Program 4:

def xor():

    """Perform XOR operation."""

    global check\_value, gen\_poly

    for j in range(1, len(gen\_poly)):

        check\_value[j] = '0' if check\_value[j] == gen\_poly[j] else '1'

def crc():

    """Cyclic Redundancy Check computation."""

    global check\_value, data, gen\_poly, data\_length

    # Initialize check\_value with the initial data bits

    check\_value = list(data[:len(gen\_poly)])

    i = len(gen\_poly)

    while i <= data\_length + len(gen\_poly) - 1:

        if check\_value[0] == '1':

            xor()

        check\_value.pop(0)  # Shift left

        if i < data\_length:

            check\_value.append(data[i])

        else:

            check\_value.append('0')

        i += 1

def receiver():

    """Check for errors on the receiver side."""

    global data, check\_value

    received\_data = input("Enter the received data: ")

    print("\n-----------------------------")

    print(f"Data received: {received\_data}")

    # Set data to the received data and calculate CRC

    data = received\_data

    crc()

    # Check if there is any error

    if any(bit == '1' for bit in check\_value[:len(gen\_poly) - 1]):

        print("\nError detected\n")

    else:

        print("\nNo error detected\n")

# Main program

if \_name\_ == "\_main\_":

    # Input the data and generator polynomial

    data = input("Enter data to be transmitted: ")

    gen\_poly = input("Enter the generating polynomial: ")

    data\_length = len(data)

    # Append n-1 zeros to the data

    data += '0' \* (len(gen\_poly) - 1)

    print("\n----------------------------------------")

    print(f"Data padded with n-1 zeros: {data}")

    print("\n----------------------------------------")

    # Compute the CRC value

    crc()

    crc\_value = ''.join(check\_value[:len(gen\_poly) - 1])

    print(f"\nCRC or Check value is: {crc\_value}")

    # Append CRC to the original data

    transmitted\_data = data[:data\_length] + crc\_value

    print("\n----------------------------------------")

    print(f"Final data to be sent: {transmitted\_data}")

    print("\n----------------------------------------\n")

    # Call the receiver function

    receiver()

Program 05:

def Sliding\_window():

    window\_size = int(input("Enter the window size:"))

    num\_frames = int(input("Enter no of frames next to transmit:"))

    frames = []

    print(f"enter {num\_frames} frames:")

    for i in range(1,num\_frames+1):

        frame = int(input(f"frame{i}:"))

        frames.append(frame)

    print("-------------")

    print(f"after sending given {window\_size} frame at each stage, sender wants for acknowledgement sent by the recieve \n")

    for i in range(1,num\_frames+1):

        print(frames[i-1], end=" ")

        if i % window\_size == 0 or i == num\_frames:

            print("Acknowledge is above frame is sent is recieved by sender \n")

if \_\_name\_\_ == "\_\_main\_\_":

    Sliding\_window()

Program 06:

INF = float('inf')

def bellmanford(graph,vertices,edges,sources):

    distance = [INF]\*vertices

    distance[sources] = 0

    for \_ in range(vertices-1):

        for edge in graph:

            u,v,w = edge

            if distance[u] != INF and distance[u] + w < distance[v]:

                distance[v] = distance[u] + w

    for edge in graph:

        u,v,w = edge

        if distance[u] != INF and distance[u] + w < distance[v]:

            print("Graph contains negative weight cycle")

            return

    print("vertex distance from source")

    for i in range(vertices):

        print(f"{i}\t\t{distance[i]}")

def main():

    vertices = 6

    edges = 8

    graph = [(0,1,5),(0,2,7),(1,2,3),(1,3,4),(1,4,6),(3,4,-1),(3,5,2),(4,5,-3)]

    bellmanford(graph,vertices,edges,0)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Program 07:

def gcd(m, n):

    while n != 0:

        r = m % n

        m = n

        n = r

    return m

def rsa():

    n, e, d, phi = 0, 0, 0, 0

    encrypted = []

    decrypted = []

    # Input message and prime numbers p and q

    message = input('Enter the message to be encrypted: ')

    p = int(input('Enter the value of p (prime number): '))

    q = int(input('Enter the value of q (prime number): '))

    # Calculate n and phi(n)

    n = p \* q

    phi = (p - 1) \* (q - 1)

    # Find public key exponent e

    for i in range(2, phi):

        if gcd(i, phi) == 1:

            e = i

            break

    # Find private key exponent d

    for i in range(2, phi):

        if (e \* i - 1) % phi == 0:

            d = i

            break

    print(f"Public Key (e, n): ({e}, {n})")

    print(f"Private Key (d, n): ({d}, {n})")

    # Convert message to numbers

    nummes = [ord(c) for c in message]

    # Encrypt the message

    for num in nummes:

        encrypted.append(pow(num, e, n))

    print("Encrypted message:", encrypted)

    # Decrypt the message

    for num in encrypted:

        decrypted.append(pow(num, d, n))

    decrypted\_message = ''.join(chr(num) for num in decrypted)

    print("Decrypted message:", decrypted\_message)

if \_\_name\_\_ == "\_\_main\_\_":

    rsa()

program 08:

client:

import socket

def tcp\_client():

    # Create a socket

    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Connect to the server

    client\_socket.connect(("127.0.0.1", 4000))

    print("Connected to the server.")

    # Enter the filename

    filename = input("Enter the filename: ")

    # Send the filename to the server

    client\_socket.sendall(filename.encode())

    # Receive the file content from the server

    print("\nReceived file content:")

    while True:

        data = client\_socket.recv(1024).decode()

        if not data:

            break

        print(data, end="")

    # Close the connection

    client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

    tcp\_client()

server:

import socket

def tcp\_server():

    # Create a server socket

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Bind the socket to an address and port

    server\_socket.bind(("127.0.0.1", 4000))

    server\_socket.listen(1)

    print("Server ready for connection.")

    # Accept a client connection

    conn, addr = server\_socket.accept()

    print("Connection established with:", addr)

    # Receive the filename from the client

    filename = conn.recv(1024).decode()

    print(f"Client requested file: {filename}")

    try:

        # Open and read the file

        with open(filename, "r") as file:

            for line in file:

                conn.sendall(line.encode())  # Send file content to the client

    except FileNotFoundError:

        conn.sendall(f"Error: File '{filename}' not found.".encode())

    # Close the connection

    conn.close()

    server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

    tcp\_server()