Pratical-6

Aim: write a program to implement error detection and correction using Hamming Code concept. Make a test run to input data stream and verify error correction feature

Error Correction at Data Link layer:

Hamming code is a set of error-correction codes that can be used to detect and correction the errores it is a technique developed by R.W. Hamming for error correction.

Create Sender:

1. Input to Sender file Convert text to binary
2. Apply hamming code concept on the binary data
3. Save this concept in a file called Channel

Create Reciever

1. Reciever program Should read input
2. Apply hamming code
3. if there is an error display the pos
4. Else remove the redundant bits

Code

Sender.py

```
def string_to_binary(input_string):
    return ''.join(format(ord(char), '08b') for char in input_string)

def calculate_redundant
        for i in range(m):
            if 2**i >= m+i+1:
def position_redundant_bit(data, r):
    j = 0
    k = 0
    m = len(data)
    res = ''
    positions = []
    for i in range(1, m+r+1):
        if i == 2**j:
            res = res + data[k]
            positions.append(f'R{i}')
            j += 1
        else:
            res = res + data[k]
            positions.append(f'D{i}')
            k += 1
    return res, positions
def calculate_parity_bits(arr, r):
    n = len(arr)
    for i in range(r):
        val = 0
```

```
for j in range (1, n+1):
    if arr[j-1] = 'R' and (j & (2**i))
                    == (2**i)):
        val = val ^ int(arr[j-1])
arr = arr[:(2**i)-1] + str(val)
        + arr[(2**i):]
return arr
def detect_error(arr, nr):
    n = len(arr)
    res = 0
    for i in range(nr):
        val = 0
        for j in range(1, n+1):
            if j & (2**i) == (2**i):
                val = val ^ int(arr[j-1])
        res = res + val * (10**i)
    return int(str(res), 2)
def binary_to_string(data):
binary_values = [binary_data[i:i+8] for
                i in range(0, len(binary_data
string_output = ''.join([chr(int(bv,2)) for
        bv in binary_values])
return string_output
```

output

Enter the String : h

Binary data : 01101000

Number of data bits : 8,

Number of redundant bits : 4

Encoded message with placeholders for redundant bits : RR0R110R1000

Positions of bits : ['R1', 'R2', 'D3', 'R4', 'D5', D6, D7, R8, D9, D10, D11, D12]

Data after adding the Parity bits : 010110110000

Error detected at Position : 6

Corrected data : 010010111000

String : h

Result

Thus the hamming code output recieved Sucessfully

16/8/24