Coding LZ Phrases

Juha Kärkkäinen

University of Helsinki

Seminar on Data Compression
January 30, 2017

Coding Phrases

Each phrase in an LZ parsing is represented as a pair $(len, dist)$:

- len = length of phrase
- $dist$ = distance to the previous occurrence

We need to encode the pair and store it as part of the compressed file. To achieve good compression, we want to use as few bits as possible.

Sometimes there is a literal character *char* instead of the pair $(len, dist)$ and we need to be able to recognize when this happens: Two simple ways to do this are:

- Extra bit
- $len = 0$ means that it is followed by *char* instead of *dist*.

Binary code

len and *dist* are integers, so we need some way of encoding integers. The standard way is a binary code:

Binary code

For any $b \geq 0$ and $n \in [0..2^b - 1]$

$$\text{binary}_b(n) = b\text{-bit binary representation of } n.$$

A drawback of a binary code is that it cannot encode integers larger than $2^b - 1$ for some fixed b. If b is large, even small numbers are encoded with a large number of bits.

Because of this, some compressors limit the values of *len* and *dist*:

- To limit *len*, a long phrase is split into multiple shorter ones.
- To limit *dist*, the compressor allows previous occurrences only in a limited length sliding window.

But these limits can reduce the compression.

Unary code

Unary code a simple code with no upper bound and short codes for small integers.

Unary code

For any $n \in \mathbb{N}$

$$\text{unary}(n) = 1^n 0$$

For example, $\text{unary}(5) = 111110$.

However, it uses too many bits for encoding all except the smallest integers.

We will next look at some more sophisticated codes.

Prefix codes

To make decoder's job easier, a code should always be a prefix code.

Prefix code

No code word (binary string encoding a value) is a prefix of another code word.

With prefix codes, you can always tell where one code word ends and another begins. Both binary and unary codes are prefix codes.

Note that the no prefix rule is applied separately within each code.

- We can have one code for *len* and another code for *dist*.
- A code word for a *len*-value can be a prefix of a code word for a *dist*-value, assuming we always know which one the current code word encodes.

Kraft's Inequality

The code word lengths in a prefix code must satisfy Kraft's Inequality:

Kraft's Inequality

There exists a prefix code with codeword lengths $\ell_1, \ell_2, \ell_3, \ldots$ if and only if

$$\sum_i 2^{-\ell_i} \leq 1 \ .$$

Most good codes, including binary and unary codes, actually satisfy the equality $\sum_i 2^{-\ell_i} = 1$.

Since a code always exists if the lengths satisfy the ineqality, often the main task in designing a code is choosing the code word lengths.

Elias gamma and delta codes

Elias gamma and delta codes are more sophisticated codes for unbounded integers.

Elias γ and δ

For any $n \in \mathbb{N}$

$$\gamma(n) = \text{unary}(k) \cdot \text{binary}_k(n - 2^k + 1)$$
$$\delta(n) = \gamma(k) \cdot \text{binary}_k(n - 2^k + 1)$$

where $k = \lfloor \log(n+1) \rfloor$.

The lengths of the codes are

$$|\gamma(n)| = 2\lfloor \log(n+1) \rfloor + 1$$
$$|\delta(n)| = \lceil \log(n+2) \rceil + 2\lfloor \log \lceil \log(n+2) \rceil \rfloor$$

For example, $\gamma(6) = 110\ 11$ and $\delta(6) = 101\ 11$

Variants of Elias Codes

In Elias codes, the first part encodes the length of the second part. If there is an upper bound on the values, we can take advantage of that in encoding the first part. For example

$$\text{double-binary}_b(n) = \text{binary}_b(k) \cdot \text{binary}_k(n - 2^k + 1)$$

An interesting variant of Elias gamma code is to require the second part to be a multiple of some integer q:

$$\gamma_q(n) = \text{unary}(k) \cdot \text{binary}_{kq}\left(n - \frac{2^{kq} - 1}{2^q - 1}\right)$$

This results in a popular byte-aligned code known as v-byte if we

- set $q = 7$
- interleave the first and second part so that each byte (8 bits) has one bit from the first part and 7 bits from the second part.

Huffman coding

Suppose we want a code for the values in a finite (and relative small) set Σ. An optimal prefix code is a one that minimizes the average code word length

$$\sum_{x \in \Sigma} f(x)\ell(x)$$

where

- $f(x) \in [0, 1]$ is the frequency of the value x
- $\ell(x)$ is the code word length for the value x

The Huffman algorithm computes the optimal code word lengths given the frequencies.

The algorithm can produce the actual code words too, but these are often replaced by a canonical Huffman code with the same code lengths.

Huffman code table

Huffman code is not suitable for very large sets Σ because the codes have to be included in the compressed file separately in what is often called a Huffman code table. A Huffman code table could store for each value in Σ:

- the frequency (so that the decoder can run the Huffman algorithm),
- the actual code word, or
- the code word length (with both the encoder and the decoder using canonical Huffman codes).

The last one is usually preferred. Storing a Huffman code table in as small space as possible is an art in itself.

A huffman code can be used for example to encode

- *len*-values
- the first part in an Elias-type code for *dist* values.

Entropy

If the code word lengths could be arbitrary real numbers, there would be a simple formula for determining the optimal code word length:

$$\ell(x) = \log\left(\frac{1}{f(x)}\right).$$

The resulting average code word length is known as the entropy and represents a kind of information theoretic lower bound for compression:

$$H(f) = -\sum_{x\in\Sigma} f(x)\log f(x)\ .$$

For example

frequency	0.2	0.3	0.1	0.2	0.1	0.1
optimal code word length	2.32	1.74	3.32	2.32	3.32	3.32
Huffman code word length	2	2	3	2	4	4

$H(P) \approx 2.45$ while the average Huffman code length is 2.5.

Arithmetic coding

Huffman code can be especially bad for highly skewed frequencies:

frequency	0.99	0.01
optimal code word length	0.014	6.64
Huffman code word length	1	1

$H(P) \approx 0.081$ while the average Huffman code length is 1.

Arithmetic coding (also known as range coding) is a coding technique that can effectively use fractional code word lengths (of limited precision) and gets very close to entropy.

Drawbacks of arithmetic coding include

- Arithmetic coding is slow.
- The decoder needs to know the frequencies with high enough accuracy, which increases the size of the code table.

A recent invention is Asymmetric Numeral Systems, ANS, which achieves compression similar to arithmetic coding but at a speed close to Huffman coding.

Computing frequencies

Huffman and arithmetic coding require the knowledge of the frequencies, which has some drawbacks:

- The code tables require additional space.
- Computing the frequencies requires one pass over the file followed by another pass for encoding. Sometimes just one pass is desirable.

Furthermore, a single global frequency for a value might not always be optimal, for example:

- The file may not be uniform and it might be better to use different frequencies in different parts of the file.
- The frequencies may depend on the context. For example, the frequencies of *dist*-values may be different for small *len*-values than large *len*-values.

The region or context dependend frequencies can be implemented with multiple code tables, but this increases the size of the stored tables further.

Adaptive coding

An alternative is adaptive coding, where the frequencies (or rather the probabilities) of the next value are estimated using some dynamic, probabilistic model during the encoding. The same model must be used by the decoder. There is no need for code tables as long as only information in the already en/decoded part is used.

In the simplest case, the model simply counts the frequencies of values seen in the so far en/decoded part, but much more complicated models are possible. For example:

- Regional variation can be supported by giving higher weight to more recent observations.
- There could be a different model for each context, but there could also be a combined model that takes the context into account but can use information from other contexts too.

Adaptive models are most commonly used with arithmetic coding. A complex model can significantly slow down endcoding and decoding.

Further information

- More details on many of the topics, including Huffman and arithmetic coding, can be found in the lecture notes for the Data Compression Techniques course:

  `https://www.cs.helsinki.fi/courses/582487/2012/k/k/1`
  `https://www.cs.helsinki.fi/courses/582487/2015/k/k/1`

- Most of the topics here have wikipedia pages that then have further pointers.
- One student could take Asymmetric Numeral Systems as a topic.
- There are probably many techniques and tricks that can be found only in the source code of actual compressors. Looking at those is the main purpose of this seminar.