102

UNIVERSIDAD NACIONAL DEL CALLAO

FACULTAD DE CIENCIAS NATURALES Y MATEMÁTICA

ESCUELA PROFESIONAL DE MATEMÁTICA

IF

ABR 2014

102

"Avances en Factorización Entera. Factorización con Curvas Elípticas"

RUTH MEDINA APARCANA

(01.07.2012-28.02.14)

Resolución Rectoral Nº 674-2012-R

Callao, 2014

Índice general

1. Resumen

El presente trabajo consiste en el estudio de diferentes algoritmos matemáticos para lograr la factorización de enteros grandes, que son de trascendencia por su eficiencia y rapidez.
En las últimas décadas hemos visto la llegada del poder de la computación, que cada vez se ha hecho más accesible y más rápido; y nos encontramos con que ese crecimiento en el poder de cómputo no ha resuelto el problema de la factorización; ha dejado en evidencia que el reto matemático de la factorización requiere estudio e investigación.
Iniciamos este trabajo enfatizando en el problema de factorizar y su trascendencia de hacerlo, luego continuamos con una pequeña reseña de los aportes de algunos matemáticos desde la época griega como Euclídes, Eratóstenes y el gran aporte de Fermat con su pequeño gran teorema concido como el pequeño Teorema de Fermat y su aplicación al sistema criptográfico RSA de vigencia actual si de proteger información se trata, por medios inseguros, como lo es internet.
En realidad son inumerables los matemáticos que han investigado en el reto de factorización de los enteros, algunos sin obtener frutos y otros con resultados muy pocos trascendentes, mostraremos algunos de los más relevantes.
Luego, presentamos una serie de resultados de algebra básicos como Fracccions Continuas, entre otros temas, como el método de rho de Pollar y otros temas nada básicos como la teoría de Curvas Elípticas que constituyen el cimiento donde descanza el estudio de los los tres algoritmos más significativos en cuanto a factorización.
Seguidamente, hacemos un estudio de factorización con Fracciones Continuas, mejoras en su rendimiento y la descripción del algoritmo del mismo nombre; así también hacemos lo propio con el método de la Criba Cuadrática, la criba

del campo numérico y por último el estudio de el algoritmo de Curvas Elípticas para factorización de enteros y mostramos un ejemplo de aplicación usando este método.

En general no es fácil empezar a factorizar a un número entero si no se conoce nada acerca de él, ya que algunos métodos son más eficientes que otros, sin embargo podemos optimizar el proceso de factorizar, aplicando el mejor método conforme a la forma del número, podemos empezar por el de ensayar divisores pequeños, hasta una cota Co; luego podemos aplicar el algoritmo de Pollard, esperando encontrar un factor r, tal que $Co < r < C1$ donde $C1$ es otra cota; luego el método con curvas elípticas, esperando encontrar un factor r1, tal que $C1 < r1 < C2$ y por último aplicar un método de propósito general, como la Criba cuadrática o la Criba de campos numéricos.

2. Introducción

La factorización de números enteros es un problema muy actual, no porque se haya resuelto o se conozca un método realmente eficiente para resolverlo sino más bien por todo lo contrario.
La dificultad del problema de factorizar es importante en Criptografía; este problema, atacado infructuosamente hasta el momento, nos permite considerar la multiplicación de números enteros como una cierta función de dirección única.
La aplicación más importante de este hecho es el sistema criptogáfico de clave pública RSA, introducido por Rivest, Shamir y Adleman en 1978. Éste fue el primer sistema de cifrado en el que cada usuario tiene dos llaves, una pública y otra secreta, para la comunicación segura a través de un canal inseguro, como lo es hoy el internet. Aún hoy en día, el RSA es el estándar de comunicaciíon electrónica segura más utilizado. La seguridad de este sistema radica en la imposibilidad práctica de factorizar números grandes, con centenares de dígitos.
El algoritmo clásico de factorización con mejor complejidad demostrada es el obtenido por Pollard y Strassen en 1976 que tiene complejidad $O(2^{\frac{n}{2}})$ [10], siendo n el número de dígitos. Sin embargo existen otros algoritmos más eficientes, aunque no se haya conseguido probar rigurosamente su complejidad. Por ejemplo, se conjetura que el algoritmo de Lenstra, Manasse y Pollard [16] tiene complejidad $e^{(O\sqrt[3]{nlog^2 n})}$.
Para factorizar un número n pequeño, primero debemos decidir si un número n pequeño es primo, podemos usar el método de ensayo y error para verificar que no tiene divisores primos inferiores a $\sqrt{n}$.
Para un número un poco más grande, la estrategia usual es primero verificar si tiene divisores primos pequeños, sino se usa el test para seudoprimos fuertes de Miller-Rabin con unas pocas bases p_i (con p_i primo) y usualmente se combina

con el test de Lucas. Esta manera de proceder decide de manera correcta si un número es primo o no, hasta cierta cota 10^M.

Es decir, la combinación de algoritmos decide de manera correcta si $n < 10^M$; sino, decide de manera correcta solamente con una alta probabilidad y cabe la (remota)posibilidad de declarar un número compuesto como primo.

En el método de factorización por ensayo y error, en su versión más simple, probamos con todos los números entre 2 y $\sqrt{N}$ para hallar un factor de N. Si no lo hallamos, N es primo. En vez de hacer estos $\sqrt{N}$ pasos (en el peor caso), vamos a escoger una lista aleatoria de números, más pequeña que $\sqrt{N}$, y probar con ellos.A menudo se construyen sucesiones seudoaleatorias $x_0, x_1, x_2, ...$ usando una iteración de la forma: $x_{i+1} = f(x_i)(\ \text{mód}\ N)$, con x_0 en el rango $(0, N-1)$. Entonces $\{x_0, x_1, ...\} \subseteq Z_N$. Por lo tanto los x_i se empiezan a repetir en algún momento.

En general no existe una forma eficiente de factorizar a un número entero si no se conoce nada acerca de él, ya que algunos métodos son más eficientes en algunos casos que en otros, sin embargo podemos dar una estrategia para poder intentar factorizar un número entero.

3. Marco Teórico

3.1. El problema de factorizar

A largo de la historia, los matemáticos han tratado de comprender el comportamiento de los números primos, pues esto pertmitiría poder factorizar poder factorizar números enteros, señalaremos algunos de los principales matemáticos que realizaron aportaciones sobre el tema en tiempos pasados.

- Euclides (325-265 a. C.), demuestra que hay infinidad de números primos en la proposición número 20 del libro IX de los Elementos:
  "Ningún conjunto de números primos los incluye a todos". La demostración es muy sencilla: Supongamos que hay una cantidad finita de números primos $p_1, p_2, \cdots, p_k$. Consideramos 1 más el producto de todos ellos, es decir, $N = (p_1.p_2.\cdots p_k) + 1$. Este número podría resultar ser un número primo, o bien es divisible por algún primo no considerado.
  En cualquier caso el primo p resultante satisface $p \neq p_i; i = 1, 2, \cdots, k$.

- Eratóstenes (276-195 a. C.), cerca del año 200 a. C., el astrónomo Eratóstenes de Cirene ideó un algoritmo para calcular números primos, llamado Criba o Tamiz de Eratóstenes.
  La criba de Eratóstenes es un algoritmo que permite hallar todos los números primos menores que un número natural dado N, para ello:
  Se forma una tabla con todos los números naturales comprendidos entre 2 y N, y se van tachando los números que no son primos de la siguiente manera: se toma el 2 y se procede a tachar todos sus múltiplos, continúal proceso con el siguiente entero que no ha sido tachado y es declarado primo, después se tachan todos sus múltiplos, así sucesivamente. El proceso termina cuando ya no hay números por tachar.

- Pierre de Fermat(1601 – 1665), nació en Francia en 1601, las matemáticas eran su gran afición.

  Él demostró que cada número primo p, tal que $p-1$ es divisible entre cuatro, se puede escribir como la suma de dos cuadrados, el número dos también se incluye, ya que $1^2+1^2=2$. Ideó además un nuevo método de factorización de números grandes, y un teorema importante conocido como Pequeño Teorema de Fermat; establece que si p es un número primo, entonces para cualquier entero a primo relativo con p, obtenemos $a^{p-1} \equiv 1 \pmod{p}$.

  Fermat mantuvo correspondencia con otros matemáticos de su época, y en particular con el monje Marín Mersenne (1548-1688), que nacío en Francia y actuó de intermediario entre algunos matemáticos del siglo XVII. En una de sus cartas a Mersenne, Fermat conjetura que los números 2^N+1 eran siempre primos si N es una potencia de 2. Él había verificado esto para $N=1,2,4,8$ y 16 y sabía que si N no era una potencia de 2, el resultado fallaba.

  Los números de esta forma son llamados Números de Fermat. Pero 100 años más tarde Euler demostró que $2^{32}+1=4294967297$ es divisible por 641 y por tanto no es primo.

  Los números de la forma 2^N-1 también atrajeron la atención de muchos matemáticos porque es muy fácil demostrar que a menos que N sea primo, este número es compuesto. A menudo éstos son llamados números primos de Mersenne, dado que Mersenne los estudió. No todos los números de la forma 2^N-1 con N primo son primos, por ejemplo si $N=11,23,29,37,\cdots$ entonces 2^N-1 no es primo, en particular $2^{23}-1=8388607\times 47$.

3.2. El criptosistema Rivest, Shamir y Adleman

Este criptosistema fue propuesto en 1978, por Rivest, Shamir y Adleman, razón por la que es conocido con el nombre de *RSA*, que son las iniciales de los nombres de sus inventores.

El criptosistema *RSA* se basa en la hipótesis, de que, para n y e enteros positivos dados, donde n es el producto de dos primos grandes, la función

$m \rightarrow m^e \ \ (mod n)$ es de dirección única con trampa. La trampa, que permite invertir la función, es precisamente conocer la descomposición en factores primos de n, por lo que se puede decir que el RSA está basado en la dificultad del problema de factorización de números enteros.

Un usuario del RSA construye su clave de la siguiente manera:

Primero construye un entero $n = pq$ donde p y q son dos primos grandes, aproximadamente del mismo tamaño y que no estén muy próximos(para dificultar al máximo la factorización n).

Luego se calcula $\phi(n) = (p-1)(q-1)$ y se elige un entero e tal que $1 < e < \phi(n)$ y $mcd(e, \phi(n)) = 1$.

Finalmente, calcula el inverso multiplicativo de e módulo $\phi(n)$, es decir el único entero d tal que $1 < d < \phi(n)$ y $e.d \equiv 1 \ \ (mod \ \ \phi(n))$.

Todos estos cálculos se pueden realizar usando algoritmos conocidos. Así,el par (n, e) constituye la clave pública para el usuario; y d es su clave privada. n recibe el nombre de *módulo*, e es el *exponente de encriptación* y d el *exponente de desencriptación*

Para enviar un mensaje a este usuario sólo hay que conocer su clave pública (n, e), si el texto claro es m (donde m es un entero menor que n), éste se encripta calculando:

$$c = m^e \ \ (mod \ n)$$

Como antes se ha mencionado, la función encriptación que proporciona el criptotexto c a partir de m es una función de dirección única con trampa. El usuario por su parte, descifra el mensaje haciendo uso de su clave privada calculando:

$$c^d = m \quad (modn)$$

Si se conoce la factorización de n en producto de primos, es fácil invertir esta función y desencriptar, pues entonces se puede obtener $\phi(n)$ y luego d de la misma forma que lo ha hecho el usuario. Aunque no está demostrado, se cree que criptoanalizar RSA tiene esencialmente el mismo grado de dificultad que factorizar el módulo, es por ello que, si se resuelve factorizar de manera efectiva un entero, rompera la seguridad de este método.

El algoritmo RSA trabaja con 2 tipos de claves por cada usuario:

- Clave privada.- Esta clave es conocida tan sólo por el propio usuario.

- Clave pública.- Esta clave la puede conocer cualquier persona.

Estas claves se obtienen a partir de operaciones matemáticas que se presentan a continuación:

1. El usuario debe escoger un número n el cual sea el producto de dos números primos p y q.

$$n = p.q$$

2. Se calcula la función de Euler a n, es decir se debe calcular $\varphi(n)$.
   La función de Euler entrega el número de elementos del *Conjunto Reducido de Restos*, es decir, entrega la cantidad de números en el rango de $[1, n-1]$ los cuales no son factores de n. Es este caso, al ser n un número compuesto por dos factores primos, el resultado de la función de Euler será:

$$\varphi(n) = (p-1)(q-1)$$

3. Se elige la clave pública e, la cual corresponde a un número entero en el rango $]1, n[$ tal que e y $\varphi(n)$ sean primos relativos, o sea que el máximo común múltiplo entre ambos sea igual a 1. Es decir:

$$1 < e < n \quad / \quad MCD(e, \varphi(n)) = 1$$

4. Obtener la clave privada d, al calcular el inverso de e dentro del anillo de enteros $\mathbb{Z}_n$ (enteros no negativos menores que n). Es decir

$$d \quad / \quad e \cdot d \equiv 1 \ (\text{mód } n)$$

   Una vez finalizadas estas operaciones, **se publica la clave pública e y el número n.**

Cifrado RSA

La operación de cifrado de RSA es simple. Si se quiere cifrar un número N, habiendo elegido una clave pública e y calculado la clave privada d mediante la eleccin de n, se procede a calcular C, que corresponde a la salida de la operación de cifrado. Dependiendo de para qué se quiera usar el cifrado la operación será diferente.

Si lo que se quiere es la confidencialidad, ciframos el número N con la clave pública e del receptor, utilizando tambiń el n del receptor.

$$C = N^e (\text{mód } n)$$

En cambio, si se quiere el no repudio, utilizamos el mecanismo de firma digital, utilizando tanto clave privada d como el n del emisor.

$$C = h(N)^d (\text{mód } n)$$

En este último caso, $h(N)$ corresponde a la utilización de un algoritmo hash tal como $MD5$ o $SHA-1$ para el número N. Luego se tendrá que enviar tanto el mensaje N, como C para poder realizar la comprobación del mensaje.

Descifrado RSA

El descifrado clásico opera muy similar al cifrado. La diferencia es que se utilizan las claves contrarias al proceso de cifrado. Para que el receptor pueda obtener el número N a través de C deberá realizar la siguiente operacin

$$N = C^d (\text{mód } n)$$

En este caso, al igual que en el cifrado, se utiliza tanto la clave como el n del receptor.

Si el mensaje fue enviado con firma digital (N, C), el descifrado y posterior comprobacin se realizar calculando:

$$h(N) = C^e (\text{mód } n)$$

Luego, se procederá a calcular $h(N)$ a través del mensaje N. Si los dos $h(N)$ son congruentes, quiere decir que el mensaje no ha sido alterado, y el emisor ha sido comprobado.

3.3. Ejemplo de Rivest, Shamir y Adleman

Datos en Directorio Público

$e_B = 20803 \qquad n_B = 49477$

USUARIO A		USUARIO B
$M = 55$		$p_B = 197$
$C = 55^{20803}(\text{mód } 49447)$		$q_B = 251$
$\therefore C = 26567$	$\longrightarrow 26567 \longrightarrow$	$n_B = p_B \cdot q_B = 49477$
		$d_B = 34467$
		$e_B = 20803$
		$C = 26567$
		$M = 26567^{34467}(\text{mód } 49447)$
		$\therefore M = 55$

En este caso el usuario A quiere enviarle el mensaje $M = 55$ al usuario B, para lo cual obtiene la clave pública de B, es decir $e_B = 20803$ y el grupo finito con el que se trabaja, es decir $n_B = 49447$.
Luego se aplica las operaciones de cifrado y descifrado mencionadas en los párrafos anteriores.

3.4. Factorización por divisiones sucesivas

Un procedimiento muy sencillo para la buscar los factores primos de un entero cualquiera dado n, consiste en tomar una tabla de los primeros valores primos y proceder a calcular el módulo de dividir n por los sucesivos primos, comenzando por el primero de ellos (el 2). Cada vez que se encuentra un primo p que divide al candidato n se inicia el proceso a partir de ese primo.
Si, llegado a n no se ha encontrado ningún primo que divida a n o a lo que queda de él después de sucesivas divisiones, entonces podemos dar por terminado el proceso.
Este proceso puede ser útil para enteros pequeños (enteros que no superen el orden de 10^7), pero en cuanto se aumenta el tamaño del número a factorizar se hace impracticable pues requiere un tiempo de computación excesivo.

Un siguiente paso en los métodos de factorización será utilizar el método Rho de Pollard. Este algoritmo estén cualquier libro sobre factorización de los citados en la bibliografía, por ejemplo [9]. Es un algoritmo útil para enteros n de rango entre 10^6 y 10^{12} .

4. Materiales y Métodos

Materiales

Este trabajo se ha desarrollado sobre la base de textos, papers, artículos, software especializado experiencias propias y la combinación apropiada de técnicas del algebra, análisis, teoría de números, que han permitido un adecuado desarrollo de este trabajo. Se ha hecho uso de material de tipo técnico en el diseño e impresión de los informes trimestrales y el informe final. Toda la información ha sido procesada en una computadora personal usando un procesador científico LaTeXy un visor Adobe Acrobat para pdf, en concordancia con las directivas vigentes, mediante el cual se han editado todo el formulismo matemático.

Métodos

Realizada la elección de la bibliografía y la recolección de información, los métodos usados en el desarrollo de los temas en cada capítulo, que han permitido el avance y la exposición clara del trabajo de investigación son :

- Método Deductivo, que se caracteriza por por ser conciso y lógico lo que ha permitido desarrollar la teoría de curvas elípticas de una forma concreta y ordenada.
- Método inductivo-deductivo que ha permitido el desarrollo del formulísmo que describe los conceptos, así también como el análisis de las soluciones de los algoritmos y los resultados presentados.

5. Resultados

5.1. Método de Pollard

Los métodos ρ y λ de Pollard corren aproximadamente en el mismo tiempo que paso de bebé, paso de gigante, pero requieren poco almacenamiento, y requiere mucho menos espacio de almacenamiento . Primero discutiremos el método ρ, luego su generalización al método λ.

Sea G un grupo de orden finito N. Elegimos una función $f : G \to G$ que se comporte más bien aleatoriamente. Entonces empezamos con un elemento aleatorio P_0 y calculamos las iteraciones $P_{i+1} = f(P_i)$. Como G es finito, habrán algunos índices $i_0 < j_0$ tales que $P_{i_0} = P_{j_0}$. Entonces

$$P_{i_0+1} = f(P_{i_0}) = f(P_{j_0}) = P_{j_0+1},$$

y similarmente, $P_{i_0+l} = P_{j_0+l}$ para todo $l \geq 0$. Por lo tanto, la sucesión P_i es periódica con periodo $j_0 - i_0$ (o posiblemente un divisor de $j_0 - i_0$). La figura que describe éste proceso se parece a la letra griega ρ, de ahí el nombre *método ρ de Pollard.* Si f es una función aleatoria elegida aleatoriamente, (no la precisaremos), entonces esperamos encontrar una coincidencia con j_0 a lo más una constante $\sqrt{N}$ veces . Para un análisis del tiempo de ejecución para varias elecciones de f.

Una implementación ingenua del método almacena todos los puntos P_i hasta encontrar una coincidencia. Esto toma alrededor de $\sqrt{N}$ de almacenamiento. Sin embargo, es posible hacerlo mucho mejor a costa de muy poco cálculo. La clave es que una vez encontrada la coincidencia para dos índices que difieren en d, todos los índices subsecuentes que difieran en d nos darán coincidencias. Esta es la periodicidad mencionada arriba. Por lo tanto, podemos calcular pares (P_i, P_{2i}) para $i = 1, 2, \ldots$, para solo quedarnos con el par

actual; no almacenamos los pares previos. Estos se pueden calcular mediante la regla

$$P_{i+1} = f(P_i), \quad P_{2(i+1)} = f(f(P_{2i})).$$

Supongamos que $i \geq i_0$ e i es un múltiplo de d. Entonces los índices $2i$ e i difieren en un múltiplo de d y obtenemos la coincidencia $P_i = P_{2i}$. Como $d \leq j_0$ e $i_0 < j_0$, se sigue fácilmente que existe una coincidencia para $i \leq j_0$. Por lo tanto, se espera que el número de pasos para encontrar una coincidencia sea a lo más un múltiplo constante de $\sqrt{N}$.

Otro método para encontrar una coincidencia es almacenar solo los puntos P_i que satisfacen cierta propiedad (llamémosles "puntos distinguidos"). Por ejemplo, podríamos requerir que los últimos k bits de la representación binaria de la x-coordenada sean 0. Entonces almacenamos, como promedio, uno de cada 2^k puntos P_i. Supongamos que hay una coincidencia $P_i = P_j$ pero P_i no es uno de estos puntos distinguidos. Esperamos que P_{i+l} sea distinguido para algún l con $1 \leq l \leq 2^k$, aproximadamente. Entonces $P_{j+l} = P_{i+l}$, y así encontramos una coincidencia entre puntos distinguidos con sólo un poco más de cálculo.

Pero, cómo elegir una función f apropiada?. Además de que f actúe de forma aleatoria, necesitamos poder extraer información útil de una coincidencia. Aquí una manera de hacerlo: Dividimos G en s subconjuntos disjuntos $S_1, S_2, \ldots, S_s$ de aproximadamente el mismo tamaño. Una buena elección para s aparenta ser alrededor de 20. Elegimos $2s$ enteros aleatoriamente a_i, b_i mód N. Sea

$$M_i = a_i P + b_i Q.$$

Finalmente definimos

$$f(g) = g + M_i \quad \text{si } g \in S_i.$$

La mejor manera de pensar en f es como dando una caminata aleatoria en G, siendo los posibles pasos los elementos M_i.

Finalmente, elegimos enteros aleatorios a_0, b_0 y sea $P_0 = a_0 P + b_0 Q$ el punto de partida para la caminata. Mientras calculamos los puntos P_j, también grabamos cómo estos puntos se expresan en términos de P y Q. Si $P_j = u_j P + v_j Q$ y $P_{j+1} = P_j + M_i$, entonces $P_{j+1} = (u_j + a_i)P + (v_j + b_i)Q$, luego $(u_{j+1}, v_{j+1}) = (u_j, v_j) + (a_i, b_i)$. Cuando encontremos una coincidencia

$P_{j_0} = P_{i_0}$, tendremos

$$u_{j_0}P + v_{j_0}Q = u_{i_0}P + v_{i_0}Q, \text{ por lo tanto } (u_{i_0} - u_{j_0})P = (v_{j_0} - v_{i_0})Q.$$

Si $\text{mcd}(v_{j_0} - v_{i_0}, N) = d$, se tiene

$$k \equiv (v_{j_0} - v_{i_0})^{-1}(u_{i_0} - u_{j_0}) \mod N/d.$$

Esto nos da d elecciones para k. Usualmente, d será pequeño, así que podemos intentar todas las posibilidades hasta tener $Q = kP$.

En aplicaciones de criptografía, N es usualmente primo, en cuyo caso $d = 1$ ó N. Si $d = N$, tendremos una relación trivial (los coeficientes de P y Q son múltiplos de N), así que empezamos de nuevo. Si $d = 1$, obtenemos k.

Un ejemplo

Sea $G = E(\mathbb{F}_{1093})$, donde E es la curva elíptica dada por $y^2 = x^3 + x + 1$. Usaremos $s = 3$.

Sean $P = (0, 1)$ y $Q = (413, 959)$.

Se puede mostrar que el orden de P es 1067. Queremos encontrar k tal que $kP = Q$. Sea

$$P_0 = 3P + 5Q, \quad M_0 = 4P + 3Q, \quad M_1 = 9P + 17Q, \quad M_2 = 19P + 6Q.$$

Sea $f : E(\mathbb{F}_{1093}) \to E(\mathbb{F}_{1093})$ definido por

$$f(x, y) = (x, y) + M_i \quad \text{si } x \equiv i \mod 3.$$

Aquí el número x es visto como un entero $0 \leq x < 1093$ y después es reducido mód 3. Por ejemplo,

$$f(P_0) = P_0 + M_2 = (727, 589),$$

pues $P_0 = (326, 69)$ y $326 \equiv 2 \mod 3$.

Podemos definir $f(\infty) = \infty$ si queremos. Sin embargo, si nos encontramos con $f(\infty)$, habremos encontrado una relación del tipo $aP + bQ = \infty$ y podemos encontrar k fácilmente (si la relación no es algo trivial como $1067P + 2134Q = \infty$). Por lo tanto, no nos preocupamos por el ∞.

Si calculamos P_0, $P_1 = f(P_0)$, ..., obtenemos

$$P_0 = (326, 69),\ P_1 = (727, 589),\ P_2 = (560, 365),\ P_3 = (1070, 260),$$
$$P_4 = (473, 903),\ P_5 = (1006, 951),\ P_6 = (523, 938), \ldots,$$
$$P_{57} = (895, 337),\ P_{58} = (1006, 951),\ P_{59} = (523, 938), \ldots$$

Por lo tanto, la sucesión empieza a repetirse en $P_5 = P_{58}$.

Si seguimos de cerca los coeficientes de P y Q en los cálculos, encontramos que

$$P_5 = 88P + 46Q \text{ y } P_{58} = 685P + 620Q.$$

Por lo tanto,

$$\infty = P_{58} - P_5 = 597P + 574Q.$$

Como P tiene orden 1067, calculamos

$$-574^{-1}597 \equiv 499 \mod 1067.$$

Por lo tanto, $Q = 499P$, luego $k = 499$.

Hemos almacenado todos los puntos $P_0, P_1, \ldots, P_{58}$ hasta encontrar una coincidencia. En cambio, repitamos el cálculo, pero calculando los pares (P_i, P_{2i}) y sólo almacenamos el par actual. Entonces encontramos que para $i = 53$ existe la coincidencia $P_{53} = P_{106}$. Así obtenemos

$$620P + 557Q = P_{53} = P_{106} = 1217P + 1131Q.$$

Por lo tanto, $597P + 574Q = \infty$, de donde $k = 499$, como antes. □

El *método λ de Pollard* usa una función f como en el método ρ, pero se usan varios puntos iniciales aleatorios $P_0^{(1)}, \ldots, P_0^{(r)}$. Así obtenemos sucesiones definidas por

$$P_{i+1}^{(l)} = f(P_i^{(l)}), \quad 1 \leq l \leq r, \quad i = 1, 2, \ldots.$$

Estas se pueden calcular con varias computadoras en paralelo. Los puntos que satisfacen ciertas condiciones son llamados distinguidos y son reportados a una computadora central. Cuando se encuentra una coincidencia entre las entradas de las computadoras, tenemos una relación que nos debería permitir resolver el problema de log discreto, como en el método ρ. Cuando hay una coincidencia

entre dos sucesiones, estas dos sucesiones coincidirán siempre a partir de ese punto. Sólo necesitamos ver puntos distinguidos porque los puntos distinguidos deberían ocurrir poco después de una coincidencia.

Cuando sólo hay dos puntos iniciales aleatorios, tenemos dos caminatas aleatorias. Eventualmente estas tendrán un punto en común, y coincidirán de ahí en adelante. La figura de este proceso se asemeja a la letra griega λ, de ahí el nombre.

A veces el método λ se describe en términos de canguros saltando alrededor de un campo (esta es la caminata aleatoria). Una variante del método λ con dos caminatas graba uno de cada 10 puntos, por ejemplo, en la primera sucesión y luego comprueba si la segunda sucesión coincide con cualquiera de estos puntos. En este caso, la primera sucesión es llamada un canguro domado, y la segunda un canguro salvaje. La idea es usar el canguro domado para atrapar al canguro salvaje.

Se espera que el método λ encuentre una coincidencia en a lo más una constante veces $\sqrt{N}$ pasos. Si se corre en paralelo con varios puntos iniciales, el tiempo se puede mejorar significativamente.

Finalmente, debemos señalar que los métodos ρ y λ son *probabilísticos*, lo que significa que existe una gran probabilidad de que vayan a terminar dentro del tiempo estipulado, pero esto no está garantizado.

5.2. Fracciones Continuas

Cualquier número real x puede representarse por fracciones continuas de la forma siguiente:

$$x = a_0 + \cfrac{z_1}{a_1 + \cfrac{z_2}{a_2 + \cfrac{z_3}{a_3 + \cdots}}}$$

donde $a_0 \in \mathbb{Z}, ai \in \mathbb{N}$, para $i \geq 1$.

Si $z_i = 1$ para todo i esto es llamado una fracción continua simple y a menudo es escrita como sigue:

$$x = [a_0, a_1, a_2, \cdots]$$

El siguiente algoritmo, conocido como algoritmo de las fracciones continuas, nos da la pauta para encontrar la representación de una fracción continua simple.

Algoritmo : Representación de una fracción continua

La entrada: número real x.

La respuesta: representación de $\sqrt{x}$ como fracción continua $[a_0, a_1, a_2, \cdots]$.

$$\begin{array}{rclrcl} & & & x_0 & = & \sqrt{x} \\ a_0 & = & \lfloor x_0 \rfloor & x_1 & = & \frac{1}{x_0 - a_0} \\ a_1 & = & \lfloor x_1 \rfloor & x_2 & = & \frac{1}{x_1 - a_1} \\ & \vdots & & & \vdots & \\ a_i & = & \lfloor x_i \rfloor & x_{i+1} & = & \frac{1}{x_i - a_i} \\ & \vdots & & & \vdots & \end{array}$$

Debemos resaltar que x es racional si y sólo si su fracción continua es finita.

El número racional

$$\frac{m_i}{n_i} = [a_0, a_1, a_2, \cdots, a_i]$$

se llama el i-ésimo convergente de x.

Se tiene que x es el límite de la sucesión de racionales $\frac{m_i}{n_i}$ y de hecho proporcionan una aproximación óptima a x en el sentido de que los denominadores son primos relativos.

El numerador y el denominador se pueden obtener recursivamente de la siguiente forma:

$$\begin{aligned}
\frac{m_0}{n_0} &= \frac{a_0}{1} \\
\frac{m_1}{n_1} &= \frac{a_0 a_1 + 1}{a_1} \\
&\vdots \\
\frac{m_i}{n_i} &= \frac{a_i m_{i-1} + m_{i-2}}{a_i n_{i-1} + n_{i-2}}, \quad i \geq 2
\end{aligned}$$

Una vez calculada la fracción continua simple se procede en el algoritmo a calcular el valor de $\frac{m_i}{n_i}$, el i-ésimo convergente de $\sqrt{N}$ y se calcula $b_i \equiv m_i^2($ mód $N)$. De los valores b_i resultantes sólo se tomarán aquellos cuyos factores primos sean menores a una cota C, el valor máximo de la cota usualmente usado es de 10000, claro que esto depende del valor que se desea factorizar pues un número de 20 dígitos puede ser factorizado con una cota no menor a 3000, pero entre mayor sea la cota utilizada el Algoritmo de factorización tardará un poco más en obtener la factorización del número.

Factorización con Fracciones Continuas

En 1975, Michael A. Morrizon y Jhon Billhart desarrollaron el método de Fracciones Continuas, que les permitió factorizar el séptimo número de Fermat $F_7 = 2^{2^7} + 1$. Para poder comprender este método de factorización recordaremos algunos conceptos relacionados con el tema de Fracciones continuas.

Una vez calculada la fracción continua simple se procede, a calcular el valor de $\frac{m_i}{n_i}$, el i-ésimo convergente de $\sqrt{N}$ y se calcula $b_i \equiv m_i^2($ mód $N)$. De los valores b_i resultantes sólo se tomarán aquellos cuyos factores primos sean menores a una cota C, el valor máximo de la cota usualmente usado es de 10000, claro que esto depende del valor que se desea factorizar pues un número de 20 dígitos puede ser factorizado con una cota no menor a 3000, pero entre mayor sea la cota utilizada el Algoritmo siguiente tardará un poco más en obtener la factorización del número.

5.3. Algoritmos de Factorización con Fracciones Continuas

Esta técnicas de factorización utilizando las Fracciones Continuas(CFRAC) es de velocidad exponencial; la técnica de la Criba Cuadrática(QS) introducida en las décadas de los ochenta por Carl Pomerance y la técnica de la criba del campo numérico introducida por A.K. Lenstra, H.W Lenstra Jr.,J.M. Pollard y C. Pomerance en la década de los noventa.
Todas estas técnicas se basan en la técnica de factorización introducida por Fermat que pretende la búsqueda de una relación de la forma

$$X^2 \equiv Y^2(\mod N).....(1)$$

siendo que N es el número que se quiere factorizar.
De esta relación se demuestra que $x^2 - Y^2 = kN$ es decir que

$$(x-y)(x+y) = kN.....(2)$$

y buscando mediante el algoritmo de Euclides el máximo común divisor de $(x+y)$ con N y $(x-y)$ con N tenemos un cincuenta por ciento de probabilidad de que el resultado sea un factor de N.
Los métodos señalados dedican todos sus esfuerzos en lograr hallar una relación como la recogida en la ecuación del inicio. Todos ellos buscan métodos de generación de relaciones

$$P^2 \equiv C(\mod N).....(3)$$

que verifiquen que C tiene todos sus divisores primos menores que un valor límite prefijado B . Una vez encontradas y almacenadas suficientes relaciones de esta forma (3) se buscan subconjuntos de estas relaciones halladas, que llamaremos subconjuntos S , que verifiquen que el producto de todos los C de cada uno de esos subconjuntos sea cuadrado perfecto.
Para la búsqueda de los diferentes subconjuntos S se utiliza la técnica de eliminación gaussiana. El producto de todos los elementos de uno subconjunto S nos llevará a una expresión de la forma

$$\prod P_i^2 \equiv \prod C_i(\mod N)....(4)$$

donde al haberse logrado que la parte derecha de la congruencia sea cuadrado perfecto, tendremos que la expresión (4) es análoga a la (1). De los tres algoritmos subexponenciales, cada nuevo que apareció superaba en velocidad al anterior a partir de un tamaõ del entero a factorizar. Entre ellos se diferencian fundamentalmente en el modo de buscar relaciones de la forma (3), y el factor diferenciador de cada uno de ellos reside en el modo de lograr cuánto antes una cantidad suficiente de esas relaciones donde sus respectivos valores C tengan todos sus divisores primos menores que un límite B prefijado.
Una vez hallada una relación de esa forma se debe probar, como acabamos de decir, que C tiene todos sus divisores primos menores que un valor límite prefijado; si no es así, la relación es desechada y se pasa a probar una nueva relación. Por tanto, ganamos velocidad si

1. generamos nuevas relaciones de forma más rápida;

2. generamos relaciones que tengan mayor probabilidad de que C sea tenga todos sus divisores primos por debajo del límite marcado; y esa probabilidad aumenta a medida que se logran valores de C menores (principal ventaja del método CFRAC); o a medida que se logra determinar con cierta probabilidad que esa propiedad se cumple antes de generar el valor de C (principal ventaja del método QS);

3. ganaremos velocidad si optimizamos el código implementado.

Este tercer camino,el de la optimización del código, tomar una implementación de un algoritmo de factorización e intentar reducir los tiempos de ejecución. Para este proceso, debemos escoger uno de los algoritmos con los que vamos a trabajar, implementarlo, y posteriormente optimizarlo. Cuál de los tres algoritmos escoger? Nosotros hemos implementado y posteriormente optimizado el algoritmo basado en el desarrollo de las fracciones continuas.

Los métodos QS MPQS y NFS aventajan en velocidad al algoritmo CFRAC. Como podemos ver en [Stin00], los tiempos asintóticos para los algoritmos de factorización posteriores al CFRAC son:
QS, su comportamiento queda definido con los parámetros de la funciń $Ln(g;c)$ de valores $c = 1 + O(1)$ y $g = 12$.
NFS: Su comportamiento queda definido con los parámetros de la función

$Ln(g;c)$ de valores $c = 1.923 + O(1)$ y $g = 13$.
De estos valores, se deduce que cada algoritmo presentado supera en velocidad a anterior. De todas formas, se debe tener en cuenta que estas expresiones reflejan un comportamiento asintótico y, como recuerdan algunos autores , hay tamaños de enteros (rango entre 100 y 150 dígitos) donde QS o CFRAC pueden resultar más rápido que NFS.
Esta notación consigue clasificar muy bien a los algoritmos que están cerca de ser polinómicos. Si se quiere hacer una equivalencia entre la Onotación y la función $Ln(g;c)$, diremos que una función de tiempo subexponencial es aquella cuyo comportamiento asintótico es de la forma $f(n)$, donde $f(n) = O(n)$.

Mejoras en el rendimiento del código. Optimización

Hemos hablado de las medidas teóricas de la complejidad de nuestros algoritmos de factorización. Disponemos también de herramientas que nos permiten medir el número exacto de instrucciones ejecutadas en un determinado proceso. También podemos obtener el número de ciclos que ha empleado un proceso hasta llegar a su término.
Entendemos por productividad la cantidad de trabajo hecho en un cierto tiempo. La productividad la medimos con el parámetro rendimiento. Principalmente, cuando hablamos del rendimiento de un procesador nos ocupamos del tiempo de respuesta o tiempo de ejecución. El tiempo es la medida del rendimiento del ordenador. Patterson y Hennessy, traen una ecuación del rendimiento, que recoge la medida del tiempo de ejecución: Según señalan ambos autores, el tiempo de ejecución de una aplicación depende de tres factores: El número de instrucciones que se ejecutan (NI); el promedio de ciclos empleados en cada instrucción (CPI); el tiempo de ciclo (que depende de la frecuencia de reloj del ordenador: T ciclo).

Optimizar un código consiste en modificarlo para lograr que realice la misma tarea en un tiempo menor. En el proceso de optimización debemos tener en cuenta la ley de AMDHAL:
cuando realizamos una mejora en una función dentro de la aplicación, el peso

de esa mejora será proporcional al peso que tenía esa función antes de ser optimizada: no compensa mejorar lo que apenas tiene peso en un proceso.
Las dos vías de mejora que hemos seguido a lo largo de nuestra investigación para lograr la optimización de nuestro código son: análisis del software del algoritmo en C (mejora por software); y análisis de la interacción entre el código compilado y la máquina (mejora por hardware).
El primero de los dos pasos consiste en realizar búsqueda de algoritmos mejores que los utilizados en los procesos que deseamos optimizar: podemos encontrar algoritmos que realicen el mismo proceso en un tiempo menor.

5.4. Descripción del Algoritmo de Factorización por la técnica de las Fracciones Continuas

Sea $N > 1$, un entero compuesto impar. El procedimiento a seguir en el método de factorización de CFRAC es el que sigue:

1. Mediante el algoritmo, basado en la técnica de las fracciones continuas, buscamos una expresión racional, aproximada del valor real $\sqrt{N}$ ó $\sqrt{kN}$, para algún $k > 1$ entero. En el proceso generamos cinco secuencias $(A_i, B_i, C_i, P_i \ y \ Q_i)$, donde la fracción $\frac{P_i}{Q_i}$ ofrece la aproximación iésima en forma racional del valor real $\sqrt{N}$ .En cada nueva iteración, el valor $\frac{P_i}{Q_i}$ queda más próximo al valor de $\sqrt{N}$. En el desarrollo del algoritmo de generación de estas cinco secuencias, se verifica la siguiente expresin:

$$P_{n-1}^2 - kNQ_{n-1}^2 = (-1)^n C_n$$

y si aplicamos el operador $mod N$ a ambas partes de la igualdad, tenemos que

$$P_{n-1}^2 \equiv (-1)^n C_n(\quad \text{mód } N)$$

Expresión que es de la misma forma que la recogida en (3). Llamaremos, a cada par de enteros positivos (P_{n-1}, C_n) en esta congruencia, un **par** PC.

2. El segundo paso consiste en encontrar, de entre el conjunto de pares $P - C$, ciertos subconjuntos, que llamaremos conjuntos S, cada uno de

ellos con la propiedad de que el producto

$$\Pi(-1)^i C_i$$

de todos los C_is de cada conjunto S sea un valor cuadrado perfecto. Si no se encuentra ningún subconjunto de esas características, entonces se debe volver al paso 1 en busca de más pares $P - C$.

3. Cada conjunto S encontrado en el paso anterior verifica la congruencia

$$P^2 \equiv \prod P_i^2 \equiv \Pi(-1)^i C_i = C^2 (\mod N),$$

donde $1 \leq P < N$. Calculamos los valores de P y de C y $mcd(P-C, N) = D$, para los diferentes conjuntos S hallados. Si en algún conjunto S tenemos que $1 < D < N$, entonces el método ha logrado encontrar un factor no trivial de N , que es precisamente D.

Distribución de los números primos

Este tema es lo suficientemente complejo para que este interés, se haya podido mantener hasta la actualidad, el conjunto de los números primos obedece a un orden asimétrico, por lo que siempre ha resultado un problema entender sus leyes de distribución.

En 1859, Riemann abrió una nueva puerta para la investigación de los números primos y las investigaciones sobre números primos se dividieron en dos líneas una que se encarga de ver la cantidad de números primos hasta un número dado (Teorema de los Números Primos) y la otra que busca probar primalidad.

La función zeta forma parte de la primera línea, logrando una fusión entre la aritmética y el análisis.

Riemann trató de encontrar una fórmula o un método simple para calcular la cantidad de primos hasta un número x, y aunque no cumplió su cometido sentó las bases para que posteriormente se pudiesen lograr algunas aproximaciones al Teorema de los Números Primos.

Definición:[Función de Euler]Se define la función

$$\pi : \mathbb{R} \to \mathbb{N}, \quad \pi(x) = \sum_{p \leq x} 1$$

donde el índice p varía sólo en el conjunto de números primos.

Este resultado fue conjeturado por varios matemáticos, entre ellos, por Legendre y Gauss, pero la demostración completa sólo fue presentada en 1896, independientemente por De la Vallée Poussin y Hadamard. No daremos la demostraremos este teorema, pero demostraremos del siguiente resultado debido a Tchebycheff que es claramente una versión débil del TNP.

Tenemos que

$$\binom{2n}{n} < \sum_{0\le k\le 2n} \binom{2n}{k} = 2^{2n},$$

y como $\binom{2n}{n} = \frac{(2n)!}{n!n!}$ es múltiplo de todos los primos p que satisfacen $n < p \le 2n$, se sigue que el producto de los primos entre n y $2n$ es menor que 2^{2n}. Como hay $\pi(2n) - \pi(n)$ primos como esos, se sigue que $n^{\pi(2n)-\pi(n)} < 2^{2n}$ (pues todos eso primos son mayores que n), donde $(\pi(2n) - \pi(n)) \log n < 2n \log 2$ y

$$\pi(2n) - \pi(n) < \frac{2n \log 2}{\log n}.$$

Eso implica fácilmente, por inducción, que

$$\pi(2^{k+1}) \le \frac{5 \cdot 2^k}{k}$$

(a partir de $k = 5$; hasta $k = 5$ se sigue de $\pi(n) \le n/2$). De ahí se sigue que si $2^k < x \le 2^{k+1}$ entonces

$$\pi(x) \le \frac{5 \cdot 2^k}{k} \le \frac{5x \log 2}{\log x}$$

pues $f(x) = x \log / \log x$ es una función creciente para $x \ge 3$.

Vamos ahora a probar otra desigualdad. El exponente del primo p en la factorización de $n!$ es

$$\begin{aligned} \omega_p(n) &= \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \cdots \\ &= \sum_{k=1}^{\infty} \left\lfloor \frac{n}{p^k} \right\rfloor \end{aligned}$$

(esta es una suma finita pues si $k > \log_p n = \log n / \log p$ entonces $\left\lfloor \frac{n}{p^k} \right\rfloor = 0$). De hecho, $\left\lfloor \frac{n+1}{p^j} \right\rfloor - \left\lfloor \frac{n}{p^k} \right\rfloor$ es siempre 0 ó 1, y es igual a 1 si, y sólo si, p^j divide

$n+1$. Así, $\omega_p(n+1) - \omega_p(n)$ es igual al exponente de p en la factorización de $n+1$, lo que provee una prueba por inducción del hecho arriba.

Así, el exponente de p en $\binom{2n}{n} = (2n)!/n!^2$ es

$$\sum_{k=1}^{\infty}\left(\left\lfloor\frac{2n}{p^k}\right\rfloor - 2\left\lfloor\frac{n}{p^k}\right\rfloor\right).$$

Tenemos ahora que $\left\lfloor\frac{2n}{p^k}\right\rfloor - 2\left\lfloor\frac{n}{p^k}\right\rfloor$ es siempre 0 ó 1 (pues $0 \leq x - \lfloor x\rfloor < 1$ para todo x), donde el exponente de p en $\binom{2p}{p}$ es como máximo $\log_p n = \log n/\log p$ para todo primo p. Por otra parte, si $n < p \leq 2n$, el exponente de p en $\binom{2p}{p}$ es 1. Así, si $\binom{2n}{n} = \prod_{p<2n} p^{\alpha_p}$ es la factorización de $\binom{2n}{n}$ entonces

$$\begin{aligned}
\log\binom{2n}{n} &= \sum_{p<2n} \alpha_p \log p \\
&= \sum_{p\leq n} \alpha_p \log p + \sum_{n<p\leq 2n} \log p \\
&\leq \pi(n)\log n + (\pi(2n) - \pi(n))\log(2n) \\
&\leq \pi(2n)\log(2n)
\end{aligned}$$

donde

$$\pi(2n) \geq \log\binom{2n}{n} * \log(2n) \geq n\log 2/\log(2n)$$

para todo x par, lo que implica la misma estimación para todo x entero, pues $\pi(2k-1) = \pi(2k)$.

Se tiene el siguiente resultado: Dada $f : \mathbb{N} \to [0, +\infty)$ una función decreciente. La serie

$\sum_{p \text{ primo}} f(p)$ converge si, y sólo si, la serie $\sum_{n=2}^{\infty} \frac{f(n)}{\log n}$ converge.

En particular, $\sum_{p \text{ primo}} \frac{1}{p} = +\infty$.

Una aproximación más precisa para $\pi(x)$ es dada por

$$\mathrm{Li}(x) = \int_0^x \frac{dt}{\log t},$$

donde tomamos el valor principal de esta integral, o sea,

$$\mathrm{Li}(x) = \lim_{\varepsilon\to 0}\int_{\varepsilon}^{1-\varepsilon} \frac{dt}{\log t} + \int_{1+\varepsilon}^{x} \frac{dt}{\log t};$$

claramente

$$\lim_{x\to\infty} \frac{\mathrm{Li}(x)}{\log(x)/x} = 1.$$

Se sabe, en tanto, que

$$|\pi(x) - \mathrm{Li}(x)| \leq Cxe^{-a(\log x)^{3/5}(\log\log x)^{-1/5}}$$

para algún valor de las constantes a y C (independientes de x). En particular, para cualquier $k > 0$ existe $C > 0$ tal que para todo x,

$$|\pi(x) - \mathrm{Li}(x)| \leq C\frac{x}{(\log x)^k},$$

lo que muestra que $\mathrm{Li}(x)$ (y aún $x/(\log x - 1)$) es una aproximación de $\pi(x)$ mejor que $x/\log x$.

La hipótesis de Riemann, ya mencionada, equivale a decir que para todo $\varepsilon > 0$ existe C con

$$|\pi(x) - \mathrm{Li}(x)| \leq Cx^{1/2+\varepsilon};$$

nadie aún puede demostrar que esta estimación sea correcta, incluso para algún valor de $\varepsilon < 1/2$. La hipótesis de Riemann también implica que existe C con

$$|\pi(x) - \mathrm{Li}(x)| \leq Cx^{1/2}\log x,$$

lo que daría una estimación de la magnitud de este error de forma mucho mejor de las que se saben demostrar. Por otra parte, se sabe demostrar que no puede existir ninguna estimación mucho mejor que esta para $|\pi(x) - \mathrm{Li}(x)|$: existen una constante $C > 0$ y enteros x_1 y x_2 arbitrariamente grandes con

$$\pi(x_1) - \mathrm{Li}(x_1) < -C\frac{\sqrt{x_1}\log\log\log x_1}{\log x_1}$$
$$\pi(x_2) - \mathrm{Li}(x_2) > C\frac{\sqrt{x_2}\log\log\log x_2}{\log x_2}.$$

Existen varios refinamientos conocidos del teorema de Dirichlet. Definimos $\pi_{d,a}(x)$ como el número de primos de la forma $a+dn$ en el intervalo $[2, x]$. De la Vallée Poussin probó que

$$\lim_{x\to+\infty} \frac{\pi_{a,d}(x)}{\pi(x)} = \frac{1}{\varphi(d)},$$

esto es, todas las posibles clases módulo d tienen aproximadamente la misma proporción de primos.

Por otro lado, Tchebycheff observó que para valores pequeños de x, $\pi_{3,2}(x) - \pi_{3,1}(x)$ y $\pi_{4,3}(x) - \pi_{4,1}(x)$ son positivos, después Littlewood, demuestró que estas funciones cambian de signo infinitas veces.
En 1957, Leech demostró que el menor valor de x para el cual $\pi_{4,3}(x) - \pi_{4,1}(x) = -1$ es 26861 y en 1978.
Luego, Bays y Hudson demostraron que el menor valor de x para el cual $\pi_{3,2}(x) - \pi_{3,1}(x) = -1$, es 608981813029.

Sean $p(d, a)$ el menor primo de la forma $a + dn$, con n entero, y

$$p(d) = \text{máx}\{p(d, a) \mid 0 < a < d,\ \text{mcd}(a, d) = 1\}.$$

En 1944, Linnin demostró que existe $L > 1$, con $p(d) < d^L$ para todo d suficientemente grande; pero la mejor estimación conocida para L es $L \leq 5.5$ debida a Heath y Brown en 1992, que también conjeturaron que

$$p(d) \leq Cd(\log d)^2.$$

Por otra parte, no se sabe demostrar que existen infinitos primos de la forma $n^2 + 1$, de hecho, no existe ningún polinomio P en una variable y de grado mayor que 1 para el cual se sepa demostrar que existen infinitos primos de la forma $P(n)$, $n \in \mathbb{Z}$. Por otra parte, existen muchos polinomios en mas de una variable que asumen infinitos valores primos, por ejemplo, se prueba fácilmente que todo primo de la forma $4n+1$ puede ser escrito también en la forma a^2+b^2, $a, b \in \mathbb{Z}$. Por otra parte, existen infinitos primos de la forma $a^2 + b^4$.

Uno de los problemas abiertos más famosos de la matemática es la conjetura de Goldbach: todo número par mayor o igual a 4 es la suma de dos números primos. Chen demostró que todo número par suficientemente grande es la suma de un primo con un número entero con a lo más dos factores primos. Vinogradov demostró que todo impar suficientemente grande (por ejemplo, mayor que $3^{3^{15}}$) es una suma de tres primos.

No hay ninguna fórmula simple para generar primos arbitrariamente grandes. Existen fórmulas que generan números primos, mas son todas complicadas y no ayudan mucho ni a generar números primos explícitamente ni a responder preguntas teóricas sobre la distribución de los primos.

La inutilidad de esta última fórmula viene del hecho que para calcular c debemos encontrar todos los primos; la fórmula sería más interesante si hubiera otra interpretación para el número real c, lo que parece muy improbable. Por otra parte, existe un número real $\alpha > 1$ tal que $\lfloor a^{3^n} \rfloor$ es siempre primo.

Proposición 5.4.1

Sea $n > 1$. Si para cada factor primo q de $n-1$ existe un entero a_q tal que $a_q^{n-1} \equiv 1 \pmod{n}$ y $a_q^{(n-1)/q} \not\equiv 1 \pmod{n}$, entonces n es primo.

Prueba.-

Sea q^{k_q} la mayor potencia de q que divide $n-1$. El orden de a_q en $(\mathbb{Z}/(n))^*$ es un múltiplo de q^{k_q}, donde $\varphi(n)$ es un múltiplo de q^{k_q}. Como esto vale para todo factor primo q de $n-1$, $\varphi(n)$ es un múltiplo de $n-1$ y n es primo.

Proposición 5.4.2

Si $n-1 = q^k R$, donde q es primo y existe un entero a tal que $a^{n-1} \equiv 1 \pmod{n}$ y $\mathrm{mcd}\left(a^{(n-1)/q} - 1, n\right) = 1$, entonces cualquier factor primo de n es congruente a 1 módulo q^k.

Prueba.-

Si p es un factor primo de n, entonces $a^{n-1} \equiv 1 \pmod{p}$ y p no divide $a^{(n-1)/q} - 1$, donde $\mathrm{ord}_p a$, el orden de a módulo p, divide $n-1$ mas no divide $(n-1)/q$. Así, $q^k \mid \mathrm{ord}_p a \mid p-1$, donde $p \equiv 1 \pmod{q}^k$.

Corolario 5.4.4

Si $n-1 = FR$, con $F > R$ y para todo factor primo q de F existe $a > 1$ tal que $a^{n-1} \equiv 1 \pmod{n}$ y $\mathrm{mcd}\left(a^{(n-1)/q} - 1, n\right) = 1$, entonces n es primo.

Prueba.-

Sean q un factor primo de F y q^k la mayor potencia de q que divide F; por la proposición anterior, todo factor primo de n debe ser congruente a 1 módulo q^k. Como esto vale para cualquier factor primo de F, se sigue que cualquier factor primo de n debe ser congruente a 1 módulo F. Como $F > \sqrt{n}$, entonces n es primo.

De hecho, basta conocer un conjunto de factores primos cuyo producto sea menor que $(n-1)^{1/2}$ para, con ayuda el resultado de Pocklington, intentar demostrar la primalidad de n (lo que dejamos como ejercicio). Los siguientes criterios son consecuencias directas de las proposiciones anteriores.

Fermat conjeturó que todo número de la forma $F_n = 2^{2^n} + 1$ era primo, y se verifica la fórmula para $n \leq 4$. Observe que $2^n + 1$ (y en general $a^n + 1$, con $a \geq 2$) no es primo si n no es una potencia de 2; si p es un factor primo impar de n, podemos escribir $a^n + 1 = b^p + 1 = (b+1)(b^{p-1} - b^{p-2} + \cdots + b^2 - b + 1)$, donde $b = a^{n/p}$. Euler mostraría más tarde que F_5 no es primo (tenemos $F_5 = 4294967297 = 641 \cdot 6700417$) y ya se demostró que F_n es compuesto para varios otros valores de n; ningún otro primo de la forma $F_n = 2^{2^n} + 1$ es conocido, mas se conocen muchos primos (algunos bastante grandes) de la forma $a^{2^n} + 1$, que son conocidos como primos de Fermat generalizados. La prueba a seguir muestra como probar eficientemente la primalidad de F_n.

Corolario 5.4.4 Sea $F_n = 2^{2^n} + 1$. Luego F_n es primo si, y sólo si,

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n}$$

. **Prueba.-**
Si $3^{(F_n-1)/2} \equiv -1 \pmod{F_n}$, entonces la primalidad de F_n se sigue de la Proposición ??. Por otra parte, si F_n es primo entonces $3^{(F_n-1)/2} \equiv \left(\frac{3}{F_n}\right) = \left(\frac{F_n}{3}\right) = \left(\frac{2}{3}\right) = -1 \pmod{F_n}$.

Corolario 5.4.5
Sea $n = h \cdot 2^k + 1$, con $2^k > h$. Luego, n es primo si, y sólo si, existe un entero a, con $a^{(n-1)/2} \equiv -1 \pmod{n}$.
Prueba.-
Si n es primo, podemos tomar a cùalquiera con $\left(\frac{a}{n}\right) = -1$; es decir, la mitad enteros entre 1 y $n-1$ sirve como a. La recíproca se sigue del Corolario con $F = 2^k$.

Corolario 5.4.6
Sea $n = h \cdot q^k + 1$, con q primo y $q^k > h$. Luego, n es primo si, y sólo si, existe

un entero a, con $a^{n-1} \equiv 1 \pmod{n}$ y $\mathrm{mcd}\left(a^{(n-1)/q} - 1, n\right) = 1$.

Prueba.-

Si n es primo, podemos tomar a cualquiera que no sea de la forma x^q módulo n; es decir, una proporción de $(q-1)/q$ de entre los enteros entre 1 y $n-1$ sirve como a. La recíproca se sigue del Corolario con $F = q^k$.

Una gran mayoría de los 100 mayores primos conocidos están en las condiciones del teorema de Proth (ver tablas). Esto se debe al hecho de que los primos de esa forma son frecuentes (más frecuentes que, por ejemplo, los primos de Mersenne) y que su primalidad es fácilmente demostrada usando este resultado.

Búsqueda de elementos que forman el conjunto S

Encontrar relaciones o pares $P - C$ es tarea relativamente sencilla. La dificultad del proceso empieza ahora que podemos tener almacenadas un montón de ellas: Cómo entresacar, de todo el vasto conjunto de relaciones, estos subconjuntos S de pares $P - C$ que verifiquen que el producto de todos sus correspondientes C_is resulte un cuadrado perfecto? Cómo saber siquiera si existe uno solo de esos subconjuntos?

Por suerte se conoce un método que podria dar respuesta a esta interrogante. Existe una forma de detectar subconjuntos S dentro de la enormidad de posibles pares $P-C$ candidatos. Para ello debemos exigir una fuerte restriccin a los posibles candidatos a formar parte de esos subconjuntos: únicamente podrán formar parte de un subconjunto S aquellos pares $P - C$ para los que logremos conocer todos los factores de su correspondiente C_i .

5.5. La Criba Cuadrática

Carl Pomerance introdujo una variación a la criba lineal y definió lo que ha dado en llamarse la Criba Cuadrática (popularmente conocido como QS). Fue Richard Schroeppel el creador de la búsqueda de congruencias $u^2 \equiv v(modn)$,conocido como **método de la criba lineal**. La aportación principal de Schroeppel es una ingeniosa forma de reemplazar el tiempo invertido en las pruebas de división por otro en principio más breve: el tiempo necesario para realizar una criba como la de Eratóstenes. Su idea era encontrar

otros modos de producir residuos cercanos a $\sqrt{n}$ que tuviesen la ventaja de que pudieran factorizarse colectivamente mediante la criba.
El algoritmo de Schroeppel el siguiente. Sea $K = \lfloor \sqrt{u} \rfloor$ y consideremos la funcin:

$$S(A, B) = (K + A)(K + B) - n$$

donde A y B son enteros mucho menores, en valor absoluto, que K . Por ese motivo tendremos que $S(A, B)$ no será mucho mayor que K. La idea vuelve a ser intentar encontrar pares (Ai, Bi) tales que

$$\prod_{i=1}^{k} S(Ai, Bi)$$

sea un cuadrado, como con el algoritmo de CFRAC. Además, se debe lograr que cada valor distinto de A1...Ak , B1...Bk sea asumido un número par de veces. Así entonces

$$\prod_{i=1}^{k} (K + Ai)(K + Bi)$$

será también un cuadrado.
Por tanto, si podemos encontrar tales colecciones fortuitas de pares Ai, Bi , entonces podemos encontrar enteros u y v tales que

$$u^2 \equiv v^2 (mod n)$$

El procedimiento para realizar la criba es tomar un valor fijo $A = Ao$ y variar B sobre enteros consecutivos.
Estos números forman una progresión aritmética de tal manera que si $p|S(Ao, Bo)$, entonces $p|S(Ao, Bo + p), p|S(Ao, Bo + 2p)$,...
En esta criba, en lugar de ir tachando los números múltiplos de cada primo que entra en juego en el proceso de la criba de Eratoóstenes,podemos ir dividiendo los compuestos por los sucesivos primos que los forman, llegando a transformar todos los compuestos al número 1 al final del proceso, cuando ya estén completamente factorizados. Si, además, en lugar de cribar por todos los primos hasta llegar a un valor de B igual a la raíz cuadrada del rango de la criba, lo hacemos sólo hasta el valor primo menor y más próximo al límite de suavidad tendremos a 1, al final del proceso, únicamente a aquellos valores que sean suaves.

Pero no todos los valores suaves quedarán reducidos al valor 1 al final del proceso: tenemos que considerar los casos en que un determinado valor $S(Ao, Bi)$ verifique que un mismo primo lo factorice más de una vez: tenemos el problema de las potencias de primos menores que el límite superior de suavidad. Debemos rectificar el método cribando también con estas potencias de primos menores que B. Gracias a la criba, conocemos de antemano qué valores de B tendrán $S(Ao, B)$ divisibles por los primos p de la base de factores: los que después de la criba hayan quedado a 1 son valores de B para los que $S(Ao, B)$ es suave. Y entonces no debemos hacer un derroche de intentos de división en aquellos valores donde de antemano sabemos que no son suaves. Esto supondrá una ventaja importante sobre el algoritmo CFRAC, donde los valores a factorizar surgen de modo aparentemente aleatorio y no es posible realizar un proceso de criba.
La ventaja de la criba lineal de Schroeppel sobre CFRAC es que un proceso de criba sustituye el proceso de intentos por división. La desventaja es que el método de Schroeppel produce residuos que no necesariamente son cuadráticos.

La criba cuadrática

Dado $K = \lfloor\sqrt{u}\rfloor$, tomamos $S(A) = (K - A)^2 - n$, donde $S(A) = S(A, A)$
El método seguido en QS es considerar el polinomio

$$Q(a) = (\lfloor\sqrt{u}\rfloor + a)^2 - n$$

o, lo que es lo mismo, $Q(a) \equiv x^2(modn)$, donde $x = \lfloor\sqrt{u}\rfloor + a$
A primera vista, este es un método como otro cualquiera para buscar congruencias, no aporta en principio valor alguno al definido con CFRAC; pero el punto crucial, que marca la verdadera diferencia, es que con QS no es necesario factorizar todos los valores $Q(a)$ sobre la base de factores.
Aquí, $Q(a)$ es un polinomio con coeficientes enteros, que podemos usar para realizar una criba.
Veamos cómo funciona: Supongamos cierto número m del que sabemos que $m|Q(a)$. Entonces, para cada entero k , se cumple automáticamente que $m|Q(a + km)$. Para encontrar un valor de a (si existe) para el que se verifique que $m|Q(a)$ basta resolver la expresión $x^2 \equiv n(modm)$ y tomando luego el valor $a = x - \lfloor\sqrt{u}\rfloor modm$.

Al hacer la criba, sin peligro de perder generalidad, podemos tomar la restricción con las potencias de factores primos $m = p^k$. Si p es un primo impar, entonces $x^2 \equiv n(mod\ \ p^k)$ tiene solución (en concreto dos) si y solo si $(n/p) = +1$. Por tanto, incluimos únicamente en la base de factores a aquellos primos menores que el límite B y que verifiquen esta expresión del símbolo de Jacobi. Y calculamos explícitamente los dos valores posibles de $a(modp^k)$ tales que $p^k|Q(a)$, y que llamamos $p_k a$ y $p_k b$. Si $p = 2$ y $k \geq 3$, entonces $x^2 \equiv n(mod\ \ 2^k)$ tiene una solución (de hecho cuatro) si y solo si $n \equiv 1(mod\ \ 8)$ y podemos proceder entonces a su cálculo.

Finalmente, si $p = 2$ y $k = 2$, tomamos $x = 1$ si $n \equiv 1(mod\ \ 4)$ (de otra forma, a no existe); y si $p = 2$ y $k = 1$, tomamos $x = 1$. Tomando un intervalo suficientemente grande (para la criba), calculamos de modo aproximado $ln|Q(a)|$. Almacenamos estos valores en un vector indexado con a. Ahora, para cada primo p de la base de factores y más genéricamente, para cada pequeña potencia de primo (cuando p es pequeño: es buena regla limitar a valores tales que p^k no superen un determinado límite) vamos restando el valor aproximado de lnp a cada elemento del vector que sea congruente con a_{p^k} o con b_{p^k} módulo p^k.

Cuando todos los primos de la base de factores han sido utilizados en la criba, queda claro que un valor $Q(a)$ será factorizado en nuestra base de factores si y solo si lo que queda en la posición de índice a de nuestro vector está próximo a cero (si los logaritmos fueran exactos, únicamente si es idénticamente igual a cero). De hecho, si $Q(a)$ no queda completamente factorizado, entonces el valor restante en la posición de índice a será al menos igual a lnB ; dado que este valor es bastante mayor que uno, queda justificado entonces que no sea necesario tomar valores muy exactos de los logaritmos.

El polinomio $Q(a)$ introducido antes es bastante útil para el proceso que deseamos realizar, pero desgraciadamente no es útil en la práctica dado que los valores de $Q(a)$ crecen rápidamente, más de lo que desearíamos. La idea (que debemos a Jim Davis y a Peter Montgomery[11]) que surge entonces es la de MPQS: criba cuadrática con polinomios múltiples.

El nuevo método consiste en usar una serie de polinomios Q de forma que los valores de a vayan permaneciendo lo más pequeños posibles.

Tomamos polinomios cuadráticos de la forma $Q(x) = Ax^2 + 2Bx + C$, don-

de $A > 0, B2 - AC > 0$ y tales que $n|B^2 - AC$. Estos polinomios ofrecen congruencias semejantes a las de antes:

$$AQ(x) = (Ax + b)^2 - (B^2 - AC) \equiv ((Ax + b)^2 (mod n)$$

Además, queremos que los valores de $Q(x)$ sean tan pequeños como podamos en el intervalo de criba. Si queremos cribar en un intervalo de longitud $2M$, es natural que centremos el intervalo en el mínimo de la función Q ; por ejemplo, la criba en el intervalo

$$I = [-B/A - M, -B/A + M].$$

Y entonces, para cada $x \in I$, tenemos que $Q(-BA) \leq A(x) \leq Q(-BA + M)$. Por lo tanto, para minimizar el valor absoluto de $Q(x)$ averiguamos si $Q(-B/A) \approx -Q(-B/A+M)$, lo que es equivalente a que $A^2M^2 \approx 2(B^2-AC)$ y, por tanto

$$A \approx \frac{\sqrt{2(B^2 - AC)}}{M}$$

y entonces tenemos

$$max_{x \in I}|Q(x)| \approx \frac{B^2 - AC}{A} \approx M\sqrt{(B^2 - AC)/2}$$

Como queremos que sea lo más pequeño posible, pero como también tenemos que $n|B^2 - AC$, escogemos entonces A, B y C tal que $B^2 - AC = n$; y el máximo valor para $Q(x)$ será aproximadamente igual a $M\sqrt{n/2}$. Este es un orden de magnitud similar al caso QS (de hecho algo inferior), pero ahora ha quedado añadida la libertad de introducir un nuevo polinomio tan pronto como nuestros residuos alcancen valores demasiado altos. En resumen: Primero escogemos una longitud de intervalo de criba apropiado, M . Luego, tomamos un valor para A cercano a $\sqrt{2n}/M$tal que A sea primo y $(n/A) = 1$. Luego buscamos un valor para B tal que $B^2 \equiv n(mod A)$, y finalmente buscamos un $C = (B^2 - n)A$. Ahora, como en el caso del QS ordinario, debemos computar para cada potencia de primo p^k en nuestra base de factores los valores $a_{p^k}(Q)$ y $b_{p^k}(Q)$, con los que iniciamos la criba. Son simples raíces módulo p^k de $Q(a) = 0$. Así, como el discriminante de Q ha sido escogido igual que n , éstas raíces son iguales que $(-B + a_{p^k})/A$ y $(-B + b_{p^k})/A$, donde

$$a_{p^k} \ , \ b_{p^k}$$

son las raíces de n módulo p^k que hemos de calcular una vez para todo el proceso.

5.6. La Criba de Campo Numérico

El método más reciente y más poderoso conocido hasta la fecha para factorizar números es el de la **criba de campo numérico (NFS)**. La idea básica es similar a la de QS: mediante un proceso de criba buscamos congruencias módulo n trabajando sobre una base de factores y obteniendo, mediante la técnica de eliminacin gaussiana sobre Z/nZ una congruencia de cuadrados hasta encontrar la deseada factorización de n.
En algunos aspectos, el algoritmo de factorización NFS es similar a los últimos presentados, pues su objetivo, una vez más, se centra en combinar congruencias hasta lograr una expresión de la forma presentada por Fermat. Se emplea para enteros que puedan expresarse de la forma

$$n = r^e - s$$

donde r y $|s|$ son enteros positivos pequeños, $r > 1$ y e grande.
El algoritmo ha venido en llamarse de la criba de campo numérico porque su proceso depende de las propiedades aritméticas de un campo numérico seleccionado con adecuadas propiedades algebraicas, combinado con las técnicas tradicionales de criba, como las recogidas para el caso QS. Es, sin duda, el más complejo algoritmo de factorización conocido.
Los requerimientos básicos de este algoritmos pueden resumirse de la siguiente manera: Dado el entero n que se desea factorizar, se elige un grado d y se expresa n como el valor obtenido, a partir del polinomio irreductible de grado d , sobre un determinado entero m :

$$n = f(m) = m^d + a(d-1).m^{d-1} + a(d-2).m^{d-2} + ... + a1.m + ao$$

donde m y ak son enteros de un orden $n^{1/d}$. Una vía para encontrar un polinomio así es tomar m como la parte entera de la raíz de grado d de n y obtener posteriormente la expansin de n en la base m. Para un tamaño de entero a factorizar de, por ejemplo, 125 dígitos decimales el valor para d recomendado es 5, de forma que los coeficientes y el mismo valor de m tienen

en torno a los 25 dígitos. El proceso de la criba del campo numérico busca entonces (mediante un proceso de criba similar al descrito para QS) tantos pares (a, b) como sea posible tales que tanto

$$a + bm$$

y también

$$b^d f(-a/b) = (-a)^d + a(d-1).(-a)^{d-1}.b + a(d-2).(-a)^{d-2}.b^2 + ... + (-a1)ab^{d-1} + aod^d$$

sean suaves sobre la base de factores dada.

5.7. Curvas Elípticas y Factorización

Sea $\overline{K}$ una clausura algebraica fija de un cuerpo K. Una *curva elíptica* sobre K es una curva proyectiva plana $E \subseteq \mathbb{P}^2(\overline{K})$ no singular definida por una ecuación de la forma

$$y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3. \tag{1}$$

donde $a_1, a_2, a_3, a_4, a_6 \in K$.
Si afinizamos la curva elíptica E haciendo $z = 1$, obtenemos una curva dada por la ecuación afín

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2}$$

Si $car(K) \neq 2$, hacemos el siguiente cambio

$$\begin{cases} x &= x \\ y &= \frac{1}{2}(Y - a_1x - a_3), \end{cases}$$

y conseguimos describir la curva con la ecuación

$$Y^2 = x^3 + \frac{b_2}{4}x^2 + \frac{b_4}{2}x + \frac{b_6}{4} \tag{3}$$

donde

$$b_2 = a_1^2 + 4a_2, \quad b_4 = 2a_4 + a_1a_3, \text{ y } b_6 = a_3^2 + 4a_6.$$

Si además $car(K) \neq 3$. En la ecuación (3), hacemos el cambio

$$\begin{cases} x &= X - \frac{1}{12}b_2 \\ Y &= Y, \end{cases}$$

obteniendo la ecuación

$$Y^2 = X^3 - \frac{c_4}{2^4 \cdot 3}X - \frac{c_6}{2^5 \cdot 3^3} \tag{4}$$

con

$$c_4 = b_2^2 - 24b_4, \quad \text{y} \quad c_6 = b_2^3 + 36b_2b_4 - 216b_6.$$

Y así tenemos que E esta dada por la ecuación

$$y^2 = x^3 + Ax + B \tag{5}$$

donde $A = -\frac{c_4}{2^4 \cdot 3}$ y $B = -\frac{c_6}{2^5 \cdot 3^3}$.

La ecuación dada en (5) es conocida como la **ecuación de Weierstrass** de la curva elíptica E, mientras (2) es llamada *ecuación general de Weierstrass.*

Suma de puntos en una curva elíptica

Las curvas elípticas definen de manera natural una operación llamada suma, con la cual se obtiene un grupo abeliano sobre el conjunto de sus puntos,
y cumple con las propiedades dadas en la siguiente proposición:

Proposición 5.7.1
Sea E una curva elíptica sobre un cuerpo K definida por $y^2 = x^3 + Ax + B$. Sean $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2)$ puntos en E con $P_1 \neq \mathcal{O}$ y $P_2 \neq \mathcal{O}$. Definimos $P_1 + P_2 = P_3 = (x_3, y_3)$ como sigue:

i) Si $x_1 \neq x_2$, entonces

$$x_3 = m^2 - x_1 - x_2, \quad y_3 = m(x_1 - x_3) - y_1, \quad \text{donde } m = \frac{y_2 - y_1}{x_2 - x_1}.$$

ii) Si $x_1 = x_2$ pero $y_1 \neq y_2$, entonces $P_1 + P_2 = \mathcal{O}$.

iii) Si $P_1 = P_2$ y $y_1 \neq 0$, entonces

$$x_3 = m^2 - 2x_1, \quad y_3 = m(x_1 - x_3) - y_1, \quad \text{donde } m = \frac{3x_1^2 + A}{2y_1}.$$

iv) Si $P_1 = P_2$ y $y_1 = 0$, entonces $P_1 + P_2 = \mathcal{O}$.

Además, definimos $P + \mathcal{O} = P$ para todo P en E.

Por lo tanto $E(K)$ es cerrado bajo la ***suma de puntos***.

Teorema 5.7.2
La suma de puntos en una curva elíptica E, satisface las siguientes propiedades:

1. Conmutatividad: $P_1 + P_2 = P_2 + P_1$ para todo $P_1, P_2 \in E$.
2. Existencia del neutro: $P + \mathcal{O} = P$ para todo $P \in E$.
3. Existencia de inverso: Dado $P \in E$, existe $P' \in E$ tal que $P + P' = \mathcal{O}$.
4. Asociatividad: $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ para todo $P_1, P_2, P_3 \in E$.

Esto significa que los puntos de E forman un grupo abeliano aditivo con $\mathcal{O}$ como elemento neutro.

Prueba. La conmutatividad, es obvia pues la recta que une P_1 y P_2 es la misma que la recta que une P_2 y P_1.
La propiedad de existencia del elemento neutro $\mathcal{O}$, se cumple por definición. Ahora, si P está en E y P' es la reflexión de P con respecto al eje x, entonces $P + P' = \mathcal{O}$. La asociatividad es la propiedad menos obvia de la suma de puntos de E, los detalles de la prueba los podemos encontrar en [1].

Observación 1: En la ecuación de Weierstrass, si $P = (x, y)$, entonces $-P = (x, -y)$. Para la ecuación general de Weierstrass (2), si $P = (x, y)$, se prueba que $-P = (x, -a_1x - a_3 - y)$.

Observación 2: Si P es un punto de una curva elíptica E, entonces $[k]P$ denota $P + P + \cdots + P$ (con k sumandos, $k > 0$). Si $k < 0$, entonces $[k]P = (-P) + (-P) + \cdots + (-P)$, con $|k|$ sumandos. Para calcular *la multiplicación puntual o multiplicación escalar* $[k]P$, cuando k es un entero muy grande, es más rápido y eficiente usar el método de las *doblar y sumar* puntos elípticos.Este método nos permite calcular $[k]P$ para k grande, más rápidamente.

5.8. Algoritmo de Factorización de Lenstra

Hendrik Lenstra descubrió, a finales de los años ochenta, un algoritmo basado en la teoría de curvas elípticas para factorizar enteros, usualmente conocido como el Método de Factorización de Curva Elíptica (Elliptic curve factorization method, ECM) o la Factorización de Curva Elíptica de Lenstra. Este es un algoritmo de tiempo de ejecución sub-exponencial para la factorización de enteros; para una factorización de propósito general, ECM es el tercer método más rápido conocido de factorización. El segundo más rápido es la criba cuadrática de múltiples polinomios y el más rápido es la criba general del cuerpo de números.
El ECM es considerado un algoritmo de factorización de propósito especial así como el más adecuado para encontrar factores pequeños.
En la actualidad, es todavía el mejor algoritmo para divisores de no más de los 20 a 25 dígitos decimales (64 a 83 bits respectivamente), así como su tiempo de ejecución está dominado por el tamaño del factor más pequeño p en lugar de por el tamaño del número n a ser factorizado.
El método ECM se usa para eliminar factores pequeños de un entero muy grande con muchos factores; si el entero resultante todavía es compuesto, entonces sólo tiene factores grandes y es factorizado mediante el uso de técnicas de propósito general.
El factor más grande encontrado usando ECM cuenta con 75 dígitos y fue descubierto el 2 de agosto de 2012 por Samuel Wagstaff.

Este algoritmo está basado en otro algoritmo conocido como el algoritmo $\rho - 1$ de Pollard (ver [4]); presentaremos un resumen de este algoritmo el cual se basa en calcular $[k]P$ para distintos enteros k y para un punto $P \in E(F_p)$ donde E es una curva elíptica sobre el cuerpo finitoF_p.

Algoritmo : Curva elíptica
La entrada: $N \in \mathbb{N}$ compuesto, con $mcd(N;6) = 1$, y B para número máximo de iteraciones.
La respuesta: cualquiera de los divisores no triviales de N o fracaso.

PASO 1 Escoger aleatoriamente $(E; P)$ donde E es una curva elíptica $y^2 = x^3 + ax + b$ sobre Z_N y $P(x; y) \in E(Z_N)$.

PASO 2 Hacer $a = 2$.

PASO 3 Mientras $a \leq B$ hacer

- Calcular el punto $P_a \in Z_N$, que se realiza mediante las sumas repetidas: El punto $P_a(x_a; y_a) = P_1(x_1; y_1) + P_{a-1}(x_{a-1}; y_{a-1})(\ \text{mód}\ N)$, utilizando la expresión

$$(x_3; y_3) = (\lambda^2 - x_1 - x_2(mod N); \lambda(x_1 - x_3) - y_1(\ \ \text{mód}\ N))$$

  donde

- Si $c = O$ o si no es posible calcular d_2 entonces, calcular $p = mcd(d_2; N)$, si $1 < p < N$ entonces salida p y terminar el proceso.

- En otro caso incrementar en una unidad el valor de a.

PASO 4 Fracaso, el número de iteraciones no fue suficiente para encontrar un factor de N.

5.9. Un ejemplo

Utilizamos la curva elíptica $y^2 = x^3 + x + 1$ sobre Z_{35} para factorizar 35. Tomemos $B = 11$ y el punto $P = (0, 1)$, notemos que pertenece a la curva. Ahora se procede a calcular los múltiplos enteros de P:

	mód 35
P	$(0; 1)$
$2P$	$(9; 12)$
$3P$	$(2; 16)$
$4P$	$(28; 34)$
$5P$	

Cuando intentamos calcular $5P$, el calcular $P + 4P$ (mód 35) involucra la expresión $\frac{(y_2 - y_1)}{(x_2 - x_1)}$ y el denominador es $x_2 - x_1 = 28 - 0$, el cual no es invertible módulo 35 ya que $mcd(28, 35) = 7$.
Así hemos encontrado un factor no trivial de 35, por lo que termina el proceso.

6. Discusión

En este trabajo realizamos un estudio de diferentes algoritmos matemáticos con el objetivo de lograr la factorización de enteros grandes, que son de trascendencia por su eficiencia y rapidez y que se bazan en el cristosistema RSA que es de vigencia actual en cuanto a seguridad de envio de información por medios inseguros como lo es internet.

En general no existe una forma eficiente de factorizar a un número entero si no se conoce nada acerca de él, ya que algunos métodos son más eficientes en algunos casos que en otros, sin embargo podemos dar una estrategia para poder intentar factorizar un número entero, con lo que esperamos optimizar el proceso de factorizar, aplicando el mejor método conforme a la forma del número. Si ya se conoce una estratégia para factorizar al número entero, el parámetro más importante es el tiempo que tomará para lograr el objetivo:

1. Aplicar el método de ensayar divisores pequeños, hasta una cota C_0.
2. Aplicar el algoritmo de Pollard, esperando encontrar un factor r, tal que $C_0 < r < C_1$ donde C_1 es otra cota.
3. Aplicar el método con curvas elípticas, esperando encontrar un factor r_1 tal que $C_1 < r_1 < C_2$
4. Aplicar un método de propósito general, como la Criba cuadrática o la Criba de campos numéricos.

7. Referenciales

LIBROS

L1 COHEN, HENRI. *A Course in Computational Algebraic Number Theory* New York:Springer Verlag 2nd edition 1993.

L2 RIESEL, HANS. *Prime Numbers and Computer Methods for Factorization* New York: Springer Verlag 2nd edition 1994.

L3 YAN, SONG Y. *Number Theory for Computing* Berlyn: Springer 2nd edition 2002.

L4 BRESSOUD, D.M. *Factorization and primality testing* New York: Springer-Verlag 3ra edition 1989.

L5 SILVERMAN, JOSEPH. *The Arithmetic of Elliptic Curves* Graduate Texts Math 106 New York: Springer Verlag 1986.

L6 KATZ, N.M. and MAZUR, B. *Arithmetic Moduli of Elliptic Curves* Princeton: Princeton University Press 1ra edition 1985.

L7 KNUTH, DONALD. E. *The Art of Computer Programming,* Vol. 2 Addison Wesley 2nd edition 1982.

L8 LANG, S. *Elliptic Curves Diophantine Analysis* New York: Springer-Verlag 2nd edition 1978.

ARTÍCULOS

1. MCKEE, J. *Speeding Fermat's factoring method* Mathematics of Computation 1999 vol. 68 1729-1737.

2. RIESEL, H. *Prime Numbers and Computer Methods for Factorization* Progress in Mathematics 1996 vol. 57 123-154.

3. GOLDWASSER, S. *Twenty Some Years Later (or Cryptography and Complexity: A Match Made in Heaven).*New Directions in Cryptography 1997 vol.38 314-324.

4. LENSTRA, H. W. JR. *Factoring integers with elliptic curves* Annals of Mathematics 1987 vol.126 649-673.

5. BRENT, R.- POLLARD, J. M. *Factorization of the Eighth Fermat Number* Mathematics of Computation 1981 vol.36 627-630.

6. HERNANDEZ ENCINAS, L.- MUÑOZ MASQUÉ, J.- MONTOYA VITINI, F. ÁLVAREZ MARAÑÓN, G. PEINADO DOMÍNGUEZ, A.*Algoritmo de Cifrado con Clave Pública mediante una Función Cuadrática en el Grupo de los Enteros Módulo n*, Actas de la IV Reunión Española de Criptología 1996 101 - 108.

7. POMERANCE, C. *Analysis and comparation of some integer factoring algorithms.* Computational Methods in Number Theory 1 982 vol 21 89-141.

8. RIVEST, R.L.- SHAMIR, A.- ADLEMAN, L. *A method for obtaming digital signatures and public-key cryptosystems* Comm ACM 1978 vol. 21 120-128.

9. SCHOOF, R.J. *Elliptic curves over finite fields and the computation of square roots mod p,* Math Comp 1985 vol 44 483-494. American Mathematical Society.

10. SCHNORR, C.P. - LENSTRA, H.W. JR *A Monte Carlo factoring algorithm with linear storage* Math Comp 1984 vol.43 289-311.

11. MONTGOMERY, P.L. *Speeding the Pollard and elliptic curve methods of factorization* Math of Comp 1987 vol 48 243-264.

12. WILLIAMS, H.C. *Primality testing on a Computer* Ars Combm 1978 vol.5 127-185

13. KIESEL, H. *Prime Numbers and Computer Methods for Factorization* Progi Math 1985 57

14. BRILLHART, JHON.- MONTGOMERY, PETER.- SILVERMAN,ROBERT D. *Tables of Fibonacci and Lucas Factorizations* Mathematics of Computation 1988 vol.181 251-260.

15. WATERHOUSE, W.C. *Abelian varieties over finite fields* Ann Sei Ecole Norm Sup 1969 vol.4 521-560

8. Apéndice

8.1. Algoritmo del Método rho de Pollard

El Algoritmo Método rho de Pollard

Entrada: $N, f, x0$
Resultado: Un factor p de N o mensaje de falla.

1. salir=false;
2. $k = 0$;
3. xi = x0 ;
4. while salir=false do
5. i = 2k ;
6. 1 for $j = i + 1, i + 2, ..., 2i + 1$ do
7. xj = f (x0) (mod N) ;
8. if xi = xj then
9. salir=true;
10. Imprimir: El método falló. Reintentar cambiando f o x0 h;
11. Exit For;
12. g =MCD(xi.xj,N) ;

13. if $1 < g < N$ then

14. salir=true;

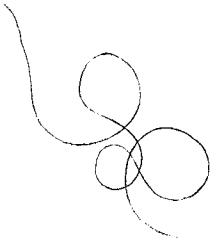
15. Imprimir N = N/g Eg;

16. Exit For;

17. x0 = xj ;

18. xi = xj ;

19. k++;



8.2. Algoritmos de Factorización de Enteros

- El algoritmo de CFRAC.
- El algoritmo de Dixon.
- El Método de Curvas Elípticas.
- El Método de Euler.
- El Método de rho Pollard.
- El Método de p - 1.
- El Método de p + 1.
- El Método de QS.
- El Método de GNFS.
- El Método de SNFS.
- El Método de rational sieve.
- El Método de Fermat.
- El Método de formas caudráticas de Shanks.
- El Método de Trial division.
- El Método de Shor.

ANEXOS

En este Informe Final no se considera ANEXOS.