Number Theory

Md. Tariqul Islam
IIT-44, JU.

How Many ways you can divide n chocolates evenly among some number of people ?

Everytime someone spin it, it stop k step forward, how many ways you can choose k so that it will stop at every step? When number of step is n.

Given a set of available bill/banknote

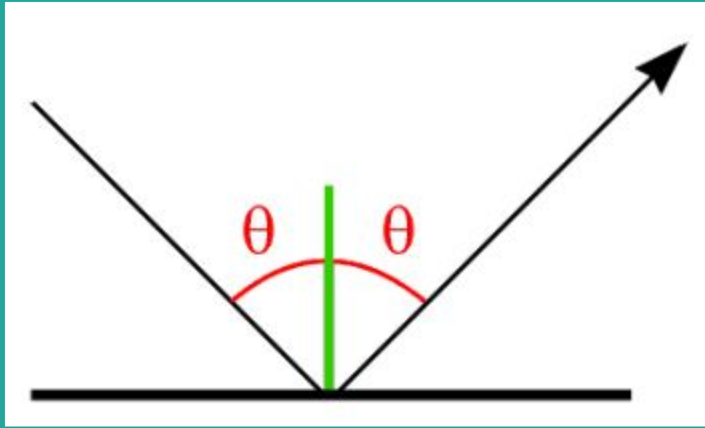
- What's the minimum amount you can changes ?
- Given a amount , how to decide is it possible to make change of given amount ?

{12,15,24,45}

What is the minimum amount you can make changes?

Can we change 25 ?

You have a nxm rectangle , which has 4 vertex at (0,0) , (n,0) (n,m) and (0,m) accordingly. Also you have a ball at point (x,y) inside the rectangle , which is moving at a speed (vx,vy). Meaning at t=0 , the position of the ball is (x,y) , in next second it will be at point (x+vx,y+vy) . given that |vx|=|vy|. When ball hit at any boundary , the direction changes but velocity remains the same. Direction changes it a way such incoming angle is same as outgoing angle.
The question is what is the position of the ball after t seconds?



Sets of Integers

Set	Name	Symbol
{...-2,-1,0,1,2…}	Integers	Z
{0,1,2,3,4,5,....}	Non negative integers	Z^*
{1,2,3,4,5,....}	Positive Integers	Z^+ , N
{0,1,2,...,n-1}	Additive group modulo n	Z^+_n
$\{1,3,7,9\}_{10}$	Multiplicative group modulo n	Z^*_n

Divisors

- Let $a,b,c \in Z$, that is a , b and c are 3 integers
- And $c=ab$
- Then a and b both are divisor of c
- We write $a|c$, means a divide c
- In other word , a will be a divisor of c if there exist a integer b for which $c=ab$ is true.

Multiples

- A integer c is multiple of another integer a iff there exist a integer b or which c=ab is true.

Properties

- If $d|a$ and $d|b$ then $d|(a-b)$ and $d|(a+b)$
    - Prove
- If $d|a$ and $d|b$ then $d|(ax+by)$
    - Prove
- $a|b$ and $b|a$ implies $a=\pm b$

Reminders

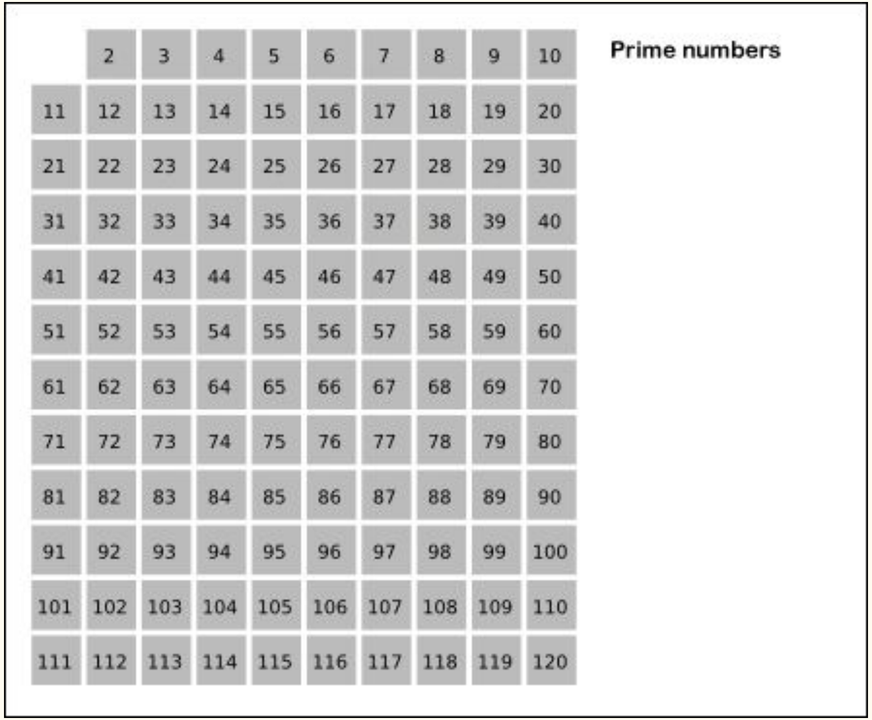
- $a\%b = a - \lfloor a/b \rfloor b$
- Sometime we call a modulo b , or simply a mod b
- a modulo b = a - (largest multiple of b which is less than or equal a)
- Well defined when a is negative
- Not defined when b is zero
- If a % m = b % m , then a and b are modular equivalent in modulo m , we write $a \equiv b \bmod m$
    - We'll talk more in later

Primes

- Any positive integer n has two trivial divisor , namely 1 and n.
- If n has no other divisor other than 1 and n , then n is called prime number.
- With the exception of 1, so to be precise , a prime number has exacltly two distinct divisor (1 has one divisor)
- To check whether a number is prime or not , we can search for a non trivial divisor .
- If c=ab and $a \leq b$ then $a \leq \sqrt{c}$ and $b \geq \sqrt{c}$
- So we can bound our search till $\sqrt{c}$

Sieve / pruning and segmentation

- Normal sieve
- Mark all even number first
- Ignore even numbers (when you need the list only)
- Bitwise (using 32/64/128 bit number)
- Segmentation
- Variation (precalculating all prime factor/sod/nod till n)



GCD and LCM

- Gcd is greatest common divisor of 2 or more integer
  - gcd(10,15,25)=5
- LCM is Lowest common multiple of two or more number.
- gcd(a,b) is smallest linear combination of a b
  - Let s=ax+by , s is positive and smallest possible for some x,y ϵ Z
  - Let q=$\lfloor a/s \rfloor$, a%s=a-qs=a-q(ax+by)=a(1-qx)+b(-qy)
  - So a%s is also a linear combination of a,b and we have $0 \leq a\%s < s$
  - But a%s cant be positive and less than s at the same time
  - So a%s=0 and s|a , with the same argument , we can show s|b
  - So s | gcd(a,b) [any common divisor divides gcd]
  - But gcd(a,b)|s [if d|a and d|b then d|(ax+by)]
  - So s=gcd(a,b)

- gcd(a,b)=gcd(b,a)
- gcd(a,b)=gcd(-a,b)
- gcd(a,b)=gcd(|a|,|b|)
- gcd(a,0)=|a|
- If d|a , d|b then d|gcd(a,b)
- gcd(xa,xb)=x.gcd(a,b)
- gcd(a,ka)=|a|,for any k ϵ Z
- gcd(a,b)=gcd(b,a%b)
- For any x,y ϵ Z ax+by=z.gcd(a,b) for some z ϵ Z
- gcd(a,b) is smallest linear combination of a b

Coprime / Relatively primes

- Two or more integer are called relatively prime is they has no common divisor other than 1.
- In other word , if $gcd(a,b) = 1$ then a and b are relatively prime

Integer factorization

- Any positive integer > 1 can be represented as a multiple of prime numbers only.
    - A prime number can contribute more than one time
        - $100=2x2x5x5=2^2 5^2$
        - If no prime number occurs more than once , the number is called square-free
    - $n = p_1^{k1} p_2^{k2} p_3^{k3} p_4^{k4} \ldots p_m^{km}$
- Such representation called prime factorization and it's unique for any positive integer
- We can calculate prime factorization of a integer in $O(\sqrt{n})$ time
- We can do some better if we have list of all prime numbers till $\sqrt{n}$

Basic counting functions ($\varphi,\sigma_0,\sigma_1$)

$\phi(n) = \sum[a > 0 \wedge a < n \wedge \gcd(a, n) = 1]$

$\sigma_k(n) = \sum_{d|n} d^k$

- Let's start with σ_0 , which simplifies to number of divisor of n.
  - Given a number , think we have it's prime factorization , then how can we construct a divisor ? how manys of them
- All three functions are multiplicative.
- $\varphi(p)=p-1,\sigma_0(p)=2,\sigma_1(p)=p+1$ for any prime p
- $\varphi(p^k)=p^k-p^{k-1},\sigma_0(p^k)=k+1,\sigma_1(p^k)=1+p+p^2+p^3+..+p^k=(p^{k+1}-1)/(p-1)$
  - $p,2p,3p,4p,.....,p^{k-1}p(=p^k)$, whose are the number who is not co prime with p^k
- For any other number , prime factorize it and multiply the result from each prime component

GCD , Euclidean Algorithm, proofs and Applications

- gcd(a,b)=gcd(b,a%b)
- Let d=gcd(a,b) and q=⌊a/b⌋ then a%b=a-qb
- So d|(a%b) , since d|b so d|gcd(b,a%b) ⇒ gcd(a,b)|gcd(b,a%b)
- Now let d=gcd(b,a%b) and q=⌊a/b⌋ then a=a%b +qb
- So d|a , since d|b then d|gcd(a,b) ⇒ gcd(b,a%b)|gcd(a,b)
- So gcd(a,b)=gcd(b.a%b)

Extended euclid

- gcd(a,b)=ax+by
- gcd(a,0)=a=a.1+0.0
- gcd(a,b)=gcd(b,a%b)
- ax+by=bu+(a%b)v [assuming gcd(b,a%b)=bu+(a%b)v, and we know u and v]
- ax+by=bu+(a-bq)v [q=⌊a/b⌋]
- ax+by=av+b(u-qv)
- So we got, x=v,y=u-qv

Modular Arithmetics

- If a%m=b%m then a≡b (mod m)
- a%m= a - ⌊a/m⌋m
  - So -16%10=-16- ⌊-16/10⌋10=-16-(-2)10=-16+20=4
- (a+b)%m=((a%m)+(b%m))%m
- (a*b)%m=((a%m)*(b%m))%m
- a%m is always non-negative regardless of a, assuming m is positive
- a%m=(a%m+m)%m
  - Since m%m=0
- If a<0 then (a%m+m)%m will give a non-negative result (c/c++)
- Notice (a/b)%m is not well defined since we're dealing with integers only

Prove that $p \equiv \mp 1 \bmod 6$

Where p is prime and $p > 3$

work : try proving $p^2 \equiv 1 \bmod 24$

,where p is a prime and > 3

Multiplicative inverse

- Remember reciprocals ? dividing by 3 is same as multiplying by ⅓ or 3^{-1}
- If we multiply 3 and it's inverse ⅓ or 3^{-1} , we should get 1
  - $3*3^{-1}=3*⅓=1$, $a*a^{-1}=a*(1/a)=1$
- We can apply same argument here , like if $a*b\equiv1 \pmod m$, we can treat b as a^{-1} or $1/a$ [and vice versa] whenever needs
  - $a*b\equiv1 \Rightarrow b=1/a \Rightarrow b=a^{-1} \pmod m$
  - $3*7\equiv1 \pmod{10}$ so , $8/3=8*7=6 \pmod{10}$
- Does every number has a modular inverse ? does it depends on modulo ?
  - Prove that $a*b\equiv1 \pmod m$ is possible iff $gcd(a,m)=gcd(b,m)=1$
  - That is modular inverse of a number a exist in modulo m , if and only if $gcd(a,m)=1$

Find a^{-1} modulo m

- Given gcd(a,m)=1 , find a^{-1} , that is find a number b such that $a*b \equiv 1 \pmod m$
- Let's say we have gcd(a,m)=ax+my=1
    - We can find such x y by extended euclid algorithm
- Take the modulo m , we have $ax \equiv 1 \pmod m$ [since my %m=0]
    - Bingo , x is our modular inverse , $a^{-1}=x$
- If you dont wanna write extended euclid , here is another way for you
    - Let $q=\lfloor m/a \rfloor$, then $m\%a=m-qa$
    - $(m\%a) \equiv -qa \pmod m$
    - $(m\%a).a^{-1} \equiv -qa.a^{-1} \pmod m$
    - $(m\%a).a^{-1} \equiv -q \pmod m$
    - $a^{-1} \equiv -q(m\%a)^{-1} \pmod m$

```
int modular_inverse(int a,int m){
        if(a==1)
                return 1;
        return ((m-(m/a))*modular_inverse(m%a,m))%m;
}
```

- We have (at least) two other ways for finding modular inverse. Will see later on

Integer hashing

- How to calculate a%m when a is way big and m fit's in a integer variable ?
- We can extract digit by digit from a , and maintain the modulo m instead of actual a
- 6896454679%999=(6896454670+9)%999=(6896454670%999+9)%999
- In that way we can determine whether the number a is a multiple of m or not.
- If sum of digit of n is divisible by 3 or 9 respectively then n is also divisible by 3 or 9 , but why ?

```
rem=0

for(i=0;i<n;i++)

    rem=(rem*10+a[i])%mod
```

Fermat little Theorem / Euler theorem

- Imagine a sequence $1, a, a^2, a^3, a^4, a^5, a^6, \ldots$ In modulo m
- The sequence must end up with some kind of cycle [why?]
- What is the maximum and minimum length of that cycle ?
    - If $\gcd(a,m)=1$, cycle will always form a circle , means $a^i \% = 1$ for some $i>0$
    - If $\gcd(a,m)\neq 1$ then cycle is likely to shape a ρ form other than circle
    - Maximum cycle length is $\varphi(m)$, when m is prime it’s m-1
    - For any a such $\gcd(a,m)=1$, cycle length is a divisor of $\varphi(m)$
- $a^{p-1} \equiv 1 \bmod p$ when p is a prime and $0<a<p$ [Fermat Little theorem]
- $a^{\varphi(m)} \equiv 1 \bmod m$, where $\gcd(a,m)=1$ [Euler theorem]
- If we agree on those fact above , these two theorem are direct conclusion from them
- Moreover , Fermat little theorem is just a special case of euler theorem.

Calculating modular inverse (again)

- Since $a^{\varphi(m)} \equiv 1 \bmod m$ we have $a.a^{\varphi(m)-1} \equiv 1 \bmod m$
- So , $a^{\varphi(m)-1}$ is the modular inverse of a in modulo m
- Moreover when m is prime , (say $m = p$) then $a^{-1} \equiv a^{p-2}$ modulo p

Binary Exponentiation

```
int pow(a,b):
    if b==0:
        return 1
    res=pow(a,b/2)
    res=res*res
    if b is even:
        return res
    Else :
        return res*a
```

```
int pow(a,b):
    res=1
    for(i=b;i>0;i=(i>>
1))
        If i is odd:
            res=res*a
        a=a*a
    return res
```

```
int pow(a,b):
    res=1
    for i=0 to 30:
        If i’th bit of b
is SET:
            res=res*a
        a=a*a
    return res
```

- If we need to calculate a^b mod m , we can do that in exact same ways , just need to take modulo each time after any operation on res or a.
- The main problem here that we need a multiplication here , which limits the value of mod roughly in 10^9
- To demonstate last two function , expand b as a sum of distinct powers of 2

Binary Exponentiation (cont..)

- $a^{25} = a^{16+8+1} = a^{16}.a^{8}.a^{1}$
- On each iteration we square a , means it's generates this sequence below
  - $a, a^{2}, a^{4}, a^{8}, a^{16}, a^{32}, a^{64}, a^{128}, a^{256}, \ldots.$
- We check corresponding bit of b to decide whether we need the current term in the results or not?

```
int pow(a,b,m):
    res=1%m
    for(i=b;i>0;i=(i>>1))
        If i is odd:
            res=res*a % m
        a=a*a%m
    return res
```

We still have the problem with the limits of m. we can have 64 bit unsigned variable in our computer, but our ab mod m only work if m is 32 bit . We need to work it out.

Matrix exponentiation

- We can use the same idea to calculate exponent of a matrix (square matrix to be more specific)
- Just use the same code bellow , res= 1 , need to change it as res=I_n , where res is a nXn matrix , I_n means identity matrix.
- a=a*a , just squaring a and then saving it into a again.
- res=res*a(or you can write res=a*res) , is multiplying res with a.

Up to the limit

- We want to work out the overflow
- So, we'll just get rid of multiplication operator , (mod operator as well).
- More specifically , we want none of our calculation to exceed m
- We can pay extra O(log m) and do the multiplication exactly same way of pow()
- But now the problem is addition , though it's allow m to as large as $(2^{64}+2)/2$ -1 with 64 bit unsigned int. But not quite 2^{64}-1
- With the function add(), m is not a problem now. Note that add() has no cost.

```
int pow(a,b,m):
    res=1%m
    for(i=b;i>0;i=(i>>1))
        If i is odd:
            res=multiply(res,a,m)
        a=multiply(a,a,m)
    return res
int multiply(a,b,m):
    int res=0
    for(i=b;i>0;i=(i>>1))
        for(i=b;i>0;i=(i>>1))
        If i is odd:
            res=add(res,a,m)
        a=add(a,a,m)
    return res
int add(a,b,m):
    if m-b>a:
        return a+b
    else : return a-(m-b)
```

Multilevel exponent , applying modulo on exponent

- Recall fermat little theorem
  - $a^{p-1} \equiv 1$ mod p when p is a prime and $0<a<p$
- Let’s say we want to calculate a^x mod p , where p is a prime , $0<a<p$ and x is way to large , (Like 10000 digit decimal or 10^5 bit binary)
- With the previous algorithm for pow(a,x,m) , everything is fine . infact complexity is not gonna improved but it’s not clear how we will check the last bit of x each time , or how will we divide it by 2 , or right shift by 1. If we need any extra cost to do that , our complexity will rise . Either way , it’s a mess to work with such number.
- Let's assume $q=\lfloor x/(p-1) \rfloor$, and $r=x\%(p-1)$

Multilevel exponent , applying modulo on exponent

- $a^x \bmod p = a^{q(p-1)+r} = a^{q(p-1)}\, a^r = (a^{p-1})^q\, a^r \equiv 1^q a^r \bmod p \quad \equiv a^r \bmod p$
- What we just showed is $a^x \bmod p \equiv a^{x \bmod p-1} \bmod p$, when p is prime and $1<a<p$
- Using same set of arguments , with euler theorem we can conclude $a^x \bmod m \equiv a^{x \bmod \varphi(m)} \bmod m$, the only restriction here is a and m needs to be co-prime.
- But , what if a and m is not coprime , can we do something ?
- $a^x \% m = a^{\varphi(m) + (x \bmod \varphi(m))} \% m$ if $x >= \log 2(m)$
  - If m fit's in 64 bit then x just need to larger than 64
- But wait ? why this is true why we are multiplying $a^{\varphi(m)}$, which is just one?
  - $a^{\varphi(m)}$ is not just one , not always . only when a and m co-prime
    - $5^4 \% 10$, $0^{anything}$ mod anything ,
- Well , why this is true ?
  - ~~Still a question to me.~~ Will not fit here.

Generator

- Remember Z^*_n ? Z^*_n={x: 0<x<n and gcd(x,n)=1} , the set of all positive coprime number with n which are smaller than n
- Also recall that φ(n) is the cardinality of that set.
- Any element g from this set , from a circular cycle when taking its exponent
  - $1 , g , g^2,g^3,g^4,....$ mod n
- Which is means there exist a (many) positive j for which g^j mod n = 1
- Let's say x is smallest positive such j for which g^x mod n=1
  - x <= φ(n) , [conclusion from euler theorem]
- If x=φ(n) that means , it's 'generate' every single element of the set before it's getting back to 1
- If x <φ(n) , then it's must be a divisor of φ(n) [again , euler theorem but how?]
- Z^*_n has a (or more) generator if and only if n=2, 4, p^i and $2p^i$, where p is prime and i is positive integer.

Generator

- To find a generator (after confirming that the exist one) , we can do no better that bruteforce search
    - At least as far as i know
- However , we can speed up our search by using some facts
- If a^x mod m = 1 and x<φ(m) , x could not be a generator
- We do not need to check all x , any x for which a^x mod m = 1 holds , must be a divisor of φ(m)
- To check a candidate a , we need to verify a^x mod m ≢ 1, where x|φ(m)
- Yes , we need to check all a, but you'll find one before you get to the 100 in most cases.
- You can still improve it algorithm, by checking those divisor who cover others , namely for each prime divisor p, check x=φ(m)/p . [see Emaxx tutorial]

Discrete log

- $a^x \equiv b \bmod m$, given a , b , m find x or report no such x exist .
- If any such x exist , one of them must be $<$ m . in other word if there is any solution , one solution must less than m.[why?]
- Let $p=\sqrt{m}$, and let $x=ip+j$, such $j=x\%p$ and $i=floor(x/p)$
  - Note that , $i<p$ and $j<p$
- $a^{ip+j}\equiv b \quad \Rightarrow a^{ip}a^{j}\equiv b \quad \Rightarrow a^{ip} \equiv b.a^{-j} \bmod m$
- $a^{ip} \equiv b.a^{-j} \bmod m$
- We can calculate a^{ip} and store in a map for all $i<p$ then for each $b.a^{-j}$ we can just look for the value in map.
- Cost of this algorithm is $O(\sqrt{m})$ if we can have constant lookup in map.

Discrete root

- $x^a = b \bmod m$
- $x = b^{1/a} \bmod m$
- $x = b^{1/a \bmod \varphi(m)}\ m$
- But $a^{-1} \bmod \varphi(m)$ only exist if a and $\varphi(m)$ are co-prime. Even if they are not coprime we can still have such x but we we'll not be able to calculate them in this way.
- Let $x = g^y$ where g is a generator modulo m. (yeah, m must satisfy those condition to have a generator)
- For a constant modulo m, we can precal it's generator and use it for for free , when m is a variable , well we have to find a generator
    - We know how to find a generator , right ?

Discrete root

- If i replace x with g^y we got $(g^y)^a = b \bmod m$
- $g^{ya} = b \bmod m$, we need to find y
- $(g^a)^y = b \bmod m$, now t
- This is the exact same problem with solving discrete logarithm
- So , we can find such y and result will be g^y

Linear equations (diophantine)

- $ax=b \bmod m$, given a,b,m find a x which will satisfy this equation.
- $ap + mq=g$, where $g=gcd(a,m)$, we can find such p and q by running egcd
- If we take modulo m , we got $ap\equiv g \bmod m$
- By comparing 2 equation we can have $x=p(b/g)$, and that tells us b must be a multiple of g in order to apply this argument
  - Indeed , b must be a multiple of gcd(a,b) in order to have ANY solution to this equation
- If x is a solution of this equation then $x+(m/g)$ and $x-(m/g)$ is also satisfy the equation

Linear Diophantine Equation

- ax+by=c , given a ,b,c
  - Find ANY x,y which will satisfy this equation
  - Find all solution in a given range
  - Or just count them
- ap+by=g where g=gcd(a,b) we can fnd such x y by running extended euclid

Let's say we found x_0 and y_0 , which is a solution for given a,b,c, by running extended euclid and further calculation. And let g =gcd(a,b)

Then $ax_0 + by_0 = c$

$=> ax_0 + by_0 + ab/g - ab/g = c$

$=> ax_0 + ab/g + by_0 - ab/g = c$

$=> a(x_0 + b/g) + b(y_0 - a/g) = c$

So, $(x_0 + b/g)$ and $(y_0 - a/g)$ is also a solution to the given equation, moreover , for any integer k

$a(x_0 + bk/g) + b(y_0 - ak/g) = c$

So, that's all the solution.

To find all solution in a range , see My notes

System of linear equation , CRT

Factorial modulo , ${}^{n}C_{r}$ modulo , wilson theorem

Fibonacci's

Roots of unity

- $x^n = 1 \bmod m$
- Some application needs to have n distinct x to satisfy this equation
- If Z^*_m has a generator g , and n is divisor of $\varphi(m)$,then Let $p = \varphi(m)/n$
- Then the sequence below will be n distinct root to the given equation
  - $1 , g^p , g^{2p} , ... g^{(n-1)p}$
- Let's check $(g^{pi})^n = (g^{i\,\varphi(m)/n})^n = g^{i\,\varphi(m)} = (g^{\varphi(m)})^i = 1^i = 1$

Trivial square roots of unity

Randomized Primality test

Millar robin test

Pollard pho factorization

Linear congruential generator

Next to learn

- Mobius inversion
- FFT and NTT
- Solving linear recurrence with matrix exponent
-