

EXPERIMENT – 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.

1001101'						
i'	power of 2	action	res'	y'	x'	
1	1=2**0=yes	insert '0'	'0'	1	1	
2	2=2**1=yes	insert '0'	'00'	2	1	
3	3=2**2[NO]	insert data	'001'	2	2	
4	4=2**2[YES]	insert 0	'0010'	3	2	
5	5=2**3	insert data	'00100'	3	3	
6	6=2**3	insert data	'001001'	3	4	
7	7=2**3	insert data	'0010011'	3	5	
8	8=2**3[YES]	insert '0'	'00100110'	4	5	
9	9=2**3[no]					

CODE:-

```
def calcRedundantBits(m):
    # Use the formula  $2^r \geq m + r + 1$ 
    for i in range(m):
        if( $2^{**i} \geq m + i + 1$ ):
            return i

def posRedundantBits(data, r):
    # Redundancy bits are placed at the positions
    j = 0
    k = 1
    m = len(data)
    res = ''
    # If position is power of 2 then insert '0' Else append the data
    for i in range(1, m+r+1):
        if(i ==  $2^{**j}$ ):
            res = res + '0'
            j += 1
        else:
            res = res + data[-1 * k]
            k += 1
    # The result is reversed since positions are counted backwards. ( $m + r + 1$ )
... 1)
    return res[::-1]
```

```

def calcParityBits(arr, r):
    n = len(arr)
    # For finding rth parity bit, iterate over

    # 0 to r - 1
    for i in range(r):
        val = 0
        for j in range(1, n + 1):

            # If position has 1 in ith significant
            # position then Bitwise OR the array value
            # to find parity bit value.
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])
                # -1 * j is given since array is reversed

        # String Concatenation
        # (0 to n - 2^r) + parity bit + (n - 2^r + 1 to n)
        arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
    return arr

def detectError(arr, nr):
    n = len(arr)
    res = 0

    # Calculate parity bits again
    for i in range(nr):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])

    # Create a binary no by appending
    # parity bits together.

    res = res + val*(10**i)

    # Convert binary to decimal
    return int(str(res), 2)

# Enter the data to be transmitted
data = '1011001'

# Calculate the no of Redundant Bits Required
m = len(data)
r = calcRedundantBits(m)

# Determine the positions of Redundant Bits
arr = posRedundantBits(data, r)

```

```

# Determine the parity bits
arr = calcParityBits(arr, r)

# Data to be transferred
print("Data transferred is " + arr)

# Stimulate error in transmission by changing
# a bit value.
# 10101001110 -> 11101001110, error in 10th position.

arr = '10101001110'
print("Error Data is " + arr)
correction = detectError(arr, r)
if(correction==0):
    print("There is no error in the received message.")
else:
    print("The position of error is ",len(arr)-correction+1,"from the left")

```

OUTPUT: -

```

main.py:
1: def calculateParityBits():
2:     # Parity bits calculation
3:     for i in range(0, n):
4:         if(i % m == 0):
5:             even[i] = 1
6:
7: def positionErrorInData(data, r):
8:     # Positioning error bits in the received data
9:     j = 0
10:    k = 0
11:    R = [None]
12:    for i in range(0, n+r-1):
13:        if(i % (m+r) == 0):
14:            R.append(data[i])
15:        else:
16:            R.append(0)
17:    return R
18:
19: def detectError(arr, r):
20:     res = 0
21:     for i in range(0, n+r-1):
22:         if(i % r == 0):
23:             res = res ^ arr[i]
24:     return res
25:
26: def calcParityBits(arr, r):
27:     n = len(arr)
28:     m = len(arr) // r
29:     even = [0] * n
30:     for i in range(0, n):
31:         val = 0
32:         for j in range(0, r-1):
33:             val = val ^ arr[i * r + j]
34:         even[i] = val
35:     arr = list(arr)
36:     for i in range(0, n):
37:         arr[i] = even[i]
38:     return arr
39:
40: arr = '10101001110'
41: print("Data transferred is 10101001110")
42: print("Error Data is 11101001110")
43: print("The position of error is 4 from the left")
44:
45: --- (Code Execution Successful) --+

```

RESULT: -

The code for HAMMING CODE have been executed successfully and the output is verified.



