Experiment No: 01

Experiment Name: Write a program to implement Huffman code using Symbols with their corresponding probabilities.

Theory:

Huffman coding is a technique of compressing data to reduce its size without losing any of the details. Huffman coding is generally useful to compress the data in which there are frequently occurring characters.

suppose the string below is to be sent over a network.

BCAADDDCCACACAC

Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of $8 \times 15 = 120$ bits are required to send this string.

Huffman coding first creates a tree using frequencies of the character and then generates code for each character.

Once the data is encoded, it has to be decoded. Decoding is done using the same tree.

Huffman coding is done with the help of the following steps.

1. Calculate the frequency of each character in the string.

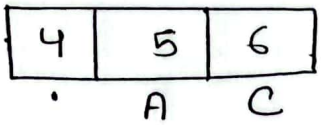
1	6	5	3
B	C	A	D

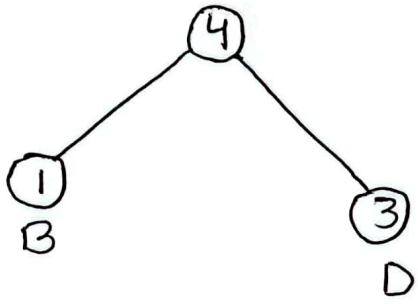
2. Sort the characters in increasing order of the frequency. These are stored in a priority queue.

1	3	5	6
B	D	A	C

3. Make each unique character as a leaf node.

4. Create an empty node(Z). Assign the minimum frequency to the left child of Z and assign the second minimum frequency to the right child of Z. Set the value of the Z as the sum of the above two minimum frequencies.

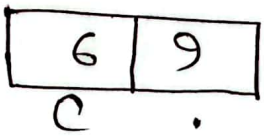


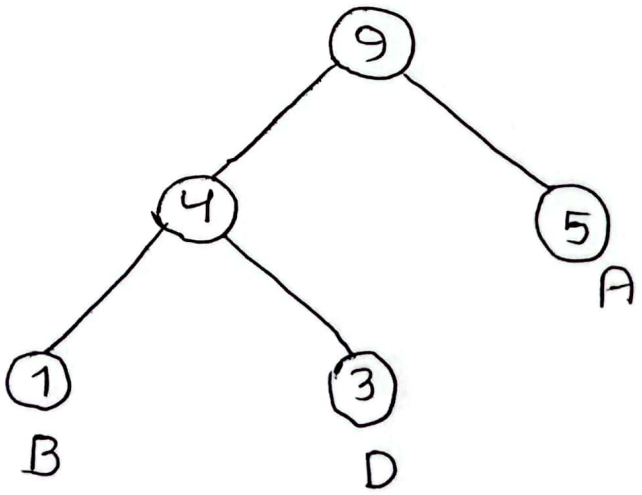


5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies.

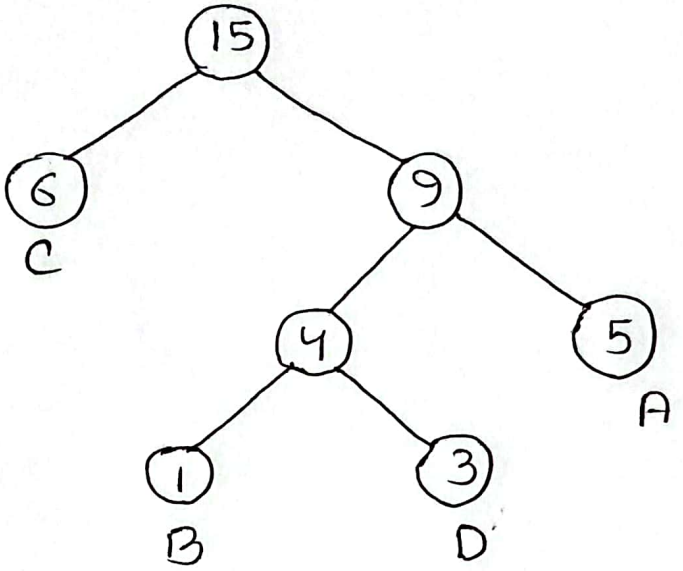
6. Insert node z into the tree.

7. Repeat steps 3 to 5 for all the characters.

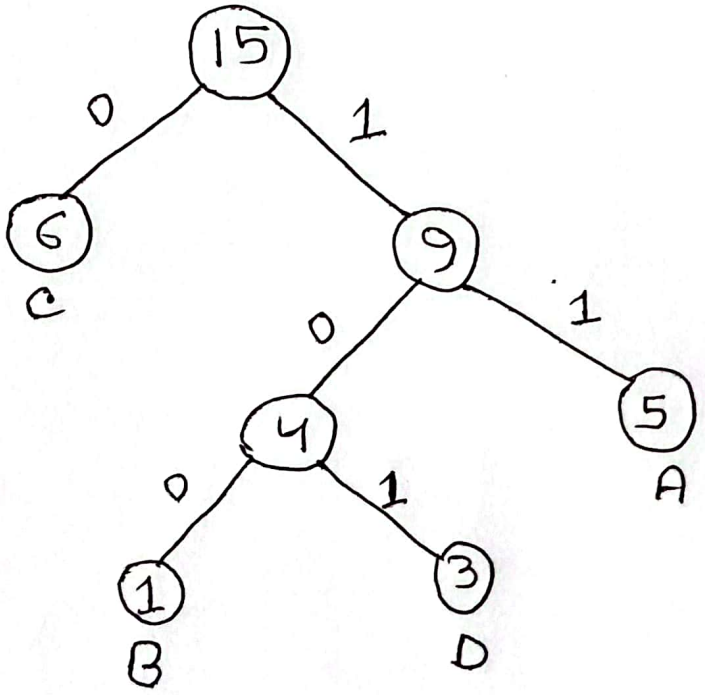




15



8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge.



For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

character	Frequency	code	size
A	5	11	5×2=10
B	1	100	1×3=3
C	6	0	6×1=6
D	3	101	3×3=9
4×8=32bits	15 bits		28 bits

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to

$32+15+28 = 75$

Decoding the code:

For decoding the code, we can take the code and traverse through the tree to find the characters.

Source code in python:

```
import heapq
from collections import Counter

def probabilities(message):
    message = message.upper().replace(' ','')
    freq = Counter(message)
    P = [fre/len(message) for fre in
                                freq.values()]
    return P, freq

def build_heap(freq):
    heap = [[weight/sum(freq.values()),
            [char, ''] for char, weight in
                                freq.items()]
    heapq.heapify(heap)
    return heap

def build_tree(heap):
    while len(heap)>1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        for pair in left[1:]:
            pair[1] = '0' + pair[1]
        for pair in right[1:]:
            pair[1] = '1' + pair[1]
```

```
        heapq.heappush(heap,[left[0]+right
        [0]]+left[1:]+right[1:])
    return heap[0]

message='bcaaadddccacacac'
P,freq=Probabilities(message)
Print('Probabilities :',P)
heap=build_heap(freq)
tree=build_tree(heap)
print('Tree:',tree)
for pair in tree[1:]:
    print(pair[0],'->',pair[1])
```

Output:

Probabilities : [0.0667, 0.4, 0.333, 0.2]

Tree [1,0,['C','0'],['B','100'],['D','101'],
['A','11']]

C -> 0

B -> 100

D -> 101

A -> 11

Experiment No: 02

Experiment Name: Write a program to simulate convolutional coding based on their encoder structure.

Theory:

Convolution codes or Trellis code introduce memory into the coding process to improve the error-correcting capabilities of the codes. The coding and decoding process 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.

Say, we have a message source that generates a sequence of information digits $\{U_K\}$. We will assume that the information digits are binary 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. In the example its memory is 2 bits it Contains a shift-register that keeps stored the values of the last two information bits. The encoder has several modulo-2 adders. Hence the encoder rate is

$$R_t = \frac{1}{2} \text{ bits}$$

In general the encoder can take n_i information bits to generate n_c codeword bits yielding on encoder rate of $R_t = \frac{n_i}{n_c}$ bits.

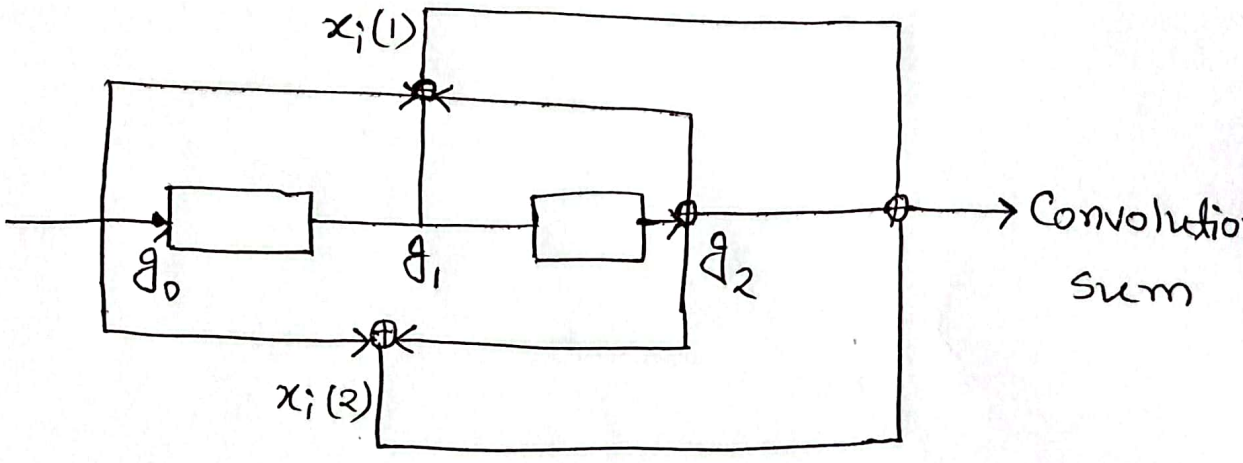


Fig-1: Convolution encoder

Here in the above diagram,

l = the message length

m = number of shift register

n = number of modulo-2 adders

Output = $n(m+l)$ bit

Code rate, $r = l/(n(m+l))$; $l >> m$

$= l/ln$

$= \frac{1}{n}$

Constraint length, $K = m+1$.

if $g^{(1)}_0 . g^{(1)}_1 . g^{(1)}_2 \cdots \cdots g^{(1)}_m$ are the state of shift register then the input top adder output path is given by

$$g^{(1)}_0 , g^{(1)}_1 , g^{(1)}_2 \cdots g^{(1)}_m$$

and the input-bottom adder output path is given by,

$$g^{(2)}_0 . g^{(2)}_1 . g^{(2)}_2 \cdots \cdots g^{(2)}_m$$

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

$$x_i^{(1)} = \sum_{l=0}^{m} g_l^{(1)} m_{i-l} \; ; \quad i = 0, 1, 2 \cdots n$$

and, then convolution sum for (1)

$$x_i^{(1)} = \sum_{l=0}^{m} g_l^{1} m_{i-l} \quad ; \quad i = 0, 1, 2 \cdots n$$

So, if output, $x_i = \{ x_0^{(1)} x_0^{(2)} x_1^{(1)} x_1^{(2)} x_2^{(1)} x_2^{(2)} \cdots \}$

Example:

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,

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

where, $j = 1$ and $i = 0$ then,

$$x_0^1 = g_0^1 m_0$$

$$= 1 \times 1$$

$$= 1 \% 2$$

$$= 1$$

$x_1^1 = g_0^1 m_1 + g_1^1 m_0 = 1\times0 + 1\times1 = 0+1 = 1\%2 = 1$

$x_2^1 = g_0^1 m_2 + g_1^1 m_1 + g_2^1 m_0 = 1\times0 + 1\times0 + 1\times1 = 1\%2 = 1$

$x_3^1 = g_0^1 m_3 + g_1^1 m_2 + g_2^1 m_1 = 1\times1 + 1\times0 + 1\times0 = 1\%2 = 1$

$x_4^1 = g_0^1 m_4 + g_1^1 m_3 + g_2^1 m_2 = 1\times1 + 1\times1 + 1\times0 = 2\%2 = 0$

$x_5^1 = g_1^1 m_4 + g_2^1 m_3 = 1\times1 + 1\times1 = 1+1 = 2\%2 = 0$

$x_6^1 = g_2^1 m_5 = 1\times1 = 1\%2 = 1$

$\therefore x_i^1 = 1111001$

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

$$x_0^2 = g_0^2 m_0 = 1\times1 = 1\%2 = 1$$

Then for successive i, we got:

$x_1^2 = g_0^2 m_1 + g_1^2 m_0 = 1\times0 + 0\times1 = 0 = 0\%2 = 0$

$x_2^2 = g_0^2 m_2 + g_1^2 m_1 + g_2^2 m_0 = 1\times0 + 0\times0 + 1\times1 = 1\%2 = 1$

$x_3^2 = g_0^2 m_3 + g_1^2 m_2 + g_2^2 m_1 = 1\times1 + 0\times0 + 1\times0 = 1\%2 = 1$

$x_4^2 = g_0^2 m_4 + g_1^2 m_3 + g_2^2 m_2 = 1\times1 + 0\times1 + 1\times0 = 1\%2 = 1$

$x_5^2 = g_1^2 m_4 + g_2^2 m_3 = 0\times1 + 1\times1 = 1\%2 = 1$

$x_6^2 = g_2^2 m_5 = 1\times1 = 1\%2 = 1$

$\therefore x_i^2 = 1011111$

$x_i = 11101111010111$

Source Code in Python:

```
import numpy as np
def encode(msg, k, n):
    g, v = [], []
    for i in range(n):
        sub_g = list(map(int, input('Enter
                bits for generator{i}: ').split
                ())))
        if len(sub_g) != k:
            raise valueError(f'You entered
            {len(sub_g)}bits.\n need to enter
                                {k} bits')
        g.append(sub_g)
    for i in range(n):
        res = list(np.poly1d(g[i]) * np.poly1d(msg))
        v.append(res)
    listMax = max(len(l) for l in v)
    for i in range(n):
        if len(v[i]) != listMax:
            tmp = [0] * [listMax - len(v[i])]
            v[i] = tmp + v[i]
    res = []
    for i in range(listMax):
        res += [v[j][i] %2 for j in range(n)]
    return res
```

```
message = list(map(int, input('Enter message:
                                    split()))
k = int(input('Constraints : '))
n = int(input('Number of output(generator)
print('Encoded Message', encode(message,
                                    k, n))
```

Output:

```
Enter message: 1 0 0 1
Constraints: 3
Number of output(generator): 2
Enter bits for generator 0: 1 1 1
Enter bits for generator 1: 1 0 1
Encoded Message: [1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0
                                    1, 1, 1]
```

Experiment No: 03

Experiment Name: Write a program to implement Lempel-Ziv Code.

Theory:

Lempel-Ziv coding is a lossless data compression algorithm that works by finding repeated sequence of data in a stream and replacing them with references to a dictionary of previously encountered sequences.

Basic steps of encoding a given data sequence of letters by lempel Ziv Coding.

1. It is accomplised by dividing data into sequently.
2. These segments are shortest sequence no encountered ~~priously~~ previously.
3. All segments return as phrase.
4. we encode by phrase by prefix and last bit.
5. Symbol A → 0 , B → 1

Example:

A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
1	2	3	4	5	6	7	8	9

A = 0 , B = 1

Position:	1	2	3	4	5	6	7	8	9
Sequence:	A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
Numerical representation:	φA	1B	2B	φB	2A	5B	4B	3A	7
code :	0000	0011	0101	0001	0100	1011	1001	0110	0111

Source code in python:

```
import string
message = 'AABABBBABAABABBBABBABB'
dictionary = { }
tmp, i, last = '', 1, 0
Flag = True
for x in message:
    tmp += x
    Flag = False
    if tmp not in dictionary.keys():
        dictionary[tmp] = i
        tmp = ''
        i += 1
        Flag = True
```

```
if not Flag:
    last = dictionary[tmp]
res []
for char, idx in dictionary.items():
    ~~tem~~,
    tmp, s = '' , ''
    for x, j in zip(char[:-1], range
                                (len(char))):
        tmp += x
        if tmp in dictionary.keys():
            take = dictionary [tmp]
            s = str(take) + char[j+1:]
    if len(char) == 1:
        s = char
    res.append(s)
if last:
    res.append(str(last))
alphabet = string.ascii_uppercase
mark = dict(zip(alphabet, range(len(alphabet
finalres = []
for x in res:
    tmp = ""
    for char in x:
        if char.isalpha():
            tmp += bin(mark[char])[
        else:
```

```
            tmp += bin(int(char))[2:]
        final_res.append(tmp.zfill(4))

print(res)
print("Encoded : ", final_res)
```

Output:

```
['A', '1B', '2B', 'B', '2A', '5B', '4B', '3A', '7']
Encoded: ['0000', '0011', '0101', '0001', '0100',
          '1011', '1001', '0110', '0111']
```

Experiment No: 04

Experiment Name: Write a program to implement Hamming Code.

Theory:

Hamming code method is one of the most effective ways to detect single-data bit errors in the original data at the receiver end. It is not only used for error detection but is also for correcting errors in the data bit.

Redundant Bits - These are the extra binary bits added externally into the original data bit to prevent damage to the transmitted data and are also needed to recover the original data.

The expression applied to deduce the redundant value is,

$2^r >= d + r + 1$

where,

d = data bits

r = redundant bits, $r = 1, 2, 3 \ldots n$

Here,

The number of data bits is 7,

$2^r \geq r + 7 + 1$

$2^r \geq r + 8$

$2^4 \geq 4+8$

The number of redundant bits = 4

Parity Bits - The parity bit is the method to append binary bits to ensure that the total count of 1's in the original data is even bit or odd. It is also applied to detect errors on the receiver side and correct them.

Types of parity bits:

Odd parity bits - In this parity type, the total number of 1's in the data bit should be odd in count, then the parity value is 0, and the value is 1.

Even parity bits - In this parity type, the total number of 1's in the data bit should be even in count, then the parity value is 0, and the value is 1.

To solve the data bit issue with the hamming code method, some steps need to be followed:

Step-1: The position of the data bits and the number of redundant bits in the original data. The number of redundant bit is deduced from the expression $[2^r \geq d+r+1]$

Step-2: Fill in the data bits and redundant bit, and find the parity bit value using the expression $[2^p$, where, $p=0,1,2---n]$

Step-3: Fill the parity bit obtained in the original data and transmit the data to the receiver side.

Step-4: Check the received data using the parity bit and detect any error in the data and in case damage is present, use the parity bit value to correct the error.

Example:

The data bits = 7

The redundant bit,

$2^r \quad 2^r \geq d+r+1$

$\Rightarrow 2^4 \geq 7+4+1$

$\Rightarrow 16 \geq 12$ [So, the value of $r=4$]

position of the redundant bit, applying the 2^P expression:

$2^0 - P_1$

$2^1 - P_2$

$2^2 - P_4$

$2^3 - P_8$

11	10	9	8	7	6	5	4	3	2	1
1	0	1	P_8	1	0	1	P_4	0	P_2	P_1

Finding the parcity Bits for Even parcity bits,

1. P_1 parcity bit is deduced by checking all the bits with 1's in the least Significant location.

P_1: 1, 3, 5, 7, 9, 11

P_1: P_1, 0, 1, 1, 1, 1

P_1: 0

2. P_2 parcity bit is deduced by checking all the bits with 1's in the second Significant location.

P_2: 2, 3, 6, 7, 10, 11

P_2: P_2, 0, 0, 1, 0, 1

P_2: 0

3. P_4 parity bit is deduced by checking all the bits with 1's in the third significant location.

P_4 : 4, 5, 6, 7

P_4 : P_4, 1, 0, 1

P_4 : 0

4. P_8 Parity bit is deduced by checking all the bits with 1's in the fourth significant location.

P_8 : 8, 9, 10, 11

P_8 : P_8, 1, 0, 1

P_8 : 0

So, the original data to be transmitted to the receiver side is:

11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	1	0	0	0	0

Source code in python:

```
def isPowerOfTwo(n):
    if n==0:
        return False
    if n & (n-1)==0:
        return True

inp = input('Enter Input (hamming code):')
hamming = ''
length =len(inp)

r=0
for i in range(length):
    if 2**i==length+i+1:
        r=i
        break

k=0
for i in range(1,length+r+1):
    if isPowerOfTwo(i):
        hamming += 'P'
    else:
        hamming += inp[k]
        k+=1

print('Position generate parity bit:',hamming)
```

```
res=0
for i in range(len(hamming)):
    if hamming[i]=='1':
        res^=(i+1)
P=bin(res)[2:].zfill(r)[::-1]
k=0
hamming_list=list(hamming)
for i in range(len(hamming)):
    if hamming_list[i]=='P':
        hamming_list[i]=P[k]
        k+=1
hamming=''.join(hamming_list)
print('Hamming Code :',hamming)

rcv=input('Enter Received Code: ')
res=0
for i in range(len(rcv)):
    if rcv[i]=='1':
        res^=(i+1)
if res==0:
    print('No Error')
else:
    print('Error at :',res)
```

Output :

```
Enter Input(hamming code):0110
Position generate parity bit : ppop110
Hamming code : 1100110
Enter Received Code : 1100111
Error at : 7
```

Experiment No: 05

Experiment Name: A binary symmetric channel has the following noise matrix with probability,

$$P(Y/X) = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

Now find the channel capacity C.

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 bit (a zero or a one) and the receiver will receive a bit. It is assumed that the bit is usually transmitted correctly, but that it will be "flipped" with a small probability ("the crossover probability"). A binary symmetric channel with crossover probability p and denoted by is a channel with binary input and binary output and probability of error p, that is if x is the transmitted

random variable and Y the received variable, then the channel is characterized by the ~~ad~~ conditional probabilities.

$P_r(Y=0|x=0) = 1-P$ $\quad P(Y=1|x=0) = P$

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

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 whenitsees 0, and vice versa) and obtain an equivalent channel with crossover probability $1-P \leq \frac{1}{2}$.

2. This channel is often used by theorists because it is one of the simplest noisy channels to analyze many problems in communication theory can be reduced to a binary symmetric channel.

3. Conversely, being able to transmit effectively over the bsc can give rise to solutions for more complicated channels.

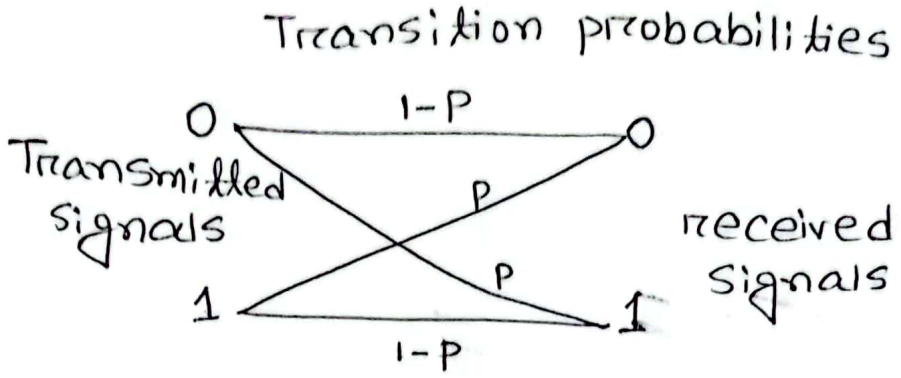


Fig-1: Binary symmetric channel

This is a binary symmetric channel in which the input symbols are complemented with probability. The information capacity of a binary Symmetric channel with parameter P is,

$$C = 1 - H(P) \text{ bits} \quad \text{-----(i)}$$

Here, Conditional probability H(P) on H(x/y) calculate using following formula,

$$H(P) = (1-P)\log\frac{1}{1-P} + P\log\frac{1}{P} \quad \text{----(ii)}$$

Source code in Python:

```
import math
matrix = [[2/3, 1/3], [1/3, 2/3]]
print("Symmetric matrix is : ")
for i in range(0, 2):
    for j in range(0,2):
        print('%.2f ' % matrix[i][j], end='')
    print()
Hp = matrix[0][0] * math.log2(1.0/matrix[0][0]
    + matrix[0][1] * math.log2(1.0/matrix[0][1]
print("Conditional probability H(Y/x) is =
        %.3f" % Hp, "bits/msg symbol")
C = 1-Hp
print("Channel Capacity is =%.3f " %C,
        "bits/msg symbol")
```

Output:

```
Symmetric matrix is :
0.67  0.33
0.33  0.67
Conditional probability H(Y/x) is = 0.918 bits/msg symbo
Channel Capacity is = 0.082 bits/msg symbol
```

Experiment No: 06

Experiment Name: Write a program to check the optimality of Huffman code.

Theory:

To check the optimality of a huffman code, we can use the Kraft inequality, which states that the sum of all the code word lengths (in bits) raised to the power of -1 must be less than or equal to 1. If a codes satisfies the Kraft inequality, it is considered optimal. Another way to check the optimality of Huffman code is to verify that the 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 Huffman code is by comparing the average length of the Huffman code with the entropy if its average length is equal to the entropy of the source. In all cases, if the code is not optimal

We can use the standard algorithm for constructing a huffman code to generate a new, optimal code.

Source code in python:

```
import heapq
import math
from collections import Counter

def calculate_frequency(my_text):
    my_text = my_text.upper().replace(' ','')
    frequency = dict(Counter(my_text))
    return frequency

def build_heap(freq):
    heap = [[weight, [char, ""]] for char,
            weight in freq.items()]
    heapq.heapify(heap)
    return heap

def build_tree(heap):
    while len(heap)>1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
```

```
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [lo[0] +
            hi[0]] + lo[1:] + hi[1:])
    return heap[0]
def compute_huffman_avg_length(freq, tree
                               length):
    huffman_avg_length = 0
    for pair in tree[1:]:
        huffman_avg_length += (len(pair[1]*
        (freq[pair[0]] / length))
    return huffman_avg_length

def entropy(freq, length):
    H = 0
    P = [fre/length for _, fre in freq.items()
    for x in p:
        H += -(x * math.log2(x))
    return H
```

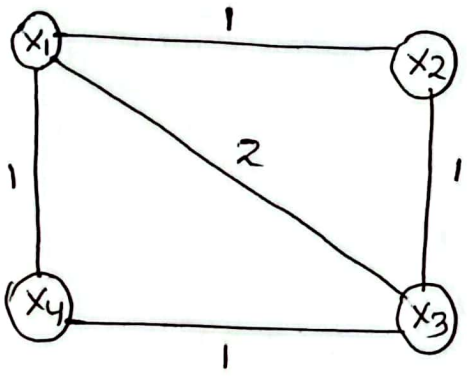
```
message = 'aaaabbbbbccccccddddee'
freq = calculate_frequency (message)
heap = build_heap(freq)
tree = build_tree(heap)
huffman_avg_length = compute_huffman_avg
                     length (freq, tree,
                         len (message))
H = entropy(freq, len(message))
print ("Huffman : %.2f bits" % huffman_avg-
                              length )
print ("Hu. Entropy: %.2f bits" % H )
if huffman_avg_length >= H
    print ("Huffman code is optimal")
else:
    print ("Code is not optimal")
```

Output:

```
Huffman: 2.25 bits
Entropy: 2.23 bits
Huffman code is optimal
```

Experiment NO: 08

Experiment Name: Write a code to find the entropy rate of a random walk on the following weighted graph.



Theory:

The entropy of a stochastic process $\{x_i\}$ is defined by.

$$H(X) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2 \cdots X_n)$$

when the limit exists.

Consider a graph with m nodes labeled $\{1, 2 \cdots m\}$ with weight $w_{ij} \geq 0$ on the edge joining node i to node j. A particle walks randomly from node to node in this graph. The random walk $\{X_n\}$. $X_n \in \{1, 2, \cdots m\}$ is a sequence of vertices of the graph. Given $X_n = i$, the

next vertex j is chosen from among the nodes connected to node i with a probability proportional to the weight of the edge connecting i to j. Thus, $P_{ij} = W_{ij} / \Sigma_k W_{ik}$.

The stationary distribution for markov chain assigns probability to node i proportional to the total weight of the edges emanating from node i. Let,

$$W_i = \sum_j W_{ij}$$

be the total weight of edges emanating from node i and let,

$$W = \sum_{i,j \; j>i} W_{ij}$$

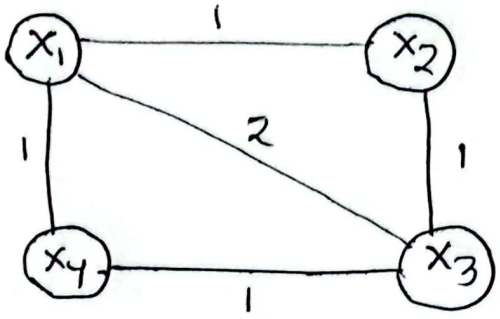
be the sum of the weights of the edges. Then $\sum_i W_i = 2W$. We now guess that the stationary distribution is

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

The entropy rate,

$$H(\mathcal{X}) = H\left(\frac{W_{ij}}{2W}\right) - H\left(\frac{W_i}{2W}\right)$$

Solution:



There are four nodes x_1, x_2, x_3, x_4.
So W_1, W_2, W_3 and W_4.

$$W_1 = \sum_j W_{ij}$$
$$= 1+2+1$$
$$= 4$$

$$W_2 = \sum_j W_{ij}$$
$$= 1+1$$
$$= 2$$

$$W_3 = \sum_j W_{ij}$$
$$= 2+1+1$$
$$= 4$$

$$W_4 = \sum_j W_{ij}$$
$$= 1+1$$
$$= 2$$

$$\Sigma_i W_i = 2W$$
$$\Rightarrow W = \frac{\Sigma_i W_i}{2}$$
$$= \frac{4+2+4+2}{2}$$
$$= \frac{12}{2}$$
$$\therefore W = 6$$

the stationary distribution is,

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

$$= \frac{W_1, W_2, W_3, W_4}{2W}$$

$$= \frac{W_1}{2W}, \frac{W_2}{2W}, \frac{W_3}{2W}, \frac{W_4}{2W}$$

$$= \frac{4}{12}, \frac{2}{12}, \frac{4}{12}, \frac{2}{12}$$

$$= \frac{1}{3}, \frac{1}{6}, \frac{1}{3}, \frac{1}{6}$$

$$= 0.333, 0.1667, 0.333, 0.1667$$

$$H\left(\frac{W_i}{2W}\right) = \mu_i$$

$$= (0.333, 0.1667, 0.333, 0.1667)$$

$$H\left(\frac{W_i}{2W}\right) = -\sum_{i=1}^{4} P(x) \log P(x)$$

$$= -0.333 \log_2(0.333) - 0.1667 \log(0.1667)$$
$$- 0.333 \log_2(0.333) - 0.1667 \log(0.1667)$$

$$= 0.5282 + 0.43086 + 0.5282 + 0.43086$$

$$= 1.918$$

$$H\left(\frac{W_{ij}}{2W}\right) = \frac{W_1}{2W}, \frac{W_2}{2W}, \frac{W_3}{2W}, \frac{W_4}{2W}$$

$$= \left(\frac{0}{12}, \frac{1}{12}, \frac{2}{12}, \frac{1}{12}\right), \left(\frac{1}{12}, \frac{0}{12}, \frac{1}{12}, \frac{0}{12}\right)$$

$$\left(\frac{2}{12}, \frac{1}{12}, \frac{0}{12}, \frac{1}{12}\right), \left(\frac{1}{12}, \frac{0}{12}, \frac{1}{12}, \frac{0}{12}\right)$$

$$= 0 \quad 0.0833 \quad 0.1667 \quad 0.0833 \quad 0.0833$$

$$0 \quad 0.0833 \quad 0 \quad 0.1667 \quad 0.0833$$

$$0 \quad 0.0833 \quad 0.0833 \quad 0 \quad 0.0833$$

$$0$$

$$H\left(\frac{W_{ij}}{2W}\right) = -\sum_{i=1}^{12} P(x) \log P(x)$$

$$= -0\log_2 0 - 0.0833\log_2(0.0833) - 0.1667\log_2(0.1667$$

$$-0.0833\log_2(0.0833) - 0.0833\log_2(0.0833)$$

$$-0\log_2 0 - 0.0833\log_2(0.0833) - 0\log_2 0$$

$$-0.1667\log_2(0.1667) - 0.0833\log_2(0.0833$$

$$-0\log_2 0 - 0.0833\log_2(0.0833) -$$

$$0.0833\log_2(0.0833) - 0\log_2 0 - 0.0833 \times$$

$$\log_2(0.0833) - 0\log_2 0$$

$$= 0.2986 + 0.43086 + 0.2986 + 0.2986 +$$

$$0.2986 + 0.43086 + 0.2986 + 0.2986 +$$

$$0.2986 + 0.2986$$

$$= 3.251$$

Entropy rate,

$$H(x) = H\left(\frac{W_{ij}}{2W}\right) - H\left(\frac{W_i}{2W}\right)$$

$$= 3.251 - 1.918$$

$$= 1.333$$

Source code in python:

```
import math
from collections import defaultdict

g = defaultdict(list)
xij = [[1,1,2], [1,1], [1,2,1], [1,1]]

def makegraph(li):
    for node in range(len(li)):
        for x in li[node]:
            g[node].append(x)

def entropy(li):
    H = 0
    for x in li:
        if x == 0:
            continue
        H += -(x * math.log2(x))
    return H
```

```
makeGraph(xij)
Wi = []
for node in range(len(g)):
    Wi.append(sum(g[node]))
W = sum(Wi)/2
ui = [weight/(2*W) for weight in Wi]
H_Wi_div_2W = entropy(ui)
Wij_div_2W_list = []
for i in range(len(g)):
    Wij_div_2W_list += [weight/(2*W) for
                        weight in g[i]]
H_Wij_div_2W = entropy(Wij_div_2W_list)
H_X = H_Wij_div_2W - H_Wi_div_2W
print('Entropy Rate: %.2f' % H_X)
```

output:

Entropy Rate: 1.33

Experiment No: 08

Experiment Name: Write a program to find Conditional entropy and joint entropy and mutual information based on the following matrix.

Y \ X	1	2	3	4
1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$
2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$
3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
4	$\frac{1}{4}$	0	0	0

Theory:

Entropy: Entropy is a measure of uncertainty of a random variable.

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

Conditional Entropy: If $(X, Y) \sim P(x, y)$ then the conditional entropy $H(Y/X)$ is defined as,

$$H(Y/X) = \sum_{x \in \mathcal{X}} P(x) H(Y/X=x)$$

$$H(Y/x) = \sum_{x \in X} P(x) \left\{ - \sum_{y \in Y} P(Y/x) \log P(Y/x) \right\}$$

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

$$= - E_{P(x,y)} \log P(Y/x)$$

Similarly,

$$H(X/Y) = - E_{P(x,y)} \log P(X/Y)$$

Joint Entropy: The point Entropy $H(X,Y)$ of a pair of discrete random variables (X, Y) with a joint distribution $P(x, y)$ is defined as,

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

$$H(X,Y) = H(X) + H(Y/x)$$

Mutual Information: The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution $P(x)$, $P(y)$,

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x) P(y)}$$

$$I(X;Y) = H(Y) - H(Y/x)$$

solution:

Y \ X	1	2	3	4
1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$
2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$
3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
4	$\frac{1}{4}$	0	0	0

The marginal distribution of $x =$

$\left(\frac{1}{8}+\frac{1}{16}+\frac{1}{16}+\frac{1}{4}, \frac{1}{16}+\frac{1}{8}+\frac{1}{16}+0, \frac{1}{32}+\frac{1}{32}+\frac{1}{16}+0, \frac{1}{32}+\frac{1}{32}+\frac{1}{16}+0\right)$

$= \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right)$

The marginal distribution of $Y = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$

$$H(X) = -\sum_{i=1}^{4} P(x) \log P(x)$$

$$= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8}$$

$$= -\frac{1}{2}(-1) - \frac{1}{4}(-2) - \frac{1}{8}(-3) - \frac{1}{8}(-3)$$

$$= \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8}$$

$$= \frac{4+4+3+3}{8}$$

$$= \frac{14}{8}$$

$$= \frac{7}{4}$$

$$H(Y) = -\sum_{i=1}^{4} P(x) \log P(x)$$

$$= -\frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4}$$

$$= -\frac{1}{4}(-2) - \frac{1}{4}(-2) - \frac{1}{4}(-2) - \frac{1}{4}(-2)$$

$$= \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}$$

$$= 2$$

Conditional Entropy:

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

$$= P(Y=1) H(X/Y=1) + P(Y=2) H(X/Y=2) + P(Y=3) H(X/Y=3)$$
$$+ P(Y=4) H(X/Y=4)$$

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

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

$$= \frac{7}{16} + \frac{7}{16} + \frac{1}{2}$$

$$= \frac{7+7+8}{16}$$

$$= \frac{22}{16}$$

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

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

$$= P(X=1)H(Y/X=1) + P(X=2)H(Y/X=2) +$$
$$P(X=3)\,H(Y/X=3) + P(X=4)\,H(Y/X=4)$$

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

$$= \frac{1}{2} \times \frac{7}{4} + \frac{1}{4} \times \frac{3}{2} + \frac{1}{8} \times \frac{3}{2} + \frac{1}{8} \times \frac{3}{2}$$

$$= \frac{7}{8} + \frac{3}{8} + \frac{3}{16} + \frac{3}{16}$$

$$= \frac{14+6+3+3}{16}$$

$$= \frac{26}{16}$$

$$= \frac{13}{8}$$

Joint Entropy:

$$H(x,y) = H(X) + H(Y/X)$$

$$= \frac{7}{4} + \frac{13}{8}$$

$$= \frac{14+13}{8}$$

$$= \frac{27}{8}$$

Mutual Information:

$$I(X;Y) = H(Y) - H(Y/X)$$

$$= 2 - \frac{13}{8}$$

$$= \frac{16-13}{8}$$

$$= \frac{3}{8}$$

Source code in python:

```
import math

matrix = [
        [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]
]
marginal_x = []
for i in range(len(matrix[0])):
    marginal_x.append(sum(matrix[j][i]
        for j in range(len(matrix))))
marginal_y = []
for i in range(len(matix)):
    marginal_y.append(sum(matrix[i][j] for
        j in range(len(matrix[0]))))
```

```
def entropy(marginal_var):
    H = 0
    for x in marginal_var:
        if x == 0:
            continue
        H += -(x * math.log2(x))
    return H

H_x = entropy(marginal_x)
H_y = entropy(marginal_y)

H_xy = 0
for i in range(len(matrix)):
    tmp = [(1/marginal_y[i]) * matrix[i][
        for j in range(len(matrix[0]))]
    H_xy += entropy(tmp * marginal_y[i

H_yx = 0
for i in range(len(matrix[0])):
    tmp = [(1/marginal_x[i]) * matrix[i][j
        for j in range(len(matrix))]
print('Conditional Entropy H(X|Y): ', H_xy)
print('Conditional Entropy H(Y/x): ', H_yx)
```

```
H_of_xy = H_x + H_yx
print('Joint Entropy H(x,y) : ', H_of_xy)

I_of_xy = H_y - H_yx
print('Mutual Information: ', I_of_xy)
```

output:

```
Conditional Entropy H(X/Y) : 1.375
Conditional Entropy H(Y/X) : 1.625
Joint Entropy H(X,Y) : 3.375
Mutual Information : 0.375
```