Exp. No: 06

DATE: 16-08-24

# ERROR CORRECTION AT DATA LINK LAYER

## AIM:

Write a program to implement error detection & correction using HAMMING code concept. Make a test run to input data stream & 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 & correct the errors that can occur when the data is transmitted from the sender to reciever. ~~It is a techniq~~

## Create Sender Program with below features:

→ Input to sender file should be a text of any length. Program should convert the text ~~into~~ binary.

→ Apply hamming code concept on the binary data & add redundant bits to it.

→ Save this output in ~~a fil~~ a file called channel.

## Create a Reciever program with below features:

→ Reciever program should read the input from Channel file.

→ Apply Hamming code on the binary data to check for errors.

→ If there is an error, display the position of the error.

→ Else remove the redundant bits & convert the binary data ascii & display the input.

PROGRAM:

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

def position_redundant_bits (data, r):
    k =0
    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

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 ^= 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 ^= int(arr[j-1])
        res += val * (10 **i)
    return int(str(res), 2)
def string_to_binary(string):
    return ''.join(format(ord(char), '08b') for char in string)
def cal_redundant_bits(m):
    for i in range(m):
        if (2**i >= m+i+1):
            return i
def pos_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 calc_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 ^= int (arr[-j])
        arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
    return arr

def main():
    data_name = input ("Enter the data string to be transmitted: ").strip()
    binary_data = string_to_binary (data_name)
    m = len (binary_data)
    r = calc_redundant_bits (m)
    arr = pos_redundant_bits (binary_data, r)
    arr = calc_parity_bits (arr, r)
    print(f"Number of redundant bits (r) is : {r}")
    print(f"Data ~~transmitted~~ transferred with redundent bits is: {arr}")
    recieved = list (arr)
    error_pos = int (input (f"Enter the position to introduce an error
                                               (1 to {len(recieved)}): "))
    recieved [error_pos-1] = '1' if recieved[error_pos-1] == '0' else '0'
    recieved = ''.join (recieved)
    print(f"Recieved data with error introduced: {recieved}")
```

```
error_bit = detect_error (received, r)
if error_bit == 0;
    print (" No error detected. ")
else:
    print ( f" Error detected at position : {error_bit}")
    corrected = list (recieved)
    corrected[error_bit-1] = '1' if corrected [error_bit-1]==
    corrected = ''.join (corrected)
    print (f" Corrected data : {corrected} ")
if _name_ == " _main_ ":
    main ()
```

OUTPUT:

Binary representation of 'John Allan':

0100101001 101 111 0 11 010 000 110 111000 1000000 0010000 101 10110
11011000 11000 010110 1110

Hamming code with parity bits: [0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0]

Introducing a single-bit error for demonstration...

Error!

Enter the bit position (1-87) to introduce an error: 3

Error!

Hamming code with error: [0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,
0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1,
1, 1, 0]

Error detected at position: 3

Corrected Hamming Code: [0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0]

Final output after correcting Hamming code: 'John Allan'

20/8/24

RESULT:

Thus the given Hamming code is executed successfully & output is verified.