Crc

*def* find\_remainder(*data\_stream*, *divisor*, *crc*):

    print("############ find\_remainder start #######################")

    data = *data\_stream* + *crc*

    print(*f*"data = {data}")

    print(*f*"len(data) - len(crc) {len(data) - len(*crc*)}")

    print(''.join(data))

    for i in range(len(data) - len(*crc*)):

        print(' '\*i + ''.join(data[i:i+len(*crc*)+1]))

        print(*f*"i = {i} data[i:i+len(crc)+1] = {data[i:i+len(*crc*)+1]}")

        if data[i] == '1':

            for j in range(len(*divisor*)):

                print(*f*"j = {j} str((int(data[{i + j}]) ^ int(divisor[{j}]))): {data[i + j]} ^ {*divisor*[j]} = {str((int(data[i + j]) ^ int(*divisor*[j])))}")

                data[i + j] = str((int(data[i + j]) ^ int(*divisor*[j])))

    print("############ find\_remainder end #######################")

    return ''.join(data[-len(*crc*):])

*def* main():

    # data\_stream = [bit for bit in input('Data stream: ')]

    data\_stream = ["1","0","1","0","0","0","1","0","1","1","1","1","0","0"]

    print(*f*"data stream {data\_stream}")

    # divisor = [bit for bit in input('Divisor: ')]

    divisor = ["1","1","1","0","1"]

    print(*f*"divisor {divisor}")

    crc = ['0'] \* (len(divisor) - 1)

    print(*f*"crc {crc} (len(divisor) - 1) {(len(divisor) - 1)}")

    remainder = find\_remainder(data\_stream, divisor, crc)

    print(*f*"remainder {remainder}")

    if int(remainder) == 0:

        print('No error as Remainder = 0')

    else:

        print('Error as Remainder = ' + str(int(remainder)))

        print('Correct code generated => ' + ''.join(data\_stream) + remainder)

main()

Byte-stuffing

*def* byte\_stuff(*msg*, *flag*, *esc*):

    print("############# byte\_stuff start ###############")

    encoded\_msg = []

    head = 0

    encoded\_msg.append(*flag*)

    print(*f*"encoded\_msg {encoded\_msg}")

    i=1

    while head < len(*msg*):

        print(*f*"^^^^^^^^^^^^^^ loop {i} start ^^^^^^^^^^^^^^^")

        if *msg*[head] == *flag* or *msg*[head] == *esc*:

            print(*f*"msg[head] == flag or msg[head] == esc : {*msg*[head]} == flag or {*msg*[head]} == esc ")

            encoded\_msg.append(*esc*)

        encoded\_msg.append(*msg*[head])

        print(*f*"encoded\_msg {encoded\_msg}")

        head += 1

        print(*f*"head {head}")

        print(*f*"^^^^^^^^^^^^^^ loop {i} end ^^^^^^^^^^^^^^^")

        i+=1

    encoded\_msg.append(*flag*)

    print("############# byte\_stuff end ###############")

    return ' '.join(encoded\_msg)

# flag, esc = input('Enter FLAG and ESC strings(bytes) - space separated: ').split(' ')

# msg = input("Enter the message strings(bytes) - space separated: ")

flag, esc = "FLAG" , "ESC"

msg = "A FLAG B A ESC ESC FLAG C B ESC"

encoded = byte\_stuff(msg.split(' '), flag, esc)

print(*f*"encoded {encoded}")

print('Before Byte stuffing: ' + msg)

print("After Byte stuffing: " + encoded)

Bit-stuffing

*def* check\_bs(*msg*):

    count = 0

    b\_msg = []

    head = 0

    while head < len(*msg*):

        b\_msg.append(*msg*[head])

        if *msg*[head] == '1':

            count += 1

        if *msg*[head] == '0':

            count = 0

        if count == 5:

            b\_msg.append('0')

            count = 0

        head += 1

    return ''.join(b\_msg)

msg = input("Enter Bit String: ")

# Flag Bits 01111110

encoded = check\_bs(msg)

print(encoded)

Bellman ford

*def* main():

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

    # m = int(input("Enter number of edges "))

    n = 4

    m=4

    nodes = ["A","B","C","D", "E","F", "G"]

    edges = {

        "AB" : 2,

        "AC" : 5,

        "BC" : 6,

        "BD" : 1,

        "BE" : 3,

        "CF" : 8,

        "DE" : 4,

        "EG" : 9,

        "FG" : 9,

    }

    dist = {

        "A" : 1000,

        "B" : 1000,

        "C" : 1000,

        "D" : 1000,

        "E" : 1000,

        "F" : 1000,

        "G" : 1000

    }

    # for i in range(m):

    #     s = input("Enter source node: ")

    #     d = input("Enter destination node: ")

    #     w = input("Enter weight : ")

    #     edges[s+d] = w

    #     if s not in nodes:

    #         nodes.append(s)

    #     if d not in dist.keys():

    #         dist[d] = 1000

    # src = input("Enter start node: ")

    src = "A"

    dist[src] = 0

    for i in range(n):

        for key,value in edges.items():

            u = key[0]

            v = key[1]

            w = value

            dist[v] = min(dist[v], w+dist[u])

    print(dist)

main()

Dikstra

*def* minDistance(*dist*, *sptSet*, *node*):

    minimum = 1000

    min\_index=0

    for i in *node*:

        if *sptSet*[i] == False and *dist*[i] <= minimum:

            minimum = *dist*[i]

            min\_index = i

    return min\_index

*def* main():

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

    # m = int(input("Enter number of edges "))

    n = 4

    m=4

    nodes = ["A","B","C","D", "E","F", "G"]

    edges = {

        "AB" : 2,

        "AC" : 5,

        "BC" : 6,

        "BD" : 1,

        "BE" : 3,

        "CF" : 8,

        "DE" : 4,

        "EG" : 9,

        "FG" : 7,

    }

    dist = {

        "A" : 1000,

        "B" : 1000,

        "C" : 1000,

        "D" : 1000,

        "E" : 1000,

        "F" : 1000,

        "G" : 1000

    }

    sptSet = {

        "A" : False,

        "B" : False,

        "C" : False,

        "D" : False,

        "E" : False,

        "F" : False,

        "G" : False

    }

    parent = {

        "A" : "A"

    }

    # for i in range(m):

    #     s = input("Enter source node: ")

    #     d = input("Enter destination node: ")

    #     w = input("Enter weight : ")

    #     edges[s+d] = w

    #     if s not in nodes:

    #         nodes.append(s)

    #     if d not in dist.keys():

    #         dist[d] = 1000

    # src = input("Enter start node: ")

    src = "A"

    dist[src] = 0

    for i in range(n+1):

        u = minDistance(dist,sptSet,nodes)

        sptSet[u] = True

        for j in nodes:

            if u+j in edges.keys():

                # if not sptSet[j] and edges[u+j] and dist[u] != 1000 and dist[u] + edges[u+j] < dist[j]:

                if edges[u+j] > 0 and sptSet[j] == False and dist[j] > dist[u] + edges[u+j]:

                    dist[j] = dist[u] + edges[u+j]

                    parent[j] = u

    print(sptSet)

    print(dist)

    print(parent)

    for i in nodes:

        print(i,*end*=" ")

        print(dist[i], *end*=" ")

        print(" Path: ", *end*=" ")

        parnode = parent[i]

        temp = []

        temp.append(i)

        while parnode != src:

            # print(f" <- {parnode} ",end =" ")

            temp.append(parnode)

            parnode = parent[parnode]

        temp.append("A")

        temp.reverse()

        print(temp)

main()

Tcp

Server

from http import client

import socket

if \_\_name\_\_ == '\_\_main\_\_':

    ip = "127.0.0.1"

    port = 1234

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

    server.bind((ip, port))

    server.listen(5)

    while True:

        client, address = server.accept()

        print(*f*"Connection Established - {address[0]} : {address[1]}")

        string = client.recv(1024)

        string = string.decode("utf-8")

        string = string.upper()

        client.send(bytes(string, "utf-8"))

        client.close()

Client

import socket

if \_\_name\_\_ == '\_\_main\_\_':

    ip = "127.0.0.1"

    port = 1234

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

    server.connect((ip, port))

    string = input("Enter string: ")

    server.send(bytes(string, "utf-8"))

    buffer = server.recv(1024)

    buffer = buffer.decode("utf-8")

    print(*f*"Server {buffer}")

Udp

Server

import socket

socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

mesg = "hello from client"

socket.sendto(mesg.endcode("utf-8"), ("127.0.0.1", "12345"))

data, addr = socket.recvfrom(1024)

print("Server says")

print(str(data))

socket.close()

Client

import socket

socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) #

socket.bind(("127.0.0.1",12345))

while True:

    data, address = socket.recvfrom(1024)

    print(*f*"client data {str(data)}")

    message = "hello froom server"

    socket.sendto(bytes(message, "utf-8"), address)