Lempel-Ziv Lossless Compression

• Developed by Jacob Ziv of AT&T Bell Labs/Technion-Israel and Abraham Lempel of IBM in 1978.

• LZ algorithm does not require any knowledge of the source probabilities. The algorithm is inherently adaptive in that it handles sources with any probability law.

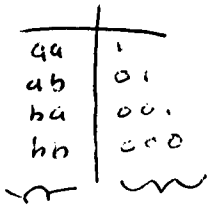
• It is possible to prove the algorithm achieves entropy H(X) for any stationary source and hence it is optimum. No proof is given here.

• LZ is the basis of virtually all commercially available file compression packages (compress, Stacker, PK Zip, stuff-it, etc).

• There are hundreds of variations of the LZ algorithm. The version presented is the simplest to implement, and in some instances may not be as good as those used in commercially available software.

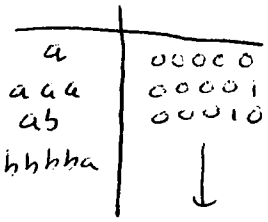
Basic Idea of LZ algorithm: Use simple adaptation rules to construct the table of source outputs to encode.

Huffman code:



- We're given the source output strings to encode and we must design a code using their known probabilities.
- Using the table at left, we encode 2 consecutive source outputs.
- The encoding table is selected ahead of time and is used to encode the source for all time.
- Huffman code is a *fixed-to-variable length code.*

LZ code:



- Probability law is unknown and table is constructed as the encoding progresses.
- Table grows to some fixed length. Usually something like 2^6 - 2^{20} entries.
- The number of bits in each codeword are fixed ahead of time.
- LZ code is a *variable-to-fixed length code.*

The LZ algorithm: 2 steps

1) Given a source output $\underline{x} = (X_1, ..., X_N)$ to encode, the encoder looks for the longest sourceword in the table that prefixes $\underline{x}$, say $\underline{w}_i$ and sends the codeword for it.

2) Table is updated with a new source word $\underline{w}_i$ + next source output (this gets added to the table.

Example: algorithm best shown by example. Suppose that we must encode the following source output: $\underline{x}$ = a a a b a a b a a b a a b b a

Time 0: start with 2 table entries, a and b . Codewords 1 and 2 are L bits long

Encoding table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)

Time 1: Begin encoding.

1)Look for the longest sourceword in the table that prefixes $\underline{x}$. It's 'a'. Send codeword for 'a' (codeword 1)

1
x = a a a b a a b a a b a a b b a

2) Add 'a+next source output' to list. Therefore add 'aa' to table.

Encoding table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3

Time 2: Continue encoding.

1)Look for the longest sourceword in the table that prefixes $\underline{x}$. It's 'aa'. Send codeword for 'aa' (3).

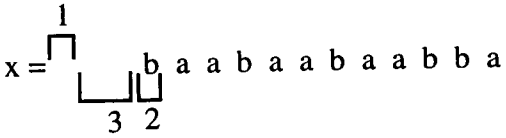
1 3
x = a a b a a b a a b a a b b a

2) Add 'aa+next source output' to list (add 'aab' to table).

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aab	4

Time 3: Continue encoding.

1)Look for the longest sourceword in the table that prefixes x. It's 'b'. Send codeword for 'b' (2).

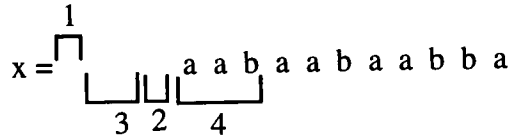


2) Add 'b+next source output' to list (add 'ba' to table).

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aab	4
ba	5

Time 4: Continue encoding.

1)Look for the longest sourceword in the table that prefixes x. It's 'aab'. Send codeword for 'aab' (4).

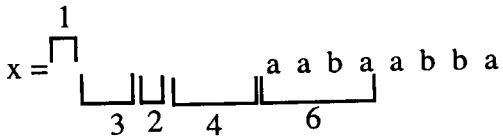


2) Add 'aab+next source output' to list. Therefore add 'aaba' to table.

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aab	4
ba	5
aaba	6

Time 5: Continue encoding.

1)Look for the longest sourceword in the table that prefixes x. It's 'aaba'. Send codeword for 'aaba' (6).



2) Add 'aaba+next source output' to list. Therefore add 'aabaa' to table.

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aab	4
ba	5
aaba	6
aabaa	7

Continue on until data file is exhausted.

Constructing the decoder. The decoder must be able to accept the fixed length codewords produced by the encoder and construct the original sequence of a's and b's. To do this it must also reconstruct the original encoding table on the fly.

In our previous example, the decoder receives: 1 3 2 4 6 (each number represents a different codeword of length L). The decoder must produce the original sequence of a's and b's and construct the original encoding table.

Time 0: Decoder starts with same table that encoder started with

Decoder Table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)

Time 1: Decoder receives the codeword 1. Therefore it decodes it as an 'a'. What does it add to the list? It must add 'a_' to the list. This is what the encoder did, except decoder does not know what's coming next and cannot fill in the blank (yet). Decoder defers the decision unitl the next time

Decoder Table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aa_	4

Time 2: Decoder receives codeword 3, which cannot be decoded yet. But since it received a 3 rather than a 2, it knows the blank must be filled with an 'a'. Thus it adds 'aa' to the list. Encoder added 'aa_' to the list when it encoded during time 2. Decoder defers until next symbol to decide on 'aa_'

Decoder Table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aa_	4

Time 3: Decoder receives a 2, which is decoded as a 'b'. Knowing this the decoder can fill in the blank of 'aa_' from time 2 to 'aab'. Encoder added 'b_' to the list when it encoded during time 3. Decoder defers until next symbol to decide on 'b_'

Decoder Table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aab	4
ba	5
aab_	6

Time 4: Decoder receives a 4, which is decoded as a 'aab'. Knowing this the decoder can fill in the blank of 'aab_' from time 3 to 'aaba'. Encoder added 'aab_' to the list when it encoded during time 4. Decoder defers until next symbol to decide on 'aab_'

Decoder Table

Sourceword	codeword
a	1 = (000000...0)
b	2 = (000000...1)
aa	3
aab	4
ba	5
aab_	6

And so on ...