Exp 6
Date : 20/8/24

## 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

## CODE

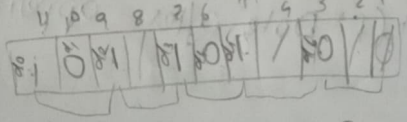
```
def calculate_parity_bits(data):
    P1 = (data[0] + data[2] + data[3] + data[5] +
          data[6]) % 2
    P2 = (data[0] + data[1] + data[3] + data[4] +
          data[6]) % 2
    P4 = (data[3] + data[4] + data[5]) % 2
    P8 = (data[0] + data[1] + data[2]) % 2
    return P1, P2, P4, P8

def parity_bits(data):
    P1 = (data[10] + data[8] + data[6] + data[4]
          + data[2] + data[0]) % 2
    P2 = (data[9] + data[8] + data[5] + data[4]
          + data[1] + data[0]) % 2
    P4 = (data[7] + data[6] + data[5] + data[4]) % 2
    P8 = (data[0] + data[3] + data[2] + data[1] % 2
    return P1, P2, P4, P8
```

```
def generate_hamming_code(data):
    P1, P2, P3, P4 = Calculate_parity_bits(data)
    hamming_code = [data[0], data[1], data[2],
                    P8, data[3], data[4], data[5]
                    P4, data[6], P2, P1]
    return hamming_code

def detect_error(hamming code):
    P1, P2, P4, P8 = parity_bits(hamming_code)
    error_position = P1*1 + P2*2 + P3*4 + P4*8
    return error position

data = []
print("enter 4 bits of data one by one:")
for i in range(7):
    bit = int(input(f"Bit {i+1}: "))
    data.append(bit)
print(f"Date after appending all bits : {data}")

hamming_code = generate_hamming_code(data)
print("The hamming Code ", ' '.join(st(bit) for
                                    bit in hamming-
                                    code))

Corrupt_code = []
print("enter 11 bit code with error")
```

```
for i in range(11):
    bit = int(input(f"Bit {i+1}: "))
    Corrupted_code.append(bit)

error_pos = detect_error(Corrupted_code)
print(f"calculated error position: {11-
                                    error_pos+1}")
if Corrupted_code[11-error_pos]==0:
    Corrupted_code[11-error_pos]=1
else:
    Corrupted_code[11-error_pos]=0
print("Data :" {Corrupted_code}")
```

Output

Enter 7 bits one by one:
Bit 1: 1
Bit 2: 0
Bit 3: 1
Bit 4: 1
Bit 5: 0
bit 6: 1
Bit 7: 0



Data after appending [1, 0, 1, 1, 0, 1, 0]
11 bit hamming code is 10101010000
Enter 11 bit hamming code with error

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

Calculated error position: 4

Data after error correcting : [1,0,1,0,1,0,1,0,0,0,0]

## Hamming Code for any String

### Code

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

def binary_to_string(binary_data):
    chars = []
    for i in range(0, len(binary_data), 8):
        byte = binary_data[i:i+1]
        chars.append(chr(int(byte,2)))
    return ''.join(chars)

def calculate_parity_bits(data):
    n = len(data)
    r = 0
    while (2**r) < (n+r+1):
        r += 1
    return r

def insert_parity_bits(data, r):
    n = len(data)
    j = 0
    k = 0
    m = n+r
    hamming_code = []
    for i in range(1, m+1):
        if i == 2**j:
            hamming_code.append(0)
            j += 1
        else:
            hamming_code.append(int(data[k]))
            k += 1
    return hamming_code
```

```
def detect_and_correct_error(hamming_code, r):
    n = len(hamming_code)
    error_position = 0
    for i in range(r):
        parity_pos = 2**i
        parity_val = 0
        for j in range(1, n+1):
            if j & parity_pos:
                parity_val ^= hammingcode[j-1]
        if parity_val != 0:
            error_position += parity_pos
    if error_position
        print(f"Error at : {error_position}")
        hamming_code[error_position-1] ^= 1
        printf" Corrected hamming code: {hamming_code}")
    else:
        print("No error detected")
    return hamming_code

def extract_data_from_hamming_code, r):
    j = 0
    data = []
    for i in range(1, len(hamming_code)+1):
        if i != 2**j:
            data.append(hamming_code[i-1])
        else:
            j += 1
    return ''.join(map(str, data))

def main():
    input_string = input("enter a string: ")
    binary_data = string_to_binary(input_string)
    print(f"Binary is '{input_string}' : {binary_data}")
```

```
r = calculate_parity_bits(binary_data)
hamming_code = insert_parity_bits(binary_data, r)
hamming_code = calculate_parity_values(hamming_
                                       code, r)
print(f"hamming code : {hamming_code}")
print("\n Introduce error")
error_bit = int(input(f"error the bit position
                      (1-{len(hamming_code)}) : "))
hamming_code[error_bit-1] ^= 1
print(f"hamming code with error : {hamming_code}
                                  ")
hamming_code = detect_and_error(hamming_code
                                , r)
Corrected_binary_data = extract_data_from
                        hamming(hamming_
                        code, r)
Corrected_string = binary_to_string(corrected_
                                    binary_data)
print(f"final output after correcting : '{Corrected_
                                          string}'")

if __name__ == "__main__":
    main()
```

## Output

enter a string : hi

Binary representation of 'hi' is 0110100001101001

hamming code with parity : [0,0,0,1,1,1,0,1,1,0,0,0,0,1,1,0,0,1,0,0,1]

introducing a single bit error

Enter the bit (1-21), to introduce an error : 2

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

correcting hamming code : [0,0,0,1,1,1,0,1,1,0,0,0,0,1,1,0,0,1,0,0,1]

final output after correcting 'hi'

20/8/24

## Result

Thus the program of error correction at datalink layer by hamming code is successfully executed & output is verified