1.Implement the data link layer farming methods such as character-stuffing and bit stuffing.

```
Program .  def char_stuff(data):
    flag = 'F'
    esc  = 'E'
    stuffed = ""
 for ch in data:
        if ch == flag or ch == esc:
            stuffed += esc + ch   # Stuffing
        else:
            stuffed += ch
    return flag + stuffed + flag  # Final frame
data = input("Enter data for Character Stuffing: ")
print("Stuffed Frame:", char_stuff(data))
def bit_stuff(data):
    count = 0
    res = ""
    for b in data:
        res += b
        if b == '1':
            count += 1
            if count == 5:       # after 5 ones → stuff 0
                res += '0'
                count = 0
        else:
            count = 0
    return "01111110" + res + "01111110"   # Add flags

data = input("Enter bit stream: ")
print("Stuffed Frame:", bit_stuff(data))
```

2. Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP

Program.

```
def crc(data, poly):
    # Append zeros
    d = list(data + '0' * (len(poly) - 1))
    p = list(poly)

    # Perform division
    for i in range(len(data)):
        if d[i] == '1':    # Only XOR when bit is 1
            for j in range(len(p)):
                d[i + j] = str(int(d[i + j]) ^ int(p[j]))

    # Return last n-1 bits
    return ''.join(d[-(len(poly) - 1):])



data = input("Data poly: ")

print("CRC-12 :", crc(data, "1100000001111"))
print("CRC-16 :", crc(data, "11000000000000101"))
print("CRC-CCITT :", crc(data, "10001000000100001"))
```

3. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.

Program.

```
def go_back_n():
    frames = int(input("Enter total number of frames: "))
    window_size = int(input("Enter window size: "))
  print("\nOUTPUT:\n")
    base = 0
    next_frame = 0
    while base < frames:
# Sender sends frames in the window
        while next_frame < base + window_size and next_frame < frames:
            print(f"Sender : Sending Frame {next_frame}")
            next_frame += 1

        # Receiver acknowledges all sent frames
        for f in range(base, next_frame):
            print(f"Receiver : Acknowledgement received for Frame {f}")

        # Slide the window
        base = next_frame
    print("\nTransmission Complete.")
go_back_n()
```

4. Implement Dijsktra's algorithm to compute the shortest path through a network .

Program.

```
import heapq
def dijkstra(graph, start):
    n = len(graph)
    dist = [float('inf')] * n
    dist[start] = 0
    pq = [(0, start)]
    while pq:
        d, u = heapq.heappop(pq)
        if d > dist[u]:
            continue
        for v, w in enumerate(graph[u]):
            if w > 0 and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                heapq.heappush(pq, (dist[v], v))
    return dist
graph = [
    [0, 4, 0, 0, 0, 0, 8, 0],
    [4, 0, 8, 0, 0, 0, 11, 0],
    [0, 8, 0, 7, 0, 4, 0, 2],
    [0, 0, 7, 0, 9, 14, 0, 0],
    [0, 0, 0, 9, 0, 10, 0, 0],
    [0, 0, 4, 14, 10, 0, 2, 0],
    [8, 11, 0, 0, 0, 2, 0, 1],
    [0, 0, 2, 0, 0, 0, 1, 0]
]
print(dijkstra(graph, 0))
```

5. Take an example subnet of hosts and obtain a broadcast tree for the subnet.

Program.

```
import sys

def main():

    n = int(input("Enter the number of nodes: "))


    # Read adjacency matrix

    adj = [[0]*(n+1) for _ in range(n+1)]

    print("Enter adjacency matrix (0 or 1):")

    for i in range(1, n+1):

        for j in range(1, n+1):

            adj[i][j] = int(input())


    root = int(input("Enter the root node: "))


    print(f"\nBroadcast Tree from Root {root}:")

    for j in range(1, n+1):

        if adj[root][j] == 1:

            print(f"{root} → {j}")


if __name__ == "__main__":

    main()
```

6. Implement distance vector routing algorithm for obtaining routing tables at each node.

Program.

```
print("Distance Vector Routing Algorithm\n")


n = int(input("Enter number of nodes: "))
cost = [list(map(int, input().split())) for _ in range(n)]
dist = [row[:] for row in cost]


for it in range(1, n):
    print(f"\nIteration {it}:")
    for i in range(n):
        for j in range(n):
            for k in range(n):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
        print(f"Routing table for node {i}: {dist[i]}")


print("\nFinal Routing Tables:")
for i in range(n):
    print(f"Node {i}: {dist[i]}")
```

7. Implement data encryption and data decryption .

Program.

```
from cryptography.fernet import Fernet

# Step 1: Generate a key for encryption and decryption
key = Fernet.generate_key()
cipher = Fernet(key)

# Step 2: Take input from user
data = input("Enter text to encrypt: ").encode()   # convert string to bytes

# Step 3: Encrypt the data
encrypted_data = cipher.encrypt(data)
print("\n🔐 Encrypted text:", encrypted_data.decode())

# Step 4: Decrypt the data
decrypted_data = cipher.decrypt(encrypted_data)
print("🔓 Decrypted text:", decrypted_data.decode())

# Step 5: Display the key
print("\nKeep this secret key safe if you want to decrypt later:")
print(key.decode())
```

8. Write a program for congestion control using Leaky bucket algorithm.

Program.

```
bucket_size = int(input("Bucket size: "))
output_rate = int(input("Output rate: "))
n = int(input("No. of packets: "))

bucket = 0

for i in range(n):
```

```
incoming = int(input(f"Packet {i+1} size: "))

if bucket + incoming > bucket_size:
    print("Packet dropped!")
else:
    bucket += incoming

# leak
bucket -= output_rate
if bucket < 0:
    bucket = 0

print("Bucket now:", bucket)
```

9. Write a program for frame sorting techniques used in buffers.

Program.

```
# Frame Sorting with user input
# Taking number of frames
n = int(input("Enter number of frames: "))
frames = []

# Taking frame sequence numbers from user
for i in range(n):
    num = int(input(f"Enter frame {i+1} sequence number: "))
    frames.append(num)

print("\nFrames received (unsorted):")
print(frames)

# Sorting the frames
frames.sort()

print("\nFrames after sorting:")
print(frames)

# Delivering frames
print("\nDelivering frames in order:")
for f in frames:
    print("Delivering frame:", f)
```