FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING

ENEE5304 - INFORMATION AND CODING THEORY

Second semester 2024/2025

Course Project on Source Coding

Prepared by:

Sara Ewaida – 1203048

Section: 1

Instructor: Prof. Wael Hashlamoun

Date: 1st Jun 2025

Abstract

This report explores the application of Lempel-Ziv (LZ78) source coding to compress sequences of random symbols generated with non-uniform probabilities. The symbol set {a, b, c, d} is defined with respective probabilities of 0.5, 0.25, 0.2, and 0.05. After calculating the source entropy, sequences of varying lengths are generated and encoded using a custom implementation of the LZ78 algorithm. The report documents the resulting dictionaries, encoded outputs, and the total number of binary digits used for transmission. It further analyzes the compression performance in terms of bits per symbol and compression ratio, using standard ASCII coding as a benchmark. Results show that the compression improves with longer sequences, approaching the theoretical entropy limit, and demonstrate the practical benefits of dictionary-based encoding for symbol sequences with skewed distributions.

Table of Contents

Table of Figures

Table of Tables

1. Lempel-Ziv Encoding of Random Symbols

1.1. Introduction

We have 4 different symbols: a, b, c, d. Their probabilities are P(a) = 0.5, P(b) = 0.25, P(c) = 0.2, P(d) = 0.05. The task starts from generating multiple sequences with different lengths (20, 50, 100, 200, 400, 800, 1000, 5000) until encoding them using the Lempel-Ziv method. The first task of this part is to calculate the source entropy H. Then, generate five random sequences of length 20 symbols and find the average value of N_B, where N_B is the number of binary digits needed to encode the sequence. Then, the compression ratio relative to the ASCII code (average value of $N_B/(N*8)$) is found. Then, the size of the encoded sequence, compression ratio, and the number of bits per symbol are calculated for the rest of the lengths.

1.2. Theoretical Background

1.2.1. Source entropy

Entropy is a fundamental concept in information theory that quantifies the average amount of information produced by a stochastic source of data. For a discrete source emitting symbols with known probabilities, entropy H (S) is defined as

$$H(s) = -\sum_i p_i \cdot \log_2 p_i$$

where $p_{(x_i)}$ is the probability of symbol X_i. Entropy represents the theoretical lower bound on the average number of bits required to encode each symbol without loss. A higher entropy indicates greater uncertainty in the source and more bits needed per symbol, while lower entropy implies more predictability and better potential for compression. In the context of source coding, entropy sets the benchmark for how efficiently a coding algorithm like Huffman or Lempel-Ziv can compress data. No lossless compression scheme can achieve an average bit rate lower than the source entropy. [1]

1.2.1 Lempel-Ziv encoding

Brief Introduction

LZ78 is a lossless data compression algorithm introduced by Abraham Lempel and Jacob Ziv in 1978. It is part of the Lempel-Ziv family of dictionary-based encoders. Unlike fixed-length encoding schemes, LZ78 builds a dictionary of previously encountered substrings dynamically during the encoding process. Each new sequence found in the input is added to the dictionary with a unique index, allowing repeated sequences to be represented compactly in the encoded output. [2]

How LZ78 Works

1. The algorithm starts with an empty dictionary.
2. Then, the algorithm reads the sequence character by character.
3. For each character or sequence, it checks whether it already exists in the dictionary.
4. If the character or sequence is not in the dictionary, the algorithm adds it as a new entry and outputs a pair consisting of the index of the longest matching prefix (or 0 if none) and the next unmatched character.
5. If the character or sequence is already in the dictionary, the algorithm continues reading the next character to extend the match.
6. The algorithm repeats steps 2–5 until it reaches the end of the sequence.

The following figure is a detailed example of how Lempel-Ziv works:

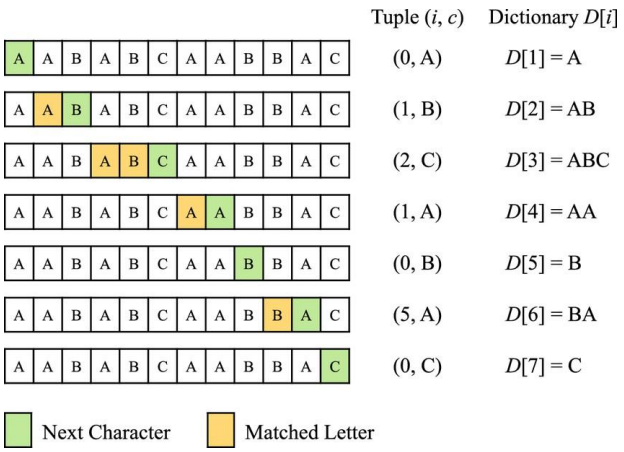

Figure 1: how Lempel-Ziv works [3]

1.3. Results and Analysis

1.3.1. Part1: Source entropy

This part calculates the source entropy H in bits, the code of this part is in appendix.

$$H(S) = -\sum_i p_i \cdot log_2 p_i$$

$$H(S) = -((0.5 \cdot log_2 0.5) + (0.25 \cdot log_2 0.25) + (0.2 \cdot log_2 0.2) + (0.05 \cdot log_2 0.05))$$

$$H(S) = 1.6804820237218405$$

```
C:\Users\Asus\Desktop\CodingProject\CodingProject\
Entropy: 1.6804820237218405 bits/symbol
----------------------------------------
```

Figure 2: source entropy

1.3.2. Part2: Lempel-Ziv Parsing and Encoding Table for a 20-Symbol Random Sequence

This part generates a single random sequence of 20 symbols using predefined symbol probabilities. The program parses the sequence using the Lempel-Ziv (LZ78) algorithm and constructs the dictionary of unique phrases along with their corresponding encoded packets and binary transmitted codes. The output also includes the total number of binary digits needed to encode the sequence (NB), the number of bits per phrase, bits per symbol, and the compression ratio relative to 8-bit ASCII encoding.

```
------
Generated sequence: aaabbbacabcacadcaabb
Encoded sequence: 0011000011011000010011000101101100010101100011101100010001100011101011000010011001001110110000111001100010

--------------------------------------------------------------------------------
Phrase Address   Dictionary Content   Encoded Packet     Transmitted Code
--------------------------------------------------------------------------------
1                a                   (0, 'a')           001100001
2                aa                  (1, 'a')           101100001
3                b                   (0, 'b')           001100010
4                bb                  (3, 'b')           1101100010
5                ac                  (1, 'c')           101100011
6                ab                  (1, 'b')           101100010
7                c                   (0, 'c')           001100011
8                aca                 (5, 'a')           10101100001
9                d                   (0, 'd')           001100100
10               ca                  (7, 'a')           11101100001
11               abb                 (6, 'b')           11001100010

----------------------------------------
# of phrases = 11
Bits per phrase = 12 → [4 + 8]
NB = 132 → [12 * 11]
Bits per symbol = 6.6000
Compression ratio = 0.8250
----------------------------------------

Process finished with exit code 0
```

Figure 3: LZ78 Encoding Output (N = 20)

Table 1 LZ78 Encoding Table for a Random Sequence of 20 Symbols:

Phrase Address	Dictionary Content	Encoded Packet	Transmitted Code
1	a	(0, 'a')	001100001
2	aa	(1, 'a')	101100001
3	b	(0, 'b')	001100010
4	bb	(3, 'b')	1101100010
5	ac	(1, 'c')	101100011
6	ab	(1, 'b')	101100010
7	c	(0, 'c')	001100011
8	aca	(5, 'a')	10101100001
9	d	(0, 'd')	00110100
10	ca	(7, 'a')	11101100001
11	abb	(6, 'b')	11001100010

1.3.3. Part3: Generating sequences with different number of symbols

In this part, the objective is to observe how the efficiency of compression changes as the length of the input sequence increases. The program is executed for different values of N (20, 50, 100, 200, 400, 800, 1000, 5000). For each sequence, the size of the encoded output (N_B), the compression ratio ($N_B/8N$), and the average number of bits per symbol (N_B/N) are calculated and recorded. This allows us to analyze the behavior of the Lempel-Ziv encoding method as the input grows.

```
C:\Users\Asus\Desktop\CodingProject\CodingProject\.v
N       NB      Compression Ratio   Bits per Symbol
-------------------------------------------------------
20      120     75.0%               6.0
50      286     71.5%               5.72
100     518     64.75%              5.18
200     854     53.37%              4.27
400     1620    50.62%              4.05
800     3056    47.75%              3.82
1000    3728    46.6%               3.728
5000    16416   41.04%              3.2832

Process finished with exit code 0
```

Figure 4: comparison table of encoding sequences with different lengths

Table 2: comparison table of encoding sequences with different lengths

Sequence length N	Size of encoded sequence (NB)	Compression ratio NB /(8*N)	Compression ratio NB /(8*N)
20	120	75%	6.0
50	286	71.5%	5.72
100	518	64.75%	5.18
200	854	53.37%	4.27
400	1620	50.62%	4.05
800	3056	47.75%	3.82
1000	3728	46.6%	3.728
5000	16416	41.04%	3.2832

3. Conclusion

In this project, we explored the application of Lempel-Ziv (LZ78) encoding for compressing randomly generated symbol sequences with predefined probability distributions. By generating sequences of varying lengths and applying LZ78 compression, we were able to observe how the structure and frequency of symbols influence the number of bits required for encoding. The results demonstrated that as the sequence length increases, the compression ratio improves and the average number of bits per symbol decreases, highlighting the efficiency of LZ78 in handling repetitive patterns. This project emphasizes the relationship between source entropy, phrase generation, and binary encoding efficiency, reinforcing the value of dictionary-based compression in real-world scenarios where symbol probabilities are skewed. Through this study, we gained a deeper understanding of how Lempel-Ziv operates and how data characteristics impact compression performance.

4. References

[1] ScienceDirect. (n.d.). *Source Entropy – an overview*. Retrieved May 10, 2025, from https://www.sciencedirect.com/topics/engineering/source-entropy

[2] Saquib, M. (n.d.). *Unit 31: Lempel-Ziv Compression (LZ78)*. King Fahd University of Petroleum and Minerals. Retrieved May 10, 2025, from https://faculty.kfupm.edu.sa/ics/saquib/ICS202/Unit31_LZ78.pdf

[3] Rao, K. P., & Rani, B. U. (2019). *A Modified Lempel–Ziv–Welch (LZW) Lossless Compression Technique for Wireless Sensor Networks*. *Wireless Personal Communications*, Springer. Retrieved May 10, 2025, from https://link.springer.com/article/10.1007/s11277-019-06979-7

5. Appendix

5.1. Entropy code

```python
import math
1 usage
def calculate_entropy(probability):
    return -sum([prob * math.log2(prob) for prob in probability])
probabilities = [0.5, 0.25, 0.2, 0.05]
entropy = calculate_entropy(probabilities)
entropy_str = "Entropy: " + str(entropy) + " bits/symbol"
print(entropy_str)
print('-' * len(entropy_str) + '\n')
```

5.2. LimpelZiv code

```python
import random
import math

#Generate N = 20 symbol sequence
symbols = ['a', 'b', 'c', 'd']
probabilities = [0.5, 0.25, 0.2, 0.05]
N = 20
sequence = ''.join(random.choices(symbols, probabilities, k=N))
print(f"\nN = {N}")
print("------")
print(f"Generated sequence: {sequence}")

#LZ78 parsing
dictionary = {}
phrases = []
encoded_packets = []
transmitted_codes = []
binary_addresses = []

current = ""
index = 1

```

```
for char in sequence:
    current += char
    if current not in dictionary:
        dictionary[current] = index
        if len(current) == 1:
            packet = (0, current)
            address_bin = '0'
        else:
            packet = (dictionary[current[:-1]], current[-1])
            address_bin = format(packet[0], 'b')  # binary of index
        phrases.append(current)
        encoded_packets.append(packet)
        binary_addresses.append(format(index, '03b'))  # binary of phrase address
        char_bin = format(ord(packet[1]), '08b')       # 8-bit ASCII for character
        transmitted_codes.append(address_bin + char_bin)
        index += 1
        current = ""

print(f"Encoded sequence: {''.join(transmitted_codes)}")

#Calculate NB and bits per phrase
num_phrases = len(phrases)
bits_for_index = math.ceil(math.log2(num_phrases)) if num_phrases > 1 else 1
bits_per_phrase = bits_for_index + 8
NB = bits_per_phrase * num_phrases

#Compression Ratio
compression_ratio = NB / (N * 8)
bits_per_symbol = NB / N
```

```

print("\n" + "-" * 85)
print(f"{'Phrase Address':<16} {'Dictionary Content':<20} {'Encoded Packet':<18} {'Transmitted Code'}")
print("-" * 85)
for i in range(num_phrases):
    print(f"{i+1:<16} {phrases[i]:<20} {str(encoded_packets[i]):<18} {transmitted_codes[i]}")

print("\n" + "-" * 40)
print(f"# of phrases = {num_phrases}")
print(f"Bits per phrase = {bits_per_phrase} → [{bits_for_index} + 8]")
print(f"NB = {NB} → [{bits_per_phrase} * {num_phrases}]")
print(f"Bits per symbol = {bits_per_symbol:.4f}")
print(f"Compression ratio = {compression_ratio:.4f}")
print("-" * 40)
```

5.3. Generating sequences with different number of symbols

```python
import random
import math
import pandas as pd
symbols = ['a', 'b', 'c', 'd']
probabilities = [0.5, 0.25, 0.2, 0.05]
sequence_lengths = [20, 50, 100, 200, 400, 800, 1000, 5000]
results = []
for N in sequence_lengths:
    sequence = ''.join(random.choices(symbols, probabilities, k=N))
    dictionary = {}
    phrases = []
    transmitted_codes = []
    current = ""
    index = 1
    for char in sequence:
        current += char
        if current not in dictionary:
            dictionary[current] = index
            if len(current) == 1:
                packet = (0, char)
                address_bin = '0'
            else:
                packet = (dictionary[current[:-1]], char)
                address_bin = format(packet[0], 'b')
            char_bin = format(ord(packet[1]), '08b')
            transmitted_codes.append(address_bin + char_bin)
            phrases.append(current)
            index += 1
            current = ""
```

```python
    # NB and ratios
    num_phrases = len(phrases)
    bits_for_index = math.ceil(math.log2(num_phrases)) if num_phrases > 1 else 1
    bits_per_phrase = bits_for_index + 8
    NB = bits_per_phrase * num_phrases
    compression_ratio_percent = (NB / (N * 8)) * 100
    bits_per_symbol = NB / N

    results.append({
        "N": N,
        "NB": NB,
        "Compression Ratio": f"{round(compression_ratio_percent, 2)}%",
        "Bits per Symbol": round(bits_per_symbol, 4)
    })

df = pd.DataFrame(results)
print(f"{'N':<8}{'NB':<8}{'Compression Ratio':<20}{'Bits per Symbol':<20}")
print("-" * 70)
for row in results:
    print(f"{row['N']:<8}{row['NB']:<8}{row['Compression Ratio']:<20}{row['Bits per Symbol']:<20}")
```