**HAMMING**

def calcRedundantBits(m):

for i in range(m):

if(2\*\*i >= m + i + 1):

return i

def posRedundantBits(data, r):

j = 0

k = 1

m = len(data)

res = ''

for i in range(1, m+r+1):

if(i == 2\*\*j):

res = res + '0'

j += 1

else:

res = res + data[-1 \* k]

k += 1

return res[::-1]

def calcParityBits(arr, r):

n = len(arr)

for i in range(r):

val = 0

for j in range(1, n + 1):

if(j & (2\*\*i) == (2\*\*i)):

val = val ^ int(arr[-1 \* j])

arr = arr[:n-(2\*\*i)] + str(val) + arr[n-(2\*\*i)+1:]

return arr

# 1011

# 1010101 r=3

# 1 0 1 0 1 0 0

def detectError(arr, nr):

n = len(arr)

res = 0

for i in range(nr):

val = 0

for j in range(1, n + 1):

if(j & (2\*\*i) == (2\*\*i)):

val = val ^ int(arr[-1 \* j])

res = res + val\*(10\*\*i)

return int(str(res), 2)

data = input("Enter the data bits : ")

m = len(data)

r = calcRedundantBits(m)

arr = posRedundantBits(data, r)

arr = calcParityBits(arr, r)

print("Data transferred is : " + arr)

recieved = input("Enter the recieved data : ")

print("Recieved Data is : " + recieved)

correction = detectError(recieved, r)

if(correction!=0):

print("The position of error is ",len(recieved)-correction+1," from the left")

else:

print("There is no error in recieved message")

**CRC**

def xor(a, b):

result = []

for i in range(1, len(b)):

if a[i] == b[i]:

result.append('0')

else:

result.append('1')

return ''.join(result)

def mod2div(dividend, divisor):

pick = len(divisor)

tmp = dividend[0: pick]

while pick < len(dividend):

if tmp[0] == '1':

tmp = xor(divisor, tmp) + dividend[pick]

else:

tmp = xor('0'\*pick, tmp) + dividend[pick]

pick += 1

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

tmp = xor('0'\*pick, tmp)

remainder = tmp

return remainder

def encodeData(data, divisor):

l\_divisor = len(divisor)

appended\_data = data + '0'\*(l\_divisor-1)

remainder = mod2div(appended\_data, divisor)

codeword = data + remainder

print("Remainder : ", remainder)

print("Encoded Data (Data + Remainder) : ", codeword)

data = input("Enter the data : ")

divisor = input("Enter the divisor : ")

encodeData(data, divisor)

data2 = input("Enter the recieved data : ")

rem = mod2div(data2,divisor)

if(int(rem)==0):

print("There is no error in recieved data")

else:

print("There is error in recieved data")

**GO BACK N**

#include<bits/stdc++.h>

#include<ctime>

#define ll long long int

using namespace std;

void transmission(ll &i,ll &N, ll &tf, ll &tt)

{

while(i<=tf)

{

int z=0;

for(int k=i;k<i+N&&k<=tf;k++)

{

cout<<"Sending frame "<<k<<"...."<<endl;

tt++;

}

for(int k=i;k<i+N&&k<=tf;k++)

{

int f=rand()%2;

if(!f)

{

cout<<"Acknoledgement for frame "<<k<<"...."<<endl;

z++;

}

else

{

cout<<"Timeout! Frame number "<<k<<" not rcieved"<<endl;

cout<<"Retransmitting Window ...."<<endl;

break;

}

}

cout<<endl;

i+=z;

}

}

int main()

{

ll tf,N,tt=0;

srand(time(NULL));

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

cin>>tf;

cout<<"enter the window size : ";

cin>>N;

ll i=1;

transmission(i,N,tf,tt);

cout<<"Total number of frames which were sent and resent are : "<<tt<<endl;

return 0;

}

**IPV4**

import math

def findClass(ip):

if 0 <= ip[0] <= 127:

print("Network Address is : ", ip[0])

print('No. of IP addresses possible : ', 2 \*\* 24)

return "A", '255.0.0.0'

elif 128 <= ip[0] <= 191:

ip = [str(i) for i in ip]

print("Network Address is : ", ".".join(ip[0:2]))

print('No. of IP addresses possible : ', 2 \*\* 16)

return "B", '255.255.0.0'

elif 192 <= ip[0] <= 223:

ip = [str(i) for i in ip]

print("Network Id is : ", ".".join(ip[0:3]))

print('No. of IP addresses possible : ', 2 \*\* 8)

return "C", '255.255.255.0'

elif 224 <= ip[0] <= 239:

print("In this Class, IP address is not divided into Network and Host ID")

return "D"

else:

print("In this Class, IP address is not divided into Network and Host ID")

return "E"

def Subnetting(ip, num, className, ip\_addresses):

temp = 0

if className == "A":

place2 = ip\_addresses / (256 \*\* 2)

for i in range(num):

print(f"Subnet {i} => ")

print(temp)

print("Subnet Address : ", ip[0] + '.' + str(temp) + '.0' + '.0')

temp += int(place2)

print("Broadcast address : ",ip[0] + '.' + str(temp - 1) + '.255' + '.255')

print("Valid range of host IP address : ", ip[0] + '.' + str(temp - int(place2)) + '.' + '0' + '.1' + '\t-\t' + ip[0] + '.' + str(temp - 1) + '.254' + '.254')

print()

elif className == "B":

place2 = ip\_addresses / 256

for i in range(num):

print(f"\nSubnet {i} => ")

print("Subnet Address : ", ".".join(ip[0:2]) + '.' + str(temp) + '.0')

temp += int(place2)

print("Broadcast address : ", ".".join(ip[0:2]) + '.' + str(temp - 1) + '.255')

print("Valid range of host IP address : ", ".".join(ip[0:2]) + '.' + str(temp - int(place2)) + '.1\t-\t' + ".".join(ip[0:2]) + '.' + str(temp - 1) + '.254')

print()

elif className == "C":

for i in range(num):

print(f"\nSubnet {i} => ")

print("Subnet Address : ", ".".join(ip[0:3]) + '.' + str(temp))

temp += int(ip\_addresses)

print("Broadcast address : ", ".".join(ip[0:3]) + '.' + str(temp - 1))

print("Valid range of host IP address : ", ".".join(ip[0:3]) + '.' + str(temp - int(ip\_addresses) + 1) + '\t-\t' + ".".join(ip[0:3]) + '.' + str(temp - 2))

print()

else:

print("In this Class, IP address is not divided into Network and Host ID")

def subnetmask(num, network\_mask):

var = '1' \* int(math.log(num, 2))

var1 = '0' \* (8 - int(math.log(num, 2)))

binary\_num = var + var1

network\_mask = network\_mask.split('.')

network\_mask = [i for i in network\_mask if i != '0']

network\_mask.append(str(int(binary\_num, 2)))

while len(network\_mask) < 5:

network\_mask.append('0')

print('Subnet Mask – ', ".".join(network\_mask[0:4]))

ip = input("Enter the IP address : ")

ip = ip.split(".")

ip = [int(i) for i in ip]

lst = findClass(ip)

networkClass = lst[0]

print("Given IP address belongs to class : ", networkClass)

ip = [str(i) for i in ip]

network\_mask = lst[1]

print('Network Mask : ', network\_mask)

num\_subnet = int(input('\nNo. of subnets(power of 2) : '))

num\_ip = int(2 \*\* (8 \* (68 - ord(networkClass))) / num\_subnet)

print('The no. of bits in the subnet id : ', int(math.log(num\_subnet, 2)))

if ord(networkClass) < 68:

print('Total no. of IP addresses possible in each subnet : ', num\_ip)

Subnetting(ip, num\_subnet, networkClass, num\_ip)

subnetmask(num\_subnet, network\_mask)

**MyServer**

import java.io.\*;

import java.net.\*;

public class MyServer{

public static void main(String[] args){

try{

ServerSocket ss = new ServerSocket(8080);

Socket s = ss.accept(); //establish connection

DataInputStream din = new DataInputStream(s.getInputStream());

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

String str = (String)din.readUTF();

System.out.println("Number : "+str);

int t = Integer.parseInt(str);

int ans = 1;

while(t>0){

ans\*=t;

t--;

}

dout.writeUTF(Integer.toString(ans));

dout.flush();

dout.close();

ss.close();

}catch(Exception e){

System.out.println(e);

}

}

}

**MyClient**

import java.io.\*;

import java.net.\*;

public class MyClient{

public static void main(String[] args){

try{

Socket s = new Socket("localhost",8080);

DataInputStream din = new DataInputStream(s.getInputStream());

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the number : ");

String str = bfr.readLine();

dout.writeUTF(str);

String str2 = din.readUTF();

System.out.println("The factorial of number is : "+str2);

dout.flush();

dout.close();

s.close();

}catch(Exception e){

System.out.println(e);

}

}

}