Data Compression: Lempel-Ziv Algorithm

I. Huffman encoding is not efficient enough – it encodes only fixed units of the alphabet: like 26 letters.

II. But if we want to compress an English text, many words or even long phrases occur with high frequency, like “the”, “and”, “of the” ... or fraction of a word “ion”, “ing” ...

III. If we also encode such highly frequent subsequences, we can probably compress more.

IV. Lempel-Ziv algorithm tries to do this. It is the most important practical data compression algorithm.

The Lempel-Ziv Algorithm

- Let w be the text to be encoded.

- The algorithm keeps a "dictionary" of encoded words that are dynamically chosen from w.

- With each character sequence s (subsequence of w), the dictionary assigns it a **code**, $\#(s)$. All codes have the same length, typically 12 bits (encoding up to $2^{12} = 4096$ sequences).

- The dictionary is initialized with the codes for all characters in the alphabet Σ (assigned with codes $0, \ldots, |\Sigma| - 1$).

- At each step:

- Determine the longest prefix of p of w that is in the dictionary, p is called the **current match**.

- Output the code of p (by the dictionary); delete p from w.

- Add to the dictionary the current match of the *previous* step (for decoding) concatenated with the first character in w after that, assigning it the next unused code.

- After we have encoded the whole text w, the dictionary is *thrown away*.

An Example.

Consider the text $w =$ COCOA AND BANANAS.

Step	Output	Add to Dictionary
1	#(C)	–
2	#(O)	CO
3	#(CO)	OC
4	#(A)	COA
5	#(□)	A□
6	#(A)	□A
7	#(N)	AN
8	#(D)	ND
9	#(□)	D□
10	#(B)	□B
11	#(AN)	BA
12	#(AN)	ANA
13	#(A)	ANA
14	#(S)	AS

Decoding Algorithm

Since we do not save the dictionary (except for the initial dictionary, to save space), the decoding process mimics the encoding process to re-construct the dictionary dynamically. In above example:

- It first reads code for C, this is in the initial dictionary.

- Then it reads the code for O.

- Mimic the encoder, add CO to the dictionary with next unused code (which is the same code the encoding has chosen).

- Then read the code #(CO) and look up the dictionary and decode to CO, and add OC to the dictionary with the next unused code.

- And so on.

What if the dictionary is full?

- Stop placing new entries. Just use the dictionary created so far.

- Clear dictionary (except for 1 character sequences) and start afresh.

- Discard infrequently used sequences and re-use their codes.

- Switch to longer codes — adding single bits doubles the number of available codes.

An experiment on the Lewis-Denenberg book:

- Before compression, the book has 1322028 characters (8 bits each) = 10576224 bits.
- 2 pass Huffman Algorithm compresses the book to 6469752 bits, 61% of original.
- A variant of Lempel-Ziv with 12-bit codes allowing code size to grow compresses the book to 4493168, 42.5% of original.