EXTENDED HUFFMAN CODING (BLOCK HUFFMAN CODING)

- Prepared By:
- Jay Kakdiya (160410116046)
- Kaushiki Kansara (160410116048)
- Devangini Kathad (160410116049)
- Kaustubh Wade (160410116050)

1

Introduction

Huffman coding is not the best coding method now, but it may be the most-cited coding method until today. Huffman published his paper on coding in 1952, and it instantly became the most imperative work in information theory. Huffman's original work spawned many variations. And it dominated the world of coding until the early 1980s. But there are some practical problems in using original Huffman coding. One of the most prominent problems is that we have to read the whole stream prior to coding. This is a major problem when (i) the file size is too large; (ii) the source stream is continuous.

2

Introduction Continued

When the file size is large it will take much a longer time to build the Huffman header for compression. This happens because we have to read the whole file twice from the source stream, that is, in most cases hard disks. In the first read pass, we have to build the Huffman tree to build codes for individual character. In the second read pass, we do the real work of compression. If the file size is small enough to be stored in the main memory, reading in the second pass can be done from the main memory instead of the hard disk. This will reduce the overhead time of reading from a slow speed device twice. But when the file size is large, we cannot store it in the main memory.

3

Introduction Continued

And thus, we are unable to avoid the second time reading from the hard disk drive. With the original method of Huffman coding, there is no way to avoid it.

When the source stream is continuous, the original Huffman coding is simply inapplicable. Because without knowledge of the total stream, we cannot build the Huffman tree to compress the input stream. However, there is a remedy for this problem, which is known as dynamic Huffman coding.

4

Proposed Alternative Method

Our proposed method is pretty simple. The main idea is to break the input stream into blocks and compress each block separately. We choose block size in such a way that we can store one full single block in main memory. Our proposed method is termed block Huffman coding. We have used BHC for block Huffman coding and PHC for pure Huffman coding. The proposed methods for static and continuous data are outlined below.

5

Algorithm for Static Data

1. Read a block from the stream into the main memory.
2. Build the Huffman tree and code for this block.
3. Compress this block by reading it from the main memory.
4. Put the header and compressed data to the output stream.
5. If there is more data in the input stream, go to Step 1. Otherwise coding is ended.

6

How It Solves Our Problem

This method can handle both the drawbacks mentioned in the previous section. In the first case, as we are reading the file from the hard disk only once, compression speed increases significantly, because the second pass reading is done from the main memory that is much faster than the hard disk. Now the file size may be as large as we can imagine without suffering double penalties for reading two times from the hard disk drive. So our first problem is practically solved.

7

Algorithm for the Continuous Stream

Now let us consider the second shortcoming. In information transfer in a network environment, we have to face a continuous stream quite often. For this case, we can modify the above idea in the following way.

1. Read data from the stream into the main memory.
2. If the block is not completed, then go to Step 1.
3. Build the Huffman tree and code for this block.
4. Compress this block by reading it from the main memory.

8

Algorithm for Continuous Stream Cont.

5. Put the header and compressed data to the output stream.
6. Go to Step 1. However, during Steps 3-5, we have to store the incoming data in parallel mode. In this way. we can solve the problem in the case of a continuous stream.

9

Block Size

Another thing we have to consider in this proposed method is the block size. The main limiting factor here is the size of the usable main memory. If we take block size to be as small as 1 Kbyte, there would be a large number of blocks. However, this will incur much less memory overhead. But as for each block, we have to store a header, the storage overhead for headers is significant. This decreases with the increasing size of the block. But increase in block size must put up with the usable size of physical memory. We can use a block size as moderate as 5 Kbytes, 10 Kbytes, or 12 Kbytes.

10

Locality

In original Huffman coding, the code tree is built after reading the whole file. The algorithm assumes that the probability of every character is almost the same in the entire file. Hence, the code tree is built for global data. But in practical cases, we have seen that characters are not scattered randomly in the entire file. For example, the ASCII character zero (null character) is found in groups as large as 10Kbytes in some executable (.exe) files. In fact, it is almost always a better idea to compress a data block with the local code tree for that block. And this consideration of locality is an inherent feature of our proposed method. So we can expect that our proposed method will perform better than the original method.

11

Compression ratio in Pure and Block Huffman Coding.

File Name	Size in Bytes	Compression Ratio in PHC(%)	Compression Ratio in BHC(%)
CDPLAYER.EXE	106496	32.83	36.44
PAINT.C	50900	36.89	37.61
GRAPHICS.LIB	29263	16.85	16.32
TEST.TXT	46186	39.55	40.92
HELP.EXE	35541	17.53	20.24
DAILER.EXE	63240	16.59	19.22
FINAL.DOC	401408	48.28	54.52
NFPE.DOC	61440	53.49	57.71
01.BMP	118448	84.66	84.6

12

Finding the Middle Ground

If we increase the block size, we can minimize the overhead of storing multiple headers. However, by taking a small size block, we can acquire greater advantage of locality feature. So we have to find the middle ground. This may vary from file to file as it is related to inherent characteristics of the input file.

13

The Results

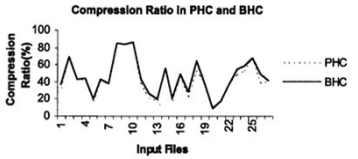
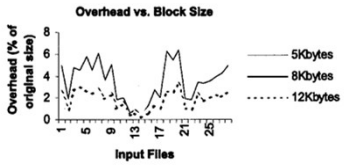


Figure 3. Compression ratio of PHC and BHC.



14

Conclusions

So we can conclude from the discussion above that to obtain better efficiency from our proposed block Huffman coding, a moderate sized block is better. Another thing we may notice is that the block size does not depend on file types. So we can use this proposed method as a general method of coding.

15

REFERENCES

1. R.G. Gallager, Variations on a theme by Huffman, IEEE tins. Infown. Theory IT-24, 668-674, (1978).
2. D.E. Knuth, Dynamic Huffman coding, Journal of Algorithms 6, 163-180, (1985).
3. M.A. Mannan, R.A. Chowdhury and M. Kaykobad, A storage efficient header for Huffman coding, Proceedings ICCIT, 57-59, (2001).
4. D.A. Huffman, A method for the construction of minimum-redundancy codes, Proceedings of the IRE 40 (9). 1098-1101, (September 1952).

16