Algoritmos de factorización de números enteros

Miguel Clavo López

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Directores del trabajo: Álvaro Lozano & Jorge Martín
4 de septiembre de 2023

Summary

The set of natural numbers under standard addition and multiplication have the property of uniqueness factorization into primes. That means, for every $n \in \mathbb{N}$ we can decompose $n = p \cdot k$ with p a prime number and k a natural number. This factorization is easy to do but requires many steps to get it. Studying algorithms that make this factorization faster is important now more than ever.

Most of now-a-days encryption systems depend on the difficulty of factorizing a number in comparison with the quickness to find a prime number, like in the RSA encryption systems. But this is not always the case, there exists special numbers that can be factorized in less steps by the Elliptic Curve Method of Lenstra. This algorithm is based on a previous algorithm, the so-called $p-1$ Pollard method.

Chapter 1 introduces the basics on arithmetic and modular arithmetic needed for the rest of the manuscript. One of the key ingredients needed by Pollard's method is Fermat's Little theorem on modular arithmetic:

Theorem (Fermat's Little theorem (proved in 1.10))**.** *Let $a, p \in \mathbb{Z}$ such that p is a prime number and such that p does not divide a. Then*

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

Chapter 2 describes how the Pollard method works. Suppose we want to factorize a natural number n that is product of two primes p and q. Now the idea is to take a number $a \in \mathbb{N}$ and raise it to a multiple of $p-1$, lets say $(p-1) \cdot g$. By Little Fermat's theorem we obtain

$$a^{(p-1)\cdot g} \equiv 1 \pmod{p}.$$

Therefore $\gcd(a^{(p-1)\cdot g}, n) > 1$, that means it is a factor of n (prime as n is multiple of 2 primes).

The number a can be any number lower than p, in practice, we can take $a = 2$. To get the multiple of $p-1$ we can just compute the factorial of a big enough number L, so that $L!$ is divided by $p-1$. If $e_1^{q_1} \cdots e_k^{q_k}$ is the factorization of $p-1$, then it is enough to choose $L = \max_i e_i \cdot q_i$. Note that if $p-1$ factors into *small* primes, then the method is specially fast.

Until now, we have been looking for factors of n in the group of multiplicative units of $\mathbb{Z}/p\mathbb{Z}$. In Chapter 3 is explained how this algorithm can be extended by changing this group with more general groups.

Here we deal with the group of points on an elliptic curve. We consider the locus of the points $P = (x, y)$ that are solutions of the Weierstrass equation in a field $\mathbb{K}$:

$$y^2 = x^3 + Ax + B \qquad \text{with } \triangle := -16(4A^3 + 27B^2) \neq 0.$$

We add a virtual *point at infinity* O, so that all vertical lines will cut the previous set in 2 point and O. The union of this point and the previous set is called the *Elliptic curve* $E(\mathbb{K})$ given by the Weierstrass equation. Now consider the following procedure for adding two points P and Q: the line that passes through P and Q cuts the curve in a third point $P * Q$. Taking the vertical line from this point, it cuts

the curve in another point that will be the sum $P + Q$. This addition defines the internal operation of the group of points from $E(\mathbb{K})$.

If the curve is not defined over a field $\mathbb{K}$, then the operation is not well defined, because at some point we will have to find the inverse of a coordinate. This is the key for the Lenstra's algorithm.

We need another result for more general groups that plays the same role as the Little Fermat Theorem for the units of $\mathbb{Z}/p\mathbb{Z}$, this is Lagrange's theorem:

Theorem (Lagrange's theorem 3.5)**.** *Given a group G and a subgroup H then $|H|$ divides $|G|$.*

With all of these properties for the points in a curve, we can generalize the Pollard algorithm. This is done in Chapter 4.

Observe that $\mathbb{Z}/k\mathbb{Z}$ is a field if and only if k is prime. With this property, we can factorize a composite number $n = pq$ with the following method:

Take a random elliptic curve $E(\mathbb{Z}/n\mathbb{Z})$ and continue as if n was a prime (we will not be able to continue at some point, that is what we seek). Take a point $P \in E(\mathbb{Z}/n\mathbb{Z})$, and compute $[i!]P$ (adding P to itself $i!$ times).

From a theoretical point of view we can also consider the same curve with coefficients in the (unknown) field $\mathbb{Z}/p\mathbb{Z}$. By the Lagrange theorem we know that if $i!$ divides $\#E(\mathbb{Z}/p\mathbb{Z})$ then $[i!]P = O$ in $E(\mathbb{Z}/p\mathbb{Z})$.

This implies that it is not possible to compute $[(i-1)!]P + [i]P$ in $E(\mathbb{Z}/n\mathbb{Z})$, since, as we have mentioned above, we will have to find the inverse of a non-invertible number of $\mathbb{Z}/n\mathbb{Z}$. That number is a divisor of n.

As in the Pollard algorithm, this method is specially fast finding prime factors p of n such that $p - 1$ has small prime factors. An improvement over Pollard's method is given by Hasse's theorem.

Theorem. *Let $E(\mathbb{K})$ then*

$$|\#E(\mathbb{K}) - (p+1)| \leq 2\sqrt{p}.$$

The improvement comes from the fact that set of points where we search for a divisor of n can be smaller than in Pollard method. In fact we can build an elliptic curve with the number of points that we want inside the Hasse's theorem range.

Índice general

Introducción

En la actualidad, la mayoría de los criptosistemas en uso dependen de la dificultad para factorizar números en comparación con la rapidez para encontrar números primos. Un ejemplo son los sistemas RSA que para romperlo es necesario factorizar un número que es múltiplo de dos primos. En general, no se conoce un algoritmo determinista para factorizar números en tiempo polinomial. Sin embargo, sí que existen algoritmos que son capaces de factorizar enteros que cumplen unas propiedades específicas en un tiempo subexponencial.

El algoritmo de Lenstra para curvas elípticas es uno de los más importantes. Su rapidez factorizando números con factores primos p tales que $p-1$ se factorice en números primos *pequeños*, ha hecho que no se puedan utilizar estos números en las claves de cifrado RSA.

Este algoritmo se basa en otro anterior, el $p-1$ de Pollard. En este texto vamos a explicar cómo funciona este algoritmo y como pasar de este al de Lenstra. Para ello primero se explican las bases teóricas, que son propiedades de la aritmética entera, para comprobar el funcionamiento de este algoritmo. En concreto el fin va a ser demostrar el pequeño teorema de Fermat 1.10 y a través de éste construir el algoritmo de Pollard. Después se describe como implementarlo.

El paso del algoritmo de Pollard al de Lenstra requiere cambiar la aritmética entera por operaciones en grupos. Además esto nos servirá para comprobar la ventaja del segundo respecto al primero. En particular, usaremos grupos de curvas elípticas. De esta manera describiremos el algoritmo de Lenstra, y de nuevo se dan algunas notas sobre como se puede implementar.

Por último se discute cómo el algoritmo de Lenstra es especialmente rápido para números n concretos y cómo estos números son los mismo para los que el algoritmo de Pollard es especialmente eficiente.

En este trabajo está dividido en 5 capítulos.

En el capítulo 1 se desarrollan los conceptos básicos de la aritmética entera para entender de qué trata la factorización de números y qué se necesita para llevarla a cabo. Toda esta base nos sirve para enunciar el pequeño teorema de Fermat (teorema 1.10), que es el teorema fundamental para entender como funcionan los algoritmos que se explican en el trabajo.

En el capítulo 2 se describe el algoritmo $p-1$ de Pollard, basado directamente en el teorema de Fermat (Algoritmo 3). También se discute cómo llevar a cabo su implementación, a través del estudio del tipo de números para los que el algoritmo es especialmente rápido.

En el capítulo 3 se introducen las nociones necesarias de teoría de grupos con el objetivo, primero de comprobar que la eficiencia del algoritmo $p-1$ radica en que busca factores únicamente en el grupo de unidades de $\mathbb{Z}/p\mathbb{Z}$. Segundo, para entender el teorema de Lagrange (teorema 3.5), que es la generalización del pequeño teorema de Fermat para cualquier tipo de grupo, y será la clave para entender el algoritmo de Lenstra. Después se define cuerpo y el grupo de puntos de una curva elíptica sobre éste.

En el capítulo 4 se muestra el algoritmo de Lenstra de factorización por curvas elípticas (algoritmo 5). Este algoritmo es en esencia el mismo algoritmo que el de Pollard pero generalizando el pequeño teorema

de Fermat al teorema de Lagrange aplicado al grupo de una curva elíptica. La ventaja de este algoritmo se verá que está en que el orden del grupo de una curva elíptica puede ser considerablemente menor que el grupo de unidades utilizado en el algoritmo de Pollard.

En el capítulo 5 se presentan algunas conclusiones y lineas de investigación futuras.

Capítulo 1

Aritmética entera

1.1. Definiciones y propiedades de los números naturales y enteros

Se denota como $\mathbb{N}$ al conjunto de números naturales: $0, 1, 2, 3...$ y como $\mathbb{Z}$ al conjunto de números enteros $... -2, -1, 0, 1, 2....$ Ambos conjuntos con las operaciones de suma y multiplicación habituales.

Se dice que n tiene como factores a los números a y b si $n = a \cdot b$.

Se dice que a divide a n (representado por $a \mid n$) si existe $k \in \mathbb{N}$ tal que $a \cdot k = n$.

Los números naturales se pueden clasificar según su divisibilidad de esta forma:

Definición 1.1. Sea $p \in \mathbb{N}$ se dice que p es un número primo si $p \geq 2$ y no se puede factorizar en dos números menores que p. Si un número $n \geq 2$ no es primo se dice que es compuesto.

También podemos definir una relación entre dos o más números enteros:

Definición 1.2. Dados $a, b \in \mathbb{N}$ el *máximo común divisor de a y b* (denotado por $\text{mcd}(a, b)$) es mayor número natural que es divisor de a y b.

Sean $a, b \in \mathbb{N}$ se dice que son *coprimos* si a y b no tienen divisores comunes mayores que 1, es decir si $\text{mcd}(a, b) = 1$.

Vamos a recordar algunos resultados que pueden parecer elementales pero son fundamentales para la construcción de los métodos tratados en este trabajo. Como primera propiedad de los números naturales tenemos la existencia del cociente y resto.

Teorema 1.3 (de la división)**.** *Sean $a > 0$ y b dos números naturales, existen $q > 0$ y $0 \leq r < a$ naturales tales que $b = aq + r$. A los números q y r se les llama* cociente *y* resto *respectivamente.*

Demostración. Sea

$$S = \{b - ax \mid x \in \mathbb{N}, b - ax \geq 0\}.$$

Por el principio de buena ordenación de $\mathbb{N}$, S tiene un elemento mínimo r. Luego existe $q \in \mathbb{N}$ tal que $r = b - aq$.

Como $S \subset \mathbb{N}$ se tiene $r \in \mathbb{N}$.

Para ver que $r < a$ se observa que si no lo fuera $0 \leq r - a$, y

$$0 \leq r - a = b - qa - a = b - (q + 1)a.$$

Así $r - a \in S$ y contradice que r sea el menor valor en S. Luego $r < a$ y $b = aq + r$, con $0 \leq r < a$. □

Además ambos, cociente y resto, son únicos:

Teorema 1.4. *Sean $a > 0$ y $b \in \mathbb{N}$ tal que $b = aq + r$ como en el teorema anterior. Entonces cociente y resto son únicos.*

Demostración. Suponer que hay dos expresiones $b = aq + r$ y $b = as + t$, con q, r, s, t como en el teorema anterior.

Se puede asumir que $r \geq t$. Entonces

$$0 = b - b = aq + r - as + t = r - t - a(s - q)$$

y

$$a(s - q) = r - t.$$

Pero $0 \leq r - t \leq r < a$. Dividiendo entre a se tiene $0 \leq s - q = (r - t)/a < 1$. Dado que $s - q$ es natural, $s - q = 0$. Se tiene $r = t$ y $q = s$. □

Además se puede ver que todo número natural es divisible por al menos un primo.

Proposición 1.5. *Todo número natural mayor que 2 es divisible por un primo.*

Demostración. Sea $P(n)$ la sentencia "n es divisible por un número primo" comprobemos por inducción que $P(n)$ es cierta para todo $n \in \mathbb{N}$.

$P(2)$ es trivial, ya que 2 es primo.

Si n es primo entonces $P(n)$ es cierta ya que es divisible por si mismo. Por otro lado, si n es compuesto entonces $n = a \cdot b$ con $2 \leq a, b < n$. Por la hipótesis de inducción sabemos que $P(a)$ es cierta, luego el primo que divide a a también divide a n. □

Con los anteriores resultados podemos concluir que todo número tiene una factorización única en números primos:

Corolario 1.6. *Para todo natural $n > 1$ existen $\{p_1, p_2, \ldots, p_k\}$ primos tales que $n = p_1 \cdot p_2 \cdots p_k$ y están únicamente determinados para cada n.*

Demostración. Demostremos primero la existencia y luego la unicidad:

Si n es primo entonces $n = n$ es una factorización en números primos.

Si n es compuesto, existen $a, b \in \mathbb{N}$ con $a, b < n$ tales que $n = a \cdot b$. Si a o b son compuestos podemos aplicar de nuevo el proceso obteniendo divisores cada vez más pequeños hasta llegar a obtener una factorización en primos.

Con este método en un número de pasos finitos obtenemos una factorización en primos.

La unicidad se prueba por inducción. Suponer que para todo número $m < n$, m tiene factorización en primos única.

Si n es primo, entonces ya conocemos su factorización, $n = n$ y no puede haber otra factorización ya que $n = p_1 p_2$ contradice que n sea primo.

Si n no es primo entonces, supongamos que tiene dos factorizaciones distintas $n = p_1 \cdots p_s$ y $n = q_1 \cdots q_r$.

Observar que

$$p_i \neq q_j \qquad \text{para } 1 \leq i \leq s \text{ y } 1 \leq j \leq r.$$

ya que por la hipótesis de inducción no existen dos factorizaciones distintas de n/p_1 al ser menor que n.

Supongamos pues, sin perdida de generalidad, que $p_1 < q_j$ para todo $1 \leq j \leq r$. Por el teorema de la división existen d y r tales que: $q_1 = dp_1 + r$. Entonces

$$\frac{q_1}{p_1} = d + \frac{r}{p_1},$$

y $0 < r < p_1 < q_1$ ya que si $r = 0$ entonces q_1 no sería primo. Al multiplicar ambos lados por $\frac{n}{q_1}$ se obtiene:

$$p_2 \cdots p_s = \left(d + \frac{r}{p_1}\right) q_2 \cdots q_r = d \cdot q_2 \cdots q_r + \frac{r}{p_1} \cdot q_2 \cdots q_r.$$

Observemos que $k := \frac{r}{p_1} \cdot q_2 \cdots q_r$ debe ser un número natural puesto que es la diferencia (positiva) de dos naturales y $0 < \frac{q_1}{p_1}$. Entonces

$$p_1 \cdot k = r \cdot q_2 \cdots q_r$$

es menor que n. Tenemos pues dos factorizaciones de un número menor que n lo cual contradice la hipótesis de inducción. □

1.1.1. Algoritmo de Euclides para calcular el máximo común divisor

Como resultado del corolario 1.6 podemos enunciar un algoritmo para hallar el $\text{mcd}(a, b)$ de dos números $a, b \in \mathbb{N}$.

Algoritmo 1 Algoritmo de Euclídeas para hallar $\text{mcd}(a, b)$

Entrada $a, b \in \mathbb{N}$.

1: Entrada: $a, b \in \mathbb{N}$.
2: **while** $b \neq 0$ **do**
3: Calcular $q, r \in \mathbb{N}$ tales que $a = q \cdot b + r$. ▷ Ver teorema 1.3
4: $a \leftarrow b$
$b \leftarrow r$

Salida a.

Proposición 1.7. *El algoritmo 1 devuelve en un número finito de pasos* $\text{mcd}(a, b)$.

Demostración. Primero observar que $\text{mcd}(a, b) = \text{mcd}(q \cdot b + r, b)$, como b divide a $q \cdot b$, el mayor número que divide a $q \cdot b + r$ y a b es el mismo que a r y b.

De esta manera se tiene $\text{mcd}(a, b) = \text{mcd}(b, r)$ y podemos repetir este proceso.

Llamando r_i al resto de la iteración número i.

Por cada iteración se tiene:

$$\begin{aligned} \text{mcd}(a, b) &= \text{mcd}(b, r) \\ \text{mcd}(b, r) &= \text{mcd}(r, r_1) \\ &\vdots \\ \text{mcd}(r_{i-1}, r_i) &= \text{mcd}(r_i, 0) = r \end{aligned}$$

Comprobando finalmente que si $a = b \cdot q$ entonces $\text{mcd}(a, b) = a$, se tiene que el algoritmo devuelve $\text{mcd}(a, b)$. □

Tenemos pues un algoritmo para hallar el máximo común divisor de dos números enteros. A través de él podemos obtener una igualdad que será útil para cálculos posteriores.

Tomando $d = \text{mcd}(a, b)$ se tiene la igualdad

$$d = r_{i-2} - r_{i-1} \cdot q_{i-1}.$$

A su vez

$$r_{i-1} = r_{i-3} - r_{i-2} \cdot q_{i-2},$$

para cada r_k hasta llegar a

$$r_1 = a - b \cdot q_1.$$

Así sustituyendo en la primera igualdad se obtiene

$$d = a \cdot x + b \cdot y$$

Es decir, podemos expresar

$$\text{mcd}(a, b) = a \cdot x + b \cdot y$$

para algunos $x, y \in \mathbb{Z}$. A esta identidad se le llama *igualdad de Bézout*.

1.1.2. Algoritmo prueba y error

Gracias al corolario 1.6 también podemos enunciar un primer algoritmo, que es el que se utiliza para factorizar a mano.

De ahora en adelante N denotará un número compuesto que queremos factorizar en primos. Para factorizarlo, podemos probar uno a uno los primos menores a $\sqrt{N}$ para ver cual divide a N, una vez encontrado ese primer primo p_1 realizamos el mismo proceso con N/p_1.

Algoritmo 2 Prueba-error para la factorización de N

Entrada $N \in \mathbb{Z}$.

```
1: n ← N
2: lista ← ∅
3: while n > 1 do
4:     for p ∈ 2, 3, 5, 7 ··· do
5:         if p | n then
6:             p es factor primo de N.
7:             lista ← lista ∪{p}
8:             n ← n/p
```

Salida lista ▷ listado de factores primos de N

Este método presenta algunos inconvenientes. Primeramente hay que tener una lista de números primos, la cual podemos tener guardada o podemos generarla con la "criba de Eratostenes". Pero el problema principal es que hay que probar con todos los primos menores al factor p de N para poder obtener p.

Como vemos, el algoritmo recorre una sucesión de números primos hasta dar con uno que divide a N. Si conociéramos alguna propiedad sobre N podríamos sustituir la sucesión de números primos por otra que tenga menos elementos previos al primer factor primo.

1.2. Congruencias módulo *p*

A través de la noción de divisibilidad aparece el concepto de congruencia.

Definición 1.8. Dos naturales a y b se dicen *congruentes módulo n* si n divide a $a - b$. Se denota a este hecho como:

$$a \equiv b \pmod{n}.$$

Equivalentemente, si realizamos las divisiones $a = qn + r$ y $b = sn + t$ tenemos:

$$a \equiv b \pmod{n} \Leftrightarrow t = r.$$

Podemos definir operaciones módulo n de la siguiente forma:

$$(a \pmod n) + (b \pmod n) = a + b \pmod n,$$
$$(a \pmod n) \cdot (b \pmod n) = a \cdot b \pmod n.$$

Es decir, se heredan de los números enteros y las propiedades de estos.

Observar que para un natural $n > 1$, todo $m \in \mathbb{N}$ es congruente módulo n con algún número entre 0 y $n - 1$.

Esto hace que pueda verse la definición anterior defina una relación de equivalencia:

- Reflexividad: $a \equiv a \pmod n$.
- Simetría: $a \equiv b \pmod n \implies b \equiv a \pmod n$.
- Transitividad: $a \equiv b \pmod n$, $b \equiv c \pmod n \implies a \equiv c \pmod n$.

Por ser relación de equivalencia define una partición de $\mathbb{Z}$.

Denotaremos el conjunto de todas las clases de equivalencia módulo n como $\mathbb{Z}/n\mathbb{Z}$. Así tomaremos como representantes de las clases de equivalencia $\pmod n$ los números menores a n.

Entonces trabajaremos con $\mathbb{Z}/n\mathbb{Z} = \{0, 1, 2, \ldots, n-1\}$ con las operaciones ya descritas

Además observar una primera propiedad:

Proposición 1.9. *Sean $a, b \in \mathbb{N}$. Se tiene*

$$\operatorname{mcd}(a, b) = 1 \Leftrightarrow \operatorname{mcd}(a \pmod b, \quad b) = 1.$$

Demostración. Por el teorema de la división $a = kb + r$ y $a \pmod b = r$ con $r < b$.

Sea $b = p_1 \cdot p_2 \cdots p_m$ la descomposición en primos de b, dividiendo $a = qb + r$ entre cada p_i para cada i:

$$\frac{a}{p_i} = \frac{kb}{p_i} + \frac{r}{p_i} \text{ con } r < b.$$

Se tiene que

$$\frac{r}{p_i} \in \mathbb{N} \Leftrightarrow p_i \mid \operatorname{mcd}(a, b).$$

Como esto se puede realizar para cada p_i con $i \in \{1, 2, \cdots, m\}$ tenemos que ningún divisor primo de b divide a a si y solo si ningún divisor primo de b divide a r.

Como r es el representante de a en $\mathbb{Z}/b\mathbb{Z}$, tenemos la equivalencia. □

De la manera habitual podemos definir la exponencial:

$$q^e \pmod n = \overbrace{q \cdot q \cdots q}^{e \text{ veces}} \pmod n.$$

Esta operación tiene una propiedad especial en $\mathbb{Z}/p\mathbb{Z}$ con p primo que hace las operaciones más rápidas y que da una característica particular de $\mathbb{Z}/p\mathbb{Z}$.

Teorema 1.10 (Pequeño Teorema de Fermat)**.** *Sean a, $p \in \mathbb{Z}$ con p primo y tal que $p \nmid a$ entonces:*

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

Demostración. En toda la demostración los valores representaran la clase de equivalencia en $\mathbb{Z}/p\mathbb{Z}$.

Tomamos los conjuntos de clases de equivalencia:

$$A = \{a, 2a, \ldots, (p-1)a\},$$
$$B = \{1, 2, \ldots, p-1\} = \mathbb{Z}/p\mathbb{Z} - \{0\}.$$

Observar que por lo visto antes B contiene representantes de todas las clases de equivalencia módulo p excepto 0. Además $p \nmid a$ y p no divide a ninguno de los valores $1, 2, 3, \ldots, p-1$, por lo que $0 \notin A$. Así que $A \subseteq B$.

Ahora, suponer que existen $r, s \in \mathbb{N}$ con $1 \leq r < p$ y $1 \leq s < p$ tales que $ra = sa$.

Tenemos entonces

$$0 = ra - sa \iff 0 = ra - sa = (r-s)a \iff r - s = 0,$$

ya que por hipótesis $p \nmid a$ y $a \not\equiv 0$. Pero, como $r < p$ y $s < p$, y $r - s \equiv 0 \pmod{p}$ se tiene $r = s$.

Esto implica que las clases de A (que son de la forma ka con $k \in \mathbb{N}$) son diferentes entre si. Es decir la cardinalidad de A es $p-1$, y como $A \subseteq B$ y $\#A = \#B$ tenemos $A = B$. Por todo ello:

$$a \cdot 2a \cdot 3a \cdots (p-1)a = 1 \cdot 2 \cdot 3 \cdots (p-1),$$
$$a^{p-1}(p-1)! = (p-1)!,$$

con $(p-1) \not\equiv 0 \pmod{p}$. Entonces dividiendo entre $(p-1)!$ resulta

$$a^{p-1} = 1. \qquad \square$$

Con este resultado sabemos que $p \mid a^{p-1} - 1$ para cualquieras $a, p \in \mathbb{N}$ con $1 < a < p$ con p primo. Pero también que para n compuesto tal que $n = pq$ se tiene

$$\operatorname{mcd}(n, a^{p-1} - 1) > 1.$$

Es decir, n y $a^{p-1} - 1$ tienen un factor primo común. Esto será la clave para el siguiente capítulo.

Capítulo 2

Algoritmo $p-1$ de Pollard

Acabamos de ver que para cualquier n compuesto tal que $n = p \cdot m$ con p primo, tomando $a \in \mathbb{N}$ arbitrario se tiene $\text{mcd}(n, a^{p-1} - 1) > 1$. Esto puede ser útil para factorizar un número n con factores primos desconocidos. Si tomamos un a y lo elevamos a distintas potencias llegará un momento en el que obtengamos $\text{mcd}(n, a^i - 1) > 1$, para algún i tal que $i \mid (p-1)$.

Además por lo visto anteriormente sabemos que:

$$\text{mcd}(n, a^{p-1} - 1) \neq 1 \Leftrightarrow \text{mcd}(n, \quad a^{p-1} - 1 \pmod{n}) \neq 1.$$

Luego podemos realizar las operaciones de elevar a i en $\mathbb{Z}/n\mathbb{Z}$ directamente esperando que podamos agilizar el proceso por el Pequeño Teorema de Fermat.

El algoritmo de Pollard es el siguiente:

Sea un número N compuesto, el algoritmo $p-1$ de Pollard opera como sigue:

Algoritmo 3 $p-1$ de Pollard para factorización de N

Entrada $N \in \mathbb{Z}$ compuesto.

1: Tomar un valor $1 < a < N$ y $A \leftarrow a$.
2: **if** $\text{mcd}(a, N) \neq 1$ **then** devolver $\text{mcd}(a, N)$.
3: **for** $i \in \{1, 2, ...L\}$ **do**
4: $\quad A \leftarrow A^i \pmod{N}$
5: $\quad F \leftarrow \text{mcd}(A - 1, N)$
6: $\quad$ **if** $1 < F < N$ **then** break. F es un factor no trivial de N.
7: $\quad$ **else if** $F = N$ **then** Volver al paso 1 y tomar un nuevo valor de a.

Salida F. ▷ Devuelve un factor de N

Veamos cómo y por qué funciona:

Teorema 2.1. *Sea $N \in \mathbb{Z}$ con un factor primo p tal que*

$$p - 1 = \prod_{j=1}^{t} e_j^{q_j}.$$

Entonces el bucle (pasos (2) a (6)) del algoritmo 3 con:

$$L = \max_{1 \leq j \leq t} e_j q_j,$$

termina (para casi todos los valores de a) devolviendo un factor no trivial F de N.

Demostración. Notar en primer lugar que en la i-ésima iteración del bucle, $A = a^{i!}$.

Por la definición de L si $e_j q_j \mid L$ entonces $e_j s \leq L$ para cada $1 \leq s \leq q_j$. Por tanto

$$e_j^{q_j} \mid L! \text{ para cada } 1 \leq j \leq t, \text{ y } p-1 \mid L!.$$

Por el teorema 1.10 tenemos que $a^{L!} = a^{(p-1)K} \equiv 1^K = 1 \pmod{p}$. Es decir, sabemos que $a^{L!}$ es múltiplo de p siendo este un factor primo de N diferente de 1, así que, como tarde, obtendremos un divisor F de N en la iteración L.

Puede darse el caso de $N \mid a^{i!} - 1$ y obtendríamos $F = N$, pero esto sólo ocurre con 1 de cada $\frac{N}{p}$ múltiplos de p, así que podemos afirmar que es un suceso poco común. □

El caso en el que el bucle (pasos (2) a (6)) no halle un divisor no trivial y devuelva N se soluciona cambiando la base a, de esta manera podemos asegurar que el algoritmo siempre va a hallar un divisor no trivial.

Corolario 2.2. *El algoritmo 3 termina en un tiempo finito.*

Demostración. Hemos visto por el teorema anterior que el único caso en el que los pasos (2) a (6) no terminan con un factor no trivial de N es cuando $a^{i!} \equiv 1 \pmod{N}$, y en ese caso, cambiaremos el valor a en el paso (1). Como el algoritmo comprueba si $a^1 - 1$ es divisor de N, para $a = p + 1$ tendremos un divisor de N y el algoritmo lo devolverá en un tiempo finito. □

Para ver un ejemplo podemos considerar el número $527 = 31 \cdot 17$. El algoritmo de Pollard encuentra el factor primo 17 en la iteración número 4, es decir, tomando la base $A = 2$ tenemos $\text{mcd}(2^{4!} - 1, 527) = 17$.

2.1. Implementación del algoritmo $p-1$ de Pollard

Si observamos en el algoritmo 3, se ha de elegir una cota L para el número de iteraciones del bucle (pasos (2) a (6)). Para la elección de esta cota lo más eficaz sería elegirla como en el teorema 2.1, pero esta cota se define de manera recursiva, ya que requiere de conocer los factores primos del número N, que es justo lo que queremos saber con el algoritmo.

Vamos a dar una definición para entender en qué condiciones el algoritmo 3 es eficaz y cómo elegir una cota para estos casos.

Definición 2.3. Sea B un entero positivo (habitualmente primo) decimos que un entero n es *B-liso* si y solo si todos sus factores primos son menores o iguales a B.

Por ejemplo, $153 = 3^2 \cdot 17$ es 17-liso (y también es n-liso para todo $n \geq 17$).

Entonces para un número $N \in \mathbb{N}$ tal que tenga un factor primo p de manera que

$$p - 1 = \prod_{j=1}^{t} e_j^{q_j} \qquad \text{con } e_j \leq B \ \forall j$$

como en el teorema 2.1, se tiene que $p-1$ es B-liso y el algoritmo 3 devolverá un factor de N.

De esta manera, podemos emplear una cota B en vez de L y podremos factorizar un número N siempre y cuando sus factores primos menos 1 sean B-lisos.

Se puede utilizar esto para buscar una cota B en el algoritmo 3 que sepamos que está directamente relacionada con tiempo en el que esperamos encontrar una cota como se explica con detalle en [1] y páginas 149 a 152 de [2].

Una implementación del algoritmo en C++ para factorizar números que son múltiplos de dos primos pequeños se da en el apendice C.

Capítulo 3

Grupos y curvas elípticas

3.1. Comparación entre conjuntos de búsqueda

Sabemos cual es el conjunto de potenciales divisores de N para el algoritmo prueba-error. Son los números primos menores que $\sqrt{N}$, o, en el caso de no tener una lista de esos primos, son los números naturales menores que $\sqrt{N}$. Veamos sobre qué estructura trabaja el algoritmo $p-1$ Pollard.

3.1.1. Grupos

Para poder entender el conjunto de números del que toma posibles factores de N el algoritmo 3 tenemos primero que definir la estructura de grupo:

Definición 3.1. Sea G un conjunto no vacío con una operación binaria interna (por defecto la llamaremos multiplicación) decimos que G es un *grupo* si se cumple:

- Asociatividad: $a \cdot (b \cdot c) = (a \cdot b) \cdot c \quad \forall a, b, c \in G$.
- Elemento neutro: $\exists e \in G$ tal que $e \cdot a = a \quad \forall a \in G$.
- Elemento inverso: Para todo $a \in G$ se tiene $\exists\, a^{-1} \in G$ tal que $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Además, si la operación cumple la propiedad conmutativa ($a \cdot b = b \cdot a$) diremos que es un *grupo abeliano*.

Denotamos como $|G|$ el número de elementos del grupo G y lo llamaremos *orden del grupo G*.

Vamos a ver algunos ejemplos de la definición anterior:

Ejemplos 3.2.

- Los conjunto $\mathbb{Z}$ y $\mathbb{Z}/n\mathbb{Z}$ con la suma son grupos abelianos ya que la suma es asociativa y conmutativa, el elemento neutro es 0 y el inverso de un elemento a es $-a$ en $\mathbb{Z}$ y $-a = n - a$ en $\mathbb{Z}/n\mathbb{Z}$.
- Los conjuntos $\mathbb{Z}$ y $\mathbb{Z}/n\mathbb{Z}$ con la multiplicación no son grupos, ya que el 0 no tiene elemento inverso.

Sin embargo, podemos tomar un subconjunto que sí es grupo. Denotamos $U(\mathbb{Z}/n\mathbb{Z})$ al conjunto formado por elementos con inverso de $\mathbb{Z}/n\mathbb{Z}$. Entonces se tiene que $U(\mathbb{Z}/n\mathbb{Z})$ con la multiplicación es un grupo:

- La operación $\cdot$ es binaria e interna, ya que para $a, b \in U(\mathbb{Z}/n\mathbb{Z})$ se tiene $(a \cdot b) \cdot (b^{-1} \cdot a^{-1}) = 1$ luego $a \cdot b \in U(\mathbb{Z}/n\mathbb{Z})$.

- La multiplicación es asociativa por ser la misma que en $\mathbb{Z}/n\mathbb{Z}$.
- Cada elemento tiene inverso por construcción del subconjunto.
- El elemento neutro de $U(\mathbb{Z}/n\mathbb{Z})$ es 1.

Llamaremos a $U(\mathbb{Z}/n\mathbb{Z})$ grupo de unidades de $\mathbb{Z}/n\mathbb{Z}$ y a sus elementos unidades del conjunto. Esta misma construcción se puede hacer con $\mathbb{Z}$. En este caso $U(\mathbb{Z}) = \{1, -1\}$.

El caso de $U(\mathbb{Z}/n\mathbb{Z})$ tiene más interés para nuestros propósitos, pero requiere una descripción más detallada:

Teorema 3.3. *El conjunto de unidades de $\mathbb{Z}/n\mathbb{Z}$ es el conjunto de números coprimos con n.*

Demostración. Sea $a \in \mathbb{Z}/n\mathbb{Z}$. Tomamos el producto de a con todos los elementos del conjunto:

$$A = \{0, 1 \cdot a, \ldots, (n-1) \cdot a\}.$$

Si a no es coprimo con n, tenemos que $\operatorname{mcd} a, n = q$ y $q \cdot k = n$ para algún $k \in \mathbb{Z}/n\mathbb{Z}$ con $k < n$. Entonces $n \mid a \cdot k$ y por lo tanto $a \cdot k \equiv 0 \pmod{n}$.

Además, como

$$a \cdot (k + i) = a \cdot k + a \cdot i \equiv 0 + a \cdot i \pmod{n}$$

tenemos que el conjunto A tiene elementos equivalentes a $\{0 \cdot a, 1 \cdot a, \ldots, (k-1) \cdot a\}$, con exactamente k elementos.

Supongamos ahora que existe $i \in \mathbb{Z}/n\mathbb{Z}$ tal que $a \cdot i \equiv 1 \pmod{n}$ y definimos el conjunto $B = \{i \cdot a, i^2 \cdot a, \ldots, i^{(n-1)} \cdot a\}$ y sus elementos son equivalentes en $\mathbb{Z}/n\mathbb{Z}$ a:

$$\begin{aligned} i \cdot a &\equiv 1 \pmod{n}, \\ 2i \cdot a \equiv i \cdot a + i \cdot a &\equiv 1 + 1 \pmod{n}, \\ &\vdots \\ (n-1)i \cdot a &\equiv (n-1) \pmod{n} \end{aligned}$$

luego se tienen $n-1$ elementos diferentes dos a dos de $\mathbb{Z}/n\mathbb{Z}$, lo que contradice que tenemos k elementos diferentes. No existe entonces un elemento i tal que $a \cdot i \equiv 1$.

En sentido contrario la implicación: si existe $i \in \mathbb{Z}/n\mathbb{Z}$ tal que $a \cdot i \equiv 1$ tenemos $n-1$ elementos distintos en el conjunto A y que exista un $k < n$ tal que $ak = n$ contradice que existan $n-1$ elementos distintos. □

Nota 3.4. Se puede obtener un inverso para un elemento $a \in \mathbb{Z}/n\mathbb{Z}$ con a coprimo con n. Para ello tomamos la igualdad de Bézout definida en el apartado 1.1.1. Observar que

$$\operatorname{mcd}(a, b) = 1 = c_1 \cdot n + c_2 \cdot a,$$

tomando módulos

$$1 \pmod{n} = 0 + c_2 \cdot a \pmod{n}.$$

Es decir, c_2, dado por la igualdad de Bézout, es inverso de a en $\mathbb{Z}/n\mathbb{Z}$.

Además podemos ver que para p primo,

$$U(\mathbb{Z}/p\mathbb{Z}) = \{1, 2, \ldots, p-1\}.$$

Es decir, son los elementos del conjunto excepto el 0.

Con estos conceptos ya podemos estudiar el comportamiento de $p-1$ de Pollard.

3.1.2. Algoritmo $p-1$ de Pollard

Para ver qué valores toma A en el algoritmo de Pollard vamos a ver cómo se comportan las potencias de A. Vamos a utilizar la misma notación que en el enunciado del algoritmo 3.

Supongamos que hay algún valor de A que se repite antes de $A^{L!}$, es decir, existen $i, t \in \mathbb{N}$ con $0 < i, t < L$ tales que $A^{i\cdot t} = A^i \pmod N$ se tiene:

$$A^{i\cdot t} \equiv A^i \pmod N \Leftrightarrow A^i \cdot (A^{i(t-1)} - 1) \equiv 0 \pmod N.$$

Entonces sabemos que o bien A^i o bien $(A^{i(t-1)} - 1)$ es 0. Si $A^i \equiv 0$ el algoritmo habría parado ya que la base a sería divisor de N. Así que vamos a suponer que $(A^{i(t-1)} - 1) \equiv 0$ o lo que es lo mismo $A^{i(t-1)} \equiv 1$ y el algoritmo igualmente para en ese momento.

Vamos a suponer el caso en el que el valor de A no se repite.

Sea $N \in \mathbb{N}$ el número que queremos factorizar en primos, y sea p un factor primo de N sabemos que: Si $p \mid a$ entonces $\text{mcd}(a, N) \neq 1$.

Partiendo de la hipótesis de que A no se repite antes de $A^{L!}$ tenemos que A toma valores en $U(\mathbb{Z}/p\mathbb{Z})$ hasta que $p \mid A$. Esto ocurre como tarde cuando el exponente de A^i, es divisible entre $p-1$. En ese momento por el pequeño teorema de Fermat se tiene $p \mid (A^{i!} - 1)$ y por lo tanto $\text{mcd}((A^{i!} - 1), N) \neq 1$.

Hemos visto que el algoritmo $p-1$ de Pollard toma valores de A en $U(\mathbb{Z}/p\mathbb{Z})$ sin repetirse, luego a lo sumo tomará $p-1$ valores.

Se puede pensar que el proceso aceleraría cambiando el conjunto $U(\mathbb{Z}/p\mathbb{Z})$ por otro. Eso es lo que hace el algoritmo de Lenstra.

3.1.3. Teorema de Lagrange

Para entenderlo primero tenemos que ver el Teorema de Lagrange, que generaliza el Pequeño teorema de Fermat 1.10 para cualquier grupo.

Teorema 3.5 (de Lagrange)**.** *Sea G un grupo tal que $|G| < \infty$ y $H \leq G$ un subgrupo de G, es decir, un subconjunto de G que es grupo.*

Entonces

$$|H| \,\Big|\, |G|$$

Demostración. Por ser G finito y $H \subset G$ podemos tomar $H = \{h_1, \cdots, h_k\}$.

Tomando $g, g' \in G$ y multiplicando a izquierda por cada elemento de H:

$$gH := \{gh_1, gh_2, \cdots, gh_k\}$$

Realizando el mismo proceso con g', observamos que:

$$gH \cap g'H \neq \emptyset \implies gh = g'h' \text{ para algunos } h, h' \in H$$

Por ser G grupo

$$g = g'h'h^{-1}.$$

Tomando $h_i \in H$ se tiene

$$gh_i \in H \implies gh_i = g'h'h^{-1}h_i$$

Como $h'h^{-1}h_i \in H$ por ser multiplicación de elementos de H:

$$gh_i \in g'H \implies gH = g'H.$$

Es decir, sean $g, g' \in G$ los conjuntos gH y $g'H$ son o bien el mismo o bien no tienen elementos en común.

De esta manera $\{gH \mid g \in G\}$ es una partición de G.

Además $gh_i = gh_j \iff h_i = g^{-1}gh_j = h_j$ entonces

$$|G| = k = |gH|.$$

De esta manera, $G = \bigcup gH$ y por tanto

$$n = |G| = k + \cdots + k \implies k \mid n. \qquad \square$$

Un corolario nos da la generalización del Pequeño Teorema de Fermat para el orden de un grupo. Pero primero vamos a ver como se define el subgrupo de potencias de un elemento de G.

Definición 3.6. Sea G un grupo finito y $g \in G$, llamamos subgrupo generado por g, denotado $\langle g \rangle$, al subconjunto formado por las potencias de g,

$$\langle g \rangle = \{1, g^1, \cdots, g^i\},$$

dotado de la misma operación que G.

Decimos que $|\langle g \rangle| = i$ es el orden de g.

Corolario 3.7. *Sea G un grupo de orden $|G| = n$, y sea $g \in G$ se tiene $g^n = 1$ siendo 1 el elemento neutro del grupo.*

Demostración. Tomamos el conjunto de potencias de g: $\{g, g^2, \cdots, g^i, \cdots\}$.

Por ser $|G|$ finito, tenemos que existen $i, j \in \mathbb{N}$ tales que $g^i = g^j$. De lo que se sigue:

$$g^i = g^j \iff g^i \cdot g^{-j} = g^{i-j} = g^j \cdot g^{-j} = 1.$$

Es decir, existe un mínimo $k \in \mathbb{N}$ tal que $g^k = 1$.

De esta manera $\langle g \rangle = \{1, g, g^2, \cdots, g^{k-1}\}$, ya que $g^k = 1$.

Tomando ordenes

$$|\langle g \rangle| \,\Big|\, |G|.$$

De esta forma $g^k = 1$, y $k \,\Big|\, |G|$. Por lo que $g^n = g^{k \cdot m} = (g^k)^m = 1^m = 1$. $\square$

Se comprueba que para el grupo $|U(\mathbb{Z}/p\mathbb{Z})|$ con p primo,

$$|U(\mathbb{Z}/p\mathbb{Z})| = p - 1.$$

Como $U(\mathbb{Z}/p\mathbb{Z})$ es un grupo

$$a^{p-1} = 1 \pmod{p} \qquad \forall a \in \mathbb{Z}/p\mathbb{Z}.$$

3.2. Grupo de puntos en una curva elíptica

El conjunto por el que sustituimos $U(\mathbb{Z}/p\mathbb{Z})$ en el algoritmo de Lenstra requiere construir una curva elíptica sobre un cuerpo.

Definición 3.8. Un *cuerpo* $\mathbb{K}$ es un conjunto con dos operaciones binarias internas, suma y multiplicación, ambas operaciones con las propiedades asociativa y conmutativa como las definidas para grupos y las siguientes propiedades:

- Elementos neutro para ambas:

$$\exists\, 0 \in \mathbb{K} : a + 0 = a \quad \forall\, a \in \mathbb{K},$$

$$\exists\, 1 \in \mathbb{K} : b \times 1 = b \quad \forall\, b \in \mathbb{K}$$

- Elemento opuesto y elemento inverso:

$$\forall\, a \in \mathbb{K} \;\; \exists\, -a : -a + a = 0,$$

$$\forall\, 0 \neq b \in \mathbb{K} \;\; \exists\, b^{-1} : b^{-1} \times b = 1$$

- Distribuidad:

$$a, b, c \in \mathbb{K} \implies a \times (b + c) = (a \times b) + (a \times c)$$

Algunos ejemplos de esta definición son:

Ejemplos 3.9.

- El conjunto de números reales $\mathbb{R}$ con la suma y la multiplicación real es un cuerpo.
- El conjunto $\mathbb{Z}$ no es un cuerpo ya que solo 1 y -1 tienen elemento inverso.
- El conjunto $\mathbb{Z}/n\mathbb{Z}$ es un cuerpo si y solo si n es primo, ya que, por el teorema 3.3, solo tienen inverso los números coprimos con n.

Podemos definir ecuaciones polinómicas con coeficientes en cuerpos ya que tenemos las operaciones de suma y multiplicación.

Por ejemplo podemos definir la ecuación $y = x + 1$ sobre $\mathbb{Z}/2\mathbb{Z}$, cuyas soluciones son $(0, 1)$ y $(1, 0)$.

3.2.1. Grupo de una curva elíptica

Las curvas elípticas están definidas en el plano proyectivo de en cuerpo:

Definición 3.10. Dado $\mathbb{K}$ un cuerpo, el plano proyectivo sobre $\mathbb{K}$ es el cociente

$$\mathbb{P}^2(\mathbb{K}) = \mathbb{K}^3 \setminus \{(0,0,0)\} \Big/ \sim,$$

con la relación de equivalencia

$$(x, y, z) \sim (tx, ty, tz) \text{ para } t \in \mathbb{K} \setminus 0.$$

La clase de equivalencia de (x, y, z) se denota por $[x : y : z]$.

Observemos que si $F \in \mathbb{K}[X, Y, Z]$ es homogéneo y $F(x, y, z) = 0$, entonces $F(tx, ty, tz) = 0$ para todo $t \in \mathbb{K}$. Por lo que tiene sentido hablar ceros de F es $\mathbb{P}^2(\mathbb{K})$.

Definición 3.11. Una *curva* en $\mathbb{P}^2(\mathbb{K})$ es el lugar de ceros de un polinomio homogéneo $F \in \mathbb{K}[X, Y, Z]$. Decimos que es *lisa* si

$$\left\{(X, Y, Z) \in \mathbb{P}^2(\mathbb{K}) \;\middle|\; \frac{\sigma F}{\sigma X} = 0,\ \frac{\sigma F}{\sigma Y} = 0,\ \frac{\sigma F}{\sigma Z} = 0,\ F(X, Y, Z) = 0\right\} = \emptyset. \tag{3.1}$$

El *género* de una curva lisa es la cantidad

$$g = \frac{(d-1)(d-2)}{2} = 1 \text{ con } d \text{ el grado de la ecuación que la define.}$$

Una curva elíptica es una curva lisa de género 1.

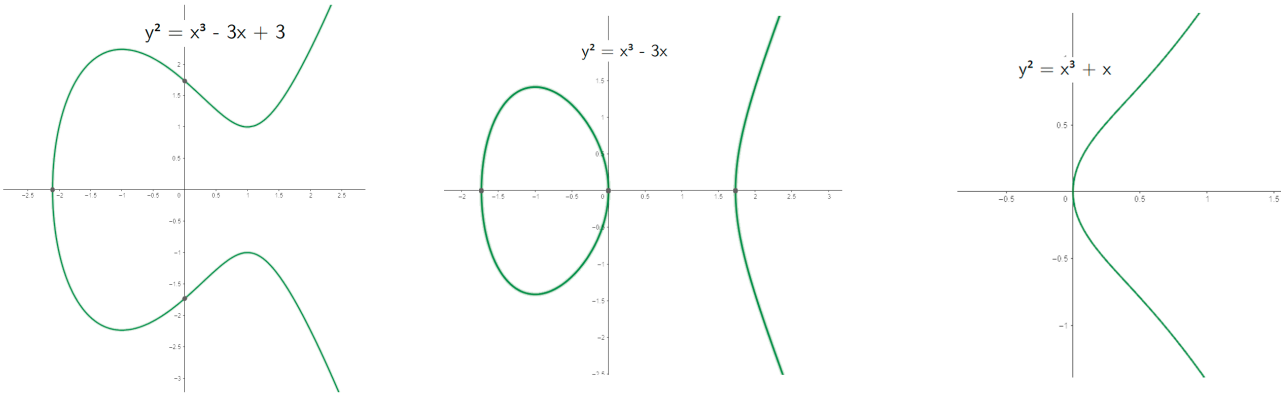

Figura 3.1: Algunos ejemplos de formas de Weierstrass.

La ecuación general de una curva elíptica es

$$F(X,Y,Z) = a_1X^3 + a_2Y^3 + a_3Z^3 + a_4X^2Y + a_5XY^2 + a_6X^2Z + a_7XZ^2 + a_8Y^2Z + a_9YZ^2 + a_{10}XYZ = 0,$$

cumpliendo la condición de la ecuación 3.1.

Resultará más sencillo trabajar en el espacio afín (no homogéneo) $(\mathbb{Z}/p\mathbb{Z})^2$. Para ello vamos a considerar la recta $[0:1:0]$ como la *recta del infinito* sobre la que deshomogeneizaremos. Por el teorema de Bézout [12], al ser una recta (curva de grado 1) corta a la curva elíptica en 3 puntos. En nuestro caso queremos que corte a $[0:1:0]$ en un punto triple.

Para realizar este proceso primero tomamos la intersección con el plano $Z = 0$, obteniendo

$$F(X,Y,Z) \cap \{z = 0\} = a_1X^3 + a_2Y^3 + a_4X^2Y + a_5XY^2 = 0.$$

Para que $[0:1:0]$ tenga un punto de intersección triple, necesitamos que la ecuación sea de la forma $aX^3 = 0$. De esta forma nos queda la ecuación con un punto de intersección triple con $[0:1:0]$:

$$Y^2Z + AXYZ + CYZ^2 = X^3 + BX^2Z + DXZ^2 + FZ^3.$$

Finalmente, para obtener una ecuación de la curva elíptica en el plano afín vamos a escribir la forma de Weierstrass, definida ya en coordenadas no-homogéneas $\mathbb{K}^2$ y obtenida a través de realizar un cambio de variable a la ecuación general. Para ver el cambio de variable completo consultar el apéndice A.

Definición 3.12. Sean $\mathbb{K}$ un cuerpo de característica[1] distinta de 2 y 3, y A y $B \in \mathbb{K}$. Se llama *ecuación de una curva elíptica en la forma de Weierstrass definida en* $\mathbb{K}$ (de ahora en adelante nos referiremos a ella como "forma de Weierstrass") a una igualdad de la forma:

$$E : y^2 = x^3 + Ax + B$$

tal que el discriminante es no nulo en el cuerpo, en símbolos $\triangle := -16(4A^3 + 27B^2) \neq 0$.

La figura 3.1 muestra algunos ejemplos de curvas elípticas en forma de Weierstrass. Se observa que son simétricas respecto del eje $y = 0$, y que no hay puntos de corte consigo mismas.

[1]La característica de un cuerpo $\mathbb{K}$ es un primo p que cumple $p \cdot 1 = 0 \in \mathbb{K}$. Si no existe dicho primo, decimos que la característica es 0.

Podemos utilizar la ecuación de Weierstrass para trabajar en $\mathbb{K}^2$, pero realmente, una ecuación de una curva elíptica está definida en $\mathbb{P}^2(\mathbb{K})$, que es un espacio homogéneo. Veamos cómo se pasa de $\mathbb{P}^2(\mathbb{K})$ a $\mathbb{K}^2$.

Homogeneizando la ecuación de Weierstrass nos queda:

$$y^2z = x^3 + Axz^2 + Bz^3.$$

Como para homogeneizar la ecuación hemos añadido z el cambio de variable para volver a la forma no-homogénea será $x \to x/z$ y $y \to y/z$ con lo que queda

$$y^2z^3 = x^3z^3 + Axz^3 + B.$$

Para $z = 1$ (que es equivalente a $z \neq 0$ en $\mathbb{P}^2(\mathbb{K})$) obtenemos de nuevo la forma de Weierstrass, pero nos quedaría ver que ocurre cuando $z = 0$. Tomando $z = 0$ observamos que

$$\begin{aligned} y^2 \cdot 0 &= x^3 + Ax \cdot 0 + B \cdot 0, \\ 0 &= x^3, \end{aligned}$$

para cualquier y.

Es decir, el punto $[0, 1, 0] \in \mathbb{P}^2$ es solución de la ecuación de Weierstrass en el plano proyectivo, pero no está representado en las coordenadas no-homogéneas $\mathbb{K}^2$. Se le llama *punto del infinito* y lo denotaremos como O.

Podemos dar una versión afín de la definición de curva elíptica que usaremos en este trabajo.

Definición 3.13. Sea $\mathbb{K}$ un cuerpo y E una ecuación de una curva elíptica definida en $\mathbb{K}$.

Al conjunto de puntos que cumplen la ecuación E en $\mathbb{K}^2$ con el punto del infinito:

$$\{(x, y) \in \mathbb{K}^2 \text{ tales que } E(x, y) : y^2 = x^3 + Ax + B\} \cup \{O\}$$

se le llama *curva elíptica dada por E*.

Denotamos a este conjunto como $E(\mathbb{K})$.

Además denotaremos por $\#E(\mathbb{K})$ al número de puntos que tiene la curva elíptica definida en $\mathbb{K}$.

Veamos algunas observaciones:

- Toda recta que corte a la curva elíptica en el punto del infinito y en otro punto, es equivalente en $\mathbb{K}^2$ a una recta vertical:

  Una recta que corte al punto $O = [0 : 1 : 0]$ y a un punto $p_0 = [a : b : c]$ es de la forma

  $$[x : y : z] = [0 : 1 : 0] + t(p_1 - [0 : 1 : 0]) \tag{3.2}$$

  $$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ta \\ 1 + t(b-1) \\ tc \end{bmatrix} \tag{3.3}$$

  Aplicando el cambio de variable no-homogéneo $x/z \to x, y/z \to y$

  obtenemos:

  $$\frac{x}{z} = \frac{ta}{tc} = \frac{a}{c} = cte \tag{3.4}$$

  $$\frac{y}{z} = \frac{1 + t(b-1)}{tc}. \tag{3.5}$$

  Es decir, en la recta solo varía la coordenada y, luego todas las rectas que pasen por el punto del infinito $[0, 1, 0]$ serán rectas verticales en $\mathbb{K}^2$.

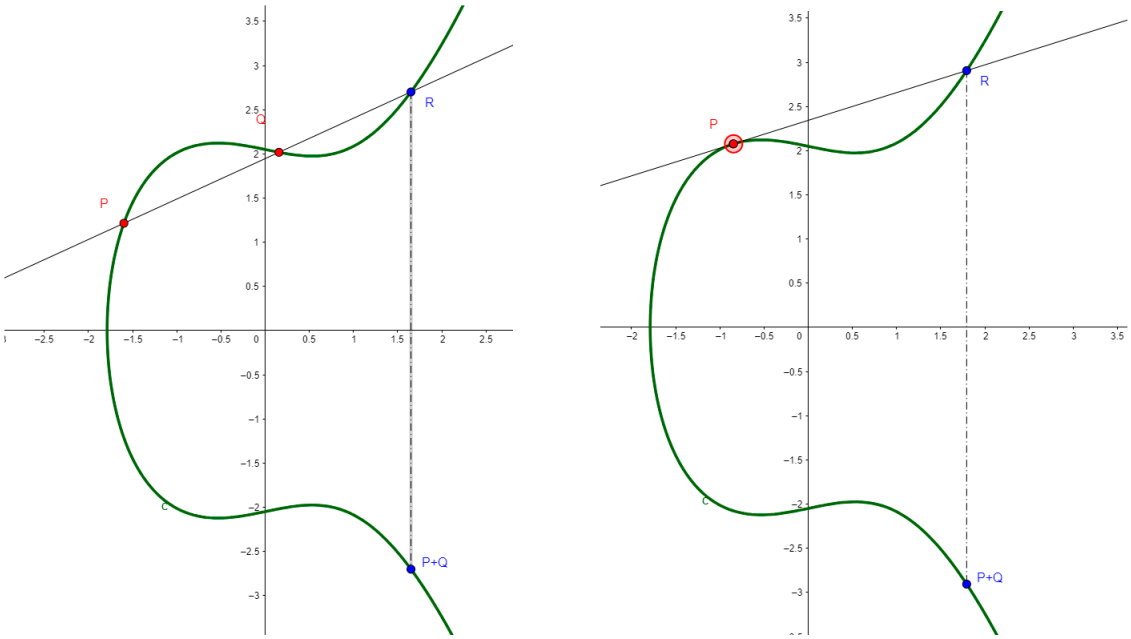

Figura 3.2: Suma de puntos en una curva elíptica.

- En sentido contrario, toda recta de la forma $x = c$ corta a la curva elíptica en el punto del infinito, en el punto $(c, \sqrt{c^3 + Ac})$ y en el punto $(c, -\sqrt{c^3 + Ac})$.
- Toda recta $r : y = ax + c$ (incluida la recta vertical con $a = \infty$) corta a la curva elíptica en exactamente 3 puntos, si consideramos que si es tangente a la curva en un punto, la recta corta a la recta en un punto doble:

  Sea E la ecuación como en la definición, sustituyendo y nos queda:

  $$(ax + c)^2 = x^3 + Ax + B.$$

  Que es una ecuación de grado 3 y por lo tanto con 3 soluciones x_1, x_2, x_3 y que definen los 3 puntos de corte: $(x_1, a \cdot x_1 + c), (x_2, a \cdot x_2 + c), (x_3, a \cdot x_3 + c)$.
- Toda curva elíptica dada por una ecuación en la forma de Weierstrass es simétrica respecto del eje $y = 0$.

Tenemos ya el conjunto en el que va a trabajar el algoritmo de Lenstra: $E(\mathbb{Z}/n\mathbb{Z})$. Nos falta ver hay una estructura de grupo en este conjunto:

Veamos primero la explicación geométrica de la operación suma en el conjunto $E(\mathbb{Z}/n\mathbb{Z})$.

Definición 3.14. Veamos el proceso que da la suma de puntos en E (véase la figura 3.2):

1. Sean $P, Q \in E(\mathbb{K})$ puntos en una curva elíptica y sea L la recta que une P y Q (Si $P = Q$ tomamos la recta tangente a E por ese punto).
2. La recta L corta a E en un tercer punto al que llamamos R.
3. Trazamos la recta L' que une el punto del infinito O con el punto R. Como hemos explicado antes, L' es la recta vertical que pasa por R.
4. La recta L' corta a la curva por un tercer punto al que llamaremos $P + Q$.

El punto $P + Q$ será la suma de P y Q.

Efectivamente esta operación entre puntos de $E(\mathbb{K})$ define una estructura de grupo:

Proposición 3.15. *Sea E una curva elíptica definida en* $\mathbb{K}$*, la operación descrita arriba cumple:*

1. *Es interna: Sean* $P, Q \in E(\mathbb{K})$ *entonces* $P + Q \in E(\mathbb{K})$.
2. *Sea L una recta que interseca a E en 3 puntos* P, Q, R *entonces* $(P + Q) + R = O$.
3. *Es conmutativa:* $P, Q \in E$ *entonces* $P + Q = Q + P$.
4. *Tiene elemento neutro:* $P \in E$ *entonces* $P + O = P$.
5. *Tiene elemento inverso:* $P \in E$ *entonces existe* $-P$ *tal que* $P + (-P) = O$.

Demostración.

1. Si P y Q tienen coordenadas en $\mathbb{K}^2$, entonces la ecuación de la recta conectando los dos puntos tiene coeficientes en $\mathbb{K}$, y puesto que $\mathbb{K}$ es un cuerpo, podremos despejar las coordenadas del punto $P + Q$ de la ecuación de la recta. (El cálculo específico lo veremos cuando hayamos definido las operaciones aritméticas.)
2. El punto R está en la misma vertical que $P + Q$ luego la recta que los une tendrá como tercer punto O y como suma el punto del infinito de nuevo.
3. Puesto que la recta que une P y Q solo corta a E en un punto más, ese tercer punto será igual para $P + Q$ que para $Q + P$ y por lo tanto la suma será la misma.
4. Tomando $Q = O$ en la definición de suma, vemos que las lineas L y L' coinciden, luego el proceso devuelve $P + O = P$.
5. Por los apartados 2 y 4 respectivamente en las siguientes igualdades:

$$O = (P + O) + R = P + R.$$

Luego $-P$ es el otro corte con E de la vertical que pasa por P. □

La demostración de la asociatividad requiere de un desarrollo más amplio de la aplicación del teorema de Cayley-Bachalach. Puede verse esta demostración en el Apéndice B.

Veamos la definición en coordenadas de la suma de puntos de una curva elíptica en forma de Weierstrass.

Sea E una curva elíptica dada por la ecuación

$$F(x, y) = y^2 - x^3 + Ax - B = 0.$$

Sea $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2)$ puntos de la curva elíptica E.

En el caso en que $x_1 = x_2$ pero $y_1 \neq y_2$ tenemos dos puntos en misma linea vertical, luego $P + Q = O$. En el caso en que $x_1 \neq x_2$ la recta L que pasa por P_1 y P_2 tiene una ecuación de la forma:

$$L : y = \lambda \cdot x + \nu.$$

Sustituyendo las coordenadas de P y Q para que pase por ellos se obtiene:

$$y_1 \lambda \cdot x_1 + \nu \quad \text{y} \quad y_2 \lambda \cdot x_2 + \nu.$$

Despejando λ y ν obtenemos:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{y} \quad \nu = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}.$$

En el caso en que $P_1 = P_2$ derivando la ecuación de la curva obtenemos

$$2y(x)y'(x) = 3x_1^2 + A,$$

y de ahí la pendiente

$$\lambda = \frac{3x_1^2 + A}{2y_1}.$$

Para hallar ν despejamos de las ecuaciones de la recta y la curva

$$y_1 = (\lambda x_1 + \nu)^2 = x_1^3 + Ax_1 + B.$$

Resolviendo queda

$$\nu = \frac{-x_1^3 + Ax_1 + 2B}{2y_1}.$$

Observar que $y_1 = 0$ si y solo si $(x_1, y_1) = (x_2, y_2) = (x_1, 0)$ que coincide con el vértice de la curva.

Ahora, por el teorema anterior, tenemos que si una recta corta a E en tres puntos P_1, P_2, P_3, entonces

$$(P_1 + P_2) + P_3 = O.$$

Además como esos tres puntos es donde corta la recta a la curva, sabemos que van a ser raíces de $F(x, \lambda x + \nu)$. Luego solo queda igualar coeficientes:

$$F(x, \lambda x + \nu) = \lambda^2 x^2 + 2\upsilon\lambda x + \nu^2 = (x - x_1)(x - x_2)(x - x_3).$$

Obtenemos entonces

$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = -\lambda x_3 - \nu.$$

Nota 3.16. Observar que para realizar la suma de puntos necesitamos calcular el inverso de un número en $\mathbb{Z}/n\mathbb{Z}$.

Para ello recordar que podemos utilizar el método de Bézout descrito en 1.1.1 y aplicado como se explica en la nota 3.4.

Ahora que tenemos las formulas aritméticas para realizar la suma de puntos en una curva elíptica podemos reescribir las propiedades:

1. Sea $P = (x, y) \in E$ entonces $-P = (x, -y)$.
2. Sea $P = (x_1, y_1) \in E$ y $Q = (x_2, y_2) \in E$ entonces $P + Q$ tiene coordenadas:

$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = -\lambda x_3 - \nu.$$

Hemos demostrado finalmente que $E(\mathbb{K})$ tiene estructura de grupo.

Además observemos que en el caso en que $\mathbb{K}$ no fuera un cuerpo y no tuviera inverso para la multiplicación, no podríamos realizar la suma, sólo la podríamos realizar para los elementos que tuvieran inverso en $\mathbb{Z}/n\mathbb{Z}$. Por lo que perdemos la estructura de grupo si la curva no está definida en un cuerpo.

Capítulo 4

El algoritmo de Lenstra

El algoritmo de Lenstra para factorizar un número compuesto N consiste en:

1. Tomamos una ecuación de curva elíptica $E(x, y)$, definida en el conjunto (no en el cuerpo) $\mathbb{Z}/N\mathbb{Z}$. Es decir, tomamos valores de x e y en $\mathbb{Z}/N\mathbb{Z}$ que sean soluciones de la ecuación.

2. Realizamos sumas de puntos en $E(x, y)$ con el mismo proceso explicado en el apartado anterior.

3. Habrá puntos que no podamos sumar, por que sus coordenadas no tendrán inverso en $\mathbb{Z}/N\mathbb{Z}$. En ese caso, por el teorema 3.3, habremos encontrado un número con algún factor primo en común con N.

Nota 4.1. En lo que sigue usaremos la notación $E(\mathbb{Z}/N\mathbb{Z})$ para el conjunto de ceros de ecuación E con coeficientes en $\mathbb{Z}/N\mathbb{Z}$ que no es un cuerpo. Evidentemente $\#E(\mathbb{Z}/N\mathbb{Z}) < \infty$ ya que el plano $(\mathbb{Z}/N\mathbb{Z})^2$ tiene ya un número finito de puntos, y estamos tomando un subconjunto de este conjunto más el punto del infinito.

Para encontrar los puntos que no podemos sumar, podemos probar cualquier combinación de dos puntos de $E(\mathbb{Z}/N\mathbb{Z})$ hasta que obtengamos una pareja de puntos en la que no podamos realizar la suma. Sin embargo este proceso sería poco eficiente, así que vamos a tomar la idea del algoritmo $p-1$ de Pollar de utilizar el Teorema de Lagrange para acelerar este proceso.

Partimos de un punto $P \in E(\mathbb{Z}/N\mathbb{Z})$ y calculamos $[n]P$ (sumaremos n veces P), es decir, de manera que $|G| \mid n$ y por el Teorema de Lagrange tengamos

$$[n]P = [m][|G|]P = m \cdot 0.$$

Para conseguir este múltiplo calcularemos de manera similar al algoritmo $p-1$ de Pollard, $[i!]P$. Para ello se usa el algoritmo dobla y suma.

4.1. Dobla y suma

Sea $E(\mathbb{K})$ una curva elíptica sobre el cuerpo $\mathbb{K}$ y sea $P \in E(\mathbb{K})$. Si queremos calcular $[n]P$ una manera obvia de hacerlo sería ir sumando P hasta llegar a n veces.

$$P,\ [2]P = P + P, [3]P = [2]P + P, \ldots, [n]P = [n-1]P + P.$$

Pero este primer método es demasiado costoso para n muy grande.

Vamos a ver un algoritmo válido para hallar la potencia n-ésima de un elemento de cualquier grupo de manera más eficaz.

Algoritmo 4 Algoritmo dobla y suma para calcular la potencia n de un elemento de un grupo

Entrada 1: $P \in G$ con G grupo. **Entrada 2:** $n \in \mathbb{N}$.

1: Escribir n como expresion binaria:
$$n = \epsilon_0 + \epsilon_1 \cdot 2 + \epsilon_2 \cdot 2^2 + \epsilon_3 \cdot 2^3 + \cdots + \epsilon_t \cdot 2^t \qquad \text{con } \epsilon_i \in 0, 1 \text{ y } \epsilon_t = 1.$$
2: $Q \leftarrow P$.
3: $R = \begin{cases} O & \text{si } \epsilon_0 = 0, \\ P & \text{si } \epsilon_0 = 1. \end{cases}$
4: **for** $i \in \{1, 2, \cdots, t\}$. **do**
5: $\quad Q \leftarrow [2]Q$.
6: $\quad$ **if** $\epsilon_i = 1$. **then**
7: $\quad\quad R \leftarrow R + Q$.

Devuelve: R. ▷ $R = [n]P$

Teorema 4.2. *El algoritmo 4 calcula* $[n]P$ *siendo* $P \in G$ *un elemento del grupo* G*, en no más que* $2 \cdot \log_2(n)$ *sumas.*

Demostración. Durante la i-ésima iteración del bucle (pasos(4)-(7)), el valor de Q es $[2^i]P$. Como el valor de R aumenta si y solo si $\epsilon_i = 1$, el valor final de R es

$$\sum_{i \text{ tal que } \epsilon_i = 1} [2^i]P = \sum_{i=0}^{t} [\epsilon_i 2^i]P = \left[\sum_{i=0}^{t} \epsilon_i 2^i\right] P = [n]P.$$

Cada iteración del bucle realiza una vez $Q + Q$ y a lo sumo una suma $R + Q$. Como $t \leq \log_2(n)$, el número de pasos requerido es el establecido en el enunciado. □

Debemos calcular $[n!]P$ y para ello calcularemos iterativamente $[i!]P = [(i-1)!]P + [i]P$.

Además, recordemos que al no ser un cuerpo $\mathbb{Z}/N\mathbb{Z}$, no existirá $[2]Q$ o $R + Q$ para alguna iteración i del bucle (pasos(4)-(7)). En el momento que esto ocurra, tendremos un divisor de N.

4.2. Algoritmo para curvas elípticas

El algoritmo de factorización de Lenstra sobre curvas elípticas es:

Algoritmo 5 Algoritmo de Lenstra para la factorización de N

Entrada: $N \in \mathbb{Z}$ compuesto.

1: Elegir una cota L
2: Escoger una curva elíptica $E \pmod N$ y un punto $P \in E(\mathbb{Z}/N\mathbb{Z})$
3: $Q \leftarrow P$
4: **for** $i \in \{1, 2, \ldots, L.\}$ **do**
5: $\quad Q \leftarrow [i]Q$ ▷ Siempre trabajando en $\mathbb{Z}/N\mathbb{Z}$
6: $\quad$ **if** no existe inverso de un elemento $a \in \mathbb{Z}/N\mathbb{Z}$ **then**
7: $\quad\quad$ **if** $\text{mcd}(a, N) \neq N$ **then**
8: $\quad\quad\quad f \leftarrow \text{mcd}(a, N)$
9: $\quad\quad$ **Break for**
10: **if** $f = N$ **then** Ir al paso(1) y Elegir una nueva curva y un nuevo punto P.

Salida: f. ▷ f factor no trivial de N

Nota 4.3. Los pasos (5) y (6) requieren recordar algoritmos explicados anteriormente:

- Para el cálculo del paso (5) utilizamos el algoritmo Dobla y Suma 4 sobre el punto Q y con $n = i$.
- Para el paso (6) primero tendremos que calcular $\text{mcd}(a, N)$ por el método de Euclides 1 y de ser coprimos calculamos el inverso por Nota 3.4.

Una vez visto como funciona el algoritmo, comprobemos primero que funciona el bucle y más tarde veremos que eligiendo bien la curva E y el punto P el algoritmo finaliza con un factor no trivial de N.

Teorema 4.4. *Sea N un número compuesto. Supongamos que N tiene un factor primo p tal que L y la curva E satisfacen*

$$\#E(\mathbb{Z}/p\mathbb{Z}) = q_1^{e_1} q_2^{e_2} \cdots q_t^{e_t} \text{ y } L \geq \text{máx}\{q_1 e_1, \ldots, q_t e_t\},$$

con $q_1, \ldots, q_t$ primos distintos. Entonces, el bucle del algoritmo 5 (pasos (4-9)) termina devolviendo un factor distinto de 1 de N o con $a = N$.

Demostración. Consideremos una curva elíptica tal que su grupo $E(\mathbb{Z}/p\mathbb{Z})$ que tiene orden

$$\#E(\mathbb{Z}/p\mathbb{Z}) = q_1^{e_1} q_2^{e_2} \cdots q_t^{e_t}.$$

Por el teorema de Lagrange, para cualquier elemento P se tiene

$$[\#E(\mathbb{Z}/p\mathbb{Z})]P = \mathcal{O},$$

y por lo tanto

$$[L!]P = [k][\#E(\mathbb{Z}/p\mathbb{Z})]P = [k]\mathcal{O} = \mathcal{O} \text{ para algún } k \in \mathbb{Z}/p\mathbb{Z}.$$

Ahora, vamos a ver qué ocurre cuando el algoritmo llega a una suma de puntos $P_1, P_2 \neq \mathcal{O}$ en $E(\mathbb{Z}/N\mathbb{Z})$ cuyas coordenadas en $\pmod{(p)}$ son dos puntos P'_1, P'_2 en $E(\mathbb{Z}/p\mathbb{Z})$ tales que $P'_1 + P'_2 = \mathcal{O}$.

Observar que para P_1, P_2 con coordenadas $(x_1, y_2), (x_2, y_2)$, para que $P'_1 + P'_2 = \mathcal{O}$, tiene que ocurrir que $x_1 = x_2$ y $y_1 + y_2 = 0 \pmod{(p)}$. Es decir

$$p \mid (x_1 - x_2) \quad \text{y} \quad p \mid (y_1 + y_2).$$

Por lo tanto, para sumar los puntos P_1 y P_2 en $E(\mathbb{Z}/N\mathbb{Z})$, necesitamos el inverso de $(x_1 - x_2)$ o de (y_1) o (y_2) (que serían el mismo). Pero este inverso no existe, ya que son múltiplos de un factor primo p de N, es decir, no son coprimos con N.

Por lo tanto el algoritmo encontrará $a = (x_1 - x_2)$ o $a = (y_1)$ y sabremos que $\text{mcd}(a, N) \neq 1$. □

Nota 4.5. Aunque es posible trabajar sobre el plano proyectivo, es más sencillo realizar el algoritmo sobre la ecuación de Weierstrass 3.12

$$E : y^2 = x^3 + Ax + B \qquad \text{y} \qquad \triangle = -16(4A^3 + 27B^2),$$

con $A, B \in \mathbb{Z}/N\mathbb{Z}$ y trabajar con coordenadas en este conjunto $(\mathbb{Z}/N\mathbb{Z})^2$. Esto facilita la implementación.

Nota 4.6. Elegir un punto en una curva elíptica $\pmod{N}$ con N compuesto es una tarea difícil, pero no tenemos por qué realizarla. Podemos elegir la curva elíptica y el punto a la vez de la siguiente manera. Elegimos un A para la ecuación de Weierstrass y un punto $P = (x_0, y_0)$. Definimos $B = y_0^2 - x_0^3 - Ax_0$.

En el caso en que nuestro discriminante sea nulo,

$$\triangle = -16(4A^3 + 27B^2) = 0,$$

tendremos que $\text{mcd}(\triangle, N) \neq 1$.

Nota 4.7. El algoritmo tal cual está enunciado puede no terminar por la elección aleatoria de curva y punto. Sin embargo, si empezando con $P = (1, 1)$ y en cada nueva elección del punto hacer el cambio de coordenadas $x_0 \leftarrow x_0 + 1, y_0 \leftarrow y_0 + 1$, en algún momento ocurrirá que para hacer $[2]P$ en la primera iteración obtendremos

$$x_0 = x_0 \quad \text{y} \quad \text{mcd}(y_0, N) = y_0 \neq 1.$$

Por lo tanto, no podremos realizar la suma ya que y_0 no tendría inverso. Habiendo encontrado el factor primo más bajo de N.

El apéndice D contiene una implementación del algoritmo en C++ que factoriza números que sean producto de dos primos.

4.3. Mejora respecto del grupo de unidades

Hemos visto que el funcionamiento del algoritmo es igual que el de Pollard. Pero nos queda ver la diferencia respecto a este.

Para ello vamos a comparar el orden del grupo de unidades de $\mathbb{Z}/N\mathbb{Z}$ con el orden del grupo de puntos de $E(\mathbb{Z}/p\mathbb{Z})$. Utilizamos el grupo $E(\mathbb{Z}/p\mathbb{Z})$ ya que por el teorema 4.4 el algoritmo utiliza un número de iteraciones basado en $\#E(\mathbb{Z}/p\mathbb{Z})$.

Veamos primero (sin demostración) un resultado por el cual $|E(\mathbb{Z}/p\mathbb{Z})| \leq |U(\mathbb{Z}/p\mathbb{Z})|$.

Teorema 4.8 (de Hasse)**.** *Sea $E(\mathbb{Z}/p\mathbb{Z})$ una curva elíptica con p primo. Se tiene que*

$$|\#E(\mathbb{Z}/p\mathbb{Z}) - (p + 1)| \leq 2\sqrt{p}.$$

La demostración se puede encontrar en [8].

Podemos observar que $\#E(\mathbb{Z}/p\mathbb{Z})$ está acotado por una función de p, es decir, toda curva elíptica sobre $\mathbb{Z}/p\mathbb{Z}$ va a tener la misma cota del teorema de Hasse. Además, se sabe que para todo entero $a \in [p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$ existe una curva E tal que $\#E(\mathbb{Z}/p\mathbb{Z}) = a$, que se puede construir ([5] y [10]). Sin embargo, este no es el método que comúnmente se usa ya que el construir la curva requiere de muchos pasos, lo que haremos será tomar curvas aleatorias y tomar una cota para el número de iteraciones.

El siguiente teorema nos da una cota y un número de pasos esperado.

Teorema 4.9 (Canfield, Ërdos y Pomerance [11].)**.** *Sea $N \in \mathbb{N}$, p un factor primo de N y B la cota óptima para encontrar p por el algoritmo 5.*

Entonces, $B = L(p)^{\frac{\sqrt{2}}{2}}$ y el tiempo de ejecución del algoritmo es

$$\exp \sqrt{\log(x)\log(\log(x))} \quad \textit{pasos.}$$

Con este resultado podemos ver que, para N que cumplen la condición de que sus factores primos menos 1 cumplen la propiedad de ser B-liso para B bajo, el algoritmo de Lenstra funciona especialmente bien [4]. Sin embargo, en general para cualquier natural, el tiempo de ejecución es mayor que el del algoritmo de "criba general del cuerpo de número". El cual toma un tiempo

$$\exp\left(c^3 \sqrt[3]{(\log(N)(\log\log N)^2)}\right)$$

para factorizar un natural N cualquiera.

Capítulo 5

Conclusiones y futuro

El algoritmo $p-1$ de Pollard resulta ser un algoritmo eficaz para factorizar números tales que uno de sus factores primos p, tal que $p-1$ se factoriza en primos pequeños. El algoritmo se basa en el pequeño teorema de Fermat, el cual nos asegura que cualquier número a coprimo con p cumple que $a^{p-1}-1$ es múltiplo de p, y por lo tanto tiene un factor común con n.

Este algoritmo no es en principio especialmente eficiente, pero se puede modificar generalizando el concepto del pequeño teorema de Fermat y aplicarlo a otros grupos. En el caso de la modificación de Lenstra, se aplica el teorema de Lagrange al grupo de puntos de una curva elíptica, convirtiendo el algoritmo en el más potente para los números descritos, y uno de los más potentes en general para la factorización de números enteros.

En este último caso, podemos tomar un punto P en una supuesta curva elíptica E sobre el conjunto $\mathbb{Z}/n\mathbb{Z}$. Como $\mathbb{Z}/n\mathbb{Z}$ no es un cuerpo $E(\mathbb{Z}/n\mathbb{Z})$ no es un grupo con la estructura habitual. Por lo que, al intentar sumar puntos podemos encontrarnos con una operación que no está definida. La razón de fondo está en el hecho de que $\mathbb{Z}/n\mathbb{Z}$ no es un cuerpo, y por lo tanto no todo elemento no nulo es inversible. En ese momento habremos encontrado un factor del número n. Podemos asegurar que encontraremos el factor gracias al teorema de Lagrange.

A pesar de que se carece de un estudio riguroso del coste computacional del algoritmo de Lenstra, los experimentos computacionales muestran su eficacia en la práctica. Esto ocasiona que en métodos como el cifrado RSA en los que se busca que un número sea muy costoso de factorizar, no sea buena opción utilizar números que son producto de primos pequeños, es por ello que se utilizan números que son el producto de dos primos p y q grandes y tales $p-1$ y $q-1$ no son tampoco múltiplos de números primos pequeños.

Sobre el futuro del algoritmo hay varias consideraciones que se deben hacer si se desea obtener la mayor eficiencia.

Actualmente hay lineas de investigación que modifican el algoritmo para trabajar sobre curvas hiperelípticas de género $g > 1$, es decir de la forma $y^2 = f(x)$ con $f(x)$ un polinomio de grado 5. Para ello se encaja la curva en el grupo de la Jacobiana de la curva. Este proceso se consiguió optimizar en 2010 [14] consiguiendo que fuera más rápido que el método original con curvas elípticas.

Aún con estas modificaciones, no se conoce un método de factorización de números naturales en tiempo polinomial. Sin embargo hay versiones del algoritmo de Lenstra para sistemas no-deterministas que se suponen en "la mayoría de caso" más rápidas que el algoritmo no-determinista Shor que encuentra factores primos en tiempo sub-polinomial [15].

Bibliografía

[1] A-S. Charest *Pollard's $p-1$ and Lenstra's factoring algorithms*, McGill university 2005.

https://www.math.mcgill.ca/darmon/courses/05-06/usra/charest.pdf

[2] S. S. Wagstaff. *Cryptanalysis of Number Theoretic Ciphers*. 1ª edición. Chapman and Hall/CRC 2002.

https://doi.org/10.1201/9781315275765

[3] J. H. Silverman y J. Tate. *Rational Points on Elliptic Curves*. 2ª edición. Springer 2015.

[4] J. H. Silverman. *The Arithmetic of Elliptic Curves*. 2ª edición. Springer 2010.

[5] R. Bröker. *Constructing elliptic curves of prescribed order* Leiden University scholarly publications 2006.

https://scholarlypublications.universiteitleiden.nl/access/item%3A2895327/view

[6] Q. Wang, X. Fan, H, Zang y Y. Wang. The Space Complexity Analysis in the General Number Field Sieve Integer Factorization. *Theoretical Computer Science* **630** (2016), 76–94.

[7] MIT Libraries. *18.783 Elliptic Curves*, Spring 2017

https://hdl.handle.net/1721.1/122962

[8] K. Hendricks. *On the Proof of Hasse's Theorem*

https://cocalc.com/share/5d54f9d642cd3ef1affd88397ab0db616c17e5e0/www/edu/2004/24g/projects/kristen.pdf?viewer=download

[9] I. Tolkov. *Counting points on elliptic curves: Hasse's theorem and recent developments*. University of Washintong 2009

https://sites.math.washington.edu//~morrow/336_09/papers/Igor.pdf

[10] M. Deuring. Die Typen der Multiplikatorenringe elliptischer Funktionenköper. *Abh.Math.Semin.Univ.Hambg.* **14** (1941), 197–272.

https://hdl.handle.net/1721.1/122962

[11] E.R.Canfield, P. Erdös y C. Pomerance On a Problem of Oppenheim concerning "Factorisatio Numerorum´´. *Journal of Number Theory* **17** (1983), 1–28.

[12] R. P. Hulst. *A proof of Bezout's theorem using the Euclidean algorithm*

https://www.universiteitleiden.nl/binaries/content/assets/science/mi/scripties/bachelor/-2016/110830_btheses_hulst_fwn_math.pdf

[13] H. M. Hashim y R. K. K. Ajeena The Computational Complexity of the Elliptic Curve Factorization Algorithm over Real Field. *J. Phys.: Conf. Ser.* (2021) 1897 012046

doi:10.1088/1742-6596/1897/1/012046

[14] R. Cosset. Factorization with genus 2 curves. *arXiv*. doi.org/10.48550/arXiv.0905.2325

[15] D. J. Bernstein; N. Heninger; P. Lou; L. Valenta. Post-quantum RSA *Lecture Notes in Computer Science, Springer* **10346** (2017) doi:10.1007/978-3-319-59879-6_18

Apéndice A

Paso de la ecuación general de una curva elíptica a la forma de Weierstrass

Comenzamos con una ecuación definida sobre el cuerpo $\mathbb{K}$ de la forma

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3,$$

cuyas soluciones en el plano proyectivo $\mathbb{P}^2(\mathbb{K})$ forman una curva elíptica.

Para pasar del plano elíptico al plano afín se realiza un cambio de coordenadas no-homogéneo $x = X/Z$ e $y = Y/Z$. Resulta la ecuación

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

Observar que, como decimos en 3.2.1, existe un punto extra, el punto del infinito $O = [0, 1, 0]$.

Suponiendo que tiene característica $\text{char}(\mathbb{K}) \neq 2$, entonces podemos simplificar la ecuación completando cuadrados, es decir, con el cambio de variable

$$y \to \frac{1}{2}(y - a_1x - a_3).$$

Obtenemos una ecuación de la forma

$$E : y^2 = 4x^3 + b_2x^2 + 2b_4x + b_6.$$

con

$$b_2 = a_1^2 + 4a_4, \quad b_4 = 2a_4 + a_1a_3, \quad b_6 = a_3^2 + 4a_6.$$

Suponiendo ahora que $\text{char}(\mathbb{K}) \neq 2, 3$ entonces el cambio de variable

$$(x, y) \to \left(\frac{x - 3b_2}{36}, \frac{y}{108}\right)$$

nos da la ecuación

$$E : y^2 = x^3 - 27c_4x - 54c_6$$

con

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

Obteniendo la ecuación de Weierstrass.

Apéndice B

Asociatividad del grupo de puntos de la curva elíptica

La demostración geométrica de la asociatividad de la operación de suma de puntos en una curva elíptica $E(K)$.

Para la demostración añadiremos la notación $P * Q$ para el tercer punto de intersección de la recta que pasa por P y Q con la curva E, y denotaremos $\overline{A, B}$ a la recta que pasa por los puntos A, B, y si hay un tercer punto C por el que pasa la linea $\overline{A, B, C}$

Sean P, Q y R tres puntos en la curva. Queremos probar que $(P + Q) + R = P + (Q + R)$.

Notar que para hallar $P + Q$ se toma $P * Q$ y se halla el punto de corte de la recta vertical que pasa por ese punto y la curva elíptica. De esta manera se tiene que demostrar la asociatividad es equivalente a demostrar $(P + Q) * R = P * (Q + R)$.

Tomamos los puntos $O, P, Q, R, P * Q, Q * R, P + Q$ y $Q + R$. A partir de estos puntos tomamos los conjuntos de rectas:

$$C_1 = \left\{\overline{P, Q, P * Q},\ \ \overline{O, Q * R, Q + R},\ \ \overline{R, P + Q}\right\},$$
$$C_2 = \left\{\overline{Q, R, Q * R},\ \ \overline{O, P * Q, P + Q},\ \ \overline{P, Q + R}\right\}.$$

En la figura B.1, las rectas punteadas corresponden a C_1 y las sólidas a C_2.

Observar que las lineas $\overline{R, P + Q}$ y $\overline{P, Q + R}$ cortan a la curva elíptica E en los puntos $(P + Q) * R$ y $P * (Q + R)$ respectivamente, y además, se cortan entre ellas en un punto C. Luego si estos tres puntos coinciden, es decir, si el punto C está sobre la curva elíptica, habremos probado la asociatividad.

Observar que tenemos 9 puntos : $O, P, Q, R, P * Q, Q * R, P + Q, Q + R$ y C que comparten ambos conjuntos de rectas. Cada línea está definida por una ecuación lineal, por lo que multiplicando las 3 ecuaciones obtendríamos la ecuación de una cúbica, en la que las soluciones son las tres rectas. Tenemos pues dos curvas cúbicas C_1 y C_2 definidas por 9 puntos, entonces por el teorema de Cayley-Bacharach la curva E que pasa por 8 de esos puntos también pasa por el noveno. Es decir, $C \in E$ y por lo tanto $P * (Q + R) = (P + Q) * R = C$.

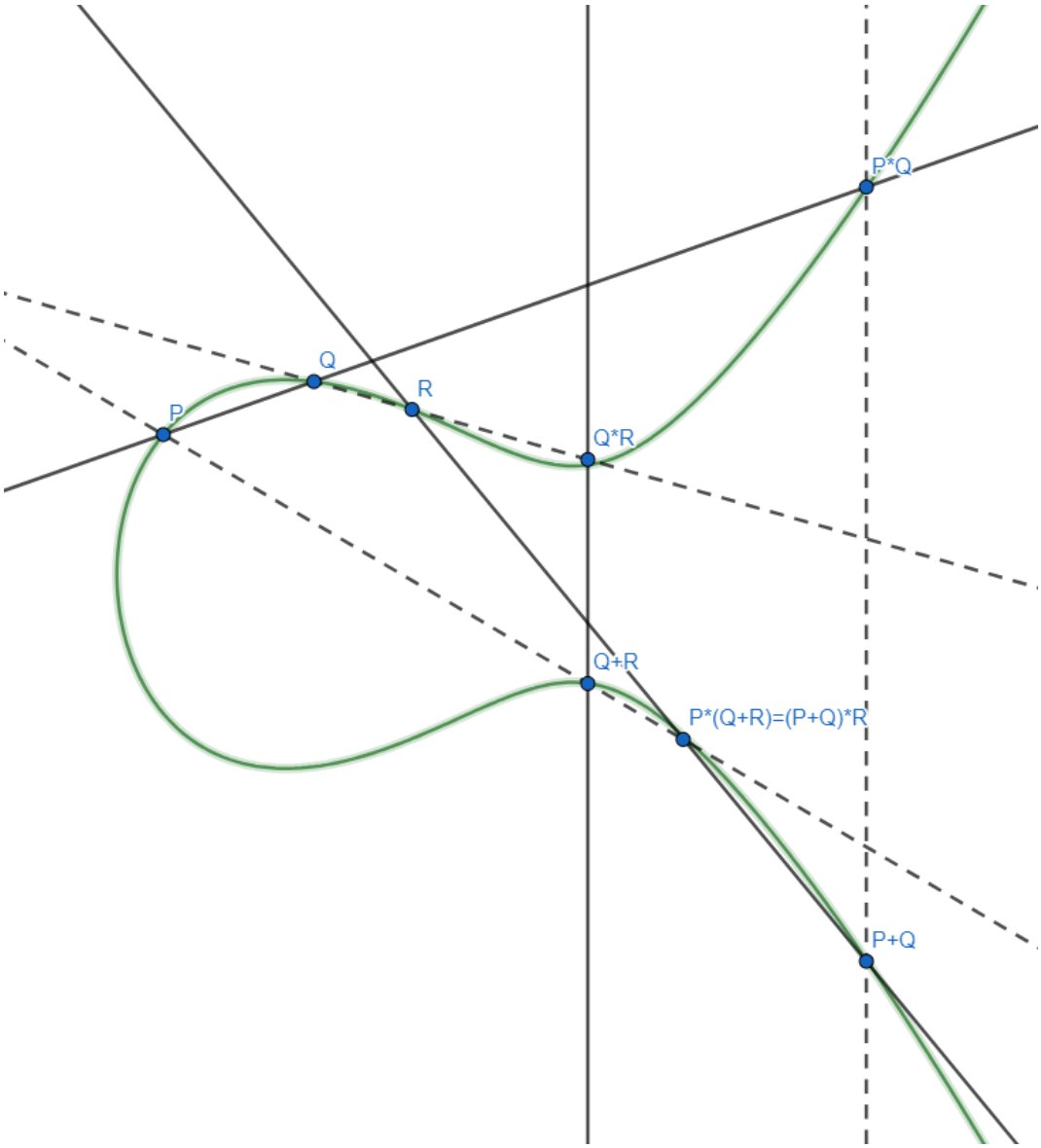

Figura B.1: Asociatividad de la suma del grupo de una curva elíptica.

Apéndice C

Implementación en C++ del método $p-1$ de Pollard

```cpp
#include <bits/stdc++.h>
#include <iostream>
#include <vector>
#include <chrono>
#include <numeric>

using namespace std;

//dobla-y-suma
long long modPot(long long x, unsigned int y, long long b)
{
    int res = 1; //resultado
    x = x % b;

    if (x == 0) return 0; // si x es divisible entre la base b;

    while (y > 0)
    {
        // si el último bit es 1, "y" es impar, entonces multiplica por el resultado
        if (y & 1) res = (res*x) % b;

        y = y>>1; //quitamos el último bit (si era 1 ya hemos añadido la potencia en el "if")
        x = (x*x) % b; //elevamos al cuadrado
    }
    return res;
}

long long pollard(long long n, long long a)
{
    //bucle hasta factor primo
    for(long i = 2 /* exponente */ ;; i++ /* incrementamos el exponente*/)
    {

        //potencia
        a =  ((long long) modPot(a,i,n));

        //gcd
        long long d = ((long long) gcd(a-1,n));

```

```
        // si es factor no trivial break
        if (d > 1)
        {
            //devolver el factor
            return d;
            break;
        }

        //caso en que tenemos múltiplo de n, por recursividad
        if (d == n)
        {
            return pollard(n, a ++);
            cout << "cambio de base ";
            break;
        }
    }
}


int main()
{
    auto start = chrono::high_resolution_clock::now();

    // N A FACTORIZAR, vamos a suponer que es producto de dos primos
    long long n = 2;

    // lista de factores
    vector<long> factor;

    long long f = pollard(n, 2);

    factor.push_back(f);
    long long r = (n/f);
    factor.push_back(r);

    cout << "Los factores primos de " << n << " son: ";

    for (int elem : factor){
        cout << elem << " ";
    }
    cout << endl;

    auto end = chrono::high_resolution_clock::now();

    /* cantidad de milisegundos como entero */
    auto duration = chrono::duration_cast<chrono::milliseconds>(end - start).count()
    ;

    cout << "Time taken by program is : " << duration << "ms " << endl;

    return 0;
}
```

Apéndice D

Implementación en C++ del método de Lenstra de factorización por curvas elípticas

```cpp
#include <bits/stdc++.h>
#include <iostream>
#include <vector>
#include <chrono>
#include <numeric>
#include <cmath>
using namespace std;

//este es el n a factorizar
static long long n =7*31;

long long gcdExtended(long long a, long long b, long long* x, long long* y)
{
    //caso base
    if (a == 0) {
        *x = 0, *y = 1;
        return b;
    }

    // guardar resultados recursivamente
    long long x1, y1;
    long long mcd = gcdExtended(b % a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b / a) * x1;
    *y = x1;

    return mcd;
}

//inverso en modulo
long long modInverse(long long A, long long M)
{
    long long x, y;
    if(A<0) A=A+M;
    long long g = gcdExtended(A, M, &x, &y);
```

```
38
39
40      //cout << A << " A " << M << " B " ;
41      if (g != 1){
42          cout << "factor encontrado" << g << endl;
43          throw g;
44      }
45      else {
46
47          // m se añade para lidiar con los casos negativos
48          long long res = (x % M + M) % M;
49
50          return res;
51      }
52  }
53
54  long long modPot(long long x, unsigned int y, long long b)
55  {
56      int res = 1;     // resultado
57
58      x = x % b; // chequear que sea mas pequeño
59
60      if (x == 0) return 0; // si x es divisible entre la base b;
61
62      while (y > 0)
63      {
64          // si el ultimo bit es 1, y es impar, entonces multiplica por el resultado
65          if (y & 1)
66              res = (res*x) % b;
67
68          y = y>>1; //quitamos el último bit (si era 1 ya hemos añadido la potencia en
        el if)
69          x = (x*x) % b;//elevamos al cuadrado
70      }
71      return res;
72  }
73
74
75  //suma de puntos
76  vector<long long> ECS(vector<long long> EC, vector<long long> P, vector<long long> Q
        )
77  {
78      //variables
79      long long m, yint;
80      long long x3, y3;
81
82      long long x1 = P.front();
83      long long y1 = P.back();
84      long long x2 = Q.front();
85      long long y2 = Q.back();
86
87      long long a = EC.front();
88      long long b = EC.back();
89
90      //inverso del grupo
91      long long invdif;
92      long long invig;
93
```

```
94      if(x2 == x1){
95          try{
96          invig = modInverse((2*y1),n); //lanza excepción
97          m = ((3*modPot(x1,2,n)+a)*invig)%n; //pendiente de la derivada
98          }
99          catch(long long g){
100         throw;
101         }
102
103     }
104     else{
105         try{
106         invdif = modInverse((x2-x1), n);  //lanza excepcion
107         m = (y2-y1)*invdif %n;
108         }
109         catch(long long g){
110         throw;
111         }
112     }
113
114     x3 =  (modPot(m,2,n)-x1-x2)%n;
115     y3 = y1-m*(x1-x3);
116
117     vector<long long> PmQ{x3,y3};
118
119     return PmQ;
120 }
121
122 vector<long long> crearEC(long long a, vector<long long> punto){
123
124     vector<long long> EC;
125
126     EC.push_back(a); //elegimos a
127
128     long long aux = (modPot(punto.back(),2,n) - modPot(punto.front(),3,n)-EC.front()
    *punto.front())%n;
129
130     EC.push_back(aux);
131
132     //puede ser que la curva no sea lisa, y tendriamos un factor
133     long long dis = (-16*(4*modPot(a,3,n)+27*modPot(aux,2,n)))%n;
134     try{
135     }
136     catch(long long g)
137     {
138         throw;
139         return EC;
140     }
141     return EC;
142 }
143
144
145 //dobla y suma
146 vector<long long> dobysum(vector<long long> EC, vector<long long> P, long k){
147
148     vector<long long> res;
149
150     while(k & 0){
```

```
151         try{
152             P = ECS(EC, P, P);
153 
154         }
155         catch(long long g)
156         {
157             throw;
158         }
159         k = k>>1;
160 
161     }
162 
163     res = P;
164 
165     try{
166         P = ECS(EC, P, P);
167     }
168     catch(long long g)
169     {
170         throw;
171     }
172 
173     k=k>>1;
174 
175     while(k>0){
176 
177         if (k & 1){
178             try{
179                 res = ECS(EC, res, P);
180             }
181             catch(long long g)
182             {
183                 throw;
184             }
185          }
186         k = k>>1;
187         try{
188             P = ECS(EC, P, P);
189         }
190         catch(long long g)
191         {
192             throw;
193         }
194 
195 
196     }
197     return res;
198 }
199 
200 
201 long long Lenstra(){
202 
203     long long factor;
204     vector<long long> P{1,2};
205     vector<long long> Paux{1,2};
206     long A = 1;
207     vector<long long> EC;
208 
```

```
209
210     while (true)
211     {
212         try{
213             EC = crearEC(A, P);
214             break;
215         }
216         catch(long long g)
217         {
218             if(g!=n)return g;
219             else Paux.back()++;
220         }
221     }
222
223
224     long k = 1;
225     while (k<sqrt(n))
226     {
227         try{
228         P=Paux;
229         P = dobysum(EC, P, k);
230         k++;
231         }
232         catch(long long g){
233             if(g!=n){
234                 factor=g;
235                 break;
236             }
237             else{
238                 Paux.back()++;
239                 while (true){
240                     try{
241                     EC = crearEC(A, Paux);
242                     break;
243                     }
244                     catch(long long g)
245                     {
246                         cout << "excepcion en crear curva" << endl;
247                         if(g!=n){
248
249                             return g;
250                         }
251                         else Paux.back()++;
252                     }
253                 }
254                 k=1;
255             }
256         }
257     }
258     return factor;
259 }
260
261
262 int main(){
263
264     cout << n << endl;
265
266     long long factor = Lenstra();
```

```
267     long long r = n/factor;
268     cout << r <<" y " << factor << " son factores de " << n;
269 
270     return 0;
271 }
```