ASSESSMENT-4

Network and communication lab

Rishabh Johri

19BDS0021

1. Write a menu driven code for Decimal, Binary a. To check the class, network id and host id of an IPv4 address. (Use function wherever necessary) b. To check whether given IP address is valid or not. c. To find first address, last address and number of addresses in the block.

AIM

Write a menu driven code for Decimal, Binary a. To check the class, network id and host id of an IPv4 address. b. To check whether given IP address is valid or not. c. To find first address, last address and number of addresses in the block.

ALGORITHM

a)

1)

For determining the class: The idea is to check first octet of IP address. As we know, for class A first octet will range from 1 – 126, for class B first octet will range from 128 – 191, for class C first octet will range from 192- 223, for class D first octet will range from 224 – 239, for class E first octet will range from 240 – 255.

2)

For determining the Network and Host ID: We know that Subnet Mask for Class A is 8, for Class B is 16 and for Class C is 24 whereas Class D and E is not divided into Network and Host ID.

b)

To validate the IP address we should follow these steps:

(i)Tokenize the string (IP address) using the dot “.” delimiter

(ii)If the sub strings are containing any non-numeric character, then return false

(iii)If the number in each token is not in range 0 to 255, then return false

(iv)If there are exactly three dots and four parts then it is a valid IP address

c)

1) number of addresses in block N = NOT(mask) +1

2) 1st address in block = (any address) AND (mask)

3) last address = (any address) OR (NOT(mask))

SOURCE CODE ( language : Python3)

def b2d(n):

    s = 0

    for i in range(len(n)):

        j = len(n)-i-1

        s = s + (2\*\*i)\*int(n[j])

    return s

def d2b(n):

    s = ''

    if n==0:

        s = '0'

    while n>0:

        s = s + str(n%2)

        n = n//2

    b = s[::-1]

    if len(b)==8:

        return b

    else:

        return (8-len(b))\*'0'+b

def dec2bin(IP):

    l = IP.split(".")

    l1 = []

    for i in l:

        l1.append(d2b(int(i)))

    return l1

def bNOT(B):

    B1 = ''

    for i in B:

        if i == '0':

            B1 = B1+'1'

        else:

            B1 = B1+'0'

    return B1

def AND(b1,b2):

    if b1=='1' and b2=='1':

        return '1'

    else:

        return '0'

def OR(b1,b2):

    if b1=='0' and b2=='0':

        return '0'

    else:

        return '1'

def bAND(B1,B2):

    ans = ''

    for i in range(len(B1)):

        ans = ans + AND(B1[i],B2[i])

    return ans

def bOR(B1,B2):

    ans = ''

    for i in range(len(B1)):

        ans = ans + OR(B1[i],B2[i])

    return ans

def a\_c():

    IP = input("Enter IP address in binary:")

    MASK = input("Mask in binary:")

    l1 = IP.split()

    l2 = MASK.split()

    mask = ''

    for m in l2:

        mask = mask + m

    print("Number of addressses:",b2d(bNOT(mask))+1)

    print("1st address:")

    first = ''

    for i in range(len(l1)):

        first = first + bAND(l1[i],l2[i])+"."

    print(first[0:len(first)-1])

    print("last address:")

    l22 = []

    for var in l2:

        l22.append(bNOT(var))

    last = ''

    for i in range(len(l1)):

        last = last + bOR(l1[i],l22[i])+"."

    print(last[0:len(last)-1])

def bin2dec(IP):

    l = IP.split()

    ip=""

    c = 1

    for i in l:

        if c!=4:

            ip = ip + str(b2d(i))+"."

        else:

            ip = ip + str(b2d(i))

        c=c+1

    return ip

def IPclass(IP\_d):

    l = IP\_d.split('.')

    a = int(l[0])

    if a in range(0,128):

        c = 'A'

    elif a in range(128,192):

        c = 'B'

    elif a in range(192,224):

        c = 'C'

    elif a in range(224,240):

        c = 'D'

    elif a in range(240,256):

        c = 'E'

    else :

        c = 'invalid'

    return c

def net\_host\_id(IP\_d):

    l = IP\_d.split('.')

    c  = IPclass(IP\_d)

    if c == 'A':

        print("Network id:",l[0])

        print("Host id:",l[1]+"."+l[2]+"."+l[3])

    elif c == 'B':

        print("Network id:",l[0]+"."+l[1])

        print("Host id:",l[2]+"."+l[3])

    elif c == 'C':

        print("Network id:",l[0]+"."+l[1]+"."+l[2])

        print("Host id:",l[3])

    else:

        print("In this Class, IP address is not divided into Network and Host ID")

def validate\_IP\_b():

    flag = 1

    IP = input("Enter IP address (eg: 11000000 10101010 11110000 00000000)")

    l = IP.split()

    if len(l)!=4:

        flag=0

    for var in l:

        if len(var)!=8:

            flag=0

            break

    for i in l:

        if set(i).union({'0','1'})!= {'0','1'}:

            flag = 0

            break

    if flag ==1:

        print("Valid")

    else:

        print("Invalid")

def validate\_IP\_d():

    IP = input("Enter IP addess(Example: 191.10.10.1)")

    l = IP.split(".")

    flag = 1

    if len(l) != 4:

        flag=0

    for var in l:

        if var.isnumeric()=="False":

            flag = 0

            break

    if flag==1:

        for v in l:

            if v[0]=='0':

                flag = 0

                break

        for i in l:

            if int(i) not in range(0,256):

                flag =0

                break

    if flag == 0:

        print("invalid")

    else:

        print("Valid")

def b\_c():

    IP = input("Enter IP address:")

    MASK = input("Mask:")

    l1 = dec2bin(IP)

    l2 = dec2bin(MASK)

    mask = ''

    for m in l2:

        mask = mask + m

    print("Number of addressses:",b2d(bNOT(mask))+1)

    print("1st address:")

    first = []

    for i in range(len(l1)):

        first.append(b2d(bAND(l1[i],l2[i])))

    f = ''

    for var in first:

        f = f + str(var) +"."

    print(f[0:len(f)-1])

    print("last address:")

    l22 = []

    for var in l2:

        l22.append(bNOT(var))

    last = []

    for i in range(len(l1)):

        last.append(b2d(bOR(l1[i],l22[i])))

    L = ''

    for var in last:

        L = L + str(var) +"."

    print(L[0:len(L)-1])

def menu():

    print("a. To check the class, network id and host id of an IPv4 address.\nb. To check whether given IP address is valid or not.\nc. To find first address, last address and number of addresses in the block.")

print("a)Binary\nb)Decimal")

c = input("Enter choice a or b: ")

if(c not in {'a','b'}):

    print("Invalid choice")

if c=='a':

    menu()

    choice = input("Enter a,b or c: ")

    if choice == "a":

        IP = input("Enter IP address(eg:11111111 10101010 00000000 00000000)")

        IP\_d = bin2dec(IP)

        address\_class = IPclass(IP\_d)

        print("Class :",address\_class)

        net\_host\_id(IP\_d)

    elif choice =="b":

        validate\_IP\_b()

    elif choice == "c":

        a\_c()

    else:

        print("invalid choice")

elif c=='b':

    menu()

    choice = input("Enter a,b or c: ")

    if choice == "a":

        IP = input("Enter IP address(eg: 111.56.45.78)")

        print("Class :",IPclass(IP))

        net\_host\_id(IP)

    elif choice =="b":

        validate\_IP\_d()

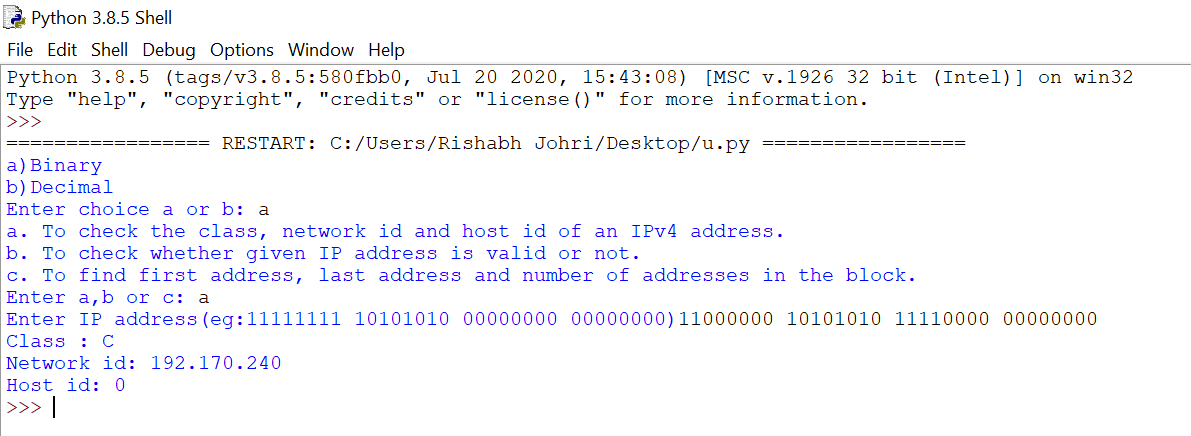
    elif choice == "c":

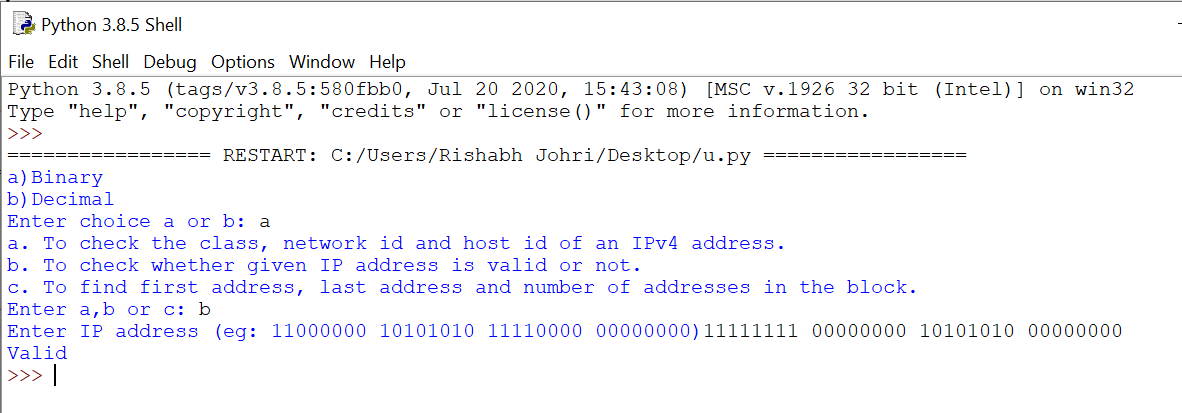
        b\_c()

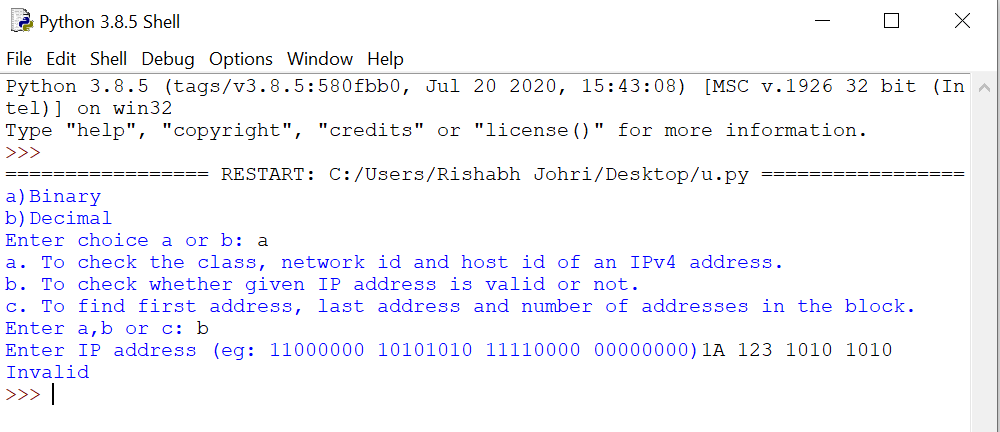
    else:

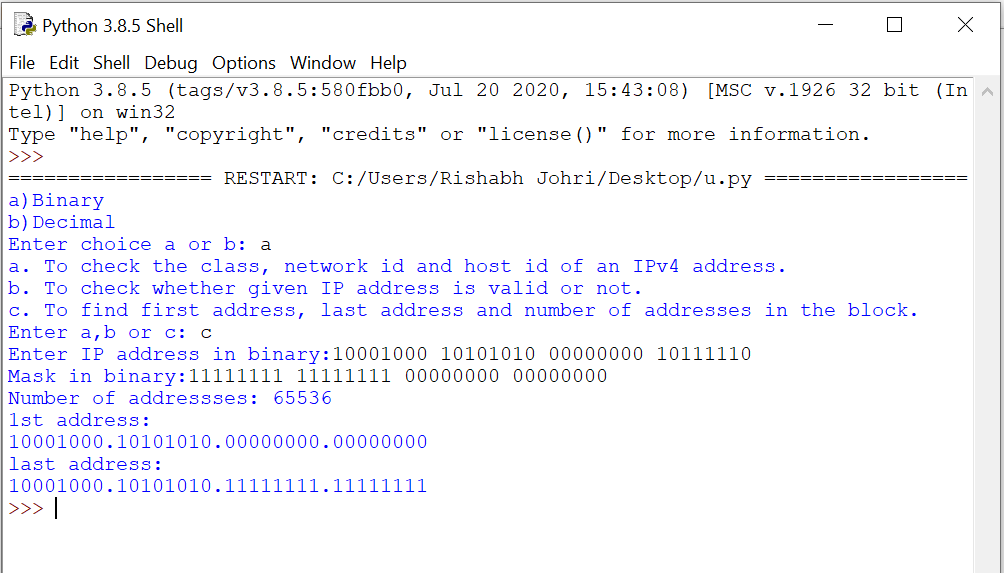
        print("invalid choice")

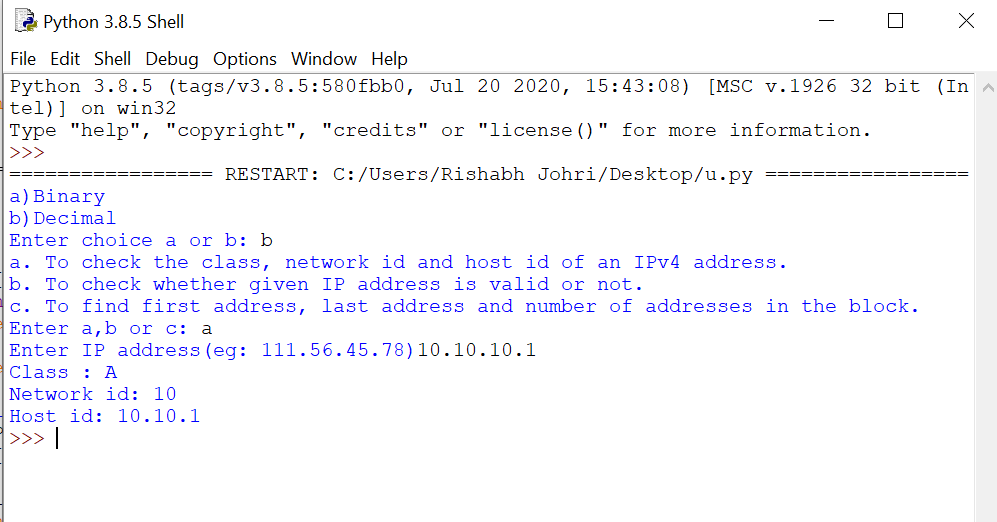
OUTPUT

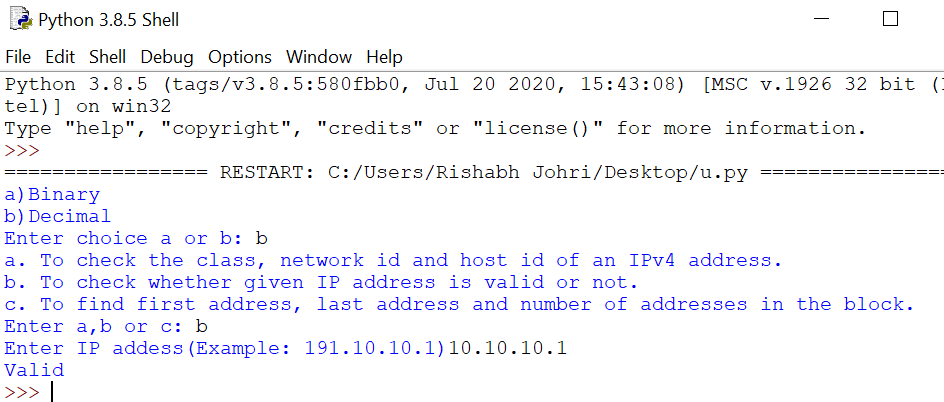


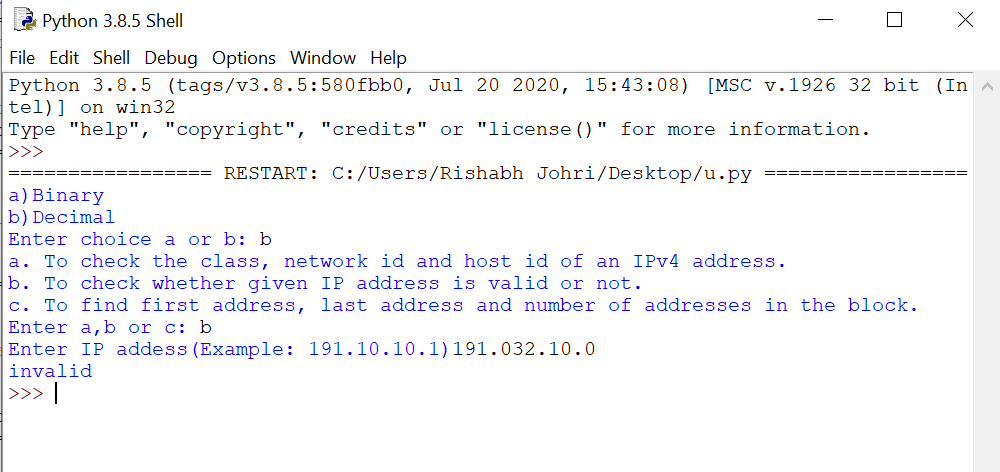


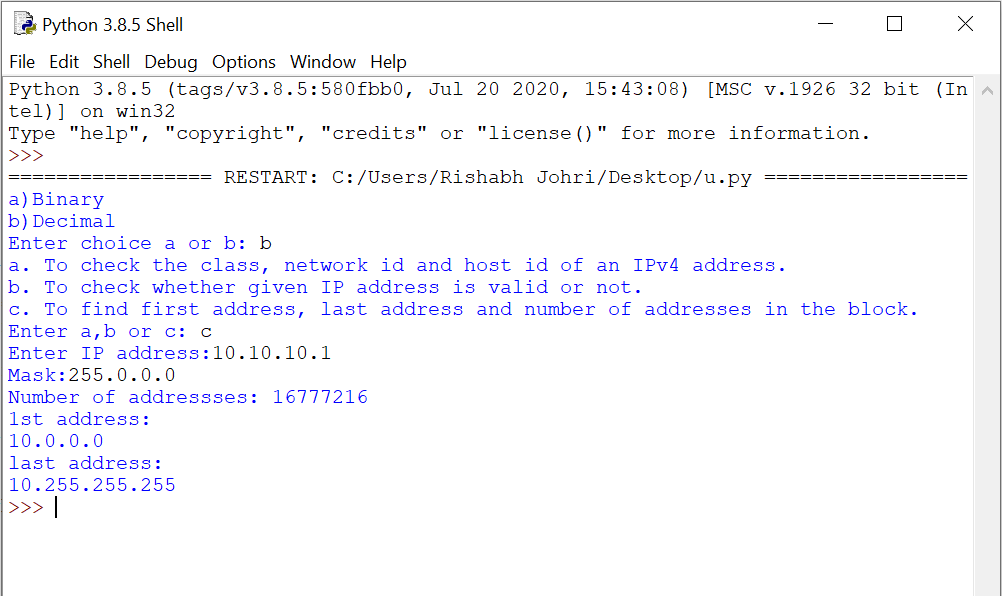












LINK TO CODE Repository

<https://github.com/rishabhjohri/NetworkCommunication_IP_Addressing>

2. Implement the following unicast routing algorithms using functions. a. Distance Vector Routing b. Link State Routing

AIM

Implement the following unicast routing algorithms using functions. a. Distance Vector Routing b. Link State Routing

ALGORITHM

a) Distance Vector Routing

At each node x,

Initialization

for all destinations y in N:

Dx(y) = c(x,y) // If y is not a neighbor then c(x,y) = ∞

for each neighbor w

Dw(y) = ? for all destination y in N.

for each neighbor w

send distance vector Dx = [ Dx(y) : y in N ] to w

loop

wait(until I receive any distance vector from some neighbor w)

for each y in N:

Dx(y) = minv{c(x,v)+Dv(y)}

If Dx(y) is changed for any destination y

Send distance vector Dx = [ Dx(y) : y in N ] to all neighbors

forever

b) Link State Routing

The three keys to understand the Link State Routing algorithm:

Knowledge about the neighborhood: Instead of sending its routing table, a router sends the information about its neighborhood only. A router broadcast its identities and cost of the directly attached links to other routers.

Flooding: Each router sends the information to every other router on the internetwork except its neighbors. This process is known as Flooding. Every router that receives the packet sends the copies to all its neighbors. Finally, each and every router receives a copy of the same information.

Information sharing: A router sends the information to every other router only when the change occurs in the information.

c( i , j): Link cost from node i to node j. If i and j nodes are not directly linked, then c(i , j) = ∞.

D(v): It defines the cost of the path from source code to destination v that has the least cost currently.

P(v): It defines the previous node (neighbor of v) along with current least cost path from source to v.

N: It is the total number of nodes available in the network.

**Initialization**

N = {A} // **A is a root node**.

for all nodes v

if v adjacent to A

then D(v) = c(A,v)

else D(v) = infinity

**loop**

find w not in N such that D(w) is a minimum.

Add w to N

Update D(v) for all v adjacent to w and not in N:

D(v) = min(D(v) , D(w) + c(w,v))

Until all nodes in N

SOURCE CODE

#include<stdio.h>

#include <string.h>

struct node

{

    unsigned dist[20];

    unsigned from[20];

}rt[10];

void a()

{

    int costmat[20][20];

    int nodes,i,j,k,count=0;

    printf("\nEnter the number of nodes : ");

    scanf("%d",&nodes);//Enter the nodes

    printf("\nEnter the cost matrix :\n");

    for(i=0;i<nodes;i++)

    {

        for(j=0;j<nodes;j++)

        {

            scanf("%d",&costmat[i][j]);

            costmat[i][i]=0;

            rt[i].dist[j]=costmat[i][j];//initialise the distance equal to cost matrix

            rt[i].from[j]=j;

        }

    }

        do

        {

            count=0;

            for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct distance from the node i to k using the cost matrix

            //and add the distance from k to node j

            for(j=0;j<nodes;j++)

            for(k=0;k<nodes;k++)

                if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])

                {//We calculate the minimum distance

                    rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];

                    rt[i].from[j]=k;

                    count++;

                }

        }while(count!=0);

        for(i=0;i<nodes;i++)

        {

            printf("\n\n For router %d\n",i+1);

            for(j=0;j<nodes;j++)

            {

                printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);

            }

        }

    printf("\n\n");

    getch();

}

void b()

{

int count,src\_router,i,j,k,w,v,min;

int cost\_matrix[100][100],dist[100],last[100];

int flag[100];

 printf("\n Enter the no of routers");

scanf("%d",&count);

printf("\n Enter the cost matrix values:");

for(i=0;i<count;i++)

{

for(j=0;j<count;j++)

{

 printf("\n%d->%d:",i,j);

scanf("%d",&cost\_matrix[i][j]);

if(cost\_matrix[i][j]<0)cost\_matrix[i][j]=1000;

}

}

 printf("\n Enter the source router:");

scanf("%d",&src\_router);

for(v=0;v<count;v++)

{

flag[v]=0;

last[v]=src\_router;

dist[v]=cost\_matrix[src\_router][v];

}

flag[src\_router]=1;

for(i=0;i<count;i++)

{

min=1000;

for(w=0;w<count;w++)

{

if(!flag[w])

if(dist[w]<min)

{

v=w;

min=dist[w];

}

}

flag[v]=1;

for(w=0;w<count;w++)

{

if(!flag[w])

if(min+cost\_matrix[v][w]<dist[w])

{

dist[w]=min+cost\_matrix[v][w];

last[w]=v;

}

}

}

for(i=0;i<count;i++)

{

 printf("\n%d==>%d:Path taken:%d",src\_router,i,i);

w=i;

while(w!=src\_router)

{

 printf("\n<--%d",last[w]);w=last[w];

}

 printf("\n Shortest path cost:%d",dist[i]);

}

}

main(){

    printf("Implement the following unicast routing algorithms using functions.\na. Distance Vector Routing\nb. Link State Routing");

    printf("\nEnter the choice a or b: ");

    char choice;

    scanf("%c",&choice);

    switch(choice){

        case 'a':

            a();

            break;

        case 'b':

            b();

            break;

        default :

            printf("\nInvalid choice!");

    }

}

OUTPUT

