Homework4

October 8, 2021

[1]:
```
########## Preamble: Important functions from last homeworks

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
```

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

def naiveDLP(g,h,p):
    i = 0
```

```
    candidate = 1
    while(True):
        if candidate == h:
            break
        candidate = (candidate*g) % p
        i += 1
    return i
```

[4]:
```
#########Problem 2
#Here is a version using a hash table or set
#I think some folks implemented this using a library which was able to store␣
↪indices in a hash table, that's probably better than my implementation to be␣
↪honest.
def babyGiant(g,h,p,N = -1):
    #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
```

[9]:

```
###### Problem 3
# Part (a)
p = 113
g = 3
h = 19
print("Computing log_",g,"(",h,") mod",p)
print("Naive: ", naiveDLP(g,h,p))
print("BabyGiant: ", babyGiant(g,h,p))
print('\n')

#part (b)
p = 1073741827
g = 2
h = 54382
print("Computing log_",g,"(",h,") mod",p)
print("BabyGiant: ", babyGiant(g,h,p))
print("Naive: ", naiveDLP(g,h,p))
print('\n')

#part (c)
p = 302353671346363311149
g = 6
h = 3295
print("Computing log_",g,"(",h,") mod",p)
#print("Naive: ", naiveDLP(g,h,p))
#print("BabyGiant: ", babyGiant(g,h,p))
print("Didn't run!")
print('\n')
```

```
Computing log_ 3 ( 19 ) mod 113
Naive:  99
BabyGiant:  99
```

```
Computing log_ 2 ( 54382 ) mod 1073741827
BabyGiant:  811057010
Naive:  811057010


Computing log_ 6 ( 3295 ) mod 302353671346363311149
Didn't run!
```

```
[0]:
```