Description of compressor/decompressor

Jason Leake

1. Introduction

The compressor consists of two programs.

jlcompress	Performs compression and decompression. It will automatically decompress a compressed file and compress an uncompressed one
jldecompress	Performs decompression only. It will complain if given an uncompressed file, and decompress a compressed one. It is strictly not necessary since jlcompress can decompress files, but is included because the specification asked for a pair of programs.

2. Command line

./jlcompress <switches> inputFile [outputFile]

--help or -h	Print some help
--force or -f	Overwrite output file if it doesn't exist
--flip	Disable default compression and flip the order of bits in the file
--huffman	Disable default compression and Huffman encode the file
--rle	Disable default compression and run length encode the file

The default compression is identical to specifying --rle --huffman. The order of compression is always flip, run-length encode and finally Huffman although steps may be left out (and in the default case the flipping always is). The compression switches have no effect when a file is being decompressed.

./jldecompress <switches> inputFile [outputFile]

--help or -h	Print some help
--force or -f	Overwrite output file if it doesn't exist

Default output files

If no output file is specified, then the input filename with the suffix ".decompressed" or ".compressed" is used as appropriate. If the input file already has one of these suffixes then the output file has this removed and the alternative substituted.

3. Compressed file structure

The compressed file has the following format:

4 byte header "JLCM"	Identifies the file as a compressed file. In a "real" program the magic number would be non-printable to make it harder to change with a text editor
1 byte compression algorithm bitmask	Indicates how the file was compressed
Compressed data	

The compression code can therefore determine whether the file is compressed or not, and if it is compressed which algorithms were used. The jlcompress program uses this to determine whether to compress or decompress a file and how to decompress it. The jldecompress just decompresses and so uses this information to generate an error message if it is not given a compressed file.

4. Compression algorithms

The compressor can carry out three operations on an HTML file to compress it. They are always carried out in the same order. One of the operations simply improves the efficiency of run-length encoding and therefore does not itself change the size of the file.

"Flipping" the data

The most significant bit of every byte is concatenated together to form the start of the output block, followed by the second most significant byte of every byte and so on. It doesn't result in any compression, but since in a seven bit ASCII file the most significant of every byte is 0, it coalesces these together to make for more efficient run length encoding.

Run length encoding

Handles sequences of 4-256 identical characters. If there are less than four characters in succession then since an identical.

The format of a repeat sequence is:

<repeat code byte><count><repeated byte>

An <escape code byte> is used to prefix a <repeat code byte> when the latter appears in the text being compressed, and to escape itself when it appears in the text. The repeat and escape characters are non-ASCII characters to make it less likely that they will appear in the text being compressed, since it takes two bytes to encode them. Since a repeat byte sequence is three bytes long, only four byte and upwards repeats are represented in this way. Smaller repeats are left as short runs of identical characters. The only exception is if the <repeat code byte> or <escape code byte> appear in a short run in the file, since it takes two bytes to encode each one of these it is profitable to repeat them even if they only occur twice because the <repeated byte> need not be escaped whatever character it is.

The repeat count is one less than the number of times that the character actually occurs, so a repeat count of 255 indicates that the character occurs 256 times in succession.

Huffman encoding

The compressor will also Huffman compress the text file. The frequency table used to generate the Huffman table is prepended to the start of the compressed data. The format of the frequency table is as follows, although the table columns do not represent bytes or words:

Number of bytes in the file (unsigned long)		
Number of frequency table entries – 1 (unsigned char)		
Byte value (unsigned char)	Number of bytes of count (unsigned char)	Count (size specified by previous field)
Byte value (unsigned char)	Number of bytes of count (unsigned char)	Count (size specified by previous field)
etc		
Compressed text		

Number of bytes in the file

This is needed because the Huffman compression does not necessarily finish on a byte boundary. A better representation might have been to record how many bits were unused at the end of the last byte, since this would fit into a byte, but this was easier to implement.

Number of frequency table entries

Only entries for bytes which occur in the file are included, i.e. if the byte value isn't present in the uncompressed text file then there is no frequency table entry for it. Since the count can range from 1 to a very large number a variable number of bytes is used to store it, and consequently the number of bytes is stored in the frequency table entry as well as the bytes themselves.

5. Test programs

There are three Perl scripts used for testing. They can be run in sequence using “make alltests”.

generalTests.pl

This compresses and decompresses Huffman_coding.html in all of the combinations of algorithms that the program provides, and using both decompression programs. As well as being run directly, it can me run using the “test” Makefile build target.

rleTests.pl

This just performs run length compression. It generates files containing sequences of 1 to 999 repeat code bytes, escape code bytes or normal characters and compresses and decompresses them. The main reason for this test program is to allow edge conditions such as sequences of 256 or 257 identical bytes to be tested.

bigFile.pl

This program tests that the code can handle large files. It creates a file of just over 100 million bytes and compresses and decompresses it using the various permutations.

6. Observations

For the sample HTML file used by generalTests.pl, the percentage compression obtained is:

Algorithm	**Compression**	**Comment**
RLE only	0.2%	Only frequent repeats are whitespace
Bit flip + RLE	15.3%	MSBs in file are all 0, and coalesced. So 1/8 of the file is a single run of 0s
Huffman	33.7%	Virtually the same as RLE + Huffman since RLE contributes very little for this file.
RLE + Huffman	33.7%	The default algorithm that the program uses. RLE is fast to perform and may provide benefits in files with much whitespace
Bit flip + Huffman	10.7%	Worse than Huffman on its own because the input to the Huffman compressor contains more or less equal numbers of all 256 byte values so the tree depth is greater. Consequently the average bit pattern length is greater
Bit flip + RLE + Huffman	16.9%	Worse than RLE + Huffman because the input to the Huffman compressor contains more or less equal numbers of all 256 byte values so the tree depth is greater, consequently the average bit pattern length is greater

7. Possible improvements

General

The compressed files are not portable across architectures. This is because architecture-dependent constants have been used, such as the size of an unsigned long in the byte count used in the Huffman compression. The program has also not taken any account of little endian versus big endian architectures in any multi-byte counts. The solution would probably be to use the XDR library to convert data to a standard format.

The algorithms are probably not the best ones for text compression. A better choice may be to replace RLE with LZW before the Huffman compression.

RLE

Since runs of three or less characters are not used, there are three unused values in the repeat count byte. 0, 1 and 2 could be used to represent 257, 258 and 259 repeated characters.

Huffman

The Huffman tree is generated using a single priority queue, which, according to the Wikipedia article on Huffman coding [1], means that takes n log n time to construct the tree. The article also describes a linear-two queue technique which could be used instead.

8. Acknowledgements

I drew extensively on the explanation in the Wikipedia article on Huffman Coding [1] to write the program. In particular I used the simpler of the two algorithms described in it for constructing the Huffman tree, which uses a single priority queue.

I also used the source HTML of this article, Huffman_coding.html, as my primary test file, and have included it in my submission since it is used by the main regression test script generalTests.pl.

9. References

[1] http://en.wikipedia.org/wiki/Huffman_coding, *Huffman coding*, Wikipedia