San Francisco Bay University

CE305 - Computer Organization
2023 Fall Homework #2

Kritika Regmi
ID:19702

1. Cyclic Redundancy Check (CRC) is one of the popular coding and decoding techniques in the data transmitted over the network for error detection and correction. Given $x^5 + x^2 + 1$ as a CRC generation polynomial from International Telegraph and Telephone Consultative Committee (CCITT), write the encoding and decoding *def* functions in Python for the only 4-bits original binary data. The examples and testcases of the encoding and decoding processes are shown as follows for your programming. After that, discuss how many bits errors CRC can detect.

ANS: Since we are using a polynomial of degree 5, this can detect 5 bits errors. If we need a CRC that can detect a specific number of bit errors, we will need to select a CRC with the appropriate polynomial length.

CODE:

```
#Week2_CE305_Question1_19702_Kritika_Regmi
#Cyclic_Redundancy_Check

def encoding(msg, poly):
    msg_temp=msg
    msg = msg + '0' * (len(poly) - 1)
    remainder = list(msg)

    for i in range(len(msg) - len(poly) + 1):
        if remainder[i] == '1':
            for j in range(len(poly)):
                remainder[i + j] = str(int(remainder[i + j]) ^ int(poly[j]))

    encoded_msg = ''.join(remainder)
    return msg_temp + encoded_msg[-(len(poly) - 1):]
```

```
def decoding(rcv, poly):
    received_msg, received_remainder = rcv.split()
    received_msg += received_remainder
    received_msg = list(received_msg)

    for i in range(len(received_msg) - len(poly) + 1):
        if received_msg[i] == '1':
            for j in range(len(poly)):
                received_msg[i + j] = str(int(received_msg[i + j]) ^ int(poly[j]))

    remainder = ''.join(received_msg)[-len(poly) + 1:]

    if all(bit == '0' for bit in remainder):
        return 'No error'
    else:
        return 'Error'
```

TEST CASES AND RESULT:

```
# Test cases
org_sig1 = '1010'
poly = '100101'
print(encoding(org_sig1, poly))
received_sig1 = '1010 00111'
print(decoding(received_sig1, poly))

org_sig2 = '1100'
poly = '100101'
print(encoding(org_sig2, poly))
received_sig2 = '1010 01111'
print(decoding(received_sig2, poly))
```

```
101000111
No error
110011001
Error
```

2. Hamming code is one important error correcting code in computer science and telecommunication as well. Standard Hamming code can only detect and correct a **single bit** error.

ANS:

CODE:

```
#Week2_CE305_Question2_19702_Kritika_Regmi
#Hamming_code

def HamEncoding(msg):
    n = len(msg)
    k = 0
    while 2**k < n + k + 1:
        k += 1

    encoded_msg = ['0'] * (n + k)

    j = 0
    for i in range(1, n + k + 1):
        if i == 2**j:
            j += 1
        else:
            encoded_msg[i - 1] = msg[i - j - 1]

    for i in range(k):
        parity_bit_index = 2**i - 1
        count = 0
        for j in range(parity_bit_index, n + k, 2**(i + 1)):
            for l in range(j, min(j + 2**i, n + k)):
                count ^= int(encoded_msg[l])
        encoded_msg[parity_bit_index] = str(count)

    return ''.join(encoded_msg)
```

```python
def HamDecoding(rcv, k):
    n = len(rcv)
    decoded_msg = list(rcv)

    errors = []
    for i in range(k):
        parity_bit_index = 2**i - 1
        count = 0
        for j in range(parity_bit_index, n, 2**(i + 1)):
            for l in range(j, min(j + 2**i, n)):
                count ^= int(decoded_msg[l])
        if count != 0:
            errors.append(parity_bit_index + 1)

    errors.reverse()
    error_positions_str = ', '.join(map(str, errors))

    if not errors:
        return 'No error'
    else:
        for pos in errors:
            decoded_msg[pos - 1] = str(1 - int(decoded_msg[pos - 1]))
        corrected_msg = ''.join(decoded_msg)
        return f'Error at Position(s) {error_positions_str}, and correct data: {corrected_msg}'
```

TEST CASES AND RESULT:

```python
# Test cases
org_sig1 = '1101'
encoded_sig1 = HamEncoding(org_sig1)
print(encoded_sig1)

received_sig1 = '1010101'
k = 3
print(HamDecoding(received_sig1, k))

org_sig2 = '1001011'
encoded_sig2 = HamEncoding(org_sig2)
print(encoded_sig2)

received_sig2 = '1010001'
k = 3
print(HamDecoding(received_sig2, k))
```

```
1010101
No error
10110010011
Error at Position(s) 4, 1, and correct data: 0011001
```