TRIBHUVAN UNIVERSITY

Institute of Engineering

HIMALAYA COLLEGE OF ENGINEERING

Department of Electronics and Computer Engineering

Report on LZW Compression

Submitted by:

Name: Dristi Dugar

Roll no: 2073/BCT/13

Submitted to: Subarna Shakya

1. Introduction

LZW compression is the compression of a file into a smaller file using a table-based lookup algorithm invented by Abraham Lempel, Jacob Ziv, and Terry Welch. Two commonly-used file formats in which LZV compression is used are the GIF image format served from Web sites and the TIFF image format. LZW compression is also suitable for compressing text files.

A particular LZW compression algorithm takes each input sequence of bits of a given length (for example, 12 bits) and creates an entry in a table (sometimes called a "dictionary" or "codebook") for that particular bit pattern, consisting of the pattern itself and a shorter code. As input is read, any pattern that has been read before results in the substitution of the shorter code, effectively compressing the total amount of input to something smaller. Unlike earlier approaches, known as LZ77 and LZ78, the LZW algorithm does include the look-up table of codes as part of the compressed file. The decoding program that uncompresses the file is able to build the table itself by using the algorithm as it processes the encoded input.

The LZW algorithm is a very common compression technique. This algorithm is typically used in GIF and optionally in PDF and TIFF. Unix's 'compress' command, among other uses. It is lossless, meaning no data is lost when compressing. The 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.

2. Objective

The major objectives of this report can be listed as:

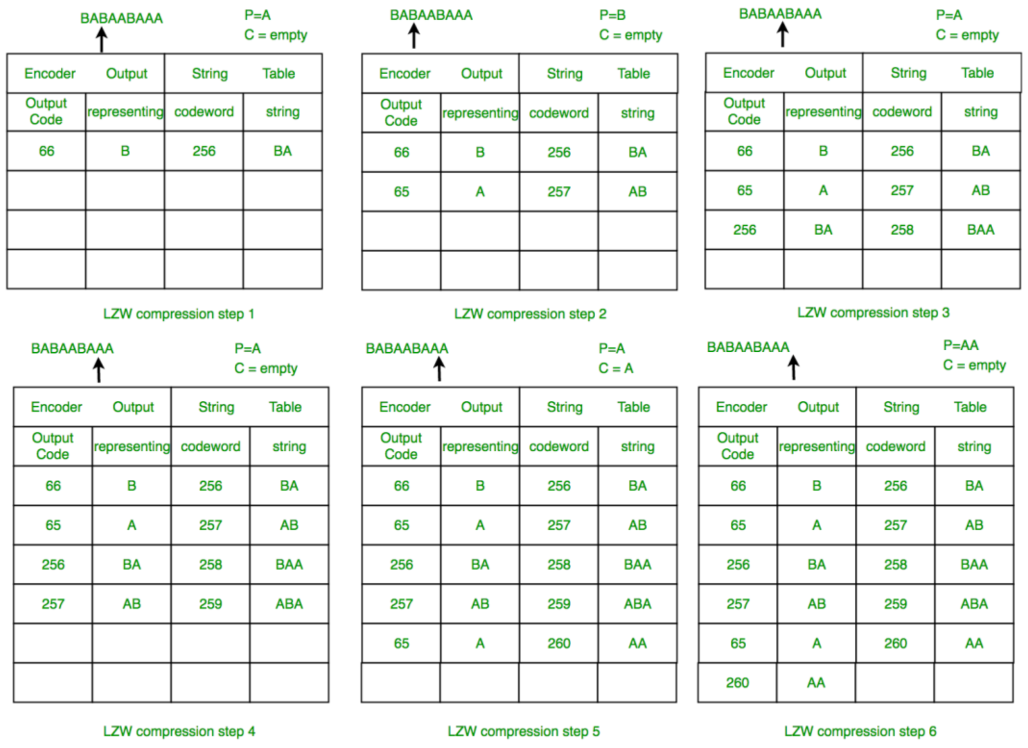
1. To understand LZW algorithm for lossless compression.
2. To apply LZW compression and decompression and observe the results.

3. Methodology

3.1 Encoding

A high level view of the encoding algorithm is shown here:

1. Initialize the dictionary to contain all strings of length one.
2. Find the longest string W in the dictionary that matches the current input.
3. Emit the dictionary index for W to output and remove W from the input.
4. Add W followed by the next symbol in the input to the dictionary.
5. Go to Step 2.



BABAABAAA P=A C = empty

Encoder	Output	String	Table
Output Code	representing	codeword	string
66	B	256	BA

LZW compression step 1

BABAABAAA P=B C = empty

Encoder	Output	String	Table
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB

LZW compression step 2

BABAABAAA P=A C = empty

Encoder	Output	String	Table
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA

LZW compression step 3

BABAABAAA P=A C = empty

Encoder	Output	String	Table
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA

LZW compression step 4

BABAABAAA P=A C = A

Encoder	Output	String	Table
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA
65	A	260	AA

LZW compression step 5

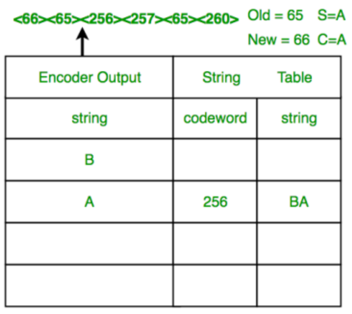
BABAABAAA P=AA C = empty

Encoder	Output	String	Table
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA
65	A	260	AA
260	AA		

LZW compression step 6

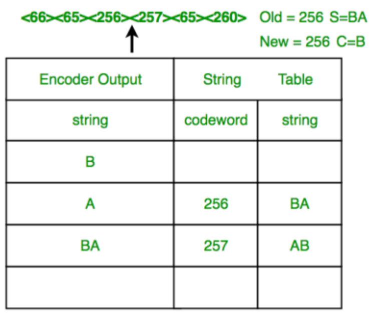
3.2 Decoding

The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the dictionary. However, the full dictionary is not needed, only the initial dictionary that contains single-character strings (and that is usually hard coded in the program, instead of sent with the encoded data). Instead, the full dictionary is rebuilt during the decoding process the following way: after decoding a value and outputting a string, the decoder concatenates it with the first character of the *next* decoded string (or the first character of current string, if the next one can't be decoded; since if the next value is unknown, then it must be the value added to the dictionary in this iteration, and so its first character is the same as the first character of the current string), and updates the dictionary with the new string. The decoder then proceeds to the next input (which was already read in the previous iteration) and processes it as before, and so on until it has exhausted the input stream.


<66><65><256><257><65><260>
Old = 65 S=A
New = 66 C=A

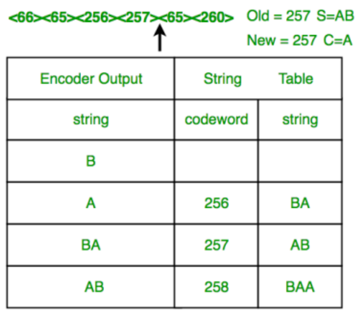
Encoder Output	String Table	
string	codeword	string
B		
A	256	BA

LZW compression step 1

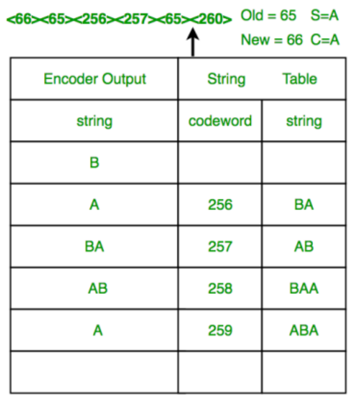

<66><65><256><257><65><260>
Old = 256 S=BA
New = 256 C=B

Encoder Output	String Table	
string	codeword	string
B		
A	256	BA
BA	257	AB

LZW compression step 2

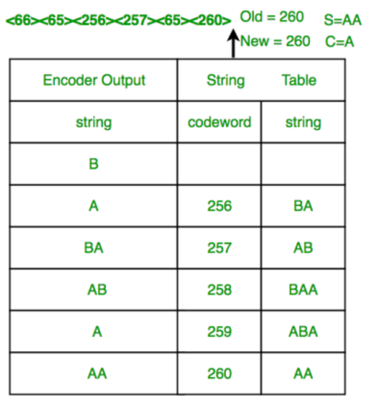

<66><65><256><257><65><260>
Old = 257 S=AB
New = 257 C=A

Encoder Output	String Table	
string	codeword	string
B		
A	256	BA
BA	257	AB
AB	258	BAA


<66><65><256><257><65><260>
Old = 65 S=A
New = 66 C=A

Encoder Output	String Table	
string	codeword	string
B		
A	256	BA
BA	257	AB
AB	258	BAA
A	259	ABA

LZW compression step 4


<66><65><256><257><65><260>
Old = 260 S=AA
New = 260 C=A

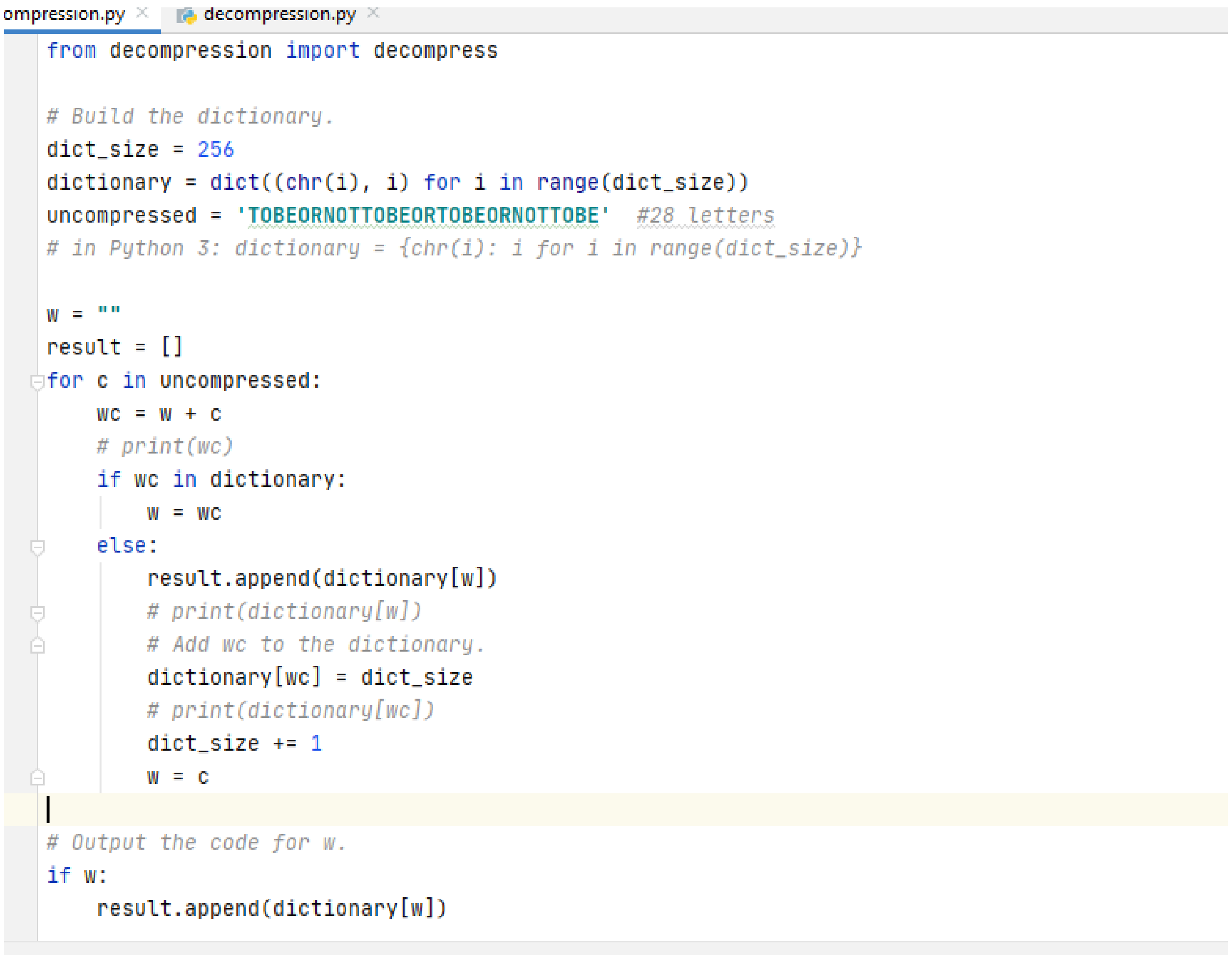
Encoder Output	String Table	
string	codeword	string
B		
A	256	BA
BA	257	AB
AB	258	BAA
A	259	ABA
AA	260	AA

LZW compression step 5

4. Results and Analysis

The final result can be viewed as below where a 28 lettered string was converted into a list of 17 items:

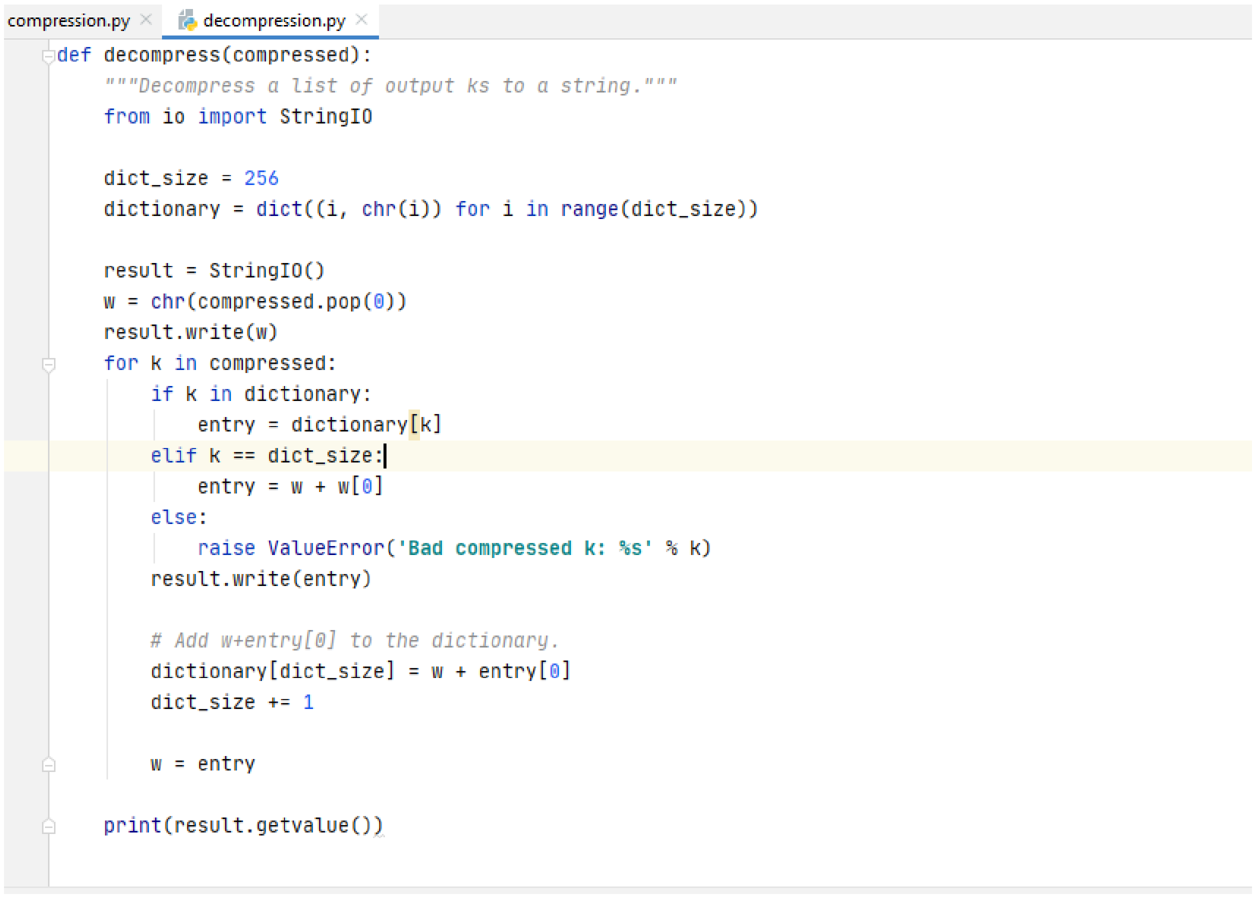
Compression algorithm:



ompression.py × decompression.py ×

```
from decompression import decompress

# Build the dictionary.
dict_size = 256
dictionary = dict((chr(i), i) for i in range(dict_size))
uncompressed = 'TOBEORNOTTOBEORTOBEORNOTTOBE'  #28 letters
# in Python 3: dictionary = {chr(i): i for i in range(dict_size)}

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

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

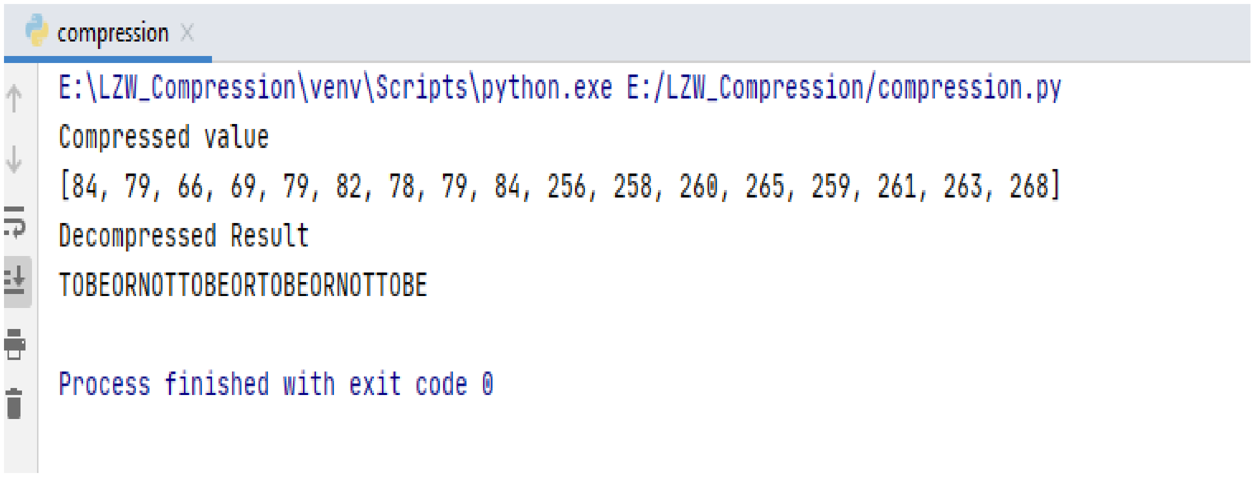
Decompression algorithm:



compression.py decompression.py

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

    dict_size = 256
    dictionary = dict((i, chr(i)) for i in range(dict_size))

    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

    print(result.getvalue())
```

Final result:



compression

```
E:\LZW_Compression\venv\Scripts\python.exe E:/LZW_Compression/compression.py
Compressed value
[84, 79, 66, 69, 79, 82, 78, 79, 84, 256, 258, 260, 265, 259, 261, 263, 268]
Decompressed Result
TOBEORNOTTOBEORTOBEORNOTTOBE

Process finished with exit code 0
```

5. Conclusion

Thus, the LZW algorithm was explained as a lossless compression method which can be used in PDF, GIF and images as well as other text files. Also, compression of a 28 lettered string to 17 itemed list was also observed by running the algorithm using python coding to better understand the working.

References

1. https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
2. https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/
3. https://rosettacode.org/wiki/LZW_compression
4. https://whatis.techtarget.com/definition/LZW-compression#:~:text=LZW%20compression%20is%20the%20compression,and%20the%20TIFF%20image%20format.