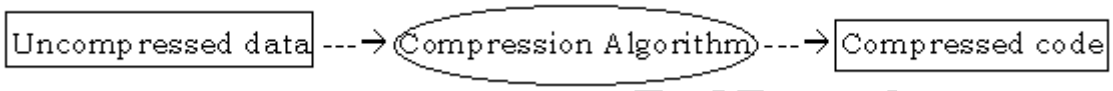
Data Representation: Coding and Compression Techniques

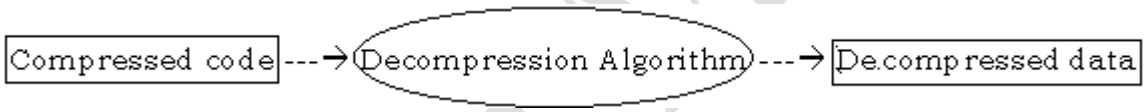
Digitization of data was a revolutionary step for the knowledge industry that made it extremely easy, effortless and space-time-cost effective to create, edit, store, transmit, retrieve, use, and reuse data. Today data representation has reached at a highly sophisticated state evolving through various coding phases. Here is a given brief overview of the commonly used digital coding and compression techniques.

Data Compression:

It is the process of reducing the amount of data required to represent a given quantity of information. All compression techniques aim at reducing certain types of redundancies present in data. The compression and decompression processes can be understood from the following block diagrams.



Compression



Decompression

Compression Error (CErr)= Original data – decompressed data

Types of compression

Compression techniques can be broadly divided into two classes

i. Lossless compression

In this case the original data can be exactly retrieved without any loss after the decompression. In other words the decompressed data is exactly same as the original uncompressed data i.e., CErr=Null

ii. Lossy compression

Original data cannot be retrieved once it is compressed using these techniques i.e., the decompressed data is not exactly same as the original uncompressed data or CErr != Null

Before entering into further discussion about the compression techniques let us first understand certain terminologies such as redundancy, compression ratio, signal to noise ratio (SNR) etc.

Redundancy:

It is the amount of irrelevant and repeated information present in a particular coding scheme. In other words, coding schemes or data representation standards those contain repetitive and extraneous information are said to contain redundant data.

e.g., If b_1 and b_2 represents the number of bits required to represent a certain amount of information using codes C_1 and C_2 respectively then the coding redundancy R_c will be given by

$$R_c = 1 - \frac{1}{C} \qquad \text{--------(1)}$$

Where, C is the compression ratio defined as

$$\text{Compression ratio (C)} = \frac{\text{Number of bits in uncompressed representation}}{\text{Number of bits in compressedrepresentation}}$$

$$= b_1 / b_2 \qquad \text{--------(2)}$$

Example 1

Let us assume that a piece of information ***I*** is represented by 500 bits using a code C_1. If the same piece of information is represented by 100 bits using another code C_2 then the compression ratio

Compression ratio (C) = b_1 / b_2 = 500/100 = 5:1

And

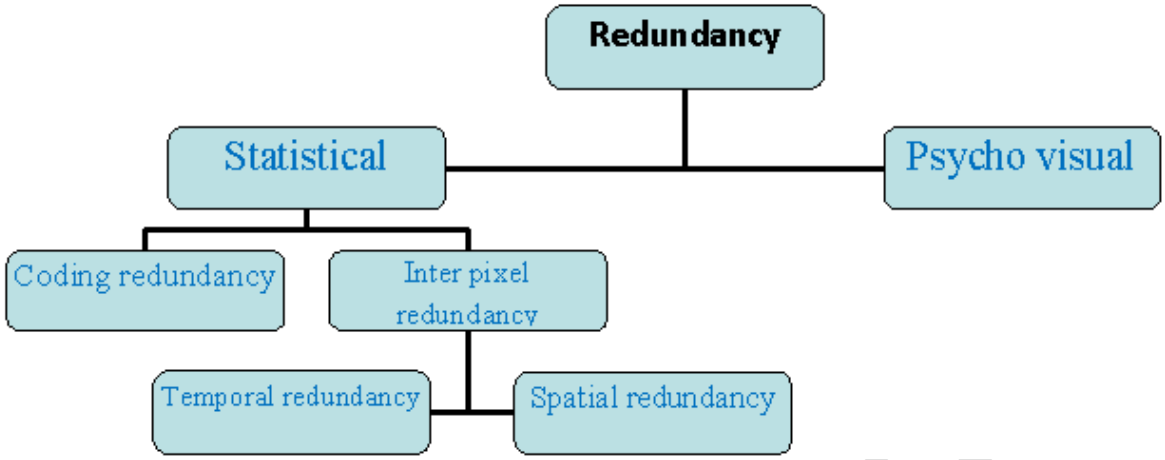
Coding redundancy $R_c = 1 - \frac{1}{C}$ = 1- 1/5 = 4/5 =.8 = 80%

Which implies, code C_1 has 80% of redundancy in comparison to code C_2

Types of Redundancy

Redundancy can be broadly classified into two types such as Statistical redundancy and Psycho visual redundancy. Statistical redundancy further can be of two types such as coding redundancy and inter pixel redundancy. Inter-pixel redundancy is further classified into spatial redundancy and temporal redundancy. Spatial redundancy represents the correlation between neighboring pixel values whereas spectral redundancy represents the correlation between different color

planes or spectral bands. Similarly temporal redundancy gives the correlation between adjacent frames in a sequence of images in video data.



Coding redundancy: It is associated with the coding system used for information representation. Information is general is represented using some standard codes such as ASCII, Unicode, ISO etc for textual information and 8-bit gray levels or 24-bit color for images etc. These standard representations generally use more number of bits in comparison to the number of bits actually necessary for representing that information giving rise to coding redundancy. If $P(S_k)$ is the probability of occurrence of a particular symbol S_k and if each symbol S_k requires $n(S_k)$ number of bits then the average number of bits those will be required to represent each symbol in a text containing a total of N symbols will be given by

$$\text{bps}_{avg}\ \text{(Average bits per symbol)} = \sum_{k=0}^{N-1} P(S_k) * n(S_k) \qquad \text{---------(3)}$$

Example 2

Let us consider that the probabilities corresponding to set of symbols are as given below:

S_k	S1	S2	S3	S4
$P(S_k)$	0.2	0.3	0.12	0.38

The average bits per symbol Bps_{avg} can be computed as follows for two different codes C_1 and C_2, where C_1 is a fixed length code with 8 bits per symbol and C_2 is a variable length code.

S_k	$P(S_k)$	Number of bits/ Code in C_1	Code using C_2	Number of bits/ Code in C_2
S1	0.2	8	111	3
S2	0.3	8	10	2
S3	0.12	8	110	3
S4	0.38	8	0	1

From the above table,

bps_{avg} (Average bits per symbol) for code C_1

= (0.2+0.3+0.12+0.38)*8= 8 bits/symbol

bps_{avg} (Average bits per symbol) for code C_2

= 0.2*3 + 0.3*2 + 0.12*3 + 0.38*1

= 0.6+0.6+.36+.38

= 1.94 bps

Compression ratio C= 8/1.94= 4.123:1 (Approx)

Coding Redundancy $R_c = 1 - \frac{1}{C}$ = 1-1/(8/1.94)=.7575= 75.75%

Therefore, code ***C_1*** contains 75.75% of redundancy against code ***C_2***

Huffman code

Proposed during 1952 by David A. Huffman, a pioneer in computer science, Huffman Code is an optimal data compression algorithm based on entropy encoding. The properties of this code are:

1. This is a variable length code which means every character or symbol is represented by a variable number of bits unlike the ASCII code in which every symbol is represented by a fixed number of bits. The number of bits per symbol is calculated taking into account the frequency of occurrences of symbols present in a document. The symbol with highest frequency or probability is assigned with a minimum length code whereas a symbol with minimum probability with highest number of bits. For example,

   Example 3: if there are only three symbols A, B and C with probabilities 0.52, 0.31 and 0.17 respectively then A to be represented by minimum possible number of bits. The codes for A, B and C can be something as follows:

A=0
B=10
C=11

This will provide compression in the sense, if there are 100 characters in a text then,

Total number of bits required in ASCII = 100*8=800.

Total number of bits in above case = (.52*1 + .31*2 + .17*2) *100= 148

$$\text{Compression ratio (C)} = \frac{\text{Number of bits in uncompressed representation}}{\text{Number of bits in compressedrepresentation}}$$

$$= 800/148$$

$$= 5.405:1$$

$$\text{Redundancy (R)} = 1 - \frac{1}{C} = 0.815 = 81.5\%$$

2. Huffman code is a prefix code (or better can be termed as a prefix free code) in the sense the code or bit pattern assigned to one character or symbol is not a prefix of the code assigned to any other character/symbol. The prefix free property of the code ensures error free decoding during the decompression phase.

   For example,

   Example 4: Let us encode/decode the word CAB using the coding scheme as shown in example 3 above

   Encoding:

   Initialization steps:

   W=the sequence of characters to be encoded

   CW=Null

   Step1: Read the first character (Symbol) S from W scanning it from left to right

   Step2: CW=CW + code corresponding to S

   W= W-S.

   Step3: Is W =NULL? If yes stop else go to step 1

   Continuing in this way CAB can be encoded as 11010

Decoding:

Initialization steps:

W=NULL

CW=the code word

Step1: Read the first bit ***b*** from CW scanning it from left to right

Step2: Is ***b*** a code for any symbol?

If yes,

Step3: W=W + Character corresponding to code ***b***.

CW=CW-***b***

If no,

Step4: read the next bit N

b= ***b***+N

go to step 2

Step5: Is CW =NULL? If yes stop else go to step 1

NOTE: In this case, + symbol is used to represent concatenation and – represents removal of a symbol or a set of symbols.

Continuing through the above steps,

Initially, W=Null and CW=11010

Iteration No	**CW**	**b**	**Is b a code**	**W**
1	11010	1	N	NULL
		11	Y	C
2	010	0	Y	CA
3	10	1	N	
		10	Y	CAB

So, the code is correctly decoded to the initial sequence of symbols. Now, instead of assigning the codes as above let us assign the codes as follows:

A=0
B=01
C=11

In this coding scheme, 0 (the code of A) is a prefix of 01 (the code of B).

Now, CAB can be coded as 11001

Decoding,

Initially, W=Null and CW=11001

Iteration No	CW	b	Is b a code	W
1	11001	1	N	NULL
		11	Y	C
2	001	0	Y	CA
3	01	0	N	CAA
4	1	1	N	
5	NULL	?	?	?

It can be seen from the above example that the decoding is not unambiguous and error free if the code is not prefix free.

Steps of the Huffman algorithm

Huffman Coding consists of two major steps such as,

i. Building the Huffman Tree from the input sequence of characters/ symbols.
ii. Traversing the Huffman Tree and assigning codes to each character/symbol.

Building the Huffman Tree

Input: Unique characters or symbols (S) and their probabilities of occurrence (P(S))

If (S, P(S)) pair is not given then can be computed as follows:

a. Read the sequence (W) to be encoded
b. List the unique symbols (S) in W
c. Count the number of occurrence (frequency(F)) of each symbol
d. P(S)= probability of occurrence of S = F(S)/length(W)

Example 5, let W=KAKAKABAB

Length(W)=9

F(A)=4
F(B)=2
F(K)=3

Therefore, P(A)=4/9, P(B)=2/9 and P(K)=3/9

Note: sum of probabilities should always be equal to ONE.

Step1. Create a leaf node (S, P(S)) for each unique character and build a min heap of all leaf nodes.

Step2. Extract two nodes with the minimum P(S) from the min heap.

Step3. Create a new internal node with P(S) equal to the sum of the P(S)s of two nodes. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

Step4. Repeat steps 2 through 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

The above process can be understood through the following example.

Example 6:

Let us consider the symbols and their probabilities as given in example 2

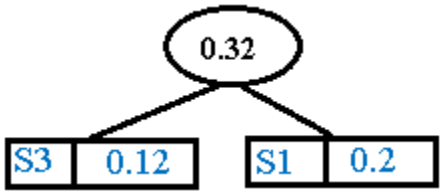
S_k	S1	S2	S3	S4
$P(S_k)$	0.2	0.3	0.12	0.38

Solution

Step1: Build a min heap containing 4 nodes where each node represents root of a tree with single node.

S_k	**$P(S_k)$**
S3	0.12
S1	0.2
S2	0.3
S4	0.38

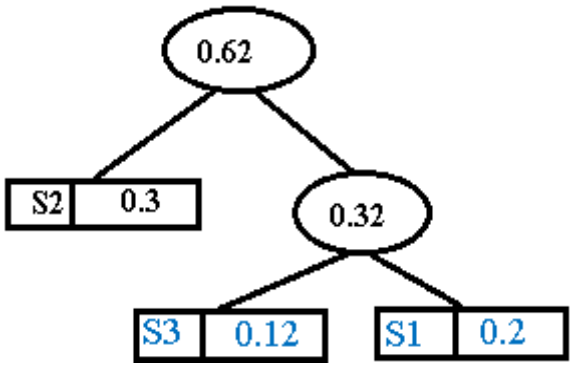
Step2: Extract two minimum frequency nodes from min heap. Add a new internal node with the probability equal to sum of the probabilities of these two nodes.



Now min heap contains 3 nodes out of which 2 nodes are roots with single element each, and one heap node is root with 2 leaves.

S_k	**$P(S_k)$**
S2	0.3
Internal node	0.32
S4	0.38

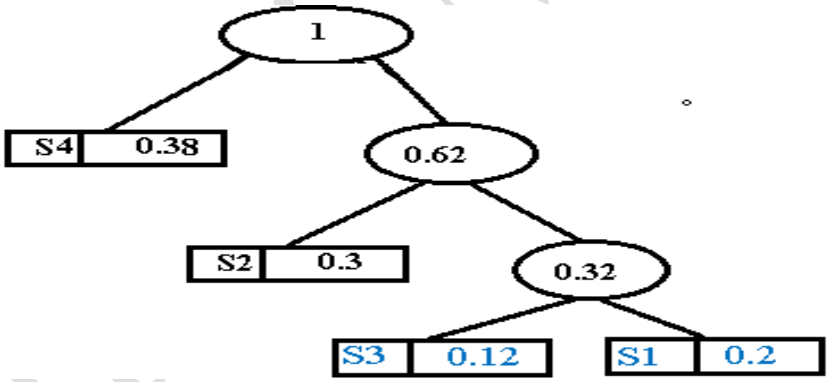
Step3: Extract two nodes 0.3 and 0.32 with minimum probabilities from heap. Add a new internal node with probability 0.62



Now min heap contains 2 nodes, one root with a single element and another root with 4 children.

S_k	$P(S_k)$
S4	0.38
Internal node	0.62

Step4: Extract two nodes 0.38 and 0.62 with minimum probabilities from heap. Add a new internal node with probability 1

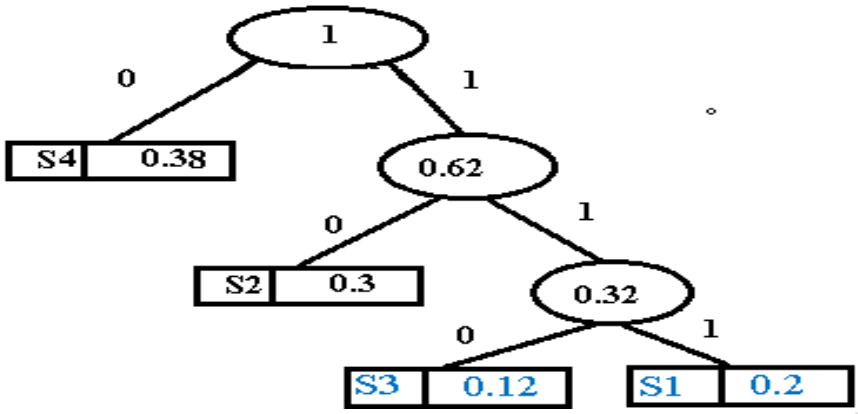


S_k	$P(S_k)$
Internal node	1

Since the min heap contains a single node so, stop.

Traversing the Huffman Tree and assigning codes to each character/symbol

1. Traverse the above Huffman tree starting from the root.
2. Maintain an auxiliary array.
    a. Write 0 to the array while moving to the left child and
    b. Enter a 1 to the array while traversing the right child.
3. Print the array when a leaf node is encountered.



The symbols and their corresponding codes are:

S_k	Code
S4	0
S2	10
S3	110
S1	111

Computation of compression ratio and redundancy:

Total number of bits required to represent a sequence of 100 S_k those are are represented in ASCII = 100*8=800.

Total number of bits in above case = (.38*1 + .3*2 + .12*3+ .2*3) *100= 194

$$\text{Compression ratio (C)} = \frac{\text{Number of bits in uncompressed representation}}{\text{Number of bits in compressedrepresentation}}$$

$$= 800/194$$

$$= 4.123:1 \text{ (Approx)}$$

$$\text{Redundancy (R)} = 1 - \frac{1}{C} = 0.7575 = 75.75\%$$

Over the years, the Huffman tree has got several types of representation but careless implementation of those variants of Huffman tree may result with non-optimal code. Let us try to understand this through the following three examples.

Example 7:

S_k	A	B	C	H
$P(S_k)$	0.52	0.1	0.25	0.13

Example 8:

S_k	A	B	C	H
$P(S_k)$	0.47	0.1	0.25	0.18

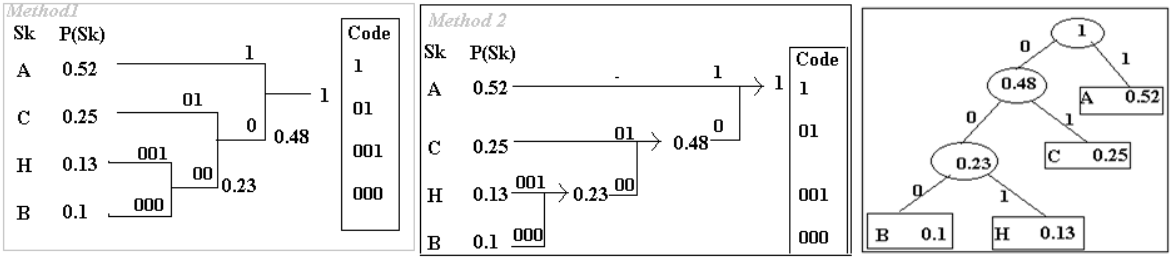
Example 9:

S_k	A	B	C	H
$P(S_k)$	0.32	0.2	0.26	0.22

Solutions:

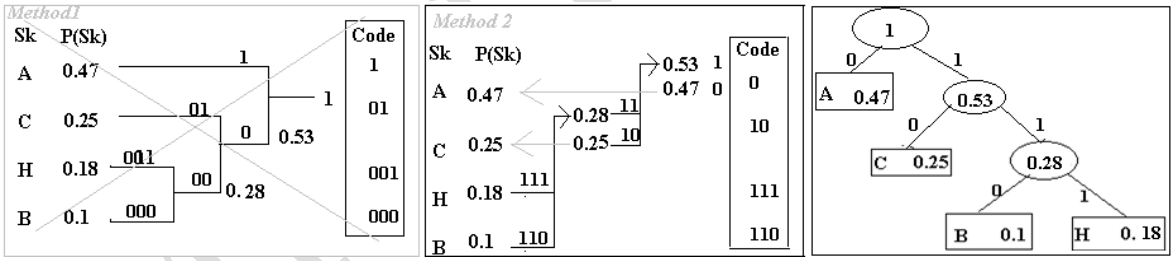
Arrange the symbols according to the decreasing order of their probabilities and then construct the Huffman tree as follows:

Solutions of Example 7



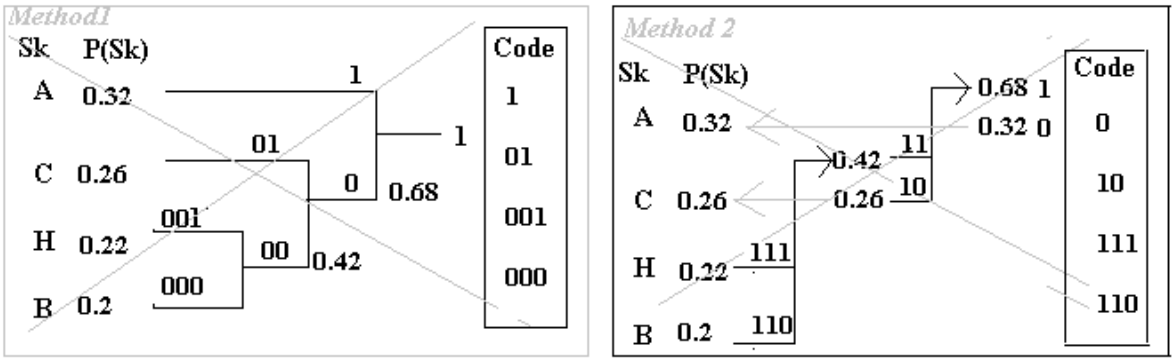
For this example, both the methods generated the same codes correctly.

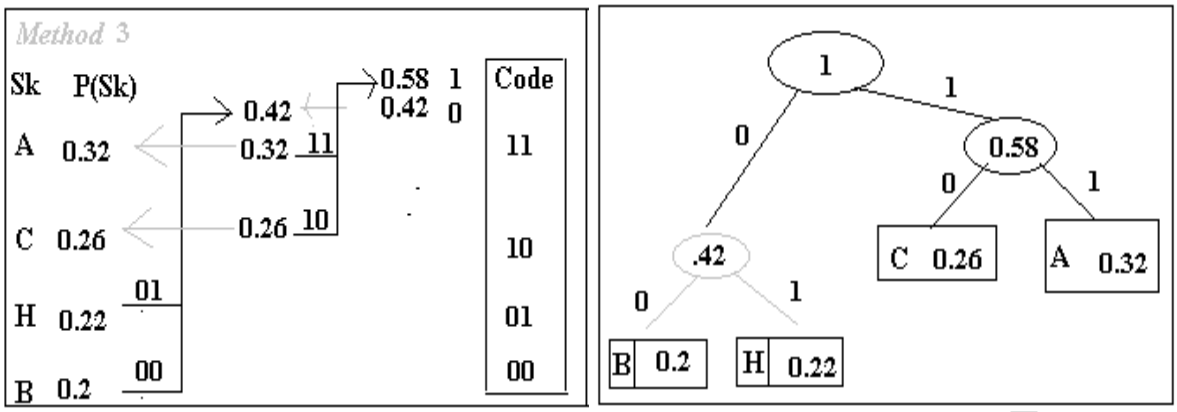
Solutions of Example 8



For the above example 8, method 1 generated a wrong sequence of codes as the probabilities are not rearranged according to their decreasing order.

Solutions of Example 9





In this case method 2 uses a local comparison method instead of global arrangement of the probabilities at step2 hence produces incorrect codes. Hence, Method 3 is the correct solution to problem 3.

Now if we have to encode BAHBAHBACHA which have 1C, 3 Hs, 4 As and 3 Bs using the above solutions (of example -9) then the coded sequence will have

1*2+3*3+4*1+3*3= 2+9+4+9=24 bits using solution given by method 1

1*2+3*3+4*1+3*3= 2+9+4+9=24 bits using solution given by method 2

1*2+3*2+4*2+3*2= 1+6+8+6=21 bits using solution given by method 3

Hence method 3 generates the optimal code and is correct.

Review Questions:

1. Encode the following sequence using Huffman code
   ABRAKADABRABABARDARBARDABARDARBAR

2. Find Huffman code for the following symbols

S_k	A	B	C	H	N	_
$P(S_k)$	0.12	0.21	0.23	0.22	0.19	0.03

   i) How many bits will be required to encode the following sequence of characters?

      BACHAN_CHHABAN_NANHABAHAN

   ii) Find the compression ratio and redundancy against the ASCII representation.