NAME : Y PRAHASITH
USN : 01JST18CS177
ROLL NO: 60
SEM & SEC : 5 A

COMPUTER NETWORKS LAB

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

```
->  import java.util.Scanner;
public class CRC16CCITT
 {
    public static void main(String[] args) {
        int crc = 0xFFFF;
        int polynomial = 0x1021;
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        sc.close();
        byte[] bytes = s.getBytes();
        for (byte b: bytes) {
            for (int i = 0; i < 8; i++) {
              boolean bit = ((b >> (7-i) & 1) == 1);
              boolean c15 = ((crc >> 15 & 1) == 1)
                crc <<= 1;
                if (c15 ^ bit) {
                    crc ^= polynomial;
                }
            }
        }
        crc &= 0xffff;
        System.out.println("CRC16-CCITT = " +
Integer.toHexString(crc));
    }
}
```

2. Write a program to find the shortest path between vertices using bellman-ford algorithm.

```
-> import java.util.*;
class BellmanFord {
```

```
static ArrayList<ArrayList<Integer[]>>
readGraph(Scanner sc) {
        int n = sc.nextInt();
        int e = sc.nextInt();
        ArrayList<ArrayList<Integer[]>> graph =
new ArrayList<ArrayList<Integer[]>>(n);
        for (int i = 0; i < n; i++) {
            graph.add(new
ArrayList<Integer[]>());
        }
        for (int i = 0; i < e; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            int w = sc.nextInt();
            Integer[] edge = {v, w};
            graph.get(u).add(edge);
        }
        return graph;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<ArrayList<Integer[]>> graph =
readGraph(sc);
        int n = graph.size();
        int s = sc.nextInt();
        sc.close();
        int[] dist = new int[n];
        Integer[] prev = new Integer[n];
        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
            prev[i] = null;
        }
        dist[s] = 0;
        for (int i = 1; i < n; i++) {
       for (int source = 0; source < n; source++) {
       for (Integer[] edge: graph.get(source)) {
      if (dist[edge[0]] > dist[source] + edge[1]) {
        dist[edge[0]] = dist[source] + edge[1];
```

```
                 prev[edge[0]] = source;
                          }
                     }
                }
           }
           boolean nCycle = false;
          for (int source = 0; source < n; source++) {
           for (Integer[] edge: graph.get(source)) {
            if (dist[edge[0]] > dist[source] + edge[1])
{
                          nCycle = true;
                          dist[edge[0]] = dist[source] +
edge[1];
                          prev[edge[0]] = source;
                     }
                }}
           if (nCycle) {
           System.out.println("The graph contains
negative weight cycle.");
```

```
  }
      else {
          System.out.println("The graph does not
contain negative weight cycle.");
      }
      System.out.println(Arrays.toString(dist));
      System.out.println(Arrays.toString(prev));
   }
}
```

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

-> Here we have two programs , the client side and the server side program.

Client - program :

```
import socket
HOST = "127.0.0.1"
PORT = 55007
```

```
if __name__ == "__main__":
   filename = input("Enter filename: ")
with
socket.socket(socket.AF_INET,socket.SOCK_ST
as s:
      s.connect((HOST, PORT))
      s.sendall(bytes(filename, "UTF-8"))
      contents = ""
      while True:
         data = s.recv(1024)
         if not data:
            break
         contents += data.decode("UTF-8")
   print(f"Contents of file
'{filename}":\n\n{contents}")
```

Server- Program :

```
import socket
HOST = ""
```

```
PORT = 55007

if __name__ == "__main__":
    with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen(1)
        conn, addr = s.accept()
        with conn:
            print(f"Connected accepted from
{addr}")
            data = conn.recv(1024)
            try:
                file = open(data.decode("UTF-8"), "r")
                contents = file.read()
                file.close()
            except FileNotFoundError as error:
                contents = f"ERROR: {error}"
            conn.sendall(bytes(contents, "UTF-8"))
```

4. Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

-> Here also we have two programs , one works on the client end and the other one for the server side.

Client - side program :

```
import socket
HOST = '127.0.0.1'
PORT = 65432
BYTE_SIZE = 1024

with socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) as s:
    s.bind((HOST,PORT))
    print('Successfully connected to server!')
    while True:
        message, _ = s.recvfrom(BYTE_SIZE)
        if not message:
            print()
            break
        print(message.decode(), end='')
    s.close()
```

Server-side program :

```
import socket
HOST = '127.0.0.1'
PORT = 65432
BYTE_SIZE = 1024

with socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) as s:
    while True:
        message = input('Message = ')
        if not message:
            s.sendto(b'', (HOST, PORT))
            break
        message += '\n'
        s.sendto(message.encode(), (HOST, PORT)
    s.close()
```

5. Write a program for simple RSA algorithm to encrypt and decrypt the data.

```
-> from Crypto.Util import number
def gen_keys():
    bit_length = 256
    p = number.getPrime(bit_length)
    while True:
        q = number.getPrime(bit_length)
        if p != q:
            break
    N = p * q
    ctf = (p-1) * (q-1)
    while True:
        e = number.getPrime(8)
        if ctf % e != 0:
            break
    d = number.inverse(e, ctf)
    return (e, N), (d, N)

def encrypt(plaintext, public_key):
    num = int(plaintext)
    enc = pow(num, public_key[0], public_key[1])
    return enc

def decrypt(ciphertext, private_key):
    cipher = int(ciphertext)
    dec = pow(cipher, private_key[0],
private_key[1])
    return dec

if __name__ == '__main__':
    public_key, private_key = gen_keys()
    while True:
        print()
        print('1. Encrypt\n2. Decrypt\n3. Exit')
        choice = int(input())

        if choice == 1:
            plaintext = input('Enter text to be
encrypted: ')
            print("Encrypted = ", encrypt(plaintext,
public_key))

        elif choice == 2:
            ciphertext = input('Enter text to be
decrypted: ')
            print("Decrypted = ", decrypt(ciphertext
private_key))

        else:
            break
```

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

```
-> import java.util.Scanner;
public class LeakyBucket
{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int capacity = sc.nextInt();
        int inflow = sc.nextInt();
        int outflow = sc.nextInt();
        int n = sc.nextInt();
        sc.close();
        int filled = 0;
        while (n != 0 ) {
            if (inflow <= (capacity-filled)) {
                filled += inflow;
            }
            else {
                System.out.println((inflow - capacity
+ filled) + "packets overflowed and discard.");
                filled = capacity;
            }
            filled -= outflow;
            n--;
            System.out.println(filled + " out of " +
capacity + " packets remaining in bucket.");
            try {
                Thread.sleep(2000);
            }
            catch (Exception e) {
                continue;
            }
        }
    }
}
```