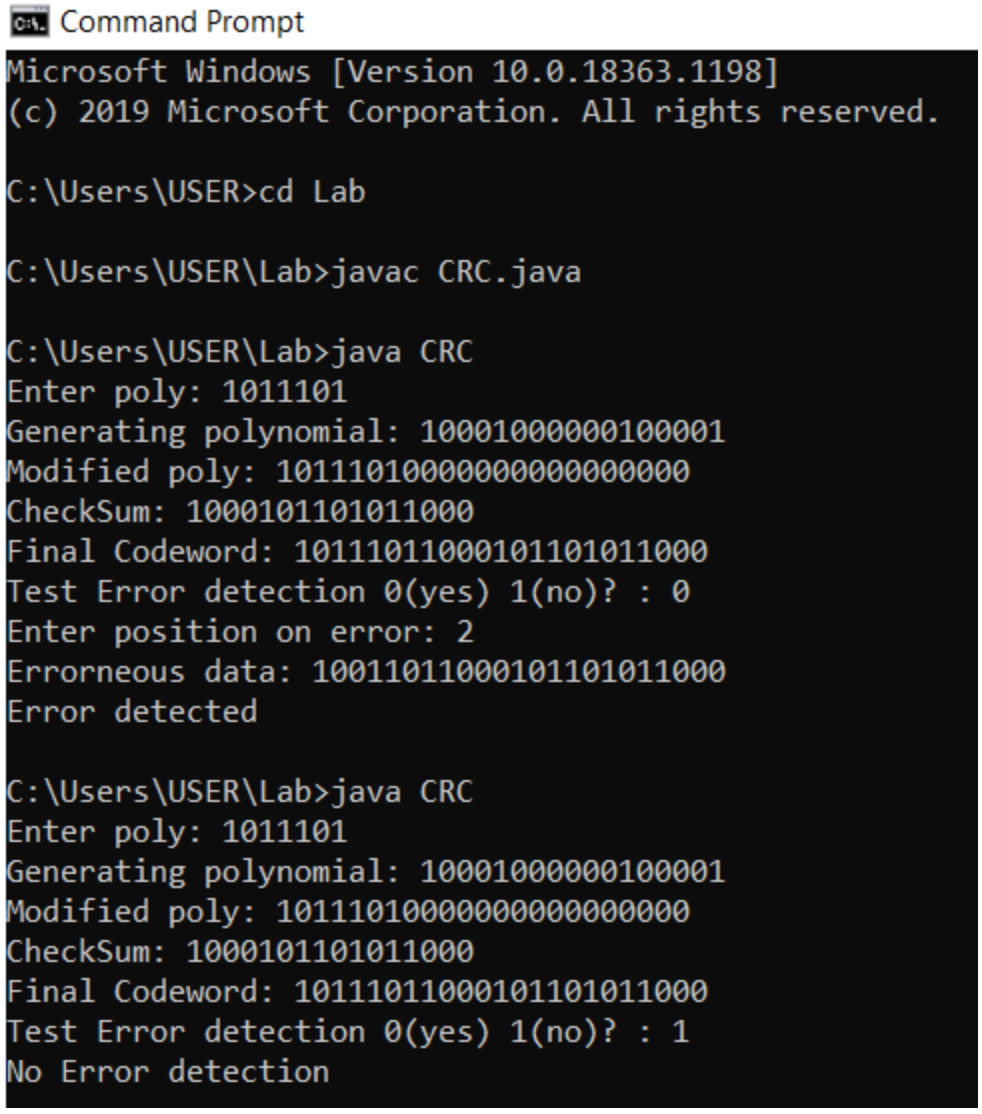
1.Write a program for error detecting code using CRC-CCITT (16-bits)

```
import java.util.*;
public class Crc
{
	public static int n;
	public static void main(String[] args)
{
		Scanner in=new Scanner(System.in);
		 Crc ob=new Crc();
		 String code, copy, rec,zero="0000000000000000";
		 System.out.println("Enter message");
		 code=in.nextLine();
		 n=code.length();
		 copy=code;
		 code+=zero;
		 code=ob.divide(code);
		 System.out.println("Message="+copy);
		 copy=copy.substring(0,n)+code.substring(n);
		 System.out.println("CRC=");
		 System.out.println(code.substring(n));
		 System.out.println("transmitted frame is "+copy);
		 System.out.println("Enter recived data");
		 rec=in.nextLine();
		 if(zero.equals(ob.divide(rec).substring(n)))
```

```
                System.out.println("Correct bits recieved");
        else
                System.out.println("Recieved frame contains one or more
errors");
        in.close();
    }
    public String divide(String s)
{
                int i,j;
                char x;
                String div="10001000000100001";
                for(i=0;i<n;i++)
                {
                    x=s.charAt(i);
                    for(j=0;j<17;j++)
                    {
                        if(x=='1')
                        {if(s.charAt(i+j)!=div.charAt(j))
                            s=s.substring(0,i+j)+"1"+s.substring(i+j+1);
                        else
                            s=s.substring(0,i+j)+"0"+s.substring(i+j+1);
                        }
                    }
                }
                return s;
    }
}
```



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

```
class Topology:
    def __init__(self, n):
        self.matrix = []
        self.n = n

    def addlink(self, u, v, w):
        self.matrix.append((u, v, w))

    def table(self, dist, src):
        print("Vector Table of {}".format(chr(ord('A')+src)))
        print("Dest\tcost")
```

```
        for i in range(self.n):
            print("{0}\t{1}".format(chr(ord('A')+i), dist[i]))

    def bellmanford(self, src):
        dist = [9999] * self.n
        dist[src] = 0

        for _ in range(self.n - 1):
            for u, v, w in self.matrix:
                if dist[u] != 9999 and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w

        self.table(dist, src)


def main():
    matrix = []
    n = int(input("Enter number of Nodes : "))
    print("Enter the Adjacency Matrix :")
    for i in range(n):
        m = list(map(int, input().strip().split()))
        matrix.append(m)
    topo = Topology(n)
    for i in range(n):
        for j in range(n):
            if matrix[i][j] == 1:
                topo.addlink(i, j, 1)

    for a in range(n):
        topo.bellmanford(a)


main()
```

```
Enter number of Nodes : 5
Enter the Adjacency Matrix :
0 1 1 9999 9999
1 0 9999 9999 9999
1 9999 0 1 1
9999 9999 1 0 9999
9999 9999 1 9999 0
Vector Table of A
Dest    cost
A       0
B       1
C       1
D       2
E       2
Vector Table of B
Dest    cost
A       1
B       0
C       2
D       3
E       3
Vector Table of C
Dest    cost
A       1
B       2
C       0
D       1
E       1
Vector Table of D
Dest    cost
A       2
B       3
C       1
D       0
E       2
Vector Table of E
Dest    cost
A       2
B       3
C       1
D       2
E       0
```

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

```
#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;
}
```

```
Please Enter The Graph (!!! Use 99 for infinity):
0 4 99 99 99 99 99 8 99
4 0 8 99 99 99 99 11 99
99 8 0 77 99 4 99 99 2
99 99 7 0 9 14 99 99 99
99 99 99 9 0 10 9 99 99
99 99 4 99 10 0 2 99 99
99 99 99 14 99 2 0 1 6
8 11 99 99 99 99 1 0 7
99 99 2 99 99 99 6 7 0
Enter the source vertex:
0
Vertex   Distance   Path

0 -> 1      4       0 1
0 -> 2      12      0 1 2
0 -> 3      23      0 7 6 3
0 -> 4      21      0 7 6 5 4
0 -> 5      11      0 7 6 5
0 -> 6      9       0 7 6
0 -> 7      8       0 7
0 -> 8      14      0 1 2 8
```

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

```
#include<bits/stdc++.h>
#include<unistd.h>
using namespace std;
#define bucketSize 500

void bucketInput(int a,int b)
{
	if(a > bucketSize)
		cout<<"\n\t\tBucket overflow";
	else{
		sleep(5);
		while(a > b){
			cout<<"\n\t\t"<<b<<" bytes outputted.";
			a-=b;
			sleep(5);
		}
		if(a > 0)
			cout<<"\n\t\tLast "<<a<<" bytes sent\t";
```

```
                cout<<"\n\t\tBucket output successful";
        }
}
int main()
{
        int op,pktSize;
        cout<<"Enter output rate : ";
        cin>>op;
        for(int i=1;i<=5;i++)
        {
                sleep(rand()%10);
                pktSize=rand()%700;
                cout<<"\nPacket no "<<i<<"\tPacket size = "<<pktSize;
                bucketInput(pktSize,op);
        }
        cout<<endl;
        return 0;
}
```

"C:\Users\Prashanth\Documents\java programs\dij.exe"

```
Enter output rate : 100

Packet no 1     Packet size = 267
                100 bytes outputted.
                100 bytes outputted.
                Last 67 bytes sent
                Bucket output successful
Packet no 2     Packet size = 600
                Bucket overflow
Packet no 3     Packet size = 324
                100 bytes outputted.
                100 bytes outputted.
                100 bytes outputted.
                Last 24 bytes sent
                Bucket output successful
Packet no 4     Packet size = 658
                Bucket overflow
Packet no 5     Packet size = 664
                Bucket overflow

Process returned 0 (0x0)   execution time : 61.603 s
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.

Client.py

```
import socket

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 65432

print('\033[32m======== CLIENT ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((SERVER_HOST, SERVER_PORT))
    while True:
        filename = input('Enter file name: ')

        if not filename:
            break

        sock.sendall(bytes(filename, 'utf-8'))
        print(f'Sent: {filename}')

        data = sock.recv(1024)
        contents = data.decode('utf-8')
        print(f'Received: {contents}')
        print()
```

Server.py

```
import socket

HOST = '127.0.0.1'
PORT = 65432

print('\033[36m======== SERVER ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.bind((HOST, PORT))
    sock.listen(1)
    conn, addr = sock.accept()
```

```
with conn:
    print(f'Connected by: {addr}')
    while True:
        data = conn.recv(1024)
        if not data:
            break

        filename = data.decode('utf-8')
        print(f'Received Filename: {filename}')

        try:
            with open(filename, 'r') as f:
                data = f.read()
            data = bytes(data, 'utf-8')
        except:
            data = bytes(f'File {filename} not found', 'utf-8')

        conn.sendall(data)
        print(f'Sent: {data}')
        print()
```

```
======== CLIENT ========
Enter file name: testfile.txt
Sent: testfile.txt
Received: Hello world! I was sent by the TCP Server.

Enter file name: nofile
Sent: nofile
Received: File nofile not found

Enter file name:
```

```
======== SERVER ========
Connected by: ('127.0.0.1', 45380)
Received Filename: testflle.txt
Sent: b'File testflle.txt not found'

Received Filename: nofile
Sent: b'File nofile not found'
```

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.

Client.py

```
import socket

HOST = '127.0.0.1'
PORT = 65432

print('\033[32m======== CLIENT ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    sock.connect((HOST, PORT))
    while True:
        filename = input('Enter file to request from server: ')

        if not filename:
            break

        sock.sendall(bytes(filename, 'utf-8'))
        print(f'Sent: {filename}')

        data = sock.recv(1024).decode('utf-8')
        print(f'Received: {data}')
        print()
```

Server.py

```
import socket

HOST = '127.0.0.1'
PORT = 65432

print('\033[36m======== SERVER ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    sock.bind((HOST, PORT))
    while True:
        data, addr = sock.recvfrom(1024)
        if not data:
            break
```

```
filename = data.decode('utf-8')
print(f'Received Filename: {filename} From: {addr}')

try:
    with open(filename, 'r') as f:
        data = f.read()
    data = bytes(data, 'utf-8')
except:
    data = bytes(f'File {filename} not found', 'utf-8')

sock.sendto(data, addr)
print(f'Sent: {data} To: {addr}')
print()
```