REPORT

On

COMPUTER NETWORKS LAB (Course Code: V23CSL09)

B. TECH V Semester (2023-2027 Batch)

Academic Year:2025-26

SRI VASAVI ENGINEERING COLLEGE(Autonomous)

Pedatadepalli, Tadepalligudem-534101

Department of Computer Science & Engineering
(Accredited by NBA)

SRI VASAVI ENGINEERING COLLEGE(Autonomous)

PEDATADEPALLI, TADEPALLIGUDEM.

Certificate

This is to certify that this a bonafide record of practical work done in **Computer Networks Lab (Course Code : V23CSL09)** by Mister **Kommini Joseph Graham Staines** bearing roll number **23A81A0G1** of **CSE** Branch **V** Semester during the academic year 2025-26.

No of Experiments Done: 16

Faculty In Charge of the Laboratory

Head of the Department

Examiner 1

Examiner 2

INDEX

SNO.	EXPERIMENT	DATE	Pg No	SIGNATURE
1.	Study of network devices in detail and connect computers in Local area Network.		**4-7**	
2.	Develop a program to implement data link layer framing methods such as i) Character stuffing ii) Bit stuffing			
3.	Develop a program to implement data link layer framing method checksum.			
4.	Develop a program for Hamming code generation for error detection and correction.			
5.	Develop a program to implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCITT.			
6.	Develop a programto implement Sliding Window Protocol for Go Back N.			
7.	Develop a program to implement Sliding Window Protocol for Selective Repeat.			
8.	Develop a program to implement Stop and Wait Protocol.			
9.	Develop a program for congestion control using Leaky Bucket Protocol.			
10.	Develop a program to implement Dijkstra's algorithm to compute the Shortest path through a graph.			
11.	Develop a program to implement Distance vector routing algorithm by obtaining routing table at each node.			
12.	Develop a program to implement Broadcast tree by taking subnet of hosts.			
13.	Wireshark : i) packet capture using wireshark ii) starting wireshark iii)viewing captured packets iv) analysis and statistics & filters.			
14.	Discover active hosts,open ports and running services in a network using Nmap.			
15.	Find operating system in a host using Nmap.			
16.	Do the following using NS2 Simulator i) NS2 simulator -Introduction ii) simulate to find the number of packets dropped. iii) simulate to find the number of packets dropped by TCP/UDP iv) simulate to find the number of packets dropped due to congestion. v) simulate to compare Data Rate & Throughput.			

EXPERIMENT 1:

Study of Network devices in detail and connect the computers in Local Area Network.

A) Network devices:

these are hardware components used to connect computers, printers, phones and other devices in a network. They help in transmitting and receiving data efficiently.

Information about Network Devices

1) Router

Function: Connects different networks (e.g. home network to the internet)
key role: Routes data between networks using IP address
Example: Home WiFi Router

2) Switch

Function: Connects multiple devices within the same local area network

3)LAN

key Role: Uses MAC address to forward data to correct devices.
Note: More efficient than hubs; supports full duplex communication.

4) Modem

Function: Modulates & demodulates signals for internet access.
Keyrole: Connects a home or office network to ISP via telephone, cable or fiber line.

5) Hub (Deprecated)

Function: Broadcast data to all devices on a network segment
key role: Simple data distribution, no intelligence
limitation: Creates more network traffic, replaced by switches

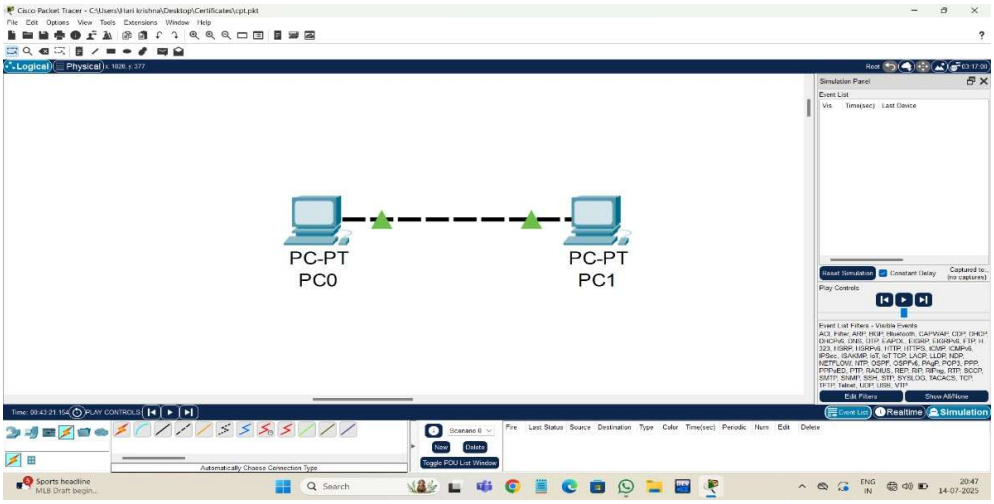
6)Repeater

Function: Boosts signal over longer distances.
key role: Extends the coverage area of a network, especially wireless
usecase: large buildings, or areas with signal dead zones,

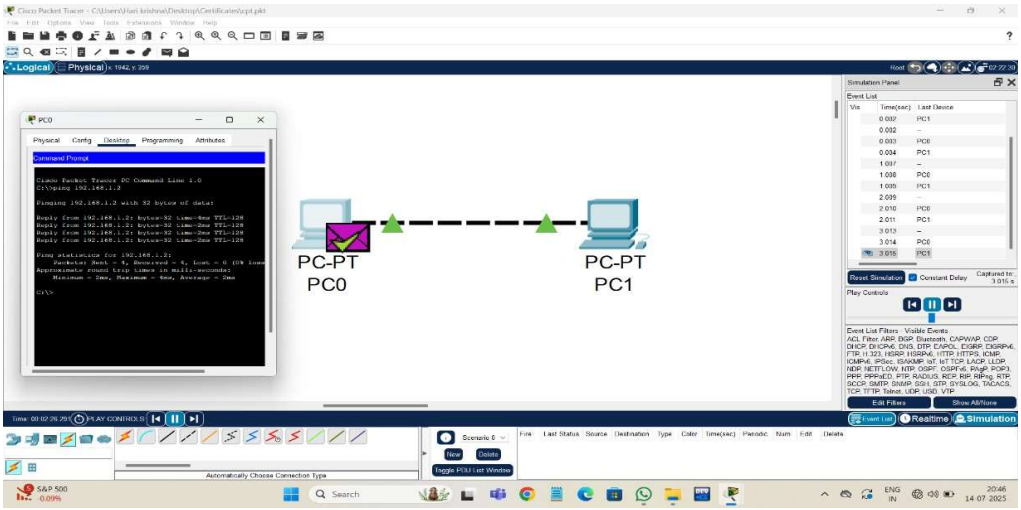
7)Network Interface Card (NIC)

Function: Allows a device to connect to a network
keyrole: Communicate with other devices over a LAN or the internet
Form: Can be wired (Ethernet) or wireless (wi-fi)

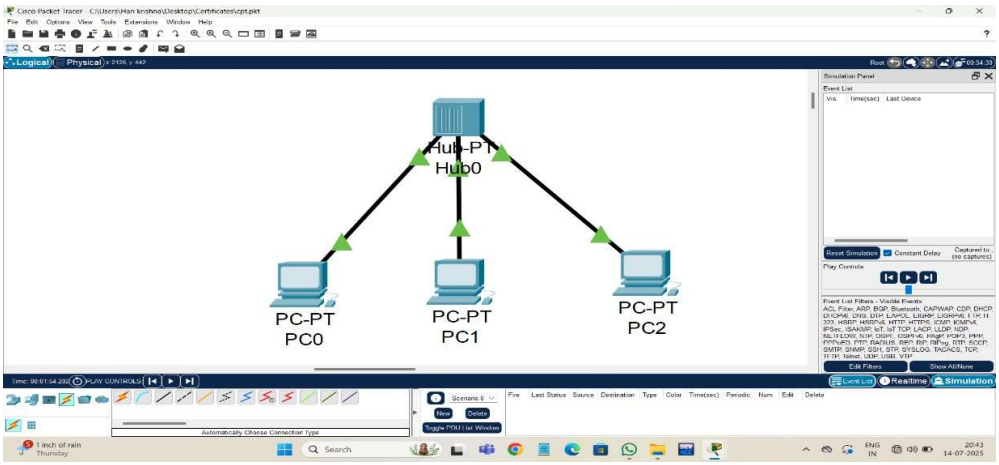
1(a) Connecting 2 devices using a cable:



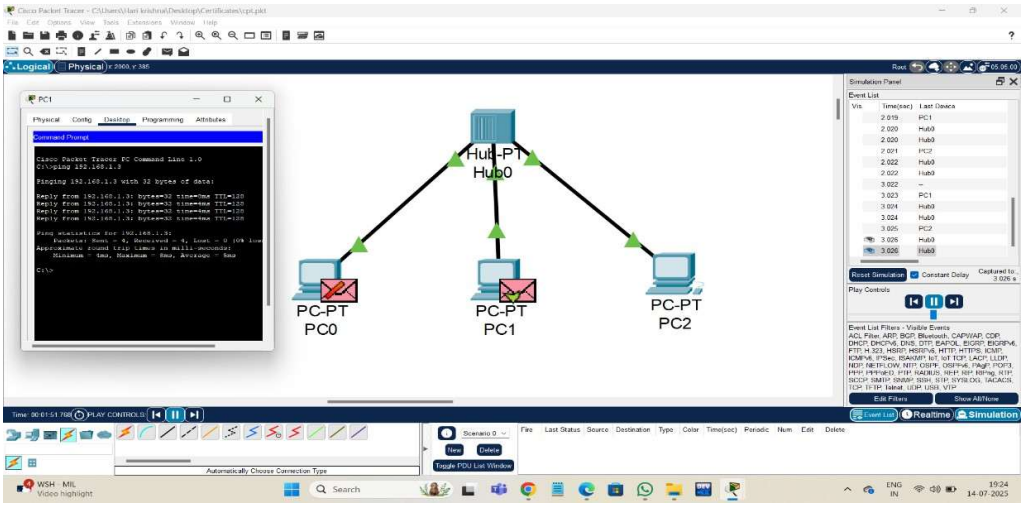
1(b) passing data packet from one PC to another which are connected using a cable:



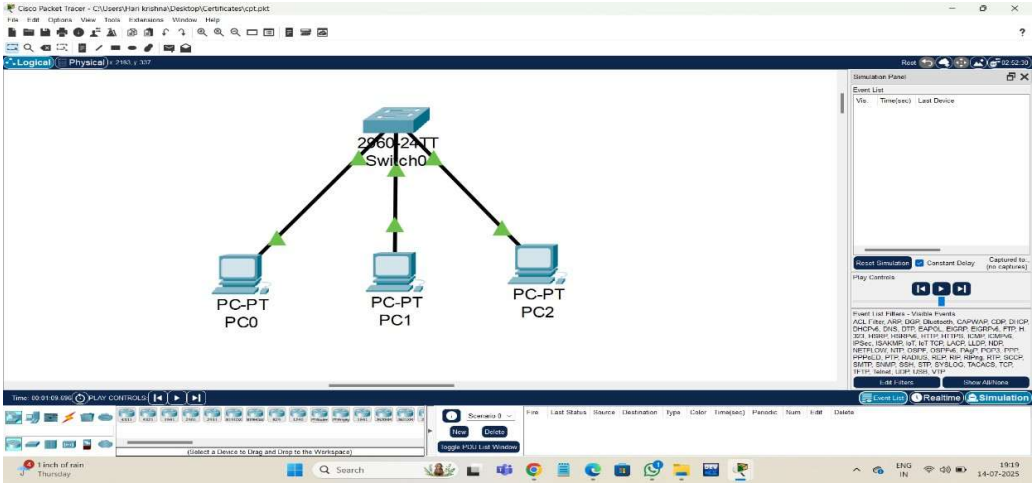
2(a) connecting devices using a hub:



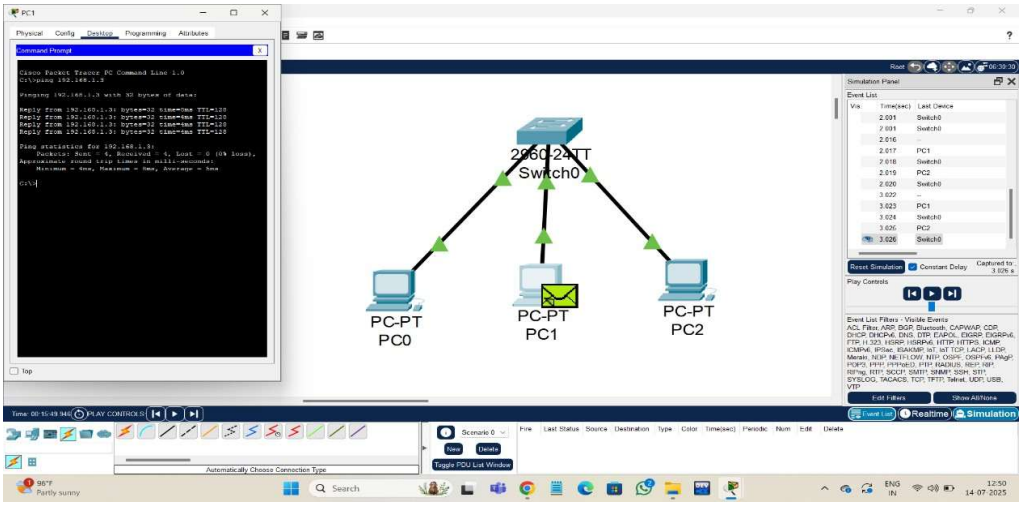
2(b) sending data packets between devices connecting using a hub:



3(a) connecting PC's using Switch:



3(b) sending a data packet from one PC to another using a switch:



EXPERIMENT 2:

Develop a Program to implement the data link layer framing methods such as i) Character stuffing ii) bit stuffing.

i)Character Stuffing:

```
FRAME_FLAG = "01111110"
ESCAPE_BYTE = "01111101"

def perform_stuffing(input_data):
    stuffed_data = FRAME_FLAG
    for i in range(0, len(input_data), 8):
        byte_chunk = input_data[i:i+8]
        if byte_chunk == FRAME_FLAG or byte_chunk == ESCAPE_BYTE:
            stuffed_data += ESCAPE_BYTE
        stuffed_data += byte_chunk
    stuffed_data += FRAME_FLAG
    return stuffed_data

if __name__ == "__main__":
    total_packets = int(input("Enter number of packets: "))
    combined_stuffed = ''
    for i in range(total_packets):
        packet = input("Enter binary data for packet {}: ".format(i+1))
        combined_stuffed += perform_stuffing(packet)
    print("Stuffed Data (combined binary string): ")
    print(combined_stuffed)
```

output:

```
Enter number of packets: 2
Enter binary data for packet 1: 01111001011101001110010011110 0111100 01111000 0111100
Enter binary data for packet 2:  0111101 0111101 0111100 0111101 0111101 0111101
Stuffed Data (combined binary string):
011111100111100101110100111001001111100 0111100 01111000 011111000111111001111110 0111101 0111101 0111100 0111101
 0111101 011110101111110
```

i)Character Destuffing:

```
FRAME_FLAG = "01111110"
ESCAPE_BYTE = "01111101"

def perform_destuffing(stuffed_input):
    if not stuffed_input.startswith(FRAME_FLAG) or not stuffed_input.endswith(FRAME_FLAG):
        return "ERROR: Invalid frame structure"
    content = stuffed_input[len(FRAME_FLAG):-len(FRAME_FLAG)]
    original_data = ""
    i = 0
    while i < len(content):
        byte_chunk = content[i:i+8]
        if byte_chunk == ESCAPE_BYTE:
            if i + 16 <= len(content):
                original_data += content[i+8:i+16]
                i += 16
            else:
                return "Error"
        else:
            original_data += byte_chunk
            i += 8
    return original_data

def perform_destuffing_multiple(stuffed_input):
    index = 0
    packet_count = 1
    while index < len(stuffed_input):
        start = stuffed_input.find(FRAME_FLAG, index)
        if start == -1:
            break
        end = stuffed_input.find(FRAME_FLAG, start + len(FRAME_FLAG))
        if end == -1:
            break
        one_frame = stuffed_input[start:end + len(FRAME_FLAG)]
        original_data = perform_destuffing(one_frame)
        if not original_data.startswith("ERROR"):
            print("Original packet {}: {}".format(packet_count, original_data))
```

```
        packet_count += 1
        index = end + len(FRAME_FLAG)

def main():
    stuffed_data = input("Enter stuffed binary string: ")
    print("Destuffed Packets:")
    perform_destuffing_multiple(stuffed_data)

if __name__ == "__main__":
    main()
```

output:

```
Enter stuffed binary string: 0111111001111001011101001110010011110 0 0111100 01111000 0111100011111100111111 0
11101 0111101 0111100 0111101 0111101 011110101111110
Destuffed Packets:
Original packet 1: 0111100101110100111001001111100 0111100 01111000 0111100
Original packet 2:  0111101 0111101 0111100 0111101 0111101 0111101
```

ii)BIT Stuffing

```
Frame_Flag = "01111110"

def perform_bit_stuffing(data):
    result = ""
    count = 0
    for bit in data:
        result += bit
        if bit == '1':
            count += 1
            if count == 5:
                result += '0'
                count = 0
        else:
            count = 0
    return result

def main():
    total_packets = int(input("Enter number of packets: "))
    combined_data = ""
    for i in range(total_packets):
        packet = input("Enter binary data for packet {}: ".format(i+1))
        if not all(bit in '01' for bit in packet):
            print("ERROR")
            return
```

```
        combined_data += perform_bit_stuffing(packet)
    final_output = Frame_Flag + combined_data + Frame_Flag
    print("Bit stuffed Data:")
    print(final_output)

if __name__ == "__main__":
    main()
```

output:

```
Enter number of packets: 2
Enter binary data for packet 1: 1111101111110111111011111101111110111111011111101111110
Enter binary data for packet 2: 1010101010101010101010101010101010101010101010101010
Bit stuffed Data:
0111111011111001111101011111010111110101111101011111010111110101010101010101010101010101010101010101010101010101010
01111110
```

ii)BIT Destuffing

```
FRAME_FLAG = "01111110"

def perform_bit_destuffing(data):
    result = ""
    count = 0
    i = 0
    while i < len(data):
        bit = data[i]
        result += bit
        if bit == '1':
            count += 1
            if count == 5 and i + 1 < len(data) and data[i+1] == '0':
                i += 1
                count = 0
        else:
            count = 0
        i += 1
    return result

def extract_frames(stuffed_input):
    packets = []
    start = 0
    while True:
        frame_start = stuffed_input.find(FRAME_FLAG, start)
        if frame_start == -1:
            break
        frame_end = stuffed_input.find(FRAME_FLAG, frame_start + len(FRAME_FLAG))
        if frame_end == -1:
            break
        frame_content = stuffed_input[frame_start + len(FRAME_FLAG):frame_end]
        if frame_content:
            packets.append(frame_content)
```

```
        start = frame_end
    return packets

def main():
    stuffed_input = input("Enter stuffed binary string: ")
    frames = extract_frames(stuffed_input)
    if not frames:
        print("ERROR: No packets found.")
        return
    print("Destuffed packets:")
    for idx, packet in enumerate(frames, start=1):
        destuffed_packet = perform_bit_destuffing(packet)
        print("Packet {}: {}".format(idx, destuffed_packet))

if __name__ == "__main__":
    main()
```

output:

```
Enter the number of bits in stuffed packet:
52
0 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 1 0
Destuffed Output:
01111100110111111001111111111111111001100111110
```

EXPERIMENT 3:

Develop a Program to implement data link layer farming method checksum.

```
def add_bin(x, y, size):
    s = bin(int(x, 2) + int(y, 2))[2:]
    if len(s) > size:
        # Adjust for wrap-around: add overflow carry to least significant bits
        s = bin(int(s[-size:], 2) + int(s[:-size], 2))[2:]
    return s.zfill(size)

def ones_complement(bits):
    return ''.join('0' if ch == '1' else '1' for ch in bits)

def sender():
    bits = input("Enter continuous data bits: ").strip()
    size = int(input("Enter block size (m): "))
    parts = [bits[i:i+size] for i in range(0, len(bits), size)]
    temp = parts[0]
    for p in parts[1:]:
        temp = add_bin(temp, p, size)
    chk = ones_complement(temp)
    print("checksum:", chk)
    print("Final stream to send:", bits + chk)
```

```
if __name__ == "__main__":
    sender()
```

output:

```
Enter continuous data bits: 1100101011111000100100111101111000100100 11
Enter block size (m): 8
checksum: 10100010
Final stream to send: 1100101011111000100100111101111000100100 1110100010
```

```
def add_bin(x, y, size):
    s = bin(int(x, 2) + int(y, 2))[2:]
    if len(s) > size:
        s = bin(int(s[-size:], 2) + int(s[:-size], 2))[2:]
    return s.zfill(size)

def ones_complement(bits):
    return ''.join('0' if ch == '1' else '1' for ch in bits)

def receive():
    received = input("Enter received data bits (with checksum): ").strip()
    size = int(input("Enter block size (m): "))
    parts = [received[i:i+size] for i in range(0, len(received), size)]
    temp = parts[0]
    for p in parts[1:]:
        temp = add_bin(temp, p, size)
    final_check = ones_complement(temp)
    if set(final_check) == {'0'}:
        print("No error detected!")
    else:
        print("Error detected in received data!!")

if __name__ == "__main__":
    receive()
```

output:

```
Enter received data bits (with checksum): 1100101011111000100100111101111000100100 1110100010
Enter block size (m): 8
Error detected in received data!!
```

EXPERIMENT 5:

Develop a Program to implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCITT.

CRC:CRC is a error detection technique where the sender divides the data by a fixed generator polynomial and the remainder is appended to the data for transmission .

```
def required_redundant_bits(m):
    r = 1
    while (2 ** r) < (m + r + 1):
        r += 1
    return r


def create_hamming(data):
    m = len(data)
    r = required_redundant_bits(m)
    n = m + r
    code = ["0"] * n
    d = 0
    for i in range(n):
        # Insert data bits into non-parity positions (skip powers of 2)
        if (i + 1) & (i) != 0:  # If (i+1) is not power of two
            code[i] = data[d]
            d += 1
    # Calculate parity bits
    for i in range(r):
        pos = 2 ** i
        ones = 0
        for j in range(pos - 1, n, 2 * pos):
            ones += code[j:j + pos].count("1")
        code[pos - 1] = str(ones % 2)
    return "".join(code), r
```

```
m = int(input("Enter number of code bits (m): "))
data = input(f"Enter {m} bit data: ").strip()
codeword, r = create_hamming(data)
print("Number of redundant bits:", r)
print("Hamming code to send:", codeword)
print("Length of codeword:", len(codeword))
```

output:

```
Enter number of code bits (m): 7
Enter 7 bit data: 1011001
Number of redundant bits: 4
Hamming code to send: 10100111001
Length of codeword: 11
```

```
def locate_error(code, r):
    n = len(code)
    err = 0
    for i in range(r):
        pos = 2**i
        ones = 0
        for j in range(pos-1, n, 2*pos):
            ones += code[j:j+pos].count("1")
        if ones % 2 != 0:
            err += pos
    return err

def retrieve_data(code):
    n = len(code)
    res = []
    skip_pow = 1
    for i in range(1, n+1):
        if i == skip_pow:
            skip_pow *= 2
        else:
            res.append(code[i-1])
    return "".join(res)

cw = input("Enter received Hamming code: ").strip()
n = len(cw)
r = 1
while (2**r) < (n+1):
    r += 1
err = locate_error(cw, r)
cw = list(cw)
if err == 0:
    print("NO error detected!")
```

```
else:
    print("Error detected at position:", err)
    cw[err-1] = "1" if cw[err-1] == "0" else "0"
corrected = "".join(cw)
print("Corrected code word:", corrected)
print("Extracted data word:", retrieve_data(corrected))
```

output:

```
Enter received Hamming code:  10100111001
NO error detected!
Corrected code word: 10100111001
Extracted data word: 1011001
```

EXPERIMENT 5:

Develop a Program to implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCITT.

CRC:CRC is a error detection technique where the sender divides the data by a fixed generator polynomial and the remainder is appended to the data for transmission .

```
dataword = input("Enter the dataword (binary): ").strip()
print("Choose Generator:")
print("1. User Generator\n2. CRC-12\n3. CRC-16\n4. CRITT")
choice = input("choice (1-4): ").strip()
if choice == '1':
    generator = input("Enter generator polynomial (binary): ").strip()
elif choice == '2':
    generator = "110000000111"
elif choice == '3':
    generator = "11000000000000101"
elif choice == '4':
    generator = "10001000000100001"
else:
    raise SystemExit("Invalid choice.")
r = len(generator) - 1
data = [int(b) for b in dataword] + [0] * r
m = len(dataword)
g = [int(b) for b in generator]

for p in range(m):
    if data[p] == 1:
        for j in range(len(g)):
            data[p + j] ^= g[j]
remainder = data[m:m + r]
```

```
codeword = dataword + ''.join(str(b) for b in remainder)
print("Generator:", generator)
print("Remainder:", ''.join(str(b) for b in remainder))
print("Code word:", codeword)
```

output 1:

```
Enter the dataword (binary): 11010011101100
Choose Generator:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 1
Enter generator polynomial (binary): 10111
Generator: 10111
Remainder: 1100
Code word: 11010011101001100
```

output 2:

```
Enter the dataword (binary): 10110011
Choose Generator:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 2
Generator: 110000000111
Remainder: 11000010011
Code word: 10110011111000010011
```

output 3:

```
Choose Generator:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 3
Generator: 11000000000000101
Remainder: 0000000001001110
Code word: 1110100000000001001110
```

output 4:

```
Enter the dataword (binary): 11101
Choose Generator:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 4
Generator: 10001000000100001
Remainder: 1100001110011100
Code word: 111011100001110011100
```

RECEIVER:

```
codeword = input("Enter received code word (dataword + CRC): ").strip()
print("Choose Generator Used for Verification:")
print("1. User Generator\n2. CRC-12\n3. CRC-16\n4. CRITT")
choice = input("choice (1-4): ").strip()
if choice == '1':
   generator = input("Enter generator polynomial used (binary): ").strip()
elif choice == '2':
   generator = "110000000111"
elif choice == '3':
   generator = "11000000000000101"
elif choice == '4':
   generator = "10001000000100001"
else:
   raise SystemExit("Invalid choice.")
r = len(generator) - 1
cw = [int(b) for b in codeword]
g = [int(b) for b in generator]
for p in range(len(codeword) - r):
   if cw[p] == 1:
      for j in range(len(g)):
         cw[p + j] ^= g[j]
final_rem = cw[-r:] if r > 0 else []
print("Verification remainder (should be all zeros):", ''.join(str(b) for b in final_rem))
if all(b == 0 for b in final_rem):
   print("No error detected!")
else:
   print("Error detected in received data!")
```

output :

```
C:\Users\stain\OneDrive\Desktop\CN RECORD>python -u "c:\Users\stain\OneDrive\Desktop\CN RECORD\exp4b.py"
Enter received code word (dataword + CRC): 11101000000001001110
Choose Generator Used for Verification:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 3
Verification remainder (should be all zeros): 0000000000000000
No error detected!

C:\Users\stain\OneDrive\Desktop\CN RECORD>python -u "c:\Users\stain\OneDrive\Desktop\CN RECORD\exp4b.py"
Enter received code word (dataword + CRC): 11101110001110011100
Choose Generator Used for Verification:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 4
Verification remainder (should be all zeros): 0000001001010010
```

```
Enter received code word (dataword + CRC): 111011111
Choose Generator Used for Verification:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 1
Enter generator polynomial used (binary): 10111
Verification remainder (should be all zeros): 0000
No error detected!

C:\Users\stain\OneDrive\Desktop\CN RECORD>python -u "c:\Users\stain\OneDrive\Desktop\CN RECORD\exp4b.py"
Enter received code word (dataword + CRC): 11101000011010010
Choose Generator Used for Verification:
1. User Generator
2. CRC-12
3. CRC-16
4. CRITT
choice (1-4): 2
Verification remainder (should be all zeros): 00000010110
Error detected in received data!
```

EXPERIMENT:6

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

Go Back N:

Go Back N is a sliding window ARQ protocol used for error and flow control in Data Link Layer. Sender can transmit multiple frames (up to a window size) without waiting for acknowledgement. if an error or loss occur in a error the receiver discards that frame and all subsequent frames.

Program:

```
import random


class Receiver:
    def __init__(self, Sender):
        self.Sender = Sender


    def receive(self, frame_no):
        print(f"Receiver got frame {frame_no} [Seq={frame_no}] without error")
        print(f"Receiver replies with ACK {frame_no}")


class GoBackN:
    def __init__(self, total_frames, window_size, frame_loss_p=0.25, ack_loss_p=1/3, seed=None):
        self.total_frames = total_frames
        self.window_size = window_size
        self.sf = 0
        self.sn = 0
        self.receiver = Receiver(self)
        self.rng = random.Random(seed)
        self.frame_loss_p = frame_loss_p
        self.ack_loss_p = ack_loss_p


    def frame_ok(self):
        return self.rng.random() > self.frame_loss_p


    def ack_ok(self):
```

```
        return self.rng.random() > self.ack_loss_p


    def can_send_more(self):
        return self.sn < self.total_frames and ((self.sf + self.window_size) > self.sn)


    def start(self):
        print(f"\nTransmission initiated with {self.total_frames} frames (window
size={self.window_size})")
        while self.sf < self.total_frames:
            window_end = min(self.sf + self.window_size, self.total_frames)
            print(f"\nSender window active: frames {self.sf} to {window_end - 1}")
            while self.sn < window_end:
                print(f"Sender is transmitting frame {self.sn} [Seq={self.sn}]")
                self.sn += 1
                ack_expected = self.sf
                timeout = False
                while ack_expected < window_end:
                    frame_ok = self.frame_ok()
                    if frame_ok:
                        self.receiver.receive(ack_expected)
                        ack_ok = self.ack_ok()
                        if ack_ok:
                            print(f"Sender received ACK for frame {ack_expected}")
                            ack_expected += 1
                            self.sf += 1
                        else:
                            print(f"ACK for frame {ack_expected} got lost (timeout)")
                            self.sn = self.sf
                            timeout = True
                            break
                    else:
                        print(f"Frame {ack_expected} lost during transmission")
                        self.sn = self.sf
```

```
                timeout = True
                break
            if not timeout and self.sf < self.total_frames:
                print(f"Sliding window moved: sf={self.sf}, sn={self.sn}")
        print("\nAll frames delivered successfully using Go-Back-N ARQ!")


if __name__ == "__main__":
    total = int(input("Enter no. of frames to send: "))
    win = int(input("Enter window size: "))
    gbn = GoBackN(total, win, frame_loss_p=0.25, ack_loss_p=1/3, seed=None)
    gbn.start()
```

output :

```
Enter no. of frames to send: 7
Enter window size: 4

Transmission initiated with 7 frames (window size=4)

Sender window active: frames 0 to 3
Sender is transmitting frame 0 [Seq=0]
Frame 0 lost during transmission
Sender is transmitting frame 0 [Seq=0]
Frame 0 lost during transmission
Sender is transmitting frame 0 [Seq=0]
Frame 0 lost during transmission
Sender is transmitting frame 0 [Seq=0]
Receiver got frame 0 [Seq=0] without error
Receiver replies with ACK 0
Sender received ACK for frame 0
Receiver got frame 1 [Seq=1] without error
Receiver replies with ACK 1
ACK for frame 1 got lost (timeout)
Sender is transmitting frame 1 [Seq=1]
Receiver got frame 1 [Seq=1] without error
Receiver replies with ACK 1
ACK for frame 1 got lost (timeout)
Sender is transmitting frame 1 [Seq=1]
Receiver got frame 1 [Seq=1] without error
Receiver replies with ACK 1
Sender received ACK for frame 1
Receiver got frame 2 [Seq=2] without error
Receiver replies with ACK 2
Sender received ACK for frame 2
Receiver got frame 3 [Seq=3] without error
Receiver replies with ACK 3
Sender received ACK for frame 3
Sliding window moved: sf=4, sn=2
Sender is transmitting frame 2 [Seq=2]
Sliding window moved: sf=4, sn=3
Sender is transmitting frame 3 [Seq=3]
Sliding window moved: sf=4, sn=4
```

```
Sender window active: frames 4 to 6
Sender is transmitting frame 4 [Seq=4]
Receiver got frame 4 [Seq=4] without error
Receiver replies with ACK 4
Sender received ACK for frame 4
Frame 5 lost during transmission
Sender is transmitting frame 5 [Seq=5]
Receiver got frame 5 [Seq=5] without error
Receiver replies with ACK 5
Sender received ACK for frame 5
Receiver got frame 6 [Seq=6] without error
Receiver replies with ACK 6
Sender received ACK for frame 6
Sender is transmitting frame 6 [Seq=6]

All frames delivered successfully using Go-Back-N ARQ!
```

EXPERIMENT-7:

Develop a Program to implement Sliding window protocol for Selective repeat.

Selective repeat is a sliding window protocol where only the erroneous or lost frames are retransmitted, Instead of resending all frames after an error.

Program:

```
import random
from collections import OrderedDict


class ReceiverSide:
    def __init__(self, Sender):
        self.Sender = Sender


    def receive_frame(self, frame_num: int):
        print(f"Receiver received frame {frame_num} correctly")
        print(f"Receiver sending ack {frame_num}")


class SelectiveRepeatProtocol:
    def __init__(self, total_packets: int, window_limit: int, seed=None):
        self.total_packets = total_packets
        self.window_limit = window_limit
        self.base = 0
        self.next_seq = 0
        self.sent_frames = OrderedDict((i, False) for i in range(total_packets))
        self.receiver = ReceiverSide(self)
```

```
        self.random = random.Random(seed)


    def print_status(self):
        pending_keys = list(self.sent_frames.keys())
        print(f"\n[Window update] Base={self.base}, Next Seq={self.next_seq}, window
size={self.window_limit}")


    def begin_transmission(self):
        print("\n--- Selective Repeat ARQ Demonstration ---")
        print(f"Total frames: {self.total_packets}, window size: {self.window_limit}\n")
        while self.base < self.total_packets:
            while self.next_seq < self.base + self.window_limit and self.next_seq < self.total_packets:
                print(f"Transmitter: Sending frame #{self.next_seq}")
                self.sent_frames[self.next_seq] = False
                self.next_seq += 1
            self.print_status()
            for frame_id in list(self.sent_frames.keys()):
                if not self.sent_frames[frame_id]:
                    frame_delivered = self.random.randrange(4) != 0  # 3/4 chance delivered
                    if frame_delivered:
                        self.receiver.receive_frame(frame_id)
                        ack_delivered = self.random.randrange(4) != 0
                        if ack_delivered:
                            print(f"Transmitter received ack {frame_id}")
                            self.sent_frames[frame_id] = True
                        else:
                            print(f"Ack {frame_id} lost, retransmitting...")
                            print(f"Resending frame {frame_id}...")
                    else:
                        print(f"Frame {frame_id} lost in transit, resending...")

            while self.base < self.total_packets and self.sent_frames[self.base]:
                self.base += 1
```

```
def main():
    try:
        total = int(input("Enter total number of frames to send: ").strip())
        win = int(input("Enter window size: ").strip())
    except Exception:
        print("Error: Invalid input. Please enter integers for total frames and window size.")
        return

    sr = SelectiveRepeatProtocol(total, win)
    sr.begin_transmission()

if __name__ == "__main__":
    main()
```

output :

```
Enter total number of frames to send: 7
Enter window size: 4

--- Selective Repeat ARQ Demonstration ---
Total frames: 7, window size: 4

Transmitter: Sending frame #0
Transmitter: Sending frame #1
Transmitter: Sending frame #2
Transmitter: Sending frame #3

[Window update] Base=0, Next Seq=4, window size=4
Receiver received frame 0 correctly
Receiver sending ack 0
Transmitter received ack 0
Receiver received frame 1 correctly
Receiver sending ack 1
Transmitter received ack 1
Receiver received frame 2 correctly
Receiver sending ack 2
Transmitter received ack 2
Receiver received frame 3 correctly
Receiver sending ack 3
Ack 3 lost, retransmitting...
Resending frame 3...
Frame 4 lost in transit, resending...
Receiver received frame 5 correctly
Receiver sending ack 5
Ack 5 lost, retransmitting...
Resending frame 5...
Receiver received frame 6 correctly
Receiver sending ack 6
Transmitter received ack 6
Transmitter: Sending frame #4
Transmitter: Sending frame #5
Transmitter: Sending frame #6
```

```
[Window update] Base=3, Next Seq=7, window size=4
Receiver received frame 3 correctly
Receiver sending ack 3
Transmitter received ack 3
Frame 4 lost in transit, resending...
Receiver received frame 5 correctly
Receiver sending ack 5
Transmitter received ack 5
Receiver received frame 6 correctly
Receiver sending ack 6
Ack 6 lost, retransmitting...
Resending frame 6...

[Window update] Base=4, Next Seq=7, window size=4
Frame 4 lost in transit, resending...
Receiver received frame 6 correctly
Receiver sending ack 6
Transmitter received ack 6

[Window update] Base=4, Next Seq=7, window size=4
Receiver received frame 4 correctly
Receiver sending ack 4
Transmitter received ack 4

C:\Users\stain\OneDrive\Desktop\CN RECORD>
```

EXPERIMENT 8:

Develop a Program to implement Stop and Wait Protocol.

It is a flow control method in the data link layer where the sender transmits new frame and waits for its acknowledgement before sending the next frame. In duplex mode, the sender and receiver can send and acknowledge frames simultaneously, but only one frame per direction is in transmit at any time.

Program:

SENDER:

```
import socket
import time


def send_packet(packet, server_address, max_retries=3, timeout=2, retry_delay=1):
    sender_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sender_socket.settimeout(timeout)


    attempt = 0
    while attempt < max_retries:
        try:
            print(f"[Sender] Sending: {packet} (Attempt {attempt + 1})")
            sender_socket.sendto(packet.encode(), server_address)
```

```
            ack, _ = sender_socket.recvfrom(1024)
            ack_msg = ack.decode()
            if ack_msg == f"ACK-{packet}":
                print(f"[Sender] Received ACK for: {packet}")
                sender_socket.close()
                return True

        except socket.timeout:
            attempt += 1
            print(f"[Sender] Timeout! No ACK received for {packet}. Retrying in {retry_delay}s...")

            time.sleep(retry_delay)

    print(f"[Sender] Failed to receive ACK after {max_retries} attempts for: {packet}")
    sender_socket.close()
    return False

if __name__ == "__main__":
    server_address = ('localhost', 25253)
    packets = ["Packet1", "Packet2", "Packet3", "Packet4", "Packet5", "Packet6"]  # 6 packets now

    for packet in packets:
        success = send_packet(packet, server_address)
        if not success:
            print(f"[Sender] Giving up on {packet}, moving to next packet.")

    # After all packets, send QUIT to tell receiver to stop
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.sendto("QUIT".encode(), server_address)

    print("[Sender] All done. Sender socket closed.")
```

```
    input("enter to stop")
```

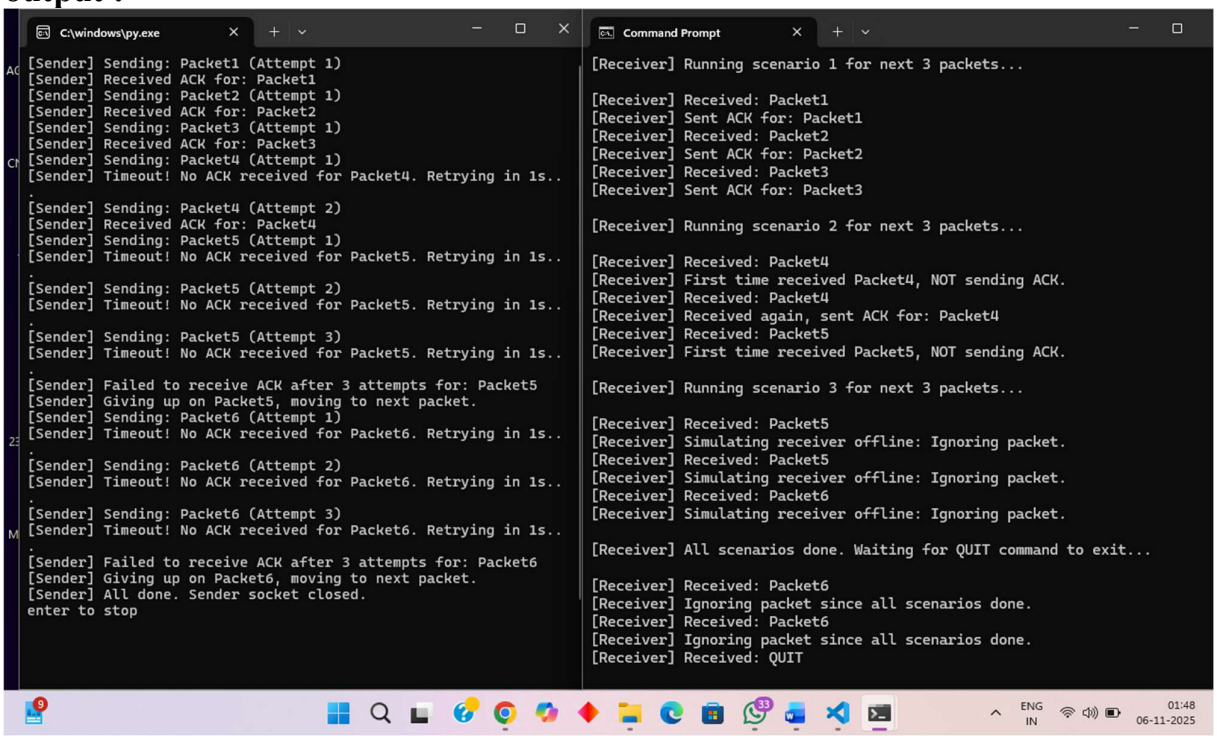
RECEIVER

```
import socket
import time

def run_receiver():
    receiver_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    receiver_socket.bind(('localhost', 25253))

    print("[Receiver] Ready to receive data...\n")

    received_once = set()  # for scenario 2 tracking

    scenario_steps = [
        ("1", 3),
        ("2", 3),
        ("3", 3),
    ]

    for scenario, packet_count in scenario_steps:
        print(f"\n[Receiver] Running scenario {scenario} for next {packet_count} packets...\n")

        count = 0
        while count < packet_count:
            data, sender_address = receiver_socket.recvfrom(1024)
            packet = data.decode().strip()
            print(f"[Receiver] Received: {packet}")

            if packet == "QUIT":
                print("[Receiver] Shutdown command received. Exiting...")
```

```
            receiver_socket.close()
            print("[Receiver] Socket closed.")
            return

        if scenario == "1":
            ack = f"ACK-{packet}"
            receiver_socket.sendto(ack.encode(), sender_address)
            print(f"[Receiver] Sent ACK for: {packet}")

        elif scenario == "2":
            if packet not in received_once:
                print(f"[Receiver] First time received {packet}, NOT sending ACK.")
                received_once.add(packet)
            else:
                ack = f"ACK-{packet}"
                receiver_socket.sendto(ack.encode(), sender_address)
                print(f"[Receiver] Received again, sent ACK for: {packet}")

        elif scenario == "3":
            # Receiver offline: ignore packets
            print("[Receiver] Simulating receiver offline: Ignoring packet.")

        count += 1


print("\n[Receiver] All scenarios done. Waiting for QUIT command to exit...\n")
while True:
    data, sender_address = receiver_socket.recvfrom(1024)
    packet = data.decode().strip()
    print(f"[Receiver] Received: {packet}")

    if packet == "QUIT":
```

```
            print("[Receiver] Shutdown command received. Exiting...")
            break
        else:
            print("[Receiver] Ignoring packet since all scenarios done.")

    receiver_socket.close()
    print("[Receiver] Socket closed.")
if __name__ == "__main__":
    run_receiver()
```

output :



EXPERIMENT 9:

Develop a program for congestion control using leaky bucket algorithm

It is a congestion control technique that regulates data transmission rate in network. Data packets are added to a bucket and sent out in a fixed rate.

Program:

import threading

import time

```
def leaky_bucket():
    max_size = int(input("Enter max bucket size: "))
    output_rate = int(input("Enter outflow rate (bytes per second): "))
    bucket_size = [0]

    def empty_bucket():
        while True:
            if bucket_size[0] <= 0:
                print("Releasing 0 bytes (bucket empty).")
            else:
                released = min(bucket_size[0], output_rate)
                bucket_size[0] -= released
                print(f"Releasing {released} bytes. Remaining: {bucket_size[0]}")
            time.sleep(2)

    empty_thread = threading.Thread(target=empty_bucket, daemon=True)
    empty_thread.start()

    while True:
        print()
        data = int(input("Enter data size (-1 to stop): "))
        print()
        if data == -1:
            break
        print(f"Incoming data size: {data}")
        if bucket_size[0] + data > max_size:
            overflow = bucket_size[0] + data - max_size
            bucket_size[0] = max_size
            print(f"Data added. Current bucket size: {bucket_size[0]}")
            print(f"Overflow! {overflow} bytes dropped.")
        else:
            bucket_size[0] += data
```

```
            print(f"Data added. Current bucket size: {bucket_size[0]}")
    while bucket_size[0] > 0:
        time.sleep(2)
    for _ in range(5):
        print("Releasing 0 bytes (bucket empty).")
        time.sleep(2)
    print("All data released. Bucket is empty.")
if __name__ == "__main__":
    leaky_bucket()
```

output :

```
Enter max bucket size: 10
Enter outflow rate (bytes per second): 3
Releasing 0 bytes (bucket empty).

Enter data size (-1 to stop): 5

Incoming data size: 5
Data added. Current bucket size: 5

Enter data size (-1 to stop): 4Releasing 3 bytes. Remaining: 2


Incoming data size: 4
Data added. Current bucket size: 6

Enter data size (-1 to stop): 6

Incoming data size: 6
Data added. Current bucket size: 10
Overflow! 2 bytes dropped.

Enter data size (-1 to stop): Releasing 3 bytes. Remaining: 7
2

Incoming data size: 2
Data added. Current bucket size: 9

Enter data size (-1 to stop): 1

Incoming data size: 1
Data added. Current bucket size: 10

Enter data size (-1 to stop): Releasing 3 bytes. Remaining: 7
Releasing 3 bytes. Remaining: 4
Releasing 3 bytes. Remaining: 1
-1

Releasing 1 bytes. Remaining: 0
Releasing 0 bytes (bucket empty).
Releasing 0 bytes (bucket empty).
Releasing 0 bytes (bucket empty).
Releasing 0 bytes (bucket empty).
```

```
Releasing 0 bytes (bucket empty).
Releasing 0 bytes (bucket empty).
Releasing 0 bytes (bucket empty).
Releasing 0 bytes (bucket empty).
All data released. Bucket is empty.
```

EXPERIMENT 10:

Develop a Program to implement Dijkstra's algorithm to compute the Shortest path through a graph.

It is a shortest path routing algorithm used find minimum cost path from a source node to all nodes in a network.

Program:

```
INF = 9999

def dijkstra(adj_matrix, src):
    n = len(adj_matrix)
    dist = [INF] * n
    pred = [-1] * n
    visited = [False] * n
    dist[src] = 0


    for _ in range(n):
        u = -1
        min_dist = INF
        for i in range(n):
            if not visited[i] and dist[i] < min_dist:
                min_dist = dist[i]
                u = i
        if u == -1:
            break
        visited[u] = True
        for v in range(n):
            if adj_matrix[u][v] != 0 and not visited[v]:
                if dist[u] + adj_matrix[u][v] < dist[v]:
                    dist[v] = dist[u] + adj_matrix[u][v]
                    pred[v] = u
```

```
    return dist, pred

def print_path(pred, dest):
    path = []
    cur = dest
    while cur != -1:
        path.insert(0, cur)
        cur = pred[cur]
    return path

def main():
    n = int(input("Enter the number of vertices: "))
    adj_matrix = [[0 for _ in range(n)] for _ in range(n)]

    print("Enter edges as (source destination weight):")
    print("To quit, enter -1 -1 -1\n")

    while True:
        try:
            u, v, w = map(int, input("Enter edge (src dest weight): ").split())
        except ValueError:
            print("Please enter three integers like: 0 2 5")
            continue
        if u == -1 and v == -1 and w == -1:
            break
        if 0 <= u < n and 0 <= v < n:
            adj_matrix[u][v] = w
            adj_matrix[v][u] = w
        else:
            print("Invalid vertices; must be between 0 and", n-1)
    print("\nAdjacency matrix:")
    for i in range(n):
```

```
        print(" ".join(str(adj_matrix[i][j]).rjust(3) for j in range(n)))

    src = int(input("\nEnter source vertex: "))
    dist, pred = dijkstra(adj_matrix, src)

    print("\n--- Shortest paths and costs ---")
    for i in range(n):
        if i == src:
            continue
        if dist[i] >= INF:
            print(f"No path exists from vertex {src} to vertex {i}")
        else:
            path = print_path(pred, i)
            print(f"Shortest path from vertex {src} to vertex {i}: {' -> '.join(map(str, path))}")
            print(f"Cost from vertex {src} to vertex {i}: {dist[i]}")

if __name__ == "__main__":
    main()
```

output :

```
Enter the number of vertices: 5
Enter edges as (source destination weight):
To quit, enter -1 -1 -1

Enter edge (src dest weight): 1 2 3
Enter edge (src dest weight): 0 1
Please enter three integers like: 0 2 5
Enter edge (src dest weight): 0 2 5
Enter edge (src dest weight): 0 4 5
Enter edge (src dest weight): 2 3 1
Enter edge (src dest weight): -1 -1 -1

Adjacency matrix:
  0   0   5   0   5
  0   0   3   0   0
  5   3   0   1   0
  0   0   1   0   0
  5   0   0   0   0

Enter source vertex: 0

--- Shortest paths and costs ---
Shortest path from vertex 0 to vertex 1: 0 -> 2 -> 1
Cost from vertex 0 to vertex 1: 8
Shortest path from vertex 0 to vertex 2: 0 -> 2
Cost from vertex 0 to vertex 2: 5
Shortest path from vertex 0 to vertex 3: 0 -> 2 -> 3
Cost from vertex 0 to vertex 3: 6
Shortest path from vertex 0 to vertex 4: 0 -> 4
Cost from vertex 0 to vertex 4: 5
```

EXPERIMENT 11:

Develop 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:

```
def print_routing_tables(routing_tables, node_names):
    print("\nFinal Routing Tables:")
    for i in range(len(node_names)):
        print(f"\nRouter {node_names[i]}")
        print(f"{'Destination':<15}{'Next Hop':<12}{'Distance':<10}")
        for j in range(len(node_names)):
            next_hop = routing_tables[i][j][1]
            distance = routing_tables[i][j][0]
            print(f"{node_names[j]:<15}{node_names[next_hop] if next_hop != -1 else '-
':<12}{distance:<10}")


def distance_vector_routing(n, cost_matrix):
    routing_tables = [
        [[cost_matrix[i][j], j if cost_matrix[i][j] != 999 else -1] for j in range(n)]
        for i in range(n)
    ]
    updated = True
    while updated:
        updated = False
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if cost_matrix[i][k] == 999 or routing_tables[k][j][0] == 999:
                        continue
                    new_dist = cost_matrix[i][k] + routing_tables[k][j][0]
                    if new_dist < routing_tables[i][j][0]:
                        routing_tables[i][j][0] = new_dist
```

```
                routing_tables[i][j][1] = k
                updated = True
    return routing_tables

def main():
    INF = 999
    node_names = []
    n = int(input("Enter number of nodes: "))
    print("Enter node names:")
    node_names = input().split()

    print("Enter the cost matrix (use 999 for infinity):")
    cost_matrix = []
    for i in range(n):
        row = list(map(int, input(f"Row for {node_names[i]}: ").split()))
        cost_matrix.append(row)

    routing_tables = distance_vector_routing(n, cost_matrix)
    print_routing_tables(routing_tables, node_names)

if __name__ == "__main__":
    main()
```

output :

```
Enter number of nodes: 8
Enter node names:
A B C D E F G H
Enter the cost matrix (use 999 for infinity):
Row for A: 0 4 3 999 999 999 999 999
Row for B: 4 0 999 999 52 999 999 999
Row for C: 3 999 0 999 999 4 999 999
Row for D: 999 5 999 0 999 999 3 999
Row for E: 999 2 999 999 0 6 3 999
Row for F: 999 999 4 999 6 0 999 2
Row for G: 999 999 999 3 3 999 0 1
Row for H: 999 999 999 999 999 2 1 0

Final Routing Tables:

Router A
Destination    Next Hop    Distance
A              A           0
B              B           4
C              C           3
D              C           13
E              C           13
F              C           7
G              C           10
H              C           9

Router B
Destination    Next Hop    Distance
A              A           4
B              B           0
C              A           7
D              A           17
E              A           17
F              A           11
G              A           14
H              A           13
```

```
Router C
Destination    Next Hop    Distance
A              A           3
B              A           7
C              C           0
D              F           10
E              F           10
F              F           4
G              F           7
H              F           6

Router D
Destination    Next Hop    Distance
A              B           9
B              B           5
C              G           10
D              D           0
E              G           6
F              G           6
G              G           3
H              G           4

Router E
Destination    Next Hop    Distance
A              B           6
B              B           2
C              B           9
D              G           6
E              E           0
F              F           6
G              G           3
H              G           4
```

```
Router F
Destination    Next Hop    Distance
A              C           7
B              E           8
C              C           4
D              H           6
E              E           6
F              F           0
G              H           3
H              H           2

Router G
Destination    Next Hop    Distance
A              E           9
B              E           5
C              H           7
D              D           3
E              E           3
F              H           3
G              G           0
H              H           1

Router H
Destination    Next Hop    Distance
A              F           9
B              G           6
C              F           6
D              G           4
E              G           4
F              F           2
G              G           1
H              H           0
```

EXPERIMENT 12:

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

A Broadcast tree is a spanning tree used in computer networks to ensure that broadcast packets are delivered efficiently to all nodes in a subnet without creating loops or redundant transmission. Often implemented using algorithm like STP (Spanning tree protocols in LAN's)

Program:

```
import heapq


def show_path(v, prev):
    path = []
    while v != -1:
        path.append(v)
        v = prev[v]
    path.reverse()
    print(" -> ".join(map(str, path)))


def broadcast_tree():
    n = int(input("How many vertices? "))
    m = int(input("How many edges? "))
    adj = [[] for _ in range(n)]
    print("Enter edges as: from to weight")
    for _ in range(m):
        a, b, w = map(int, input().split())
        adj[a].append((b, w))
        adj[b].append((a, w))


    src = int(input("Pick starting vertex: "))
    dist = [float('inf')] * n
    prev = [-1] * n
    dist[src] = 0
    pq = [(0, src)]
```

```
    while pq:
        d, u = heapq.heappop(pq)
        if d > dist[u]:
            continue
        for v, wt in adj[u]:
            if dist[u] + wt < dist[v]:
                dist[v] = dist[u] + wt
                prev[v] = u
                heapq.heappush(pq, (dist[v], v))

    print(f"\n=== Broadcast Tree (shortest paths) from source {src} ===")
    for v in range(n):
        if dist[v] == float('inf'):
            print(f"{src} to {v}: No path")
        else:
            print(f"{src} to {v}: cost={dist[v]}; path=", end=" ")
            show_path(v, prev)
    print("\nEdges in Broadcast Tree (no loops):")
    for v in range(n):
        if prev[v] != -1:
            for u, w in adj[prev[v]]:
                if u == v:
                    print(f"{prev[v]} -> {v} (cost {w})")
                    break

broadcast_tree()
```

E

output :

```
How many vertices? 8
How many edges? 10
Enter edges as: from to weight
0 1 3
0 2 6
1 2 2
1 3 1
2 4 4
3 4 5
3 5 3
4 6 2
5 6 1
6 7 7
Pick starting vertex: 0

=== Broadcast Tree (shortest paths) from source 0 ===
0 to 0: cost=0; path= 0
0 to 1: cost=3; path= 0 -> 1
0 to 2: cost=5; path= 0 -> 1 -> 2
0 to 3: cost=4; path= 0 -> 1 -> 3
0 to 4: cost=9; path= 0 -> 1 -> 3 -> 4
0 to 5: cost=7; path= 0 -> 1 -> 3 -> 5
0 to 6: cost=8; path= 0 -> 1 -> 3 -> 5 -> 6
0 to 7: cost=15; path= 0 -> 1 -> 3 -> 5 -> 6 -> 7

Edges in Broadcast Tree (no loops):
0 -> 1 (cost 3)
1 -> 2 (cost 2)
1 -> 3 (cost 1)
3 -> 4 (cost 5)
3 -> 5 (cost 3)
5 -> 6 (cost 1)
6 -> 7 (cost 7)
```