Lempel-Ziv-Welch (LZW) Data Compressing Model

Martin Chakravorti

Information

What is information? Any interaction between objects, when one of them acquires some substance, and the other(s) don't lose it, is called information interaction, and the transmitted substance is called information. Multimedia information (MMI) is understood, as a rule, as sound (audio stream), two-dimensional pictures, video (2D pictures stream) and three-dimensional images.

Units

A **Bit** is an "atom" of digital information (Data): A finite sequence of bits is called a **Code**. A **Byte** consists of eight bits and can have 256 different values (0...255). For computers it is easier to deal with bytes than with bits, because each byte has a unique address in memory, each address points to a particular byte.

History

Claude Shannon formulated in his 1948 paper, “A Mathematical Theory of Communication” the theory of data compression and found the Shannon-Fano compressor. Huffman Coding was another compressor. But, it was only optimal for a fixed block length, assuming that the source statistics were known before.

History

The underlying data compression models were found by Jacob Ziv and Abraham Lempel in 1977 (LZ-77) and 1978 (LZ-78), respectively. Some years later, in 1984, Terry Welch refined the scheme. Together, they stand for the current name: LZW.

Compression Possible

Examples for file compression:

Texts in any languages, HTML files, Acrobat Reader 6.0, Graphics with Bitmap (JPEG), PDF from Macromedia Flash MX Manual, Adobe Acrobat documents etc.

LZ-77 and LZ-78

The two most widely used technique for lossless file compression are LZ-77 and LZ-78. LZ-77 exploits the fact that words and phrases within a text file are likely to be repeated. When they do repeat, they can be encoded as a pointer to an earlier occurrence, with the pointer accompanied by the number of characters to be matched. Incoming data is split into blocks which are then transformed as a whole. It is handled either as stream or as blocks. The more homogeneous and bigger the data and memory, the more effective are block algorithms, the less homogeneous and smaller data and memory, the better stream methods.

LZ-77

As a matter of fact, LZ-77 will typically compress text to a third or less of its original size. The hardest part to implement, is the search for matches in buffer.

LZ-77

Key to the operation of LZ-77 is a sliding history buffer, also known as a "sliding window", which stores the most recently transmitted text. When this look- ahead-buffer fills up, its oldest contents are discarded. The size of the buffer is important. If it is too small, finding string matches will be less likely. If it is too large, the pointers will be larger, working against compression.

Difference between LZ-77 & LZW

In comparison to the **LZ-77**, which uses pointers to previous words or parts of words in a file to obtain compression, the **LZW** takes that scheme one step further. Basically, the **LZW** is constructing a "dictionary" of words or parts of words in a message, and then using pointers for the dictionary entries.

LZW-Binary Code

There are only two possible states: full(1, one, true, yes, exists) or empty (0, zero, false, no, doesn't exist). Actually, the dictionary size is limited to 12 bits per index, which results to a maximal dictionary size of 4096 (4K) words.

Concept of LZW

Many files, especially text files, have certain strings that repeat very often, for example " the ". With the spaces, the string takes 5 bytes, or 40 bits to encode. But it is better to add the whole string to the list of characters after the last one, at 256. Then every time it reaches the word "the", it just sends the code 256. This would take 9 bits instead of 40 (since 256 does not fit into 8 bits).

Example for LZW

The_rain_in_Spain_falls_mainly_in_the_plain. The underscores ("_") indicate spaces. This uncompressed message is 43 bytes, or 344 bits, long.

At first, LZW simply outputs uncompressed characters, since there are no previous occurrences to refer back to. It starts with the words:

The_rain_. Then, the following word arrives:

in_. This word has occurred earlier in the message, and can be represented as a pointer back to that earlier text, along with a length field. This gives:

The_rain_ <3,3>, where the pointer syntax hints "look back three characters and take three characters from that point." There are two different binary formats for the pointer: a) an 8-bit pointer plus 4-bit length, which assumes a maximum offset of 255 and a maximum length of 15. and b) a 12-bit pointer plus 6-bit length, which assumes a maximum offset size of 4096, implying a 4 kilobyte buffer, and a maximum length of 63.

Decompression

In fact, the decompressor builds its own dictionary on its side, that matches exactly with the compressor's, so that only the codes need to be sent. Therefore, decompression works in the reverse fashion as compression. The decoder knows that the last symbol of the most recent dictionary entry is the first symbol of the next parse block. Consequently, the codes, generated by the compressor, are generally at least one step "behind" the data of the decompressor.

Criticism about LZW

There is a limit imposed in the original LZW implementation by the fact that once the 4096-bit dictionary is complete, no more strings can be added. Defining a larger dictionary of course results in greater string capacity, but also longer pointers, reducing compression for messages that do not fill up the dictionary.