CRC:

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 mod2div(dividend, divisor):

pick = len(divisor)

tmp = dividend[0:pick]

while pick < len(dividend):

if tmp[0] == '1':

tmp = xor(divisor, tmp) + dividend[pick]

else:

tmp = xor('0' \* pick, tmp) + dividend[pick]

pick += 1

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

tmp = xor('0' \* pick, tmp)

return tmp

def encodeData(data, key):

l\_key = len(key)

appended\_data = data + '0' \* (l\_key - 1)

remainder = mod2div(appended\_data, key)

codeword = data + remainder

return codeword

def checkData(received\_data, key):

remainder = mod2div(received\_data, key)

if '1' in remainder:

print("Error detected in the received data.")

else:

print("No error detected in the received data.")

data = '11010011101100'

key = '1011'

print("Original Data: ", data)

codeword = encodeData(data, key)

print("Data after encoding (CRC Code): ", codeword)

print("\nAt Receiver's End:")

received\_data = codeword

checkData(received\_data, key)

print("\nSimulating error in transmission:")

received\_data\_with\_error = '11010011101101'

print("Received Data with Error: ", received\_data\_with\_error)

checkData(received\_data\_with\_error, key)

Checksum:

def check(b\_chunks):

checksum = 0

for chunk in b\_chunks:

checksum += int(chunk,2)

if checksum> 0xFF:

checksum = (checksum & 0xFF) + 1

return format(~checksum & 0xFF,'08b')

def validate(b\_chunks,checksum):

cal\_checksum = check(b\_chunks)

return cal\_checksum == checksum

print("Sender Side:")

n = int(input("Enter the number of binary chunks to send: "))

b\_chunks = []

for i in range(n):

b\_chunks.append(input("Enter a binary chunk: "))

checksum = check(b\_chunks)

print(f"Computed Checksum (Sender): {checksum}")

print("\nReceiver Side:")

b\_chunks\_r = []

for i in range(n):

b\_chunks\_r.append(input("Enter a binary chunk: "))

validated = validate(b\_chunks\_r,checksum)

if validated:

print("Checksum valid: No error detected.")

else:

print("Checksum invalid: Error detected.")

Dijkstras:

import heapq

def dijkstra(graph, start):

queue = []

distances = {node: float('infinity') for node in graph}

distances[start] = 0

heapq.heappush(queue, (0, start))

while queue:

current\_distance, current\_node = heapq.heappop(queue)

if current\_distance > distances[current\_node]:

continue

for neighbor, weight in graph[current\_node].items():

distance = current\_distance + weight

if distance < distances[neighbor]:

distances[neighbor] = distance

heapq.heappush(queue, (distance, neighbor))

return distances

def main():

graph = {

'A': {'B': 1, 'C': 4},

'B': {'A': 1, 'C': 2, 'D': 5},

'C': {'A': 4, 'B': 2, 'D': 1},

'D': {'B': 5, 'C': 1}

}

start\_node = 'A'

shortest\_distances = dijkstra(graph, start\_node)

print("Shortest distances from node", start\_node)

for node, distance in shortest\_distances.items():

print(f"Distance to {node}: {distance}")

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

main()

server

import socket

def start\_server(host='127.0.0.1', port=65432):

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

server\_socket.bind((host, port))

server\_socket.listen()

print(f'Server listening on {host}:{port}')

while True:

conn, addr = server\_socket.accept()

print(f'Connection from {addr}')

data = conn.recv(1024)

print(f'Received: {data.decode()}')

conn.sendall(b'Hello from server!')

conn.close()

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

start\_server()

Client

import socket

def start\_client(host='127.0.0.1', port=65432):

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

client\_socket.connect((host, port))

client\_socket.sendall(b'Hello from Client!')

data = client\_socket.recv(1024)

print(f'Received: {data.decode()}')

client\_socket.close()

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

start\_client()