Lempel-Ziv-Welch (LZW) Compression

Data Structures and Algorithms

Ho Thuy Tram, Thai Gia Huy

University of Science - VNU HCMC

Basic introduction to data compression

What is Data Compression, at first?

- A process of reducing the amount of data "represented by bits" needed for the storage or transmission.
- Important in storing information digitally on computer disks and in transmitting it over communication channels.
- Since the size of data increases over time, compression methods enable efficient data transfer at lower bandwidth by utilizing payload.

1. Introduction
   - Basic introduction to Data Compression
   - Historical Background of LZW Compression
   - Applications of LZW Compression
2. Understanding the Algorithm
3. Performance Analysis
4. Overview of LZW-Related Algorithms
5. Conclusion

Historical Background

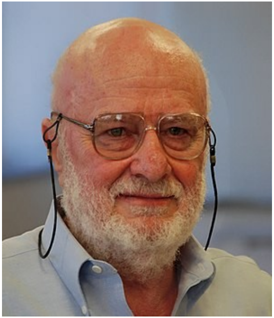


Figure 1: Abraham Lempel

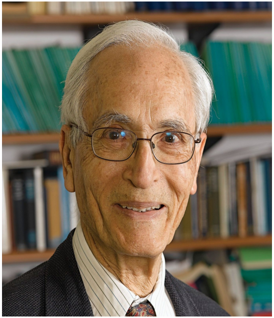


Figure 2: Jacob Ziv

- Upon the pioneering work of Abraham Lempel and Jacob Ziv on LZ7, LZ78, published in *IEEE Transactions on Information Theory*.

Historical Background

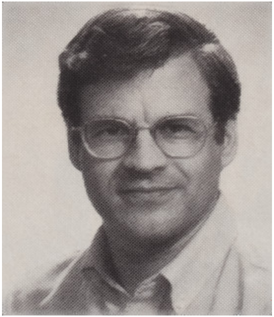


Figure 3: Abraham Lempel

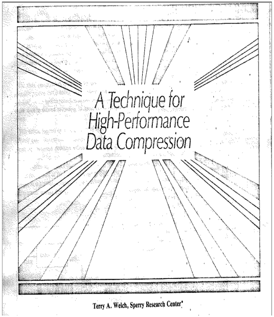
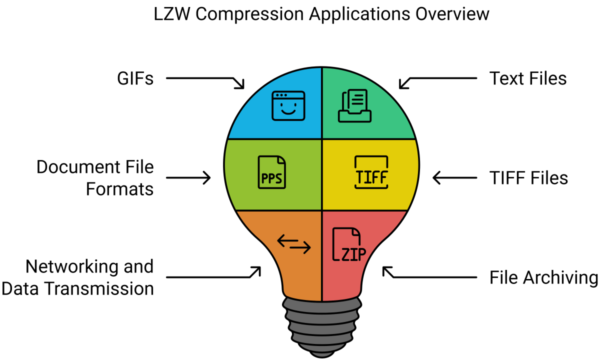


Figure 4: Lempel-Ziv-Welch paper

- Terry Welch further refined the algorithm to improve its efficiency and adaptiveness, leading to his 1984 article, "A Technique for High-Performance Data Compression", published in *Computer*.

1. Introduction
   - Basic introduction to Data Compression
   - Historical Background of LZW Compression
   - Applications of LZW Compression
2. Understanding the Algorithm
3. Performance Analysis
4. Overview of LZW-Related Algorithms
5. Conclusion

Applications



1. Introduction
2. Understanding the Algorithm
   - Key Concepts and Mechanisms
   - Compression Phase with Examples
   - Decompression Phase with Examples
3. Performance Analysis
4. Overview of LZW-Related Algorithms
5. Conclusion

Key Concepts and Mechanisms (1)

Lossless Dictionary-based method

- To "memorize" the substrings of characters that occurred before in the text.
- Uses the indices of the place in the dictionary where the substring is stored.

"Greedy" parsing algorithm

- The text is looped over exactly once, during this parsing, the longest recognized substring is saved to the encoded results.
- "The current substring + the next character" is added to the dictionary.

Key Concepts and Mechanisms (2)

- No dictionary transmission which reduces transmission overhead. The dynamic dictionary is created by both compressor and decompressor with a small dictionary.
- Relies heavily on the nature of the input data. Does not attempt to optimally select strings by making use of probability estimation.
- Therefore, its effectiveness is less than optimal but creates great usability by the simplicity of the algorithm.

Compression Phase

Algorithm 1 LZW Compression Process

Initialize an empty dictionary.
Populate the emptied dictionary with all possible one-length characters from the extended ASCII set (values 0 to 255).
Initialize an empty string **P**.
while not at the end of the character stream **do**
    C = next character.
    if P + C exists in the dictionary **then**
        P = **P** + **C** (Extend by adding C to the string P).
    else
        Output the code for **P**.
        Append the string **P** + **C** to the dictionary.
        P = **C** (Replace the string P with the current character).
    end if
end while

Compression Process with Step by Step

Step	Encoder Output		Dictionary		Explaination
	Output	Represents	Key	Value	
1					P = "", C = "A", assign P+C = "A" to P
2	193	"A"	"AB"	256	P = "A", C = "B", assign C = "B" to P
3	194	"B"	"BC"	257	P = "B", C = "C", assign C = "C" to P
4	195	"C"	"CB"	258	P = "C", C = "B", assign C = "B" to P
5					P = "B", C = "C", assign P+C = "BC" to P
6	257	"BC"	"BCC"	259	P = "BC", C = "C", assign C = "C" to P
7	195	"C"	"CA"	260	P = "C", C = "A", assign C = "A" to P
8					P = "A", C = "B", assign P+C = "AB" to P
9	256	"AB"			P = "AB", C = ""
					end of stream

1. Introduction
2. **Understanding the Algorithm**
   - Key Concepts and Mechanisms
   - Compression Phase with Examples
   - **Decompression Phase with Examples**
3. Performance Analysis
4. Overview of LZW-Related Algorithms
5. Conclusion

Decompression Phase

Algorithm 2 LZW Decompression Process

```
Initialize the dictionary with all possible single-length characters.
OLD = first input code.
Output translation of OLD (decoded sequence).
while not at the end of the encoded sequence do
    NEW = next input code.
    if NEW exists in the dictionary then
        S = translation of NEW.
    else
        S = translation of OLD + C.
    end if
    Output S.
    C = first character of S.
    Append translation of OLD + C to the dictionary.
    OLD = NEW.
end while
```

Deompression Process with Step by Step

Step	Decoder Output	Dictionary		Explaination
		Key	Value	
1	"A"			OLD = 193
2	"B"	256	"AB"	NEW = 194, S = "B", C = "B", OLD = 194
3	"C"	257	"BC"	NEW = 195, S = "C", C = "C", OLD = 195
4	"BC"	258	"CB"	NEW = 257, S = "BC", C = "B", OLD = 257
5	"C"	259	"BCC"	NEW = 195, S = "C", C = "C", OLD = 195
6	"AB"	260	"CA"	NEW = 256, S = "AB", C = "A", OLD = 256
				end of stream

Compression Ratios

Data Type	Compression Ratio
English Text	1.8
Cobol Files	2 to 6
Floating Point Arrays	1.0
Formatted Scientific Data	2.1
System Log Data	2.6
Program Source Code	2.3
Object Code	1.5

Best Case for English Text

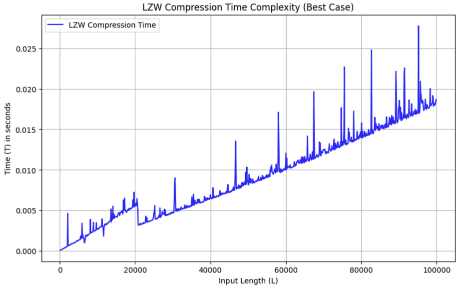


Figure 5: LZW Compression Runtime in Best Case

Best Case for English Text

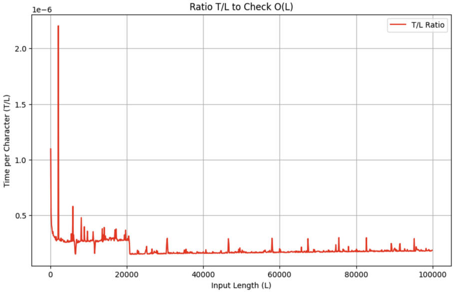


Figure 6: Best Case: $T = \mathcal{O}(L)$

Worst Case for English Text

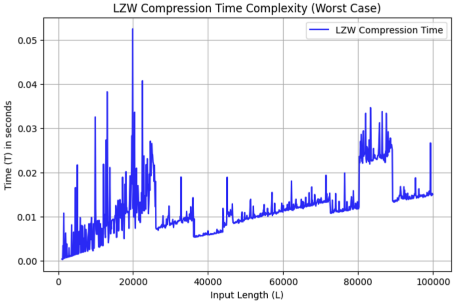


Figure 7: LZW Compression Runtime in Worst Case

Worst Case for English Text

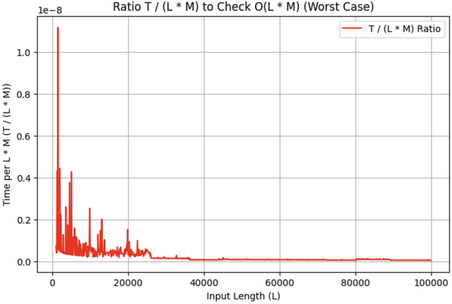


Figure 8: Best Case: $T = \mathcal{O}(L \cdot M)$

1. Introduction
2. Understanding the Algorithm
3. Performance Analysis
4. Overview of LZW-Related Algorithms
   - LZ77
   - LZ78
   - Comparison between LZ77, LZ78, and LZW
5. Conclusion

LZ77

Concept: Uses a *sliding window* to find matches between the lookahead buffer and previous data.

Compression Output: A triplet (offset, length, symbol).

Strengths	**Weaknesses**
Efficient for localized patterns	Less efficient for globally repetitive data
Simple implementation	Limited range due to sliding window

1. Introduction
2. Understanding the Algorithm
3. Performance Analysis
4. Overview of LZW-Related Algorithms
    - LZ77
    - LZ78
    - Comparison between LZ77, LZ78, and LZW
5. Conclusion

LZ78

Concept: Dynamically builds a dictionary to store symbol sequences.

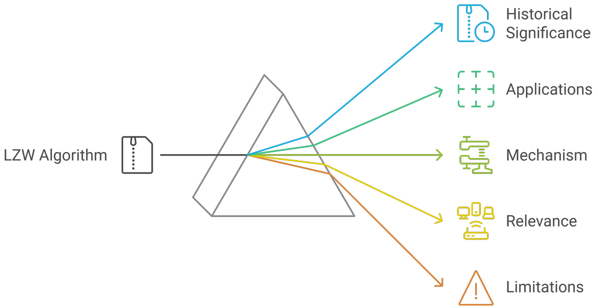
Compression Output: A pair $(\text{index}, \text{symbol})$.

Strengths	**Weaknesses**
Captures patterns over longer ranges	Includes raw symbols in the output
Does not require a fixed window size	May create a large dictionary, increasing memory usage

Comparison between LZ77, LZ78, and LZW

Feature	**LZ77**	**LZ78**	**LZW**
Dictionary Type	Sliding Window	Dynamic Dictionary	Pre-initialized Dictionary
Output	(offset, length, symbol)	(index, symbol)	(index)
Strengths	Local Patterns	Long-Range Patterns	High Compression Ratios
Weaknesses	Limited Range	Includes Raw Symbols	Complex Initialization
Applications	ZIP, PNG	Basis for LZW	GIF, Legacy Systems

Conclusion


LZW Algorithm
Historical Significance
Applications
Mechanism
Relevance
Limitations