FACULTY OF ENGINEERING AND TECHNOLOGY DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

INFORMATION AND CODING THEORY

ENEE5304

Course Project

Limple-Ziv Code

Prepared by:

#Raneen Mahmoud #1181375

#Nemat Mimi#1181766

Section:

1

Instructor: Dr. **Wael Hashlamoun**

01/06/2022

Second Semester 2021/2022

Abstract

The aim of this project is to get familiar with the compression techniques especially with Limpel-Ziv, also to improve our knowledge in calculating a lot of things such as number of bits used in encoding paragraphs using ASCII code, average number of bits per symbol of the encoded paragraphs, and entropy.

Table of Contents

1. Introduction 1

2. Method 2

2.1 What is Limpel-Ziv 2

2.2 Advantages of the Limpel-Ziv algorithm 2

2.3 How does it work 3

3. Procedure and discussion: 4

3.1 The number of bits used to encode the paragraph if ASCII code is used: 4

3.2 Find the number of bits used to encode the paragraph if the Lempel-Ziv code is used. 4

3.3 Find the average number of bits/symbol of the encoded paragraph 5

3.4 Find the entropy of the source alphabet. 6

3.5 Find the percentage of compression accomplished by using the Lempel-Ziv encoding as compared to ASCII code. 6

Conclusion 7

References 8

4. Appendix: 9

Code: 9

input text: 10

List of Figures

Figure 1:Limpel-Ziv example [2] 3
Figure 2 3.1 code 4
Figure 3 3.1 Result 4
Figure 4 3.2 code 4
Figure 5 3.3 code 5
Figure 6 3.3 result 6
Figure 7 3.4 code 6
Figure 8 3.4 result 6
Figure 9 3.5 code 6
Figure 10 3.5 result 6

1. Introduction

Always the majority of data exhibits patterns which is subject to specific limitations. This applies to text as well as images, sound, and video, to mention a few. Consider the following text below which was taken from" Harry Potter and the Philosopher's Stone "book.

"Harry Potter has never even heard of Hogwarts when the letters start dropping on the doormat at number four, Privet Drive. Addressed in green ink on yellowish parchment with a purple seal, they are swiftly confiscated by his grisly aunt and uncle. Then, on Harry's eleventh birthday, a great beetle-eyed giant of a man called Rubeus Hagrid bursts in with some astonishing news: Harry Potter is a wizard, and he has a place at Hogwarts School of Witchcraft and Wizardry."

This text, like practically every other English work, contains letters that appear more frequently than others. For example, there are much more 'e' which is about 43 letters, and on other way there where 9 letters of 'w' in this paragraph. In addition to that we noticed that several substrings appear repeatedly in this text.

Data compression methods like Huffman, Lossless and Limpel-Ziv use such properties to make compressed data smaller than the original data. In this project, Limpel zif compression techniques in this project will be discussed and explained in detail and use it to compress a story to a minim number of bits, as well as the development of a Python software that aids in the compression of any given paragraphs.

2. Method

2.1 What is Limpel-Ziv

Abraham Lempel and Jacob Ziv created this technique, which was later published by Terry Welch in 1984. Unlike other compression techniques, the LZW algorithm is lossless, which means that no data is lost during file compression and decompression.
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. 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, The Idea relies on reoccurring patterns to save data space. 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".[1]

2.2 Advantages of the Limpel-Ziv algorithm

The Limpel-Ziv algorithm has the following advantages:

- The LZW algorithm is faster than the other algorithms.
- The algorithm is clear, basic, and effective.
- For files with a high volume of repeating data, the LZW method performs better.
- Another essential feature of the LZW compression technique is that it generates the encoding and decoding tables on the go, with no prior knowledge of the input.
- The LZW technique has a compression ratio of 60-70 percent for various text files.

2.3 <u>How does it work</u>

LZW compression works by reading a series of symbols, collecting them into strings, and then turning the strings to codes. We achieve compression because the codes take up less space than the strings they replace.

The following figure Gives an example that explains the algorithm in a simple way, so in short Identifying and storing sentences of the shortest length that have not yet appeared in a dictionary need to done, and after that when a new phrase is discovered, it is encoded as the union of the prior phrase and the new source. Each phrase begins with a previously occurring phrase (head) and ends with the new source output (tail).

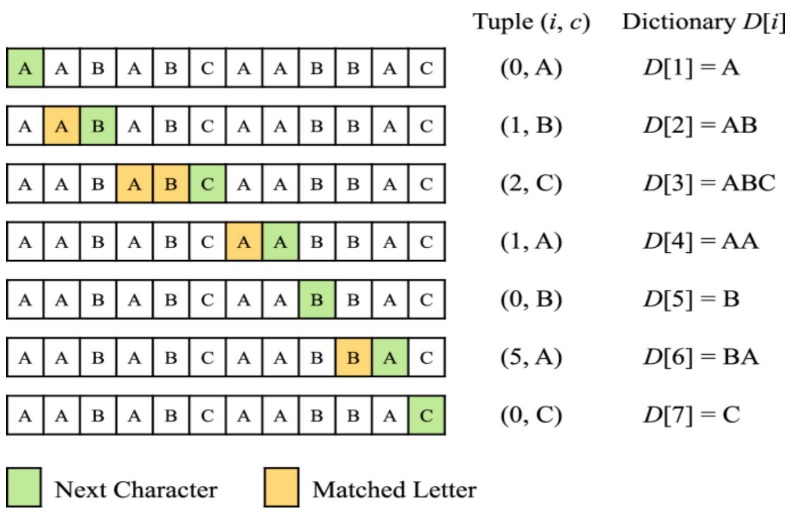


Figure 1:Limpel-Ziv example [2]

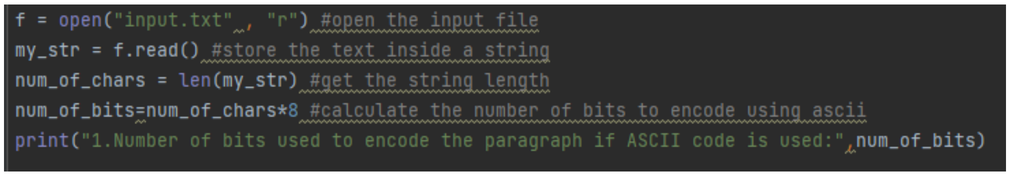
3. Procedure and discussion:

At this project, we were given a paragraph and asked to write a computer program that simulates the Lempel-Ziv code and do the following:

3.1 The number of bits used to encode the paragraph if ASCII code is used:

This requirement can be easily found by calculating the total number of the characters in the text and multiplying it by 8 since each character needs 8 bits to be represented.

The following code part shows the implementations.

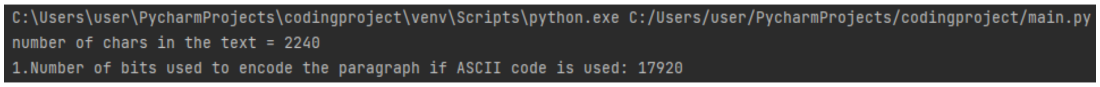


```
f = open("input.txt", "r") #open the input file
my_str = f.read() #store the text inside a string
num_of_chars = len(my_str) #get the string length
num_of_bits=num_of_chars*8 #calculate the number of bits to encode using ascii
print("1.Number of bits used to encode the paragraph if ASCII code is used:",num_of_bits)
```

Figure 2 3.1 code

From the above figure it can be noticed that the input file that contains the text was opened and its content was saved into a string, then the length of the string was got using the function len. Finally, the length was multiplied by 8.

The calculated value was printed as it is shown below:

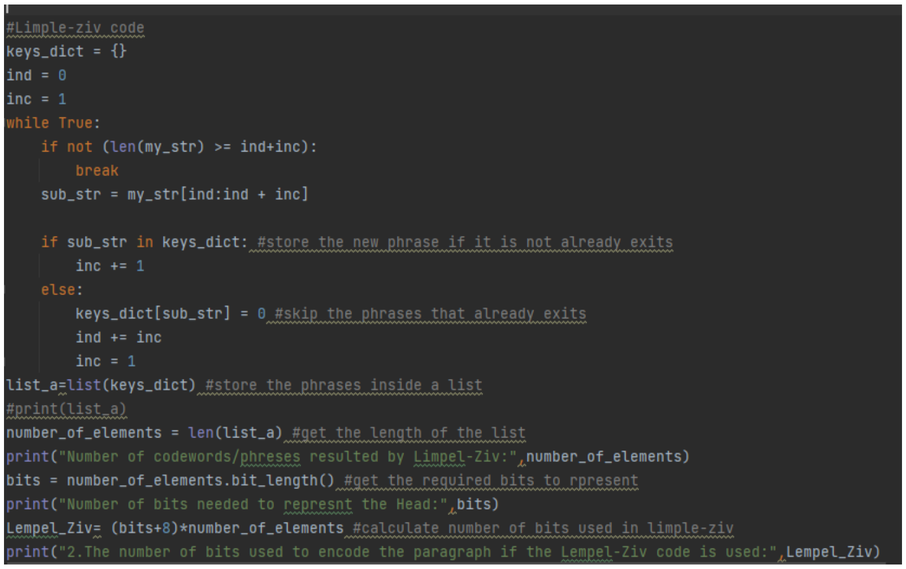


```
C:\Users\user\PycharmProjects\codingproject\venv\Scripts\python.exe C:/Users/user/PycharmProjects/codingproject/main.py
number of chars in the text = 2240
1.Number of bits used to encode the paragraph if ASCII code is used: 17920
```

Figure 3 3.1 Result

3.2 Find the number of bits used to encode the paragraph if the Lempel-Ziv code is used.

In order to find this requirement, the following code was implemented.



```
#Limple-ziv code
keys_dict = {}
ind = 0
inc = 1
while True:
    if not (len(my_str) >= ind+inc):
        break
    sub_str = my_str[ind:ind + inc]

    if sub_str in keys_dict: #store the new phrase if it is not already exits
        inc += 1
    else:
        keys_dict[sub_str] = 0 #skip the phrases that already exits
        ind += inc
        inc = 1
list_a=list(keys_dict) #store the phrases inside a list
#print(list_a)
number_of_elements = len(list_a) #get the length of the list
print("Number of codewords/phreses resulted by Limpel-Ziv:",number_of_elements)
bits = number_of_elements.bit_length() #get the required bits to rpresent
print("Number of bits needed to represnt the Head:",bits)
Lempel_Ziv= (bits+8)*number_of_elements #calculate number of bits used in limple-ziv
print("2.The number of bits used to encode the paragraph if the Lempel-Ziv code is used:",Lempel_Ziv)
```

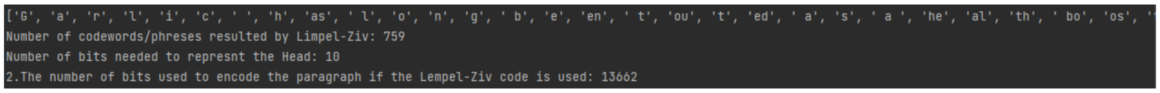
Figure 4 3.2 code

As it can be seen from the previous figure a loop was written in order to get the Limple-Ziv phrases this loop would check the characters and save it as a new phrase if it is tis first time or skip it until a new character appears.

Then, to find the number of bits that used in Limple-Ziv, the total number of codewords/phrases resulted by Limple-Ziv were calculated using len function and the required bits to represent the head was found using the bit_length function in order to implement the following equation.

Number of bits used by Limple-Ziv = (#of bits to represent the old phrase+ #of bits to represent the new phrase “which is 8 bits”) * total number of codewords.

After these implementations were made the following results were got.

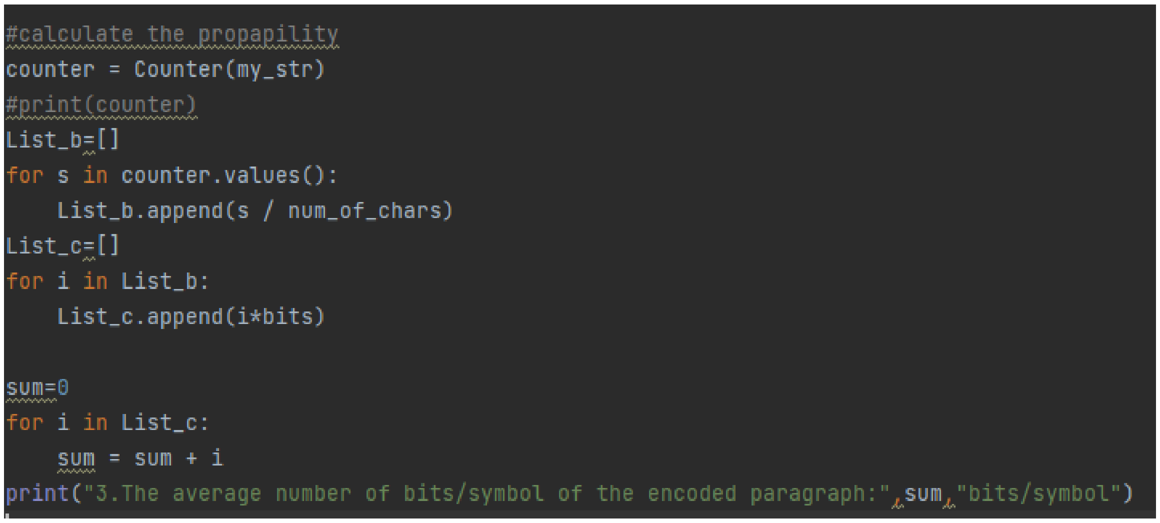


```
['G', 'a', 'r', 'l', 'i', 'c', ' ', 'h', 'as', ' l', 'o', 'n', 'g', ' b', 'e', 'en', ' t', 'ou', 't', 'ed', ' a', 's', ' a ', 'he', 'al', 'th', ' bo', 'os', '
Number of codewords/phreses resulted by Limpel-Ziv: 759
Number of bits needed to represnt the Head: 10
2.The number of bits used to encode the paragraph if the Lempel-Ziv code is used: 13662
```

Notice that how number od bits used in Limple-Ziv in much lower than those used in Ascii.

3.3 Find the average number of bits/symbol of the encoded paragraph

In order to find the average number of bits/symbol for the encoded paragraph the following code was written.



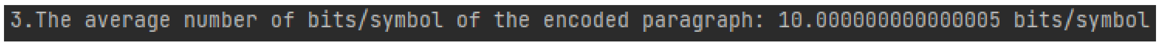
```
#calculate the propapility
counter = Counter(my_str)
#print(counter)
List_b=[]
for s in counter.values():
    List_b.append(s / num_of_chars)
List_c=[]
for i in List_b:
    List_c.append(i*bits)

sum=0
for i in List_c:
    sum = sum + i
print("3.The average number of bits/symbol of the encoded paragraph:",sum,"bits/symbol")
```

Figure 5 3.3 code

in order to implement the following equation:

Avg = **Σ**probability of char * needed bits to represent its codeword

So, at first, the probability of each character was found by calculating how many times it occurs in the paragraph and divide it by the total number of characters. Second, each character was multiplied with the needed bits to represent its codeword which was calculated in the previous part. Finally, the total sum was calculated and the following result was got.



```
3.The average number of bits/symbol of the encoded paragraph: 10.000000000000005 bits/symbol
```

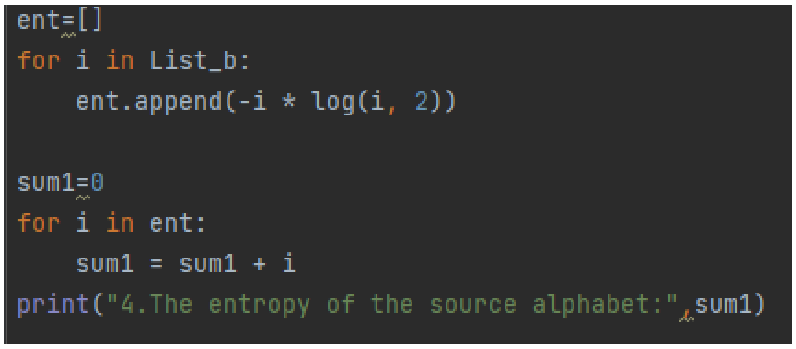
Figure 6 3.3 result

3.4 Find the entropy of the source alphabet.

Since the entropy is calculated by:

H = **Σ**- probability of char * $\log_2$ probability of char

The following code was implemented.

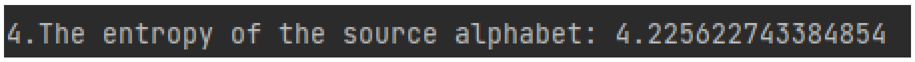


```
ent=[]
for i in List_b:
    ent.append(-i * log(i, 2))

sum1=0
for i in ent:
    sum1 = sum1 + i
print("4.The entropy of the source alphabet:",sum1)
```

Figure 7 3.4 code

The probability of each character was calculated in the previous part. So, here it was recalled as negative and multiplied by the logarithm of base 2 of it as it shown above.

The total summation was calculated and printed as following.

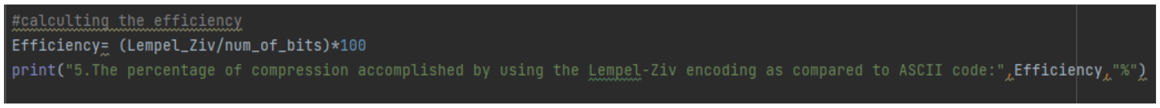


```
4.The entropy of the source alphabet: 4.225622743384854
```

Figure 8 3.4 result

3.5 Find the percentage of compression accomplished by using the Lempel-Ziv encoding as compared to ASCII code.

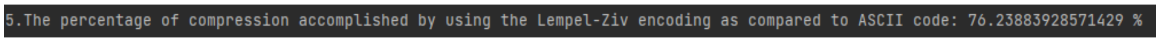
In order to find the efficiency, the used bits to encode the paragraph by using Ascii and Limple-Ziv were recalled, divided and multiplied by 100% as it shown in the following code part.



```
#calculting the efficiency
Efficiency= (Lempel_Ziv/num_of_bits)*100
print("5.The percentage of compression accomplished by using the Lempel-Ziv encoding as compared to ASCII code:",Efficiency,"%")
```

Figure 9 3.5 code

The final result was printed and the following number was got.



```
5.The percentage of compression accomplished by using the Lempel-Ziv encoding as compared to ASCII code: 76.23883928571429 %
```

Figure 10 3.5 result

Note: it can be seen that Limple-Ziv is more efficient then Ascii in encoding the paragraph by 76%.

Conclusion

All of the objectives of this project were achieved, first a good understanding of several compression techniques, particularly the Limpel-Ziv technique, was developed. In addition, a Python program was written to simulate the Lempel-Ziv code, and the number of bits required in encoding paragraphs using ASCII code, the average number of bits per symbol of the encoded paragraphs, and entropy were computed.

References

[1]https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/[accessed in 31 May 2022, 18:33]

[2]https://link.springer.com/article/10.1007/s11277-019-06979-7[accessed in 31 May 2022, 18:50]

4. Appendix:

Code:

```
from collections import Counter
from math import log
f = open("input.txt" , "r") #open the input file
my_str = f.read() #store the text inside a string
num_of_chars = len(my_str) #get the string length
print("number of chars in the text =",num_of_chars)
num_of_bits=num_of_chars*8 #calculate the number of bits to encode
using ascii
print("1.Number of bits used to encode the paragraph if ASCII code is
used:",num_of_bits)

#Limple-ziv code
keys_dict = {}
ind = 0
inc = 1
while True:
    if not (len(my_str) >= ind+inc):
        break
    sub_str = my_str[ind:ind + inc]

    if sub_str in keys_dict: #store the new phrase if it is not already
exits
        inc += 1
    else:
        keys_dict[sub_str] = 0 #skip the phrases that already exits
        ind += inc
        inc = 1
list_a=list(keys_dict) #store the phrases inside a list
#print(list_a)
number_of_elements = len(list_a) #get the length of the list
print("Number of codewords/phreses resulted by Limpel-
Ziv:",number_of_elements)
bits = number_of_elements.bit_length() #get the required bits to
rpresent
print("Number of bits needed to represnt the Head:",bits)
Lempel_Ziv= (bits+8)*number_of_elements #calculate number of bits used
in limple-ziv
print("2.The number of bits used to encode the paragraph if the Lempel-
Ziv code is used:",Lempel_Ziv)


#calculate the propapility
counter = Counter(my_str)
#print(counter)
List_b=[]
for s in counter.values():
    List_b.append(s / num_of_chars)
List_c=[]
for i in List_b:
    List_c.append(i*bits)

sum=0
for i in List_c:
    sum = sum + i
```

```
print("3.The average number of bits/symbol of the encoded
paragraph:",sum,"bits/symbol")

#calculting the entropy
ent=[]
for i in List_b:
    ent.append(-i * log(i, 2))

sum1=0
for i in ent:
    sum1 = sum1 + i
print("4.The entropy of the source alphabet:",sum1)


#calculting the efficiency
Efficiency= (Lempel_Ziv/num_of_bits)*100
print("5.The percentage of compression accomplished by using the
Lempel-Ziv encoding as compared to ASCII code:",Efficiency,"%")
```

input text:

```
Garlic has long been touted as a health booster but it has never been
clear why the herb might be good for you. Now new research is beginning
to unlock the secrets of the odoriferous bulb. In a study published
today in the proceedings of the national academy of sciences
researchers show that eating garlic appears to boost our natural supply
of hydrogen sulfide. Hydrogen sulfide is actually poisonous at high
concentrations. It is the same noxious byproduct of oil refining that
smells like rotten eggs. But the body makes its own supply of the stuff
which acts as an antioxidant and transmits cellular signals that relax
blood vessels and increase blood flow. In the latest study performed at
the university of alabama at birmingham researchers extracted juice
from supermarket garlic and added small amounts to human red blood
cells. The cells immediately began emitting hydrogen sulfide the
scientists found.  The power to boost hydrogen sulfide production may
help explain why a garlic rich diet appears to protect against various
cancers including breast prostate and colon cancer say the study
authors. Higher hydrogen sulfide might also protect the heart according
to other experts. Although garlic has not consistently been shown to
lower cholesterol levels researchers at albert einstein college of
medicine earlier this year found that injecting hydrogen sulfide into
mice almost completely prevented the damage to heart muscle caused by a
heart attack. People have known garlic was important and has health
benefits for centuries said doctor Kraus associate professor of
environmental science and biology at the university of alabama. Even
the greeks would feed garlic to their athletes before they competed in
the olympic games. Now the downside. The concentration of garlic
extract used in the latest study was equivalent to an adult eating
about two medium sized cloves per day. In such countries as italy korea
and china where a garlic rich diet seems to be protective against
disease per capita consumption is as high as eight to twelve cloves per
day. While that may sound like a lot of garlic doctor kraus noted that
increasing your consumption to five or more cloves a day is not hard if
you use it every time you cook.
```