Data Compression

Fourth Class

Computer Science

Department, CSW

Definition of Data Compression:

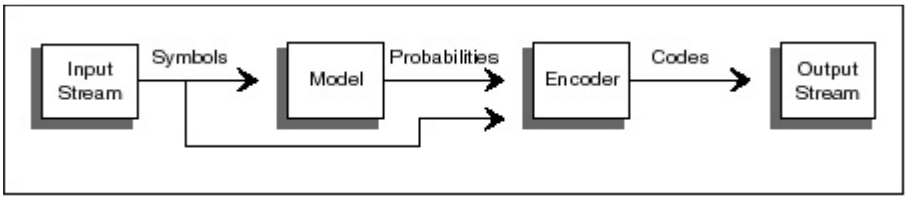
Data compression refers to the process of encoding information with the explicit purpose ofminimizing its memory/transmission capacity requirements.
Despite the exponential growth inmemory and transmission capacity, many high-bandwidth applications, such as digital storage andtransmission of video, would not be possible without compression.The goal of this course is to give graduate students a conceptual understanding, and hands-onexperience, of the state-of-the-art compression algorithms and approaches. These include bothlossless and lossy compression techniques with an emphasis on widely deployed, standardizedcoding schemes.

The discipline of data compression has its origins in the 1950s and 1960s and has experienced rapid growth in the 1980s and 1990s. Currently, data compression is a vastfield encompassing many approaches and techniques.
The input data stream is also referred as the source stream or the original raw data, the output data stream is referred as the output, the bitstream,or the compressed stream.

Data compression is popular for two reasons:
(1) People like to accumulate data in one place and hate to throw anything away. No matter how big a storage device one has, sooner or later it is going to overflow.Data compression seems useful because it delaysthis inevitability.
(2) People hate to wait a long time for data transfers.

The source can be ***memoryless***, or it can have ***memory***. In the former case,each symbol is independent of its predecessors. In the latter case, each symbol dependson some of its predecessors and, perhaps, also on its successors, so they are correlated.

Data Compression = Modeling + Coding:
In general, data compression consists of taking a stream of symbols and transforming them into codes. If the compression is effective, the resulting stream of codes will be smaller than the original symbols. The decision to output a certain code for a certain symbol or set of symbols is based on a model. The model is simply a collection of dataand rules used to process input symbols and determine which code(s) to output.



A Statistical Model with a Huffman Encoder.

Data Compression motivates Whom?

Typical examples of application areas that are relevant to and motivated bydata compression include

- Personal communication systems such asvoice mail and telephony systems.
-Computer systems such as memory structures, disks and tapes.
- Mobile computing.
-Distributed computer systems.
-Computer networks, especially the Internet.
-Multimedia evolution, imaging, signal processing.
-Image archival and videoconferencing.
-Digital and satellite TV.

Objectives of Data Compression:

By 'compressing data', we actually meanderiving techniques or, more specifically, designing efficient algorithms to:

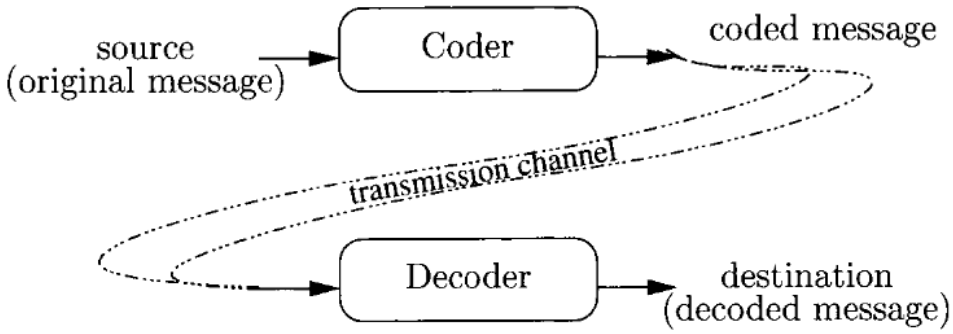
-Represent data in a less redundant fashion.
-Remove the redundancy in data.
-Implement compression algorithms, including both compression and decompression.

Data Compression Intrinsic Idea:

The general question to ask here wouldbe, for example, given a string s, what is the alternative sequence of symbolswhich takes less storage space? The solutions to the compression problems wouldthen be the compression algorithms that will derive an alternative sequence of symbols which contains fewer number of bits in total, plus the decompressionalgorithms to recoverthe original string.

Decompression:

Any compression algorithm will not work unless a means of decompression is alsoprovided due to the nature of data compression. When compression algorithmsare discussed in general, the word compression alone actually implies the contextof both compression and decompression.



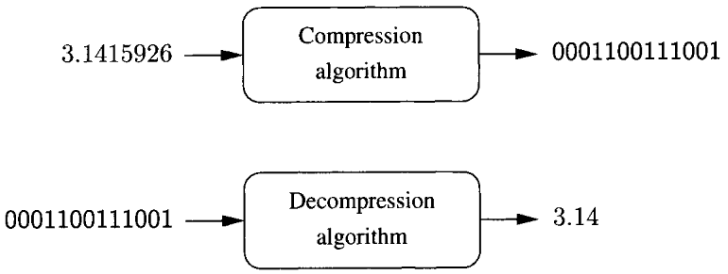
Coder (compressor) and decoder (decompressor)

Classification of Compression Techniques:

Data-compression techniques can be divided into two major families; lossy and lossless.

- **Lossy Compression (Irreversible Compression)**

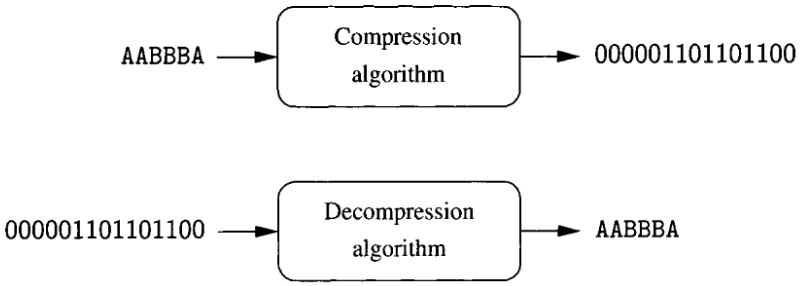
Lossy data compression concedes a certain loss of accuracy in exchange for greatlyincreased compression. It is not possible to reconstruct the original exactly from the compressed version.Lossy compression proves effective when applied to graphicsimages and digitized voice. By their very nature, these digitized representations of analogphenomena are not perfect to begin with, so the idea of output and input not matchingexactly is a little more acceptable. Lossy compression is called irreversible compression since it is impossible torecover the original data exactly by decompression.



Lossy Compression Algorithms

- **Lossless Compression**

A compression approach is lossless only if it is possible to exactly reconstruct theoriginal data from the compressed version. There is no loss of any informationduring the compression process.



Lossless Compression Algorithms

Terms to Know:

(1) A code is a symbol that stands for another symbol.

(2) An adaptive method examines the raw data and modifiesits operations and/or its parameters accordingly. An example is the adaptive Huffmanmethod.

(3) A nonadaptive compression method is rigid and does not modify its operations, itsparameters, or its tables in response to the particular data being compressed. Sucha method is best used to compress data that is all of a single type. Examples arethe Group 3 and Group 4 methods for facsimile compression.

(4) Asemiadaptive uses a 2-pass algorithm, where thefirst pass reads the input stream to collect statistics on the data to be compressed, andthe second pass does the actual compressing using parameters set by the first pass.

(5) Locallyadaptive, meaning it adapts itself to local conditions in the input stream and varies thisadaptation as it moves from area to area in the input. An example is the move-to-frontmethod.

(6) Cascaded compression is the providing of serious of compressor that feeds each other respectively, but, those should be lossless coders since the lossy will produce different output during the decodeing operation and this process will be accumulated until the final decoder will faced by a non-readable file.

(7) Perceptive compression is the process of lossy compressor that needs to eliminate the data whose absence would not be detected by our sence.

(8) Symmetrical compression is the case where the compressor and decompressor usebasically the same algorithm but work in "opposite" directions.Such a method makessense for general work, where the same number of files is compressed as is decompressed.

(9) Asymmetric compression means either the compressor or the decompressor may have to work significantly harder. useful in environments where files are updated all the time and backups are made. There is a small chance that a backup file will be used, so the decompressor isn't used very often.

(10)Universalcompressor means the compressor and decompressor do not know the statistics of the input stream. A universal method is optimal if the compressor can

produce compression factors that asymptotically approach the entropy of the input stream for long inputs.

(11) The term file differencing refers to any method that locates and compresses thedifferences between two files. Instead of transferring the whole file especially for a file that existed in several computers.

(12) Streaming mode and block mode, in the former one, the codec inputs a byte or several bytes, processes them, and continues until an end-of-file is sensed. Some methods, such as Burrows-Wheeler, work in the block mode, where the input stream is read block by block and each block is encoded separately.

(13) Most compression methods are physical. They look only at the bits in the input stream and ignore the meaning of the data items in the input. Some compression methods are logical. They look at individual data items in the source stream and replace common items with short codes.

(14) Compression performance: Several measures are commonly used to express theperformance of a compression method.

- Compression Ratio

$$Compression\ ratio = \frac{\text{size of the output stream}}{\text{size of the input stream}}$$

A value of 0.6 means that the data occupies 60% of its original size after compression. Values greater than 1 imply an output stream bigger than the input stream (negative compression). The compression ratio can also be called bpb (bit per bit). In image compression, the same term, bpb stands for "bits per pixel."alsobpc (bits per character)—the number of bits it takes, on average, to compress one character inthe input stream.

Note that, bitrate is a general term for bpb and bpc. Thus, the maingoal of data compression is to represent any given data at low bit rates.

- Compression Factor

The inverse of the compression ratio is called the compression factor:

$$Compression\ factor = \frac{\text{size of the input stream}}{\text{size of the output stream}}$$

In this case, values greater than 1 indicate compression and values less than 1 implyexpansion. This measure seems natural to many people, since the bigger the factor,the better the compression.

- Saving Percentage shows the shrinkage of data as a percentage.

$$Saving\ Percentage = \frac{\text{size before compression} - \text{size after compression}}{\text{size before compression}}$$

(14) Other quantities, such as mean square error (MSE) and peak signal to noise ratio(PSNR), are used to measure the distortion caused by lossy compression of images andmovies, where PSNR for image is 37, and for wav is 79.82.

Entropy:

Information Theory uses the term entropy as a measure of how much information isencoded in a message. The word entropy was borrowed from thermodynamics, and it hasa similar meaning. The higher the entropy of a message, the more information it contains.The entropy of a symbol is defined as the negative logarithm of its probability. Todetermine the information content of a message in bits, we express the entropy using the**base 2** logarithm:

$$H(\mathcal{P}) = \sum_{j=1}^{n} p_j I(s_j), \text{ or}$$

$$H(\mathcal{P}) = -\sum_{j=1}^{n} p_j \log p_j$$

So, accordingly, Shannon also defined the notion of the self-information (I or i)of a message as:

$$i(s) = \log_2 \frac{1}{p(s)}$$

Entropy fits with data compression in its determination of how many bits of informationare actually present in a message. If the probability of the character 'e' appearing in thismanuscript is 1/16, for example, the information content of the character is four bits. Sothe character string "eeeee" has a total content of 20 bits. If we are using standard 8-bitASCII characters to encode this message, we are actually using 40 bits. The differencebetween the 20 bits of entropy and the 40 bits used to encode the message is where thepotential for data compression arises.

One important fact to note about entropy is that, unlike the thermodynamic measure ofentropy, we can use no absolute number for the information content of a given message.The problem is that when we calculate entropy, we use a number that gives us theprobability of a given symbol. The probability figure we use is actually the probabilityfor a given model, not an absolute number. If we change the model, the probability willchange with it.

Examples:

Example 1 : *Consider a binary source* $S = (a, b)$ *with probabilities* $P = (3/4, 1/4)$ *respectively.* $I(a) = -\log(3/4) = 0.415$ *bits and* $I(b) = -\log(1/4) = 2$ *bits.*

Here the self-information of each symbol in the alphabet cannot, on its individualbasis, represent easily the whole picture of the source. One common wayto consider a set of data is to look at its average value of the self-information ofall the symbols, like:

$$\overline{I_1} = p_a I(a) + p_b I(b) = \frac{3}{4} \times 0.415 + \frac{1}{4} \times 2 = (1.245 + 2)/4 \approx 0.81 \text{ bit}$$

Example 2 : *Suppose we have another binary source* $S_2 = (c, d)$ *with probabilities* $(\frac{1}{2}, \frac{1}{2})$ *respectively.* $I(c) = I(d) = -\log 1/2 = 1$ *bit and* $\overline{I_2} = p_c I(c) + p_d I(d) = 1$ *bit.*

We now see clearly that source S2 contains on average more informationthan source S.

Example 3: *Suppose the probabilities are 1, 0, 0, 0. The entropy is:*

$$H(P) = H(1, 0, 0, 0) = 0 \text{ bit}$$

This suggests that there is zero amount of information conveyed in thesource. In other words, there is simply no need to encode the message.

The Evolution of the Compression Algorithms:

There are generally two kinds of algorithms based on their historical evolutionary, which are statistical and dictionary based models.

- **Statistical Modeling**

The simplest forms of statistical modeling use a static table of probabilities, and this table varies according to the input data stream and the table is rebuilt every time new input is presented, this model was dominated until 1980, example is Huffman coding algorithm.

Statistical models generally encode a single symbol at a time, reading it in, calculating a probability, then outputting a single code.

- **Dictionary Schemes**

In this case a lookup table is used instead of statistical approach,It reads in input data and looks for groups of symbols that appear in adictionary. If a string match is found, a pointer or index into the dictionary can be outputinstead of the code for the symbol. The longer the match, the better the compression ratio.
Example is In LZWcompression in which in 1977and 1978, Jacob Ziv and Abraham Lempel described a pair of compression methodsusing an adaptive dictionary.

It is worth to mention that Huffmanrepresents fixed length symbols with variable length codes. While LZW represents variable length symbols with fixed length codes.

Compression Algorithms:
In the forthcoming lectures, we will go through different compression methods that have been used for data compression.

Basin Techniques for Data Compression

1- Run-Length Encoding (RLE)

This method means the process of replacing the number of runs of each repeated characters in that sequence.
If a data item d occurs nconsecutive times in the input stream, replace the n occurrences with the single pair*nd*. The n consecutive occurrences of a data item are called a run length of n, and thisapproach to data compression is called run-length encoding or RLE and belongs to lossless compression techniques.

Consider the following strings:
1. KKKKKKKKK
2. ABCDEFG
3. ABABBBC
4. abc123bbbbCDE

We highlight the runs in each instance by a small shade.
1. KKKKKKKKK : There is a run of length 9 on symbol k.
2. ABCDEFG : There is no run.
3. ABABBBC : There is a run of length 3 on symbol B.
4. abc123bbbbCDE : There is a run of length 4 on symbol b.

Before we finalize the way of coding, let us examine the following scenarios:

Input: 2._all_is_too_well

First scenario:coding: 2._a2_is_t2_we2
This scenario will not work, clearly, thedecompressor should have a way to tell that the first 2 is part of the text and the othersare repetition factors for the letters o and l.

Result: Failed

Second scenario:coding: 2._a2l_is_t2o_we2l
This scenario will not work, clearly, same problem again.

Result: Failed

Third scenario:coding: 2._a@2l_is_t@2o_we@2l
This canbe decompressed unambiguously. However, this string is longer than the original string,because it replaces two consecutive letters with three characters. We have to adopt theconvention that only three or more repetitions of the same character will be replacedwith a repetition factor.

Result: Passed

So, we now agreed to put a ***codeword*** and not just a single coding to observance and regard the entirecompression of the input text.

Example: String KKKKKKKKK, containing a run containing 9 Ks, can be replacedby triple ('r', 9, 'K'), or a short unit **r9K** consisting of the symbol r, 9and K, where r represents the case of 'repeating symbol', 9 means '9 times of occurrence'and K indicates that this should be interpreted as 'symbol K' (repeating9 times).

Example: When there is no run, in ABCDEFG for example, the run-flag n is assignedto represent the non-repeating symbols and l (length), the length of the longest nonrecurrentsymbols is counted. Finally, the entire non-recurrent string is copiedas the third element in the triple. This means that non-repeating stringABCDEFGis replaced by ('n', 7, 'ABCDEFG'), and **n7ABCDEFG** for short.

Run-length algorithms are very effective for the following cases:

- if the source contains many runs of consecutive symbols.
- the symbols can be characters in a text file with many repetitions
- 0s and ls in a binary file
- colour pixels in an image
- component blocks of larger sound files.

2- Hardware Data Compression (HDC):

This is an improvement of the RLE used by tape drivesconnected to IBM computer systems, and a similar algorithm used in the IBMSystem Network Architecture (SNA) standard for data communications are stillin use today.

We assume each run or the non-repeatingsymbol sequence contains no more than 64 symbols. There are two types ofcontrol characters. One is a flag for runs and the other is for non-run sequences.
We define the repeating control characters as r3, r 4 , . . . , r63.and
Non-repeating symbolsri, where i = 2,... , 63, which gives a total of 123 control characters, and ri will specify the number of spaces (blanks) if mentioned alone without following symbol.

Example: GGG_ _ _ _ _ _BCDEFG_ _55GHJK_LM777777777777
can be compressed to
r3Gr6n6BCDEFGr2n955GHJK_LMr127.

Size of input: 38 size of output: 30
Compression ratio= 30/38=0.79
Compression factor=38/30= 1.26
Saving percentage=(38-30)/38= 0.21

Question: how can we understand the meaning of **r127**at decompressor stage?
Sol: r127 gives one interpretation, since the number after r starts from 3, so, in this case the number taken after r should be 12 which means repetition of number 7 twelve times and not repetition of number 27 one time.

r317 ?which one is correct, r3_17 or r31_7 ?and why ?

3- **Move to Front coding (MtF)**

The idea of MtF is to encode a symbol with a '0' as long as it is a recentlyrepeating symbol. In this way, if the source contains a long run of identicalsymbols, the run will be encoded as a long sequence of zeros.Initially, the alphabet of the source is stored in an array and the index ofeach symbol in the array is used to encode a corresponding symbol. On eachiteration, a new character is read and the symbol that has just been encodedis moved to the front of the array. This process can be seen easily from theexample below.

Example: consider the following input: abcddcbamnopponm
With MtFcoding:C = (0, 1, 2, 3, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3), average=2.5 T1
Without MtF coding: C' = (0, 1, 2, 3, 3, 2, 1, 0, 4, 5, 6, 7, 7, 6, 5, 4), average=3.5 T2
So, average of C is smaller numbers which is the aim of this technique.

Another input: abcdmnopabcdmnop
With MtF coding: C = (0, 1, 2, 3, 4, 5, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7), average=5.25 T3
Without MtF coding: C'=(0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7), average=3.5 T4
Now, C' gets less average than C but, the move-to-front rule createsa worse result in this case, since the input does not contain concentrations of identicalsymbols (it does not satisfy the concentration property).

But, however, we can assign Huffman codes or variable codes for the resulted C to obtain the compressed data, in which the more occurrencesget the less representation storage.

The solution can be done by getting the ascendingalphabetic that constitutes the input data and assigning a corresponding index from an array as follows:

a b c d m n o p
0 1 2 3 4 5 6 7

a	abcdmnop	0	a	abcdmnop	0	a	abcdmnop	0	a	abcdmnop	0
b	abcdmnop	1	b	abcdmnop	1	b	abcdmnop	1	b	abcdmnop	1
c	bacdmnop	2	c	abcdmnop	2	c	bacdmnop	2	c	abcdmnop	2
d	cbadmnop	3	d	abcdmnop	3	d	cbadmnop	3	d	abcdmnop	3
d	dcbamnop	0	d	abcdmnop	3	m	dcbamnop	4	m	abcdmnop	4
c	dcbamnop	1	c	abcdmnop	2	n	mdcbanop	5	n	abcdmnop	5
b	cdbamnop	2	b	abcdmnop	1	o	nmdcbaop	6	o	abcdmnop	6
a	bcdamnop	3	a	abcdmnop	0	p	onmdcbap	7	p	abcdmnop	7
m	abcdmnop	4	m	abcdmnop	4	a	ponmdcba	7	a	abcdmnop	0
n	mabcdnop	5	n	abcdmnop	5	b	aponmdcb	7	b	abcdmnop	1
o	nmabcdop	6	o	abcdmnop	6	c	baponmdc	7	c	abcdmnop	2
p	onmabcdp	7	p	abcdmnop	7	d	cbaponmd	7	d	abcdmnop	3
p	ponmabcd	0	p	abcdmnop	7	m	dcbaponm	7	m	abcdmnop	4
o	ponmabcd	1	o	abcdmnop	6	n	mdcbapon	7	n	abcdmnop	5
n	opnmabcd	2	n	abcdmnop	5	o	nmdcbapo	7	o	abcdmnop	6
m	nopmabcd	3	m	abcdmnop	4	p	onmdcbap	7	p	abcdmnop	7
	mnopabcd						ponmdcba				
	T1			T2			T3			T4	

Example:
Suppose that the following sequence of symbols is to be compressed:**DDCBEEEFGGAA** from a source alphabet (A, B, C, D, E, F, G). Showhow the MtF method works.
Sol: Initially, the alphabet is stored in an array:

	0 1 2 3 4 5 6 A B C D E F G	
D	ABCDEFG	3
D	DABCEFG	0
C	DABCEFG	3
B	CDABEFG	3
E	BCDAEFG	4
E	EBCDAFG	0
E	EBCDAFG	0
F	EBCDAFG	5
G	FEBCDAG	6
G	GFEBCDA	0
A	GFEBCDA	6
A	AGFEBCD	0
	AGFEBCD	

Encoding is: 303340056060

Decoding Stage: same arrangement of letters

	0 1 2 3 4 5 6 A B C D E F G	
3	ABCDEFG	D
0	DABCEFG	D
3	DABCEFG	C
3	CDABEFG	B
4	BCDAEFG	E
0	EBCDAFG	E
0	EBCDAFG	E
5	EBCDAFG	F
6	FEBCDAG	G
0	GFEBCDA	G
6	GFEBCDA	A
0	AGFEBCD	A
	AGFEBCD	

Decoding is: DDCBEEEFGGAA.

4- Burrows-Wheeler Transform (BWT)

The BWT algorithmwas introduced by Burrows and Wheeler in 1994 and is the base of a recent powerful softwareprogramfor conservative data compression bzip which is currently one ofthe best general purpose compression methods for text.

The encoding algorithm manipulates the symbols of S, the entire sourcesequence by changing the order of the symbols. The decoding process transformsthe original source sequence back. During the encoding process, the entire inputsequence of symbols is permutated and the new sequence contains hopefullysome favorable features for compression. It is classified as lossless compression technique.

The original input is a sting S, as a resulted operation, the encoding process produces a sequence L and an index s (the index of the first character in S), the decoding process reproduces the original source sequence back using L and an index s.

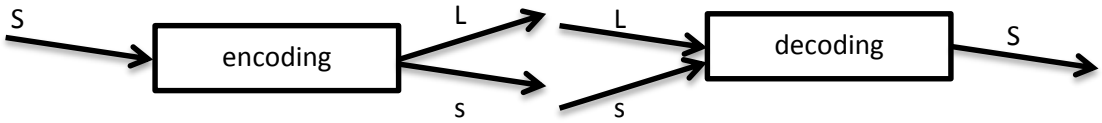


Figure represents the operation of BWT.

As we know, the process involved two steps, encoding and decoding, the encoding process in this method actually is just a permutation of the original string S **which allows a better compression.**

Example:

Consider a string S = 'ACCELERATE' of n = 10 characters, storedin a one-dimensional array. Show how BWT can be realized for encoding anddecoding purposes.

Sol: The purpose of this process is to shuffle the symbols of the source sequence Sin order to derive L. Thelengths of the original array S and of the resulting array L are the same becauseL is actually a permutation of S. In other words, we only want to change theorder of the symbols in the original array S to get a new array L which canhopefully be compressed more efficiently.

Encoder:

Deriving L

1. We first shift the string S one symbol to the left in a circular way. Bycircular, we mean that the leftmost symbol in the array is shifted out ofthe array, and then added back from the right and becomes the rightmostelement in the array. For example,
'A C C E L E R A T E'willbecome'C C E L E R A T E A ' after sucha circular-to-left shift.

Repeating the circular shift n - 1 times, we can generate the n x n matrixbelow where n is the number of symbols in the array, and each row andthe column is a particular permutation of S.

	0	1	2	3	4	5	6	7	8	9
0	A	C	C	E	L	E	R	A	T	E
1	C	C	E	L	E	R	A	T	E	A
2	C	E	L	E	R	A	T	E	A	C
3	E	L	E	R	A	T	E	A	C	C
4	L	E	R	A	T	E	A	C	C	E
5	E	R	A	T	E	A	C	C	E	L
6	R	A	T	E	A	C	C	E	L	E
7	A	T	E	A	C	C	E	L	E	R
8	T	E	A	C	C	E	L	E	R	A
9	E	A	C	C	E	L	E	R	A	T

2. We now sort the rows of the matrix in lexicographic order so the matrixbecomes:

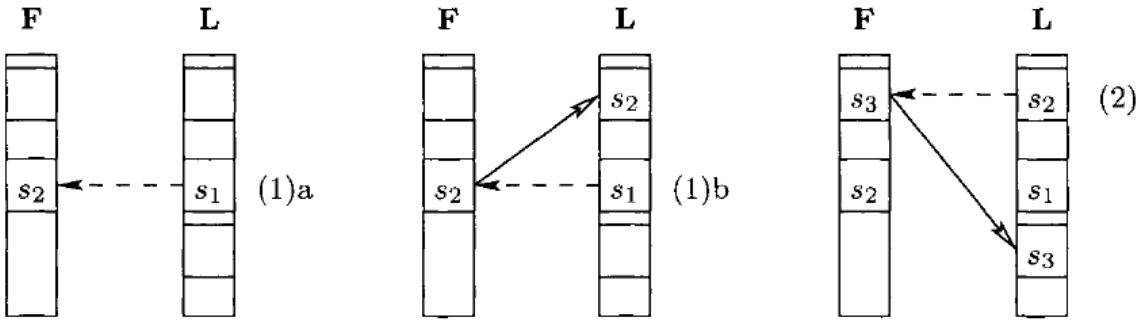
	0	1	2	3	4	5	6	7	8	9
0	A	C	C	E	L	E	R	A	T	E
1	A	T	E	A	C	C	E	L	E	R
2	C	C	E	L	E	R	A	T	E	A
3	C	E	L	E	R	A	T	E	A	C
4	E	A	C	C	E	L	E	R	A	T
5	E	L	E	R	A	T	E	A	C	C
6	E	R	A	T	E	A	C	C	E	L
7	L	E	R	A	T	E	A	C	C	E
8	R	A	T	E	A	C	C	E	L	E
9	T	E	A	C	C	E	L	E	R	A

3. We name the last column L, which is what we need in the BWT forencoding, where sl indicates the first symbol of the given array S.

	F									L	
0	A	C	C	E	L	E	R	A	T	E	
1	A	T	E	A	C	C	E	L	E	R	
2	C	C	E	L	E	R	A	T	E	A	$\leftarrow s_1$
3	C	E	L	E	R	A	T	E	A	C	
4	E	A	C	C	E	L	E	R	A	T	
5	E	L	E	R	A	T	E	A	C	C	
6	E	R	A	T	E	A	C	C	E	L	
7	L	E	R	A	T	E	A	C	C	E	
8	R	A	T	E	A	C	C	E	L	E	
9	T	E	A	C	C	E	L	E	R	A	

Note that, F is the sorted version of the L, however, the receiver will receive the L and s1 which is the index of the first character in S, as follows:
L=ERACTCLEEA, s1=A

Decoder:
The goal of the reverse process of the transform is to recover the original stringS from L. Since L is a permutation of S, what we need to find is the orderrelationship in which the symbols occurred in the original string S.Note that both L and F are permutations of the original string S, and foreach symbol in L, we know the next symbol in the original string S would bethe one in F with the same index (because, during the encoding process, theleftmost element was shifted out and added to the right end to become therightmost element).



Chain Relationship

Note that two types of links exist between the items in F and those in L. Thefirst type is, from L to F, to link two items s_i and its follower s_{i+1} by an identicalindex. If s_i = L[k], for some index k, then the next symbol s_{i+1} = F[k]. Thesecond type is, from F to L, to link by the same symbol s_{i+1} from its locationin F to its location in L.Last figure (2) shows thetwo types of links.

We can define an auxiliary array T to store, for each elementin F, its index in L. T is sometimes called transformation vector and is criticalfor deriving the original string S from L.

Now, for the previous encoded message, L=ERACTCLEEA, s_1=A

```
   0 1 2 3 4 5 6 7 8 9
L: E R A C T C L E E A

   0 1 2 3 4 5 6 7 8 9
F: A A C C E E E L R T
T: 2 9 3 5 0 7 8 6 1 4
```

The process starts from s_1 = L[2], we then know the next symbol in S:s2 = F[2] = ' C ' = L[T[2]] = L[3], From s_2 = L[3], we know thenext symbol is s_3 = F[3] = 'C' = L[T[3]] = L[5], From s3 = L[5],we know the next symbol s_4= F [5] = ' E ' = L[T[5]] = L [7] . This processcontinues until the entire string S = ACCELERATE is derived.

So, however, The BWT cannot be performed until the entire input file has been processed.

Ex: consider the following message, compress and decompress using BWT.
S= DOLLSTICKETS, assume there are no spaces between the words.
Sol:

D	O	L	L	S	T	I	C	K	E	T	S
O	L	L	S	T	I	C	K	E	T	S	D
L	L	S	T	I	C	K	E	T	S	D	O
L	S	T	I	C	K	E	T	S	D	O	L
S	T	I	C	K	E	T	S	D	O	L	L
T	I	C	K	E	T	S	D	O	L	L	S
I	C	K	E	T	S	D	O	L	L	S	T
C	K	E	T	S	D	O	L	L	S	T	I
K	E	T	S	D	O	L	L	S	T	I	C
E	T	S	D	O	L	L	S	T	I	C	K
T	S	D	O	L	L	S	T	I	C	K	E
S	D	O	L	L	S	T	I	C	K	E	T

Now, we sort the resulted circular shift.

C	K	E	T	S	D	O	L	L	S	T	**I**
D	O	L	L	S	T	I	C	K	E	T	**S**
E	T	S	D	O	L	L	S	T	I	C	**K**
I	C	K	E	T	S	D	O	L	L	S	**T**
K	E	T	S	D	O	L	L	S	T	I	**C**
L	L	S	T	I	C	K	E	T	S	D	**O**
L	S	T	I	C	K	E	T	S	D	O	**L**
O	L	L	S	T	I	C	K	E	T	S	**D**
S	D	O	L	L	S	T	I	C	K	E	**T**
S	T	I	C	K	E	T	S	D	O	L	**L**
T	I	C	K	E	T	S	D	O	L	L	**S**
T	S	D	O	L	L	S	T	I	C	K	**E**

Then, the last column is L=**ISKTCOLDTLSE,** s_1=**7**

Now, at the decompress side:

	0	1	2	3	4	5	6	↓7	8	9	10	11
L=	I	S	K	T	C	O	L	**D**	T	L	S	E
F=	C	D	E	I	K	L	L	O	S	S	T	T
T=	4	7	11	0	2	6	9	5	1	10	3	8

So, the original string is: S=**DOLLSTICKETS**

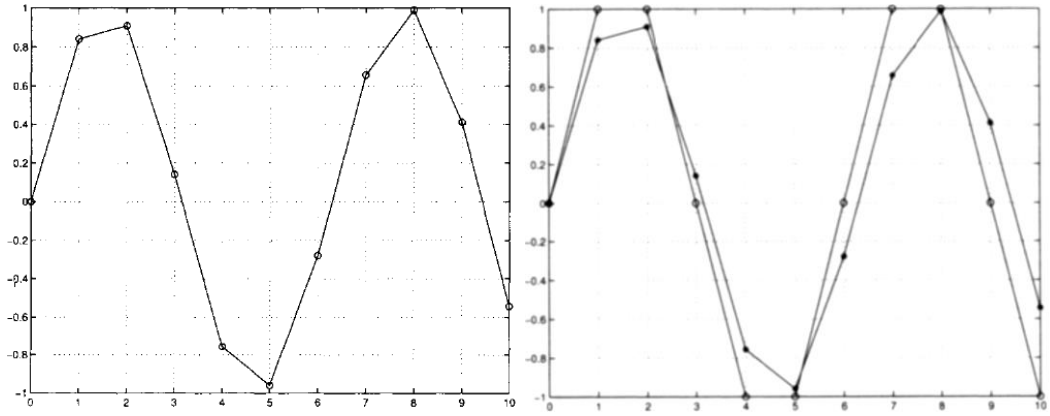
5- Quantization

The dictionary definition of the term "quantization" is "to restrict a variable quantityto discrete values rather than to a continuous set of values." In the field of datacompression, quantization is used in two ways:

1. If the data to be compressed is in the form of large numbers, quantization is usedto convert it to small numbers. Small numbers take less space than large ones, soquantization generates compression. On the other hand, small numbers generally containless information than large ones, so quantization results in lossy compression.
2. If the data to be compressed is analog (i.e., a voltage that changes with time)quantization is used to digitize it into small numbers. The smaller the numbers thebetter the compression, but also the greater the loss of information. This aspect ofquantization is used by several speech compression methods.

So, the mapping in this case is many-to-one mapping and belongs to lossy compression technique since it is irreversible due to lose of information during the compression process.

We can assume **number examples**and not charactersdue to the nature of this method which can ne applied on wav, image and video streaming.



Sampling before quantization after quantization

So, quantization can be applied on real numbers and non-real numbers for the purpose of reducing the possible values of any quantity.The function used tomap the input sequence of values to the output values is called quantizer.

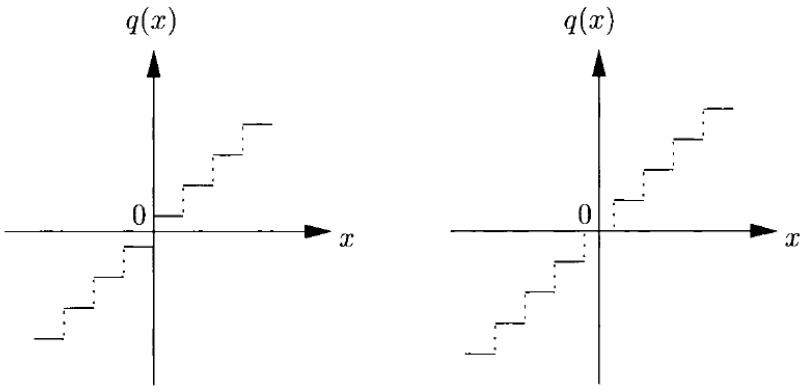
A quantizationis called ***scalar quantization*** if each of the samples is quantized separately. Itis called ***vector quantization*** if at least two samples are quantized at the sametime.

Scalar Quantization:

Wenormally get a staircase shape of curves as shown in following figure if we plot the output values against the input values of a quantizer.

There are two types (outputview or staircase values)of scalar quantizerscalled:

- midrisequantizer: which does not have a zero output level.
- midtreadquantizer: where zero is one of the output values.

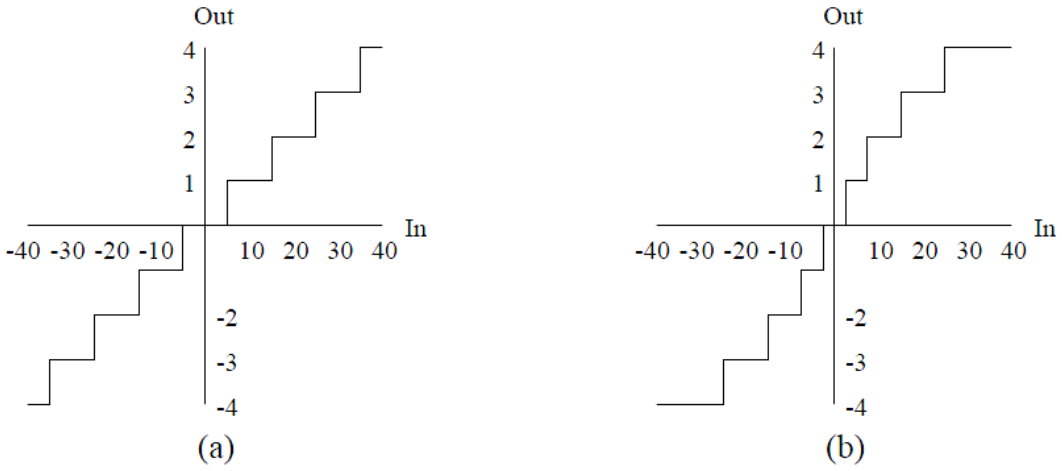


(a) Midrise (b) Midtread

Types of scalar quantizer

A scalar quantizer can be divided (staircase shape or behavior) into:

- uniformquantizer: typically used when the mapping is linear. Again,the example of dividing 8-bit integers by 4 is a linear mapping.
- non-uniformquantizer:which is used typically when the mapping is non-linear. For example, it turns out that the eye is more sensitive to low values of red than to high values. Therefore we can get better quality compressed images by making the regions in the low values smaller than the regions in the high values.



(a) uniform and (b) non-uniform scalar quantization.

The uniform quantized values can be selected according to the following formula:
0,s, 2s,. . . ,ks, such that $(k + 1)s > m$ and $ks \leq m$

Where m is the maximum potential data value and s is the step size (spacing parameter).

Ex. Let us assume m=256 for 8-bits number,

- s=3 leads to producing the following sequence: 0, 3, 6, 9, 12, …, 252, 255
- s=4 leads to producing the following sequence: 0, 4, 8, 12, 16, …, 252, 255 (since the next multiple value of 4 after 252 is 256).

So, any value comes for quantization should be attached to its nearest value from the above sequence, for example:

Ex: quantize the following data to 2-bits representation:
Data={5, 6, 13, 0, 10, 12, 8, 2, 9}, using midtread quantizer.

Sol: 2-bits representation leads to 2^2=4 cases, maximum number=14 which holds 4 bits, which means maximum potential number is 15.

16/4=4 =s, so, we have the quantized sequence ={ 0, 4, 8, 12 }

Quantized data= {4, 4, 12, 0, 8, 12, 8, 0, 8}
So, since the output is in 4-cases, then can be formulated into 2-bits only.

0	00
4	01
8	10
12	11

Note: in case of midrise quantizer, the quantized sequence is ={2, 6, 10, 14}, which means centering the sequence over the potential range.

Ex:consider you have a data with 8-bits number each, find the quantized sequence for to just eight numbers (so each can be expressed in 3 bits), midrise quantizer.

Sol: input is256 numbers (2^8), and is output 8 numbers,
s=256/8=32step size.

We can apply this method tocompute the sequence ={16, 48, 80, 112, 144, 176, 208, 240}.

Note: there are other methods to calculate the quantized sequence and this method is not unique.

We should note that vector quantization, as well as scalar quantization, can be used as part ofa **lossless compression technique.** In particular if in addition to sending the closest representative,the coder sends the distance from the point to the representative, then the original point can bereconstructed. The distance is often referred to as the residual. In general this would not lead to anycompression, but if the points are tightly clustered around the representatives, then the techniquecan be very effective for lossless compression since the residuals will be small and probabilitycoding will work well in reducing the number of bits.

Statistical Methods

The methods discussed so far have one common feature, they assign fixed-size codes tothe symbols (characters or pixels) they operate on. In contrast, statistical methods use**variable-size codes**, with the shorter codes assigned to symbols or groups of symbols thatappear more often in the data (have a higher probability of occurrence).

It is worth to mention that the commonly-occurring letters in English language are E and T, while the rare letters andpunctuation marks are Q,Z, comma.

1- Prefix Codes

A prefix code is a variable-size code that satisfies the prefix property.This property requires that once a certain bit pattern has been assigned asthe code of a symbol, no other codes should start with that pattern (the pattern cannotbe the prefix of any other code).

For example: once the string "1" was assigned as the code of a1,no other codes could start with 1 (i.e., they all had to start with 0). Once "01" wasassigned as the code of a2, no other codes could start with 01. This is why the codes ofa3 and a4 had to start with 00. Naturally, they became 000 and 001.

Designing variable-size codes is therefore done by following two principles:
(1) Assignshort codes to the more frequent symbols
(2) Obey the prefix property.

Ex.: Consider the four symbols a1, a2, a3, and a4. If they appear in our data strings withequal probabilities (= 0.25), then the entropy of the data is $-4(0.25 \log_2 0.25) = 2$ bits, so, 00, 01, 10, and 11 can be assigned as a codewords for each of themconsecutively.

Ex.: consider the table of probabilities of the letters and their assigned codewords:

Symbol	Prob.	Code1	Code2
a_1	.49	1	1
a_2	.25	01	01
a_3	.25	010	000
a_4	.01	001	001

Variable-Size Codes.

The entropy, similarly, H(P)=−(0.49 log2 0.49+0.25 log2 0.25+0.25 log2 0.25+0.01 log2 0.01) ≈−(−0.050−0.5−0.5−0.066) = 1.57 bits per symbol, which is better than 2-bits in previous codewords assigning.
Let us examine the assigned codewords in latter example, let us start with **code1**, 0101 can give different interpretations, as follows:

Case 1: 0101 ➔ 01 | 01 ➔ decoded message = a_2a_2
Case 2: 0101 ➔ 010 | 1 ➔ decoded message = $a_3\ a_1$

So, this is because the prefix property is not considered during this assigning of the codewords, but, however, **code2** satisfies this property.

But, we can assign **code3**:
a_1=1, a_2=001, a_3=010, a_4=001

Our goal therefore is to minimize the average length of the code:

$$\bar{l}(\mathcal{P},\mathcal{L}) = \sum_{j=1}^{n} p_j l_j$$

In case of code2: average length=1.77
In case of code3: average length=2.02

So, it is obvious that **code2** is better than **code3**.

Self-punctuating property: it is the property of discriminating the codewordsfrom each other, like a_1=1, a_2=001, a_3=010, a_4=001, contrarily, a_1=1, a_2=01, a_3=101, a_4=010 is not since 101 has different meanings like a_1a_2 or a_2 ?

2- The Unary Code

The unary code of the positive integer n is defined as n − 1 ones followed by a single 0(as shown in the table below) or, alternatively, as n − 1 zeros followed by a single one. The length of theunary code for the integer n is therefore n bits. Stone-age people indicated the integern by marking n adjacent vertical bars on a stone, so the unary

code is sometimes calleda stone-age binary and each of its n − 1 ones is called a stone-age bit.

n	Code	Alt. Code
1	0	1
2	10	01
3	110	001
4	1110	0001
5	11110	00001

Some Unary Codes.

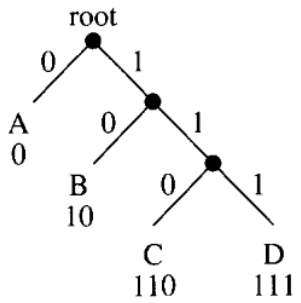
Ex: code the following data123432325?
Sol: as seen in the latter table, the codes are ready to use:
Original message=1 2 3 4 3 2 3 2 5
Coded message= 0|10|110|1110|110|10|110|10|11110

Note that: alphabetic data can be applied as well.

Ex: try to code the following message ABCFFAADFBGHHF

3- Binary Tree

A binary tree can be used as well to construct the codewords for the predefined symbols that construct the message.
For example: message =CBDAAABDA, the codes can be obtained from :



a binary tree

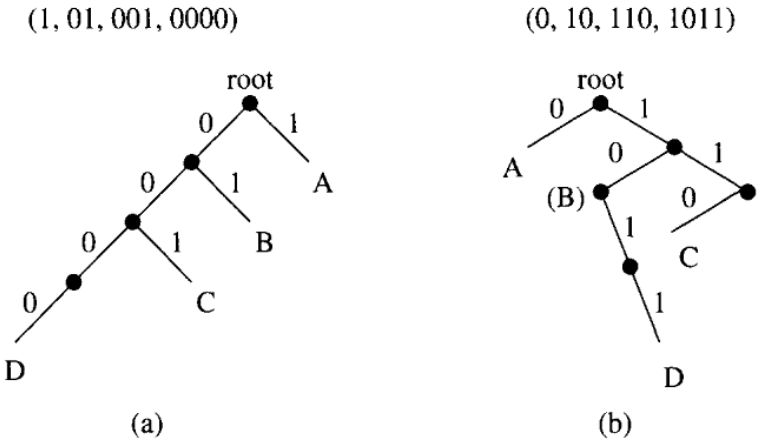
The corresponding codewords are: A=0, B=10, C=110, D=111
So, the coded message is: 110|10|111|0|0|0|10|111|0

Other usage of the binary tree is, to check the prefix,A prefix is the first few consecutive bits of a codeword. When two codewordsare of different lengths, it is possible that the shorter codeword is identical tothe first few bits of the longer

codeword. In this case, the shorter codeword issaid to be a prefix of the longer one, like (01 is the prefix of 0111).

However, after constructing the binary tree, If all the codeword labels are only associated with the leaves, then the codeword is a prefix code. Otherwise, it is not.

Ex: Decide whether the codes (1, 01, 001, 0000) and(0, 10, 110, 1011) for alphabet (A, B, C, D) are prefix codes.



Binary trees for equivalent codewords.

For a prefix code, the codewords are only associated with the leaves. Sinceall the codewords in (1, 01, 001, 0000) are at leaves (last figure (a)), wecan easily conclude that (1, 01,001, 0000) is a prefix code.Since codeword 10 (B) is associated with an internal node of the 0-1 tree(last figure (b)), we conclude that (0, 10, 110, 1011) is not a prefix code.

4- Shannon-Fano Coding

This method was suggested by Shannon and Weaver in 1949 and modified by Fano in 1961.
It goes as follows: First sort all the symbols in nonincreasing frequency order. Then split this list in a way that the first part's sum of frequencies is as equal as possible to the second part's sum of frequencies. This should give you two lists where the probability of any symbol being a member of either list is as close as possible to one half. When the split is done, prefix all the codes of the symbols in the first list with 0 and all the codes of the symbols of the second list with 1. Repeat recursively this procedure on both sublists until you get lists that contain a single symbol. At the end

of this procedure, you will get a uniquely decodable codebook for the arbitrary distribution you input.

EXAMPLE: Find a code using Fano-Shannon coding for a source of five symbols of probabilities 0.5 , 0.2 , 0.1, 0.1, 0.1? Then find its efficiency?
Sol:- Dividing lines are inserted to successively divide the probabilities into halves, quarters ,…etc as shown in the following Figure. A '0' and '1' are added to the code at each division and the final code obtained by reading from the right towards each symbol, writing down the appropriate sequence of 0's and 1 's.

			CODE
S_1	0.5	0	0
S_2	0.2	1 0 0	100
S_3	0.1	1 0 1	101
S_4	0.1	1 1 0	110
S_5	0.1	1 1 1	111

Entropy H =1.96
Average length=0.5x1+0.2x3+0.1x3+0.1x3+0.1x3=2.0
Efficiency=1.96/2.0x100% = 98%

5- Huffman Coding

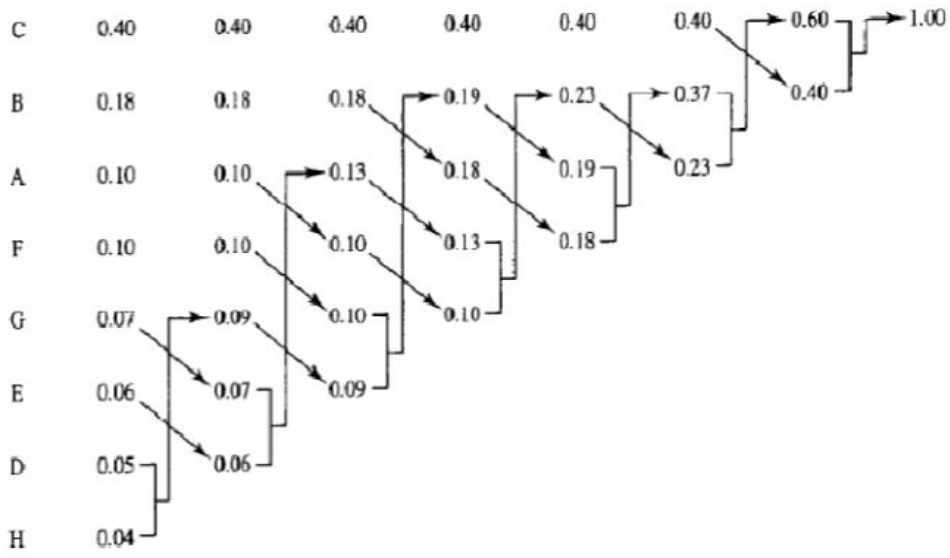
Huffman (1952) devised a variable-length encoding algorithm, based on the source letter probabilities P(xi), i = 1, 2,..., L.
Huffman codes solve the problem of finding an optimal codebook for an arbitrary probability distribution of symbols.
There are, of course, many different ways of devising codewords. However, Huffman's algorithm produces optimal codewords, in the sense that while there might exist many equivalent codewords, none will have a smaller average code length.

Example: find out the equivalent codewords for the following data:

Symbol	*C*	*B*	*A*	*F*	*G*	*E*	*D*	*H*
Probability	0.40	0.18	0.10	0.10	0.07	0.06	0.05	0.04

?

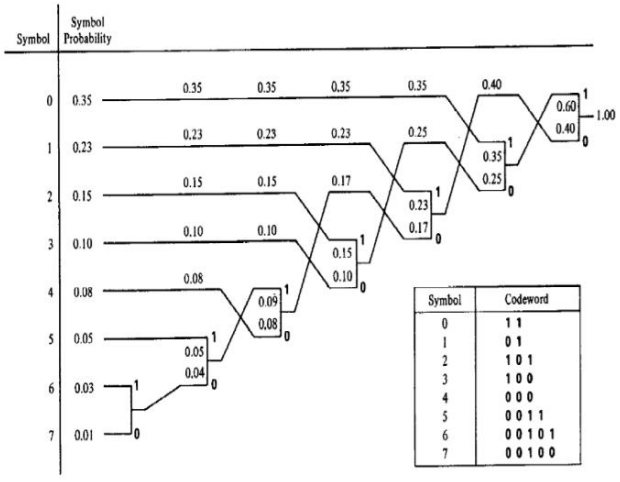
Sol:



Symbol	C	B	A	F	G	E	D	H
Probability	0.40	0.18	0.10	0.10	0.07	0.06	0.05	0.04
Codeword	1	001	011	0000	0100	0101	00010	00011

Find the efficiency ?

Another example: probabilities: 0.35, 0.23, 0.15, 0.10, 0.08, 0.5, 0.03, 0.01
Sol:



Note: solve the above question using shanon-fano algorithm.

References:

1- Introduction to Data Compression, by Guy E. Blelloch, Carnegie Mellon University, 2013.
2- Data Compression Objectives, University of New Orleans, Department of Computer Science, 2008.
3- The Data Compression Book (Second Edition), by Mark Nelson and JeanloupGailly, Cambridge, 2004.
4-Data Compression; The Complete Reference (fourth edition), by David Salomon, published by Springer, 2007.
5- Fundamental Data Compression, by Ida MengyiPu, Published by ELSEVIER, 2006.
6- Variable-Length Codes for Data Compression, by David Salomon, Published by Springer, 2007.