[ 2CEIT503 COMPUTER NETWORKS]

# Practical: 5

**AIM- Write a program to implement various Error Detection Mechanisms.**
**a. find minimum hamming distance**
**b. Checksum**
**c. CRC.**

Submitted By:
21012021003_AMIT GOSWAMI


Department of
Information Technology

## a) find minimum hamming distance

```
def hamming_distance(word1, word2):
    if len(word1) != len(word2):
        raise ValueError("Input words must have the same length")
    distance = sum(bit1 != bit2 for bit1, bit2 in zip(word1, word2))
    return distance

def minimum_hamming_distance():
    num_bits = int(input("Enter the number of bits: "))
    num_codewords = int(input("Enter the number of codewords: "))
    codewords = []
    for i in range(num_codewords):
        codeword = input(f"Enter codeword {i + 1} : ")
        if len(codeword) != num_bits:
            print(f"Error: Codeword {i + 1} must have {num_bits} bits.")
            return
        codewords.append(codeword)
    min_distance = float('inf')
    for i in range(num_codewords):
        for j in range(i + 1, num_codewords):
            distance = hamming_distance(codewords[i], codewords[j])
            min_distance = min(min_distance, distance)
    print(f"Minimum Hamming distance between codeword is : {min_distance}")
minimum_hamming_distance()
```

```
Enter the number of bits: 4
Enter the number of codewords: 2
Enter codeword 1 : 1100
Enter codeword 2 : 1010
Minimum Hamming distance between codeword is : 2
```

## b) Checksum

```
def calculate_checksum(data_segments):
    # Calculate the sum of all data segments using 1's complement arithmetic
    checksum_sum = sum(int(segment, 2) for segment in data_segments)  # Assuming
binary segments

    # Perform 1's complement on the sum
    checksum = format(checksum_sum, 'b')
    while len(checksum) > len(data_segments[0]):
        checksum = checksum[1:]  # Remove carry if present
    wrapsum = checksum
    # Take 1's complement of the checksum
```

```
    checksum = ''.join(['1' if bit == '0' else '0' for bit in checksum])

    print(f"Wrapsum: {wrapsum}")
    print(f"Checksum: {checksum}")

    return checksum
    return wrapsum

def receive_and_validate(received_data_segments, received_checksum):
    # Calculate the sum of all received segments using 1's complement arithmetic
    received_sum = sum(int(segment, 2) for segment in received_data_segments)  #
Assuming binary segments

    # Add the received checksum to the sum
    received_sum += int(received_checksum, 2)

    # Perform 1's complement on the sum
    received_sum = format(received_sum, 'b')
    while len(received_sum) > len(data_segments[0]):
        received_sum = received_sum[1:]  # Remove carry if present

    # Take 1's complement of the received sum
    received_sum = ''.join(['1' if bit == '0' else '0' for bit in received_sum])
     # If the result is zero, the received data is accepted; otherwise, it's discarded
    acceptable_data = ''.join(['0' for i in range(m)])

    if received_sum == acceptable_data:
        return f"complemented total sum: {received_sum}\nData Accepted"
        # return f"acceptable data: {acceptable_data}"
        # return "Data Accepted"
    else:
        return f"complemented total sum: {received_sum}\nData Discarded"
        # return "Data Discarded"

k = int(input("Enter the number of segments(k): "))
m = int(input("Enter the number of bits per segment (m): "))
data_segments = []
for i in range(k):
    segment = input(f"Enter segment {i + 1} (a binary number with {m} bits): ")
    if len(segment) != m:
        print(f"Error: Segment {i + 1} must have exactly {m} bits.")
        exit(1)
    data_segments.append(segment)

checksum = calculate_checksum(data_segments)

received_checksum = checksum
received_data_segments = []
print("=== for receiver side ===")
for i in range(k):
```

```
    segment = input(f"Enter segment {i + 1}(a binary number with {m} bits): ")
    if len(segment) != m:
        print(f"Error: Segment {i + 1} must have exactly {m} bits.")
        exit(1)
    received_data_segments.append(segment)

result = receive_and_validate(received_data_segments, received_checksum)
print(result)
```

```
Enter the number of segments(k): 2
Enter the number of bits per segment (m): 4
Enter segment 1 (a binary number with 4 bits): 1100
Enter segment 2 (a binary number with 4 bits): 1010
Wrapsum: 0110
Checksum: 1001
=== for receiver side ===
Enter segment 1(a binary number with 4 bits): 1100
Enter segment 2(a binary number with 4 bits): 1010
complemented total sum: 0000
Data Accepted
```

## c) CRC

```
def xor_operation(n1, n2):
    return "".join(['1' if a != b else '0' for a, b in zip(n1, n2)])

def division(data, divisor):
    lenDivis = len(divisor)
    invData = "0" * lenDivis
    lenCode = len(data)
    i = lenDivis
    codePart = data[:lenDivis]
    while i <= lenCode:
        if codePart[0] == "1":
            temp = xor_operation(codePart, divisor)
        else:
            temp = xor_operation(codePart, invData)
        if i != lenCode:
            codePart = temp[1:] + data[i]
        else:
            codePart = temp
        i += 1
    return codePart[1:]
```

```
def sender_and_receiver_CRC(data, divisor):
    lenDivis = len(divisor)
    codeWord = data + "0" * (lenDivis - 1)
    syndrome = division(codeWord, divisor)
    encoded_message = xor_operation(codeWord, syndrome)

    return encoded_message

data = input("Enter the data (binary): ")
divisor = input("Enter the divisor polynomial (binary): ")

encoded_message = sender_and_receiver_CRC(data, divisor)

print("Encoded Message:", encoded_message)

def receiverCRC(data, divisor):
    return division(data, divisor)
received_message = input("Enter the received message (binary): ")

if receiverCRC(received_message, divisor) == "0" * (len(divisor) - 1):
    print("Message is error-free.")
else:
    print("Message contains errors.")
```

```
Enter the data (binary): 101110
Enter the divisor polynomial (binary): 1011
Encoded Message: 011
Enter the received message (binary): 101101
Message contains errors.
```