Ex NO : 6

10/8/24

Practical - 6.

Aim :

write a program to implement error detection & correction using HAMMING code concept. Make a test run to input data stream & verify error correction features.

Error correction at data link layer :

Hamming code is a set of link error detection codes that can be used to detect & correct the errors that can occur when the data is transmitted from the sender to the receiver. Developed by R.W hamming for error corrections.

Create Sender program :

1) Input to sender file should be a text

of any length. program should convert the text to binary.

2) Apply hamming code on the binary data to check for errors.

3) If there is an error, concept output is a file called channel.

create receiver program:

1) receiver program should read the input from channel file.

2) Apply hamming code on the binary data to check for errors.

3) If there is an error, display the position of the error.

4) Else remove the redundant bits & convert binary data to ascii & display the output.

Student's observation:

write the code here.

Sender.py (filename)

```
import os

def text_to_binary(text):
    return ''.join(format(ord(char), '08b')
                   for char in text)

def calculate_redundant_bits(m):
    r = 0
    while(2**r) < (m+r+1):
        r += 1
    return r

def position_redundant_bits(data, r):
    j = 0
    k = 1
    m = len(data)
    res = ''
    for i in range(1, m+r+1):
        if i == 2**j:
            res += '0'
            j += 1
        else:
            res += data[-k]
            k += 1
    return res[::-1]
```

```
def calculate_parity_bits(arr, r):
    n = len(arr)
    for i in range(r):
        val = 0
        for j in range(1,n+1):
            if j & (2**i) == (2**i):
                val = val ^ int(arr[-j])
        arr = arr[:n-(2**i)] + str(val) +
arr[n-(2**i)+1:]
    return arr.

def apply_hamming_code(data):
    m = len(data)
    r = calculate_redudant_bits(m)
arranged_data = position_redundant_bits(data,r)
hamming_code = calculate_parity_bits
(arranged_data, r)
return hamming_code.

def save_to_channel(hamming_code)
    with open('channel','w') as file:
        file.write(hamming_code)
```

```
if __name__ == "__main__":

    text = input("Enter the text: ")

    binary_data = text_to_binary(text)

    hamming_code = apply_hamming_code
                   (binary_data)

    save_to_channel(hamming_code)
```

receiver.py

```
def read_from_channel():
    with open('channel', 'r') as file:
        return file.read()

def calculate_redundant_bits
    r = 0
    while (2**r) < (m+r+1)
        r += 1
    return r

def detect_error(arr, nr):
    n = len(arr)
    res = 0
    for i in range(nr):
        val = 0
```

```
for j in range (n//2, n+1):
    if j & (2**j) == (2**j):
        val = val ^ int (arr[-j])
    res = res + val * (10**j)
return int (str(res), 2)

def correct_error (arr, pos):
    if pos >=1
    arr = arr[: len[arr] - pos] + str(1 - int(arr[
    len(arr) - pos]))
    return arr

def remove_redundant_bits
    n = len(arr)
    j = 0
    res = 1,1
    for i in range (1, n+1)
        if (i != 2**j)
        res! = Arr[-i];
    else
        j+=1
    return res[::-1]
```

```
def binary_to_text (binary_data)
    text = ''
for i in range (0, len (binary_data), 8):
    byte = binary_data [i : i+8]
    text += chr (int (byte, 2))
return return text.
```

output :

Data transffered is 10101001110

Error Data is 11101001110

The position of error is 2 from the left

Result :

Thus the output is verified successfully.

18/9/24