1.(a) IMPLEMENTATION OF DATA LINK LAYER FRAMING METHOD BIT STUFFING

AIM

To implement **BIT STUFFING** - the data link layer framing method.

ALGORITHM

1. Read the string (0's and 1's).
2. Add 0 if consecutive five bits containing 1's.
3. Add the flag bits 01111110 in starting and end of the input string.
4. Print the stuffing string.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{
 char ch, arr[50], fs[50]="", t[50]="";
 int count=0, i=0, j=0;
 clrscr();
 printf("Enter the bit srting : \t");
 scanf("%s",arr);
 strcat(fs,"01111110-");
 if(strlen(arr)<5)
 {
  strcat(fs,arr);
  strcat(fs,"-01111110");
  printf("The bits after stuffing are \n %s",fs);
  }
 else
 {
  while((ch=arr[j]) !='\0')
  {
   if(ch=='1')
          ++count;
```

```
 else
        count=0;
 t[i++]=ch;
 if(count==5)
 {
  t[i++]='0';
  count=0;
  }
 j++;
}
t[i]='\0';
strcat(t,"-01111110");
strcat(fs,t);
printf("The bits after bit stuffing are\n %s", fs);
}
getch();
return 0;
}
```

SAMPLE OUTPUT -1

Enter the bit srting : 111111
The bits after bit stuffing are
01111110-1111101-01111110

SAMPLE OUTPUT -2

Enter the bit srting : 11111111111111111111
The bits after bit stuffing are
01111110-11111011111011111011111011-01111110

SAMPLE OUTPUT -3

Enter the bit srting : 111
The bits after stuffing are
01111110-111-01111110

RESULT

Thus the program was executed and verified successfully for **BIT STUFFING**.

1.(b) IMPLEMENTATION OF DATA LINK LAYER FRAMING METHOD CHARACTER (OR) BYTE STUFFING

AIM

To implement **BYTE STUFFING** or **CHARACTER STUFFING** - the data link layer framing method.

ALGORITHM

1. Read the string (Character string).
2. Convert the input string into capital letters by using ASCII.
3. Add the flag DLESTX (Start of Text) at starting and DLEETX (End of Text) at end of the input string.
4. Add DLE if the input string contains DLE.
5. Print the string after stuffing.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{
 char sdel[]="DLESTX", data[50]="", sdata[50]="";
 int i=0, j=0, k;
 clrscr();
 printf("Enter the message \t\t: ");
 scanf("%s",data);
 for ( k=0; data [k]!='\0'; k++)
   if(data[k]>='a' && data[k]<='z')
      data[k]=data[k]-32;

 printf("Original Message is \t: %s \n",data);
 if(strlen(data)<3)
 {
 strcat(sdel, data);
 strcat(sdel,"DLEETX");
 printf("Message after character stuffing is \t: %s",sdel);
 }
```

```
else
{
 while(data[i] != '\0')
 {
  if(data[i] == 'D' && data[i+1] == 'L' && data[i+2] == 'E')
  {
   strcat(sdata, "DLEDLE");
   i=i+3;
   j=j+6;
   continue;
   }
  sdata[j]=data[i];
  j++;
  i++;
 }
 strcat(sdel, sdata);
 strcat(sdel,"DLEETX");
 printf("Message after stuffing is \t : %s",sdel);
 }
getch();
return 0;
}
```

SAMPLE OUTPUT -1

Enter the message : hai
Original Message is : HAI
Message after stuffing is : DLESTXHAIDLEETX

SAMPLE OUTPUT -2

Enter the message : adle
Original Message is : ADLE
Message after stuffing is : DLESTXADLEDLEDLEETX

SAMPLE OUTPUT -3

Enter the message : adlestxadleetx
Original Message is : ADLESTXADLEETX
Message after stuffing is : DLESTXADLEDLESTXADLEDLEETXDLEETX

RESULT

Thus the program was executed and verified successfully for **CHARACTER STUFFING**.

2. COMPUTATION OF CRC CODE FOR THE POLYNOMIALS CRC-12, CRC-16 and CRC CCIP

AIM

To write a program to compute **CRC code for the polynomials** – CRC-12, CRC-16, CRC-CCIP.

DESCRIPTION

1. **Calculation of CRC at Sender Side**
   - A string of n 0's is appended to the data unit to be transmitted.
   - Here, n is one less than the number of bits in CRC generator.
   - Binary division is performed of the resultant string with the CRC generator.
   - After division, the remainder so obtained is called as **CRC**.
   - It may be noted that CRC also consists of n bits.
2. **Appending CRC to Data Unit**
   - The CRC is obtained after the binary division.
   - The string of n 0's appended to the data unit earlier is replaced by the CRC remainder.
3. **Transmission to Receiver**
   - The newly formed code word (Original data + CRC) is transmitted to the receiver.
4. **Checking at Receiver Side**
   - The transmitted code word is received.
   - The received code word is divided with the same CRC generator.
   - On division, the remainder so obtained is checked.

The following two cases are possible now.

Case-1: Remainder = 0

- Receiver assumes that no error occurred in the data during the transmission.
- Receiver accepts the data.

Case-02: Remainder ≠ 0

- Receiver assumes that some error occurred in the data during the transmission.
- Receiver rejects the data and asks the sender for retransmission.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main(void)
{
 int data[50],div[16],rem[16];
 int datalen, divlen, i,j,k;
 int ch;
 clrscr();
 printf("Enter the data: ");
 i = 0;
 while((ch = fgetc(stdin)) != '\n')
 {
  if(ch == '1')
    data[i] = 1;
  else
    data[i] = 0;
  i++;
  }
datalen = i;
printf("\nEnter the divisor: ");
i = 0;
while((ch = fgetc(stdin)) != '\n')
{
 if(ch == '1')
   div[i] = 1;
 else
   div[i] = 0;
 i++;
 }
divlen = i;
for(i = datalen ; i < datalen + divlen - 1 ; i++)
   data[i] = 0;
datalen = datalen + divlen - 1;
for(i = 0 ; i < divlen ; i++)
   rem[i] = data[i];
k = divlen-1;
while(k < datalen)
   if(rem[0] == 1)
   {
    for(i = 0 ; i < divlen ; i++)
       rem[i] = rem[i] ^ div[i];
```

```
  }
 else
 {
  if(k == datalen-1)
     break;
  for(i = 0 ; i < divlen-1 ; i++)
  {
   rem[i] = rem[i+1];
   printf("%d",rem[i]);
  }
  rem[i] = data[++k];
  printf("%d\n",rem[i]);
 }

j=1;
for(i = datalen - divlen + 1 ; i < datalen ; i++)
     data[i] = rem[j++];
printf("\nThe data to be sent is\n");
for(i = 0 ; i < datalen ; i++)
     printf("%d",data[i]);
getch();
return 0;
}
```

SAMPLE OUTPUT

```
Enter the data: 10101011
Enter the divisor: 101
000
001
010
101
001
010
100

The data to be sent is
1010101101
```

RESULT

Thus the program was executed and verified successfully for **CRC code for the**

polynomials.

3.(a) FLOW CONTROL USING THE SLIDING WINDOW PROTOCOL

AIM

To develop a simple date link layer that performs the **flow control using the sliding window protocol**.

DESCRIPTION

1. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer data is in the form of frames.
2. Both sender and receiver agrees on some window size. If window size=w then after sending w frames sender waits for the acknowledgement (ack) of the first frame.
3. As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender.
4. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int w,i,f,frames[50];
   clrscr();
   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");
  getch();
  return 0;
 }
```

SAMPLE OUTPUT

Enter window size: 3
Enter number of frames to transmit: 5
Enter 5 frames: 21
25
45
65
29

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

21 25 45
Acknowledgement of above frames sent is received by sender

65 29
Acknowledgement of above frames sent is received by sender

RESULT

Thus the program was executed and verified successfully for **flow control using the sliding window protocol**.

3.(b) LOSS RECOVERY USING THE GO-BACK-N MECHANISM

AIM

To develop a simple date link layer that performs the **loss recovery using the Go-Back-N Mechanism**.

DESCRIPTION

1. The sender can transmit N frames before receiving the ACK frame.
2. The size of the send sliding window is N.
3. The copy of sent data is maintained in the sent buffer of the sender until all the sent packets are acknowledged.
4. If the timeout timer runs out then the sender will resend all the packets.
5. Once the data get acknowledged by the receiver then that particular data will be removed from the buffer.
6. Whenever a valid acknowledgement arrives then the send window can slide one or more slots.

PROGRAM

```
#include<stdio.h>
#include<contio.h>
int main()
{
 int windowsize, sent=0, ack, i;
 clrscr();
 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 \t");
        scanf("%d",&ack);
```

```
if(ack == windowsize)
        {
                printf("All the frames transmitted successfully ");
                break;
              }
        else
                sent = ack;
        }
 getch();
 return 0;
 }
```

SAMPLE OUTPUT

```
enter window size
5
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.

Please enter the last Acknowledgement received      4
Frame 4 has been transmitted.

Please enter the last Acknowledgement received      5
All the frames transmitted successfully
```

RESULT

Thus the program was executed and verified successfully for **Loss recovery using the go-back-n mechanism**.

4. DIJKSTRA'S ALGORITHM TO COMPUTE SHORTEST PATH THROUGH A NETWORK

AIM

To implement Dijkstra's algorithm to compute shortest path through a network.

DESCRIPTION

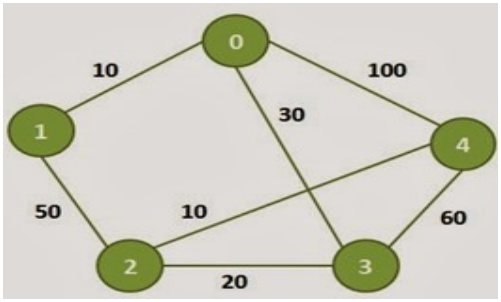
1. Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
2. The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
3. Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
4. The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

PROGRAM

```
#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;
 clrscr();
 printf("Enter no. of vertices :\t");
 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 :\t");
 scanf("%d",&u);
 dijkstra(G,n,u);
 getch();
 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, i, j;
 //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("\n Distance of node%d = %d", i, distance[i]);
   printf("\n\t Path = %d",i);
   j = i;
   do
   {
     j = pred[j];
     printf(" <-- %d",j);
   }while(j != startnode);
  }
}
```



SAMPLE OUTPUT

```
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 node1 = 10
        Path = 1 <-- 0
 Distance of node2 = 50
        Path = 2 <-- 3 <-- 0
 Distance of node3 = 30
        Path = 3 <-- 0
 Distance of node4 = 60
        Path = 4 <-- 2 <-- 3 <-- 0
```

RESULT

Thus the program was executed and verified successfully for shortest path through a network using Dijkstra's algorithm.

5. BROADCAST TREE FOR THE SUBNET

AIM

To obtain broadcast tree for the subnet of host.

DESCRIPTION

- A router creates a data packet and then sends it to each host one by one. In this case, the router creates multiple copies of single data packet with different destination addresses.
- All packets are sent as unicast but because they are sent to all, it simulates as if router is broadcasting.
- This method consumes lots of bandwidth and router must destination address of each node.
- Secondly, when router receives a packet that is to be broadcasted, it simply floods those packets out of all interfaces. All routers are configured in the same way.
- This method is easy on router's CPU but may cause the problem of duplicate packets received from peer routers.

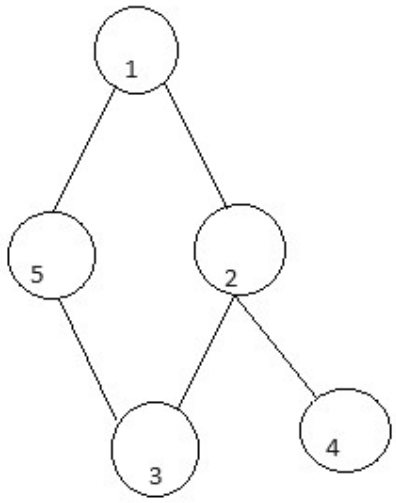
PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n;

void main()
{
 int i,j,root;
 clrscr();
 printf("Enter no.of nodes : ");
 scanf("%d",&n);
 printf("Enter adjacent matrix \n");
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
 {
        printf("Enter connecting of %d --> %d :: ",i,j);
        scanf("%d",&a[i][j]);
 }

 printf("Enter root node : ");
 scanf("%d",&root);
 adj(root);
 getch();
 }
```

```
adj(int k)
{
 int i,j;
 printf("Adjacent node of root node ::\n");
 printf("%d\n",k);
 for(j=1;j<=n;j++)
{
  if(a[k][j]==1 || a[j][k]==1)
          printf("%d \t",j);
 }
printf("\n");
for(i=1;i<=n;i++)
{
  if((a[k][j]==0) && (a[i][k]==0) && (i!=k))
          printf("%d \t",i);
 }
}
```

SAMPLE OUTPUT

```
Enter no.of nodes : 5
Enter adjacent matrix
Enter connecting of 1 --> 1 :: 0
Enter connecting of 1 --> 2 :: 1
Enter connecting of 1 --> 3 :: 0
Enter connecting of 1 --> 4 :: 0
Enter connecting of 1 --> 5 :: 1
Enter connecting of 2 --> 1 :: 1
Enter connecting of 2 --> 2 :: 0
Enter connecting of 2 --> 3 :: 1
Enter connecting of 2 --> 4 :: 1
Enter connecting of 2 --> 5 :: 0
Enter connecting of 3 --> 1 :: 0
Enter connecting of 3 --> 2 :: 1
Enter connecting of 3 --> 3 :: 0
Enter connecting of 3 --> 4 :: 0
Enter connecting of 3 --> 5 :: 1
Enter connecting of 4 --> 1 :: 0
Enter connecting of 4 --> 2 :: 1
Enter connecting of 4 --> 3 :: 0
Enter connecting of 4 --> 4 :: 0
Enter connecting of 4 --> 5 :: 0
Enter connecting of 5 --> 1 :: 1
Enter connecting of 5 --> 2 :: 0
Enter connecting of 5 --> 3 :: 1
Enter connecting of 5 --> 4 :: 0
Enter connecting of 5 --> 5 :: 0
Enter root node : 2
Adjacent node of root node ::
2
1       3       4
5
```



RESULT

Thus the program was executed and verified successfully for obtaining broadcast tree for the subnet of host.

6. IMPLEMENTATION OF DISTANCE VECTOR ROUTING ALGORITHM FOR OBTAINING ROUTING TABLES AT EACH NODE

AIM

To implement the distance vector routing algorithm for obtaining routing tables at each node.

DESCRIPTION

Let $d_x(y)$ be the cost of the least-cost path from node x to node y. The least costs are related by Bellman-Ford equation,

$$d_x(y) = \min_v\{c(x,v) + d_v(y)\}$$

Where the minv is the equation taken for all x neighbors. After traveling from x to v, if we consider the least-cost path from v to y, the path cost will be $c(x,v)+d_v(y)$. The least cost from x to y is the minimum of $c(x,v)+d_v(y)$ taken over all neighbors.

With the Distance Vector Routing algorithm, the node x contains the following routing information:

- For each neighbor v, the cost c(x,v) is the path cost from x to directly attached neighbor, v.
- The distance vector x, i.e., $D_x = [\ D_x(y) : y \text{ in } N\ ]$, containing its cost to all destinations, y, in N.
- The distance vector of each of its neighbors, i.e., $D_v = [\ D_v(y) : y \text{ in } N\ ]$ for each neighbor v of x.

Distance vector routing is an asynchronous algorithm in which node x sends the copy of its distance vector to all its neighbors. When node x receives the new distance vector from one of its neighboring vector, v, it saves the distance vector of v and uses the Bellman-Ford equation to update its own distance vector. The equation is given below:

$$d_x(y) = \min_v\{\ c(x,v) + d_v(y)\} \quad \text{for each node y in N}$$

The node x has updated its own distance vector table by using the above equation and sends its updated table to all its neighbors so that they can update their own distance vectors.

PROGRAM

```
#include<stdio.h>
#include<conio.h>

struct node
{
 unsigned dist[20];
 unsigned from[20];
 }rt[10];

int main()
{
 int dmat[20][20];
 int n,i,j,k,count=0;
 clrscr();
 printf("\nEnter the number of nodes : ");
 scanf("%d",&n);
 printf("\nEnter the cost matrix :\n");
 for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 {
  scanf("%d",&dmat[i][j]); dmat[i][i]=0;
  rt[i].dist[j]=dmat[i][j]; rt[i].from[j]=j;
  }

 do
 {
  count=0;
  for(i=0;i<n;i++)
  for(j=0;j<n;j++)
  for(k=0;k<n;k++)
          if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
          {
           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<n;i++)
{
 printf("\n\nState value for router %d is \n",i+1);
 for(j=0;j<n;j++)
 {
  printf("\t\nnode %d via %d Distance is %d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
  }
 }
printf("\n\n");
getch();
return 1;
}
```

SAMPLE OUTPUT

```
Enter the number of nodes : 3
Enter the cost matrix :
0   2   4
2   0   5
4   5   0
State value for router 1 is

node 1 via 1 Distance is 0
node 2 via 2 Distance is 2
node 3 via 3 Distance is 4

State value for router 2 is

node 1 via 1 Distance is 2
node 2 via 2 Distance is 0
node 3 via 3 Distance is 5

State value for router 3 is

node 1 via 1 Distance is 4
node 2 via 2 Distance is 5
node 3 via 3 Distance is 0
```

RESULT

Thus the program was executed and verified successfully for distance vector routing algorithm for obtaining routing tables at each node.