HW7ImplementationSolutions

November 9, 2021

[1]:
```
########## 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
```

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

def quadraticSieve(a,b,B,N):
    B+=1 #The upper bound shouldn't be sharp
    sieveList = []
    primes = prime_range(3,B)
    primeDataList = [0 for i in range(0,len(primes)+1)]  #This i'th spot␣
 ↪primeDataList keeps track of how many factors of the i'th prime we have
```

```
    for t in range(a,b):
        sieveList.append([t*t - N] + primeDataList)
        #factor out powers of 2 right away
        while(sieveList[t-a][0]%2 == 0):
            sieveList[t-a][0] = sieveList[t-a][0]//2
            sieveList[t-a][1] += 1

    #now do the odd primes
    i = 2
    for p in primes:
        if fastPowerSmall(N,(p-1)//2,p) == 1: #First make sure N can even be a␣
↪square mod p
            pPower = p                          #We will in fact do this for␣
↪prime powers too
            while(pPower < 2*(b-a)):
                alpha = int(Mod(N,pPower).sqrt())      #First we compute the␣
↪square roots (casting to an integer)
                beta = pPower-alpha

                #Next we find the smallest number >= a which is congruent to␣
↪the square roots mod pPower
                if a%pPower < alpha:
                    t1 = a + alpha - (a%pPower)
                else:
                    t1 = a + alpha - (a%pPower) + pPower
                if a%pPower < beta:
                    t2 = a + beta - (a%pPower)
                else:
                    t2 = a + beta - (a%pPower) + pPower

                while(t1<b):
                    sieveList[t1-a][0] = sieveList[t1-a][0]//p  #We divide the␣
↪associated numbers by p
                    sieveList[t1-a][i] += 1                      #Keeping track␣
↪of how many factors to remove
                    t1 += pPower
                while(t2<b):
                    sieveList[t2-a][0] = sieveList[t2-a][0]//p
                    sieveList[t2-a][i] += 1
                    t2 += pPower
                pPower *= p
        i+=1
    return sieveList
```

[3]:
```
##########  Problem 2
```

```
#####You can just print your Quadratic sieve output.  I'm doing this to make␣
↪solutions more readable.

def printQuadraticSieve(a,b,B,N):
    primes = prime_range(0,B+1)
    sieve = quadraticSieve(a,b,B,N)
    listOfCs = [i^2 - N for i in range(a,b)]
    print("Sieving:",listOfCs)
    print("...")
    for i in range(0,len(sieve)):
        print(listOfCs[i],"sieves to",sieve[i][0])
        if(sieve[i][0]==1):
            print("It is smooth!")
            factorList = [["There are",sieve[i][j+1],"factors of",primes[j]]␣
↪for j in range(0,len(primes))]
            print(factorList)
        print("...")


print("===========Trying B=7============")
printQuadraticSieve(15,30,7,221)
print("===========Trying B=11============")
printQuadraticSieve(15,30,11,221)
```

```
===========Trying B=7============
Sieving: [4, 35, 68, 103, 140, 179, 220, 263, 308, 355, 404, 455, 508, 563, 620]
…
4 sieves to 1
It is smooth!
[['There are', 2, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 0, 'factors of', 5], ['There are', 0, 'factors of', 7]]
…
35 sieves to 1
It is smooth!
[['There are', 0, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 1, 'factors of', 5], ['There are', 1, 'factors of', 7]]
…
68 sieves to 17
…
103 sieves to 103
…
140 sieves to 1
It is smooth!
[['There are', 2, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 1, 'factors of', 5], ['There are', 1, 'factors of', 7]]
…
179 sieves to 179
```

```
…
220 sieves to 11
…
263 sieves to 263
…
308 sieves to 11
…
355 sieves to 71
…
404 sieves to 101
…
455 sieves to 13
…
508 sieves to 127
…
563 sieves to 563
…
620 sieves to 31
…
============Trying B=11=============
Sieving: [4, 35, 68, 103, 140, 179, 220, 263, 308, 355, 404, 455, 508, 563, 620]
…
4 sieves to 1
It is smooth!
[['There are', 2, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 0, 'factors of', 5], ['There are', 0, 'factors of', 7], ['There are', 0,
'factors of', 11]]
…
35 sieves to 1
It is smooth!
[['There are', 0, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 1, 'factors of', 5], ['There are', 1, 'factors of', 7], ['There are', 0,
'factors of', 11]]
…
68 sieves to 17
…
103 sieves to 103
…
140 sieves to 1
It is smooth!
[['There are', 2, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 1, 'factors of', 5], ['There are', 1, 'factors of', 7], ['There are', 0,
'factors of', 11]]
…
179 sieves to 179
…
220 sieves to 1
It is smooth!
```

```
[['There are', 2, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 1, 'factors of', 5], ['There are', 0, 'factors of', 7], ['There are', 1,
'factors of', 11]]
…
263 sieves to 263
…
308 sieves to 1
It is smooth!
[['There are', 2, 'factors of', 2], ['There are', 0, 'factors of', 3], ['There
are', 0, 'factors of', 5], ['There are', 1, 'factors of', 7], ['There are', 1,
'factors of', 11]]
…
355 sieves to 71
…
404 sieves to 101
…
455 sieves to 13
…
508 sieves to 127
…
563 sieves to 563
…
620 sieves to 31
…
```

[4]:

```
########## Problem 3

def sieveFactor(a,b,B,N):
    sieve = quadraticSieve(a,b,B,N) #First run the sieve doing the relation␣
↪building step.  Next we do the elimination step
    primes = prime_range(0,B)
    A = []
    C = []
    E = []

    for i in range(0,len(sieve)):          #These are the a_i, such that c_i =␣
↪a_i^2-N is B-smooth, and the e_ij are the exponents of the prime factors p_j
        if sieve[i][0]==1:
            A.append(a + i)
            C.append((a+i)^2 - N)
            nextRow = [sieve[i][j] for j in range(1,len(sieve[i]))]
            E.append(nextRow)

    M = matrix(GF(2),E)                                     #Use Sage to convert E to␣
↪a matrix over F_2. Dont forget E
    basis = M.kernel().basis()                              #compute the basis of the␣
↪nullspace of this matrix
```

```
    for b in range(0,len(basis)):
        #Each entry here will the sum of the column of the E associated to a␣
↪prime, if it appears in the basis  This gives us the exponenets of the c_is
        exponent = [0 for i in range(0,len(primes))]
        a0 = 1
        for i in range(0,len(basis[b])):
            if basis[b][i] == 1:
                a0 = a0 * A[i] % N                         #We're also computing the␣
↪products of the a_i associated to the basis element of the nullspace
                for j in range(0,len(primes)):
                    exponent[j] += E[i][j]
        #Next we compute the product of the square roots off the c_i that␣
↪appear in our factorization using the exponenets we computed in the previous␣
↪loop
        b0 = 1
        for j in range(0,len(primes)):
            b0 = b0 * primes[j]**(exponent[j]//2) % N #since a^2 =␣
↪p_j^(exp[j]), we divide by 2 to take the square root in Z.
        #In this case there's no hope
        if a0==b0:
            continue
        divisor = extendedEuclideanAlgorithm(N,a0-b0)[0] #Here's our candidate!␣
↪ Let's see if it works!
        if(divisor != 1 and divisor !=N and divisor !=-1 and divisor !=-N):
            return[abs(divisor),abs(N//divisor)] #we use absolute values to␣
↪ensure positive factors
    print("none found")
```

[5]:
```
##########  Problem 4
#####Part (a)
print("Part (a): Let's try to factor 221:")
print(sieveFactor(15,30,7,221))

#####Part(b)
#Let's first define this L function:
def ell(x):
    return float(e^((ln(x)*ln(ln(x)))^.5))

#Let's use the ranges given in the problem
def sieveLFactor(N):
    L = int(ell(N))
    B = int(L^(.5^.5))
    a = math.floor(sqrt(N))
    b = a + L
    return sieveFactor(a,b,B,N)
```

```
#####b(i)
print("Part b(i): Let's try to factor 8249")
print(sieveLFactor(8249))

#####b(ii)
print("Part b(ii): Let's try to factor 7799773")
print(sieveLFactor(7799773))

#####b(iii)
print("Part b(iii): Let's try to factor 9488773076569")
print(sieveLFactor(9488773076569))

#####b(iv)
print("Part b(iv): Let's try to factor 1182692471909987")
print(sieveLFactor(1182692471909987))
```

```
Part (a): Let's try to factor 221:
[13, 17]
Part b(i): Let's try to factor 8249
[73, 113]
Part b(ii): Let's try to factor 7799773
[7507, 1039]
Part b(iii): Let's try to factor 9488773076569
[7340269, 1292701]
Part b(iv): Let's try to factor 1182692471909987
[33895067, 34892761]
```

[18]:
```
########## Problem 7 Computations:

#Part (a)
print(ell(2^100)/10^9,"seconds")

#Part (b)
print(ell(2^250)/(10^9*60),"minutes")

#part (c)
print(ell(2^500)/(10^9*60*60*24*365),"years")

#part (d)
print(ell(2^1000)/(10^9*60*60*24*365*10^12),"trillion years")
```

```
0.027802429905024805 seconds
159.2147074064945 minutes
1130.0731911459704 years
5.553235322322046 trillion years
```

[0]:

```

```