HW8ImplentationSolutions

November 9, 2021

[13]:

```
########### Preamble

def fastPowerSmall(g,A,N):
    a = g
    b = 1
    while A>0:
        if A % 2 == 1:
            b = b * a % N
        A = A//2
        a = a*a % N
    return b

def extendedEuclideanAlgorithm(a,b):
    u = 1
    g = a
    x = 0
    y = b
    while true:
        if y == 0:
            v = (g-a*u)/b
            return [g,u,v]
        t = g%y
        q = (g-t)/y
        s = u-q*x
        u = x
        g = y
        x = s
        y = t

def findInverse(a,p):
    inverse = extendedEuclideanAlgorithm(a,p)[1] % p
    return inverse

def textToInt(words):
    number = 0
    i = 0
    for letter in words:
```

```
        number += ord(letter)*(256**i)
        i+=1
    return number

def intToText(number):
    words = ""
    while number>0:
        nextLetter = number % 256
        words += chr(nextLetter)
        number = (number-nextLetter)/256
    return words

def millerRabin(a,n):

    #first throw out the obvious cases
    if n%2 == 0 or extendedEuclideanAlgorithm(a,n)[0]!=1:
        return True

    #Next factor n-1 as 2^k m
    m = n-1
    k = 0
    while m%2 == 0 and m != 0:
        m = m//2
        k = k+1
    #Now do the test:
    a = fastPowerSmall(a,m,n)
    if a == 1:
        return False

    for i in range(0,k):
        if (a + 1) % n == 0:
            return False
        a = (a*a) % n

    #If we got this far a is not a witness
    return True

# This function runs the Miller-Rubin test on 20 random numbers between 2 and
 ↪p-1.  If it returns true there is a probability of (1/4)^20 that p is prime.
def probablyPrime(p):
    for i in range(0,20):
        a = ZZ.random_element(2,p-1)
        if millerRabin(a,p):
            return False
    return True

def findPrime(lowerBound,upperBound):
```

```
    while True:
        candidate = ZZ.random_element(lowerBound,upperBound)
        if probablyPrime(candidate):
            return candidate


def SunTzuPairs(m1,m2,a1,a2):
    #Run the Euclidean algorithm on a1 and a2
    GCDplus = extendedEuclideanAlgorithm(m1,m2)

    #Make sure our moduli are coprime
    if GCDplus[0]!=1:
        print("The moduli are not coprime! CRT will not work!")
        return -1

    #Otherwise the inverse of m1 mod m2 has already been computed
    m1Inverse = GCDplus[1]

    #We know x = a1 + m1*y, let's find y
    y = (a2 - a1)*m1Inverse % m2
    x = a1 + m1*y % (m1*m2) #we mod out by m1m2 to be in the right range

    return x
```

[4]:

```
########### Problem 1

#Helper Function
def factorOut2(n): #returns m,k such that n = m2^k and m is odd
    k = 0
    while n%2 == 0:
        n = n//2
        k = k+1
    return n,k

#####Part (a)
def legendreSymbol(a,p):
    #Make sure the base is even
    if p == 2:
        print("The base of a Legendre symbol must be odd!")
        return

    #Then use Euler's criterion
    else:
        a = a % p
        m = (p-1)//2
        return fastPowerSmall(a,m,p)
```

```python
#####Part (b)
def jacobiSymbol(a,b):
    #print("(",a,",",b,")")
    #Make sure the base is even
    if b%2 == 0:
        print("The base of a Jacobi symbol must be odd!")
        return

    #The value only depends on the class of a modulo b:
    a = a%b
    #If a=-1, 0, 1, or 2 we can compute this directly using quadratic␣
↪reciprocity
    if a==b-1:
        #This depends on the congruence class of b modulo 4
        if (b%4)==1:
            return 1
        else:
            return -1

    if a==0:
        return 0

    if a==1:
        return 1

    if a==2:
        #This depends on the congruence class of b modulo 8
        if (b%8 == 1) or (b%8 == 7):
            return 1
        else:
            return -1

    #If the base is prime we can just compute a Legendre symbol using Euler's␣
↪formula
    if probablyPrime(b):
        return legendreSymbol(a,b)

    #If not we use quadratic reciprocity to flip the jacobi symbol.  Since the␣
↪base of the jacobi symbol must be even, we need to factor all powers of 2␣
↪out of a.
    m,k = factorOut2(a)

    #since a = m2^k then (a/b) = (m/b)(2/b)^k.  The even part is easy to␣
↪compute:
    evenPart = jacobiSymbol(2,b)**k
```

```
    #Using quadratic reciprocity we compute that the odd part is either (b/m)␣
↪or -(b/m) depending on the residues of b and m modulo 4
    if (m%4==1) or (b%4==1):
        oddPart = jacobiSymbol(b,m)
    else:
        oddPart = -jacobiSymbol(b,m)
    return evenPart*oddPart

#####Part (c): Testing
tests = [
    [8,15],
    [11,15],
    [12,15],
    [171337608,536134436237]
]

for pairs in tests:
    a = pairs[0]
    b = pairs[1]
    print("(",a,"/",b,") =",jacobiSymbol(a,b))
```

```
( 8 / 15 ) = 1
( 11 / 15 ) = -1
( 12 / 15 ) = 0
( 171337608 / 536134436237 ) = -1
```

[4]:

```
########## Problem 2
g = 17
p = 19079

##### Part (a)
iList = [3030,6892,18312]
for i in iList:
    gi = fastPowerSmall(g,i,p)
    print(gi,"=",factor(gi))
```

```
14580 = 2^2 * 3^6 * 5
18432 = 2^11 * 3^2
6000 = 2^4 * 3 * 5^3
```

[19]:

```
##### Part (c)
q = (p-1)//2
E = [[2,6,1],[11,2,0],[4,1,3]]
b = [3030,6892,18312]

#make it into a matrix and vector over Fq
Mq = Matrix(GF(q),E)
```

```
vq = vector(GF(q),b)

#The solve_right command will return the solution to Mx = v
print("Mod q we have",Mq.solve_right(vq))

#Let's also do this over F2
M2 = Matrix(GF(2),E)
v2 = vector(GF(2),b)
print("Mod 2 we have",M2.solve_right(v2))
print("Therefore mod 2q:")
#now use SunTzu to glue to mod p-1 solution.
print("x2 =", SunTzuPairs(q,2,8195,0))
print("x3 =", SunTzuPairs(q,2,1299,0))
print("x5 =", SunTzuPairs(q,2,7463,0))
```

```
Mod q we have (8195, 1299, 7463)
Mod 2 we have (0, 0, 0)
Therefore mod 2q:
x2 = 17734
x3 = 10838
x5 = 17002
```

[28]:
```
##### Part (d)
#First compute 19g^-12400
ginv = findInverse(g,p)
ginvPower = fastPowerSmall(ginv,12400,p)

#Then factor it
print((19*ginvPower)%p,"=",factor(19*ginvPower % p))
```

```
384 = 2^7 * 3
```

[32]:
```
##### Part (e)
print((7*17734 + 10838 + 12400)%(p-1))
```

```
13830
```

[33]:
```
print(fastPowerSmall(g,13830,p))
```

```
19
```

[0]:
```

```