Homework5

October 13, 2021

[1]:
```
########## Preamble
########## Loading in fastpowering and euclidean algorithm and find inverse



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

#We'll also need BabySteps GiantSteps to call from Pohlig-Hellman
def babyGiant(g,h,p,N = -1):
```

```python
    #If we don't know n we assume it is p-1
    if N==-1:
        N = p-1

    #We should also reduce g and h mod p
    g = g % p
    h = h % p
    #We need both a list and a set in order to remember the logarithm
    babyStepList = []
    babyStepSet = set()
    n = math.floor(math.sqrt(N)) + 1

    #Set x to 1 and add it to both lists
    x = 1
    babyStepList.append(x)
    babyStepSet.add(x)
    #Generate your babysteps list
    for i in range(0,n):
        x = x*g % p
        babyStepList.append(x)
        babyStepSet.add(x)

    #x is now g^n.  Compute the inverse and that will be our giant step. Our giant steps start at h
    giantStep = findInverse(x,p)
    x = h

    #Then compute your giant steps check if they are in your set
    #Note, we go all the way to n+1 here because we do the multiplication at the end.
    for j in range(0,n+1):
        if x in babyStepSet:
            #If we're in the set find the index!
            #Notice we only have to do this once!
            for i in range(0,n+1):
                if x == babyStepList[i]:
                    #We found the match! Since g^i = hg^(-nj) the discrete log is i+nj
                    return i+n*j

        #Otherwise we take one more giant step and try again
        x = x*giantStep % p
    #If we got here then there was no match and this means that h is not a power of g
    print("h is not a power of g, there is no log!")
    return -1
```

[2]:
```
########## Problem 1

#####First have a Sun Tzu's theorem for pairs of moduli
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

def SunTzu(moduli,residues):

    #First make sure the lists match
    if len(moduli)!=len(residues):
        print("You have a different number of moduili and residues!  CRT will␣
↪not work!")
        return -1

    while len(moduli)>1:
        #Run CRTPairs on the last two pairs of data
        a1 = residues.pop()
        a2 = residues.pop()
        m1 = moduli.pop()
        m2 = moduli.pop()
        x = SunTzuPairs(m1,m2,a1,a2)

        #Make sure you didn't get thrown an error
        if x==-1:
            return x

        #Replace the last elements of your list with the solutions from the␣
↪last two to continue inductively
        residues.append(x)
        moduli.append(m1*m2)

    #Once the lists are length one our remaining residue is our solution!
```

```
    return residues[0]
```

[3]:
```
########## Problem 2
def PohligHellman(g,h,p,factors):

    #Compute the given order of g.
    N = 1
    for m in factors:
        N = N*m
    #Make sure that it is at least a multiple of the actual order
    if(fastPowerSmall(g,N,p)!=1):
        print("The given factors can't be right.  This Pohlig-Hellman won't␣
 ↪work.")
        return -1

    localSolutions = []
    for m in factors:
        #compute your hi and gi
        gi = fastPowerSmall(g,N//m,p)
        hi = fastPowerSmall(h,N//m,p)

        #run babygiant on your new stuff.  Make sure to feed it your new order␣
 ↪to speed things up!
        x = babyGiant(gi,hi,p,m)

        #x is -1 if we got an error from babyGiantHash
        if x==-1:
            print("h is not a power of g")
            return -1
        #This is your local solution for m!
        localSolutions.append(x)
    #Now use the CRT to stitch it all together
    return(SunTzu(factors,localSolutions))
```

[6]:
```
########## Problem 3
##### Part (a)
moduli = [23,41]
residues = [9,25]
print("A number that is congruent to 9 mod 23 and congruent to 25 mod 41 is:")
print(SunTzu(moduli,residues))

##### Part (b)
m = [2,3,5,7,11,13,17]
r = [1,2,4,6,10,1,16]
print("A number that modulo the first 7 primes is congruent to 1,2,4,6,10,1,␣
 ↪and 16 respectively is:")
print(SunTzu(m,r))
```

```
A number that is congruent to 9 mod 23 and congruent to 25 mod 41 is:
722
A number that modulo the first 7 primes is congruent to 1,2,4,6,10,1, and 16
respectively is:
314159
```

[7]:

```
########## Problem 4
##### Part (a)
factors = [2^4,7]
print("Given the factors of 113-1 we compute log_3(19) mod 113 to be:")
print(PohligHellman(3,19,113,factors))

##### Part (b)
p = 30235367134636331149
b = 6
h = 3295

factors = [4,9, 13, 41143, 335341, 4682597]

print("Given the factors of p-1 we can actually compute log base 6 of 3295 mod␣
↪a large p=30235367134636331149",)
print(PohligHellman(b,h,p,factors))
print("Let's check it worked:")
print(fastPowerSmall(b,16203647288039693568,p))
```

```
Given the factors of 113-1 we compute log_3(19) mod 113 to be:
99
Given the factors of p-1 we can actually compute log base 6 of 3295 mod a large
p=30235367134636331149
16203647288039693568
Let's check it worked:
3295
```