Bibliography

Ahmed, N., T. Natarajan, and R. K. Rao (1974) "Discrete Cosine Transform," *IEEE Transactions on Computers*, **C-23**:90–93.

Bell, Timothy C., John G. Cleary, and Ian H. Witten (1990) *Text Compression*, Englewood Cliffs, Prentice Hall.

BOCU (2001) is `http://oss.software.ibm.com/icu/docs/papers/binary_ordered_compression_for_unicode.html`.

BOCU-1 (2002) is `http://www.unicode.org/notes/tn6/`.

Bookstein, Abraham and S. T. Klein (1993) "Is Huffman Coding Dead?" *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 80–87. Also published in *Computing*, **50**(4):279–296, 1993, and in *Proceedings of the Data Compression Conference, 1993*, Snowbird, UT. p. 464.

Bradley, Jonathan N., Christopher M. Brislawn, and Tom Hopper (1993) "The FBI Wavelet/Scalar Quantization Standard for Grayscale Fingerprint Image Compression," *Proceedings of Visual Information Processing II*, Orlando, FL, SPIE vol. 1961, pp. 293–304, April.

Brandenburg, Karlheinz, and Gerhard Stoll (1994) "ISO-MPEG-1 Audio: A Generic Standard for Coding of High-Quality Digital Audio," *Journal of the Audio Engineering Society*, **42**(10):780–792, October.

brucelindbloom (2007) is `http://www.brucelindbloom.com/` (click on "info").

Burrows, Michael, and D. J. Wheeler (1994) *A Block-Sorting Lossless Data Compression Algorithm*, Digital Systems Research Center Report 124, Palo Alto, CA, May 10.

Calude, Cristian and Tudor Zamfirescu (1998) "The Typical Number is a Lexicon," *New Zealand Journal of Mathematics*, **27**:7–13.

Campos, Arturo San Emeterio (2006) *Range coder*, in `http://www.arturocampos.com/ac_range.html`.

Carpentieri, B., M. J. Weinberger, and G. Seroussi (2000) "Lossless Compression of Continuous-Tone Images," *Proceedings of the IEEE*, **88**(11):1797–1809, November.

Chaitin (2007) is `http://www.cs.auckland.ac.nz/CDMTCS/chaitin/sciamer3.html`.

Choueka Y., Shmuel T. Klein, and Y. Perl (1985) "Efficient Variants of Huffman Codes in High Level Languages," *Proceedings of the 8th ACM-SIGIR Conference*, Montreal, pp. 122–130.

Deflate (2003) is `http://www.gzip.org/zlib/`.

Elias, P. (1975) "Universal Codeword Sets and Representations of the Integers," *IEEE Transactions on Information Theory*, **21**(2):194–203, March.

Faller N. (1973) "An Adaptive System for Data Compression," in *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, pp. 593–597.

Fano, R. M. (1949) "The Transmission of Information," Research Laboratory for Electronics, MIT, Tech Rep. No. 65.

Federal Bureau of Investigation (1993) *WSQ Grayscale Fingerprint Image Compression Specification, ver. 2.0*, Document #IAFIS-IC-0110v2, Criminal Justice Information Services, February.

Feldspar (2003) is `http://www.zlib.org/feldspar.html`.

Fenwick, Peter (1996a) "Punctured Elias Codes for Variable-Length Coding of the Integers," Technical Report 137, Department of Computer Science, University of Auckland, December. This is also available online.

Fenwick, P. (1996b) *Symbol Ranking Text Compression*, Tech. Rep. 132, Dept. of Computer Science, University of Auckland, New Zealand, June.

Fraenkel, Aviezri S. and Shmuel T. Klein (1996) "Robust Universal Complete Codes for Transmission and Compression," *Discrete Applied Mathematics*, **64**(1):31–55, January.

funet (2007) is `ftp://nic.funet.fi/pub/graphics/misc/test-images/`.

G.711 (1972) is `http://en.wikipedia.org/wiki/G.711`.

Gallager, Robert G. (1978) "Variations on a Theme by Huffman," *IEEE Transactions on Information Theory*, **24**(6):668–674, November.

Gardner, Martin (1972) "Mathematical Games," *Scientific American*, **227**(2):106, August.

Gemstar (2007) is `http://www.gemstartvguide.com`.

Gilbert, E. N. and E. F. Moore (1959) "Variable Length Binary Encodings," *Bell System Technical Journal*, **38**:933–967.

Gray, Frank (1953) "Pulse Code Communication," United States Patent 2,632,058, March 17.

Haar, A. (1910) "Zur Theorie der Orthogonalen Funktionensysteme," *Mathematische Annalen* first part **69**:331–371, second part **71**:38–53, 1912.

Heath, F. G. (1972) "Origins of the Binary Code," *Scientific American*, **227**(2):76, August.

Hecht, S., S. Schlaer, and M. H. Pirenne (1942) "Energy, Quanta and Vision," *Journal of the Optical Society of America*, **38**:196–208.

hffax (2007) is `http://www.hffax.de/html/hauptteil_faxhistory.htm`.

Hilbert, D. (1891) "Ueber stetige Abbildung einer Linie auf ein Flächenstück," *Math. Annalen*, **38**:459–460.

Hirschberg, D., and D. Lelewer (1990) "Efficient Decoding of Prefix Codes," *Communications of the ACM*, **33**(4):449–459.

Holzmann, Gerard J. and Björn Pehrson (1995) *The Early History of Data Networks*, Los Alamitos, CA, IEEE Computer Society Press. This is available online at `http://labit501.upct.es/ips/libros/TEHODN/ch-2-5.3.html`.

Huffman, David (1952) "A Method for the Construction of Minimum Redundancy Codes," *Proceedings of the IRE*, **40**(9):1098–1101.

incredible (2007) is `http://datacompression.info/IncredibleClaims.shtml`.

ITU-T (1989) CCITT Recommendation G.711: "Pulse Code Modulation (PCM) of Voice Frequencies."

JuergenAbel (2007) is file `Preprint_After_BWT_Stages.pdf` in `http://www.data-compression.info/JuergenAbel/Preprints/`.

Karp, R. S. (1961) "Minimum-Redundancy Coding for the Discrete Noiseless Channel," *Transactions of the IRE*, **7**:27–38.

Knuth, Donald E. (1985) "Dynamic Huffman Coding," *Journal of Algorithms*, **6**:163–180.

Kraft, L. G. (1949) *A Device for Quantizing, Grouping, and Coding Amplitude Modulated Pulses*, Master's Thesis, Department of Electrical Engineering, MIT, Cambridge, MA.

Linde, Y., A. Buzo, and R. M. Gray (1980) "An Algorithm for Vector Quantization Design," *IEEE Transactions on Communications*, **COM-28**:84–95, January.

Lloyd, S. P. (1982) "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, **IT-28**:129–137, March.

Manber, U., and E. W. Myers (1993) "Suffix Arrays: A New Method for On-Line String Searches," *SIAM Journal on Computing*, **22**(5):935–948, October.

Max, Joel (1960) "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, **IT-6**:7–12, March.

McCreight, E. M (1976) "A Space Economical Suffix Tree Construction Algorithm," *Journal of the ACM*, **32**(2):262–272, April.

McMillan, Brockway (1956) "Two Inequalities Implied by Unique Decipherability," *IEEE Transactions on Information Theory*, **2**(4):115–116, December.

MNG (2003) is http://www.libpng.org/pub/mng/spec/.

Moffat, Alistair, Radford Neal, and Ian H. Witten (1998) "Arithmetic Coding Revisited," *ACM Transactions on Information Systems*, **16**(3):256–294, July.

Motil, John (2007) Private communication.

Mulcahy, Colm (1996) "Plotting and Scheming with Wavelets," *Mathematics Magazine*, **69**(5):323–343, December. See also http://www.spelman.edu/~colm/csam.ps.

Mulcahy, Colm (1997) "Image Compression Using the Haar Wavelet Transform," *Spelman College Science and Mathematics Journal*, **1**(1):22–31, April. Also available at URL http://www.spelman.edu/~colm/wav.ps. (It has been claimed that any smart 15-year-old could follow this introduction to wavelets.)

Osterberg, G. (1935) "Topography of the Layer of Rods and Cones in the Human Retina," *Acta Ophthalmologica*, (suppl. 6):1–103.

Paez, M. D. and T. H. Glisson (1972) "Minimum Mean Squared Error Quantization in Speech PCM and DPCM Systems," *IEEE Transactions on Communications*, **COM-20**(2):225–230,

patents (2007) is http://www.datacompression.info/patents.shtml.

PDF (2001) *Adobe Portable Document Format Version 1.4*, 3rd ed., Reading, MA, Addison-Wesley, December.

Pennebaker, William B., and Joan L. Mitchell (1992) *JPEG Still Image Data Compression Standard*, New York, Van Nostrand Reinhold.

Phillips, Dwayne (1992) "LZW Data Compression," *The Computer Application Journal* Circuit Cellar Inc., **27**:36–48, June/July.

PKWare (2003) is http://www.pkware.com.

PNG (2003) is http://www.libpng.org/pub/png/.

RFC1945 (1996) *Hypertext Transfer Protocol—HTTP/1.0*, available at URL http://www.faqs.org/rfcs/rfc1945.html.

RFC1950 (1996) *ZLIB Compressed Data Format Specification version 3.3*, is http://www.ietf.org/rfc/rfc1950.

RFC1951 (1996) *DEFLATE Compressed Data Format Specification version 1.3*, is http://www.ietf.org/rfc/rfc1951.

RFC1952 (1996) *GZIP File Format Specification Version 4.3*. Available in PDF format at URL http://www.gzip.org/zlib/rfc-gzip.html.

RFC1962 (1996) *The PPP Compression Control Protocol (CCP)*, available from many sources.

RFC1979 (1996) *PPP Deflate Protocol*, is http://www.faqs.org/rfcs/rfc1979.html.

RFC2616 (1999) *Hypertext Transfer Protocol – HTTP/1.1*. Available in PDF format at URL http://www.faqs.org/rfcs/rfc2616.html.

Rice, Robert F. (1979) "Some Practical Universal Noiseless Coding Techniques," Jet Propulsion Laboratory, JPL Publication 79-22, Pasadena, CA, March.

Rice, Robert F. (1991) "Some Practical Universal Noiseless Coding Techniques—Part III. Module PSI14.K," Jet Propulsion Laboratory, JPL Publication 91-3, Pasadena, CA, November.

Robinson, Tony (1994) "Simple Lossless and Near-Lossless Waveform Compression," Technical Report CUED/F-INFENG/TR.156, Cambridge University, December. Available at `http://citeseer.nj.nec.com/robinson94shorten.html`.

Salomon, David (1999) *Computer Graphics and Geometric Modeling*, New York, Springer.

Salomon, David (2006) *Curves and Surfaces for Computer Graphics*, New York, Springer.

Salomon, D. (2007) *Data Compression: The Complete Reference*, London, Springer Verlag.

Schindler, Michael (1998) "A Fast Renormalisation for Arithmetic Coding," a poster in the Data Compression Conference, 1998, available at URL `http://www.compressconsult.com/rangecoder/`.

Shannon, Claude E. (1948), "A Mathematical Theory of Communication," *Bell System Technical Journal*, **27**:379–423 and 623–656, July and October,

Shannon, Claude (1951) "Prediction and Entropy of Printed English," *Bell System Technical Journal*, **30**(1):50–64, January.

Shenoi, Kishan (1995) *Digital Signal Processing in Telecommunications*, Upper Saddle River, NJ, Prentice Hall.

Sieminski, A. (1988) "Fast Decoding of the Huffman Codes," *Information Processing Letters*, **26**(5):237–241.

Softsound (2007) is `http://mi.eng.cam.ac.uk/reports/ajr/TR156/tr156.html`.

Strang, Gilbert, and Truong Nguyen (1996) *Wavelets and Filter Banks*, Wellesley, MA, Wellesley-Cambridge Press.

technikum29 (2007) is `http://www.technikum29.de/en/communication/fax.shtm`.

Tetrachromat (2007) is `http://en.wikipedia.org/wiki/Tetrachromat`.

Unicode (2007) is `http://unicode.org/`.

Vitter, Jeffrey S. (1987) "Design and Analysis of Dynamic Huffman Codes," *Journal of the ACM*, **34**(4):825–845, October.

Wallace, Gregory K. (1991) "The JPEG Still Image Compression Standard," *Communications of the ACM*, **34**(4):30–44, April.

Watson, Andrew (1994) "Image Compression Using the Discrete Cosine Transform," *Mathematica Journal*, **4**(1):81–88.

Weisstein-pickin (2007) is Weisstein, Eric W. "Real Number Picking." From MathWorld, A Wolfram web resource. `http://mathworld.wolfram.com/RealNumberPicking.html`.

Welch, T. A. (1984) "A Technique for High-Performance Data Compression," *IEEE Computer*, **17**(6):8–19, June.

Wirth, N. (1976) *Algorithms + Data Structures = Programs*, 2nd ed., Englewood Cliffs, NJ, Prentice-Hall.

Witten, Ian H., Radford M. Neal, and John G. Cleary (1987) "Arithmetic Coding for Data Compression," *Communications of the ACM*, **30**(6):520–540.

Wolf, Misha et al. (2000) "A Standard Compression Scheme for Unicode," Unicode Technical Report #6, available at `http://unicode.org/unicode/reports/tr6/index.html`.

Zhang, Manyun (1990) *The JPEG and Image Data Compression Algorithms* (dissertation).

Ziv, Jacob, and A. Lempel (1977) "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, **IT-23**(3):337–343.

Ziv, Jacob and A. Lempel (1978) "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, **IT-24**(5):530–536.

zlib (2003) is `http://www.zlib.org/zlib_tech.html`.

A literary critic is a person who finds meaning in literature that the author didn't know was there.

—Anonymous

Glossary

A glossary is a list of terms in a particular domain of knowledge with the definitions for those terms. Traditionally, a glossary appears at the end of a book and includes terms within that book which are either newly introduced or at least uncommon.

In a more general sense, a glossary contains explanations of concepts relevant to a certain field of study or action. In this sense, the term is contemporaneously related to ontology.

—From `Wikipedia.com`

Adaptive Compression. A compression method that modifies its operations and/or its parameters in response to new data read from the input. Examples are the adaptive Huffman method of Section 2.3 and the dictionary-based methods of Chapter 3. (See also Semiadaptive Compression.)

Alphabet. The set of all possible symbols in the input. In text compression, the alphabet is normally the set of 128 ASCII codes. In image compression, it is the set of values a pixel can take (2, 16, 256, or anything else). (See also Symbol.)

Arithmetic Coding. A statistical compression method (Chapter 4) that assigns one (normally long) code to the entire input file, instead of assigning codes to the individual symbols. The method reads the input symbol by symbol and appends more bits to the code each time a symbol is input and processed. Arithmetic coding is slow, but it compresses at or close to the entropy, even when the symbol probabilities are skewed. (See also Model of Compression, Statistical Methods.)

ASCII Code. The standard character code on all modern computers (although Unicode is fast becoming a serious competitor). ASCII stands for American Standard Code for Information Interchange. It is a $(1+7)$-bit code, with one parity bit and seven data bits per symbol. As a result, 128 symbols can be coded. They include the uppercase and lowercase letters, the ten digits, some punctuation marks, and control characters. (See also Unicode.)

Bark. Unit of critical band rate. Named after Heinrich Georg Barkhausen and used in audio applications. The Bark scale is a nonlinear mapping of the frequency scale over the audio range, a mapping that matches the frequency selectivity of the human ear.

Bi-level Image. An image whose pixels have two different colors. The colors are normally referred to as black and white, "foreground" and "background," or 1 and 0. (See also Bitplane.)

Bitplane. Each pixel in a digital image is represented by several bits. The set of all the kth bits of all the pixels in the image is the kth bitplane of the image. A bi-level image, for example, consists of one bitplane. (See also Bi-level Image.)

Bitrate. In general, the term "bitrate" refers to both bpb and bpc. However, in audio compression, this term is used to indicate the rate at which the compressed file is read by the decoder. This rate depends on where the file comes from (such as disk, communications channel, memory). If the bitrate of an MPEG audio file is, e.g., 128 Kbps, then the encoder will convert each second of audio into 128 K bits of compressed data, and the decoder will convert each group of 128 K bits of compressed data into one second of sound. Lower bitrates mean smaller file sizes. However, as the bitrate decreases, the encoder must compress more audio data into fewer bits, eventually resulting in a noticeable loss of audio quality. For CD-quality audio, experience indicates that the best bitrates are in the range of 112 Kbps to 160 Kbps. (See also Bits/Char.)

Bits/Char. Bits per character (bpc). A measure of the performance in text compression. Also a measure of entropy. (See also Bitrate, Entropy.)

Bits/Symbol. Bits per symbol. A general measure of compression performance.

Block Coding. A general term for image compression methods that work by breaking the image into small blocks of pixels, and encoding each block separately. JPEG (Section 5.6) is a good example, because it processes blocks of 8×8 pixels.

Burrows–Wheeler Method. This method (Section 7.1) prepares a string of data for later compression. The compression itself is done with the move-to-front method (see item in this glossary), perhaps in combination with RLE. The BW method converts a string S to another string L that satisfies two conditions:

1. Any region of L will tend to have a concentration of just a few symbols.

2. It is possible to reconstruct the original string S from L (a little more data may be needed for the reconstruction, in addition to L, but not much).

CCITT. The International Telegraph and Telephone Consultative Committee (Comité Consultatif International Télégraphique et Téléphonique), the old name of the ITU, the International Telecommunications Union. The ITU is a United Nations organization responsible for developing and recommending standards for data communications (not just compression). (See also ITU.)

CIE. CIE is an abbreviation for Commission Internationale de l'Éclairage (International Committee on Illumination). This is the main international organization devoted to light and color. It is responsible for developing standards and definitions in this area. (See also Luminance.)

Circular Queue. A basic data structure (see the last paragraph of Section 1.3.1) that moves data along an array in circular fashion, updating two pointers to point to the start and end of the data in the array.

Codec. A term that refers to both encoder and decoder.

Codes. A code is a symbol that stands for another symbol. In computer and telecommunications applications, codes are virtually always binary numbers. The ASCII code is the defacto standard, although the new Unicode is used on several new computers and the older EBCDIC is still used on some old IBM computers. In addition to these fixed-size codes there are many variable-length codes used in data compression and there are the all-important error-control codes for added robustness (See also ASCII, Unicode.)

Compression Factor. The inverse of compression ratio. It is defined as

$$\text{compression factor} = \frac{\text{size of the input file}}{\text{size of the output file}}.$$

Values greater than 1 indicate compression, and values less than 1 imply expansion. (See also Compression Ratio.)

Compression Gain. This measure is defined as

$$100 \log_e \frac{\text{reference size}}{\text{compressed size}},$$

where the reference size is either the size of the input file or the size of the compressed file produced by some standard lossless compression method.

Compression Ratio. One of several measures that are commonly used to express the efficiency of a compression method. It is the ratio

$$\text{compression ratio} = \frac{\text{size of the output file}}{\text{size of the input file}}.$$

A value of 0.6 indicates that the data occupies 60% of its original size after compression. Values greater than 1 mean an output file bigger than the input file (negative compression).

Sometimes the quantity $100 \times (1 - \text{compression ratio})$ is used to express the quality of compression. A value of 60 means that the output file occupies 40% of its original size (or that the compression has resulted in a savings of 60%). (See also Compression Factor.)

Continuous-Tone Image. A digital image with a large number of colors, such that adjacent image areas with colors that differ by just one unit appear to the eye as having continuously varying colors. An example is an image with 256 grayscale values. When adjacent pixels in such an image have consecutive gray levels, they appear to the eye as a continuous variation of the gray level. (See also Bi-level image, Discrete-Tone Image, Grayscale Image.)

Decoder. A program, an algorithm, or a piece of hardware for decompressing data.

Deflate. A popular lossless compression algorithm (Section 3.3) used by Zip and gzip. Deflate employs a variant of LZ77 combined with static Huffman coding. It uses a 32-Kb-long sliding dictionary and a look-ahead buffer of 258 bytes. When a string is not found in the dictionary, its first symbol is emitted as a literal byte. (See also Zip.)

Dictionary-Based Compression. Compression methods (Chapter 3) that save pieces of the data in a "dictionary" data structure. If a string of new data is identical to a piece that is already saved in the dictionary, a pointer to that piece is output to the compressed file. (See also LZ Methods.)

Discrete Cosine Transform. A variant of the discrete Fourier transform (DFT) that produces just real numbers. The DCT (Sections 5.5 and 5.6.2) transforms a set of numbers by combining n numbers to become an n-dimensional point and rotating it in n dimensions such that the first coordinate becomes dominant. The DCT and its inverse, the IDCT, are used in JPEG (Section 5.6) to compress an image with acceptable loss, by isolating the high-frequency components of an image, so that they can later be quantized. (See also Fourier Transform, Transform.)

Discrete-Tone Image. A discrete-tone image may be bi-level, grayscale, or color. Such images are (with some exceptions) artificial, having been obtained by scanning a document, or capturing a computer screen. The pixel colors of such an image do not vary continuously or smoothly, but have a small set of values, such that adjacent pixels may differ much in intensity or color. (See also Continuous-Tone Image.)

Discrete Wavelet Transform. The discrete version of the continuous wavelet transform. A wavelet is represented by means of several filter coefficients, and the transform is carried out by matrix multiplication (or a simpler version thereof) instead of by calculating an integral.

Encoder. A program, algorithm, or hardware circuit for compressing data.

Entropy. The entropy of a single symbol a_i is defined as $-P_i \log_2 P_i$, where P_i is the probability of occurrence of a_i in the data. The entropy of a_i is the smallest number of bits needed, on average, to represent symbol a_i. Claude Shannon, the creator of information theory, coined the term *entropy* in 1948, because this term is used in thermodynamics to indicate the amount of disorder in a physical system. (See also Entropy Encoding, Information Theory.)

Entropy Encoding. A lossless compression method where data can be compressed such that the average number of bits/symbol approaches the entropy of the input symbols. (See also Entropy.)

Facsimile Compression. Transferring a typical page between two fax machines can take up to 10–11 minutes without compression. This is why the ITU has developed several standards for compression of facsimile data. The current standards (Section 2.4) are T4 and T6, also called Group 3 and Group 4, respectively. (See also ITU.)

Fourier Transform. A mathematical transformation that produces the frequency components of a function. The Fourier transform represents a periodic function as the sum of sines and cosines, thereby indicating explicitly the frequencies "hidden" in the original representation of the function. (See also Discrete Cosine Transform, Transform.)

Gaussian Distribution. (See Normal Distribution.)

Golomb Codes. The Golomb codes consist of an infinite set of parametrized prefix codes. They are the best variable-length codes for the compression of data items that are distributed geometrically. (See also Unary Code.)

Gray Codes. Gray codes are binary codes for the integers, where the codes of consecutive integers differ by one bit only. Such codes are used when a grayscale image is separated into bitplanes, each a bi-level image. (See also Grayscale Image,)

Grayscale Image. A continuous-tone image with shades of a single color. (See also Continuous-Tone Image.)

Huffman Coding. A popular method for data compression (Chapter 2). It assigns a set of "best" variable-length codes to a set of symbols based on their probabilities. It serves as the basis for several popular programs used on personal computers. Some of them use just the Huffman method, while others use it as one step in a multistep compression process. The Huffman method is somewhat similar to the Shannon–Fano method. It generally produces better codes, and like the Shannon–Fano method, it produces best code when the probabilities of the symbols are negative powers of 2. The main difference between the two methods is that Shannon–Fano constructs its codes top to bottom (from the leftmost to the rightmost bits), while Huffman constructs a code tree from the bottom up (builds the codes from right to left). (See also Shannon–Fano Coding, Statistical Methods.)

Information Theory. A mathematical theory that quantifies information. It shows how to measure information, so that one can answer the question, how much information is included in a given piece of data? with a precise number! Information theory is the creation, in 1948, of Claude Shannon of Bell Labs. (See also Entropy.)

ITU. The International Telecommunications Union, the new name of the CCITT, is a United Nations organization responsible for developing and recommending standards for data communications (not just compression). (See also CCITT.)

JFIF. The full name of this method (Section 5.6.7) is JPEG File Interchange Format. It is a graphics file format that makes it possible to exchange JPEG-compressed images between different computers. The main features of JFIF are the use of the YCbCr triple-component color space for color images (only one component for grayscale images) and the use of a marker to specify features missing from JPEG, such as image resolution, aspect ratio, and features that are application-specific.

JPEG. A sophisticated lossy compression method (Section 5.6) for color or grayscale still images (not video). It works best on continuous-tone images, where adjacent pixels have similar colors. One advantage of JPEG is the use of many parameters, allowing the user to adjust the amount of data loss (and thereby also the compression ratio) over

a very wide range. There are two main modes, lossy (also called baseline) and lossless (which typically yields a 2:1 compression ratio). Most implementations support just the lossy mode. This mode includes progressive and hierarchical coding.

The main idea behind JPEG is that an image exists for people to look at, so when the image is compressed, it is acceptable to lose image features to which the human eye is not sensitive.

The name JPEG is an acronym that stands for Joint Photographic Experts Group. This was a joint effort by the CCITT and the ISO that started in June 1987. The JPEG standard has proved successful and has become widely used for image presentation, especially in web pages.

Kraft–McMillan Inequality. A relation that says something about unambiguous variable-length codes. Its first part states: Given an unambiguous variable-size code, with n codes of sizes L_i, then

$$\sum_{i=1}^{n} 2^{-L_i} \le 1.$$

[This is Equation (1.4).] The second part states the opposite, namely, given a set of n positive integers $(L_1, L_2, \ldots, L_n)$ that satisfy Equation (1.4), there exists an unambiguous variable-size code such that L_i are the sizes of its individual codes. Together, both parts state that a code is unambiguous if and only if it satisfies relation (1.4).

Laplace Distribution. A probability distribution similar to the normal (Gaussian) distribution, but narrower and sharply peaked. The general Laplace distribution with variance V and mean m is given by

$$L(V, x) = \frac{1}{\sqrt{2V}} \exp\left(-\sqrt{\frac{2}{V}}|x - m|\right).$$

Experience seems to suggest that the values of the residues computed by many image compression algorithms are Laplace distributed, which is why this distribution is employed by those compression methods, most notably MLP. (See also Normal Distribution.)

Lossless Compression. A compression method where the output of the decoder is identical to the original data compressed by the encoder. (See also Lossy Compression.)

Lossy Compression. A compression method where the output of the decoder is different from the original data compressed by the encoder, but is nevertheless acceptable to a user. Such methods are common in image and audio compression, but not in text compression, where the loss of even one character may result in wrong, ambiguous, or incomprehensible text. (See also Lossless Compression.)

Luminance. This quantity is defined by the CIE (Section 5.6.1) as radiant power weighted by a spectral sensitivity function that is characteristic of vision. (See also CIE.)

LZ Methods. All dictionary-based compression methods are based on the work of J. Ziv and A. Lempel, published in 1977 and 1978. Today, these are called LZ77 and LZ78 methods, respectively. Their ideas have been a source of inspiration to many researchers, who generalized, improved, and combined them with RLE and statistical methods to form many commonly used adaptive compression methods, for text, images, and audio. (See also Dictionary-Based Compression, Sliding-Window Compression.)

LZW. This is a popular variant (Section 3.2) of LZ78, originated by Terry Welch in 1984. Its main feature is eliminating the second field of a token. An LZW token consists of just a pointer to the dictionary. As a result, such a token always encodes a string of more than one symbol. (See also Patents.)

Model of Compression. A model is a method to "predict" (to assign probabilities to) the data to be compressed. This concept is important in statistical data compression. When a statistical method is used, a model for the data has to be constructed before compression can begin. A simple model can be built by reading the entire input, counting the number of times each symbol appears (its frequency of occurrence), and computing the probability of occurrence of each symbol. The data is then input again, symbol by symbol, and is compressed using the information in the probability model. (See also Statistical Methods.)

One feature of arithmetic coding is that it is easy to separate the statistical model (the table with frequencies and probabilities) from the encoding and decoding operations. It is easy to encode, for example, the first half of a data using one model, and the second half using another model.

Move-to-Front Coding. The basic idea behind this method is to maintain the alphabet A of symbols as a list where frequently occurring symbols are located near the front. A symbol `s` is encoded as the number of symbols that precede it in this list. After symbol `s` is encoded, it is moved to the front of list A.

Normal Distribution. A probability distribution with the well-known bell shape. It occurs often in theoretical models and real-life situations. The normal distribution with mean m and standard deviation s is defined by

$$f(x) = \frac{1}{s\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right\}.$$

Patents. A mathematical algorithm can be patented if it is intimately associated with software or firmware implementing it. Several compression methods, most notably LZW, have been patented, creating difficulties for software developers who work with GIF, UNIX `compress`, or any other system that uses LZW. (See also LZW.)

Pel. The smallest unit of a facsimile image; a dot. (See also Pixel.)

Pixel. The smallest unit of a digital image; a dot. (See also Pel.)

PKZip. A compression program developed and implemented by Phil Katz for the old MS/DOS operating system. Katz then founded the PKWare company which also markets the PKunzip, PKlite, and PKArc software (`http://www.pkware.com`).

Prediction. Assigning probabilities to symbols.

Prefix Property. One of the principles of variable-length codes. It states that once a certain bit pattern has been assigned as the code of a symbol, no other codes should start with that pattern (the pattern cannot be the prefix of any other code). Once the string 1, for example, is assigned as the code of a_1, no other codes should start with 1 (i.e., they all have to start with 0). Once 01, for example, is assigned as the code of a_2, no other codes can start with 01 (they all should start with 00). (See also Variable-Length Codes, Statistical Methods.)

Psychoacoustic Model. A mathematical model of the sound-masking properties of the human auditory (ear–brain) system.

Rice Codes. A special case of the Golomb code. (See also Golomb Codes.)

RLE. A general name for methods that compress data by replacing a run of identical symbols with one code, or token, containing the symbol and the length of the run. RLE sometimes serves as one step in a multistep statistical or dictionary-based method.

Scalar Quantization. The dictionary definition of the term "quantization" is "to restrict a variable quantity to discrete values rather than to a continuous set of values." If the data to be compressed is in the form of large numbers, quantization is used to convert them to small numbers. This results in (lossy) compression. If the data to be compressed is analog (e.g., a voltage that varies with time), quantization is used to digitize it into small numbers. This aspect of quantization is used by several audio compression methods. (See also Vector Quantization.)

Semiadaptive Compression. A compression method that uses a two-pass algorithm, where the first pass reads the input to collect statistics on the data to be compressed, and the second pass performs the actual compression. The statistics (model) are included in the compressed file. (See also Adaptive Compression.)

Shannon–Fano Coding. An early algorithm for finding a minimum-length variable-size code given the probabilities of all the symbols in the data. This method was later superseded by the Huffman method. (See also Statistical Methods, Huffman Coding.)

Shorten. A simple compression algorithm for waveform data in general and for speech in particular (Section 6.5). Shorten employs linear prediction to compute residues (of audio samples) which it encodes by means of Rice codes. (See also Rice Codes.)

Sliding-Window Compression. The LZ77 method (Section 1.3.1) uses part of the already-seen input as the dictionary. The encoder maintains a window to the input file, and shifts the input in that window from right to left as strings of symbols are being encoded. The method is therefore based on a sliding window. (See also LZ Methods.)

Space-Filling Curves. A space-filling curve is a function $\mathbf{P}(t)$ that goes through every point in a given two-dimensional region, normally the unit square, as t varies from 0 to 1. Such curves are defined recursively and are used in image compression.

Statistical Methods. Statistical data compression methods work by assigning variable-length codes to symbols in the data, with the shorter codes assigned to symbols or groups of symbols that appear more often in the data (have a higher probability of occurrence). (See also Variable-Length Codes, Prefix Property, Shannon–Fano Coding, Huffman Coding, and Arithmetic Coding.)

Symbol. The smallest unit of the data to be compressed. A symbol is often a byte but may also be a bit, a trit $\{0, 1, 2\}$, or anything else. (See also Alphabet.)

Token. A unit of data written on the compressed file by some compression algorithms. A token consists of several fields that may have either fixed or variable sizes.

Transform. An image can be compressed by transforming its pixels (which are correlated) to a representation where they are decorrelated. Compression is achieved if the new values are smaller, on average, than the original ones. Lossy compression can be achieved by quantizing the transformed values. The decoder inputs the transformed values from the compressed file and reconstructs the (precise or approximate) original data by applying the opposite transform. (See also Discrete Cosine Transform, Discrete Wavelet Transform, Fourier Transform.)

Unary Code. A simple variable-size code for the integers that can be constructed in one step. The unary code of the nonnegative integer n is defined (Section 1.1.1) as $n-1$ 1's followed by a single 0 (Table 1.2). There is also a general unary code. (See also Golomb Code.)

Unicode. A new international standard code, the Unicode, has been proposed, and is being developed by the international Unicode organization (`www.unicode.org`). Unicode specifies 16-bit codes for its characters, so it provides for $2^{16} = 64\text{K} = 65{,}536$ codes. (Notice that doubling the size of a code much more than doubles the number of possible codes. In fact, it squares the number of codes.) Unicode includes all the ASCII codes in addition to codes for characters in foreign languages (including complete sets of Korean, Japanese, and Chinese characters) and many mathematical and other symbols. Currently, about 39,000 out of the 65,536 possible codes have been assigned, so there is room for adding more symbols in the future. (See also ASCII, Codes.)

Variable-Length Codes. These codes are employed by statistical methods. Most variable-length codes are prefix codes (page 28) and should be assigned to symbols based on their probabilities. (See also Prefix Property, Statistical Methods.)

Vector Quantization. Vector quantization is a generalization of the scalar quantization method. It is used in both image and audio compression. In practice, vector quantization is commonly used to compress data that has been digitized from an analog source, such as sampled sound and scanned images (drawings or photographs). Such data is called digitally sampled analog data (DSAD). (See also Scalar Quantization.)

Voronoi Diagrams. Imagine a petri dish ready for growing bacteria. Four bacteria of different types are simultaneously placed in it at different points and immediately start multiplying. We assume that their colonies grow at the same rate. Initially, each colony consists of a growing circle around one of the starting points. After a while some of them meet and stop growing in the meeting area due to lack of food. The final result

is that the entire dish gets divided into four areas, one around each of the four starting points, such that all the points within area i are closer to starting point i than to any other start point. Such areas are called Voronoi regions or Dirichlet tessellations.

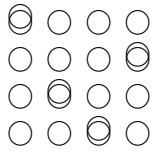
Wavelets. (See Discrete-Wavelet Transform.)

Zip. Popular software that implements the Deflate algorithm (Section 3.3) that uses a variant of LZ77 combined with static Huffman coding. It uses a 32-Kb-long sliding dictionary and a look-ahead buffer of 258 bytes. When a string is not found in the dictionary, its first symbol is emitted as a literal byte. (See also Deflate.)

Glossary (noun). An alphabetical list of technical terms in some specialized field of knowledge; usually published as an appendix to a text on that field.

—A typical dictionary definition

Solutions to Puzzles

Page 30. The diagram shows how easy it is. 

Page 47. No, because each rectangle on the chess board covers one white and one black square, but the two squares that we have removed have the same color.

Page 67. The next two integers are 28 and 102. The rule is simple but elusive. Start with (almost) any positive 2-digit integer (we somewhat arbitrarily selected 38). Multiply the two digits to obtain $3 \times 8 = 24$, then add $38 + 24$ to generate the third integer 62. Now multiply $6 \times 2 = 12$ and add $62 + 12 = 74$. Similar multiplication and addition produce the next two integers 28 and 102.

Page 76. Just me. All the others were going in the opposite direction.

Page 98. Each win increases the mount in Mr Ambler's pocket from m to $1.5m$ and each loss reduces it from n to $0.5n$. In order for him to win half the time, g, the number of games, has to be even and we denote $i = g/2$. We start with the simple case $g = 2$, where there are two possible game sequences, namely `WL` and `LW`. In the first sequence, the amount in Mr Ambler's pocket varies from a to $\frac{3}{2}a$ to $\frac{1}{2}\frac{3}{2}a$ and in the second sequence it varies from a to $\frac{1}{2}a$ to $\frac{3}{2}\frac{1}{2}a$. It is now obvious that the amount left in his pocket after i wins and i losses does not depend on the precise sequence of winnings and losses and is always $\left(\frac{1}{2}\right)^i \left(\frac{3}{2}\right)^i a = \left(\frac{3}{4}\right)^i a$. This amount is less than a, so Ambler's chance of net winning is zero.

Page 107. The schedule of the London underground is such that a Brixton-bound train always arrives at Oxford circus a minute or two after a train to Maida Vale has departed. Anyone planning to complain to London Regional Transport, should do so at `http://www.tfl.gov.uk/home.aspx`.

Page 110. The next integer is 3. The first integer of each pair is random, and the second one is the number of letters in the English name of the integer.

Page 132. Three socks.

Page 151. When each is preceded by `BREAK` it has a different meaning.

Page 179. Consider the state of the game when there are five matches left. The player whose turn it is will lose, because regardless of the number of matches he removes (between 1 and 4), his opponent will be able to remove the last match. A similar situation exists when there are 10 matches left because the player whose turn it is can remove five and leave five matches. The same argument applies to the point in the game when 15 matches are left. Thus, he who plays the first move has a simple winning strategy; remove two matches.

Page 209. This is easy. Draw two squares with a common side (five lines), and then draw a diagonal of each square.

Page 231. When fully spelled in English, the only vowel they contain is `E`.

Page 238. The letters `NIL` can be made with six matches.

Page 253. It is 265. Each integer is the sum of two consecutive squares. Thus, $1^2 + 2^2 = 5$, $3^2 + 4^2 = 25$, and so on.

Page 257. FLOUR, FLOOR, FLOOD, BLOOD, BROOD, BROAD, BREAD.

Who in the world am I? Ah, that's the great puzzle.
—Lewis Carroll

Answers to Exercises

> A bird does not sing because he has an answer, he sings because he has a song.
>
> —Chinese Proverb

Intro.1. When a software maker has a popular product it tends to come up with new versions. Typically, a user downloads the update off the maker's site anonymously. Over time, the updates become bigger and may take too long to download. This is why such updates should be supplied in compressed format. This is an example of data that is compressed once, at the source, and is decompressed once by each user downloading it. Obviously, the speed of encoder and decoder is uncritical, but efficient compression is important.

1.1. We can assume that each image row starts with a white pixel. If a row starts with, say, seven black pixels, we can prepare the run lengths 0, 7, ... and the compressor will simply ignore the run of zero white pixels.

1.2. "Cook what the cat dragged in," "my ears are turning," "sad egg," "a real hooker," "my brother's beeper," and "put all your eggs in one casket."

1.3. The following list summarizes the advantages and downsides of each approach:

- Two-passes: slow, not online, may require memory to store result of the first pass, on the other hand it achieves best possible compression.
- Training: sensitive to the choice of training documents (which may or may not reflect the characteristics of the data being compressed). Fast, because all the data symbols have been assigned variable-length codes even before the encoder starts. General purpose (this has been proved by the fax compression standard). Good performance if the training data is statistically relevant.
- Adaptive: adapts while reading and compressing the data, which is why it performs poorly on small input files. Online, generally faster than the other two methods.

1.4. 6,8,0,1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,2,2,2,2,6,1,1. The first two are the bitmap resolution (6×8). The original image occupies either 48 bits, so compression will be achieved if the resulting 25 runs can be encoded in fewer than 48 bits. Like most compression methods, RLE does not work well for small data files.

1.5. RLE of images is based on the idea that adjacent pixels tend to be identical. The last pixel of a row, however, has no reason to be identical to the first pixel of the next row.

1.6. It is not necessary if the width of a row is known a priori. For example, if image dimensions are contained in the header of the compressed image. If the eol is sent for the first row, there is no need to signal the end of a line again, since the decoder can infer the line width after decoding the first line and split the runs into lines by counting the pixels. It is possible to signal the end of a scan line without using any special symbol. The end of a line could be signalled by inserting two consecutive zeros. Since it is not possible to have two consecutive runs of zero length, the decoder may interpret two consecutive zeros as the end of a scan line.

1.7. Method (b) has two advantages as follows:

1. The paragraph following Equation (5.6) shows that a block of DCT coefficients (Section 5.5) has one large element (the DC) at its top-left corner and smaller elements (AC) elsewhere. The AC coefficients generally become smaller as we move from the top-left to the bottom-right corner and are largest on the top row and the leftmost column. Thus, scanning a block of DCT coefficients in zigzag order collects the large elements first. Notice that this order collects all the coefficients of the top row and leftmost column before it even starts collecting the elements in the bottom-right half of the block. Thus, this scan order, which is used in JPEG, transforms a two-dimensional block of DCT coefficients into a one-dimensional sequence where numbers, especially after quantization, become smaller and smaller, and include runs of zeros. Such a sequence is easy to compress with variable-length codes and RLE, which is why a zigzag scan is a key to effective lossy compression of images.

2. In a zigzag scan, every pair of pixels are near neighbors. When scanning by rows, the last element of a row is not correlated with the first element of the next row. Thus, we have to make sure that each row has an even number of elements (otherwise, the last column must be duplicated and temporarily appended to the image being scanned). The same problem exists when scanning by columns or as shown in Figure 1.12c.

Method (b) also has its own downside. It is easy to see, from Figure 1.12 or any similar grid, that the distance between diagonally-adjacent pixels is slightly larger than the distance between near neighbors on the same row or on the same column. Thus, if the distance between the centers of pixels on the same row is 1, then the distance between the centers of neighbors on a diagonal is $\sqrt{2}$, about 41% greater. If we assume that correlation is inversely proportional to distance, then a 41% greater distance translates to 41% weaker correlation and therefore worse compression.

1.8. Each of the first four rows yields the eight runs 1,1,1,2,1,1,1,eol. Rows 6 and 8 yield the four runs 0,7,1,eol each. Rows 5 and 7 yield the two runs 8,eol each. The total number of runs (including the eol's) is therefore 44.

When compressing by columns, columns 1, 3, and 6 yield the five runs 5,1,1,1,eol each. Columns 2, 4, 5, and 7 yield the six runs 0,5,1,1,1,eol each. Column 8 gives 4,4,eol, for a total of 42 runs. We conclude that this image is "balanced" with respect to rows and columns.

1.9. The straightforward answer is that the decoder doesn't know but it does not need to know. The decoder simply reads tokens and uses each offset to locate a string of text without having to know whether the string was a first or a last match.

1.10. Imagine a history book divided into chapters. One chapter may commonly employ words such as "Greek," "Athens," and "Troy," while the following chapter may have a preponderance of "Roman," "empire," and "legion." In such a document, we can expect to find more matches in the newer (recent) part of the search buffer. Now imagine a very long poem or hymn, organized in long stanzas, each followed by the same chorus. When trying to match pieces of the current chorus, the best matches may be found in the previous chorus, which may be located in the older part of the search buffer. Thus, the distribution of matches depends heavily on the type of data that is being compressed.

1.11. The next step matches the space and encodes the string ␣e.

sir␣sid	␣eastman␣easily␣	⇒	(4,1,e)
sir␣sid␣e	astman␣easily␣te	⇒	(0,0,a)

and the next one matches nothing and encodes the a.

1.12. Any two correlated pixels a and b are similar (this is what being correlated means). Thus, if the pair (a, b) is considered a point, it will be located in the xy plane on or near the 45° line $x = y$. After the rotation, the point will end up on or near the x axis, where it will have a small y coordinate, while its x coordinate will not change much (Figure 5.4). We say that the rotation squeezes the range of one of the two dimensions and slightly increases the range of the other.

1.13. This transform, like anything else in life, does not come "free" and involves a subtle cost. The original numbers were correlated, but the transform coefficients are not. The statistical cross-correlation of a set of pairs (x_i, y_i) is the sum $\sum_i x_i y_i$, and it is easy to see that the cross-correlation of our five pairs has dropped from 1729.72 before the transform to −23.0846 after it. A significant reduction!

2.1. Figure Ans.1a,b,c shows the three trees. The codes sizes for the trees are

$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$
$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$
$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

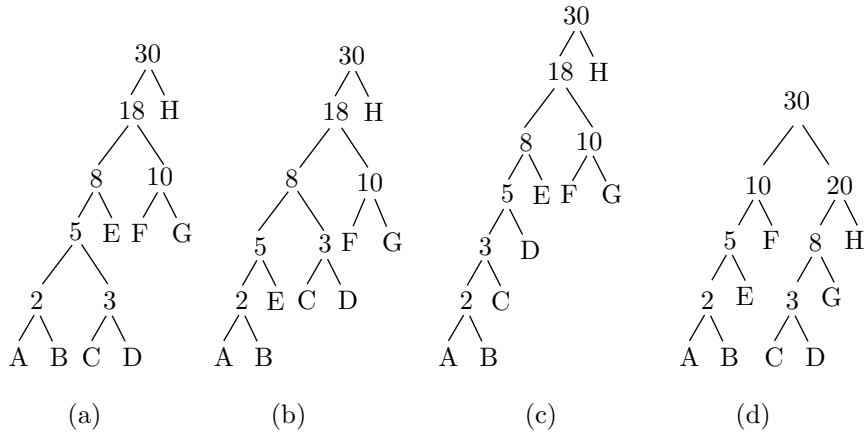


Figure Ans.1: Three Huffman Trees for Eight Symbols.

2.2. After adding symbols A, B, C, D, E, F, and G to the tree, we were left with the three symbols ABEF (with probability 10/30), CDG (with probability 8/30), and H (with probability 12/30). The two symbols with lowest probabilities were ABEF and CDG, so they had to be merged. Instead, symbols CDG and H were merged, creating a non-Huffman tree.

2.3. The second row of Table 2.2 (due to Guy Blelloch) shows a symbol whose Huffman code is three bits long, but for which $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

2.4. The explanation is simple. Imagine a large alphabet where all the symbols have (about) the same probability. Since the alphabet is large, that probability will be small, resulting in long codes. Imagine the other extreme case, where certain symbols have high probabilities (and, therefore, short codes). Since the probabilities have to add up to 1, the rest of the symbols will have low probabilities (and, therefore, long codes). We therefore see that the size of a code depends on the probability, but is indirectly affected by the size of the alphabet.

2.5. Figure Ans.2 shows Huffman codes for 5, 6, 7, and 8 symbols with equal probabilities. In the case where n is a power of 2, the codes are simply the fixed-sized ones. In other cases the codes are very close to fixed-size. This shows that symbols with equal probabilities do not benefit from variable-length codes. (This is another way of saying that random text cannot be compressed.) Table Ans.3 shows the codes, their average sizes and variances.

2.6. It increases exponentially from 2^s to $2^{s+n} = 2^s \times 2^n$.

2.7. The binary value of 127 is 01111111 and that of 128 is 10000000. Half the pixels in each bitplane will therefore be 0 and the other half, 1. In the worst case, each bitplane will be a checkerboard, i.e., will have many runs of size one. In such a case, each run requires a 1-bit code, leading to one codebit per pixel per bitplane, or eight codebits per

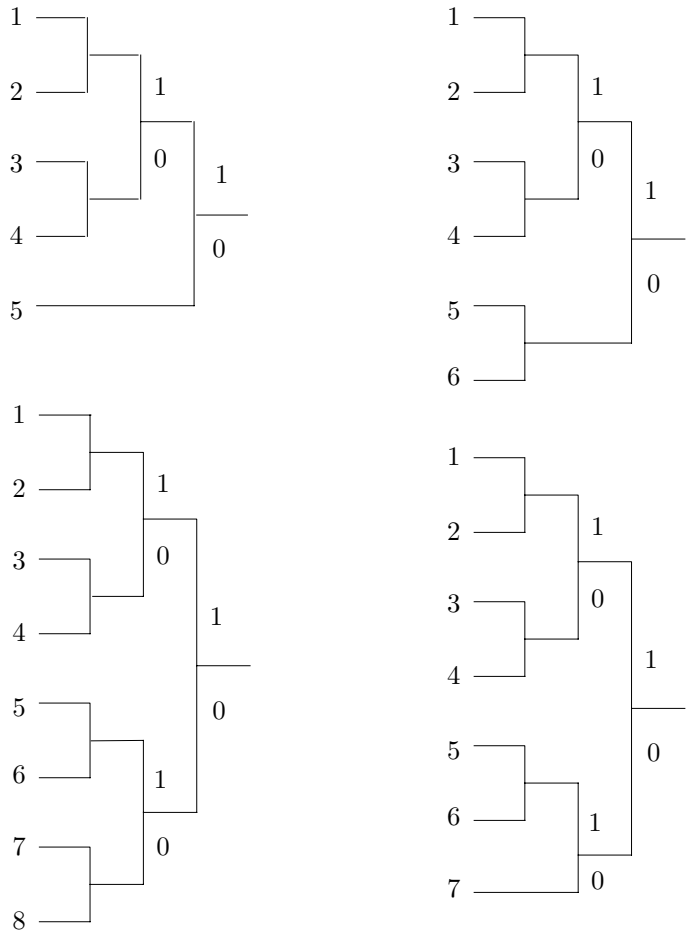


Figure Ans.2: Huffman Codes for Equal Probabilities.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Avg. size	Var.
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.143	111	110	101	100	011	010	00		2.86	0.1226
8	0.125	111	110	101	100	011	010	001	000	3	0

Table Ans.3: Huffman Codes for 5–8 Symbols.

pixel for the entire image, resulting in no compression at all. In comparison, a Huffman code for such an image requires just two codes (since there are just two pixel values) and they can be one bit each. This leads to one codebit per pixel, or a compression factor of eight.

2.8. The two trees are shown in Figure 2.13c,d. The average code size for the binary Huffman tree is

$$1\times 0.49+2\times 0.25+5\times 0.02+5\times 0.03+5\times .04+5\times 0.04+3\times 0.12=2\,\text{bits/symbol},$$

and that of the ternary tree is

$$1\times 0.26+3\times 0.02+3\times 0.03+3\times 0.04+2\times 0.04+2\times 0.12+1\times 0.49=1.34\,\text{trits/symbol}.$$

2.9. A symbol with high frequency of occurrence should be assigned a shorter code. Therefore, it has to appear high in the tree. The requirement that at each level the frequencies be sorted from left to right is artificial. In principle, it is not necessary, but it simplifies the process of updating the tree.

2.10. Figure Ans.4 shows the initial tree and how it is updated in the 11 steps (a) through (k). Notice how the *esc* symbol gets assigned different codes all the time, and how the different symbols move about in the tree and change their codes. Code 10, e.g., is the code of symbol "i" in steps (f) and (i), but is the code of "s" in steps (e) and (j). The code of a blank space is 011 in step (h), but 00 in step (k).

The final output is: "`s0i00r100␣1010000d011101000`". A total of $5\times 8+22=62$ bits. The compression ratio is thus $62/88\approx 0.7$.

2.11. Because a typical fax machine scans lines that are about 8.2 inches wide ($\approx$ 208 mm), so a blank scan line produces 1,664 consecutive white pels.

2.12. These codes are needed for cases such as example 4, where the run length is 64, 128, or any length for which a make-up code has been assigned.

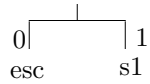
2.13. There may be fax machines (now or in the future) built for wider paper, so the Group 3 code was designed to accommodate them.

2.14. Each scan line starts with a white pel, so when the decoder inputs the next code it knows whether it is for a run of white or black pels. This is why the codes of Table 2.22 have to satisfy the prefix property in each column but not between the columns.

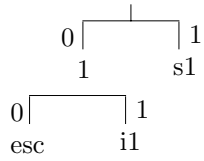
2.15. Imagine a scan line where all runs have length one (strictly alternating pels). It's easy to see that this case results in expansion. The code of a run length of one white pel is 000111, and that of one black pel is 010. Two consecutive pels of different colors are thus coded into nine bits. Since the uncoded data requires just two bits (01 or 10), the compression ratio is $9/2=4.5$ (the compressed data is 4.5 times bigger than the original data; a significant expansion).

Initial tree

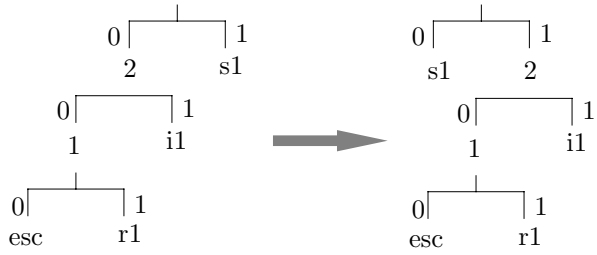
(a). Input: s. Output: ‘s’.
$esc\, s_1$



(b). Input: i. Output: 0‘i’.
$esc\, i_1\, 1\, s_1$



(c). Input: r. Output: 00‘r’.
$esc\, r_1\, 1\, i_1\, 2\, s_1 \rightarrow$
$esc\, r_1\, 1\, i_1\, s_1\, 2$



(d). Input: ␣. Output: 100‘␣’.
$esc_{\sqcup 1}\, 1\, r_1\, 2\, i_1\, s_1\, 3 \rightarrow$
$esc_{\sqcup 1}\, 1\, r_1\, s_1\, i_1\, 2\, 2$

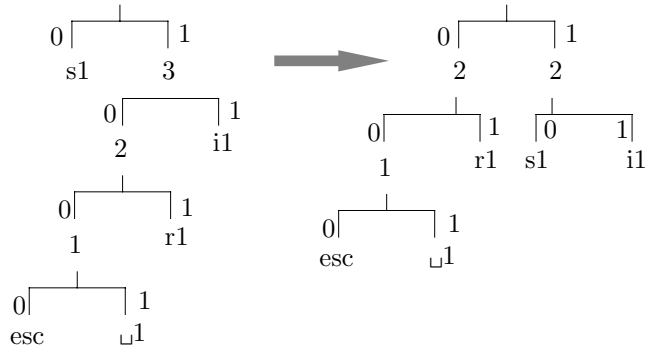
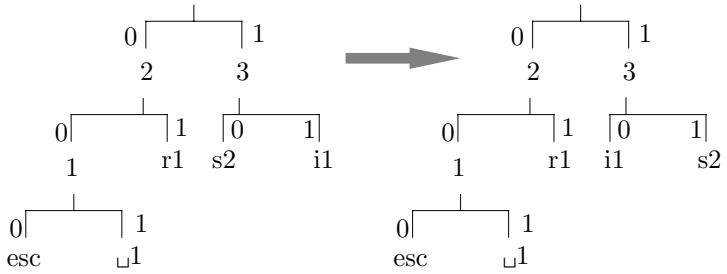
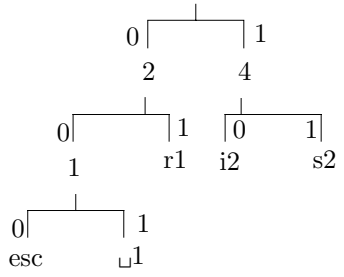


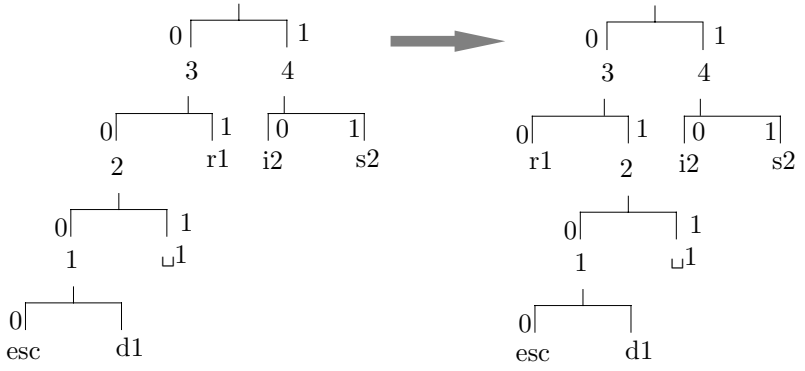
Figure Ans.4: Exercise 2.10. Part I.



(e). Input: s. Output: 10.
$esc\,␣1\,1\,r_1\,s_2\,i_1\,2\,3 \rightarrow$
$esc\,␣1\,1\,r_1\,i_1\,s_2\,2\,3$

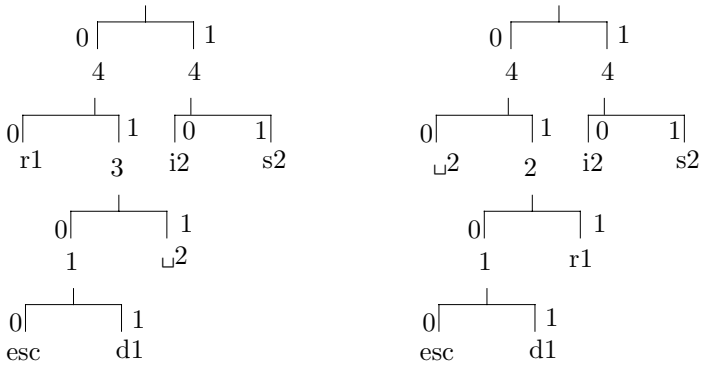


(f). Input: i. Output: 10.
$esc\,␣1\,1\,r_1\,i_2\,s_2\,2\,4$

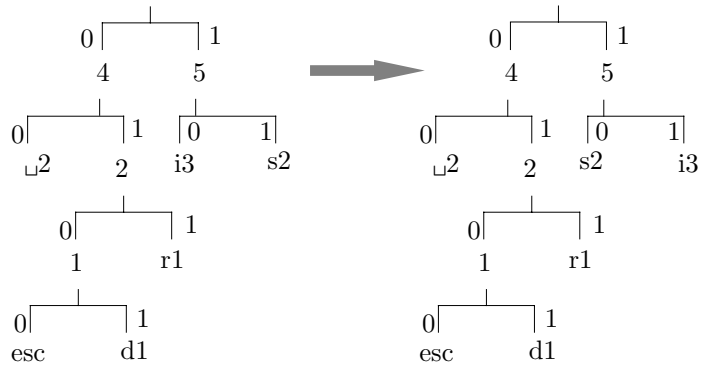


(g). Input: d. Output: 000'd'.
$esc\,d_1\,1\,␣1\,2\,r_1\,i_2\,s_2\,3\,4 \rightarrow$
$esc\,d_1\,1\,␣1\,r_1\,2\,i_2\,s_2\,3\,4$

Figure Ans.4: Exercise 2.10. Part II.

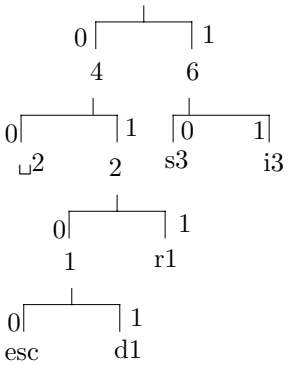


(h). Input: ␣. Output: 011.
$esc\, d_1\, 1\, \sqcup_2\, r_1\, 3\, i_2\, s_2\, 4\, 4 \rightarrow$
$esc\, d_1\, 1\, r_1\, \sqcup_2\, 2\, i_2\, s_2\, 4\, 4$

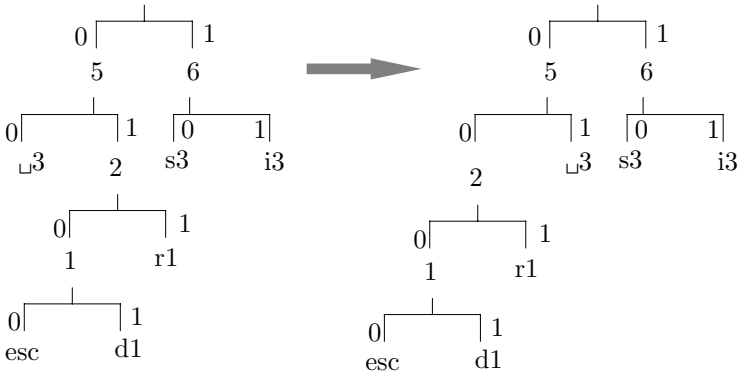


(i). Input: i. Output: 10.
$esc\, d_1\, 1\, r_1\, \sqcup_2\, 2\, i_3\, s_2\, 4\, 5 \rightarrow$
$esc\, d_1\, 1\, r_1\, \sqcup_2\, 2\, s_2\, i_3\, 4\, 5$

Figure Ans.4: Exercise 2.10. Part III.



(j). Input: s. Output: 10.
$esc\, d_1\, 1\, r_1\, \sqcup_2\, 2\, s_3\, i_3\, 4\, 6$



(k). Input: $\sqcup$. Output: 00.
$esc\, d_1\, 1\, r_1\, \sqcup_3\, 2\, s_3\, i_3\, 5\, 6 \rightarrow$
$esc\, d_1\, 1\, r_1\, 2\, \sqcup_3\, s_3\, i_3\, 5\, 6$

Figure Ans.4: Exercise 2.10. Part IV.

3.1. (1) The size of the output is $N[48 - 28P] = N[48 - 25.2] = 22.8N$. The size of the input is, as before, $40N$. The compression factor is therefore $40/22.8 \approx 1.75$. (2) Maximum compression is obtained when the output size $N[48 - 28P]$, is minimum, which occurs when $P = 1$. The compression factor in this case is $40N/N[48 - 28P] = 2$.

3.2. This is straightforward. The remaining steps are listed in Table Ans.5

Dictionary		Token	Dictionary		Token
15	␣t	(4, t)	21	␣si	(19,i)
16	e	(0, e)	22	c	(0, c)
17	as	(8, s)	23	k	(0, k)
18	es	(16,s)	24	␣se	(19,e)
19	␣s	(4, s)	25	al	(8, l)
20	ea	(4, a)	26	s(eof)	(1, (eof))

Table Ans.5: Next 12 Encoding Steps in the LZ78 Example.

3.3. Table Ans.6 summarizes the steps. The output emitted by the encoder is

97 (a), 108 (l), 102 (f), 32 (␣), 101 (e), 97 (a), 116 (t), 115 (s), 32 (␣), 256 (al), 102 (f), 265 (alf), 97 (a),

and the following new entries are added to the dictionary

(256: al), (257: lf), (258: f), (259: ␣e), (260: ea), (261: at), (262: ts), (263: s), (264: ␣a), (265: alf), (266: fa), (267: alfa).

I	In dict?	New entry	Output	I	In dict?	New entry	Output
a	Y			s	N	263-s	115 (s)
al	N	256-al	97 (a)	␣	Y		
l	Y			␣a	N	264-␣a	32 (␣)
lf	N	257-lf	108 (l)	a	Y		
f	Y			al	Y		
f	N	258-f	102 (f)	alf	N	265-alf	256 (al)
␣	Y			f	Y		
␣e	N	259-␣e	32 (w)	fa	N	266-fa	102 (f)
e	Y			a	Y		
ea	N	260-ea	101 (e)	al	Y		
a	Y			alf	Y		
at	N	261-at	97 (a)	alfa	N	267-alfa	265 (alf)
t	Y			a	Y		
ts	N	262-ts	116 (t)	a,eof	N		97 (a)
s	Y						

Table Ans.6: LZW Encoding of alf eats alfalfa.

3.4. The encoder inputs the first `a` into I, searches and finds `a` in the dictionary. It inputs the next `a` but finds that Ix, which is now `aa`, is not in the dictionary. The encoder thus adds string `aa` to the dictionary as entry 256 and outputs the token 97 (`a`). Variable I is initialized to the second `a`. The third `a` is input, so Ix is the string `aa`, which is now in the dictionary. I becomes this string, and the fourth `a` is input. Ix is now `aaa` which is not in the dictionary. The encoder thus adds string `aaa` to the dictionary as entry 257 and outputs 256 (`aa`). I is initialized to the fourth `a`. Continuing this process is straightforward.

The result is that strings `aa`, `aaa`, `aaaa`, ... are added to the dictionary as entries 256, 257, 258, ..., and the output is

$$97\ (\texttt{a}),\ 256\ (\texttt{aa}),\ 257\ (\texttt{aaa}),\ 258\ (\texttt{aaaa}),\ \ldots$$

The output consists of pointers pointing to longer and longer strings of `a`'s. The first k pointers thus point at strings whose total length is $1+2+\cdots+k=(k+k^2)/2$.

Assuming input data that consists of one million `a`'s, we can find the size of the compressed output stream by solving the quadratic equation $(k+k^2)/2 = 1{,}000{,}000$ for the unknown k. The solution is $k \approx 1{,}414$. The original, 8-million bit input is thus compressed into 1,414 pointers, each at least 9-bit (and in practice, probably 16-bit) long. The compression factor is thus either $8\text{M}/(1414\times 9) \approx 628.6$ or $8\text{M}/(1414\times 16) \approx 353.6$.

This is an impressive result but such input files are rare (notice that this particular input can best be compressed by generating an output file containing just "1,000,000 `a`", and without using LZW).

3.5. We simply follow the decoding steps described in the text. The results are:
1. Input 97. This is in the dictionary so set I=`a` and output `a`. String `ax` needs to be saved in the dictionary but x is still unknown.
2. Input 108. This is in the dictionary so set J=`l` and output `l`. Save `al` in entry 256. Set I=`l`.
3. Input 102. This is in the dictionary so set J=`f` and output `f`. Save `lf` in entry 257. Set I=`f`.
4. Input 32. This is in the dictionary so set J=␣ and output ␣. Save `f␣` in entry 258. Set I=␣.
5. Input 101. This is in the dictionary so set J=`e` and output `e`. Save ␣`e` in entry 259. Set I=`e`.
6. Input 97. This is in the dictionary so set J=`a` and output `a`. Save `ea` in entry 260. Set I=`a`.
7. Input 116. This is in the dictionary so set J=`t` and output `t`. Save `at` in entry 261. Set I=`t`.
8. Input 115. This is in the dictionary so set J=`s` and output `s`. Save `ts` in entry 262. Set I=`t`.
9. Input 32. This is in the dictionary so set J=␣ and output ␣. Save `s␣` in entry 263. Set I=␣.
10. Input 256. This is in the dictionary so set J=`al` and output `al`. Save ␣`a` in entry 264. Set I=`al`.
11. Input 102. This is in the dictionary so set J=`f` and output `f`. Save `alf` in entry 265. Set I=`f`.

12. Input 265. This has just been saved in the dictionary so set J=`alf` and output `alf`. Save `fa` in dictionary entry 266. Set I=`alf`.
13. Input 97. This is in the dictionary so set J=`a` and output `a`. Save `alfa` in entry 267 (even though it will never be used). Set I=`a`.
14. Read eof. Stop.

3.6. Assume that the dictionary is initialized to just the two entries (1: `a`) and (2: `b`). The encoder outputs

1 (`a`), 2 (`b`), 3 (`ab`), 5 (`aba`), 4 (`ba`), 7 (`bab`), 6 (`abab`), 9 (`ababa`), 8 (`baba`), ...

and adds the new entries (3: `ab`), (4: `ba`), (5: `aba`), (6: `abab`), (7: `bab`), (8: `baba`), (9: `ababa`), (10: `ababab`), (11: `babab`), ... to the dictionary. This regular behavior can be analyzed and the kth output pointer and dictionary entry predicted, but the effort is probably not worth it.

3.7. The answer to Exercise 3.4 shows the relation between the size of the compressed file and the size of the largest dictionary string for the "worst case" situation (input that creates the longest strings). For a 1 Mbyte input file, there will be 1,414 strings in the dictionary, the largest of which is 1,414 symbols long.

3.8. A simple alternative it to store the 256 8-bit values in the first (rightmost) 256 bytes of the sliding window and keep these locations fixed whenever the window is shifted.

3.9. No. Besides the slowdown caused by evaluating a much larger number of matching candidates, sending the offset of a match is expensive and matches of length shorter than 3 would probably increase the size of the compressed file.

4.1. Table Ans.7 shows the steps of encoding the string $a_2a_2a_2a_2$. Because of the high probability of a_2 the low and high variables start at very different values and approach each other slowly.

a_2	$0.0 + (1.0 - 0.0) \times 0.023162 = 0.023162$
	$0.0 + (1.0 - 0.0) \times 0.998162 = 0.998162$
a_2	$0.023162 + 0.975 \times 0.023162 = 0.04574495$
	$0.023162 + 0.975 \times 0.998162 = 0.99636995$
a_2	$0.04574495 + 0.950625 \times 0.023162 = 0.06776322625$
	$0.04574495 + 0.950625 \times 0.998162 = 0.99462270125$
a_2	$0.06776322625 + 0.926859375 \times 0.023162 = 0.08923124309375$
	$0.06776322625 + 0.926859375 \times 0.998162 = 0.99291913371875$

Table Ans.7: Encoding the String $a_2a_2a_2a_2$.

4.2. It can be written either as 0.1000... or 0.0111... .

4.3. In practice, the eof symbol has to be included in the original table of frequencies and probabilities. This symbol is the last to be encoded, and the decoder stops when it detects an eof.

4.4. The encoding steps are simple (see first example on page 124). We start with the interval $[0,1)$. The first symbol a_2 reduces the interval to $[0.4,0.9)$. The second one, to $[0.6,0.85)$, the third one to $[0.7,0.825)$ and the eof symbol to $[0.8125,0.8250)$. The approximate binary values of the last interval are 0.1101000000 and 0.1101001100, so we select the 7-bit number 1101000 as our code.

The probability of $a_2a_2a_2$eof is $(0.5)^3\times 0.1 = 0.0125$, but since $-\log_2 0.0125 \approx 6.322$ it follows that the practical minimum code size is 7 bits.

5.1. An image with no redundancy is not always random. The definition of redundancy tells us that an image where each color appears with the same frequency has no redundancy (statistically) yet it is not necessarily random and may even be interesting and/or useful.

5.2. The results are shown in Table Ans.8 together with the Matlab code used to calculate it.

43210	Gray	43210	Gray	43210	Gray	43210	Gray
00000	00000	0**1000**	01100	**10000**	11000	1**1000**	10100
00001	00001	0100**1**	01101	1000**1**	11001	1100**1**	10101
000**10**	00011	010**10**	01111	100**10**	11011	110**10**	10111
00011	00010	0101**1**	01110	1001**1**	11010	1101**1**	10110
00**100**	00110	01**100**	01010	10**100**	11110	11**100**	10010
0010**1**	00111	0110**1**	01011	1010**1**	11111	1110**1**	10011
001**10**	00101	011**10**	01001	101**10**	11101	111**10**	10001
00111	00100	01111	01000	10111	11100	11111	10000

Table Ans.8: First 32 Binary and Gray Codes.

```
a=linspace(0,31,32); b=bitshift(a,-1);
b=bitxor(a,b); dec2bin(b)
```

Code for Table Ans.8.

5.3. The code of Figure Ans.9 yields the coordinates of the rotated points

$$(7.071,0),\ (9.19,0.7071),\ (17.9,0.78),\ (33.9,1.41),\ (43.13,-2.12)$$

(notice how all the y coordinates are small numbers) and shows that the cross-correlation drops from 1729.72 before the rotation to -23.0846 after it. A significant reduction!

5.4. Numerical precision aside, the resulting quantization errors (Table Ans.10) computed in (1) and (2) should be the same (approximately 0.011 for the finest quantization, 1.1778 for the second best, and 1.244 for the coarsest). This feature is a useful consequence of the linearity of the IDCT and it can be used to estimate the quantization error in the quantized data without having to perform an IDCT.

```
p={{5,5},{6, 7},{12.1,13.2},{23,25},{32,29}};
rot={{0.7071,-0.7071},{0.7071,0.7071}};
Sum[p[[i,1]]p[[i,2]], {i,5}]
q=p.rot
Sum[q[[i,1]]q[[i,2]], {i,5}]
```

Figure Ans.9: Code for Rotating Five Points.

Items	12.000000	10.000000	8.000000	10.000000	12.000000	10.000000	8.000000	11.000000	
DCT	28.637500	0.571202	0.461940	1.757000	3.181960	-1.729560	0.191342	-0.308709	
Q1	28.600000	0.600000	0.500000	1.800000	3.200000	-1.800000	0.200000	-0.300000	
Q2	28.000000	1.000000	1.000000	2.000000	3.000000	-2.000000	0.000000	0.000000	
Q3	28.000000	0.000000	0.000000	2.000000	3.000000	-2.000000	0.000000	0.000000	
E1	0.0014062	0.0008293	0.0014486	0.0018490	0.0003254	0.0049618	0.0000750	0.0000758	0.0109712
E2	0.4064062	0.1838677	0.2895086	0.0590490	0.0331094	0.0731378	0.0366118	0.0953012	1.1769918
E3	0.4064062	0.3262717	0.2133886	0.0590490	0.0331094	0.0731378	0.0366118	0.0953012	1.2432758
IDCT1	12.025400	10.023300	7.960540	9.930970	12.016400	9.993210	7.943540	10.998900	
IDCT2	12.188300	10.231500	7.749310	9.208630	11.787600	9.545490	7.828650	10.655700	
IDCT3	11.236000	9.624430	7.662860	9.573020	12.347100	10.014600	8.053040	10.684200	
E1	0.0006452	0.0005429	0.0015571	0.0047651	0.0002690	0.0000461	0.0031877	0.0000012	0.0110143
E2	0.0354569	0.0535923	0.0628455	0.6262665	0.0451138	0.2065793	0.0293608	0.1185425	1.1777575
E3	0.5836960	0.1410528	0.1136634	0.1823119	0.1204784	0.0002132	0.0028132	0.0997296	1.2439586

Table Ans.10: Results of a DCT Quantization Error Experiment.

5.5. The *Mathematica* code of Figure 5.7 produces the eight coefficients 140, −71, 0, −7, 0, −2, 0, and 0. We now quantize this set coarsely by clearing the last two nonzero weights −7 and −2, When the IDCT is applied to the sequence 140, −71, 0, 0, 0, 0, 0, 0, it produces 15, 20, 30, 43, 56, 69, 79, and 84. These are not identical to the original values, but the maximum difference is only 4; an excellent result considering that only two of the eight DCT coefficients are nonzero.

5.6. The eight values in the top row are very similar (the differences between them are either 2 or 3). Each of the other rows is obtained as a right-circular shift of the preceding row.

5.7. It is obvious that such a block can be represented as a linear combination of the patterns in the leftmost column of Figure 5.24. The actual calculation yields the eight weights 4, 0.72, 0, 0.85, 0, 1.27, 0, and 3.62 for the patterns of this column. The other 56 weights are zero or very close to zero.

5.8. First figure out the zigzag path manually, then record it in an array `zz` of structures, where each structure contains a pair of coordinates for the path as shown, e.g., in Figure Ans.11.

If the two components of a structure are `zz.r` and `zz.c`, then the zigzag traversal can be done by a loop of the form:

```
for (i=0; i<64; i++){
```

(0,0)	(0,1)	(1,0)	(2,0)	(1,1)	(0,2)	(0,3)	(1,2)
(2,1)	(3,0)	(4,0)	(3,1)	(2,2)	(1,3)	(0,4)	(0,5)
(1,4)	(2,3)	(3,2)	(4,1)	(5,0)	(6,0)	(5,1)	(4,2)
(3,3)	(2,4)	(1,5)	(0,6)	(0,7)	(1,6)	(2,5)	(3,4)
(4,3)	(5,2)	(6,1)	(7,0)	(7,1)	(6,2)	(5,3)	(4,4)
(3,5)	(2,6)	(1,7)	(2,7)	(3,6)	(4,5)	(5,4)	(6,3)
(7,2)	(7,3)	(6,4)	(5,5)	(4,6)	(3,7)	(4,7)	(5,6)
(6,5)	(7,4)	(7,5)	(6,6)	(5,7)	(6,7)	(7,6)	(7,7)

Figure Ans.11: Coordinates for the Zigzag Path.

```
row:=zz[i].r; col:=zz[i].c
...data_unit[row][col]...}
```

5.9. The third DC difference, 5, is located in row 3 column 5, so it is encoded as 1110|101.

5.10. Thirteen consecutive zeros precede this coefficient, so Z = 13. The coefficient itself is found in Table 5.32 in row 1, column 0, so R = 1 and C = 0. Assuming that the Huffman code in position (R, Z) = (1, 13) of Table 5.33 is 1110101, the final code emitted for 1 is 1110101|0.

5.11. Figure Ans.12a shows a simple, 8×8 image with one diagonal line above the main diagonal. Figure Ans.12b,c shows the first two steps in its pyramid decomposition. It is obvious that the transform coefficients in the bottom-right subband (HH) indicate a diagonal artifact located above the main diagonal. It is also easy to see that subband LL is a low-resolution version of the original image.

12	16	12	12	12	12	12	12
12	12	16	12	12	12	12	12
12	12	12	16	12	12	12	12
12	12	12	12	16	12	12	12
12	12	12	12	12	16	12	12
12	12	12	12	12	12	16	12
12	12	12	12	12	12	12	16
12	12	12	12	12	12	12	12

(a)

14	12	12	12	4	0	0	0
12	14	12	12	0	4	0	0
12	14	12	12	0	4	0	0
12	12	14	12	0	0	4	0
12	12	14	12	0	0	4	0
12	12	12	14	0	0	0	4
12	12	12	14	0	0	0	4
12	12	12	12	0	0	0	0

(b)

13	13	12	12	2	2	0	0
12	13	13	12	0	2	2	0
12	12	13	13	0	0	2	2
12	12	12	13	0	0	0	2
2	2	0	0	4	4	0	0
0	2	2	0	0	4	4	0
0	0	2	2	0	0	4	4
0	0	0	2	0	0	0	4

(c)

Figure Ans.12: The Subband Decomposition of a Diagonal Line.

5.12. The average can easily be calculated. It turns out to be 131.375, which is exactly 1/8 of 1051. The reason the top-left transform coefficient is eight times the average is that the Matlab code that did the calculations uses $\sqrt{2}$ instead of 2 (see function `individ(n)` in Figure 5.51).

5.13. Figure Ans.13a–c shows the results of reconstruction from 3277, 1639, and 820 coefficients, respectively. Despite the heavy loss of wavelet coefficients, only a very small loss of image quality is noticeable. The number of wavelet coefficients is, of course, the same as the image resolution $128\times128 = 16{,}384$. Using 820 out of 16,384 coefficients corresponds to discarding 95% of the smallest of the transform coefficients (notice, however, that some of the coefficients were originally zero, so the actual loss may amount to less than 95%).

5.14. The Matlab code of Figure Ans.14 calculates W as the product of the three matrices A_1, A_2, and A_3 and computes the 8×8 matrix of transform coefficients. Notice that the top-left value 131.375 is the average of all the 64 image pixels.

```
clear
a1=[1/2 1/2 0 0 0 0 0 0; 0 0 1/2 1/2 0 0 0 0;
 0 0 0 0 1/2 1/2 0 0; 0 0 0 0 0 0 1/2 1/2;
 1/2 -1/2 0 0 0 0 0 0; 0 0 1/2 -1/2 0 0 0 0;
 0 0 0 0 1/2 -1/2 0 0; 0 0 0 0 0 0 1/2 -1/2];
% a1*[255; 224; 192; 159; 127; 95; 63; 32];
a2=[1/2 1/2 0 0 0 0 0 0; 0 0 1/2 1/2 0 0 0 0;
 1/2 -1/2 0 0 0 0 0 0; 0 0 1/2 -1/2 0 0 0 0;
 0 0 0 0 1 0 0 0; 0 0 0 0 0 1 0 0;
 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0 1];
a3=[1/2 1/2 0 0 0 0 0 0; 1/2 -1/2 0 0 0 0 0 0;
 0 0 1 0 0 0 0 0; 0 0 0 1 0 0 0 0;
 0 0 0 0 1 0 0 0; 0 0 0 0 0 1 0 0;
 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0 1];
w=a3*a2*a1;
dim=8; fid=fopen('8x8','r');
img=fread(fid,[dim,dim])'; fclose(fid);
w*img*w' % Result of the transform
```

131.375	4.250	−7.875	−0.125	−0.25	−15.5	0	−0.25
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
12.000	59.875	39.875	31.875	15.75	32.0	16	15.75
12.000	59.875	39.875	31.875	15.75	32.0	16	15.75
12.000	59.875	39.875	31.875	15.75	32.0	16	15.75
12.000	59.875	39.875	31.875	15.75	32.0	16	15.75

Figure Ans.14: Code and Results for the Calculation of Matrix W and Transform $W\cdot I\cdot W^T$.

5.15. Figure Ans.15 lists the Matlab code of the inverse wavelet transform function `iwt1(wc,coarse,filter)` and a test.

5.16. The simple equation $10\times2^{20}\times8 = (500x)\times(500x)\times8$ is solved to yield $x^2 = 40$ square inches. If the card is square, it is approximately 6.32 inches on a side. Such

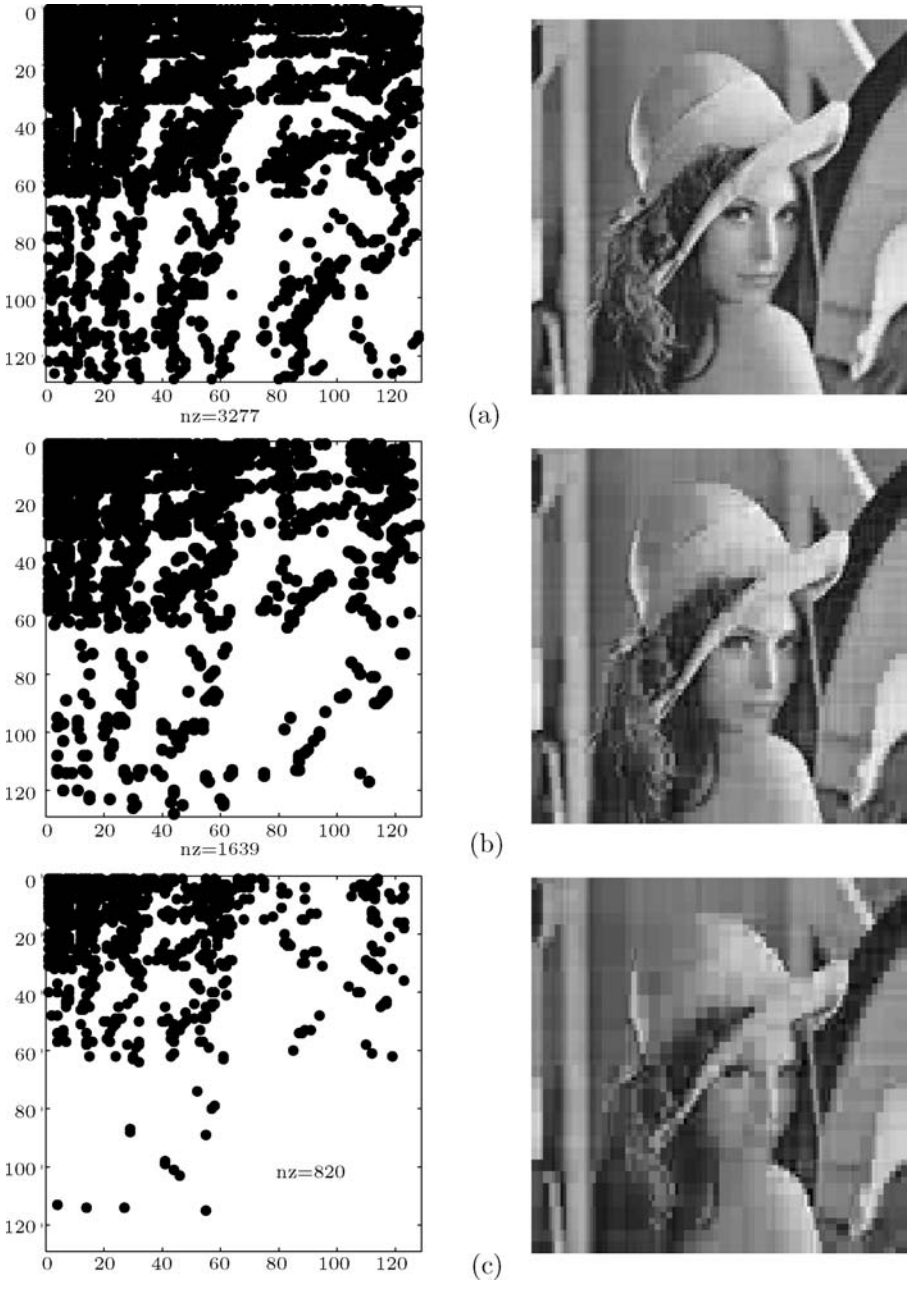


Figure Ans.13: Three Lossy Reconstructions of the 128×128 Lena Image.

```
function dat=iwt1(wc,coarse,filter)
% Inverse Discrete Wavelet Transform
dat=wc(1:2^coarse);
n=length(wc); j=log2(n);
for i=coarse:j-1
 dat=ILoPass(dat,filter)+ ...
  IHiPass(wc((2^(i)+1):(2^(i+1))),filter);
end

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;

function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;
```

A simple test of `iwt1` is

```
n=16; t=(1:n)./n;
dat=sin(2*pi*t)
filt=[0.4830 0.8365 0.2241 -0.1294];
wc=fwt1(dat,1,filt)
rec=iwt1(wc,1,filt)
```

Figure Ans.15: Code for the 1D Inverse Discrete Wavelet Transform.

a card has 10 rolled impressions (about 1.5×1.5 each), two plain impressions of the thumbs (about 0.875×1.875 each), and simultaneous impressions of both hands (about 3.125×1.875 each). All the dimensions are in inches.

6.1. An average book may have 60 characters per line, 45 lines per page, and 400 pages. This comes to $60 \times 45 \times 400 = 1{,}080{,}000$ characters, requiring one byte of storage each.

6.2. Audio samples are normally 16-bit unsigned integers, so they are in the interval $[1, 2^{16} - 1]$. The residuals tend to be small numbers, but can also be large. In principle, a residual can be as large as 2^{16} and as small as -2^{16}. Thus, enough variable-length codes are needed to encode all the residuals. The first audio sample of an input file is not subtracted from anything but is encoded with the same variable-length code used to encode the residuals and such codes exist even for the largest samples because they are needed for the residuals.

6.3. Such an experiment should be repeated with several persons, preferably of different ages. The person should be placed in a sound insulated chamber, and a pure tone of frequency f should be played. The amplitude of the tone should be gradually increased from zero until the person can just barely hear it. If this happens at a decibel value d, point (d, f) should be plotted. This should be repeated for many frequencies until a graph similar to Figure 6.2a is obtained.

6.4. We first select identical items. If all $s(t-i)$ equal s, Equation (6.5) yields the same s. Next, we select values on a straight line. Given the four values a, $a+2$, $a+4$, and $a+6$, Equation (6.5) yields $a+8$, the next linear value. Finally, we select values roughly equally-spaced on a circle. The y coordinates of points on the first quadrant of a circle can be computed by $y=\sqrt{r^2-x^2}$. We select the four points with x coordinates 0, $0.08r$, $0.16r$, and $0.24r$, compute their y coordinates for $r=10$, and substitute them in Equation (6.5). The result is 9.96926, compared to the actual y coordinate for $x=0.32r$ which is $\sqrt{r^2-(0.32r)^2}=9.47418$, a difference of about 5%. The code that did the computations is shown in Figure Ans.16.

```
(* Points on a circle. Used in exercise to check
 4th-order prediction in FLAC *)
r = 10;
ci[x_] := Sqrt[100 - x^2];
ci[0.32r]
4ci[0] - 6ci[0.08r] + 4ci[0.16r] - ci[0.24r]
```

Figure Ans.16: Code for Checking Fourth-Order Prediction.

7.1. Because the original string S can be reconstructed from L but not from F.

7.2. A direct application of Equation (7.1) eight more times produces:

```
S[10-1-2]=L[T^2[I]]=L[T[T^1[I]]]=L[T[7]]=L[6]=i;
S[10-1-3]=L[T^3[I]]=L[T[T^2[I]]]=L[T[6]]=L[2]=m;
S[10-1-4]=L[T^4[I]]=L[T[T^3[I]]]=L[T[2]]=L[3]=␣;
S[10-1-5]=L[T^5[I]]=L[T[T^4[I]]]=L[T[3]]=L[0]=s;
S[10-1-6]=L[T^6[I]]=L[T[T^5[I]]]=L[T[0]]=L[4]=s;
S[10-1-7]=L[T^7[I]]=L[T[T^6[I]]]=L[T[4]]=L[5]=i;
S[10-1-8]=L[T^8[I]]=L[T[T^7[I]]]=L[T[5]]=L[1]=w;
S[10-1-9]=L[T^9[I]]=L[T[T^8[I]]]=L[T[1]]=L[9]=s;
```

The original string `swiss␣miss` is indeed reproduced in S from right to left.

7.3. Figure Ans.17 shows the rotations of S and the sorted matrix. The last column, L of Ans.17b happens to be identical to S, so S=L=`sssssssssh`. Since A=(`s`,`h`), a move-to-front compression of L yields C = $(1,0,0,0,0,0,0,0,0,1)$. Since C contains just the two values 0 and 1, they can serve as their own Huffman codes, so the final result is 1000000001, 1 bit per character!

sssssssssh	hsssssssss
sssssssshs	shsssssssss
sssssssshss	sshsssssss
ssssssshsss	ssshssssss
sssssshssss	sssshsssss
ssssshsssss	ssssshssss
sssshssssss	sssssshsss
ssshssssss	ssssssshss
sshssssssss	sssssssshs
shsssssssss	ssssssssh
hsssssssss	
(a)	(b)

Figure Ans.17: Permutations of "sssssssssh".

7.4. The encoder starts at T[0], which contains 5. The first element of L is thus the last symbol of permutation 5. This permutation starts at position 5 of S, so its last element is in position 4. The encoder thus has to go through symbols S[T[i − 1]] for $i = 0, \ldots, n-1$, where the notation $i-1$ should be interpreted cyclically (i.e., $0-1$ should be $n-1$). As each symbol S[T[i − 1]] is found, it is compressed using move-to-front. The value of I is the position where T contains 0. In our example, T[8]=0, so I=8.

I don't have any solution, but I certainly admire the problem.
—Ashleigh Brilliant

Index

The index caters to those who have already read the book and want to locate a familiar item, as well as to those new to the book who are looking for a particular topic. I have included any terms that may occur to a reader interested in any of the topics discussed in the book (even topics that are just mentioned in passing). As a result, even a quick glance over the index gives the reader an idea of the terms and topics included in the book. Notice that the index items "data compression" and "image compression" are very general.

I have attempted to make the index items as complete as possible, including middle names and dates. Any errors and omissions brought to my attention are welcome. They will be added to the errata list and will be included in any future editions.

Any inaccuracies in this index may be explained by the fact that it has been sorted with the help of a computer.
—D. Knuth

Colophon

This book was written during the short period February through September 2007. The book was designed by the author and was typeset by him with the TEX typesetting system developed by Donald Knuth. The text and the tables were done with Textures, a TEX implementation for the Macintosh. The figures were generated in Adobe Illustrator, also on the Macintosh. Figures with calculations were computed first in *Mathematica* or Matlab, and then "polished" in Illustrator.

The CM set of fonts consists of 75 fonts that are described in Knuth's *Computer Modern Typefaces* (Volume E of the Computers and Typesetting series) as well as the "line," "circle," and symbol fonts associated with LaTeX.

Knuth started his research in computerized typesetting in 1975 and developed the fonts as part of the huge TEX project. The fonts were initially called AM (American Modern) and were later improved, to become the familiar CM fonts, released in 1985.

The Type 1 versions of the Computer Modern fonts (1990) and AMSFonts (1992) were produced by Blue Sky Research and Y&Y, who published the fonts as part of their implementations of TEX.

Character outlines for the CM fonts were derived from high-resolution METAFONT-generated character bitmaps by *ScanLab* from Projective Solutions, applied and corrected by Blue Sky Research. The outlines for the AMS Euler fonts were derived algorithmically from METAFONT code using tools developed by Y&Y. Character- and font-level hints were programmed using software from Y&Y, with extensive manual labor by programmers from both companies.

In 1988, Blue Sky Research released the PostScript Type 3 versions of the CM fonts.

The CMMI* fonts were revised in 1996 to conform to Knuth's changes to the Greek delta and arrow characters.

In 1997, the type-1 CM fonts became public domain.

Life shrinks or expands in proportion to one's courage.

—Anais Nin