

COMPUTER NETWORKS (SE-304a)

Laboratory Manual



Department of Computer Science and Engineering

DELHI TECHNOLOGICAL UNIVERSITY(DTU)

Shahbad Daulatpur, Bawana Road, Delhi-110042

09 April 2021

Submitted to: -

Dr. Anil Singh Parihar

Submitted by:-

Name:- ASHISH KUMAR

Roll number:- 2K18/SE/041

INDEX

S.No	EXPERIMENT	DATE	REMARKS
1.	Write a program to implement data link layer - Bit Stuffing.	15-01-2021	
1b.	Write a program to implement data link layer - Character Stuffing.	22-01-2021	
1c.	Write a program to implement data link layer - Byte Stuffing.	29-01-2021	
2.	Write a program to implement cyclic redundancy check (CRC).	05-02-2021	
3.	Write a program to implement Stop and Wait Protocol.	19-02-2021	
4.	Write a program to implement Sliding Window Protocol.	26-02-2021	
5.	Write a program to implement and find Network Class, Network Id and Host Id of given IPV4 address.	05-03-2021	
6.	Write a program to implement distance vector routing (DVR) algorithm.	12-03-2021	
7.	Write a program to implement link state routing algorithm.	19-03-2021	

PROGRAM 1

Aim:- Write a program to implement data link layer - Bit Stuffing.

Theory:- Bit stuffing is the mechanism of inserting one or more non-information bits into a message to be transmitted, to break up the message sequence, for synchronization purpose. It is widely used in network and communication protocols, in which bit stuffing is a required part of the transmission process.

Bit Stuffing Mechanism

In a data link frame, the delimiting flag sequence generally contains six or more consecutive 1s. In order to differentiate the message from the flag in case of the same sequence, a single bit is stuffed in the message. Whenever a 0 bit is followed by five consecutive 1bits in the message, an extra 0 bit is stuffed at the end of the five 1s.

When the receiver receives the message, it removes the stuffed 0s after each sequence of five 1s. The un-stuffed message is then sent to the upper layers.

CODE:-

```
#include<iostream>

#include<bits/stdc++.h>

#define MAX 100

using namespace std;

int main(){

    int i,j,a[MAX],n,new_n;

    cout<<"Enter string length:";

    cin>>n;
```

```

    if(n<=0){
        cout<<"Invalid size specified.\n";
        exit(0);
    }
    cout<<"\nEnter input string (0's & 1's only):";
    for(int i=0;i<n;i++)
    {
        cin>>a[i]; //storing each 0 & 1 in array
    }

//Stuffing
new_n=n;
for(i=0;i<=MAX;i++){
    if(a[i]==0 && i+7 < new_n){
        if(a[i+1] == a[i+2] == a[i+3] == a[i+4] == a[i+5] == a[i+6] == 1 &&
a[i+7]==0) {
            new_n++;
            for(j=new_n-1;j>i+6;j--){
                a[j] = a[j-1];
            }
            a[i+6] = 0; // stuff a bit 0
            i=i+8;
        }
    }
}

```

```

    }

    cout<<"\nAfter bit stuffing the string becomes:";

    for(i=0;i<new_n;i++){

        cout << a[i] << " ";

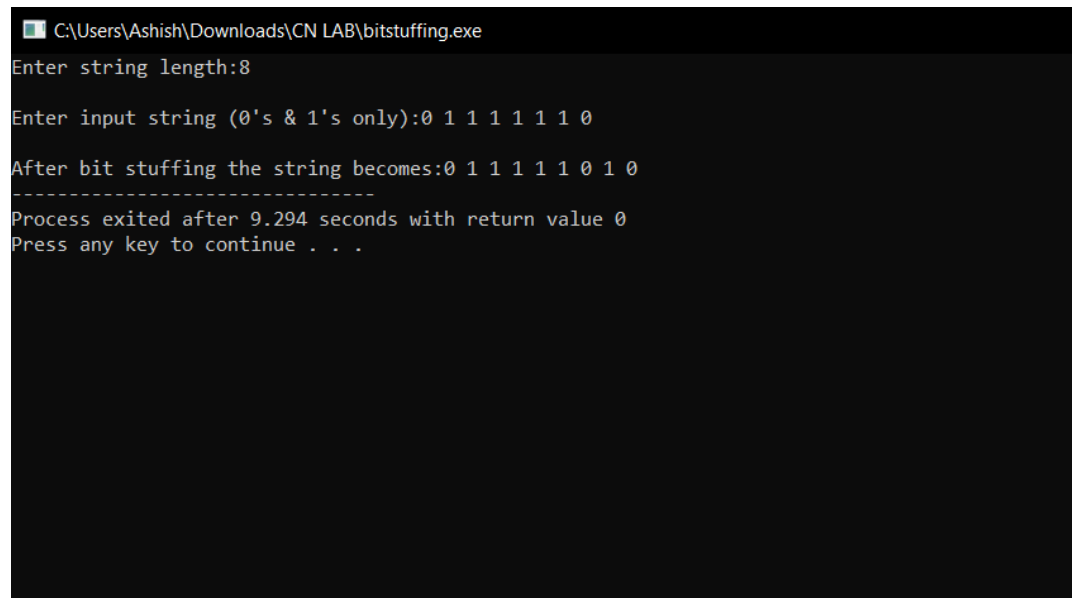
    }

    return 0;

}

```

OUTPUT:-



```

C:\Users\Ashish\Downloads\CN LAB\bitstuffing.exe
Enter string length:8
Enter input string (0's & 1's only):0 1 1 1 1 1 0
After bit stuffing the string becomes:0 1 1 1 1 1 0 1 0
-----
Process exited after 9.294 seconds with return value 0
Press any key to continue . . .

```

Learning Outcome:- We have successfully implemented bit stuffing and we learnt its application in computer network.

PROGRAM 1b

Aim:- Write a program to implement data link layer - Character Stuffing.

Theory:- In byte stuffing(or character stuffing), a special byte called the escape character (ESC) is stuffed before data section of the frame when there is a character with the same pattern as the flag byte. If the ESC sequence is found in the message byte, then another ESC byte is stuffed before it. Whenever the receiver encounters the ESC character, it deletes it from the data section and treats the next character as data, not a delimiting flag.

CODE:-

```
#include<iostream>

#include<bits/stdc++.h>

#include<string.h>

using namespace std;

int main()

{

    char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3];

    int i, j;

    /*we will consider only three types of byte sequences in the sent data, as :

        F : Flag Sequence

        E : Escape Sequence

        D : Any other Data Sequence*/
```

```

cout<<" Enter characters to be stuffed:";

cin>>a;

cout<<"\n Enter a character that represents starting delimiter:";

cin>>sd;

cout<<"\n Enter a character that represents ending delimiter:";

cin>>ed;

x[0] = s[0] = s[1] = sd;

x[1] = s[2] = '\0';

y[0] = d[0] = d[1] = ed;

d[2] = y[1] = '\0';

strcat(fs, x);

for(i = 0; i < strlen(a); i++)
{
    t[0] = a[i];

    t[1] = '\0';

    if(t[0] == 'F')    //if we see a flag byte,then insert E before F
        strcat(fs, "EF");

    else if(t[0] == 'E')    //if we see a ESC byte, then insert E before E
        strcat(fs, "EE");

    else
        strcat(fs, "D");    //if we see data, then nothing will be done
}

strcat(fs, y);

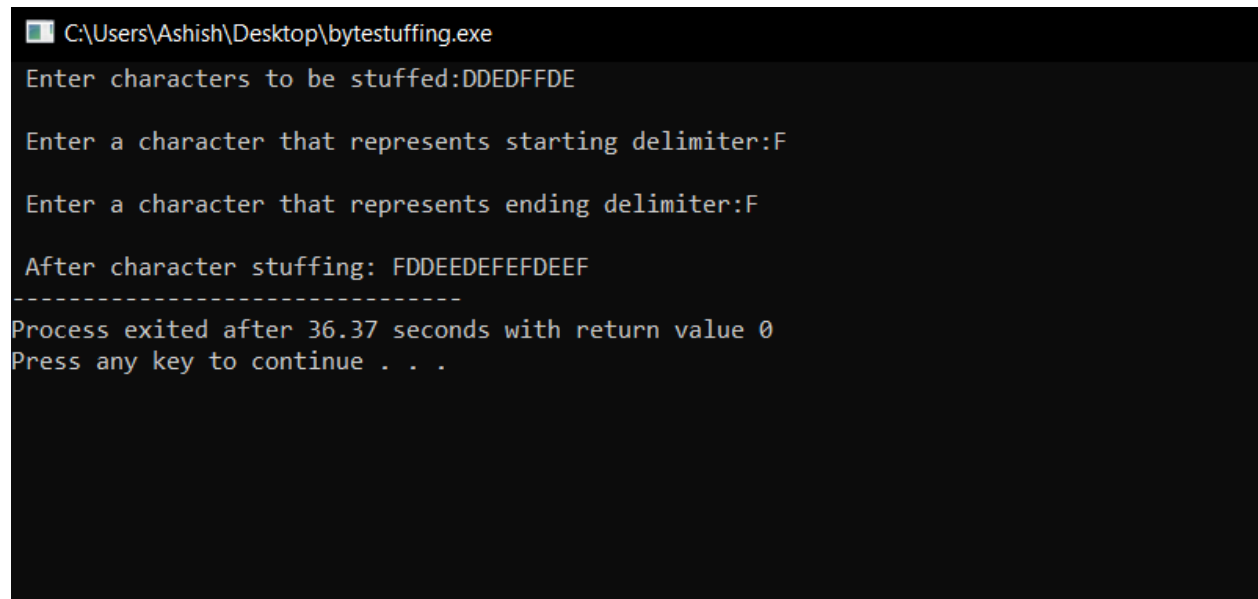
```

```
cout<<"\n After character stuffing:"<< fs;

return 0;

}
```

OUTPUT:-



```
C:\Users\Ashish\Desktop\bytestuffing.exe
Enter characters to be stuffed:DDEDFDE
Enter a character that represents starting delimiter:F
Enter a character that represents ending delimiter:F
After character stuffing: FDDEEDEFDFDEEF
-----
Process exited after 36.37 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have successfully implemented Character stuffing and we learnt its application in computer network.

PROGRAM 1c

Aim:- Write a program to implement data link layer - Byte Stuffing.

Theory:- In byte stuffing, a special byte called the escape character (ESC) is stuffed before data section of the frame when there is a character with the same pattern as the flag byte. If the ESC sequence is found in the message byte, then another ESC byte is stuffed before it. Whenever the receiver encounters the ESC character, it deletes it from the data section and treats the next character as data, not a delimiting flag.

CODE:-

```
#include<stdio.h>

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

int main(){

    char input[100];

    cout<<"Enter Data: ";

    for(int i=0;i<20;i++){

        cin>>input[i];

    }

    cout<<"\nFlag: 11111111\n";

    cout<<"\nEscape Seq: 00000000\n";

    cout<<"\nOutput after byte stuffing:";
```

```
cout<<"11111111 ";
int count=0,ce=0;
for(int i=0;i<20;i++){
    if(input[i]=='1'){
        if(count==7){
            cout<<" ";
            for(int j=0;j<8;j++){

                cout<<"0";
            }
            cout<<" ";
            count=0;
        }
        else{
            count++;
        }
    }
    else{
        count=0;
    }
    if(input[i]=='0'){
        if(ce==7){
            cout<<" ";
```

```
        for(int j=0;j<8;j++){  
            cout<<"0";  
        }  
        cout<<" ";  
        ce=0;  
    }  
    else{  
        ce++;  
    }  
}  
else{  
    ce=0;  
}  
cout<<input[i];  
}  
  
cout<<" 11111111";  
return 0;  
}
```

OUTPUT:-

```
C:\Users\Ashish\Desktop\program1c.exe
Enter Data: 0011010101111111111000110
Flag: 11111111
Escape Seq: 00000000
Output after byte stuffing:11111111 0011010101111111 00000000 1110 11111111
-----
Process exited after 1.921 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have successfully implemented byte stuffing and we learnt its application in computer network.

PROGRAM 2

Aim:- Write a program to implement cyclic redundancy check(CRC).

Theory:- CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel. CRC uses **Generator Polynomial** which is available on both sender and receiver side.

Following are the steps used in CRC for error detection:

- In CRC technique, a string of n 0s is appended to the data unit, and this n number is less than the number of bits in a predetermined number, known as division which is n+1 bits.
- Secondly, the newly extended data is divided by a divisor using a process is known as binary division. The remainder generated from this division is known as CRC remainder.
- Thirdly, the CRC remainder replaces the appended 0s at the end of the original data. This newly generated unit is sent to the receiver.
- The receiver receives the data followed by the CRC remainder. The receiver will treat this whole unit as a single unit, and it is divided by the same divisor that was used to find the CRC remainder.

If the resultant of this division is zero which means that it has no error, and the data is accepted.

If the resultant of this division is not zero which means that the data consists of an error. Therefore, the data is discarded.

CODE:-

```
#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int main()
```

```
{  
    int i,j,k,l;  
  
    int fs;  
  
    cout<<"\n Enter Frame size: ";  
  
    cin>>fs;  
  
    int f[20];  
  
    cout<<"\n Enter Frame:";  
  
    for(i=0;i<fs;i++)  
    {  
        cin>>f[i];  
    }  
  
  
    int gs;  
  
    cout<<"\n Enter Generator bits size: ";  
  
    cin>>gs;  
  
  
    int g[20];  
  
  
    cout<<"\n Enter Generator bits(or key):";  
  
    for(i=0;i<gs;i++)  
    {  
        cin>>g[i];  
    }  
}
```

```
cout<<"\n On Sender Side:";
```

```
//Append 0's
```

```
int rs=gs-1;
```

```
for (i=fs;i<fs+rs;i++)
```

```
{
```

```
    f[i]=0;
```

```
}
```

```
int temp[20];
```

```
for(i=0;i<20;i++)
```

```
{
```

```
    temp[i]=f[i];
```

```
}
```

```
cout<<"\n Message after appending 0's :";
```

```
for(i=0; i<fs+rs;i++)
```

```
{
```

```
    cout<<temp[i];
```

```
}
```

```
//Division
```

```
for(i=0;i<fs;i++)
```

```
{
```

```
    j=0;
```

```
    k=i;
```

```
    //check whether it is divisible or not
```

```
    if (temp[k]>=g[j])
```

```
    {
```

```
        for(j=0,k=i;j<gs;j++,k++)
```

```
        {
```

```
            if((temp[k]==1 && g[j]==1) || (temp[k]==0 && g[j]==0))
```

```
            {
```

```
                temp[k]=0;
```

```
            }
```

```
        else
```

```
        {
```

```
            temp[k]=1;
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
int crc[15];
```



```
for(i=0,j=fs;i<rs;i++,j++)
```

```
{
```

```
    crc[i]=temp[j];
```

```
}
```

```
cout<<"\n CRC bits: ";
```

```
for(i=0;i<rs;i++)
```

```
{
```

```
    cout<<crc[i];
```

```
}
```

```
cout<<"\n Transmitted data: ";
```

```
int tf[15];
```

```
for(i=0;i<fs;i++)
```

```
{
```

```
    tf[i]=f[i];
```

```
}
```

```
for(i=fs,j=0;i<fs+rs;i++,j++)
```

```
{
```

```
    tf[i]=crc[j];
```

```
}
```

```
for(i=0;i<fs+rs;i++)
```

```
{
```

```

        cout<<tf[i];
    }

    cout<<"\n\n On Receiver side : ";
    cout<<"\n Received data: ";
    for(i=0;i<fs+rs;i++)
    {
        cout<<tf[i];
    }

    for(i=0;i<fs+rs;i++)
    {
        temp[i]=tf[i];
    }

    for(i=0;i<fs+rs;i++)
    {
        j=0;
        k=i;
        if (temp[k]>=g[j])
        {
            for(j=0,k=i;j<gs;j++,k++)
            {

```

```

        if((temp[k]==1 && g[j]==1) || (temp[k]==0 && g[j]==0))
        {
            temp[k]=0;
        }
        else
        {
            temp[k]=1;
        }
    }
}

```

```

cout<<"\n Remainder: ";
int rrem[15];
for (i=fs,j=0;i<fs+rs;i++,j++)
{
    rrem[j]= temp[i];
}
for(i=0;i<rs;i++)
{
    cout<<rrem[i];
}

```

```
int flag=0;
for(i=0;i<rs;i++)
{
    if(rrem[i]!=0)
    {
        flag=1;
    }
}

if(flag==0)
{
    cout<<"\n\n Since Remainder is all zeros. Hence Message Transmitted From
Sender To Receiver Is Correct";
}
else
{
    cout<<"\n\n Since remainder is not all zeroes. Hence Message Transmitted
From Sender To Receiver Contains Error";
}
return 0;
}
```

OUTPUT:-

```
C:\Users\Ashish\Downloads\CN LAB\CRC.exe

Enter Frame size: 6

Enter Frame:1 0 0 1 0 0

Enter Generator bits size: 4

Enter Generator bits(or key):1 1 0 1

On Sender Side:
Message after appending 0's :10010000
CRC bits: 001
Transmitted data: 100100001

On Receiver side :
Received data: 100100001
Remainder: 000

Since Remainder is all zeros. Hence Message Transmitted From Sender To Receiver Is Correct
-----
Process exited after 14.03 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have successfully implemented cyclic redundancy check(CRC) and we learnt its application in computer network.

PROGRAM 3

Aim:- Write a program to implement Stop and Wait Protocol.

Theory:- Stop-and-wait ARQ, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order.

It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time; it is a special case of the general sliding window protocol with transmit and receive window sizes equal to one and greater than one respectively. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement (ACK) signal. After receiving a valid frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again. The timeout countdown is reset after each frame transmission. The above behavior is a basic example of Stop-and-Wait. However, real-life implementations vary to address certain issues of design.

CODE:-

```
#include<iostream>

#include<bits/stdc++.h>

#include<dos.h>

#include <unistd.h>

#define time 5

#define max_seq 1

#define tot_pack 5

using namespace std;
```

```

int randn(int n)
{
    return rand()%n + 1;
}

typedef struct
{
    int data;
}packet;

typedef struct
{
    int kind;
    int seq;
    int ack;
    packet info;
}frame;

typedef enum{ frame_arrival,error,time_out}event_type;

frame data1;

//creating prototype

void from_network_layer(packet *);
void to_physical_layer(frame *);
void to_network_layer(packet *);
void from_physical_layer(frame*);

```

```
void sender();  
void receiver();  
void wait_for_event_sender(event_type *);  
void wait_for_event_receiver(event_type *);
```

```
#define inc(k) if(k<max_seq)k++;else k=0;
```

```
int i=1;  
char turn;  
int disc=0;  
int main()  
{  
    while(!disc)  
    { sender();  
      //delay(400);  
      receiver();  
    }  
    getchar();  
}  
void sender()  
{  
    static int frame_to_send=0;  
    static frame s;
```



```

    packet buffer;

    event_type event;

    static int flag=0;    //first place

    if (flag==0)
    {
        from_network_layer(&buffer);

        s.info=buffer;

        s.seq=frame_to_send;

        cout<<"\n\nSender information : "<<s.info.data<<"\n";

        turn='r';

        to_physical_layer(&s);

        flag=1;

    }

    wait_for_event_sender(&event);

    if(turn=='s')
    {
        if(event==frame_arrival)
        {
            from_network_layer(&buffer);

            inc(frame_to_send);

            s.info=buffer;

            s.seq=frame_to_send;

```

```

        cout<<"\n\nSender information : "<<s.info.data<<"\n";
        turn='r';
        to_physical_layer(&s);
    }
}
//end of sender function

```

```

void from_network_layer(packet *buffer)
{
    (*buffer).data=i;
    i++;
}
//end of from network layer function

```

```

void to_physical_layer(frame *s)
{
    data1=*s;
}
//end of to physical layer function

```

```

void wait_for_event_sender(event_type *e)
{
    static int timer=0;
    if(turn=='s')
    {
        timer++;
    }
}

```

```

        return ;
    }
else          //event is frame arrival
    {
        timer=0;
        *e=frame_arrival;
    }
}          //end of wait for event function

```

```

void receiver()
{
    static int frame_expected=0;
    frame s,r;
    event_type event;
    wait_for_event_receiver(&event);
    if(turn=='r')
    { if(event==frame_arrival)
    {
        from_physical_layer(&r);
        if(r.seq==frame_expected)
        {
            to_network_layer(&r.info);
            inc (frame_expected);
        }
    }
}

```

```

    }
else
    cout<<"\nReceiver :Acknowledgement resent \n";
    turn='s';
    to_physical_layer(&s);
    }
}
} //end of receiver function

```

```

void wait_for_event_receiver(event_type *e)
{
    if(turn=='r')
    {
        *e=frame_arrival;
    }
}

```

```

void from_physical_layer(frame *buffer)
{
    *buffer=data1;
}

```

```
void to_network_layer(packet *buffer)
{
    cout<<"\nReceiver : "<<" Packet "<<i-1<<" received \t";
    cout<<"\nAcknowledgement sent ";
    if(i>tot_pack)
    {
        disc=1;
        cout<<"\n\nDISCONNECTED...\n";
    }
}
```

OUTPUT:-

```
C:\Users\Ashish\Desktop\stop&wait.exe

Sender information : 1

Receiver : Packet 1 received
Acknowledgement sent

Sender information : 2

Receiver : Packet 2 received
Acknowledgement sent

Sender information : 3

Receiver : Packet 3 received
Acknowledgement sent

Sender information : 4

Receiver : Packet 4 received
Acknowledgement sent

Sender information : 5

Receiver : Packet 5 received
Acknowledgement sent

DISCONNECTED...
```

Learning Outcome:- We have successfully implemented stop and wait protocol and we learnt its application in computer network. The Stop and Wait ARQ solves main three problems, but may cause big performance issues as sender always waits for acknowledgement even if it has next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country though a high speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence numbers. So Stop and Wait ARQ may work fine where propagation delay is very less for example LAN connections, but performs badly for distant connections like satellite connection.

PROGRAM 4

Aim:- Write a program to implement Sliding Window Protocol.

Theory:- The sliding window is a technique for sending multiple frames at a time. It controls the data packets between the two devices where reliable and gradual delivery of data frames is needed. It is also used in TCP (Transmission Control Protocol).

In this technique, each frame has sent from the sequence number. The sequence numbers are used to find the missing data in the receiver end. The purpose of the sliding window technique is to avoid duplicate data, so it uses the sequence number.

CODE:-

```
#include<bits/stdc++.h>

#include<cstdio>

#include<iostream>

using namespace std;

int main(){

int f,w;

int cnt=0;

cout<<"\nEnter the Window Size : ";

cin>>w;
```

```

cout<<"Enter the number of frames : ";

cin>>f;

int i=1; int k=1;

cout<<"\nAfter sending "<<w<<" frames, sender waits for acknowledgment to be
sent by receiver."<<endl;

while(i<=f) {

if(i==1){

for(int j=1;j<=w && j<=f;j++){

    cout<<"\nFrame "<<j<<" is sent "<<endl;

    cnt++;

}

}

if(i != 1){

cout<<"Sent Frame "<<i<<endl;

cnt++;

}

cout<<"\nAcknowledgment is received for frame "<<k<<" by sender"<<endl;

k++;

cout<<endl;

if(i==1){

    i += w ;

}

else{

```



```
        i++;  
    }  
}  
  
for(int a=k; a<=f; a++){  
    cout<<"\nAcknowledgment is received for frame "<<a<<" by sender"<<endl;  
}  
cout<<"\nTotal number of transmissions : "<<cnt<<endl;  
return 0;  
}
```

OUTPUT:-

```
C:\Users\Ashish\Desktop\slidingwindow.exe
Enter the Window Size : 3
Enter the number of frames : 8

After sending 3 frames, sender waits for acknowledgment to be sent by receiver.

Frame 1 is sent
Frame 2 is sent
Frame 3 is sent

Acknowledgment is received for frame 1 by sender
Sent Frame 4

Acknowledgment is received for frame 2 by sender
Sent Frame 5

Acknowledgment is received for frame 3 by sender
Sent Frame 6

Acknowledgment is received for frame 4 by sender
Sent Frame 7

Acknowledgment is received for frame 5 by sender
Sent Frame 8

Acknowledgment is received for frame 6 by sender

Acknowledgment is received for frame 7 by sender
Acknowledgment is received for frame 8 by sender

Total number of transmissions : 8

-----
Process exited after 10.37 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have successfully implemented sliding window protocol and we learnt its application in computer network.

PROGRAM 5

Aim:- Write a program to implement and find Network Class, Network Id and Host Id of given IPV4 address.

Theory:- Given a valid IPv4 address in the form of string and it follows Class Full addressing. The 32 bit IP address is divided into five subclasses. These are:

1. Class A
2. Class B
3. Class C
4. Class D
5. Class E

Each of these classes has a valid range of IP addresses. Classes D and E are reserved for multicast and experimental purposes respectively.

- 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.
- For determining the Network and Host ID: We know that Subnet Mask for Class A is 8, for Class B is 16 and for Class C is 24 whereas Class D and E is not divided into Network and Host ID.

CODE:-

```
#include<iostream>

#include<bits/stdc++.h>

#include<string.h>

using namespace std;

char findClass(char str[])

{

char arr[4];
```

```
int i = 0;
while (str[i] != '.')
{
    arr[i] = str[i];
    i++;
}
i--;
int ip = 0, j = 1;
while (i >= 0)
{
    ip = ip + (str[i] - '0') * j;
    j = j * 10;
    i--;
}
if (ip >= 0 && ip <= 127)
    return 'A';
else if (ip >= 128 && ip <= 191)
    return 'B';
else if (ip >= 192 && ip <= 223)
    return 'C';
else if (ip >= 224 && ip <= 239)
    return 'D';
else
```

```
return 'E';
```

```
}
```

```
void separate(char str[], char ipClass)
```

```
{
```

```
char network[12], host[12];
```

```
for (int k = 0; k < 12; k++)
```

```
network[k] = host[k] = '\0';
```

```
if (ipClass == 'A')
```

```
{
```

```
int i = 0, j = 0;
```

```
while (str[j] != '.')
```

```
network[i++] = str[j++];
```

```
i = 0;
```

```
j++;
```

```
while (str[j] != '\0')
```

```
host[i++] = str[j++];
```

```
std::cout<<"\nNetwork ID is : "<<network;
```

```
std::cout<<"\nHost ID is : "<<host;
```

```
}
```

```
else if (ipClass == 'B')
```

```
{
```

```
int i = 0, j = 0, dotCount = 0;
```

```

while (dotCount < 2)
{
network[i++] = str[j++];
if (str[j] == '.')
dotCount++;
}

i = 0;

j++;

while (str[j] != '\0')
host[i++] = str[j++];

std::cout<<"\nNetwork ID is : "<< network;

std::cout<<"\nHost ID is : "<< host;

}

else if (ipClass == 'C')
{
int i = 0, j = 0, dotCount = 0;

while (dotCount < 3)
{
network[i++] = str[j++];

if (str[j] == '.')
dotCount++;
}

i = 0;

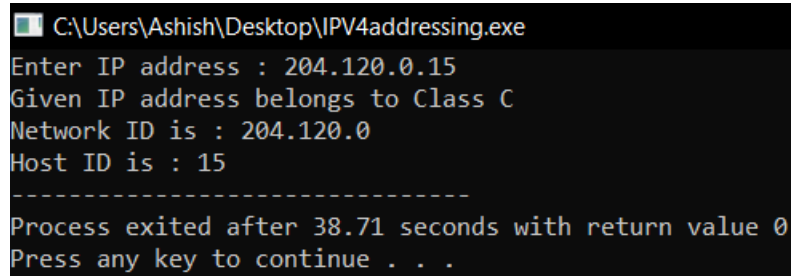
```

```
j++;  
while (str[j] != '\0')  
    host[i++] = str[j++];  
std::cout<<"\nNetwork ID is : "<< network;  
std::cout<<"\nHost ID is : "<< host;  
}
```

```
else  
    std::cout<<"\nIn this Class, IP address is not"  
    "divided into Network and Host ID\n";  
}
```

```
int main()  
{  
    char str[50];  
    std::cout<<"Enter IP address : ";  
    std::cin>>str;  
    char ipClass = findClass(str);  
    std::cout<<"Given IP address belongs to Class "<<ipClass;  
    separate(str, ipClass);  
    return 0;  
}
```

OUTPUT:-



```
C:\Users\Ashish\Desktop\IPv4addressing.exe
Enter IP address : 204.120.0.15
Given IP address belongs to Class C
Network ID is : 204.120.0
Host ID is : 15
-----
Process exited after 38.71 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have learned following things :

IPv4 address is divided into two parts:

- Network ID
- Host ID

The class of IP address is used to determine the bits used for network ID and host ID and the number of total networks and hosts possible in that particular class.

Each ISP or network administrator assigns IP address to each device that is connected to its network.

PROGRAM 6

Aim:- Write a program to implement distance vector routing(DVR) algorithm.

Theory:- A distance-vector routing algorithm also called the Bellman-Ford algorithm is a routing algorithm in which every router maintains a database with one entry for each possible destination on the network. It requires that a router inform its neighbours of topology changes periodically. Routers select the route with the lowest cost to each possible destination and add this to their own routing tables. These neighbors propagate the information to their neighbors hop by hop until information from all routers has spread throughout the entire internetwork.

CODE:-

```
#include<iostream>

using namespace std;

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

int main()
{
    int costmat[20][20];
    int nodes,i,j,k,count=0;
    cout<<"\nEnter the number of nodes : ";
```

```

cin>>nodes;

cout<<"\nEnter the cost matrix :\n";

for(i=0;i<nodes;i++)
for(j=0;j<nodes;j++)
{
cin>>costmat[i][j];

costmat[i][i]=0;

rt[i].dist[j]=costmat[i][j];

rt[i].parent[j]=j;

}

do

{

count=0;

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

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

for(k=0;k<nodes;k++)

if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])

{

rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];

rt[i].parent[j]=k;

count++;

}

}while(count!=0);

```

```
for(i=0;i<nodes;i++)
{
cout<<"\n\nFor " <<"router " <<i+1;
for(j=0;j<nodes;j++)
cout<<"\nNode " <<j+1<<" via " <<rt[i].parent[j]+1<<" Distance " <<rt[i].dist[j]<<"
";
}
cout<<endl;
return 0;
}
```

OUTPUT:-

```
C:\Users\Ashish\Downloads\CN LAB\distance_vector_routing.exe

Enter the number of nodes : 3

Enter the cost matrix :
1 2 3
4 6 1
6 7 9

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 4
Node 2 via 2 Distance 0
Node 3 via 3 Distance 1

For router 3
Node 1 via 1 Distance 6
Node 2 via 2 Distance 7
Node 3 via 3 Distance 0

-----
Process exited after 4.8 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have learned following things :

1. Distance vector algorithm is a dynamic algorithm. Each of the routers present in the network maintains a distance vector to store information regarding the shortest route to any given node from that node.
2. **Advantages of Distance Vector routing :-**
 - It is easy to administer and implement.
 - It is simpler to configure and maintain than link state routing.
 - Requires less hardware and processing power than other routing methods.

PROGRAM 7

Aim:- Write a program to implement link state routing algorithm.

Theory:- Link state routing is a technique in which each router shares the knowledge of its neighborhood with every other router in the internetwork.

Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation.

Features of link state routing protocols –

- Link state packet – A small packet that contains routing information.
- Link state database – A collection information gathered from link state packet.
- Shortest path first algorithm (Dijkstra algorithm) – A calculation performed on the database results into shortest path.

To find shortest path, each node need to run the famous Dijkstra algorithm. This famous algorithm uses the following steps:

- Step-1: The node is taken and chosen as a root node of the tree, this creates the tree with a single node, and now set the total cost of each node to some value based on the information in Link State Database
- Step-2: Now the node selects one node, among all the nodes not in the tree like structure, which is nearest to the root, and adds this to the tree. The shape of the tree gets changed.
- Step-3: After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.
- Step-4: The node repeats the Step 2. and Step 3. until all the nodes are added in the tree.

CODE:-

```
#include<iostream>

#include<bits/stdc++.h>

#include <string.h>

using namespace std;

int main()

{

int count,src_router,i,j,k,w,v,n;

int cost_matrix[100][100],dist[100],last[100],min;

int flag[100];

cout<<"\n Enter the number of routers : ";

cin>>count;

cout<<"\n Enter the cost matrix values : \n";

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

{

for(j=0;j<count;j++)

{

cout<<" \n "<<i<<" to "<<j<<": ";

cin>>cost_matrix[i][j];

if(cost_matrix[i][j]<0)cost_matrix[i][j]=1000;

}

}

cout<<"\n Enter the source router: ";
```

```
cin>>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++)
{
cout<<"\n "<<src_router<<" to "<<i<<" : Path taken : "<<i;
w=i;
while(w!=src_router)
{
cout<<" <-- "<<last[w];
w=last[w];
}
cout<<"\n Shortest path cost: " <<dist[i]<<"\n";
}
return 0;
}

```


OUTPUT:-

```
C:\Users\Ashish\Desktop\Link State Routing Algorithm.exe

Enter the number of routers : 3

Enter the cost matrix values :

0 to 0: 2
0 to 1: 1
0 to 2: 4
1 to 0: 2
1 to 1: 4
1 to 2: 6
2 to 0: 3
2 to 1: 7
2 to 2: 8

Enter the source router: 0

0 to 0 : Path taken : 0
Shortest path cost: 2

0 to 1 : Path taken : 1 <-- 0
Shortest path cost: 1

0 to 2 : Path taken : 2 <-- 0
Shortest path cost: 4

-----
Process exited after 360.1 seconds with return value 0
Press any key to continue . . .
```

Learning Outcome:- We have learned and understand the concept of link state protocol and found this :

Link State protocols in comparison to Distance Vector protocols have:

1. It requires large amount of memory.
2. Shortest path computations require many CPU cycles.
3. If network use the little bandwidth; it quickly reacts to topology changes
4. All items in the database must be sent to neighbors to form link state packets.
5. All neighbors must be trusted in the topology.