

EBU5303

Multimedia Fundamentals

Lossless Compression

Dr. Marie-Luce Bourguet
marie-luce.bourguet@qmul.ac.uk

Learning Objectives

- Compare lossless and lossy compression
- Calculate a compression rate
- Distinguish between data and information
- Distinguish between different types of redundancy
- Describe lossless compression techniques: RLE, LZW, Huffman
- Use appropriate compression technique given data properties

Reading



<http://burg.cs.wfu.edu/TheScienceOfDigitalMedia/Chapter1/Ch1ScienceOfDigitalMedia.pdf>

1.5 Compression Methods

<http://burg.cs.wfu.edu/TheScienceOfDigitalMedia/Chapter3/Ch3ScienceOfDigitalMedia.pdf>

3.10.1 LZW Compression

3.10.2 Huffman Encoding

Reading



[Fundamentals of Multimedia](#), by Ze-Nian Li, Mark S. Drew, Jiangchuan Liu (3rd edition)

Chapter 7: Lossless Compression Algorithms

Agenda

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

Media Compression Techniques

- For **storage**
 - Typical uncompressed colour image requires about 1 MB storage
- For **transmission** over Internet, local networks and other networks with restricted bandwidth

Basics of Media Compression

- **Lossless encoding**

- redundant data is not encoded
- no information is lost in compression and decompression cycle
- possible because some data appear more often than other and some data normally appear together

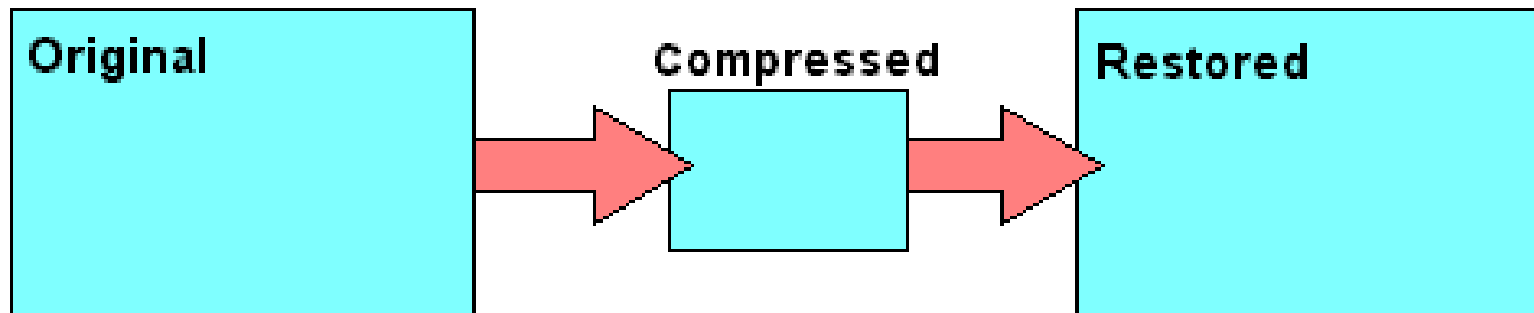
- **Lossy encoding**

- 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

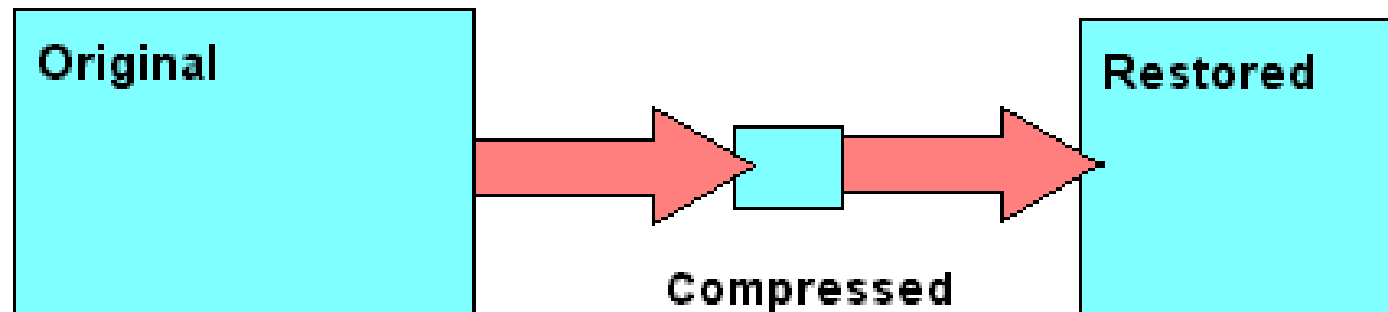
Lossless versus Lossy

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

LOSSLESS



LOSSY



Compression rate

- The ***compression rate*** is the ratio of the original file size a to the size of the compressed file b , expressed as $a:b$.
- Alternatively, you can speak of the ratio of b to a as a percentage.
- For example, for a file that is reduced by compression to half its original size, you could say that 50% compression is achieved, or alternatively, that the compression rate is 2:1.

Exercise



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

Types of Compression Algorithms

- Types of compression algorithm include ***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**.

Dictionary-based Encoding

@yatishparmar

yatishparmar.com

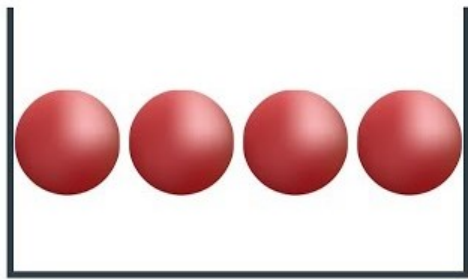
Example

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!

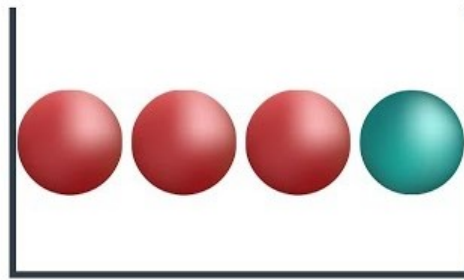
Pattern	Reference	Binary value
To	0	0000
the	1	0001
swinging	2	0010
and	3	0011
ringing	4	0100
Of	5	0101
bells	6	0110
Bells	7	0111
rhyming	8	1000
chiming	9	1001
,	10	1010
-	11	1011
!	12	1100

Entropy Compression

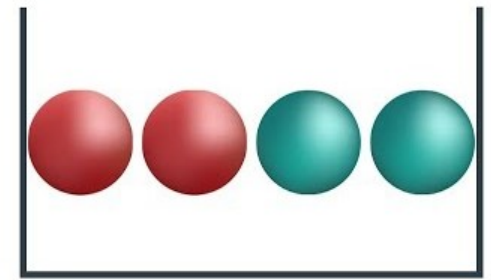
Entropy



Low



Medium



High

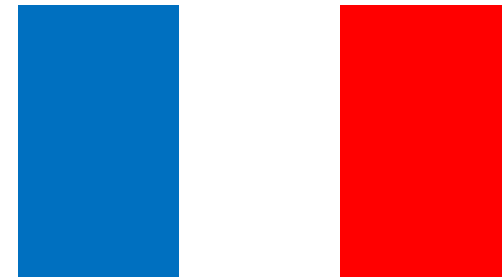
short ← Code length → long

Agenda

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

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
- Lossless
- Exploits **spatial redundancy**
- The simple idea is that instead of storing each pixel as an individual value, it *can be* more concise to store number pairs (c, n) , where c indicates the pixel value and n is how many consecutive pixels have that value.



RLE 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

- The run-length encoding of this sequence would be:

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

- Without RLE, 20 pixels require:

20 pixels * 1 byte/pixel = 20 bytes

- With RLE: 8 bytes
- Compression rate: 20:8 = **5:2**

RLE: Calculating the length

- 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:

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

RLE Example revisited

- Let's consider this sequence of pixels again:
255 255 255 255 255 255 242 242 242 242 238
238 238 238 238 238 255 255 255 255
- The run-length encoding of the sequence is:
(255, 6), (242, 4), (238, 6), (255, 4)
- If after pre-processing the image, the longer run is found to be $r = 300$, 2 bytes will be needed to store n .
- With RLE, the 20 pixels now require 12 bytes.
- Compression rate: 20:12

Calculating the length (another way)

- 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



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

AABCACAAAAABBBBBBCCCCCCCCCCCCCD

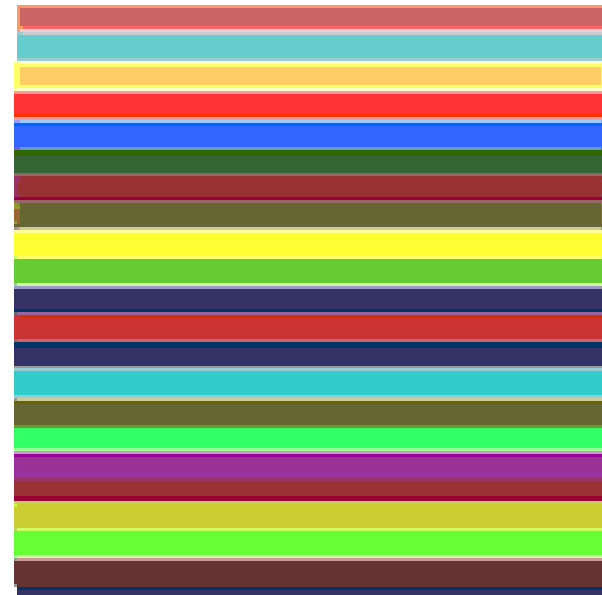
- What is the compression rate?

Exercise

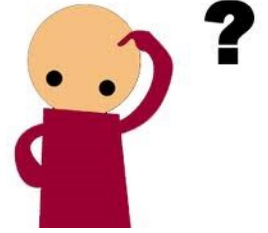


- How about this sequence of bytes?
AABCAABACABBDCBCCACBCCCCBBB
- What is the compression rate?

RLE suitable image examples



Question



Images of country flags can usually be compressed very efficiently using RLE. Why? Choose one answer.

- Because they contain a small number of colours.
- Because the colours are almost identical from one pixel to the next.
- Because of their statistical properties.
- Because they contain large areas of uniform colour.

Exercise



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



Agenda

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

LZW

- LZW stands for Lempel-Ziv-Welch, the creators of the algorithm.
- Lossless and exploits *spatial redundancy* (like RLE)
- 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.
- LZW is an example of *dictionary-based encoding*.

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.
- Dictionary-based compression uses a strategy that programmers are familiar with, i.e. using indexes into databases to retrieve information from large amounts of storage.

LZW Algorithm

```
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;
```

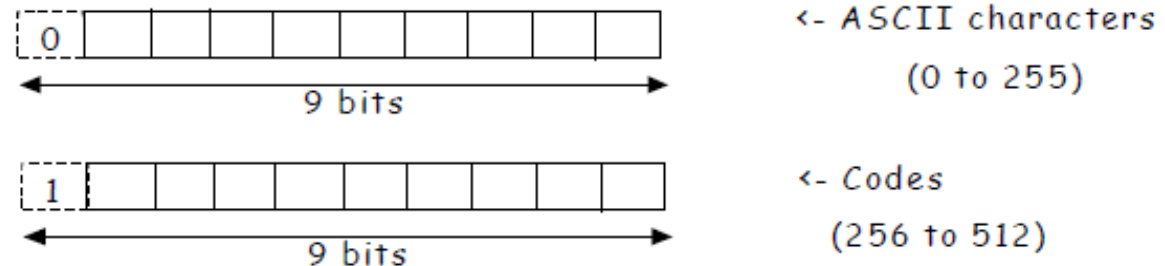
```
END
```

LZW Example

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	-

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).
- Original LZW uses dictionary with 4K entries, with the length of each symbol/code being 12 bits.

Exercise













- What is the encoded string?
- What is the original string?
- What is the compression rate?

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	-

LZW for images

- 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																	



LZW for images

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
}
```

Agenda

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

Huffman encoding

- Encode frequent bit patterns with short code, infrequent bit patterns with longer codes
- Example of *statistical coding* or *entropy compression*
- Effective when probabilities vary widely
- Lossless
- Fixed-length inputs become variable-length outputs

Entropy Compression

- Entropy encoding works by means of variable-length codes, using fewer bits to encode symbols that occur more frequently, while using more bits for symbols that occur infrequently.
- 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.

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

Entropy and bit depth

If you now have an image file of 256 pixels and only 8 colours in the image with the following frequencies:

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

Shannon's equation tells us that the *minimum value for the average number of bits* required to represent each colour in this file is 2.006

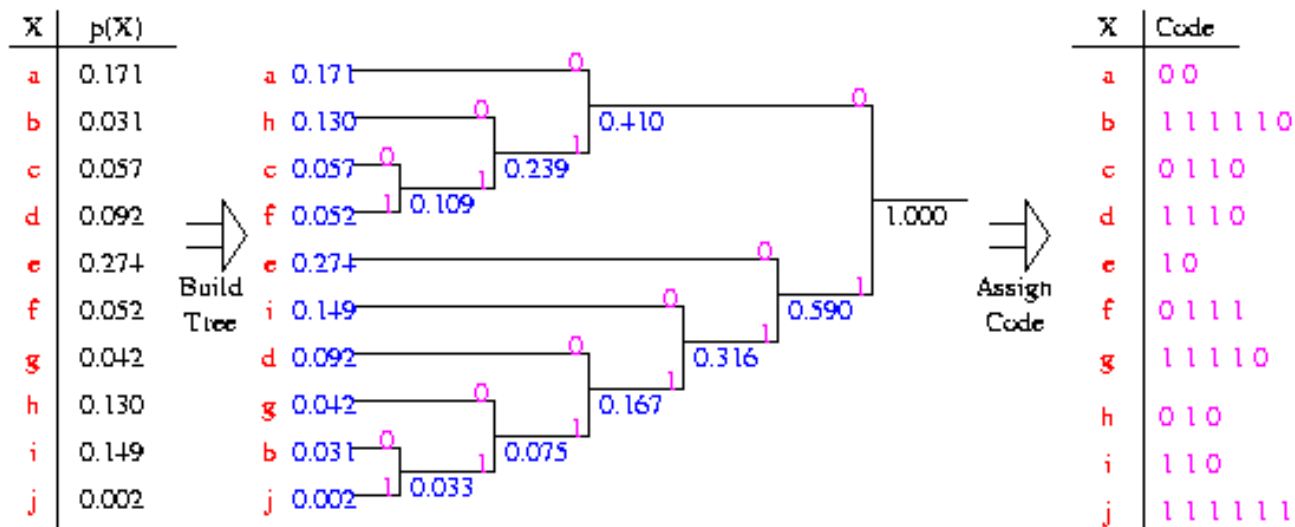
$$\begin{aligned} & \frac{100}{256} \log_2 \left(\frac{256}{100} \right) + \frac{100}{256} \log_2 \left(\frac{256}{100} \right) + \frac{20}{256} \log_2 \left(\frac{256}{20} \right) + \frac{5}{256} \log_2 \left(\frac{256}{5} \right) \\ & + \frac{5}{256} \log_2 \left(\frac{256}{5} \right) + \frac{3}{256}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

Entropy and bit depth

- The implication in Shannon's equation is that we don't necessarily need to use the same number of bits to represent each symbol.
- A better compression ratio is achieved if we use fewer bits to represent symbols that appear more frequently in the file.

Huffman encoding

- The Huffman encoding algorithm requires two passes: (1) determining the codes for the colours and (2) compressing the image file by replacing each colour with its code.
- To determine the codes, a tree data structure is built from the colour frequency table.



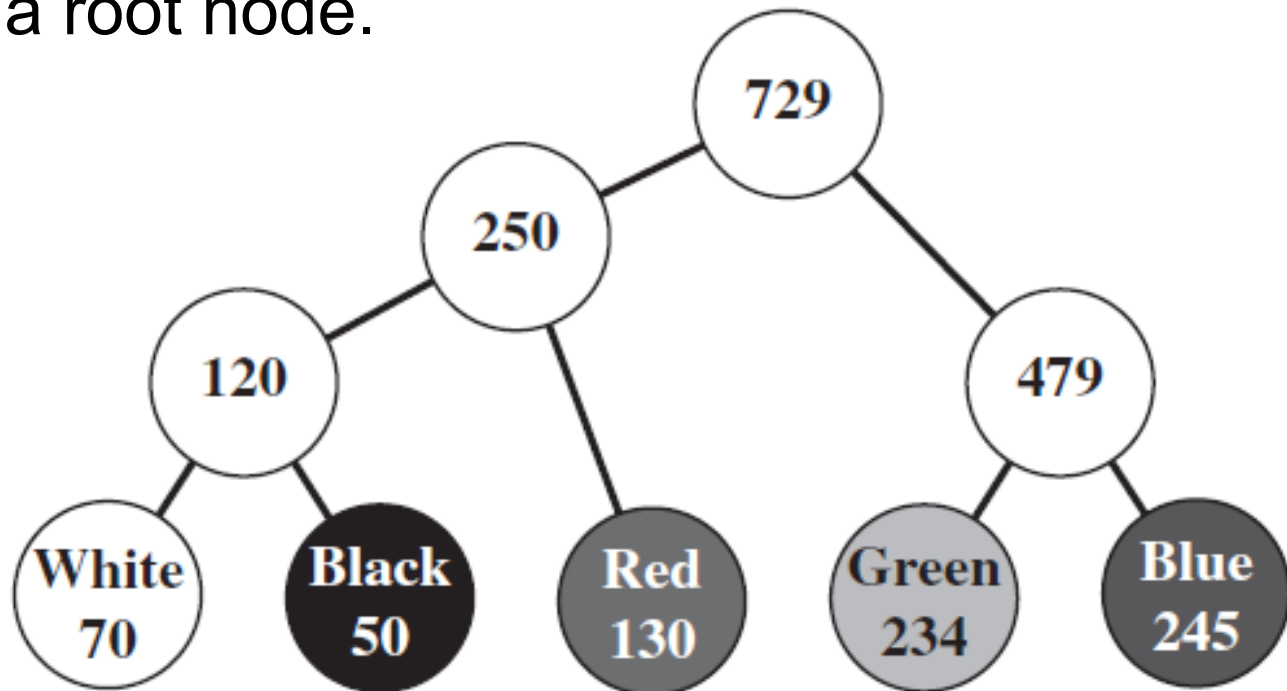
Huffman encoding

A node is created for each of the colours in the image, with the frequency of that colour's appearance stored in the node. These nodes will be the leaves of the code tree.



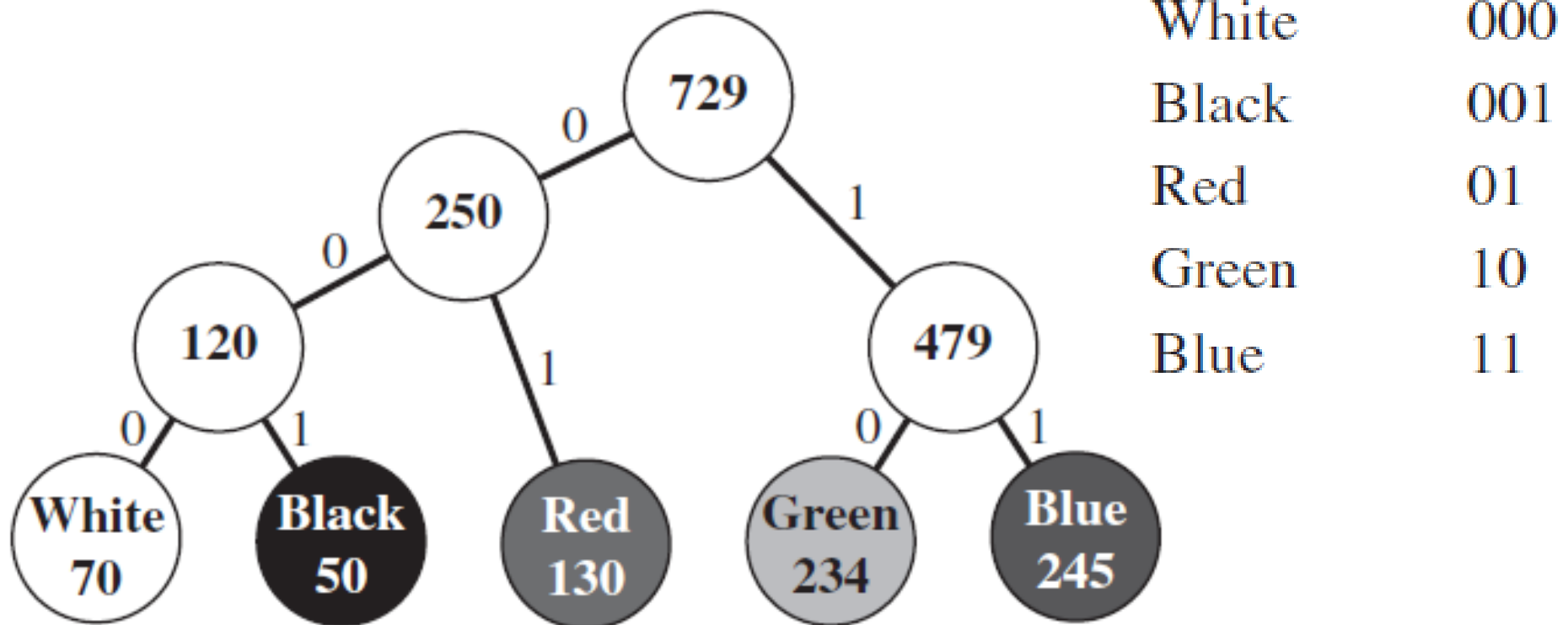
Huffman encoding

The two nodes with the smallest frequency values are joined such that they are the children of a common parent node, and the parent node's frequency value is set to the sum of the frequency values in the children nodes. This node-combining process repeats until you arrive at the creation of a root node.



Huffman encoding

Once the tree has been created, the branches are labeled with 0s on the left and 1s on the right. Then for each leaf node, you traverse the tree from the root to the leaf, gathering the code on the way down the tree.



Exercise



- Find a Huffman code using the frequency table below (there may be more than one solution).
- Then encode the following sequence:
ABBAAADCBA
- Finally, calculate the compression rate.

Symbol/Colour	probability	code
A	0.40	?
B	0.20	?
C	0.20	?
D	0.10	?
E	0.10	?

Exercise



Using the table below, decode the following message:

11111011110100110111101011011001101111
10101011001100100111111

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

Exercise

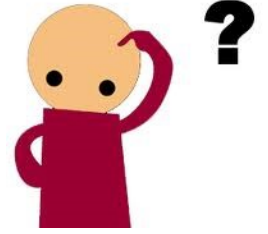


Consider the following sequence of letters (each letter represents one byte):

AABBBBCDDDA AAAA

- Encode the sequence using RLE (Run Length 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?

Question



Which of the following statements about lossless image encoding is not true?

- With lossless encoding, redundant data is not encoded.
- No information is lost in a compression and decompression cycle.
- For encoding to remain lossless, an image should be compressed only once.
- Lossless encoding is possible when some data appear more often than other.

Summary

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

Problem

Q9: Is lossless compression useful to you? Why?

