## 2. i) Write a Program to implement the data link layer farming methods such as --> character stuffing.

**PROGRAM:**

```
head = input ("Enter character that represents the starting delimiter: ")
tail = input (" Enter character that represents the ending delimiter: ")
st = input("Enter the characters to be stuffed: ")
res=head
for i in st:
    if i==head or i==tail:
        res = res + i + i
    else:
        res = res + i
res = res+tail
print("Frame after character stuffing: ", res)
```

**OUTPUT:**

```
Enter character that represents the starting delimiter: d

Enter character that represents the ending delimiter: g

Enter the characters to be stuffed: goodday

Frame after character stuffing: dggooddddayg
```

## 2. ii) Write a Program to implement the data link layer farming methods such as --> bit stuffing.

**PROGRAM:**

```
st = input ("Enter the frame: ")
count = 0
res = ""
for i in st:
    if i == '1' and count < 5:
        res += '1'
        count += 1
    elif i == ' ':
        pass
    else:
        res += i
        count = 0
    if count == 5:
        res += '0'
        c = 0
print ("Frame after bit stuffing: ", res)
```

**OUTPUT:**

```
Enter the frame: 01111110

Frame after bit stuffing: 011111010
```

3. Write a Program to implement data link layer farming method checksum.

**PROGRAM:**

```
s1 = input("Enter the string of 0's and 1's as subunit1: ")
s2 = input("Enter the string of 0's and 1's as subunit2: ")
s1 = s1[::-1]
s2 = s2[::-1]

res = ""
c = '0'

for i,j in zip(s1, s2):
    if i == '0' and j == '0' and c == '0':
        res += '0'
        c = '0'
    elif i == '0' and j == '0' and c == '1':
        res += '1'
        c = '0'
    elif i == '0' and j == '1' and c == '0':
        res += '1'
        c = '0'
    elif i == '0' and j == '1' and c == '1':
        res += '0'
        c = '1'
    elif i == '1' and j == '0' and c == '0':
        res += '1'
        c = '0'
    elif i == '1' and j == '0' and c == '1':
        res += '0'
        c = '1'
    elif i == '1' and j == '1' and c == '0':
        res += '0'
        c = '1'
    elif i == '1' and j == '1' and c == '1':
        res += '1'
        c = '1'


if c == '1':
    ans = ""
    for i in res:
        if i == '1' and c == '1':
            ans += '0'
            c = '1'
        elif i == '0' and c == '0':
            ans += '0'
            c = '0'
        else :
            ans += '1'
            c = '0'
    res = ans
```

```
final = ""
for i in res:
    if i == '1':
        final += '0'
    else:
        final += '1'
print("Checksum of two subunits: ", final[::-1].strip())
```

**OUTPUT:**

```
Enter the string of 0's and 1's as subunit1: 10101001

Enter the string of 0's and 1's as subunit2: 00111001

Checksum of two subunits:  00011101
```

4. Write a program for Hamming Code generation for error detection and correction.

**PROGRAM:**

```
li = list(map(int,input("Enter 7 bits data of 0's  and 1's  separated by
spaces: ").split()))
rec = list(map(int,input("Enter the received 11 data bits of 0's and 1's
separated by spaces: ").split()))
# reverse the list
li = li[::-1]
# parity bits of 0 are added at the place of 2 pow's i.e. at positions of
1,2,4,6 remaining places data bits are added
li = [0,0] + li[0:1] + [0] + li[1:4] + [0] + li[4:]
#now find the even parity bit position
li[0] = (li[2] + li[4] + li[6] + li[8] + li[10]) % 2
li[1] = (li[2] + li[5] + li[6] + li[9] + li[10]) % 2
li[3] = (li[4] + li[5] + li[6]) % 2
li[7] = (li[8] + li[9] + li[10]) % 2
# reverse the list
li = li[::-1]
# reverse the receiver side data and check the parity bits position values
rec = rec[::-1]
r1 = (rec[0] + rec[2] + rec[4] + rec[6] + rec[8] + rec[10]) % 2
r2 = (rec[1] + rec[2] + rec[5] + rec[6] + rec[9] + rec[10]) % 2
r3 = (rec[3] + rec[4] + rec[5] + rec[6]) % 2
r4 = (rec[7] + rec[8] + rec[9] + rec[10]) % 2

bit = str(r4) + str(r3) + str(r2) + str(r1)
bit = int(bit,2)
if bit :
    print("received data is having error at position: ", bit)
else:
    print("received data doesn't have any error")
```

**OUTPUT:**

```
Enter 7 bits data of 0’s and 1’s separated by spaces: 1 0 1 0 1 0 1

Enter the received 11 data bits of 0's and 1's separated by spaces1 0 1 0 0 1 0 1 1
0 1

Received data is having error at position:  2


Enter 7 bits data of 0's  and 1's  separated by spaces: 1 0 1 0 1 0 1

Enter the received 11 data bits of 0's and 1's separated by spaces: 1 0 1 0 0 1 0 1
1 1 1

received data doesn't have any error
```

## 5. Write a Program to implement on a data set of characters the three CRC polynomials CRC 12, CRC 16 and CRC CCITT.

**PROGRAM:**

```
def xor(x, y):
    ans = ""
    for i in range(1, len(y)):
        if x[i] == y[i]:
            ans += '0'
        else:
            ans += '1'
    return ans


def divide(dividend, divisor):
    a = len(divisor)
    temp = dividend[0:a]
    while a < len(dividend):
        if temp[0] == '1':
            temp = xor(divisor, temp) + dividend[a]
        else:
            temp = xor('0' * a, temp) + dividend[a]
        a += 1
    if temp[0] == '1':
        temp = xor(divisor, temp)
    else:
        temp = xor('0' * a, temp)
    return temp


keys = ['1100000001111', '11000000000000101', '10001000000100001']

print("Choose the CRC")
print("1. CRC - 12")
print("2. CRC - 16")
print("3. CRC - CCITT ")
n = int(input())
send = input("Enter the string of code word of binary data bits of 0's and 1's to be
sent from the sender: ")
rec = input(" Enter the string of code word of binary data received at the receiver
side: ")
key = keys[n - 1]

# encoding sender side
length = len(key)
send1 = send + '0' * (length - 1)
rem = divide(send1, key)

# decoding receiver side
ans = divide(rec, key)
if (ans == '0' * (len(key) - 1)):
    print("no error")
else:
    print("frame error")
```

**OUTPUT:**

```
Choose the CRC

1. CRC - 12

2. CRC - 16

3. CRC - CCITT

2

Enter the string of code word of binary data bits of 0's and 1's to be sent from the
sender: 101110111010101

 Enter the string of code word of binary data received at the receiver side:
1011101110101010100110011111011

no error


Choose the CRC

1. CRC - 12

2. CRC - 16

3. CRC - CCITT

1

Enter the string of code word of binary data bits of 0's and 1's to be sent from the
sender: 1010101

 Enter the string of code word of binary data received at the receiver side:
1010101001000000010

no error
```

6. Write a Program to implement Sliding window protocol for Go back N.

**PROGRAM:**

**SENDER SIDE:**

```
import socket
import random
import time

s = socket.socket()
s.bind(("localhost", 1450))
s.listen(5)
c, adr = s.accept()
print(str(adr))
n = int(input("Enter number of frames: "))
N = int(input("Enter window size: "))
seq = 1  # is used to keep track of the window starting
frame = 1  # frame to send starts with 1

# send first N window size frames
for i in range(N):
    print('Frames sent ->', frame)
    c.send(str(frame).encode())
    frame += 1
    time.sleep(2)

timer = 5

# will start with acknowledgement frame of 1
while frame <= n:
    t = random.randint(1, 7)
    msg = c.recv(1).decode()
    msg = int(msg)

    if (msg != seq):
        # here we try to discard the already sent frames after failed frame
        continue
    if (timer > t):
        # if the timer is greater than random number be consider it as ack
        print("acknowledgement received")
        print('Frames sent ->', str(frame))
        # we will send next frame
        c.send(str(frame).encode())
        seq += 1
        frame += 1
        time.sleep(2)
    else:
    # if timer is less than the random number we consider as not received ack
        print('acknowledgement not received')
        frame = seq
        # we will again send the frames from window starting i.e. seq
        for i in range(N):
            print('Frames sent ->', frame)
            c.send(str(frame).encode())
            frame += 1
            time.sleep(2)
```

**RECEIVER SIDE:**

```
import socket
import time
s=socket.socket()
s.connect(("localhost", 1450))
while 1:
    msg=s.recv(2).decode()
    print("Received --> ",int(msg))
    s.send(str(msg).encode())
    time.sleep(1)
```

**OUTPUT:**

```
SENDER SIDE:                          RECEIVER SIDE:
Enter number of frames: 8             Received --> 1
Enter window size: 4                  Received --> 2
Frames sent -> 1                      Received --> 3
Frames sent -> 2                      Received --> 4
Frames sent -> 3                      Received --> 5
Frames sent -> 4                      Received --> 6
acknowledgement received              Received --> 3
Frames sent -> 5                      Received --> 4
acknowledgement received              Received --> 5
Frames sent -> 6                      Received --> 6
acknowledgement not received          Received --> 3
Frames sent -> 3                      Received --> 4
Frames sent -> 4                      Received --> 5
Frames sent -> 5                      Received --> 6
Frames sent -> 6                      Received --> 7
acknowledgement not received          Received --> 8
Frames sent -> 3
Frames sent -> 4
Frames sent -> 5
Frames sent -> 6
acknowledgement received
Frames sent -> 7
acknowledgement received
Frames sent -> 8
```

7. Write a Program to implement Sliding window protocol for Selective repeat.

**PROGRAM:**

**SENDER SIDE:**

```
import socket
import random
import time
s = socket.socket()
s.bind(("localhost",8038))
s.listen(5)
```

```
c, adr = s.accept()
print("from address”, str(adr), "connection has established")
n = int(input("Enter number of frames: "))
N = int(input("Enter window size: "))
seq = 1 # is used to keep track of the window starting
frame = 1 # frame to send starts with 1
# send first N window size frames
for i in range(N):
    print('Frames sent ->',frame)
    c.send(str(frame).encode())
    frame += 1
    time.sleep(2)
timer = 5 # will start with acknowledgement frame of 1
while frame <= n :
    t = random.randint(1,7)
    msg = c.recv(1).decode()
    msg = int(msg)
    print("Frame ", msg)
    if(timer > t):
        # if the timer is greater than random number be consider it as ack
        print("acknowledgement received")
        print('Frames sent ->',str(frame))
        # we will send next frame
        c.send(str(frame).encode())
        seq += 1
        frame += 1
        time.sleep(2)
    else:
        # if timer is less than the random number we consider as not received ack
        print('acknowledgement not received')
        # we will again send the frames from window starting i.e seq
        print('Frames sent ->',msg)
        c.send(str(msg).encode())
        time.sleep(2)
```

**RECEIVER SIDE:**

```
import socket
import time
s=socket.socket()
s.connect(("localhost", 8038))
while 1:
    msg=s.recv(2).decode()
    print("Received --> ",int(msg))
    s.send(str(msg).encode())
    time.sleep(1)
```

**OUTPUT:**

```
SENDER SIDE:                              RECEIVER SIDE:
Enter number of frames: 8                 Received --> 1
Enter window size: 4                      Received --> 2
Frames sent -> 1                          Received --> 3
Frames sent -> 2                          Received --> 4
Frames sent -> 3                          Received --> 1
Frames sent -> 4                          Received --> 5
```

```
Frame  1
acknowledgement not received
Frames sent -> 1
Frame  2
acknowledgement received
Frames sent -> 5
Frame  3
acknowledgement received
Frames sent -> 6
Frame  4
acknowledgement received
Frames sent -> 7
Frame  1
acknowledgement not received
Frames sent -> 1
Frame  5
acknowledgement received
Frames sent -> 8
```

```
Received --> 6
Received --> 7
Received --> 1
Received --> 8
```

## 8. Write a Program to implement Stop and Wait Protocol.

**PROGRAM:**

**SENDER SIDE:**

```
import socket
import time
import random
s=socket.socket()
s.bind(("localhost", 8020))
s.listen(5)
c, adr = s.accept()
print("connection to " + str(adr) + " established")
a=int(input("enter total number of frames"))
x = 0
print("sending -->", x)
c.send(str(x).encode())
while( a > 1 ):
    timer = 5
    t=random.randint(1,7)
    msg = c.recv(1).decode()
    if( timer > t):
        time.sleep(3)
        print("ack-->", msg)
        x=int(msg)
        print("sending -->", str(x))
        c.send(str(x).encode())
    else:
        time.sleep(3)
        print("timeout")
        print("sending again-->", x)
        c.send(str(x).encode())
        a=a+1
    a = a-1
```

**RECEIVER SIDE:**

```
import socket
s=socket.socket()
s.connect(("localhost", 8020))
while(1):
    msg=s.recv(1).decode()
    print("Received --> ", msg)
    x=int(msg)
    if(x==0):
        x=x+1
        s.send(str(x).encode())
    else:
        x=x-1
        s.send(str(x).encode())
```

**OUTPUT:**

**SENDER SIDE:**

```
enter total number of frames6
sending -->0
timeout
sending again-->0
timeout
sending again-->0
ack--> 1
sending -->1
ack--> 0
sending -->0
timeout
sending again-->0
timeout
sending again-->0
timeout
sending again-->0
ack--> 1
sending -->1
timeout
sending again-->1
timeout
sending again-->1
ack--> 0
sending -->0
timeout
sending again-->0
ack--> 1
sending -->1
```

**RECEIVER SIDE:**

```
Received --> 0
Received --> 0
Received --> 0
Received --> 1
Received --> 0
Received --> 0
Received --> 0
Received --> 0
Received --> 1
Received --> 1
Received --> 1
Received --> 0
Received --> 0
Received --> 1
```

## 9. Write a program for congestion control using leaky bucket algorithm

**PROGRAM:**

```
print("Enter bucket size, outgoing rate, number of inputs and incoming size")
bucketsize = int(input())
outgoing = int(input())
n = int(input())
incoming = int(input())
store=0
while n!= 0:
    print("Incoming size is ", incoming)
    if incoming <= (bucketsize-store):
        store += incoming
        print("Bucket buffer size is  " , store ," out of " , bucketsize)
    else:
        print("Packet loss : " , (incoming-(bucketsize-store)))
        store=bucketsize
        print("Bucket buffer size is  " ,store ," out of " , bucketsize)
    store -= outgoing;
    print("After outgoing: " ,store , " packets left out of ", bucketsize      ,"in
buffer")
    n=n-1
```

**OUTPUT:**

```
Enter bucket size, outgoing rate, number of inputs and incoming size

300

50

2

200

Incoming size is  200

Bucket buffer size is   200 out of 300

After outgoing:  150 packets left out of 300 in buffer

Incoming size is 200

Packet loss:  50

Bucket buffer size is   300 out of 300

After outgoing:  250 packets left out of 300 in buffer
```

10. Write a program to implement Dijkstra's algorithm to compute the shortest path through a graph.

**PROGRAM:**

```
INF = 1000

# search min function
def search_min(length, se, n):
    global v
    mi = 100
    for i in range(n):
        if se[i] == 0:
            if length[i] < mi:
                mi = length[i]
                v = i
    return v

se = [0] * 10
length = []
path = []
graph = []

n = int(input("Enter No of Vertexes: "))
print("enter the adjacency matrix: ")
for i in range(n):
    graph.append(list(map(int, input().split())))
s = int(input("Enter Source node: "))
# INTIALIZATION PART
for i in range(n):
    if graph[s][i] == 0:
        length.append(INF)
        path.append(0)
    else:
        length.append(graph[s][i])
        path.append(s)
se[s] = 1
length[s] = 0

# ITERATION PART
c = 1
while c:
    c = 0
    j = search_min(length, se, n)
    se[j] = 1
    for i in range(n):
        if se[i] != 1:
            if graph[i][j] != 0:
                if length[j] + graph[i][j] < length[i]:
                    length[i] = length[j] + graph[i][j]
                    path[i] = j
    for i in range(0, n):
        if se[i] == 0:
            c += 1
# OUTPUT
print("From(sourcevertex) To ", s)
print("\tPath\t\tLength\t\t\tShortest path ")
for i in range(n):
    if i != s:
        print("\t\t%d\t\t\t%d" % (i, length[i]), end='\t')
    j = i
    while j != s:
        print("\t%d->%d" % (j, path[j]), end='\t')
        j = path[j]
    print()
```

**OUTPUT:**

```
Enter No of Vertexes: 4

enter the adjacency matrix:

0 6 0 1

6 0 2 4

0 2 0 1

1 4 1 0

Enter Source node: 0

From(sourcevertex) To  0



      Path        Length                  Shortest path


            1                 4           1->2        2->3        3->0

            2                 2           2->3        3->0

            3                 1           3->0
```

11. Write a Program to implement Distance vector routing algorithm by obtaining routing table at each node (Take an example subnet graph with weights indicating delay between nodes).

**PROGRAM:**

```
# IMPLEMENTATION OF DISTANCE VECTOR:
INFINITY = 10000
length = [[0 for _ in range(10)] for _ in range(10)]
path = [[0 for _ in range(10)] for _ in range(10)]
se = [0] * 10
adj = []
s = c = 0

n = int(input("Enter No of Routers: "))
print("Enter Adjacency Matrix")
for i in range(n):
    adj.append(list(map(int, input().split())))

# Initialization Part
for i in range(n):
    for j in range(n):
        if adj[i][j] == 0 and i != j:
            length[i][j] = INFINITY
            path[i][j] = 0

        else:
            length[i][j] = adj[i][j]
            path[i][j] = j

        if i == j:
            path[i][j] = 30

# Iteration Part
c = 1
while c:
    c = 0
    for s in range(n):
        for j in range(n):
            if adj[s][j]:
                for i in range(n):
                    if (length[s][j] + length[j][i]) <
length[s][i]:
```

```
                        length[s][i] = length[s][j] + 
length[j][i]
                        path[s][i] = j

    for s in range(n):
        for i in range(n):
            if length[s][i] == INFINITY:
                c += 1

print("\nRouting table\n\n")
for i in range(65, 65 + n):
    print("    ", chr(i), "   ", end=' ')
print("\n---------------------------------------")

for i in range(n):
    print(chr(i + 65), end=' ')
    for s in range(n):
        print(" %3d%3c |" % (length[s][i], path[s][i] + 65), 
end='')
    print()
```

**OUTPUT:**

```
Enter No of Routers: 4
Enter Adjacency Matrix
0 6 0 1
6 0 2 4
0 2 0 1
1 4 1 0

Routing table


     A          B          C          D
-------------------------------------
A    0  _ |   4  D |   2  D |   1  A |
B    5  D |   0  _ |   2  B |   3  C |
C    2  D |   2  C |   0  _ |   1  C |
D    1  D |   3  C |   1  D |   0  _ |
```

12. Write a Program to implement Broadcast tree by taking subnet of hosts.

**PROGRAM:**

```
# IMPLEMENTATION OF BROADCASTING
gptr = [[0 for _ in range(10)] for _ in range(10)]
bt = [[0 for _ in range(10)] for _ in range(10)]
st = [[0 for _ in range(10)] for _ in range(10)]

a = b = stcost = l = s = m = k = p = t = mi = 0
x = y = [0] * 10
n = int(input("Enter No of Routers: "))
print("Enter time delays between routers")

for i in range(0, n):
    for j in range(i + 1, n):
        print(i, "->", j, " time delay ")
        gptr[i][j] = int(input())
        gptr[j][i] = gptr[i][j]

for i in range(0, n):
    for j in range(1, n):
        st[i][j] = 0
        bt[i][j] = 0
        if i == j:
            gptr[i][j] = 0

# /***********Iteration************/
t = n
while t > 1:
    mi = 100
    for i in range(0, n):
        for j in range(i + 1, n):
            if gptr[i][j]:
                if gptr[i][j] < mi:
                    mi = gptr[i][j]
                    l = i
                    s = j
    gptr[l][s] = 0
    c1 = 0
    for i in range(a):
        if x[i] == l:
            c1 += 1
        if x[i] == s:
            c1 += 1

    if c1 == 0:
        x[a] = l
        a += 1
    c2 = 0
    for i in range(b):
        if y[i] == s:
            c2 += 1
        if y[i] == l:
            c2 += 1
    if c2 == 0:
        y[b] = s
        b += 1
    if c1 != 2 and c2 != 2:
        st[l][s] = mi
        t -= 1

print("Path\tTimedelay")

for i in range(0, n):
    for j in range(i + 1, n):
        if (st[i][j]):
            print(i, " --> ", j, " ", st[i][j])
            stcost += st[i][j]
            bt[i][j] = 1
            bt[j][i] = 1

print("It takes minimum ", stcost, " seconds to broad cast
data in given subnet")
print("\nBroad cast tree is \n")

for i in range(0, n):
    for j in range(0, n):
        print(bt[i][j], end=' ')
    print()
```

**OUTPUT:**

```
Enter No of Routers: 4
Enter time delays between routers
0 -> 1 time delay
6
0 -> 2 time delay
0
0 -> 3 time delay
1
1 -> 2 time delay
2
1 -> 3 time delay
4
2 -> 3 time delay
1
Path        Timedelay
0  -->  3       1
1  -->  2       2
2  -->  3       1
It takes minimum  4 seconds to broad cast data in given subnet

Broad cast tree is

0 0 0 1
0 0 1 0
0 1 0 1
1 0 1 0
```