OliCyber.IT 2023 - Approfondimenti
Introduzione alla Teoria dei Numeri

Contenuti

1 Massimo Comun Divisore e Algoritmo di Euclide (Esteso)

Dati due numeri interi a, b, il loro *Greatest Common Divisor* (Massimo Comun Divisore in italiano) è il più grande numero naturale (quindi intero e positivo) che divide sia a che b (con resto 0).

$$g = GCD(a,b) \implies \begin{cases} a = k \cdot g \\ b = h \cdot g \\ h, k \in \mathbb{Z} \end{cases}$$

Questo si può scrivere anche come $\begin{cases} g|a \\ g|b \end{cases}$ ove "|" indica "divide (con resto 0)".

Come calcolarlo? Una prima idea può essere quella di scomporre a, b nelle rispettive fattorizzazioni in (potenze di) primi, per poi andare a prendere quelli comuni con l'esponente minore.

$$\begin{cases} a = 18 \\ b = 12 \end{cases} \implies \begin{cases} a = 2^1 \cdot 3^2 \\ b = 2^2 \cdot 3^1 \end{cases} \implies g = GCD(a,b) = 2^1 \cdot 3^1 = 6$$

Il problema di questo algoritmo sta nel primo step: in generale, ottenere la fattorizzazione di un numero non è affatto semplice! Non per niente RSA fonda la sua sicurezza proprio su questa difficoltà.
Ciononostante esiste un algoritmo molto efficiente per ovviare al problema: quello di Euclide[1]. Esso parte dall'osservare che, se g divide sia a che b, allora deve necessariamente dividere anche il *resto* della divisione di a per b (supponendo $a \geq b$, altrimenti basta scambiarli).

$$g = GCD(a,b) \implies \begin{cases} a = k \cdot g \\ b = h \cdot g \\ h, k \in \mathbb{Z} \end{cases} \quad \textit{(Definizione di GCD)}$$

$$\begin{cases} a = q \cdot b + r \\ q, r \in \mathbb{Z} \\ 0 \leq r < b \end{cases} \quad \textit{(Divisione Euclidea)}$$

$$k \cdot g = q \cdot h \cdot g + r \implies (k - q \cdot h) \cdot g = r \implies g|r.$$

Indichiamo $r := a\%b = a$ *modulo* b.
Quindi $GCD(a,b) = GCD(b, a\%b) = GCD(b,r)$.
Effettuare la divisione Euclidea tra due numeri è molto semplice! Si può quindi iterare il ragionamento su b, r, fintantoché non otterremo $r = 0$ (N.B.: $GCD(b,0) = b$).

$$\begin{cases} a = 18 \\ b = 12 \end{cases} \implies g = GCD(18,12) = GCD(12,6) = GCD(6,0) = 6.$$

Ma c'è di più! Tenendo traccia dei quozienti (i q delle varie divisioni Euclidee effettuate), è possibile trovare due interi (eventualmente negativi) x, y tali che:

$$x \cdot a + y \cdot b = g = GCD(a,b).$$

Questa si chiama *Identità di Bézout*.
L'algoritmo ottenuto si chiama *algoritmo di Euclide esteso* (XGCD[2]).
Consiglio di leggere la tabellina nell'esempio fornito nel link per un'intuizione sul suo funzionamento.

[1] https://it.wikipedia.org/wiki/Algoritmo_di_Euclide)
[2] https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm)

2 Inversi modulari

Riprendiamo ora in mano gli accenni visti a lezione riguardo le congruenze.
In particolare, voglio sottolineare che quando si va a considerare il *resto* della divisione di un intero per un numero naturale, esso è sempre positivo.
Questo si può ricavare facilmente dalle proprietà che si sono viste:

$$0 \equiv n \pmod{n} \implies 0 - 1 \equiv n - 1 \pmod{n} \implies -1 \equiv n - 1 \pmod{n}$$

Potrai aver già intuito come una congruenza sia fondamentalmente un'equazione. Per manipolarla, puoi applicare tutti i principi di equivalenza che hai visto a scuola: *ad eccezione della divisione!* Come mai la divisione è così fastidiosa in questo caso?
Ebbene, la risposta sta nell'insieme di numeri con il quale si sta lavorando. In particolare, a scuola probabilmente hai sempre lavorato con i numeri reali; nota che vale la seguente:

$$\text{se } x \text{ è un numero reale } \textit{diverso da zero}, \frac{1}{x} = x^{-1} \text{ è a sua volta un numero reale.}$$

Questo non vale negli interi (e in particolare negli interi modulo n):

$$\frac{1}{3} \text{ non è un numero intero.}$$

Quindi, visto che si sta lavorando con un particolare sottoinsieme dei numeri interi, non può andar bene effettuare dei passaggi che ci fanno "uscire" dall'insieme in cui siamo.
Dove sta dunque il trucco? Nella *definizione* di *divisione* come *operazione su un insieme.*
Definiamo come *divisione per un numero* $x \neq 0$ il *prodotto per l'inverso moltiplicativo di* x, che chiamiamo x^{-1}.
Nei numeri reali, questo coincide con $\frac{1}{x}$, poiché $x \cdot \frac{1}{x} = \frac{x}{x} = 1$.
Nei numeri interi, $x^{-1} \pmod{n}$ (se esiste) è *un altro numero intero*!

- $2^{-1} \pmod{5} = 3$, perché $2 \cdot 3 = 6 \equiv 1 \pmod{5}$.
- $2^{-1} \pmod{5}$ *non è* 0.5
- nei reali, x^{-1} esiste se e solo se $x \neq 0$.
- negli interi modulo n, x^{-1} esiste se e solo se $GCD(x, n) = 1$, ossia se x ed n sono *coprimi.*

Abbiamo dunque la stessa notazione per *lo stesso concetto* (inverso moltiplicativo), ma che negli interi modulo n è diverso da "dividere 1 per x".
Il passo immediatamente successivo consiste nell'avere un modo di trovare l'inverso moltiplicativo di un numero modulo n in maniera efficiente.
Per dire, per cercare l'inverso di 3 modulo 11 (nota bene: $GCD(3, 11) = 1$) potrebbe venire in mente di andare per tentativi: i numeri da controllare sono quelli compresi tra 1 e 10, che sono pochi.

- $3 \cdot 1 = 3 \pmod{11} \longrightarrow$ no
- $3 \cdot 2 = 6 \pmod{11} \longrightarrow$ no
- $3 \cdot 3 = 9 \pmod{11} \longrightarrow$ no
- $3 \cdot 4 = 12 \equiv 1 \pmod{11} \longrightarrow$ sì! (potrei fermarmi qui, *l'inverso se esiste è unico*)
- $3 \cdot 5 = 15 \equiv 4 \pmod{11} \longrightarrow$ no
- ...
- $3 \cdot 10 = 30 \equiv 8 \pmod{11} \longrightarrow$ no

Quando il modulo è molto grande, però, non possiamo permetterci di controllarli tutti: serve un algoritmo migliore. Detto fatto! In realtà abbiamo già lo strumento che risponde a questa necessità.
Ricorda cosa ci permette di ottenere l'algoritmo di Euclide esteso (che è molto efficiente): dati $a, b \in \mathbb{Z}$, ci vengono restituiti $x, y \in \mathbb{Z}$ e $g \in \mathbb{N}$ tali che $x \cdot a + y \cdot b = g = GCD(a, b)$. Questa è un'equazione senza modulo!
Che succede dunque se consideriamo $a \pmod{n}$, con $GCD(a, n) = g = 1$ (a, n coprimi)?
Ebbene, otteniamo $x, y \in \mathbb{Z}$ tali che $x \cdot a + y \cdot n = GCD(a, n) = 1$. Equazione senza modulo!
Quindi, se riduciamo entrambi i membri dell'equazione modulo n, otteniamo:

- $x \cdot a + y \cdot n \pmod{n} = 1 \pmod{n}$, ma $y \cdot n$ è un multiplo di n, quindi
- $y \cdot n \equiv 0 \pmod{n}$, dunque
- $x \cdot a \equiv 1 \pmod{n}$.

Quindi x rispetta la definizione di *inverso moltiplicativo di a modulo n*!

3 Il Teorema Cinese del Resto

Ora arriva la parte più tosta: un ultimo sforzo! Con tutte queste informazioni, possiamo vedere anche un'altra cosa molto importante: una prima formulazione del *Teorema Cinese del Resto* (Chinese Remainder Theorem, CRT[3] in inglese).
Partiamo da un esempio:

- Sia $x \equiv 47 \pmod{143}$. Nota: $143 = 11 \cdot 13$.
- Un possibile x intero che rispetta l'equazione modulare precedente, è proprio 47.
- In generale, per ogni k intero, $x = 47 + k \cdot 143$ è anch'essa una soluzione.
- $x = 47 \equiv 3 \pmod{11}$.
- $x = 47 \equiv 8 \pmod{13}$.

Supponiamo di non sapere che $x \equiv 47 \pmod{143}$, ma solo che $x \equiv 3 \pmod{11}$ e $x \equiv 8 \pmod{13}$.
Possiamo recuperare l'informazione che $x \equiv 47 \pmod{143 = 11 \cdot 13}$? La risposta è sì, proprio attraverso il CRT.
Vediamo subito come:

- Dato che $GCD(11, 13) = 1$, possiamo dedurre che $6 \cdot 11 - 5 \cdot 13 = 1$, grazie all'algoritmo di Euclide esteso.
- Nota bene: dall'equazione precedente, $6 \cdot 11 \equiv 1 \pmod{13}$ e $-5 \cdot 13 \equiv 1 \pmod{11}$. Inoltre $6 \cdot 11 \equiv 0 \pmod{11}$ e $-5 \cdot 13 \equiv 0 \pmod{13}$.
- Osserva che $X = 8 \cdot (6 \cdot 11) + 3 \cdot (-5 \cdot 13)$ è congruo a $3 \pmod{11}$ e congruo a $8 \pmod{13}$.
- $X = 333 \equiv 47 \pmod{143}$.

In generale, se abbiamo un sistema di congruenze in un'unica incognita x in cui i vari moduli sono *a due a due coprimi*, il CRT ci permette di recuperare il resto di x modulo *il prodotto dei moduli nel sistema.*
In equazioni:

$$\text{Se } \begin{cases} x \equiv x_0 \pmod{n_0} \\ x \equiv x_1 \pmod{n_1} \\ \dots \\ x \equiv x_m \pmod{n_m} \end{cases} \quad \text{e } GCD(n_i, n_j) = 1 \ \forall \ i, j \text{ tali che } i \neq j :$$

$\implies$ è possibile recuperare X tale che $x \equiv X \pmod{N}$, ove $N = n_0 \cdot n_1 \cdot \dots \cdot n_m$.
Per calcolare la soluzione, basta iterare il ragionamento fatto con solo due congruenze come nell'esempio: ogni volta che lo si applica, "si rimuove un'equazione".
Nota come, anche qui, l'unica cosa che è stata utilizzata è l'algoritmo di Euclide esteso.
Per far sedimentare questi importantissimi concetti, ti consiglio di provare ad implementare XGCD e CRT in Python e di usarli per rispondere ad alcuni quesiti delle challenge. In bocca al lupo!

[3] https://it.wikipedia.org/wiki/Teorema_cinese_del_resto