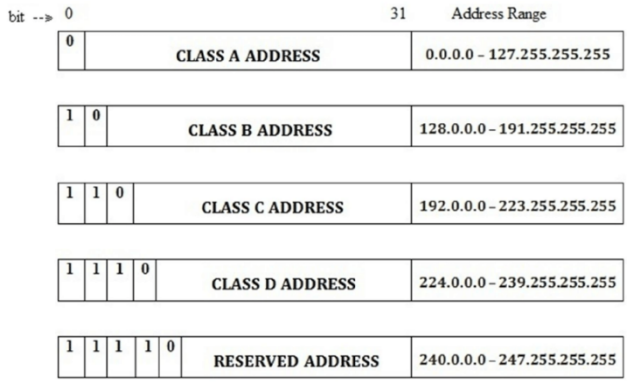
# *Program 1:*
# *Program to determine class, Network and Host ID of an IPv4 address*

AIM: Given a valid IPv4 address in the form of string and it follows Class Full addressing. The task is to determine class of given IPv4 address as well as separate Network and Host ID part from it.

**Approach**

1. **For determining the class:** The idea is to check first octet of IP address. As we know, for class **A** first octet will range from **1 – 126**, for class **B** first octet will range from **128 – 191**, for class **C** first octet will range from **192- 223**, for class **D** first octet will range from **224 – 239**, for class **E** first octet will range from **240 – 255**.

2. **For determining the Network and Host ID:** Network Id for Class **A** is **8bit(first Octat)**, for Class **B** is **16** bits(first and second Octat)and for Class **C** is **24 bits(first, second and third octat)** whereas Class **D** and **E** is not divided into Network and Host ID.




## Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main()
{
   char Ipadd[20],temp[20],NetId[10],HostId[20],temp1[20];
   int FByte,i,j,dot_count;
   printf("Enter IP address in format X.Y.Z.A\n");
   scanf("%s",Ipadd);
   i=0;
   while(Ipadd[i]!='.')
   {
      temp[i]=Ipadd[i];
      i++;
   }
   temp[i]='\0';
   FByte=atoi(temp);
   if(FByte>=0 && FByte<128)
   {
```

```
    printf("Class A\n");
    printf("Subnet Mask is 255.0.0.0\n");
    i=0,j=0;
    while(Ipadd[i]!='.')
    {
       NetId[j]=Ipadd[i];
       i++,j++;
    }

    NetId[j]='\0';
    strcat(NetId,".0.0.0");
    j=0,i++;
    while(Ipadd[i]!='\0')
    {
       temp1[j]=Ipadd[i];
       i++,j++;
    }
    temp1[j]='\0';
    strcpy(HostId,"0.");
    strcat(HostId,temp1);
    printf("Network ID is %s\n",NetId);
    printf("Host ID is %s\n", HostId);
}
else if(FByte>=128 && FByte<192)
{
    printf("Class B\n");
    printf("Subnet Mask is 255.255.0.0\n");
    i=0,j=0;

    while(Ipadd[i]!='\0')
    {

       while(dot_count<=1)
       {
          NetId[j]=Ipadd[i];
          if(Ipadd[i]=='.')
          {
             dot_count++;
          }
          i++,j++;
       }
       NetId[j]='\0';
       strcat(NetId,"0.0");
       j=0;
       while(Ipadd[i]!='\0')
       {
          temp1[j]=Ipadd[i];
          i++,j++;
       }
       temp1[j]='\0';
       strcpy(HostId,"0.0.");
       strcat(HostId,temp1);
    }
    printf("Network ID is %s\n",NetId);
    printf("Host ID is %s\n", HostId);
```

```
    }
    else if(FByte>=192 && FByte<224)
    {
        printf("Class C\n");
        printf("Subnet Mask is 255.255.255.0\n");
        i=0,j=0;

        while(Ipadd[i]!='\0')
        {

            while(dot_count<=2)
            {
                NetId[j]=Ipadd[i];
                if(Ipadd[i]=='.')
                {
                    dot_count++;
                }
                i++,j++;
            }
            NetId[j]='\0';
            strcat(NetId,"0");
            j=0;
            while(Ipadd[i]!='\0')
            {
              temp1[j]=Ipadd[i];
                i++,j++;
            }
            temp1[j]='\0';
            printf("temp1 is %s\n", temp1);
            strcpy(HostId,"0.0.0.");
            strcat(HostId,temp1);
        }
        printf("Network ID is %s\n",NetId);
        printf("Host ID is %s\n", HostId);
    }
     else if(FByte>=224 && FByte<240)
    {
        printf("Class D");
        printf("Subnet Mask is 255.255.255.255");

    }
    else if(FByte>=240 && FByte<=255)
    {
        printf("Class E");
    }
    else
    {
        printf("Invalid IP Address\n");
    }
}
```

Output:

**Input :** 1.4.5.5
**Output :**

```
Given IP address belongs to Class A
Subnet Mask is 255.0.0.0
```

Network ID is 1.0.0.0
Host ID is 0.4.5.5

**Input :** 130.45.151.154
**Output :**
Given IP address belongs to Class B
Subnet Mask is 255.255.0.0
Network ID is 130.45.0.0
Host ID is 0.0.151.154

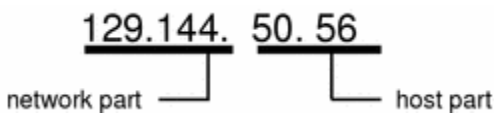
Viva Questions

**1. What is an IP Address?**
**Ans:** An IP address is a unique address that identifies a device on the internet or a local network. IP address is used by IP (Internet Protocol) at network layer for delivering data packet between two end devices.

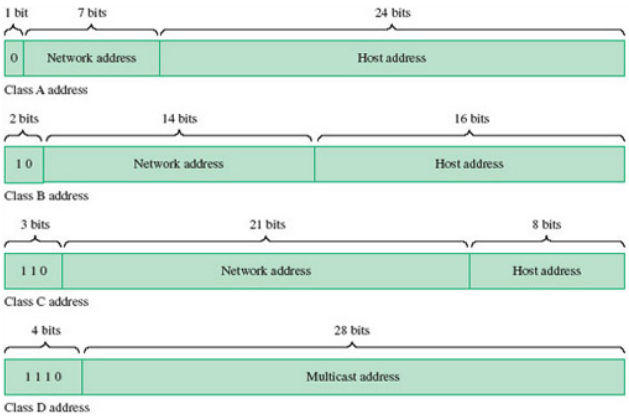
**2. How assigns IP address?**
Ans. Allocated by the Internet Assigned Numbers Authority (IANA), a division of the Internet Corporation for Assigned Names and Numbers (ICANN).

**3. What is the size of IPv4 address?**
Ans. The size IPv4 address is a **32-bit or 4 bytes**. An IPv4 address is typically written in decimal digits, formatted as four 8-bit fields that are separated by periods. Each 8-bit field represents a byte of the IPv4 address. This form of representing the bytes of an IPv4 address is often referred to as **the dotted-decimal format.**

**4. What are the parts of IPv4 address?**
IPv4 addresses consist of three parts: a prefix identifying the class of address, the address of a network, and the address of a host within the network.




**Network Part:** The network part specifies the unique number that is assigned to your network.
**Host Part:** This is the part of the IPv4 address that you assign to each host. The host part uniquely identifies this machine on your network. Note that for each host on your network, the network part of the address is the same, but the host part must be different.



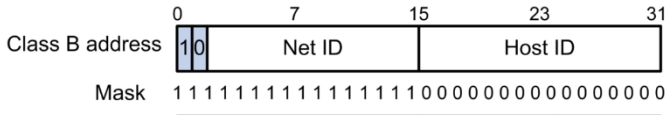

**5. What is Classful Addressing in IPv4?**

Ans. IPv4 addressing, at its inception, used the concept of classes. This architecture is called classful addressing. In classful addressing, the address space is divided into five classes: A, B, C, D,and E. Each class occupies some part of the address space

6. What is a Masking?

Ans: Masking identifies the boundary between the host ID and the combination of net ID and subnet ID. Each subnet mask comprises 32 bits that correspond to the bits in an IP address. In a subnet mask, the consecutive ones represent the net ID and consecutive zeros represent the host ID. Class A, B, and C networks use these default masks (also called natural masks): 255.0.0.0, 255.255.0.0, and 255.255.255.0 respectively.

Class B network




7. What is the address space of IPv4?

Ans: An address space is the total number of addresses used by the protocol. If a protocol uses $N$ bits to define an address, the address space is $2^N$ because each bit can have two different values (0 or 1) and $N$ bits can have $2^N$ values. IPv4 uses 32-bit addresses, which means that the address space is $2^{32}$ or 4,294,967,296 (more than 4 billion

# *Program 2 : C Program to simulate working of ARP and RARP protocol.*

**AIM:** To write a C program to simulate working of ARP and RARP protocol.

## Description :

**ARP and RARP** are protocols of the network layer. When an IP datagram needs to be sent between one host to another, the sender would require the receiver's physical address as well as its logical address.

The term ARP is an abbreviation for Address Resolution Protocol. The ARP maps the node's IP address (32-bit logical address) to the MAC address/physical address (48-bit address).

The term RARP is an abbreviation for Reverse ARP. The RARP is also a protocol of the network layer. The RARP maps the 48-bit address (MAC address/physical address) to the logical IP address (32-bit).

## Program:

```
#include<stdio.h>
#include<string.h>
void 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;

        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 for mapping : ");
                                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(MAC 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:

                return ;
        }
    }
}
```

# *Program 3: C Program to implement the data link layer character stuffing framing methods .*

Objective:

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagrams passed down by above layers and convert them into frames ready for transfer. This is called Framing.

Overview:

Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is upto the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters. The three framing methods that are widely used are

- Character count
- Starting and ending characters, with character stuffing
- Starting and ending flags, with bit stuffing

***Character Count***

This method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count,it knows how many characters follow, and hence where the end of the frame is. The disadvantage is that if the count is garbled by a transmission error, the destination will lose synchronization and will be unable to locate the start of the next frame. So, this method is rarely used.

## AIM: To write a C Program to implement the data link layer framing Character stuffing methods

***Character stuffing***

- In the second method, each frame starts and Ends with the ASCII character FLAG.
- This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for FLAG characters.
- If however, binary data is being transmitted then there exists a possibility of the characters FLAG occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII ESC character just before the FLAG character in the data.

- The receiver's data link layer removes this ESC before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

**Algorithm:**

Step1: Define FLAG and ECS character

Step2 : Rread the message

Step3: Scan the message looking for FLAG and ESC character in the message

If present insert ESC character

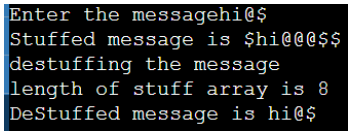
Step4: For destuffing scan the received message looking for ESC character in the message if present remove and read the next character.

**Program:**

```
#include <stdio.h>
#include<string.h>
#define FLAG '$'
#define ESC '@'
int main()
{
   char data[25],stuff[50],dstuff[50];
   int length,i,j;
   printf("Enter the message");
   scanf("%s",data);
   length=strlen(data);
   stuff[0]=FLAG;
   for(i=0,j=1;data[i]!='\0';i++)
   {
      if(data[i]!=FLAG && data[i]!=ESC)
      {
         stuff[j++]=data[i];
      }
      if(data[i]==FLAG || data[i]==ESC)
      {
         stuff[j++]=ESC;
         stuff[j++]=data[i];
      }
   }
   stuff[j++]=FLAG;
```

```
    stuff[j]='\0';
    printf("Stuffed message is %s\n",stuff);
    printf("destuffing the message\n");
    length=strlen(stuff);
    printf("length of stuff array is %d\n",length);
    for(i=1,j=0;stuff[i]!='\0';i++)
    {
       if(stuff[i]!=FLAG && stuff[i]!=ESC)
       {
          dstuff[j++]=stuff[i];
       }
       if(stuff[i]==ESC)
       {
          i++;
          dstuff[j++]=stuff[i];
       }
    }
    dstuff[j]='\0';
    printf("DeStuffed message is %s\n",dstuff);
    return 0;
}
```

OUTPUT:



# *Program 4: C Program to implement the data link layer bit stuffing framing methods.*

## AIM: To write a C Program to implement the data link layer framing Bit stuffing methods

***Bit stuffing***

- The third method allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.
- At the start and end of each frame is a flag byte consisting of the special bit pattern 01111110 .
- Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a zero bit into the outgoing bit stream. This technique is called bit stuffing.
- When the receiver sees five consecutive 1s in the incoming data stream, followed by a zero bit, it automatically destuffs the 0 bit.
- The boundary between two frames can be determined by locating the flag pattern.

## Program for Bit Stuffing.

```
#include<stdio.h>
#include<string.h>
char a[50];
char b[50];
char flag[8] = "01111110";

int main()
{
        int i = 0, j = 8, n, k;
          printf("Enter the message:\n");
        gets(a);
        strcpy(b, flag);

        n = strlen(a);
        for (k = 0; k < n; k++)
        {
          if (a[k] == '0' &&a[k + 1] == '1' && a[k + 2] == '1' && a[k + 3] == '1' && a[k
+ 4] == '1' && a[k + 5] == '1')
                    {
                              b[j] = '0';
                              b[j + 1] = '1';
                              b[j + 2] = '1';
                              b[j + 3] = '1';
                              b[j + 4] = '1';
                              b[j + 5] = '1';
                              b[j + 6] = '0';
```

```
                    i = i + 6;
                j = j + 7;
            }
            b[j++] = a[i++];
            }
        strcat(b, flag);
        printf("\nMessage after bit Stuffing");
        puts(b);
    }
```

**Output:**

Enter the message:
10111111101

Message after Bit Stuffing:
0111111010111110110101111110

## VIVA Question

**1. What are the design issues of Data Link Layer?**

**Ans:** The following are the design issues in the Data Link Layer −

- The services that are provided to the Network layer.
- Framing
- Error control
- Flow control

**2.What are the different types of services provide to network layer?**

The types of services provided can be of three types −

- Unacknowledged connectionless service
- Acknowledged connectionless service
- Acknowledged connection - oriented service

**3. What is a frame?**

Ans. The data link layer encapsulates each data packet from the network layer into frames that are then transmitted. A frame has three parts, namely −

- Frame Header
- Payload field that contains the data packet from network layer
- Trailer

**4. What is Flow Control?**

Ans .The data link layer regulates flow control so that a fast sender does not drown a slow receiver. When the sender sends frames at very high speeds, a slow receiver may not be able to handle it. There will be frame losses even if the transmission is error-free. The two common approaches for flow control are

# *Program to implement on a data set of characters the three CRC polynomials CRC 12, CRC 16 and CRC CCIP .*

Description:

- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

- **Algorithm for Encoding using CRC**
  - The communicating parties agrees upon the size of message,*M(x)* and the generator polynomial, *G(x)*.
  - If *r* is the order of *G(x)*,*r*, bits are appended to the low order end of *M(x)*. This makes the block size bits, the value of which is $x^rM(x)$.
  - The block $x^rM(x)$ is divided by *G(x)* using modulo 2 division.
  - The remainder after division is added to $x^rM(x)$ using modulo 2 addition. The result is the frame to be transmitted, *T(x)*. The encoding procedure makes exactly divisible by *G(x)*.
- **Algorithm for Decoding using CRC**
  - The receiver divides the incoming data frame T(x) unit by G(x) using modulo 2 division. Mathematically, if E(x) is the error, then modulo 2 division of [*M(x)* + *E(x)*] by *G(x)* is done.
  - If there is no remainder, then it implies that *E(x)*. The data frame is accepted.
  - A remainder indicates a non-zero value of E(x), or in other words presence of an error. So the data frame is rejected. The receiver may then send an erroneous acknowledgment back to the sender for retransmission.

**Generator Polynomial G normally as you choose a prime number that begins and ends with 1, some examples are:**

**1. 1001**

**2. 11101**

**3. 1100000001111 (CRC12)**

**4. 11000000000000101 (CRC16)**

## Main Polynomials

- **CRC-12** : $X^{12} + X^{11} + X^3 + X^2 + X + 1$
  1100000001111
- **CRC-16** : $X^{16} + X^{15} + X^2 + 1$
  11000000000000101
- **CRC CCITT V41** : $X^{16} + X^{12} + X^5 + 1$ (This code is primarily used in the HDLC)
  10001000000100001
- **CRC-32** (**Ethernet**) : = $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$
- **CRC ARPA** : $X^{24} + X^{23} + X^{17} + X^{16} + X^{15} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^5 + X^3 + 1$

- **SDLC**: $X^{16} + X^{12} + X^5 + 1$

Program:

```
#include<stdio.h>
#include<string.h>
char data[50],g[20],temp[20];
void XOR()
{
	for(int i=0;i<strlen(g);i++)
	temp[i]=(temp[i]==g[i]?'0':'1');

}
void CRC()
{
	int i,j;
	for(i=0;i<strlen(g);i++)
	temp[i]=data[i];
	do
	{
		if(temp[0]=='1')
			XOR();
		for(j=0;j<strlen(g)-1;j++)
		{
			temp[j]=temp[j+1];
		}
		temp[j]=data[i++];

	}while(i<=strlen(data));
}
int main()
{
	int flag=0,n,m,k;
	int opt;
	while(1)
	{
		printf("\n 1.CRC12 \n 2.CRC16 \n 3.CRC CCIT \n 4.Default \n5.exit \n\n Enter your option
\n");
		scanf("%d",&opt);
		switch(opt)
		{
			case 1:strcpy(g,"11000000011");
			break;
			case 2:strcpy(g,"11000000000000101");
			break;
			case 3:strcpy(g,"10001000000100001");
			break;
			case 4:strcpy(g,"1101");
			break;
			case 5:return 0;
		}
		printf("\n Enter data");
		scanf("%s",data);
		n=strlen(data);
		m=strlen(g);
		printf("\n Generate polynomial is:%s",g);
```

```
                printf("length of g is %d  length of data is %d\n",m,n);

                for(k=n;k<n+m-1;k++)
                        data[k]='0';

                printf("\n Modified datais %s\n",data);
                CRC();

                printf("\n cyclic redundancy check remainder value us%s",temp);
                int j;

                for(int k=n,j=0;k<n+m-1;k++,j++)
                        data[k]=temp[j];

                printf("\n Modified code word is:%s",data);
        }
}
```

output:

```
 1.CRC12
 2.CRC16
 3.CRC CCIT
 4.Default
 5.exit

 Enter your option 4

 Enter data100100

 Generate polynomial is:1101length of g is 4  length of data is 6

 Modified datais 100100000

 cyclic redundancy check remainder value us001

 Modified code word is:100100001
```

**VIVA Questions:**

**1. What is error?**

Ans. It is condition when the receiver's information does not match with the sender's information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from sender to receiver. That means a 0 bit may change to 1 or a 1 bit may change to 0.

**2.What are different error handling methods?**

1.Error Detection Method
2.Error Correction Method

**3. What are different Error detection Methods?**

Some popular techniques for error detection are:
1. Simple Parity check
2. Two-dimensional Parity check
3. Checksum
4. Cyclic redundancy check

**4. what is the purpose of redundant bits in error handling method?**

To detect or correct error some extra bit is added is known as redundant bit. Redundant bits are added by the sender and removed by the receiver.

**5. What is the difference between error detection and error correction?**

Error detection means to check if error has occurred and error correction is, if error has occurred then which bit is corrupted for that we need to consider whole data bit to check which bit is corrupted.

**6. How error correction  can be handled?**

Error Correction can be handled in two ways:

- **Backward error correction:** Once the error is discovered, the receiver requests the sender to retransmit the entire data unit.
- **Forward error correction:** In this case, the receiver uses the error-correcting code which automatically corrects the errors.

7. Which layer deals with error handling?

8.

# Implementation of Dijkstra‘s Algorithm

Aim: To write C Program to implement Dijkstra’s algorithm to find shortest path

Objective:

Dijkstra's Algorithm allows you to calculate the shortest path between one node (you pick which one) and every other node in the graph. Dijkstra algorithm is also called single source shortest path algorithm. It is based on greedy technique.

**Dijkstra’s Algorithm:**

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[ ].

**Step 1 :**

Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ].
C[i][j] is the cost of going from vertex i to vertex j.
If there is no edge between vertices i and j then C[i][j] is infinity.

**Step 2** : Array visited[ ] is initialized to zero.

```
for(i=0;i<n;i++)
        visited[i]=0;
```

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

**Step 4**:

1. 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];
```

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

**Step 5 :** for(i=1;i<n;i++)

1. Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as
2. Recalculate the shortest distance of remaining vertices from the source.
3. Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

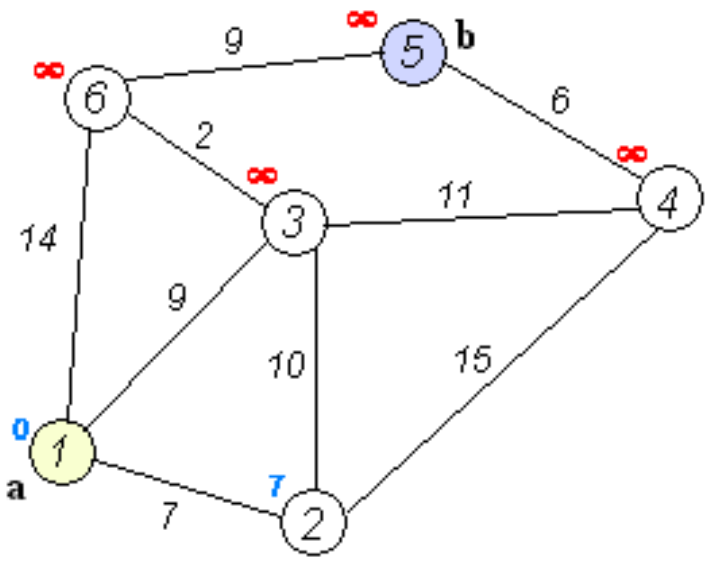
```
if(visited[v]==0)
    distance[v]=min(distance[v],distance[w]+cost[w][v])
```

**C Program:**

```
#include<stdio.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-1);
	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("\nTotal Cost from node %d  to %d is %d",startnode+1, i+1,distance[i]);
          printf("\nPath : %d",i+1);
                                                      j=i;
                           do
                           {
                                    j=pred[j];
                                    printf(" <-- %d",j+1);
                           }while(j!=startnode);
         }
}
```

**OUTPUT:**




Enter no. of vertices:6
Enter the adjacency matrix:
0 7 9 0 0 14
7 0 10 15 0 0
9 10 0 11 0 2
0 15 11 0 6 0
0 0 0 6 0 9
4 0 2 0 9 0

Enter the starting node:1

Total Cost from node 1 to 2 is 7
Path : 2 <-- 1
Total Cost from node 1 to 3 is 9
Path : 3 <-- 1
Total Cost from node 1 to 4 is 20
Path : 4 <-- 3 <-- 1
Total Cost from node 1 to 5 is 20
Path : 5 <-- 6 <-- 3 <-- 1
Total Cost from node 1 to 6 is 11
Path : 6 <-- 3 <-- 1

Enter the starting node: 3

Total Cost from node 3 to 1 is 6
Path : 1 <-- 6 <-- 3
Total Cost from node 3 to 2 is 10
Path : 2 <-- 3
Total Cost from node 3 to 4 is 11
Path : 4 <-- 3
Total Cost from node 3 to 5 is 11
Path : 5 <-- 6 <-- 3
Total Cost from node 3 to 6 is 2
Path : 6 <-- 3

Enter the starting node:4

Total Cost from node 4 to 1 is 17
Path : 1 <-- 6 <-- 3 <-- 4
Total Cost from node 4 to 2 is 15
Path : 2 <-- 4
Total Cost from node 4 to 3 is 11
Path : 3 <-- 4
Total Cost from node 4 to 5 is 6
Path : 5 <-- 4
Total Cost from node 4 to 6 is 13
Path : 6 <-- 3 <-- 4

VIVA Questions:

**1. Which technique the Dijkstra's algorithm is based on?**
The solution is based on a well-known algorithm from graph theory—**Dijkstra's shortest-path algorithm**. Basically, the algorithm works as follows. We start with containing this node and then initialize the table of costs (the s) to other nodes using the known costs to directly connected nodes.

**2. What is routing?**
Routing is the process of selecting a path for traffic in a network or between or across multiple networks.

**3. What is routing algorithm?**
In order to transfer the packets from source to the destination, the network layer must determine the best route through which packets can be transmitted. The routing protocol is a routing algorithm that provides the best path from the source to the destination. The best path is the path that has the "least-cost path" from source to the destination.

**4. What are the categories of routing algorithm?**
The Routing algorithm is divided into two categories:
Adaptive Routing algorithm
Non-adaptive Routing algorithm

**5. What is the difference between adaptive and non-adaptive routing algorithm?**

| Adaptive vs. Non-adaptive Routing | |
|---|---|
| Adaptive routing | Non-adaptive routing |
| Routers exchange updates and router table information. | Network administrator manually enters routing paths into the router. |
| Routes adjust automatically in response to changes in network topology. | Adjustments to changes in network topology require manual update. |
| Prevents packet delivery failure and improves network performance. | Provides granular control over packet paths. |

**6. What is packetizing?**
The process of encapsulating the data received from upper layers of the network(also called as payload) in a network layer packet at the source and decapsulating the payload from the network layer packet at the destination is known as packetizing.

**7. What is Forwarding Process?**
When a packet arrives at a router's input link, the router must move the packet to the appropriate output link. A router forward a packet by examining the value of a field in the arriving packet's header, and then using this header value to index into the router's forwarding table. The value stored in the forwarding table entry for that header indicates the router's outgoing link interface to which the packet is to be forwarded.

## 8. Difference between Connection-oriented and Connection-less Services

Connectionless service is used in the network system to transfer data from one end to another end without creating any connection. So it does not require establishing a connection before sending the data from the sender to the receiver. It is not a reliable network service because it does not guarantee the transfer of data packets to the receiver, and data packets can be received in any order to the receiver. Therefore we can say that the data packet does not follow a **defined** path. In connectionless service, the transmitted data packet is not received by the receiver due to network congestion, and the data may be lost.

# *Distance Vector Routing Algorithm and display the same.*

AIM: Program to take an subnet graph with weights indicating delay between nodes and obtain Routing table at each node using distance vector routing algorithm and display the same.

**Objective:**

**Distance-vector protocols** update the **routing** tables of **routers** and determine the **route** on which a packet will be sent by the next hop which is the exit interface of the **router** and the IP address of the interface of the receiving **router**. **Distance** is a measure of the cost to reach a certain node.

**Algorithm:**

**Step 1 :**

1. For each neighbour V, Create route struture rt(distance,from) from adjacency matrix adj[ ][ ].
2. C[i][i] is the set to 0.
3. If there is no edge between vertices i and j then C[i][j] is infinity.

**Step 2** : Choose arbitary vertex k and we calculate the direct distance from the node i to k using the cost matrix

**Step 3:** calculate the minimum distance using the formula.

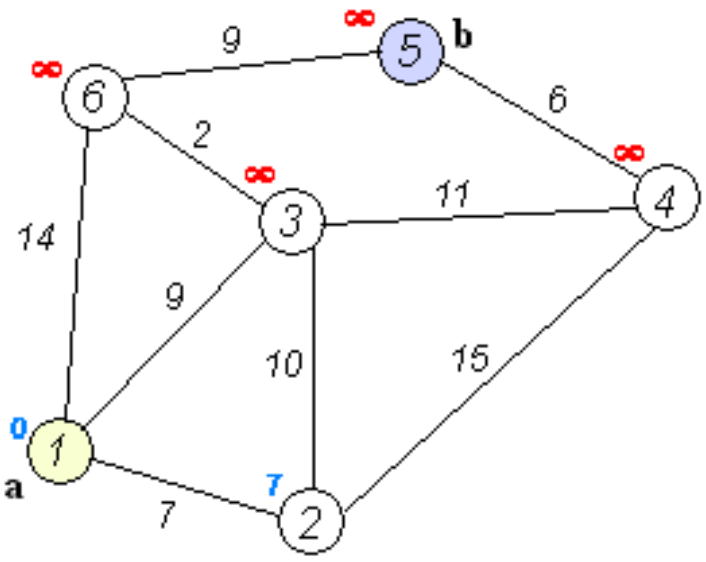
$$D_i(j) = \min_j \{ d_i(j) , c(i,k) + d_k(j)\} \quad \text{for each node j in N}$$

**C Program:**

```
#include<stdio.h>
#define INFINITY 99
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=1;i<=nodes;i++)
   {
      for(j=1;j<= nodes;j++)
      {
         scanf("%d",&costmat[i][j]);

            if(costmat[i][j] == 0)
                  costmat[i][j] = INFINITY;
```

```
            rt[i].dist[j] = costmat[i][j];//initialise the distance equal to cost matrix
            rt[i].from[j]=j;
        }
    }

//We choose arbitary 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( i=1;i<=nodes;i++)
        for ( j=1;j<=nodes;j++)
            if(i==j)
                {
                  rt[i].dist[j]  =  0;
                rt[i].from[j]  =  i;

                 }
            else
              for( k=1 ;  k <=nodes  ; k++)
                 if(  rt[i].dist[j]    >   costmat[i][k] + rt[k].dist[j]   )
                    {
                            rt[i].dist[j]  =   costmat[i][k] +  rt[k].dist[j];
                               rt[i].from[j]  =  k;
                    }
    for(i=1;i<=nodes;i++)
      {
        printf("\n\n Router %d : Routing Table\n",i);

        for(j=1; j<=nodes;j++)
        {
                printf("\t\nnode %d via %d Distance %d ",j,rt[i].from[j],rt[i].dist[j]);

        }
      }
   printf("\n\n");

}
```

**OUTPUT:**




Enter no. of vertices:6

Enter the adjacency matrix:

0 7 9 0 0 14
7 0 10 15 0 0
9 10 0 11 0 2
0 15 11 0 6 0
0 0 0 6 0 9
14 0 2 0 9 0

For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 7
node 3 via 3 Distance 9
node 4 via 3 Distance 20
node 5 via 6 Distance 23
node 6 via 3 Distance 11

For router 2

node 1 via 1 Distance 7
node 2 via 2 Distance 0
node 3 via 3 Distance 10
node 4 via 4 Distance 15
node 5 via 4 Distance 21
node 6 via 3 Distance 12

For router 3
node 1 via 1 Distance 9
node 2 via 2 Distance 10
node 3 via 3 Distance 0
node 4 via 4 Distance 11
node 5 via 6 Distance 11
node 6 via 6 Distance 2
For router 4
node 1 via 3 Distance 20
node 2 via 2 Distance 15
node 3 via 3 Distance 11
node 4 via 4 Distance 0
node 5 via 5 Distance 6
node 6 via 3 Distance 13

For router 5
node 1 via 6 Distance 23
node 2 via 4 Distance 21
node 3 via 6 Distance 11
node 4 via 4 Distance 6

**VIVA Questions**

**1. What is an autonomous system?**

**An** Autonomous System **(**AS**) is a collection of routers whose prefixes and routing policies are under common administrative control. This could be a network service provider, a large company, a university, a division of a company, or a group of companies.**

**2. Difference between Internet, Intranet and Extranet**

**1. Internet :** The network formed by the co-operative interconnection of millions of computers, linked together is called Internet

**2. Intranet :** It is an internal private network built within an organization using Internet and World Wide Web standards and products that allows employees of an organization to gain access to corporate information.

**3. Extranet :** It is the type of network that allows users from outside to access the Intranet of an organization.

**4. What are the features of DVR?**

Following are the features of the distance vector routing are –

- Router communicates all of its connected knowledge about the network to its neighbors.
- Sharing of data takes place only with the neighbors.
- Sending of data holds place at constant, ordinary intervals, declared every 30 seconds.

**5. What are the two types of update in DVR?**

Trigger update and Peridic update

**Periodic Update:** A node sends its routing table, normally every 30 s. The period depends on the protocol that is using distance vector routing.
**Triggered Update:** A node sends its two-column routing table to its neighbors anytime there is a change in its routing table.

**6. What is count to infinity problem?**

Count to infinity problem is one of the important issue in Distance Vector Routing is County of Infinity Problem. Counting to infinity is just another name for a **routing loop or Route Poisoning**.

In distance vector routing, routing loops usually occur when an interface goes down. Here Router will then go on feeding each other bad information toward infinity which is called as **Count to Infinity problem**.

**7. Solution to Count to infinity problem?**

**1. Defining Infinity** The first obvious solution is to redefine infinity to a smaller number, such as 100.

**2. Split horizon:** if router receive routing information on one interface, sending that information back out of that interface is not likely to be productive. So instead of flooding the table through each interface, each node sends only part of its table through each interface .i.e it does not need to advertise information learnt form a router back to it.

**3. Split Horizon and Poison Reverse**: A Router can still advertise the full routing table, but if the source of information is same router , it can replace the distance with infinity as a warning: "Do not use this value; what I know about this route comes from you."

**8. What is RIP?**

The Routing Information Protocol (RIP) is an intradomain routing protocol used inside an autonomous system. It is a very simple protocol based on distance vector routing. RIP implements distance vector routing directly with some considerations:

1. In an autonomous system, we are dealing with routers and networks (links). The routers have routing tables; networks do not.
2. The destination in a routing table is a network, which means the first column defines a network address.
3. The metric used by RIP is very simple; the distance is defined as the number of links (networks) to reach the destination. For this reason, the metric in RIP is calleda hop count.
4. Infinity is defined as 16, which means that any route in an autonomous system using RIP cannot have more than 15 hops.
5. The next-node column defines the address of the router to which the packet is to be sent to reach its destination.

**9. What are routing protocols?**

A routing protocol is a combination of rules and procedures that lets routers in the internet inform each other of changes. It allows routers to share whatever they know about the internet or their neighborhood. The sharing of information allows a router in San Francisco to know about the failure of a network in Texas. The routing protocols also include procedures for combining information received from other routers.

# *Congestion control using Leaky Bucket algorithm*

Aim: To Write C Program to implement Leaky bucket algorithm for Congestion control.

Description:

**Traffic Shaping** is a mechanism to control the amount and the rate of the traffic sent to the network. This approach of congestion management is called Traffic shaping. Traffic shaping helps to regulate rate of data transmission and reduces congestion..

There are 2 types of traffic shaping algorithms:

1. Leaky Bucket
2. Token Bucket

In networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.

Algorithm:

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define NOF_PACKETS 10
int rand(int a)
{
   int rn = (random() % 10) % a;
   return  rn == 0 ? 1 : rn;
}

int main()
{
   int packet_sz[NOF_PACKETS], i, clk, b_size, o_rate, p_sz_rm=0, p_sz, p_time, op;
   for(i = 0; i<NOF_PACKETS; ++i)
      packet_sz[i] = rand(6) * 10;
   for(i = 0; i<NOF_PACKETS; ++i)
      printf("\npacket[%d]:%d bytes\t", i, packet_sz[i]);
   printf("\nEnter the Output rate:");
   scanf("%d", &o_rate);
   printf("Enter the Bucket Size:");
   scanf("%d", &b_size);

for(i = 0; i<NOF_PACKETS; ++i)
   {
```

```
      if( (packet_sz[i] + p_sz_rm) > b_size)
         if(packet_sz[i] > b_size)/*compare the packet siz with bucket size*/
            printf("\n\nIncoming packet size (%dbytes) is Greater than bucket capacity (%dbytes)-PACKET
REJECTED", packet_sz[i], b_size);
         else
            printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!");
      else
      {
         p_sz_rm += packet_sz[i];
         printf("\n\nIncoming Packet size: %d", packet_sz[i]);
         printf("\nBytes remaining to Transmit: %d", p_sz_rm);
         p_time = rand(4) * 10;
         printf("\nTime left for transmission: %d units", p_time);
         for(clk = 10; clk <= p_time; clk += 10)
         {
            sleep(1);
            if(p_sz_rm)
            {
               if(p_sz_rm <= o_rate)/*packet size remaining comparing with output rate*/
                  op = p_sz_rm, p_sz_rm = 0;
               else
                  op = o_rate, p_sz_rm -= o_rate;
               printf("\nPacket of size %d Transmitted", op);
               printf("----Bytes Remaining to Transmit: %d", p_sz_rm);
            }
            else
            {
               printf("\nTime left for transmission: %d units", p_time-clk);
               printf("\nNo packets to transmit!!");
            }
         }
      }
   }
}
```

Output:

```
packet[0]:30 bytes
packet[1]:10 bytes
packet[2]:10 bytes
packet[3]:50 bytes
packet[4]:30 bytes
packet[5]:50 bytes
packet[6]:10 bytes
packet[7]:20 bytes
packet[8]:30 bytes
packet[9]:10 bytes
Enter the Output rate:5
Enter the Bucket Size:30


Incoming Packet size: 30
```

Bytes remaining to Transmit: 30
Time left for transmission: 20 units
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 25
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 20

Incoming Packet size: 10
Bytes remaining to Transmit: 30
Time left for transmission: 30 units
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 25
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 20
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 15

Incoming Packet size: 10
Bytes remaining to Transmit: 25
Time left for transmission: 10 units
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 20

Incoming packet size (50bytes) is Greater than bucket capacity (30bytes)-PACKET REJECTED

Bucket capacity exceeded-PACKETS REJECTED!!

Incoming packet size (50bytes) is Greater than bucket capacity (30bytes)-PACKET REJECTED

Incoming Packet size: 10
Bytes remaining to Transmit: 30
Time left for transmission: 10 units
Packet of size 5 Transmitted----Bytes Remaining to Transmit: 25

Bucket capacity exceeded-PACKETS REJECTED!!

Bucket capacity exceeded-PACKETS REJECTED!!

Bucket capacity exceeded-PACKETS REJECTED!!

# *Broadcast tree for a given subnet.*

**Aim:** To write c program to take an subnet of hosts and obtain broadcast tree for it.

**Description:**
Broadcasting refers to transmitting a packet that will be received by every device in the network. Broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast—or any other convenient spanning tree for that matter.
A spanning tree is a subset of the subnet that includes all the routers but contains no loops. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some spanning tree for the method to be applicable.
In this program we use Kruskal's Algorithm to find the sink tree for the give graph.

**Kruskal's Algorithm**
This algorithm will create spanning tree with minimum weight, from a given weighted graph.

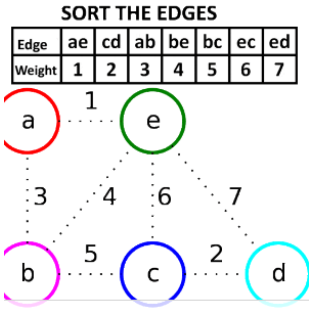
1. Begin
2. Create the edge list of given graph, with their weights.
3. Sort the edge list according to their weights in ascending order.
4. Draw all the nodes to create skeleton for spanning tree.
5. Pick up the edge at the top of the edge list (i.e. edge with minimum weight).
6. Remove this edge from the edge list.
7. Connect the vertices in the skeleton with given edge. If by connecting the vertices, a cycle is created in the skeleton, then discard this edge.
8. Repeat steps 5 to 7, until n-1 edges are added or list of edges is over.
9. Return

```
#include<stdio.h>
#define MAX 30

typedef struct edge
{
   int u,v,w;
}edge;

typedef struct edgelist
{
   edge data[MAX];
   int n;
}edgelist;

edgelist elist;

int G[MAX][MAX],n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();
```

```
void main()
{
   int i,j,total_cost;
   printf("\nEnter number 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]);
   kruskal();
   print();
}

void kruskal()
{
        int belongs[MAX],i,j,cno1,cno2;
        elist.n=0;

        for(i=1;i<n;i++)
                for(j=0;j<i;j++)
                        {
                          if(G[i][j]!=0)
                          {
                                  elist.data[elist.n].u=i;
                                  elist.data[elist.n].v=j;
                                  elist.data[elist.n].w=G[i][j];
                                  elist.n++;
                          }
                }
                sort();
                for(i=0;i<n;i++)
                belongs[i]=i;
                spanlist.n=0;
                for(i=0;i<elist.n;i++)
                {
                        cno1=find(belongs,elist.data[i].u);
                        cno2=find(belongs,elist.data[i].v);
                        if(cno1!=cno2)
                        {
                                spanlist.data[spanlist.n]=elist.data[i];
                                spanlist.n=spanlist.n+1;
                                union1(belongs,cno1,cno2);
                        }
                }
}

int find(int belongs[],int vertexno)
{
   return(belongs[vertexno]);
}



void union1(int belongs[],int c1,int c2)
```

```
{
   int i;
   for(i=0;i<n;i++)
      if(belongs[i]==c2)
         belongs[i]=c1;
}

void sort()
{
   int i,j;
   edge temp;
   for(i=1;i<elist.n;i++)
      for(j=0;j<elist.n-1;j++)
         if(elist.data[j].w>elist.data[j+1].w)
         {
            temp=elist.data[j];
            elist.data[j]=elist.data[j+1];
            elist.data[j+1]=temp;
         }
}

void print()
{
   int i,cost=0;
   for(i=0;i<spanlist.n;i++)
   {
      printf("\nEdge %d -- %d : cost%d",spanlist.data[i].u+1,spanlist.data[i].v+1,spanlist.data[i].w);
      cost=cost+spanlist.data[i].w;
   }

   printf("\n\nCost of the spanning tree=%d",cost);
}
```

Output:




Enter number of vertices:6

Enter the adjacency matrix:
0 7 9 0 0 14
7 0 10 15 0 0
9 10 0 11 0 2
0 15 11 0 6 0
0 0 0 6 0 9
14 0 2 0 9 0

Edge 6 -- 3 : cost2
Edge 5 -- 4 : cost6
Edge 2 -- 1 : cost7
Edge 3 -- 1 : cost9
Edge 6 -- 5 : cost9

Cost of the spanning tree=33

**VIVA Question**

**1. What is congestion?**

Congestion in a network may occur if the **load** on the network-the number of packets sent to the network-is greater than the *capacity* of the network-the number of packets a network can handle.

**2. What is congestion control?**

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

**3. Different congestion control mechanism?**

We can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal)

- In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.
- Closed-loop congestion control mechanisms try to alleviate congestion after it happens.

**4. Name the policies that can prevent congestion.**

The following policies can help to prevent congestion: ***a good retransmission policy***,***use of the selective-repeat window***, ***a good acknowledgment policy***, ***a good discard policy***, and ***a good admissions policy***.

**5. Name the mechanisms that can alleviate congestion**

Congestion can be alleviated by ***back pressure***, ***a choke point***, implicit signaling and ***explicit signaling***.

**Backpressure :** Backpressure is a technique in which a congested node stops receiving packets from upstream node.

**Choke Packet Technique :** Choke packet technique is applicable to both virtual networks as well as datagram subnets. A choke packet is a packet sent by a node to the source to inform it of congestion

**Implicit Signaling :** In implicit signaling, there is no communication between the congested nodes and the source. The source guesses that there is congestion in a network. For example when sender sends several packets and there is no acknowledgment for a while, one assumption is that there is a congestion.

**Explicit Signaling :** In explicit signaling, if a node experiences congestion it can explicitly sends a packet to the source or destination to inform about congestion.

**Forward Signaling :** In forward signaling, a signal is sent in the direction of the congestion. The destination is warned about congestion. The receiver in this case adopt policies to prevent further congestion.

**Backward Signaling :** In backward signaling, a signal is sent in the opposite direction of the congestion. The source is warned about congestion and it needs to slow down.

**6. How congestion control is implemented in virtual circuit sub-network?**

One technique that is widely used in virtual-circuit networks to keep congestion is **admission control**. The idea is simple: do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested.

# *Write a program for frame sorting technique used in buffers*

**Description:**

Data-link layer takes the packets from the Network Layer and encapsulates them into frames. If the frame size becomes too large, then the packet may be divided into small sized frames. Smaller sized frames makes flow control and error control more efficient.

Receiver Data link layer may receive the out of order frame. These frame are sorted based on sequence number and order message is delivered to network layer.

**Algorithm:**

Program input in unsorted frames with sequence number and data. Algorithm sorts the frame based on sequence number and prints the message in order.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct frame
{
   int fslno;
   char finfo[20];
};
struct frame arr[10];
int n;
void sort()
{
    int i,j,ex;
   struct frame temp;
    for(i=0;i<n;i++)
      {
         ex=0;
         for(j=0;j<n-i-1;j++)
       if(arr[j].fslno>arr[j+1].fslno)
             {
                temp=arr[j];
        arr[j]=arr[j+1];
          arr[j+1]=temp;
           ex++;
       }

       if(ex==0) break;
   }
}

void main()
{

    int i;
```

```
    char msg[100];
    printf("\n Enter the number of frames \n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {

        printf("\n Enter the frame sequence number & data\n");
        scanf("%d",&arr[i].fslno);
        //printf("\n Enter the frame data\n");
        gets(arr[i].finfo);
    }

    sort();
    strcpy(msg,arr[0].finfo);
    int j=0;
    for(i=0;i<n;i++)
      {
            for(int k=0;k<3;k++)
            {
               msg[j++]=arr[i].finfo[k];
            }

      }
      msg[j]='\0';
   puts(msg);
}
```

# *Implementation of Caesar Cypher Algorithm to encrypt and decrypt the string*

**Description:**

- Encryption is a way of scrambling data so that only authorized parties can understand the information. In technical terms, it is the process of converting plaintext to ciphertext.

- Decrytion is a process of converting ciphertext to plaintext.

Encryption and decryption requires the use of a cryptographic key: a set of mathematical values that both the sender and the recipient of an encrypted message agree on.

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials. Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down

**Algorithm:**

Encryption of a letter by a shift n can be described mathematically as. $E_n(x) = (x + n) mod\ 26$

Decryptio of a letter described mathematically as $D_n(x) = (x - n) mod\ 26$

Program take

.

Program

```
#include <stdio.h>
#include<string.h>
#include <ctype.h>
int main()
{
        int i, x;
        char str[100];
        printf("\nPlease choose following options:\n");
        printf("1 = Encrypt the string.\n");
        printf("2 = Decrypt the string.\n");
        scanf("%d", &x);
```

```
        //using switch case statements
        switch(x)
        {
                case 1:
                        printf("\nPlease enter a string:\t");
                        //fgets(str,100,stdin);
                        scanf("%s",str);
                        int length=strlen(str);
                        for (int i=0;i<length;i++)
                         {
                            if (isupper(str[i]))
                                str[i]= (str[i]+3-65)%26 +65;
                         else
                                str[i]= (str[i]+3-97)%26 +97;
                        }
                  printf("\nEncrypted string: %s\n", str);
                  break;
        case 2:
                printf("\nPlease enter a string:\t");
                scanf("%s",str);
                length=strlen(str);
                for (int i=0;i<length;i++)
                 {
                    if (isupper(str[i]))
                        str[i]= (str[i]-3-65)%26 +65; ////the key for decryption is 3 that is substracted to ASCII value
                  else
                                str[i]= (str[i]-3-97)%26 +97;
                 }
                 printf("\nDecrypted string: %s\n", str);
                  break;
        default:
                  printf("\nError\n");
  }
  return 0;
}
```