1. Name of the Experiment: Explain and implementation of Huffman code.

Huffman Coding

An important class of prefix codes known as Huffman codes. The basic idea behind coding is to assign to each symbol of an alphabet a sequence of bits roughly equal in length to the amount of information conveyed by the symbol in equation. Huffman codes are compact codes. That is, the Huffman algorithm produces a code with an average length, L, which is the smallest possible to achieve for the given number of source symbols, code alphabet and source statistics.

Binary Huffman Coding Algorithm

For the design of binary Huffman codes the Huffman coding algorithm is as follows:

1. The source symbol are listed in order of decreasing probability. The two source symbols of lowest probability are assigned a 0 and a 1.This part of the step is referred to as a splitting stage.

2. These two source symbols are regarded as being combined into a new source symbol with probability equal to the sum of the two original probabilities. The probability of the new symbol is placed in the list in accordance with its value.

3. The procedure is repeated until we are left with a final list of source statistics of only two for which a 0 and a 1 are assigned.

The code for each source symbol is found by working backward and tracing the sequence of Os and 1s assigned to that symbol as well as its successors.

As Example: Consider a 5 symbol source with the following probability assignments:

$P(S_1) = 0.2$ $P(S_2) = 0.4$ $P(S_3) = 0.1$ $P(S_4) = 0.1$ $P(S_5) = 0.2$

Re-ordering in decreasing order of symbol probability produces $\{S_2, S_1, S_5.S_3, S_4\}$. The re ordered source is then reduced to the source S_3, with only two symbols as shown in Figure 1, where the arrow-heads point to the combined symbol created in Sy by the combination of the last two symbols from S_J-1. Starting with the trivial compact code of {0, 1} for S_3, and working back to S a compact code is designed for each reduced source S, and shown in Figure 1. In each S, the code word for the last two symbols is produced by taking the code word of the symbol pointed to by the arrow-head and appending a 0 and 1 to form two new code

words. The Huffman code itself is the bit sequence generated by the path from the root to the corresponding leaf node.

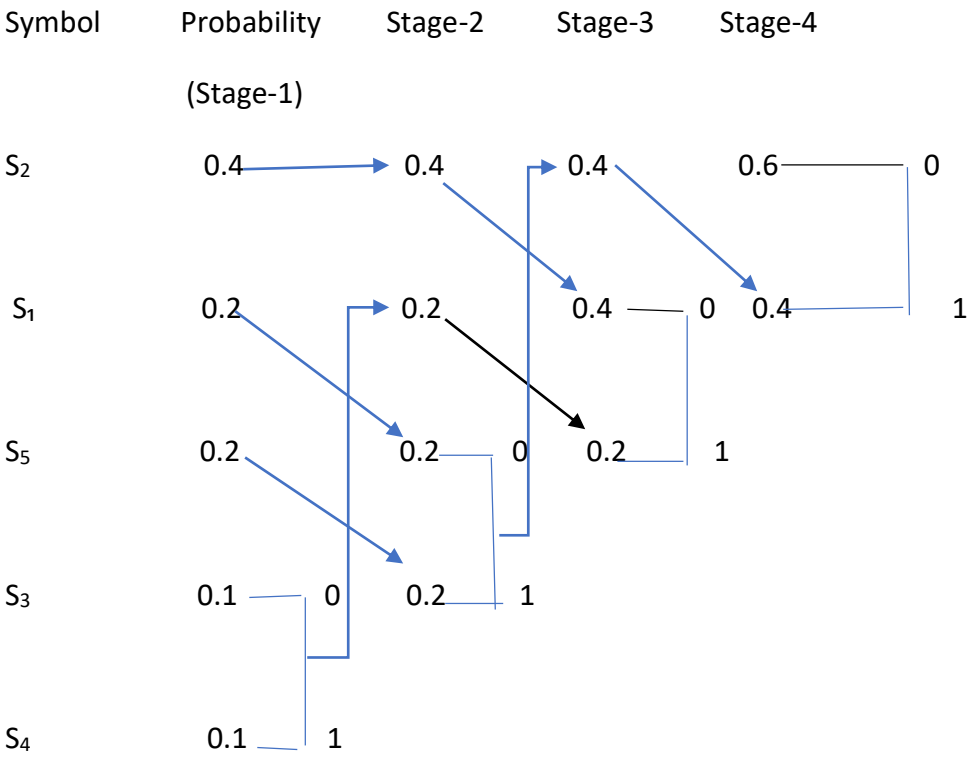


FIGURE 1: Binary Huffman Coding Table

Binary Huffmnan code is:

Symbol	$P(S_i)$	Huffman Code
S_1	0.2	10
S_2	0.4	00
S_3	0.1	010
S_4	0.1	011
S_5	0.2	11

2. Explain & implementation of convolutional coding.

Problem Description:

Convolution coding:

Convolution codes or Trellis Code introduce memory into the coding process to improve the error-correcting capabilities of the codes. The coding and decoding processes that are applied to error-correcting block codes are memory less. The encoding and decoding of a block depends only on that block and is independent of any other block. They do this by making the parity checking bits dependent on bit values in several consecutive blocks.

Say we have a message source that generates a sequence of information digits (U_k). We will assume that the information digits are binary, i.e., information bits. These information bits are fed into a convolutional encoder. As an example consider the encoder shown below. This encoder is a finite state machine that has (a finite) memory: Its current output depends on the current input and on a certain number of past inputs. In the example its memory is 2 bits, i.e., it contains a shift register that keeps stored the values of the last two information bits. Moreover, the encoder has several modulo-2 adders. The output of the encoder is the codeword bits that will be then transmitted over the channel. In our example for every information bit, two codeword bits are generated. Hence the encoder rate is

R_t =1/2bits.

In general the encoder can take n, information bits to generate n. codeword bits yielding an encoder rate of $R_t = n_i/n_e$ bits.

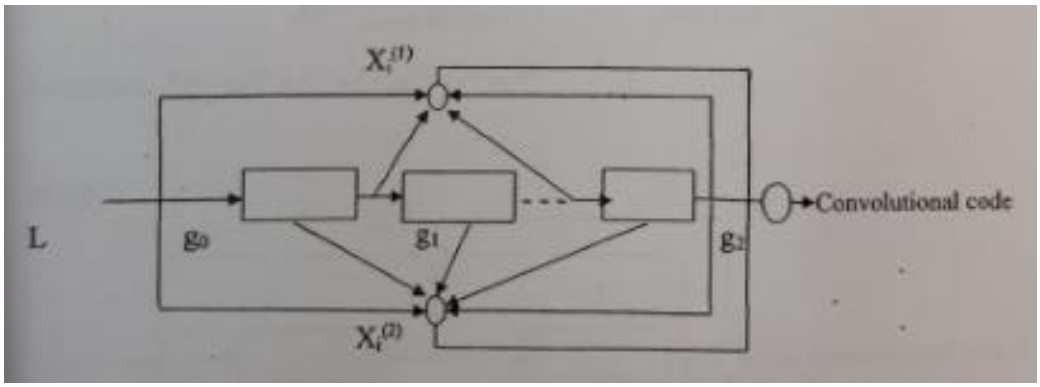


Fig :Convolution Encoder.

To make sure that the outcome of the encoder is a deterministic function of the sequence of input bits, we ask the memory cells of the encoder to contain zeros at the beginning of the encoding process. Moreover, once L, Information bits have been encoded, we stop the information bit sequence and will feed T dummy zero bits as inputs instead,

where T is chosen to be equal to the memory size of the encoder. These dummy bits will make sure that the state of the memory cells are turned back to zero. Here in the above diagram,

L=the message length

m=no. of shift register
n=no of modulo-2 adder

Output -=n(m+L)bit and Code rate, r=L/(n(m+L)); L>>m

=L/(Ln)
=l/n
Constraint Length ,K=m+I If $g^{(1)}_0$, $g^{(1)}_1$, $g^{(1)}_2$…… $g^{(l)}_m$ are the state of shift register then the input-top adder output path is given by
$g^{(1)}_0$, $g^{(1)}_1$, $g^{(1)}_2$…… $g^{(l)}_m$
and the input-bottom adder output path is given by
$g^{(2)}_0$, $g^{(2)}_1$, $g^{(2)}_2$…… $g^{(2)}_m$

Let the message sequence be m_o,m_1,m_2…….m_a, then convolution sum for (1)
$X_i^{(l)}$
= $\sum_{l=0}^{m} 1.\, g_1{}^1\, m_{i-1}$ i=0,1,2,n

and, then convolution sum for (1)
$X_i^{(l)}$
= $\sum_{l=0}^{m} 1.\, g_1{}^1\, m_{i-1}$

So output, Xi=($x_o{}^1\, x_o{}^2\, x_1{}^1, x_1{}^{2,}\, x_2{}^{1,}\, x_2{}^2$…….)

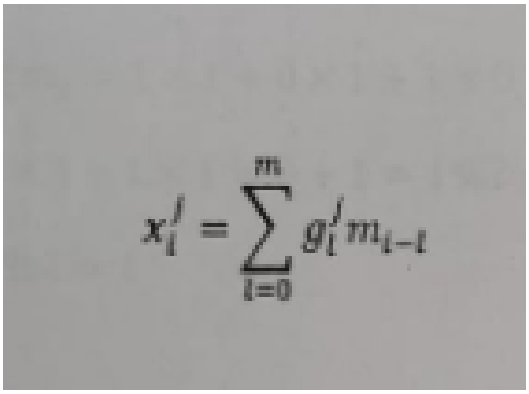
Input:

Top output path: ($g^{(1)}_0$, $g^{(1)}_1$, $g^{(1)}_2$) = (1, 1, 1)

Bottom output path: ($g^{(2)}_0$, $g^{(2)}_1$, $g^{(2)}$)=(1,0,1)
Message bit sequence-(m_o, m_1, m_2, m_3, m_4)=(1,0,0,1,1)

Solution:

We know that,



$$x_i^j = \sum_{i=0}^{m} g_i^j m_{i-i}$$

When j=1 & i=0 then,

$X_0^1 = g_0{}^1 m_o = 1x1=1\%2=1$

Then for successive i, we get:

$X_1^1 = g_0{}^1 m_1 + g_1{}^1 m_0 = 1x0+1x1 = 0+1=1\%2 = 1$
$X_2^1 =, 1\times0+1x0+1x1=0+1=1\%2 = 1$
$X_3^1 = 1x1+1x0+1x0=1+0+0=1\%2 = 1$
$X_4^1 = 1x1+1x1+1x0=1+1+0=2\%2 = 0$
$X_5^1 = 1x1+1x1=1+1=2\%2 = 0$
$X_6^1 = g_1{}^2 m_3 = 1x1 =1\%2 = 1$

$\therefore x_i{}^1 = 1111001.$

When j=2 & i=0 then,
$X_2^0 = g_0{}^2 m_0 = 1x1 = 1\%\ 2=1$
Then for successive i, we get:
$X_1^2 = 1x0+0x1=0+0=0$
$X_2^2 = 1x0+ 0x0+1\times1= 0+1=1\%2 = 1$
$X_3^2 = 1x1+0x0+1x0=1+0+0=1\%2 = 1$
$X_4^2 = 1\times1+0x1+1x0=1+0+0=1\%2 = 1$
$X_5^2 = 0x1+1x1 = 0+1=1\%2 = 1$

$X_6^2 = 1 x 1 =1\%2 = 1$
So, $x_i^2 = 1011111$
And
$X_i = 11101111010111$

3. Name of the Experiment: Explain and implementation of Lempel-Ziv code.

Theory:

Lempel–Ziv is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv. It was the algorithm of the widely used Unix file compression utility compress and is used in the GIF image format.

Lempel–Ziv algorithm is accomplished by parsing the source data stream into segments that are the shortest subsequences not encountered previously. To illustrate, let us consider an input binary sequences as follows:

000101110010100101……

Let us assume 0 & 1 are already stored so,

Subsequence stored: 0,1

Data to be parsed: 000101110010100101…….

The encoding process begins at left. As 0 & 1 are already stored, the shortest sub-sequence of data stream are written as,

Subsequence stored: 0, 1, 00

Data to be parsed: 0101110010100101……

The second and the next sequences are,

Subsequence stored: 0, 1, 00, 01

Data to be parsed: 01110010100101…….

We continue this until the given data stream is completely parsed. Now the binary code blocks of the sequences are,

Numerical Positions:	1	2	3	4	5	6	7	8	9
Subsequence:	0	1	00	01	011	10	010	100	101
Numerical representation:		11	12	42	21	41	61	62	
Binary encoded blocks:		0010	0011	1001	0100	1000	1100	1101	

Figure: Illustrating the encoding process performed by the Lempel-Ziv algorithm.

From the figure, the first row show the numerical position of individual subsequences in the code. A sequence of data stream 010 consists of the concatenation of the sequence 01 in position 4 and symbol 0 in position 1; hence the numerical representation is 41. Similarly others are.

The last row in figure is the binary subsequence is Binary encoded blocks. The last symbol of each sequence in the code book is an innovation symbol, which is so called in recognition of the fact that can distinguishes it from all previous subsequence stored in the code book. The last bit of each uniform block of bits represents the innovation symbol and the remaining bits provide the equivalent binary representation of the “pointer” to the root subsequence that matches the one in question expect for the innovation symbol.

The decoder is just simple as the encoder. Use the pointer to identify the root subsequence and appends the innovation symbol. Such as, the binary block encoded block 1101 in position 9. The last bit is the innovation symbol. Here 110 point to the root subsequence 10 in position 6. Hence, the block 1101 is decoded into 101, which is correct.

In contrast to Huffman coding, the Lempel-Ziv algorithm uses fixed length code to represent a variable number of source symbols. That makes Lempel-Ziv coding suitable for synchronous transmission.

4.Name of the Experiment: Explain and implementation of Hamming code.

Theory:

In telecommunication, Hamming codes are a family of linear error-correcting codes that generalize the Hamming (7,4) code invented by Richard Hamming in 1950. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are perfect codes, that is, they achieve the highest possible rate for codes with their block length and minimum distance 3.

In mathematical terms, Hamming codes are a class of binary linear code. For each integer $r \geq 2$ there is a code-word with block length $n = 2^r - 1$ and message length $k = 2^r - r - 1$. Hence the rate of Hamming codes is $R = k / n = 1 - r / (2^r - 1)$, which is the highest possible for codes with minimum distance of three (i.e., the minimal number of bit changes needed to go from any code word to any other code word is three) and block length $2^r - 1$. The parity-check matrix of a Hamming code is constructed by listing all columns of length r that are non-zero, which means that the dual code of the Hamming code is the shortened Hadamard code. The parity-check matrix has the property that any two columns are pair-wise linearly independent.

Due to the limited redundancy that Hamming codes add to the data, they can only detect and correct errors when the error rate is low. This is the case in computer memory (usually RAM), where bit errors are extremely rare and Hamming codes are widely used, and a RAM with this correction system is a ECC RAM (ECC memory). In this context, an extended Hamming code having one extra parity bit is often used. Extended Hamming codes achieve a Hamming distance of four, which allows the decoder to distinguish between when at most one one-bit error occurs and when any two-bit errors occur. In this sense, extended Hamming codes are single-error correcting and double-error detecting, abbreviated as SECDED.

Parity:

Parity adds a single bit that indicates whether the number of ones (bit-positions with values of one) in the preceding data was even or odd. If an odd number of bits is changed in transmission, the message will change parity and the error can be detected at this point; however, the bit that changed may have been the parity bit itself. The most common convention is that a parity value of one indicates that there is an odd number of ones in the data, and a parity value of zero indicates that there is an even number of ones. If the number of bits changed is even, the check bit will be valid and the error will not be detected.

Moreover, parity does not indicate which bit contained the error, even when it can detect it. The data must be discarded entirely and re-transmitted from scratch. On a noisy transmission medium, a successful transmission could take a long time or may never occur.

However, while the quality of parity checking is poor, since it uses only a single bit, this method results in the least overhead.

Construction of G and H

The matrix $\mathbf{G}:=(I_k|-A^T)$ is called a (canonical) generator matrix of a linear (n,k) code,

And $\mathbf{H}:=(A|I_{n-k})$ is called a parity-check matrix.

This is the construction of **G** and **H** in standard (or systematic) form. Regardless of form, **G** and **H** for linear block codes must satisfy

$\mathbf{H}\,\mathbf{G}^T=\mathbf{0}$, an all-zeros matrix.

Since [7, 4, 3] = [n, k, d] = [$2^m - 1$, $2^m - 1 - m$, 3]. The parity-check matrix **H** of a Hamming code is constructed by listing all columns of length *m* that are pair-wise independent.

Thus **H** is a matrix whose left side is all of the nonzero n-tuples where order of the *n*-tuples in the columns of matrix does not matter. The right hand side is just the (*n* − *k*)-identity matrix.

So **G** can be obtained from **H** by taking the transpose of the left hand side of **H** with the identity *k*-identity matrix on the left hand side of **G**.

The code generator matrix **G** and the parity-check matrix **H** are:

$$\mathbf{G}:=\begin{pmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}_{4,7}$$

And

$$\mathbf{H}:=\begin{pmatrix}1&1&0&1&1&0&0\\1&0&1&1&0&1&0\\0&1&1&1&0&0&1\end{pmatrix}_{3,7}$$

Finally, these matrices can be mutated into equivalent non-systematic codes by the following operations:

- Column permutations (swapping columns)
- Elementary row operations (replacing a row with a linear combination of rows)

[7,4] Hamming code with an additional parity bit:

The [7,4] Hamming code can easily be extended to an [8,4] code by adding an extra parity bit on top of the (7,4) encoded word (see Hamming(7,4)). This can be summed up with the revised matrices:

$$\mathbf{G:=}\begin{pmatrix}1&1&1&0&0&0&0&1\\1&0&0&1&1&0&0&1\\0&1&0&1&0&1&0&1\\1&1&0&1&0&0&1&0\end{pmatrix}_{4,8}$$

And

$$\mathbf{H} := \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}_{4,8}$$

Note that H is not in standard form. To obtain G, elementary row operations can be used to obtain an equivalent matrix to H in systematic form:

$$\mathbf{H} := \left(\begin{array}{cccc|cccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)_{4,8}$$

For example, the first row in this matrix is the sum of the second and third rows of H in non-systematic form. Using the systematic construction for Hamming codes from above, the matrix A is apparent and the systematic form of G is written as

$$\mathbf{G} := \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array}\right)_{4,8}$$

The non-systematic form of G can be row reduced (using elementary row operations) to match this matrix.

The addition of the fourth row effectively computes the sum of all the codeword bits (data and parity) as the fourth parity bit.

For example, 1011 is encoded (using the non-systematic form of G at the start of this section) into 01100110 where blue digits are data; red digits are parity bits from the [7,4] Hamming code; and the green digit is the parity bit added by the [8,4] code. The green digit makes the parity of the [7,4] code-words even.

Finally, it can be shown that the minimum distance has increased from 3, in the [7,4] code, to 4 in the [8,4] code. Therefore, the code can be defined as [8,4] Hamming code.