def f(x,a,b,G,H,modn): '''Used for pr.'''

def frho(x,n): '''Used for prho.'''

def lcm(\*a):

def gcd(\*a):

def inv(u, v): '''Return u^-1 mod v'''

def div(x,y,n):'''Return x/y mod n'''

def po3(x,y,n):'''Return x^y mod n'''

def ord(x,n): '''Return order of x in Z\*n'''

def rot(n,sqn,modn): '''Return n^(1/sqn) mod modn'''

def crt(a\_l,m\_l):

    '''

    Use Chinese Remainder Theorem to solve the euation system:

    x=a\_l[0] mod m\_l[0]

    x=a\_l[1] mod m\_l[1]

    ...

    x=a\_l[l] mod m\_l[l]

    Output (res,modn), where res is the minimum positive integer solution,

    modn is the multiple. General solution is res+k\*modn.

'''

def bsgs(base,power,order,modn):

    '''Solve log\_base(power) mod modn using BSGS algorithm.

Return an integer, or -1 if error.'''

def ph(base,power,order,p\_l,e\_l,modn):

    '''

    Solve log\_base(power) mod modn using Pohlig-Hellman algorithm.

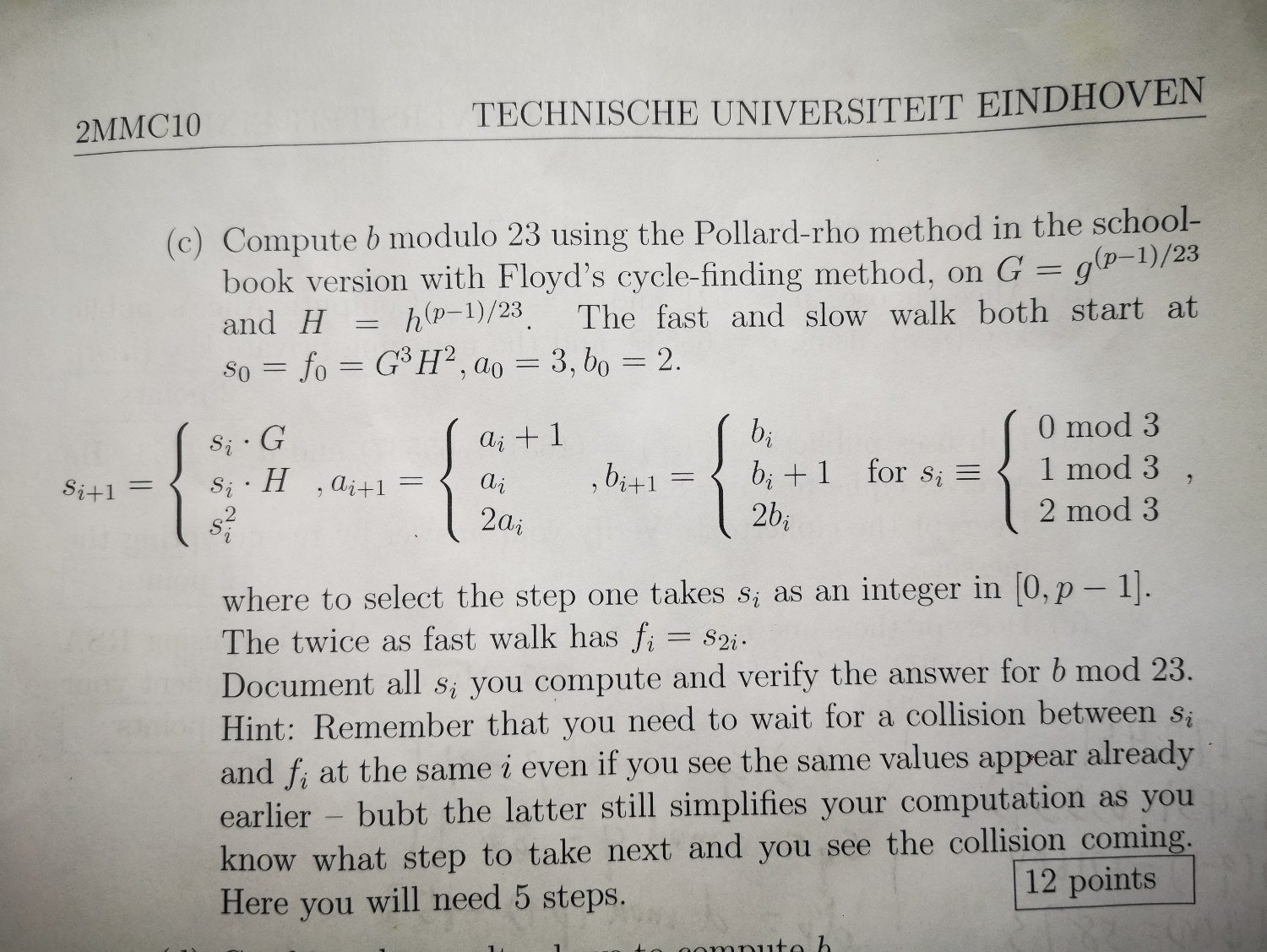
    Return an integer, or -1 if error.

    The order of base is order=p\_l[0]^e\_l[0]\*p\_l[1]^e\_l[1]...\*p\_l[l]^e\_l[l].

'''

def pr(s0,f0,a0,b0,G,H,modn):

'''Print i,s,ai,bi,f,a2i,b2i of each round. Use f(x,a,b,G,H,modn).'''



def prho(n,x1):

'''Print xi,x2i of each round. Use frho(x,n).'''

def inv(u, v): '''Return u^-1 mod v'''

def div(x,y,n):'''Return x/y mod n'''

def po3(x,y,n):'''Return x^y mod n'''

def dot(al,bl):

def rou(x):

    '''round x to nearest integer.'''

def gs(M,n):

    '''

    Input a n\*n matrix M. Each row is a base.

    Return a n\*n marix M\_ret, with Gram–Schmidt process.

    '''

def lll(M,n,delta=3/4):

    '''

    Input a n\*n matrix M. Each row is a base.

    Return a n\*n marix M\_ret, with lll lattice basis reduction.

'''

Example:

n=7684607040813031964568123727442263397500506224420545927139570285341

pt=3855587076697238083701498334674944

X=pow(2,36)

M=[[n,0,0,0],[pt,X,0,0],[0,pt\*X,X\*X,0],[0,0,pt\*X\*X,X\*X\*X]]

M1=lll(M,4)

f=gef(M1,4,X)

v=sol(f,4,X)

print(pt+v)

def gef(M,n,X):

    '''

    Return the function array arr from n\*n Matrix M.

    The function is F(x)=arr[0]+arr[1]x+...+arr[n-1]x^n-1.

    '''

def gev(arr,n,x):

    ''' Return the value F(x).'''

def sol(arr,n,sol):

''' sol F(x) by Newton's method. sol is estimated value of x.'''

def douw(x,y,a,b,modn):

'''double (x,y) on '''

def addw(x\_p,y\_p,x\_q,y\_q,a,b,modn):

'''Add two poins (x\_p,y\_p) and (x\_q,y\_q) modulp modn. '''

def mulw(x,y,a,b,exp,modn):

    '''Calculate the exp times of the point (x,y) modulo modn. Result is (x\_r,y\_r).'''

def bsgsw(x,y,x\_A,y\_A,a,b,order,modn):

    '''

    Solve DLP using Baby Step Giant Step.

    Return log\_{(x,y)}(x\_A,x\_A) modulo modn.

    The point (x,y) has order "order".

'''

def ps(modn):

'''Print i^2 mod modn.'''

def pt(typ,arg,modn):

'''

Print Point. typ is:

0: Edwards Curve, arg=d;

1: Twisted Edwards Curve, arg=[a,d]

2: Short Weierstrass Curve, arg=[a,b];

3: Montgomery curve, arg=[A,B]

'''

def inv(u, v): '''Return u^-1 mod v'''

def div(x,y,n):'''Return x/y mod n'''

def po3(x,y,n):'''Return x^y mod n'''

def adde(x1,y1,x2,y2,d,modn):

'''(x1,y1)+(x2,y2) on Edwards curve x^2+y^2=1+dx^2y^2 mod modn'''

def mule(x,y,times,d,modn):

'''times\*(x,y) on Edwards curve x^2+y^2=1+dx^2y^2 mod modn'''

def orde(x,y,d,modn):

'''order of (x,y) on Edwards curve x^2+y^2=1+dx^2y^2 mod modn'''

def loge(x1,y1,x2,y2,d,modn):

'''log\_(x1,y1)(x2,y2) on Edwards curve x^2+y^2=1+dx^2y^2 mod modn'''

def addm(x\_p,y\_p,x\_q,y\_q,A,B,modn):

def mulm(x,y,A,B,exp,modn):

def doum(x,y,A,B,modn):

def logm(x1,y1,x2,y2,A,B,modn):

def prt(n): '''Return if n is a prime (True or False).'''