

PRACTICAL - 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.

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. It is a technique developed by R.W. Hamming for error correction.

Create sender program with below features.

1. Input to sender file should be a text of any length. Program should convert the text to binary.
2. Apply hamming code concept on the binary data and add redundant bits to it.
3. Save this output in a file called channel.

Create a receiver program with below 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.

Student observation:-

Sender Program - Hamming Code

```
def calc_redundant_bits(data_bits):
    r = 0
    while (2 ** r) < (len(data_bits) + r + 1):
        r += 1
    return r

def position_redundant_bits(data_bits, r):
    j = 0
    k = 1
    m = len(data_bits)
    res = ""
    for i in range(1, m + r + 1):
        if i == 2 ** j:
            res = res + '0'
            j += 1
        else:
            res = res + data_bits[-1 * k]
            k += 1
    return res[::-1]
```

```
def calc_parity_bits(arr, r):
    n = len(arr)
    arr = list(arr)
    for i in range(r):
        val = 0
        for j in range(1, n + 1):
            if j & (2 ** i) == (2 ** i):
                val = val ^ int(arr[-1 * j])
        arr[-1 * (2 ** i)] = str(val)
    return ''.join(arr)

# Input from user
text = input("Enter text to send: ")

# Convert text to binary
binary_data = ''.join(format(ord(x), '08b') for x in text)
print("Binary Data:", binary_data)

# Apply Hamming code
r = calc_redundant_bits(binary_data)
arr = position_redundant_bits(binary_data, r)
arr = calc_parity_bits(arr, r)

# Save to channel file
with open("channel.txt", "w") as f:
    f.write(arr)

print("Data written to 'channel.txt':", arr)
```

Receiver Program - Hamming Code

```
def detect_error(arr, r):
    n = len(arr)
    res = 0
    for i in range(r):
        val = 0
        for j in range(1, n + 1):
            if j & (2 ** i) == (2 ** i):
                val = val ^ int(arr[-1 * j])
        res = res + val * (10 ** i)
    return int(str(res), 2)

# Read received data
with open("channel.txt", "r") as f:
    arr = f.read().strip()

print("Received data from channel:", arr)
n = len(arr)
```

```
r = 0
while (2 ** r) < n:
    r += 1

error_pos = detect_error(arr, r)
arr = list(arr)

if error_pos == 0:
    print("No error detected.")
else:
    print("Error detected at bit position:", error_pos)
    # Correct the bit
    arr[-error_pos] = '1' if arr[-error_pos] == '0' else '0'
    print("Corrected Data:", "".join(arr))

# Remove parity bits to get original binary
j = 0
res = ""
for i in range(1, len(arr) + 1):
    if i != 2 ** j:
        res = res + arr[-i]
    else:
        j += 1

res = res[::-1]

# Convert binary to text
decoded_text = ""
for i in range(0, len(res), 8):
    byte = res[i:i+8]
    decoded_text += chr(int(byte, 2))

print("Decoded text:", decoded_text)
```

Input:- Enter text to send: A

Output:

Received data from channel: 011100011010
Error detected at bit position: 7
Corrected Data: 011100001010
Decoded text: A

Result:

- The program successfully detected and corrected the single-bit error.
- The received message matched the original transmitted data.