**[ 2CEIT503 COMPUTER NETWORKS]**

Practical: 5



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

1. **find minimum hamming distance**
2. **Checksum**
3. **CRC**

Submitted By:

20012021007\_ Gupta Samarth Rajeshkumar

5CEIT-D\_AB15



# Department of Computer Engineering/Information Technology

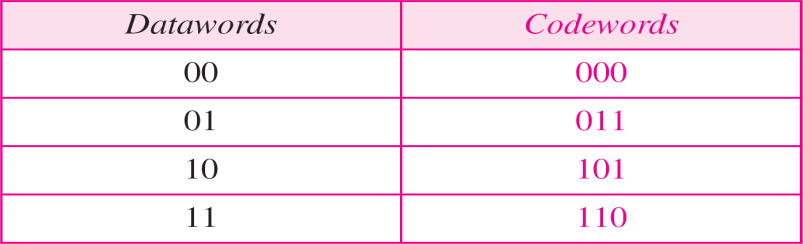
**Hamming Distance**



* 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



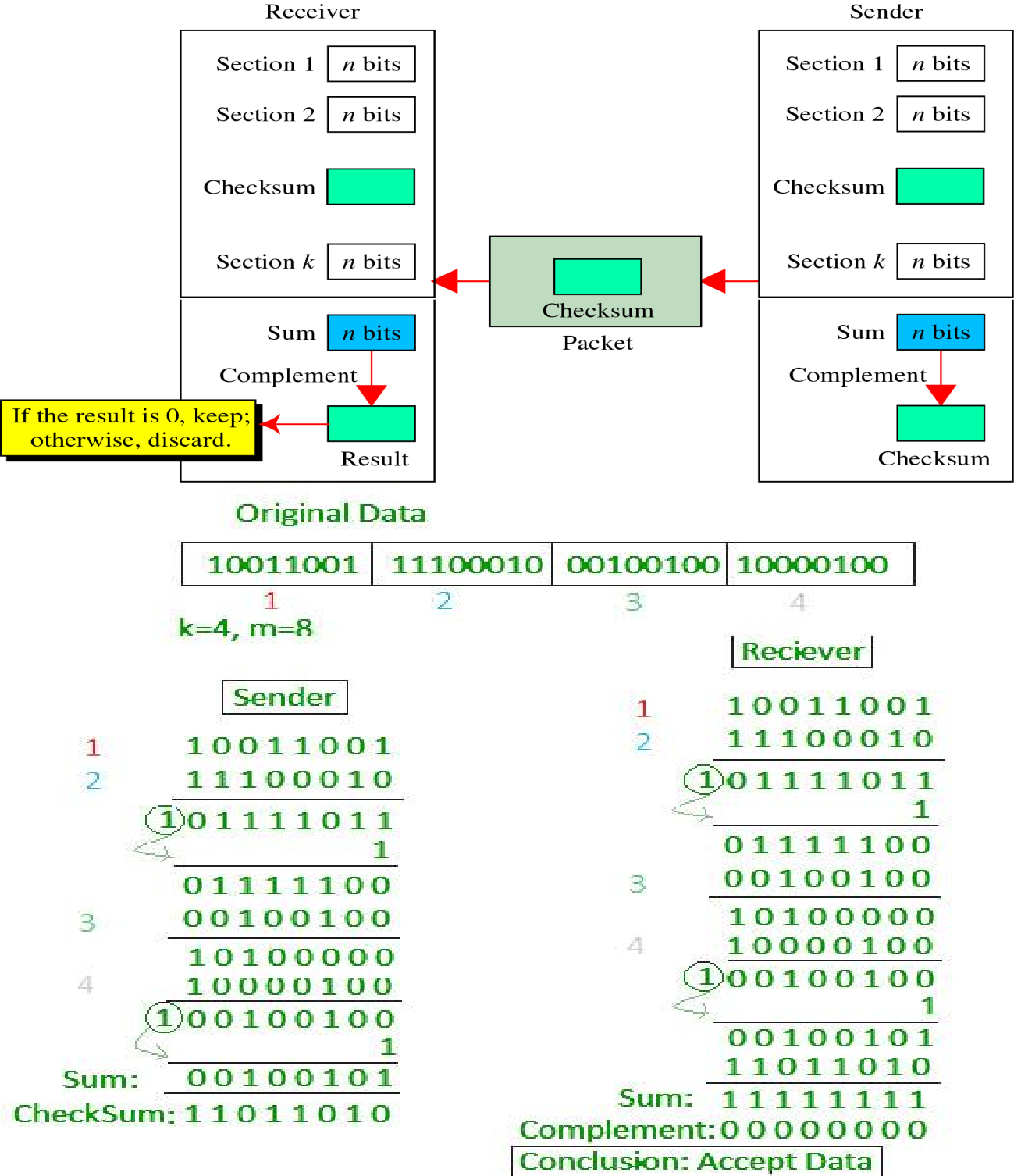
* The **minimum Hamming distance** is the smallest Hamming distance between all possible pairs in a set of words.



The dmin in this case is 2.

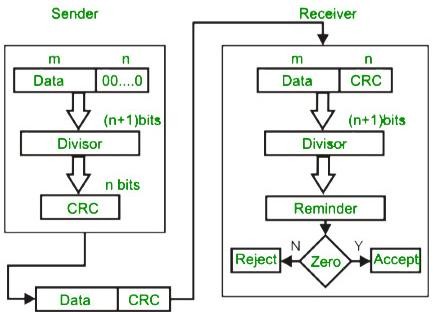
# Checksum

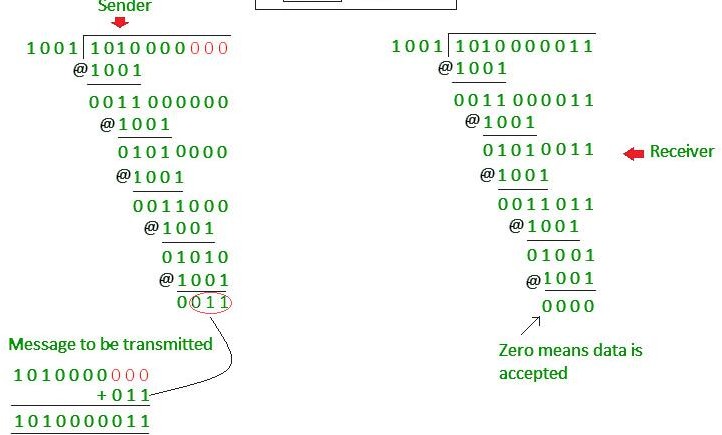
* In checksum error detection scheme, the data is divided into *k* segments each of m bits.
* In the sender’s end the segments are added using 1’s complement arithmetic to get the sum. The sum is complemented to get the checksum.
* The checksum segment is sent along with the data segments.
* At the receiver’s 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.



# 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.



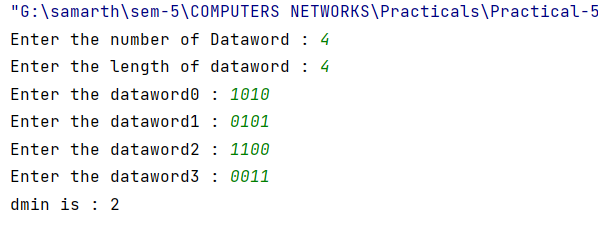


1. Hamming Distance:

**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:**

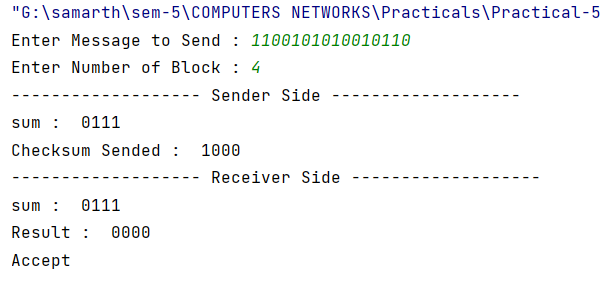


1. Checksum:

**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:**

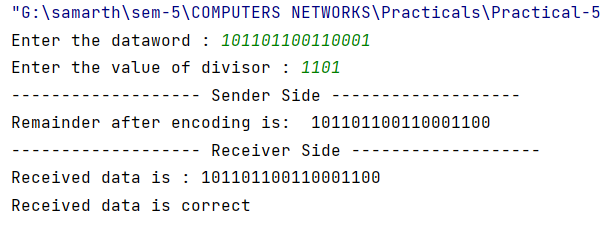


1. Cyclic Redundancy Check:

**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:**

****