CB.EN.P2CSE19012(Maya Manish Kumar)

In [1]:

```
#QUES 1 generating prime numbers of order 10^20
#quick generation of nos
from random import randrange, getrandbits
def is_prime(n, m=10^20):
    # Test if n is not even.
    # But, 2 is prime !
    if n == 2 or n == 3:
        return True
    if n <= 1 or n % 2 == 0:
        return False
    # find r and s
    s = 0
    r = n - 1
    while r & 1 == 0:
        s += 1
        r //= 2
    # do m tests
    for _ in range(m):
        a = randrange(2, n - 1)
        x = pow(a, r, n)
        if x != 1 and x != n - 1:
            j = 1
            while j < s and x != n - 1:
                x = pow(x, 2, n)
                if x == 1:
                    return False
                j += 1
            if x != n - 1:
                return False
    return True
def generate_prime_candidate(length):
    # generate random bits
    p = getrandbits(length)
    # apply a mask to set MSB and LSB to 1
    p |= (1 << length - 1) | 1
    return p
def generate_prime_number(length=(10^20)):
    p = 200000000000000

    # keep generating while the primality test fail
    while not is_prime(p, 10^20):
        p = generate_prime_candidate(length)
    return p
print(generate_prime_number())
```

```
877476667
```

In [2]:

```
import time
def GeneratingTable():
    A = {v for i in range (1,10) for j in range (0,10) for m in range (0,10)
         for v in [i*1000000000+j*100000000+m*10000000+7777777,i*1000000000+j*100000000+
77777770+m,i*1000000000+777777700+10*j+m,7777777000+i*100+j*10+m,7777777000+j*10+m]}
    print(len(A))
    return A


def ifPrime(n):
    i = 2
    while i * i <= n:
        if n%i == 0:
            return False
        i += 1
    return True

def check():
    return sorted([p for p in GeneratingTable() if ifPrime(p)])


start = time.clock()
x = check()
print(len(x),x)
end = time.clock()
time = end - start
print(time)
```

```
C:\Users\mayam\anaconda3\lib\site-packages\ipykernel_launcher.py:21: DeprecationWarning:
time.clock has been deprecated in Python 3.3 and will be removed from Python 3.8: use tim
e.perf_counter or time.process_time instead
```

```
3420
203 [1247777777, 1277777771, 1277777773, 1457777777, 1487777777, 1577777771, 1577777777,
1657777777, 1777777741, 1777777751, 1777777777, 1787777777, 1877777773, 1877777777, 18777
77779, 1927777777, 1957777777, 2017777777, 2027777777, 2077777771, 2377777771, 2437777777
, 2467777777, 2507777777, 2567777777, 2647777777, 2677777771, 2777777707, 2777777711, 277
7777719, 2777777741, 2777777759, 2777777777, 2777777797, 2917777777, 3037777777, 30777777
77, 3137777777, 3197777777, 3247777777, 3257777777, 3377777773, 3377777779, 3407777777, 3
427777777, 3527777777, 3557777777, 3577777771, 3777777701, 3777777767, 3777777793, 382777
7777, 3937777777, 3977777773, 3977777777, 4027777777, 4097777777, 4177777771, 4277777773,
4297777777, 4307777777, 4327777777, 4447777777, 4567777777, 4687777777, 4747777777, 47777
77703, 4777777717, 4777777727, 4777777729, 4777777759, 4777777769, 4777777789, 4777777793
, 4777777799, 4867777777, 4937777777, 4997777777, 5177777777, 5237777777, 5387777777, 547
7777777, 5527777777, 5567777777, 5617777777, 5627777777, 5647777777, 5777777701, 57777777
71, 5777777791, 5877777779, 6037777777, 6077777773, 6077777777, 6177777773, 6277777777, 6
317777777, 6577777771, 6577777777, 6637777777, 6757777777, 6767777777, 6777777731, 677777
7737, 6777777757, 6777777791, 6847777777, 6857777777, 6947777777, 6977777771, 6977777773,
7037777777, 7087777777, 7327777777, 7387777777, 7487777777, 7537777777, 7547777777, 75977
77777, 7607777777, 7727777777, 7777777019, 7777777027, 7777777057, 7777777069, 7777777081
, 7777777103, 7777777127, 7777777169, 7777777199, 7777777207, 7777777211, 7777777229, 777
7777237, 7777777261, 7777777327, 7777777361, 7777777369, 7777777379, 7777777391, 77777774
21, 7777777429, 7777777453, 7777777493, 7777777517, 7777777549, 7777777577, 7777777597, 7
777777633, 7777777639, 7777777649, 7777777663, 7777777669, 7777777691, 7777777703, 777777
7741, 7777777781, 7777777783, 7777777789, 7777777823, 7777777849, 7777777853, 7777777871,
7777777937, 7777777963, 7777777993, 7837777777, 7957777777, 8087777777, 8117777777, 82277
77777, 8277777773, 8347777777, 8387777777, 8477777771, 8577777773, 8627777777, 8737777777
, 8777777713, 8777777717, 8777777759, 8777777777, 8807777777, 8947777777, 8977777777, 906
7777777, 9137777777, 9177777773, 9197777777, 9257777777, 9467777777, 9477777773, 94777777
79, 9547777777, 9617777777, 9677777771, 9777777767, 9777777787, 9777777799, 9817777777, 9
887777777, 9937777777, 9977777773]
8.3433706
```

```
C:\Users\mayam\anaconda3\lib\site-packages\ipykernel_launcher.py:24: DeprecationWarning:
time.clock has been deprecated in Python 3.3 and will be removed from Python 3.8: use tim
e.perf_counter or time.process_time instead
```

In [3]:

```
#Ques2
# Python3 program to find primitive root
# of a given number n
from math import sqrt

# Returns True if n is prime
def isPrime( n):

    # Corner cases
    if (n <= 1):
        return False
    if (n <= 3):
        return True

    # This is checked so that we can skip
    # middle five numbers in below loop
    if (n % 2 == 0 or n % 3 == 0):
        return False
    i = 5
    while(i * i <= n):
        if (n % i == 0 or n % (i + 2) == 0) :
            return False
        i = i + 6

    return True

def power( x, y, p):

    res = 1 # Initialize result

    x = x % p # Update x if it is more
              # than or equal to p

    while (y > 0):

        # If y is odd, multiply x with result
        if (y & 1):
            res = (res * x) % p

        # y must be even now
        y = y >> 1 # y = y/2
        x = (x * x) % p

    return res

# Utility function to store prime
# factors of a number
def findPrimefactors(s, n) :

    # Print the number of 2s that divide n
    while (n % 2 == 0) :
        s.add(2)
        n = n // 2

    # n must be odd at this po. So we can
    # skip one element (Note i = i +2)
    for i in range(3, int(sqrt(n)), 2):

        # While i divides n, print i and divide n
        while (n % i == 0) :

            s.add(i)
            n = n // i

    # This condition is to handle the case
    # when n is a prime number greater than 2
    if (n > 2) :
        s.add(n)

# root of n
def findPrimitive( n) :
    s = set()

    # Check if n is prime or not
    if (isPrime(n) == False):
        return -1

    # Find value of Euler Totient function
    # of n. Since n is a prime number, the
    # value of Euler Totient function is n-1
    # as there are n-1 relatively prime numbers.
    phi = n - 1

    # Find prime factors of phi and store in a set
    findPrimefactors(s, phi)

    # Check for every number from 2 to phi
    for r in range(2, phi + 1):
        # Iterate through all prime factors of phi.
        # and check if we found a power with value 1
        flag = False
        for it in s:

            # Check if r^((phi)/primefactors)
            # mod n is 1 or not
            if (power(r, phi // it, n) == 1):

                flag = True
                break

        # If there was no power with value 1.
        if (flag == False):
            return r

    # If no primitive root found
    return -1

# Driver Code
n = 1017356827

print("Smallest primitive root of",n, "is", findPrimitive(n))
from math import gcd

# Function to return the count of primitive roots modulo p
def countPrimitiveRoots(p):
    result = 1
    for i in range(2, p, 1):
        if (gcd(i, p) == 1):
            result += 1017356827

    return result

# Driver code
if __name__ == '__main__':
    p = 1017356827
```

```
Smallest primitive root of 1017356827 is 2
```

In [ ]:

```
#Primitive Roots
def gcd(a,b):
    while b != 0:
        a, b = b, a % b
    return a

def primRoots(modulo):
    roots = []
    required_set = set(num for num in range (1, modulo) if gcd(num, modulo) == 1)

    for g in range(1, modulo):
        actual_set = set(pow(g, powers) % modulo for powers in range (1, modulo))
        if required_set == actual_set:
            roots.append(g)
    return roots

if __name__ == "__main__":
    p =1017356827
    primitive_roots = primRoots(p)
    print(primitive_roots)
```

In [87]:

```
# Q3 DH Algorithm
from random import getrandbits
from random import randint
import sys

def is_prime_calc(num):
    return all(num % i for i in range(2, num))

def is_prime(num):
    return is_prime_calc(num)

def get_random_prime():
    while True:
        n = getrandbits(12) + 3;
        if is_prime(n):
            return n

def gcd(a,b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

def primitive_root(modulo):
    required_set = set(num for num in range (1, modulo) if gcd(num, modulo) == 1)
    for g in range(1, modulo):
        actual_set = set(pow(g, powers) % modulo for powers in range (1, modulo))
        if required_set == actual_set:
            return g

# Generating private keys
aragon_private = randint(999, 999999)
print ('Aragon private key is %d' % aragon_private)
dragon_private = randint(999, 999999)
print ('Dragon private key is %d' % dragon_private)

# Generating p-g parameters
p = get_random_prime()
g = primitive_root(p)

print ('\n p parameter is %d, g parameter is %d \n' % (p, g))

# Generating public keys
aragon_public = pow(g, aragon_private) % p
dragon_public = pow(g, dragon_private) % p

print ('Aragon public key is %d' % aragon_public)
print ('Dragon public key is %d' % dragon_public)

aragon_key = (pow(dragon_public, aragon_private)) % p
dragon_key = (pow(aragon_public, dragon_private)) % p

print ('\n Common secret: %d == %d' % (aragon_key, dragon_key))
```

```
Aragon private key is 128844
Dragon private key is 348760

 p parameter is 977, g parameter is 3

Aragon public key is 930
Dragon public key is 39

 Common secret: 431 == 431
```

In [ ]:

```
# Q4 Find an integer k such that a^k is congruent modulo b
import math;

def discreteLogarithm(a, b, m):

    n = int(math.sqrt (m) + 1);

    # Calculate a ^ n
    an = 1;
    for i in range(n):
        an = (an * a) % m;

    value = [0] * m;

    # Store all values of a^(n*i) of LHS
    cur = an;
    for i in range(1, n + 1):
        if (value[ cur ] == 0):
            value[ cur ] = i;
        cur = (cur * an) % m;

    cur = b;
    for i in range(n + 1):

        # Calculate (a ^ j) * b and check
        # for collision
        if (value[cur] > 0):
            ans = value[cur] * n - i;
            if (ans < m):
                return ans;
        cur = (cur * a) % m;

    return -1;

# Driver code
a = 123;
b = 45;
m = 89;
print(discreteLogarithm(a, b, m));

a = 233;
b = 71;
m = 113;
print(discreteLogarithm(a, b, m));
```

In [90]:

```
# Q4 Find an integer k such that a^k is congruent modulo b
import math;

def discreteLogarithm(a, b, m):

    n = int(math.sqrt (m) + 1);

    # Calculate a ^ n
    an = 1;
    for i in range(n):
        an = (an * a) % m;

    value = [0] * m;

    # Store all values of a^(n*i) of LHS
    cur = an;
    for i in range(1, n + 1):
        if (value[ cur ] == 0):
            value[ cur ] = i;
        cur = (cur * an) % m;

    cur = b;
    for i in range(n + 1):

        # Calculate (a ^ j) * b and check
        # for collision
        if (value[cur] > 0):
            ans = value[cur] * n - i;
            if (ans < m):
                return ans;
        cur = (cur * a) % m;

    return -1;
```

```
# Driver code
a = 5;
b = 7;
m = 9;
print(discreteLogarithm(a, b, m));

a = 23;
b = 29;
m = 31;
print(discreteLogarithm(a, b, m));
```

```
8
17
```

In [ ]:

```
#Q5 Decryption refering previous code
#1
from decimal import Decimal

def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
    if gcd(e,t)== 1:
        break


for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n

dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(c
t)+' decrypted text = '+str(dt))
```

In [97]:

```
#Q5 Decryption refering previous code 2
from decimal import Decimal

def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
    if gcd(e,t)== 1:
        break
```

```
for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n

dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(c
t)+' decrypted text = '+str(dt))
```

```
Enter the value of p = 23
Enter the value of q = 9
Enter the value of text = 14
n = 207 e = 3 t = 176 d = 59 cipher text = 53 decrypted text = 152
```

In [ ]:

```
#Q5 Decryption refering previous code 3
from decimal import Decimal

def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
    if gcd(e,t)== 1:
        break


for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n

dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(c
t)+' decrypted text = '+str(dt))
```

In [ ]:

```

```