[ 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

1. **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):

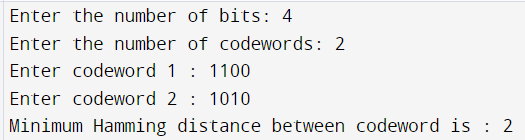
for j in range(i + 1, num\_codewords):

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()



1. **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:

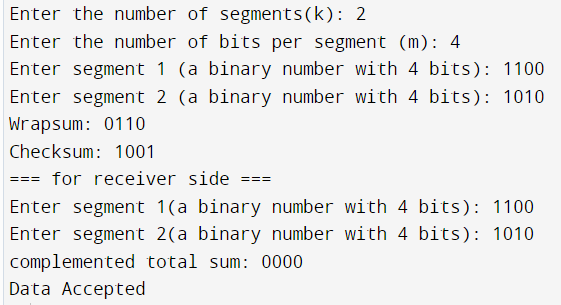
print(f"Error: Segment {i + 1} must have exactly {m} bits.")

exit(1)

received\_data\_segments.append(segment)

result = receive\_and\_validate(received\_data\_segments, received\_checksum)

print(result)



1. **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.")

