**SUBMITTED BY:** Prachi Aggarwal

**COURSE:** Bsc(H) Computer Science

**ROLL NO:** 21570015

**SUBJECT:** Computer Networks

**UNIQUE PAPER CODE:** 32341303

**SEMESTER:** IIIrd

**YEAR:** 2nd

**SUBMITTED TO:** Ms. Anshula

**Examination Rollno:** 21033570042

**TOPIC:** Practical File

**1 ) WAP to simulate Cyclic Redundance check (CRC) error detection algorithm for noisy channel.**

#include <iostream>

#include <conio.h>

using namespace std;

int main()

{

int i,j,k,l;

//Get Dataword

int fs;

cout<<"\n Enter Dataword size: ";

cin>>fs;

int f[20];

cout<<"\n Enter Dataword:";

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

{

cin>>f[i];

}

//Get Generator

int gs;

cout<<"\n Enter Generator size: ";

cin>>gs;

int g[20];

cout<<"\n Enter Generator:";

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

{

cin>>g[i];

}

cout<<"\n Sender Side:";

cout<<"\n Dataword: ";

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

{

cout<<f[i];

}

cout<<"\n Generator :";

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

{

cout<<g[i];

}

//Append 0's

int rs=gs-1;

cout<<"\n Number of 0's to be appended: "<<rs;

for (i=fs;i<fs+rs;i++)

{

f[i]=0;

}

int temp[20];

for(i=0;i<20;i++)

{

temp[i]=f[i];

}

cout<<"\n Message after appending 0's :";

for(i=0; i<fs+rs;i++)

{

cout<<temp[i];

}

//Division

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

{

j=0;

k=i;

//check whether it is divisible or not (XOR)

if (temp[k]>=g[j])

{

for(j=0,k=i;j<gs;j++,k++)

{

if((temp[k]==1 && g[j]==1) || (temp[k]==0 && g[j]==0))

{

temp[k]=0;

}

else

{

temp[k]=1;

}

}

}

}

//CRC

int crc[15];

for(i=0,j=fs;i<rs;i++,j++)

{

crc[i]=temp[j];

}

cout<<"\n CRC bits: ";

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

{

cout<<crc[i];

}

cout<<"\n Transmitted Dataword: ";

int tf[15];

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

{

tf[i]=f[i];

}

for(i=fs,j=0;i<fs+rs;i++,j++)

{

tf[i]=crc[j];

}

for(i=0;i<fs+rs;i++)

{

cout<<tf[i];

}

cout<<"\n Receiver side : ";

cout<<"\n Received Dataword: ";

for(i=0;i<fs+rs;i++)

{

cout<<tf[i];

}

for(i=0;i<fs+rs;i++)

{

temp[i]=tf[i];

}

//Division (XOR)

for(i=0;i<fs+rs;i++)

{

j=0;

k=i;

if (temp[k]>=g[j])

{

for(j=0,k=i;j<gs;j++,k++)

{

if((temp[k]==1 && g[j]==1) || (temp[k]==0 && g[j]==0))

{

temp[k]=0;

}

else

{

temp[k]=1;

}

}

}

}

cout<<"\n Reaminder: ";

int rrem[15];

for (i=fs,j=0;i<fs+rs;i++,j++)

{

rrem[j]= temp[i];

}

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

{

cout<<rrem[i];

}

int flag=0;

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

{

if(rrem[i]!=0)

{

flag=1;

}

}

if(flag==0)

{

cout<<"\n Since Remainder Is 0 Hence Message Transmitted From Sender To Receriver Is

Correct";

}

else

{

cout<<"\n Since Remainder Is Not 0 Hence Message Transmitted From Sender To Receriver

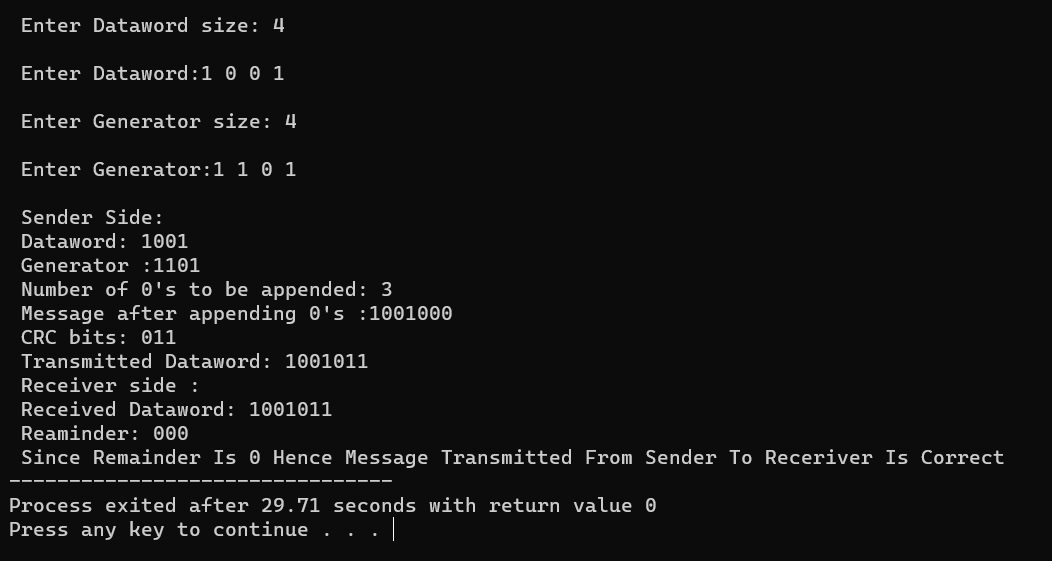
Contains Error ";

}

// getch();

}

**OUTPUT**

****

**Ques 2.) Simulate and implement stop and wait protocol for noisy channel.**

#include<iostream>

#include <time.h>

#include <cstdlib>

#include<ctime>

#include <unistd.h>

#include<iomanip>

using namespace std;

class timer {

privat e:

unsigned long begTime;

public:

void start() {

begTime = clock();

}

unsigned long elapsedTime() {

return ((unsigned long) clock() - begTime) / CLOCKS\_PER\_SEC;

}

bool isTimeout(unsigned long seconds) {

return seconds >= elapsedTime();

}

};

int main()

{

int frames[] = {1,2,3,4,5,6,7,8,9,10};

unsigned long seconds = 5;

srand(time(NULL));

timer t;

cout<<"Sender has to send frames : ";

for(int i=0;i<10;i++)

cout<<frames[i]<<" ";

cout<<endl;

int count = 0;

bool delay = false;

cout<<endl<<"Sender\t\t\t\t\tReceiver"<<endl;

do

{

bool timeout = false;

cout<<"Sending Frame : "<<frames[count];

cout.flush();

cout<<"\t\t";

t.start();

if(rand()%2)

{

int to = 24600 + rand()%(64000 - 24600) + 1;

for(int i=0;i<64000;i++)

for(int j=0;j<to;j++) {}

}

if(t.elapsedTime() <= seconds)

{

cout<<"Received Frame : "<<frames[count]<<" ";

if(delay)

{

cout<<"Duplicate";

delay = false;

}

cout<<endl;

count++;

}

else

{

cout<<"---"<<endl;

cout<<"Timeout"<<endl;

timeout = true;

}

t.start();

if(rand()%2 || !timeout)

{

int to = 24600 + rand()%(64000 - 24600) + 1;

for(int i=0;i<64000;i++)

for(int j=0;j<to;j++) {}

if(t.elapsedTime() > seconds )

{

cout<<"Delayed Ack"<<endl;

count--;

delay = true;

}

else if(!timeout)

cout<<"Acknowledgement : "<<frames[count]-1<<endl;

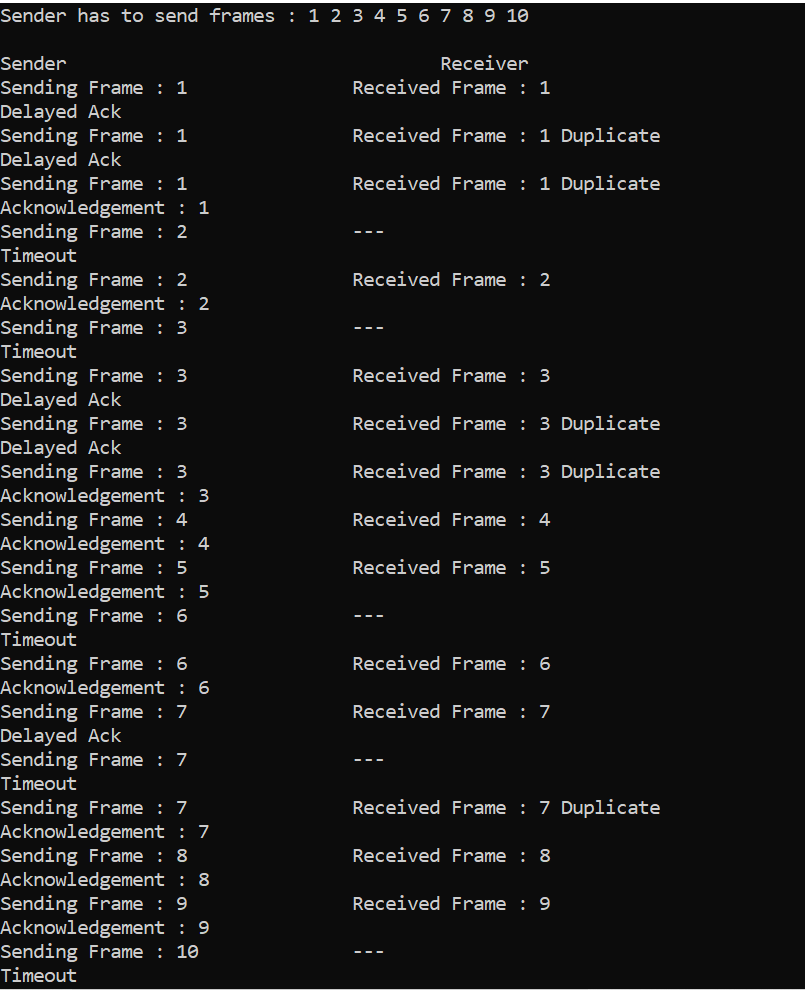
}

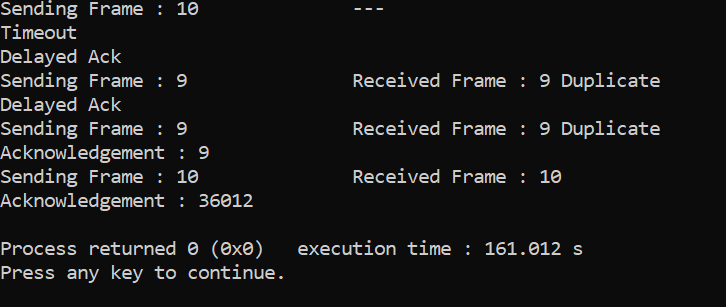
}while(count!=10);

return 0;

}

**OUTPUT**





**QUES 3.) Simulate and implement go back n sliding window protocol.**

#include<iostream>

#include<bits/stdc++.h>

#include<ctime>

using namespace std;

void transmission(long int start,long int w,long int n ,long int c) {

while (start <= n) {

int z = 0;

for (int k = start; k < start + w && k <= n; k++) {

cout << "Sending Frame " << k << "..." << endl;

c++;

}

for (int k = start; k < start + w && k <= n; k++) {

int f = rand() % 2;

if (!f) {

cout << "Acknowledgment for Frame " << k << "..." << endl;

z++;

} else {

cout << "Timeout!! Frame Number : " << k << " Not Received" << endl;

cout << "Retransmitting Window..." << endl;

break;

}

}

cout << "\n";

start = start + z;

}

cout << "Total number of frames which were sent and resent are : " << c <<endl;

}

int main() {

long int n, w, c = 0;

cout << "Enter the Total number of frames : ";

cin >> n;

cout << "Enter the Window Size : ";

cin >> w;

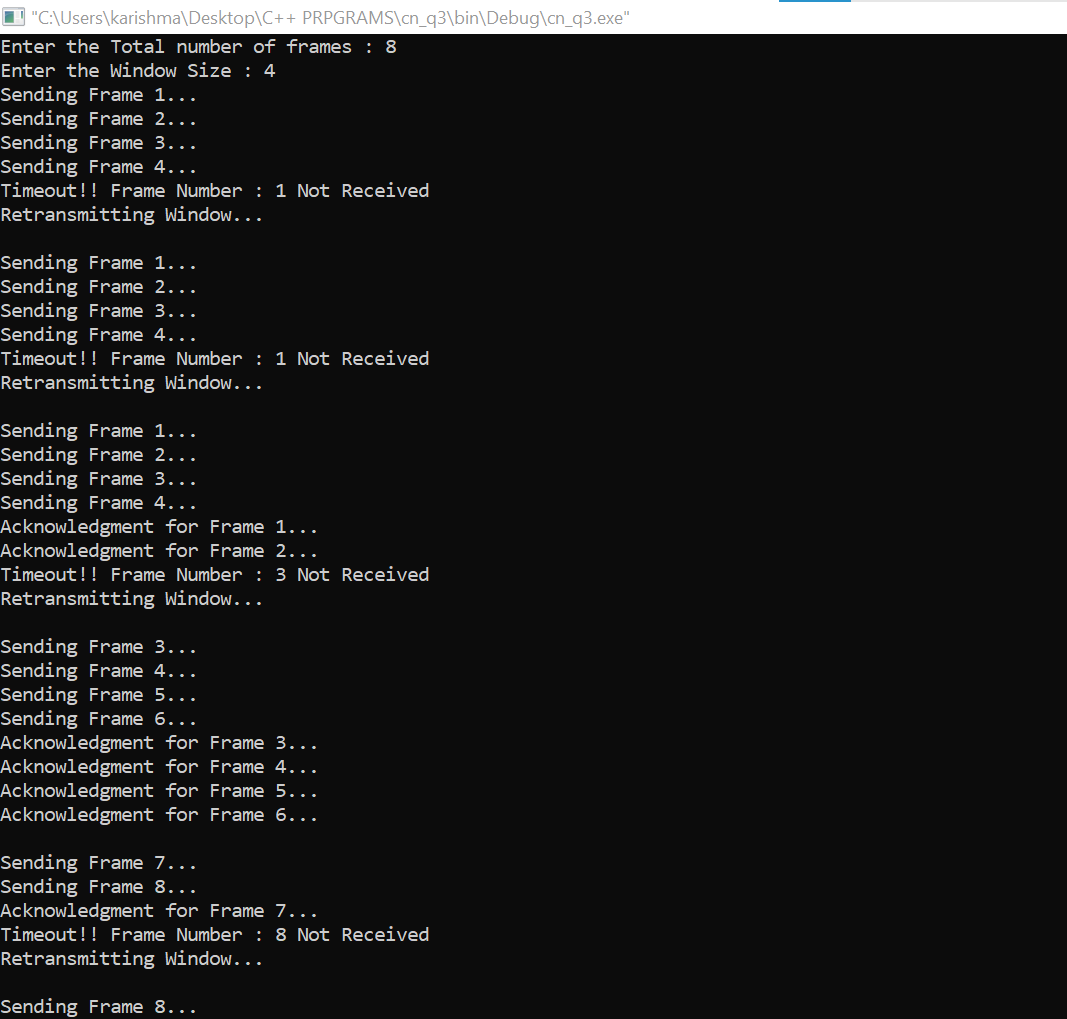
long int start = 1;

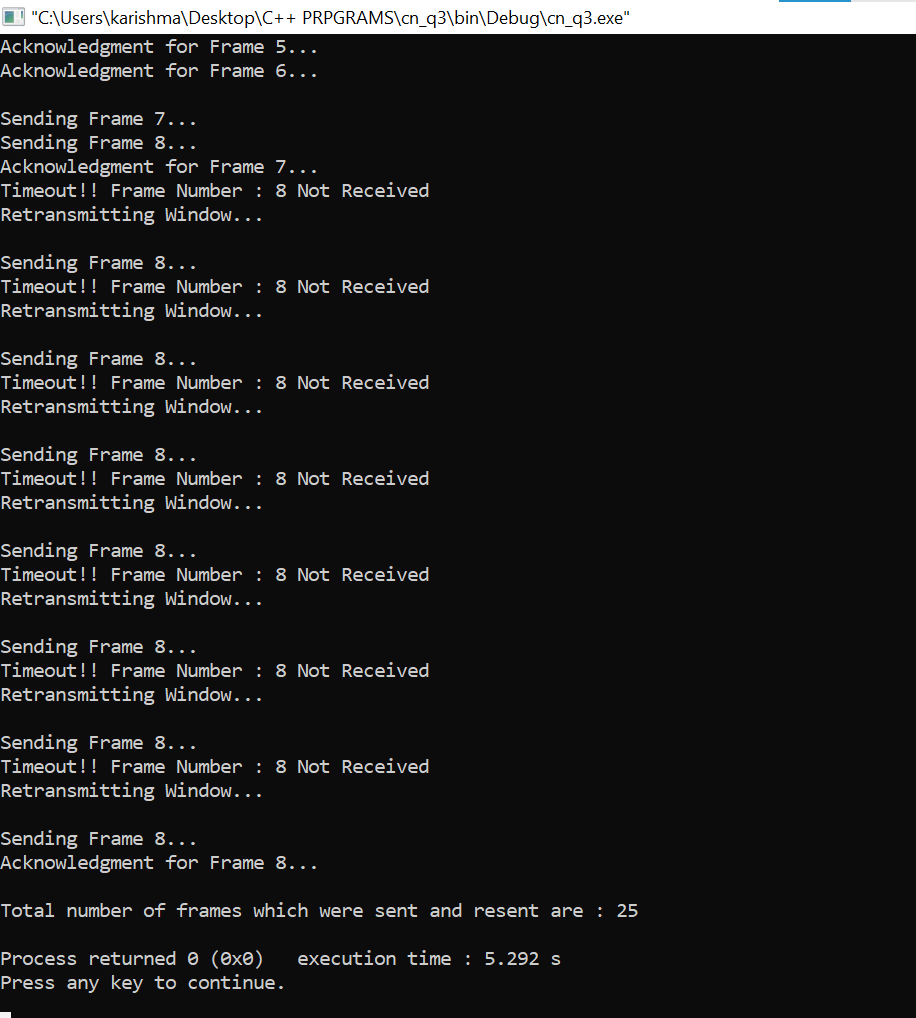
transmission(start, w, n, c);

return 0;

}

**OUTPUT**

****



**QUES 4.) Simulate and implement selective repeat sliding window protocol.**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

#include<time.h>

#include<math.h>

using namespace std;

#define TOT\_FRAMES 500

#define FRAMES\_SEND 10

class sel\_repeat

{

private:

int fr\_send\_at\_instance;

int arr[TOT\_FRAMES];

int send[FRAMES\_SEND];

int rcvd[FRAMES\_SEND];

char rcvd\_ack[FRAMES\_SEND];

int sw;

int rw;

public:

void input();

void sender(int);

void receiver(int);

};

void sel\_repeat::input()

{

int n;

int m;

int i;

cout<<"\*\*\*\*\*Please enter the no. of bits for the sequence no.\*\*\*\* : ";

cin>>n;

m=pow(2,n);

int t=0;

fr\_send\_at\_instance=(m/2);

for(i=0;i<TOT\_FRAMES;i++)

{

arr[i]=t;

t=(t+1)%m;

}

for(i=0;i<fr\_send\_at\_instance;i++)

{

send[i]=arr[i];

rcvd[i]=arr[i];

rcvd\_ack[i]='n';

}

rw=sw=fr\_send\_at\_instance;

sender(m);

}

void sel\_repeat::sender(int m)

{

cout<<"--------------------------------------------------------"<<endl;

for(int i=0;i<fr\_send\_at\_instance;i++)

{

if(rcvd\_ack[i]=='n')

cout<<"SENDER : Frame "<<send[i]<<" is============> SENT!!\n";

}

cout<<"\n----------------------------------------------------"<<endl;

receiver(m);

}

void sel\_repeat::receiver(int m)

{

time\_t t;

int f;

int j;

int f1;

int a1;

char ch;

srand((unsigned)time(&t));

for(int i=0;i<fr\_send\_at\_instance;i++)

{

if(rcvd\_ack[i]=='n')

{

f=rand()%10;

if(f!=5)

{

for(int j=0;j<fr\_send\_at\_instance;j++)

if(rcvd[j]==send[i])

{

cout<<"reciever:Frame"<<rcvd[j]<<"recieved correctly\n";

rcvd[j]=arr[rw];

rw=(rw+1)%m;

break;

}

int j;

if(j==fr\_send\_at\_instance)

cout<<"reciever:Duplicate frame"<<send[i]<<"discarded\n";

a1=rand()%5;

if(a1==3)

{

cout<<"(acknowledgement "<<send[i]<<" lost)\n";

cout<<"(sender timeouts-->Resend the frame)\n";

rcvd\_ack[i]='n';

}

else

{

cout<<"(acknowledgement "<<send[i]<<" recieved)\n";

rcvd\_ack[i]='p';

}

}

else

{int ld=rand()%2;

if(ld==0)

{

cout<<"RECEIVER : Frame "<<send[i]<<" is damaged\n";

cout<<"RECEIVER : Negative Acknowledgement "<<send[i]<<" sent\n";

}

else

{

cout<<"RECEIVER : Frame "<<send[i]<<" is lost\n";

cout<<"(SENDER TIMEOUTS-->RESEND THE FRAME)\n";

}

rcvd\_ack[i]='n';

}

}

}

for(int j=0;j<fr\_send\_at\_instance;j++)

{

if(rcvd\_ack[j]=='n')

break;

}

int i=0;

for(int k=j;k<fr\_send\_at\_instance;k++)

{

send[i]=send[k];

if(rcvd\_ack[k]=='n')

rcvd\_ack[i]='n';

else

rcvd\_ack[i]='p';

i++;

}

if(i!=fr\_send\_at\_instance)

{

for(int k=i;k<fr\_send\_at\_instance;k++)

{

send[k]=arr[sw];

sw=(sw+1)%m;

rcvd\_ack[k]='n';

}

}

cout<<"Do you want to continue?[Y/n]";

cin>>ch;

cout<<"\n";

if(ch=='y')

sender(m);

else

exit(0);

}

int main()

{

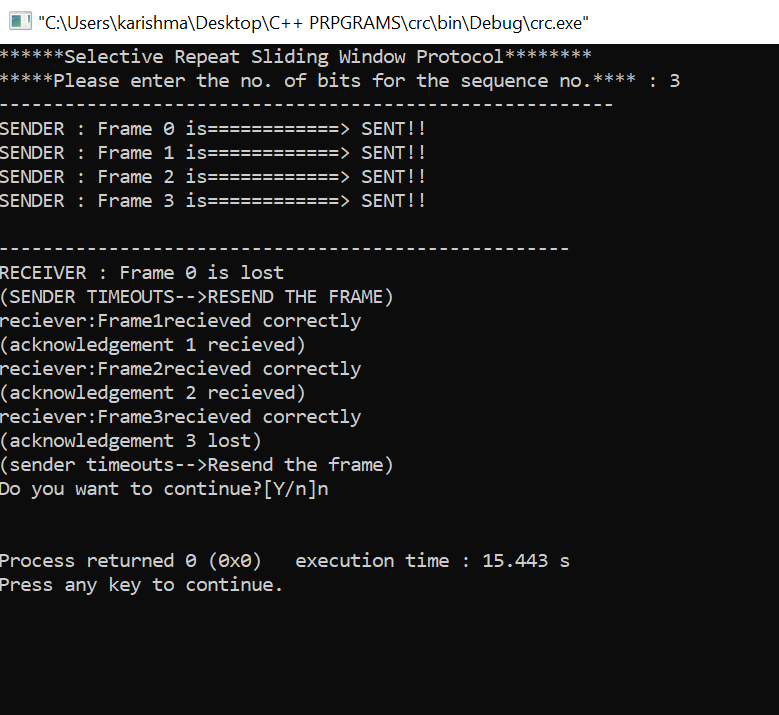
cout<<"\*\*\*\*\*\*Selective Repeat Sliding Window Protocol\*\*\*\*\*\*\*\*\n";

sel\_repeat sr;

sr.input();

return 0;

}

**OUTPUT**

**QUES 5.) Simulate and implement distance vector routing algorithm**

#include <iostream>

using namespace std;

int dist[50][50],temp[50][50],n,i,j,k,x;

void dvr();

int main()

{

cout << "Enter the number of nodes\n";

cin>>n;

cout<<"Enter the distance matrix\n";

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

{

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

{

cin>>dist[i][j];

dist[i][i]=0;

temp[i][j]=j;

}

cout<<endl;

}

dvr();

cout<<"Enter the value of i and j\n";

cin>>i>>j;

cout<<"Enter the new cost\n";

cin>>x;

dist[i][j]=x;

cout<<"After Update\n\n";

dvr();

return 0;

}

void dvr()

{

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

{

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

{

for(k=0;k<n;k++)

{

if(dist[i][k]+dist[k][j]<dist[i][j])

{

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

temp[i][j]=k;

}

}

}

}

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

{

cout<<"\nState value for router "<<i+1<<" is \n";

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

{

cout<<"\t\nnode"<<j+1<<" via "<<temp[i][j]<<" Distance "<<dist[i][j];

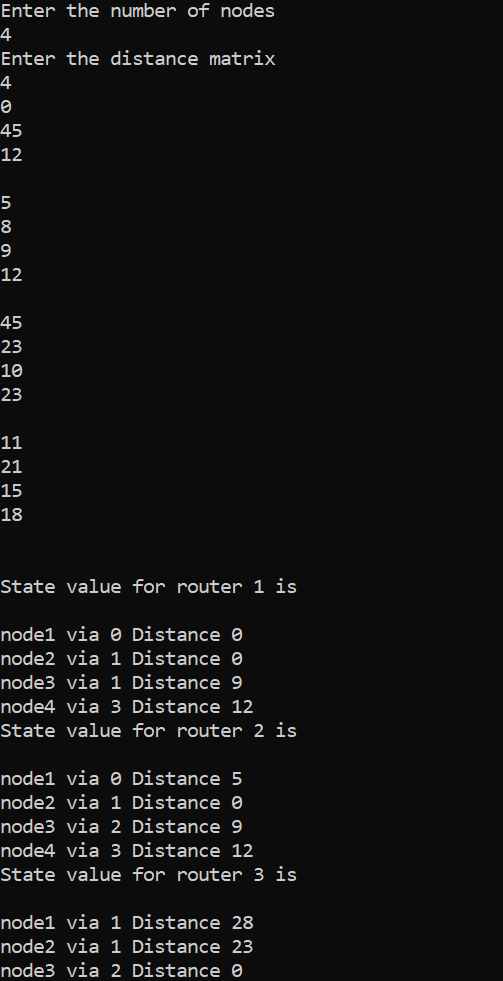
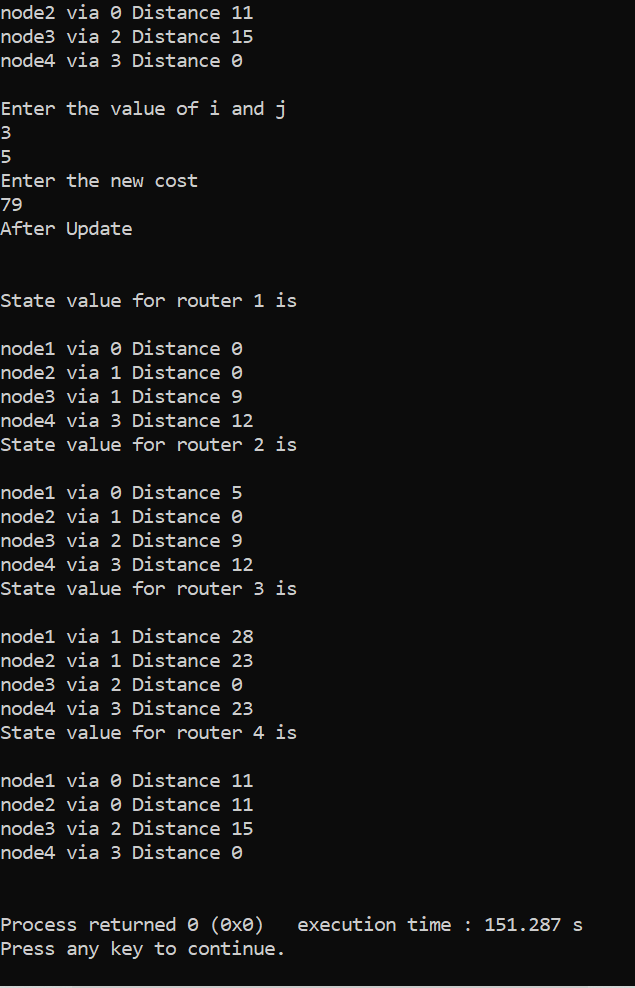
}

}

cout<<"\n\n";

}

**OUTPUT**



**QUES 6.) Simulate and implement Dijkstra algorithm for shortest path routing.**

#include <iostream>

using namespace std;

#include <limits.h>

int minDistance(int dist[], bool sptSet[]){

int min = INT\_MAX, min\_index;

for (int v = 0; v < 6; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[]){

cout <<"Vertex \t Distance from Source" << endl;

for (int i = 0; i < 6; i++)

cout << i << " \t\t"<<dist[i]<< endl;

}

void dijkstra(int graph[6][6], int src){

int dist[6];

bool sptSet[6];

for (int i = 0; i < 6; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < 6 - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < 6; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main(){

cout<<"for 6 vertexes\n";

int graph[6][6] = {

{0,1,2,0,0,0},

{1,0,0,5,1,0},

{2,0,0,2,3,0},

{0,5,2,0,2,2},

{0,1,3,2,0,1},

{0,0,0,2,1,0}

};

dijkstra(graph, 0);

return 0;

}

**OUTPUT**