CN LAB RECORD(CYCLE-2)

Name : N.Akhilesh

USN:1BM19CS092

Section:5B

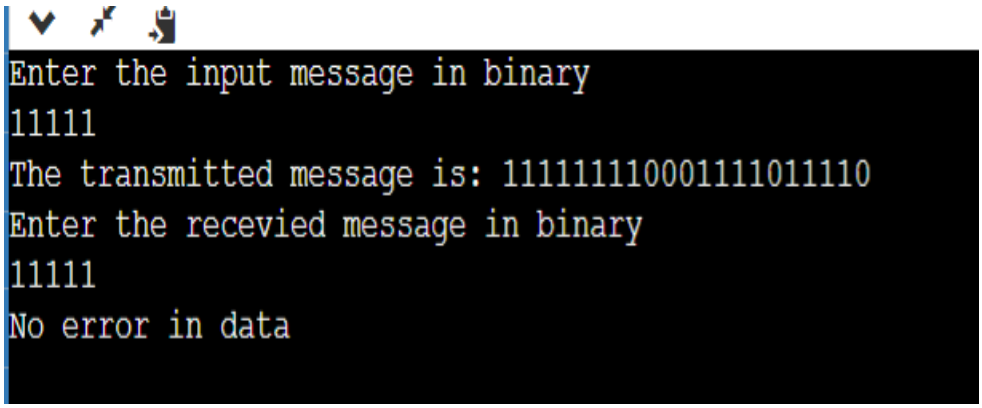
1. Write a program for error detecting code using CRC-CCITT (16 BITS).

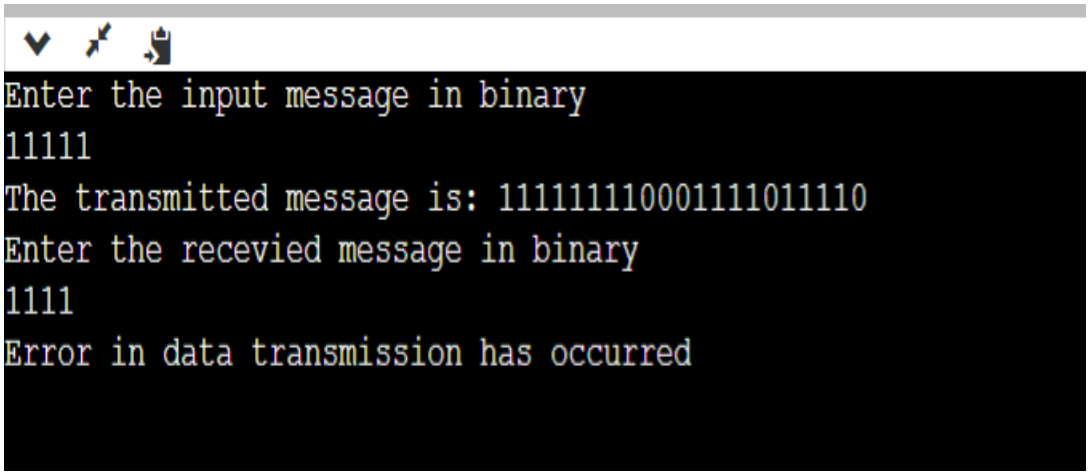
i) Program

```
#include <iostream>
#include <string.h>
using namespace std;
int crc(char *ip, char *op, char *poly, int mode)
{
 strcpy(op, ip);
 if (mode) {
 for (int i = 1; i < strlen(poly); i++)
 strcat(op, "0");
 }
 /* Perform XOR on the msg with the selected polynomial */
 for (int i = 0; i < strlen(ip); i++) {
 if (op[i] == '1') {
 for (int j = 0; j < strlen(poly); j++) {
 if (op[i + j] == poly[j])
 op[i + j] = '0';
 else
 op[i + j] = '1';
```

```
 }
 }
 }
 /* check for errors. return 0 if error detected */
 for (int i = 0; i < strlen(op); i++)
 if (op[i] == '1')
 return 0;
 return 1;
}
int main()
{
 char ip[50], op[50], recv[50];
 char poly[] = "10001000000100001";
 cout << "Enter the input message in binary"<< endl;
 cin >> ip;
 crc(ip, op, poly, 1);
 cout << "The transmitted message is: " << ip << op + strlen(ip) << endl;
 cout << "Enter the recevied message in binary" << endl;
 cin >> recv;
 if (crc(recv, op, poly, 0))
 cout << "No error in data" << endl;
 else
 cout << "Error in data transmission has occurred" << endl;
 return 0;
}
```

ii) Output


Enter the input message in binary
11111
The transmitted message is: 111111110001111011110
Enter the recevied message in binary
11111
No error in data


Enter the input message in binary
11111
The transmitted message is: 111111110001111011110
Enter the recevied message in binary
1111
Error in data transmission has occurred

2. Write a program for distance vector algorithm to find suitable path for transmission.

i. Program

```
class Topology:

    def __init__(self, array_of_points):
        self.nodes = array_of_points
        self.edges = []

    def add_direct_connection(self, p1, p2, cost):
        self.edges.append((p1, p2, cost))
        self.edges.append((p2, p1, cost))

    def distance_vector_routing(self):
        import collections
        for node in self.nodes:
            dist = collections.defaultdict(int)
            next_hop = {node: node}
            for other_node in self.nodes:
                if other_node != node:
                    dist[other_node] = 100000000 # infinity

            # Bellman Ford Algorithm
            for i in range(len(self.nodes)-1):
                for edge in self.edges:
                    src, dest, cost = edge
```

```
                if dist[src] + cost < dist[dest]:
                    dist[dest] = dist[src] + cost
                    if src == node:
                        next_hop[dest] =dest
                    elif src in next_hop:
                        next_hop[dest] = next_hop[src]

            self.print_routing_table(node, dist, next_hop)
            print()

    def print_routing_table(self, node, dist, next_hop):
        print(f'Routing table for {node}:')
        print('Dest \t Cost \t Next Hop')
        for dest, cost in dist.items():
            print(f'{dest} \t {cost} \t {next_hop[dest]}')

    def start(self):
        pass

nodes = ['A', 'B', 'C', 'D', 'E']

t = Topology(nodes)

t.add_direct_connection('A', 'B', 1)
t.add_direct_connection('A', 'C', 5)
t.add_direct_connection('B', 'C', 3)
```

t.add_direct_connection('B', 'E', 9)

t.add_direct_connection('C', 'D', 4)

t.add_direct_connection('D', 'E', 2)

t.distance_vector_routing()

ii. Output

```
Routing table for A:
Dest      Cost     Next Hop
B         1        B
C         4        B
D         8        B
E         10       B
A         0        A

Routing table for B:
Dest      Cost     Next Hop
A         1        A
C         3        C
D         7        C
E         9        E
B         0        B

Routing table for C:
Dest      Cost     Next Hop
A         4        B
B         3        B
D         4        D
E         6        D
C         0        C

Routing table for D:
Dest      Cost     Next Hop
A         8        C
B         7        C
C         4        C
E         2        E
D         0        D

Routing table for E:
Dest      Cost     Next Hop
A         10       B
B         9        B
C         6        D
D         2        D
E         0        E
```

3. Implement Dijkstra's algorithm to compute the shortest path for a given topology.

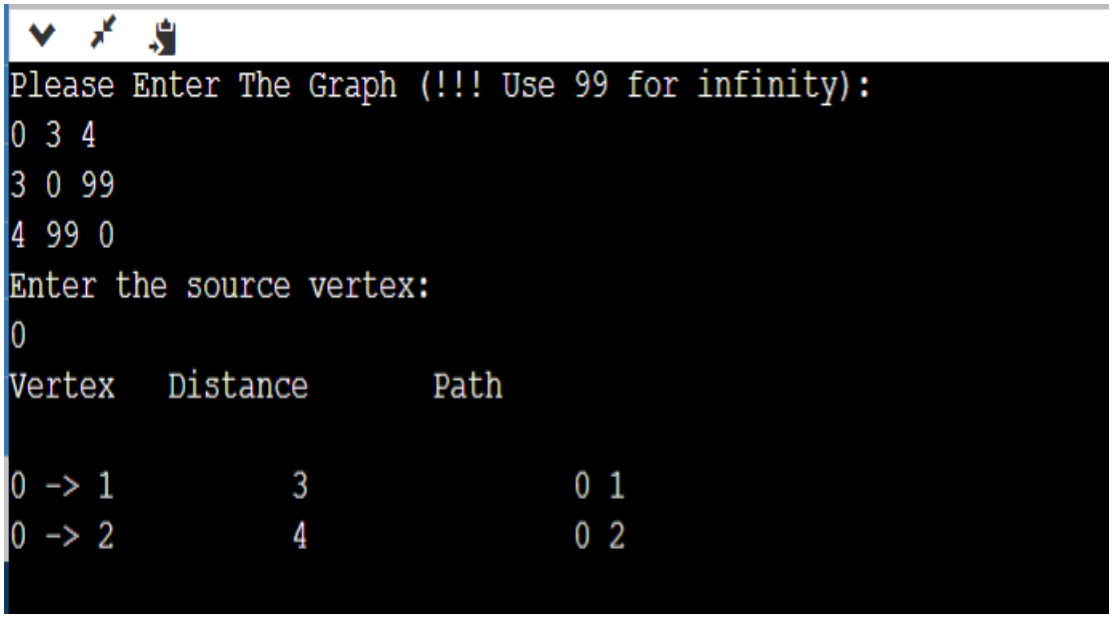
i. Program

```
#include<bits/stdc++.h>
using namespace std;

#define V 9

int minDistance(int dist[],
bool sptSet[])
{
int min = 9999,
min_index; for (int v = 0; v
< V; v++)
if (sptSet[v] == false &&
dist[v] <= min) min =
dist[v], min_index = v;

return min_index;
}

void printPath(int
parent[], int j)
{
if (parent[j] == - 1) return;

printPath(parent,
parent[j]);

cout<<j<<" ";
}

void printSolution(int
dist[], int n, int parent[])
{
int src = 0;
```

```
cout<<"Vertex\t
Distance\tPath"<<endl;
for (int i = 1; i < V; i++)
{
cout<<"\n"<<src<<" ->
"<<i<<" \t
"<<dist[i]<<"\t\t"<<src<<"
"; printPath(parent, i);
}
}

void dijkstra(int
graph[V][V], int src)
{

int dist[V]; bool sptSet[V];
int parent[V];
for (int i = 0; i < V; i++)
{

parent[0] = -1;
dist[i] = 9999; sptSet[i] =
false;
}

dist[src] = 0;

for (int count = 0; count <
V - 1; count++)
{
int u = minDistance(dist,
sptSet); sptSet[u] = true;
for (int v = 0; v < V; v++)

if (!sptSet[v] &&
graph[u][v] && dist[u] +
graph[u][v] < dist[v])
{
parent[v] = u;
```

```
dist[v] = dist[u] +
graph[u][v];
}
}

printSolution(dist, V,
parent);
}

int main()
{
int graph[V][V];
cout<<"Enter the graph
(Enter 99 for infinity):
"<<endl; for(int i = 0; i<V;
i++)
{
for(int j = 0; j<V; j++)
cin>>graph[i][j];
}
cout<<"Enter the source:
"<<endl; int src;
cin>>src;

dijkstra(graph, src);
cout<<endl;
return 0;
}
```

ii. Output


Please Enter The Graph (!!! Use 99 for infinity):
0 3 4
3 0 99
4 99 0
Enter the source vertex:
0
Vertex Distance Path
0 -> 1 3 0 1
0 -> 2 4 0 2

4. Write a program for congestion control using leaky bucket algorithm.

i. Program

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

#define NOF_PACKETS 10

int ran(int a)
{
   int rn = (rand() % 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_time, op;
   for(i = 0; i<NOF_PACKETS; ++i)
      packet_sz[i] = ran(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 = ran(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!!");
            }
         }
      }
   }
}
```

ii. Output

C:\WINDOWS\SYSTEM32\cmd.exe

```
packet[2]:40 bytes
packet[3]:10 bytes
packet[4]:30 bytes
packet[5]:40 bytes
packet[6]:20 bytes
packet[7]:20 bytes
packet[8]:20 bytes
packet[9]:40 bytes
Enter the Output rate:10
Enter the Bucket Size:15


Incoming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0

Incoming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0

Incoming packet size (40bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

Incoming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0

Incoming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

Incoming packet size (40bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

Incoming packet size (20bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

Incoming packet size (20bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

Incoming packet size (20bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

Incoming packet size (40bytes) is Greater than bucket capacity (15bytes)-PACKET REJECTED

-----------------
(program exited with code: 0)

Press any key to continue . . .
```

5. Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

i. Program

Client_TCP.py

```
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input("\nEnter file name: ")
clientSocket.send(sentence.encode())
filecontents = clientSocket.recv(1024).decode()
print ('\nFrom Server:\n')
print(filecontents)
clientSocket.close()
```

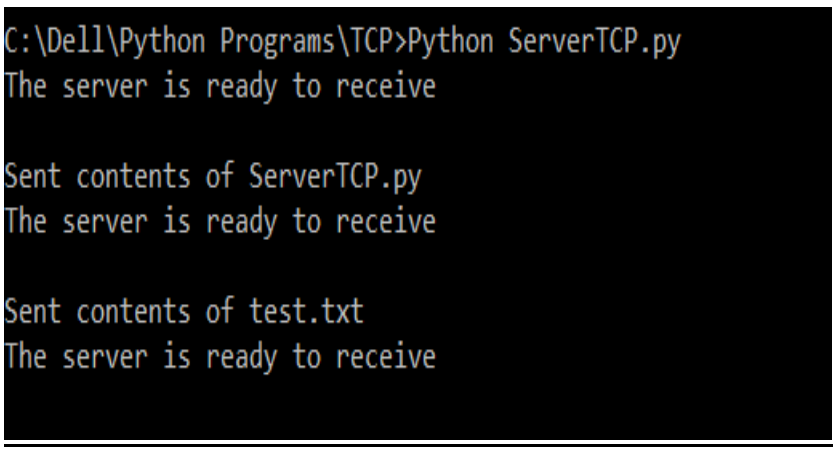
Server_TCP.py

```
from socket import *
serverName="127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((serverName,serverPort))
serverSocket.listen(1)
while 1:
 print ("The server is ready to receive")
 connectionSocket, addr = serverSocket.accept()
```

```
sentence = connectionSocket.recv(1024).decode()

file=open(sentence,"r")
l=file.read(1024)

connectionSocket.send(l.encode())
print ('\nSent contents of ' + sentence)
file.close()
connectionSocket.close()
```

ii. Output

Server_TCP.py



Client_TCP.py

Command Prompt

```
from socket import *
serverName="127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((serverName,serverPort))
serverSocket.listen(1)
while 1:
 print ("The server is ready to receive")
 connectionSocket, addr = serverSocket.accept()
 sentence = connectionSocket.recv(1024).decode()

 file=open(sentence,"r")
 l=file.read(1024)

 connectionSocket.send(l.encode())
 print ('\nSent contents of ' + sentence)
 file.close()
 connectionSocket.close()


C:\Dell\Python Programs\TCP>Python ClientTCP.py

Enter file name: test.txt

From Server:

Hello world! I was sent by the TCP Server.
```

6. Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

i. Program

Server_UDP.py

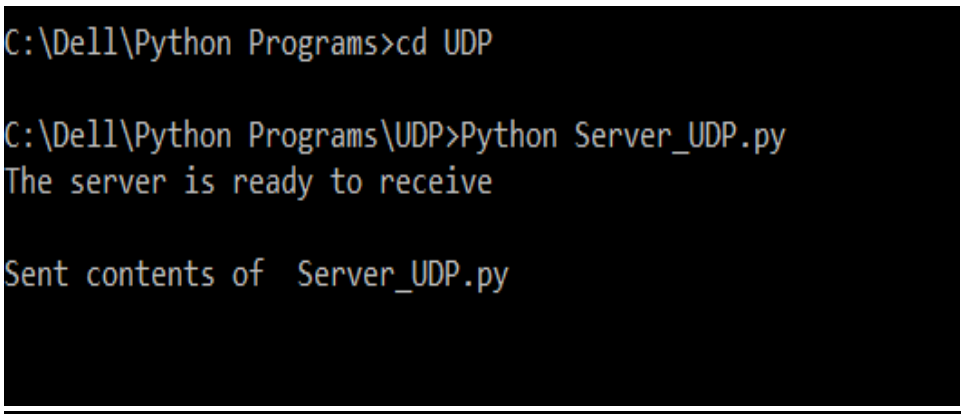
```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("127.0.0.1", serverPort))
print ("The server is ready to receive")
while 1:
 sentence, clientAddress = serverSocket.recvfrom(2048)
 sentence = sentence.decode("utf-8")
 file=open(sentence,"r")
 l=file.read(2048)

 serverSocket.sendto(bytes(l,"utf-8"),clientAddress)
 print ('\nSent contents of ', end = ' ')
 print (sentence)
 # for i in sentence:
 # print (str(i), end = '')
 file.close()
```

Client_UDP.py

```
from socket import *
serverName = "127.0.0.1"
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
sentence = input("\nEnter file name: ")
clientSocket.sendto(bytes(sentence,"utf-8"),(serverName, serverPort))
filecontents,serverAddress = clientSocket.recvfrom(2048)
print ('\nReply from Server:\n')
print (filecontents.decode("utf-8"))
# for i in filecontents:
 # print(str(i), end = '')
clientSocket.close()
clientSocket.close()
```

ii. Output

Server_UDP.py



Client_UDP.py

```
C:\Dell\Python Programs\UDP>Python Client_UDP.py

Enter file name: Server_UDP.py

Reply from Server:

from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("127.0.0.1", serverPort))
print ("The server is ready to receive")
while 1:
 sentence, clientAddress = serverSocket.recvfrom(2048)
 sentence = sentence.decode("utf-8")
 file=open(sentence,"r")
 l=file.read(2048)

 serverSocket.sendto(bytes(l,"utf-8"),clientAddress)
 print ('\nSent contents of ', end = ' ')
 print (sentence)
 # for i in sentence:
 # print (str(i), end = '')
 file.close()
```