A Lecture on the Lempel-Ziv Compression Method

Jacob Bettencourt

March 30, 2018

This is the augmented transcript of a lecture given by Luc Devroye on the 15th of March 2018 for a Data Structures and Algorithms class (COMP 252) at McGill University. The subject was the Lempel-Ziv method of compression.

Lempel-Ziv Method

Definition 1. The **Lempel-Ziv (LZ) Method** is a method of compression that can be used on any alphabet of any size.

Introduced in 1977 by Abraham Lempel and Jacob Ziv in their paper "A Universal Algorithm for Sequential Data Compression," the LZ algorithm self-adjusts so it can be used on a variety of input data using a digital search tree data structure, described below[1]. To apply the method we parse each input symbol into the smallest unseen sequence of symbols, as in the following example:

Example 2. Given alphabet: a, b, c.
Input:

abaabcaaabbcaaaa

Parsed input:

_	a	b	aa	bc	aaa	bb	c	aaaa
0	1	2	3	4	5	6	7	8

The parsed input consists of "pieces", numbered consecutively, starting with 1. Piece 0 corresponds to the empty set. Observe that each piece consists of a previous piece and precicely one new symbol. We can replace it by an integer (the number of that embolded piece) followed by a symbol, as shown below:

_	0a	0b	1a	2c	3a	2b	0c	5a

Next, we construct the bit sequence in the output. The number of bits needed for the coded output seqence is the minimal number of bits needed to code the integer pointing to the prefix of the current piece plus the number of bits required to code one symbol, B[1], as seen below:

_	0a	0b	1a	2c	3a	2b	0c	5a
0	0+B	1+B	2+B	2+B	3+B	3+B	3+B	3+B

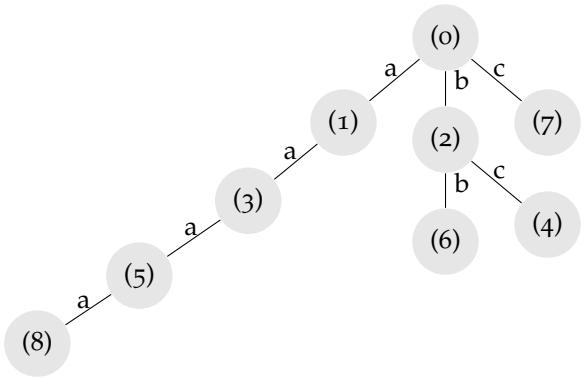
[1] B is the number of bits required to encode a symbol from an alphabet of size k:

$$B = \lceil \log_2 k \rceil$$

In the example above, piece number 8, "aaaa", is to be coded as "5a", which in turn causes 3 bits to be reserved for the "5" (as that integer ranges from 0 to 7) and B bits for "a" according to a fixed-width code for symbols.

Digital Search Tree (DST)

Definition 3. A digital search tree is a k-ary tree where k is the size of the input alphabet, and each edge corresponds to one of k symbols.



The digital search tree for the Lempel-Ziv code associates one ”piece” with each node, starting with the piece ”0” as the root. The edge values seen on the path from the root to piece ”i” describe the symbols of piece ”i”. Thus, if piece ”i” is the concatenation of piece ”j” and symbol S, then j is the parent of i in the DST, and the edge value of (i, j) is S.

Coder

The coder creates the above digital search tree while parsing the input string by processing each symbol to trace the correct path in the DST, until a nil is reached or a new branch is necessary, in which case the coder inserts the corresponding node in the tree. Since each input symbol causes one step in the coder it is clear that this takes $O(n)$ time, where n is the number of input symbols. After the coder has finished encoding it can discard the DST, as the compressed information is contained entirely in the output sequence, described above. Equivalently, one can think of the encoded sequence as the DST itself.

Decoder

The decoder for the Lempel-Ziv method observes that the coded data is the digital search tree with parent pointers, and so an array implementation can be applied to reconstruct the data:

pointer	parent	edge value
1	0	a
2	0	b
3	1	a
4	2	c
5	3	a
6	2	b
7	0	c
8	5	a

The time taken to decode the compressed information from the LZ method is $O(n)$ since, as with the coder, each symbol is decompressed one at a time by reverse-engineering the DST as an array. Each array index has a pointer to its parent, which allows the decoder to recreate the original information in $O(n)$ time.

Remark: Assume that under a probabilistic model the input sequence has entropy $\mathcal{E}$. Then for a wide variety of models (thus, assumptions), the expected length of the output sequence is very close to its theoretical lower bound, $\mathcal{E}$. Models covered include a stream of independent words taken from a set of words with given probabilities. Interestingly, unlike in prefix coding, we do not need to know these probabilities beforehand, as the Lempel-Ziv method automatically adjusts itself.

References

(i) Jacob Ziv, Abraham Lempel. *A Universal Algorithm for Sequential Data Compression.* IEEE Transactions on Information Theory Vol. 23 Issue 3, pg. 337-343, May 1977.