SOMME DI GAUSS E TEST DI PRIMALITÀ: ANALISI TEORICA E IMPLEMENTAZIONE

RELATORE: Ch.mo Prof. Alberto Tonolo

LAUREANDO: Daniela Adore

A.A. 2010-2011

UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

SOMME DI GAUSS E TEST DI PRIMALITÀ: ANALISI TEORICA E IMPLEMENTAZIONE

RELATORE: Ch.mo Prof. Alberto Tonolo

LAUREANDO: *Daniela* Adore

Padova, 12 Luglio 2011

Indice

Capitolo 1

Introduzione

Un numero primo è un numero naturale che ammette esattamente due divisori: se stesso e uno.
I numeri primi sono stati studiati sin dall'antichità: la prima dimostrazione dell'esistenza di infiniti numeri primi compare negli *Elementi* di Euclide scritti attorno al 300 a.C.
Stabilire se un numero è primo o meno non è un problema facile; tuttavia esistono vari metodi (detti test di primalità) per generare, controllare, certificare la primalità di un numero, il più antico dei quali è il *crivello di Eratostene* noto già in epoca classica ma computazionalmente molto costoso.
I test di primalità si distinguono in test di primalità deterministici e test di primalità probabilistici: i primi sono algoritmi che in un numero finito di passaggi riconoscono se il numero testato è primo o è composto mentre i test probabilistici, in un numero finito di passaggi, o riconoscono che il numero testato è composto o dichiarano che il numero è primo con una certa probabilità. I test di primalità più efficienti sono i test probabilistici come ad esempio il test di Fermat o il test di Miller-Rabin.
I numeri primi hanno trovato applicazione nella crittografia: ad esempio in RSA, uno degli algoritmi di crittografia a chiave pubblica più diffuso, la creazione della chiave in fase di cifratura richiede la conoscenza di numeri primi abbastanza grandi.
Adleman, Pomerance e Rumely nel 1980 idearono un test di primalità (APR test) diverso dai precedenti: è un test che non da infatti soltanto una risposta al quesito di primalità ma, se il numero è composto, è in grado di fornire un insieme ristretto all'interno del quale è presente il suo più piccolo fattore primo. A.K. Lenstra successivamente modificò e riformulò il test APR introducendo l'uso delle somme di Gauss come strumento per la verifica della primalità del numero testato, ed è proprio quest'ultima versione del test l'oggetto di studio di questo elaborato.
Tutte le versioni del test APR hanno complessità $O(\log N^{c \log\log\log N})$ e pertanto quasi polinomiale in $\log N$ (indicato con N il numero testato) ed il test APR è il più veloce tra i test di primalità con complessità quasi polinomiale. L'implementazione onerosa rende il test poco utile per fini commerciali, dove i numeri da testare sono relativamente piccoli; in ogni caso il test APR resta tutt'oggi il

test più efficiente per la verifica della primalità di numeri grandi.
Nell'elaborato che segue si inizia definendo gli strumenti algebrici utilizzati, da quelli più comuni, trattati nei corsi di Algebra, a quelli più avanzati come i *caratteri di Dirichlet* o le *somme di Gauss* elementi chiave del test APR-L considerato. Si procede dunque a descrivere completamente il test (dallo pseudocodice alla dimostrazione della correttezza, alla sua implementazione) studiando sia la versione deterministica che quella probabilistica.

Capitolo 2

Preliminari Matematici

Prima di definire nello specifico le somme di Gauss, elemento fondamentale dell'algoritmo analizzato, è utile ricordare alcuni concetti matematici che verranno ripresi poi di seguito.

2.1 Residui Quadratici

Dato $n \in \mathbb{N}$ indichiamo con $\mathbb{Z}_n$ l'insieme delle classi resto modulo n e con $\mathbb{Z}_n^*$ l'insieme formato dagli elementi di $\mathbb{Z}_n$ che sono relativamente primi con n.

Definizione 2.1.1. Sia p un numero primo dispari e $a \in \mathbb{Z}$. Definiamo il *simbolo di Legendre* come:

$$\left(\frac{a}{p}\right) := \begin{cases} 0 & \text{se } p \mid a. \\ 1 & \text{se } p \nmid a \text{ e l'equazione } x^2 \equiv a \bmod p \text{ è risolubile.} \\ -1 & \text{se } p \nmid a \text{ e l'equazione } x^2 \equiv a \bmod p \text{ non è risolubile.} \end{cases} .$$

Diremo che a è un *residuo quadratico modulo* p se $\left(\frac{a}{p}\right) = 1$ e che a è un *non residuo quadratico* se $\left(\frac{a}{p}\right) = -1$.

Lemma 2.1.2. *Il* simbolo di Legendre *è completamente moltiplicativo nell'argomento.* $\forall\ a, b \in \mathbb{Z}$ *e* p *primo dispari, si ha*

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right).$$

Proposizione 2.1.3. *Sia* p *un numero primo dispari e siano* a *e* $b \in \mathbb{Z}$*; se* $a \equiv b \pmod{p}$*, allora:*

$$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right).$$

Teorema 2.1.4 (**Gauss**)**.** *Se* p *e* q *sono primi distinti dispari, allora*

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{(p-1)(q-1)/4}, \quad \textit{mentre} \quad \left(\frac{2}{p}\right) = (-1)^{\left(p^2-1\right)/8}.$$

Per la dimostrazione di questo Teorema, che esula dai confini di questa trattazione, si può fare riferimento all'opera di Hardy e Wright [10]. Il Teorema 2.1.4 ci permette di definire le due *leggi di reciprocità quadratica*: siano p e q primi dispari,

$$\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right) \cdot \begin{cases} -1 & \text{se } p \equiv q \equiv 3 \pmod 4; \\ 1 & \text{altrimenti.} \end{cases}$$

$$\left(\frac{2}{p}\right) = \begin{cases} 1 & \text{se } p \equiv \pm 1 \pmod 8; \\ -1 & \text{se } p \equiv \pm 3 \pmod 8. \end{cases}$$

Le leggi di reciprocità quadratica consentono di determinare facilmente se la congruenza

$$x^2 \equiv a \pmod p$$

ha soluzioni; per la determinazione delle soluzioni (nel caso in cui esse esistano) non esiste invece un modo altrettanto diretto.

Esempio 2.1.5. Data la congruenza $x^2 \equiv 42 \pmod{47}$ si vuole determinare se essa ammette soluzioni.

$$\begin{aligned} \left(\frac{42}{47}\right) &= \left(\frac{2}{47}\right)\left(\frac{3}{47}\right)\left(\frac{7}{47}\right) = 1\left(\frac{47}{3}\right)(-1)\left(\frac{47}{7}\right)(-1) \\ &= \left(\frac{2}{3}\right)\left(\frac{5}{7}\right) = (-1)\left(\frac{7}{5}\right) = (-1)\left(\frac{2}{5}\right) = (-1)(-1) = 1 \end{aligned}$$

La congruenza $x^2 \equiv 42 \pmod{47}$ è perciò risolubile.

Esempio 2.1.6. Consideriamo un altro esempio: si vuole determinare se la congruenza $x^2 \equiv 57 \pmod{71}$ è risolubile. Come nell'esempio precedente applichiamo le leggi di reciprocità quadratica.

$$\begin{aligned} \left(\frac{57}{71}\right) &= \left(\frac{3}{71}\right)\left(\frac{19}{71}\right) = \left(\frac{71}{3}\right)(-1)\left(\frac{71}{19}\right)(-1) \\ &= \left(\frac{2}{3}\right)\left(\frac{14}{19}\right) = (-1)\left(\frac{2}{19}\right)\left(\frac{7}{19}\right) = (-1)(-1)\left(\frac{19}{7}\right) \\ &= \left(\frac{5}{7}\right) = \left(\frac{7}{5}\right) = \left(\frac{2}{5}\right) = -1 \end{aligned}$$

Avendo ottenuto come risultato finale il valore -1 si deduce che la congruenza $x^2 \equiv 57 \pmod{71}$ non è risolubile.

Definizione 2.1.7. Sia $m \in \mathbb{Z}$ dispari, e sia a un intero qualsiasi; il *simbolo di Jacobi* è definito come:

$$\left(\frac{a}{m}\right) = \prod \left(\frac{a}{p_i}\right)^{t_i},$$

dove $m = \prod_{i=1}^{l} p_i^{t_i}$ indica la scomposizione in fattori primi di m e per ogni i tale che $1 \leq i \leq l$, $\left(\frac{a}{p_i}\right)$ è il *simbolo di Legendre*.

Si osservi che il fatto che *il simbolo di Jacobi* $\left(\frac{a}{m}\right)$ sia uguale ad 1 è solo una condizione necessaria affinchè a sia *un residuo quadratico* modulo m.

Esempio 2.1.8. Sia $m = 15$ e $a = 2$, il simbolo di Jacobi è quindi $\left(\frac{2}{15}\right)$. Applicando le leggi di reciprocità quadratica otteniamo:

$$\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = (-1)(-1) = 1$$

Ma la congruenza $x^2 \equiv 2 \pmod{15}$ non ammette soluzioni.

Oss 2.1.9. Sia $m = p \cdot q$, con p e q numeri primi dispari, e sia a un intero qualsiasi; a è un *residuo quadratico* modulo m se e solo se è un *residuo quadratico* sia modulo p che modulo q.

Oss 2.1.10. Se il simbolo di Jacobi $\left(\frac{a}{m}\right) = 0$ ciò implica che $gcd(a, m) > 1$; se invece $\left(\frac{a}{m}\right) = -1$ allora a e m sono relativamente primi tra loro e la congruenza $x^2 \equiv a \pmod{m}$ non ammette soluzioni.

Oss 2.1.11. Essendo il simbolo di Jacobi ottenuto a partire dal simbolo di Legendre, vale anche per esso la proprietà moltiplicativa espressa nel Lemma 2.1.2.

2.2 Caratteri di Dirichlet

Definizione 2.2.1. Sia $D \in \mathbb{Z}$, $D > 0$ e sia $\chi : \mathbb{Z} \to \mathbb{C}$ una funzione. Allora χ è detto *carattere di Dirichlet modulo D* se soddisfa le seguenti condizioni:

- $\forall\ m, n \in \mathbb{Z}$ si ha $\chi(mn) = \chi(m)\chi(n)$;
- χ è periodica modulo D, cioè $\chi(m + D) = \chi(m)$;
- $\chi(n) = 0 \iff gcd(n, D) > 1$.

Definizione 2.2.2. Sia $D \in \mathbb{Z}, D > 0$, si definisce *carattere di Dirichlet principale* il carattere $\chi_0 : \mathbb{Z} \to \mathbb{C}$ tale per cui:

$$\forall\ n \in \mathbb{Z}, \qquad \chi_0(n) = \begin{cases} 1 & \text{se } gdc(n, D) = 1 \\ 0 & \text{altrimenti.} \end{cases}$$

Oss 2.2.3. Considerando un carattere χ modulo D e analizzando la definizione di *carattere di Dirichlet* possiamo osservare che la proprietà di periodicità propria del carattere stesso permette di passare dalla funzione χ ad una mappa definita dall'insieme delle classi resto modulo D, $\mathbb{Z}_D$, ai complessi. La nuova funzione (che chiameremo $\overline{\chi}$) è così definita:

$$\overline{\chi} : \mathbb{Z}_D \to \mathbb{C},$$

e $\overline{\chi}(a) = \overline{\chi}(a + D\mathbb{Z}) = \chi(a)$.
Analogamente, tenendo conto della terza proprietà del *carattere di Dirichlet* si può costruire una nuova mappa, χ^*, considerando come dominio i soli elementi di $\mathbb{Z}_D$ che siano relativamente primi con D e come codominio l'insieme dei numeri complessi privati dello zero,

$$\chi^* : \mathbb{Z}_D^* \to \mathbb{C}\setminus\{0\}\,.$$

La mappa χ^* è un morfismo tra gruppi moltiplicativi; χ induce quindi ad un morfismo tra $\mathbb{Z}_D^*$ e $\mathbb{C}\setminus\{0\}$.
Viceversa, sia $\psi : \mathbb{Z}_D^* \to \mathbb{C}\setminus\{0\}$ un morfismo che associa ad ogni elemento di $\mathbb{Z}_D^*$ un numero complesso non nullo; è possibile passare ad una mappa $\overline{\psi} : \mathbb{Z}_D \to \mathbb{C}$ così definita:

$$\overline{\psi}(a) = \begin{cases} \psi(a) & \text{se } gcd(a, D) = 1; \\ 0 & \text{altrimenti.} \end{cases}$$

Infine, dalla funzione $\overline{\psi} : \mathbb{Z}_D \to \mathbb{C}$ è possibile passare ad una funzione ψ^* che associa ad ogni numero intero un numero complesso ponendo $\psi^*(a) = \overline{\psi}(a + D\mathbb{Z})$ per ogni $a \in \mathbb{Z}$. Ma allora ψ^* è un *carattere di Dirichlet* perchè soddisfa tutte le condizioni espresse in 2.2.1.
Concludendo si può dire che ogni *carattere di Dirichlet* modulo D induce un morfismo tra $\mathbb{Z}_D^*$ e $\mathbb{C}\setminus\{0\}$; e ogni morfismo così definito induce un *carattere di Dirichlet* modulo D.

Proposizione 2.2.4. *Sia χ un carattere di Dirichlet modulo D, sia $n \in \mathbb{Z}$ tale per cui $gcd(n, D) = 1$.*
Allora $\chi(n)^{\varphi(D)} = 1$; $\chi(n)$ è una radice $\varphi(D)$-esima dell'unità*.*

Dimostrazione.

$$\chi(n)^{\varphi(D)} = \chi(n^{\varphi(D)}) = \chi(1)$$

per il teorema di Eulero. Ma $\chi(1) = 1$ infatti

$$\chi(1) = \chi(1 \cdot 1) = \chi(1)\chi(1) = \chi(1)^2 \neq 0,$$

perchè $gcd(1, D) = 1$; quindi $\chi(1) = 1$. □

La Proposizione 2.2.4 ha due conseguenze fondamentali:

- Il numero di caratteri di Dirichlet modulo D con $D \in \mathbb{N} \setminus \{0\}$ è finito.
- $\forall\, n \in \mathbb{Z}$ ed ogni carattere $\chi : \mathbb{Z} \to \mathbb{C}$, posto $\overline{\chi}(n) = \overline{\chi(n)}$ per ogni $n \in \mathbb{Z}$, si ha

$$\chi\overline{\chi}(n) = \chi(n)\overline{\chi}(n) = \chi(n)\overline{\chi(n)} = |\chi(n)|^2 = \begin{cases} 1 & \text{se } gcd(n, D) = 1 \\ 0 & \text{se } gcd(n, D) > 1 \end{cases} = \chi_0(n)$$

Proposizione 2.2.5. *Siano χ_1 modulo D_1 e χ_2 modulo D_2 due distinti caratteri di Dirichlet, con $D_1 \neq D_2$. Allora $\chi_1\chi_2$ è ancora un carattere di Dirichlet modulo il minimo comune multiplo tra D_1 e D_2; si ha inoltre che*

$$(\chi_1\chi_2)(n) = \chi_1(n)\chi_2(n).$$

Proposizione 2.2.6. *Sia χ un carattere di Dirichlet modulo D e sia $p_1^{a_1} \cdots p_k^{a_k}$ la scomposizione in fattori primi di D. Allora il carattere χ modulo D può essere univocamente fattorizzato come $\chi_1 \cdots \chi_k$ dove χ_j è un carattere modulo $p_j^{a_j}$.*

Una volta definito come si costruisce un carattere di Dirichlet modulo la potenza di un primo, sarà quindi immediato definire un carattere modulo D con D intero qualsiasi. Il prossimo passo consiste quindi nel descrivere come si costruisce un carattere $\chi \pmod{p^a}$ dove p è un numero primo.
Si consideri il gruppo moltiplicativo $\mathbb{Z}^*_{p^a}$; per il Teorema di Gauss ([12], pag.52) $\mathbb{Z}^*_{p^a}$ è un gruppo ciclico, esiste quindi un elemento $g \in \mathbb{Z}^*_{p^a}$ detto *generatore* tale per cui ogni elemento di $\mathbb{Z}^*_{p^a}$ è esprimibile come potenza di g.
Sia η una radice $\varphi(p^a)$-esima dell'unità e sia $\chi(g) = \eta$; una volta determinato il valore del carattere per il generatore, sono automaticamente determinati i valori per ogni elemento di $\mathbb{Z}^*_{p^a}$.
Infatti, sia $x \in \mathbb{Z}^*_{p^a}$, allora

$$\chi(x) = \chi(g^k) = \chi(g)^k = \eta^k, \quad k \in \mathbb{Z}.$$

Esempio 2.2.7. Si vuole definire un carattere χ modulo 5. Il gruppo moltiplicativo $\mathbb{Z}^*_5$ è costituito dalle classi resto $[1], [2], [3], [4]$; per prima cosa si deve determinare il generatore di $\mathbb{Z}^*_5$. Si vede subito che un generatore di $\mathbb{Z}^*_5$ è $[2]$, infatti:

$$[2]^1 = [2] \qquad [2]^2 = [4] \qquad [2]^3 = [8] = [3] \qquad [2]^4 = [16] = [1]$$

Essendo $\varphi(5) = 4$, si considerano le radici quarte dell'unità; esse sono ± 1 e $\pm i$ e rappresentano i possibili valori da associare a $\chi(2)$. Si deduce quindi che i caratteri di Dirichlet modulo 5 sono 4. Consideriamo $\chi(2) = i$, vogliamo determinare il valore del carattere per gli altri elementi di $\mathbb{Z}^*_5$.

$$\begin{aligned}
&\chi(4) = \chi(2^2) = \chi(2)^2 = i^2 = -1 \\
&\chi(3) = \chi(2^3) = \chi(2)^3 = \chi(2)\chi(4) = i(-1) = -i \\
&\chi(1) = \chi(2^4) = \chi(2)^4 = \chi(2)\chi(3) = i(-i) = 1.
\end{aligned}$$

Definizione di un carattere di Dirichlet modulo 5.

Esempio 2.2.8. Si vuole definire un carattere di Dirichlet χ modulo D, con $D = 15$.
Innanzitutto si nota che $15 = 3 \cdot 5$, quindi per la Proposizione 2.2.6 si dovrannno prima definire i caratteri χ_1 e χ_2 che indicano rispettivamente un carattere di Dirichlet modulo 3 e modulo 5; quindi:

$$\chi = \chi_1\chi_2.$$

Passiamo quindi a costruire il carattere χ_1; il gruppo moltiplicativo $\mathbb{Z}_3^*$ è costituito dalle sole due classi resto [1] e [2] e quest'ultima è il generatore di $\mathbb{Z}_3^*$. Essendo poi $\varphi(3) = 2$, i possibili valori (le radici dell'unità) associabili a $\chi(2)$ sono 1 e -1. Se a $\chi_1(2)$ è associato il valore -1, allora

$$\chi_1(1) = \chi_1(2^2) = \chi_1(2)^2 = (-1)^2 = 1.$$

Per quanto riguarda invece il carattere χ_2, consideriamo il carattere definito nell'esempio precedente; si ha quindi:

$$\chi_2(1) = 1 \qquad \chi_2(2) = i \qquad \chi_2(3) = -i \qquad \chi_2(4) = -1$$

Ritornando quindi al carattere di partenza χ modulo 15, si definisce il carattere per i valori di $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ così come definito nella Proposizione 2.2.5.

$$\begin{aligned}
&\chi(1) = (\chi_1\chi_2)(1) = \chi_1(1)\chi_2(1) = 1 \cdot 1 = 1\\
&\chi(2) = (\chi_1\chi_2)(2) = \chi_1(2)\chi_2(2) = -1 \cdot i = -i\\
&\chi(4) = (\chi_1\chi_2)(4) = \chi_1(4)\chi_2(4) = \chi_1(1)\chi_2(4) = 1 \cdot (-1) = -1\\
&\chi(7) = (\chi_1\chi_2)(7) = \chi_1(7)\chi_2(7) = \chi_1(1)\chi_2(2) = 1 \cdot i = i\\
&\chi(8) = (\chi_1\chi_2)(8) = \chi_1(8)\chi_2(8) = \chi_1(2)\chi_2(3) = -1 \cdot (-i) = i\\
&\chi(11) = (\chi_1\chi_2)(11) = \chi_1(11)\chi_2(11) = \chi_1(2)\chi_2(1) = -1 \cdot 1 = -1\\
&\chi(13) = (\chi_1\chi_2)(13) = \chi_1(13)\chi_2(13) = \chi_1(1)\chi_2(3) = 1 \cdot (-i) = -i\\
&\chi(14) = (\chi_1\chi_2)(14) = \chi_1(14)\chi_2(14) = \chi_1(2)\chi_2(4) = -1 \cdot (-1) = 1
\end{aligned}$$

Definizione di un carattere di Dirichlet modulo 15.

Proposizione 2.2.9. *Il* simbolo di Jacobi *è un carattere di Dirichlet.*

Dimostrazione. Sia $D > 1$ un numero intero dispari, dobbiamo dimostrare che il simbolo di Jacobi $\left(\frac{n}{D}\right)$ con $n \in \mathbb{Z}$ è un carattere di Dirichlet modulo D. Devono quindi essere verificate le tre proprietà che definiscono tale carattere:

- $\forall\ m, n \in \mathbb{Z}$ si ha $\left(\frac{mn}{D}\right) = \left(\frac{m}{D}\right)\left(\frac{n}{D}\right)$; vale infatti la relazione moltiplicativa propria del simbolo di Jacobi.
- $\left(\frac{n+D}{D}\right) = \left(\frac{[n+D]_D}{D}\right) = \left(\frac{[n]_D}{D}\right) = \left(\frac{n}{D}\right)$; il simbolo di Jacobi $\left(\frac{n}{D}\right)$ é periodico modulo D.
- Se D è un numero composto, allora $D = \prod_{i=1}^{l} p_i^{t_i}$,
  quindi $\left(\frac{n}{D}\right) = 0 \Rightarrow \prod_{i=1}^{l} \left(\frac{n}{p_i}\right)^{t_i} = 0$. Ciò vuol dire che almeno un primo p_i divide n e di conseguenza $gcd(n, D) > 1$.

□

Proposizione 2.2.10. *I caratteri di Dirichlet modulo D formano un* gruppo moltiplicativo.

Dimostrazione. Verifichiamo che l'insieme X_D di caratteri modulo D è un gruppo.

- $\chi_1(\chi_2\chi_3)(n) = (\chi_1\chi_2)\chi_3(n)$, $\forall\ n \in \mathbb{Z}$; la moltiplicazione di caratteri è quindi un'operazione associativa.
- $\chi_0(n)\chi_1(n) = \chi_1(n)\chi_0(n) = \chi_1\chi_0(n) = \begin{cases} \chi_1(n) & \text{se gcd(n,D)=1} \\ 0 & \text{altrimenti.} \end{cases} = \chi_1(n).$

  Il carattere principale di Dirichlet χ_0 è quindi l'elemento neutro del gruppo moltiplicativo.
- L'inverso di un carattere di Dirichlet è pari al suo complesso coniugato,

$$\chi(n)\overline{\chi}(n) = \chi(n)\overline{\chi(n)} = \chi_0(n).$$

□

Teorema 2.2.11. *Il numero dei caratteri di Dirichlet modulo D è* $\varphi(D)$.

Dimostrazione. Se $D = p^a$ con p numero primo, dalla descrizione della costruzione di un carattere di Dirichlet χ modulo la potenza di un primo fatta ad inizio sezione, le radici associabili a $\chi(g)$ (dove g rappresenta un generatore del gruppo moltiplicativo $\mathbb{Z}^*_{p^a}$) sono $\varphi(p^a)$. Esistono quindi $\varphi(p^a)$ caratteri di Dirichlet distinti modulo p^a.
Se D invece è un numero composto, allora $D = p_1^{a_1} \cdots p_k^{a_k}$ e, come descritto nella Proposizione 2.2.6, un carattere χ modulo D si può scrivere come prodotto di caratteri $\chi_1 \cdots \chi_k$, dove χ_j rappresenta un carattere modulo $p_j^{a_j}$. Avendo appena visto che il numero dei caratteri χ_j è $\varphi(p_j^{a_j})$ con $j \in \{1, ..., k\}$ si ottiene che il numero di caratteri modulo D è pari a:

$$\varphi(p_1^{a_1}) \cdots \varphi(p_k^{a_k}) = \varphi(p_1^{a_1} \cdots p_k^{a_k}) = \varphi(D).$$

□

Esempio 2.2.12. Riprendendo l'esempio 2.2.7, se si vuole definire un carattere modulo 5, i possibili valori da associare a $\chi(2)$ sono ± 1 e $\pm i$, quindi i caratteri di Dirichlet modulo 5 sono $\varphi(5) = 4$.
Nello specifico i caratteri sono così definiti:

$\chi_1(1) = 1$	$\chi_2(1) = 1$	$\chi_3(1) = 1$	$\chi_4(1) = 1$
$\chi_1(2) = \mathbf{1}$	$\chi_2(2) = -\mathbf{1}$	$\chi_3(2) = \mathbf{i}$	$\chi_4(2) = -\mathbf{i}$
$\chi_1(3) = 1$	$\chi_2(3) = -1$	$\chi_3(3) = -i$	$\chi_4(3) = i$
$\chi_1(4) = 1$	$\chi_2(4) = 1$	$\chi_3(4) = -1$	$\chi_4(4) = -1$
$\chi_1(5) = 0$	$\chi_2(5) = 0$	$\chi_3(5) = 0$	$\chi_4(5) = 0$

I caratteri di Dirichlet modulo 5.

Lemma 2.2.13. *(Relazioni di Ortogonalità). I* caratteri di Dirichlet *modulo D soddisfano le seguenti relazioni:*

$$\sum_{n=1}^{D} \chi(n) = \begin{cases} \varphi(D) & se\ \chi = \chi_0 \\ 0 & se\ \chi \neq \chi_0 \end{cases} \tag{2.1}$$

$$\sum_{\chi (mod\ D)} \chi(n) = \begin{cases} \varphi(D) & se\ n \equiv 1 \pmod{D} \\ 0 & se\ n \not\equiv 1 \pmod{D} \end{cases} \tag{2.2}$$

Dimostrazione. Dimostriamo innanzitutto la prima relazione di ortogonalità.
Se $\chi \neq \chi_0 \Rightarrow \exists\, a \mid gcd(a, D) = 1\ e\ \chi(a) \neq 1$. Definiamo

$$S = \sum_{x=1}^{D} \chi(x); \qquad allora \qquad \chi(a)S = \sum_{x=1}^{D} \chi(ax).$$

Essendo $gcd(a, D) = 1$, ax al variare di x tra 1 e D descrive tutti gli elementi di $\mathbb{Z}_D$.
Quindi:

$$\chi(a)S = \sum_{y=1}^{D} \chi(y) = S.$$

Ricordando che $\chi(a) \neq 1$ l'equazione trova soluzione solo se $S = \sum_x \chi(x) = 0$.

Se invece $\chi = \chi_0$, allora $\chi(n) = \begin{cases} 1 & \text{se gcd(n,D)=1} \\ 0 & \text{se gcd(n,D)>1} \end{cases}$

Quindi

$$S = \sum_{n=1}^{D} \chi(n) = \sum_{\substack{n=1 \\ gcd(n,D)=1}}^{D} 1 = |\mathbb{Z}_D^*| = \varphi(D).$$

Consideriamo ora la seconda relazione di ortogonalità; la dimostrazione è molto simile alla precedente.
Se $n \equiv 1 \pmod{D} \Rightarrow \chi(n) = \chi(1) = 1$, quindi per il Teorema 2.2.11

$$\sum_{\chi(\text{mod } D)} \chi(n) = \sum_{\chi(\text{mod } D)} \chi(1) = \sum_{\chi(\text{mod } D)} 1 = \varphi(D).$$

Se $n \not\equiv 1 \pmod{D} \Rightarrow \exists\, \overline{\chi} \mid \overline{\chi}(n) \neq 1$.

Infatti, sia $D = p_1^{a_1} \cdots p_k^{a_k}$, se $n \not\equiv 1 \pmod{D}$ allora $\exists\, i$ tale che $n \not\equiv 1 \pmod{p_i^{a_i}}$ ed esiste un carattere $\chi_i \pmod{p_i^{a_i}}$ tale che $\chi_i(n) \neq 1$.
Sia g un generatore del gruppo $\mathbb{Z}^*_{p_i^{a_i}}$, e sia $n \in \mathbb{Z}^*_{p_i^{a_i}}$; allora n può essere scritto come g^k con k numero intero, $k \neq \varphi(p_i^{a_i})$ perchè $n \not\equiv 1 \pmod{p_i^{a_i}}$. Esiste sicuramente un carattere χ_i tale che $\chi_i(n) \neq 1$ ed è un carattere che associa a $\chi_i(g)$ una radice $\varphi(p_i^{a_i})$-esima primitiva dell'unità. In questo modo si ha:

$$\chi_i(n) = \chi_i(g^k) = \chi_i(g)^k \neq 1,$$

perchè $k \neq \varphi(p_i^{a_i})$.
Il carattere χ_i è anche un carattere modulo D e $\chi_i(n) \neq 1$. Definendo quindi $S = \sum_\chi \chi(n)$ si ha:

$$\chi_i(n)S = \sum_{\chi} \chi_i\chi(n) = \sum_{\chi} \chi(n) = S.$$

Ma $\chi_i(n) \neq 1$ quindi $S = \sum_\chi \chi(n) = 0$. □

Esempio 2.2.14. Si consideri l'esempio 2.2.12; applicando la prima relazione di ortogonalità ad esempio a χ_1 e χ_2 si ottiene:

$$\sum_{k=1}^{5} \chi_1(k) = 1 + 1 + 1 + 1 + 0 = 4 = \varphi(5)$$

$$\sum_{k=1}^{5} \chi_2(k) = 1 + (-1) + (-1) + 1 + 0 = 0$$

Consideriamo ora la seconda relazione di ortogonalità; riprendendo i caratteri di Dirichlet determinati nell'esempio 2.2.12, valutiamo la relazione per $n = 1$ e $n = 2$. Si ottiene:

$$\sum_{\chi(\text{mod } 5)} \chi(1) = 1 + 1 + 1 + 1 + 0 = 4 = \varphi(5)$$
$$\sum_{\chi(\text{mod } 5)} \chi(2) = 1 + (-1) + i + (-i) = 0$$

Capitolo 3

Le somme di Gauss

Le somme di Gauss sono lo strumento matematico che è alla base dell'algoritmo di primalità ideato da Adleman, Pomerance e Rumely nel 1980 e dunque in questo capitolo verranno descritte in maniera esaustiva, dalla loro definizione, al legame con i *caratteri di Dirichlet* fino ad arrivare poi alla dimostrazione del Teorema chiave per l'implementazione dell'algoritmo APR.

3.1 Definizioni e proprietà

Definizione 3.1.1. Sia $\chi \pmod D$ un carattere di Dirichlet modulo D e $n \in \mathbb{Z}$. Si definisce

$$G(n,\chi) = \sum_{m=1}^{D} \chi(m) e^{2\pi i m n/D}$$

la *somma di Gauss* associata a χ e a n.

Se $\chi = \chi_0$ allora

$$G(n,\chi_0) = \sum_{\substack{m=1 \\ (m,D)=1}}^{D} e^{2\pi i m n/D}$$

$G(n,\chi_0)$ viene detta *somma di Ramanujan* ed indicata con $c_D(n)$.

Teorema 3.1.2. *Sia χ un carattere di Dirichlet modulo D, allora:*

$$G(n,\chi) = \overline{\chi(n)} G(1,\chi) \qquad \text{se } gcd(n,D) = 1$$

Dimostrazione. Dato un carattere di Dirichlet χ modulo D, sia $n \in \mathbb{Z}$ tale che $gdc(n,D) = 1$. Allora, per la Proposizione 2.2.4 $\chi(n)$ è una radice $\varphi(D)$-esima dell'unità. Quindi:

$$|\chi(n)|^2 = \chi(n)\overline{\chi(n)} = 1 \quad e \quad \chi(r) = \overline{\chi(n)}\chi(n)\chi(r) = \overline{\chi(n)}\chi(nr), \quad \forall r \in \mathbb{Z}$$

per la proprietà associativa della moltiplicazione tra caratteri.
La *somma di Gauss* vale:

$$G(n,\chi) = \sum_{r=1}^{D-1} \chi(r) e^{2\pi i n r/D} = \overline{\chi(n)} \sum_{r=1}^{D-1} \chi(nr) e^{2\pi i n r/D} = (\text{essendo } gcd(n,D)=1)$$
$$= \overline{\chi(n)} \sum_{m=1}^{D-1} \chi(m) e^{2\pi i m/D} = \overline{\chi(n)} G(1,\chi).$$

□

Definizione 3.1.3. La somma di Gauss $G(n,\chi)$ è detta *separabile* se:

$$G(n,\chi) = \overline{\chi(n)} G(1,\chi)$$

Oss 3.1.4. Dal Teorema 3.1.2 segue che una somma di Gauss è *separabile* se n e D sono relativamente primi tra loro.

Teorema 3.1.5. *Sia χ un carattere di Dirichlet modulo D, la* somma di Gauss $G(n,\chi)$ *è* separabile *per ogni n se e solo se:*

$$G(n,\chi) = 0 \quad \textit{se } gcd(n,D) > 1.$$

Dimostrazione. Se $gcd(n,D) = 1$ si sa che la somma di Gauss $G(n,\chi)$ è sempre separabile; se invece $gcd(n,D) > 1$ si ha che $\chi(n) = 0$ (per definizione) e quindi l'equazione

$$G(n,\chi) = \overline{\chi(n)} G(1,\chi)$$

è verificata solo se $G(n,\chi) = 0$. □

Teorema 3.1.6. *Sia χ un carattere di Dirichlet modulo D e sia $G(n,\chi)$ una somma di Gauss separabile per ogni $n \in \mathbb{Z}$. Allora*

$$|G(1,\chi)|^2 = D.$$

Dimostrazione.

$$|G(1,\chi)|^2 = G(1,\chi)\overline{G(1,\chi)} = G(1,\chi) \sum_{m=1}^{D} \overline{\chi}(m) e^{-2\pi i m/D}$$
$$= \sum_{m=1}^{D} G(m,\chi) e^{-2\pi i m/D} = \sum_{m=1}^{D} \sum_{r=1}^{D} \chi(r) e^{2\pi i m r/D} e^{-2\pi i m/D}$$
$$= \sum_{r=1}^{D} \chi(r) \sum_{m=1}^{D} e^{2\pi i m(r-1)/D}$$

Se $r = 1$ si ha:

$$\chi(1) \sum_{m=1}^{D} 1 = D;$$

se $r \neq 1$ si ottiene invece che

$$\chi(r) \sum_{m=1}^{D} e^{2\pi i m(r-1)/D} = 0;$$

infatti

$$\sum_{m=1}^{D} e^{2\pi i m(r-1)/D} = \sum_{m=1}^{D} e^{2\pi i m a/b} \quad \text{con} \quad gcd(a,b) = 1 \quad \text{e} \quad b|D.$$

$$\sum_{m=1}^{D} e^{2\pi i m a/b} = \sum_{m=1}^{kb} e^{2\pi i m/b} = k \sum_{m=1}^{b} e^{2\pi i m/b} = k \cdot 0 = 0;$$

perché la somma delle b radici b-esime dell'unità è zero. Si ottiene quindi:

$$|G(1,\chi)|^2 = \sum_{r=1}^{D} \chi(r) \sum_{m=1}^{D} e^{2\pi i m(r-1)/k} = D$$

□

3.2 Le somme di Gauss e i caratteri di Dirichlet primitivi

Dai risultati ottenuti finora si è giunti alla conclusione che la condizione di separabilità di una somma di Gauss implica l'annullamento della stessa nel caso in cui $gcd(n, D) > 1$, dove n è un numero intero e D è il modulo del carattere di Dirichlet associato a $G(n, \chi)$.
In questa sezione verrà definito il carattere di Dirichlet *primitivo* e le relazioni con le somme di Gauss definite su di esso; inoltre il Teorema che segue fornisce la condizione necessaria affinché $G(n, \chi)$ sia diversa da zero quando $gcd(n, D) > 1$.

Teorema 3.2.1. *Sia χ un carattere di Dirichlet modulo D, sia $G(n, \chi) \neq 0$ per qualche $n \in \mathbb{Z}$ tale che $gcd(n, D) > 1$. Allora esiste d, un divisore di D, con $d < D$ tale che per ogni a con $gcd(a, D) = 1$ e $a \equiv 1 \pmod{d}$ si ha:*

$$\chi(a) = 1.$$

Dimostrazione. Sia $q = gcd(n, D) > 1$ e sia $d = D/q$. Allora di sicuro $d \mid D$ e $d < D$ perchè $q > 1$.
Si consideri $a \in \mathbb{Z}$ tale che $gcd(a, D) = 1$ e $a \equiv 1 \pmod{d}$; si deve provare che $\chi(a) = 1$.
Tornando alla definizione della somma di Gauss si ha:

$$G(n,\chi) = \sum_{m=1}^{D} \chi(m) e^{2\pi i m n/D} = \sum_{m=1}^{D} \chi(am) e^{2\pi i a m n/D}$$

Applicando la proprietà moltiplicativa del carattere di Dirichlet si ha che

$$\chi(am) = \chi(a)\chi(m),$$

quindi la sommatoria diventa:

$$\chi(a) \sum_{m=1}^{D} \chi(m) e^{2\pi i amn/D}.$$

A partire dalle condizioni iniziali $a \equiv 1 \pmod{d}$ e $d = D/q$ si ha

$$a = 1 + \frac{bD}{q}, \qquad b \in \mathbb{Z};$$

poiché $q \mid n$ si ottiene quindi

$$\frac{anm}{D} = \frac{nm}{D} + \frac{bDmn}{qD} = \frac{nm}{D} + \frac{bnm}{q} = \frac{nm}{D} + bn'm$$

e

$$e^{2\pi i amn/D} = e^{2\pi i nm/D}.$$

La somma di Gauss si può quindi scrivere come:

$$G(n,\chi) = \chi(a) \sum_{m=1}^{D} \chi(m) e^{2\pi i mn/D} = \chi(a) G(n,\chi).$$

Essendo $G(n,\chi) \neq 0$ per ipotesi, si deduce quindi che:

$$\chi(a) = 1.$$

□

Definizione 3.2.2. Sia χ un carattere di Dirichlet modulo D, e sia d un intero positivo tale che $d \mid D$. Allora d è detto *modulo indotto* per χ se:

$$\chi(a) = 1 \qquad \text{quando} \quad gcd(a, D) = 1 \quad \text{e} \quad a \equiv 1 \pmod{d}.$$

Teorema 3.2.3. *Sia χ un carattere di Dirichlet modulo D. Allora* 1 *è un* modulo indotto *per χ se e solo se $\chi = \chi_0$.*

Dimostrazione. Se $\chi = \chi_0$ allora $\chi(a) = \begin{cases} 1 & \text{se } gcd(a, D) = 1, \\ 0 & \text{altrimenti.} \end{cases}$

Essendo ovviamente $a \equiv 1 \pmod{1}$ si deduce che 1 è quindi modulo indotto. Viceversa se 1 è un modulo indotto, allora:

$$\chi(a) = 1 \quad \text{se } gcd(a, D) = 1 \quad e \quad a \equiv 1 \pmod{1}, \quad \text{ovvero se } gcd(a, D) = 1.$$

Si ottiene così proprio la definizione del carattere di Dirichlet principale. □

Definizione 3.2.4. Il carattere di Dirichlet modulo D è detto *primitivo* se non esistono moduli indotti $d < D$; in altre parole χ è *primitivo* modulo D se e solo se per ogni divisore d di D con $0 < d < D$, esiste un intero a tale che

$$a \equiv 1 \pmod{d}, \quad gcd(a, D) = 1 \quad \text{e} \quad \chi(a) \neq 1.$$

Oss 3.2.5. Se $D > 1$, il carattere di Dirichlet principale modulo D *non è primitivo*, perchè 1 è un modulo indotto.

Teorema 3.2.6. *Ogni carattere di Dirichlet non principale modulo p, dove p è un numero primo, è un* carattere primitivo *modulo p.*

Dimostrazione. Sia p un numero primo, i suoi divisori sono quindi 1 e p, e rappresentano quindi i due soli candidati ad essere moduli indotti. Dal Teorema 3.2.3 si sa che 1 non può essere modulo indotto perchè $\chi \neq \chi_0$ per ipotesi; quindi χ non possiede moduli indotti minori di p; χ è dunque *primitivo.* □

Consideriamo ora i risultati ottenuti nella sezione precedente, che legano le somme di Gauss ai caratteri di Dirichlet, nel caso in cui χ sia un carattere primitivo.

Teorema 3.2.7. *Sia χ un carattere di Dirichlet primitivo modulo D. Allora si ha:*

(a) $G(n, \chi) = 0 \quad \forall\ n$ *con* $gcd(n, D) > 1$*;*

(b) $G(n, \chi)$ *é separabile per ogni n;*

(c) $|G(1, \chi)|^2 = D$.

Dimostrazione. Se $G(n, \chi) \neq 0$ per qualche n con $gcd(n, D) > 1$, allora il Teorema 3.2.1 mostra che χ ha un *modulo indotto* $d < D$, quindi χ non può essere un carattere primitivo; si giunge perciò ad un assurdo, dimostrando così il punto (a).
Il punto (b) è facilmente dimostrato come conseguenza del punto (a) e del Teorema 3.1.5.
Infine, anche il punto (c) è una conseguenza del punto (b) e del Teorema 3.1.6. □

Oss 3.2.8. La somma di Gauss $G(n, \chi)$ è separabile se χ è primitivo.

3.3 Somme di Gauss particolari

In questa sezione si vuole definire un carattere di Dirichlet con caratteristiche (modulo e ordine) note a priori, per analizzare poi le dipendenze della somma di Gauss ad esso associata e giungere quindi alla *relazione fondamentale* che lega somme di Gauss e caratteri. Questa relazione rappresenta l'idea chiave dell'algoritmo APR.

Sia q un numero primo, g un generatore del *gruppo ciclico* $\mathbb{Z}_q^*$; si indichi inoltre con ζ una radice $(q-1)$-esima primitiva dell'unità. Si definisce un carattere di Dirichlet χ modulo q nel seguente modo: sia $m = g^k \in \mathbb{Z}_q^*$, allora

$$\chi(m) = \chi(g^k) = \zeta^k.$$

A tale carattere è stata associata la somma di Gauss:

$$G(n,\chi) = \sum_{m=1}^{q-1} \chi(m) e^{2\pi i m/q} = \sum_{m=1}^{q-1} \chi(m)\zeta_q^m = \sum_{k=1}^{q-1} \chi(g^k)\zeta_q^{g^k} = \sum_{k=1}^{q-1} \zeta^k \zeta_q^{g^k},$$

dove ζ_q indica la *radice q-esima primitiva* dell'unità, $\zeta_q = e^{2\pi i/q}$.

Il gruppo dei caratteri di Dirichlet modulo q ha $\varphi(q) = q-1$ elementi. Si supponga p sia un fattore primo di $q-1$; si vuole costruire un carattere di Dirichlet $\chi_{p,q}$ modulo q di ordine p.
Sia $g = g_q$ il più piccolo generatore in $\mathbb{Z}_q^*$; poniamo $\chi_{p,q}(g_q^k) = \zeta_p^k$ per ogni intero k.
In questa maniera si è appena definito un carattere modulo q, infatti

$$\zeta_p^{q-1} = e^{2\pi i(q-1)/p} = 1,$$

perchè p è un divisore di $q-1$.
Sia $m \in \mathbb{Z}_q^*$ allora $m = g_q^k$ per qualche k intero; si ha dunque

$$\chi_{p,q}(m)^p = \chi_{p,q}(g_q^k)^p = \zeta_p^{kp} = e^{2\pi i kp/p} = 1,$$

e

$$\chi_{p,q}(g_q) = \zeta_p \neq 1.$$

Si può quindi concludere che χ ha ordine p. La somma di Gauss associata al carattere di Dirichlet modulo q e di ordine p, con p e q primi, è così definita:

$$G(p,q) = \sum_{m=1}^{q-1} \chi_{p,q}(m)\zeta_q^m = \sum_{k=1}^{q-1} \chi_{p,q}(g_q^k)\zeta_q^{g_q^k} = \sum_{k=1}^{q-1} \zeta_p^k \zeta_q^{g_q^k}.$$

Lemma 3.3.1. *Siano p e q due numeri primi, se $p \mid q-1$ allora*

$$G(p,q)\overline{G(p,q)} = q.$$

Dimostrazione. Per comodità si indichi con χ il carattere $\chi_{p,q}$ modulo q e di ordine p; si ha:

$$G(p,q)\overline{G(p,q)} = \sum_{m_1=1}^{q-1}\sum_{m_2=1}^{q-1} \chi(m_1)\overline{\chi(m_2)}\zeta_q^{m_1-m_2}.$$

Sia m_2^{-1} l'inverso moltiplicativo di m_2 modulo q, allora $\overline{\chi(m_2)} = \chi(m_2^{-1})$. Infatti,

$$\chi(m_2)\chi(m_2^{-1}) = \chi(m_2 m_2^{-1}) = \chi(1) = 1 = |\chi(m_2)|^2 = \chi(m_2)\overline{\chi(m_2)}.$$

Si supponga inoltre che $m_1 m_2^{-1} \equiv a \pmod q$, allora

$$\chi(m_1)\overline{\chi(m_2)} = \chi(m_1 m_2^{-1}) = \chi(a) \qquad e \qquad m_1 - m_2 \equiv (a-1)m_2 \pmod q.$$

Dunque,

$$\begin{aligned} G(p,q)\overline{G(p,q)} &= \sum_{m_2=1}^{q-1} \sum_{m_1=1}^{q-1} \chi(m_1 m_2^{-1}) \zeta_q^{m_1 - m_2} \\ &= \sum_{m_2=1}^{q-1} \sum_{a=1}^{q-1} \chi(a) \zeta_q^{(a-1)m_2} \\ &= \sum_{a=1}^{q-1} \chi(a) \sum_{m_2=1}^{q-1} \zeta_q^{(a-1)m_2}. \end{aligned}$$

Se $a = 1$ si ha:

$$\chi(a) \sum_{m_2=1}^{q-1} \zeta_q^{(a-1)m_2} = \chi(1) \sum_{m_2=1}^{q-1} 1 = q - 1;$$

se invece $a > 1$ si ha:

$$\sum_{m_2=1}^{q-1} \zeta_q^{(a-1)m_2} = \sum_{m_2=0}^{q-1} \zeta_q^{(a-1)m_2} - 1 = \sum_{k=0}^{q-1} \zeta_q^k - 1 = \sum_{k=0}^{q-1} e^{2\pi i k/q} - 1 = -1,$$

perchè come ricordato nella dimostrazione del Teorema 3.1.6 la somma delle q radici q-esime dell'unità vale 0.

Pertanto

$$G(p,q)\overline{G(p,q)} = q - 1 - \sum_{a=2}^{q-1} \chi(a) = q - \sum_{a=1}^{q-1} \chi(a) = q - 0 = q.$$

Nel calcolo dell'ultima sommatoria si è utilizzata la prima relazione di ortogonalità dei caratteri, vedi Lemma 2.2.13. □

Lemma 3.3.2. *Siano p, q, n dei numeri primi tali che $p \mid q-1$ e $gcd(pq, n) = 1$. Allora*

$$G(p,q)^{n^{p-1}-1} \equiv \chi_{p,q}(n) \pmod n$$

Dimostrazione. Sia $\chi = \chi_{p,q}$; essendo n primo si ha

$$G(p,q)^{n^{p-1}} = \left(\sum_{m=1}^{q-1} \chi(m) \zeta_q^m \right)^{n^{p-1}} \equiv \sum_{m=1}^{q-1} \chi(m)^{n^{p-1}} \zeta_q^{m n^{p-1}} \pmod n.$$

Per il Piccolo Teorema di Fermat, $n^{p-1} \equiv 1 \pmod p$, quindi, essendo χ un carattere di ordine p, $\chi(m)^{n^{p-1}} = \chi(m)$. Indicato con n^{-1} l'inverso moltiplicativo di n modulo q, e considerando $\chi(n^p) = \chi(n)^p = 1$ (perchè p è l'ordine di χ) si ha

$$\chi(n) = \chi(n^p n^{-(p-1)}) = \chi(n)^p \chi(n)^{-p-1} = \chi(n)^{-p-1};$$

si ottiene quindi:

$$\sum_{m=1}^{q-1} \chi(m)^{n^{p-1}} \zeta_q^{mn^{p-1}} = \sum_{m=1}^{q-1} \chi(m)\zeta_q^{mn^{p-1}} = \sum_{m=1}^{q-1} \chi(n^{-(p-1)})\chi(mn^{p-1})\zeta_q^{mn^{p-1}}$$
$$= \chi(n) \sum_{m=1}^{q-1} \chi(mn^{p-1})\zeta_q^{mn^{p-1}} = \chi(n)G(p,q).$$

Abbiamo fin qua ottenuto che

$$G(p,q)^{n^{p-1}} \equiv \chi(n)G(p,q) \pmod{n};$$

sia q^{-1} l'inverso moltiplicativo di q modulo n; moltiplicando entrambi i membri della congruenza per $q^{-1}\overline{G(p,q)}$ per il Lemma 3.3.1 si ottiene:

$$\begin{aligned} q^{-1}\overline{G(p,q)}G(p,q)^{n^{p-1}} &\equiv q^{-1}\overline{G(p,q)}\chi(n)G(p,q) \pmod{n} \\ q^{-1}qG(p,q)^{n^{p-1}-1} &\equiv q^{-1}q\chi(n) \pmod{n} \\ G(p,q)^{n^{p-1}-1} &\equiv \chi(n) \pmod{n}. \end{aligned}$$

□

Capitolo 4

L'algoritmo APR

Nel 1980 Adleman, Pomerance e Rumely idearono un test di primalità usando come punto di partenza un test ideato tre anni prima da Solovay e Strassen. L'idea fondamentale dell'APR test è stata la definizione di una sequenza di test di pseudo primalità che produce un insieme limitato il quale, nel caso in cui N sia composto, contiene certamente un divisore non banale di N. Si verifica quindi se in tale insieme è presente un divisore non banale di N, in caso affermativo si deduce che il numero testato è composto, in caso contrario si può affermare che N è primo.
Il test APR si differenzia dai precedenti test di primalità sostanzialmente per tre motivi:

- il test risolve il problema della primalità senza fornire la fattorizzazione del numero testato N;
- la sua complessità è $O(\log N^{c \log\log\log N})$, cioè quasi polinomiale in $\log N$;
- la verifica della correttezza del test necessita della conoscenza di concetti algebrici avanzati come i *caratteri di Dirichlet* o le *somme di Gauss*.

Successivamente A.K.Lenstra semplificò e riformulò il test APR, definendo un test di primalità basato sull'uso delle somme di Gauss. Lenstra ideò una prima versione, alquanto semplice nei concetti (si utilizzavano le somme di Gauss) ma difficile da implementare; in seguito, aiutato dal matematico A.Cohen, creò una versione più pratica nell'implementazione basata sulle somme di Jacobi.
In quest'elaborato verranno descritte e analizzate le versioni (deterministica e probabilistica) dell'algoritmo di Adleman, Pomerance e Rumely modificate da A.K.Lenstra, che indicheremo con la sigla APR-L.

4.1 Idee di base

In questa sezione verranno esposte le idee e i concetti basilari che portarono poi alla definizione del test di primalità APR. Il punto di partenza come già detto è

il test di primalità di Solovay-Strassen, ideato nel 1977 e che si basa sul concetto di *pseudoprimalità di Eulero*.

Definizione 4.1.1. Diciamo che $n \in \mathbb{Z}$ è uno *pseudoprimo di Eulero* in base $a \in \mathbb{N}^*$ se è composto e $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$.

Rispetto al classico concetto di *pseudoprimalità* basato sul Teorema di Eulero ([12], pag.71), non esistono interi composti N che sono pseudoprimi di Eulero rispetto ad ogni base a relativamente prima con N (ossia non esiste l'analogo dei numeri di Carmichael).
Sfruttando questa congruenza Solovay e Strassen idearono quindi un test di primalità, il cui algoritmo ha complessità polinomiale; iterando il test su k basi distinte si può provare che N è composto (se anche solo una congruenza non è verificata) oppure, se tutti i test sono superati, che la probabilità che il numero testato N sia primo è maggiore di $1 - \frac{1}{2}^k$.
L'algoritmo di Solovay-Strassen, dato in ingresso un numero intero dispari N è il seguente:

[STEP 1]
Si acquisisca l'intero k, pari al numero di test da effettuare.

[STEP 2]
Si scelga in maniera random un valore $b \in \mathbb{Z}_N^*$;
$\mathtt{k} = \mathtt{k} - 1$.

[STEP 3]
Si calcoli $\mathtt{b}^{(\mathtt{N}-1)/2}$ *e* $\left(\frac{\mathtt{b}}{\mathtt{N}}\right)$ *e si verifichi la congruenza*

$$\mathtt{b}^{(\mathtt{N}-1)/2} \equiv \left(\frac{\mathtt{b}}{\mathtt{N}}\right) \pmod{\mathtt{N}}.$$

[STEP 4]
Se la congruenza al punto 3 non vale allora si conclude che il numero N è composto, altrimenti se k è diverso da zero, si ritorna allo step 2.
Se $\mathtt{k} = 0$ *allora l'algoritmo termina e la probabilità che* N *sia primo è maggiore di* $1 - \frac{1}{2}^{\mathtt{k}}$.

L'esecuzione dell'algoritmo termina quando si è provato che N è composto oppure dopo che sono state fatte k scelte allo step 2 (con k pari appunto al numero di basi b da estrarre casualmente) e N ha sempre verificato la condizione di pseudo primalità dello step 3.
Adleman, Pomerance e Rumely partirono dal concetto di *pseudo primalità di Eulero* e crearono il loro test di primalità basandosi sulle seguenti osservazioni:

1. Si sostituiscono le congruenze del tipo

$$b^{(N-1)/2} \equiv \left(\frac{b}{N}\right) \pmod{N},$$

utilizzate nell'algoritmo di Solovay-Strassen con congruenze che si basano sulle somme di Gauss. Se anche solo una congruenza non è verificata, allora sicuramente N è composto.

2. Se N passa il punto 1, il test fornisce un insieme di interi tra i quali, se N è composto, è presente il più piccolo fattore r di N con $r \leq \sqrt{N}$. Rimane quindi da verificare se N è divisibile per almeno un elemento dell'insieme, ovviamente diverso da 1.

4.2 Algoritmo APR-L deterministico

In questa sezione viene presentato lo schema dell'algoritmo APR-L nella sua versione deterministica. Per aiutare il lettore nella comprensione dei vari passaggi si è eseguito il test su quattro valori di N sufficientemente piccoli, $N_1 = 101$, $N_2 = 111$, $N_3 = 121$ e $N_4 = 703$ e ad ogni step sono riportati i corrispondenti calcoli.

[STEP 1]
$\mathtt{I} = 1$;
$\mathtt{j} = 0$.

Il primo passo dell'algoritmo è sostanzialmente un passo base in cui vengono fissati i valori iniziali di I, prodotto dei primi iniziali, e del contatore j che tiene nota del numero di primi utilizzati nella costruzione di I.

[STEP 2]
$\mathtt{pr} = \mathtt{nextprime}(\mathtt{pr})$;
$\mathtt{j} = \mathtt{j} + 1$;
$\mathtt{I} = \mathtt{I} * \mathtt{pr}$;
Costruisci l'intero F *come prodotto dei primi* q *con* $\mathtt{q} - 1 \mid \mathtt{I}$;
se $\mathtt{F}^2 \leq \mathtt{N}$ *allora torna a STEP 2*;
Se $\gcd(\mathtt{N}, \mathtt{I} \cdot \mathtt{F}) > 1$ *e* $\gcd(\mathtt{N}, \mathtt{I} \cdot \mathtt{F})! = \mathtt{N}$ *, return 'N è composto!';*

In questo step si costruiscono essenzialmente gli interi I e F di riferimento per l'intero algoritmo; i fattori primi di I sono detti *primi iniziali* mentre i fattori primi di F si definiscono *primi di Euclide* perché sono costruiti con lo stesso metodo utilizzato per dimostrare l'infinità dei numeri primi espressa nel Teorema di Euclide. Ogni fattore primo q di F è pari infatti al prodotto di alcuni primi p_i, con $p_i \mid I$ più uno,

$$q = p_1 \cdot_2 \cdots p_k + 1.$$

$N_1 = 101$;
$pr = NP(pr) = NP(0) = 2$;
$j = j + 1 \Rightarrow j = 1$;
$I = I * pr \Rightarrow I = 2$;
$F = 2 * 3 = 6$;
$F^2 > N_1$? NO
$pr = NP(pr) = NP(2) = 3$;
$j = j + 1 \Rightarrow j = 2$;
$I = I * pr \Rightarrow I = 6$;
$F = 2 * 3 * 7 = 42$;
$F^2 > N_1$? SI
$gcd(N_1, I \cdot F) = gcd(101, 252) = 1$.

Step 2 superato con $N_1 = 101$.

$N_2 = 111$;
$pr = NP(pr) = NP(0) = 2$;
$j = j + 1 \Rightarrow j = 1$;
$I = I * pr \Rightarrow I = 2$;
$F = 2 * 3 = 6$;
$F^2 > N_2$? NO
$pr = NP(pr) = NP(2) = 3$;
$j = j + 1 \Rightarrow j = 2$;
$I = I * pr \Rightarrow I = 6$;
$F = 2 * 3 * 7 = 42$;
$F^2 > N_2$? SI
$gcd(N_2, I \cdot F) = gcd(111, 252) = 3$.
STOP

Step 2 fallito con $N_2 = 111$.

L'intero F per come è stato costruito è un numero libero dal quadrato, cioè un numero non divisibile per nessun quadrato perfetto tranne che per 1; F inoltre deve essere tale che

$$F > \sqrt{N}$$

in caso contrario si ripete il passo 2, aggiornando il valore dell'intero I e di conseguenza anche quello dell'intero F. Per valori di N minori di 10^{350} questo passo è ripetuto al più $j = 9$ volte e in tal caso $I = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$. Sempre in questo passo si calcola il *Massimo Comune Divisore* tra N e il prodotto $I \cdot F$ degli interi I e F; nel caso in cui $gcd(N, I \cdot F)$ sia maggiore di 1 e diverso da N allora si può già dichiarare che N è composto.

Lo step 2 produce i medesimi risultati con tutti i valori di N utilizzati, si ha infatti:

$$F^2 > N_4 > N_3 > N_1.$$

[STEP 3]

Per ogni primo $\mathtt{p} \mid \mathtt{I}$ *scomponi* $\mathtt{N}^{\mathtt{p}-1} - 1 = \mathtt{p}^{\mathtt{s_p}}\mathtt{u_p}$ *con* $\mathtt{u_p}$ *non divisibile per* p;
Per ogni coppia di primi p *e* q *con* $\mathtt{p} \mid \mathtt{I}$, $\mathtt{q} \mid \mathtt{F}$ *e* $\mathtt{p} \mid \mathtt{q} - 1$
calcola il più piccolo intero $\mathtt{w}(\mathtt{p}, \mathtt{q}) \leq \mathtt{s_p}$ *tale che*

$$\mathtt{G}(\mathtt{p}, \mathtt{q})^{\mathtt{p}^{\mathtt{w}(\mathtt{p},\mathtt{q})}\mathtt{u_p}} \equiv \zeta_{\mathtt{p}}^{\mathtt{j}} \pmod{\mathtt{N}} \qquad \textit{per qualche intero } j,$$

ma se nessun $\mathtt{w(p,q)}$ *soddisfa la congruenza allora, return 'N è composto!'.*

Nello step appena descritto per ogni fattore primo p di I si calcola la scomposizione di $N^{p-1}-1$ come prodotto di p_p^s e u_p, con p che non divide u_p. Il passo 3 è caratterizzato dalla verifica della condizione espressa nel Lemma 3.3.2 relativo alle somme di Gauss. Se N è un numero primo allora, proprio per tale Lemma si ha che:

$$G(p,q)^{N^{p-1}-1} = G(p,q)^{p^{s_p}u_p} \equiv \chi_{p,q}(N) \pmod{N}.$$

Se N è primo allora la congruenza presente al passo 3 è sicuramente verificata per $w(p,q) = s_p$; se invece non esiste nessun valore $w(p,q)$ tale per cui la congruenza è soddisfatta, allora si può concludere che N è composto e l'algoritmo termina.

$N_1 = 101;$
$s_1(2) = 2;$
$u_1(2) = 25;$
$s_1(3) = 1;$
$u_1(3) = 3400;$
$w_1(2,3) = 2;$
$w_1(2,7) = 2;$
$w_1(3,7) = 1.$

Step 3 superato, $N_1 = 101$

$N_3 = 121;$
$s_3(2) = 2;$
$u_3(2) = 15;$
$s_3(3) = 1;$
$u_3(3) = 4880;$
$w_3(2,3) = 1;$
$w_3(2,7) = \nexists;$
STOP

Step 3 fallito, $N_3 = 121$

$N_4 = 703;$
$s_4(2) = 1;$
$u_4(2) = 351;$
$s_4(3) = 3;$
$u_4(3) = 18304;$
$w_4(2,3) = 1;$
$w_4(2,7) = 1;$
$w_4(3,7) = 3.$

Step 3 superato, $N_4 = 703$

[STEP 4]

Per ogni primo $\mathtt{p}$ *con* $\mathtt{p} \mid \mathtt{I}$

$$\mathtt{w(p)} = \max\{\mathtt{w(p,q)} |\ \mathtt{q} \mid \mathtt{F\ e\ p} \mid \mathtt{q}-1\}$$

Per ogni coppia di primi $\mathtt{p}$ *e* $\mathtt{q}$ *con* $\mathtt{p} \mid \mathtt{I}$, $\mathtt{q} \mid \mathtt{F}$ *e* $\mathtt{p} \mid \mathtt{q}-1$ *trova un intero* $\mathtt{l(p,q)} \in [0, \mathtt{p}-1]$ *tale per cui*

$$\mathtt{G(p,q)}^{\mathtt{p}^{w(p)}\mathtt{u_p}} \equiv \zeta_{\mathtt{p}}^{\mathtt{l(p,q)}} \pmod{\mathtt{N}}.$$

Per dare un significato al quarto step dell'algoritmo è necessario fare una precisazione; l'algoritmo descritto si basa essenzialmente su due concetti fondamentali:

1. la proprietà moltiplicativa delle somme di Gauss espressa nel Lemma 3.3.2;

2. se N è composto e ha superato gli step $1-3$ dell'algoritmo allora esiste un fattore primo r di N tale che

$$r = [l^a]_F.$$

In questo passo e anche in quello successivo si calcolano quindi gli interi necessari per determinare r e sono:

- $w(p)$ uguale al massimo tra i $w(p,q)$ determinati nello step 3 al variare di q, con $q \mid F$ e $p \mid q-1$;
- $l(p,q)$, un intero compreso tra 0 e $p-1$ per cui sia verificata la congruenza

$$G(p,q)^{p^{w(p)}u_p} \equiv \zeta_p^{l(p,q)} \pmod{n};$$

  $l(p,q)$ non è altro che l'intero j per il quale è verificata la congruenza nel passo precedente (ed esso esiste sicuramente dal momento in cui il passo 3 è stato superato).

- $q_0(p)$ pari al più piccolo primo q per cui $w(p,q) = w(p)$; se ad esempio $w(p) = w(p,q_1) = w(p,q_2)$ con $q_1 < q_2$, allora $q_0(p) = q_1$.

$N_1 = 101;$ $w_1(2) = 2;$ $w_1(3) = 1;$ $l_1(2,3) = 1;$ $l_1(2,7) = 1;$ $l_1(3,7) = 1.$	$\Leftarrow$ Step 4, $N_1 = 101$ Step 4, $N_4 = 703$ $\Rightarrow$	$N_4 = 703;$ $w_4(2) = 1;$ $w_4(3) = 3;$ $l_4(2,3) = 0;$ $l_4(2,7) = 1;$ $l_4(3,7) = 0.$

[STEP 5]

$\mathtt{q_0(p)} = \min(\mathtt{q}) \mid \mathtt{w(p)} = \mathtt{w(p,q)}$;
Per ogni primo p *con* p $\mid$ I *calcola*

$$\mathtt{H} = \mathtt{G(p, q_0(p))}^{\mathtt{p^{w(p)-1}u_p}} \pmod{\mathtt{N}};$$

Al variare di j *tra* 1 *e* p, *calcola* $\gcd(\mathtt{N}, \mathtt{c}(\mathtt{H} - \zeta_{\mathtt{p}}^{\mathtt{j}}))$*: se è diverso da* 1, *return 'N è composto!'.*

Le *somme di Gauss*, per come sono state definite nel capitolo precedente, sono elementi dell'anello $\mathbb{Z}[\zeta_p, \zeta_q]$ e possono essere espresse come somme $\sum_{j=0}^{p-2}\sum_{k=0}^{q-2} a_{j,k}\zeta_p^j\zeta_q^k$.

Dire che due elementi in $\mathbb{Z}[\zeta_p, \zeta_q]$ sono congrui modulo N, significa che i corrispondenti coefficienti interi sono congrui modulo N. Sia α un elemento dell'anello $\mathbb{Z}[\zeta_p, \zeta_q] \setminus \{0\}$, si indica con $c(\alpha)$ il *Massimo Comune Divisore* dei coefficienti $a_{j,k}$.

Nello *step 5* dell'algoritmo si calcola il *Massimo Comune Divisore* tra N e $c(H - \zeta_p^j)$ al variare di j tra 0 e $p-1$ e $H = G(p, q_0(p)^{p^{w(p)-1}u_p}$. Se anche solo uno di questi valori è diverso da 1 allora N è composto e si restituisce come output la stringa 'N è composto!'.

$q_1(2) = 3$ $q_1(3) = 7$	$\Leftarrow$ Step 5.1, $N_1 = 101$ Step 5.1, $N_4 = 703$ $\Rightarrow$	$q_4(2) = 3$ $q_4(3) = 7$

$p = 2$	
$j = 1 \Rightarrow c_2(1) = 90;$	$gcd(101, 90) = 1$
$j = 2 \Rightarrow c_2(2) = 9;$	$gcd(101, 9) = 1$
$p = 3$	
$j = 1 \Rightarrow c_3(1) = 3;$	$gcd(101, 3) = 1$
$j = 2 \Rightarrow c_3(2) = 4;$	$gcd(101, 4) = 1$
$j = 3 \Rightarrow c_3(2) = 1;$	$gcd(101, 1) = 1$
Step superato!	

Step 5.2: si calcola il Massimo Comune Divisore tra $N_1 = 101$ e $c_i(j) = c(H - \zeta_i^j)$ al variare di j tra 1 e p. Per la definizione di H si veda lo Step 5 dell'algoritmo.

Step 5.2: si calcola il Massimo Comune Divisore tra $N_4 = 703$ e $c_i(j) = c(H - \zeta_i^j)$ al variare di j tra 1 e p.

$p = 2$	
$j = 1 \Rightarrow c_2(1) = 242;$	$gcd(703, 242) = 1$
$j = 2 \Rightarrow c_2(2) = 459;$	$gcd(703, 459) = 1$
$p = 3$	
$j = 1 \Rightarrow c_3(1) = 3;$	$gcd(703, 3) = 1$
$j = 2 \Rightarrow c_3(2) = 37;$	$gcd(703, 37) = 37$
STOP!	

[STEP 6]

Per ogni primo $\mathtt{q}$ *tale che* $\mathtt{q} \mid \mathtt{F}$ *trova il più piccolo generatore* $\mathtt{g_q}$ *di* $\mathbb{Z}_{\mathtt{q}}^*$. $\mathtt{l}(2) = 0$;
Per ogni primo dispari $\mathtt{q}$ *con* $\mathtt{q} \mid \mathtt{F}$ *determina un intero* $\mathtt{l}(\mathtt{q})$ *risolvendo il sistema di congruenze*

$$\mathtt{l}(\mathtt{q}) \equiv \mathtt{l}(\mathtt{p}, \mathtt{q}) \pmod{\mathtt{p}} \qquad \textit{per ogni primo } \mathtt{p} \mid \mathtt{I};$$

Determina un intero $\mathtt{l}$ *risolvendo il sistema di congruenze*

$$\mathtt{l} \equiv \mathtt{g_q}^{\mathtt{l(q)}} \pmod{\mathtt{q}} \qquad \textit{per ogni primo } \mathtt{q} \mid \mathtt{F};$$

Al variare di $\mathtt{j}$ *tra* $\mathtt{1}$ *e* $\mathtt{I}$, *se* $\mathtt{l}^{\mathtt{j}} \pmod{\mathtt{F}}$ *è un fattore non banale di* $\mathtt{N}$ *allora return 'N è composto!';*
Altrimenti, return 'N è primo!'.

Nello step 6 si costruisce l'insieme dei potenziali divisori di N, un insieme che, se N è composto, contiene il suo più piccolo fattore primo.

$N_1 = 101;$
$q = 2 \Rightarrow g_2 = 1 \pmod 2$
$q = 3 \Rightarrow g_3 = 2 \pmod 3$
$q = 7 \Rightarrow g_7 = 3 \pmod 7$

Step 6.1: si determina il generatore g_q del gruppo ciclico $\mathbb{Z}_q^*$ per ogni primo $q \mid F$.

Step 6.2.1: utilizzando il *Teorema Cinese del Resto (CRT)* si risolve il sistema di congruenze per $q = 3$.

$$\begin{cases} l_1(3) \equiv l_1(2,3) = 1 \pmod 2 \\ l_1(3) \equiv l_1(3,3) = 0 \pmod 3 \end{cases} \Rightarrow l_1(3) \equiv 3 \pmod 6$$

Step 6.2.2: Analogo allo step 6.2.1 ma con $q = 7$.

$$\begin{cases} l_1(7) \equiv l_1(2,7) = 1 \pmod 2 \\ l_1(7) \equiv l_1(3,7) = 1 \pmod 3 \end{cases} \Rightarrow l_1(7) \equiv 1 \pmod 6$$

Step 6.3: si risolve il sistema di congruenze usando il CRT.

$$\begin{cases} l_1 \equiv 1 \pmod 2 \\ l_1 \equiv 2^3 \equiv 2 \pmod 3 \\ l_1 \equiv 3 \pmod 7 \end{cases} \Rightarrow l \equiv 17 \pmod{42}$$

$j = 1 \Rightarrow gcd(17, 101) = 1$
$j = 2 \Rightarrow gcd(37, 101) = 1$
$j = 3 \Rightarrow gcd(41, 101) = 1$
$j = 4 \Rightarrow gcd(25, 101) = 1$
$j = 5 \Rightarrow gcd(5, 101) = 1$
$j = 6 \Rightarrow gcd(1, 101) = 1$
N_1 è primo!

Step 6.4: al variare di j tra 0 e I si determina $l^j \pmod F$ e si calcola il Massimo Comune Divisore tra $N_1 = 101$ e $[l^j]_F$.

□

4.3 Ulteriori dettagli dell'algoritmo

L'algoritmo APR-L deterministico è già stato descritto passo per passo nella sezione precedente; in questa sezione si dimostra la Proposizione che, assieme al Lemma 3.3.2 rappresenta l'idea chiave dell'algoritmo.

Proposizione 4.3.1. *Sia N un numero composto e si supponga che N abbia superato gli step 1-3 dell'algoritmo presentato nella Sezione 4.2; sia r il più piccolo fattore primo di N. Allora*

$$r = [l^a]_F,$$

con l intero modulo F.

Dimostrazione. Innanzitutto si vuole dimostrare che

$$p^{w(p)} \mid r^{p-1} - 1 \quad \text{per ogni primo } p \mid I.$$

Se $w(p) = 1$ allora si sa che p e r sono due numeri primi, quindi $gcd(p, r) = 1$; per il Piccolo Teorema di Fermat si ha:

$$r^{p-1} \equiv 1 \pmod{p},$$

quindi $p \mid r^{p-1} - 1$.
Assumiamo quindi $w(p) \geq 2$ e che esista qualche intero $l(p, q) \neq 0$. Allora per il Lemma 3.3.2 si ha

$$G(p, q)^{p^{w(p)} u_p} \equiv \zeta_p^{l(p,q)} \not\equiv 1 \pmod{N}$$

Infatti $l(p, q)$, per come è stato definito, è un intero il cui valore è compreso tra 0 e $p - 1$, quindi se $l(p, q) \neq 0$ allora $\zeta_p^{l(p,q)}$ è sicuramente diverso da 1 perchè ζ_p è una radice p-esima primitiva dell'unità. Essendo r un fattore di N, la congruenza è comunque verificata anche se si considera modulo r.
Elevando entrambi i membri della congruenza per p si ottiene:

$$G(p, q)^{p^{w(p)+1} u_p} \equiv \zeta_p^{l(p,q)p} \equiv 1 \pmod{N}.$$

Sia h l'ordine moltiplicativo di $G(p, q)$ modulo r; si ha quindi

$$G(p, q)^h \equiv 1 \pmod{r};$$

siccome $h \mid p^{w(p)+1} u_p$ e $h \nmid p^{w(p)} u_p$ si ha che $p^{w(p)+1} \mid h$. Ma $h \mid p(r^{p-1} - 1)$, infatti, sempre per la condizione di primalità espressa nel Lemma 3.3.2 essendo r un numero primo, la congruenza

$$G(p, q)^{r^{p-1}-1} \equiv \chi(r) \pmod{r}$$

è verificata. Se si considera quindi $G(p, q)^{p(r^{p-1}-1)}$ si ha che

$$G(p, q)^{p(r^{p-1}-1)} \equiv \chi(r)^p \equiv 1 \pmod{r},$$

perchè $\chi(r)$ è un carattere di Dirichlet modulo q e di ordine p.
Essendo h l'ordine di $G(p, q)$ modulo r deve quindi necessariamente essere che $h \mid p(r^{p-1} - 1)$. Sapendo quindi che $p^{w(p)+1} \mid h$ e $h \mid p(r^{p-1} - 1)$ si ottiene facilmente che $p^{w(p)+1} \mid p(r^{p-1} - 1)$ e quindi

$$p^{w(p)} \mid r^{p-1} - 1.$$

Si supponga ora invece che ogni intero $l(p, q)$ valga 0, allora si ha:

$$G(p, q_0)^{p^{w(p)} u_p} \equiv 1 \pmod{r}.$$

Considerando ancora h pari all'ordine di $G(p, q_0)$ modulo r si ha $p^{w(p)} \mid h$ perchè $h \mid p^{w(p)u_p}$ e $h \nmid p^{w(p)-1}u_p$. Per il Lemma 3.3.2 si ha:

$$G(p, q_0)^{r^{p-1}-1} \equiv \zeta_p^{l(p,q)} = 1 \pmod{r},$$

quindi analogamente al caso precedente si ha che $h \mid r^{p-1} - 1$ e quindi:

$$p^{w(p)} \mid r^{p-1} - 1.$$

Ora che è stato dimostrato che $p^{w(p)} \mid r^{p-1} - 1$ si può affermare che questo implica l'esistenza di due interi a e b con

$$\frac{r^{p-1} - 1}{p^{w(p)}u_p} = \frac{a}{b} \quad \text{e } b \equiv 1 \pmod{p}.$$

Si vuole quindi dimostrare il punto chiave: sia r un fattore di N, allora

$$r = [l^a]_F;$$

quindi se esiste un fattore r di N, questo di sicuro è determinato nello step 6. Prima di continuare con la dimostrazione è necessario descrivere come sono definiti gli interi $l(q)$ e l; essi vengono costruiti nello step 6 dell'algoritmo. Entrambi gli interi sono ottenuti utilizzando il Teorema Cinese del Resto (CRT) per la risoluzione di sistemi di congruenze; $l(q)$ e costruito risolvendo il sistema di congruenze

$$l(q) \equiv l(p, q) \pmod{p} \quad \text{per ogni primo } p \mid q - 1,$$

mentre l è ottenuto risolvendo il sistema di congruenze

$$l \equiv g_q^{l(q)} \pmod{q} \quad \text{per ogni primo } q \mid F.$$

Dalle definizioni di $\chi_{p,q} = \chi$ e l si ottiene

$$G(p, q)^{p^{w(p)u_p}} \equiv \zeta_p^{l(p,q)} = \zeta_p^{l(q)} = \chi(g_q^{l(q)}) = \chi(l) \pmod{r},$$

per ogni coppia di primi p e q con $q \mid F$ e $p \mid q - 1$. Dunque si ottiene:

$$\chi(r) = \chi(r)^b \equiv G(p, q)^{(r^{p-1}-1)b} = G(p, q)^{p^{w(p)}u_p a} \equiv \chi(l)^a = \chi(l^a) \pmod{r},$$

quindi:

$$\chi(r) = \chi(l^a).$$

Il prodotto dei caratteri $\chi_{p,q}$ con p primo, $p \mid I$ e $p \mid q - 1$, è un carattere χ_q di ordine $\prod_{p|q-1} p = q - 1$, perchè $q - 1 \mid I$ e I è un intero libero dal quadrato. Quindi

$$\chi_q(r) = \prod_{p|q-1} \chi_{p,q}(r) = \prod_{p|q-1} \chi_{p,q}(l^a) = \chi_q(l^a).$$

Come notato nel Capitolo 2, un carattere modulo q può essere visto come un morfismo tra $\mathbb{Z}_q^*$ e $\mathbb{C} \setminus \{0\}$, gli argomenti r e l^a sono elementi di $\mathbb{Z}_q^*$ quindi si può affermare che $r \equiv l^a \pmod{q}$, per ogni primo $q \mid F$. Essendo F un intero libero dal quadrato ne consegue che

$$r \equiv l^a \pmod{F}.$$

Essendo $F \geq \sqrt{n} \geq r$ e $F \neq r$ si ha che r è uguale al più piccolo residuo positivo di $l^a \pmod{F}$, quindi

$$r = [l^a]_F.$$

□

Capitolo 5

Implementazione APR-L deterministico

In questo capitolo si presenta il codice utilizzato nell'implementazione dell'algoritmo APR-L deteministico. Per una più facile comprensione del codice si è deciso di dedicare ad ogni step dell'algoritmo una sezione e riportare in essa le funzioni create per implementare il relativo passo dell'algoritmo. Ovviamente, essendo il primo step dell'algoritmo un passo base che non richiede l'implementazione di nessuna funzione, esso viene tralasciato. Come già fatto precedentemente si considera un valore N e si riportano i corrispondenti risultati nell'applicazione della funzione analizzata; il valore scelto è $N = 101$ che, come già dimostrato negli esempi del capitolo precedente, è un numero primo.
Il software utilizzato per l'implementazione degli algoritmi APR è *PARI/Gp*; originariamente sviluppato da Henri Cohen (lo stesso che modificò l'algoritmo APR) e dai colleghi dell'università di Bordeaux, attualmente *PARI/Gp* è un software free diffuso su licenza GPL e mantenuto aggiornato con la collaborazione di molti volontari.
PARI/Gp è uno dei più utilizzati sistemi algebrici per computer; progettato per rapidi calcoli in teoria dei numeri (fattorizzazioni, teoria algebrica dei numeri, curve ellittiche,...) contiene inoltre un ampio numero di utili funzioni per manipolare vari oggetti matematici (matrici, polinomi, serie esponenziali, numeri algebrici, ecc.) e molte funzioni trascendenti. PARI può essere utilizzato sia come collezioni di librerie da importare in programmi C (libpari) oppure come un avanzato calcolatore (Gp). In tale modalità è anche disponibile un linguaggio di scripting la cui sintassi è analoga a quella del C e risulta quindi di facile utilizzo per la maggioranza di utenti. *PARI/Gp* è diponibile su varie piattaforme (Mac OS X, MS Windows, Linux, UniX).

5.1 Step 2: Preparation Step

```
[STEP 2]
    pr = nextprime(pr);
```

$j = j + 1$;
$I = I * pr$;
Costruisci l'intero F *come prodotto dei primi* q *con* $q - 1 \mid I$;
se $F^2 \leq N$ *allora torna a STEP 2*;
Se $\gcd(N, I \cdot F) > 1$ *e* $\gcd(N, I \cdot F)! = N$ *, return 'N è composto!';*

La funzione prep.gp

```
{prep(n,pr)=
 local(j);
 \\ quando viene richiamata per la prima volta la funzione prep(n,pr), pr vale 0.
 j+=1;
 pr=nextprime(pr);
 i=i*pr;
 f=1;
 forprime(q=2,i+1,
   if (i%(q-1)==0,
       f*=q));
 if(sqr(f)<=n,
   prep(n,pr+1));
 g=[i,f];
 if (gcd(n,(i*f))>1  && gcd(n,(i*f))!= n,
       stringa="composto";
       return (stringa));
}
```

La funzione *prep* svolge esattamente quanto descritto nello step 2, restituisce la stringa 'composto' solo nel caso in cui il Massimo Comune Divisore tra N e il prodotto $I \cdot F$ sia diverso da 1 e da N; nel caso invece in cui la condizione sia verificata la funzione termina senza dare un output in uscita. Sono state però create due variabili globali i e f (tutto ciò che non è dichiarato locale con il comando *local*, è globale) che rappresentano rispettivamente i valori I e F descritti nell'algoritmo.

```
gp > n = 101
%2 = 101
gp > prep(n,pr)
gp > i
%3 = 6
gp > f
%4 = 42
gp > n= 111
%5 = 111
gp > prep(n,pr)
% 6 = 'composto!'
```

5.2 Step 3: Probable-Prime Computation

Prima di cominciare a descrivere le funzioni che caratterizzano questo passo dell'algoritmo è utile una precisazione; per comodità nell'implementazione e per evitare di ripetere più volte gli stessi calcoli si è deciso di far uso di matrici per la memorizzazione di alcuni valori ricavati durante l'esecuzione delle funzioni. Tali matrici hanno lunghezza pari al numero di primi che dividono I e larghezza pari al numero di primi che dividono F. Per calcolare lunghezza e larghezza della matrici si è utilizzato una funzione (**omega**) predefinita in Gp; *omega (x)* restituisce appunto il numero di divisori primi di x.

Le matrici create sono le seguenti:

- **matr_w**: matrice in cui sono memorizzati i valori $w(p,q)$ definiti nel terzo step dell'algoritmo;
- **matr_lpq** : matrice in cui sono memorizzati i valori $l(p,q)$ definiti nel quarto step dell'algoritmo;
- **matr_gauss**: matrice contentente le somme di Gauss $G(p,q)$; si evita quindi di ripetere lo stesso calcolo in più punti del programma;
- **matr_su**: matrice in cui sono memorizzati i valori del tipo $G(p,q)^u$ con u ottenuto dalla scomposizione di $N^{p-1}-1$ (terzo step dell'algoritmo). Si è reso necessario memorizzare tale valore perchè, come si vedrà successivamente, è un calcolo che richiede molto tempo e una notevole disponibilità di memoria.

L'idea è quella di associare ad ogni riga un primo p tale che $p \mid I$ e ad ogni colonna un primo q con q fattore primo di F. Considerando quindi i dati dell'esempio di partenza con $N = 101$ e $I = 6 = 2*3$, $F = 42 = 2*3*7$ le matrici descritte in questa sezione hanno due righe e tre colonne; ad esempio matr_lpq $[1,2] = l(2,3)$, matr_w$[2,3] = w(3,7)$.

[STEP 3]
Per ogni primo $\mathtt{p} \mid \mathtt{I}$ *scomponi* $\mathtt{N}^{\mathtt{p}-1} - 1 = \mathtt{p}^{\mathtt{s_p}}\mathtt{u_p}$ *con* $\mathtt{u_p}$ *non divisibile per* p*;*
Per ogni coppia di primi p *e* q *con* $\mathtt{p} \mid \mathtt{I}$, $\mathtt{q} \mid \mathtt{F}$ *e* $\mathtt{p} \mid \mathtt{q}-1$
calcola il più piccolo intero $\mathtt{w(p,q)} \leq \mathtt{s_p}$ *tale che*

$$\mathtt{G(p,q)}^{\mathtt{p}^{\mathtt{w(p,q)}}\mathtt{u_p}} \equiv \zeta_{\mathtt{p}}^{\mathtt{j}} \pmod{\mathtt{N}} \qquad \textit{per qualche intero } j,$$

ma se nessun $\mathtt{w(p,q)}$ *soddisfa la congruenza allora, return 'N è composto!'.*

La funzione scomp

```
{scomp(n,p) =
 local (u,s);
 u=n^(p-1)-1;
 s=0;
 while(u%p == 0,
       u= u\p;
       s=s+1);
 vett_scomp=[p,s,u];
 return (vett_scomp);
}
```

Applicando la funzione *scomp* ad N e p ciò che si ottiene è un vettore composto da tre celle in cui sono memorizzati rispettivamente il primo p utilizzato per la scomposizione, l'esponente s e il fattore u non divisibile per p. Si ricorda infatti che $N^{p-1}-1$ viene scomposto in $p^s u$.

```
gp > n = 101
%22 = 101
gp > scomp (n,2)
%23 = [2,2,25]
gp > scomp (n,3)
%24 = [3,1,3400]
```

La funzione gsum

```
{gsum (p,q)=
 local(k,a,b,coeffzetap,coeffzetaq,S);
 print("Computation of the coefficients of a Gauss sum with p= ", p , " and q= ",q);
 if ((q-1)%p != 0,
       error("p does not divide q-1"));
 coeffzetap=vector(q-1);
 coeffzetaq=vector(q-1);
 gq=znprimroot(q);
 polp=polcyclo(p,x);
 polq=polcyclo(q,y);

 ZETAp=Mod(x,polp);
 ZETAq=Mod(y,polq);

 for (k=1, q-1,
       a=k%p;
       b=lift(Mod(gq^k,q));
       coeffzetap[k]=a;
       coeffzetaq[k]=b);
 S=0;
 for (k=1,q-1,
       S +=(ZETAp^coeffzetap[k])*(ZETAq^coeffzetaq[k]));
 return(S);
}
```

La funzione *gsum* come si può intuire dal nome è la funzione che calcola la somma di Gauss così come definita nel secondo capitolo. La somma di Gauss associata ai primi p e q vale:

$$G(p,q)=\sum_{k=1}^{q-1}\zeta_p^k\zeta_q^{g_q^k}.$$

Analizziamo ora la funzione gsum:

- gq rappresenta il più piccolo generatore in $\mathbb{Z}_q^*$; viene determinato mediante la funzione *znprimroot* (predefinita in Gp) che restituisce come output un elemento $[m]$ del quoziente $\mathbb{Z}_n$ indicato con *Mod(m,n).*

- *polcyclo(p,x)* è un'altra funzione predefinita di Gp e restituisce il p-esimo polinomio ciclotomico valutato in x; un polinomio ciclotomico è un polinomio le cui radici sono tutte le radici primitive dell'unità.
  L'idea è quella di definire le due radici primitive ζ_p e ζ_q come elementi del quoziente $\mathbb{Q}[x]/ < polcicl >$ definendole rispettivamente come *Mod(x, polp)* e *Mod (y,polq).*

- i coefficienti di ζ_p e ζ_q sono costruiti seguendo la definizione di somma di Gauss e quindi considerando rispettivamente il modulo della divisione tra k e p e tra g_q^k e q al variare di k tra 1 e $q-1$.

```
gp > gq=znprimroot(2)
%22 = Mod(1,2)
gp > gq=znprimroot(3)
%23 = Mod(2,3)
gp > gq=znprimroot(7)
%24 = Mod(3,7)
gp > polp=polcyclo(2,x)
%25= x+1
gp > polq=polcyclo(7,y)
%26= y^6 + y^5 + y^4 + y^3 + y^2 + y + 1
```

```
gp > gsum(2,3)
Computation of the coefficients of a Gauss sum
with p=2 and q=3
%27 = Mod(Mod(2*y + 1, y^2 + y + 1), x + 1)
gp > gsum (2,7)
Computation of the coefficients of a Gauss sum
with p=2 and q=7
%28 = Mod(Mod(2*y^4 + 2*y^2 + 2*y + 1,
y^6 + y^5 + y^4 + y^3 + y^2 + y + 1), x + 1)
```

Esempi di applicazioni di funzioni in Gp.

La funzione gauss_modn

```
{gauss_modn(S,p,q)=
 local(a,b,c,d,l,j,S1,polp,polq);
 polp=polcyclo(p,x);
 polq=polcyclo(q,y);
 a=Vec(lift(S));
 l=matsize(a);
 l=l[2];
 for (j=1,l,
       b=Vec(lift(a[j]));
       c=Mod(b,n);
       d=Pol(c,y);
       a[j]=Mod(d,polq));
 S1=Mod(Pol(a),polp);
 return(S1);
}
```

La funzione *gauss_modn* è una funzione che dato in ingresso una somma di Gauss S da in uscita una somma di Gauss $S1$ i cui coefficienti sono classi resto modulo N. Una somma di Gauss $G(p,q)$ è infatti un elemento dell'anello $\mathbb{Z}[\zeta_p,\zeta_q]$ e può essere espressa come somma $\sum_{j=0}^{p-2}\sum_{k=0}^{q-2} a_{j,k}\zeta_p^j\zeta_q^k$. Quindi, ciò che in effetti esegue la funzione *gauss_modn* è proprio considerare i singoli coefficienti $a_{j,k}$ e valutarli modulo N. Per fare ciò sono state utilizzate alcune funzioni predefinite in Gp e sono:

- *lift*:consente di passare da un elemento dello spazio $\mathbb{Z}_n$ ad uno di $\mathbb{Z}$, o da un elemento dello spazio $\mathbb{Z}[x]/n\mathbb{Z}[x]$ a uno di $\mathbb{Z}[x]$.
- *Vec(x)*: trasforma l'oggetto x in un vettore; generalmente l'oggetto x su cui è applicata questa funzione è un polinomio.

- *Pol(c,v=x)*: trasforma l'oggetto c (generalmente un vettore) in un polinomio di varibile v partendo dal coefficiente di grado massimo.

- *matsize (x)*: restituisce un vettore di due celle contenente il numero di righe e di colonne della matrice o del vettore x.

```
gp > lift(Mod(2,3))
%29 = 2
gp > lift(Mod(2*y+1, y^2+y+1))
%30 = 2*y+1
gp > a=2*y+1
%31 = 2*y+1
gp > Vec(a)
%32 = [2,1]
gp > a=y^6+y^5+3y^4+y^2+y-1
%33 = y^6+y^5+3y^4+y^2+y-1
gp > Vec(a)
%34 = [1,1,3,0,1,1,-1]
```

```
gp > matsize(Vec(a))
%35 = [1, 7] gp > Pol(Vec(a))
%36 = y^6+y^5+3y^4+y^2+y-1
gp > S=gsum(2,3)
Computation of the coefficients of a Gauss sum
with p=2 and q=3
%37 = Mod(Mod(2*y+1, y^2+y+1),
x+1)
gp > n=101 %38 = 101 gp > gauss_modn(S,2,3)
%39 = Mod(Mod(Mod(2, 101)*y+
Mod(1, 101), y^2+y+y), x+1)
```

Esempi di applicazioni di funzioni in Gp.

La funzione new_a

```
new_a(S1,q,vett,ri,co)=
local(a,b,c,e,j,k,q,w,check,p,u,m);
p=vett[1];
u=vett[3];
check=0;
vector_root(p);
bs=quadrati_ripetuti(S1,u),
matr_su[ri,co]=bs;
e=valuta_S(bs);
for (j=1,p,
      if (e==vett_root[j],
            check=1;
            matr_w[ri,co]=k;
            j=p+1));

for (k=1,vett[2],
      bs=quadrati_ripetuti(bs,p);
      e=valuta_S(bs);
    for (j=1,p,
            if (e==vett_root[j],
                  check=1;
                  matr_w[ri,co]=k;
                  k=vett[2]+1;
                  j=p+1)));

if (check==0,
      stringa="composto";
      return (stringa));
}
```

La funzione *new_a* è una delle funzioni principali del programma; attraverso l'applicazione di tale funzione si verifica la condizione di primalità espressa nel Lemma 3.3.2 che, come già detto più volte, rappresenta una delle idee chiave dell'algoritmo APR-L deterministico. L'obiettivo è quello di confrontare due oggetti: una somma di Gauss elevata ad una certa potenza e una radice primitiva dell'unità; questi due elementi appartengono però ad anelli differenti (le somme di Gauss appartengono a $\mathbb{Z}[\zeta_p, \zeta_q]$ mentre le radici p-esime primitive appartengono a $\mathbb{Z}[\zeta_p]$), quindi è stato necessario definire una funzione (*valuta_S*) in grado di rendere la somma di Gauss fornita in ingresso, un elemento confrontabile con le potenze della radice ζ_p (nel codice indicata con *zp*) definita attraverso la funzione *primitiveroot*. Oltre alla funzione *new_a* sono state definite altre tre funzioni, il cui codice è riportato di seguito:

1. *vector_root(p)*: funzione che crea un vettore di p celle contenenti valori del tipo ζ_p^j con j compreso tra 0 e $p-1$. La radice p-esima primitiva(*zp*) è definita richiamando la funzione *primitiveroot*.

2. *quadrati_ripetuti(S,u)*: funzione che calcola S^u. L'algoritmo per il calcolo delle potenze attraverso il metodo dei quadrati ripetuti è molto utile quando si devono eseguire calcoli modulo un numero molto grande N, basta far seguire ad ogni operazione di somma o di prodotto il calcolo del resto modulo N. Nel nostro caso però è stata definita la funzione che realizza il calcolo della potenza con il metodo dei quadrati ripetuti classico; infatti, ciò che viene passato come 'base' è una somma di Gauss ed essa è gia definita come un oggetto che vive in un quoziente (ogni coefficiente è già considerato modulo N). Non appena si effettua un prodotto di un polinomio per un oggetto che e' definito vivere in un quoziente, quello che si ottiene è un elemento del quoziente stesso.

3. *valuta_S(a)*: data in ingresso una somma di Gauss i cui coefficienti sono già stati ridotti modulo N, restituisce in uscita un oggetto (vettore) confrontabile con gli elementi del vettore creato nella funzione *vector_root*.

Le funzioni vector_root e primitiveroot

```
{vector_root(p)=
 local(j,q,w,zp);
 vett_root=vector(p);
 zp=primitiveroot(p);
 for(j=0,p-1,
       q=zp^j;
       w=Mod(Vec(lift(q)),n);
       vett_root[j+1]=w);
}
```

```
{primitiveroot (p)=
 local (polp,zp);
 polp=polcyclo(p);
 zp=Mod(x,polp);
 return(zp);
}
```

La funzione quadrati_ripetuti

```
{quadrati_ripetuti(a,m)=
 local(P,M,A,r);
 P=1;
 M=m;
```

```
A=a;
while (M>0,
   r=M%2;
   if (r==1,
      P=P*A);
   A=A^2;
   M=ceil(M\2));
return (P);
}
```

gp > vector_root(2)	gp > S1=gauss_modn(S,2,3)
gp > vett_root	%6 = Mod(Mod(Mod(2, 101) * y+
%2 = [[Mod(1,101)],[Mod(100,101)]]	Mod(1, 101), y^2 + y + y), x + 1)
gp > vector_root(3)	gp >
gp > vett_root	a=quadrati_ripetuti(S1,n)
%3 =[[Mod(1, 101)], [Mod(1, 101), Mod(0, 101)],	%7 = Mod(Mod(Mod(99, 101) * y+
[Mod(100, 101), Mod(100, 101)]]	Mod(100, 101), y^2 + y + y), x + 1)
gp > S=gsum(2,3)	gp > valuta_S(a)
Computation of the coefficients of a Gauss sum	%8 = [Mod(98,101)]
with p=2 and q=3	
%4 = Mod(Mod(2 * y + 1, y^2 + y + 1),	
gp > n=101 %5 = 101	

Esempi di applicazioni di funzioni in Gp.

La funzione valuta_S

```
{valuta_S(a)=
 local(b,e);
 b=lift(lift(a));
 y=1;
 e=Mod(Vec(eval(b)),n);
 y='y;
 return(e);
}
```

5.3 Step 4: Maximal Order Search

[STEP 4]

Per ogni primo p *con* $\mathtt{p} \mid \mathtt{I}$

$$\mathtt{w}(\mathtt{p}) = \max\{\mathtt{w}(\mathtt{p},\mathtt{q}) \mid \ \mathtt{q} \mid \mathtt{F} \text{ e } \mathtt{p} \mid \mathtt{q}-1\}$$

Per ogni coppia di primi p *e* q *con* $\mathtt{p} \mid \mathtt{I}$, $\mathtt{q} \mid \mathtt{F}$ *e* $\mathtt{p} \mid \mathtt{q}-1$ *trova un intero* $\mathtt{l}(\mathtt{p},\mathtt{q}) \in [0, \mathtt{p}-1]$ *tale per cui*

$$\mathtt{G}(\mathtt{p},\mathtt{q})^{\mathtt{p}^{\mathtt{w}(\mathtt{p})}\mathtt{u}_\mathtt{p}} \equiv \zeta_\mathtt{p}^{\mathtt{l}(\mathtt{p},\mathtt{q})} \pmod{\mathtt{N}}.$$

Per realizzare questo passo dell'algoritmo si sono utilizzate, oltre alla funzioni già create, due nuove funzioni:

1. *create_matrice_info*: viene creata una matrice di *lung* righe e 5 colonne, dove *lung* è pari al numero di primi che dividono l'intero I. Nelle 5 colonne sono memorizzati rispettivamente:

   - il primo p che divide I;
   - $w(p) = max\{w(p,q)\}$;
   - $q(p) = min\{q \mid w(p) = w(p,q)\}$;
   - $G(p, q(p))$;
   - $G(p, q(p))^u$.

   I valori delle ultime due colonne, come anticipato precedentemente, sono memorizzzati con lo scopo di evitare di ripetere calcoli che, per le dimensioni dei dati coinvolti, appesantirebbero ulteriormente il programma. Osservando il codice della funzione si può notare la presenza della funzione *vettore_q*: applicandola sugli interi I e F crea un vettore contenente tutti i primi q che dividono l'intero F.

2. *new_new_a*: simile alla funzione *new_a* provvede a determinare e memorizzare nella corripondente matrice (matr_lpq) un intero $l(p,q) \in [0, p-1]$ tale per cui
$$G(p,q)^{p^{w(p)}u_p} \equiv \zeta_p^{l(p,q)} \pmod{N}.$$

La funzione create_matrice_info

```
{create_matrice_info (matrice,i,f)=
 local(a,b,p);
 a=0;
 b=0;
 matr_info=matrix(lung, 5);
 vettore_q(i,f);
 print("il vettore q e'", vett_q);
 forprime(p=2,i,
       if(i%p==0,
          a+=1;
          matr_info[a,1]=p));
 for (a=1,lung,
          matr_info [a,2]=vecmax(matrice[a,]));
 for (a=1, lung,
       for (b=1, larg,
             if (matrice[a,b]== matr_info [a,2],
                   matr_info[a,3]=vett_q[b];
                   matr_info [a,4]=matr_gauss[a,b];
                   matr_info [a,5]=matr_su[a,b];
                   b=larg+1)));
}
```

La funzione new_new_a

```
{new_new_a(S,p,s,u,ri,co)=
 local(a,b,bs,e,j,k,q,w,check);
 vector_root(p);
```

```
 bs=matr_su[ri,co];
 a=quadrati_ripetuti(bs,(p^s));
 e=valuta_S(a);
 for (j=1,p,
      if (e==vett_root[j],
               matr_lpq[ri,co]=(j-1);
               j=p+1));
}
```

La funzione vettore_q

```
{vettore_q (i,f) =
 local (b,p);
 b=1;
 vett_q= vector(larg);
 vett_q[1]=2;
 p=2;
 forprime (q=3,i+1,
      if (f%q==0,
            b+=1;
            vett_q[b] =q));
}
```

5.4 Step 5: Coprime Check

[STEP 5]

$\mathtt{q_0(p)} = \min(\mathtt{q}) \mid \mathtt{w(p)} = \mathtt{w(p,q)}$;

Per ogni primo p *con* p $\mid$ I *calcola*

$$\mathtt{H} = \mathtt{G(p, q_0(p))}^{\mathtt{p^{w(p)-1}u_p}} \pmod{\mathtt{N}};$$

Al variare di j *tra* 1 *e* p*, calcola* $\gcd(\mathtt{N}, \mathtt{c}(\mathtt{H} - \zeta_{\mathtt{p}}^{\mathtt{j}}))$*: se è diverso da* 1*, return 'N è composto!'.*

La funzione coprime_check

```
{coprime_check(p,s,ri)=
 local (H,b,e,m,w,j,S1);
 S1=matr_info[ri,5];
 H=quadrati_ripetuti(S1,p^(s-1));
 e=valuta_S(H);
 \\in questo momento ho g^(p^(w(p)-1)*up mod n
 vector_root(p);
 a=Pol(lift(e));
 \\print("a vale ",a);
 for (j=1,p,
      b=Pol(lift(vett_root[j]));
      c=a-b;
      d=gcd(c);
      if (gcd(n,d)>1,
            stringa="composto";
            return (stringa)));
}
```

Il quinto passo dell'algoritmo è realizzato attraverso la funzione *coprime_check*; in tale funzione viene determinato H sfruttando quanto memorizzato precedentemente (i valori della quinta colonna della matrice matr_info) e utilizzando nuovamente la funzione *quadrati_ripetuti* per il calcolo dell'elevamento a potenza.

Anche in questo caso per consentire il confronto e le operazioni tra la somma di Gauss H e le varie potenze della radice p-esima primitiva si è ricorsi all'uso della funzione *valuta_S* e delle funzioni predefinite *lift* e *Pol*.

```
gp > n = 101
%22 = 101
gp > scomp (n,2)
%23 = [2,2,25]
gp > coprime_check(2,vett_scomp[2],1)
gp > scomp (n,3)
%24 = [3,1,3400]
gp > coprime_check(vett_scomp[1],vett_scomp[2],2)
gp >
```

Esempi di applicazioni di funzioni in Gp.

5.5 Step 6: Divisor Search

[STEP 6]

Per ogni primo q *tale che* $\mathtt{q} \mid \mathtt{F}$ *trova il più piccolo generatore* $\mathtt{g_q}$ *di* $\mathbb{Z}_{\mathtt{q}}^*$.
$\mathtt{l}(2) = 0$;
Per ogni primo dispari q *con* $\mathtt{q} \mid \mathtt{F}$ *determina un intero* $\mathtt{l}(\mathtt{q})$ *risolvendo il sistema di congruenze*

$$\mathtt{l}(\mathtt{q}) \equiv \mathtt{l}(\mathtt{p}, \mathtt{q}) \pmod{\mathtt{p}} \qquad \textit{per ogni primo } \mathtt{p} \mid \mathtt{I};$$

Determina un intero l *risolvendo il sistema di congruenze*

$$\mathtt{l} \equiv \mathtt{g_q}^{\mathtt{l}(\mathtt{q})} \pmod{\mathtt{q}} \qquad \textit{per ogni primo } \mathtt{q} \mid \mathtt{F};$$

Al variare di j *tra* 1 *e* I, *se* $\mathtt{l}^{\mathtt{j}} \pmod{\mathtt{F}}$ *è un fattore non banale di* N *allora return 'N è composto!';*
Altrimenti, return 'N è primo!'.

La funzione divisor_search

```
{divisor_search (matr_lpq,i)=
 local(a,c,stringa);
 matr_lpq=matrice_modp(matr_lpq,matr_info);
 vettore_lq=vector(larg);
 vettore_lq[1]=0;
 vettore_gq= vector(larg);
 vettore_gqlq= vector(larg);
 for (c=2,larg,
```

```
        vettore_lq[c]=chinese(matr_lpq[,c]));
 print ("il vettore_lq è ", vettore_lq);
 for (a=1, larg,
        vettore_gqlq[a]= Mod(vettore_gq[a])^lift (vettore_lq[a]),vett_q[a]));
 print("il vettore_gqlq è", vettore_gqlq);

 l=chinese (vettore_gqlq);
 print(l);
 if(gcd(lift(l),n)!=1 && gcd (lift(l),n)!=n,
        stringa="composto";
        return(stringa));
}
```

Nella funzione *divisor_search* si calcolano i possibili fattori di N nel caso esso sia un numero composto e abbia superato gli step precedenti. Per fare ciò si è ricorso all'uso di vettori di lunghezza pari al numero di primi q che dividono F, sfruttando la comodità di eseguire le operazioni tra di essi. Nella funzione *divisor_search* si identificano i seguenti vettori:

- **vettore_gq**: ogni cella contiene il generatore del gruppo ciclico $\mathbb{Z}_q^*$, con q fattore primo di F; gli elementi gq sono ricavati richiamando la funzione predefinita *znprimroot*.

- **vettore_lq**: vettore che contiene i valori $l(q)$ ottenuti risolvendo il sistema di congruenze come indicato nello step 6. Per risolvere il sistema di congruenze si utilizza il Teorema Cinese del Resto; analogamente per trovare i valori $l(q)$ si utilizza la funzione *chinese*, predefinita in Gp, che calcola appunto la soluzione del sistema di congruenze. Come input della funzione si fornisce quindi l'opportuna colonna della matrice *matr_lpq* contenente i valori $l(p, q)$ calcolata nello step 4 attraverso la funzione *new_new_a*.

- **vettore_gqlq**: contiene valori del tipo $g_q^{l(q)} \mod q$ ottenuti operando sui due vettori appena descritti.

La funzione *matr_modp* converte gli elementi (interi) della matrice in ingresso in classi resto modulo p con p fattore primo di I; in base alla definzione delle matrici data nella Sezione 5.2, questa è un'operazione che va effettuata riga per riga (ad ogni riga è associato un primo p diverso, con $p \mid I$).

La funzione matrice_modp

```
{matrice_modp(matr_l_pr,matr_info)=
 local(a,b);
 matr_modp=matrix(lung,larg);
 for(a=1,lung,
        matr_modp[a,]= Mod(matr_l_pr[a,],matr_info[a,1]));
 return(matr_modp);
}
```

```
gp > matr_info=[2,3,3;3,1,7]
%30 =
[2 3 3]
[3 1 7]
gp > matr_a=[0,1,6;0,2,3]
%31 =
[0 1 6]
[0 2 3]
gp > matr=
matrice_modp(matr_a,matr_info)
%32 =
[Mod(0,2) Mod(1,2) Mod(1,2)]
[Mod(0,3) Mod(1,3) Mod(1,3)]
```

```
gp > vett_l=vector(3)
%33 = [0,0,0]
gp > vett_l[1]=chinese(matr_b[,1])
%34 = Mod(0,6)
gp > vett_l[2]=chinese(matr_b[,2])
%34 = Mod(1,6)
gp > vett_l[3]=chinese(matr_b[,3])
%34 = Mod(1,6)
```

Esempi di applicazioni di funzioni in Gp.

5.6 La funzione principale: Union

La funzione union

```
{union(n)=
 local(vett_p,j,p,k,prov);
 i=1;
 pr=0;
 stringa="";
 print ("2) PREPARATION STEP");
 stringa=prep(n,pr);
 if (stringa=="composto",
       print ("RISULTATO");
       print ("n è composto!");
       break);
 print("I vale ",i);
 print("F vale ",f);

\\ creo matrici e vettori che mi serviranno di seguito
 lung=omega(i);
 larg=omega(f);
 matr_w=matrix(lung,larg);
 matr_lpq=matrix(lung,larg);
 matr_gauss=matrix(lung,larg);
 vett_p=vector(lung);
 matr_su=matrix(lung,larg);

 j=1;
 print ("");
 print ("3) PROBABLE-PRIME COMPUTATION");

 forprime (p=2,i,
       if (i%p==0,
             vett_p[j]=scomp(n,p);
             j+=1));
 print(vett_p);

 j=0;k=0;
 forprime (p=2,i,
```

```
      if (i%p==0,
            j+=1;
        vett_prov=vett_p[j]; \\vettore = vett_scomp
            forprime(q=2,i+1,
                  if (f%q==0,
                        k+=1;
                        if ((q-1)%p==0,
                              S=gsum(p,q); \\somma di Gauss con i valori di p e q
                              matr_gauss[j,k]=S;
                              S1=gauss_modn(S,p,q);
                     stringa=new_a(S1,q,vett_prov,j,k))));
                              if (stringa=="composto",
                                    print ("RISULTATO");
                                    print ("n è composto!");
                                    break(2););
            k=0));
 print("la matrice matr_w è ",matr_w);
 print ("");
 print ("4) MAXIMAL ORDER SEARCH");
 create_matrice_info(matr_w,i,f);
 j=0; k=0;
 forprime(p=2,i,
      if(i%p==0,
            j+=1;
            vett_prov=vett_p[j]; \\vettore = vett_scomp
            forprime(q=2,i+1,
                  if (f%q==0,
                        k+=1;
                        if ((q-1)%p==0,
                              S=matr_gauss[j,k];
                              S1=gauss_modn(S,p,q);
                        new_new_a(S1,p,matr_info[j,2],vett_prov[3],j,k))));
            k=0));

 print("la matrice l(p,q) vale ", matr_lpq);


 print ("");
 print ("5) COPRIME_CHECK");

 for(j=1,lung,
      prov=vett_p[j];
      stringa=coprime_check(matr_info[j,1],matr_info[j,3],matr_info[j,2],j);
 if (stringa=="composto",
      print ("RISULTATO");
      print ("n è composto!");
      break(2),
      print("Step COPRIME CHECK superato!")));

 print ("");
 print ("6) DIVISOR_SEARCH");
 stringa=divisor_search(matr_lpq,i);
 print ("");
 print ("RISULTATO");
 if (stringa=="composto",
      stringa1="n è composto!";
      print(stringa1),
      stringa1="n è primo!";
     print(stringa1));
}
```

```
gp > n=101
%21 = 101
gp > union(n)
2) PREPARATION STEP
I vale 6
F vale 42
3) PROBABLE-PRIME COMPUTATION
[[2, 2, 25], [3, 1, 3400]]
Computation of the coefficients of a Gauss sum with p= 2 and
q= 3
Computation of the coefficients of a Gauss sum with p= 2 and
q= 7
Computation of the coefficients of a Gauss sum with p= 3 and
q= 7
la matrice matr_w è [0, 2, 2; 0, 0, 1]

4) MAXIMAL ORDER SEARCH
il vettore q e'[2, 3, 7]
la matrice l(p,q) vale [0, 1, 1; 0, 0, 1]

5) COPRIME_CHECK
Step COPRIME CHECK superato!
Step COPRIME CHECK superato!

6) DIVISOR_SEARCH
la matrice lpq mod p è stata creata
il vettore_lq è [0, Mod(3,6), Mod(1,6)]
il vettore_gqlq è[Mod(1,2), Mod(2,3), Mod(3,7)]
Mod(17,42)

RISULTATO
n è primo!
```

La funzione union in Gp

5.7 Osservazioni

La difficoltà iniziale nell'implementazione dell'algoritmo APR-L deterministico consiste nel capire come definire e trattare correttamente oggetti complessi come le somme di Gauss definite in un anello ($\mathbb{Z}[\zeta_p, \zeta_q]$) in cui non si è abituati a lavorare. Fortunatamente Gp mette a disposizione un gran numero di funzioni predefinite, che permettono di agevolare e snellire la stesura del codice, e una guida ben strutturata in cui poter trovare tutte le informazioni necessarie.
Il collo di bottiglia dell'algoritimo è rappresentato dal calcolo dell'elevamento a potenza $G(p,q)^u$ con u definito nella scomposizione di $N^{p-1}-1$ in $p^s u$ con $p \nmid u$. Infatti u è all'incirca dello stesso ordine di grandezza di N^{p-1} quindi, ad esempio, con un numero N di 20 cifre e $p = 7$ l'esponente u è un numero con più di 100 cifre decimali.

Questo è il motivo per cui l'algoritmo APR-L è ritenuto importante per i concetti utilizzati nel determinare la primalità di un numero, ma la cui implementazione è difficile e comunque poco efficiente; fu per questo infatti che A.K.Lenstra e Cohen idearono un test di primalità basato sulle somme di Jacobi, elementi che, a differenza delle somme di Gauss, sono definiti nell'anello $\mathbb{Z}[\zeta_p]$ e quindi più facili da trattare.

Capitolo 6

L'algoritmo APR-L probabilistico

In questo capitolo ci si occupa di descrivere completamente la versione probabilistica dell'argoritmo APR-L; così come già fatto per la versione deterministica il punto di partenza resta sempre la dimostrazione della validità algebrica dei concetti utilizzati. Si passa quindi a descrivere l'algortimo vero e proprio affiancando alla descrizione di ogni step un esempio per facilitare la comprensione dei concetti esposti; infine si riporta e descrive il codice utilizzato nell'implementazione.

6.1 Aspetti algebrici

Le versioni (deterministica e probabilistica) dell'algoritmo di Adleman, Pomerance e Rumely modificate da A.K.Lenstra si basano principalmente sui concetti di *somme di Gauss* e *caratteri di Dirichlet* e sulle proprietà a loro associate descritte nei primi capitoli di quest'eleborato.
Si ricorda la definizione di *somma di Gauss*:

Definizione 6.1.1. Siano p e q due numeri primi tali che che $p \mid q-1$ e sia χ un carattere di Dirichlet di ordine p e modulo q.
La somma di Gauss $\tau(\chi)$ associata al carattere χ è pari a

$$\tau(\chi) = \sum_{m=1}^{q-1} \chi(m)\zeta_q^m.$$

La somma di Gauss come già ricordato precedentemente è un elemento dell'anello $\mathbb{Z}[\zeta_p, \zeta_q]$; tale anello verrà d'ora in poi indicato con R.

Lemma 6.1.2. *Sia χ un carattere di Dirichlet di ordine p e modulo q, con p e q due numeri primi; sia $\tau(\chi)$ la somma di Gauss associata al carattere χ. Sia n un numero intero tale che $gcd(n, pq) = 1$; se n è primo allora*

$$(\tau(\chi))^n \equiv (\chi(n))^{-n}\tau(\chi^n) \pmod{n}.$$

Dimostrazione. Se n è un numero primo, allora si ha che

$$(a+b)^n \equiv a^n + b^n \pmod{n} \quad \forall a, b, \in \mathbb{Z}_n^*,$$

quindi si ottiene facilmente che:

$$(\tau(\chi))^n \equiv \sum_{x=1}^{q-1} (\chi(x))^n \zeta_q^{nx} \equiv \tau((\chi)^n)(\chi(n))^{-n} \pmod{n}.$$

□

L'idea è quella di costruire un test di primalità che sfrutti le due seguenti condizioni:

- Esiste $\eta(\chi) \in \langle \zeta_p \rangle$ tale che

$$(\tau(\chi))^n \equiv (\eta(\chi))^{-n} \tau(\chi^n) \pmod{nR}. \tag{6.1}$$

  Questa condizione deve essere verificata per ogni coppia di primi p e q.

- Per ogni primo $r|n$ si ha

$$\nu_p(r^{p-1} - 1) \geq \nu_p(n^{p-1} - 1), \tag{6.2}$$

  dove con $\nu_p(m)$ s'intende il numero di fattori p in m.

Se la condizione 6.2 vale per ogni primo $r \mid n$ allora chiaramente vale per ogni $r \mid n$. Inoltre si può definire l'intero $l_p(r)$ come

$$l_p(r) = \frac{r^{p-1} - 1}{n^{p-1} - 1} \pmod{p} \quad \forall r \mid n.$$

Si ha:

$$l_p(r) = l_p(r_1) + l_p(r_2) \quad \text{con } r = r_1 r_2 \quad \text{e} \quad r \mid n,$$

e

$$l_p(n) = 1.$$

Infatti, sia $r = r_1 r_2$ un divisore di n e siano r_1 e r_2 numeri primi. Allora

$$\begin{aligned}(r_1^{p-1} - 1)(r_2^{p-1} - 1) &= (r_1 r_2)^{p-1} - r_1^{p-1} - r_2^{p-1} + 1 \\ &= (r^{p-1} - 1) - (r_1^{p-1} - 1) - (r_2^{p-1} - 1) \equiv 0 \pmod{p}\end{aligned}$$

Quindi

$$r^{p-1} - 1 \equiv (r_1^{p-1} - 1) + (r_2^{p-1} - 1) \pmod{p}$$

Applicando la definizione di $lp(r)$ si ottiene:

$$\begin{aligned}l_p(r) = l_p(r_1 r_2) &= \frac{r^{p-1} - 1}{n^{p-1} - 1} \\ &\equiv \frac{(r_1^{p-1} - 1) + (r_2^{p-1} - 1)}{n^{p-1} - 1} \pmod{p} \\ &\equiv \frac{r_1^{p-1} - 1}{n^{p-1} - 1} + \frac{r_2^{p-1} - 1}{n^{p-1} - 1} \pmod{p} \\ &= l_p(r_1) + l_p(r_2),\end{aligned}$$

e inoltre

$$l_p(n) = \frac{n^{p-1} - 1}{n^{p-1} - 1} \pmod{p} = 1.$$

Proposizione 6.1.3. *Se le condizioni 6.1 e 6.2 sono soddisfatte allora si ha che* $\chi(r) = \eta(\chi)^{l_p(r)} \quad \forall r \mid n$. *In particolare* $\chi(n) = \eta(\chi)$ *quindi* $\chi(r) = \chi(n)^{l_p(r)}$.

Dimostrazione. Si considera l'omomorfismo $\sigma : R \to R$ tale per cui $\sigma(\zeta_p) = \zeta_p^{n^i}$ e $\sigma(\zeta_q) = \zeta_q$. Applicando tale funzione a ciascun elemento della condizione 6.1 si ottiene

$$\tau(\chi^{n^i})^n \equiv \eta(\chi)^{-n^{i+1}} \tau(\chi^{n^{i+1}}) \pmod{nR} \tag{6.3}$$

Infatti si ha:

- $\sigma(\tau(\chi)^n) = (\sum_{x=1}^{q-1} \chi(x)^{n^i} \zeta_q^x)^n = \tau(\chi^{n^i})^n$;
- $\sigma(\eta(\chi)^{-n}) = \eta(\chi)^{-nn^i} = \eta(\chi)^{-n^{i+1}}$;
- $\sigma(\tau(\chi^n)) = \tau(\chi^{nn^i}) = \tau(\chi^{n^{i+1}})$.

Si vuole ora dimostrare per induzione la validità della seguente congruenza:

$$\tau(\chi)^{n^i} \equiv \eta(\chi)^{-in^i} \tau(\chi^{n^i}) \pmod{nR}.$$

Per $j = 0$ si ottiene:

$$\tau(\chi) \equiv \tau(\chi) \pmod{nR}.$$

Supponiamo che l'equivalenza sia stata dimostrata per valori di j minori o uguali a $i - 1$, si vuole provare che essa è verificata anche per i. Per ipotesi induttiva vale quindi:

$$\tau(\chi)^{n^{i-1}} \equiv \eta(\chi)^{-(i-1)n^{i-1}} \tau(\chi^{n^{i-1}}) \pmod{nR};$$

Valutando la congruenza per $j = i$ si ottiene:

$$\begin{aligned}
\tau(\chi)^{n^i} &= (\tau(\chi)^{n^{i-1}})^n \\
&\equiv (\eta(\chi)^{-(i-1)n^{i-1}})^n (\tau(\chi^{n^{i-1}}))^n && \pmod{nR} \\
&\equiv \eta(\chi)^{-(i-1)n^i} (\tau(\chi^{n^{i-1}}))^n && \pmod{nR} \\
&\equiv \eta(\chi)^{-(i-1)n^i} (\eta(\chi)^{-n^i} \tau(\chi^{n^i})) && \pmod{nR} \\
&\equiv \eta(\chi)^{-in^i} \tau(\chi^{n^i}) && \pmod{nR}.
\end{aligned}$$

Consideriamo adesso la congruenza appena dimostrata nel caso in cui $i = p-1$; si ottiene dunque:

$$\begin{aligned}
&\tau(\chi)^{n^{p-1}} \equiv \eta(\chi)^{-(p-1)n^{p-1}} \tau(\chi^{n^{p-1}}) && \pmod{nR} \\
&\tau(\chi)^{n^{p-1}} \equiv \eta(\chi)\tau(\chi) && \pmod{nR} \\
&\tau(\chi)^{n^{p-1}} \tau(\chi)^{-1} \equiv \eta(\chi)\tau(\chi)\tau(\chi)^{-1} && \pmod{nR} \\
&\tau(\chi)^{n^{p-1}-1} \equiv \eta(\chi) && \pmod{nR}.
\end{aligned}$$

Al risultato finale si è giunti osservando i seguenti aspetti:

- $\eta(\chi) \in \langle \zeta_p \rangle$, quindi $\eta(\chi)$ ha ordine p;
- $gcd(n, p) = 1$ quindi $n^{p-1} \equiv 1 \pmod{p}$;
- La somma di Gauss $\tau(\chi)$ è un elemento invertibile in R; si indica con $\tau(\chi)^{-1}$ il suo inverso.

Sia r un divisore primo di n, allora per r vale la congruenza

$$\tau(\chi)^{r^{p-1}-1} \equiv \chi(r) \pmod{rR};$$

essendo

$$l_p(r) = \frac{r^{p-1} - 1}{n^{p-1} - 1}$$

si ottiene che $r^{p-1} - 1 = l_p(r)(n^{p-1} - 1)$ e quindi

$$\chi(r) \equiv \eta(\chi)^{l_p(r)} \pmod{rR}.$$

Essendo $\chi(r)$ che $\eta(\chi)$ radici p-esime primitive dell'unità, si ha che

$$\chi(r) = \eta(\chi)^{l_p(r)}.$$

Infine, se n soddisfa le condizioni 6.1 e 6.2 allora:

$$\chi(n) = \eta(\chi)^{l_p(n)} = \eta(\chi).$$

□

Lemma 6.1.4. *Se $p^2 \nmid (n^{p-1} - 1)$ allora la condizione 6.2 è vera.*

Dimostrazione. Se $p^2 \nmid (n^{p-1} - 1)$ allora sicuramente si avrà che $\nu_p(n^{p-1} - 1) = 1$; quindi per ogni divisore r di n vale sicuramente la relazione

$$\nu_p(r^{p-1} - 1) \geq \nu_p(n^{p-1} - 1),$$

quindi la condizione 6.2 è certamente verificata. □

Lemma 6.1.5. *Se la condizione 6.1 è verificata e $\eta(\chi) \neq 1$, allora $p = ord(\chi)$ soddisfa la condizione 6.2.*

Dimostrazione. Sia ω l'ordine di $\tau(\chi)$ tale quinid che $\tau(\chi)^{\omega} \equiv 1 \pmod{rR}$. Allora $\omega \mid p(r^{p-1} - 1)$ e $\omega \mid p(n^{p-1} - 1)$ ma $\omega \nmid n^{p-1} - 1$ perchè $\eta(\chi) \neq 1$. Quindi $\nu_p(\omega) = \nu_p(p(n^{p-1} - 1))$ e $\nu_p(\omega) \leq \nu_p(p(r^{p-1} - 1))$. Si ottiene allora:

$$\nu_p(p(n^{p-1} - 1)) = 1 + \nu_p(n^{p-1} - 1) \leq \nu_p(p(r^{p-1} - 1)) = 1 + \nu_p(r^{p-1} - 1),$$

e quindi:

$$\nu_p(n^{p-1} - 1) \leq \nu_p(r^{p-1} - 1).$$

□

6.2 L'algoritmo APR-L probabilistico

In questa sezione viene presentato lo schema dell'algoritmo APR-L nella sua versione probabilistica. Per aiutare il lettore nella comprensione dei vari passaggi si è eseguito il test su tre valori di N sufficientemente piccoli, $N_1 = 101$, $N_2 = 111$, $N_3 = 121$ e ad ogni step sono riportati i corrispondenti calcoli.

[STEP 1]

$\mathtt{I} = 1$;
$\mathtt{j} = 0$;

[STEP 2]

$\mathtt{pr} = \mathtt{nextprime}(\mathtt{pr})$;
$\mathtt{j} = \mathtt{j} + 1$;
$\mathtt{I} = \mathtt{I} * \mathtt{pr}$;
Costruisci l'intero F *come prodotto dei primi* q *con* $\mathtt{q} - 1 \mid \mathtt{I}$;
se $\mathtt{F}^2 \leq \mathtt{N}$ *allora torna a STEP 2*;
Se $\gcd(\mathtt{N}, \mathtt{I} \cdot \mathtt{F}) > 1$ *e* $\gcd(\mathtt{N}, \mathtt{I} \cdot \mathtt{F})! = \mathtt{N}$ *, return 'N è composto!';*

I primi due step dell'algortimo APR-L probabilistico sono identici alla versione deterministica e per gli esempi ad essi associati si rimanda il lettore alla Sezione 4.2 del Capitolo 4. Per $N_2 = 111$ l'algoritmo termina già nel secondo step, rilevando un fattore non banale nel calcolo del Massimo Comune Divisore tra N e il prodotto $I \cdot F$; per gli altri due valori di N il secondo step è superato fornendo i valori $I = 6$ e $F = 42$.

[STEP 3]

Per ogni primo p *con* $\mathtt{p} \mid \mathtt{I}$ *ripetere gli Step 3.1, 3.2, 3.3*

[STEP 3.1]

Per ogni primo q *con* $\mathtt{q} \mid \mathtt{F}$ *scegliere un carattere di Dirichlet* $\chi_{\mathtt{p},\mathtt{q}}$ *di modulo* q *e ordine* p*;*
Se ho esaurito tutti i valori di q *allora non esiste nessun carattere di Dirichlet* χ *di ordine* p *e modulo* q *che soddisfa le condizioni degli step 3.2 e 3.3, quindi return 'N è composto!'*

[STEP 3.2]

Verificare se $\chi(\mathtt{N}) \neq 1$*; in caso affermativo si passa allo step successivo altrimenti, si torna allo step 3.1*

[STEP 3.3]

$\chi_{\mathtt{p},\mathtt{q}} = \chi$; *Se la congruenza*

$$\mathtt{G}(\mathtt{p}, \mathtt{q}) = \tau(\chi)^{\mathtt{N}} \equiv \chi(\mathtt{N})^{-\mathtt{N}} \tau(\chi^{\mathtt{N}}) \pmod{\mathtt{N}}$$

non è verificata ritornare allo Step 3.1

Essendo un carattere di Dirichlet di ordine p e modulo q un morfismo tra $\mathbb{Z}_q^*$ e $\langle \zeta_p \rangle$ ciò che sostanzialmente si esegue nello step 3 è il seguente:

1. Fissato un primo p si sceglie un primo q tale che $q \mid F$ e $p \mid q-1$; per tale valore di q si determina g, il generatore del gruppo ciclico $\mathbb{Z}_q^*$;
2. Si associa a $\chi(g)$ una radice p-esima primitiva diversa da 1; se si sono già utilizzate tutte le radici p-esime primitive dell'unità allora si torna al punto 1 e si sceglie un altro valore q;
3. Si calcola $\chi(N)$ sapendo che $\chi(N) = \chi(g^k) = \chi(g)^k$, dove k è l'intero per cui $N \equiv g^k \pmod{q}$;
4. Si verifica che $\chi(N)$ sia diverso da 1, in caso affermativo si può procedere, altrimenti si torna al punto 2 associando a $\chi(g)$ un'altra radice p-esima primitiva dell'unità;
5. Si verifica la condizione fondamentale espressa in 6.1; se è superata allora si procede, altrimenti si torna al punto 2;
6. Se non esiste nessun primo q con $q \mid F$ e $p \mid q-1$ tale per cui la condizione sia verificata allora N è composto.

Lo step 3 dell'algoritmo è caratterizzato quindi dalla verifica delle condizioni 6.1 e 6.2; se N è un numero primo allora si sa che:

$$\tau(\chi)^n \equiv \chi(n)^{-n}\tau(\chi^n) \pmod{nR}.$$

Se N è primo allora la congruenza al passo 3 è sicuramente verificata per un valore $\chi(n) \neq 1$; se invece non esiste nessun carattere di Dirichlet tale per cui la congruenza sia soddisfatta allora si può concludere che N è composto e l'algortimo termina.

$N_1 = 101$
$p = 2,\ q = 3$
$g = 2\ k = 1$
$\chi(g) = -1$
Condizione ok!
$p = 3\ q = 7$
$g = 3\ k = 1$
$\chi(N) \neq 1$
Condizione ok!

Step 3 superato, $N_1 = 101$

$N_3 = 121$
$p = 2,\ q = 3$
$g = 2\ k = 0$
$\chi(N) = 1$
$p = 2\ q = 7$
$g = 3\ k = 2$
$\chi(N) \neq 1$
Condizione fallita!

Step 3 fallito, $N_3 = 121$

[STEP 4]
Al variare di j *tra* 1 *e* I − 1, *se* $\mathtt{N}^{\mathtt{j}}$ (mod F) *è un fattore non banale di* N *allora return 'N è composto!';*
Altrimenti, return 'N è primo!'.

Nello step 4 si costruisce l'insieme dei potenziali divisori di N , un insieme che, se N è composto, contiene il suo più piccolo fattore primo. L'idea fondamentale è la seguente: se N è un numero composto che ha superato lo step 3 dell'algortimo allora esiste un suo divisore $r \leq \sqrt{N}$ e $\chi(r) = \chi(N^l)$ con l un intero ottenuto risolvendo il sistema di congruenze

$$l \equiv l_p(r) \pmod{p} \quad \forall p \mid n.$$

Da un ragionamento analogo a quello esposto nella Sezione 4.3 si ottiene quindi che il fattore r è pari a

$$r \equiv N^l \pmod{F};$$

essendo $F \geq \sqrt{n} \geq r$ e $F \neq r$ si ha che r è uguale al più piccolo residuo positivo di $N^l \pmod{F}$, quindi

$$r = [N^l]_F.$$

$N_1 = 101$
$j = 1 \Rightarrow Mod(101, 42) = 17 \Rightarrow gcd(17, 101) = 1$
$j = 2 \Rightarrow Mod(101^2, 42) = 37 \Rightarrow gcd(37, 101) = 1$
$j = 3 \Rightarrow Mod(101^3, 42) = 41 \Rightarrow gcd(41, 101) = 1$
$j = 4 \Rightarrow Mod(101^4, 42) = 25 \Rightarrow gcd(25, 101) = 1$
$j = 5 \Rightarrow Mod(101^5, 42) = 5 \Rightarrow gcd(5, 101) = 1$
$j = 6 \Rightarrow Mod(101^6, 42) = 1 \Rightarrow gcd(1, 101) = 1$
N_1 è primo!

Step 4: al variare di j tra 0 e I si determina $N^j \pmod{F}$ e si calcola il Massimo Comune Divisore tra $N_1 = 101$ e $[N^j]_F$.

Oss 6.2.1. Il carattere probabilistico dell'algoritmo dipende solamente dalla scelta del carattere χ di ordine p e modulo q che verifichi $\chi(n) \neq 1$; e quindi sostanzialmente dalla verifica della condizione 6.2. Questa verifica, se N è un numero primo, può richiedere molto tempo; l'aspetto non deterministico dell'algoritmo può essere rimosso introducendo le ipotesi generalizzate di Riemann.

6.3 Implementazione APR-L probabilistico

In questa sezione si presenta il codice utilizzato nell'algoritmo APR-L probabilistico; molte delle funzioni utilizzate sono le stesse descritte nell'analisi dell'implementazione dell'algoritmo APR-L deterministico nel Capitolo 5 e quindi per

la loro descrizione si rimanda il lettore a tale capitolo.
Le funzioni create sono due: la prima, la funzione *confronto* esegue sostanzialmente lo step 3 dell'algoritmo, dalla determinazione del generatore del gruppo ciclico $\mathbb{Z}_q^*$ e dell'ordine di N in $\mathbb{Z}_q^*$ alla verifica della condizione 6.1 (non prima di aver accertato che il valore $\chi(N)$ sia diverso da 1); la seconda funzione è la funzione *union* che, come nella versione deterministica, è la funzione principale in cui sono combinate le varie procedure al fine di realizzare quanto descritto nell'algoritmo.

La funzione confronto.gp

```
{confronto(S1,S2,p,q)=
 local(vett,a,b,c,e,stringa,l,gq,z,ord);
 vector_root(p);
 e=valuta_S(S1);
 gq=znprimroot(q);
 z=Mod(n,q);
 ord=znlog(z,gq);
 for (j=2,p,
    v=Mod(Pol(vett_root[j])^ord,polp);
      if(lift(v)!=Mod(1,n),
            v=v^(-1);
            a=quadrati_ripetuti(v,n);
            b=a*S2;
            c=valuta_S(b);
            if(c==e,
                  print ("congruenza verificata!");
                  check=1;
                  j=p+1)));
}
```

La funzione union.gp

```
{union(n)=
 i=1;
 pr=0;
 stringa="";
 print (" STEP 2");
 stringa=prep(n,pr);
 if (stringa=="composto",
      print ("RISULTATO");
      print ("n è composto!");
      break);
 print("I vale ",i);
 print("F vale ",f);
 j=0;
 k=0;
 print (" STEP 3");
 forprime (p=2,i,
      if(i%p==0,
            check=0;
            forprime(q=2,i+1,
                  if(f%q==0,
                        k+=1;
                        if((q-1)%p==0,
                              S=gsum(p,q);
                              S1=gauss_modn(S,p,q);
                              S2=gsum_n(p,q);
                              S2=gauss_modn(S2,p,q);
                              S1=quadrati_ripetuti(S1,n);
                              confronto(S1,S2,p,q);
                              if (check==1,
                                    q=i+2))));
```

```
            if (check==0,
                print("'RISULTATO"');
                print("n è composto!");
                break(3))));

 print (" STEP 4");
 res=Mod(n,f);
 if (gcd(lift(res),n)>1 && gcd(lift(res),n)!=n,
       print("RISULTATO");
       print("n è composto!");
       break);
 for (j=2,i-1,
      res=quadrati_ripetuti(res,n);
      if (gcd(lift(res),n)>1 && gcd(lift(res),n)!=n,
      print("RISULTATO");
      print("n è composto!");
      break(2)));

 print("RISULTATO");
 print("n è primo!!");
}
```

E' stata creata una nuova funzione *gsum_n* che calcola $\tau(\chi^n)$; praticamente uguale alla funzione *gsum*, si differenzia soltanto nel calcolo dei coefficienti associati a ζ_p ottenuti considerando il modulo della divisione tra kN (anzichè solo k) e p al variare di k tra 1 e $q-1$.

```
gp > n=101
%21 = 101
gp > union(n)
STEP 2
I vale 6
F vale 42
STEP 3 Computation of the coefficients of a Gauss sum with p=
2 and q= 3
Computation of the coefficients of a Gauss sum with p= 2 and
q= 3
Congruenza verificata!
Computation of the coefficients of a Gauss sum with p= 3 and
q= 7
Computation of the coefficients of a Gauss sum with p= 3 and
q= 7
Congruenza verificata!
STEP 4
RISULTATO
n è primo!
```

La funzione union in Gp

```
gp > n=121
%30 = 121
gp > union(n)
STEP 2
I vale 6
F vale 42
STEP 3
Computation of the coefficients of a Gauss sum with p= 2 and
q= 3
Computation of the coefficients of a Gauss sum with p= 2 and
q= 3
Computation of the coefficients of a Gauss sum with p= 2 and
q= 7
Computation of the coefficients of a Gauss sum with p= 2 and
q= 7
RISULTATO
n è composto!
```

La funzione union in Gp

Capitolo 7

Complessità Computazionale del test APR-L

7.1 Analisi del test APR-L

Il test APR e le versioni modificate (APR-L) hanno complessità pari a

$$O(\log N^{c \log\log\log N}),$$

dove c è una costante positiva per la quale un limite superiore potrebbe essere calcolato in principio.
La complessità sostanzialmente dipende dalla scelta dell'intero I definito come specificato nella Sezione 4.2. Uno dei punti fondamentali dell'algoritmo è infatti la costruzione degli interi I e F, ottenuti dal prodotto rispettivamente dei primi p e dei primi q; I e F devono soddisfare tre proprietà:

1. devono essere entrambi liberi dal quadrato;
2. $q - 1 \mid I \quad \forall\, q \mid F$;
3. $F > \sqrt{N}$.

L'intero F dipende quindi dall'intero I e quest'ultimo numero deve essere sufficientemente grande da garantire che F soddisfi le condizioni appena elencate. Inoltre, un risultato di Teoria Analitica dei Numeri permette di limitare superiormente il valore di I. Quando $N > 100$ si ottiene

$$I = \prod p < \log N^{c \log\log\log N}.$$

Definita $f(N)$ una funzione che associa ad ogni valore di N il giusto valore dell'intero I che soddisfi le condizioni richieste nel test, si può concludere che ogni passo dell'algoritmo (sia nella versione deterministica che in quella probabilistica) ha complessità *polinomiale* in $f(N)$.
E' quindi necessario conoscere con maggiore dettaglio $f(N)$, cercando di dare una valutazione dell'ordine di grandezza di tale funzione; il Teorema che segue caratterizza la funzione $f(N)$.

Teorema 7.1.1. *Sia n un intero, $n > 100$; esistono due costanti c_1, c_2 positive e calcolabili tali che:*

$$\log n^{c_1 \log\log\log n} < f(n) < \log n^{c_2 \log\log\log n}.$$

Nella dimostazione di questo Teorema si utilizzano dei concetti di Teoria Analitica dei Numeri che vanno al di là dello scopo di questo elaborato e per tale motivo non verrà trattata; in ogni caso una dimostazione dettagliata del Teorema 7.1.1 è presente nel lavoro di Adleman, Pomerance e Rumely [1] in cui viene descritto il test APR.

7.2 Analisi dei tempi di esecuzione

In questa sezione si riportano e si analizzano i tempi di esecuzione dei programmi implementati, eseguiti su un notebook dalle prestazioni standard.
Alcune considerazioni possono già essere fatte:

- generalmente per un qualsiasi test di primalità la versione probabilistica offre prestazioni nettamente superiori rispetto alla versione deterministica; il carattere probabilistico, e quindi l'incertezza del risultato nel caso in cui il numero testato sia valutato primo, sono dovuti al fatto che il numero di confronti eseguiti nel test probabilistico è di molto inferiore rispetto al test deterministico.
  Nel caso in esame, nel test APR-L deterministico la condizione di primalità espressa nel Lemma 3.3.2 è verificata per ogni coppia di primi (p, q) dove p è un fattore di I e q è un fattore di F e $p \mid q-1$; se il primo step è superato, l'insieme dei potenziali divisori (indicato con R) che si viene a determinare contiene certamente il più piccolo fattore primo di N, nel caso in cui N sia composto. Se in R non è presente alcun fattore primo di N allora si può certamente concludere che il numero testato è primo.
  Nel test APR-L probabilistico le condizioni testate sono due e sono le condizioni espresse in 6.1 e 6.2; tali condizioni non sono verificate per tutte le coppie (p, q) come nel caso deterministico ma, per ogni primo $p \mid I$ è sufficiente trovare un primo $q \mid F$ con $p \mid q-1$ che soddisfi le condizioni. Il carattere probabilistico dipende quindi dalla scelta del carattere di Dirichlet di ordine p e modulo q che verifichi $\chi(N) \neq 1$. Ovviamente, avendo verificato le condizioni per un numero limitato di coppie di primi (p, q) l'insieme R che si viene a determinare non contiene più con certezza il più piccolo fattore primo di N nel caso in cui esso sia composto.

- Analizziamo brevemente le condizioni che devono essere verificate nei test. Nell'algoritmo APR-L deterministico la condizione è:

  $$G(p, q)^{N^{p-1}-1} \equiv \chi_{p,q}(N) \pmod{N};$$

  mentre nel test APR-L probabilistico la condizione che viene verificata è:

  $$\tau(\chi)^N \equiv \chi(N)^{-N} \tau(\chi^N) \pmod{N},$$

dove $G(p,q)$ e $\tau(\chi)$ sono scritture diverse per rappresentare lo stesso concetto e cioè la somma di Gauss associata al carattere di Dirichlet di ordine p e modulo q con $p \mid q-1$. Si può notare facilmente che la differenza principale tra le due condizioni è l'esponente a cui deve essere elevata la somma di Gauss: nella versione deterministica l'esponente può essere molto più grande di N con conseguente appesantimento dei calcoli.
Consideriamo per esempio un numero N di 20 cifre decimali; nella fase iniziale dei test si determina l'intero I pari a $I = 210 = 2 \cdot 3 \cdot 5 \cdot 7$, quindi la somma di Gauss nel test APR-L deterministico al variare di p in $\{2, 3, 5, 7\}$ è elevata ad esponenti di circa 20, 40, 80 e 120 cifre decimali. Nel test APR-L probabilistico invece l'esponente è sempre lo stesso ed è pari a N. Anche questo fattore contribuisce quindi in modo determinante all' allungamento dei tempi di esecuzione del programma che implementa la versione deterministica del test.

Si riporta di seguito la tabella dei tempi di esecuzioni dei programmi; i tempi sono riferiti al caso peggiore cioè quando il numero testato è primo.

n°digits	**APR-L det**	**APR-L prob**
3	16 ms	0 ms
7	530 ms	93 ms
10	1min 38s	532 ms
15	5min 38s	1435 ms
20	10min	2792 ms
30	>15min	7862 ms

Tempi di esecuzione dei test APR-L det e APR-L prob

Di seguito si riportano i tempi di esecuzione del test probabilistico per valori di N superiori a quelli riportati nella tabella sopra; si tralasciano i tempi di esecuzione del test deterministico che si è visto essere già molto elevati quando N è un numero di 30 cifre decimali.

n°digits	**APR-L prob**
40	12,6s
50	1 min 48s
60	2min 19 s
70	2min 54 s
80	>5min

Tempi di esecuzione del test APR-L probabilistico

Bibliografia

[1] Adleman, L., C. Pomerance e R.S. Rumely: *On distinguishing prime numbers from composite numbers*. Annals of Mathematics, 117:173–206, 1983.

[2] Alford, W.R., A. Granville e C. Pomerance: *There are infinitely many Carmichael numbers*. Annals of Mathematics, 1994.

[3] Apostol, T. M.: *Introduction to Analytic Number Theory*. Springer-Verlag, quarta edizione, 2010.

[4] Bosma, W.: *Primality Proving with Cyclotomy*. tesi di laurea, Universiteit van Amsterdam, 1990.

[5] Chmielowiec, A.: *Primality Proving with Gauss and Jacobi Sums*. Enigma Information Security Systems.

[6] Cohen, H.: *Test de Primalité d'apres Adleman, Pomerance, Rumely*. Nel *Séminaire de Theorie des Nombres*, 1981.

[7] Cohen, H. e A.K. Lenstra: *Implementation of a New Primality Test*. Mathematics of Computation, Vol.48:103–121, 1987.

[8] Cohen, H. e H.W.Jr. Lenstra: *Primality Testing and Jacobi Sums*. Mathematics of Computation, Vol.42:297–330, 1984.

[9] Crandall, R. e C. Pomerance: *Primes Numbers: A Computational Perspective*. Springer, seconda edizione, 2005.

[10] Hardy, G.H. e E.M. Wright: *An Introduction to the Theory of Numbers*. Oxford Science Publication, prima edizione, 1979.

[11] Languasco, A.: *Codici a chiave pubblica ed algoritmi di primalità*. tesi di laurea, Università degli Studi di Genova, A.A 1988/89.

[12] Languasco, A. e A. Zaccagnini: *Introduzione alla Crittografia*. Ulrico Hoepli Milano, prima edizione, 2004.

[13] Lenstra, H.W.Jr.: *Primality Testing Algorithms (after Adleman, Pomerance, Rumely)*. Nel *Seminaire Bourbaki 33*, pagine 246–257, 1980/1981.

[14] Manin, Y. Ivanovic e A.A. Panchishkin: *Introduction to Modern Number Theory*. Springer, 2005.

[15] Miller, S.J. e T.B. Ramin: *An Invitation to Modern Number Theory*. Princeton University Press, 2006.

[16] Murty, M.Ram: *Problems in Analytic Number Theory*. Springer, terza edizione, 2008.

[17] Pomerance, C.: *On the distribution of pseudoprimes*. Mathematics of Computation, 37:587–593, 1981.

[18] Pomerance, C.: *Recent Development in Primality Testing*. Mathematical Intelligencer, 3:97–105, 1981.

[19] Rumely, R.S.: *Recent Advances in Primality Testing*. Notices of American Mathematical Society, pagina 475, agosto 1983.

[20] Schoof, R.: *Four Primality Testing Algorithms*. Algorithmic Number Theory, Vol.44, 2008.

[21] Vàrilly, A.: *Dirichlet's Theorem on Arithmetic Progression*.

Ringraziamenti

Mi piacerebbe ringraziare le molte persone che mi sono state vicine in questo mio percorso, non solo di studi ma anche di vita.
Un particolare ringraziamento al Prof. Alberto Tonolo per la fiducia e per gli stimoli che ha saputo offrirmi e al Prof. Alessandro Languasco per la grande disponibilità e il suo continuo aiuto.
Non avrei neppure preso in considerazione questo cammino senza il supporto di mia madre Giovanna, che fin dall'inizio ha creduto in me.
Grazie a mio padre, Alfredo, per avermi dimostrato lo spirito giusto nell'affrontare i problemi e a mio fratello, Cristian, per avermi sopportata nei momenti di tensione.
A Domenico per essermi stato vicino e per avermi dedicato così tanto tempo e attenzioni.
A tutti i parenti, amici e colleghi, troppo numerosi per nominarli, un grazie di cuore per la loro gentilezza ed il loro sostegno.