Martin Chakravorti

ISNM
Media Technology
Prof. Andreas Schrader

Summary of my Slide Presentation from November 26, 2003 on Lempel-Ziv-Welch's data compression model

In my introduction I explained what information is: 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.

Then I defined the most important physical units in computer science. 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.

After that, I referred to the history of data compression. Claude E. Shannon formulated in his 1948 paper, "A Mathematical Theory of Communication" the theory of data compression and founded the first, the so-called Shannon-Fano compressor.
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.

On my next slide I showed what kind of files can be send in a compressed form:
Texts in any languages, HTML files, Acrobat Reader 6.0, Graphics with Bitmap (for JPEG-images), MPEG for videos, PDF from Macromedia Flash MX Manual and Adobe Acrobat documents.

I pointed out, that the original technique for lossless file compression is LZ-77 and LZ-78. LZ-77 makes use of the circumstance 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. 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, however.

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 lookahead-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.

Hamburger Str. 57, D-23558 Lübeck, Tel: 0451 / 893683, Fax: 0451 / 8714784, Handy: 0173 / 6719422, Email: chakravorti@t-online.de

On the following slide I explained the difference between LZ-77 and LZW, which is until now the most widely used technique for data compression. 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.

Subsequently, I presented the LZW-Binary Code. Therein are only two states possible: full (1, one, true, yes, exists) or empty (0, zero, false, no, does not exist). Actually, the dictionary size is limited to 12 bits per index, which results to a maximal dictionary size of 4096 (4K) words.

Later on I gave a short example how the LZW-concept actually works:
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).

Then, I talked about the decompression process. In fact, the decompressor builds its own dictionary on its side that matches exactly with the compressors, 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.

I concluded the presentation with criticism about the model, because there is a limit imposed on the original LZW implementation. 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.

Reference is made to the example for data compression on slide #13.

Hamburger Str. 57, D-23558 Lübeck, Tel: 0451 / 893683, Fax: 0451 / 8714784, Handy: 0173 / 6719422, Email: chakravorti@t-online.de