Exp. no:

Practical - 06

Date : 29/08/2024

AIM :

To 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 correct the errors that can occur when the data is transmitted from sender to the receiver. It is a technique developed by. R. W. Hamming for error correction.

Sender program features :

1) Input to sender file should be a text of any length. Program should convert it to binary.

2) Apply hamming code concept on the binary data and redundant bits to it.

3) Save this output in a file called channel.

Receiver program features :

1) Receiver program should read the input from channel file.

2) Apply hamming code on the binary data to check for errors.

3) If there is an error, display the position of the error.

4) Else remove the redundant bits and convert the binary data to ascii and display the output.

```
import math
def char_to_binary(ch):
    binary = []
    for i in range(7, -1, -1):
        binary.append((ord(ch) >> i) & 1)
    return binary
def calculate_parity_bits(hamming_code, n, r):
    for i in range(r):
        parity_pos = 2**i
        parity = 0
        for j in range(parity_pos-1, n, 2*parity_pos):
            if k < n:
                parity ^= hamming_code[k]
        hamming_code[parity_pos] = parity
def generate_hamming_code(data_bits, m):
    r = 0
    n = m
    while n + r + 1 > 2**r:
        r += 1
        n = m + r
    hamming_code = [0] * (n+1)
    j = 0
    k = 0
    for i in range(1, n+1):
        if i == 2**k:
            k += 1
        else:
            hamming_code[i] = data_bits[j]
            j += 1
calculate_parity_bits(hamming_code, n, r)
    return hamming_code, n, r
def detect_and_correct_error(hamming_code, n, r)
    error_pos = 0
    for i in range(r):
```

```
        parity_pos = 2 ** i
        parity = 0
        for j in range(parity_pos, n+1, 2 * parity_pos):
            for k in range(j, j + parity_pos):
                if k <= n:
                    parity ^= hamming_code[k]
        if parity != 0:
            error_pos += parity_pos
    return error_pos

def binary_to_char(binary):
    output = ""
    for i in range(0, len(binary), 8):
        ch = 0
        for j in range(8):
            ch = (binary[i+j] << (7-j))
        output += chr(ch)
    return output

def main():
    input_string = input("Enter the input str
    )
    binary = []
    for ch in input_string:
        binary.extend(char_to_binary(ch))
    data_bits = binary[:]
    hamming_code, n, r = generate_hamming_cod
        (data_bits, len(data_bits))
    print("Generated hamming code:", ' '.join
        (map(str, hamming_code[1:])))
    error_pos = int(input("Enter the position to
        simulate error (0 for no error):"))
```

```
if error_pos > 0 and error_pos <= n:
    hamming_code[error_pos] ^= 1
print("Hamming code with error:", ''.join
    (map(str, hamming_code[1:])))

detected_error_pos = detect_and_correct_error(
    hamming_code, n, int(math.log2(n+1)))
if detected_error_pos == 0:
    print("No error detected.")

else:
    print(f"Error detected at position: {detected_
                                    error_pos}")
    hamming_code[detected_error_pos] ^= 1
    print("Corrected Hamming code:", ''.join(map(
        str, hamming_code[1:])))
    print(f"corrected bit at position {detected_error_
            pos}: {hamming_code[detected_error_
                                    pos]}")

    corrected_data_bits = []
    k = 0
    for i in range(1, n+1):
        if i != 2**k:
            corrected_data_bits.append(hamming
                                    _code[i])
        else:
            k += 1
    corrected_string = binary_to_char(corrected_
                                    data_bits)
    print("corrected string:", corrected_string)
```

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

OUTPUT:

```
Enter the input string : apple
Generated hamming code : 110011000010
111100000111000000111011000110101
Enter the position to stimulate error: 3
Hamming code with error = 111011000010111100000111000000111011000110101
Error detected at pos : 3
corrected bit at position 3 (binary: 11):
corrected hamming code: 110011000010111100000111000000111011000110101
corrected string : apple
```

RESULT:

Thus the program is executed and output is verified.

28/8/24