**Lab Program-1**

**Write a program for error detecting code using CRC-CCITT (16-bits).**

def xor1(a, b):

    x = ""

    # print(len(a),len(b))

    for i in range(1, len(a)):

        if a[i] == b[i]:

            x += "0"

        else:

            x += "1"

    return x

def modulo2(divident, divisor):

    divlen = len(divisor)

    temp = divident[0:divlen]

    # print(temp)

    while(divlen < len(divident)):

        if temp[0] == "1":

            temp = xor1(temp, divisor)+divident[divlen]

        else:

            temp = temp[1:divlen]+divident[divlen]

        # print(temp)

        divlen += 1

    # print(temp)

    if temp[0] == "1":

        temp = xor1(temp, divisor)

        # return "0"+temp

    # print(len(temp),)

    if len(temp) < len(divisor):

        return "0"+temp

    return temp

def encode(data, key):

    append = data+"0"\*(len(key))

    # print(code)

    rem = modulo2(append, key)

    print("remaindar="+rem)

    code = data+rem

    print("code="+code)

    # Checking the logic:

    rem = modulo2(code, key)

    print("Remaindar we get when we do not have error="+rem)

    code = code.replace("011", "101")

    rem = modulo2(code, key)

    print("Remaindar we get when we have error="+rem)

def polytobin(string):

    keys = []

    key = ""

    for i in string:

        if i == '+':

            keys.append(int(key[1:]))

            key = ""

            continue

        key += i

    if key != "":

        keys.append(0)

    bina = ""

    j = 0

    print(keys)

    for i in range(keys[0], -1, -1):

        if i == (keys[j]):

            bina += "1"

            j += 1

        else:

            bina += "0"

    print(bina)

    return bina

string = input("Enter the key polynomial:\n")

key = polytobin(string)

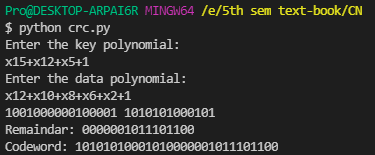
string = input("Enter the data polynomial:\n")

data = polytobin(string)

print(key, data)

encode(data, key)

Output:



**Lab Program-2**

**Write a program for distance vector algorithm to find suitable path for transmission.**

from collections import defaultdict

# from typing import DefaultDict

import math

class Graph:

    def \_\_init\_\_(self,vertices):

        self.vertices = vertices

        self.graph = defaultdict(list)

        # self.pc = defaultdict(list)

    # def vertexList(self):

        for i in range(self.vertices):

            self.graph[i]=[]

            for j in range(self.vertices):

                if i==j:

                    self.graph[i].append(0)

                else:

                    self.graph[i].append(1000)

    def addEdge(self,u,v,pc):

        self.graph[u][v]=self.graph[v][u]=pc

    def distVector(self):

        l=1

        while l<3:

            # For each vertex

            for i in range(0,self.vertices):

                # To find its neighbour

                for j in range(0,self.vertices):

                    # If it is a neighbour

                    if self.graph[i][j]!=1000:

                        x=self.graph[i][j]

                        # Loop through its table

                        for k in range(0,self.vertices):

                            self.graph[i][k]=min(self.graph[i][k],self.graph[j][k]+x)

            print(self.graph)

            l+=1

g=Graph(7)

g.addEdge(0,1,2)

g.addEdge(0,3,3)

g.addEdge(1,4,4)

g.addEdge(1,2,5)

g.addEdge(2,5,4)

g.addEdge(2,6,3)

g.addEdge(4,3,5)

g.addEdge(4,5,2)

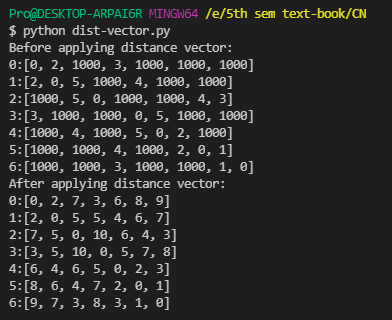
g.addEdge(5,6,1)

print(g.graph)

g.distVector()

print(g.graph)

**Output:**

****

**Lab Program-3**

**Implement Dijkstra’s algorithm to compute the shortest path for a given topology.**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self,vertices):

        self.vertices = vertices

        self.graph = defaultdict(list)

        self.dist = []

        for i in range(0,vertices):

            self.dist.append(1000)

    def addEdge(self,u,v,pc):

        self.graph[u].append([v,pc])

        self.graph[v].append([u,pc])

    def dijakstra(self,root):

        visited=[root]

        self.dist[root]=0

        while len(visited)<self.vertices:

            ver=None

            dist=1000

            for i in visited:

                for j in self.graph[i]:

                    if j[0] not in visited:

                        if self.dist[i]+j[1]<dist:

                            ver=j[0]

                            dist=self.dist[i]+j[1]

            visited.append(ver)

            self.dist[ver]=dist

        for i in visited:

            print(str(i)+"  Path cost:"+str(self.dist[i]))

g=Graph(7)

g.addEdge(0,1,2)

g.addEdge(0,3,3)

g.addEdge(1,4,4)

g.addEdge(1,2,5)

g.addEdge(2,5,4)

g.addEdge(2,6,3)

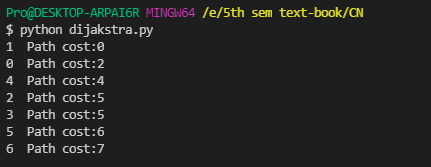
g.addEdge(4,3,5)

g.addEdge(4,5,2)

g.addEdge(5,6,1)

g.dijakstra(1)

**Output**

****

**Lab Program-4**

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

import random

import time

def main():

    SIZE=int(input("enter the number of packets: "))

    pk=[]

    for i in range(0,SIZE):

        pk.append(random.randint(1,100))

        print("packet["+str(i)+"]="+str(pk[i]))

    out=int(input("Enter the output rate: "))

    bucks=int(input("Enter the bucket size: "))

    curr=0

    for i in range(0,SIZE):

        if pk[i]+curr>bucks:

            if pk[i]>bucks:

                print("Packet is bigger than the bucket")

            else:

                print("Incoming packet cannot be contained in the bucket")

        else:

            curr+=pk[i]

            print("Incoming packet size: "+str(pk[i]))

            print("Packets left to transmit: "+str(curr))

            while curr>0:

                time.sleep(1)

                op=0

                if curr<=out:

                    op=curr

                    curr=0

                else:

                    op=out

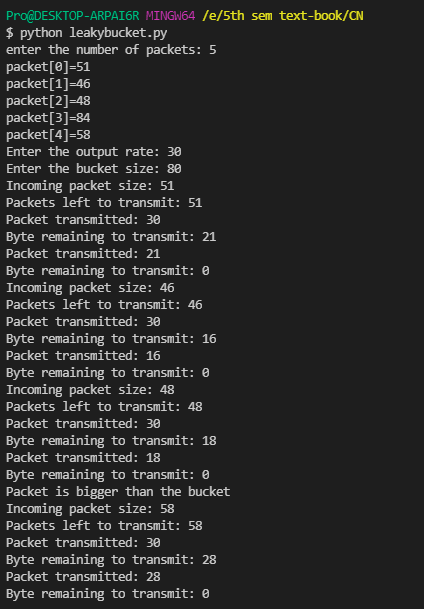
                    curr-=out

                print("Packet transmitted: "+str(op))

                print("Byte remaining to transmit: "+str(curr))

main()

**Output:**

****

**Lab Program-5**

**Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

**Server.py**

from socket import \*

serverName="127.0.0.1"

serverPort = 12000

serverSocket = socket(AF\_INET,SOCK\_STREAM)

serverSocket.bind((serverName,serverPort))

serverSocket.listen(1)

while 1:

    print ("The server is ready to receive")

    connectionSocket, addr = serverSocket.accept()

    # print(connectionSocket,addr)

    sentence = connectionSocket.recv(1024).decode()

    print(sentence)

    l=""

    try:

        file=open(sentence,"r")

        l=file.read(1024)

        file.close()

    except FileNotFoundError:

        l="File Not found"

    # print(str(file))

    connectionSocket.send(l.encode())

    print ("Sent contents of " + sentence)

    connectionSocket.close()

    exit(0)

**Client.py**

from socket import \*

serverName='127.0.0.1'

serverPort=12000

clientSocket=socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((serverName, serverPort))

sentence=input("Enter the filename: ")

clientSocket.send(sentence.encode())

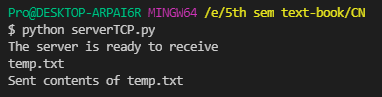
filecontents=clientSocket.recv(1024).decode()

print("Response from server:")

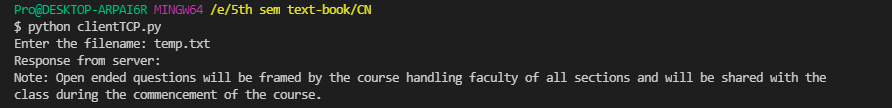
print(filecontents)

**Output:**

**Client.py output**

****

**Server.py output**

****

**Lab Program-6:**

**Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

**Server.py**

from socket import \*;

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

server.bind(("127.0.0.1",12000))

print("Server is ready")

while 1:

    sentence,addr=server.recvfrom(2048)

    sentence=sentence.decode("utf-8")

    file=open(sentence,'r')

    contents=file.read(2048)

    server.sendto(contents.encode('utf-8'),addr)

    print("Sent contents of "+sentence)

    file.close()

**Client.py**

from socket import \*;

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

sentence=input("Enter the file name")

client.sendto(sentence.encode("utf-8"),("127.0.0.1",12000))

contents,addr=client.recvfrom(2048)

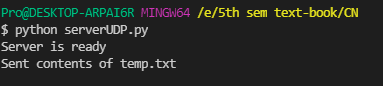
print("Reply from server")

print(contents.decode("utf-8"))

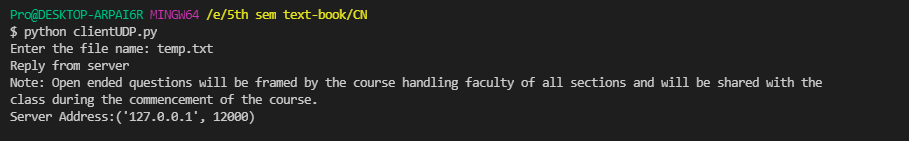
print("Server Address:"+str(addr))

**Output:**

**Server.py**

****

**Client.py**

****