Aritmètica Modular

Aritmètica Modular

En certs aspectes de la informàtica (per exemple, la criptografia) és important una variació de l'aritmètica sobre els nombres enters: **l'aritmètica modular**.

Serveix per operar amb rangs restringits d'enters.

Definim **x mòdul N** com **la resta de dividir x per N**, és a dir, si x=qN+r amb 0<=r<N, llavors el x mòdul N és r.

La complexitat és $O(n^2)$

Això permet definir una equivalència (**congruència**) entre nombres (inclosos els negatius!):

$x \equiv y \pmod{N}$ si i només si N divideix $(x-y)$

Com que 10 divideix (133-3), 133 és congruent amb 3 mòdul 10

Aritmètica Modular

Això afecta a les operacions aritmètiques:

Substitution rule *If* $x \equiv x' \pmod N$ *and* $y \equiv y' \pmod N$, *then:*

$$x + y \equiv x' + y' \pmod N \quad \textbf{and} \quad xy \equiv x'y' \pmod N.$$

Però les seves propietats es conserven:

$x + (y + z) \equiv (x + y) + z \pmod N$	Associativity
$xy \equiv yx \pmod N$	Commutativity
$x(y + z) \equiv xy + yz \pmod N$	Distributivity

Si ajuntem les dues coses, tenim que és legal reduir els resultats intermedis al seu mòdul N en qualsevol moment.

$$2^{345} \equiv (2^5)^{69} \equiv 32^{69} \equiv 1^{69} \equiv 1 \pmod{31}.$$

Aritmètica Modular

La suma i la multiplicació no són gaire complexes.

Suma: Si dos nombres estan el rang $0..(N-1)$ la seva **suma** ho està en el $0..2(N-1)$ (que només és un bit més). Si el resultat passa de $N-1$ el que hem de fer és simplement **restar** del resultat N. És evident que **la complexitat és lineal $O(n)$**, on n = log N, la mida de N.

Recordem que necessitem $log_b N$ dígits per escriure N en base b.

Multiplicació: De forma semblant, fem la multiplicació normal i transformem al rang $0..(N-1)$ si és que ens hem passat.
El **producte** pot ser fins $(N-1)^2$ però això es pot representar amb $2n$ bits. Per transformar el resultat hem de **dividir** per N (amb complexitat $O(n^2)$). Per tant, **la complexitat és $O(n^2)$**

Aritmètica Modular

Divisió: Aquesta operació no és tant trivial (no està definida per tots els nombres) i té **una complexitat $O(n^3)$.** (*veure més endavant*)

Exponenciació: Ara imaginem que volem calcular expressions com aquesta amb nombres molt grans (centenars de bits):

$$x^y \bmod N$$

Aquest resultat entremig pot necessitar molts bits per ser representat!

El resultat necessita *log N* bits

Si els operadors tenen 20 bits, aquest valor pot necessitar 10 milions de bits!

$$(2^{19})^{(2^{19})} = 2^{(19)(524288)}$$

La complexitat és molt alta!

Aritmètica Modular

Una solució és fer totes les operacions intermèdies mòdul N!

O sigui, calcular $x^y \bmod N$ fent y multiplicacions successives per x mòdul N.

$$x \bmod N \to x^2 \bmod N \to x^3 \bmod N \to \cdots \to x^y \bmod N,$$

Tots aquests resultats són menors que N i per tant no necessiten tants bits. Per tant, les multiplicacions són de complexitat $O(n^2)$.

El problema és que si y té 500 bits, n=500 bits, hem de fer $y - 1 \approx 2^{500}$ multiplicacions i **l'algorisme és exponencial sobre n**, la mida de y.

Aritmètica Modular

Però una petita modificació pot ser un gran canvi!

Observem que es pot calcular x elevat a y, on y és una potència de 2, elevant al quadrat, mòdul N, successivament:

$$x \bmod N \to x^2 \bmod N \to x^4 \bmod N \to x^8 \bmod N \to \cdots \to x^{2^{\lfloor \log y \rfloor}} \bmod N.$$

Cada potència pren un temps proporcional a $O(\log^2 N)$ i hi ha $\log y$ multiplicacions: **l'algorisme és polinòmic respecte la mida de N i lineal respecte la mida de y**!

Log y és la mida de y: $\log_2 8 = 3$

log N és la mida de N.

Aritmètica Modular

Per un **valor concret** (que no sigui potència de 2) només hem multiplicar les potències de 2 que corresponen a la representació binaria de y:

$$x^{25} = x^{11001_2} = x^{10000_2} \cdot x^{1000_2} \cdot x^{1_2} = x^{16} \cdot x^{8} \cdot x^{1}.$$

Aritmètica Modular

Aquesta operació es pot expressar **recursivament** fent aquestes operacions mòdul N:

$$x^y = \begin{cases} (x^{\lfloor y/2 \rfloor})^2 & \text{Si y és parell} \\ x \cdot (x^{\lfloor y/2 \rfloor})^2 & \text{Si y és senar} \end{cases}$$

$x^{25} = x \cdot (x^{12})^2 \mod N$

$\rightarrow (x^6)^2 \mod N$

$\rightarrow (x^3)^2 \mod N$

$\rightarrow x \cdot (x)^2 \mod N$

Aritmètica Modular

L'algorisme queda:

```
function modexp(x, y, N)
Input:  Two n-bit integers x and N, an integer exponent y
Output:  x^y mod N

if y = 0:  return 1
z = modexp(x, ⌊y/2⌋, N)
if y is even:
    return z^2 mod N
else:
    return x · z^2 mod N
```

La complexitat és $O(n^3)$
n crides recursives en les que fa una multiplicació mòdul N.

Cap a la divisió modular: L'algorisme d'Euclides

Fa més de 2000 anys que Euclides va enunciar un **algorisme** per trobar el **màxim comú divisor** de dos nombres *a* i *b*.

La forma més obvia de buscar-ho és **trobar els factors dels dos nombres i multiplicar llavors els seus factors comuns**.

Exemple pel *mcd* de 1035 i 759:

$1035=3^2*5*23$ i $759 = 3*11*23$, per tant $mcd=3*23=69$

El problema és que no tenim cap algorisme eficient per factoritzar els nombres!

L'algorisme d'Euclides

Euclides va utilitzar aquesta **regla simple**: Si x i y són enters positius amb x>=y, llavors mcd(x,y)=mcd(x mod y,y).

Proof. It is enough to show the slightly simpler rule $\gcd(x, y) = \gcd(x - y, y)$ from which the one stated can be derived by repeatedly subtracting y from x.

Here it goes. Any integer that divides both x and y must also divide $x - y$, so $\gcd(x, y) \leq \gcd(x - y, y)$. Likewise, any integer that divides both $x - y$ and y must also divide both x and y, so $\gcd(x, y) \geq \gcd(x - y, y)$. ∎

Això ens permet escriure aquest algorisme:

```
function Euclid(a, b)
Input:  Two integers a and b with a ≥ b ≥ 0
Output:  gcd(a, b)

if b = 0:  return a
return Euclid(b, a mod b)
```

mcd(1071,462)=21

Pas	Equació	
0	$1071 = q_0\ 462 + r_0$	$q_0 = 2$ and $r_0 = 147$
1	$462 = q_1\ 147 + r_1$	$q_1 = 3$ and $r_1 = \mathbf{21}$
2	$147 = q_2\ 21 + r_2$	$q_2 = 7$ and $r_2 = 0$;

Quan triga?....

L'algorisme d'Euclides

La primera cosa que hem de veure és com es van reduint els nombres a mesura que anem calculant.

Cal fixar-se que a cada iteració els arguments (a,b) es converteixen a (b, a mod b): canviem l'ordre i el més gran queda reduït al mòdul del petit.

A més a més, es fàcil veure que si a>=b, llavors (a mod b)<a/2.

Proof. Witness that either $b \leq a/2$ or $b > a/2$. These two cases are shown in the following figure. If $b \leq a/2$, then we have $a \bmod b < b \leq a/2$; and if $b > a/2$, then $a \bmod b = a - b < a/2$.

L'algorisme d'Euclides

Això vol dir que **en dos iteracions successives** els dos arguments decreixen al menys a la meitat, és a dir, **perden un bit en la seva representació.**

Si inicialment eren enters de n bits, en 2n crides recursives arribarem al final de l'algorisme. Com que cada crida implica una divisió d'ordre quadràtic, (a mod b) , **el temps total serà $O(n^3)$.**

Una extensió de l'algorisme d'Euclides

Suposem que algú ens diu que d és el mcd(a,b). **Com podem comprovar-ho?**

No és simple: no n'hi ha prou amb dir que d divideix a i b, perquè això només vol dir que és factor comú, no el més gran!

Però podem usar aquest lema:

Si d divideix a i b, i d=ax+by per alguns enters x i y, llavors necessariament d=mcd(a,b).

Proof. By the first two conditions, d is a common divisor of a and b and so it cannot exceed the greatest common divisor; that is, $d \leq \gcd(a, b)$. On the other hand, since $\gcd(a, b)$ is a common divisor of a and b, it must also divide $ax + by = d$, which implies $\gcd(a, b) \leq d$. Putting these together, $d = \gcd(a, b)$. ∎

Una extensió de l'algorisme d'Euclides

Però per usar el lema, hem de trobar els nombres....

Doncs resulta que aquests nombres es podem trobar amb una petita extensió de l'algorisme d'Euclides:

```
function extended-Euclid(a, b)
Input:  Two positive integers a and b with a ≥ b ≥ 0
Output:  Integers x, y, d such that d = gcd(a, b) and ax + by = d

if b = 0:  return (1, 0, a)
(x', y', d) = Extended-Euclid(b, a mod b)
return (y', x' − ⌊a/b⌋y', d)
```

La complexitat és $O(n^3)$,
no canvia respecte a l'algorisme original.

Una extensió de l'algorisme d'Euclides

Example. To compute $\gcd(25, 11)$, Euclid's algorithm would proceed as follows:

$$\begin{aligned} \underline{25} &= 2 \cdot \underline{11} + 3 \\ \underline{11} &= 3 \cdot \underline{3} + 2 \\ \underline{3} &= 1 \cdot \underline{2} + 1 \\ \underline{2} &= 2 \cdot \underline{1} + 0 \end{aligned}$$

(at each stage, the gcd computation has been reduced to the underlined numbers). Thus $\gcd(25, 11) = \gcd(11, 3) = \gcd(3, 2) = \gcd(2, 1) = \gcd(1, 0) = 1$.

To find x and y such that $25x + 11y = 1$, we start by expressing 1 in terms of the last pair $(1, 0)$. Then we work backwards and express it in terms of $(2, 1)$, $(3, 2)$, $(11, 3)$, and finally $(25, 11)$. The first step is:

$$1 = \underline{1} - \underline{0}.$$

Una extensió de l'algorisme d'Euclides

To rewrite this in terms of $(2, 1)$, we use the substitution $0 = 2 - 2 \cdot 1$ from the last line of the gcd calculation to get:

$$1 = \underline{1} - (\underline{2} - 2 \cdot \underline{1}) = -1 \cdot \underline{2} + 3 \cdot \underline{1}.$$

The second-last line of the gcd calculation tells us that $1 = 3 - 1 \cdot 2$. Substituting:

$$1 = -1 \cdot \underline{2} + 3(\underline{3} - 1 \cdot \underline{2}) = 3 \cdot \underline{3} - 4 \cdot \underline{2}.$$

Continuing in this same way with substitutions $2 = 11 - 3 \cdot 3$ and $3 = 25 - 2 \cdot 11$ gives:

$$1 = 3 \cdot \underline{3} - 4(\underline{11} - 3 \cdot \underline{3}) = -4 \cdot \underline{11} + 15 \cdot \underline{3} = -4 \cdot \underline{11} + 15(\underline{25} - 2 \cdot \underline{11}) = 15 \cdot \underline{25} - 34 \cdot \underline{11}.$$

We're done: $15 \cdot 25 - 34 \cdot 11 = 1$, so $x = 15$ and $y = -34$.

Divisió Modular: el problema.

A l'aritmètica real, cada nombre *a* diferent de zero té un invers $1/a$, i **dividir per a és el mateix que multiplicar pel seu invers.**

A l'aritmètica modular podem fer una definició semblant: Direm que *x* és **l'invers multiplicatiu** de *a (mod N)* si

$$ax \equiv 1 \pmod{N}$$

Aquest invers, però, no sempre existeix: 2 no és invertible mòdul 6 perquè no hi ha cap *x* que faci

$$2x \equiv 1 \bmod 6$$

Divisió Modular: Quins nombres no poden tenir invers?

De forma més general, podem estar segurs que *mcd(a,N)* divideix *a·x mod N*, perquè aquesta quantitat es pot escriure com *ax+kn*. Per tant, si *mcd(a,N)*>1 llavors

$$ax \not\equiv 1 \pmod{N}$$

independentment del valor x, i per tant a **no pot tenir un invers multiplicatiu** mòdul N!

Quan *mcd(a,N)=1* diem que *a* i *N* són **relativament primers**. **Només els nombres relativament primers tenen invers multiplicatiu mòdul N.**

Divisió Modular

Per nombres **relativament primers** l'algorisme modificat d'Euclides ens retorna dos enters, x i y tals que ax+Ny=1, lo que significa que ax és congruent mòdul N, i per tant **x és l'invers de a!**

Example. Continuing with our previous example, suppose we wish to compute $11^{-1} \bmod 25$. Using the extended Euclid algorithm, we find that $15 \cdot 25 - 34 \cdot 11 = 1$. Reducing both sides modulo 25, we have $-34 \cdot 11 \equiv 1 \bmod 25$. So $-34 \equiv 16 \bmod 25$ is the inverse of $11 \bmod 25$.

Teorema de la divisió modular

Per qualsevol *a mod N*, *a* té un invers multiplicatiu mòdul *N* si i només si és relativament primer per *N*. Quan aquest invers existeix, es pot trobar amb un temps $O(n^3)$ amb l'algorisme extès d'Euclides.

Amb això resolem el problema de la divisió modular: quan treballem mòdul N, podem dividir pels nombres relativament primers per N, i només per aquests. I per dividir, el que hem de fer és multiplicar per l'invers

És un nombre primer el vostre DNI?

Comprovar si un nombre més o menys gran és primer és una tascar a priori dura, perquè **hi ha molts factors per provar**. Però hi ha alguns fets que ens poden estalviar feina:

- No cal considerar com a factor cap nombre parell excepte el 2. De fet, podem obviar tots els factors que no són primers.
- Podem dir que un nombre és primer si no hem trobat cap candidat a factor menor que arrel de N, atès que N=K*L, i per tant és impossible que els dos nombres siguin més grans que arrel de N.

Fins aquí, bé, però no trobarem més maneres d'eliminar més candidats!

Això podria fer dir que provar la primalitat d'un nombre és un problema dur, però això no és veritat: només és dur si ho intentem pel camí de la factorització!

Primalitat

Una de les activitats bàsiques de la informàtica, la **criptografia**, es basa en el següent fet: **la factorització és dura, però la primalitat és fàcil.**

O el que és el mateix, **no podem factoritzar grans nombres, però podem mirar fàcilment si grans nombres són primers** (evidentment, sense buscar els factors!).

Per fer-ho, ens basarem en un teorema de 1640...

Primalitat

Teorema petit de Fermat

Si p és primer, llavors per a qualsevol enter a, 1<= a <p, es compleix que,

$$a^{p-1} \equiv 1 \pmod{p}$$

Això ens suggereix un test directe per comprovar si un nombre és primer:

```
function primality(N)
Input:  Positive integer N
Output:  yes/no

Pick a positive integer a < N at random
if a^(N-1) ≡ 1 (mod N):
   return yes
else:
   return no
```

Podríem repetir aquest test un munt de vegades i decidir

Primalitat

Els problema és que aquest teorema és necessari però no suficient: no diu què passa quan *N* no és primer!

És més, es coneixen uns certs nombres compostos, anomenats nombres de Carmichael, que passen el test per tots els enters *a* relativament primers a *N*. Per tant, és perillós passar la funció tal i com l'hem escrit...

Primalitat

Per exemple, el nombre 561 és un nombre de Carmichael:

a^{560} és congruent amb 1 mòdul 561

per tots els nombres *a* relativament primers a 561, i 561=3x11x17.

I per tant no és congruent amb 1 per *a* igual a 3, 11, 17 i múltiples d'aquests nombres. Per tots els altres valors compleix el test de Fermat.

Primalitat

Suposem de moment que no existeixen els nombres de Carmichael. Què passa amb els nombres compostos?

Lema

Si $a^{N-1} \not\equiv 1 \pmod{N}$ per algun *a* relativament primer a N (*o sigui, els nombres compostos que no són de Carmichael*), llavors com a mínim en **la meitat dels casos** en que *a*<N el teorema petit de Fermat fallarà.

Primalitat

Una petita variació de l'algorisme que veurem, coneguda com l'algorisme de *Rabin & Miller* soluciona aquest problema.

Si ignorem els nombres de Carmichael, podem dir que:

- Si N és primer, llavors $a^{N-1} \equiv 1 \pmod{N}$ per tots a<N
- Si N no és primer, llavors $a^{N-1} \equiv 1 \pmod{N}$ fallarà per almenys la meitat dels valors a<N

I per tant el comportament de l'algorisme proposat és:

- El test retornarà `yes` en tots els casos si N és primer.
- El test retornarà `yes` per la meitat o menys dels casos en que N no és primer.

Primalitat

Si repetim l'algorisme *k* vegades per nombres *a* escollits aleatòriament, llavors:

$$\Pr(\text{"}yes\text{"} \text{ quan } N \text{ no és primer}) \leq \frac{1}{2^k}$$

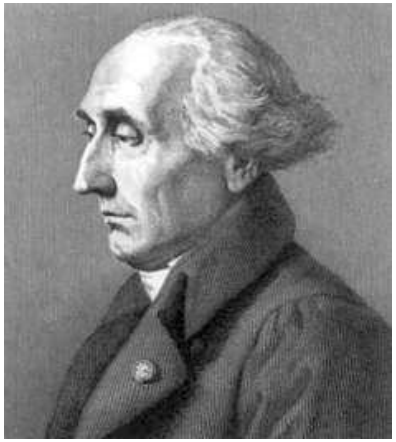
Si k=100, la probabilitat d'error és menor que 2^{-100}

Amb un nombre moderat de tests podem determinar si un nombre és primer.

Primalitat i grans nombres

Com ho fem per trobar primers formats per uns quants centenars de bits?

Si n'hi ha pocs tenim un problema amb l'algorisme anterior, doncs l'haurem de repetir moltes vegades!



El teorema dels nombres primers de Lagrange ens assegura que no tindrem problemes: la probabilitat de que un nombre de *n* bits sigui primer és aproximadament:

$$\frac{1}{\ln 2^n} \approx \frac{1.44}{n}$$

Primalitat i grans nombres

Com és de ràpid aquest mètode per trobar primers de n bits?

A cada iteració té una probabilitat 1/n d'aturar-se, per tant s'aturarà quan hagi fet O(n) iteracions.

Com l'apliquem?

N'hi ha prou amb provar per a=2, 3 i 5! (De fet, la probabilitat de fallar és molt menor que ½!)

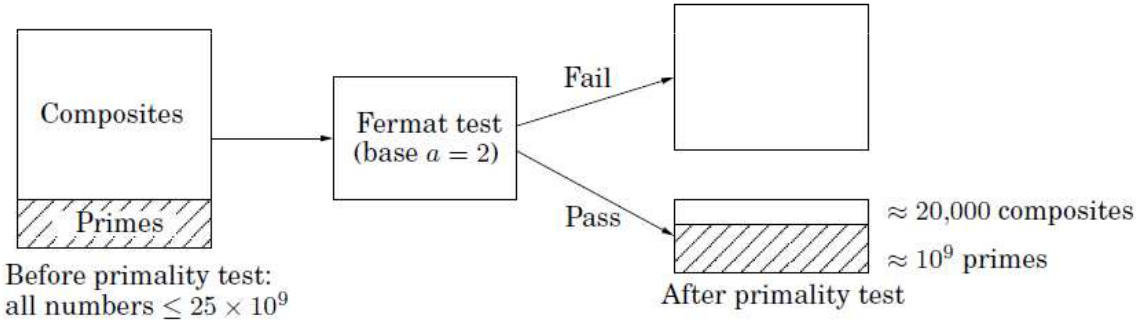
Quina és la probabilitat de que el nombre seleccionat sigui primer?

Suposem que volem passar el test per a=2 a tots els nombres < 25 x 10^9

Primalitat i grans nombres

Quina és la probabilitat de que el nombre seleccionat sigui primer?

En aquest rang hi ha al voltant de 10^9 primers i 20.000 compostos que passen el test.

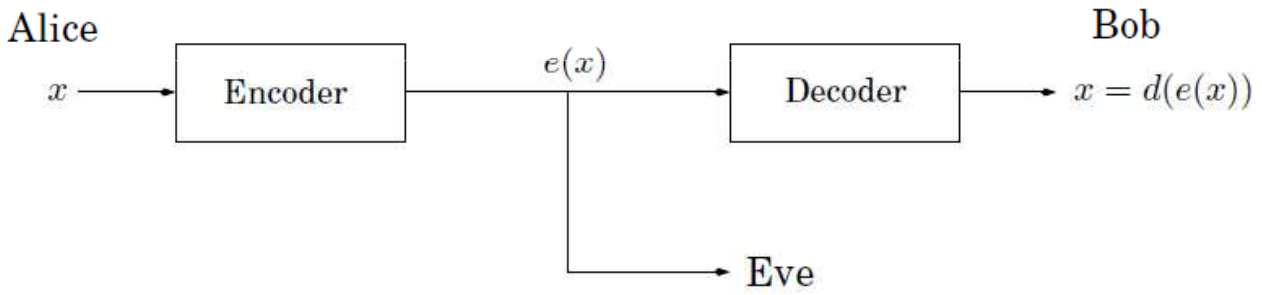


Per tant, la probabilitat d'error és $20.000/10^9 = 2 \times 10^{-5}$

Aquesta probabilitat va baixant a mesura que creix N.

Criptografia

El problema és que l'Alice pugui codificar amb una clau secreta x, generant e(x), i que en Bob sigui l'únic que pugui descodificar-ho:



Anem a veure com implementar un sistema segur amb clau pública!

Criptografia

Bob chooses his public and secret keys.

- He starts by picking two large (n-bit) random primes p and q.
- His public key is (N, e) where $N = pq$ and e is a $2n$-bit number relatively prime to $(p-1)(q-1)$. A common choice is $e = 3$ because it permits fast encoding.
- His secret key is d, the inverse of e modulo $(p-1)(q-1)$, computed using the extended Euclid algorithm.

Alice wishes to send message x to Bob.

- She looks up his public key (N, e) and sends him $y = (x^e \bmod N)$, computed using an efficient modular exponentiation algorithm.
- He decodes the message by computing $y^d \bmod N$.

Criptografia

És molt segur?

És segur si donats N, e, i $y=x^e \ mod \ N$, és computacionalment intractable trobar x.

Com podria l'Eve trobar x?

Podria provar per tots els valors de x si $y=x^e \ mod \ N$ però això té una complexitat exponencial!

També podria factoritzar N per trobar p i q, i llavors determinar d invertint $e \ mod \ (p-1)(q-1)$, però això és també intractable!