

Lossless Compression

Agenda

- Compression can be lossless or lossy
- Lossless techniques include:
 - RLE
 - LZW
 - Huffman

Compression can be lossless or lossy

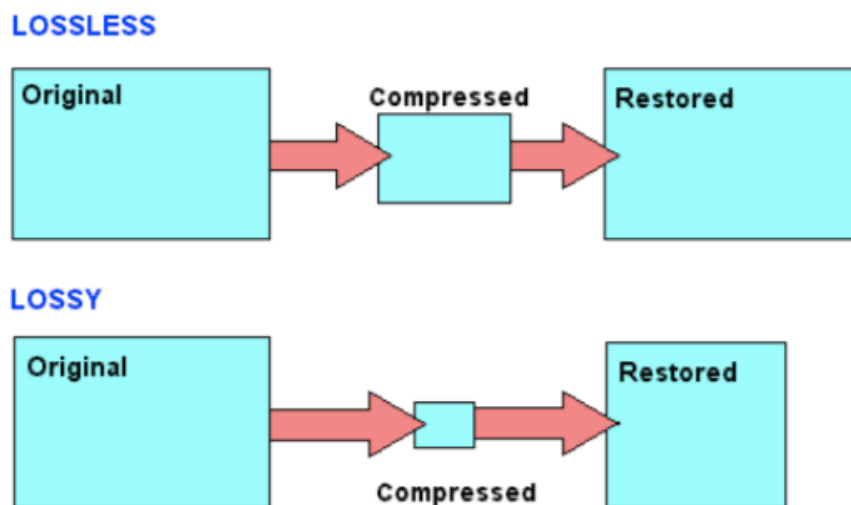
Basics of Media Compression

Lossless encoding

- Redundant data is not encoded
- No information is lost in compression and decompression
- Possible because some data appear more often than other and some data normally appear together

Lossy encoding

- details not perceived by users are discarded; e.g. small change in lightness (luminance) noticed more than change of colour details
- approximation of the real data (sacrifice some information)



Since lossless compression does not loss information, it can be applied as many time as necessary, whereas the lossy compression loss information, so it should be applied only once.

Compression rate

Compression rate: ratio of the original file size a to the size of the compressed file b, expressed as a:b

Example:

What is the compression rate of a digital image knowing that its uncompressed size is 5MB and its compressed size is 5KB?

Solution: $\frac{5000000}{5000} = 1000 : 1$

Types of Compression Algorithms

dictionary-based, entropy, arithmetic, adaptive, perceptual and differential compression methods

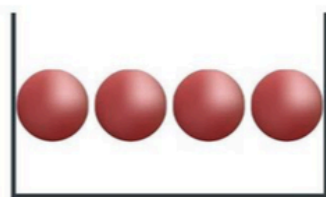
- Dictionary-based methods (e.g. **LZW compression**) : use a look-up table of fixed-length codes, where one code word may correspond to a string of symbols rather than to a single symbol
- Entropy compression (e.g. **Huffman encoding**) : uses a statistical analysis of the frequency of symbols and achieves compression by encoding more frequently-occurring symbols with shorter code words

Example

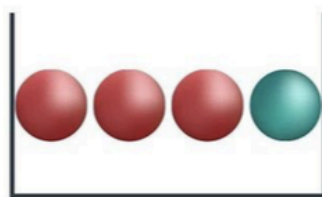
To the swinging and the
ringing
of the bells, bells, bells-
of the bells, bells, bells, bells
Bells, bells, bells-
To the rhyming and the
chiming of the bells!

Pattern	Reference	Binary value
To	0	0000
the	1	0001
swinging	2	0010
and	3	0011
ringing	4	0100
Of	5	0101
bells	6	0110
Bells	7	0111
rhyming	8	1000
chiming	9	1001
,	10	1010
-	11	1011
!	12	1100

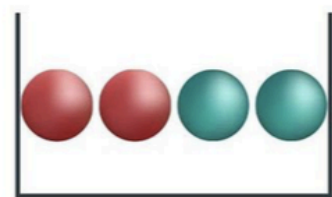
Dictionary-based compression



Low



Medium



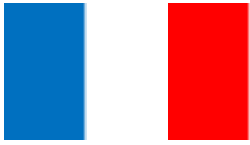
High

short ← Code length → long

Entropy compression

Run-Length Encoding (RLE)

Physically reduce any type of repeating sequence (**run**), once the sequence reach a predefined number of occurrence (**length**)



e.g. area of solid colour (contain great area of uniform colours)

Exploits **spatial redundancy**

Idea: store number pairs (c, n), where c indicates the pixel value and n is how many consecutive pixels have that value.

Example:

For example, say that in a 10,000-pixel grayscale image, the first 20 pixels are:
255 255 255 255 255 255 242 242 242 242 238 238 238 238 238 238 255 255 255
255

Solution:

(255,6), (242,4), (238,6), (255,4)

Without RLE, 20 pixels \cdot 1 byte/pixel = 20 bytes

With RLE, (1 byte + 1 byte) \cdot 4 pairs = 8 bytes

Compression rate = 20 : 8 = 5 : 2

RLE: Calculating the length (Method 1)

The run-length encoding algorithm scans the image file in a preprocessing step to determine the size of the largest run of colors, r

The formula for figuring out how many bytes you need to represent a number that can be anywhere between 0 and r is:

$$b = \left\lceil \frac{\log_2(r + 1)}{8} \right\rceil$$

Example:

For example, say that in a 10,000-pixel grayscale image, the first 20 pixels are:
255 255 255 255 255 255 242 242 242 242 238 238 238 238 238 238 255 255 255
255, **after pre-processing the image, the longer run is found to be $r = 300$**

Solution:

$r = 300$, $b = 2$ bytes

(255, 6), (242, 4), (238, 6), (255, 4)

With RLE, (1 byte + 2 byte) \cdot 4 = 12 bytes

Compression rate = 20 : 12 = 5 : 3

RLE: Calculating the length (Method 2)

Rather than determining the largest n in advance and then setting the bit depth accordingly, we could choose a bit depth d in advance (e.g. 8 bits)

Then if more than 2^d consecutive pixels of the same colour are encountered, **the runs are divided into blocks**

For example, if 1 byte is used to represent each n , then the largest value for n is 255. If

1000 consecutive whites exist in the file they would be represented as 4 different runs:
(255, 255), (255, 255), (255, 255), (255, 235)

Exercise 1

How would you encode the following sequence of symbols using RLE, assuming each symbol represents a grayscale pixel value (i.e. 1 byte) ?

AABCAAAAAABBBBBBCCCCCCCCCCCCD

What is the compression rate?

Solution:

(A,2),(B,1),(C,1),(A,6),(B,5),(C,11),(D,1)

Without RLE, 1 byte · 27 = 27 bytes

With RLE, (1 byte + 1 byte) · 7 = 14 bytes

Compression rate = 27 : 14

How about this sequence of bytes? AABCAABACABBDCBCCACBCCCCBBB

What is the compression rate?

Solution:

(A,2)(B,1)(C,1)(A,2)(B,1)(A,1)(C,1)(A,1)(B,2)(D,1)(C,1)(B,1)(C,2)(A,1)(C,1)(B,1)(C,4)(B,3)

Without RLE, 1 byte · 27 = 27 bytes

With RLE, (1 byte + 1 byte) · 18 = 36 bytes

Compression rate = 27 : 36 = 3 : 4

RLE suitable image examples

1. Country flags
2. Long identical pixels

Example

Consider the 8 bits gray scale image shown in the Figure below. It contains 1050 x 780 pixels. Calculate the maximum compression rate you can achieve on that image by using Run Length Encoding. Explain your calculations.

Solution:



Run length method 1

(0,350),(255,350),(127,350),.....(0,350),(255,350),(127,350)

(In total, there are 780 cycles)

For each cycle, b = 2 bytes, (1 byte + 2 bytes) · 3 = 9 bytes

Without RLE, 1050 · 780 pixels · 1 byte/pixels = 819000 bytes

With RLE, 9 · 780 = 7020 bytes

Compression rate = $\frac{819000}{7020} = 350 : 3$

Run length method 2

(0,255),(0,95),(255,255),(255,95),(127,255),(127,95),.....(0,255),(0,95),(255,255),

(255,95),(127,255),(127,95)

(In total, there are 780 cycles)

For each cycle, $(1 \text{ byte} + 1 \text{ byte}) \cdot 6 = 12 \text{ bytes}$ (Which is larger then the former method)

Compression rate = $\frac{1050}{12} = 175 : 2$ (smaller than the former method)

LZW (Lempel-Ziv-Welch)

exploits spatial redundancy

The method is commonly applied to GIF and TIFF image files

The LZW algorithm is based on the observation that sequences of color in an image file (or sequences of characters in a text file, i.e. words) are often repeated

(Some data normally appears together)

Dictionary-based encoding

- Dictionary-based algorithms encode variable-length strings of symbols as single tokens (indexes).
- The tokens form an index into a phrase dictionary.
- If the tokens are smaller than the phrases they replace, compression occurs.

LZW algorithm (text)

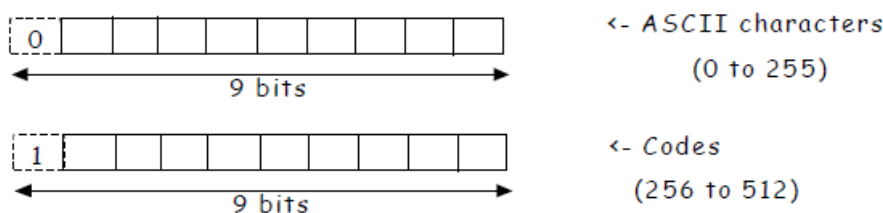
```
BEGIN
  s = next input character;
  while not EOF
  {
    c = next input character;

    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
```

LZW for text

For text, the dictionary is initialised with 256 entries (indexed with ASCII codes 0 through 255) representing the ASCII table

Instead of using 2 or more bytes per code, we could use 9 bits:



With 9 bits we can only have a maximum of 256 codes for strings of length 2 or above (with the first 256 entries for ASCII characters)

($2^9 = 512$, 256 for ASCII characters, 256 for new codes)

Original LZW uses dictionary with 4K entries, with the length of each symbol/code being 12 bits ($\log_2(4000) = 11.96 \rightarrow 12$ bits)

Exercise

Current Char	Next Char	CurrentChar + NextChar is in the dictionary?	Output Index	[New Index] New String
"B"	"A"	No	66	[256]"BA"
"A"	"B"	No	65	[257]"AB"
"B"	"A"	Yes	-	-
"BA"	"A"	No	256	[258]"BAA"
"A"	"B"	Yes	-	-
"AB"	"A"	No	257	[259]"ABA"
"A"	"A"	No	65	[260]"AA"
"A"	"A"	Yes	-	-
"AA"	"A"	No	260	[261]"AAA"
"A"	EOF	-	65	-

What is the encoded string?

What is the original string?

What is the compression rate?

Solution:

(1) B A BA AB A AA A

(2) BABAABAAAA

(3) Without LZW: $10 \cdot 8 \text{ bits} = 80 \text{ bits}$. With LZW: $7 \cdot 9 \text{ bits} = 63$

Compression rate = $80 : 63$

LZW algorithm (image)











```

algorithm LZW
/*Input: A bitmap image.
Output: A table of the individual colors in the image and a compressed version of the
file.
Note that + is concatenation.*/
{
    initialize table to contain the individual colors in bitmap
    pixelString = first pixel value
    while there are still pixels to process {
        pixel = next pixel value
        stringSoFar = pixelString + pixel
        if stringSoFar is in the table then
            pixelString = stringSoFar
        else {
            output the code for pixelString
            add stringSoFar to the table
            pixelString = pixel
        }
    }
    output the code for pixelString
}

```

With a first pass over the image file, the code table is **initialised** to contain **all the individual colours** that exist in the image file

These colours are encoded in consecutive integers

Code	0	1	2	3	4	5	6	7	8	9								etc.
Color																		



It will be useful if the images has a large distribution of similar color distribution blocks

Huffman encoding

statistical coding or entropy compression

Encode frequent bit patterns with short code, infrequent bit patterns with longer codes

Effective when probabilities vary widely

Fixed-length inputs become variable-length outputs

Shannon's entropy equation:



KEY EQUATION

Let S be a string of symbols and p_i be the frequency of the i^{th} symbol in the string. (p_i can equivalently be defined as the probability that the i^{th} symbol will appear at any given position in the string.) Then

$$H(S) = \eta = \sum_i p_i \log_2 \left(\frac{1}{p_i} \right)$$

Entropy and bit depth

For an image file that has exactly 256 pixels in it, each pixel being of a different colour, then the frequency of each colour is $1/256$. The average number of bits needed to encode each colour is 8

$$\sum_0^{255} \frac{1}{256} \left(\log_2 \left(\frac{1}{\frac{1}{256}} \right) \right) = \sum_0^{255} \frac{1}{256} (\log_2(256)) = \sum_0^{255} \frac{1}{256} (8) = 8$$

Another Example:

Color	Frequency
black	100
white	100
yellow	20
orange	5
red	5
purple	3
blue	20
green	3

$$\begin{aligned}
 & \frac{100}{256} \log_2 \left(\frac{256}{100} \right) + \frac{100}{256} \log_2 \left(\frac{256}{100} \right) + \frac{20}{256} \log_2 \left(\frac{256}{20} \right) + \frac{5}{256} \log_2 \left(\frac{256}{5} \right) \\
 & + \frac{5}{256} \log_2 \left(\frac{256}{5} \right) + \frac{3}{256} \log_2 \left(\frac{256}{3} \right) + \frac{20}{256} \log_2 \left(\frac{256}{20} \right) + \frac{3}{256} \log_2 \left(\frac{256}{3} \right) \\
 & \approx 0.530 + 0.530 + 0.287 + 0.111 + 0.111 + 0.075 + 0.287 + 0.075 \approx 2.006
 \end{aligned}$$

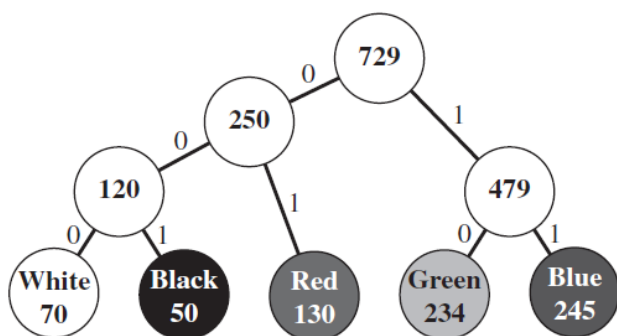
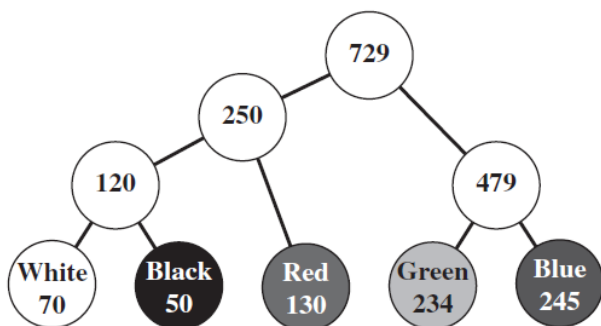
the minimum value for the average number of bits required to represent each colour in this file is 2.006

A better compression ratio is achieved if we use fewer bits to represent symbols that appear more frequently in the file

Huffman encoding

- (1) determining the codes for the colours and
- (2) compressing the image file by replacing each colour with its code

Example:



White	000
Black	001
Red	01
Green	10
Blue	11

Example:

Find a Huffman code using the frequency table below (there may be more than one solution). Then encode the following sequence: (3 bits/symbol)

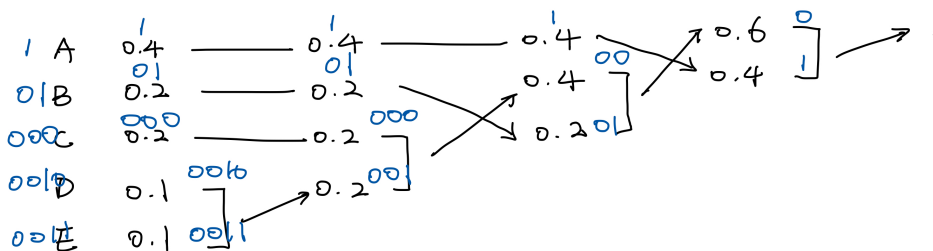
ABBAAADCBA

Finally, calculate the compression rate.

Symbol/Colour	Probability	Code
A	0.4	1
B	0.2	0 1
C	0.2	0 0 0
D	0.1	0 0 1 0
E	0.1	0 0 1 1

Solution:

Compression rate = $33 : (6+6+4+4) = 33 : 20$



Exercise

Consider the following sequence of letters (each letter represents one byte):

AABBBBCDDDA

- Encode the sequence using RLE (RunLength Encoding) and calculate the compression rate.
- Considering the following Huffman codes : 00, 01, 10, 110; how would you encode the sequence of bytes using Huffman encoding? What is the compression rate you achieve?
- Which of the two techniques works best on this data?

Solution:

(1) (A,2) (B,4) (C,1) (D,3) (A,5)

Without RLE : 15 bytes, Without RLE : 10 bytes

Compression rate = $3 : 2 = 1.5$

(2) A:00, B:01, D:10, C:110, Compression rate = $\frac{15 \cdot 8}{(14 \cdot 2 + 3)} = 120 : 31 = 3.87$

(3) RLE, Huffman encoding

Questions in test:

b) Image lossless compression.

[10 marks]

- i) What image property is used in Huffman encoding to achieve compression? Justify your answer. (3 marks)
- ii) Decode the following binary message using the Huffman encoding table provided in Table 1.
Binary message: 1111101111010011011110101101100110111110101011001100100111111 (2 marks)
- iii) Assuming that each symbol of Table 1 would normally be encoded using 3 bits (which is enough to encode 5 different symbols), how much compression is achieved in the binary message of question ii) above? (3 marks)
- iv) Consider the following statement: "For compression to remain lossless, an image should be encoded/decoded only once". Is it correct? Justify your answer. (2 marks)

symbol	probability	code
A	0.40	00
B	0.20	01
C	0.20	10
D	0.10	110
E	0.10	111

Table 1: Huffman encoding table

- (1) In most images, some colours may appear more frequently than other colours. (statistical properties (or the fact that some colours appear more often than others)).

The main idea of Huffman encoding is to use shorter code to encode pixels with more frequent colours, vice versa. In this case, though we use longer code for pixels with less frequent colours, it appears less, the reduction cause by shorter code weight more than the increasement cause by longer code, hence, we will reach compression.

- (2) EDECCBCECCDDBCEDCCDBCBAEE

- (3) Compression rate = $\frac{25 \cdot 3}{11 \cdot 3 + 14 \cdot 2} = 1.23$

- (4) No. Since the encoded/decoded process is lossless, no matter how many times you encoded/decoded, it won't loss any information. Hence, for compression, encoded/decoded more than once can also remain lossless.

If your image contains large areas of uniform colour, which lossless compression technique will work best? Why and how does it work?

Solution: In this case RLE will work best. RLE is a compression technique that utilizes the spatial redundancy in an image. Since this is an image which contains large areas of uniform colour, it contains large spatial redundancy. Instead of storing

values for each pixels, it uses number pairs (c,n) to store the information, where c is the pixel value and n is the number of consecutive pixels having that value.