COMP482- Programming Studio

Project 1-
Image Compression Using LZW Coding

Instructor: Prof.Dr. Muhittin Gökmen

Teaching Assistants: Mustafa Ersen, MSc
Fatih Said Duran , MSc

Student Assistants: Beyzanur Yıldız
Onat Kaan Atılgan
Yusuf Kartal

Project 1: Image Compression using LZW coding

- In this project, you will develop a program in Python to compress an image by using Lempel-Ziv-Welch (LZW) method, save the compressed file, and decompress the image from the compressed file.
- Input to your program is an image file in *.png or *.bmp format
- You will implement methods by yourself without using any image processing libraries other than reading, writing and showing images.
- You will prepare a report to show your implementation and results.

Compression

Why do we need compression?

- Image: 6.0 million pixel camera, 3000x2000
  - 18 MB per RGB image → 56 pictures / 1GB

- Video: DVD Disc 4.7 GB
  - video 720x480, RGB, 30 f/s → 31.1MB/sec
  - audio 16bits x 44.1KHz stereo → 176.4KB/s
    - → 1.5 min per DVD disc

- Send video from cellphone:
  352*240, RGB, 15 frames / second
  - 3.8 MB/sec → $38.00/sec levied by AT&T

Data Compression

- Wikipedia: “data compression, or source coding, is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use through use of specific encoding schemes.”

- Applications
  - General data compression: .zip, .gz …
  - Image over network: telephone/internet/wireless/etc
  - Slow device:
    - 1xCD-ROM 150KB/s, bluetooth v1.2 up to ~0.25MB/s
  - Large multimedia databases

How do we compress?

- Goals of compression
  - Remove redundancy
  - Reduce irrelevance

- irrelevance or perceptual redundancy
  - not all visual information is perceived by eye/brain, so throw away those that are not.

a b c

FIGURE 8.4 (a) Original image. (b) Uniform quantization to 16 levels. (c) IGS quantization to 16 levels.

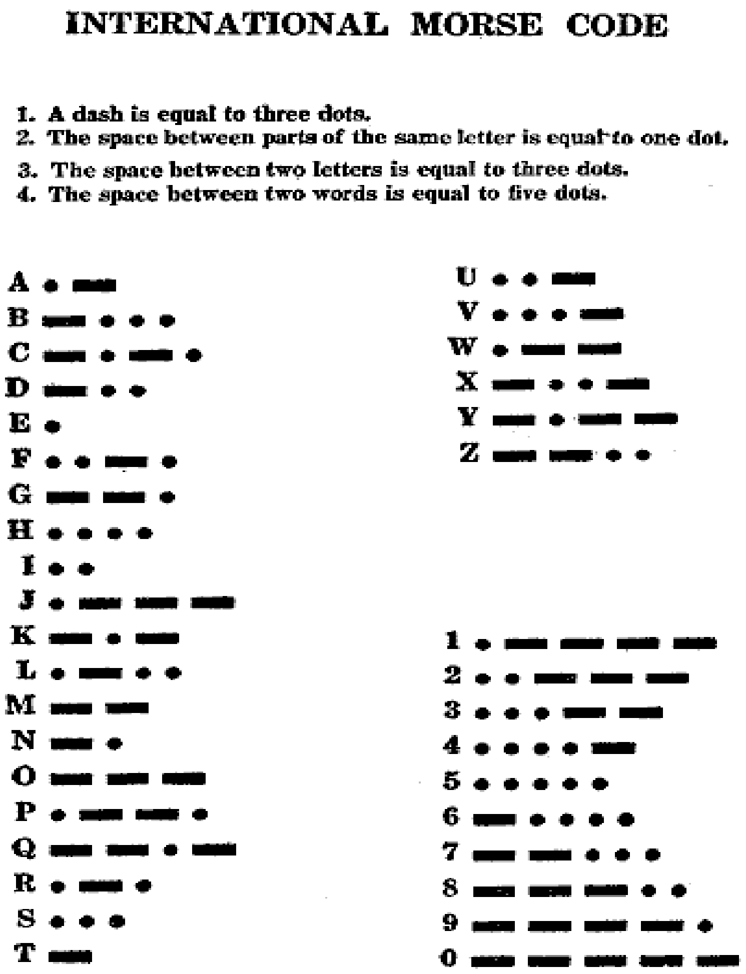
what can we compress?

- Goals of compression
  - Remove redundancy
  - Reduce irrelevance

- redundant : exceeding what is necessary or normal
  - symbol (statistical) redundancy
    - the common and uncommon values cost the same to store
  - spatial and temporal redundancy
    - adjacent pixels are highly correlated.

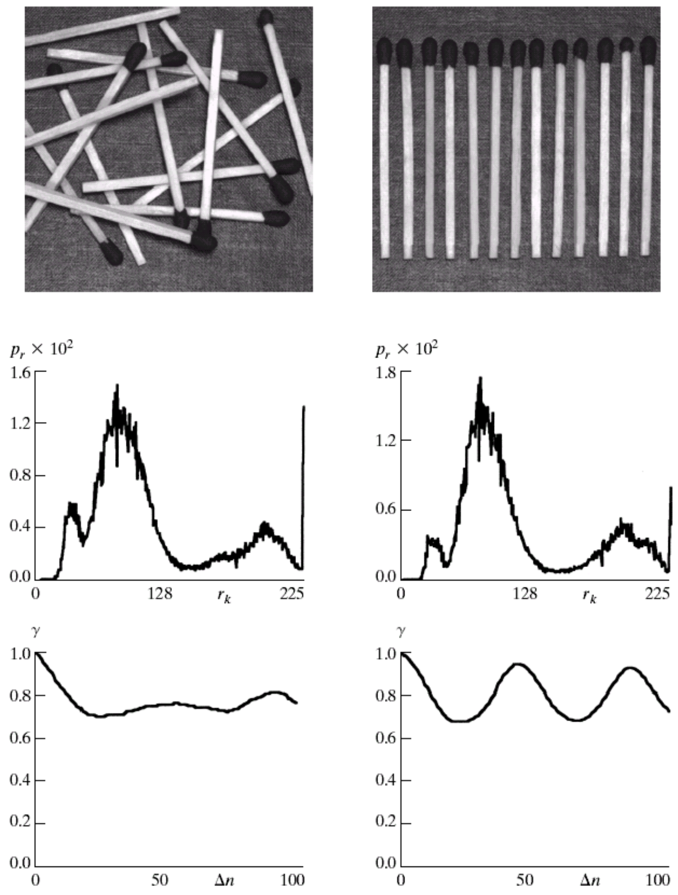
symbol/inter-symbol redundancy

- Letters and words in English
  - e, a, i, s, t, …
    q, y, z, x, j, …
  - a, the, me, I …
    good, magnificent, …
  - fyi, btw, ttyl …
- In the evolution of language we naturally chose to represent frequent meanings with shorter representations.



pixel/inter-pixel redundancy

- Some gray level value are more probable than others.
- Pixel values are not i.i.d. (independent and identically distributed)

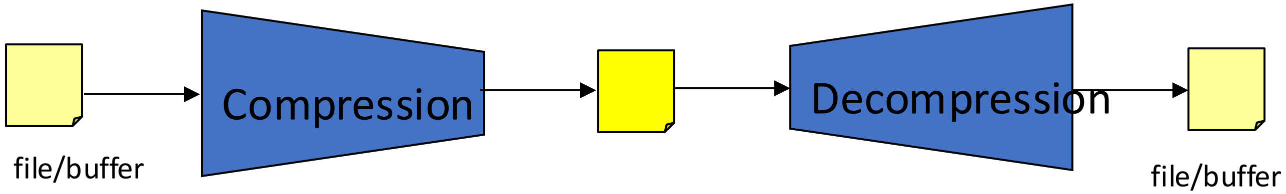


a b
c d
e f

FIGURE 8.2 Two images and their gray-level histograms and normalized autocorrelation coefficients along one line.

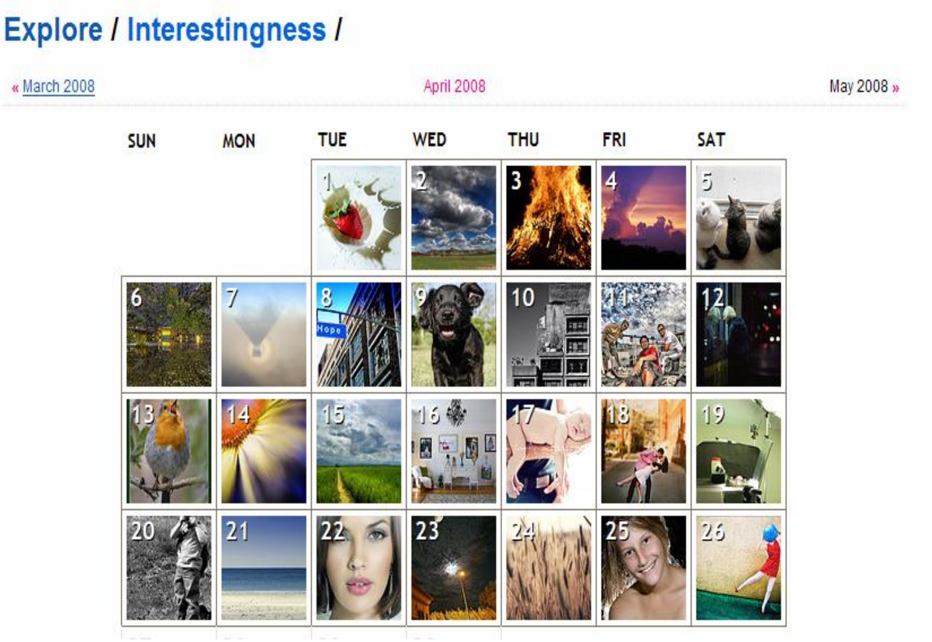
modes of compression

- Lossless Compression
  - preserve all information, perfectly recoverable
  - examples: morse code, zip/gz
- Lossy Compression
  - throw away perceptually insignificant information
  - cannot recover all bits
  - Examples: jpeg, mpeg



how much can we compress a picture?

IMG_0470.jpg	944 KB	JPEG Image	11/11/2006 4:29 PM	11/11/2006 3:29 PM	1600 x 1200
IMG_0471.jpg	892 KB	JPEG Image	11/11/2006 4:30 PM	11/11/2006 3:30 PM	1600 x 1200
IMG_0472.jpg	876 KB	JPEG Image	11/11/2006 4:31 PM	11/11/2006 3:31 PM	1600 x 1200
IMG_0473.jpg	1,214 KB	JPEG Image	11/11/2006 4:38 PM	11/11/2006 3:38 PM	1600 x 1200
IMG_0474.jpg	1,117 KB	JPEG Image	11/11/2006 4:38 PM	11/11/2006 3:38 PM	1600 x 1200
IMG_0475.jpg	1,208 KB	JPEG Image	11/11/2006 4:38 PM	11/11/2006 3:38 PM	1600 x 1200
IMG_0476.jpg	795 KB	JPEG Image	11/11/2006 4:39 PM	11/11/2006 3:39 PM	1600 x 1200
IMG_0477.jpg	1,042 KB	JPEG Image	11/11/2006 4:39 PM	11/11/2006 3:39 PM	1600 x 1200
IMG_0478.jpg	1,027 KB	JPEG Image	11/11/2006 4:40 PM	11/11/2006 3:40 PM	1600 x 1200
IMG_0479.jpg	1,010 KB	JPEG Image	11/11/2006 4:40 PM	11/11/2006 3:40 PM	1600 x 1200
IMG_0480.jpg	790 KB	JPEG Image	11/11/2006 4:41 PM	11/11/2006 3:41 PM	1600 x 1200
IMG_0481.jpg	959 KB	JPEG Image	11/11/2006 4:41 PM	11/11/2006 3:41 PM	1600 x 1200
IMG_0482.jpg	1,073 KB	JPEG Image	11/11/2006 4:42 PM	11/11/2006 3:42 PM	1600 x 1200
IMG_0483.jpg	990 KB	JPEG Image	11/11/2006 4:43 PM	11/11/2006 3:43 PM	1600 x 1200
IMG_0484.jpg	1,046 KB	JPEG Image	11/11/2006 4:45 PM	11/11/2006 3:45 PM	1600 x 1200
IMG_0485.jpg	878 KB	JPEG Image	11/11/2006 4:46 PM	[illegible]	[illegible]
IMG_0486.jpg	774 KB	JPEG Image	11/11/2006 4:46 PM	11/11/20	
IMG_0487.jpg	830 KB	JPEG Image	11/11/2006 4:47 PM	11/11/20	
IMG_0488.jpg	1,011 KB	JPEG Image	11/11/2006 4:47 PM	11/11/20	
IMG_0489.jpg	957 KB	JPEG Image	11/11/2006 4:47 PM	11/11/20	
IMG_0490.jpg	961 KB	JPEG Image	11/11/2006 4:48 PM	11/11/20	



- same dimensions (1600x1200), same original accuracy -- 3 bytes/pixel, same compressed representation, same viewer sensitivity and subjective quality …
- Different compressed file sizes
- Because of different "information content" in each image!

Project1

Image Compression using LZW Coding

- The project will consist of 6 Levels:
  - Level 1: LZW Encoding and Decoding (Text)
  - Level 2: Image Compression (Gray Level)
  - Level 3: Image Compression (Gray Level differences)
  - Level 4: Image Compression (Color)
  - Level 5: Image Compression (Color differences)
  - Level 6: GUI

Level 1: LZW Encoding and Decoding

Lossless Compression

Summary:

- Dictionary based Compression
- Adaptive Mechanism
- Limpel Ziv Welch (LZW) mechanism

Sources:

- *The Data Compression Book,* 2nd Ed., Mark Nelson and Jean-Loup Gailly.
- LZW Compression Article from Dr. Dobbs Journal: *Implementing LZW compression using Java,* ***by Laurence Vanhelsuwé***

LZW (dictionary Coding)

LZW (Lempel-Ziv-Welch) coding, assigns fixed-length code words to variable length sequences of source symbols, but requires no *a priori* knowledge of the probability of the source symbols.

LZW is used in:

•*Tagged Image file format (TIFF)*

•*Graphic interchange format (GIF)*

•*Portable document format (PDF)*

LZW was formulated in 1984

Dictionary-Based Compression

- The Huffman and Arithmetic coding algorithms use a statistical model to encode single symbols
  - Compression: Encode symbols into bit strings that use fewer bits.
- Dictionary-based algorithms do not encode single symbols as variable-length bit strings; they encode variable-length strings of symbols as single tokens
  - The tokens form an index into a phrase dictionary
  - If the tokens are smaller than the phrases they replace, compression occurs.
- Dictionary-based compression is easier to understand because it uses a strategy that programmers are familiar with -> using indexes into databases to retrieve information from large amounts of storage.
  - Telephone numbers
  - Postal codes

Dictionary-Based Compression: Example

- Consider the Random House Dictionary of the English Language, Second edition, Unabridged. Using this dictionary, the string:

  A good example of how dictionary-based compression works

  can be coded as:

  1/1 822/3 674/4 1343/60 928/75 550/32 173/46 421/2

- Coding:
  - Uses the dictionary as a simple lookup table
  - Each word is coded as x/y, where x gives the page in the dictionary and y gives the number of the word on that page.
  - The dictionary has 2,200 pages with less than 256 entries per page: Therefore, x requires 12 bits and y requires 8 bits, i.e., 20 bits per word (2.5 bytes per word).
  - Using ASCII coding the above string requires 48 bytes, whereas our encoding requires only 20 (<-2.5 * 8) bytes: 50% compression.

Adaptive Dictionary-based Compression

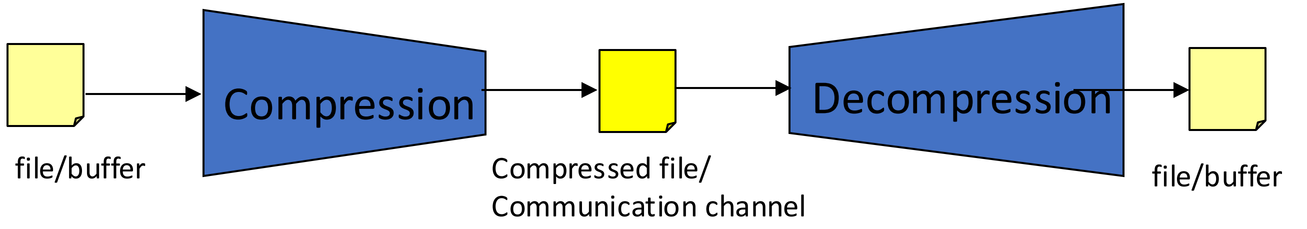
- Build the dictionary adaptively
  - Necessary when the source data is not plain text, say audio or video data.
  - Is better tailored to the specific source.
- Original methods due to Ziv and Lempel in 1977 (LZ77) and 1978 (LZ78). Terry Welch improved the scheme in 1984 (called LZW compression). It is used in UNIX *compress* and GIF.
- LZ77: A sliding window technique in which the dictionary consists of a set of fixed length phrases found in a window into the previously processed text
- LZ78: Instead of using fixed-length phrases from a window into the text, it builds phrases up one symbol at a time, adding a new symbol to an existing phrase when a match occurs.

LZW Algorithm

Index	Symbol
…	…
256	TH
257	THE
---	---

Preliminaries:

- ❑A dictionary that is indexed by “codes” is used.
- ❑The dictionary is assumed to be initialized with 256 entries (indexed with ASCII codes 0 through 255) representing the ASCII table.
- ❑The compression algorithm assumes that the output is either a file or a communication channel. The input being a file or buffer.
- ❑Conversely, the decompression algorithm assumes that the input is a file or a communication channel, and the output is a file or a buffer.



LZW Algorithm

LZW Compression:

```
set w = NIL
loop
    read a character k
    if wk exists in the dictionary
        w = wk
    else
        output the code for w
        add wk to the dictionary
        w = k
endloop
```

The program reads one character at a time.

- *If the code is in the dictionary, then it adds the character to the current work string, and waits for the next one. This occurs on the first character as well.*
- *If the work string is not in the dictionary, (such as when the second character comes across), it adds the work string to the dictionary and sends over the wire (or writes to a file) the code assigned to the work string without the new character. It then sets the work string to the new character.*

Example of LZW: Compression

Input String: ^WED^WE^WEE^WEB^WET

w	k	Output	Index	Symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

```
set w = NIL
loop
    read a character k
    if wk exists in the dictionary
        w = wk
    else
        output the code for w
        add wk to the dictionary
            w = k
endloop
```

LZW Algorithm

LZW Decompression:

```
read fixed length token k (code or char)
output k
w = k
loop
   read a fixed length token k
   entry = dictionary entry for k
   output entry
   add w + first char of entry to
      the dictionary
   w = entry
endloop
```

The nice thing is that the decompressor builds its own dictionary on its side, that matches exactly the compressor's, so that only the codes need to be sent.

Example of LZW

Input String (to decode): **^WED<256>E<260><261><257>B<260>T**

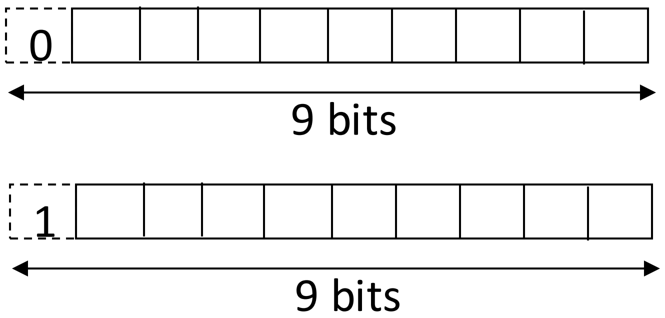
w	k	Output	Index	Symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
^W	E	E	260	^WE
E	<260>	^WE	261	E^
^WE	<261>	E^	262	^WEE
E^	<257>	WE	263	E^W
WE	B	B	264	WEB
B	<260>	^WE	265	B^
^WE	T	T	266	^WET

```
read a fixed length token k
      (code or char)
output k
w = k
loop
    read a fixed length token k
        (code or char)
    entry = dictionary entry for k
    output entry
    add w + first char of entry to
        the dictionary
    w = entry
endloop
```

Decoder builds the dictionary from the input string while generating the output

LZW Algorithm - Discussion

- Where is the compression?
  - Original String to decode : ^WED^WE^WEE^WEB^WET
  - Decoded String : ^WED<256>E<260><261><257>B<260>T
  - Plain ASCII coding of the string : 19 * 8 bits = 152 bits
  - LZW coding of the string: 12*9 bits = 108 bits (7 symbols and 5 codes, each of 9 bits)
- Why 9 bits?
  - An ASCII character has a value ranging from 0 to 255
  - All tokens have fixed length
  - There has to be a distinction in representation between an ASCII character and a Code (assigned to strings of length 2 or more)
  - Codes can only have values 256 and above



<- ASCII characters

(0 to 255)

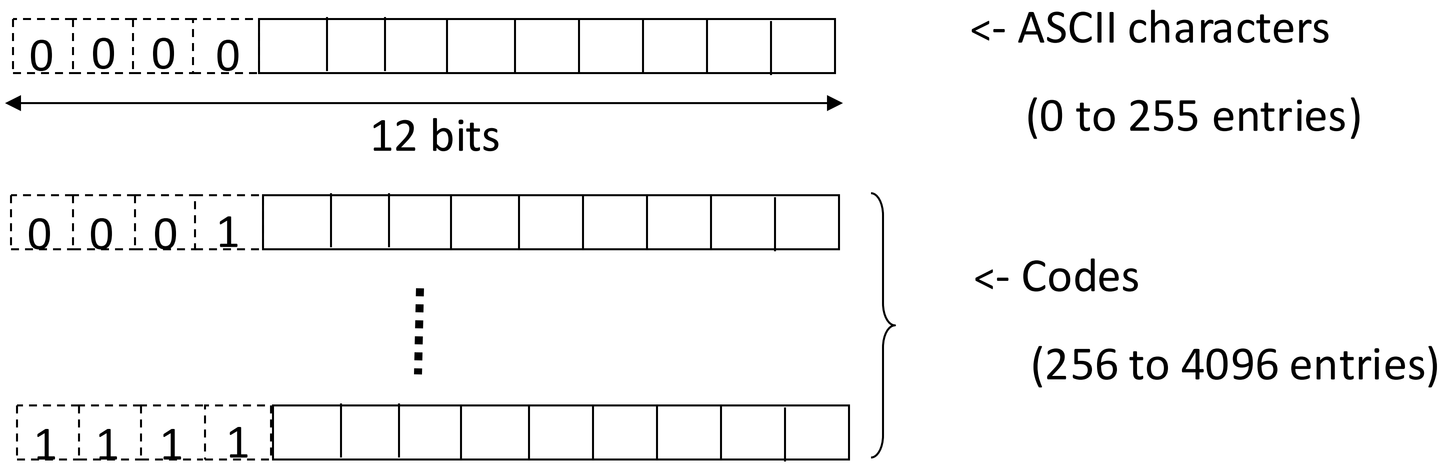
<- Codes

(256 to 512)

LZW Algorithm – Discussion (continued)

- 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



❑ With 12 bits, we can have a maximum of $2^{12} - 256$ codes.

- Practical implementations of LZW algorithm follow the two approaches:
  - Flush the dictionary periodically. (fixed n bits)
    – no wasted codes
  - Grow the length of the codes as the algorithm proceeds (progressive)
    - First start with a length of 9 bits for the codes.
    - Once we run out of codes, increase the length to 10 bits. When we run out of codes with 10 bits then we increase the code length to 11 bits and so on.
    - more efficient.

0	ASCII
1	Codes 256-512

Out of codes

0	0	ASCII
0	1	Codes 256-511
1	0	Codes 512-767
1	1	Codes 768-1023

Out of codes

0	0	0	ASCII
0	0	1	Codes 256-511
0	1	0	Codes 512-767
0	1	1	Codes 768-1023
1	0	0	Codes 1024-1279
1	0	1	Codes 1280-1535
1	1	0	Codes 1536-1791
1	1	1	Codes 1792-2047

Example:

39	**39**	**126**	**126**
39	**39**	**126**	**126**
39	**39**	**126**	**126**
39	**39**	**126**	**126**

EXAMPLE 8.7:
LZW coding.

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Some Basic Compression Methods: LZW Coding

39 39 126 126
39 39 126 126
39 39 126 126
39 39 126 126

TABLE 8.7
LZW coding example.

w	k	output	Index	Symbol
Currently Recognized Sequence	**Pixel Being Processed**	**Encoded Output**	**Dictionary Location (Code Word)**	**Dictionary Entry**
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

Decoding LZW

Let the bit stream received be:

39 39 126 126 256 258 260 259 257 126

In LZW, the dictionary which was used for encoding need not be sent with the image. A separate dictionary is built by the decoder, on the “fly”, as it reads the received code words.

Recognized	Encoded value	pixels	Dic. address	Dic. entry
	39	39		
39	39	39	256	39-39
39	126	126	257	39-126
126	126	126	258	126-126
126	256	39-39	259	126-39
256	258	126-126	260	39-39-126
258	260	39-39-126	261	126-126-39
260	259	126-39	262	39-39-126-126
259	257	39-126	263	126-39-39
257	126	126	264	39-126-126

Exercises

- **Use LZW to trace encoding the string ABRACADABRA.**
- **Write a Python program that encodes a given string using LZW.**
- **Write a Python program that decodes a given set of encoded codewords using LZW.**

Level 1 actions:

Compression (Encoding)

- Read the characters from text file
- Construct the LZW dictionary
- Generate the output
- Encode the symbols in the input file
- Save the compressed file
- Calculate code length and compression ratio

Decompression (Decoding)

- Read the stored data
- Restore the LZW dictionary
- Restore the symbols from the compressed data
- Save the restored text
- Compare the original and restored text

Level 1 actions-details:

Compression (Encoding)

- Read the characters from text file
- Construct the LZW dictionary by using the LZW compression algorithm
- Generate the output codes in the form of an integer array
- Convert integer array to a binary string by extracting codelength bits from each integer
- If the length of the binary string is not a multiple of 8, add zeros to the string (padding)
    - Add a 8 bits to the beginning of the string to indicate the number of zeros added to the end of the string
    - Now the_string_length % 8 == 0
- Extract bytes in the string and write them into a binary output file (compressed file)
- Calculate code length and compression ratio

Level 1 actions-details:

Decompression (Decoding)

- Read the bytes from binary file (compressed file)
- Get binary string and unpad the string
- Generate integer array
- Construct the LZW dictionary by using the LZW decompression algorithm
- Generate the decompressed text
- Save the decompressed text into decompressed.txt file
- Calculate the Compression Ratio (CR), Compression Factor(CF) and Space Saving(SS)

CR = size_of_compressed_file / size_of_original file

CF = size_of_original file / size_of_compressed_file

SS = (size_of_original_file - size_of_compressed_file) / size_of-original file

A sample Class

```
import math
import os
import image_tools
import numpy as np

class LZWCoding:

    def __init__(self, path, data_type):
        self.path = path
        self.data_type = data_type
        self.filename, self.file_extension = os.path.splitext(self.path)
        self.file_size = os.path.getsize(self.path)
        self.compressed_file_size = 0
        self.compression_ratio = 0
        self.code_length = 9
        if data_type == "image":
            self.img = image_tools.readPILimg(path)
            self.img_cols, self.img_rows = self.img.size
            if self.img.mode == "RGBA":
                self.img.convert("RGB").save(path)

    def compress(self, uncompressed):
        """Compress a string to a list of output symbols."""

    def calculate_compression_ratio(self, type=""):

    def pad_encoded_text(self, encoded_text):

    def get_byte_array(self, padded_encoded_text):

    def int_array_to_binary_string(self, int_array):

    def compress_text(self):

    def remove_padding(self, padded_encoded_text):

    def decompress_file(self, input_path):


def main():
    sample_text = LZWCoding("sample.txt", "text")
    compressed_file = sample_text.compress_text()
    sample_text.decompress_file(compressed_file)

if __name__ == "__main__":
    main()
```

LZW Compression

- ```
  def compress(uncompressed):
      """Compress a string to a list of output symbols."""

      # Build the dictionary.
      dict_size = 256
      dictionary = {chr(i): i for i in range(dict_size)}

      w = ""
      result = []
      for c in uncompressed:
          wc = w + c
          if wc in dictionary:
              w = wc
          else:
              result.append(dictionary[w])
              # Add wc to the dictionary.
              dictionary[wc] = dict_size
              dict_size += 1
              w = c

      # Output the code for w.
      if w:
          result.append(dictionary[w])
      return result
  ```
  ```

LZW Decompression

```
def decompress(compressed):
    """Decompress a list of output ks to a string."""
    from io import StringIO

    # Build the dictionary.
    dict_size = 256
    dictionary = {i: chr(i) for i in range(dict_size)}
    # use StringIO, otherwise this becomes O(N^2)
    # due to string concatenation in a loop
    result = StringIO()
    w = chr(compressed.pop(0))
    result.write(w)
    for k in compressed:
      if k in dictionary:
        entry = dictionary[k]
      elif k == dict_size:
        entry = w + w[0]
      else:
        raise ValueError('Bad compressed k: %s' % k)
      result.write(entry)

      # Add w+entry[0] to the dictionary.
      dictionary[dict_size] = w + entry[0]
      dict_size += 1

      w = entry
    return result.getvalue()
```

```
# How to use compression and decompression :

data1 = "TOBEORNOTTOBEORTOBEORNOT"
data="ababababab"
print("input data:",data)
compressed = compress(data)
print("compressed_data: ", compressed)
decompressed = decompress(compressed)
print("decompressed_data : ",decompressed)


# Program output:

input data: ababababab
compressed_data:  [97, 98, 256, 258, 257, 98]
decompressed_data :  ababababab
```

String-to-bytes.py

```
from LZW import LZWCoding
from integer_to_bytes import int_array_to_binary_string
import sys
array1 = [84,79,66]
path = "tobe.txt"
codelength = 12
l = LZWCoding(path, codelength)
bitstring = int_array_to_binary_string(array1, codelength)
print("integer codes: ", array1)
bitstr = int_array_to_binary_string(array1, codelength)
print("bit string: ",bitstr)
print("total number of bits: ", len(bitstr))
#print("integer from first 12 bits: ",int(bitstr[0:12],2))

padded = l.pad_encoded_text(bitstring)
print("length of padded string",len(padded), ",padded string: ", padded)
my_byte_array = l.get_byte_array(padded)
#print(my_byte_array)

bit_string = ""
for byte in my_byte_array:

    #byte = ord(byte)
    bits = bin(byte)[2:].rjust(8, '0')
    bit_string += bits
print("padded_bit_string:", bitstring)
encoded_text = l.remove_padding(bit_string)
decompressed_text = l.decompress(encoded_text)
print("decompressed data:",decompressed_text )
```

Extracting codelength bits from integers: Generate bit string

- ```
  def int_array_to_binary_string(self, int_array):
      import math
      bitstr = ""
      bits = self.codelength
      for num in int_array:
          for n in range(bits):
              if num & (1 << (bits - 1 - n)):
                  bitstr += "1"
              else:
                  bitstr += "0"
      return(bitstr)
  ```
  ```

Padding and extracting bytes

```
def pad_encoded_text(self, encoded_text):
    extra_padding = 8 - len(encoded_text) % 8
    for i in range(extra_padding):
        encoded_text += "0"

    padded_info = "{0:08b}".format(extra_padding)
    print("padded info: ", padded_info)
    encoded_text = padded_info + encoded_text
    return encoded_text

def get_byte_array(self, padded_encoded_text):
    if (len(padded_encoded_text) % 8 != 0):
        print("Encoded text not padded properly")
        exit(0)

    b = bytearray()
    for i in range(0, len(padded_encoded_text), 8):
        byte = padded_encoded_text[i:i + 8]
        b.append(int(byte, 2))
    return b
```

Decoder: remove_padding to get the encoded integer codes of LZW

```
def remove_padding(self, padded_encoded_text):
    padded_info = padded_encoded_text[:8]
    extra_padding = int(padded_info, 2)
    padded_encoded_text = padded_encoded_text[8:]
    encoded_text = padded_encoded_text[:-1 * extra_padding]
    int_codes = []
    for bits in range(0, len(encoded_text),self.codelength):
        int_codes.append(int(encoded_text[bits:bits+self.codelength],2))
    return int_codes
```

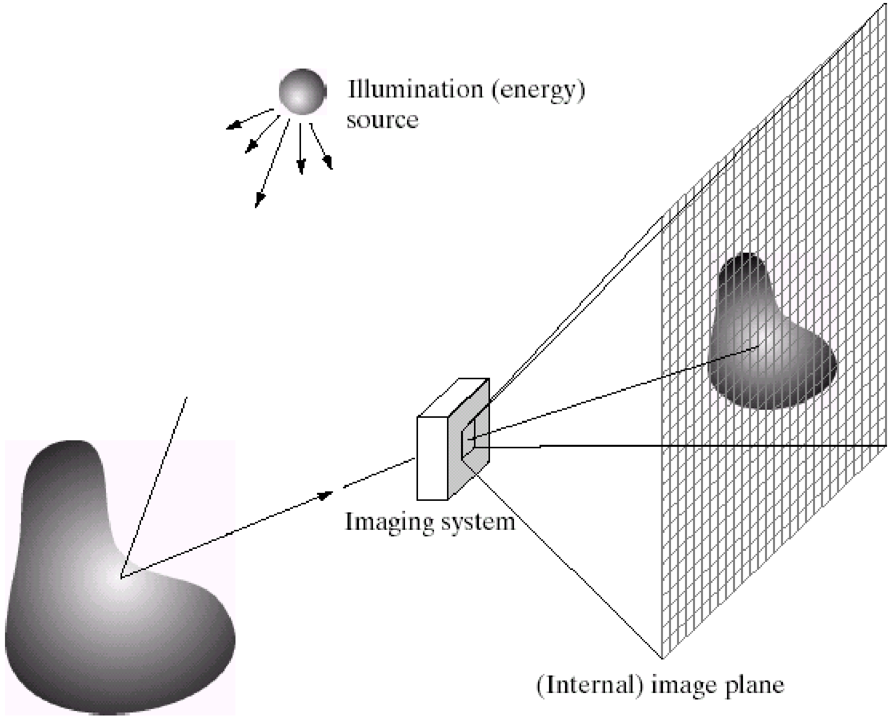
String to bytes – sample run

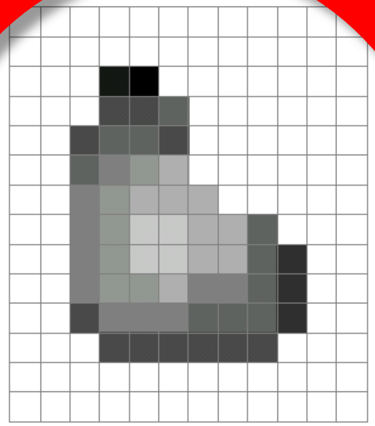
```
/Users/muhittingokmen/PycharmProjects/labeling/venv/bin/python /Users/muhittingokmen/PycharmProjects/LZW/String-to-bytes.py
integer codes:  [84, 79, 66]
bit string:  000001010100000001001111000001000010
total number of bits:  36
integer from first 12 bits:  84
12 bit slice:  000001010100 , integer code:  84
12 bit slice:  000001001111 , integer code:  79
12 bit slice:  000001000010 , integer code:  66
[84, 79, 66]
integer codes:  [84, 79, 66]
bit string:  000001010100000001001111000001000010
total number of bits:  36
padded info:  00000100
length of padded string 48 ,padded string:  000001000000010101000000010011110000010000100000
padded_bit_string: 000001010100000001001111000001000010
from decompressed: compressed data:  [84, 79, 66]
decompressed data: TOB

Process finished with exit code 0
```

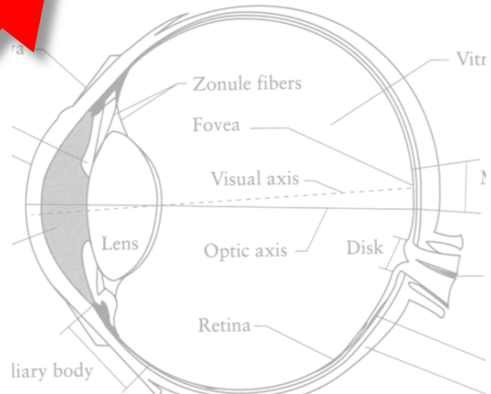
Level 2:Image Compression

What is an image?





Digital Camera



The Eye

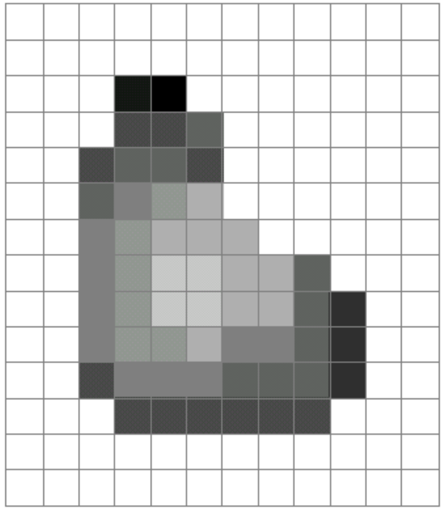
We' ll focus on these in this class

(More on this process later)

Source: A. Efros

What is an image?

- A grid (matrix) of intensity values



255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

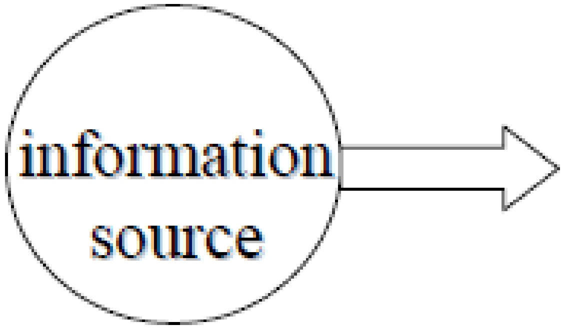
Lossless vs. Lossy Compression

Compression techniques:

- lossless
- Lossy

- Lossless (or information preserving) compression: Images can be compressed and restored without any loss of information. The original image and restored image are bit by bit identical.(.e.g. medical imaging, satellite imaging). LZW coding is lossless compression technique.
- Lossy (or information reducing) compression: perfect recovery not possible. Larger data compression (e.g.,TV signals, teleconferencing)

Entropy



Source alphabet

$$\{a_1, a_2, \ldots, a_L\}$$

- Let $p(a_l), l = 1, 2, \ldots, L$ be the probability of each symbol

- Then the entropy (or uncertainty) of the source is given by

$$H = -\sum_{l=1}^{L} p(a_l) \log(p(a_l)) \quad \text{bits/symbol}$$

Computing the Entropy of an Image

8-bit gray level source- statistically independent pixels emission

- Consider the 8-bit image:

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

Gray Level	Counts	Probability
21	12	3/8
95	4	1/8
169	4	1/8
243	12	3/8

$$H = 1.81 \text{ bits/pixel}$$

(first-order estimate)

Example:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

EXAMPLE 8.7:
LZW coding.

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Some Basic Compression Methods: LZW Coding

39 39 126 126
39 39 126 126
39 39 126 126
39 39 126 126

TABLE 8.7
LZW coding example.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

Decoding LZW

Let the bit stream received be:

39 39 126 126 256 258 260 259 257 126

In LZW, the dictionary which was used for encoding need not be sent with the image. A separate dictionary is built by the decoder, on the "fly", as it reads the received code words.

Recognized	Encoded value	pixels	Dic. address	Dic. entry
	39	39		
39	39	39	256	39-39
39	126	126	257	39-126
126	126	126	258	126-126
126	256	39-39	259	126-39
256	258	126-126	260	39-39-126
258	260	39-39-126	261	126-126-39
260	259	126-39	262	39-39-126-126
259	257	39-126	263	126-39-39
257	126	126	264	39-126-126

Level 2 actions:

Compression

- Read the image file
- Scan the image and construct the LZW Dictionary
- Store the binary code instead of gray levels
- Save the compressed file
- Calculate entropy, average code length, size of the compressed file and compression ratio

Decompression

- Read the stored data
- Restore the LZW dictionary
- Restore the image from the compressed data
- Save the restored image
- Compare the original image and restored image

Level 3:Image Compression (Gray level differences)

Difference image

- Since the entropy of the source is the lower limit of the average code length, we would like to reduce the entropy of the image by taking the differences between successive pixels.
- Keep the first column and replace following with the aritmetic difference between adjacent columns. And then keep the first row, and replace following pixels in the first column with the arithmetic difference between adjacent pixel in the first column.

Difference image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

Original Image

arr[i][j]
N=5
M=7

Gray Level	Counts	Probability
21	12	3/8
95	4	1/8
169	4	1/8
243	12	3/8
	H = 1.65 bits/pixel	

21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0

Intermediate Difference Image:

```
for i in range(N):
    for j in range(1,M):
            darr[i][j] = arr[i][j] - arr[i][j-1]
```

Gray Level	Counts	Probability
0	16	1/2
21	4	1/8
74	12	3/8
	H = 1.40 bits/pixel	

21							
0	0	0	74	74	74	0	0
0	0	0	74	74	74	0	0
0	0	0	74	74	74	0	0
0	0	0	74	74	74	0	0

Difference Image:

```
Diff_arr = darr
pivot = darr[0[0]
diff_arr[0][0] = darr[0[0] – pivot
for i in range(1, N):
    diff_arr[i][0] = darr[i][0] - darr[i-1][0]
```

Gray Level	Counts	Probability
0	20	1/2
74	12	3/8
	H = 1.03 bits/pixel	

Level 3 actions:

Compression

- Read the image file
- Obtain the difference image by taking the row-wise differences between successive gray levels in each row, starting from the second pixel in a row.
- Take the column-wise differences between successive pixels in the first column, starting from second pixel.
- Scan the difference image,
- Construct the LZW dictionary from the difference image
- Store the binary code instead of gray levels
- Save the compressed file
- Calculate entropy, average code length, the size of the compressed file and compression ratio

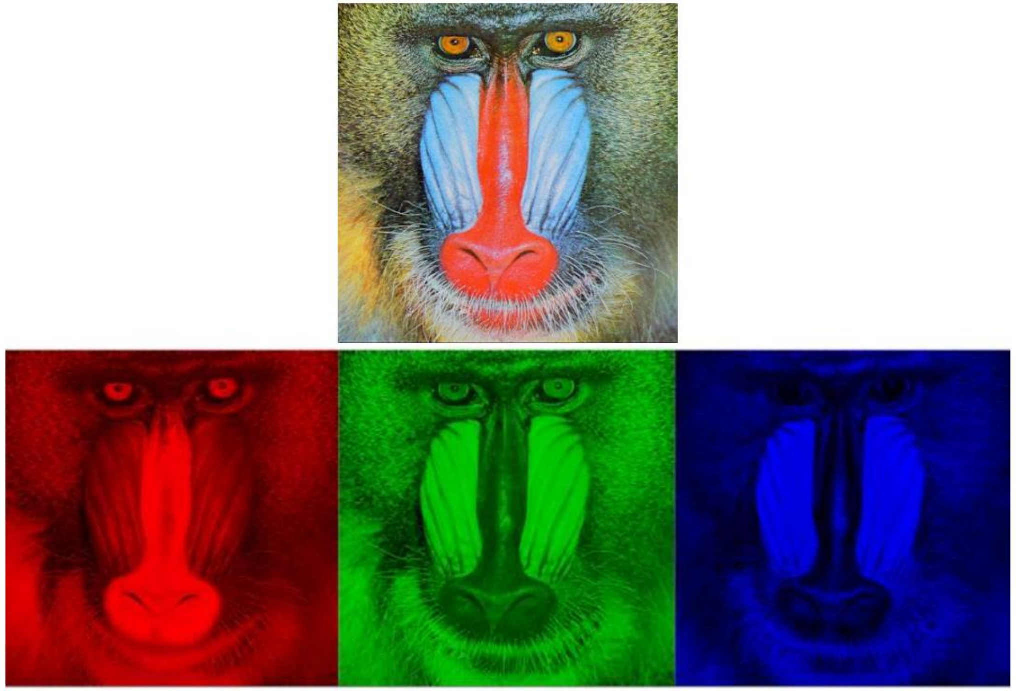
Decompression

- Read the stored data
- Restore the LZW dictionary
- Restore the difference image from the compressed data
- Restore the original image from differences
- Save the restored image
- Compare the original image and restored image

Level 4: Color Image Compression

Color Images

- Image has three channels (bands)
- Each channel spans a-bit values.



- Level 4: Apply LZW coding to each color components. (repeat level 2 for each color component)
- Level 5: Apply LZW coding to the gray level differences at each components. (repeat level 3 for each color component)

Level 4 actions:

Compression

- Read the image file, decompose into color components (you will generate three gray level images from a color image by separating RGB components)
- Scan each image and construct the LZW Dictionary
- Store the binary code instead of gray levels
- Save the compressed file
- Calculate entropy, average code length, size of the compressed file and compression ratio

Decompression

- Read the stored data
- Restore the LZW dictionary
- Restore the image from the compressed data
- Save the restored image
- Compare the original image and restored image

Level 5 actions:

Compression

- Read the color image file, decompose into color componts (you will generate three gray level image from a color image by separating R,GiB components)
- Read the image file
- Obtain the difference image by taking the row-wise differences between successive gray levels in each row, starting from the second pixel in a row.
- Take the column-wise differences between successive pixels in the first column, starting from second pixel.
- Scan the difference image,
- Construct the LZW dictionary from the difference image
- Store the binary code instead of gray levels
- Save the compressed file
- Calculate entropy, average code length, the size of the compressed file and compression ratio

Decompression

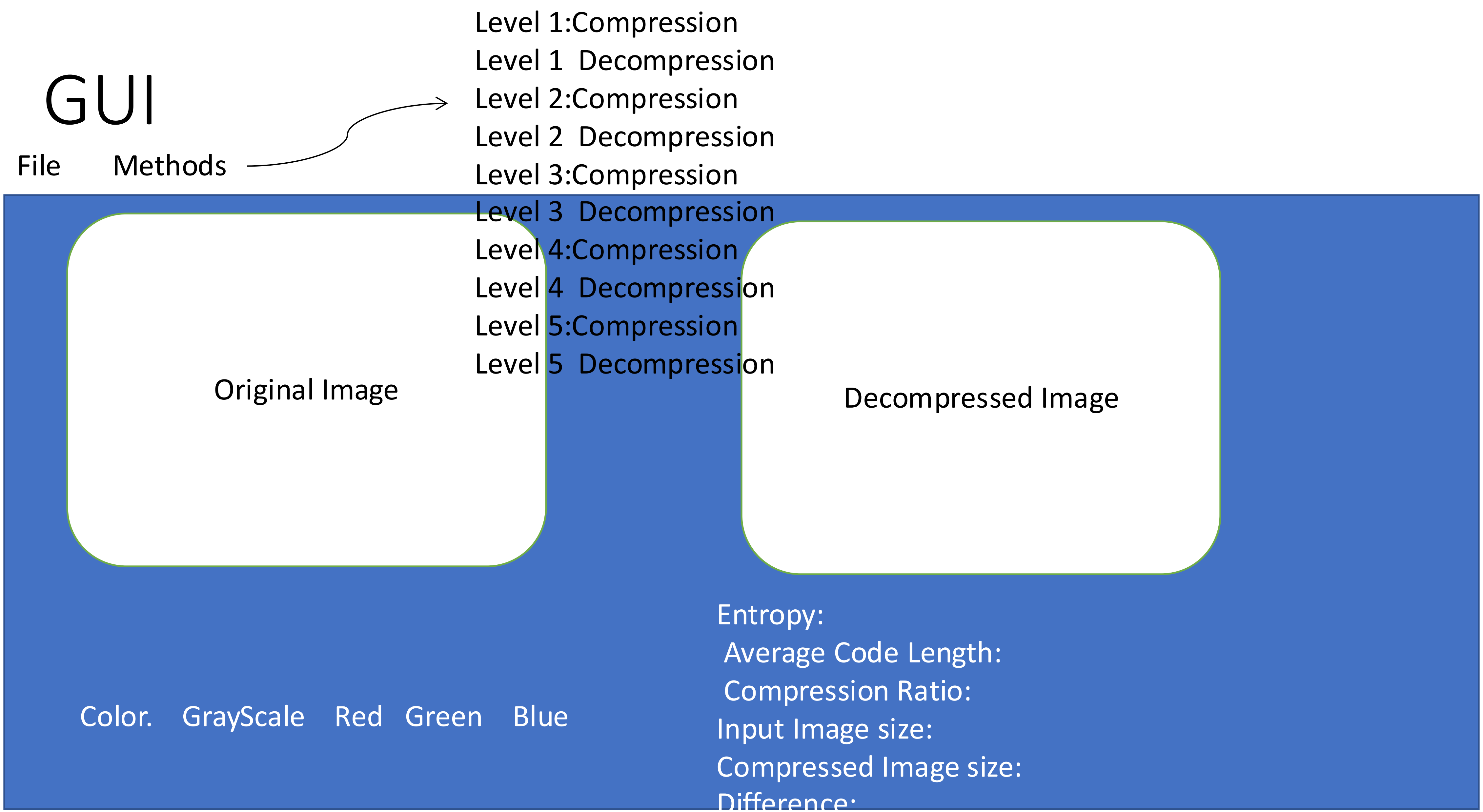
- Read the stored data
- Restore the LZW dictionary
- Restore the difference image from the compressed data
- Restore the original image from differences
- Save the restored image
- Compare the original image and restored image

GUI

Design and Develop a GUI to achieve the following:

- The user will be able to
  - Select a file from a directory and load it
  - Select image or text file
  - Select a method (gray levels or differences)
  - Save the compressed image into a file
  - Reads a compressed file, show the decompressed text/image file
  - Shows entropy, average code length and compression ratio, size of compressed file.

GUI


File
Methods
Level 1:Compression
Level 1 Decompression
Level 2:Compression
Level 2 Decompression
Level 3:Compression
Level 3 Decompression
Level 4:Compression
Level 4 Decompression
Level 5:Compression
Level 5 Decompression
Original Image
Decompressed Image
Color.
GrayScale
Red
Green
Blue
Entropy:
Average Code Length:
Compression Ratio:
Input Image size:
Compressed Image size:
Difference:

Project 1 Self Evaluation guide: You are expected to fill the row 10 of this file to evaluate your work on Project1. After entering your student info in the first 5 columns, enter a single number between 0-5 to each cell to evaluate your achievement. You enter a number between 0 to 5 to each cell, meaning 0: 0%, 1: 10%. 2: 25%, 3: 50%, 4: 75%, 5: 100% of the expected work was completed.

					Level1(30%)												Level2(30%)									Level3(10%)				Level4 (10%)				Level5 (10%)				Level6 (10%)			80%	6%	6%	8%
					Read the text file	Calculate probality	Construct the LZW dictionary	Encode the file content	Save the compressed file	Calculate entropy, average code length and compression ratio	Read the stored data	Restore the LZW dictionary	Fixed or Progressive code length	Restore the symbols from the compressed data	Save the restored text	Compare the original and restored text	Read image	Construct the LZW dictionary	Read the stored data	Encode the image and save into file	Packcodes into bytearray	Calculate entropy, average code length and compression ratio	Read compressed file, Construct tree/intensities	Restore image intensities	Compare images	Calculate differences	Encode the difference image and save into file	Restore image intensities	Compare images	Color decomposition	Encoding each component and saving into file(s)	Restoring the components and color image	Compare images	Color decomposition and differences	Encoding each component and saving into file(s)	Restoring the components and color image	Compare images	Select files, methods	Compare images	Show calculations	CodingTotal	Demo/video	Presentation	Report
				Percentages	1	2	4	4	4	3	2	2	2	2	2	2	2	5	2	4	4	3	4	4	2	3	3	3	1	3	3	3	1	3	3	3	1	3	4	3	100	100	100	100
STUDENT NO	Class	Name	SURNAME	EMAIL																																								

Deliverables

At the end of the project

- You should prepare a project report with
    - Title page
    - Abstract
    - Description of the project
    - Description of your solution
        - Use UML diagrams to demonstrate your program
    - Examples of test files, input, outputs showing that your program works correctly
    - Screen shots of the GUI
    - List of achievements (what did you learn during the project)
    - References
- Prepare a 5-minute video presentation together with a power point file (Do not directly show your code, but start with Use cases, class diagrams etc.)
- Submit your report, presentation and code to the blackboard
- Submit the self-evaluation rubric to the blackboard

Recommended Calendar for the project 1

- February 14, assignment of the project, design and implement LZW Encoding & Decoding
- February 21, design and implement Gray Level Image compression (Level 2 and 3)
- February 28, design and implement Color image compression and GUI (Level 4,5)
- March 7, design and implement Color image compression and GUI (Level 4,5 and 6)
- March 14, project demonstration and submission of the report, code and presentation