[2CEIT503 COMPUTER NETWORKS]

Practical: 5

AIM- Write a program to implement various Error Detection Mechanisms.

a. find minimum hamming distance

b. Checksum

c. CRC

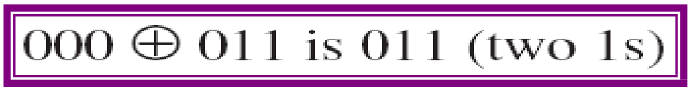
Submitted By:
20012021007_ Gupta Samarth Rajeshkumar
5CEIT-D_AB15

Ganpat University
|| विद्यया समाजोत्कर्षः ||
U.V. Patel College of Engineering

Department of Computer Engineering/Information Technology

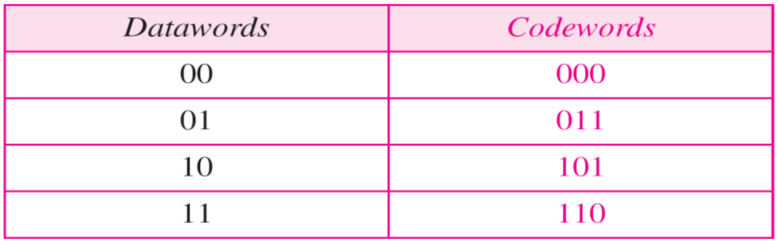
Hamming Distance

- The Hamming distance between two words is the number of differences between corresponding bits.
- Hamming distance between two words x and y as d(x,y)
- The Hamming distance d(000, 011) is 2 because

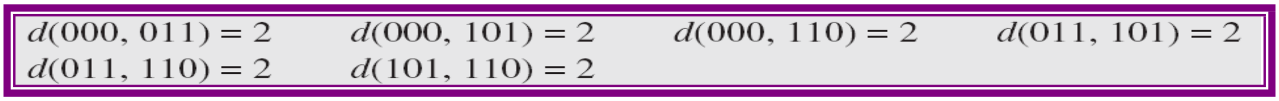


$000 \oplus 011$ is 011 (two 1s)

- The **minimum Hamming distance** is the smallest Hamming distance between all possible pairs in a set of words.



Datawords	*Codewords*
00	000
01	011
10	101
11	110



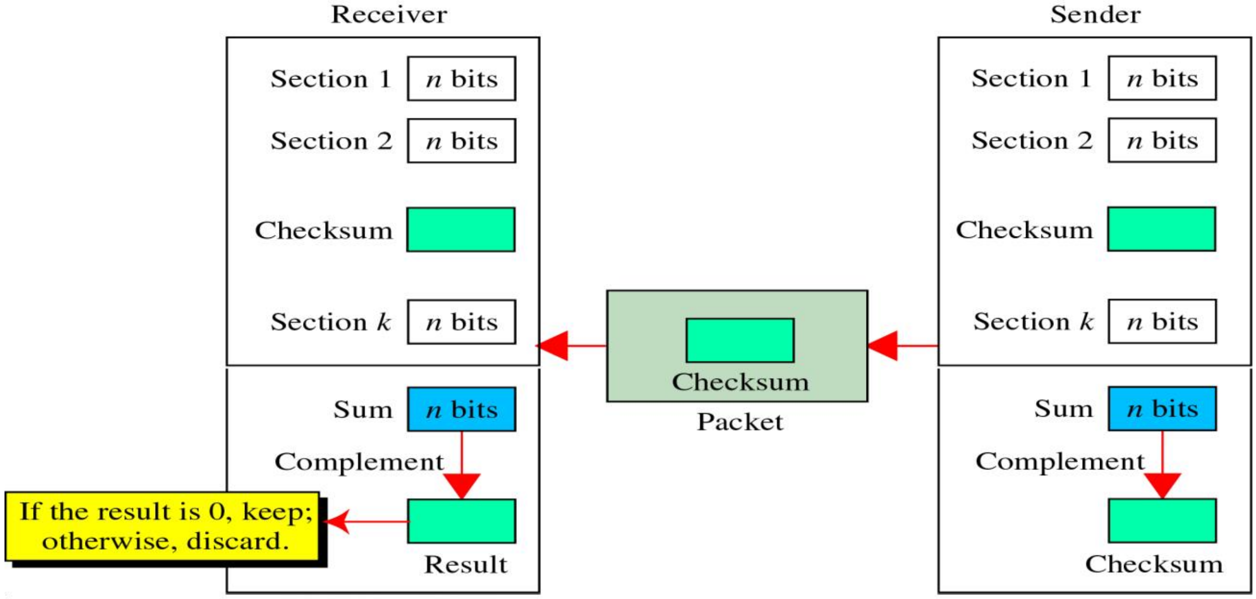
$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$

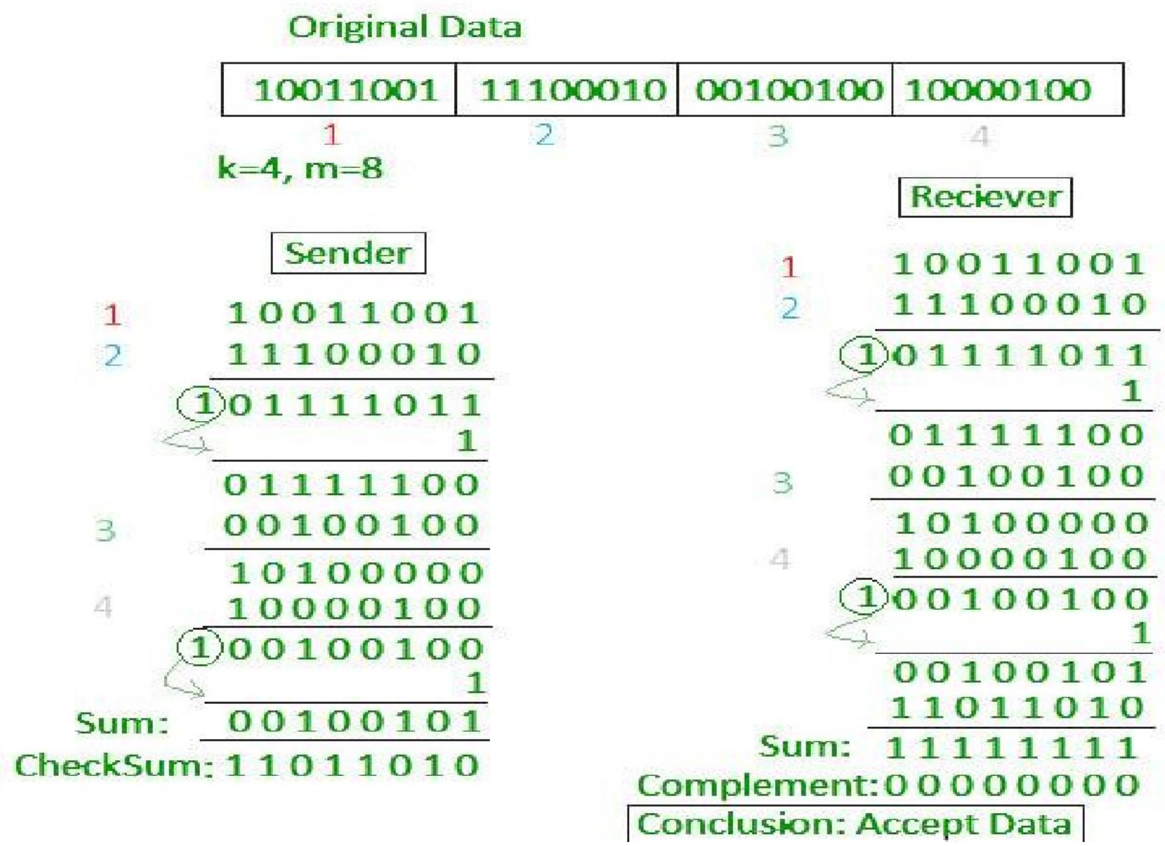
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Checksum

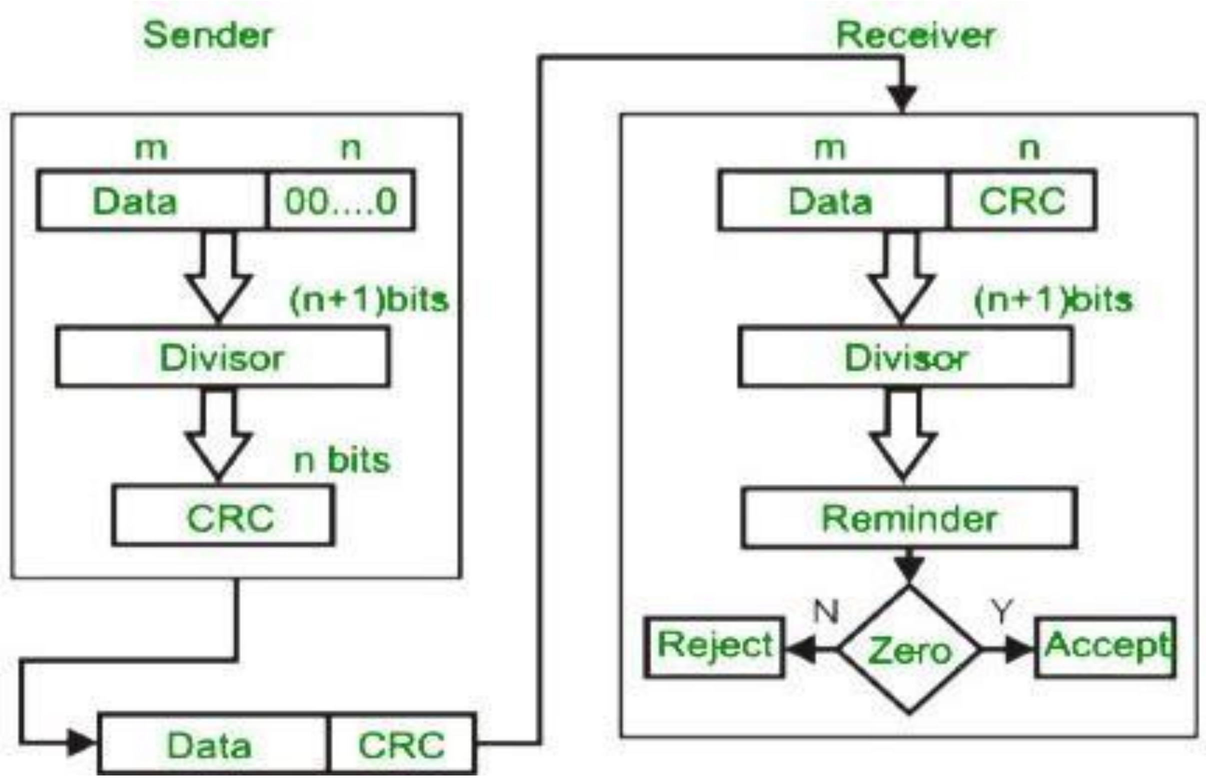
- In checksum error detection scheme, the data is divided into *k* segments each of m bits.
- In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum.
- The checksum segment is sent along with the data segments.
- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded.

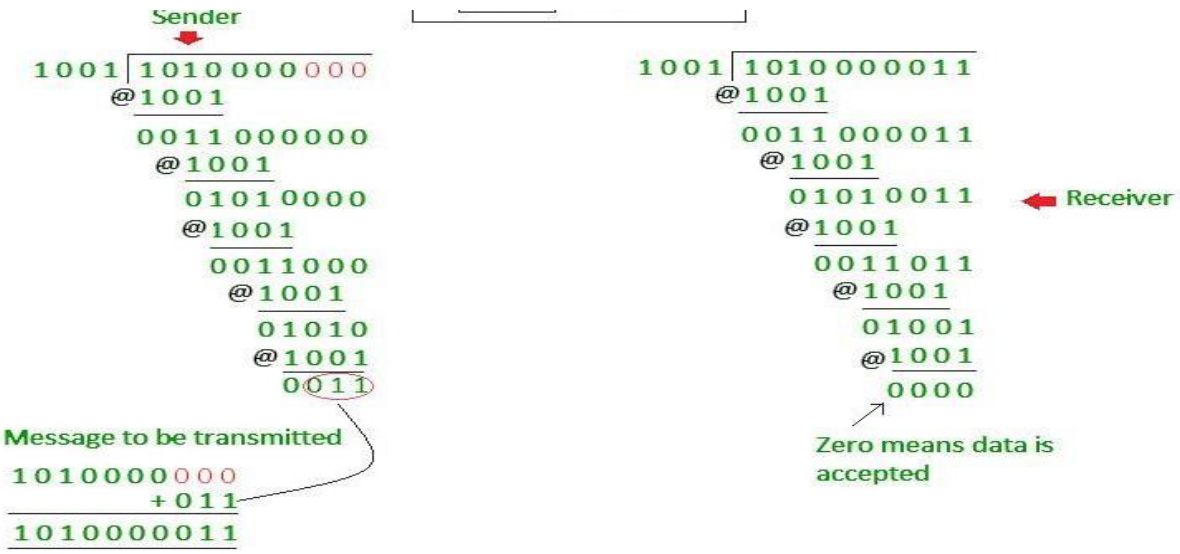

Receiver
Section 1
n bits
Section 2
n bits
Checksum
Section k
n bits
Sum
n bits
Complement
If the result is 0, keep; otherwise, discard.
Result
Checksum
Packet
Sender
Section 1
n bits
Section 2
n bits
Checksum
Section k
n bits
Sum
n bits
Complement
Checksum


Original Data
10011001
11100010
00100100
10000100
1
2
3
4
k=4, m=8
Sender
1 10011001
2 11100010
1 01111011
1
01111100
3 00100100
10100000
4 10000100
1 00100100
1
Sum: 00100101
CheckSum: 11011010
Reciever
1 10011001
2 11100010
1 01111011
1
01111100
3 00100100
10100000
4 10000100
1 00100100
1
00100101
11011010
Sum: 11111111
Complement: 00000000
Conclusion: Accept Data

CRC

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



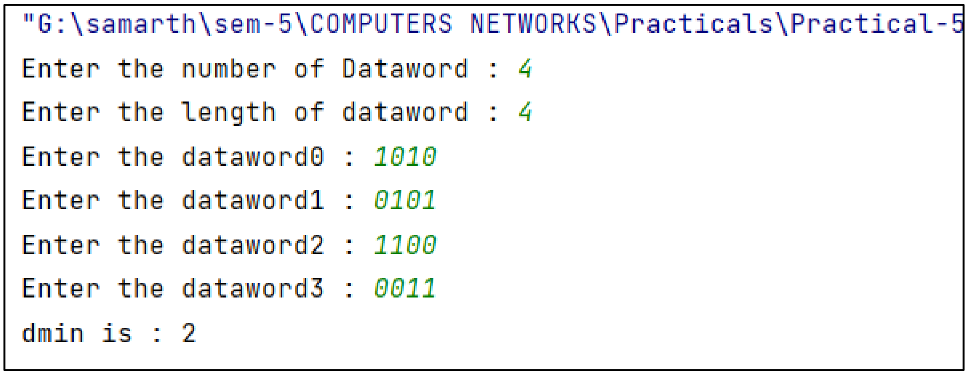


1. Hamming Distance:

CODE:

```
n = int(input("Enter the number of Dataword : "))
m = int(input("Enter the length of dataword : "))
l1 = []
a = ""
i = 0
while (i < n):
    a = list(input(f"Enter the dataword{i} : "))
    if (len(a) == m):
        l1.append(a)
        i += 1
    else:
        print(f"Enter dataword{i} again : ")

# print(l1)

def xor(p,q,m,l1):
    result=[]
    for i in range(m):
        if(l1[p][i] == l1[q][i]):
            result.append(0)
        else:
            result.append(1)
    return result

def hamming_dis():
    dmin=0
    res=[]
    for i in range(0, n):
        for j in range(i+1,n):
            c=xor(i,j,m,l1)
            res.append(c)

    # print(res)
    count = []
    counter = 0
    for i in res:
        for j in range(m):
            if(i[j] == 1):
                counter +=1
        count.append(counter)
        counter = 0
        dmin = min(count)

    print(f"dmin is : {dmin}")

hamming_dis()
```

OUTPUT:



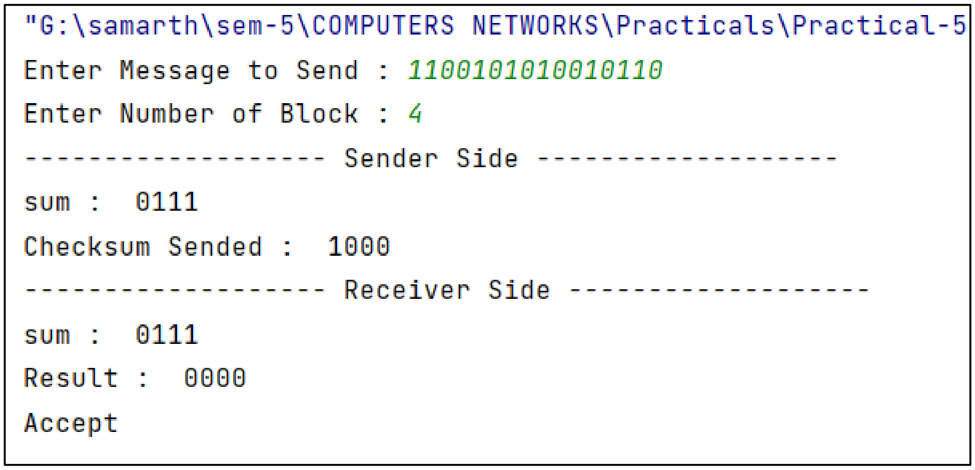
```
"G:\samarth\sem-5\COMPUTERS NETWORKS\Practicals\Practical-5
Enter the number of Dataword : 4
Enter the length of dataword : 4
Enter the dataword0 : 1010
Enter the dataword1 : 0101
Enter the dataword2 : 1100
Enter the dataword3 : 0011
dmin is : 2
```

2. Checksum:

CODE:

```
data = input("Enter Message to Send : ") # 1100101010010110
k = int(input("Enter Number of Block : ")) # 4

sum =""
def findsum(data , k):
    d1 = data[0:k]
    d2 = data[k:2*k]
    d3 = data[2*k:3*k]
    d4 = data[3*k:4*k]

    temp = bin(int(d1,2) + int(d2,2) +int(d3,2) + int(d4,2))[2:]
    temp1 = temp[0:len(temp)-k]
    temp2 = temp[len(temp1):]
    sum = bin(int(temp1,2) + int(temp2,2))[2:]
    num = len(sum)
    final_sum = ""
    while(num<k):
        final_sum = "0" + sum
        num+=1
    print("sum : ", final_sum)
    return final_sum

def complement(sum):
    checksum=""
    for i in sum:
        if i == '1':
            checksum += "0"
        else :
            checksum += "1"
    return checksum

print("------------------- Sender Side -------------------")

sum = findsum(data,k)
scheckcum = complement(sum)
print("Checksum Sended : " ,scheckcum)

print("------------------- Receiver Side -------------------")

rsum = findsum(data,k)

rchecksum = bin(int(scheckcum,2)+int(rsum,2))[2:]

final_checksum = complement(rchecksum)
print("Result : ",final_checksum)

if(int(final_checksum,2)==0):
    print("Accept")
else:
    print("Resend")
```

OUTPUT:



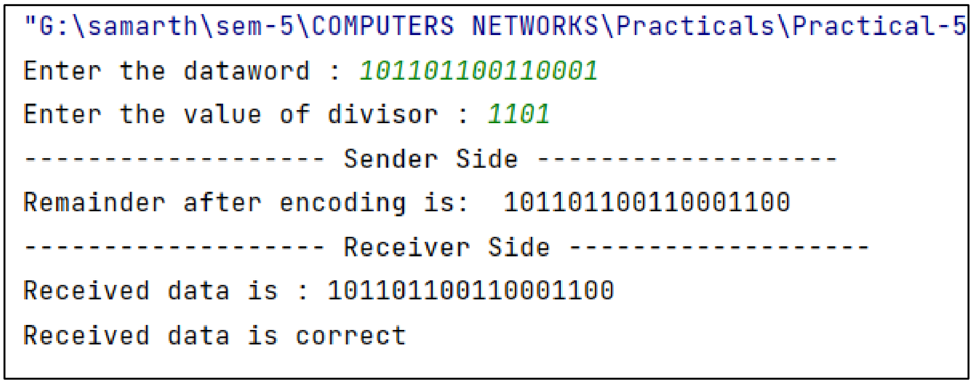
```
"G:\samarth\sem-5\COMPUTERS NETWORKS\Practicals\Practical-5
Enter Message to Send : 1100101010010110
Enter Number of Block : 4
------------------- Sender Side -------------------
sum :  0111
Checksum Sended :  1000
------------------- Receiver Side -------------------
sum :  0111
Result :  0000
Accept
```

3. Cyclic Redundancy Check:

CODE:

```
def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)

def binary_division(divident, divisor):
    pick = len(divisor)
    tmp = divident[0: pick]
    while pick < len(divident):
        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + divident[pick]
        else:
            tmp = xor('0'*pick, tmp)  + divident[pick]
        pick += 1

    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)
    checkword = tmp
    return checkword

def encodeData(data, key):
    l_key = len(key)
    send_data = data + '0'*(l_key-1)
    rem = binary_division(send_data, key)
    codeword = data + rem
    return codeword
def decodeData(receivedData, key):
    l_key = len(key)
    receive_data = receivedData + '0'*(l_key-1)
    remainder = binary_division(receive_data, key)
    return remainder

data = input("Enter the dataword : ")
divisor = input("Enter the value of divisor : ")
answer = encodeData(data, divisor)
print("------------------- Sender Side -------------------")
print("Remainder after encoding is: ", answer)
print("------------------- Receiver Side -------------------")
print(f"Received data is : {answer}")
receivedData=answer
remainder = decodeData(receivedData, divisor)
if remainder == '0'*(len(divisor) - 1):
    print("Received data is correct")
else:
    print("Received data is incorrect")
```

OUTPUT:



```
"G:\samarth\sem-5\COMPUTERS NETWORKS\Practicals\Practical-5
Enter the dataword : 101101100110001
Enter the value of divisor : 1101
------------------- Sender Side -------------------
Remainder after encoding is:  101101100110001100
------------------- Receiver Side -------------------
Received data is : 101101100110001100
Received data is correct
```