### NAME OF THE EXPERIMENT: Bit Stuffing.

**OBJECTIVE**: Implement the data link layer framing method.

**RESOURCE:** gcc

### PROGRAM LOGIC:

The new technique allows data frames to contain an arbitrary number if bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with special bit pattern, 01111110, called a flag byte. Whenever the s e n d e r ’ s d a t a l i n k l a y e r e n c o u n t e r s f i v e c o n s e c u t i v e o n e s i n t h e d a t a , i t a u toma t ica l ly stu f fs a 0 bit into th e ou t goi ng bit strea m. T h i s bit s tu f fing i s a na logou s to c ha ra cter stu f fin g, in w hic h a D LE i s stu f fe d into the ou t goin g character stream before DLE in the data.

### SOURCE CODE:

// BIT Stuffing 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];

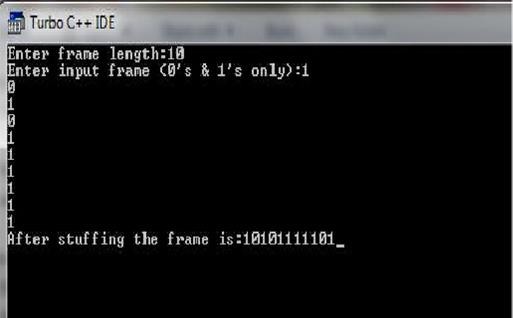
} i++; j++;

}

printf("After stuffing the frame is:"); for(i=0;i<j;i++)

printf("%d",b[i]);

} OUTPUT:



**NAME OF THE EXPERIMENT:** Character Stuffing. **OBJECTIVE:** Implement the data link layer framing methods. **RESOURCE:** gcc

### PROGRAM LOGIC:

The framing method gets around the problem of resynchronization after an error by ha vi ng ea ch fra me sta r t with th e ASCII c ha ra cter s equ en ce D L E ST X a nd 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 la yer on t he recei vin g en d re mo ve s th e D LE b efore th e da ta a re giv en t o th e network layer. This technique is called character stuffing.

### SOURCE CODE:

//PROGRAM FOR CHARACTER STUFFING

#include<stdio.h> #include<string.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';

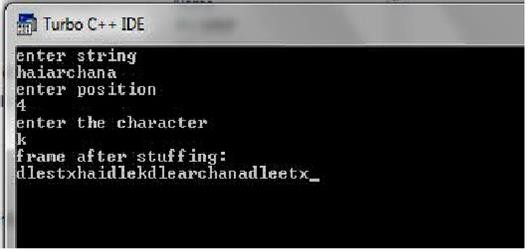
b[j+5]='x';

b[j+6]='\0';

printf("\nframe after stuffing:\n"); printf("%s",b);

}

OUTPUT:



**NAME OF THE EXPERIMENT:** Cyclic Redundancy Check.

**OBJECTIVE:** Implement on a data set of characters the three CRC polynomials – CRC 12,CRC 16 and CRC CCIP.

**RESOURCE:** gcc

### PROGRAM LOGIC:

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 ha v e th e c orrect pa r i ty tha t i s 0 . 5 . so th e proba bilit y of a ba d blo ck bei ng expected when it should not be 2 power(-n). This scheme sometimes known as Cyclic Redundancy Code

### SOURCE CODE:

//PROGRAM FOR CYCLIC REDUNDENCY 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++;

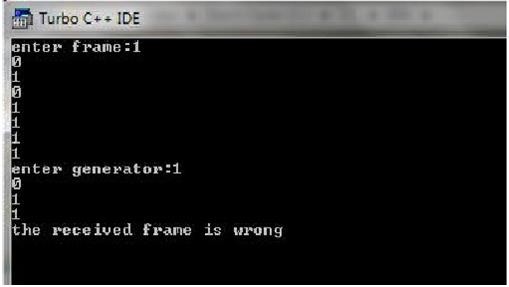
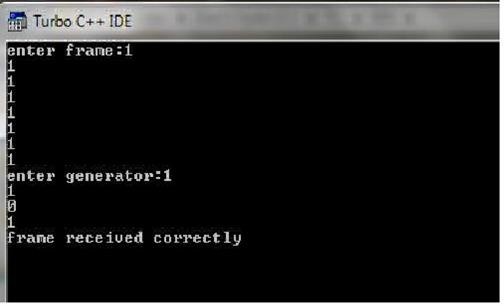
}

}

}

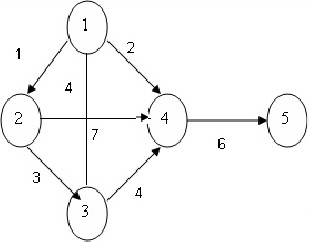
}

OUTPUT:



**NAME OF THE EXPERIMENT:** Shortest Path.

**OBJECTIVE:** Implement Dijkstra‘ s algorithm to compute the Shortest path thru a given graph.



**RESOURCE:** gcc

**Program Logic: Dijkstra's algorithm** is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree.

### SOURCE CODE:

// .PROGRAM FOR FINDING SHORTEST PATH FOR A GIVEN GRAPH//

#include<stdio.h> void main()

{

int path[5][5],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index;

printf("enter the cost matrix\n"); for(i=1;i<=5;i++) for(j=1;j<=5;j++) scanf("%d",&a[i][j]); printf("enter the paths\n"); scanf("%d",&p);

printf("enter possible paths\n"); for(i=1;i<=p;i++) for(j=1;j<=5;j++) scanf("%d",&path[i][j]);

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

{ t[i]=0;

stp=st; for(j=1;j<=5;j++)

{

edp=path[i][j+1]; t[i]=t[i]+a[stp][edp]; if(edp==ed)

break; else stp=edp;

}

}min=t[st]; index=st; for(i=1;i<=p;i++)

{

if(min>t[i])

{

min=t[i]; index=i;

}

}

printf("minimum cost %d",min); printf("\n minimum cost path "); for(i=1;i<=5;i++)

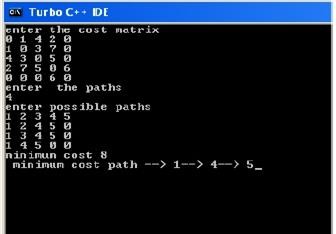
{

printf("--> %d",path[index][i]); if(path[index][i]==ed)

break;

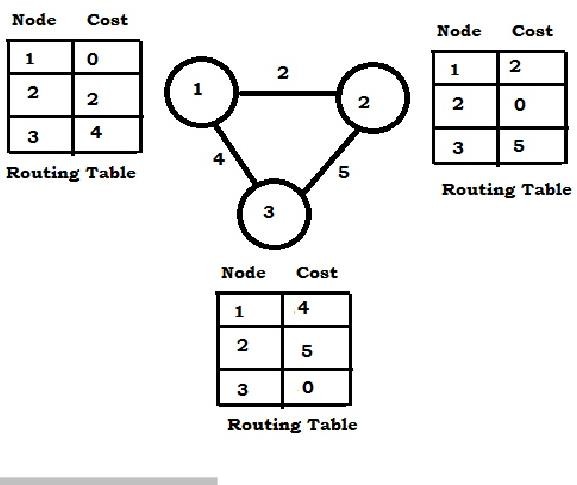
}

Output:



**NAME OF THE EXPERIMENT:** Distance Vector routing.

**OBJECTIVE:** Obtain Routing table at each node using distance vector routing algorithm for a given subnet.



**RESOURCE:** gcc

### PROGRAM LOGIC:

Distance Vector Routing Algorithms calculate a best route to reach a destination based solely on distance. E.g. RIP. RIP calculates the reach ability based on hop count. It’ s different from link state algorithms which consider some other factors like bandwidth and other metrics to reach a destination. Distance vector routing algorithms are not preferable for complex networks and take longer to converge.

### SOURCE CODE:

#include<stdio.h> struct node

{

unsigned dist[20]; unsigned from[20];

}rt[10];

int main()

{

int dmat[20][20]; int n,i,j,k,count=0;

printf("\nEnter the number of nodes : "); scanf("%d",&n);printf("Enter 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("\nState value for router %d is \n",i+1); for(j=0;j<n;j++)

{

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

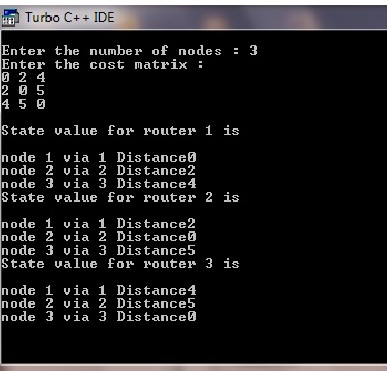
}

}

printf(“\n”);

}

Output:



**NAME OF THE EXPERIMENT:** Broadcast Tree.

**OBJECTIVE:** Implement broadcast tree for a given subnet of hosts

**RESOURCE:** gcc

### PROGRAM LOGIC:

This technique is widely used because it is simple and easy to understand. The idea o f this a lgorith m i s t o bu i ld a gra ph o f th e su bnet with ea c h n ode o f t he gra ph representing a router and each arc of the graph representing a communication line. T o ch oo s e a rou te betw ee n a giv en pa i r o f rou ter s th e a lgorith m ju st f ind s the broadcast between them on the graph.

### SOURCE CODE:

// Write a ‘c’ program for Broadcast tree from subnet of host #include<stdio.h>

int p,q,u,v,n;

int min=99,mincost=0; int t[50][2],i,j;

int parent[50],edge[50][50]; main()

{

printf("\n Enter the number of nodes"); scanf("%d",&n);

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

{

printf("%c\t",65+i); parent[i]=-1;

}

printf("\n"); for(i=0;i<n;i++)

{

printf("%c",65+i); for(j=0;j<n;j++) scanf("%d",&edge[i][j]);

}

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

{

for(j=0;j<n;j++) if(edge[i][j]!=99)

if(min>edge[i][j])

{

min=edge[i][j]; u=i;

v=j;

}

p=find(u); q=find(v); if(p!=q)

{ t[i][0]=u;

t[i][1]=v; mincost=mincost+edge[u][v]; union(p,q);

}

else

{

t[i][0]=-1;t[i][1]=-1;

}

min=99;

}

printf("Minimum cost is %d\n Minimum spanning tree is\n" ,mincost); for(i=0;i<n;i++)

if(t[i][0]!=-1 && t[i][1]!=-1)

{

printf("%c %c %d", 65+t[i][0],65+t[i][1],edge[t[i][0]][t[i][1]]);

printf("\n");

}

union(int l,int m)

{

parent[l]=m;

}

find(int l)

{

if(parent[l]>0)

i=parent[i]; return i;

}

Output:

