TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

FOURTH YEAR MULTIMEDIA SYSTEM PROJECT REPORT

ON

IMPLEMENTATION OF LZW ALGORITHM

SUBMITTED TO:

Prof. Dr. SUBARNA SHAKYA

SUBMITTED BY:

SHIVA SHRESTHA(073/BCT/41)

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

DECEMBER, 2020

TITLE: IMPLEMENTATION OF LZW ALGORITHM

OBJECTIVES

- To compress and decompress the size of file using LZW algorithm
- To understand the working principle of LZW algorithm

INTRODUCTION

Lempel–Ziv–Welch (**LZW**) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. LZW algorithm is a greedy algorithm in that it tries to recognize increasingly longer and longer phrases that are repetitive, and encode them. A typical file data compression algorithm used in many file compression schemes such as GIF files, PDF, etc. These are lossless compression algorithms in which no data is lost, and the original file can be entirely reconstructed from the encoded message file.

LZW algorithm is simple to implement and has the potential for very high throughput in hardware implementations. It is the algorithm of the widely used Unix file compression utility compress, and is used in the GIF image format. LZW is the foremost technique for general purpose data compression due to its simplicity and versatility. It is the basis of many PC utilities that claim to “double the capacity of your hard drive”.

METHODOLOGY

WORKING PRINCIPLE

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression.

Characteristic features of LZW includes,

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

LZW COMPRESSION ALGORITHM

The idea of the compression algorithm is the following: as the input data is being processed, a dictionary keeps a correspondence between the longest encountered words and a list of code values. The words are replaced by their corresponding codes and so the input file is compressed. Therefore, the efficiency of the algorithm increases as the number of long, repetitive words in the input data increases.

LZW ENCODING

```
PSEUDOCODE
Initialize table with single character strings
P = first input character
WHILE not end of input stream
     C = next input character
     IF P + C is in the string table
       P = P + C
     ELSE
       output the code for P
     add P + C to the string table
        P = C
      END WHILE
 output code for P
```

LZW DECOMPRESSION ALGORITHM

The LZW decompressor creates the same string table during decompression. It starts with the first 256 table entries initialized to single characters. The string table is updated for each character in the input stream, except the first one. Decoding achieved by reading codes and translating them through the code table being built.

LZW DECODING

```
*   PSEUDOCODE
1   Initialize table with single character strings
2   OLD = first input code
3   output translation of OLD
4   WHILE not end of input stream
5       NEW = next input code
6       IF NEW is not in the string table
7               S = translation of OLD
8               S = S + C
9       ELSE
10              S = translation of NEW
11      output S
12      C = first character of S
13      OLD + C to the string table
14      OLD = NEW
15  END WHILE
```

IMPLEMENTATION

As we know about what and how the LZW algorithm works, we are going to implement this in real system by compressing and decompressing the particular files that locates in our system. We will be using Python, as a programming language to implement this algorithm as it is robust and easy to understand language.

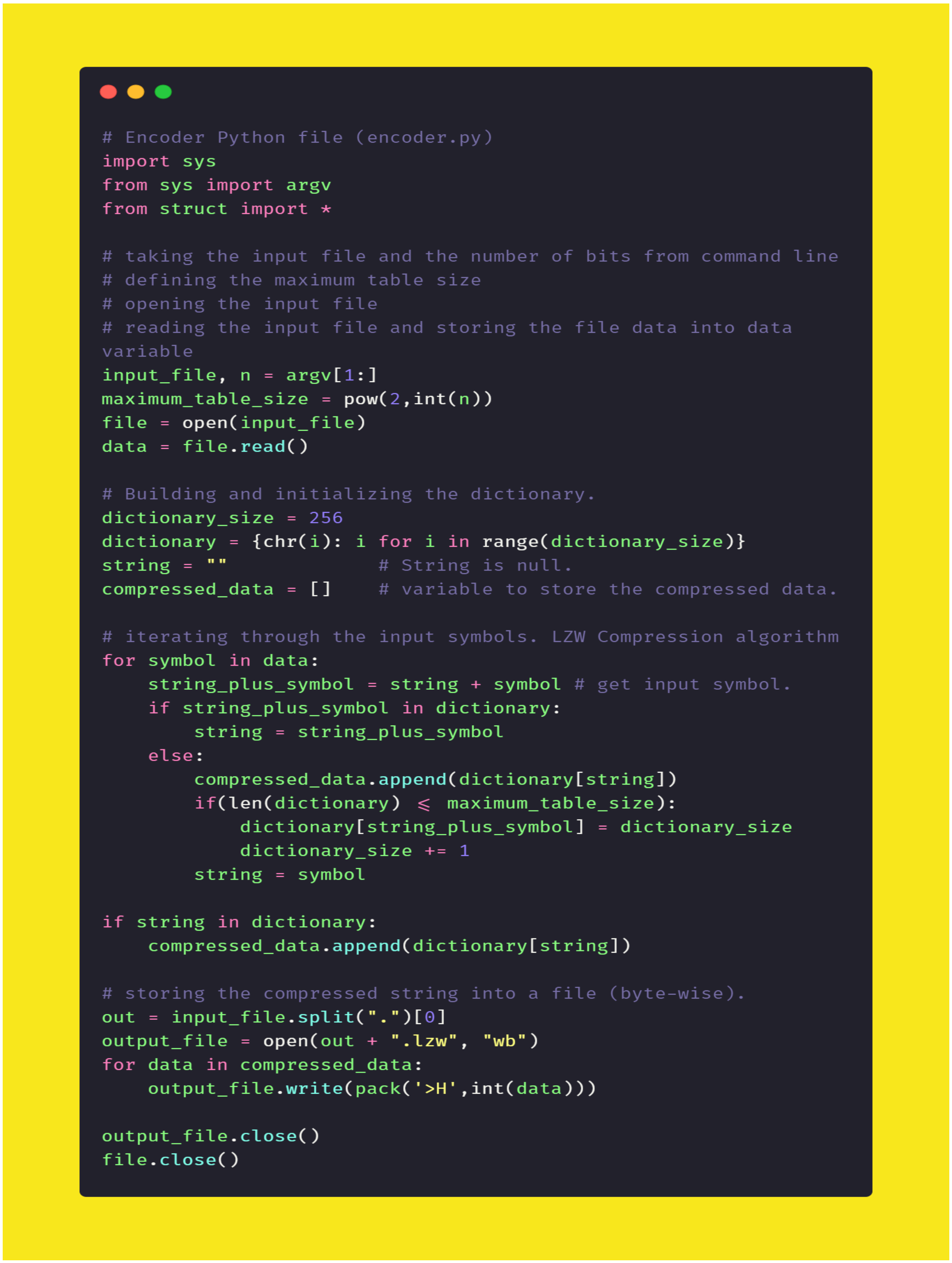
We made two separate python files named encoder.py and decoder.py respectively.

encoder.py

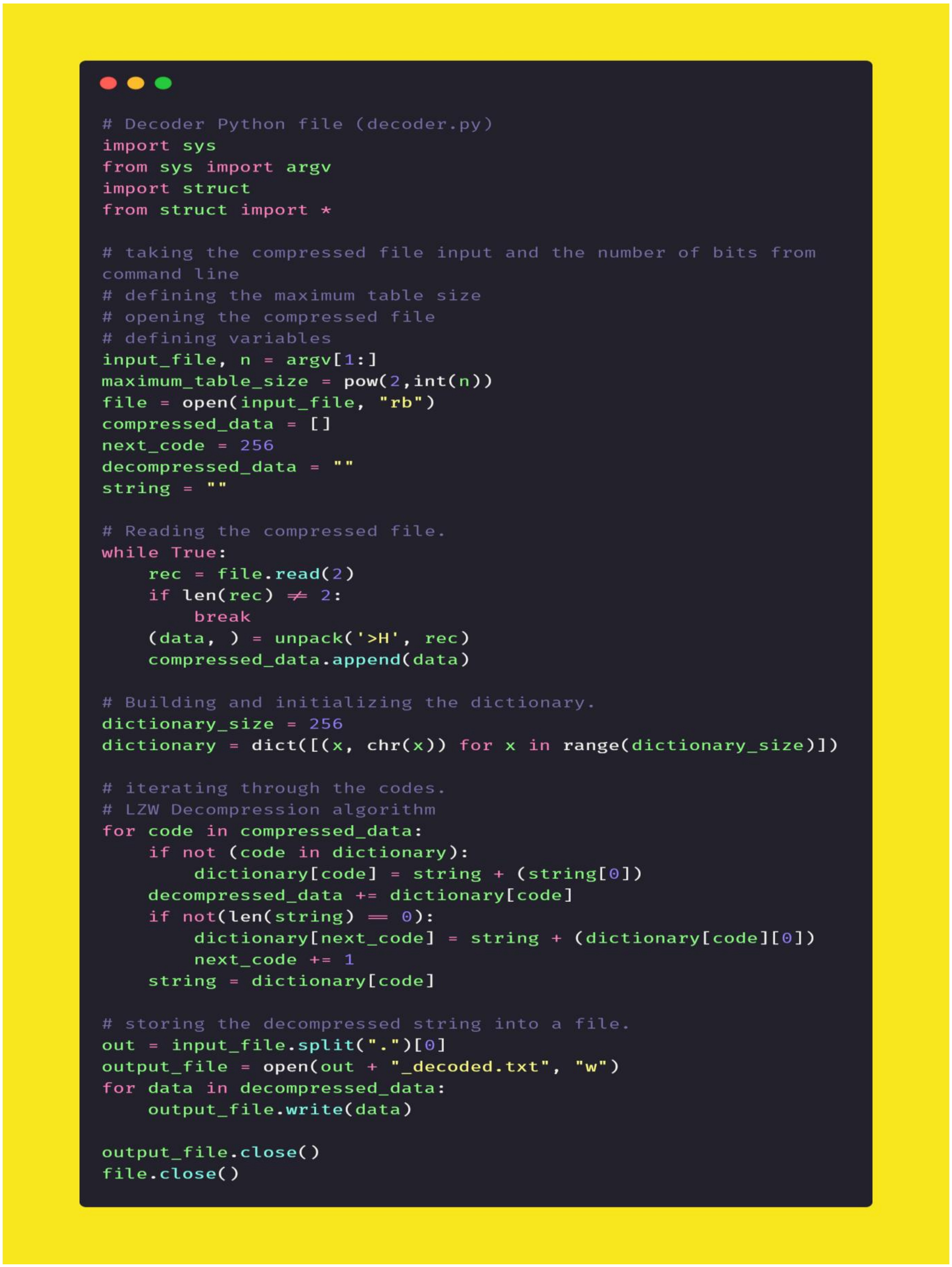
The input data is encoded using the encoder.py file. The dictionary of size 256 is built and initialized, using the python dictionary data structure in the dictionary, key are characters and values are the ASCII values, the LZW compression algorithm is applied and we get the compressed data. The program outputs the compressed data and stores it to an output file named inputFileName.lzw.

decoder.py

The compressed data is decompressed using the decoder.py file. The dictionary of size 256 is built and initialized, using the python dictionary data structure in the dictionary, key are characters and values are the ASCII values, the LZW decompression algorithm is applied and we get the decompressed data, the program outputs the decompressed data and stores it to an output file named inputFileName_decoded.txt



```python
# Encoder Python file (encoder.py)
import sys
from sys import argv
from struct import *

# taking the input file and the number of bits from command line
# defining the maximum table size
# opening the input file
# reading the input file and storing the file data into data
variable
input_file, n = argv[1:]
maximum_table_size = pow(2,int(n))
file = open(input_file)
data = file.read()

# Building and initializing the dictionary.
dictionary_size = 256
dictionary = {chr(i): i for i in range(dictionary_size)}
string = ""             # String is null.
compressed_data = []    # variable to store the compressed data.

# iterating through the input symbols. LZW Compression algorithm
for symbol in data:
    string_plus_symbol = string + symbol # get input symbol.
    if string_plus_symbol in dictionary:
        string = string_plus_symbol
    else:
        compressed_data.append(dictionary[string])
        if(len(dictionary) <= maximum_table_size):
            dictionary[string_plus_symbol] = dictionary_size
            dictionary_size += 1
        string = symbol

if string in dictionary:
    compressed_data.append(dictionary[string])

# storing the compressed string into a file (byte-wise).
out = input_file.split(".")[0]
output_file = open(out + ".lzw", "wb")
for data in compressed_data:
    output_file.write(pack('>H',int(data)))

output_file.close()
file.close()
```



```python
# Decoder Python file (decoder.py)
import sys
from sys import argv
import struct
from struct import *

# taking the compressed file input and the number of bits from
command line
# defining the maximum table size
# opening the compressed file
# defining variables
input_file, n = argv[1:]
maximum_table_size = pow(2,int(n))
file = open(input_file, "rb")
compressed_data = []
next_code = 256
decompressed_data = ""
string = ""

# Reading the compressed file.
while True:
    rec = file.read(2)
    if len(rec) != 2:
        break
    (data, ) = unpack('>H', rec)
    compressed_data.append(data)

# Building and initializing the dictionary.
dictionary_size = 256
dictionary = dict([(x, chr(x)) for x in range(dictionary_size)])

# iterating through the codes.
# LZW Decompression algorithm
for code in compressed_data:
    if not (code in dictionary):
        dictionary[code] = string + (string[0])
    decompressed_data += dictionary[code]
    if not(len(string) == 0):
        dictionary[next_code] = string + (dictionary[code][0])
        next_code += 1
    string = dictionary[code]

# storing the decompressed string into a file.
out = input_file.split(".")[0]
output_file = open(out + "_decoded.txt", "w")
for data in decompressed_data:
    output_file.write(data)

output_file.close()
file.close()
```

RESULT AND ANALYSIS

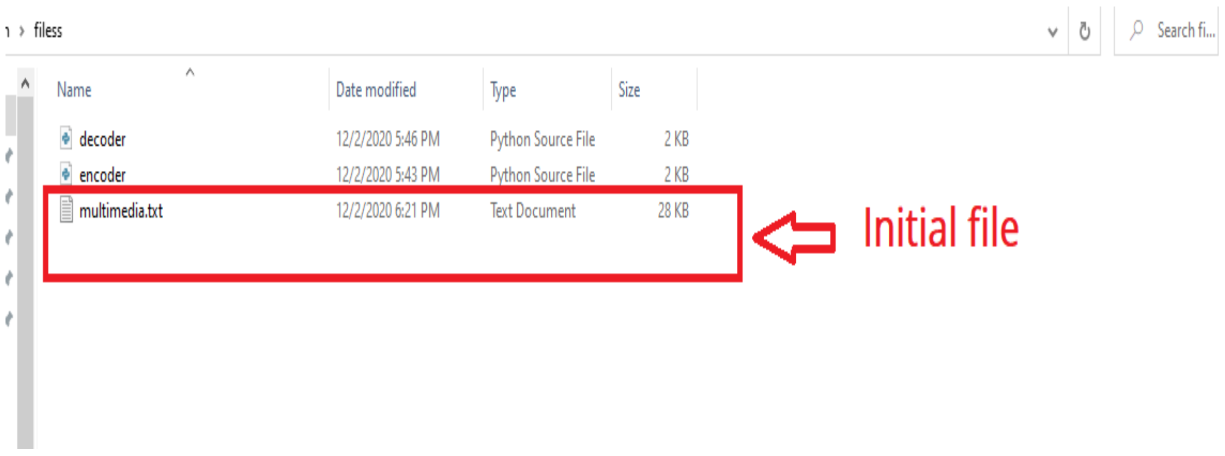
We successfully added two python files in our working directory plus one file that needs to be compressed. We created text file named "multimedia.txt" that size of (10,723 bytes(about 11KB). Now we compressed that file using python file 'encoder.py', which creates new compressed version of actual text file named "multimedia.lzw". Our compressed new file size is 5208 bytes(about 6KB) that reduced by about 46% which is huge reduction.

Similarly, we can decompressed the compressed file using python file 'decoder.py', which creates new text file named "multimedia_decoded.txt" which file size(11KB) is same as of the file "multimedia.txt". Here we proved that LZW is lossless algorithm.

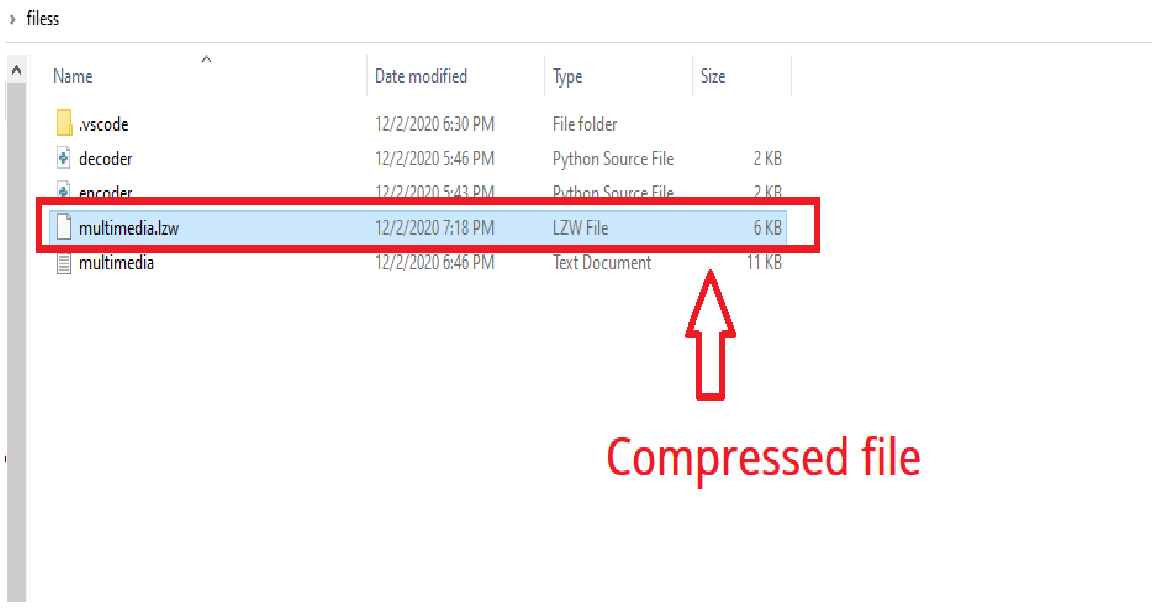
DISCUSSTION AND CONCLUSION

Implementing LZW algorithm in Python needs little bit knowledge of python data structures which we used while writing codes. We know actually how LZW algorithm works and we proved that LZW is lossless algorithm by analyzing the decoding file of compressed one. We were familiar with huge applicability of this method and how internet data's are heavily dependent on this algorithm. It reduces the cost of extra storage needed while performing huge computations.

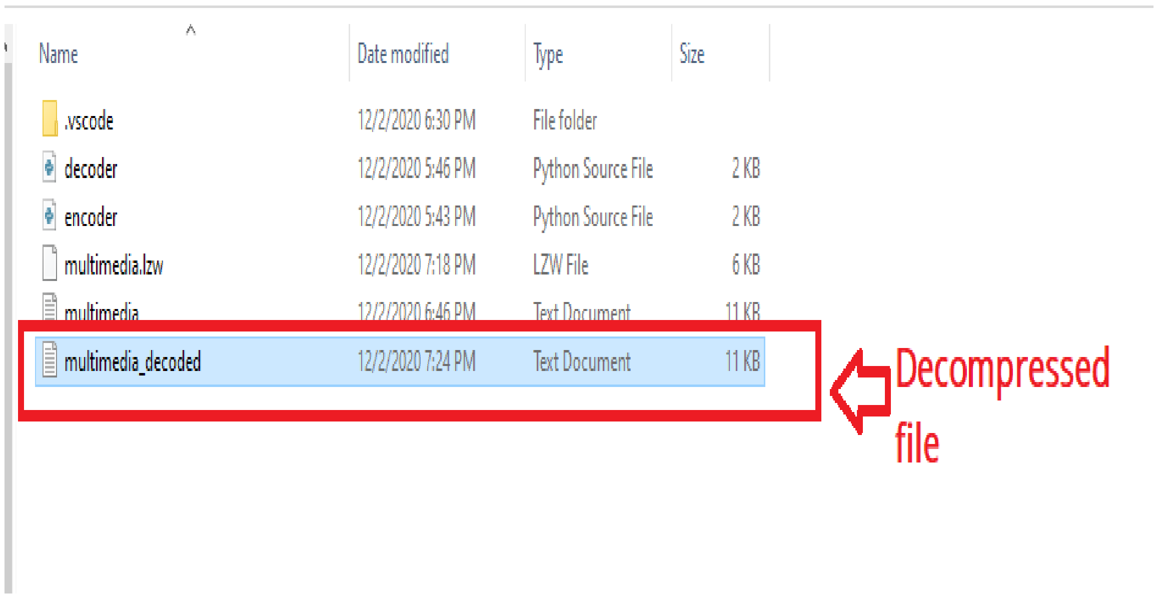
During research period, we found that this algorithm is widely used by top tech companies for data compression and it compresses repetitive sequences of data very well. At a glance, we familiar with compression and decompression of particular file using LZW algorithm.



Name	Date modified	Type	Size
decoder	12/2/2020 5:46 PM	Python Source File	2 KB
encoder	12/2/2020 5:43 PM	Python Source File	2 KB
multimedia.txt	12/2/2020 6:21 PM	Text Document	28 KB



Name	Date modified	Type	Size
.vscode	12/2/2020 6:30 PM	File folder	
decoder	12/2/2020 5:46 PM	Python Source File	2 KB
encoder	12/2/2020 5:43 PM	Python Source File	2 KB
multimedia.lzw	12/2/2020 7:18 PM	LZW File	6 KB
multimedia	12/2/2020 6:46 PM	Text Document	11 KB



Name	Date modified	Type	Size
.vscode	12/2/2020 6:30 PM	File folder	
decoder	12/2/2020 5:46 PM	Python Source File	2 KB
encoder	12/2/2020 5:43 PM	Python Source File	2 KB
multimedia.lzw	12/2/2020 7:18 PM	LZW File	6 KB
multimedia	12/2/2020 6:46 PM	Text Document	11 KB
multimedia_decoded	12/2/2020 7:24 PM	Text Document	11 KB

References

[1] "The Data Compression Guide," [Online]. Available: https://sites.google.com/site/datacompressionguide/lzw.

[2] "LZW-Compressor-in-Python," [Online]. Available: https://github.com/adityagupta3006/LZW-Compressor-in-Python.

[3] A. R. Saikia, "LZW (Lempel–Ziv–Welch) Compression technique," GeeksforGeeks, 09 09 2019. [Online]. Available: https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/. [Accessed 02 12 2020].