Malla Reddy Engineering College (Autonomous)

(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad). Accredited 2nd time by NAAC with 'A' Grade, Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad Telangana-500100 www.mrec.ac.in

LABORATORY RECORD

LABORATORY NAME: COMPUTER NETWORKS LAB

LABORATORY CODE: 80519

YEAR/SEMESTER: II/II

REGULATIONS: MR18 (2019-20 ADMITTED BATCH)

DEPARTMENT OF INFORMATION TECHNOLOGY

JULY-2021

Malla Reddy Engineering College (Autonomous)

(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad). Accredited 2nd time by NAAC with 'A' Grade, Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad Telangana-500100 www.mrec.ac.in

Department of Information Technology

CERTIFICATE

THIS IS TO CERTIFY THAT IT IS BONAFIED RECORD OF LABORATORY WORK DONE BY MR./MS.____________________________________BEARING THE ROLL NUMBER ________________OF___________________YEAR/SEMESTER ________________________ DEPARTMENT IN THE ____LABORATORY DURING THE ACADEMIC YEAR__________________UNDER OUR OBSERVATION.

LAB IN-CHARGE HEAD OF DEPARTMENT

INTERNAL EXAMINER EXTERNAL EXAMINER

INDEX

1. **Implement the Data Link Layer Farming Methods:**

a) Character Count

```
#include <stdio.h>

#include <string.h>

  int main()

{

   char string[] = "The best of both worlds";

   int count = 0;

       //Counts each character except space

   for(int i = 0; i < strlen(string); i++) {

       if(string[i] != ' ')

           count++;

   }

       //Displays the total number of characters present in the given string

   printf("Total number of characters in a string: %d", count);

      return 0;

}
```

Output:

```
Total number of characters in a string: 19
```

b) Character stuffing and destuffing

The framing method gets around the problem of resynchronization after an error by having each frame start the ASCII character sequence DLE STX and the sequence DLE ETX. If the destination ever losses the track of the frame boundaries all it has to do is look for DLE STX or DLE ETX characters to figure out. The data link the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing

Algorithm:

```
Begin

Step 1: Initialize I and j as 0

Step 2: Declare n and pos as integer and a[20],b[50],ch as character

Step 3: read the string a

Step 4: find the length of the string n, i.e n-strlen(a)

Step 5: read the position, pos

Step 6: if pos > n then

Step 7: print invalid position and read again the position, pos

Step 8: end if

Step 9: read the character, ch

Step 10: Initialize the array b , b[0…5] as ’d’, ’l’, ’e’, ’s’ ,’t’ ,’x’ respectively

Step 11: j=6;

Step 12: Repeat step[(13to22) until i<n

Step 13: if i==pos-1 then

Step 14: initialize b array,b[j],b[j+1]…b[j+6] as‘d’, ‘l’, ‘e’ ,’ch, ’d’, ‘l’,‘e’ respectively

Step 15: increment j by 7, i.e j=j+7

Step 16: end if

Step 17: if a[i]==’d’ and a[i+1]==’l’ and a[i+2]==’e’ then Step 18: initialize array b,
b[13…15]=’d’, ‘l’, ‘e’ respectively Step 19: increment j by 3, i.e j=j+3

Step 20: end if
Step 21: b[j]=a[i]
Step 22: increment I and j;
Step 23: initialize b array,b[j],b[j+1]…b[j+6] as'd', 'l','e' ,'e','t', 'x','\0' respectively
Step 24: print frame after stiuffing
Step 25: print b
End
```

Source Code:

```
//PROGRAM FOR CHARACTER STUFFING
#include<stdio.h>
#include<string.h>
#include<process.h>
void main()
{
int i=0,j=0,n,pos;
 char a[20],b[50],ch;
printf("enter string\n");
scanf("%s",&a);
n=strlen(a);
printf("enter position\n");
 scanf("%d",&pos);
 if(pos>n)
{
printf("invalid position, Enter again :");
scanf("%d",&pos);}
printf("enter the character\n");
```

```
ch=getche();
b[0]='d';
b[1]='l';
b[2]='e';
b[3]='s';
b[4]='t';
b[5]='x'; j=6;
while(i<n)
{
if(i==pos-1)
{
b[j]='d';
b[j+1]='l';
b[j+2]='e';
b[j+3]=ch; b[j+4]='d';
b[j+5]='l';
b[j+6]='e';
j=j+7;
}
if(a[i]=='d' && a[i+1]=='l' && a[i+2]=='e')
{
b[j]='d';
b[j+1]='l';
b[j+2]='e';
j=j+3;
}
b[j]=a[i];
i++; j++;
}
b[j]='d';
b[j+1]='l';
b[j+2]='e';
b[j+3]='e';
b[j+4]='t';
b[j+5]='x';
b[j+6]='\0';
printf("\nframe after stuffing:\n");
printf("%s",b);
}
```

Output:

```
Enter string MREC
enter position : 2
enter the character frame after stuffing:
dlestxMdldleRECdleetx
(program exited with code: 0)
 Press return to continue
```

c) Bit stuffing and destuffing

Algorithm:

```
Begin
Step 1: Read frame length n
Step 2: Repeat step (3 to 4) until i<n(: Read values into the input frame (0's and1's) i.e.
Step 3: initialize I i=0;
Step 4: read a[i] and increment i
Step 5: Initialize i=0, j=0,count =0
Step 6: repeat step (7 to 22) until i<n
Step 7: If a[i] == 1 then
Step 8: b[j] = a[i]
Step 9: Repeat step (10 to 18) until (a[k] =1 and k<n and count <5)
Step 10: Initialize k=i+1;
Step 11: Increment j and b[j]= a[k];
Step 12: Increment count ;
Step 13: if count =5 then
Step 14: increment j,
Step 15: b[j] =0
Step 16: end if
Step 17: i=k;
Step 18: increment k
Step 19: else
Step 20: b[j] = a[i]
Step 21: end if
Step 22: increment I and j
Step 23: print the frame after bit stuffing
Step 24: repeat step (25 to 26) until i< j
Step 25: print b[i]
Step 26: increment i
End
```

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
int a[20],b[30],i,j,k,count,n;
printf("Enter frame length:");
scanf("%d",&n);
printf("Enter input frame (0's & 1's only):");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
i=0; count=1; j=0;
while(i<n)
{
if(a[i]==1)
{
b[j]=a[i];
for(k=i+1;a[k]==1 && k<n && count<5;k++)
{
j++;
b[j]=a[k];
count++;
if(count==5)
{
j++;
b[j]=0;
}
i=k;
}}
else
{
b[j]=a[i];
}
i++;
j++;
}
printf("After stuffing the frame is:");
for(i=0;i<j;i++)
printf("%d",b[i]);
}
```

Program Output:

```
Enter frame length:5
Enter input frame (0's & 1's only):
1
1
1
1
1
After stuffing the frame is:111110
------------------
(program exited with code: 6)
Press return to continue
```

2. Implement on a data set of characters the three CRC polynomials: CRC-12, CRC-16 and CRC-32.

Theory:

CRC method can detect a single burst of length n, since only one bit per column will be changed, a burst of length n+1 will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be 2 power(-n). This scheme sometimes known as Cyclic Redundancy Code

Algorithm:

Begin

Step 1: Declare I,j,fr[8],dupfr[11],recfr[11],tlen,flag,gen[4],genl,frl, rem[4] as integer

Step 2: initialize frl=8 and genl=4

Step 3: initialize i=0

Step 4: Repeat step(5to7) until i<frl

Step 5: read fr[i]

Step 6: dupfr[i]=fr[i]

Step 7: increment i

Step 8: initialize i=0

Step 9: repeat step(10to11) until i<genl

Step 10: read gen[i]

Step 11: increment i

Step 12: tlen=frl+genl-1

Step 13: initialize i=frl

Step 14: Repeat step(15to16) until i<tlen

Step 15: dupfr[i]=0

Step 16: increment i

```
Step 17: call the function remainder(dupfr)
Step 18: initialize i=0
Step 19: repeat step(20 to 21) until j<genl
Step 20: recfr[i]=rem[j]
Step 21: increment I and j
Step 22: call the function remainder(dupfr)
Step 23: initialize flag=0 and i=0
Step 24: Repeat step(25to28) until i<4
Step 25: if rem[i]!=0 then
Step 26: increment flag
Step 27: end if
Step 28: increment i
Step 29: if flag=0 then
Step 25: print frame received correctly
Step 25: else
Step 25: print the received frame is wrong
End
Function: Remainder(int fr[]) Begin
Step 1: Declare k,k1,I,j as integer
Step 2: initialize k=0;
Step 3: repeat step(4 to 14) until k< frl
Step 4: if ((fr[k] == 1) then
Step 5: k1=k
Step 6: initialize i=0, j=k
Step 7: repeat step(8 to 9) until i< genl
Step 8: rem[i] =fr[j] exponential gen[i]
Step 9: increment I and j
Step 10: initialize I = 0
Step 11: repeat step(12to13) until I <genl
Step 12: fr[k1] = rem[i]
Step 13: increment k1 and i
Step 14: end if
End
```

//Program for Cyclic Redundancy Check

```
#include<stdio.h>
int gen[4],genl,frl,rem[4]; void main()
{
int i,j,fr[8],dupfr[11],recfr[11],tlen,flag; frl=8; genl=4;
printf("enter frame:"); for(i=0;i<frl;i++)
{
scanf("%d",&fr[i]); dupfr[i]=fr[i];
}
printf("enter generator:");
 for(i=0;i<genl;i++)
scanf("%d",&gen[i]);
tlen=frl+genl-1;
for(i=frl;i<tlen;i++)
{
dupfr[i]=0;
}
remainder(dupfr);
for(i=0;i<frl;i++)
{
recfr[i]=fr[i];
}
for(i=frl,j=1;j<genl;i++,j++)
{
recfr[i]=rem[j];
}
remainder(recfr);
flag=0;
for(i=0;i<4;i++)
{
if(rem[i]!=0) flag++;
}
if(flag==0)
{
printf("frame received correctly");
}
else
{
printf("the received frame is wrong");
}
}
remainder(int fr[])
```

```
{
int k,k1,i,j;
for(k=0;k<frl;k++)
{
if(fr[k]==1)
{
k1=k;
for(i=0,j=k;i<genl;i++,j++)
{
rem[i]=fr[j]^gen[i];
}
for(i=0;i<genl;i++)
{
fr[k1]=rem[i]; k1++;
}
}
}
}
```

OUTPUT:

```
enter frame: MREC
enter generator: frame received correctly
(program exited with code: 24)
 Press return to continue
```

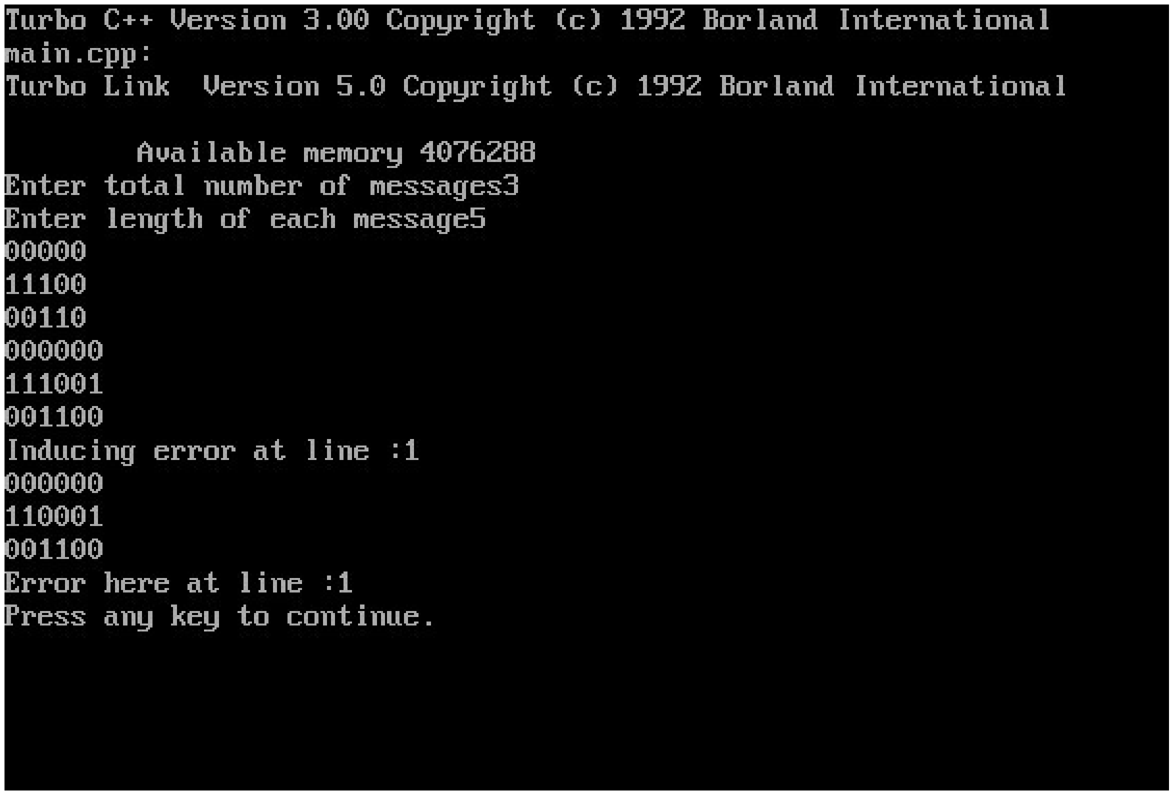
3. Implement Parity Check using the following techniques

a) Single Dimensional Data

b. Multi Dimensional Data

```
#include<iostream>
#include<stdlib.h>
using namespace std;
#define maxlength 10
#define maxmessages 10
void initialize(int arr[][10],int m,int n)
{
for(int i =0;i<m;i++)
for(int j=0;j<n;j++)
{
arr[i][j] = rand()%2;
}
}
void print(int arr[][10],int m,int n)
{
for(int i =0;i<m;i++)
{
 for(int j=0;j<n;j++)
{
cout<<arr[i][j]<<" ";
}
cout<<endl;
}
}
void addparbit(int arr[][10],int m,int n)  // Even Parity
{
for(int i=0;i<m;i++)
{
int count = 0;
for(int j=0;j<n;j++)
{
if(arr[i][j] == 1)
count++;
}
if(count%2 == 0)
arr[i][n] = 0;
else
arr[i][n] = 1;
}
}
void induceerror(int arr[][10],int m,int n)
```

```
{
int k1,k2;
k1= rand()%m;
k2 = rand()%n;
if(arr[k1][k2]==0)
arr[k1][k2]=1;
else
arr[k1][k2]=0;
cout<<"Inducing error at line : "<<k1<<endl;
}
void checkerror(int arr[][10],int m,int n)
{
for(int i=0;i<m;i++)
{
int count = 0;
for(int j=0;j<n;j++)
{
if(arr[i][j] == 1)
count++;
}
if(count%2 == 0 && arr[i][n] != 0)
{
cout<<"Error here at line : " <<i;
}
else if(count%2 == 1 && arr[i][n] != 1)
{
cout<<"Error here at line : " <<i;
}
}
}
int main()
{
  int m,n,arr[maxmessages][maxlength];

cout<<"Enter total number of messages";

cin>>m;

cout<<"Enter length of each message";

cin>>n;

initialize(arr,m,n);

print(arr,m,n);
addparbit(arr,m,n);

print(arr,m,n+1);

induceerror(arr,m,n);

print(arr,m,n+1);

checkerror(arr,m,n);

return 0;

}
```



```
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
main.cpp:
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4076288
Enter total number of messages3
Enter length of each message5
00000
11100
00110
000000
111001
001100
Inducing error at line :1
000000
110001
001100
Error here at line :1
Press any key to continue.
```

4. Impelementation of Sliding Window Protocols

a).One bit sliding window protocol

```
#include<stdio.h>
int main()
{
   int w,i,f,frames[50];
   printf("Enter window size: ");
   scanf("%d",&w);
   printf("\nEnter number of frames to transmit: ");
   scanf("%d",&f);
    printf("\nEnter %d frames: ",f);
    for(i=1;i<=f;i++)
       scanf("%d",&frames[i]);
  printf("\nWith sliding window protocol the frames will be sent in the following manner
(assuming no corruption of frames)\n\n");
   printf("After sending %d frames at each stage sender waits for acknowledgement sent by the
receiver\n\n",w);
    for(i=1;i<=f;i++)
    {
       if(i%w==0)
       {
          printf("%d\n",frames[i]);
          printf("Acknowledgement of above frames sent is received by sender\n\n");
       }
       else
```

```
        printf("%d ",frames[i]);
    }


    if(f%w!=0)
        printf("\nAcknowledgement of above frames sent is received by sender\n");
    return 0;
}
```

Output:

```
Enter window size: 3
Enter number of frames to transmit: 5
Enter 5 frames: 12 5 89 4 6
With sliding window protocol the frames will be sent in the following manner (assuming no
corruption of frames)
After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver
12 5 89
Acknowledgement of above frames sent is received by sender
4 6
Acknowledgement of above frames sent is received by sender
```

b). Go Back N sliding window protocol

Code in C:

```
#include<stdio.h>
int main()
{
        int windowsize,sent=0,ack,i;
        printf("enter window size\n");
        scanf("%d",&windowsize);
        while(1)
        {
                for( i = 0; i < windowsize; i++)
                        {
                                printf("Frame %d has been transmitted.\n",sent);
                                sent++;
                                if(sent == windowsize)
                                        break;
                        }
                        printf("\nPlease enter the last Acknowledgement received.\n");
                        scanf("%d",&ack);

                        if(ack == windowsize)
                                break;
                        else
                                sent = ack;
        }
return 0;

}
```

output:-

```
enter window size 8

Frame 0 has been transmitted.

Frame 1 has been transmitted.

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.


Please enter the last Acknowledgement received.2

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.


Please enter the last Acknowledgement received.8
```

1. --------------------------------

5. Write a code simulating ARP /RARP protocols

Algorithm:

```
Step 1: Initialize the string of ip addresses
Step 2: For every ip address, assign an ethernet address
Step 3: To Perform ARP, enter the ip address
Step 4: The equivalent ethernet address is displayedStep
Step 5: If the ethernet address is not found, then 'not found' message is displayeS
Step 6: To Perform RARP, enter the ethernet address
Step 7: The equivalent ip address is displayed
Step 8: If the ip address is not found, then 'not found' message is displayed
Step 9: Provide option to perform both ARP and RARP
Step 10:Choose Exit option to terminate the program
```

Program:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
char ip[10][20]={"192.168.0.64","192.168.0.60","192.168.0.68","132.147.3.3"};
char
et[10][20]={"00_A8_00_40_8E_FS","00_16_17_31_8e_22","00_16_17_31_8E_F7","00_16_17
_31_8E_08"};
char ipaddr[20],etaddr[20];
int i,op;
int x=0,y=0;
clrscr();
while(1)
{
printf("\n\n 1.ARP 2.RARP 3.EXIT");
printf("\n enter the choice");
scanf("%d",&op);
switch(op)
{
case 1:
printf("\n enter ip address");
scanf("%s",ipaddr);
for(i=0;i<=20;i++)
{
if(strcmp(ipaddr,ip[i])==0)
{
printf("Ethernet address is%s",et[i]);
x=1;
} }
if(x==0)
printf("invalid ip address");
x=0;
break;
case 2:
printf("enter ethernet address");
scanf("%s",etaddr);
for(i=0;i<=20;i++)
{
if(strcmp(etaddr,et[i])==0)
{
printf("IP address is %s",ip[i]);
y=1;
} }
if(y==0)
printf("Invalid ethernet address");
y=0;
break;
case 3:
exit(0);
} } }
```

6. Impelementation of Routing Protocols

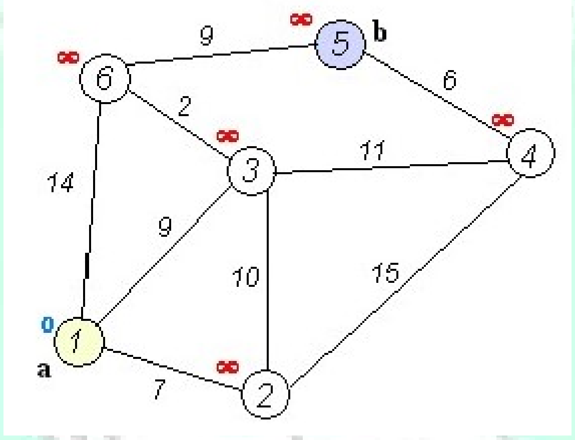
Dijkstra’s Algorithm in C

Dijkstra algorithm is also called single source shortest path algorithm. It is based on greedy technique. The algorithm maintains a list visited[ ] of vertices, whose shortest distance

from the source is already known.

If visited[1], equals 1, then the shortest distance of vertex i is already known. Initially, visited[i] is marked as, for source vertex.

At each step, we mark visited[v] as 1. Vertex v is a vertex at shortest distance from the source vertex. At each step of the algorithm, shortest distance of each vertex is stored in an array distance [ ].



Also Read: Prim’s

0

Dijkstra’s Algorithm

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the

cost of going from verte vertices i and j then C[i]i to vertex j. If there is no edge between j] is infinity.

2. Array visited[ ] is initialized to zero. for(i=0;i<n;i++)

visited[i]=0;

3. If the vertex 0 is the source vertex then visited[0] is marked as 1.

4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.

for(i=1;i<n;i++)

distance[i]=cost[0][i];

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

5. for(i=1;i<n;i++)

- Choose a vertex w, such that distance[w] is minimum and visited[w] is

  0. Mark visited[w] as 1.

- Recalculate the shortest distance of remaining vertices from the source.

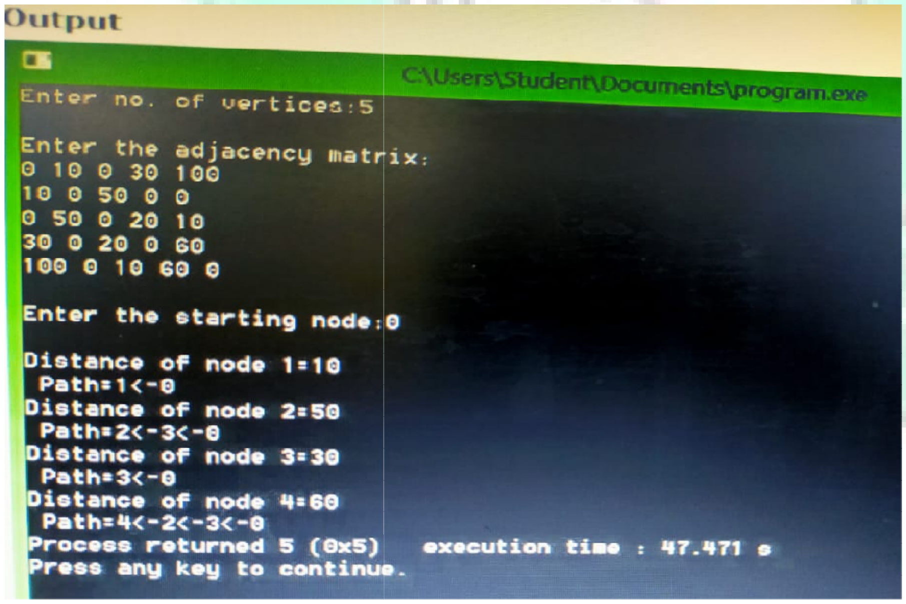
- Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

if(visited[v]==0)

distance[v]=min(distance[v], distance[w]+cost[w][v])

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u; printf("Enter no. of vertices:"); scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n"); for(i=0;i<n;i++)
for(j=0;j<n;j++) scanf("%d",&G[i][j]); printf("\nEnter the starting node:"); scanf("%d",&u);
dijkstra(G,n,u); return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
 int visited[MAX],count,mindistance,nextnode,
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0;i<n;i++)
 for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
 if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j]; printf("<-%d",j);
}while(j!=startnode);
}
}
```

Output:



Output

C:\Users\Student\Documents\program.exe

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node 1=10
 Path=1<-0
Distance of node 2=50
 Path=2<-3<-0
Distance of node 3=30
 Path=3<-0
Distance of node 4=60
 Path=4<-2<-3<-0
Process returned 5 (0x5)   execution time : 47.471 s
Press any key to continue.
```

b).Distance Vector routing protocol

Algoithm:

1. Start
2. By convention, the distance of the node to itself is assigned to zero and when a node is unreachable the distance is accepted as 999.
3. Accept the input distance matrix from the user (*dm[][]*) that represents the distance between each node in the network.
4. Store the distance between nodes in a suitable varible.
5. Calculate the minimum distance between two nodes by iterating.
   - If the distance between two nodes is larger than the calculated alternate available path, replace the existing distance with the calculated distaance.
6. Print the shortest path calculated.
7. Stop.

```
#include<stdio.h> struct node
{
unsigned dist[20];
unsigned from[20];
}rt[10];
int main()
{
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 thedistance equal to cost matrix
rt[i].from[j]=j;
}
}
do
{
count=0;
for(i=0;i<nodes;i++)
//We choose arbitrary 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();
}
/*A sample run of the program works as:-
Enter the number of nodes : 3
Enter the cost matrix :
0 2 7
2 0 1
7 1 0
        For router      1
        node 1 via      1   Distance    0
        node 2 via      2   Distance    2
        node 3 via      3   Distance    3
        For router      2
        node 1 via      1   Distance    2
        node 2 via      2   Distance    0
        node 3 via      3   Distance    1
```

```
For router      3
node 1 via      1   Distance     3
node 2 via      2   Distance     1
node 3 via      3   Distance     0
*/
```

c).Link State routing protocol

Overview Link-state routing protocol is a main class of routing protocols. It is performed by every router (switching node) in the network. The base concept of link-state routing is that every node constructs a map of the connectivity to the network, showing which nodes are connected to which other nodes. Each node then independently calculates the next bestlogical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table. Objective In this term project, you are asked to develop a program to implement Link-State Routing Protocol. Your program should have two functions:

Simulate the process of generating routing tables for each router in a given network,

Compute optional path with least cost between any two specific routers.

Problem Description Suppose we have a network with arbitrary number of routers. The network topology is given by a matrix, called the original routing table, which only indicates the costs of links between all directly connected routers. We assume each router only knows its own information and has no knowledge about others at the beginning. In this project, to implement Link-State Routing Protocol, first your program is required to create the state of the links by each router after the input file containing the network information been loaded. This is called the link state packet or LSP. Then, the program need to flood LSPs of each router to all other routers and build the routing table for each router. A Dijkstra's algorithm could be applied to find shortest path tree. Finally, your program should be able to output the routing table of any router, and output the optimal path between any two selected routers.

Program:

```
#include <stdio.h>
#include <string.h>
int main() 4. {
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]);
          }
          }
```

Output:

```
Enter the no of routers3
 Enter the cost matrix values: 0->0:
Enter the no of routers2
Enter the cost matrix values: 0->0:3
0->1:4
1->0:5
1->1:6
Enter the source router:1 1==>0:
Path taken:0<--1
```

7. Implement RSA algorithm.

Algorithms

Begin

1. Choose two prime numbers p and q.
2. Compute n = p*q.
3. Calculate phi = (p-1) * (q-1).
4. Choose an integer e such that 1 < e < phi(n) and gcd(e,phi(n)) = 1; i.e., e and phi(n) are coprime.
5. Calculate d as d ≡ e−1 (mod phi(n)); here, d is the modular multiplicative inverse of e modulo phi(n).
6. For encryption, c = me mod n, where m = original message.
7. For decryption, m = c d mod n.End

```
#include<iostream>
 #include<math.h>
 using namespace std;
// find gcd
int gcd(int a, int b)
{
int t;
while(1)
{
t= a%b; if(t==0) return b; a = b;
b= t;
}
}
```

Program:

```
int main()
{

//2 random prime numbers
double p = 13;
double q = 11;

double n=p*q;//calculate n double track;
double phi= (p-1)*(q-1);//calculate phi

//public key

//e stands for encrypt
double e=7;
//for checking that 1 < e < phi(n) and gcd(e, phi(n)) = 1; i.e., e and phi(n) are coprime.

while(e<phi) {

track = gcd(e,phi); if(track==1)

break; else
e++;

}

//private key
//d stands for decrypt
//choosing d such that it satisfies d*e = 1 mod phi double d1=1/e;
double d=fmod(d1,phi);
double message = 9;
double c = pow(message,e); //encrypt the message
double m = pow(c,d);
c=fmod(c,n);
m=fmod(m,n);
cout<<"Original Message = "<<message;
cout<<"\n"<<"p = "<<p;
cout<<"\n"<<"q = "<<q;
cout<<"\n"<<"n = pq = "<<n;
cout<<"\n"<<"phi = "<<phi;
```

```
 cout<<"\n"<<"e = "<<e;
cout<<"\n"<<"d = "<<d;
cout<<"\n"<<"Encrypted message = "<<c;
cout<<"\n"<<"Decrypted message = "<<m;
return 0;
}
}
 return 0;
```

Output File:

result.txt – it will contain our decrypted text.

TEMP FILES:

- bits.txt – it will contain our plain text converted in bits.
- cipher.txt – it will contain our encrypted text in bits.
- decrypted.txt – it will contain our decrypted text in bits (same as bits.txt in content)

Output:

8. Write a program to implement client-server application using TCP

If we are creating a connection between client and server using TCP then it has few functio ality like, TCP is suited for applications that require high reliability, and transmission time is relatively less c itical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified.

There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control an requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken do

TCP Server –

- using create(), Create TCP socket.
- using bind(), Bind the socket to server address.
- using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
- using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
- Go back to Step 3.

TCP Client –

Create TCP socket.

Connect newly created client socket to server.

•TCP Server:

•TCP Client:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
// Function designed for chat between client and server.

void func(int sockfd)
{
char buff[MAX];
int n;
// infinite loop for chat
for (;;) {
bzero(buff, MAX);
// read the message from client and copy it in buffer
read(sockfd, buff, sizeof(buff));
// print buffer which contains the client contents
printf("From client: %s\t To client : ", buff);
bzero(buff, MAX);
n = 0;
// copy server message in the buffer
while ((buff[n++] = getchar()) != '\n')
;
// and send that buffer to client
write(sockfd, buff, sizeof(buff));
// if msg contains "Exit" then server exit and chat ended.

if (strncmp("exit", buff, 4) == 0) {
printf("Server Exit...\n");
break;
}
}
}
// Driver function
```

```
int main()
{
int sockfd, connfd, len;
struct sockaddr_in servaddr, cli;
// socket create and verification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
printf("socket creation failed...\n");
exit(0);
}
else
printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
printf("socket bind failed...\n");
exit(0);
}
else
printf("Socket successfully binded..\n");
// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
printf("Listen failed...\n");
exit(0);
}
else
printf("Server listening..\n");
len = sizeof(cli);
// Accept the data packet from client and verification
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
printf("server acccept failed...\n");
exit(0);
}
else
printf("server acccept the client...\n");
// Function for chatting between client and server
func(connfd);
// After chatting close the socket
close(sockfd);
}
```

Compilation –

Server side:

```
gcc server.c -o server./server
```

Client side:

```
gcc client.c -o client./client
```

Output –

Server Side:

```
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hi
To client : hello
From client: exit
To client : exit
Server Exit...
```

Client side:

```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello

Enter the string : exit
 From Server : exit
Client: Exit
```