A few methods of compression

Wicher Heldring

February 8, 2015

Abstract

This is a brief introduction of different methods of compression.

Contents

1 LZ77

LZ77 (Lempel-Ziv 77) was released in 1977. This compression method is used in the popular zip format. In LZ77 there are 2 types of units that describe a compressed stream, the first is a literal, normally any byte character. The other is a tuple of 2 integers where the first entry describes the distance and the second describes the length. For instance a decompression of $A, (-1, 4)$ would be $AAAAA$ since we go back distance or offset = -1 and then we read 4 (length) characters and write them to the current output position. So $A, (-1, 4) = AA, (-1, 3) = AAA, (-1, 2) = AAAA, (-1, 1) = AAAAA$. A sidenote is that LZ77 is not a bijection, there are multiple ways to describe a given input stream. This means that some compressors can give better results and efficiency than other compressors. An example of this is if you are to compress $AB...BC...ABC$, there would be 2 possible ways to compress this. A greedy compressor would compress this as $..., (\text{pointer to A}, 2), C$ while an other way to describe this stream is $..., A, (\text{pointer to B}, 2)$. Depending on the implementation sometimes the second option achieves a better compression ratio since the distance is shorter and thus can be described in less bits.

The main advantage of LZ77 is the decompression is extremely fast, a lot of LZ77 variants don't improve a lot on compression ratio, but they do improve on speed instead.

1.1 LZSS

In the original LZ77 algorithm suggested that you should encode every possible match, so even matches with only length 1. However this is not a proper way to do it since the implementing this means that you often use more bits to encode a single match, than to just write it down as a literal. A variant of LZ77 is therefore LZSS (Lempel-Ziv-Storer-Szymanski). LZSS starts of by reading one bit. The bit 0 means that a literal is next (8 bits of data), reading a 1 means that a distance-length pair is next. So a better representation of a compression stream would be for instance $(0, A), (0, B), (1, -2, 2)$. Which will be decompressed in $ABAB$. The distance-length pair can be encoded in a few different ways. LZSS encodes this distance-length pair by reserving 12 bits for distance and 4 bits for length. This means that we can encode lengths up to $2^4 + 2 = 18$ since there are 4 bits and any match with length 1 or 0 is not worth the trouble to encode: encoding a literal takes 9 bits while encoding a match takes $16 + 1 = 17$ bits, 1 for the flag and 16 for the distance-length pair.

1.2 LZS

LZSS uses a fixed-size distance-length pair. LZS (LempelZivStac) improves on LZSS by using huffman codes do denote the length-distance pair. This pair is encoded by first writing down the distance. If the distance is larger than 128, write a 0 and then the length in 11 bits, otherwise write a 1 and then 7 bits for the length. The length is encoded using a fixed Huffman tree which can be found at `https://tools.ietf.org/html/rfc2395#page-3`.

2 LZW

LZW (Lempel-Ziv-Welch) uses references to a dictionary instead of a references to previous data. To encode a given input stream, first a dictionary is initialized with all the possible ASCII values mapping to themselves: $[0, 1, ..., 255] \implies [\mathrm{chr}(0), \mathrm{chr}(1), ..., \mathrm{chr}(255)]$. Next when we read we try to match the input to the longest string that we can find in the dictionary. After this add the longest string + the next input symbol to the dictionary and write down the dictionary entry of this longest string to the output stream. First use 9 bits to encode the data but as soon as the length of the dictionary exceeds 9 bits = 512, then use 10 bits etc. If the dictionary size is over 16 bits we discard the current dictionary and reinitialize it to the default ASCII table.

2.1 LZMW

LZMW (Lempel-Ziv-Miller-Wegman) is a variant of LZW that improves over LZW in 2 ways. The first is that instead of reinitializing the dictionary when it is full, it removes the least used phrase instead. Any deterministic way to determine the least used phrase will work. However the initial 256 ASCII values should never be deleted. The second improvement over LZW is that instead of adding the longest match so far + the first unknown character, we add the concatenation of the previous and current match to the dictionary. This means that instead of growing dictionary entries only letter by letter, we can add complete words to the dictionary. Which means that if you are compressing text that the dictionary are often words or series of words.

2.2 LZAP

LZAP (Lempel-Ziv All Prefixes) is also a variant of LZW that adds more entries by appending all prefixes to the dictionary. For instance if we first read 'ABC' that is in our dictionary and then 'DEF'; 'ABCD', 'ABCDE', 'ABCDEF' would be added to our dictionary, instead of only 'ABCD' in the case of LZW. This means that overall we get a better compression ratio in most cases.

3 Arithmetic codes

Huffman codes have mostly been used in compression so far, arithmetic codes are however increasingly used because they have a few advantages over huffman codes. So to encode a given string for instance 'ABCDD', the first thing is to model the distribution and range. This would be the model of 'ABCDD'.

Key	Probabilty	Range
A	0.2	0.0-0.2
B	0.2	0.2-0.4
C	0.2	0.4-0.6
D	0.4	0.6-1

The algorithm to encode a given input stream recursively is as follows.

$$L_0 = 0.0 \qquad f_L(x) = \text{low range in tabel above}$$
$$H_0 = 1.0 \qquad f_H(x) = \text{high range in tabel above}$$

$$L_{n+1} = L_n + (H_n - L_n) * f_L(x_n)$$
$$H_{n+1} = L_n + (H_n - L_n) * f_H(x_n)$$

Where x_n is the n'th input symbol in the input stream. So encoding 'ABCDD' would be done like this.

Input symbol	L_n	H_n
A	0.0	0.2
B	0.24	0.28
C	0.256	0.264
D	0.2608	0.264
D	0.26272	0.264

Any float between 0.26272 and 0.264 would describe the string 'ABCDD' in this model. So given a probability distribution this is an efficient way of encoding this information.

4 PAQ

A whole different subset of compressors is the PAQ family. This tree of compressors all follow the modeling way of compression. They try to predict the next n bits given the previous m bytes of data. This distribution of what the next n bits can be will then be encoded using arithmetic codes. The models however are static and often PAQ algorithms first try to detect the kind of data it has to compress before actually choosing the required model. New modern version of PAQ are extremely slow, so slow that previous mentioned algorithms often outperform them by 100x or more. This is because for most of these models the only goal is to reach optimal compression speed. Thus the newer algorithms use large neural networks for predicting this probability distribution. Another disadvantage is that the decompresser has to do exactly the same amount of work as the compressor. Since they bot contain the same prediction model for the data and have to predict it to decompile the arithmetic codes.

References

Michael Dipperstein. LZSS (LZ77) Discussion and Implementation. `http://michael.dipperstein.com/lzss/`, 2014. [Online; accessed Jan-2015].

Matt Mahoney. Data compression explained. `http://mattmahoney.net/dc/dce.html`, 2013. [Online; accessed Jan-2015].

David Salomon. *Data Compression: The Complete Reference*. Springer Science & Business Media, 2007.