EX: 6

Date: 20/8/2024

Error Correction at Data Link Layer:

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 the sender to the receiver.

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[1] + data[0] + 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[3]) % 2
    P8 = (data[0] + data[3] + data[2] + data[1]) % 2
    return P1, P2, P4, P8
```

```
def generate_hamming_code(data):
    P1, P2, P4, P8 = 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)
    # calculate the error position
    error_position = P1*1 + P2*2 + P4*4 + P8*8
    return error_position

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

hamming_code = generate_hamming_code(data)
print("The 11-bit Hamming code is:", ''.join(str(bit) for bit in
hamming_code))

corrupted_code = []
print("Enter the 11-bit Hamming code with a possible error
(bit by bit):")
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(f"Data after errorcorrecting all bits: {corrupted_code}")
```

output:

```
Enter 7 bits of data one by one:
Bit 1:1
Bit 2:0
Bit 3:1
Bit 4:1
Bit 5:0
Bit 6:1
Bit 7:1
Data after appending all bits: [1,0,1,1,0,1,0]
The 11-bit Hamming code is: 10101010000
Enter the 11-bit Hamming code with a possible error(bit by
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 errorcorrecting all bits: [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+8]
        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 calculate_parity_values(hamming_code, n):
    n = len(hamming_code)
    for i in range(n):
        Parity_pos = 2**i
        Parity_val = 0
        for j in range(1, n+1):
            if j & Parity_pos and j != Parity_pos:
                Parity_val ^= hamming_code[j-1]
        hamming_code[Parity_pos-1] = Parity_val
    return hamming_code

def detect_and_correct_error(hamming_code, n):
    n = len(hamming_code)
    error_position = 0

    for i in range(n):
        Parity_pos = 2**i
        Parity_val = 0
        for j in range(1, n+1):
            if j & Parity_pos:
                Parity_val ^= hamming_code[j-1]

        if Parity_val != 0:
            error_position += Parity_pos

    if error_position:
        print(f"Error detected at position: {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(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 representation of '{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("\nIntroducing a single-bit error for demonstration:")
    error_bit = int(input(f"Enter the bit position (1-{len(hamming_code)}) to introduce an error: ")) hamming_code[error_bit-1] ^= 1
    print(f"Hamming code with error: {hamming_code}")
    hamming_code = detect_and_correct_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 Hamming code: {corrected_string}")
if __name__ == "__main__":
    main()
```

Output:

```
ENTER A STRING hi
Binary representation of 'hi': 0110100001101001
Hamming code with parity bits: [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 for demonstration...
ERROR!
Enter the bit position (1-21) to introduce an error: 5
Hamming code with error: [0,0,0,1,0,1,0,1,1,0,0,0,
    1,1,0,0,1,0,0,1]
Error detected at position: 5
Corrected Hamming code: [0,0,0,1,1,1,0,1,1,0,0,0,
    1,1,0,0,1,0,0,1]
Final output after correcting Hamming code: 'hi'
```

20/8/24

Result:

Thus this was successfully executed & output is verified.