Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Matematica

IL PROBLEMA DEL LOGARITMO DISCRETO

Tesi di Laurea in Algortimi della Teoria dei Numeri e Crittografia

Relatrice:
Chiar.ma Dott.ssa
MARTA MORIGI

Presentata da:
LUCA CASOLI

III Sessione
Anno Accademico 2009-10

Introduzione

Nell'ultimo secolo, grazie al rapido sviluppo delle tecnologie informatiche, la matematica ha trovato numerose nuove applicazioni. In particolare lo sviluppo di internet e delle tecnologie di comunicazione ha portato alla nascita della crittografia, campo ove tecniche matematiche precedentemente ritenute puramente teoriche si sono rivelate fondamentali.

Un sistema crittografico ha come obiettivo quello di celare un messaggio a tutti coloro che sono sprovvisti di una speciale chiave. Per celare il messaggio si utilizzano operazioni matematiche che siano facili da eseguire ma difficili da invertire se non si è in possesso di specifici dati. Il problema più celebre in questo senso è quello della fattorizzazione in primi; mentre moltiplicare due numeri primi è un'operazione semplice, risalire ai fattori con la sola conoscenza del loro prodotto è un problema molto difficile da risolvere quando i numeri in gioco sono grandi.

Un'idea simile sta alla base del problema del logaritmo discreto. In questo caso, in un senso si ha l'operazione di esponenziazione mentre nell'altro l'operazione è quella di ricerca dell'esponente da dare alla base nota per ottenere l'argomento noto. Sui numeri reali quest'operazione è il ben noto logaritmo che da luogo ad una funzione analitica con tutte le proprietà che ne conseguono. Lavorando in un gruppo generico però la situazione è diversa; l'operazione prende il nome di logaritmo discreto ed è un problema di non facile risoluzione. Nel Capitolo 1 saranno presentati in modo formale il logaritmo discreto e le sue proprietà. Nello stesso capitolo sono richiamate alcune nozioni riguardanti lo studio della complessità computazionale di un algoritmo che verranno poi utilizzate nei capitoli successivi. Infine viene illustrata un'applicazione del logaritmo discreto collegandolo ad un altro problema classico della crittografia, il problema di Diffie-Hellman.

Nel Capitolo 2 vengono illustrati i metodi *radice quadrata*, chiamati così per via della loro complessità computazionale. Questi metodi non sono particolarmente efficienti ma permettono un graduale approccio al problema che mette in evidenza risultati e proprietà che giocano un ruolo importante nella ricerca della soluzione. Saranno poi presentati gli algoritmi di Pollard (simili

agli algoritmi di fattorizzazione dello stesso autore) fino a giungere al metodo di Pohlig-Hellman che permette di ricondurre il problema ad una serie di problemi computazionalmente più semplici in casi particolari.

Nel Capitolo 3 sarà infine illustrato il metodo di calcolo dell'indice. Questo metodo è un algoritmo piuttosto generale che permette di trovare la soluzione del problema del logaritmo discreto e che, per la sua implementazione, può fare uso di algoritmi differenti. Ad esempio, in questa trattazione è stato scelto l'uso del crivello di polinomi per la ricerca dei B-numeri. Inoltre una delle fasi dell'algoritmo prevede la risoluzione di un sistema lineare e per essa sono stati proposti il metodo di Lanczos sui campi finiti ed una variante del metodo di eliminazione di Gauss. Alla fine sono riportate brevemente alcune note sulla complessità del metodo: quali sono i parametri in gioco, come sceglierli per ottimizzare l'algoritmo e quali sono i migliori risultati ottenibili al momento.

Indice

Capitolo 1

Il problema del logaritmo discreto

1.1 Il logaritmo discreto

Iniziamo col definire il logaritmo discreto. Consideriamo un gruppo ciclico G di ordine n con operazione di gruppo $*$ e sia g un suo generatore. Allora definiremo l'*esponenziale discreto* come

$$y = g^x = \underbrace{g * g * \ldots * g}_{x} \tag{1.1}$$

dove x è un intero non negativo. Il *problema del logaritmo discreto* (o *problema DL*) consiste nel calcolare x una volta assegnati G, g ed y come precedentemente indicato. I dati g ed y prenderanno rispettivamente il nome di *base* ed *argomento* del logaritmo discreto. Se vale 1.1 si ha che

$$x = \log_g y.$$

dove $\log_g y$ è appunto il *logaritmo discreto di y in base g.*

1.1.1 Soluzioni del problema

Se G è un gruppo ciclico generato da g, allora la sua soluzione al problema del logaritmo discreto in generale non è unica. Infatti se l'ordine di G è n, si ha $g^{x+kn} = g^x \cdot (g^n)^k = g^x = y$ per ogni $k \in \mathbb{Z}$. Possiamo dimostrare il seguente teorema.

Teorema 1.1.1. *Sia g generatore del gruppo ciclico G di ordine n. Allora*

$$g^x = g^z \Leftrightarrow x \equiv z \bmod n.$$

Dimostrazione. Supponiamo $x \equiv z \bmod n$. Allora, per definizione di congruenza, $n|(x-z)$ quindi esiste $k \in \mathbb{Z}$ tale che $x - z = kn$. Ne segue che $g^{x-z} = g^{kn} = 1$ ovvero $g^x = g^z$.

Viceversa, sia per ipotesi $g^x = g^z$. Se, per assurdo, x e z non fossero cogrui modulo n, allora n non dividerebbe $x - z$ quindi $g^{x-z} \neq 1$, contro l'ipotesi. Quindi x e z devono essere congrui modulo n. □

Dal teorema precedente segue che, se il problema del logaritmo discreto ha soluzione, allora questa sarà unica modulo l'ordine del gruppo.

Osservazione 1.1.2. Osserviamo che g^x non dipende da x ma dalla sua classe d'equivalenza modulo n che denotiamo con $[x]_n$. Perciò ha senso definire $g^{[x]_n} = g^x$.

Osservazione 1.1.3. Sarebbe possibile definire in modo analogo a quanto fatto nella sezione 1.1 il logaritmo discreto su un gruppo generico G non ciclico o con una base che non sia un generatore. In questi casi non sarebbe però garantita l'esistenza della soluzione del problema. In queste situazioni è comunque possibile restringersi al sottogruppo ciclico generato dalla base del logaritmo per poter risolvere il problema per ogni elemento di quel sottogruppo. In generale, se non diversamente specificato, nei discorsi che seguiranno considereremo gruppi ciclici.

1.1.2 Difficoltà del problema

Ci sono diversi parametri che influenzano la difficoltà del problema. Uno di questi è l'ordine del gruppo ciclico nel quale si sta lavorando; un ordine grande aumenta ovviamente i possibili elementi e quindi il numero di operazioni che l'algoritmo deve eseguire. Vedremo (sezione 2.5, in particolare osservazione 2.5.7) che non solo la grandezza dell'ordine è importante ma anche la sua fattorizzazione in primi. Vediamo ora che la difficoltà dipende anche dal gruppo scelto.

Supponiamo di considerare il gruppo ciclico $(\mathbb{Z}_{p-1}, +)$, p primo, generato da g. Allora in questo caso, applicando la definizione 1.1, si ha

$$y = g + g + \ldots + g = xg.$$

Se $g = 1$ il problema diventa banale. Scegliamo invece un generatore $g \neq 1$; se $g = [m]_{p-1}$ con $m \in \mathbb{Z}$ si ha che, essendo un generatore, $\mathrm{MCD}(m, p-1) = 1$ e dunque g è invertibile nell'anello $\mathbb{Z}_{p-1}$. Allora $y = gx$ e la soluzione è data da $[x]_{p-1} = g^{-1}y$, quindi il problema è ancora banale.

Senza cambiare p, consideriamo invece il gruppo ciclico $(\mathbb{Z}_p^\star, \cdot)$ generato da h ($\mathbb{Z}_p^\star$ indica gli elementi non nulli di $\mathbb{Z}_p$). Applicando 1.1 si ha

$$y = h^x = h \cdot h \cdot \ldots \cdot h \text{ così } \log_h y = x.$$

Questo non è un problema facile da risolvere in generale e nei capitoli seguenti vedremo dei metodi per trovarne le soluzioni.

Questo mostra come, nonostante l'isomorfismo $(\mathbb{Z}_{p-1}, +) \cong (\mathbb{Z}_p^\star, \cdot)$, il problema abbia complessità computazionale differente nelle due strutture. Questo è vero in generale: l'isomorfismo non implica uguale complessità computazionale sulle due strutture.

Nei capitoli seguenti considereremo G sempre come gruppo moltiplicativo con operazione di gruppo denotata da $\cdot$ se non diversamente specificato.

1.1.3 Proprietà del logaritmo discreto

Ricordiamo le seguenti proprietà del logaritmo definito sui numeri reali:

$$\log_g(a \cdot b) = \log_g a + \log_g b \tag{1.2}$$

$$\log_g a^e = e \cdot \log_g a \tag{1.3}$$

Queste proprietà sono ereditate dal logaritmo discreto.

Teorema 1.1.4. *Il logaritmo discreto gode delle proprietà 1.2 e 1.3.*

Dimostrazione. Supponiamo di essere in un gruppo G di ordine n generato da g. Siano $y_1 = g^{x_1}$ e $y_2 = g^{x_2}$. Allora

$$y_1 * y_2 = g^{x_1+x_2} \text{ quindi } \log_g(y_1 * y_2) \equiv x_1 + x_2 \equiv \log_g y_1 + \log_g y_2 \bmod n$$

quindi la proprietà 1.2 è verificata.

Sia $y = g^x$ e sia $e \geq 0$. Allora

$$y^e = g^{ex} \text{ quindi } \log_g y^e \equiv ex \equiv e \cdot \log_g y \bmod n$$

quindi la proprietà 1.3 è verificata. □

Teorema 1.1.5. *Siano b e g due generatori di un gruppo G avente ordine n. Allora per il logaritmo discreto vale la proprietà di cambio base*

$$[\log_b a]_n = [\log_g a]_n [\log_g b]_n^{-1}. \tag{1.4}$$

Dimostrazione. Innanzitutto i logaritmi esistono e sono ben definiti in quanto g e b sono per ipotesi due generatori. Supponiamo sia $b = g^{[x]_n}$, ovvero, di conseguenza, $[x]_n = [\log_g b]_n$. Essendo b un generatore di G per definizione ne segue che $[\log_g b]_n$ è invertibile. Così si ottiene la relazione

$$a = g^{[\log_g a]_n} = g^{[\log_g a]_n [\log_g b]^{-1} [x]_n} = b^{[\log_g a]_n [\log_g b]_n^{-1}}$$

dove abbiamo usato il fatto che, per definizione, $[x]_n [\log_g b]_n^{-1} = 1$. Applicando $\log_g$ ad entrambi i membri dell'uguaglianza otteniamo che la 1.4 risulta vera. □

Osservazione 1.1.6. Il teorema 1.1.5 ci consente di lavorare con una base generica per risolvere il problema del logaritmo discreto in ogni base.

1.2 Alcune note sulla complessità

Generalmente possono esistere più algoritmi per raggiungere il medesimo risultato a partire dai medesimi dati iniziali. Questo pone il problema di misurare l'efficacia di un algoritmo per poter poi mettere a confronto i vari metodi possibili e scegliere il migliore. Un algoritmo si considera migliore di un altro se ottiene lo stesso risultato con un minor numero di operazioni elementari (e quindi in minor tempo). La stima del tempo impiegato per raggiungere il risultato viene detta *complessità temporale* dell'algoritmo. Talvolta parleremo anche di *complessità spaziale*; questa sarà la capacità di memoria necessaria all'esecuzione dell'algoritmo. In generale, quando parleremo di complessità sottointenderemo complessità temporale se non diversamente specificato. Quale peso dare alle due complessità dipende dal tipo di prestazione che vogliamo, dal problema da risolvere e dai calcolatori in uso.

La complessità dipenderà non solo dall'algoritmo scelto ma anche dai dati in ingresso. Per questo la complessità si esprime in funzione del dato in ingresso n (o, più in generale, dei dati in ingresso $n_1, \dots n_k$). In particolare l'algoritmo rallenta quando il dato in ingresso è più grande perchè le operazioni che esso svolge richiederanno più *operazioni elementari* (ad esempio, intuitivamente, sommare numeri di m cifre richiede m operazioni). Per esprimere il concetto di *grandezza* di un numero si utilizza il suo numero di cifre (nella rappresentazione in base 10 per comodità, ma è possibile fare gli stessi discorsi per una base generica).

In seguito, dato $x \in \mathbb{R}$, si utilizzeranno le seguenti notazioni:

$$\lfloor x \rfloor = \max\{n \in \mathbb{Z} | n \leq x\}, \qquad \lceil x \rceil = \min\{n \in \mathbb{Z} | n \geq x\}.$$

In particolare $\lfloor x \rfloor$ è detta *parte intera di* x.

Proposizione 1.2.1. *Sia* k *il numero di cifre in base* 10 *di* n*. Allora*

$$k = \lfloor \log_{10} n \rfloor + 1.$$

Dimostrazione. Per definizione n ha k cifre in base 10 quindi, in particolare, $10^{k-1} \leq n < 10^k$. Applicando il logaritmo si ottiene la disuguaglianza $k - 1 \leq \log_{10} n < k$. Allora, essendo $\lfloor \log_{10} n \rfloor$ intero ed essendo $k - 1$ l'unico intero dell'intervallo, si deduce

$$\lfloor \log_{10} n \rfloor = k - 1$$

che conclude la dimostrazione. □

Un algoritmo si dice *efficiente* se la sua complessità è al più polinomiale nel numero di cifre di n. Ad esempio, un algoritmo avente complessità $O(\log n)$ è efficiente. Nei prossimi capitoli le complessità saranno espresse sempre in funzione di n e non del suo numero di cifre. Si può cambiare notazione semplicemente applicando la proposizione 1.2.1.

Osservazione 1.2.2. Un algoritmo avente complessità $O(n)$ non è efficiente in quanto la sua complessità è *polinomiale in* n e non *polinomiale nel numero di cifre di* n. Infatti, per la proposizione 1.2.1, le cifre di n sono $\lfloor \log_{10} n \rfloor + 1$ che possiamo approssimare semplicemente con $\log_{10} n$ (nella notazione O-grande questa approssimazione è lecita). Allora otteniamo una complessità

$$O(n) = O(10^k)$$

che è *esponenziale nel numero di cifre di* n. In tal caso l'algoritmo si dice *non efficiente*.

Si parla di *problemi difficili* se non esistono (o non sono conosciuti) algoritmi efficienti per la risoluzione, e di *problemi facili*, se tali algoritmi esistono.

1.3 Il problema di Diffie-Hellman

Il problema del logaritmo discreto trova le sue maggiori applicazioni nel campo della crittografia. Infatti, analogamente al problema della fattorizzazione, è un problema difficile il cui inverso è un problema facile.

Whitfield Diffie e Martin Hellman nel 1976 hanno ideato un protocollo per lo scambio di chiavi segrete da usare per sistemi crittografici a chiave pubblica basato proprio sul problema del logaritmo discreto. Un sistema a

chiave pubblica in generale richiede ai due enti in comunicazione di possedere una stessa chiave per poter decifrare la trasmissione ma questa, per poter essere condivisa, deve passare su un *canale insicuro*, ovvero un canale dove chiunque può vederla. Per poter fare lo scambio su un canale insicuro ma garantire comunque la segretezza, Diffie ed Hellman idearono il seguente metodo.

Supponiamo che Alice e Bob debbano scambiarsi un messaggio senza che nessun'altro sia in grado di leggerlo. Allora Alice e Bob dovranno possedere una chiave in grado di decifrare il messaggio che sia ignota a terzi. Per scambiarsi questa chiave si seguono i passi seguenti:

1. Alice e Bob concordano sul canale insicuro un gruppo ciclico G ed un suo generatore g.

2. Alice genera casualmente un intero a e calcola $K_A = g^a$. L'intero a sarà la *chiave privata* di Alice e dovrà esser mantenuta segreta. Alice comunica la sua *chiave pubblica* K_A a Bob sul canale insicuro.

3. In modo analogo Bob genera una chiave privata b, quindi calcola la chiave pubblica $K_B = g^b$ e la spedisce ad Alice.

4. A questo punto Alice conosce K_B e può calcolare $K_{AB} = K_B^a$. Analogamente Bob può calcolare $K_{BA} = K_A^b$.

5. Per come abbiamo calcolato le chiavi si ha

$$K_{AB} = K_B^a = (g^b)^a = (g^a)^b = K_A^b = K_{BA}$$

   ovvero Alice e Bob sono in possesso della medesima chiave condivisa $K = K_{AB} = K_{AB}$ senza che questa sia passata sul canale insicuro.

Un osservatore esterno che controlla il canale vede passare G, g, K_A, K_B ma non conosce le chiavi private a e b. Il calcolo di $K = g^{ab}$ a partire dalla conoscenza di G, g, g^a, g^b è detto *problema di Diffie-Hellman*.

Congettura 1.3.1 (di Diffie-Hellman)**.** *Il problema di Diffie-Hellman è computazionalmente difficile.*

Osservazione 1.3.2. Se il problema del logaritmo discreto fosse facile potremmo risolvere facilmente anche il problema di Diffie-Hellman. Infatti dai dati noti sarebbe facile calcolare $a = \log_g K_A$, $b = \log_g K_B$. Una volta noti a e b si avrebbe accesso alla chiave condivisa g^{ab}. Quindi la congettura di Diffie-Hellman implica che il problema del logaritmo discreto sia un problema difficile.

Attualmente questo è l'unico metodo noto per poter risolvere il problema di Diffie-Hellman. Non è stato ancora dimostrato se sia possibile risolvere il problema di Diffie-Hellman senza l'utilizzo del logaritmo discreto.

Capitolo 2

Primi metodi di calcolo

In questo capitolo introdurremo alcuni metodi elementari per il calcolo del logaritmo discreto. Partiremo dal metodo di enumerazione che affronta il problema in maniera banale per poi raffinare progressivamente i metodi e le idee di base costruendo algoritmi sempre più efficienti. Questi metodi saranno presentati nella loro forma più generale, ovvero non facendo alcuna ipotesi su G se non che sia ciclico. Spesso ci si riferisce ai metodi qui illustrati come *metodi radice quadrata* riferendosi alla loro complessità che, nel migliore dei casi, è appunto $O(\sqrt{n})$ ove n è l'ordine di G. Se ne deduce che sono metodi dalla complessità esponenziale, ovvero poco efficienti.

Al termine del capitolo illustreremo il metodo di Pohlig-Hellman che permette di velocizzare molto il problema nel caso in cui sia nota la fattorizzazione dell'ordine del gruppo.

2.1 Metodo di enumerazione

Il primo metodo per risolvere il problema $x = \log_g y$ in un gruppo ciclico G di ordine n generato da g è quello di testare tutti i possibili valori dell'esponente x a partire da 0 sostituendoli in $g^x = y$ per vedere se soddisfano l'equazione. Ovviamente il metodo è inadatto (come tutti i *metodi forza bruta*) per affrontare il problema nelle applicazioni reali. Infatti dovranno essere testati, nel peggiore dei casi, n esponenti che corrispondono ad una complessità $O(n)$. Nelle applicazioni crittografiche gli ordini dei gruppi utilizzati sono molto grandi (circa 2^{160}) rendendo impossibile una risoluzione del problema di questo tipo.

2.2 Algoritmo baby-step giant-step di Shanks

Shanks propose questo algoritmo come miglioramento del metodo di enumerazione. In questo caso non si testano tutti i possibili esponenti ma si riesce a ridurre la scelta solo agli elementi di un ristretto insieme. Ciò rende l'algoritmo più veloce ma, d'altra parte, per poter essere eseguito richiede una grande capacità di memorizzazione dei dati.

Si inizia col definire due insiemi come segue.

$$S_g = \{(i, g^{i\lceil\sqrt{n}\rceil}) | i = 0, \ldots, \lceil\sqrt{n}\rceil\}$$

$$S_b = \{(j, y \cdot g^j) | j = 0, \ldots, \lceil\sqrt{n}\rceil\}$$

S_g e S_b rappresentano rispettivamente quelli che sono detti *giant step* e *baby step*, n è l'ordine del gruppo ciclico G generato da g ed $y = g^x$, ovvero y è l'elemento di cui vogliamo calcolare il logaritmo discreto. Consideriamo ora due funzioni

$$\gamma : S_g \to G, \quad (a, b) \mapsto b$$

$$\beta : S_b \to G, \quad (a, b) \mapsto b$$

Cerchiamo ora un elemento $z \in G$ tale che $z = \gamma(a, b) = \beta(c, d)$. Quando si trova un tale z si dice che è stata trovata una *corrispondenza*. Ciò è equivalente a cercare due coppie, una in ognuno degli insiemi su definiti, tali che i secondi elementi siano uguali, ovvero $g^{i\lceil\sqrt{n}\rceil} = y \cdot g^j$. Si ha che

$$g^{i\lceil\sqrt{n}\rceil} = y \cdot g^j \text{ e quindi } y = g^{i\lceil\sqrt{n}\rceil} g^{-j} = g^{i\lceil\sqrt{n}\rceil - j}.$$

Possiamo ora applicare $\log_g$ ad entrambi i membri dell'equazione. Sappiamo però che il logaritmo discreto non è unico in generale ma è unico modulo l'ordine di G. Allora

$$x = \log_g y \equiv \log_g g^{i\lceil\sqrt{n}\rceil - j} \equiv i\lceil\sqrt{n}\rceil - j \bmod n$$

quindi possiamo calcolare x in quanto sono noti $g^{i\lceil\sqrt{n}\rceil} = z$ e gli indici i, j relativi a questo elemento rispettivamente negli insiemi S_g e S_b.

Teorema 2.2.1. *Esiste sempre $z \in G$ tale che $z = \gamma(a, b) = \beta(c, d)$.*

Dimostrazione. Dimostrare che si trova sempre una corrispondenza è equivalente a dimostrare che è sempre possibile trovare i e j non maggiori di $\lceil\sqrt{n}\rceil$ tali che

$$g^{i\lceil\sqrt{n}\rceil} = g^{x+j}.$$

Equivalentemente possiamo dimostrare che ogni intero x dell'intervallo $0 \leq x < n$ può essere scritto nella forma

$$x = i\lceil\sqrt{n}\rceil - j.$$

Supponiamo $i = 1$. Allora al variare di j nell'intervallo $0 \leq j < \lceil\sqrt{n}\rceil$, x assume tutti i valori interi tra 0 e $\lceil\sqrt{n}\rceil$. Analogamente, per un generico valore di i fissato, verranno assunti tutti i valori tra $(i-1)\lceil\sqrt{n}\rceil$ e $i\lceil\sqrt{n}\rceil$. Quindi ogni x in $0 \leq x < n$ può essere espresso tramite un'opportuna scelta di i e j, ovvero si trova sempre una corrispondenza. □

Osservazione 2.2.2. Dato x, definiamo $A = \{(i,j) | x \equiv i\lceil\sqrt{n}\rceil - j \bmod n\}$. Dal teorema 2.2.1 segue che $A \neq \emptyset$. Però A potrebbe in generale contenere più di un elemento. Infatti possiamo notare che, ad esempio,

$$x \equiv i\lceil\sqrt{n}\rceil - \lceil\sqrt{n}\rceil \equiv (i-1)\lceil\sqrt{n}\rceil \bmod n$$

quindi $(i, \lceil\sqrt{n}\rceil), (i-1, 0) \in A$.

Esempio 2.2.3. Supponiamo di avere il gruppo ciclico $\mathbb{Z}_{17}^{\star}$ e fissiamone come generatore $g = [3]_{17}$. Vogliamo calcolare il logaritmo discreto di $y = [5]_{17}$ utilizzando l'algoritmo baby-step giant-step.

Innanzitutto vediamo che l'ordine del gruppo è $n = 16$ e quindi $\lceil\sqrt{n}\rceil = 4$. Calcoliamo allora gli elementi dei due insiemi:

$$S_g = \{(0, [1]_{17}), (1, [13]_{17}), (2, [16]_{17}), (3, [4]_{17}), (4, [1]_{17})\}$$

$$S_b = \{(0, [5]_{17}), (1, [15]_{17}), (2, [11]_{17}), (3, [16]_{17}), (4, [14]_{17})\}$$

Notiamo subito che nelle coppie $(2, [16]_{17}) \in S_g$ e $(3, [16]_{17}) \in S_b$ i secondi elementi sono uguali. Allora in questo caso si ha $i = 2$ e $j = 3$ che ci porta al risultato

$$x \equiv \log_g y \equiv 2 \cdot 4 - 3 \equiv 5 \bmod 16$$

Infatti $3^5 \equiv 243 \equiv 5 \bmod 17$.

Osservazione 2.2.4. Questo metodo richiede il calcolo di due insiemi di elementi ed un confronto tra di essi. Ciò implica che di almeno uno dei due insiemi vengano memorizzati tutti gli elementi per poi poter eseguire le comparazioni con quelli dell'altro. Ogni insieme è composto da $1+\lceil\sqrt{n}\rceil$ coppie di elementi quindi se n è molto grande, come accade nelle applicazioni, è difficile usare questo metodo a causa dell'ingente quantità di memoria necessaria.

Teorema 2.2.5. *L'algoritmo baby-step giant-step richiede $O(\sqrt{n})$ operazioni di gruppo e comparazioni ed $O(\sqrt{n})$ memorizzazioni di elementi di G per risolvere il problema del logaritmo discreto in un gruppo G di ordine n.*

2.3 Algoritmo ρ di Pollard

Il prossimo algoritmo è stato presentato da Pollard [4] nel 1978 come metodo Monte Carlo per il calcolo del logaritmo discreto. La complessità è data in maniera probabilistica, ovvero viene dato quello che è il *valore atteso* della complessità. L'algoritmo ρ fa uso di una successione non lineare calcolata in modo ricorsivo ed assume che il suo comportamento sia *pseudo-casuale*. Con pseudo-casuale si intende un comportamento deterministico che però risulta difficilmente prevedibile se non si conoscono le condizioni iniziali e che, statisticamente, si avvicina al comportamento casuale.

2.3.1 Il paradosso dei compleanni

Prima di iniziare a parlare dell'algoritmo ρ, introduciamo il seguente risultato.

Teorema 2.3.1. *È sufficiente scegliere $O(\sqrt{n})$ elementi da un insieme di n elementi per avere probabilità $\frac{1}{2}$ di sceglierne due uguali.*

Dimostrazione. Siano dati un insieme N di n elementi e k variabili $b_1, \ldots, b_k$ che assumono valori in N. Un evento elementare è dato da una k-pla $(b_1, \ldots, b_k) \in N^k$. Siccome ogni b_i assume valori indipendentemente dagli altri, fissandone i valori sarà possibile individuare n^k eventi distinti. Supponendo che questi eventi siano tutti ugualmente probabili la probabilità di ciascuno di essi è $\frac{1}{n^k}$.

Vogliamo ora calcolare la probabilità p che esistano almeno due indici h, l tali che $b_h = b_l$. Per farlo, possiamo notare che $p = 1 - q$ dove q è la probabilità che i b_i siano tutti distinti. Chiamiamo E l'insieme degli eventi tali che i b_i siano tutti distinti.

Supponiamo ora di fissare l'elemento b_1. Se si vuole $b_2 \neq b_1$, allora b_2 sarà scelto nell'insieme $N - \{b_1\}$ che ha $n - 1$ elementi. Ragionando in questo modo si deduce che

$$|E| = n(n-1) \cdot \ldots \cdot (n-k+1) = \prod_{i=0}^{k-1}(n-i)$$

dove $|E|$ è il numero degli eventi con tutti i b_i distinti. La probabilità che venga considerato un evento di questo tipo è

$$q = \frac{1}{n^k}|E| = \frac{1}{n^k}\prod_{i=0}^{k-1}(n-i) = \prod_{i=1}^{k-1}\Big(1 - \frac{i}{n}\Big). \tag{2.1}$$

Osserviamo che $1 + x \leq e^x$ per ogni $x \in \mathbb{R}$. Sfruttando ciò, partendo da 2.1 possiamo maggiorare q come segue:

$$q \leq \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{1}{n}\sum_{i=1}^{k-1} i} = e^{-\frac{k(k-1)}{2n}}. \tag{2.2}$$

Supponiamo ora che sia $k \geq \frac{1}{2}(1 + \sqrt{1 + 8n \log 2})$. Allora si ha:

$$\begin{aligned}
\frac{k(k-1)}{2n} &\geq \frac{1}{2n}\Big(\frac{1}{2}\Big(1 + \sqrt{1 + 8n \log 2}\Big)\Big(\frac{1}{2}\Big(1 + \sqrt{1 + 8n \log 2}\Big) - 1\Big)\Big) \\
&= \frac{1}{8n}\Big((1 + \sqrt{1 + 8n \log 2})(\sqrt{1 + 8n \log 2} - 1)\Big) \\
&= \frac{1}{8n}\Big(1 + 8n \log 2 - 1\Big) \\
&= \log 2.
\end{aligned}$$

Essendo $-\frac{k(k-1)}{2n}$ l'esponente in 2.2, andando a sostituire quanto appena calcolato si ha

$$q \leq e^{-\frac{k(k-1)}{2n}} \leq e^{-\log 2} = \frac{1}{2}$$

ovvero, scegliendo k come indicato, la probabilità q di avere un evento in E tra tutti quelli possibili è maggiorata da $\frac{1}{2}$. Questo significa che la probabilità di avere un evento con almeno due elementi uguali è

$$p = 1 - q \geq \frac{1}{2}.$$

Questo è vero per ogni $k \geq \frac{1}{2}(1 + \sqrt{1 + 8n \log 2})$, quindi in particolare lo sarà per

$$k = \lceil \frac{1}{2}(1 + \sqrt{1 + 8n \log 2}) \rceil \leq O(\sqrt{n}).$$

□

Esempio 2.3.2 (Paradosso dei compleanni)**.** Il paradosso dei compleanni (o problema dei compleanni) è un quesito formulato come segue:

Si vuole avere il 50% di probabilità di trovare almeno due persone che compiono gli anni lo stesso giorno tra quelle presenti in una stanza. Quante persone devono esserci?

Il quesito si è meritato l'appellativo di *paradosso* in quanto il risultato è lontano dall'intuizione comune (e non perchè è una contraddizione logica).

Risolviamolo applicando il teorema 2.3.1.

In questo caso k rappresenta il numero delle persone presenti nella stanza ed n è il numero dei possibili compleanni. I compleanni corrispondono ai possibili giorni di un anno ma una data, il 29 febbraio, non è equiprobabile con le altre in quanto si presenta solo una volta ogni quattro anni. Supponiamo quindi di restringere il problema ad $n = 365$. Per i risultati ottenuti dimostrando il teorema 2.3.1 dovremo avere

$$k \geq \frac{1}{2}\Big(1 + \sqrt{1 + 8 \cdot 356 \cdot \log 2}\Big) \approx 23.$$

Così, selezionando casualmente 23 persone, si ha una probabilità maggiore o uguale al 50% di averne selezionate almeno due che compiono gli anni nello stesso giorno. Verifichiamo che sia vero.

Riprendiamo la notazione in k-ple per gli eventi utilizzata nella dimostrazione del teorema 2.3.1. Utilizzando i 365 elementi disponibili, essendo $k = 23$ possiamo costruire 365^{23} k-ple. Di queste, quelle che avranno tutti gli elementi differenti saranno $\frac{365!}{(365-23)!}$. La proporzione di tali k-ple sul totale è data da

$$q = \frac{365!}{(365-23)!} \cdot \frac{1}{365^{23}}$$

che, essendo queste tutte equiprobabili, corrisponde anche alla probabilità di sceglierne una dall'insieme totale. Così la probabilità richiesta dal problema è $p = 1 - q$. Aiutandosi con un programma di calcolo si può arrivare al risultato

$$p \approx 0.5073$$

ovvero la probabilità è di poco maggiore al 50%, in linea con quanto affermato dal teorema.

2.3.2 Funzionamento dell'algoritmo ρ

Sia data una partizione $\{G_1, G_2, G_3\}$ di G dove G_1, G_2, G_3 hanno cardinalità simili. Definiamo poi una funzione $f : G \to G$ come segue:

$$f(\alpha) = \begin{cases} y \cdot \alpha & \text{se } \alpha \in G_1 \\ \alpha^2 & \text{se } \alpha \in G_2 \\ g \cdot \alpha & \text{se } \alpha \in G_3 \end{cases}$$

Fissiamo un a_0 preso a caso dall'insieme $\{0, \ldots, n-1\}$ dove n è l'ordine di G. Definendo $\alpha_0 = g^{a_0}$ determiniamo la successione $(\alpha_i)_{i \geq 0}$ definita per ricorsione come $\alpha_{i+1} = f(\alpha_i)$. Sia ancora $y = g^x$; in generale α_i è della

forma $\alpha_i = y^{b_i} g^{a_i}$. In questo modo, avendo definito $\alpha_0 = g^{a_0}$, abbiamo già definito implicitamente $b_0 = 0$. Dalla definizione della funzione f abbiamo così determinato in modo univoco le successioni $(a_i)_{i\geq 0}$ $(b_i)_{i\geq 0}$. In particolare possiamo notare che

$$a_{i+1} = \begin{cases} a_i + 1 \bmod n & \text{se } \alpha_i \in G_1 \\ 2a_i \bmod n & \text{se } \alpha_i \in G_2 \\ a_i & \text{se } \alpha_i \in G_3 \end{cases}$$

$$b_{i+1} = \begin{cases} b_i & \text{se } \alpha_i \in G_1 \\ 2b_i \bmod n & \text{se } \alpha_i \in G_2 \\ b_i + 1 \bmod n & \text{se } \alpha_i \in G_3 \end{cases}$$

Queste sono successioni di elementi di un gruppo finito, quindi esistono i, k tali che $\alpha_i = \alpha_{i+k}$. Nell'algoritmo ρ si parla di *corrispondenza* quando si trovano due indici i, j tali che $\alpha_i = \alpha_j$. Ora:

$$y^{a_i} \cdot g^{b_i} = y^{a_{i+k}} \cdot g^{b_{i+k}} \text{ quindi } y^{a_i - a_{i+k}} = g^{b_{i+k} - b_i}. \tag{2.3}$$

Per ipotesi $y = g^x$. Sostituendo in 2.3 troviamo che $g^{x(a_i - a_{i+k})} = g^{b_{i+k} - b_i}$. Applicando $\log_g$ ad entrambi i membri dell'equazione e ricordando che il logaritmo discreto è unico solo modulo l'ordine di G possiamo dire che x deve essere soluzione della seguente congruenza:

$$x(a_i - a_{i+k}) \equiv b_{i+k} - b_i \bmod n$$

Se $a_i - a_{i+k}$ è invertibile modulo n allora x è univocamente determinato modulo n. Se non è invertibile, in generale si ottengono più soluzioni e si deve verificare quali di esse sono soluzioni di $x = \log_g y$. Se otteniamo troppe soluzioni da verificare, è possibile eseguire nuovamente l'algoritmo a partire da un α_0 differente.

Proposizione 2.3.3. *Devono esser calcolati mediamente $O(\sqrt{n})$ elementi della successione $(\alpha_i)_{i\geq 0}$ prima di trovare una coppia (i, k) tale che $\alpha_i = \alpha_{i+k}$.*

Dimostrazione. Questo è conseguenza del teorema 2.3.1. □

Osservazione 2.3.4. La successione $(\alpha_i)_{i\geq 0}$ in realtà non è una scelta casuale di elementi dal gruppo G ma è una successione ricorsiva calcolata in maniera deterministica a partire da una condizione iniziale casuale. La successione è però pseudocasuale quindi può essere trattata come se fosse casuale senza incorrere in errori apprezzabili.

Proposizione 2.3.5. *La successione $(\alpha_i)_{i\geq 0}$ è periodica dopo la prima corrispondenza trovata.*

Dimostrazione. Sia $(s, s+k)$ la coppia di indici che identificano la prima corrispondenza $\alpha_s = \alpha_{s+k}$, ove $k > 0$. La successione $(\alpha_i)_{i\geq 0}$ è una successione ricorsiva di rango 1, cioè l'i-esimo elemento dipende esclusivamente dall'$(i-1)$-esimo. Ne viene la seguente implicazione che dimostra la proposizione:

$$\alpha_s = \alpha_{s+k} \text{ e quindi } \alpha_{s+l} = \alpha_{s+k+l} \text{ per ogni } l \geq 0.$$

□

Se s è il più piccolo indice per il quale si trovi una corrispondenza, chiameremo *antiperiodo* la successione $(\alpha_i)_{0\leq i\leq s-1}$ e *periodo* la successione $(\alpha_i)_{s\leq i\leq s+k-1}$ dove k è la *lunghezza del periodo*.

Osservazione 2.3.6. L'algoritmo deve il suo nome alla forma della lettera ρ che descrive graficamente il comportamento della successione; la gamba della lettera rappresenta l'antiperiodo che si inserisce in un ciclo (periodo).

Esempio 2.3.7. Utilizziamo l'algoritmo ρ per risolvere il medesimo problema presentato nell'esempio 2.2.3, ovvero consideriamo $\mathbb{Z}_{17}^\star$ generato da $g = [3]_{17}$ e cerchiamo il logaritmo discreto di $y = [5]_{17}$.

Iniziamo col fissare una partizione degli elementi di $\mathbb{Z}_{17}^\star$. Per comodità possiamo considerare

$$\begin{aligned} G_1 &= \{[1]_{17}, [2]_{17}, [3]_{17}, [4]_{17}, [5]_{17}\} \\ G_2 &= \{[6]_{17}, [7]_{17}, [8]_{17}, [9]_{17}, [10]_{17}\} \\ G_3 &= \{[11]_{17}, [12]_{17}, [13]_{17}, [14]_{17}, [15]_{17}, [16]_{17}\} \end{aligned}$$

in modo che abbiano cardinalità simili. Fissiamo ad arbitrio a_0, ad esempio $a_0 = 2$, e calcoliamo quindi

$$[\alpha_0]_{17} = [g]_{17}^{a_0} = [3]_{17}^2 = [9]_{17}.$$

Essendo $\alpha_0 \in G_2$, per calcolare α_1 dobbiamo elevare al quadrato, ovvero:

$$[\alpha_1]_{17} = [\alpha_0]_{17}^2 = [81]_{17} = [13]_{17}.$$

Ora α_1 è in G_3. Proseguendo in questo modo possiamo calcolare:

$$[\alpha_2]_{17} = [g \cdot \alpha_1]_{17} = [39]_{17} = [5]_{17}, \text{ così } [\alpha_2]_{17} \in G_1$$

$$[\alpha_3]_{17} = [y \cdot \alpha_2]_{17} = [25]_{17} = [8]_{17}, \text{ così } [\alpha_3]_{17} \in G_2$$

$$[\alpha_4]_{17} = [\alpha_3]_{17}^2 = [64]_{17} = [13]_{17}, \text{ così } [\alpha_4]_{17} \in G_1.$$

Abbiamo trovato il ciclo in quanto $[\alpha_1]_{17} = [\alpha_4]_{17}$. In particolare, mantenendo le notazioni utilizzate nella presentazione dell'algoritmo, il ciclo inizia in $i = 1$ ed ha lunghezza $k = 3$. Sappiamo che la soluzione di $x = \log_g y$ è una soluzione di

$$x(a_1 - a_4) \equiv b_4 - b_1 \bmod 16.$$

Gli esponenti possono essere annotati durante il calcolo della successione oppure calcolati in seguito come visto durante la presentazione dell'algoritmo. Ad ogni modo, quelli dell'esempio risultano essere:

$$a_1 = 4, \;\; b_1 = 0 \qquad a_4 = 10, \;\; b_4 = 2.$$

Sostituendo i valori nella precedente congruenza si ha:

$$-6x \equiv 2 \bmod 16 \text{ ovvero } 10x \equiv 2 \bmod 16 \text{ e quindi } 5x \equiv 1 \bmod 8.$$

La soluzione è $x \equiv 5 \bmod 8$. Questa, come visto anche nell'esempio 2.2.3, è proprio la soluzione di $x = \log_3 5$ in quanto $3^5 \equiv 5 \bmod 17$.

2.3.3 Efficienza dell'algoritmo ρ

L'algoritmo così come presentato necessiterebbe della memorizzazione delle terne $A_i = (\alpha_i, a_i, b_i)$ richiedendo, così come l'algoritmo baby-step giant-step, una capacità di memoria dell'ordine di $O(\sqrt{n})$. In realtà è possibile migliorare questa caratteristica facendo alcune osservazioni che consentiranno di memorizzare solo una terna per volta.

Si memorizza innazitutto la terna iniziale $A_0 = (\alpha_0, a_0, b_0)$. Quindi si calcola $A_1 = (\alpha_1, a_1, b_1)$ e si controlla se $\alpha_0 = \alpha_1$. Se è così, l'algoritmo termina in quanto abbiamo trovato una corrispondenza, altrimenti si memorizza la terna A_1 sovrascrivendo la A_0. Si calcola quindi A_2 e nuovamente si controlla se $\alpha_1 = \alpha_2$: se cioè accade, l'algoritmo termina, altrimenti si memorizza A_2 sovrascrivendo A_1. A questo punto si calcola A_3 e si controlla se $\alpha_2 = \alpha_3$. Se ciò avviene, l'algoritmo termina, altrimenti proseguiamo senza memorizzare A_3. Si calcola A_4 e si controlla se $\alpha_2 = \alpha_4$ (non possiamo confrontare α_4 con α_3 perchè non abbiamo memorizzato A_3); se abbiamo trovato l'uguaglianza l'algoritmo termina, altrimenti si memorizza A_4.

L'algoritmo procede iterando il procedimento precedente. Supponiamo di avere memorizzato la terna A_i. Allora si calcola la terna A_{i+1} e si controlla se $\alpha_i = \alpha_{i+1}$. Se così non è, non si memorizza A_{i+1} ma si prosegue calcolando i vari A_{i+j} continuando di volta in volta a controllare se $\alpha_i = \alpha_{i+j}$. Quando

si arriva a calcolare $A_{i+i} = A_{2i}$ si fa nuovamente il confronto tra α_i e α_{2i}. Se non sono uguali, si sovrascrive A_i con A_{2i} e si prosegue con l'algoritmo.

In questo modo occuperemo un solo spazio di memoria (ogni terna memorizzata sovrascrive la precedente). In particolare saranno memorizzate solo le terne A_0 ed A_{2^m} per $m \in \mathbb{N}$.

Si deve verificare che in questo modo si trova una corrispondenza. Supponiamo allora di aver memorizzato A_i per un qualche $i = 2^m$ e sia s l'indice della prima corrispondenza. Se $i \geq s$ allora α_i appartiene al periodo. Sia k la lunghezza del periodo. Se $i \geq k$ consideriamo la successione

$$\alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_{i+2^m}.$$

Questa ha 2^m elementi. Essendo per definizione $i = 2^m$ e per ipotesi $i \geq k$, la successione ha almeno tanti elementi quant'è la lunghezza del ciclo e quindi uno di essi deve essere proprio α_i. Inoltre questa è la successione degli α_j che viene calcolata dopo la memorizzazione della terna A_i ed ognuno di questi α_j sarà confrontato con α_i. Avendo osservato che α_i appartiene al periodo e che tra gli α_j ci sono tutti gli elementi del periodo, una corrispondenza sarà trovata.

Quindi con questo metodo sicuramente è possibile localizzare una corrispondenza quando sarà memorizzata la i-esima terna dove $i \geq \max\{s, k\}$. Se $i = 2^m$, la corrispondenza sarà trovata entro il calcolo della 2^{m+1}-esima terna.

Teorema 2.3.8. *L'algoritmo ρ di Pollard richiede $O(\sqrt{n})$ operazioni e $O(1)$ spazio disponibile per trovare la soluzione cercata.*

Dimostrazione. Antiperiodo e periodo hanno lunghezza totale $O(\sqrt{n})$ quindi prima di trovare una corrispondenza tra due elementi ne dovranno essere calcolati al più $O(\sqrt{n})$. Inoltre con il metodo illustrato è sufficiente memorizzare una sola terna per volta richiedendo quindi una capacità di memoria dell'ordine di $O(1)$. □

2.4 Algoritmo λ di Pollard

Nello stesso lavoro in cui propose il suo algoritmo ρ, Pollard illustrò anche l'algoritmo λ. Questo metodo consente di migliorare l'efficienza del metodo ρ nel caso in cui sia noto (e sia abbastanza piccolo) un intervallo all'interno del quale l'esponente cercato deve cadere. L'algoritmo λ è probabilistico di tipo Monte Carlo, ovvero può fallire e non fornire una soluzione (si veda l'osservazione 2.4.5).

2.4.1 Catturare i canguri

Questo algoritmo è anche chiamato *algoritmo del canguro*. Ciò perchè Pollard lo presentò nel suo lavoro originale [4] come *A Lambda Method for Catching Kangaroos*. In questo introduceva il metodo come segue.

Supponiamo di voler catturare un *canguro selvatico* che si muove lungo una strada nota ma con una sequenza di salti apparentemente imprevedibili. In realtà la lunghezza dei salti è funzione dello stato del terreno in quel punto. Per effettuare la cattura avremo bisogno dell'aiuto di un secondo canguro, un *canguro addomesticato*, che abbia lo stesso modo di saltare sullo stesso tipo di terreno. Partendo col nostro canguro da una posizione a scelta, effettuiamo un certo numero di salti e dunque posizioniamo una trappola. Così, se il canguro selvatico dovesse passare in una delle posizioni in cui siamo passati col canguro addomesticato, seguendo le stesse regole di salto dovrà percorrere la traiettoria del canguro addomesticato fino alla fine, ovvero fino alla trappola.

Se i possibili luoghi in cui posizionarsi sulla strada sono m ed i salti effettuati dal canguro addomesticato sono N (ovvero ha toccato $N+1$ luoghi considerando il punto di partenza), definiamo $\theta = \frac{N+1}{m}$. Supponiamo che il canguro selvatico passi in ogni luogo con la stessa probabiltà $\frac{1}{m}$ e che la scelta del luogo ad ogni salto avvenga in modo indipendente, ovvero supponiamo l'indipendenza stocastica. Con queste premesse, la probabilità di effettuare la cattura sarà

$$p = 1 - \left(1 - \frac{1}{m}\right)^{N+1} = 1 - \left(1 - \frac{1}{m}\right)^{\theta m}. \tag{2.4}$$

Infatti la probabilità che il canguro non passi in un dato luogo è $1 - \frac{1}{m}$, quindi $(1 - \frac{1}{m})^{N+1}$ è la probabilità che ha di non passare in nessuno dei $N+1$ luoghi toccati dal canguro addomesticato (dov'è stata usata l'ipotesi di indipendenza stocastica). La probabilità di effettuare la cattura sarà proprio la probabilità complementare a quella trovata: $1 - (1 - \frac{1}{m})^{N+1}$. Utilizzando poi la definizione di θ si trova la probabilità espressa in 2.4.

In particolare si può osservare che, se θ è abbastanza grande, è possibile approssimare

$$1 - \left(1 - \frac{1}{m}\right)^{\theta m} \sim 1 - e^{-\theta}$$

con l'uso del seguente limite notevole (ponendo $x = m$, $\alpha = -1$, $\beta = \theta$):

$$\lim_{x \to +\infty} \left(1 + \frac{\alpha}{x}\right)^{\beta x} = e^{\alpha\beta}.$$

Fissato quindi il numero di possibili luoghi, possiamo calcolare quanti salti è necessario fare per avere una probabilità data di catturare il canguro selvatico.

2.4.2 Funzionamento dell'algoritmo λ

Supponiamo di sapere che il logaritmo cercato cada nell'intervallo $[a, b]$. Costruiamo un insieme $D = \{d_1, \ldots, d_r\}$ che sarà l'insieme delle *lunghezze dei salti* dove il j-esimo *salto* sarà definito come $e_j = g^{d_j}$. In questo modo l'insieme dei salti sarà $J = \{g^{d_1}, \ldots, g^{d_r}\}$. Costruiamo poi una funzione $H : G \to \{1, \ldots, r\}$ suriettiva in modo che la partizione

$$S_i = \{c \in G | H(c) = i\}, \;\; i = 1, \ldots, r$$

sia composta da insiemi aventi cardinalità dello stesso ordine di grandezza. Per com'è definita, H associa ad ogni elemento del gruppo G un indice tra 1 ed r. In particolare possiamo considerare che l'immagine di ogni elemento sia casuale (ma sempre la stessa). Nella pratica, per H può essere usata una funzione hash.

Definiamo quindi due successioni come segue:

- La prima successione inizia da un valore noto. Abbiamo supposto che il logaritmo cercato sia in $[a, b]$ così definiamo $t_0 = g^{\frac{a+b}{2}}$ (non è restrittivo supporre $\frac{a+b}{2} \in \mathbb{Z}$, eventualmente si può ampliare l'intervallo per far sì che $a + b$ sia un numero pari). In questo modo il punto iniziale è equidistante dagli estremi dell'intervallo. Ricorsivamente definiamo la successione come $t_{i+1} = t_i \cdot e_{H(t_i)}$. Questa successione rappresenta il percorso del canguro addomesticato; parte da una posizione nota ed è sempre prevedibile in ogni istante.

- La seconda successione inizia dall'elemento di cui stiamo cercando il logaritmo, ovvero $w_0 = y$. Ricorsivamente definiamo la successione come $w_{i+1} = w_i \cdot e_{H(w_i)}$. Questa successione parte dalla posizione y, ovvero dalla posizione con x incognita, quindi rappresenta l'imprevedibile percorso del canguro selvatico.

Consideriamo la successione $(t_i)_{i \geq 0}$. Per com'è definita, ogni nuovo elemento viene calcolato moltiplicando la posizione attuale t_i per il salto $H(t_i)$-esimo. Se la funzione H è quella precedentemente definita, allora questo significa che il salto per il quale verrà moltiplicato t_i è dato in modo casuale (o pseudocasuale) da H sull'insieme J dei salti. Notiamo che questo non significa che *ogni volta* il valore è scelto a caso ma che H è definita in modo casuale in principio e quindi rimane fissata. Questo implica che se $t_i = t_j$ si avrà $H(t_i) = H(t_j)$. Analogo discorso può essere fatto per la successione $(w_i)_{i \geq 0}$.

L'obiettivo dell'algoritmo è quello di trovare una intersezione tra le due successioni in modo da dedurre la posizione iniziale (o meglio l'esponente ad essa associato) studiando gli elementi comuni di $(t_i)_{i \geq 0}$ e $(w_i)_{i \geq 0}$.

Proposizione 2.4.1. *Siano $(t_i)_{i\geq 0}$ e $(w_i)_{i\geq 0}$ le due successioni trovate dall'algoritmo λ. Supponiamo che esistano $k = \min\{i \in \mathbb{N} |\ \exists j \in \mathbb{N} : t_i = w_j\}$ ed $h = \min\{j \in \mathbb{N} | t_k = w_j\}$. Allora $t_{k+l} = w_{h+l}$ per ogni $l \geq 0$.*

Dimostrazione. Sia $t_k = w_h$ per ipotesi e vogliamo dimostrare che $t_{k+l} = w_{h+l}$ per ogni $l \in \mathbb{N}$. Proviamolo per induzione su l. Per $l = 0$ l'affermazione si restringe alla semplice ipotesi quindi è banalmente vera. Supponiamo allora $t_{k+l} = w_{h+l}$ e mostriamo che $t_{k+l+1} = w_{h+l+1}$. Applicando l'ipotesi induttiva alla definizione ricorsiva delle successioni si trova

$$t_{k+l+1} = t_{k+l} \cdot e_{H(t_{k+l})} = w_{h+l} \cdot e_{H(w_{h+l})} = w_{k+l+1}$$

che, per il principio di induzione, conclude la dimostrazione. □

Osservazione 2.4.2. Per definizione, $t_i = t_{i-1} \cdot e_{H(t_{i-1})}$ ed $e_j = g^{d_j}$. Ricordando che $t_0 = g^{\frac{a+b}{2}}$ otteniamo $t_1 = g^{\frac{a+b}{2}} \cdot g^{d_{H(t_0)}}$. Considerando $d = d_{H(t_0)}$ si ha $t_1 = g^{\frac{a+b}{2}} \cdot g^d$. Iterando il procedimento, per ogni elemento della successione $(t_i)_{i\geq 0}$ è sempre possibile trovare un esponente $d_{i,t}$ tale che $t_i = g^{\frac{a+b}{2}} \cdot g^{d_{i,t}}$. Analogamente per ogni elemento della successione $(w_i)_{i\geq 0}$ è sempre possibile trovare un esponente $d_{i,w}$ tale che $w_i = y \cdot g^{d_{i,w}}$. Tali $d_{i,t}$ e $d_{i,w}$ rappresentano la *lunghezza totale percorsa* rispettivamente dal canguro ammaestrato e da quello selvatico.

Proposizione 2.4.3. *Sia $t_k = w_h$ il primo elemento comune alle due successioni, ovvero con h e k presi come definiti nella proposizione 2.4.1. Per l'osservazione 2.4.2 esistono d_t e d_w tali che $t_k = g^{\frac{a+b}{2}} \cdot g^{d_t}$ e $w_h = y \cdot g^{d_w}$. Allora*

$$\log_g y \equiv \frac{a+b}{2} + d_t - d_w \bmod n. \tag{2.5}$$

Dimostrazione. Per ipotesi si ha $t_k = w_h$ ovvero $g^{\frac{a+b}{2}} \cdot g^{d_t} = y \cdot g^{d_w}$. Quindi $y = g^{\frac{a+b}{2}+d_t-d_w}$. Applicando $\log_g$ ai membri dell'equazione e ricordando che il logaritmo discreto è unico solo modulo l'ordine n di G si trova 2.5. □

Teorema 2.4.4. *L'algoritmo λ di Pollard richiede $O(\sqrt{b-a})$ operazioni di gruppo e $O(\log(b-a)\log n)$ spazio disponibile per trovare la soluzione cercata dove n è l'ordine del gruppo.*

Per i dettagli della dimostrazione della complessità dell'algoritmo λ e per alcuni risultati sperimentali si può far riferimento a [5].

Osservazione 2.4.5. L'algoritmo ρ è probabilistico nella stima della complessità perchè è possibile prevedere il numero di operazioni da compiere prima di trovare un ciclo solo in termini di media. Mentre l'ingresso in un

ciclo era il fine ultimo dell'algoritmo ρ, nell'algoritmo λ questa eventualià può significare il fallimento del metodo. Infatti quando una successione entra in un ciclo non calcola nuovi valori e quindi potrebbe non avvenire mai una collisione con l'altra successione. La probabilità che ciò accada è però piuttosto bassa; in particolare si dimostra che scegliendo un intervallo $[a, b]$ di lunghezza inferiore ad un quarto dell'ordine di g, tale probabilità è praticamente trascurabile [5]. Se si capita in uno di questi casi è possibile eseguire nuovamente l'algoritmo cambiando l'intervallo scelto o l'insieme dei salti.

Osservazione 2.4.6. L'algoritmo deve il suo nome alla forma della lettera λ che descrive graficamente il comportamento delle due successioni che partono da valori distinti per poi incontrarsi e proseguire uguali.

Esempio 2.4.7. Riprendiamo sempre l'esempio 2.2.3 applicando questa volta l'agoritmo λ. Ricordiamo che consideriamo $\mathbb{Z}_{17}^{\star}$ generato da $g = [3]_{17}$ e cerchiamo il logaritmo discreto di $y = [5]_{17}$.

Supponiamo di sapere che la soluzione si trova nell'intervallo $[a, b] = [3, 5]$ (in questo caso ne siamo sicuri dall'esempio 2.2.3). Allora $\frac{a+b}{2} = 4$. Definiamo quindi, scegliendoli a caso, delle lunghezze d_i per i salti ed i rispettivi salti $e_i = g^{d_i}$. In questo esempio consideriamo:

$$\begin{aligned} D &= \{d_1 = 1, d_2 = 2, d_3 = 3, d_4 = 6\} \\ J &= \{e_1 = [3]_{17}, e_2 = [9]_{17}, e_3 = [10]_{17}, e_4 = [15]_{17}\}. \end{aligned}$$

Definiamo ora la funzione H. Questa può essere definita in modo arbitriario a patto che, come detto nella spiegazione dell'algoritmo, le preimmagini formino una partizione degli elementi di $\mathbb{Z}_{17}^{\star}$ composta da insiemi aventi cardinalità simili. In questo esempio definiamo H come segue:

$$H([1]_{17}) = H([2]_{17}) = H([3]_{17}) = H([4]_{17}) = 1$$

$$H([5]_{17}) = H([6]_{17}) = H([7]_{17}) = H([8]_{17}) = 2$$

$$H([9]_{17}) = H([10]_{17}) = H([11]_{17}) = H([12]_{17}) = 3$$

$$H([13]_{17}) = H([14]_{17}) = H([15]_{17}) = H([16]_{17}) = 4$$

Infine definiamo i punti di partenza delle successioni:

$$w_0 = y = [5]_{17}, \quad t_0 = [g^{\frac{a+b}{2}}]_{17} = [3^4]_{17} = [13]_{17}.$$

Fatto ciò iniziamo a calcolare le successioni partendo da t_1. Vediamo che $H(t_0) = H([13]_{17}) = 4$ quindi

$$t_1 = t_0 \cdot e_4 = [13]_{17} \cdot [15]_{17} = [8]_{17}.$$

Analogamente calcoliamo il primo termine della successione $(w_i)_{i\geq 0}$ notando che $H(w_0) = H([5]_{17}) = 2$:

$$w_1 = w_0 \cdot e_2 = [5]_{17} \cdot [9]_{17} = [11]_{17}.$$

Grazie alla definizione ricorsiva delle due successioni potremo calcolare tutti i successivi termini. In realtà sarà sufficiente calcolarne un altro, infatti

$$t_2 = t_1 \cdot e_2 = [8]_{17} \cdot [9]_{17} = [4]_{17}$$

$$w_2 = w_1 \cdot e_3 = [11]_{17} \cdot [10]_{17} = [8]_{17}$$

ovvero abbiamo trovato che $w_2 = t_1$. Riprendendo le notazioni utilizzate nella presentazione dell'algoritmo, avremo che $k = 1$ e $h = 2$, così $t_{k+l} = w_{h+l}$ per ogni $l \in \mathbb{N}$. Le distanze totali percorse dalle successioni sono $d_t = 6$ e $d_w = 5$, infatti

$$t_1 = [3^4]_{17} \cdot [3^6]_{17} = [5]_{17} \cdot [3^5]_{17} = w_2.$$

Per concludere, sappiamo che la soluzione si ricava da

$$x \equiv \frac{a+b}{2} + d_t - d_w \equiv 4 + 6 - 5 \equiv 5 \bmod 17$$

che è proprio la soluzione del problema del logaritmo discreto.

2.4.3 Algoritmo λ parallelo di Pollard

Allo scopo di migliorare l'efficienza dell'algoritmo λ, Pollard propose una sua parallelizzazione. Si può vedere l'idea che sta alla base di questa variante riprendendo la metafora dei canguri di Pollard. Mentre nell'algoritmo originale erano coinvolti due canguri, uno selvatico ed uno addomesticato, nell'algoritmo parallelo si suppone di lavorare con due mandrie di canguri.

Supponiamo di disporre di m calcolatori. Si considerano u canguri addomesticati e v canguri selvatici dove u e v sono coprimi vicini a $\frac{m}{2}$ e tali che $u + v \leq m$. In questo modo sarà possibile associare ad ogni canguro (e quindi ad ogni successione) ad un calcolatore avendo però u, v dello stesso ordine di grandezza. Ad ogni passo, ogni calcolatore calcolerà il valore della successione a lui associata. Come nell'algoritmo originale si costruisce l'insieme D delle lunghezze dei salti. In questo caso, però, gli elementi di D non saranno arbitrari ma dovranno essere multipli di uv. Denoteremo con T_i e W_j rispettivamente la i-esima successione relativa ai canguri addomesticati e la j-esima successione relativa a quelli selvatici. Si definiscono i valori iniziali di queste successioni come segue:

$$t_0(T_i) = g^{\frac{a+b}{2}+iv}, \text{ per ogni } i \in \{0, \ldots, u-1\}$$

$$w_0(W_j) = y \cdot g^{ju}, \text{ per ogni } j \in \{0, \dots, v-1\}$$

Chiameremo *collisione* l'uguaglianza tra due elementi di diverse successioni. Se si trovano ad esempio degli appropriati indici k, l, i, j tali che $t_k(T_i) = w_l(W_j)$ si dirà che è stata trovata una collisione.

Notiamo che canguri della stessa mandria viaggiano a distanze tali da non incontrarsi mai. Questo si formalizza con la seguente proposizione.

Proposizione 2.4.8. *Non possono avvenire collisioni tra due successioni T_i o tra due successioni W_j.*

Dimostrazione. Consideriamo $i_1 \neq i_2$ e siano T_{i_1}, T_{i_2} le rispettive successioni. Per definizione i valori iniziali delle due successioni non sono congrui modulo u, infatti

$$\frac{a+b}{2} + i_1 v \equiv \frac{a+b}{2} + i_2 v \bmod uv \text{ ovvero } vi_1 \equiv vi_2 \bmod uv.$$

Quest'ultima congruenza è equivalente a $i_1 \equiv i_2 \bmod u$. Se $i_1 \neq i_2$, essendo $i_1, i_2 \in \{0, \dots, u-1\}$ i due non possono essere congrui modulo u.

L'osservazione 2.4.2 afferma che per la successione T_{i_1} al k-esimo passo esiste d_{k,i_1} che rappresenta la distanza percorsa totale fino a quel punto, ovvero tale da essere la somma di tutte le distanze dei salti affettuati. Analogamente si può definire d_{r,i_2} per T_{i_2} all'r-esimo passo. Supponiamo fissati k ed r. Ricordando che le lunghezze dei salti (ovvero gli elementi dell'insieme D) sono tutte definite come multipli di uv è possibile definire $d_{k,i_1} = d_1 uv$, $d_{r,i_2} = d_2 uv$. Allora

$$\frac{a+b}{2} + i_1 v + d_1 uv \equiv \frac{a+b}{2} + i_2 v + d_2 uv \bmod uv \text{ cioè } i_1 \equiv i_2 \bmod u,$$

dove si sono utilizzati i risultati precedenti. Essendo per definizione i valori iniziali delle successioni T_i non congrui modulo uv ne segue che non possono avvenire collisioni tra T_{i_1} e T_{i_2} se $i_1 \neq i_2$.

Con analoghe argomentazioni si dimostra che non possono avvenire collisioni tra due successioni W_{j_1} e W_{j_2} con $j_1 \neq j_2$. □

Proposizione 2.4.9. *Esiste ed è unica una coppia (i, j) con $0 \leq i < u$ e $0 \leq j < v$ che verifica la congruenza*

$$\frac{a+b}{2} + iv \equiv x + ju \bmod uv.$$

Dimostrazione. Definiamo $\alpha = \frac{a+b}{2} - x$ e consideriamo $iv - ju \equiv \alpha \bmod uv$. Essendo $\mathrm{MCD}(u, v) = 1$, utilizzando l'identità di Bézout è possibile trovare due interi r, s non entrambi nulli tali che $sv + ru = 1$. Ne segue che tali interi soddisferanno

$$(\alpha s)v + (\alpha r)u = \alpha.$$

Così fissando $i_0 = \alpha s$ e $j_0 = \alpha r$ si ha $i_0 v - j_0 u = \alpha$. In particolare i_0 e j_0 sono soluzioni della congruenza $i_0 v - j_0 u \equiv \alpha \bmod uv$ ma non sono necessariamente tali che $0 \leq i_0 < u$ e $0 \leq j_0 < v$. Si può però osservare che se i_0 e j_0 sono soluzioni lo sono anche $i = i_0 + ku$ e $j = j_0 + hv$ per ogni $k, h \in \mathbb{Z}$. Così è possibile scegliere k e h in modo da avere $0 \leq i < u$ e $0 \leq j < v$.

Supponiamo per assurdo l'esistenza di due soluzioni $(i_1, j_1) \neq (i_2, j_2)$ con le condizioni enunciate sopra. Allora:

$$k + i_1 v - j_1 u \equiv 0 \bmod uv, \qquad k + i_2 v - j_2 u \equiv 0 \bmod uv$$

dalle quali segue

$$u(j_1 - j_2) + v(i_1 - i_2) \equiv 0 \bmod uv.$$

Essendo $\mathrm{MCD}(u, v) = 1$ si ha che $v|(j_1 - j_2)$ e $u|(i_1 - i_2)$ ovvero $i_1 \equiv i_2 \bmod u$ mentre $j_1 \equiv j_2 \bmod v$. Essendo $i_1, i_2 \in \{0, \ldots, u-1\}$ allora $i_1 = i_2$. Analogamente si ha $j_1 = j_2$. Ma per ipotesi $(i_1, j_1) \neq (i_2, j_2)$: assurdo. Quindi la soluzione è unica. □

Segue dalle precedenti proposizioni che esistono esattamente due successioni, T_i e W_j, che possono dar luogo ad una collisione. Una stima della complessità è data in [5]:

Teorema 2.4.10. *Una collisione sarà rilevata mediamente dopo* $O(\frac{\sqrt{b-a}}{u+v})$ *salti calcolati.*

Osservazione 2.4.11. Il metodo parallelo di Pollard funziona sotto ipotesi abbastanza restrittive. Innanzitutto si suppone noto fin dal principio il numero esatto dei calcolatori in gioco in modo da poter scegliere coerentemente u e v. Inoltre, i calcolatori devono essere tutti attivi fino al termine dell'algoritmo. Infatti, se un calcolatore dovesse interrompere l'esecuzione dell'algoritmo, esistendo solo un canguro di ogni mandria su ogni residuo, l'algoritmo potrebbe non terminare mai. Nella pratica si utilizzano quindi delle varianti del metodo parallelo di Pollard che rendono l'algoritmo più complesso da analizzare ma più versatile.

2.5 Metodo di Pohlig-Hellman

Pohlig ed Hellman osservarono nel 1978 che se è noto l'ordine n del gruppo ciclico G e la sua fattorizzazione in primi allora è possibile velocizzare il calcolo del logaritmo discreto riconducendosi al calcolo di logaritmi discreti in gruppi di ordine primo. Nelle prossime sezioni analizzeremo il funzionamento del metodo nei dettagli.

2.5.1 Riduzione a potenze di primi

Supponiamo che G sia un gruppo di ordine n generato da g. Sia nota la fattorizzazione

$$n = \prod_{p|n} p^{e(p)}$$

Definiamo per comodità le seguenti notazioni per ogni primo p che divide n:

$$n_p = \frac{n}{p^{e(p)}}, \qquad g_p = g^{n_p}, \qquad y_p = y^{n_p}. \tag{2.6}$$

Proposizione 2.5.1. *Siano dati $m_1, \ldots, m_r$ a due a due coprimi e siano definiti $M = m_1 \cdot \ldots \cdot m_r$ e $M_i = \frac{M}{m_i}$. Allora $MCD(M_1, \ldots, M_r) = 1$.*

Omettiamo la dimostrazione della proposizione 2.5.1 (può essere trovata in [7]). Utilizzeremo questa proposizione nella dimostrazione del seguente teorema.

Teorema 2.5.2. *Per ogni p primo divisore di n sia $x(p) = \log_{g_p} y^p$. Sia poi z la soluzione del seguente sistema di congruenze:*

$$z \equiv x(p) \bmod p^{e(p)}, \text{ per ogni } p \text{ che divide } n. \tag{2.7}$$

Allora $z = \log_g y$.

Dimostrazione. Definiamo $x = \log_g y$ e mostriamo che se z è la soluzione del sistema di congruenze 2.7 si ha $x \equiv z \bmod n$. Per il teorema 1.1.1 questo è sufficiente per completare la dimostrazione. Osserviamo inoltre che per il teorema cinese del resto la soluzione z esiste ed è unica modulo n.

Possiamo osservare che l'ordine di g_p è $p^{e(p)}$ quindi il logaritmo $x(p) = \log_{g_p} y_p$ è unico modulo $p^{e(p)}$. Questo è vero per ogni primo p che divide n. Per definizione, $z \equiv x(p) \bmod p^{e(p)}$ per tali primi quindi in particolare si avrà $z = \log_{g_p} y_p$, ovvero $y_p = g_p^z$. Questa relazione può essere riscritta come

$$y_p = g_p^z \text{ ovvero } y^{n_p} = g^{n_p z} \text{ cioè } (yg^{-z})^{n_p} = 1$$

ottenendo che l'ordine b di yg^{-z} deve essere un divisore di n_p per ogni primo p che divide n. Questo significa che b deve dividere il massimo comun divisore di tutti gli n_p ma, applicando la proposizione 2.5.1, si trova che tale massimo comun divisore è 1. Ne segue che l'ordine è $b = 1$ ovvero $yg^{-z} = 1$. Questo ci permette di concludere in quanto $z = \log_g y = x$. □

Osservazione 2.5.3. Possiamo trovare il logaritmo x applicando il *teorema cinese del resto*. La risoluzione del sistema 2.7 con il metodo indotto da questo teorema ha un costo computazionale trascurabile rispetto al calcolo dei logaritmi. Abbiamo visto nel teorema 2.2.5 che l'algoritmo baby-step giant-step ha complessità computazionale $O(\sqrt{p^{e(p)}})$ per calcolare $x(p)$ nel gruppo di ordine $p^{e(p)}$. Ne segue che il metodo introdotto risulta conveniente nel caso in cui n presenta nella sua fattorizzazione più primi distinti.

2.5.2 Riduzione ad ordini primi

Nella precedente sezione abbiamo ridotto il calcolo in un gruppo ciclico G al calcolo in sottogruppi aventi ordine una potenza di un primo. Vediamo ora che il calcolo in un gruppo ciclico avente come ordine una potenza di un primo può ridursi al calcolo in un sottogruppo avente ordine primo.

Teorema 2.5.4. *Sia $b \in \mathbb{N}^\star$. Per ogni $a \in \mathbb{N}^\star$ sono univocamente determinati $k \in \mathbb{N}^\star$ ed una successione $(a_0, \ldots, a_{k-1})$ con $a_i \in \{0, \ldots, b-1\}$, $a_{k-1} \neq 0$, tali che*

$$a = \sum_{i=1}^{k} a_{k-i} b^{k-i}. \tag{2.8}$$

Osservazione 2.5.5. Il teorema 2.5.4 garantisce l'esistenza e l'unicità della rappresentazione di un dato numero in base b. La dimostrazione di tale teorema può essere trovata in [2].

Consideriamo un gruppo ciclico G di ordine p^e con p primo ed $e \in \mathbb{N}^\star$. Per risolvere $x = \log_g y$ non è restrittivo supporre $x < p^e$. Per il teorema 2.5.4 possiamo esprimere

$$x = x_0 + x_1 p + x_2 p^2 + \ldots + x_{e-1} p^{e-1} \tag{2.9}$$

dove $x_i \in \{0, \ldots, p-1\}$ per ogni $i \in \{0, \ldots, e-1\}$. Mostriamo che gli x_i sono logaritmi discreti in un gruppo ciclico di ordine p. Eleviamo all'esponente p^{e-1} i membri dell'equazione $y = g^x$ ottenendo

$$y^{p^{e-1}} = g^{p^{e-1}x}. \tag{2.10}$$

Sostituiamo x con la rappresentazione espressa in 2.9. Si ottiene la seguente congruenza:

$$p^{e-1}x \equiv x_0 p^{e-1} + p^{e-1}(x_1 p + \ldots + x_{e-1}p^{e-1}) \equiv x_0 p^{e-1} \bmod p^e. \qquad (2.11)$$

Da 2.10 e 2.11 segue che $y^{p^{e-1}} = g^{p^{e-1}x_0}$. Così x_0 è un logaritmo discreto in un gruppo di ordine p in quanto $g^{p^{e-1}}$ ha ordine p.

Gli altri x_i sono determinati ricorsivamente. Supponiamo di aver determinato $x_0, \ldots, x_{h-1}$. Allora

$$g^{x_h p^h + \ldots + x_{e-1}p^{e-1}} = y \cdot g^{-(x_0 + \ldots + x_{h-1}p^{h-1})}$$

Elevando l'equazione all'esponente p^{e-h-1} (e denotando per comodità il membro di destra con α_h) si ottiene

$$g^{p^{e-1}x_h} = \alpha_h^{p^{e-h-1}}$$

che è un problema di calcolo del logaritmo discreto in un gruppo ciclico di ordine p con soluzione x_h. Quindi, per quanto detto, calcolare $x(p)$ richiede di risolvere e problemi di logaritmo discreto in un gruppo di ordine p.

2.5.3 Algoritmo di Pohlig-Hellman

Riassumendo, con il metodo di Pohlig-Hellman si calcolano innanzitutto gli elementi $g_p = g^{n_p}$ ed $y_p = y^{n_p}$ del gruppo per ogni primo p che divide l'ordine n del gruppo G. Vengono poi calcolati i coefficienti $x_i(p)$ per tutti i primi p divisori di n e per $i = 0, \ldots, e(p) - 1$. Viene infine utilizzato il teorema cinese del resto per determinare il logaritmo discreto.

Teorema 2.5.6. *Il metodo di Pohlig-Hellman risolve il problema del logaritmo discreto in un gruppo ciclico G di ordine n utilizzando*

$$O\Big(\sum_{p|n}(e(p)(\sqrt{p} + \log n))\Big)$$

operazioni di gruppo.

Dimostrazione. Il calcolo di g_p ed y_p richiede $O(\log n)$ operazioni di gruppo per ogni p. Supponiamo di utilizzare l'algoritmo baby-step giant-step per calcolare i coefficienti $x_i(p)$, algoritmo che sappiamo dal teorema 2.2.5 avere una complessità $O(\sqrt{p})$. I coefficienti vanno calcolati $e(p)$ volte.

Il teorema cinese del resto può essere implementato con $O(\nu(n)\log n)$ operazioni dove $\nu(n)$ è il numero dei primi della fattorizzazione di n. Così le operazioni da eseguire sono

$$O\Big(\sum_{p|n}(e(p)(\sqrt{p}+\log n))\Big)+O(\nu(n)\log n)\leq 2O\Big(\sum_{p|n}(e(p)(\sqrt{p}+\log n))\Big).$$

Notiamo però che il fattore 2 che moltiplica il termine di destra è una costante e può quindi essere trascurato nella notazione O-grande. Con ciò si ottiene la complessità enunciata. □

Per ulteriori dettagli sul calcolo della complessità del metodo di Pohlig-Hellman e del teorema cinese del resto si può fare riferimento a [8].

Osservazione 2.5.7. Il metodo illustrato da Pohlig ed Hellman mette in risalto l'importanza non solo della grandezza dell'ordine del gruppo ma anche della grandezza dei suoi fattori primi nella fattorizzazione standard. Infatti, se i primi sono tutti abbastanza piccoli, a prescindere dall'esponente che presentano nella fattorizzazione, è relativamente semplice risolvere il problema del logaritmo discreto. Quindi, nelle applicazioni in crittografia, la sicurezza potrebbe non essere garantita scegliendo un ordine con piccoli primi nella fattorizzazione.

Esempio 2.5.8. Siano $n = 390769$ ed $m = 458752$. Le fattorizzazioni saranno rispettivamente $n = 53\cdot 73\cdot 101$ ed $m = 2^{16}\cdot 7$. Allora è preferibile scegliere un gruppo di ordine n piuttosto che di ordine m nonostante sia $n < m$ perchè non ha fattori piccoli.

Esempio 2.5.9. Risolviamo il problema presentato nell'esempio 2.2.3 applicando il metodo di Pohlig-Hellman. In questo caso $n = 16 = 2^4$ dunque non è necessaria la prima parte del metodo in quanto siamo già ridotti ad ordini potenze di primi. Per riprendere le notazioni della teoria, abbiamo

$$G = \mathbb{Z}_{17}^{\star}, \quad p = 2, \quad e = 4, \quad g = [3]_{17}, \quad y = [5]_{17}.$$

Vogliamo trovare $x = \log_g y$. Esprimiamo x come in 2.9:

$$x = x_0 + 2x_1 + 4x_2 + 8x_3.$$

La prima equazione da risolvere è quella della forma $y^{p^{e-1}} = g^{p^{e-1}x_0}$ che in questo caso è:

$$5^8 \equiv 3^{8x_0} \bmod 17$$

Questo è un problema di logaritmo discreto dove $[3^8]_{17}$ ha ordine 2. In generale questi problemi si risolvono utilizzando i metodi illustrati in questo capitolo ma essendo l'ordine 2 possiamo notare che $x_0 \in \{0, 1\}$ e $5^8 \equiv 16 \bmod 17$, quindi $x_0 = 1$ (infatti $3^8 \equiv 16 \bmod 17$).

Ora dobbiamo trovare x_1 basandoci sulla conoscenza di x_0. Per farlo è necessario risolvere

$$y^{p^{e-1-1}} \cdot g^{-p^{e-1-1}x_0} = g^{8x_1} \text{ ovvero } (5 \cdot 3^{-1})^4 \equiv 3^{8x_1} \bmod 17.$$

Notiamo innanzitutto che $3^{-1} \equiv 6 \bmod 17$. Utilizzando ciò troviamo che $3^{8x_1} \equiv 1 \bmod 17$ e quindi, per quanto detto in precedenza, dev'essere $x_1 = 0$. Allo stesso modo si trovano:

$$y^{p^{e-2-1}} \cdot g^{-p^{e-2-1}(x_0+2x_1)} = g^{8x_2} \text{ ovvero } (5 \cdot 3^{-1})^2 \equiv 3^{8x_2} \bmod 17 \Rightarrow x_2 = 1$$

$$y^{p^{e-3-1}} \cdot g^{-p^{e-3-1}(x_0+2x_1+4x_2)} = g^{8x_3} \text{ ovvero } 5 \cdot 3^{-5} \equiv 3^{8x_3} \bmod 17 \Rightarrow x_3 = 0$$

Allora $x = \log_3 5$ è calcolabile da

$$x = x_0 + 2x_1 + 4x_2 + 8x_3 = 1 + 4 \cdot 1 = 5$$

che, come visto nell'esempio 2.2.3, è proprio la soluzione cercata. In particolare la rappresentazione di x è data da $k = 3$ e dalla successione $(1, 0, 1)$.

Capitolo 3

Metodo di calcolo dell'indice

Nel capitolo precedente i metodi sono sempre stati illustrati considerando un generico gruppo ciclico G. Nel caso del metodo di calcolo dell'indice, invece, si richiede che gli elementi del gruppo siano fattorizzabili con una fattorizzazione unica quindi tale metodo è presentato supponendo di scegliere $G = \mathbb{Z}_p^\star$ con p primo. D'ora in poi identificheremo una generica classe di resto $m \in \mathbb{Z}_p$ con il suo più piccolo rappresentante intero non negativo.

Ricordiamo le proprietà 1.2 e 1.3 del logaritmo discreto. Queste possono essere utilizzate per introdurre un nuovo metodo di risoluzione del problema $x = \log_g y$. Se fosse nota la fattorizzazione

$$y = \prod_{i=1}^{h} p_i^{e_i}$$

di y, applicando le proprietà del logaritmo discreto la soluzione potrebbe essere trovata da

$$x = \log_g \Big(\prod_{i=1}^{h} p_i^{e_i} \Big) = \sum_{i=1}^{h} (e_i \log_g p_i). \tag{3.1}$$

Quindi sarebbe sufficiente trovare $\log_g p_i$ per tutti i primi p_i per poter risolvere il problema. Il vantaggio di questo ragionamento è duplice. Innanzitutto, essendo p_i nella fattorizzazione di y, in generale sarà $p_i \leq y$ quindi il calcolo di $\log_g p_i$ potrebbe coinvolge numeri più piccoli rispetto al calcolo di $\log_g y$, ma soprattutto i $\log_g p_i$ calcolati possono essere utilizzati per risolvere più problemi del logaritmo discreto contemporaneamente. Infatti se y_1 e y_2 hanno entrambi un certo fattore primo p nella loro fattorizzazione, allora è sufficiente calcolare $\log_g p$ una sola volta ed utilizzare il risultato in entrambi i problemi.

Utilizzando infine la proprietà 1.4 di cambio base del logaritmo, i logaritmi dei primi calcolati potranno anche essere riutilizzati per risolvere problemi con basi differenti.

Le seguenti sezioni descriveranno nel dettaglio le tre fasi del metodo di calcolo dell'indice. Le tre fasi possono essere riassunte in questo modo:

1. Si genera una base di fattori e si costruiscono delle relazioni del tipo 3.1.

2. Una volta trovate abbastanza relazioni si risolve il sistema lineare che queste generano. Tale sistema lineare avrà come incognite i logaritmi dei primi della base di fattori fissata. Con questa fase termina quella che viene detta *fase di inizializzazione*.

3. Trovati i logaritmi dei primi della fattorizzazione di y si utilizzano le proprietà del logaritmo discreto per risolvere $x = \log_g y$ come mostrato in 3.1.

3.1 Basi di fattori e B-numeri

Sia $B \geq 2$ un intero fissato. Si definisce la *base di fattori* $F(B)$ come segue:

$$F(B) = \{p \text{ primi} \mid p \leq B\}.$$

Un numero fattorizzabile in $F(B)$, ovvero tale che presenti nella sua fattorizzazione solo primi $p \in F(B)$, si dice *B-numero*. Spesso ci si riferisce ai B-numeri anche come *primi privi di grandi fattori primi*. Se

$$a = p_1^{e_1} \cdot \ldots \cdot p_{\pi(B)}^{e_{\pi(B)}}$$

è la fattorizzazione standard del B-numero a, allora il vettore degli esponenti $(e_1, \ldots, e_{\pi(B)})$ è detto *B-vettore di a*. L'unicità della fattorizzazione standard ci garantisce l'unicità del B-vettore per ogni B-numero. Osserviamo che la base di fattori $F(B)$ contiene esattamente $\pi(B)$ primi dove $\pi : \mathbb{N} \to \mathbb{N}$ è la funzione aritmetica che ad m associa il numero dei primi minori o uguali ad m.

Il problema di scoprire se un dato numero è o non è un B-numero è un problema particolare del più generico problema della fattorizzazione. A differenza di quest'ultimo, però, essendo relativamente piccolo l'insieme dei possibili primi, il problema risulta più facile.

Esistono diversi algoritmi per scoprire se un dato n è un B-numero. Per il metodo di calcolo dell'indice che illustreremo è sufficiente l'utilizzo del crivello seguente.

3.1.1 Crivello di polinomi

Dato un polinomio f ed una base di fattori $F(B)$, il crivello di polinomi è un metodo per determinare tutti i valori di b in un dato intervallo $c \leq b < d$ tali che $f(b)$ sia un B-numero. Il metodo di calcolo dell'indice richiede l'utilizzo di questo crivello solo su polinomi di primo grado ma in questa sezione introdurremo la versione generale.

Proposizione 3.1.1. *Sia $f(x) = a_n x^n + \ldots + a_1 x + a_0$ un polinomio di grado n e sia p primo. Allora, se $f(b)$ è divisibile per p^s, lo è anche $f(b + p^s)$ per ogni $s \geq 1$.*

Dimostrazione. Si ha $f(b+p^s) = a_n(b+p^s)^n + \ldots + a_1(b+p^s) + a_0$. Per vedere che è un multiplo di p^s basta provare che $f(b + p^s) \equiv 0 \bmod p^s$. Utilizzando lo sviluppo del binomio di Newton si nota che

$$f(b + p^s) = a_n \sum_{i=0}^{n} \binom{n}{i} b^{n-i} p^{si} + \ldots + a_1(b + p^s) + a_0.$$

Sviluppando le somme, ogni volta che l'indice è $i > 0$ si ottiene un termine moltiplicato per p^s o per una sua potenza. Così, riducendo modulo p^s, si trovano solo i termini delle somme calcolati per $i = 0$, ovvero

$$f(b + p^s) \equiv a_n b^n + \ldots + a_1 b + a_0 \bmod p^s.$$

Così $f(b + p^s) \equiv f(b) \equiv 0 \bmod p^s$ quindi la proposizione risulta dimostrata. □

Questa proposizione sarà utilizzata per velocizzare il crivello. Infatti se viene trovato un p tale che $f(b)$ risulta essere suo multiplo, allora anche $f(b + kp^s)$ con $k \in \mathbb{N}$ lo è.

Presentiamo ora il crivello di polinomi. Consideriamo il polinomio f su $\mathbb{Z}_p[x]$. Se non è il polinomio nullo, i suoi zeri sono al più n e questi sono proprio quei valori b per cui $f(b) \equiv 0 \bmod p$, ovvero i valori b tali che $f(b)$ sia multiplo di p. Ricordiamo che b deve appartenere all'intervallo $c \leq b < d$.

1. Si definiscono $l_x = 0$ per ogni x dell'intervallo $c \leq x < d$. Questi numeri l_x serviranno per memorizzare per quali primi p è divisibile $f(x)$. In particolare, ogni volta che troviamo un x per cui p divide $f(x)$, sommeremo $\log p$ ad l_x dove log è in questo caso il logaritmo naturale definito sui reali.

2. Fissiamo ora un primo $p \in F(B)$. Si riduce il polinomio f modulo p denotando il risultato con f_p quindi si cercano gli zeri. Sia Z_p l'insieme degli zeri di f_p.

3. Se f_p risulta essere il polinomio nullo, significa che è multiplo di p per ogni valore di x. Aggiungiamo allora $\log p$ a l_x per ogni x dell'intervallo. Notiamo che in questo caso Z_p contiene tutti gli elementi dell'intervallo (ovvero ogni elemento può essere visto come uno zero di f_p perchè questo è il polinomio nullo). Saltiamo al punto 5.

4. Se f_p non è il polinomio nullo, fissiamo uno $z \in Z_p$ e definiamo $x_p(z)$ come il più piccolo residuo non negativo congruo a z modulo p. Così $f_p(x_p(z)) \equiv 0 \bmod p$, ovvero $f_p(x_p(z))$ è divisibile per p. Per la proposizione 3.1.1 anche $f_p(x_p(z)+kp)$ è divisibile per p. Consideriamo tutti i valori di $k \in \mathbb{N}$ tali da avere $x_p(z)+kp < d$. Allora aggiungiamo $\log p$ a $l_{x_p(z)+kp}$ per tali k. Ripetiamo questo procedimento per ogni $z \in Z_p$.

5. Arrivati a questo punto sappiamo per quali valori di x si ha che $f(x)$ è un multiplo di p. Controlliamo allora se questo vale anche per $p^2, p^3, \ldots$ ovvero cerchiamo il massimo esponente e_x tale che p^{e_x} divida $f(x)$. Questo controllo può essere fatto in maniera ricorsiva.

   Fissiamo $z \in Z_p$. Supponiamo che $f(z)$ sia divisibile per p^{e-1} e verifichiamo se è divisibile anche per p^e. Questo procedimento è corretto in quanto, dai punti precedenti, conosciamo già la divisibilità per $e = 1$.

   Sapendo che $f(z)$ divisibile per p^{e-1}, per la proposizione 3.1.1 lo è anche $f(z + kp^{e-1})$ per k intero. Ha quindi senso definire

   $$f_{p^e}(k) \equiv \frac{f(z + kp^{e-1})}{p^{e-1}} \bmod p.$$

   Sia W_{p^e} l'insieme degli interi non negativi minori di p che sono zeri di $f_{p^e}(k)$. Definiamo l'insieme

   $$Z_{p^e} = \{z' | z' \equiv z + kp^{e-1} \bmod p \text{ con } k \in W_{p^e}\}.$$

   Per ogni $z' \in Z_{p^e}$ definiamo $x_p(z')$ come il più piccolo intero non negativo congruo a z' modulo p. Analogamente a quanto fatto nel passo 4 aggiungiamo $\log p$ a $l_{x_p(z')+kp^e}$ per ogni $k \in \mathbb{N}$ tale che $x_p(z') + kp^e < d$.

   Questo passo va ripetuto finchè p^e divide $f(z)$: in questo modo troviamo il massimo esponente di p per cui ciò avviene per ogni x dell'intervallo. Terminato il procedimento per lo zero $z \in Z_p$ fissato, si sceglie un'altro zero e si ripete il procedimento fino ad esaurire tutti gli elementi di Z_p.

Dopo aver eseguito tali passaggi per ogni $p \in F(B)$ si otterranno, per ogni x nell'intervallo $c \leq x < d$, dei numeri della forma

$$l_x = e_1 \log p_1 + \ldots + e_{\pi(B)} \log p_{\pi(B)}. \tag{3.2}$$

La conoscenza di l_x è equivalente alla conoscenza di $e^{l_x} = p_1^{e_1} \cdot \ldots \cdot p_{\pi(B)}^{e_{\pi(B)}}$. Nella pratica si preferisce utilizzare la somma di logaritmi perchè la somma è un'operazione computazionalmente meno costosa del prodotto.

Teorema 3.1.2. *Sia l_x risultato del crivello di polinomi come espresso in 3.2. Allora $f(x)$ è un B-numero se e soltanto se $l_x = \log f(x)$.*

Dimostrazione. Sia $f(x)$ un B-numero. Allora il crivello lo riconosce come tale in quanto ne trova l'esatta fattorizzazione tramite i primi di $F(B)$. Supponiamo che la fattorizzazione sia $f(x) = p_1^{e_1} \cdot \ldots \cdot p_{\pi(B)}^{e_{\pi(B)}}$. Allora, utilizzando le proprietà 1.2 e 1.3 si ha

$$l_x = e_1 \log p_1 + \ldots + e_{\pi(B)} \log p_{\pi(B)} = \log(p_1^{e_1} \cdot \ldots \cdot p_{\pi(B)}^{e_{\pi(B)}}) = \log f(x).$$

Viceversa, sia $l_x = \log_g f(x)$. Se per assurdo $f(x)$ non fosse B-numero, allora sarebbe del tipo

$$f(x) = \Big(\prod_{j=1}^{h} q_j^{s_j}\Big)\Big(\prod_{i=1}^{\pi(B)} p_i^{e_i}\Big)$$

con $q_j \notin F(B)$ per ogni $1 \leq i \leq h$. Così l'algoritmo precedentemente presentato non testa q_1 (non viene testato nessun q_i quindi in particolar modo non è testato q_1) quindi il suo logaritmo discreto non può apparire in l_x. Allora

$$l_x = \sum_{i=1}^{\pi(B)} e_i \log p_i < s_1 \log q_1 + \sum_{i=1}^{\pi(B)} e_i \log p_i \leq \log f(x)$$

contro l'ipotesi. Quindi $f(x)$ deve essere un B-numero. □

Si dice *lunghezza del crivello* la lunghezza dell'intervallo dei valori che vengono testati. Utilizzando le notazioni precedenti si ha $c \leq x < d$ quindi la lunghezza del crivello è $d - c$. Schirokauer [6] da una stima della complessità computazionale del crivello di polinomi con il seguente teorema.

Teorema 3.1.3. *La complessità computazionale del crivello di polinomi è*

$$\pi(B)(n + \log B)^{O(1)} + O(l \log \log B)$$

dove $\pi(B)$ è il numero di primi della base di fattori, l è la lunghezza del crivello ed n è il grado del polinomio f.

3.2 Prima fase: ricerca delle relazioni

Abbiamo già detto che il metodo del calcolo dell'indice richiede di trovare relazioni del tipo 3.1 che più precisamente, supponendo di essere in $\mathbb{Z}_p^\star$, sono

$$u \equiv e_1 \log_g p_1 + \ldots + e_{\pi(B)} \log_g p_{\pi(B)} \bmod p-1 \tag{3.3}$$

dove la base di fattori scelta è $F(B) = \{p_1, \ldots, p_{\pi(B)}\}$. Nel seguito, quando diremo che u è un residuo modulo m considereremo sempre il più piccolo intero non negativo della classe $[u]_m$, ovvero $0 \leq u < m$.

Il metodo più semplice per trovare questo tipo di relazioni è quello di scegliere ad arbitrio un u residuo modulo $p-1$. Scelto u si calcola r come

$$r \equiv g^u \bmod p.$$

Se r risulta essere un B-numero avremo trovato una relazione utile. Così, scegliendo casualmente il residuo u si può ripetere il procedimento per cercare le relazioni necessarie. Questa procedura non è però efficiente.

Un metodo migliore consiste nel fissare $H = \lceil \sqrt{p} \rceil$ e due residui non negativi modulo p abbastanza piccoli λ, μ. Si può notare che $H^2 = p + J$ dove $J = \lceil \sqrt{p} \rceil^2 - p$, quindi è generalmente piccolo. Calcoliamo

$$(H+\lambda)(H+\mu) \equiv J + (\lambda+\mu)H + \lambda\mu \bmod p. \tag{3.4}$$

Essendo J, λ, μ piccoli per definizione, il fattore dominante del membro di destra in 3.4 è H. Quindi tale residuo è $O(\sqrt{p})$, ovvero è dell'ordine di grandezza di $\sqrt{p}$. Supponiamo allora di fissare λ. In questo modo troviamo

$$f_\lambda(\mu) = (H+\lambda)\mu + J + \lambda H$$

che è un polinomio di primo grado nell'indeterminata μ. Possiamo allora utilizzare il crivello di polinomi per cercare i valori di μ per cui $f_\lambda(\mu)$ risulta essere un B-numero. Così $\log_g f_\lambda(\mu)$ è un residuo modulo $p-1$ della forma 3.3. In particolare utilizzando le proprietà del logaritmo discreto la relazione trovata è

$$\log_g(H+\lambda) + \log_g(H+\mu) \equiv e_1 \log_g p_1 + \ldots + e_{\pi(B)} \log_g p_{\pi(B)} \bmod p-1. \tag{3.5}$$

La prima fase si può schematizzare nelle seguenti operazioni:

1. Si fissa B e si calcolano gli elementi della base di fattori $F(B)$. Questo può essere fatto ad esempio utilizzando il crivello di Eratostene.

2. Per ogni $\lambda > 0$ abbastanza piccolo si applica il crivello di polinomi a $f_\lambda(\mu)$ con la condizione $\mu \geq \lambda$. Questa condizione serve ad evitare il calcolo di relazioni inutili. Infatti se si trova una relazione per una fissata coppia (λ, μ), allora la stessa relazione sarà trovata per (μ, λ).

3. Utilizziamo le relazioni trovate per costruire un sistema lineare nell'anello $\mathbb{Z}_p$ nelle incognite $\log_g p_i$.

Osservazione 3.2.1. Definiamo i seguenti insiemi:

$$R = \{(\lambda, \mu)| \text{ è stata trovata una relazione per } \lambda \text{ e } \mu\}$$

$$M = \{\mu|\ \exists \lambda \in L : (\lambda, \mu) \in A,\ \ 0 \leq \mu < p-1\}$$

$$L = \{\lambda|\ \exists \lambda \in M : (\lambda, \mu) \in A,\ \ 0 \leq \lambda < p-1\}$$

Utilizzando il metodo del polinomio precedentemente descritto per cercare le relazioni necessarie, in realtà introduciamo delle nuove incognite ovvero $\log_g(H+\beta)$ con $\beta \in M \cup L$. Quindi al crescere del numero di relazioni trovate cresce anche il numero di incognite del sistema. Notiamo però che, fissato $\lambda \in L$, esistono in generale diversi $\mu \in M$ tali che $(\lambda, \mu) \in R$. Analogamente, dato $\mu \in M$ esisteranno diversi $\lambda \in L$ tali che $(\lambda, \mu) \in R$. Quindi in realtà il numero delle relazioni cresce più velocemente del numero delle incognite.

3.3 Seconda fase: risoluzione del sistema

Dalla prima fase si ottengono relazioni del tipo 3.3 nelle incognite $\log_g p_i$ con $p_i \in F(B)$ e $\log_g(H+\beta)$ con $\beta \in M \cup L$. Sia n il numero delle incognite. Denoteremo successivamente con $\log_g x_i$, $1 \leq i \leq n$ la i-esima incognita senza curarci del fatto che sia del tipo $\log_g p_i$ o $\log_g(H+\beta)$.

Osservazione 3.3.1. Se tutte le relazioni sono state trovate con il metodo del polinomio $f_\lambda(\mu)$, il sistema lineare trovato è in particolare omogeneo nelle incognite $\log_g p_i$, $\log_g(H+\beta)$ in quanto ogni relazione è del tipo 3.5. Questo caso non è però trattabile col metodo di Lanczos che andremo ad introdurre (sezione 3.3.2). Tratteremo comunque il problema da un punto di vista generale, ovvero supporremo il sistema non omogeneo. Nella pratica ci si può ricondurre ad un sistema non omogeneo calcolando alcune relazioni con il metodo per tentativi descritto nella sezione 3.2 (in questo caso nelle equazioni c'è un termine noto non nullo) e le altre con il metodo del polinomio $f_\lambda(\mu)$. Un procedimento di questo tipo non aumenta il numero di variabili in quanto, con il metodo per tentativi, le relazioni trovate coinvolgono solo le variabili $\log_g p_i$ quindi è lecito farlo senza ripercussioni sulla soluzione.

Supponiamo che la i-esima relazione sia della forma

$$u_i \equiv e_1(u_i) \log_g x_1 + \ldots + e_n(u_i) \log_g x_n \bmod q.$$

Così, se abbiamo m relazioni, la matrice associata al sistema lineare è del tipo

$$A = \begin{pmatrix} e_1(u_1) & \ldots & e_n(u_1) \\ \vdots & & \vdots \\ e_1(u_m) & \ldots & e_x(u_m) \end{pmatrix}$$

Se consideriamo la sottomatrice di A' di A composta dalle sole colonne relative alle ingognite $\log_g p_i$ possiamo notare che l'i-esima riga è il B-vettore di g^{u_i}. Ragionando per colonne, invece, abbiamo che la i-esima contiene come j-esimo elemento l'esponente di p_i nella fattorizzazione di g^{u_i}.

In generale, un B-numero non è multiplo di ogni primo della base di fattori quindi i B-vettori presenteranno molti elementi nulli (ovvero molti primi appariranno nella fattorizzazione con esponente $e_i = 0$). Una matrice avente pochi elementi non nulli è detta *matrice sparsa*, altrimenti si dice *matrice densa*.

Si può anche notare che le colonne relative ai primi più piccoli di $F(B)$ saranno più dense mentre quelle relative ai primi più grandi saranno sparse se non interamente nulle. In questo caso, quindi, la matrice non è *uniformemente sparsa*. Queste osservazioni possono essere generalizzate alla matrice A osservando che, fissato un β, l'incognita $\log_g(H+\beta)$ per costruzione appare solo in poche relazioni.

Ricordiamo che per poter trovare una soluzione unica al sistema lineare applicando il teorema di Rouchè-Capelli è necessario che il rango della matrice A sia massimo, ovvero n. Non è facile sapere a priori se la matrice che si sta costruendo soddisfa questa ipotesi quindi in generale si costruiscono molte più relazioni di quanto sarebbero necessarie, ovvero si considera m molto più grande di n.

Osservazione 3.3.2. I metodi che andremo ad illustrare funzionano modulo un primo q mentre i sistemi che provengono dalla prima fase del metodo del calcolo dell'indice sono modulo $p-1$ che non è un primo (se $p > 3$). Se è nota la fattorizzazione in primi di $p-1$ si possono applicare gli stessi metodi modulo tali primi (e loro potenze) per poi ricavare la soluzione modulo $p-1$ col teorema cinese del resto.

3.3.1 Eliminazione strutturata di Gauss

Il metodo che andremo a presentare è una variante del metodo di eliminazione di Gauss pensato per ottimizzare la riduzione nel caso di matrici

sparse. L'algoritmo di Gauss strutturato si basa proprio sulle osservazioni fatte sulla matrice 3.3: essendo vantaggioso dal punto di vista computazionale l'utilizzo di matrici sparse, questo algoritmo consiste in una serie di passaggi che tendono a conservare questa caratteristica. Invece, iniziando ad esempio l'applicazione dell'eliminazione di Gauss standard dalle colonne dense, la matrice tenderà velocemente a diventare densa a sua volta.

L'eliminazione di Gauss strutturata non è utilizzata per trovare la soluzione del sistema ma è estremamente utile per ridurre una grande matrice sparsa ad una matrice densa molto più piccola alla quale potranno poi essere applicate tecniche di risoluzione. Decidere quando interrompere questo metodo per utilizzare una tecnica di risoluzione è un importante parametro che influenza la complessità.

Il metodo può essere riassunto in quattro operazioni fondamentali:

1. Si cancellano tutte le colonne nulle della matrice. Questa operazione scarta quindi le colonne relative alle incognite del sistema lineare che non appaiono (ovvero che hanno coefficienti nulli in ogni equazione).

   Si procede cancellando le colonne che hanno un solo elemento non nullo e tutte le righe relative all'elemento non nullo delle colonne cancellate. Questo tipo di operazioni riducono la dimensione della matrice. L'operazione è lecita infatti, avendo la colonna eliminata un solo elemento non nullo, questo significa che l'incognita relativa a tale colonna appare in una sola equazione. Il suo valore sarà allora determinato una volta trovato il valore delle altre incognite presenti in tale equazione.

2. Si fissa un numero reale α con $0 < \alpha < 1$ e si scelgono αn colonne tra quelle aventi un maggior numero di elementi non nulli. Queste colonne vengono etichettate come *pesanti*, le rimanenti saranno invece le colonne *leggere*. Un valore tipico è $\alpha = \frac{1}{32}$. Successivamente chiameremo *peso* di una riga il numero di elementi non nulli di quella riga contenuti nelle colonne leggere.

3. Supponiamo che la j-esima riga abbia peso 1 e che l'elemento non nullo che gli conferisce tale peso sia l'i-esimo. Allora si sottraggono appropriati multipli della j-esima riga dalle righe che hanno l'i-esimo elemento non nullo. In questo modo l'i-esima incognita, relativa ad una colonna leggera, sarà rimasta solo nella j-esima equazione. Inoltre la j-esima equazione, avendo la relativa riga peso 1, presenta solo la i-esima incognita. In questo modo il valore di tale incognita è immediatamente determinabile.

Si ripete il procedimento per ogni riga di peso 1. Ripetere l'operazione non va ad influire sulle colonne precedentemente trattate in questo stesso passo e sulle righe di peso 1. Infatti sia la k-esima riga di peso 1 con $k \neq j$ e con elemento non nullo in h-esima posizione. Allora se fosse $i = h$, l'elemento non nullo della k-esima riga sarebbe stato annullato durante le operazioni sulla j-esima riga in quanto i due elementi non nulli apparterrebbero alla stessa colonna. Così $i \neq h$. Essendo l'unico elemento non nullo della j-esima riga in i-esima posizione, allora tale riga avrà uno 0 in h-esima posizione. Ne viene che non dev'essere sommato alcun multiplo della k-esima riga alla j-esima. Ciò dimostra che una riga di peso 1 non viene più modificata dopo che è stata eseguita l'operazione.

4. Se il sistema è fortemente sovradimensionato, ovvero se il numero delle equazioni è molto maggiore al numero delle incognite, è possibile eliminare dalla matrice alcune righe scelte tra quelle di peso maggiore.

Questi passaggi vengono ripetuti finchè non si ottiene una matrice dell'ordine desiderato (oppure finchè la matrice non è più abbastanza sparsa). Sono disponibili solo stime euristiche del vantaggio dato dall'utilizzo di questo metodo. Per i dettagli ed alcuni risultati sperimentali si possono vedere i lavori di Odlyzko [9], [10].

3.3.2 Metodo di Lanczos

Il metodo di Lanczos è un algoritmo iterativo che consente di trovare la soluzione di un sistema lineare della forma $Ax = b$. Nonostante questo metodo, nella sua versione originale, sia stato pensato per lavorare su $\mathbb{R}$, è possibile estenderlo ad un generico campo K. Dopo aver ridotto la matrice trovata nella prima fase col metodo dell'eliminazione di Gauss strutturata utilizzeremo il metodo di Lanczos per cercare la soluzione del sistema.

Premettiamo la definizione di *vettori A-ortogonali*: due vettori $u, v \in K^n$ con K campo generico si dicono A-ortogonali se $\langle u, Av \rangle = 0$ dove $\langle , \rangle$ indica il prodotto scalare standard.

Supponiamo che A sia una matrice $n \times n$ simmetrica definita positiva su $\mathbb{R}$. Se $b \in \mathbb{R}^n$, il metodo di Lanczos consente la risoluzione del sistema lineare $Ax = b$. Per farlo, si definiscono $w_0 = b$ ed iterativamente

$$w_i = Aw_{i-1} - \sum_{j=0}^{i-1} \frac{\langle w_j, A^2 w_{i-1} \rangle}{\langle w_j, Aw_j \rangle} w_j \text{ per } i \geq 1 \tag{3.6}$$

finchè non si trova un $w_i = \underline{0}$ dove $\underline{0}$ è il vettore nullo. La definizione 3.6 è costruita in modo da avere $\langle w_j, Aw_i \rangle = 0$ se $i \neq j$, ovvero i vettori così trovati sono a due a due A-ortogonali (per la dimostrazione si faccia riferimento alla proposizione 3.3.5).

Proposizione 3.3.3. *L'iterazione 3.6 che definisce i vettori ha termine.*

Dimostrazione. Essendo $w_i \in \mathbb{R}^n$, i vettori $w_0, \ldots, w_t$ sono linearmente dipendenti se $t > n$. Per definizione di dipendenza lineare, esisteranno quindi dei coefficienti $\lambda_j \in \mathbb{R}$ non tutti nulli tali che

$$\sum_{j=0}^{t} \lambda_j w_j = \underline{0}.$$

Supponiamo $\lambda_t \neq 0$. Moltiplicando per $w_t^T A$ entrambi i membri dell'equazione (e ricordando che w_i, w_j sono A-ortogonali se $i \neq j$) si ha l'annullamento di tutti i termini dell'addizione ad eccezione del t-esimo. Si ottiene quindi

$$\lambda_t w_t^T A w_t = 0 \Rightarrow w_t^T A w_t = 0 \Rightarrow \langle w_t, Aw_t \rangle = 0 \tag{3.7}$$

dove abbiamo utilizzato $\lambda_t \neq 0$. Per ipotesi A è definita positiva che, per definizione, significa che $\langle v, Av \rangle > 0$ per ogni vettore $v \neq \underline{0}$. Ma allora, l'equazione 3.7 è verificata se e solo se $w_t = \underline{0}$. Quindi considerando un numero di vettori maggiore ad n si trova sempre almeno un vettore nullo, ovvero l'iterazione termina al più in n passaggi. □

Osservazione 3.3.4. Essendo A simmetrica si ha che $A^T = A$. Così

$$\langle Av, Au \rangle = (Av)^T Au = v^T A^T Au = \langle v, A^2 u \rangle$$

per ogni coppia di vettori u, v.

Proposizione 3.3.5. *Due vettori w_i, w_j con $i \neq j$ definiti come in precedenza sono sempre A-ortogonali, ovvero $\langle w_i, Aw_j \rangle = 0$.*

Dimostrazione. Non è restrittivo considerare $i > j$. Sotto quest'ipotesi dimostriamo la proposizione per induzione su i.

Sia $i = 1$. Essendo $i > j$ dovrà essere $j = 0$. Allora utilizzando le proprietà di linearità del prodotto scalare:

$$\begin{aligned} \langle w_1, Aw_0 \rangle &= \langle Aw_0 - \frac{\langle w_0, A^2 w_0 \rangle}{\langle w_0, Aw_0 \rangle} w_0, Aw_0 \rangle = \\ &= \langle Aw_0, Aw_0 \rangle - \frac{\langle w_0, A^2 w_0 \rangle}{\langle w_0, Aw_0 \rangle} \langle w_0, Aw_0 \rangle = 0 \end{aligned}$$

dove si è usata l'osservazione 3.3.4. Questo dimostra la proposizione nel caso $i = 1$. Supponiamo ora che sia vera per ogni indice fino a $i - 1$ e mostriamo che vale anche per il caso i. Utilizzando la definizione ricorsiva dei vettori e la linearità si ha

$$\langle w_i, Aw_j \rangle = \langle Aw_{i-1}, Aw_j \rangle - \sum_{k=0}^{i-1} \frac{\langle w_k, A^2 w_{i-1} \rangle}{\langle w_k, A^2 w_k \rangle} \langle w_k, Aw_j \rangle.$$

I termini della sommatoria sono tutti moltiplicati per il termine $\langle w_k, Aw_j \rangle$ con $j < i - 1$ fissato e $0 \leq k \leq i - 1$. Per l'ipotesi induttiva, se $j \neq k$ questo termine è nullo. Quindi la somma si riduce al solo termine per $k = j$, ovvero

$$\langle w_i, Aw_j \rangle = \langle Aw_{i-1}, Aw_j \rangle - \frac{\langle w_j, A^2 w_{i-1} \rangle}{\langle w_j, Aw_j \rangle} \langle w_j, Aw_j \rangle = 0$$

che conclude la dimostrazione. □

Proposizione 3.3.6. *Sia* $m = \min\{i \geq 1 | w_i = \underline{0}\}$ *con* w_i *definito come in 3.6. Allora*

$$\overline{x} = \sum_{j=0}^{m-1} \frac{\langle w_j, b \rangle}{\langle w_j, Aw_j \rangle} w_j$$

è soluzione del sistema lineare $Ax = b$.

Dimostrazione. Sia $\mathcal{W} = \text{Span}(w_0, \ldots, w_{m-1})$. Per definizione, $\overline{x} \in \mathcal{W}$. Ne segue che

$$A\overline{x} - b \in \text{Span}(Aw_0, \ldots, Aw_{m-1}, b) \subseteq \mathcal{W}$$

per la definizione 3.6 dei vettori. Si ha che

$$\langle w_i, A\overline{x} \rangle = \langle w_i, A \sum_{j=0}^{m-1} \frac{\langle w_j, b \rangle}{\langle w_j, Aw_j \rangle} w_j \rangle = \langle w_i, \sum_{j=0}^{m-1} \frac{\langle w_j, b \rangle}{\langle w_j, Aw_j \rangle} Aw_j \rangle.$$

Applicando la linearità del prodotto scalare si ottiene

$$\langle w_i, \sum_{j=0}^{m-1} \frac{\langle w_j, b \rangle}{\langle w_j, Aw_j \rangle} Aw_j \rangle = \sum_{j=0}^{m-1} \frac{\langle w_j, b \rangle}{\langle w_j, Aw_j \rangle} \langle w_i, Aw_j \rangle.$$

Essendo i vettori costruiti in modo da essere a due a due A-ortogonali come visto nella proposizione 3.3.5, i termini della somma per $j \neq i$ sono tutti nulli. Quello che resta è quindi solo l'i-esimo termine, ovvero

$$\frac{\langle w_i, b \rangle}{\langle w_i, Aw_i \rangle} \langle w_i, Aw_i \rangle$$

che si può semplificare ottenendo la relazione $\langle w_i, A\overline{x}\rangle = \langle w_i, b\rangle$ per ogni $0 \leq i \leq m-1$. Utilizzando sempre la linearità del prodotto scalare si può riscrivere la relazione come

$$\langle w_i, A\overline{x} - b\rangle = 0 \text{ per ogni } 0 \leq i \leq m-1.$$

Avendo precedentemente notato che $A\overline{x} - b \in \mathcal{W}$ questo non può essere ortogonale a tutti i vettori di $\mathcal{W}$ a meno che non sia il vettore nullo. Allora $A\overline{x} = b$ ovvero $\overline{x}$ è la soluzione cercata del sistema lineare. □

Osservazione 3.3.7. Nella definizione 3.6 si nota che per il calcolo di w_i è necessario sommare, con appositi coefficienti, tutti i vettori precedenti. In realtà, però, tale calcolo viene semplificato. Infatti applicando la definizione stessa si ha che, per $j < i-2$,

$$\langle w_j, A^2 w_{i-1}\rangle = \langle Aw_j, Aw_{i-1}\rangle = \langle \Big(w_{j+1} + \sum_{k=0}^{j} \frac{\langle w_k, A^2 w_j\rangle}{\langle w_k, Aw_k\rangle} w_k \Big), Aw_{i-1}\rangle = 0$$

perchè, per costruzione, w_i e w_j sono A-ortogonali se $i \neq j$. Così la definizione 3.6 può essere semplificata con

$$w_i = Aw_{i-1} - \frac{\langle w_{i-1}, A^2 w_{i-1}\rangle}{\langle w_{i-1}, Aw_{i-1}\rangle} w_{i-1} - \frac{\langle w_{i-2}, A^2 w_{i-1}\rangle}{\langle w_{i-2}, Aw_{i-2}\rangle} w_{i-2} \tag{3.8}$$

per ogni $i \geq 2$. In questo modo per calcolare w_i è sufficiente utilizzare solo i due vettori precedenti.

Montgomery [11] da una stima della complessità computazionale e spaziale del metodo di Lanczos nel caso in cui si utilizzi la definizione 3.8. Tale stima può essere riassunta nel seguente teorema.

Teorema 3.3.8. *La complessità computazionale del metodo di Lanczos è $O(dn^2) + O(n^2)$ dove d è il numero medio di elementi non nulli per riga (o colonna) della matrice A. La complessità spaziale è invece $O(1)$.*

Riassumendo, abbiamo visto che se A è una matrice simmetrica definita positiva a coefficienti in $\mathbb{R}$ il metodo di Lanczos genera una famiglia di vettori $\{w_0, \ldots, w_{m-1}\}$ tali che:

$$\begin{aligned} &\langle w_i, Aw_i\rangle \neq 0 \quad \text{per } 0 \leq i \leq m-1 \\ &\langle w_j, Aw_i\rangle = 0 \quad \text{per } i \neq j \\ &\quad A\mathcal{W} \subseteq \mathcal{W} \quad \text{dove } \mathcal{W} = \text{Span}(w_0, \ldots, w_{m-1}) \end{aligned}$$

In particolare si ha che i w_i sono tutti diversi dal vettore nullo. Su un generico campo K, però, si potrebbe avere un vettore w_i non nullo che violi la prima

proprietà, ovvero che sia A-coniugato a se stesso. In questo caso l'algoritmo di Lanczos fallirebbe. Se però l'ordine del campo K è molto più grande di n, la probabilità che ciò accada è trascurabile [11], soprattutto se è eventualmente possibile rieseguire l'algoritmo con dati leggermente differenti (si veda l'osservazione 3.3.9). Quindi su un campo generico K l'algoritmo di Lanczos diventa probabilistico di tipo Monte Carlo.

L'algoritmo di Lanczos necessita di una matrice simmetrica A e questa è una condizione piuttosto restrittiva. Nel caso in esame, la matrice associata al sistema ottenuto dalla prima fase del metodo di calcolo dell'indice (o dall'eliminazione strutturata di Gauss) non è neanche quadrata. Riprendendo le notazioni usuali, la matrice A è una matrice $m \times n$ con $m > n$ di rango massimo. La possibilità di scegliere m molto più grande n permette di avere rango massimo con alta probabilità.

Si costruisce una matrice $m \times m$ diagonale D i cui elementi sulla diagonale sono scelti a caso tra gli elementi non nulli del campo K. Considerando il sistema lineare $Ax = b$, si moltiplicano a sinistra entrambi i membri per $A^T D^2$ ottenendo

$$A^T D^2 Ax = A^T D^2 b.$$

La matrice associata a questo sistema lineare è quadrata e simmetrica. Infatti

$$(A^T D^2 A)^T = A^T (D^2)^T (A^T)^T = A^T D^2 A$$

dove si è usato il fatto che D^2 è diagonale e dunque simmetrica. Essendo A una matrice $m \times n$, $A^T D^2 A$ è $n \times n$. Definiamo

$$M = A^T D^2 A, \qquad c = A^T D^2 b$$

Osserviamo che una soluzione di $Ax = b$ è sempre soluzione anche di $Mx = c$. Inoltre $Ax = b$ ha una soluzione unica in quanto $\mathrm{rk}(A) = n$, ovvero ha rango massimo. Ne segue che se $\mathrm{rk}(M) = n$, ovvero se M ha rango massimo, anche $Mx = c$ ha una soluzione unica. In particolare, ossendo ogni soluzione di $Ax = b$ una soluzione di $Mx = c$ ed avendo entrambi soluzione unica, le due soluzioni coincidono. Questo significa che è possibile trovare la soluzione di $Ax = b$, che proviene dal problema del logaritmo discreto, risolvendo $Mx = c$ la cui matrice associata è simmetrica e sul quale si può quindi applicare l'algoritmo di Lanczos.

Nel caso in cui $\mathrm{rk}(M) < n$ la soluzione non è unica. Osserviamo che l'insieme delle soluzioni è molto grande specialmente se $|K|$ è molto grande. In tal caso è possibile definire una nuova matrice D' diversa da D e controllare nuovamente il rango. Siccome trovare un rango non massimo è poco probabile

su campi di ordine grande, provando diverse matrici D scelte a caso si ha un'alta probabilità di ottenere $\mathrm{rk}(M) = n$.

Osservazione 3.3.9. Abbiamo già notato che su un campo K generico il metodo di Lanczos diventa un algoritmo Monte Carlo. Riassumendo, i fattori che lo rendono probabilistico sono i seguenti:

- La possibilità di costruire una matrice A di rango non massimo. La probabilità che ciò accada può esser resa trascurabile cercando un numero di relazioni molto maggiore al numero delle incognite del sistema.
- La possibilità su K di trovare vettori non nulli A-ortogonali a se stessi. Questa probabilità è minimizzata lavorando su un campo di ordine molto più grande rispetto al numero delle incognite [11]. Quindi anzichè considerare K si può lavorare in un'opportuna estensione $\overline{K}$ di K e scegliere i coefficienti della matrice D in $\overline{K}$. Inoltre è possibile scegliere B in modo che $\pi(B)$ non sia troppo grande (ricordando che $\pi(B)$ è il numero di incognite nel sistema del tipo $\log_g p_i$) e fare in modo che n sia molto più piccolo di $|\overline{K}|$ dove $n = \pi(B) + |R|$ con R definito nell'osservazione 3.2.1.
- La possibilità di ottenere una matrice M di rango non massimo durante il processo per rendere simmetrica la matrice A del sistema. Come nel punto precedente, lavorando in un campo di ordine grande tale probabilità diventa molto bassa. Inoltre, come già descritto in precedenza, è sempre possibile cambiare la scelta della matrice D.

3.4 Terza fase: calcolo del logaritmo

Finita la fase di inizializzazione del metodo di calcolo dell'indice inizia la terza fase. Arrivati a questo punto sono noti i valori di $\log_g p$ per ogni $p \in F(B)$ e di $\log_g(H + \beta)$ per ogni $\beta \in M \cup L$.

Osservazione 3.4.1. Conoscendo il valore di $\log_g p_i$ per ogni $p_i \in F(B)$ è possibile determinare immediatamente $\log_g u$ per ogni B-numero u. Infatti:

$$u = \prod_{p_i \in F(B)} p_i^{e(p_i)} \text{ quindi } \Rightarrow \log_g u \equiv \sum_{p_i \in F(B)} e(p_i) \log_g p_i \bmod p - 1$$

Ricordiamo i dati del problema: si vuole trovare x dove $x = \log_g y$ e g è un generatore di $\mathbb{Z}_p^\star$. Il logaritmo discreto esiste ed è unico modulo $p - 1$

Si definisce un intero $U > B$ e si costruisce la base di fattori $F(U)$. Per definizione la base di fattori associata ad U è più grande di quella associata

a B, in particolare $F(U) \supset F(B)$. Denotiamo con p_i i primi in $F(B)$ e con q_i i primi in $\mathcal{Q} = F(U) - F(B)$.

Si cerca un w intero non negativo tale che yg^w sia un U-numero. Per farlo, si fissa a caso w e si controlla se yg^w è un U-numero. Il controllo può essere fatto tramite il metodo di fattorizzazione ρ di Pollard; per i dettagli di questo metodo, simile all'algoritmo ρ per la ricerca del logaritmo discreto presentato nel capitolo precedente, si fa riferimento a [7]. Una volta trovato un appropriato w si avrà

$$yg^w = \Big(\prod_{p_i \in F(B)} p_i^{e(p_i)} \Big) \Big(\prod_{q_i \in \mathcal{Q}} q_i^{e(q_i)} \Big).$$

Applicando le proprietà del logaritmo discreto è possibile ottenere una relazione per il calcolo della soluzione del problema:

$$x = \log_g y \equiv \sum_{p_i \in F(B)} \Big(e(p_i) \log_g p_i \Big) + \sum_{q_i \in \mathcal{Q}} \Big(e(q_i) \log_g q_i \Big) - w \bmod p - 1. \quad (3.9)$$

In questa, sono noti w perchè fissato e $\log_g p_i$ per ogni $p_i \in F(B)$ dalla fase di inizializzazione. Resta da determinare il valore di $\log_g q_i$ per $q_i \in \mathcal{Q}$.

Sia fissato un $q \in \mathcal{Q}$. Per calcolare il suo logaritmo discreto si costruiscono i seguenti residui:

- Si fissa $u_0 = \left\lceil \frac{\sqrt{p}}{q} \right\rceil$. Si controlla se tale u_0 è un B-numero. Se non lo è si considera $u_1 = u_0 + 1$ e si itera fino a trovare un B-numero. Questo sarà della forma $u = \left\lceil \frac{\sqrt{p}}{q} \right\rceil + \alpha_u$. Essendo u un B-numero, dall'osservazione 3.4.1 è noto che è possibile determinare immediatamente $\log_g u$.

- Si fissa $v_0 = H = \lceil \sqrt{p} \rceil$. Come nel passo precedente, si aumenta v_0 aggiungendo progressivamente 1 fino a trovare un $v = H + \alpha_v$ tale che n, definito come il più piccolo residuo non negativo di quv modulo p, sia un B-numero. In questo modo, $\log_g n$ è direttamente calcolabile per quanto visto nell'osservazione 3.4.1. Inoltre, essendo $v = H + \alpha_v$, $\log_g v$ sarà con alta probabilità noto dalla fase di inizializzazione perchè della forma $\log_g(H + \beta)$.

I residui costruiti sono legati dalla relazione $n \equiv quv \bmod p$. Applicando il logaritmo discreto si ottiene

$$\log_g n \equiv \log_g q + \log_g u + \log_g v \bmod p - 1.$$

In questa $\log_g n$, $\log_g u$ e $\log_g v$ sono tutti noti quindi è possibile calcolare $\log_g q$ come

$$\log_g q \equiv \log_g n - \log_g u - \log_g v \bmod p - 1.$$

Ripetendo questo procedimento per ogni $q \in \mathcal{Q}$ si possono determinare tutti i termini ancora incogniti della relazione 3.9 calcolando infine x, la soluzione del problema del logaritmo discreto cercata.

Osservazione 3.4.2. All'inizio della terza fase si chiede di trovare un intero w tale che yg^w sia un U-numero. Questo implica la costruzione di una base di fattori $F(B)$ e la ricerca di un w per tentativi. A priori si potrebbe pensare di eliminare questo passaggio. Infatti, se fosse nota la fattorizzazione

$$y = \Big(\prod_{p_i \in F(B)} p_i^{e(p_i)} \Big) \Big(\prod_{q|y} q^{e(q)} \Big)$$

dove $q \notin F(B)$ sono primi, sarebbe comunque possibile costruire i residui u, v ed n per determinare i $\log_g q$ e calcolare infine $x = \log_g y$. In questo modo però è necessaria la fattorizzazione completa dell'intero y che non è un problema di facile risoluzione. Con l'introduzione di $F(U)$, invece, il problema si semplifica in quanto la fattorizzazione in una base di fattori è un problema molto più semplice rispetto alla fattorizzazione in generale.

3.5 Osservazioni sulla complessità

L'analisi della complessità computazionale richiede strumenti matematici che esulano da questa trattazione. Una esposizione dettagliata dei calcoli, delle approssimazioni fatte e delle stime delle funzioni aritmetiche in uso può essere trovata in [3], [12], in questa sezione ci limiteremo a descrivere i risultati principali.

3.5.1 La notazione L

Per esprimere la complessità del metodo di calcolo dell'indice usiamo la *notazione L*. Questa, analogamente alla notazione O-grande, permette di esprimere una stima asintotica. In particolare si definisce

$$L_p[s, c] = e^{(c+o(1))(\log p)^s (\log\log p)^{1-s}}$$

dove $c > 0$ ed $s \in [0, 1]$ sono costanti reali. Si può notare che:

- Se $s = 0$, allora $L_p[0, c] = e^{(c+o(1))\log\log p} = e^{\log(\log p)^{c+o(1)}} = (\log p)^{c+o(1)}$. Quindi $L_p[0, c]$ è una funzione polinomiale nel numero di cifre di p.

- Se $s = 1$, allora $L_p[1, c] = e^{(c+o(1))\log p} = e^{\log(p^{c+o(1)})} = p^{c+o(1)}$. Quindi $L_p[1, c]$ è una funzione esponenziale nel numero di cifre di p.

La notazione L è utilizzata nei casi in cui l'andamento della funzione non è nè polinomiale nè esponenziale ma in qualche modo intermedio. Quando un algoritmo ha una complessità di questo tipo si dice che ha *complessità sub-esponenziale*. Vedremo in seguito che il metodo di calcolo dell'indice risulta avere proprio questo tipo di complessità.

3.5.2 Complessità della fase di inizializzazione

Il primo importante parametro da valutare è l'intero B. La scelta di un B grande comporta la costruzione di una base di fattori con molti primi che facilita la ricerca dei B-numeri. D'altra parte, però, un B piccolo rende più efficiente il crivello di polinomi in quanto dal teorema 3.1.3 si osserva che il fattore dominante della stima della complessità è $\pi(B)$ che è una funzione crescente in B. La scelta del valore ottimale per B deve quindi tenere conto di questi due fatti.

Nella sezione 3.2 è stato presentato un metodo per la ricerca di relazioni utilizzando il polinomio $f_\lambda(\mu)$ con λ, μ abbastanza piccoli, ovvero tali che $0 < \lambda < \mu \leq C$ per una certa costante C. Un altro importante parametro per lo studio della complessità è valutare cosa si intende per *abbastanza piccoli* ovvero come scegliere C in modo ottimale.

Si dimostra [3] che la scelta ottimale dei due parametri risulta essere $B = C = L_p[\frac{1}{2}, \frac{1}{2}]$. Con questa scelta di B e C la complessità della fase di inizializzazione è $O(L_p[\frac{1}{2}, 1])$. Questa è una complessità sub-esponenziale infatti si ha

$$(\log p)^{1+o(1)} = L_p[0, 1] < L_p[\frac{1}{2}, 1] < L_p[1, 1] = p^{1+o(1)}$$

in quanto, ricordando che $x > logx$ per ogni x abbastanza grande,

$$e^{\log\log p} < e^{\sqrt{\log p}\sqrt{\log\log p}} < e^{\log p}$$

C'è da osservare però che la complessità computazionale così trovata per questa fase è solo una stima euristica ovvero non provata in maniera rigorosa.

3.5.3 Complessità della terza fase

Come nella fase di inizializzazione, anche nella terza fase è richiesta la scelta di una base di fattori. Il valore ottimale di U che determina la base di fattori e la rispettiva complessità dipendono dall'algoritmo utilizzato per testare se un

numero è o non è un U-numero. Ad esempio, utilizzando la fattorizzazione ρ di Pollard, la scelta ottimale risulta essere $U = L_p[\frac{1}{2}, 1]$. Con questa scelta di U la ricerca di w nella terza fase ha complessità computazionale $O(L_p[\frac{1}{2}, 1])$. Così la ricerca di w ha la stessa complessità della fase di inizializzazione. Utilizzando invece il test per tentativi la ricerca diventa più lenta. Esistono test molto più efficienti di questi (che utilizzano le curve ellittiche) che consentono di velocizzare la terza fase rendendola molto più veloce di quella di inizializzazione. In questo caso la scelta ottimale di U è

$$U = L_p\Big[\frac{2}{3}, \sqrt[3]{\frac{1}{3}}\Big]$$

e la rispettiva complessità della ricerca di w è

$$L_p\Big[\frac{1}{3}, \sqrt[3]{3}\Big].$$

Utilizzando questi valori la terza fase risulta avere complessità computazionale $O(\sqrt[6]{\log p})$. I dettagli di tale analisi possono essere trovati in [3].

Bibliografia

[1] Carl Pomerance, Elementary thoughts on discrete logarithms. *Algorithmic number theory: lattices, number fields, curves and cryptography*, 385-396, Math. Sci. Res. Inst. Publ., **44**, *Cambridge Univ. Press, Cambridge*, 2008.

[2] Johannes A. Buchmann, Introduction to cryptography. Second edition. Undergraduate Texts in Mathematics. *Springer-Verlag, New York*, 2004.

[3] Chris Studholme, The discrete log problem, preprint (2002).

[4] J.M. Pollard, Monte Carlo methods for index computation (mod p). *Math. Comp.* **32** (1978), no. 143, 918-924.

[5] Edlyn Teske, Computing discrete logarithms with the parallelized kangaroo method. The 2000 Com2MaC Workshop on Cryptography (Pohang). *Discrete Appl. Math.* **130** (2003), no. 1, 61-82.

[6] Oliver Schirokauer, Discrete logarithms and local units. *Philos. Trans. Roy. Soc. London Ser. A* **345** (1993), no. 1676, 409-423.

[7] M.W. Baldoni, C. Ciliberto, G.M. Piacentini Cattaneo, Aritmetica, crittografia e codici. UNITEXT / La Matematica per il 3+2. *Springer-Verlag*, 2006.

[8] S.C. Pohlig, M.E. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. Information Theory* **IT-24** (1978), no. 1, 106-110.

[9] A.M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance. *Advances in cryptology (Paris, 1984)*, 224-314, Lecture Notes in Comput. Sci., 209, *Springer, Berlin*, 1985.

[10] B.A. LaMacchia, A.M. Odlyzko, Solving large sparse linear systems over finite fields. *Advances in cryptology-CRYPTO '90 (Murray Hill, 1991)*, 109-133, Lecture Notes in Comput. Sci., 537, *Springer, Berlin*, 1991.

[11] P.L. Montgomery, A block Lanczos algorithm for finding dependencies over $GF(2)$. (English summary) *Advances in cryptology-EUROCRYPT '95 (Saint-Malo, 1995)*, 106-120, Lecture Notes in Comput. Sci., 921, *Springer, Berlin*, 1995.

[12] N.G. de Bruijn, On the number of positive integers $\leq x$ and free of prime factors $> y$. *Nederl. Acad. Wetensch. Proc. Ser. A.* **54**, (1951). 50-60.