PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Information and Communication Engineering (ICE)
Faculty of Engineering and Technology
B.Sc. (Engineering) 4th Year 1st Semester
Course Title: Information Theory and Coding Sessional
Course Code: ICE-4106
Session: 2017-2018

Lab Report

Experiment No : 01

Submitted by Rabia Siddika
Roll: 180627
Reg. No: 1065310
Session: 2017-18
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submitted to Sohag Sarker
Associate Professor
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submission date : 24.01.2023

24.01.2023

No of Experiment: 01

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

Theory:

Huffman coding: An important class of prefix codes known as Huffman codes. The basic idea behind Huffman 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. 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 1 are assigned

The code for each source symbol is found by working backward and tracing the sequence of 0s 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 $\{S_2, S_1, S_5, S_3, S_5\}$. The

re-ordered source is then reduced to the source S_3 with only two symbol as shown in Figure 1, Where the arrow-head point to the combined symbol created in S_j 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_j and shown in figure 1. In each S_j 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 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.

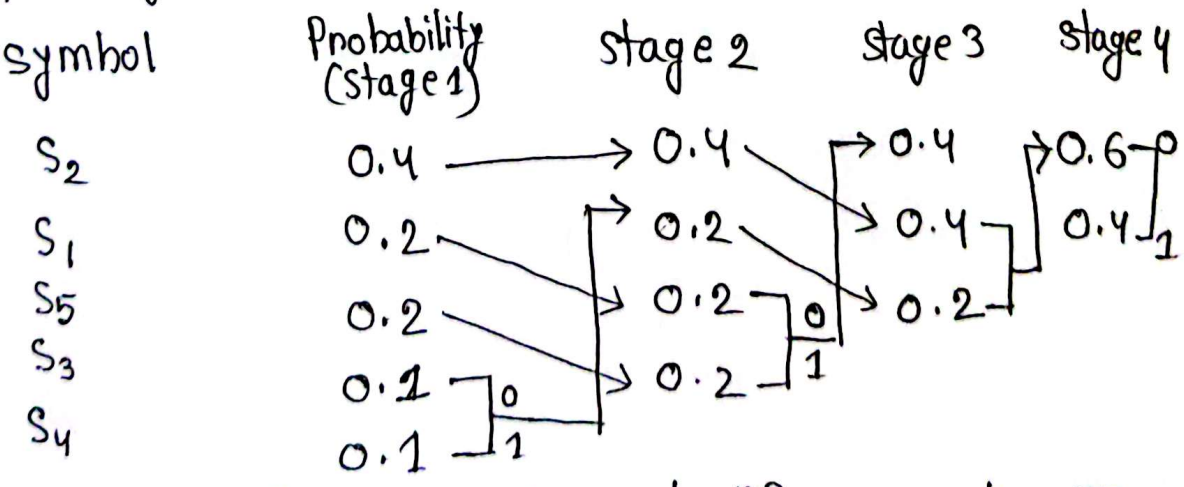


Figure-1 : Binary huffman coding Table.

The binary Huffman code is

symbol	P(Si)	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

Figure: Huffman Code Table

matlab code, we replace the symbols as:-

S1 = A, S2 = B, S_3 = C, S_4 = D, S_5 = E

Source code:

```
clc; clear all; close all;
S = ['A' 'B' 'C' 'D' 'E'];        % S = ['a', 'i', 'L', 'm', 'n',
                                  'o', 'p', 'y'];
P = [0.2  0.4  0.1  0.1  0.2];    % P = [0.1 0.1 0.2 0.1
                                  0.1 0.1 0.2 0.1 0.1]
if length(S) ~= length(P)
  error('Wrong entry ... enter again - ')
end
pos = 1 : length(P);
[P idx] = sort(P, 'descend');
S P = P;
S = S(idx);
PS = pos(idx);
W = length(P);
cnt = W + 1;
for i = 1 : W-2
for j = 1 : (W-i+1)
      flow(i, j) = j;
 end
end
i = 1;
codebook(i, W) = {'1'};
codebook(i, W-1) = {'0'};
while (length(P) > 2)
 tempsum = P(length(P) + P(length(P)-1);
 P = [P(1: length(P)-2), tempsum];
```

```
    prb = p;
    ps = ps(1:length(p));
    ps(length(p)) = cnt;
    cnt = cnt+1;
    [prb idx] = sort(prb,'descend');
    ps = ps(idx);
    idx = find(prb==p(length(p)));
  tp = ps(idx);
 tp = sort(tp,'descend');
 ps(idx) = tp;
 idx = find(cnt-1==ps);
  WL = idx
 flow(i, length(p)) = WL;
 flow(i, length(p)+1) = WL;
 if WL ~= length(p).
       for j = WL:length(p)-1
       flow(i,j) = j+1;
    end
 end
 i = i+1;
codebook(i, length(p)) = {'1'};
codebook(i, length(p)-1) = {'0'};
end
code(1:w) = {' '};
for i = 1:w
   m = i;
```

```
code(1) = codebook(1, i);
for j = 1 : w-2
m = flow(j, m);
code(i) = strcat(codebook(j+1, m), code(i));
end
code(i) = strcat(s(i), ' = ', code(i));
end
for i=1 : w
disp(code(i));
end
```

Output

'B = 00'

'A = 10'

'E = 11'

'C = 010'

'D = 011'

PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Information and Communication Engineering (ICE)
Faculty of Engineering and Technology
B.Sc. (Engineering) 4th Year 1st Semester
Course Title: Information Theory and Coding Sessional
Course Code: ICE-4106
Session: 2017-2018

Lab Report

Experiment No : 02

Submitted by
Rabia Siddika
Roll: 180627
Reg. No: 1065310
Session: 2017-18
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submitted to
Sohag Sarker
Associate Professor
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submission date : 24.01.2023

Name of Experiment: Explain and implementation of convolution coding.

Theory:

Convolution coding:

Convolution codes or Trellis code introduce memory into the coding process to impove 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 the block depends only on that block and is independent of any other block. They do this making the parity checking bits dependent on the 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 digit are binary, i.e. information bits. These information bits are fed into a convolution 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

$$R_t = 1/2 \text{ bits.}$$

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

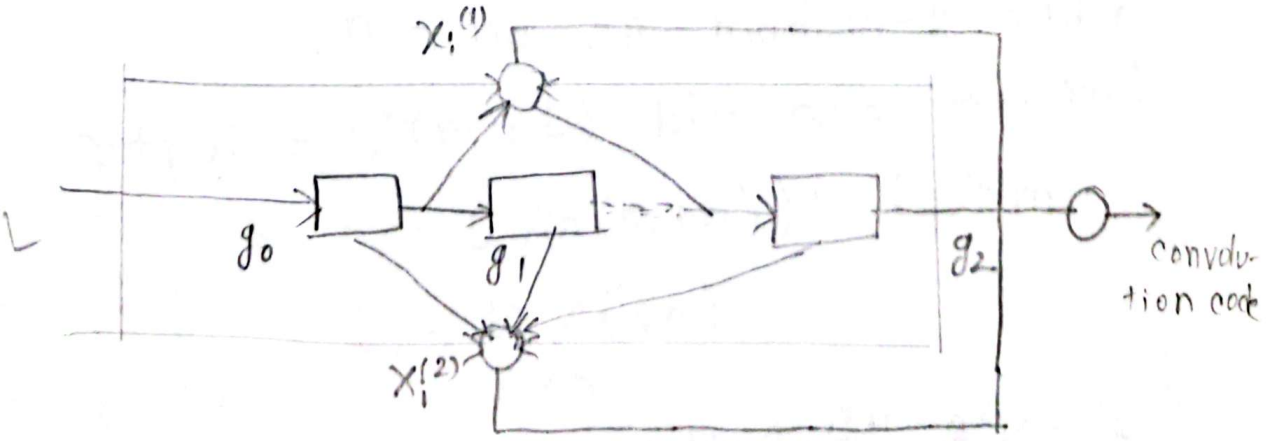


Figure - 01 : convolution encoder

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_1 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 registen
n = no of modulo-2 adden

Output = n (m+L) bit and code rate,

$$r = L/(n(m+L)); \quad L >> m$$
$$= L/(Ln)$$
$$= 1/n$$

Constraint length, k = m+1. If $g^{(1)}_0, g^{(1)}_1, g^{(1)}_2 \ldots g^{(1)}_m$ are the state of shift

register then the input - top adder output path is given by,

$g_0^{(1)}, g_1^{(1)}, g_2^{(1)} \ldots g_m^{(1)}$ are the state of shift register then the input-~~top~~ bottom adder output path is given by

$$g_0^{(2)}, g_1^{(2)}, g_2^{(2)} \ldots g_m^{(2)}$$

Let the message sequence be $m_0, m_1, m_2 \ldots m_n$, then convolution sum for (1)

$$X_i^{(1)} = \sum_{l=0}^{m} g_l^{(1)} m_{i-l} \qquad i = 0, 1, 2, n$$

and, then convolution sum for (1)

$$X_i^{(1)} = \sum_{l=0}^{m} g_l^{2} m_{i-l} \qquad i = 0, 1, 2, n$$

So outputput, $X_i = \{X_0^{(1)} X_0^{(2)} X_1^{(1)} X_1^{(2)} X_2^{(1)} X_2^{(2)} \}$

math :

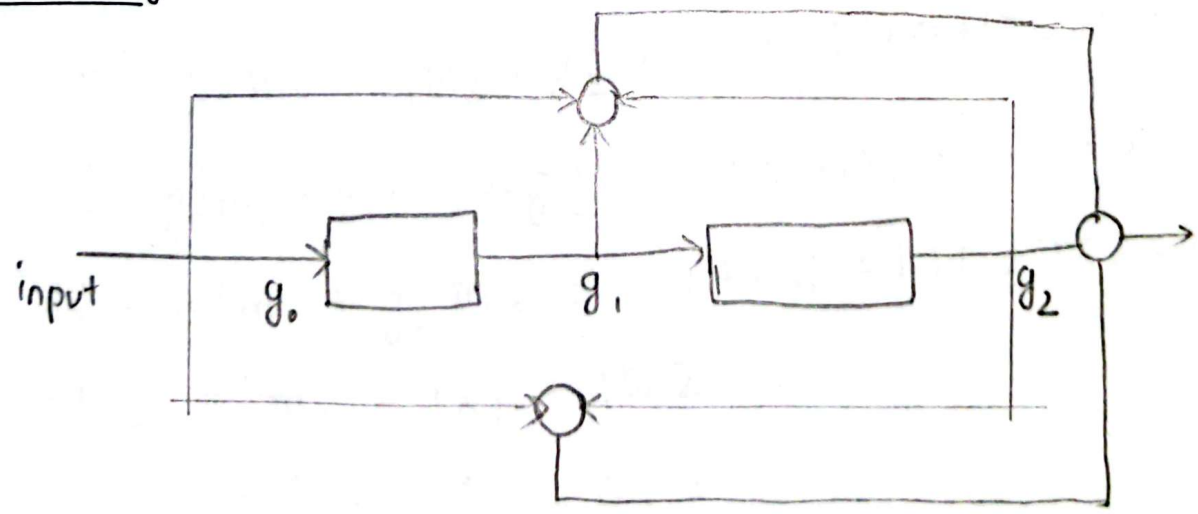


Fig-2 : convolutional encoder

Input:

Top output path : $(g_0^1, g_1^1, g_2^1) = (1,1,1)$

Bottom output path : $(g_0^2, g_1^2, g_2^2) = (1,0,1)$

message bit sequence $= (m_0, m_1, m_2, m_3, m_4) =$

$(1,0,0,1,1)$

Solution:

We know that $x_i^j = \sum_{l=0}^{m} g_l^j \, m_{i-l}$

When $j = 1$ & $i = 0$ then,

$x_0^1 = g_0^1 m_0 = 1 \times 1 = 1 \% 2 = 1$

Then for successive i, we get:

$x_1^1 = g_0^1 m_1 + g_1^1 m_0 = 1 \times 0 + 1 \times 1 = 0 + 1 = 1 \% 2 = 1$

$x_2^1 = g_0^1 m_2 + g_1^1 m_1 + g_2^1 m_0 = 1 \times 0 + 1 \times 0 + 1 \times 1 = 0 + 1 = 1 \% 2 = 1$

$x_3^1 = g_0^1 m_3 + g_1^1 m_2 + g_2^1 m_1 = 1 \times 1 + 1 \times 0 + 1 \times 0 = 1 + 0 + 0 = 1 \% 2 = 1$

$x_4^1 = g_0^1 m_4 + g_1^1 m_3 + g_2^1 m_2 = 1 \times 1 + 1 \times 1 = 1 + 1 = 2 \% 2 = 0$

$x_5^1 = g_1^1 m_4 + g_2^1 m_3 = 1 \times 1 + 1 \times 1 = 1 + 1 = 2 \% 2 = 0$

$x_6^1 = g_2^1 m_3 = 1 \times 1 = 1 \% 2 = 1$

$\therefore x_i^1 = 1111001$.

When $j=2$ and $i=0$ then

$$X_0^2 = g_0^2 m_0 = 1 \times 1 = 1 \% 2 = 1$$

Then for successive i, we get:

$$X_1^2 = g_0^2 m_1 + g_1^2 m_0 = 1 \times 0 + 0 \times 1 = 0 + 0 = 0$$

$$X_2^2 = g_0^2 m_2 + g_1^2 m_1 + g_2^2 m_0 = 1 \times 0 + 0 \times 0 + 1 \times 1 = 0 + 1 = 1 \% 2 = 1$$

$$X_3^2 = g_0^2 m_3 + g_1^2 m_2 + g_2^2 m_1 = 1 \times 1 + 0 \times 0 + 1 \times 0 = 1 + 0 + 0 = 1 \% 2 = 1$$

$$X_4^2 = g_0^2 m_4 + g_1^2 m_3 + g_2^2 m_2 = 1 \times 1 + 0 \times 1 + 1 \times 0 = 1 + 0 + 0 = 1 \% 2 = 1$$

$$X_5^2 = g_1^2 m_4 + g_2^2 m_3 = 0 \times 1 + 1 \times 1 = 0 + 1 = 1 \% 2 = 1$$

$$X_6^2 = g_2^2 m_3 = 1 \times 1 = 1 \% 2 = 1$$

$$\therefore X_i^2 = 1011111$$

So,

$$X_i = 11101111010111$$

Matlab code:

```
clc; clear all; close all;
M = [1  0  0 1 1];
g = [1 1 1 ; 1 0 1];
L = length (M); m = 2; n = 2;
x = zeros (n,(L+m));
for j = 1 : n
    for i = 1 ; (L+m)
      for k = 1 : i
        if (i-k+1) <=L
            mass = M (i-k+1);
        else
            Mass =0;
        end
        if  k > length (g (j,:))
            g (j , k ) = 0;
        end
          x (j, i ) = x ( j, i ) +g ( j, k)* Mass;
    end
      x (j, i) = mod ( x (j , i), 2);
    end
end
% code for output
x(:, :)
```

```
convolutional_code = [];
for i = 1 : (L+m)
convolution_code = strcat (convolution_code,
                   num2str(x(1,i)), num2str(x
                                    (2,i)));
end
convolutional_code.
```

Output

```
ans =
     1     1     1     1     0     0     1
     1     0     1     1     1     1     1

convolution_code =
     1 1 1 0 1 1 1 1 0 1 0 1 1 1.
```

PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Information and Communication Engineering (ICE)
Faculty of Engineering and Technology
B.Sc. (Engineering) 4th Year 1st Semester
Course Title: Information Theory and Coding Sessional
Course Code: ICE-4106
Session: 2017-2018

Lab Report

Experiment No : 03

Submitted by

Rabia Siddika
Roll: 180627
Reg. No: 1065310
Session: 2017-18
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submitted to

Sohag Sarker
Associate Professor
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submission date : 24.01.2023

Experiment no: 03

Name of the Experiment: Explain and Implentatio of Lempel-Ziv Code using MATLAB

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.

Lampel 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 sequence as follows:

0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 ....

Let us assumed 0 and 1 are already stored so, subsequence stone : 0, 1

Data to be parsed: 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 ...

The encoding process begins at left. As 0 and 1 are already stone, 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

subsequence 01 in position 4 and symbol 0 in position 1; hence the numerical representation is 41. Similarly others are.

The decoden is just simple as the encoder. use the pointen to identify the root subsequence and appends the innovation symbol. Such as, the binany block encoded block 1101 in position 9. The last bit is the innovation symbol

In contrast to Huffman coding, the lempel Zip algonithm uses fixed length codes to nepresent a variable numben of source system.

MatLAB Code:

```
X = [0 00 10 111 00101001011]
sbs = {'0', '1'};
N = length(x);
bcb = [];
lengthdict = 0;
while lengthdict < N
      i = lengthdict + 1;
      d = i;
      k = 0;
    Vs = {};
  while k == 0
      f = 0;
      h = i - lengthdict;
      vs = strcat(vs, num2str(X(i)));
    for j = 1 : length(sbs)
      if strcam(sbs(j), Vs) == 1
      if i < N
         i = i+1;
      else c = j;
      end
           f = 1;
        break
    end
    end
```

```
if  f == 0  && j == length (sbs)
      sbs = [sbs ; vs];
       k = 1;
end
end
 if   k == 1
   vp = { } ;
for t = d : i - 1
      vp = strcat (vp , num2str (x(t)));
 end
  p = 0;
 for  m = length (sbs ) : - 1 : 1
 if  strcmp (sbs (m), vp) == 1
     p = m ;
     break
 end
 end
 pointer = dec2bin (p, 3),    % pointer=dec2
                                 base(3,2,8)
 d = strcat (pointer, num2str(x (i)));
 bcb = [bcb , d],
 else
    bcb = [bcb ; bcb (e-2)];
 end
   lengthdict = lengthdict + h ;
end
   subsequnce = sbs
   Binary - encoded - Blocks = bcb.
```

Output

X =

Colums 1 through 18

0 0 0 10 1 1 1 0 0 10 100 101

Subsequences = '0' '1' '00' '01' '011' '10' '010' '100' '101'.

Binary - encoded Blocks = 0010 0011 1001 0100 1000 1100 1101

PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Information and Communication Engineering (ICE)
Faculty of Engineering and Technology
B.Sc. (Engineering) 4th Year 1st Semester
Course Title: Information Theory and Coding Sessional
Course Code: ICE-4106
Session: 2017-2018

Lab Report

Experiment No : 04

Submitted by
Rabia Siddika
Roll: 180627
Reg. No: 1065310
Session: 2017-18
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submitted to
Sohag Sarker
Associate Professor
Dept. of Information and Communication Engineering.
Pabna University of Science and Technology.
Pabna-6600, Bangladesh.

Submission date : 24.01.2023

Experiment no : 04

Experiment name: Explain and Implementation of Hamming code using MATLAB code.

Theory: Hamming codes are a family of linear error correcting codes that generalize the Hamming code invented by Richand Hamming in 1950. Hamming codes can detect up to two bit errors or connect one-bit errors without detection of un connected errors.

In mathematical terms. Hamming codes are a class of binary linear codes. For each integen $r \geq 2$ there is a code 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 highest possible for codes with minimum distance 3 and block length $2^r - 1$. The parity check matrix of a Hamming code is constructed by listing all columns of length n that are non zero, which means that the dual code of the Hamming code is the punctured

parity: Parity adds a single bit that indicates whether the number of 1 bits 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 parity and the error can be detected at this point.

Construction of G and H

The matrix $G := (I_k | -A^T)$ is called a (Canonical) generator matrix of a linear (n,k) code, and $H := (A | I_{n-k})$ 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.

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

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

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

and.

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

Finally, these matrics can be mutated into equivalent non systematic codes by the following operations.

* column permutation (swapping columns)
* Elementary row operation (replacing a row with a linear combination of rows)

Hamming code with an additional parity bit:

The example from above with an extra parity bit. This diagram is not meant a correspond to the matrix for this example

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. This can be summed up with the revised matrix

$$G_2 := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}_{4,8}$$

and

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

Gr elementary row operations can be used to obtain an equivalent matrix to H in systematic form.

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

The matrix A is apparent and the systematic form of Gr is written as

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

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

Source code

```
clc; clear all; close all;
M = input ('Enter 4 bit message :') ;%Example (1 0 1 0)
d1 = [1  0  0 0];
d2 = [0 1 0 0];
d3 = [0  0 1 0];
d4 = [0 0 0 1];
P1 = d2' + d3' + d4';
P2 = d1' + d3' + d4';
P3 = d1' + d2' + d4';
Gr = [ P1 . P2  P3 d1' d2' d3' d4']
P = [P1,  P2,  P3];
H = [eye ( [3  3]) P'];
Code_word = mod ((M * Gr), 2)
E = mod ((H * code_Word'), 2)
ERC = input ('Give an erroneous received hammi-
ng code word!');  % Example (1 0 1 1 0 1 1)
E mod ((H * ERC'), 2)
pe = 0;
for i: 1 : 7
     if (E == H (:, i))
                pe = 1;
```

```
end
end
H
Error_possition = pe
if pe ~= 0
   ERC(pe) = xor(ERC(pe), 1);
end
Hamming_code_word after_error_connection:ERC
```

output : Enter 4 bit message : [1 0 1 0]

```
G = 0 1 1 1 0 0 0
    1 0 1 0 1 0 0
    1 1 0 0 0 1 0
    1 1 1 0 0 0 1

code_word = 1 0 1 1 0 1 0

E = 0
    0
    0
```

Give an erroneous received hamming code word : [1 1 0 1 0 1 0]

```
E = 0
    1
    1

H =  1 0 0 0 1 1 1
     0 1 0 1 0 1 1
     0 0 1 1 1 0 1
```

Error_position = 4

Hamming_code_word_after_error_connection =

1 1 0 0 0 1 0

Experiment no: 05

Experiment name: A binary symmetric channel has the following noise matrix with probability.

Theory: A binary symmetric channel is a common communications channel model used in coding theory and information theory. In this model, A transmitter wishes to send a bit (a zero or a one), and the reciver will receive a bit. It is assumed that the bit is usually transmitted correctly but that it will be ~~fit~~ "flipped" with a small probability. This channel is used frequently in information theory because it is one of the simplest channels to analyze.

A binary symmetric channel with crossover probability p denoted by, is a channel with binary input and binary output and probability of error P:

$$P_r(Y=0 \mid X=0) = 1-P$$
$$P_r(Y=0 \mid X=1) = P$$
$$P_r(Y=1 \mid X=0) = P$$
$$P_r(Y=1 \mid X=1) = 1-P$$

1. It is assumed that $0 \leq P \leq 1/2$. If $P > \frac{1}{2}$

then the receiver can swap the output interpret 1 when it sees 0, and vice versa) and obtain an equivalent channel with error probability $1-P \leq 1/2$.

2. This channel is often used by theorists because it is one of the simplest noisy channel to analyze. Many problems in communication theory can be reduced to BSC

3. This converely, being able to transmit effectively over the BSC can give rise to solutions for more complicated channels

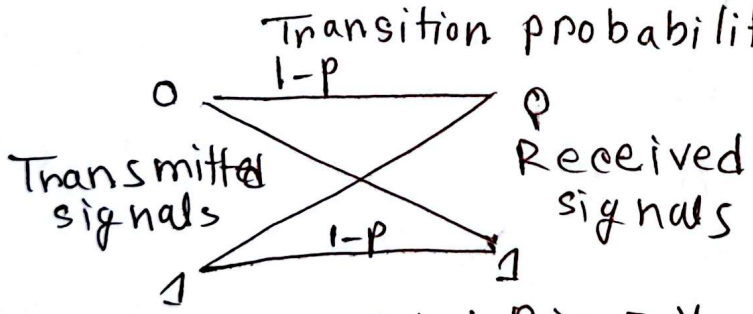


figure-01 : Binary symmetric channel

Here condition Probability H(P) or H(X/Y) calculate using following formula

$$H(P) = (1-P) \log 1/1-P + P \log 1/P$$

source code:

```
clc; clear all; close all;
[row, column] = deal(2,2)
 for i = 1 : row
 for  j = 1 : column
          matrix (i,j) = input ('Enter symmtnic matrix
element according to row → : ');
 end
 end
matrix
HP = matrix (1, 1) * log 2 (1/matrix(1,1)) +
 matrix (1, 2) * log 2(1/matrix(1,2));
Print f ('Channel capacity is : bits·message:
                                1- Hp)!
```

output: Enter symmetric matrix element according to row → : 2/3
Enter symmetric matrix element according to row → : 1/3
Enter symmetric matrix element row → : 1/3
Enter symmetric matrix element row → : 2/3

matrix = 0.6667 0.3333
0.3333 0.6667

conditional Probability H(Y|x) is 9.182985e-01
channel capacity is : 8.170417e-02

Experiment No :- 06

Experiment Name: Check optmality of Huffman code

Theory: To check the optimality of a Huffman code, you can use the kraft inequality, which states that the sum of all the code word lengths (in bits) raised to the powed of -1 must be less than or equal to 1. If a code satisfies that the kraft inequality, it is considered optimal. Another way to check the optimality of Huffman code is to verify that the code is prefix-free, meaning no code word is a prefix of any other code word. A prefix free code is always optimal A third way to check the optimality of Huffmancode is by comparing the average length is equal to the entropy of the source. In all cases; if the code is not optimal. you can use the standard algorithm for constructing a Huffman code to generate a new, optimal code, In this source code I am using kraft inequality check for proving optimality of Huffman code.

source code:

```
clc;
clear all;
close all;
x = {'a','b','c','d','e'}
disp ('symbols')
disp (x)
frequency = [25 25 20 15 15]
disp ('Corresponding frequency of symbols:')
disp ('frequency')
if length (frequency) ~= length (x);
    disp ('Equal length is expected'):
else
    Probability_x = frequency /sum (frequency);
disp ('corresponding Probability of symbols:')
disp (probability_x)
s= sort (probability_x, 'decend');
[dictionary, avg_len] = huffmandict (x,s):
 disp ('Huffman codes:'):
 inequality = 0:
 for i = 1: length (x)
 disp (dictionary {i,1})
 disp (dictionary {i;2})
  Li = length (dictionary {i,2}):
     inequaity = inequity + 2^(-Li);
 end
    disp (' Inequaity:')
    disp (inequity)
```

```
        if inequity <= 1
        disp ('This is optimal code.')
    else disp ('Not optimal.')
    end
end.
```

output:

```
symbols:
    'a' 'b' 'c' 'd' 'e'
corresponding frequency of symbols:
     25  25  20  15  15
corresponding Probability of symbols:
 0.2500   0.2500   0.2000   0.1500   0.1500

Huffmancode:

a    1  0
b    0  1
c      1  1
d
       0  0  1
e      0  0  0

Inequity : 1
This is optimal code.
```

Experiment No:- 07

Experiment Name: Numerical based on conditional Entropy and Joint Entropy.

Example 2.2.2 Let (X, Y) have the following joint distribution:

Y \ X	1	2	3	4
1	1/8	1/16	1/32	1/32
2	1/16	1/8	1/32	1/32
3	1/16	1/16	1/16	1/16
4	1/4	0	0	0

The marginal distribution of X is $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ and The marginal distribution of Y is $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ and hence $H(X) = 7/4$ bits and $H(Y) = 2$ bits. Also,

$$H(X|Y) = \sum_{i=1}^{4} P(Y=i)\, H(X|Y=i)$$

$$= \frac{1}{4} H(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}) + \frac{1}{4} H(\frac{1}{4}, \frac{1}{2}, \frac{1}{8}, \frac{1}{8}) + \frac{1}{4} H(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}) + \frac{1}{4} H(1, 0, 0, 0)$$

$$= \frac{1}{4} \times \frac{7}{4} + \frac{1}{4} \times \frac{7}{4} + \frac{1}{4} \times 2 + \frac{1}{4} \times 0$$

$$= \frac{11}{8} \text{ bits}$$

Similarly, $H(Y|X) = \frac{13}{8}$ bits and $H(X,Y) = \frac{27}{8}$ bits

Theory: Definition: The entropy $H(X)$ of a discrete random variable X is defined by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

Defination: If $(X, Y) \sim p(x, y)$ the conditional entropy $H(Y|X)$ is defined as,

$$H(Y|X) = \sum_{x \in X} p(x) H(Y | X = x)$$

$$= -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log p(y|x$$

$$= -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(y|x)$$

$$= -E \log p(y|x)$$

Source code:

```
clc;
clear all;
close all;
row = 4;
col = 4
input_dist = [1/8 1/16 1/32 1/32; 1/16 1/8 1/32 1/32;
              1/16 1/16 1/16 1/16; 1/4 0 0 0];
disp ('join distribution matrix:')
disp (input_dist)
for i = 1 : row
P_x(i) = sum (input_dist (:, i));
P_y(i) = sum (input_dist (i, :));
H_x(x) = P_x(i) * log 2 (1/P_x(i));
H_y(i) = P_y(i) * log 2 (1/P_y(i));
end
disp ('Marginal distribution of x:')
disp (P_x)
```

```
disp ('Marginal distribution of x:')
disp (P_y)
disp ('Entropy H(x) in bits:')
disp (sum (H_x))
disp ('Entropy H(Y) in bits:')
disp (sum(H_y))
for k = 1 : row
    for m = 1 : col
     tem_x (k, m ) = input_dist (k, m) P_y(k):
     tem_2 (k, m ) = tem_x (k, m ) * log 2(1/tem_x(km));
      if  isnan (tem_2 (k, m))
              tem_2 (k, m) = 0;
      end
 end
      H_x_given_(k) = sum (tem_2 (k, :))*P_y(k)
 end
disp ('H(x|y):')
disp (sum(H_x_given_y))
 for k = 1 : row
   for m = 1 : col
      tem_x (k, m)=input_dist (m, k ) / P_x (k);
      tem_2 (K, m ) = tem_x (k, m ) * log 2 (1/tem_x
                                              (K, m));
  if  isnan (tem_2 (k, m))
     team_2 (k, m ) = 0;
  end
 end
```

```
H_x given_y(k) = sum(tem_2(k,:))* P_x(k);
end
disp(sum(H_x_given_y))
```

output:

Joint distribution matrix

0.1250	0.0625	0.0313	0.0313
0.0625	0.1250	0.0313	0.0313
0.0625	0.0625	0.0625	0.0625
0.2500	0.000	0.0000	0.000

Marginal distribution of X:

0.~~05~~ 0.5000 0.2500 0.1200 0.1250

Marginal distribution of X

0.2500 0.2500 0.2500 0.2500 0.2500

Entropy H(x) in bits:

1.7500

Entropy H(x) in bits:

2

H(X|Y):

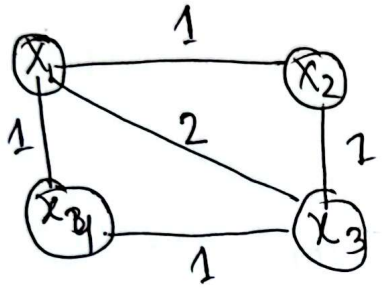
1.3750

H(Y|X):

1.6250

Experiment No: 08

Experiment name: Explain entropy rate of a random work on a weighted graph,



Theory: The entropy of a stochastic process $\{x_i\}$ is defined by, $H(X) = \lim_{n \to \infty} \frac{1}{n} H(X1, X2, \ldots Xn)$

When the limit exits

we now consider some simple of stochastic process and their corresponding entropy rates.

1. Typewriter

Consider the case of a typewriter that has m equally likely output letters. The typewriter can produce m^n sequence of length n, all of them equally likely. Hence $H(X1, X2, \ldots Xn) = \log m^n$ and the entropy rate is

$$Hf(X) = \log m \text{ bits per symbol.}$$

2. X1, X2, . . . are id random variables. Then

$$Hf(X) = \lim \frac{H(X1, X2, \ldots Xn)}{n} = \frac{\lim n H(X1)}{n}$$

$$= H(X1)$$

sequence of independent but not identically distributed random variables.

In this case,

$$H(X_1, X_2, \ldots X_n) = \sum_{i=1}^{n} H(X_i)$$

But the $H(X_i)$'s is all not equal. We can choose a sequence of distributions on $x1, x2, \ldots$ such that the limit of $1/n \sum H(X_i)$ does not exist.

consider a graph with m nodes labled $\{1, 2, \ldots m\}$ with weight $W_{ij} \geq 0$ on the edge joining nodes i to j (The graph is assumed to be undirected, so that $W_{ij} = W_{ji}$.

We verify that this is the stationary distribution by checking that $\mu P = \mu$. Here

$$\sum_i \mu_i P_{ij} = \sum_i \frac{W_i W_{ij}}{2W W_i}$$

$$= \sum_i \frac{1}{2W} W_{ij}$$

$$= \frac{W_i}{2W}$$

$$= \mu_j$$

We can now calculate the entropy rate as

$$H(X) = H(X_2 | X_1)$$

$$= -\sum_i \mu_i \sum_j P_{ij} \log P_{ij}$$

$$= -\sum_i \frac{w_i}{2w} \sum_j \frac{w_{ij}}{w_i} \log \frac{w_{ij}}{w_i}$$

$$= -\sum_i \sum_j \frac{w_{ij}}{2w} \cdot \log \frac{w_{ij}}{w_i}$$

$$= -\sum_i \sum_j \frac{w_{ij}}{2w} \log \frac{w_{ij}}{2w} + \sum_i \sum_j \frac{w_{ij}}{2w} \log \frac{w_i}{2w}$$

$$= H\left(\cdots, \frac{w_{ij}}{2w}, \ldots\right) - H\left(\cdots \frac{w_i}{2w}\right)$$

Source code:

```
clc; close all; clear all;
a=[0 1 2 1; 1 0 1 0; 2 1 0 1; 1 0 1 0];
sum=0; w1=0; w2=0; w3=0; w4=0;
for i = 1:4
    w1 = w1 + a(1,i);
    w2 = w2 + a(2,1);
    w3 = w3 + a(3,1)
    w4 = w4 + a(4,i);
 for j=1:4
      sum = sum + a(i,j)
 end
end
w = sum/2;
 wi = [w1 w2 w3 w4];
mue = [w1/(2*w) w/(2*w) w3/(2*w) w4/(2*w)];
```

w
wi
mue

```
k=1;
for i=1:4
    H2(i)=mue(i)
     for j=1:4
        H1(k)=[a(i,j)/sum];
        k=k+1;
end
end
H1
H2
ent1=0;
ent2=0
for k=1:16
if H1(k)==0
 continue;
end
 ent1=(ent1+H1(k)*log2(H1(k)));
end
for k=1:4
    ent2=ent2+H2(k).*log2(H2(k));
end
ent1=ent1*(-1)
ent2=ent2*(-1);
ent1
ent2
Entropy_rate=ent1-ent2
Entropy_rate
```

output:

W = 6

Wi = 4 2 4 2

mue = 0.3333 0.1667 0.3333 .1667

H1 = .

column 1 through 10

0 0.0833 0.1667 0.0833 0.0833 0

0.0833 0 0.1667 0.0833

Cloumns 11 through 16

0 0.0833 0.0833 . 0 0.8330

H2

0.8333 0.1667 0.333 0.1667

ent 1 = 3.2516

ent 2 = 1.9183

entropy rate = 1.3333