**ATMA RAM SANATAN DHARMA COLLEGE**

**University of Delhi**



**COMPUTER NETWORKS**

**PRACTICAL FILE**

**SUBMITTED BY:**

**EKTA ARORA**

**ROLL NO:- 18075**

**BSc. (HONS) COMPUTER SCIENCE**

**SUBMITTED TO:**

**Dr. Uma Ojha**

**Department of Computer Science**

## PRACTICAL 1

## Simulate Cyclic Redundancy Check (CRC) error detection algorithm for noisy channel.

**Code:**

**#include <iostream>**

**#include<stdio.h>**

**using namespace std;**  
   
**int** **main**()  
{  
**int** msg[20],gr[20];

**int** m,n;  
cout<<"Enter the range of the message: ";

cin>>n;  
cout<<"Enter the message: ";

**for**(**int** i=0;i<n;i++)  
  cin>>msg[i];  
cout<<"Enter the range of generator polynomial: ";

cin>>m;  
cout<<"Enter the generator polynomial: ";

**for**(**int** i=0;i<m;i++)  
  cin>>gr[i];

cout<<"Message\n";

**for**(**int** i=0;i<n;i++)  
  cout<<msg[i];

**for**(**int** i=0;i<m;i++)  
   cout<<gr[i];  
   
   
**int** codeword[n+(m-1)];

**for**(**int** i=0;i<n;i++)  
   codeword[i]=msg[i];

**for**(**int** i=n;i<n+(m-1);i++)  
    codeword[i]=0;

**int** temp[n+(m-1)];  
**for**(**int** i=0;i<n+(m-1);i++)

     temp[i]=codeword[i];  
**for**(**int** i=0;i<n;i++)  
{  
     **int** j=0,k=i;

**if**(temp[k]>=gr[j])  
       **while**(j<m)

            temp[k++]^=gr[j++];  
}  
  
**int** crc[20];  
**for**(**int** i=0,j=n;i<(m-1);i++,j++)

     crc[i]=temp[j];  
cout<<"\ncrc: \n";

**for**(**int** i=0;i<(m-1);i++)  
     cout<<crc[i];

cout<<"\n";  
**for**(**int** i=0,j=n;i<(m-1);i++,j++)

     codeword[j]=crc[i];  
cout<<"\nTransmitted Message: ";

**for**(**int** i=0;i<n+(m-1);i++)  
    cout<<codeword[i]; cout<<"\n";  
   
*//FOR NOISY CHANNEL*  
**int** nbits,pos;  
cout<<"Enter number of bits to flip: ";

cin>>nbits;  
**for**(**int** i=0;i<nbits;i++)  
{  
cout<<"Enter the position to flip: ";

cin>>pos;  
codeword[pos-1]=codeword[pos-1]==0?1:0;  
}  
   
   
cout<<"------- AT RECEIVER-------- \n";  
**for**(**int** i=0;i<n+(m-1);i++)

   cout<<codeword[i];  
   cout<<"\n";

**int** temp2[n+(m-1)];  
**for**(**int** i=0;i<n+(m-1);i++)

    temp2[i]=codeword[i];  
**for**(**int** i=0;i<n;i++)  
{  
    **int** j=0,k=i;

**if**(temp2[k]>=gr[j])  
        **while**(j<m)

            temp2[k++]^=gr[j++];  
}  
**int** rem[20];  
**for** (**int** i = n, j = 0; i < n + (m - 1); i++, j++) r

   em[j] = temp2[i];  
   
cout << "Remainder: ";

**for** (**int** i=0;i<(m-1);i++)  
   cout<<rem[i];

cout<<"\n";

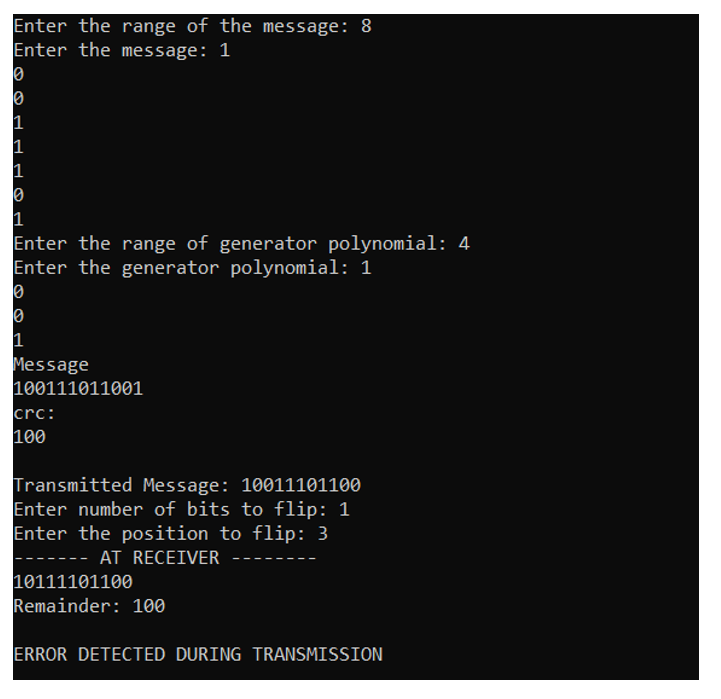
**int** flag = 0;  
**for** (**int** i = 0; i < (m - 1); i++)  
    **if** (rem[i] != 0)

       flag = 1;  
cout <<"\n";

**if** (flag==0)  
cout << "NO ERROR\n";

**else**  
   cout << "ERROR DETECTED DURING TRANSMISSION\n";  
}

**Output:**



**PRACTICAL 2**

## Simulate and implement stop and wait protocol for noisy channel.

**Code:**

**#include<iostream>**

**#include <time.h>**

**#include <cstdlib>**

**#include<ctime>**

**#include <unistd.h>**  
**using** **namespace** std;

**class** **timer**

{  
    **private**:  
      **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;

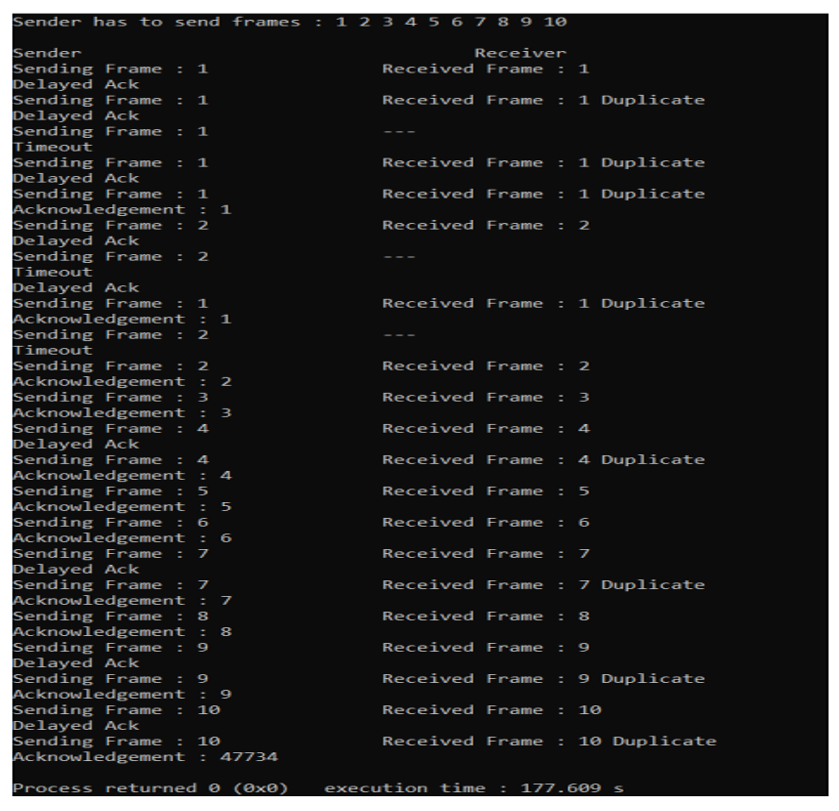
   count++;  
  
}  
**else**  
{  
    cout<<"---"<<endl;

    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:**



**PRACTICAL 3**

Simulate and implement selective repeat Go Back protocol.

**Code:**

**#include<bits/stdc++.h>**  
**#include<ctime>**  
  
**#define ll long long int**  
**using** **namespace** std;  
**void** **transmission**(ll & i, ll & N, ll & tf, ll & tt) {  
  **while** (i <= tf) {  
    **int** z = 0;  
    **for** (**int** k = i; k < i + N && k <= tf; k++) {  
      cout << "Sending Frame " << k << "..." << endl;  
      tt++;  
    }  
    **for** (**int** k = i; k < i + N && k <= tf; 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";  
    i = i + z;  
  }  
}  
  
**int** **main**() {  
  ll tf, N, tt = 0;  
  srand(time(NULL));  
  cout << "Enter the Total number of frames : ";  
  cin >> tf;  
  cout << "Enter the Window Size : ";  
  cin >> N;  
  ll i = 1;  
  transmission(i, N, tf, tt);  
  cout << "Total no of frames sent and resent are : " << tt <<endl;  
  **return** 0;  
}

**Output:**

Enter the Total number of frames : 12  
Enter the Window Size : 4  
**Sending** Frame 1...  
**Sending** Frame 2...  
**Sending** Frame 3...  
**Sending** Frame 4...  
Timeout!! Frame Number : 1 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 1...  
**Sending** Frame 2...  
**Sending** Frame 3...  
**Sending** Frame 4...  
**Acknowledgment** **for** Frame 1...  
Timeout!! Frame Number : 2 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 2...  
**Sending** Frame 3...  
**Sending** Frame 4...  
**Sending** Frame 5...  
Timeout!! Frame Number : 2 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 2...  
**Sending** Frame 3...  
**Sending** Frame 4...  
**Sending** Frame 5...  
**Acknowledgment** **for** Frame 2...  
**Acknowledgment** **for** Frame 3...  
**Acknowledgment** **for** Frame 4...  
Timeout!! Frame Number : 5 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 5...  
**Sending** Frame 6...  
**Sending** Frame 7...  
**Sending** Frame 8...  
Timeout!! Frame Number : 5 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 5...  
**Sending** Frame 6...  
**Sending** Frame 7...  
**Sending** Frame 8...  
**Acknowledgment** **for** Frame 5...  
Timeout!! Frame Number : 6 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 6...  
**Sending** Frame 7...  
**Sending** Frame 8...  
**Sending** Frame 9...  
**Acknowledgment** **for** Frame 6...  
Timeout!! Frame Number : 7 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 7...  
**Sending** Frame 8...  
**Sending** Frame 9...  
**Sending** Frame 10...  
**Acknowledgment** **for** Frame 7...  
**Acknowledgment** **for** Frame 8...  
**Acknowledgment** **for** Frame 9...  
**Acknowledgment** **for** Frame 10...  
**Sending** Frame 11...  
**Sending** Frame 12...  
Timeout!! Frame Number : 11 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 11...  
**Sending** Frame 12...  
Timeout!! Frame Number : 11 **Not** Received  
**Retransmitting** Window...  
**Sending** Frame 11...  
**Sending** Frame 12...  
**Acknowledgment** **for** Frame 11...  
**Acknowledgment** **for** Frame 12...  
Total number of frames transmitted(sent + resent) are : 38

## **PRACTICAL 4**

## Simulate and implement selective repeat sliding window protocol.

**Code:**

**#include<iostream>**

**#include<conio.h>**

**#include<time.h>**

**#include<math.h>**

**#define TOT\_FRAMES 500**  
**#define FRAMES\_SEND 10**   
**using** **namespace** std;

**class** **SelRepeat**  
{  
**private**:  
 **int** frames;  
 **int** arr[TOT\_FRAMES];

**int** send[FRAMES\_SEND];

**int** rcvd[FRAMES\_SEND];  
 **char** rcvd\_ack[FRAMES\_SEND];

**int** sw;  
 **int** rw;  
    
**public**:

**void** **input**()  
 {  
  **int** m,n,i;  
  cout<<"Enter the number of bits: ";

  cin>>n;  
  m=pow(2,n);

**int** t=0;  
  frames=(m/2);  
  **for**(i=0;i<TOT\_FRAMES;i++)  
  {  
   arr[i]=t;

   t=(t+1)%m;  
  }  
  **for**(i=0;i<frames;i++)  
  {  
   send[i]=arr[i];  
   rcvd[i]=arr[i];  
   rcvd\_ack[i]='n';  
   }  
  rw=sw=frames;

  sender(m);  
}

**void** **sender**(**int** m)  
{  
 **for**(**int** i=0;i<frames;i++)  
  {  
   **if**(rcvd\_ack[i]=='n')  
   cout<<"SENDER : Frame "<<send[i]<<" is sent\n";  
  }  
receiver(m);  
}

**void** **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<frames;i++)  
  {  
   **if**(rcvd\_ack[i]=='n')  
   {  
    f=rand()%10;  
    **if**(f!=5)*//if f=5 frame is discarded for some reason*  
*//else frame is correctly recieved*  
    {  
     **for**(**int** j=0;j<frames;j++)

      {

**if**(rcvd[j]==send[i])  
        {  
         cout<<"RECEIVER:Frame "<<rcvd[j]<<" recieved correctly\n";

         rcvd[j]=arr[rw];               
         rw=(rw+1)%m;

**break**;}

      }

    }  
 **int** j;  
 **if**(j==frames)  
    cout<<"RECEIVER:Duplicate Frame "<<send[i]<<"discarded\n"; a1=rand()%5;  
**if**(a1==3)*//if al==3 then ack is lost*  
*//else recieved*  
{  
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 =0 then frame damaged*  
  *//else frame lost*

***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 TIMEOUT-->RESEND THE FRAME\n";  
   }  
rcvd\_ack[i]='n';  
}  
}  
}  
**for**(**int** j=0;j<frames;j++)  
{  
**if**(rcvd\_ack[j]=='n')

**break**;  
}  
**int** i=0;  
**for**(**int** k=j;k<frames;k++)  
{  
send[i]=send[k];

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

**else**  
   rcvd\_ack[i]='p';

i++;  
}  
**if**(i!=frames)  
{  
   **for**(**int** k=i;k<frames;k++)  
   {  
      send[k]=arr[sw];

      sw=(sw+1)%m;

      rcvd\_ack[k]='n';  
   }  
}  
cout<<"Want to continue?";

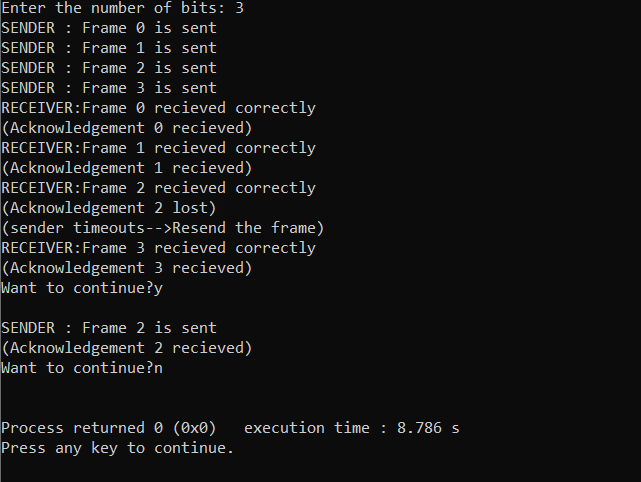
cin>>ch;  
cout<<"\n";

**if**(ch=='y')  
    sender(m);

**else**  
  exit(0);  
}};  
**int** **main**()  
{  
    SelRepeat obj;

    obj.input();  
}

**Output:**



## PRACTICAL 5

## Simulate and implement distance vector routing algorithm. **Code:**

**#include<iostream>**   
**#include<conio.h>**   
**using** **namespace** std;   
**struct** **node**  
{  
   **unsigned** dist[20];

**unsigned** from[20];  
}dvr[10];

**int** **main**()  
{  
   **int** cost[20][20];  
   **int** i,j,k,nodes,count=0;

   cout<<"\nEnter the number of nodes: ";

   cin>>nodes;  
   cout<<"\nEnter the cost matrix: \n";

**for**(i=0;i<nodes;i++)  
   {  
    **for**(j=0;j<nodes;j++)  
    {  
      cin>>cost[i][j];  
      cost[i][i]=0; dvr[i].dist[j]=cost[i][j];*//initializing distance*  
*equal to cost matrix*  
       dvr[i].from[j]=j;  
     }  
    }  
  
**do**  
{  
 count=0;

**for**(i=0;i<nodes;i++)  
  **for**(j=0;j<nodes;j++)

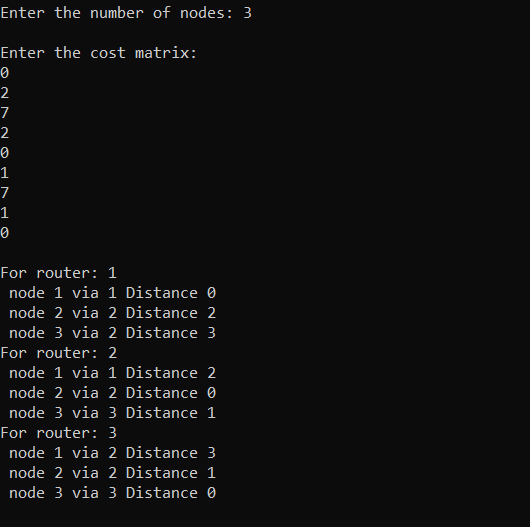
**for**(k=0;k<nodes;k++)  
          **if**(dvr[i].dist[j]>cost[i][k]+dvr[k].dist[j])  
          {*//calculate the minimum distance*  
           dvr[i].dist[j]=dvr[i].dist[k]+dvr[k].dist[j];

           dvr[i].from[j]=k; count++;  
          }  
}**while**(count!=0);

**for**(i=0;i<nodes;i++)  
{  
    cout<<"\nFor router: "<<i+1;

**for**(j=0;j<nodes;j++)  
   {  
      cout<<"\t\n node "<<j+1<<" via "<<dvr[i].from[j]+1<<" Distance "<<dvr[i].dist[j];  
   }  
}  
cout<<endl;

getch();  
}

**Output:**

## **PRACTICAL 6**

## Simulate and implement Dijkstra algorithm for shortest path routing.

**Code:**

**#include <iostream>**

**#include <iomanip>**

**#define MAX\_NODES 20**  
   
**using** **namespace** std;  
   
   
**class** **Set**  
{  
  
**public**:

**int** edge;  
**int** vertex;  
**int** path[MAX\_NODES];

**int** dist[MAX\_NODES];  
**int** adjMatrix[MAX\_NODES][MAX\_NODES];  
   
   
**void** **input**(**int** v, **int** e)  
{  
edge = e; vertex = v;  
**for** (**int** i = 0; i < v; i++)*// initializing the adjacency matrix*  
   **for** (**int** j = 0; j < v; j++)

          adjMatrix[i][j] = 0;  
**int** src, dest, weight;  
**for** (**int** i = 0; i < edge; i++)  
{  
cout << "\nEDGE " << (i + 1)  
<< "\n----- \n";  
cout << "Enter Source: ";

cin >> src;  
cout << "Enter Destination: ";

cin >> dest;  
cout << "Enter Weight: ";

cin >> weight;  
adjMatrix[src - 1][dest - 1] = weight;

adjMatrix[dest - 1][src - 1] = weight;  
}  
}

**void** **display**()  
{  
**for** (**int** i = 0; i < vertex; i++)  
{  
**for** (**int** j = 0; j < vertex; j++)  
cout << setw(5) << adjMatrix[i][j] << " "; cout <<"\n";  
}  
}  
   
   
**void** **dijkstra**(**int** src)  
{  
**bool** visited[MAX\_NODES];  
**for** (**int** i = 0; i < vertex; i++)  
{  
visited[i] = false;

dist[i] = INT\_MAX;  
}  
path[src] = -1;*//source node dist[src] = 0;*  
**for** (**int** i = 0; i < vertex- 1; i++)  
{  
**int** u = minDist(visited);*//nearest node visited[u] = true;*  
 **for** (**int** v = 0; v < vertex; v++)  
    **if** (visited[v] == false && adjMatrix[u][v] && dist[u] != INT\_MAX && dist[u] + adjMatrix[u][v] < dist[v])  
    {  
       path[v] = u;  
        dist[v] = dist[u] + adjMatrix[u][v];  
    }  
}  
cout << "\nDestn Node \t Distance \t Shortest Path";*//displaying*  
cout << "\n       \t      \t       ";  
**for** (**int** i = 0; i < vertex; i++)  
{  
cout <<"\n"<< (i + 1)<< " \t\t " << dist[i]<< " \t\t " << (src + 1);  
printShortestPath(i);  
}  
}  
   
   
**int** **minDist**(**bool** \*visited)  
{  
**int** min = INT\_MAX, min\_index;

**for** (**int** v = 0; v < vertex; v++)  
**if** (visited[v] == false && dist[v] <= min)  
   min = dist[v]; min\_index = v;  
**return** min\_index;  
}  
  
**void** **printShortestPath**(**int** node)  
{  
**if** (path[node] == -1) **return**;  
printShortestPath(path[node]); cout << " -> " << (node + 1);  
}  
};  
   
   
**int** **main**()  
{  
**int** ver, ed; Set s;  
   
cout << "Enter total number of Nodes: ";

cin >> ver;  
cout << "Enter number of Edges: ";

cin >> ed;  
s.input(ver, ed); cout<<"\nGRAPH\n";

cout<<"------- \n";  
s.display();

cout <<"\n";  
cout << "Enter Source Node: "; cin >> ver;  
s.dijkstra(ver - 1);  
**return** 0;  
}

**Output:**  
