DATA COMPRESSION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NAME :SHOUBHIK BOSE

YEAR: 4TH.

COURSE: B.TECH (2012)

CLASS ROLL NUMBER: 113

CONTENTS

INTRODUCTION

In computer science and information theory, data compression, source coding or bit-rate reduction is the process of encoding information using fewer bits than the original representation would use.

Compression is useful because it helps reduce the consumption of expensive resources, such as space or transmission bandwidth.

***On the downside*,**

Compressed data must be decompressed to be used, and this extra processing may be detrimental to some applications. For instance, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough to be viewed as it is being decompressed (the option of decompressing the video in full before watching it may be inconvenient, and requires storage space for the decompressed video).

The design of data compression schemes therefore involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced (if using a lossy compression scheme), and the computational resources required to compress and uncompress the data. Compression was one of the main drivers for the growth of information during the past two decades

WHY DATA COMPRESSION IS REQUIRED:

Compression reduces the size of a file:

- To save TIME when transmitting it.
- To save SPACE when storing it.
- Most files have lots of redundancy.

CLASSIFICATIONS OF COMPRESSION

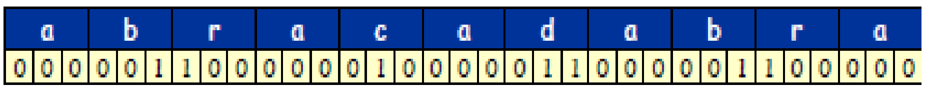
LENGTH OF CODEWORDS

1. FIXED LENGTH .
2. VARIABLE LENGTH.

INFORMATION PRESERVATION

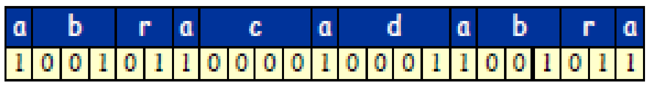
1. LOSSLESS.
2. LOSSY.

FIXED LENGTH ENCODING



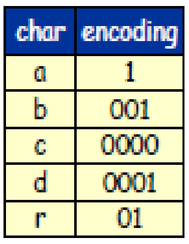
a	b	r	a	c	a	d	a	b	r	a
0 0 0	0 0 1	1 0 0	0 0 0	0 1 0	0 0 0	0 1 1	0 0 0	0 0 1	1 0 0	0 0 0

VARIABLE LENGTH ENCODING



a	b	r	a	c	a	d	a	b	r	a
1	0 0 1	0 1	1	0 0 0 0	1	0 0 0 1	1	0 0 1	0 1	1

variable length coding: 23 bits



char	encoding
a	1
b	001
c	0000
d	0001
r	01

THE BLACK BOX of COMPRESSION

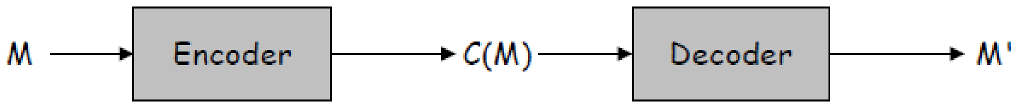
Message. Binary data M we want to compress.

Encode. Generate a "compressed" representation C(M) that hopefully uses fewer bits.

Decode. Reconstruct original message or some approximation M'.

Compression ratio: bits in C(M) / bits in M.

Lossless. M = M', 50-75% or lower.



Applications

Generic file compression:

Files: GZIP, BZIP, BOA. ; Archivers: PKZIP. ; File systems: NTFS , GFS

Multimedia,

Images: GIF, JPEG, CorelDraw. ; Sound: MP3. ; Video: MPEG, DivX™, HDTV.

Communication,

ITU-T T4 Group 3 Fax. ; V.42bis modem.

Databases.

Google (GFS)

PREFIX FREE CODES

Implementing Prefix-Free Codes

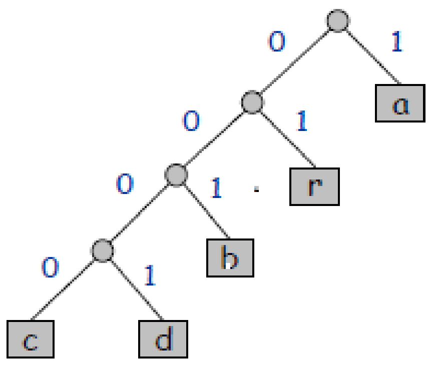
1. Use a binary trie.
2. symbols are stored in leaves
3. encoding is path to leaf.

Encoding.

1. _ Start at leaf of trie
2. corresponding to symbol s.
3. _ Follow path up to the root.
4. _ Print bits in reverse order.
5.

Decoding.

1. _ Start at root of tree.
2. _ Take right branch if bit is 0; left branch if 1.
3. _ When at a leaf node, print symbol and return to root.



char	encoding
a	1
b	001
c	0000
d	0001
r	01

HUFFMAN CODING

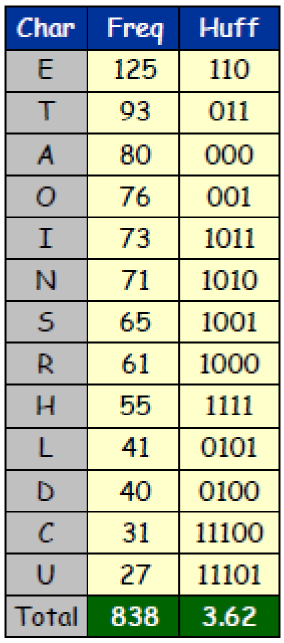
Huffman coding is optimal prefix-free code

To compute Huffman prefix-free code:

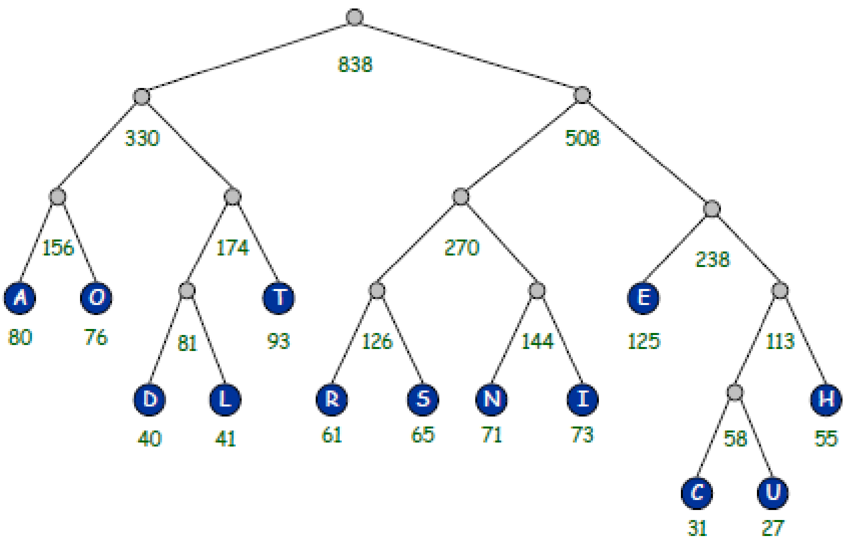
1. Count character frequencies p for each symbol s in file.
2. Start with a forest of trees, each consisting of a single vertex
3. corresponding to each symbol s with weight ps.

Repeat:

1. select two trees with min weight p1 and p2
2. merge into single tree with weight p1 + p2



Char	Freq	Huff
E	125	110
T	93	011
A	80	000
O	76	001
I	73	1011
N	71	1010
S	65	1001
R	61	1000
H	55	1111
L	41	0101
D	40	0100
C	31	11100
U	27	11101
Total	838	3.62



Run-length encoding (RLE).

1. Exploit long runs of repeated characters.
2. Replace run by count followed by repeated character, but don't bother if run is less than 3.

INPUT– AAAABBBAABBBBBCCCCCCCCDABCBAAABBBBCCCD

OUTPUT– 4A3BAA5B8CDABCB3A4B3CD

Applications.

1. Black and white graphics.
2. JPEG (Joint Photographic Experts Group).

VARIOUS COMPRESSION IMPLEMENTATIONS

Text:

- Run-length encoding (RLE) – a
simple scheme that provides good compression of data containing lots of runs of the same value.
- Lempel-Ziv 1978 (LZ78), Lempel-Ziv-Welch (LZW) – used by GIF images and compress among many other applications
- DEFLATE – used by gzip, ZIP (since version 2.0), and as part of the compression process of Portable Network Graphics (PNG), Point-to-Point Protocol (PPP), HTTP, SSH
- bzip2 - using the Burrows–Wheeler transform, this provides slower but higher compression than DEFLATE
- Lempel–Ziv–Markov chain algorithm (LZMA) - used by 7zip, xz, and other programs; higher compression than bzip2 as well as much faster decompression.
- Lempel–Ziv–Oberhumer (LZO) - designed for compression/decompression speed at the expense of compression ratios
- Statistical Lempel Ziv - a combination of statistical method and dictionary-based method; better compression ratio than using single method.

Audio

- Free Lossless Audio Codec – FLAC
- Apple Lossless – ALAC (Apple Lossless Audio Codec)
- apt-X – Lossless
- Adaptive Transform Acoustic Coding – ATRAC
- Audio Lossless Coding – also known as MPEG-4 ALS
- MPEG-4 SLS – also known as HD-AAC
- Monkey's Audio – Monkey's Audio APE
- OptimFROG
- RealPlayer – RealAudio Lossless
- Shorten – SHN
- TTA – True Audio Lossless
- WavPack – WavPack lossless
- WMA Lossless – Windows Media Lossless

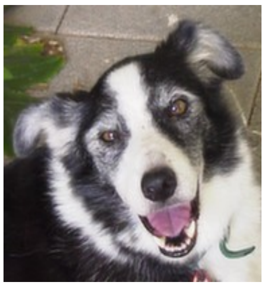
Graphics

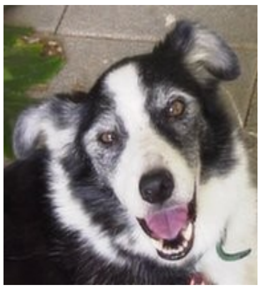
- ILBM – (lossless RLE compression of Amiga IFF images)
- JBIG2 – (lossless or lossy compression of B&W images)
- JPEG-LS – (lossless/near-lossless compression standard)
- JPEG 2000 – (includes lossless compression method, as proven by Sunil Kumar, Prof San Diego State University)
- JPEG XR – formerly *WMPhoto* and *HD Photo*, includes a lossless compression method
- PGF – Progressive Graphics File (lossless or lossy compression)
- PNG – Portable Network Graphics
- TIFF – Tagged Image File Format
- Gifsicle (GPL) – Optimize gif files
- Jpegoptim (GPL) – Optimize jpeg files

3D Graphics

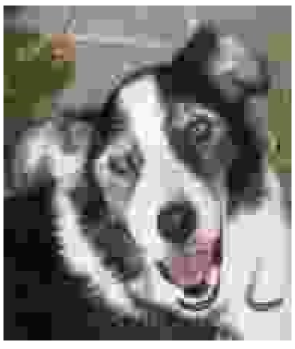
- OpenCTM – Lossless compression of 3D triangle meshes

IMAGE COMPRESSION


Original image (lossless PNG, 60.1 KB size) — uncompressed is 108.5 KB


Low compression (84% less information than uncompressed PNG, 9.37 KB)

Medium compression (92% less information than uncompressed PNG, 4.82 KB)


High compression (98% less information than uncompressed PNG, 1.14 KB)

The JPEG compression scheme – an Overview

1. Transform the image into an optimal color space.
2. Downsample chrominance components by averaging groups of pixels together.
3. Apply a Discrete Cosine Transform (DCT) to blocks of pixels, thus removing redundant image data.
4. Quantize each block of DCT coefficients using weighting functions optimized for the human eye.
5. Encode the resulting coefficients (image data) using a Huffman variable word-length algorithm to remove redundancies in the coefficients.

The BLACK-BOX

Color space converter: Convert pixels from RGB to YUV and from YUV to RGB.

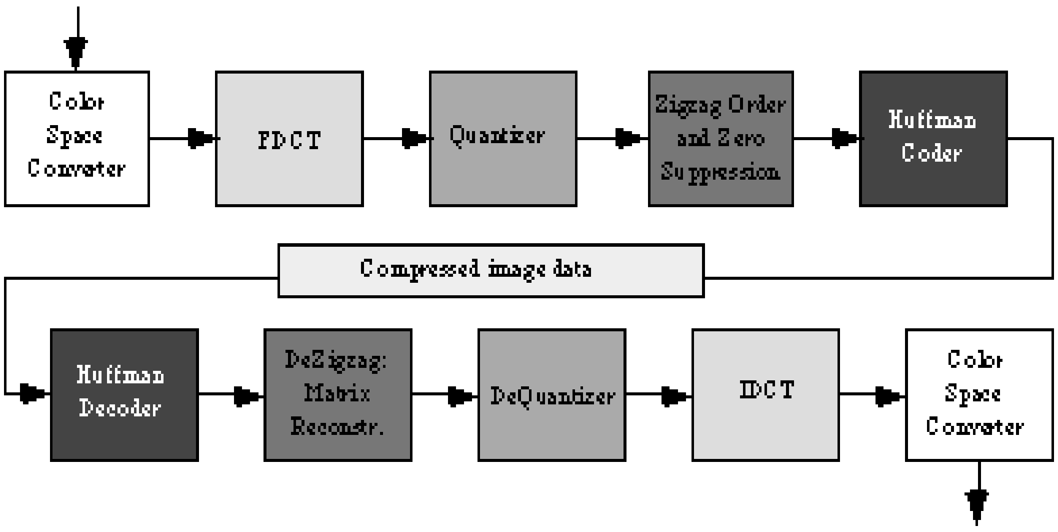
fDCT: Forward Discret Cosinus Transform, from spatial domain to frequency domain.

iDCT: Inverse Discret Cosinus Transform, from frequency domain to spatial domain.

Quantizer: Apply the lossy compression.

ZigZag: Reorder data to achieve a better compression.

Huffman coder: Compact data by representing the more common data with short codes and the less common data with longer codes.



AUDIO COMPRESSION – MP3

MP3 uses two compression techniques to achieve its size reduction ratios over uncompressed audio-one lossy and one lossless. First it throws away what humans can't hear anyway (or at least it makes acceptable compromises), and then it encodes the redundancies to achieve further compression. However, it's the first part of the process that does most of the grunt work, requires most of the complexity, and chiefly concerns us here.

Perceptual codecs are highly complex beasts, and all of them work a little differently. However, the general principles of perceptual coding remain the same from one codec to the next. In brief, the MP3 encoding process can be subdivided into a handful of discrete tasks (not necessarily in this order):

- Break the signal into smaller component pieces called " frames," each typically lasting a fraction of a second. You can think of frames much as you would the frames in a movie film.
- Analyze the signal to determine its "spectral energy distribution." In other words, on the entire spectrum of audible frequencies, find out how the bits will need to be distributed to best account for the audio to be encoded. Because different portions of the frequency spectrum are most efficiently encoded via slight variants of the same algorithm, this step breaks the signal into *sub-bands*, which can be processed independently for optimal results (but note that all sub-bands use the algorithm-they just allocate the number of bits differently, as determined by the encoder).
- The encoding bitrate is taken into account, and the maximum number of bits that can be allocated to each frame is calculated. For instance, if you're encoding at 128 kbps, you have an upper limit on how much data can be stored in each frame (unless you're encoding with variable bitrates, but we'll get to that later). This step determines how much of the available audio data will be stored, and how much will be left on the cutting room floor.
- The frequency spread for each frame is compared to mathematical models of human psychoacoustics, which are stored in the codec as a reference table. From this model, it can be determined which frequencies need to be rendered accurately, since they'll be perceptible to humans, and which ones can be dropped or allocated fewer bits, since we wouldn't be able to hear them anyway. Why store data that can't be heard?

- The bitstream is run through the process of "Huffman coding," which compresses redundant information throughout the sample. The Huffman coding does not work with a psychoacoustic model, but achieves additional compression via more traditional means. Thus, you can see the entire MP3 encoding process as a two-pass system:

1. **First you run all of the psychoacoustic models, discarding data in the process, and then you compress what's left to shrink the storage space required by any redundancies.**
2. **This second step, the Huffman coding, does not discard any data-it just lets you store what's left in a smaller amount of space.**

- The collection of frames is assembled into a serial bitstream, with header information preceding each data frame. The headers contain instructional "meta-data" specific to that frame.

CONCLUSION

1. Most people know that a compressed ZIP file is smaller than the original file, but repeatedly compressing the file will in fact usually increase in size.
2. Lossless data compression must be applied for text files.
3. Audio-Visual data may be compressed with lossy data compression, thanks to the limitations in human sensory perceptions.
4. Compression of data is inevitable for transferring data over networks to minimise network usage and therefore, costs.
5. The trade-offs between information is completely scenario-dependent.

BIBLIOGRAPHY

The image compression methods I was taught at the Image Processing sessions by Sir.J.Hazra was phenomenal and laid the foundation of understanding the concepts of data compression and its applications in real world applications.

REFERENCES

1. http://oreilly.com/catalog/mp3/chapter/ch02.html
2. http://www.cs.auckland.ac.nz/~jmor159/PLDS210/huffman.html
3. http://en.wikipedia.org/wiki/Chroma_subsampling
4. MP3: The Definitive Guide (O'Reilly).
5. www.winzip.com/wz_jpg_comp.pdf
6. http://dvd-hq.info/data_compression.php