20/08/24

EXPERIMENT NO: 06

ERROR DETECTION & CORRECTION AT DATA LINK LAYER (HAMMING CODE)

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 string_to_binary (input_string):
    return ''.join (format (ord (c), '08b') for c in input_string)
def binary_to_string (binary_data):
    char = []
    for i in range (0, len (binary_data), 8):
        byte = binary_data [i : i+1]
        char.append (char (int (byte, 2)))
    return ''.join (char)
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 and 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
        print (f" corrected hamming code: {hamming-code}")
    else:
        print (" No error detected")
    return hamming-code.
def extract_data_from_hamming-code (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'Enter 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: [0,0,0,1,1,1,0,1,1,0,0,0,0,1,1,0,0,0
Introducing a single bit error
Enter the a1(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'
```

[illegible] 20/8/24

Result:

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