code

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A				

S= entry

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2			

K = next input

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	**B**		

Entry = B
Output = B

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB

if (s != NIL)
add string s + entry[0]
to dictionary with a new code

LZW Decompression

1 2 4 5 2 3 4 6 1

S	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B				

S= entry

LZW Decompression

1 2 4 5 2 3 4 6 1

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			

S + entry[0]

- Output: “ABABBABCABABBA”,
- Truly lossless result!

Summary

- *Basics of Information Theory*
  - Data entropy
- *Fixed Length Coding*
  - Run Length Coding (RLC)
  - Dictionary-based Coding
    - *Lempel-Ziv-Welch (LZW) algorithm*