UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Algoritmos deterministas de primalidad

Autor/es

ENRIQUE ESPADA CALVO

Director/es

JUAN LUIS VARONA MALUMBRES

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Matemáticas

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2020-21

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Matemáticas

Algoritmos deterministas de primalidad

Realizado por:

Enrique Espada Calvo

Tutelado por:

Juan Luis Varona Malumbres

Logroño, septiembre de 2021

Algoritmos deterministas de primalidad

Autor: Enrique Espada Calvo
Tutor: Juan Luis Varona Malumbres

Grado en Matemáticas
Facultad de Ciencia y Tecnología
Universidad de La Rioja

Curso académico: 2020/2021

Resumen/Abstract

Resumen

En este trabajo estudiaremos algunos de los principales tests deterministas de primalidad descubiertos hasta la fecha. Para ello, primero haremos una breve introducción para contextualizar al lector en el problema de la primalidad. Después, introduciremos algunos conceptos necesarios para entender el grueso del trabajo (conceptos que puede que el lector tenga muy claros y asimilados pero que conviene recordar en este punto), y a continuación nos centraremos en los tests.

Una vez llegados a ese punto, lo primero que nos va a llamar la atención es que muchos de estos tests son aplicables solo a números que presentan formas concretas, no a todo tipo de números, por eso dividiremos los algoritmos en dos grupos. En el primero de esos grupos encontraremos los algoritmos para números de un tipo concreto, como los tests de Lucas-Lehmer y Pépin, y en el segundo estarán los algoritmos aplicables a cualquier número entero, como el célebre AKS o el conocido test de Wilson. Hechas las presentaciones, empecemos.

Abstract

In this document we will study some of the main deterministic primality tests discovered to date. To do so, we will first make a short introduction to contextualize the reader in the primality problem. Then, we will introduce some concepts necessary to understand the bulk of the work (concepts that the reader may already have very clear and assimilated but that should be recalled at this point), and then we will focus on the tests.

Once we have reached that point, the first thing that will draw our attention is that many of these tests are applicable only to numbers that have specific shapes, not to all types of numbers, so we will divide the algorithms into two groups. In the first of these groups we will find the algorithms for numbers of a specific type, such as the Lucas-Lehmer and Pépin tests, and in the second we will find the algorithms applicable to any integer, such as the famous AKS or the well-known Wilson test. Once introductions have taken place, let's get started.

Índice general

Capítulo 1

Introducción

1.1. Números primos

Para poder explicar la importancia de la primalidad en las matemáticas (y en otros campos) debemos primero introducir la definición de número primo.

Definición 1.1 (Número primo). Dado un entero $p > 1$, se dice que p es primo si sus únicos divisores positivos son 1 y el propio p. Por otra parte, a los enteros positivos mayores que 1 que no son primos se les llama compuestos. Es importante recalcar que el número 1 NO es primo. Además, decimos que dos números a, $b \in \mathbb{Z} \setminus \{0\}$ son primos entre sí, o coprimos, si $\text{mcd}(a, b) = 1$.

Las primeras evidencias históricas que hay del conocimiento de los números primos se sitúan sobre el año 300 a.C. En esta época, el referente en esta materia fue Euclides (≈325 a. C.–≈265 a. C.). En su tratado, conocido como los *Elementos* de Euclides, recogió la definición de número primo, una demostración de la existencia de infinitos números primos, la definición de máximo común divisor y la de mínimo común múltiplo ... También incluyó el conocido como teorema fundamental de la aritmética, que veremos en este capítulo.

En este nivel, son muchos los resultados básicos que podríamos incluir pero que se salen del objetivo de nuestro estudio. Nosotros tan solo introduciremos algunos enunciados de gran relevancia que nos puedan ayudar a comprender un poco más el porqué de nuestro trabajo.

Teorema 1.1 (Euclides). Existen infinitos números primos.

El número de demostraciones de este teorema que uno puede encontrar es muy grande. Nosotros solo mostraremos una. Se pueden encontrar más en [17] y algunas más en las referencias de dicho artículo. Vamos con la demostración.

Demostración. Supongamos que solo existe una cantidad finita de números primos, y que todos esos primos son $p_1, p_2, \ldots, p_r$. Tomemos ahora

$$N = p_1 p_2 \cdots p_r + 1.$$

Este N no es ninguno de los p_i, puesto que es mayor que todos ellos, luego no puede ser primo. Como N debe tener algún divisor primo, supongamos que p primo divide a N. Este p no puede ser ninguno de los p_i, ya que, si lo fuera, como $p_i \mid N$ y $p_i \mid p_1 p_2 \cdots p_r$, entonces se tendría que $p_i \mid N - p_1 p_2 \cdots p_r = 1$, y esto es imposible. Por tanto, acabamos de encontrar un nuevo primo que no está en la lista que habíamos definido, y así alcanzamos la contradicción. □

El hecho de la existencia de infinitos números primos es el que motiva el presente estudio. Si hubiera un número finito de primos, dado un entero mayor que 1, para saber si ese número es primo bastaría con comprobar si ese número está en la lista finita de los posibles números primos, pero, como acabamos de ver, esto no ocurre.

Aquí entran en juego los tests o algoritmos deterministas de primalidad. Básicamente, el objetivo de estos tests es determinar si, dado un entero $n > 1$, n es primo o compuesto, haciendo el trabajo más eficiente posible, es decir, intentando reducir el tiempo de cálculo todo lo que se pueda. Para dejar claro el concepto vamos a hacer una definición más concreta.

1.2. Algoritmos deterministas de primalidad

Definición 1.2 (Algoritmo determinista de primalidad). Dado S un conjunto de enteros positivos, un algoritmo determinista de primalidad, o test determinista de primalidad, para enteros en S es un algoritmo que, dado un entero $n \in S$, concluye si n es primo o es compuesto.

Al introducir está definición debemos mostrar la definición de lo que, por contra, sería un algoritmo no determinista (o probabilístico) de primalidad.

Definición 1.3 (Algoritmo probabilístico de primalidad). Dado S un conjunto de enteros positivos, un algoritmo probabilístico de primalidad, o algoritmo no determinista de primalidad, o test de pseudoprimalidad, para enteros en S es un algoritmo que, dado un entero $n \in S$, la salida que produce es "n es compuesto" o "n es probablemente primo". Estos algoritmos sirven para producir los denominados **pseudoprimos**.

Los tests probabilísticos son muy rápidos, normalmente bastante más que los deterministas, pero, al contrario que los deterministas, no logran garantizar de manera exacta que el número en cuestión es, en efecto, primo.

La principal utilidad de estos tests es hacer un efecto de criba. Por un lado, nos permiten demostrar rápidamente que algunos números son compuestos sin necesitar ejecutar un test determinista que tardara mucho más en dar un resultado. Por otro lado, también sirven para que, si tenemos un número que ha pasado el test probabilístico, veamos que merece la pena pasarlo por un test determinista porque es bastante probable que sea primo.

Algunos de los algoritmos probabilísticos más conocidos son el test de Miller-Rabin o el test de Solovay-Strassen. Nosotros no vamos a profundizar en este tipo de tests. Se puede encontrar más información sobre esto en [18].

Para medir la eficiencia de un algoritmo (de cualquiera de los dos tipos) nos fijamos en el número de operaciones bit que realiza, lo que denotamos por $C(n)$. Un algoritmo tiene **complejidad polinomial** si existe un polinomio $P(x)$, con coeficientes reales, tal que para todo $n \in S$ se verifica que $C(n) \leq P(\log(n))$.

Una vez definidos estos conceptos vamos a hablar de algunos algoritmos deterministas.

El algoritmo determinista de primalidad más sencillo que uno puede imaginar es el de las **divisiones sucesivas**. Para comprobar si $n > 1$ es primo o compuesto dividimos n por todos los números primos menores o iguales que $\sqrt{n}$. Si en todas las divisiones a realizar el resto es distinto de 0, entonces el número será primo, y será compuesto en otro caso. Este algoritmo hace uso de una propiedad muy interesante y sencilla de demostrar que usaremos de forma recurrente en el trabajo y que vemos a continuación.

Proposición 1.1. Dado un entero $n > 1$ compuesto, existe p primo tal que $p \mid n$ con $p \leq \sqrt{n}$.

Demostración. Sea $n = p_1 \cdots p_r$ con $r \geq 2$ la descomposición en factores primos de n. Si $p_i > \sqrt{n}$ para todo $i \in \{1, \ldots, r\}$, entonces $p_i \cdot p_j > n$ con $1 \leq i < j \leq r$, lo cual es absurdo. □

El problema del algoritmo de las divisiones sucesivas es que es tremendamente ineficiente, y en cuanto escogemos un n grande se hace inabordable.

Los algoritmos modernos más eficaces no hacen uso de las divisiones sucesivas. Prácticamente todos los algoritmos deterministas de primalidad que se utilizan hoy en día surgieron del siglo XIX en adelante.

A pesar de esto, no queremos dejar de mencionar uno de los primeros algoritmos (si no el primero) que se descubrió, la **criba de Eratóstenes**. Este método debe su nombre a su autor, Eratóstenes (276 a. C.–194 a. C.), coetáneo de Euclides en la Antigua Grecia. Este no es un algoritmo en sí como nosotros lo hemos definido, ya que no determina si un cierto n es primo. Lo que hace es encontrar una lista con los primos situados entre 2 y n. Para hacer que cumpliera nuestra definición, simplemente haríamos la criba para $n - 1$ y después comprobaríamos si n la pasaría o no. La cuestión es que el coste de seguir este procedimiento es muy grande; por eso, a pesar de ser correcto, este algoritmo tiene un valor más histórico que práctico. Vamos a explicar cómo funciona.

En primer lugar, se apuntan todos los números naturales entre 2 y n y se tachan todos los pares menos el 2 (todos los múltiplos de 2 excepto el mismo). Después, se coge el primer número no tachado y se comprueba si su cuadrado es mayor que n. Si es mayor, habremos terminado, y, si no, se tachan todos los múltiplos del número no tachado en cuestión mayores o iguales que su cuadrado. Por último se repite este segundo paso hasta que se termine.

Este método tiene sentido debido a que, si estamos estudiando el número k, los múltiplos de k menores que k^2 ya habrán sido tachados, puesto que son múltiplos de números menores que k.

Finalmente, al terminar el proceso, los números que no han sido tachados serán los primos.

Centrándonos ya en los algoritmos más recientes, hay algoritmos que se pueden aplicar a todo tipo de números, y otros que solo se pueden aplicar a números con formas concretas. Esto suele ir ligado a la necesidad de conocer la factorización parcial del número anterior o del siguiente para determinar si el número estudiado es primo, o condiciones similares. Veremos estos algoritmos en los próximos capítulos.

Por último, no queremos terminar esta introducción sin informar al lector de algunas de las aplicaciones más importantes que tienen los números primos. La importancia de los números primos en las matemáticas es incuestionable. Prueba de ello son teoremas como el que sigue.

1.3. Teorema fundamental de la aritmética

Teorema 1.2 (Teorema fundamental de la aritmética). Todo entero mayor o igual que 2 se puede poner de forma única, salvo el orden, como producto de números primos.

Demostración. Primero vamos a probar que todo $n \geq 2$ se puede poner como producto de primos, y después probaremos la unicidad. Lo probamos por inducción. Para $n = 2$ es obvio, ya que 2 es primo. Supongamos cierta nuestra hipótesis para todo i mayor que 2 y menor o igual que $n-1$ y veamos qué ocurre para el caso n. Si n es primo habríamos terminado, así que veamos qué ocurriría si no lo fuera. Sea $n = dd'$ con $1 < d \leq d' < n$. Por hipótesis de inducción, d y d' son productos de primos al ser menores que n, luego $n = dd'$ también lo es.

Vamos con la unicidad. De nuevo procedemos por inducción. Al igual que antes, el caso $n = 2$ es evidente, así que suponemos la propiedad cierta para enteros menores que n de nuevo. Ahora, debemos probar que si tenemos dos descomposiciones en factores primos $n = p_1 \cdots p_k = q_1 \cdots q_r$ (no ponemos exponentes ya que los factores primos pueden estar repetidos), se debe cumplir que $k = r$ y que existe una permutación σ en $\{1, \ldots, k\}$ tal que $p_j = q_{\sigma(j)}$. Como p_k es primo y divide a $q_1 \cdots q_r$, sabemos que p_k debe dividir a algún q_t, esto es, existe un $t_0 \in \{1, \ldots, r\}$ tal que $p_k \mid q_{t_0}$. Como q_{t_0} es primo, necesariamente $p_k = q_{t_0}$. Sea entonces

$$\frac{n}{p_k} = p_1 \cdots p_{k-1} = \prod_{\substack{t=1 \\ t \neq t_0}}^{r} q_t < n.$$

Por hipótesis de inducción, $k - 1 = r - 1$, luego $k = r$ y existe una biyección $\sigma : \{1, \ldots, k-1\} \to \{1, \ldots, r\} \setminus \{t_0\}$ tal que $p_j = q_{\sigma(j)}$ para $j \in \{1, \ldots, k-1\}$. Si extendemos σ definiendo $\sigma(k) = t_0$ ya tenemos la permutación σ en $\{1, \ldots, k\}$ que buscábamos. □

Retomando nuestra reflexión sobre las utilidades de los primos, queremos dar respuesta a una pregunta que puede que más de un lector se haga, que es ¿pueden servirnos los números primos (muy grandes o no) para algo más que las matemáticas?

Durante mucho tiempo se pensaba que no, que la utilidad de los números primos era muy limitada fuera del ámbito de las matemáticas puras. Sin embargo, esto cambió durante el siglo XX, en concreto sobre la década de los 70, con el desarrollo de la criptografía de clave pública, en la que los números primos formaban la base de los primeros algoritmos, tales como el algoritmo RSA. No es un nuestro objetivo el estudio de la criptografía, pero vamos a dar unas pinceladas de cuál es el papel de los números primos en esta materia.

1.4. El criptosistema RSA

La criptografía asimétrica es un método criptográfico en el que se utilizan dos claves para el envío de mensajes, en lugar de una sola. Una clave es pública, y sirve para cifrar mensajes, y la otra es privada, y es la que sirve para descifrarlos. La base de este método reside en que a partir de la clave para cifrar no se pueda obtener la clave para descifrar. Si esto es así, aunque cualquier persona que conozca la clave pública pueda cifrar mensajes, solo quien posea la clave privada podrá descifrarlos.

Vamos a verlo con un clásico ejemplo. Imaginémonos que Alicia y Benito quieren tener una conversación cifrada. Supongamos que Alicia quiere mandar un mensaje cifrado a Benito. Para ello, en primer lugar, utiliza la clave pública de Benito para cifrarlo. Después se lo manda a Benito, y este, con su clave privada (que por supuesto solo él debe poseer), lo descifra. A estas alturas seguro que hay alguien que se estará preguntando ¿qué tiene esto que ver con los números primos? Vamos a verlo.

En 1978, Rivest, Shamir y Adleman elaboraron un criptosistema (el conocido como RSA) que cumplía estos requisitos. Su modelo consistía en tomar dos números primos distintos p y q suficientemente grandes y tomar $m = pq$. De esta forma, cifrar requiere hacer varias operaciones sencillas con congruencias módulo m, sin la necesidad de conocer los factores de m, y para descifrar mensajes sería necesario conocer[1] p y q. Uno puede pensar ¿pero si conocemos m, factorizándolo no podríamos obtener p y q fácilmente? La respuesta es que no, y es debido a que factorizar un número lo suficientemente grande es una tarea prácticamente inabordable con los métodos de factorización existentes hoy en día (de ahí la exigencia de tomar p y q de cierto tamaño).

Esta aplicación de los números primos en la criptografía hace que la seguridad de muchos sistemas de cifrado actuales dependan de la velocidad de los algoritmos de factorización existentes.

Tras esta contextualización ya estamos en disposición de entrar en materia.

[1] En concreto, es necesario conocer $\varphi(m)$, es decir, la indicatriz de Euler (véase la definición 2.1), pero la forma más rápida para obtener $\varphi(m)$ pasa por conocer p y q. No entraremos en más detalle. El lector puede buscar más información sobre esto en [18].

Capítulo 2

Nociones previas

En este tema introduciremos algunos resultados y definiciones que necesitaremos en secciones posteriores.

2.1. Aritmética modular

Dado un número natural $n \geq 1$, se dice que dos números $a, b \in \mathbb{Z}$ son congruentes módulo n si $a - b \in n\mathbb{Z}$, donde $n\mathbb{Z}$ denota al conjunto de los enteros múltiplos de n. Esto es,

$$n\mathbb{Z} = \{nk : k \in \mathbb{Z}\}.$$

Dicho de manera equivalente, a y b son congruentes módulo n si al dividir a y b entre n obtenemos el mismo resto en ambos casos. La notación que empleamos es la siguiente:

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

Es fácil ver que, dado un cierto n, la congruencia módulo n es una relación de equivalencia, ya que verifica las siguientes propiedades:

- Reflexiva: $a \equiv a \pmod{n}$
- Simétrica: $a \equiv b \pmod{n} \iff b \equiv a \pmod{n}$
- Transitiva: $a \equiv b \pmod{n}$ y $b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n}$

La demostración de estas propiedades es muy sencilla.

Dado $a \in \mathbb{Z}$, la clase de equivalencia de a, es decir, el conjunto de los enteros congruentes con a módulo n, viene dada por la expresión

$$\widehat{a} = \{x \in \mathbb{Z} : x \equiv a \pmod{n}\} = a + n\mathbb{Z}.$$

A esta clase de equivalencia la llamamos clase de restos de a módulo n. La denotamos como $\mathbb{Z}_n$. Las clases de restos $\widehat{0}, \widehat{1}, \ldots, \widehat{n-1}$ son disjuntas y su unión

es $\mathbb{Z}$, de modo que el cardinal de $\mathbb{Z}_n$ es n. $\mathbb{Z}_n$ tiene estructura aditiva y multiplicativa, ya que la hereda de $\mathbb{Z}$. Centrándonos en el producto, este se define de la siguiente forma en $\mathbb{Z}_n$:

$$\widehat{a} \cdot \widehat{b} = \widehat{a \cdot b}, \qquad \widehat{a}, \widehat{b} \in \mathbb{Z}_n.$$

Este producto está bien definido ya que no depende de los representantes (elementos) que tomemos de la clase, es decir,

$$a \equiv a_1 \pmod{n},\ b \equiv b_1 \pmod{n} \Rightarrow ab \equiv a_1 b_1 \pmod{n}.$$

Por lo tanto, $\widehat{a} = \widehat{a_1}, \widehat{b} = \widehat{b_1} \Rightarrow \widehat{a \cdot b} = \widehat{a_1 \cdot b_1}$. En $\mathbb{Z}_n$ siempre hay inverso respecto a la suma, pero no siempre lo hay respecto al producto[1]. Esto da lugar a que nos preguntemos cuántos elementos de $\mathbb{Z}_n$ tienen inverso multiplicativo. Denotaremos $\mathbb{Z}_n^*$ al conjunto de los elementos de $\mathbb{Z}_n$ que tienen inverso multiplicativo. En el caso en el que n es primo, se tiene que todos los elementos de $\mathbb{Z}_n$ excepto el 0 poseen inverso multiplicativo. Un resultado que nos ayudará a entender esto es el siguiente.

Proposición 2.1. Sea un entero $a \neq 0$ y otro entero $n \geq 2$ tal que $\mathrm{mcd}(a, n) = 1$. Entonces, dado $b \in \mathbb{Z}$, la ecuación $ax \equiv b \pmod{n}$ tiene una única solución módulo n.

Demostración. Los n números $a, 2a, 3a, \ldots, na$ son incongruentes módulo n, ya que si $ja \equiv ka \pmod{n}$, entonces $n \mid (j-k)a$, y como $\mathrm{mcd}(a, n) = 1$, se tendría que $n \mid (j-k)$ y esto no puede ocurrir salvo que $j = k$. De este modo, al variar x entre 1 y n, mediante ax se van obteniendo los n restos módulo n posibles, y, en particular, se obtendrá algún x que verifique $ax \equiv b \pmod{n}$.

La unicidad de la solución se deduce de la misma idea, ya que si hubiera dos x distintos que dieran el mismo resto módulo n no se obtendrían los n restos posibles. □

Así, a partir de este resultado, tomando $b = 1$ se deduce que los elementos $\widehat{a}$ con $\mathrm{mcd}(a, n) = 1$ poseen inverso multiplicativo en $\mathbb{Z}_n$.

A continuación, vamos a ver una herramienta que nos va a permitir conocer el cardinal de $\mathbb{Z}_n^*$, la función indicatriz de Euler.

Definición 2.1 (Función indicatriz de Euler). Dado un entero $n \geq 2$ (podríamos considerar el 1 también pero sería un caso trivial), se define la función indicatriz de Euler de la siguiente forma:

$$\varphi(n) = \mathrm{card}\{k \in \mathbb{Z} : 1 \leq k \leq n, \mathrm{mcd}(k, n) = 1\}.$$

Es decir, $\varphi(n)$ es el número de enteros positivos menores o iguales que n que son primos con n. Esta función representa el cardinal de $\mathbb{Z}_n^*$. Obsérvese que $\varphi(n) \leq n - 1$ para todo $n \geq 2$ y que dado un primo p, $\varphi(p) = p - 1$.

[1] Veremos más adelante que esto tiene su explicación en el hecho de que $\mathbb{Z}_n$ es un anillo conmutativo unitario y es un anillo de división conmutativo solo si n es primo.

A continuación, vamos a hacer uso de este concepto para enunciar y demostrar uno de los teoremas más importantes de la aritmética modular. La afirmación general se le atribuye a Leonhard Euler (1707–1783) y el caso particular correspondiente a $n = p$, con p un número primo, a Pierre de Fermat (1607–1665).

Teorema 2.1 (Teorema de Euler-Fermat). Dados dos enteros a y n tales que $\text{mcd}(a, n) = 1$, entonces $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Demostración. Para demostrarlo reflexionemos primero la siguiente idea. Tomemos un representante de cada una de las $\varphi(n)$ clases de $\mathbb{Z}_n^*$, lo cual se denomina sistema reducido de restos módulo n. Para ello, normalmente se cogen los números primos con n de entre la siguiente lista $\{1, 2, \ldots, n\}$. Nótese que si $\{b_1, \ldots, b_{\varphi(n)}\}$ es un sistema reducido de restos módulo n y $\text{mcd}(a, n) = 1$, entonces $\{ab_1, \ldots, ab_{\varphi(n)}\}$ también sería un sistema reducido de restos módulo n.

Volviendo a la demostración, si $\{b_1, \ldots, b_{\varphi(n)}\}$ es un sistema reducido de restos módulo n, entonces $\{ab_1, \ldots, ab_{\varphi(n)}\}$ también lo es. De modo que

$$b_1 \cdots b_{\varphi(n)} \equiv a^{\varphi(n)} b_1 \cdots b_{\varphi(n)} \pmod{n}.$$

Como $\text{mcd}(b_j, n) = 1$ para cada $j \in \{1, 2, \ldots, \varphi(n)\}$, el $\text{mcd}(b_1 \cdots b_{\varphi(n)}, n) = 1$. Así, $1 \equiv a^{\varphi(n)} \pmod{n}$. Esto es porque, como $b_1 \cdots b_{\varphi(n)}(1 - a^{\varphi(n)})$ es múltiplo de n y

$$\text{mcd}(b_1 \cdots b_{\varphi(n)}, n) = 1,$$

el factor $(1 - a^{\varphi(n)})$ debe ser necesariamente múltiplo de n. □

Teorema 2.2 (Pequeño teorema de Fermat). Dado un entero a y un número primo p, se tiene que:

a) Si $p \nmid a$, entonces $a^{p-1} \equiv 1 \pmod{p}$.

b) $a^p \equiv a \pmod{p}$ (en cualquier caso, incluso si $p \mid a$).

Demostración. Si $p \nmid a$, por el teorema de Euler-Fermat $a^{p-1} \equiv 1 \pmod{p}$, lo que prueba (a). Además, en este caso, $a^p = a^{p-1}a \equiv a \pmod{p}$.

Si $p \mid a$, a^p y a serían múltiplos de p, luego $a^p \equiv 0 \equiv a \pmod{p}$. □

Definición 2.2 (Orden de a módulo n). Dado un entero $n \geq 2$ fijo y otro entero a tales que $\text{mcd}(a, n) = 1$, definimos el orden de a módulo n como sigue:

$$\text{ord}_n(a) = \text{mín}\{k \geq 1 : a^k \equiv 1 \pmod{n}\}.$$

También puede usarse con la notación $o_n(a)$. Está bien definido por el teorema de Euler-Fermat y se cumple que $1 \leq \text{ord}_n(a) \leq \varphi(n)$.

A continuación vamos a ver algunas propiedades relacionadas con esta última definición.

Definición 2.3 (Raíz primitiva módulo n). Dado un entero $n \geq 2$ fijo y otro entero a tales que $\mathrm{mcd}(a, n) = 1$, decimos que a es una raíz primitiva módulo n si se cumple

$$\mathrm{ord}_n(a) = \varphi(n).$$

Vamos a ver ahora un teorema que da una importante caracterización de las raíces primitivas.

Teorema 2.3. Sea un entero $n \geq 2$ fijo y otro entero a tales que $\mathrm{mcd}(a, n) = 1$. Entonces, a es una raíz primitiva módulo n si y solo si el conjunto $\{a, a^2, \ldots, a^{\varphi(n)}\}$ forma un sistema reducido de restos módulo n.

Demostración. Supongamos que a es raíz primitiva módulo n. Los $\varphi(n)$ números $a, a^2, \ldots, a^{\varphi(n)}$ formarán un sistema reducido de restos módulo n si no contienen repetidos, es decir, si $a^j \not\equiv a^k \pmod{n}$ para todo j y k que verifiquen que $1 \leq j < k \leq \varphi(n)$. Ahora, como $\mathrm{mcd}(a, n) = 1$, se tiene que $\mathrm{mcd}(a^j, n) = 1$, y por tanto, por la proposición 2.1, sabemos que a^j tiene inverso en $\mathbb{Z}_n$. Si ocurriera que $a^j \equiv a^k \equiv a^{k-j}a^j \pmod{n}$, multiplicando esta equivalencia por el inverso de a^j resultaría que $1 \equiv a^{k-j} \pmod{n}$, y esto no puede ser, ya que n es raíz primitiva módulo n.

Recíprocamente, supongamos que $\{a, a^2, \ldots, a^{\varphi(n)}\}$ es un sistema reducido de restos módulo n. Esto implica que todos esos elementos son distintos módulo n, y como por definición $a^{\varphi(n)} \equiv 1 \pmod{n}$, no puede haber ningún exponente $k < \varphi(n)$ tal que $a^k \equiv 1 \pmod{n}$. □

El anterior teorema nos garantiza que, si a es una raíz primitiva módulo n, $\mathbb{Z}_n^* = \langle a \rangle$.

Una pregunta que nos puede surgir en este momento es si para todo valor de n se pueden encontrar raíces primitivas. La respuesta a esa pregunta la da el siguiente teorema, el cual no demostraremos. El lector puede consultar la demostración en [2].

Teorema 2.4. Dado n un entero positivo, existe alguna raíz primitiva módulo n si y sólo si

$$n = 1,\ 2,\ 4,\ p^k \text{ o } 2p^k,$$

donde p denota un número primo impar y $k \geq 1$.

Proposición 2.2. Sea un entero $n \geq 2$ fijo y otro entero a tales que $\mathrm{mcd}(a, n) = 1$, y sea t otro entero no negativo. Entonces, $a^t \equiv 1 \pmod{n}$ si y solo si $\mathrm{ord}_n(a)$ divide a t.

Demostración. Veamos en primer lugar la implicación inversa. Si $\mathrm{ord}_n(a) \mid t$, podemos poner $t = \mathrm{ord}_n(a)q$ con q un entero positivo, y así:

$$a^t \equiv (a^{\mathrm{ord}_n(a)})^q \equiv 1^q \equiv 1 \pmod{n}.$$

Para la otra implicación, supongamos que $\mathrm{ord}_n(a) \nmid t$ y busquemos una contradicción. Entonces, $t = \mathrm{ord}_n(a)q + r$ para un cierto r que cumple que $1 \leq r < \mathrm{ord}_n(a)$. Pero, de aquí, se tiene que

$$1 \equiv a^t = (a^{\mathrm{ord}_n(a)})^q \cdot a^r \equiv 1^q \cdot a^r = a^r \pmod{n},$$

y esto es absurdo, ya que $\mathrm{ord}_n(a)$ es el menor entero positivo k que satisface que $a^k \equiv 1 \pmod{n}$. □

Lema 2.1. Sean $n \geq 2$ un entero fijo y $a \in \mathbb{Z}$ tales que $\mathrm{mcd}(a, n) = 1$. Supongamos que para ciertos enteros positivos d y k y un cierto entero no negativo r se cumple que $\mathrm{ord}_n(a^{d^r}) = d^k$. Entonces, $\mathrm{ord}_n(a) = d^{k+r}$. En particular, si $a^{2^r} \equiv -1 \pmod{n}$ y $n \geq 3$, entonces $\mathrm{ord}_n(a) = 2^{r+1}$.

Demostración. En primer lugar, cabe destacar que como $\mathrm{mcd}(a, n) = 1$, también $\mathrm{mcd}(a^{d^r}, n) = 1$, por lo que tiene sentido hablar del orden de a^{d^r} módulo n.

Ahora, sabemos que se cumple que

$$a^{d^{k+r}} = \left(a^{d^r}\right)^{d^k} \equiv 1 \pmod{n}, \tag{2.1}$$

por lo que $\mathrm{ord}_n(a) \leq d^{k+r}$. Supongamos que no se da la igualdad y busquemos una contradicción. Llamaremos $s = \mathrm{ord}_n(a) < d^{k+r}$. Como se cumple (2.1), por la proposición 2.2 se tiene que cumplir que $s \mid d^{k+r}$. Así, las descomposiciones en factores primos de d y s serán de la forma

$$\begin{aligned} d &= p_1^{b_1} p_2^{b_2} \cdots p_h^{b_h}, \qquad b_j \geq 1, \\ s &= p_1^{c_1} p_2^{c_2} \cdots p_h^{c_h}, \qquad 0 \leq c_j \leq (k+r)b_j, \end{aligned}$$

con algún $c_j < (k+r)b_j$ para que $s < d^{k+r}$. Si tomamos

$$s' = p_1^{\mathrm{máx}\{c_1, rb_1\}} p_2^{\mathrm{máx}\{c_2, rb_2\}} \cdots p_h^{\mathrm{máx}\{c_h, rb_h\}} = \mathrm{mcm}(s, d^r),$$

este s' cumplirá $s' < d^{k+r}$, $s \mid s'$, $d^r \mid s'$ y $s' \mid d^{k+r}$. Así pues, podemos escribir $s' = d^r u$ donde $1 \leq u < d^k$. Como $a^s \equiv 1 \pmod{n}$ y s' es múltiplo de s se tiene que

$$a^{s'} = a^{d^r u} \equiv 1 \pmod{n}.$$

En este punto, tenemos que

$$\begin{aligned} 1 \equiv \left(a^{d^r}\right)^{d^k} &= a^{d^{k+r} - d^r u + d^r u} = a^{(d^k - u)d^r} a^{d^r u} \\ &\equiv \left(a^{d^r}\right)^{d^k - u} \cdot 1 \equiv \left(a^{d^r}\right)^{d^k - u} \pmod{n}, \end{aligned}$$

lo que nos lleva a una contradicción, ya que $1 \leq d^k - u < d^k = \mathrm{ord}_n(a^{d^r})$.

Para probar el caso particular basta con tomar $d = 2$ y observar que $(a^{2^r})^2 \equiv (-1)^2 \equiv 1 \pmod{n}$, es decir, que el orden de a^{2^r} es 2. (Nótese que la condición de que $n \geq 3$ es importante, porque $-1 \equiv 1 \pmod{2}$.) □

Definición 2.4 (Símbolo de Legendre). Sea p un número primo impar y sea b un entero tal que $b \not\equiv 0 \pmod{p}$, definimos el símbolo de Legendre como

$$(b|p) = \begin{cases} 1, & \text{si existe } x \in \mathbb{Z} \text{ tal que } x^2 \equiv b \pmod{p}, \\ -1, & \text{si no ocurre lo anterior.} \end{cases} \tag{2.2}$$

Si $b \equiv 0 \pmod{p}$, se toma $(b|p) = 0$. Esta función nos va a permitir saber cuándo b es resto cuadrático módulo p ($(b|p) = 1$) y cuándo no lo es ($(b|p) = -1$).

A continuación, vamos a ver una extensión del símbolo de Legendre, que nos será útil en algunos casos.

Definición 2.5 (Símbolo de Jacobi). Sea b un entero y sea P un entero impar (mayor que 1) cuya descomposición en factores primos es la siguiente:

$$P = \prod_{j=1}^{s} p_j^{a_j}.$$

Llamamos símbolo de Jacobi a la expresión

$$(b|P) = \prod_{j=1}^{s} (b|p_j)^{a_j},$$

donde $(b|p_j)$ corresponde con el símbolo de Legendre. Si $P = 1$ se toma $(b|P) = 1$. Es obvio que esta función solo puede tomar los valores 1, -1 o 0 (este último si y solo si $\text{mcd}(b, P) > 1$). Nótese que si P es primo, el símbolo de Jacobi coincide con el de Legendre. Algunas de las propiedades interesantes de esta función (las cuales se prueban de forma muy sencilla) son:

- $(b|P) = (c|P)$ si $b \equiv c \pmod{P}$,
- $(bc|P) = (b|P)(c|P)$,
- $(b|PQ) = (b|P)(b|Q)$,
- $(b^2|P) = 1$ si $\text{mcd}(b, P) = 1$.

Otras fórmulas interesantes relacionadas con el símbolo de Jacobi que nos serán de ayuda son:

- $(-1|P) = (-1)^{(P-1)/2}$,
- $(2|P) = (-1)^{(P^2-1)/8}$.

Teorema 2.5 (Criterio de Euler). Dados p un número primo impar y b un entero, entonces:

$$(b|p) \equiv b^{(p-1)/2} \pmod{p}. \tag{2.3}$$

Notése que $b^{(p-1)/2}$ es congruente con 1, -1 o 0 siempre, en estas condiciones. Podemos comprobarlo con el pequeño teorema de Fermat:

$$(b^{(p-1)/2} + 1)(b^{(p-1)/2} - 1) = b^{(p-1)} - 1 \equiv 0 \pmod{p}. \tag{2.4}$$

Como vemos, al ser p primo, p tiene que ser divisor de $(b^{(p-1)/2} + 1)$ o de $(b^{(p-1)/2} - 1)$, y de ahí que la expresión $b^{(p-1)/2}$ deba ser congruente con 1 o -1. (Si b es múltiplo de p será congruente con 0.)

Teorema 2.6 (Ley de Reciprocidad Cuadrática). Sean p y q números primos impares y distintos. Entonces:

$$(p|q)(q|p) = (-1)^{(p-1)(q-1)/4}. \tag{2.5}$$

2.2. Estructuras algebraicas

Para nuestro trabajo también nos será necesario introducir algunos resultados de teoría de grupos y de anillos. Definiremos las nociones de grupo, anillo, anillo de polinomios, cuerpo... , así como ciertas propiedades relevantes.

Definición 2.6 (Grupo). Un grupo es un conjunto G dotado de una operación binaria interna

$$\begin{aligned} G \times G &\to G \\ (x, y) &\mapsto xy, \end{aligned}$$

la cual se suele llamar producto (aunque esto puede variar según el contexto) y se suele indicar como $x \cdot y$. Además, se debe verificar la siguiente lista de propiedades:

- Asociatividad. Para todo x, y, $z \in G$ se tiene que $(xy)z = x(yz)$.
- Elemento neutro. Existe un $e \in G$ tal que para todo $x \in G$ se tiene que $xe = ex = x$.
- Elemento inverso. Para todo $x \in G$ existe un $y \in G$ tal que $xy = yx = e$. Se suele denotar $y = x^{-1}$.

Con estas propiedades, se dice que $(G, \cdot)$ es un grupo, pero si además el producto es conmutativo, es decir, para todo x, $y \in G$ se cumple que $xy = yx$, entonces se dice que $(G, \cdot)$ es un grupo abeliano.

Algunos ejemplos de grupos son:

- $(\mathbb{Z}, +)$. Como vemos, en este caso la operación interna sería la suma, no el producto. En este grupo el elemento neutro es el 0. Además es abeliano.
- $V = \{e, a, b, c\}$. El grupo de Klein. Es un grupo abeliano de 4 elementos. El neutro es el e. La siguiente lista de productos lo define: $a^2 = b^2 = c^2 = e$, $ab = c$, $ac = b$ y $bc = a$.
- $(\mathbb{Q} - \{0\}, \cdot)$. Otro ejemplo de grupo abeliano. El elemento neutro sería el 1.

Definición 2.7 (Subgrupo). Sea G un grupo y sea H un subconjunto no vacío de G. Se dice que H es un subgrupo de G y se denota como $H \leq G$ si se cumplen las siguientes condiciones:

- Para todo x, $y \in H$ se tiene que $xy \in H$.
- $e \in H$ (el elemento neutro de G está en H).
- Para todo $x \in H$ se tiene que $x^{-1} \in H$.

Hay dos tipos de subgrupos, los triviales, que son el $\{e\}$ y G, y los propios, que son todos los restantes. Es obvio que todos los grupos tienen los dos subgrupos triviales.

Definición 2.8 (Orden de un grupo). El orden de un grupo es el número de elementos de dicho grupo, es decir, su cardinal.

Definición 2.9 (Orden del elemento de un grupo). Sea G un grupo y $g \in G$ un elemento de G. Hay 2 opciones:

- $g^n \neq e$ para todo $n \geq 1$ con $n \in \mathbb{Z}$. En este caso todas las potencias de g son distintas, ya que si $g^n = g^m$ con $n > m$, entonces $g^{n-m} = e$. Así pues, decimos que el orden de g es infinito.
- Existe $r = \text{mín}\{n \in \mathbb{N} : g^n = e\}$. Por lo mismo que antes, todas las potencias de g desde 0 hasta $r-1$ son distintas y $g^{n+r} = g^n$ para todo $n \in \mathbb{Z}$. En este caso se dice que el orden de g es r. Nótese que $g^m = e$ si y solo si $r \mid m$.

Teorema 2.7 (Teorema de Lagrange). El orden de un grupo finito es múltiplo del orden de cualquiera de sus subgrupos.

Corolario 2.1. En un grupo finito, el orden de cualquier elemento divide al orden del grupo.

Definición 2.10 (Anillo). Un anillo es un conjunto R dotado de dos operaciones binarias internas

$$\begin{aligned} R \times R &\to R & R \times R &\to R \\ (a,b) &\mapsto a+b & (a,b) &\mapsto ab \end{aligned}$$

(la primera la suma $(+)$ y la segunda el producto $(\cdot)$) que deben cumplir las siguientes propiedades:

- $(R,+)$ es un grupo abeliano con elemento neutro 0 y elemento inverso de un cierto a el $-a$.
- La multiplicación es asociativa, esto es, para todo a, b, $c \in R$ se tiene que $(ab)c = a(bc)$.
- La suma es distributiva respecto al producto, esto es, para todo a, b, $c \in R$ se tiene que $a(b+c) = ab + ac$ y $(a+b)c = ac + bc$.

A continuación vamos a ver algunas definiciones y propiedades más de los anillos, que nos serán de utilidad.

Un anillo es **conmutativo** si su producto es conmutativo, esto es, para todo a, $b \in R$ se tiene que $ab = ba$.

Un anillo es **unitario** si existe un elemento neutro para el producto, es decir, $1 \in R$. Nótese que consideraremos a lo largo de todo el trabajo que $0 \neq 1$. El 1 verifica que, para todo $a \in R$, $a \cdot 1 = 1 \cdot a = a$. Es claro que el menor anillo unitario que existe es el $R = \{0, 1\}$.

Un anillo de **división** es un anillo unitario en el que todo elemento no nulo de R posee inverso multiplicativo. Esto es, que, para todo elemento $a \in R$ con $a \neq 0$, existe un $b \in R$, con $b \neq 0$, tal que $a \cdot b = b \cdot a = 1$. Se denota $b = a^{-1}$.

Dado un anillo R, se dice que $a \in R$ es un **divisor de cero** si $a \neq 0$ y existe $b \in R$ con $b \neq 0$ tal que $a \cdot b = 0$ o $b \cdot a = 0$.

Dado un anillo R se dice que $a \in R$ con $a \neq 0$ es **invertible** si R es unitario y existe $b \in R$ con $b \neq 0$ tal que $a \cdot b = b \cdot a = 1$. Se denota R^* al conjunto de los elementos invertibles de R. (Nótese que esta notación ya se ha empleado previamente en el presente documento, cuando definimos $\mathbb{Z}_n$.)

Así, un **dominio de integridad** es un anillo conmutativo unitario sin divisores de cero.

Sea R un dominio de integridad; dado $r \in R$ con $r \neq 0$ y r no invertible, se dice que r es **irreducible** si, para todo a, $b \in R$ tales que $r = a \cdot b$, se tiene que o a es invertible o b es invertible.

Por último, sea R un dominio de integridad, dado $p \in R$ con $p \neq 0$ y p no invertible, se dice que p es **primo** si, para todo a, $b \in R$ tales que $p \mid a \cdot b$, se tiene que o bien $p \mid a$ o bien $p \mid b$.

Con estas definiciones, todo elemento primo es irreducible, pero el recíproco no siempre es cierto.

Observación 2.1. No debemos confundir la noción de primo que presentamos aquí con la expuesta en la definición 1.1. No entraremos en más detalles porque en este razonamiento están involucradas otras estructuras algebraicas que no mencionaremos. Como aclaración, solo diremos que $\mathbb{Z}$ es un dominio de ideales principales, y en estas estructuras todo elemento irreducible es primo. La definición 1.1 coincide aquí con la de elemento irreducible en $\mathbb{Z}$, pero como coméntabamos, en $\mathbb{Z}$ es equivalente hablar de elementos irreducibles y primos. Pueden encontrarse más detalles sobre estos hechos en [8].

Definición 2.11 (Cuerpo). Un **cuerpo** es un anillo de división conmutativo. Un ejemplo de cuerpo sería $\mathbb{Q}$. Obsérvese que un cuerpo es un anillo R que cumple que $(R \setminus \{0\}, \cdot)$ es un grupo abeliano.

Definición 2.12 (Subcuerpo). Dado un cuerpo K y dado F un subconjunto de K, se dice que F es un subcuerpo de K si se cumple que:

- $(F, +)$ es subgrupo de $(K, +)$.
- $(F \setminus \{0\}, \cdot)$ es subgrupo de $(K \setminus \{0\}, \cdot)$.

Siguiendo una idea similar a la de los subgrupos, los subcuerpos triviales de un cuerpo K serían el $\{0, 1\}$ y el propio K, y los subcuerpos propios serían los restantes.

Definición 2.13 (Anillo de polinomios). Sea R un anillo conmutativo unitario, se define el anillo de polinomios sobre R en la indeterminada x como

$$R[x] = \{a_0 + a_1 x + \cdots + a_n x^n : a_0, \ldots, a_n \in R\}.$$

$R[x]$ tiene también estructura de anillo conmutativo unitario y, además, R es dominio de integridad si y solo si $R[x]$ es dominio de integridad.

Otra observación interesante que debemos hacer es que $(R[x])^* = R^*$.

Definición 2.14 (Polinomio irreducible). Sea R un anillo conmutativo unitario; dado un polinomio $p(x) \in R[x]$ con $\deg(p(x)) \geq 1$, se dice que $p(x)$ es irreducible en $R[x]$ si no podemos poner $p(x)$ como producto de dos polinomios $f(x)$, $g(x) \in R[x]$ de grado mayor igual que 0 tales que $f(x)$ y $g(x)$ no sean invertibles.

Veamos algún ejemplo de polinomios irreducibles y no irreducibles:

- $2x+2$ no es irreducible en $\mathbb{Z}[x]$, ya que $2x+2 = 2(x+1)$ y 2 no es invertible en $\mathbb{Z}[x]$ ni tampoco lo es $(x+1)$. Sin embargo, $2x+2$ sí que es irreducible en $\mathbb{Q}[x]$.
- $2x^4 + 18x^3 + 12x + 6$ es irreducible en $\mathbb{Q}[x]$ (criterio de Eisenstein[2] con $p = 3$), pero no es irreducible en $\mathbb{Z}[x]$ ya que $2x^4 + 18x^3 + 12x + 6 = 2(x^4 + 9x^3 + 6x + 3)$.
- $x^2 + 1$ es irreducible en $\mathbb{R}[x]$.

Finalmente, denotaremos F_p al cuerpo finito de p elementos, donde p es primo, y utilizaremos en algunos puntos el siguiente resultado, que no demostraremos.

Proposición 2.3. Sea $h(x) \in F_p[x]$ un polinomio de grado d e irreducible en $F_p[x]$. Entonces, $F_p[x]/(h(x))$ (el anillo cociente de $F_p[x]$ entre el ideal $(h(x))$) es un cuerpo finito de p^d elementos.

Es importante mencionar que, para entender el siguiente enunciado, conviene conocer ciertas propiedades de los polinomios ciclotómicos. Nosotros las exponemos en la siguiente sección. Por este motivo, recomendamos al lector que haga una lectura de ese apartado antes de terminar este, si así lo precisa.

Proposición 2.4. Sea F_p el cuerpo finito de p elementos, r un entero positivo tal que $\mathrm{mcd}(r,p) = 1$, $\Phi_r(x)$ el r-ésimo polinomio ciclotómico sobre F_p y d el menor entero positivo que verifica que $p^d \equiv 1 \pmod{r}$, es decir, $d = \mathrm{ord}_r(p)$. Entonces, $\Phi_r(x)$ se factoriza en $F_p[x]$ como producto de $\frac{\varphi(r)}{d}$ factores irreducibles de grado d.

Demostración. Sea $f(x)$ un factor irreducible cualquiera de $\Phi_r(x)$, y sea ξ una raíz r-ésima primitiva de la unidad (esto es, $\xi^r = 1$) sobre F_p que sea raíz de $f(x)$. Entonces

$$\xi \in F_{p^k} \Leftrightarrow \xi^{p^k} = \xi \Leftrightarrow \xi^{p^k-1} = 1 \Leftrightarrow p^k \equiv 1 \pmod{r},$$

de donde se obtiene que $\xi \in F_{p^d}$ y además ξ no pertenece a ningún subcuerpo propio de F_{p^d}, ya que d es el menor entero positivo que cumple $p^d \equiv 1 \pmod{r}$. Por esto, se cumple que el grado del polinomio irreducible de ξ sobre F_p es igual al grado de $f(x)$, y por tanto $\deg(f(x)) = d$. Ahora, como $\deg(\Phi_r(x)) = \varphi(r)$, se tiene que el número de factores irreducibles de $\Phi_r(x)$ es $\frac{\varphi(r)}{d}$. □

[2] Consultar [8] para más información.

2.3. Polinomios ciclotómicos

Definición 2.15 (Polinomio ciclotómico de grado n). Sea n un entero mayor o igual que 1. Llamamos polinomio ciclotómico de grado n, y lo denotamos como Φ_n, al polinomio mónico cuyas raíces son todas las raíces de orden n de la unidad, es decir, las n raíces que hay en $\mathbb{C}$ del polinomio $z^n - 1$. Cabe destacar que, aunque sus raíces sean números complejos, los coeficientes de los polinomios ciclotómicos son siempre enteros, es decir, $\Phi_n(x) \in \mathbb{Z}[x]$. Otra propiedad interesante que cumplen es que son irreducibles en $\mathbb{Z}[x]$ y en $\mathbb{Q}[x]$ para todo $n \geq 1$. Presentan la forma

$$\Phi_n(x) = \prod_{k=0}^{n-1} (x - \omega^k),$$

donde $\omega = e^{\frac{2\pi i}{n}}$. Este ω se obtiene despejando la z en la ecuación $z^n = 1$, poniendo el 1 en su forma compleja.

Observación 2.2. El polinomio $x^n - 1$ tiene por raíces a todas la raíces n-ésimas de la unidad, y toda raíz n-ésima de la unidad es raíz d-ésima de la unidad para algún divisor d de n. De la misma forma, las raíces de $\Phi_d(x)$ con d un divisor de n son raíces también de $x^n - 1$, luego se tiene que

$$x^n - 1 = \prod_{d|n} \Phi_d(x).$$

Esto nos va a permitir calcular los polinomios ciclotómicos de una forma recursiva, ya que

$$\Phi_n(x) = \frac{x^n - 1}{\prod\limits_{\substack{d|n \\ d<n}} \Phi_d(x)}.$$

Es fácil ver que, dado un primo p, $\Phi_p(x) = x^{p-1} + x^{p-2} + \cdots + x + 1$. Veamos los primeros polinomios ciclotómicos:

$$\begin{aligned}
\Phi_1(x) &= x - 1, \\
\Phi_2(x) &= x + 1, \\
\Phi_3(x) &= x^2 + x + 1, \\
\Phi_4(x) &= \frac{x^4 - 1}{\Phi_1\Phi_2} = \frac{x^4 - 1}{x^2 - 1} = x^2 + 1, \\
\Phi_5(x) &= x^4 + x^3 + x^2 + x + 1, \\
\Phi_6(x) &= \frac{x^6 - 1}{\Phi_1\Phi_2\Phi_3} = \frac{x^6 - 1}{(x^3 - 1)(x + 1)} = x^2 - x + 1, \\
\Phi_7(x) &= x^6 + x^5 + x^4 + x^3 + x^2 + x + 1.
\end{aligned}$$

2.4. Fracciones continuas

Una fracción continua es una expresión de la forma

$$q_0 + \cfrac{p_1}{q_1 + \cfrac{p_2}{q_2 + \cfrac{p_3}{q_3 + \cdots}}}$$

finita o infinita, donde q_k, $p_k \in \mathbb{R}$ con $q_k \neq 0$ para todo $k \geq 1$. Nosotros nos centraremos en un tipo concreto de fracciones continuas, las simples. En las **fracciones continuas simples** $p_k = 1$ y $q_k > 0$ para todo $k \geq 1$. Utilizaremos la notación

$$\langle q_0, q_1, q_2, q_3, \ldots \rangle = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots}}},$$

y llamaremos cociente incompleto k-ésimo al número q_k y **convergente k-ésimo** al número

$$\langle q_0, q_1, \ldots, q_k \rangle = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_k}}}}.$$

Además, en las fracciones continuas simples con las que vamos a trabajar vamos a imponer alguna condición más. Estas son que $q_0 \in \mathbb{Z}$ y $q_k \in \mathbb{N}$ para todo $k \geq 1$.

En estas condiciones es fácil probar que toda fracción continua simple finita $\langle q_0, q_1, \ldots, q_k \rangle$ representa un número racional, pero además, de manera recíproca, todo número racional se puede representar mediante una fracción continua simple finita (basta utilizar el algoritmo de Euclides para la división).

Uno de los inconvenientes que surgen con las fracciones continuas simples es que calcular $\langle q_0, q_1, \ldots, q_{k+1} \rangle$ a partir de $\langle q_0, q_1, \ldots, q_k \rangle$ no es posible; sería necesario volver a evaluar todas las fracciones. Por este motivo, vamos a definir un par de sucesiones recurrentes que nos permitirán solucionar este problema:

$$\begin{array}{llll} P_{-1} = 1, & P_0 = q_0, & P_1 = q_0 q_1 + 1, & P_k = q_k P_{k-1} + P_{k-2},\ k \geq 1, \\ Q_{-1} = 0, & Q_0 = 1, & Q_1 = q_1, & Q_k = q_k Q_{k-1} + Q_{k-2},\ k \geq 1. \end{array}$$

Teorema 2.8. Sea $\langle q_0, q_1, q_2, \ldots \rangle$ una fracción continua simple (finita o infinita). Entonces su convergente n-ésimo es

$$\langle q_0, q_1, q_2, \ldots, q_n \rangle = \frac{P_n}{Q_n},$$

donde los P_n y Q_n vienen dados por las recurrencias anteriores. Además, $\text{mcd}(P_n, Q_n) = 1$, luego el cociente P_n/Q_n siempre proporciona una fracción irreducible.

La demostración de este teorema, junto con más información sobre las fracciones continuas, se pueden encontrar en [18].

Después de haber explicado (o recordado) estos conceptos, ya podemos centrarnos en nuestro objetivo principal, los tests deterministas de primalidad.

Capítulo 3

Algoritmos para números con formas concretas

Como ya anticipábamos, en este capítulo expondremos algunos de los principales algoritmos deterministas de primalidad para números con formas concretas. Además, mostraremos algunos ejemplos en los que aplicaremos algunos de los tests, tanto en este capítulo como en el siguiente. Los ejemplos que veremos aquí involucrarán números pequeños para poder mostrar adecuadamente como aplicamos cada test. Es obvio que trabajando con números tan pequeños una factorización del número en cuestión sería, en la mayoría de los casos, la forma más rápida para saber si el número es primo o no; pero como decíamos nuestro objetivo es describir minuciosamente como funcionan los tests.

En el apéndice A se mostrarán ejemplos con números más grandes, y ahí se podrá apreciar la utilidad práctica real de los tests.

3.1. Test de Pépin

Los **números de Fermat** son aquellos que tienen la siguiente forma:

$$F_n = 2^{2^n} + 1, \qquad n \geq 0. \tag{3.1}$$

Los primeros números de Fermat son

$$F_0 = 3, \qquad F_1 = 5, \qquad F_2 = 17, \qquad F_3 = 257, \qquad F_4 = 65537,$$

y los 5 son primos. En 1640, Fermat conjeturó (pero no demostró) que todos los números de Fermat eran primos. Más tarde, en 1732, Euler demostró que se equivocaba, ya que para $n = 5$ el número de Fermat obtenido no es primo. Lo demostró encontrando la siguiente descomposición de F_5:

$$F_5 = 4294967297 = 641 \cdot 6700417.$$

Es más, hasta la fecha no se conoce ningún F_n con $n > 4$ que sea primo.

Como podrá observar el lector, los números de Fermat no son precisamente pequeños. Conforme aumentamos un poco la n obtenemos números con una cantidad de dígitos muy alta, lo que hace que los algoritmos genéricos para probar la primalidad de números arbitrarios, que pudiéramos utilizar en este contexto, dejen de sernos útiles. Sin embargo, el test que da nombre a esta sección, y que presentamos a continuación, es aplicable a números de Fermat bastante grandes (no entraremos a analizar con rigor el tamaño, pues este depende de las herramientas informáticas a nuestra disposición en cada momento temporal). El test fue propuesto por el matemático y sacerdote jesuita francés Théophile Pépin (1826–1904), en 1877.

Teorema 3.1 (Test de Pépin). Sea $F_n = 2^{2^n} + 1$ con $n \geq 1$ el n-ésimo número de Fermat. El número F_n es primo si y solo si

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n}. \tag{3.2}$$

Demostración. En primer lugar vamos a suponer que F_n es primo, y veamos que se cumple la congruencia dada por (3.2). Por la ley de reciprocidad cuadrática tenemos que

$$(3|F_n)(F_n|3) = (-1)^{(3-1)(F_n-1)/4} = 1.$$

Así, $(3|F_n) = (F_n|3)$. Como además $2 \equiv -1 \pmod 3$, tenemos que

$$F_n \equiv 2^{2^n} + 1 \equiv (-1)^{2^n} + 1 \equiv 2 \equiv -1 \pmod 3.$$

Por tanto, $(F_n|3) = (-1|3) = -1$. Finalmente, utilizando el criterio de Euler, se obtiene la expresión buscada:

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

Veamos ahora la implicación inversa. Supongamos que se cumple (3.2) y veamos que F_n es primo. Dado q un divisor primo de F_n, veamos que q tiene que ser necesariamente F_n. Tenemos lo siguiente:

$$(3^{(F_n-1)/2} + 1)(3^{(F_n-1)/2} - 1) = 3^{(F_n-1)} - 1. \tag{3.3}$$

Como se tiene (3.2), F_n divide a $3^{(F_n-1)/2} + 1$ y por tanto, por (3.3), también divide a $3^{(F_n-1)} - 1$. Así,

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

Ahora, como $q \mid F_n$ (q divide a F_n, no confundir con el símbolo de Legendre) tenemos que:

$$3^{(F_n-1)} \equiv 1 \pmod q. \tag{3.4}$$

Por tanto, por el lema 2.1 se tiene que $\mathrm{ord}_q(3) = F_n - 1$, de lo que se deduce que $F_n - 1 \mid q - 1$, por lo que $q \geq F_n$. □

Vamos a ver un ejemplo de aplicación del test de Pépin.

Ejemplo. Determinar si el número de Fermat con $n = 6$ es primo o no.

Solución. En primer lugar, calculamos el número de Fermat,

$$F_6 = 18446744073709551617.$$

Ahora tenemos que calcular $3^{(F_6-1)/2} = 3^{2^{2^6}/2} = 3^{2^{2^6-1}} = 3^{2^{63}}$ módulo F_6. Para ello, podemos calcular progresivamente las potencias necesarias usando que $3^{2^k} = (3^{2^{k-1}})^2$ (omitiremos algunos cálculos intermedios para no extendernos demasiado):

$$\begin{aligned}
&3^2 \equiv 9 \pmod{F_6},\\
&3^{2^2} \equiv 9^2 \equiv 81 \pmod{F_6},\\
&3^{2^3} \equiv 81^2 \equiv 6561 \pmod{F_6},\\
&3^{2^4} \equiv 6561^2 \equiv 43046721 \pmod{F_6},\\
&3^{2^5} \equiv 43046721^2 \equiv 1853020188851841 \pmod{F_6},\\
&3^{2^6} \equiv 1853020188851841^2 \equiv 8733085925571693938 \pmod{F_6},\\
&\ldots \equiv \ldots \pmod{F_6},\\
&3^{2^{62}} \equiv 12642481979558500686^2 \equiv 4082312613475963515 \pmod{F_6},\\
&3^{2^{63}} \equiv 4082312613475963515^2 \equiv 11860219800640380469 \pmod{F_6}.
\end{aligned}$$

Por lo tanto,

$$3^{(F_6-1)/2} = 3^{2^{63}} = 11860219800640380469 \not\equiv -1 \pmod{F_6},$$

y por el test de Pépin deducimos que F_6 no es primo, es un número compuesto.

3.2. Test de Proth

Ahora vamos a explicar lo que son los números de Proth. Se denominan **números de Proth** a los números de la forma

$$k \cdot 2^n + 1, \qquad k, n \in \mathbb{N}$$

con k impar y $k < 2^n$. Obsérvese que la condición de que k sea impar es en parte innecesaria, ya que los factores 2 que haya en k siempre se pueden recoger en el 2^n, de modo que esa exigencia simplemente sirve para poder identificar de manera única a los números de Proth. Los primeros números de Proth son:

$$\begin{aligned}
&P_0 = 2^1 + 1 = 3, && P_1 = 2^2 + 1 = 5, && P_2 = 2^3 + 1 = 9,\\
&P_3 = 3 \cdot 2^2 + 1 = 13, && P_4 = 2^4 + 1 = 17, && P_5 = 3 \cdot 2^3 + 1 = 25,\\
&P_6 = 2^5 + 1 = 33, && P_7 = 5 \cdot 2^3 + 1 = 41, && P_8 = 3 \cdot 2^4 + 1 = 49.
\end{aligned}$$

A los números de Proth que son primos se les llama **primos de Proth**. Hay dos casos concretos de números de Proth que son muy interesantes. El primero de estos casos es en el que k es igual a 1 y n es una potencia de 2, es decir, nos referimos a los números de Fermat. El otro caso interesante es cuando $k = n$, y a los números resultantes se les conoce como **números de Cullen**.

Para comprobar si un número de Proth es primo o no disponemos del conocido como test de Proth, el cual fue propuesto por su autor en 1878. El nombre de los números y del test alude al matemático autodidacta francés François Proth (1852–1879).

Teorema 3.2 (Test de Proth). Sea $m = k \cdot 2^n + 1$ con $0 < k < 2^n$. Si para algún entero a se cumple que

$$a^{(m-1)/2} \equiv -1 \pmod{m}, \tag{3.5}$$

entonces m es primo. Además, si a verifica que $(a|m) = -1$ (utilizando el símbolo de Jacobi), entonces m es primo si y solo si se cumple (3.5).

Demostración. Supongamos que se cumple (3.5) (en particular, esto implica que $\text{mcd}(a, m) = 1$) y veamos que m es primo. Como $k < 2^n$, necesariamente $\sqrt{m} < 2^n$. Esto se debe a que, como k es entero, tiene que cumplirse que $k \leq 2^n - 1$, y de este modo

$$\begin{aligned}\sqrt{m} &= \sqrt{k2^n + 1} \leq \sqrt{(2^n - 1)2^n + 1} \\ &= \sqrt{2^{2n} - 2^n + 1} < \sqrt{2^{2n} - 2^n + 2^n} = 2^n.\end{aligned}$$

Ahora, supongamos que m no es primo, luego existe p primo divisor de m con $p \leq \sqrt{m}$, por lo que $p < 2^n$. Como $p \mid m$, tenemos que

$$a^{(m-1)/2} = (a^k)^{2^{n-1}} \equiv -1 \pmod{p}. \tag{3.6}$$

De aquí deducimos que $(a^k)^{2^n} \equiv 1 \pmod{p}$, y por el lema 2.1 se tiene que $\text{ord}_p(a^k) = 2^n$. Así pues, como $\text{mcd}(a^k, p) = 1$, por el teorema de Euler-Fermat, $(a^k)^{\varphi(p)} \equiv 1 \pmod{p}$, luego $\text{ord}_p(a^k) \mid \varphi(p) = p - 1$ (ya que p es primo). Por tanto, debe cumplirse que $2^n \mid (p - 1)$, lo cual es absurdo, ya que $p < 2^n$.

Finalmente, si m es primo y $(a|m) = -1$, por el criterio de Euler $(a|m) \equiv a^{(m-1)/2} \pmod{m}$, por lo que se cumplirá (3.5). □

Cabe destacar que el test de Proth es una extensión del test de Pépin; basta con tomar $a = 3$ en el test de Proth para obtener el de Pépin. Esto funciona ya que, como veíamos en la demostración del test de Pépin, $(3|F_n) = -1$.

Vamos a utilizar el test de Proth en un sencillo ejemplo.

Ejemplo. Los cinco primeros primos de Proth son 3, 5, 13, 17, 41. Comprobar que estos cinco números son primos haciendo uso del test de Proth.

Solución. Siguiendo el enunciado del test de Proth, para cada uno de los cinco números necesitamos encontrar un entero a que verifique que $a^{(m-1)/2} \equiv -1 \pmod{m}$, siendo m el número de Proth en cuestión. Si hacemos eso, habremos probado que son primos. Veamos uno por uno cada caso:

- $P_0 = 2^1 + 1 = 3$. Tomando $a = 2$, tendríamos que $2^{(3-1)/2} = 2 \equiv -1$ (mód 3).
- $P_1 = 2^2 + 1 = 5$. Tomando $a = 3$, tendríamos que $3^{(5-1)/2} = 3^2 = 9 \equiv -1$ (mód 5).
- $P_3 = 3 \cdot 2^2 + 1 = 13$. Tomando $a = 5$, tendríamos que $5^{(13-1)/2} = 5^6 = 15625 \equiv -1$ (mód 13).
- $P_4 = 2^4 + 1 = 17$. Tomando $a = 3$, tendríamos que $3^{(17-1)/2} = 3^8 = 6561 \equiv -1$ (mód 17).
- $P_7 = 5 \cdot 2^3 + 1 = 41$. Tomando $a = 7$, tendríamos que

  $$7^{(41-1)/2} = 7^{20} = 79792266297612001 \equiv -1 \text{ (mód 41)}.$$

  (Tomar $a = 3$ también serviría en este caso, y daría un número algo más pequeño, pero así vemos otros ejemplos.)

3.3. Test para números de la forma $K \cdot p^n + 1$

Vamos ahora a presentar una generalización del test de Proth.

Teorema 3.3. Sea $m = K \cdot p^n + 1$ con p un número primo y $K < p^n$. Dado $a \in \mathbb{Z}$, si a no es resto de una potencia p-ésima módulo m, es decir, si no existe un x tal que $x^p \equiv a \pmod{m}$, entonces m es primo si y solo si $\Phi_p(a^{(m-1)/p}) \equiv 0 \pmod{m}$.

Demostración. Comenzaremos con la implicación directa. Supongamos que m es primo, en cuyo caso $a^{m-1} \equiv 1 \pmod{m}$, luego $(a^{m-1} - 1)$ es divisible por m. Ahora sabemos que, al ser p primo,

$$(x^p - 1) = (x - 1)(x^{p-1} + x^{p-2} + \cdots + x + 1) = (x - 1)\Phi_p(x).$$

Sustituyendo $a^{(m-1)/p}$ en la x en la expresión anterior, resulta

$$(a^{(m-1)/p} - 1)\Phi_p(a^{(m-1)/p}) = (a^{m-1} - 1).$$

Por tanto, como m divide a $(a^{m-1} - 1)$, también divide a

$$(a^{(m-1)/p} - 1)\Phi_p(a^{(m-1)/p}).$$

Vamos a comprobar ahora que m no puede dividir a $(a^{(m-1)/p} - 1)$ (recordemos que estamos partiendo de que m es primo). Sea g una raíz primitiva módulo m, luego sabemos que $\mathbb{Z}_m^* = \langle g \rangle$. Si m dividiera a $(a^{(m-1)/p} - 1)$, también se cumpliría que $a^{(m-1)/p} \equiv 1 \pmod{m}$. Sea i el entero tal que $g^i = a$, tendríamos

$$a^{(m-1)/p} \equiv (g^i)^{(m-1)/p} = g^{i(m-1)/p} \equiv 1 \pmod{m}. \tag{3.7}$$

El orden de g en $\mathbb{Z}_m^*$ es $m-1$, o, lo que es lo mismo, Kp^n. Por tanto, por (3.7), deducimos que $Kp^n \mid i(m-1)/p$, es decir, $Kp^n \mid iKp^{n-1}$. Esto solo puede ocurrir si i es múltiplo de p, y si esto ocurriera entonces a sería resto de una potencia p-ésima módulo m, lo que contradiría nuestra hipótesis de partida. Bastaría tomar $i = h \cdot p$ y, de aquí,

$$a = g^i = g^{h \cdot p} = (g^h)^p.$$

Así llegamos a la conclusión de que m divide a $\Phi_p(a^{(m-1)/p})$, lo que prueba la primera implicación.

Vamos con la otra implicación. Supongamos que $\Phi_p(a^{(m-1)/p}) \equiv 0 \pmod{m}$, o, lo que es lo mismo, que $\Phi_p(a^{Kp^{n-1}}) \equiv 0 \pmod{m}$. Tomando $X = a^K$ tenemos

$$\Phi_p(X^{p^{n-1}}) \equiv 0 \pmod{m}.$$

De nuevo, usando las propiedades de los polinomios ciclotómicos de la misma manera que antes, se deduce que $X^{p^n} \equiv 1 \pmod{m}$. Ahora, sea q un número primo divisor de m con $q \leq \sqrt{m}$, $\Phi_p(X^{p^{n-1}}) \equiv 0 \pmod{q}$ y $X^{p^n} \equiv 1 \pmod{q}$. Por lo tanto, el orden de X en $\mathbb{Z}_q^*$ debe ser un divisor de p^n. Supongamos que el orden de X es p^j con $j < n$; esto implicaría que

$$\Phi_p(X^{p^{n-1}}) = \Phi_p(X^{p^{n-1-j+j}}) = \Phi_p((X^{p^j})^{p^{n-1-j}}) = \Phi_p(1) = p \equiv 0 \pmod{q},$$

lo cual no puede ser, por tanto el orden de X es p^n, por lo que $p^n \mid q-1$. De aquí, se sigue que $p^n < q \leq \sqrt{m}$ y por tanto $p^{2n} < m = Kp^n + 1$. Como podemos ver, esto no podría ocurrir, ya que entonces $p^{2n} < Kp^n + p^n$ y esto implicaría que $p^n \leq K$, lo que contradiría nuestra hipótesis, luego m es primo. □

Ejemplo. Vamos a comprobar que los números 19 y 67 son primos, y que el número 99 es compuesto con este test. Además, vamos a ver qué ocurre si intentamos aplicar el test al número 51.

Solución. En primer lugar, vamos a poner los números dados con la forma correcta para aplicar el test, y después vamos a identificar un a que cumpla las restricciones del test que nos ayude a dar una respuesta en cada caso.

- $19 = 2 \cdot 3^2 + 1$. Tenemos el polinomio ciclotómico de grado 3, $\Phi_3(x) = x^2 + x + 1$. Tomando $a = 2$ se puede comprobar que a no es resto de una potencia p-ésima ($p = 3$) módulo 19, y, además, $\Phi_3(2^{(19-1)/3}) = \Phi_3(2^6) = 4161$ es divisible por 19, luego 19 es primo.

- $67 = 6 \cdot 11 + 1$. Tenemos el polinomio ciclotómico de grado 11, $\Phi_{11}(x) = x^{10} + \cdots + x + 1$. Tomando $a = 3$ se puede comprobar que a no es resto de una potencia p-ésima ($p = 11$) módulo 67, y, además,

  $$\Phi_{11}(3^{(67-1)/11}) = \Phi_{11}(3^6) = 424493878882316103872536288 51$$

  es divisible por 67, luego 67 es primo.

- $99 = 2 \cdot 7^2 + 1$. Tenemos el polinomio ciclotómico de grado 7, $\Phi_7(x) = x^6 + \cdots + x + 1$. Tomando $a = 3$ se puede comprobar que a no es resto de una potencia p-ésima ($p = 7$) módulo 99, pero, $\Phi_7(3^{(99-1)/7}) = \Phi_7(3^{14})$ NO es divisible por 99, luego 99 NO es primo.
- $51 = 2 \cdot 5^2 + 1$. Este es un caso peculiar. El problema aquí es que todos los posibles restos módulo 51, que como sabemos son $\{0, 1, \ldots, 50\}$, son restos de alguna potencia p-ésima ($p = 5$) módulo 51, de modo que este test no nos va a permitir concluir si el número es primo o no.

3.4. Test para números de la forma $4K \cdot p^n - 1$

Como anuncia el título, en esta sección hablaremos de un test que nos permitirá conocer la primalidad de ciertos números de la forma $N = 4K \cdot p^n - 1$. Para explicar este test comenzaremos definiendo el siguiente grupo:

$$G_N = \{a + bi \in \mathbb{Z}_N[i] : (a + bi)(a - bi) = a^2 + b^2 \equiv 1 \pmod{N}\}.$$

Utilizaremos algunas propiedades de este grupo en esta sección, aunque no las probaremos. En particular, si tomamos G_p con p primo, se tiene que $|G_p| = p+1$ si $p \equiv 3 \pmod 4$, y $|G_p| = p - 1$ si $p \equiv 1 \pmod 4$ (se puede encontrar más información sobre este hecho en [10]). Vamos ya con el test.

Teorema 3.4. Sea $N = 4K \cdot p^n - 1$ un entero con p un número primo impar y sea $w \in G_N$. Supongamos que existe un j con $1 \leq j \leq n$ tal que

- $\Phi_p(w^{4Kp^{j-1}}) \equiv 0 \pmod{N}$,
- $2j \geq \log_p(4K) + n$.

Entonces, N es primo.

Demostración. Sea $X = w^{4K}$, entonces $\Phi_p(X^{p^{j-1}}) \equiv 0 \pmod{N}$. Supongamos que N es compuesto y que q es un factor primo impar de N con $q \leq \sqrt{N}$. (Obsérvese que N no puede ser par por su construcción, luego podemos suponer sin problema que q es impar.) Como q divide a N, $\Phi_p(X^{p^{j-1}}) \equiv 0 \pmod{q}$. Por las propiedades de los polinomios ciclotómicos, $(x-1)\Phi_p(x) = (x^p - 1)$, luego, como q divide a $\Phi_p(X^{p^{j-1}})$, también divide a $(X^{p^j} - 1)$.

Parémonos en este punto. Como $w \in G_N$ y $q \mid N$, deducimos que $w \in G_q$. Al ser G_q un grupo (es decir, cerrado por productos), $X = w^{4K} \in G_q$. Por tanto, como $X^{p^j} \equiv 1 \pmod{q}$, el orden de X en G_q es un divisor de p^j. Veamos que tiene que ser necesariamente p^j. Si $X^{p^s} \equiv 1 \pmod{q}$ con $s < j$, esto implicaría que

$$\Phi_p(X^{p^{j-1}}) = \Phi_p(X^{p^{j-1-s+s}}) = \Phi_p((X^{p^s})^{p^{j-1-s}}) = \Phi_p(1) = p \equiv 0 \pmod{q},$$

lo cual es imposible, ya que p y q son coprimos entre sí. Ahora, como tenemos que el orden de X es p^j, se debe cumplir que $p^j \mid |G_q| = q \pm 1$. Como p y

q son primos impares, no puede ocurrir que $p^j = q$ ni que $p^j = q + 1$, luego $p^j < q \leq \sqrt{N}$. De ahí se deduce que

$$p^{2j} < N = 4K \cdot p^n - 1,$$

pero esto no puede ser cierto, ya que si no $p^{2j} < 4K \cdot p^n$, y, tomando logaritmos en base p, resultaría que $2j < \log_p(4K) + n$, lo cual contradiría nuestras hipótesis, luego N debe ser primo. □

Nótese que la condición $2j \geq \log_p(4K) + n$ se verifica solo si $4K \leq p^n$. Por ello, el resultado anterior puede certificar la primalidad de N solo si $4K \leq p^n$.

Veamos un ejemplo.

Ejemplo. Determinar si el número 19 es primo con la ayuda de este test.

Solución. $19 = 4 \cdot 1 \cdot 5^1 - 1$. Como $n = 1$ solo tenemos que comprobar que ocurre en el caso $j = 1$. En primer lugar, vamos a ver si se cumple la cota del logaritmo. Tenemos

$$2 \geq \log_5(4 \cdot 1) + 1 \Leftrightarrow 2 \geq 1.86135311\ldots,$$

luego se cumple la cota. Veamos la otra condición. Tomando $\omega = 2 + 4i$ resulta

$$\Phi_5((2+4i)^{4\cdot 1\cdot 5^0}) = \Phi_5(-112 - 384i) = 10850500753 - 23166923136i,$$

y como 19 divide a 10850500753 y a 23166923136 podemos asegurar que 19 es primo por el teorema 3.4.

Trabajando con este test, los números se hacen demasiado grandes con facilidad, lo que nos complica la tarea de mostrarlos. Es por esto que solo mostramos este ejemplo, que tiene cálculos más sencillos. Se pueden ver ejemplos con números más grandes en el apéndice A (página 61).

3.5. Test de Lucas-Lehmer

Ahora vamos a pasar a hablar de los **números de Mersenne**. Estos números tienen la siguiente apariencia:

$$M_p = 2^p - 1, \qquad p \text{ primo.}$$

Llamamos **primos de Mersenne**, como cabía esperar, a los números de Mersenne que son primos.

Los números de Mersenne surgen al buscar primos de la forma $a^n - 1$ con a un entero mayor que 1. Sabemos que se cumple la siguiente igualdad:

$$(a^n - 1) = (a - 1)(a^{n-1} + a^{n-2} + \cdots + a + 1), \qquad n \in \mathbb{N}. \tag{3.8}$$

Esto provoca que, para todo divisor d de n se tiene que $(a^d - 1) \mid (a^n - 1)$. Para comprobarlo, basta tomar $n = d \cdot k$ y sustituir a y n por a^d y k respectivamente en la ecuación (3.8). Si lo hacemos, el resultado que obtenemos es el siguiente:

$$((a^d)^k - 1) = (a^d - 1)((a^d)^{k-1} + (a^d)^{k-2} + \cdots + (a^d) + 1).$$

Llegamos así a la conclusión de que los únicos números primos de la forma $a^n - 1$ tendrán que cumplir que $a = 2$ y que n sea primo, de ahí que hagamos la definición anterior de números de Mersenne. Estos números deben su nombre al monje francés Marin Mersenne (1588–1648).

En 1876, Édouard Lucas (1842–1891) presentó un algoritmo que sirve para determinar, de una forma sorprendentemente eficiente, la primalidad de un número de Mersenne. Ese algoritmo no es otro que el test de Lucas-Lehmer, el cual exponemos en esta sección. Usando este algoritmo, Lucas pudo demostrar en 1876 que el número de Mersenne $M_{127} = 2^{127} - 1$ (que posee 39 dígitos) es primo. Este número era demasiado grande para estar al alcance de los métodos de sus predecesores. Para que el lector se haga una idea de la trascendencia de tal logro, basta con mencionar que M_{127} ha sido el mayor primo de Mersenne encontrado sin ayuda de ordenadores. No fue el último encontrado, ya que posteriormente se demostró que M_{61}, M_{89} y M_{107} son primos, también sin ayuda de ordenadores, pero estos números son más pequeños.

Pasaron bastantes años hasta que se encontró un nuevo primo de Mersenne. En 1952, ya con ayuda de ordenadores, Raphael Robinson descubrió la primalidad de los números M_{521}, M_{607}, M_{1279}, M_{2203} y M_{2281}, que tienen, respectivamente, 157, 183, 386, 664 y 687 cifras.

Desde entonces, el tamaño del mayor primo conocido ha ido aumentando sin descanso, y casi siempre ha sido un número de Mersenne (entre 1989 y 1992 fue un número de la forma $k \cdot 2^n - 1$ con 65087 dígitos). El motivo por el cual los mayores primos que se suelen encontrar son primos de Mersenne es que, además de la alta eficiencia del test de Lucas-Lehmer, este test se adapta muy bien a la aritmética binaria que utilizan los ordenadores actuales más potentes. Se conocen ya medio centenar de primos de Mersenne, aunque se desconoce si hay una cantidad infinita de ellos o no.

Los últimos primos de Mersenne, han sido descubiertos por el GIMPS (Great Internet Mersenne Prime Search), que es un proyecto de computación distribuida, fundado por George Woltman, que utiliza programas gratuitos con el fin de buscar números primos de Mersenne. El mayor primo de Mersenne conocido, a fecha de 2021, fue descubierto gracias a esta entidad en 2018. Es el $M_{82589933}$, que posee 24862048 dígitos.

Vamos a mostrar ya el test. Cabe mencionar que esta versión no es la inicial ideada por Lucas en 1876, sino que es una versión refinada de Derrick Henry Lehmer (1905–1991), propuesta en 1930.

Teorema 3.5 (Test de Lucas-Lehmer). Sea $M_p = 2^p - 1$ con p un primo impar y sea S_k la sucesión recurrente dada por:

$$S_{k+1} = S_k^2 - 2, \qquad S_0 = 4.$$

El número M_p es primo si y solo si $S_{p-2} \equiv 0 \pmod{M_p}$.

Para demostrar este teorema debemos hacer unas consideraciones previas. En primer lugar, como consecuencia de las propiedades del símbolo de Jacobi,

se deduce que

$$\begin{aligned}(-2|M_p) &= (-1|M_p)\cdot(2|M_p) = (-1)^{(M_p-1)/2}(-1)^{(M_p^2-1)/8}\\ &= (-1)^{2^{p-1}-1}(-1)^{((2^p-1)^2-1)/8} = (-1)(-1)^{2^{2p-2}-2^{p-1}}\\ &= (-1)\cdot 1 = -1.\end{aligned}$$

Además, por la ley de reciprocidad cuadrática, se tiene que

$$(3|M_p)(M_p|3) = (-1)^{2(2^p-2)/4} = (-1)^{2^{p-1}-1} = -1.$$

Por lo tanto, $(3|M_p) \neq (M_p|3)$. Vamos a ver si $(M_p|3)$ es igual a 1 o a -1. Como

$$M_p = 2^p - 1 \equiv (-1)^p - 1 = -1 - 1 = -2 \equiv 1 \pmod 3,$$

entonces $(M_p|3) = (1|3) = 1$, luego $(3|M_p) = (-2|M_p) = -1$.

Tomaremos $R = \mathbb{Z}_{M_p}[\sqrt{3}]$, $\alpha = 1+\sqrt{3}$ y $\bar{\alpha} = 1-\sqrt{3}$. Obsérvese que $\alpha, \bar{\alpha} \in R$, $\alpha\bar{\alpha} = -2$ y $\frac{\bar{\alpha}}{\alpha} = -2+\sqrt{3}$. Si M_p es primo, entonces (operando en R)

$$\begin{aligned}\alpha^{M_p} &= (1+\sqrt{3})^{M_p} \equiv 1^{M_p} + \sqrt{3}^{M_p} = 1 + (3^{\frac{M_p-1}{2}})\sqrt{3}\\ &\equiv 1 + (3|M_p)\sqrt{3} = 1 - \sqrt{3} = \bar{\alpha} \pmod{M_p}.\end{aligned}$$

Para demostrar el teorema de Lucas-Lehmer vamos a enunciar un lema cuya demostración engloba a la del teorema.

Lema 3.1. Sea p un primo impar, son equivalentes las siguientes afirmaciones:

I. M_p es primo.

II. $(2-\sqrt{3})^{2^{p-1}} \equiv -1 \pmod{M_p}$.

III. La recurrencia definida por $S_0 = 4$, $S_{k+1} = S_k^2 - 2$, $k \geq 0$ verifica que $S_{p-2} \equiv 0 \pmod{M_p}$.

Como podemos ver, si probamos la equivalencia entre (I) y (III) habremos demostrado el teorema de Lucas-Lehmer. Comenzaremos probando la equivalencia entre (I) y (II).

Demostración. (I) $\Rightarrow$ (II). Sabemos que $-1 = (-2|M_p)$, y como M_p es primo por (I), por el Criterio de Euler $(-2|M_p) \equiv (-2)^{\frac{M_p-1}{2}} \pmod{M_p}$. Ahora, recuperando los valores de α y $\bar{\alpha}$, y operando en R, tenemos que

$$\begin{aligned}(-2)^{\frac{M_p-1}{2}} &= (\alpha\bar{\alpha})^{\frac{M_p-1}{2}} \equiv (\alpha^{M_p+1})^{\frac{M_p-1}{2}} = (\alpha^{M_p-1})^{\frac{M_p+1}{2}}\\ &\equiv \left(\frac{\bar{\alpha}}{\alpha}\right)^{\frac{M_p+1}{2}} = (-2+\sqrt{3})^{2^{p-1}} \equiv (2-\sqrt{3})^{2^{p-1}} \pmod{M_p}.\end{aligned}$$

(II) $\Rightarrow$ (I). Sea q un divisor primo de M_p que verifica $q \leq \sqrt{M_p}$. Consideremos el cuerpo $\mathbb{Z}_q(\sqrt{3})$, el cual está formado por los elementos que describimos a continuación:

$$\mathbb{Z}_q(\sqrt{3}) = \left\{ \frac{a+b\sqrt{3}}{c+d\sqrt{3}} : a,b,c,d \in \mathbb{Z}_q,\ c+d\sqrt{3} \neq 0 \right\}.$$

Este cuerpo tiene q^2 elementos. Ahora, si tomamos $G = \mathbb{Z}_q(\sqrt{3})^*$ (que como sabemos corresponde con el conjunto de los elementos de $\mathbb{Z}_q(\sqrt{3})$ que poseen inverso multiplicativo) vemos que este conjunto posee q^2-1 elementos, ya que el 0 no posee inverso. Además, G presenta estructura de grupo multiplicativo. Por otro lado, como se cumple (II), tenemos que $(2-\sqrt{3})^{2^{p-1}} \equiv -1 \pmod{q}$, por ser q divisor de M_p, y ello implica que

$$(2-\sqrt{3})^{2^p} \equiv 1 \pmod{q}.$$

Sabemos también que $(2-\sqrt{3}) \in G$ y que su orden en G es 2^p por lo anterior. Por el teorema de Lagrange para los grados de los grupos (véase el teorema 2.7), se tiene que el orden de todo elemento de un grupo (finito) debe dividir al orden de dicho grupo, en nuestro caso 2^p debe dividir a q^2-1 y esto nos va a llevar a una contradicción. Veamos por qué. Si 2^p divide q^2-1, entonces $2^p \leq (q^2-1)$, y por tanto $2^p < (q^2+1)$. De aquí deducimos que $\sqrt{M_p} < q$, lo cual es absurdo.

Para probar la equivalencia entre (II) y (III), vamos a probar primero que la recurrencia dada en (III) se puede representar de la siguiente manera:

$$S_k = (2-\sqrt{3})^{2^k} + (2-\sqrt{3})^{-2^k} \tag{3.9}$$

Lo probamos por inducción. Comenzamos con $k=0$. Vemos que $S_0 = 4$, y sustituyendo $k=0$ en (3.9) resulta el mismo valor, luego de momento se verifica. Ahora lo suponemos cierto para el valor k y veamos si se cumple para $k+1$. Partimos de la expresión original de la recurrencia y sustituimos en ella la hipótesis:

$$\begin{aligned} S_{k+1} = S_k^2 - 2 &= ((2-\sqrt{3})^{2^k} + (2-\sqrt{3})^{-2^k})^2 - 2 \\ &= (2-\sqrt{3})^{2^{k+1}} + (2-\sqrt{3})^{-2^{k+1}} + 2 - 2 \\ &= (2-\sqrt{3})^{2^{k+1}} + (2-\sqrt{3})^{-2^{k+1}}. \end{aligned}$$

Lo que obtenemos es la expresión que resulta de sustituir $k+1$ en (3.9), luego se tiene la igualdad.

Ahora ya estamos en disposición de probar la equivalencia entre (II) y (III). (II) $\Rightarrow$ (III). Partimos de que se cumple $(2-\sqrt{3})^{2^{p-1}} \equiv -1 \pmod{M_p}$. Sustituimos k por $p-2$ en (3.9),

$$S_{p-2} = (2-\sqrt{3})^{2^{p-2}} + (2-\sqrt{3})^{-2^{p-2}}.$$

Ahora, sacando como factor común $(2-\sqrt{3})^{-2^{p-2}}$, resulta

$$(2-\sqrt{3})^{-2^{p-2}} \cdot \left((2-\sqrt{3})^{2^{p-1}} + 1 \right).$$

Como se cumple (II), la expresión obtenida es congruente con 0 módulo M_p. (III) $\Rightarrow$ (II). Sabemos que $S_{p-2} = (2-\sqrt{3})^{2^{p-2}} + (2-\sqrt{3})^{-2^{p-2}} \equiv 0 \pmod{M_p}$. Multiplicando esta expresión por $(2-\sqrt{3})^{2^{p-2}}$ obtenemos

$$(2-\sqrt{3})^{2^{p-1}} + 1 \equiv 0 \pmod{M_p},$$

de donde se tiene de forma trivial (II). □

Observación 3.1. Es obvio que al utilizar el test de Lucas-Lehmer siempre se reduce módulo M_p cada iteración S_k, para simplificar los cálculos.

En este punto es importante hacer la siguiente reseña. Si centramos nuestra atención en los tests de Pépin y Proth, vemos que el test de Lucas (en su versión original) fue enunciado en una fecha anterior a las fechas de los otros dos tests. Esto nos indica que es a Lucas al que hay que atribuirle el mérito de descubrir el primer test de este tipo (para números de Mersenne) en 1876, y que Pépin lo adaptó en 1877 para números de Fermat, y Proth hizo lo propio para los números que llevan su nombre en 1878. De hecho, la redacción inicial del resultado de Pépin era muy parecida a la de Lucas. Esta consistía en tomar $S_0 = 3$ y $S_{k+1} = S_k^2$, y la condición de primalidad para F_n era $S_{2n-1} \equiv -1 \pmod{F_n}$.

Ejemplo. Comprobar si M_7 y M_{11} son primos.

Solución. Analicemos cada caso haciendo uso del test de Lucas-Lehmer.

- $M_7 = 2^7 - 1 = 127$. Como $p = 7$, para evaluar la primalidad de M_7 necesitaremos calcular hasta el quinto término de la sucesión del teorema:

$$\begin{aligned}
&S_0 = 4,\\
&S_1 = S_0^2 - 2 = 4^2 - 2 = 14,\\
&S_2 = S_1^2 - 2 = 14^2 - 2 = 194 \equiv 67 \pmod{M_7},\\
&S_3 = S_2^2 - 2 = 67^2 - 2 = 4487 \equiv 42 \pmod{M_7},\\
&S_4 = S_3^2 - 2 = 42^2 - 2 = 1762 \equiv 111 \pmod{M_7},\\
&S_5 = S_4^2 - 2 = 111^2 - 2 = 12319 \equiv 0 \pmod{M_7}.
\end{aligned}$$

  Como $S_5 \equiv 0 \pmod{M_7}$, podemos concluir que $M_7 = 127$ es primo.

- $M_{11} = 2^{11} - 1 = 2047$. Como $p = 11$, para evaluar la primalidad de M_{11}

necesitaremos calcular hasta el noveno término de la sucesión del teorema:

$$\begin{aligned}
&S_0 = 4,\\
&S_1 = S_0^2 - 2 = 4^2 - 2 = 14,\\
&S_2 = S_1^2 - 2 = 14^2 - 2 = 194,\\
&S_3 = S_2^2 - 2 = 194^2 - 2 = 37634 \equiv 788 \pmod{M_{11}},\\
&S_4 = S_3^2 - 2 = 788^2 - 2 = 620942 \equiv 701 \pmod{M_{11}},\\
&S_5 = S_4^2 - 2 = 701^2 - 2 = 491399 \equiv 119 \pmod{M_{11}},\\
&S_6 = S_5^2 - 2 = 119^2 - 2 = 14159 \equiv 1877 \pmod{M_{11}},\\
&S_7 = S_6^2 - 2 = 1877^2 - 2 = 3523127 \equiv 240 \pmod{M_{11}},\\
&S_8 = S_7^2 - 2 = 240^2 - 2 = 57598 \equiv 282 \pmod{M_{11}},\\
&S_9 = S_8^2 - 2 = 282^2 - 2 = 79522 \equiv 1736 \pmod{M_{11}}.
\end{aligned}$$

Como $S_9 \not\equiv 0 \pmod{M_{11}}$, podemos concluir que $M_{11} = 2047$ NO es primo. De hecho, su factorización en números primos es

$$2047 = 23 \cdot 89.$$

Este es el primer número de Mersenne que no es primo, de ahí el interés por mostrarlo como ejemplo a pesar de que los cálculos que hay que realizar sean un poco más extensos.

3.6. Test $n-1$

Este test surge de una idea de Lucas. La idea consiste en comprobar si un cierto n es primo a partir de la factorización de $n-1$. Lo vemos a continuación.

Teorema 3.6. Sean a y n dos enteros con $n > 1$ tales que $a^{n-1} \equiv 1 \pmod{n}$ y $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ para todo primo $q \mid n-1$; entonces n es primo.

Demostración. La primera condición implica que el orden de a en $\mathbb{Z}_n^*$ es un divisor de $n-1$. A su vez, la segunda condición implica que el orden de a en $\mathbb{Z}_n^*$ no puede ser un divisor de $n-1$ menor que el propio $n-1$, luego el orden de a es precisamente $n-1$. Como se ha explicado anteriormente, el orden de a tiene que dividir a $\varphi(n)$, luego $n-1 \leq \varphi(n)$. Supongamos que n es compuesto. Sea p un factor primo de n, tenemos que p y n están en $\{1, 2, \ldots, n\}$ y ambos no son coprimos con n, luego por la definición de $\varphi(n)$ tenemos que $\varphi(n) \leq n-2$, lo que nos lleva a una contradicción, así que n debe ser primo. □

El principal obstáculo que surge en esta idea es la posible complejidad de conocer la factorización de $n-1$. Hay números para los cuales esa tarea es muy compleja, pero también hay otros, como los números de Fermat (vistos anteriormente), para los que conocer la factorización de ese número menos 1 es inmediato.

En este punto uno se plantea si, conociendo parcialmente la factorización de $n-1$, sería suficiente para poder aplicar una versión parecida del test. Es decir, si $n-1 = F \cdot R$ y conocemos la factorización completa de F, ¿podemos aprovechar el test anterior? La respuesta es "depende". Dependerá del tamaño de dicha F. Iremos acotando esa F poco a poco y viendo qué resultados podemos aplicar en función de la acotación. Vamos con el primero de esos resultados.

Teorema 3.7 (Pocklington). Sean n un entero mayor que 1, a otro entero y $n-1 = F \cdot R$ donde conocemos la factorización completa de F. Si $a^{n-1} \equiv 1 \pmod{n}$ y $\mathrm{mcd}(a^{(n-1)/q} - 1, n) = 1$ para todo primo $q \mid F$, entonces todos los factores primos de n son congruentes con 1 módulo F.

Demostración. Sea p un factor primo de n. Por la primera condición, tenemos que $a^{n-1} \equiv 1 \pmod{p}$ por ser p un divisor de n, luego el orden de a^R en $\mathbb{Z}_p^*$ divide a F. Veamos por qué ese orden debe ser justamente F.

Los mayores números que dividen a F menores que F son los que obtenemos al dividir F por cada factor primo q del propio F, es decir, los F/q con q primo y $q \mid F$. Con estos números, por la segunda condición del teorema,

$$(a^R)^{F/q} \not\equiv 1 \pmod{p}.$$

Si algún divisor d de F menor que el propio F fuera el orden, entonces alguno de los F/q (el que fuera múltiplo de d) debería hacer que

$$(a^R)^{F/q} \equiv 1 \pmod{p},$$

pero esto no ocurre, luego el orden de a^R en $\mathbb{Z}_p^*$ debe ser exactamente F. Por tanto, F divide a $|\mathbb{Z}_p^*| = p-1$ y se tiene el resultado. □

Corolario 3.1. Si se cumplen las hipótesis del teorema 3.7 y $F \geq \sqrt{n}$, entonces n es primo.

Demostración. Supongamos que n es compuesto. Por el teorema 3.7, todos los factores primos de n son congruentes con 1 módulo F. Al ocurrir esto, sea p un factor primo de n, $F \mid p-1$, luego $F \leq p-1$, por lo que p es mayor que F. Como $F \geq \sqrt{n}$ llegamos a la conclusión de que todos los factores primos de n son mayores que $\sqrt{n}$, lo que contradiría la proposición 1.1, luego n debe ser primo. □

El siguiente resultado nos va a permitir reducir un poco más la F.

Teorema 3.8 (Brillhart, Lehmer, y Selfridge). Supongamos que se verifican las hipotésis del teorema 3.7 y además $n^{1/3} \leq F < n^{1/2}$. Sea la representación en base F de n como sigue: $n = c_2F^2 + c_1F + c_01$, donde c_2, c_1 y c_0 son enteros en la lista $\{0, 1, \ldots, F-1\}$. Entonces, n es primo si y solo si $c_1^2 - 4c_2$ no es un cuadrado.

Demostración. Como $n-1 = F \cdot R$, se cumple que $n \equiv 1 \pmod{F}$, luego el dígito de las "unidades" en base F de n es 1, con lo cual de aquí en adelante tomaremos $c_0 = 1$. Así, tenemos que $n = c_2F^2 + c_1F + 1$.

Comenzamos demostrando la implicación inversa. Vamos a demostrar, en ambos casos, el contrarrecíproco. Supongamos que n no es primo, y veamos que $c_1^2 - 4c_2$ es un cuadrado. Por el teorema 3.7, todos los factores primos de n son congruentes con 1 módulo F, luego dado un factor primo p de n, $F \mid p-1$, por lo que $p \geq F+1$, de donde se deduce que $p > n^{1/3}$. Por esto, llegamos a la conclusión de que n posee exactamente dos factores primos, luego lo podemos escribir como sigue:

$$n = pq,\ p = aF+1,\ q = bF+1,\ a,b \in \mathbb{N}$$

(suponemos sin pérdida de generalidad que $a \leq b$). Así, tenemos que

$$n = (aF+1)(bF+1) = abF^2 + (a+b)F + 1 = c_2F^2 + c_1F + 1.$$

Si probamos que $c_2 = ab$ y $c_1 = a+b$, demostraremos que $c_1^2 - 4c_2$ es un cuadrado, puesto que $(a+b)^2 - 4ab = (a-b)^2$. En primer lugar, como $F^3 \geq n > abF^2$, tenemos que $ab < F \Rightarrow ab \leq F-1$. Ahora hay dos opciones:

- Caso 1. $(a+b) \leq F-1$. Es el caso que nos interesa. Lo trataremos a continuación.
- Caso 2. $(a+b) > F-1$. Si ocurre esto, tendríamos que $(a+b) > F-1 \geq ab$. Como a y b son enteros positivos, la única opción que existe para que $(a+b) > ab$ es que alguno de los dos sea 1. Supongamos sin pérdida de generalidad que $a = 1$ (ya que hemos supuesto que $a \leq b$). Como $ab = b \leq F-1$ y $a+b = 1+b > F-1$, tenemos que obligatoriamente $b = F-1$, pero este caso no podría darse, ya que si sustituimos los valores obtenidos para a y b en la expresión de n resultaría lo siguiente:

  $$n = (aF+1)(bF+1) = (F+1)((F-1)F+1) = F^3+1$$

  y esto contradice la hipótesis de que $F \geq n^{1/3}$.

Por tanto, concluimos que el único caso posible es el 1. Así, tanto ab como $(a+b)$ son menores o iguales que $F-1$, y por la unicidad de la representación de un número en la base x se sigue que $c_2 = ab$ y $c_1 = a+b$.

Vamos ahora a la implicación directa. Suponemos ahora que $c_1^2 - 4c_2$ es un cuadrado y vamos a ver que n debe ser compuesto en este caso. Sea $c_1^2 - 4c_2 = u^2$. Se tiene que

$$n = \left(\frac{c_1+u}{2}F+1\right)\left(\frac{c_1-u}{2}F+1\right). \tag{3.10}$$

Las dos fracciones que aparecen en (3.10) son enteros. Veamos por qué. Como $c_1^2 - 4c_2 = u^2$, se tiene que $c_1^2 \equiv u^2 \pmod 2$, luego $(c_1-u)(c_1+u) \equiv 0 \pmod 2$. En principio, 2 podría dividir solo a uno de los factores, pero va a dividir a ambos. Esto se debe a que si $c_1 \equiv u \pmod 2$ también $c_1 \equiv -u \pmod 2$, ya que $u \equiv -u \pmod 2$. La otra implicación se comprueba de forma equivalente.

Finalmente, al ser $c_2 > 0$ se cumple que $c_1^2 > u^2$, con lo cual $c_1 > |u|$ y ninguno de los factores en (3.10) es 1, luego n es compuesto. □

Por último, la cota más pequeña de F para la que vamos a poder tener certeza de si el número es primo o no nos la va a dar el siguiente teorema.

Teorema 3.9 (Konyagin y Pomerance). Supongamos que se verifican las condiciones del teorema 3.7, que $n \geq 214$ y que $n^{3/10} \leq F < n^{1/3}$. Sea $c_3F^3 + c_2F^2 + c_1F + c_01$ la representación en la base F de n y sea $c_4 = c_3F + c_2$. Entonces, n es primo si y solo si se verifican las condiciones siguientes:

I. $(c_1 + tF)^2 + 4t - 4c_4$ no es un cuadrado para todo $t \in \{1, 2, 3, 4, 5\}$.

II. Sea u/v el convergente de la fracción continua de c_1/F elegido de tal forma que v es el máximo valor que cumple $v < F^2/\sqrt{n}$. Si $d = \lfloor c_4v/F + 1/2 \rfloor$, entonces el polinomio $vx^3 + (uF - c_1v)x^2 + (c_4v - dF + u)x - d \in \mathbb{Z}[x]$ no tiene una raíz entera a tal que $aF + 1$ sea un factor no trivial (es decir, distinto de 1) de n.

Demostración. Comenzaremos por la implicación inversa, y la demostraremos probando el contrarrecíproco por reducción al absurdo. Supongamos que n es compuesto, y veamos que, necesariamente, o no se cumple (I) o no se cumple (II). Como se cumplen las hipótesis del teorema 3.7, todos los factores primos de n son congruentes con 1 módulo F, luego n será compuesto si existen dos enteros positivos a_1 y a_2 tales que $n = (a_1F+1)(a_2F+1)$ (supondremos sin pérdida de generalidad que $a_1 \leq a_2$). Además sabemos que $c_0 = 1$ como en la demostración del teorema 3.8. Para alcanzar la contradicción comencemos suponiendo que se verifican (I) y (II). Ahora vamos a establecer unas identidades y desigualdades que usaremos en la prueba. Tenemos que

$$\begin{aligned} n &= c_3F^3 + c_2F^2 + c_1F + 1 = \frac{c_4 - c_2}{F}F^3 + c_2F^2 + c_1F + 1 = c_4F^2 + c_1F + 1 \\ &= (a_1F + 1)(a_2F + 1) = a_1a_2F^2 + (a_1 + a_2)F + 1, \end{aligned}$$

y además existe un $t \geq 0$ tal que

$$a_1a_2 = c_4 - t, \qquad a_1 + a_2 = c_1 + tF. \tag{3.11}$$

Por tanto, como $(c_1 + tF)^2 + 4t - 4c_4$ es igual a $(a_1 + a_2)^2 - 4a_1a_2$, y esto es un cuadrado, forzosamente $t \geq 6$. Por otro lado, como $a_2 \geq a_1$ se cumple también que

$$a_2 \geq \frac{a_1 + a_2}{2} \geq \frac{c_1 + 6F}{2} \geq 3F, \tag{3.12}$$

y además, como

$$\frac{n}{F^2} = c_4 + \frac{c_1}{F} + \frac{1}{F^2} > c_4 - t = a_1a_2,$$

se verifica que

$$a_1 < \frac{n}{a_2F^2} \leq \frac{n}{3F^3}. \tag{3.13}$$

Por (3.11) se tiene que[1]

$$t \leq \frac{a_1+a_2}{F} \leq \frac{a_1a_2+1}{F} < \frac{c_4}{F} < \frac{n}{F^3}. \tag{3.14}$$

Además, multiplicando por a_1 a la segunda expresión de (3.11) se obtiene

$$a_1c_1 + a_1tF = a_1^2 + c_4 - t. \tag{3.15}$$

Ahora, utilizando la notación empleada en (II), resulta

$$\begin{aligned} a_1u + a_1tv - \frac{c_4v}{F} &= a_1v\left(\frac{u}{v} - \frac{c_1}{F}\right) + (a_1c_1 + a_1tF)\frac{v}{F} - \frac{c_4v}{F} \\ &= a_1v\left(\frac{u}{v} - \frac{c_1}{F}\right) + (a_1^2 + c_4 - t)\frac{v}{F} - \frac{c_4v}{F} \\ &= a_1v\left(\frac{u}{v} - \frac{c_1}{F}\right) + (a_1^2 - t)\frac{v}{F}. \end{aligned} \tag{3.16}$$

Nótese que (3.13), (3.14) y $t \geq 6$ implican que

$$|a_1^2 - t| < \text{máx}\left\{a_1^2, t\right\} \leq \text{máx}\left\{\frac{n^2}{9F^6}, \frac{n}{F^3}\right\} \leq \frac{n^2}{6F^6}. \tag{3.17}$$

Ahora, supongamos que $u/v = c_1/F$. Entonces, de (3.16) y (3.17) se deduce que

$$\left|a_1u + a_1tv - \frac{c_4v}{F}\right| = |a_1^2 - t|\frac{v}{F} < \frac{n^2}{6F^6}\frac{v}{F} < \frac{n^2}{6F^7}\frac{F^2}{\sqrt{n}} = \frac{n^{3/2}}{6F^5} \leq \frac{1}{6}.$$

(Nótese que la última desigualdad de la expresión anterior proviene de la hipótesis del enunciado de que $n^{3/10} \leq F$.) Si, por el contrario, $u/v \neq c_1/F$, sea u'/v' el siguiente convergente de la fracción continua de c_1/F después de u/v; entonces

$$v < \frac{F^2}{\sqrt{n}} \leq v',$$

y, además, por las propiedades de las fracciones continuas (ver página 303 de [18]), se tiene que

$$\left|\frac{u}{v} - \frac{c_1}{F}\right| < \frac{1}{vv'} \leq \frac{\sqrt{n}}{vF^2}.$$

Por lo tanto,

$$\left|a_1u + a_1tv - \frac{c_4v}{F}\right| < a_1v\frac{\sqrt{n}}{vF^2} + \frac{1}{6} < \frac{n^{3/2}}{3F^5} + \frac{1}{6} \leq \frac{1}{2}.$$

Ahora, sea $d = a_1u + a_1tv$; se tiene que $|d - c_4v/F| < 1/2$, lo que implica que $d = \lfloor c_4v/F + 1/2 \rfloor$. Multiplicando (3.15) por a_1v tenemos que

$$va_1^3 - c_1va_1^2 - a_1^2tvF - a_1tv + c_4a_1v = 0,$$

[1] La segunda desigualdad proviene de que la suma de dos enteros positivos, ambos mayores o iguales que 2, es menor igual que su producto, y si los enteros son mayores o iguales que 1 entonces la suma es menor o igual que el producto de ambos más 1. La tercera desigualdad proviene de que $t \geq 6$.

y usando que $-a_1tv = a_1u - d$, se consigue

$$va_1^3 + (uF - c_1v)a_1^2 + (c_4v - dF + u)a_1 - d = 0.$$

Así, acabamos de probar que a_1 es una raíz entera no nula del polinomio descrito en (II), luego hemos llegado a una contradicción.

Vamos ahora con la implicación directa. Supongamos que n es primo. Primero vamos a ver que se debe cumplir (I) forzosamente. Si dado $t \in \{1, 2, 3, 4, 5\}$ se tiene que $(c_1 + tF)^2 + 4t - 4c_4 = u^2$ con u un entero, entonces

$$\begin{aligned} n &= (c_4 - t)F^2 + (c_1 + tF)F + 1 \\ &= \left(\frac{c_1 + tF + u}{2}F + 1\right)\left(\frac{c_1 + tF - u}{2}F + 1\right). \end{aligned}$$

Como n es primo, esta debe ser una factorización trivial de n, es decir, que alguno de los dos factores debe ser 1. Por ello, debe cumplirse que $c_1 + tF - |u| = 0$, y esto conllevaría que $c_4 = t$. Vamos a ver que esto no puede ser. Partiendo de nuestras hipótesis

$$c_4 \geq F \geq n^{3/10} \geq 214^{3/10} > 5 \geq t,$$

lo que nos lleva a una contradicción. Por tanto, (I) debe cumplirse si n es primo.

Finalmente, si n es primo es trivial comprobar que (II) se verifica. □

Una vez enunciados estos teoremas, podemos establecer formalmente el algoritmo del test $n - 1$.

Test $n-1$**.** Sean $n \geq 214$ y $n-1 = F \cdot R$ donde conocemos la factorización en números primos de F. Si $F \geq n^{3/10}$, entonces el algoritmo 1 (véase la página 43) devuelve YES si el número n es primo y NO si n es compuesto. (Obsérvese que si $n < 214$ y $F \geq n^{1/3}$ el algoritmo también responde adecuadamente, ya que la exigencia de que $n \geq 214$ está relacionada con el último teorema, el de Konyagin y Pomerance, pero, para ser rigurosos, el algoritmo funciona correctamente si se cumplen las hipótesis descritas anteriormente.)

Al igual que nos ha pasado previamente con otros tests, como el de los números de la forma $4K \cdot p^n - 1$, los cálculos que hay que realizar para usar este algoritmo son bastante extensos. Por este motivo vamos a ver un único ejemplo (no demasiado complicado) en detalle. Se pueden encontrar más ejemplos en el apéndice A, junto con una implementación del test en *Mathematica*.

Ejemplo. Vamos a verificar de dos formas distintas que el número 1471 es primo con este test.

Solución. En ejemplos sencillos como este se puede ver fácilmente que $n = 2 \cdot 3 \cdot 5 \cdot 7 + 1$. Podemos tomar distintos F y R, siempre que se cumplan las restricciones del algoritmo. El test está pensado para números en los que conozcamos la factorización de F y no de R, por lo que lo lógico sería tomar un F lo más pequeño posible. Vamos a escogerlos de dos formas distintas.

Algoritmo 1. El test $n-1$

Input: entero $n \geq 214$

Output: YES o NO: Será YES si el número n es primo y NO en otro caso

1: (Test de Pocklington)
2: **If** (Dado $a \in [2, n-2], a^{n-1} \not\equiv 1 \pmod{n}$), **Output** NO
3: **For** q tal que q es primo y $q \mid F$ **do**
4: $g = \text{mcd}(a^{(n-1)/q} \pmod{n} - 1, n)$
5: **If** $(1 < g < n)$, **Output** NO
6: **If** $(g == n)$, **go to** 1
7: (Test con la primera cota)
8: **If** $(F \geq n^{1/2})$, **Output** YES
9: (Test con la segunda cota)
10: **If** $(n^{1/3} \leq F < n^{1/2})$
11: Dados c_2, $c_1 \in \{0, 1, \ldots, F-1\}$ tales que $n = c_2F^2 + c_1F + 1$
12: **If** ($c_1^2 - 4c_2$ no es un cuadrado), **Output** YES
13: **Output** NO
14: (Test con la tercera cota)
15: **If** $(n^{3/10} \leq F < n^{1/3})$
16: **If** ((I) **and** (II)), **Output** YES
17: **Output** NO

- $F = 14$ y $R = 105$. En este caso

$$\sqrt[3]{1471} = 11.37289 \leq 14 < \sqrt{1471} = 38.35361\ldots,$$

luego podemos utilizar el test con la segunda cota, es decir, el teorema de Brillhart, Lehmer y Selfridge. En primer lugar, se puede probar que se cumplen las hipótesis del teorema 3.7. Ahora, si obtenemos la representación en base F de n resulta

$$n = c_2F^2 + c_1F + c_01 = 7 \cdot (14)^2 + 7 \cdot 14 + 1 = 1471,$$

de donde

$$c_1^2 - 4c_2 = 7^2 - 4 \cdot 7 = 21.$$

Como 21 no es un cuadrado, podemos asegurar que 1471 es primo.

- $F = 10$ y $R = 147$. Con estos valores

$$\sqrt[10]{1471^3} = 8.91833 \leq 10 < \sqrt{1471} = 11.37289\ldots,$$

luego podemos utilizar el test con la tercera cota, es decir, el teorema de Konyagin y Pomerance. Al igual que antes, se puede probar que se cumplen las hipótesis del teorema 3.7. Ahora, si obtenemos la representación en la base F de n resulta

$$n = c_3F^3 + c_2F^2 + c_1F + c_01 = 10^3 + 4 \cdot 10^2 + 7 \cdot 10^2 + 1 = 1471,$$

de donde $c_4 = c_3F + c_2 = 1 \cdot 10 + 4 = 14$. Con esto, vamos a ver si se cumplen las condiciones (I) y (II). Para la primera debemos comprobar que $g(t) = (c_1 + tF)^2 + 4t - 4c_4 = (7 + 10t)^2 + 4t - 56$ no sea un cuadrado para todo $t \in \{1, 2, 3, 4, 5\}$. Sustituyendo, como

- $g(1) = 237$,
- $g(2) = 681$,
- $g(3) = 1325$,
- $g(4) = 2169$,
- $g(5) = 3213$,

la condición (I) se cumple. Vamos con la segunda. En este ejemplo se tiene que $c_1/F = 7/10$. Calculamos todos los convergentes de la fracción continua de $7/10$, que es una fracción sencilla. El primero es 0, el segundo es 1, el tercero es $2/3$ y el cuarto es la propia fracción, $7/10$. Como

$$F^2/\sqrt{n} = 100/\sqrt{1471} = 2.60731\ldots,$$

el convergente necesario para el teorema es el 1, ya que $3 \geq 2.60731\ldots$ Así, tomando $u = 1$ y $v = 1$, se tiene que $d = \lfloor c_4v/F + 1/2 \rfloor = \lfloor 19/10 \rfloor = 1$. Ahora construimos el polinomio:

$$\begin{aligned} vx^3 + (uF - c_1v)x^2 + (c_4v - dF + u)x - d \\ = x^3 + (10 - 7)x^2 + (14 - 10 + 1)x - 1 \\ = x^3 + 3x^2 + 5x - 1. \end{aligned}$$

Este polinomio no tiene raíces enteras, tiene dos raíces complejas conjugadas y una real (como es primitivo, es irreducible en $\mathbb{Z}[x]$); por tanto, se cumple (II). Como se cumplen las dos restricciones podemos asegurar que el número 1471 es primo.

3.7. Otros algoritmos

Son muchos los tests que se pueden encontrar para determinar si un número de una forma concreta es primo no. En este trabajo nosotros solo hemos visto algunos de los más importantes.

Por mencionar algunos otros, hay uno, propuesto por Riesel, para números de la forma $N = h \cdot 2^n - 1$ en [14], y se puede encontrar otra prueba del correcto funcionamiento de dicho test en [16]. Otro test más general para números de la forma $N = h \cdot 2^n \pm 1$ puede hallarse en [6] y otro para números de la forma $N = A \cdot 3^n \pm 1$ en [5], que involucra el concepto de reprocidad cúbica.

Capítulo 4

Algoritmos para cualquier entero positivo

No siempre nuestro objetivo va a ser comprobar si un número especial como un número de Fermat o de Mersenne es primo. En ocasiones querremos ver si un número cualquiera, del que inicialmente no tenemos gran información, es primo o no. Para estas situaciones contamos con tests como los siguientes. Los algoritmos que vamos a ver aquí son correctos, pero con las herramientas de las que disponemos hoy en día también son bastante ineficientes.

4.1. Test de Wilson

Wilson conjeturó que, si un número p era primo, entonces $(p-1)!+1$ debía ser divisible por p, pero no pudo probarlo. Es importante mencionar que esta propiedad ya fue observada mucho antes por el matemático Ibn al-Haytham sobre el año 1000. A pesar de esto no fue probada hasta 1773, por Lagrange, pero aun así se atribuye el nombre del resultado a Wilson. Para demostrar este hecho Lagrange hizo uso de un teorema suyo que no veremos en este trabajo, de modo que nosotros probaremos el resultado con la ayuda del siguiente lema, que es bastante más sencillo.

Lema 4.1. Si p es primo, entonces $x^2 \equiv 1 \pmod{p}$ si y solo si $x \equiv \pm 1 \pmod{p}$.

Demostración. La implicación inversa es evidente, de modo que nos centraremos en la directa. Supongamos que $x^2 \equiv 1 \pmod{p}$. Como p divide a $x^2 - 1 = (x+1)(x-1)$ y p es primo, entonces p divide a $(x+1)$ o a $(x-1)$, luego o $x \equiv -1 \pmod{p}$ o $x \equiv +1 \pmod{p}$, respectivamente. □

Vamos ya con el teorema de Wilson.

Teorema 4.1 (Teorema de Wilson). Si p es primo, entonces $(p-1)! \equiv -1 \pmod{p}$.

Demostración. El resultado se da de forma trivial para $p = 2$ y $p = 3$, por lo que supondremos que $p \geq 5$. Como p es primo, para todo $a \in \{1, 2, \ldots, p-1\}$ se tiene $\text{mcd}(a, p) = 1$, y por la proposición 2.1 sabemos que existe un único $a^{-1} \in \{1, 2, \ldots, p-1\}$ tal que $aa^{-1} \equiv 1 \pmod{p}$. Por el lema previo, $a = a^{-1}$ si y solo si $a = 1$ o $a = p-1$. Así pues, centrándonos en el resto de los a posibles, podemos dividir el conjunto $\{2, 3, \ldots, p-2\}$ en $(p-3)/2$ parejas de elementos (obsérvese que como $p \geq 5$ esto se puede hacer) de la forma $\{a_j, a_j^{-1}\}$ con $j \in \{1, 2, \ldots, (p-3)/2\}$, verificando $a_j a_j^{-1} \equiv 1 \pmod{p}$. En consecuencia, agrupando los elementos correctamente, tenemos

$$\begin{aligned}(p-1)! &= 1 \cdot 2 \cdot 3 \cdots (p-2)(p-1) \\ &= (p-1) \prod_{j=1}^{(p-3)/2} a_j a_j^{-1} \equiv p-1 \equiv -1 \pmod{p}.\end{aligned}$$ □

Ahora, vamos a probar el inverso del teorema de Wilson.

Teorema 4.2. Si un entero $n > 1$ verifica $(n-1)! \equiv -1 \pmod{n}$, entonces n es primo.

Demostración. Vamos a probarlo por reducción al absurdo. Si n no fuera primo, existiría un primo p menor que n con $p \mid n$. Este p cumpliría que $p \mid (n-1)!$. Si se cumpliera nuestra hipótesis, entonces tendríamos que, como $n \mid ((n-1)!+1)$ y $p \mid n$, entonces $p \mid ((n-1)!+1)$, pero esto no puede ser, ya que si p dividiera a $(n-1)!$ y a $((n-1)!+1)$ entonces dividiría a la resta de ambos, es decir, $p \mid 1$, y eso es absurdo. □

El teorema de Wilson junto con su inverso proporcionan un test de primalidad correcto, pero inútil en la práctica, puesto que conocer el factorial de $(n-1)$ para determinar si n es primo es demasiado laborioso, sobre todo cuando los números empiezan a crecer.

Ejemplo. Comprobar si los números 7 y 12 son primos usando el test de Wilson.

Solución. Veamos cada caso por separado:

- 7. Calculamos el factorial de 6:

  $$6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720.$$

  Ahora, $720 + 1 = 721$ es divisible por 7, luego por el teorema de Wilson sabemos que 7 es primo.

- 12. Calculamos el factorial de 11:

  $$11! = \prod_{i=1}^{11} i = 39916800.$$

  Ahora, $39916800 + 1 = 39916801$ NO es divisible por 12, luego por el teorema de Wilson concluimos que 12 no es primo.

4.2. Algoritmo AKS

Tras el paso de muchos años intentando descubrir si el problema de determinar la primalidad de un número dado se podía resolver en tiempo polinomial, el matemático Manindra Agrawal, junto con sus dos estudiantes doctorales Neeraj Kayal y Nitin Saxena, dieron por fin respuesta a esta pregunta. Para sorpresa de gran parte de la comunidad matemática, esta respuesta fue afirmativa.

En agosto de 2002, este trío de investigadores publicó su artículo "PRIMES is in P", con el que ponían fin a años de incertidumbre. En dicho artículo propusieron un algoritmo capaz de determinar si un entero positivo dado es primo o no en tiempo polinomial. A pesar de esto, a día de hoy la aplicación de este algoritmo a los posibles primos del tamaño que requieren las aplicaciones prácticas (en particular, la criptografía) no es tan efectiva como cabría esperar. Hay ciertos algoritmos como el de las curvas elípticas (el cual no estudiaremos aquí) que son más eficientes. De todas formas nosotros no probaremos que el algoritmo tiene complejidad polinomial, ya que esto queda fuera de los objetivos de esta memoria. Se pueden encontrar más detalles sobre esto en [1].

La idea del algoritmo no deja de ser más sencilla de lo que uno podría esperar, ya que está basado en una extensión no demasiado compleja del pequeño teorema de Fermat.

A continuación, expondremos dicha extensión, enunciaremos el algoritmo en sí y pasaremos a demostrar que funciona correctamente.

Lema 4.2. Sean $a \in \mathbb{Z}$ y $n \in \mathbb{N}$ con $n \geq 2$ y $\mathrm{mcd}(a, n) = 1$. Entonces, n es primo si y solo si

$$(x + a)^n \equiv x^n + a \pmod{n}. \tag{4.1}$$

Demostración. Sea $0 < i < n$, el coeficiente de x^i en $(x+a)^n - x^n + a$ es $\binom{n}{i}a^{n-i}$ (binomio de Newton).

Supongamos que n es primo. Esto implica que $\binom{n}{i} \equiv 0 \pmod{n}$ y por tanto se tiene (4.1).

Supongamos ahora que n es compuesto y veamos que entonces (4.1) no se cumple. Sea q un primo que divide a n, y sea k la máxima potencia en la que q divide a n, esto es, $q^k \mid n$ y $q^{k+1} \nmid n$, o lo que es lo mismo, $q^k \mid\mid n$. Sabemos que $\mathrm{mcd}(q^k, a^{n-q}) = 1$, ya que n y a son coprimos. Como además $q^k \nmid \binom{n}{q}$, el coeficiente de x^q no es congruente con 0 módulo n, y por tanto no se tiene (4.1). Veamos por qué $q^k \nmid \binom{n}{q}$. Como

$$\binom{n}{q} = \frac{n(n-1)(n-2)\cdots(n-q+1)}{q!}$$

y $q^k \mid\mid n$, tomando $n = q^k \cdot s$ con $s \in \mathbb{N}$ tal que $q \nmid s$, tendríamos que

$$\binom{n}{q} = \frac{q^{k-1}s(n-1)(n-2)\cdots(n-q+1)}{(q-1)!}.$$

Así, q^k dividirá a $\binom{n}{q}$ si y solo si q divide a alguno de los elementos de la siguiente lista $\{(n-1), (n-2), \ldots, (n-q+1)\}$, y esto no ocurre. Veamos por qué. Si

$q \mid (n-1)$, como $q \mid n$, entonces $q \mid (n-(n-1)) = 1$ (absurdo, pues q es primo). Utilizando el mismo argumento sucesivamente (junto con la primalidad de q) llegamos a la conclusión de que q no puede dividir a ningún $(n-r)$ con $r \leq (q-1)$, y por tanto $q^k \nmid \binom{n}{q}$. □

Con lo expuesto en el lema 4.2 obtenemos un algoritmo que nos permitiría reconocer la primalidad de un número, pero surge un problema. El problema es que el número de operaciones necesarias es exponencial. Sin embargo, podemos reducir el número de operaciones necesarios evaluando la expresión (4.1) módulo $x^r - 1$, con un r apropiado. Así, la expresión (4.1) quedaría sustituida por la siguiente:

$$(x+a)^n \equiv x^n + a \ (\text{mód } x^r - 1, n). \tag{4.2}$$

A partir del lema es inmediato que todos los primos n satisfacen la ecuación (4.2) para todos los valores de a y r. El problema ahora es que hay algunos n compuestos que también cumplen la ecuación (4.2) para unos pocos valores de a y r. Sin embargo, podemos solucionar este problema. Veremos que para una r elegida, si la ecuación (4.2) se satisface para varios a, entonces n debe ser una potencia de un primo. El número de a y la r adecuada están acotados por un polinomio en $\log(n)$ y, por tanto, podemos conseguir un algoritmo determinista en tiempo polinómico para comprobar la primalidad.

Además, mientras no se vuelva a mencionar, se utilizará la notación $\log(x)$ para denotar el logaritmo en base 2, en lugar del logaritmo neperiano habitual. Es decir, representándolo por medio de un simple ejemplo, el lector tendrá que tener que en cuenta que

$$\log(4) = \log_2(4) = 2.$$

Lema 4.3. Si denotamos por $\text{mcm}(n)$, al mínimo común múltiplo de los n primeros números naturales, para todo $n \geq 7$ se cumple

$$\text{mcm}(n) \geq 2^n.$$

Ya estamos en disposición de enunciar el algoritmo y demostrar que funciona correctamente.

Teorema 4.3. El algoritmo 2 devuelve PRIMO si y solo si n es primo. Demostraremos este teorema a continuación, mediante el enunciado de una serie de lemas que iremos probando.

Lema 4.4. Si n es primo, el algoritmo 2 devuelve PRIMO.

Demostración. Si n es primo, los pasos 1 y 3 del algoritmo no pueden devolver COMPUESTO nunca. El bucle de los pasos 5 y 6 tampoco puede devolver COMPUESTO nunca por el lema 4.2.

Por tanto, el algoritmo detectará que n es primo en los pasos 4 o 7. □

Para probar la implicación inversa vamos a necesitar un poco más de trabajo. Si el algoritmo devuelve PRIMO en el paso 4, n debe ser primo, ya que, si no, en el paso 3 habríamos detectado algún factor no trivial de n. Esto se debe

Algoritmo 2. Algoritmo de primalidad AKS

Input: entero $n > 1$

Output: PRIMO o COMPUESTO: Será PRIMO si el número n es primo y COMPUESTO en otro caso

1: **If** ($\exists\ a \in \mathbb{N}$ y $b > 1$ tales que $n = a^b$), **Output** COMPUESTO

2: Busca el menor r tal que $\text{ord}_r(n) > \log^2(n)$

3: **If** ($\exists\ a \leq r$ tal que $1 < \text{mcd}(a, n) < n$), **Output** COMPUESTO

4: **If** ($n \leq r$), **Output** PRIMO

5: **For** $a = 1$ **to** $\lfloor \sqrt{\varphi(r)} \log(n) \rfloor$ **do**

6: **If** $((x + a)^n \not\equiv x^n + a \pmod{x^r - 1, n})$, **Output** COMPUESTO

7: **Output** PRIME

a que, al ser $n \leq r$, en el paso 3 comprobamos si existe un $a \leq n$ tal que $1 < \text{mcd}(a, n) < n$, que es la condición necesaria y suficiente para que n sea compuesto. Nos queda comprobar qué pasa si el algoritmo devuelve PRIMO en el paso 7. Para ello, vamos a centrarnos en el r que debemos obtener en el paso 2 y en la comprobación que debemos evaluar en los pasos 5 y 6.

Lema 4.5. Dado un entero $n > 1$, existe un r entero con $r \geq 2$ y $r \leq \text{máx}\{3, \lceil \log^5(n) \rceil\}$ tal que $\text{ord}_r(n) > \log^2(n)$.

Demostración. Para $n = 2$, $r = 3$ verifica las condiciones del lema, de modo que vamos a centrarnos en el caso $n > 2$. En este caso, como $n \geq 3$, se tiene la siguiente cadena de implicaciones:

$$\begin{aligned} \log(n) \geq \log(3) &\Leftrightarrow \log^5(n) \geq \log^5(3) \\ &\Leftrightarrow \log^5(n) \geq 10.00218\ldots \Leftrightarrow \lceil \log^5(n) \rceil > 10. \end{aligned}$$

Esto nos lleva a que, para probar que $r \leq \text{máx}\{3, \lceil \log^5(n) \rceil\}$, nos bastará con probar que $r \leq \lceil \log^5(n) \rceil$.

Observación 4.1. El mayor entero k que verifica $m^k \leq B = \lceil \log^5(n) \rceil$, con $m \geq 2$, es $\lfloor \log(B) \rfloor$. En efecto, partiendo de que $m \geq 2$ tenemos que

$$\log(m) \geq \log(2) \Leftrightarrow \log(m) \geq 1 \Leftrightarrow k \log(m) \geq k.$$

Utilizando este hecho,

$$m^k \leq B \Leftrightarrow k \log(m) \leq \log(B) \Leftrightarrow k \leq \log(B).$$

Puesto que k debe ser un entero, vemos que el mayor valor que puede tomar es $\lfloor \log(B) \rfloor$.

Ahora, tomamos r como el menor entero positivo que no divide al siguiente producto:

$$n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i - 1). \tag{4.3}$$

Sea $r = r_1^{c_1} \cdots r_z^{c_z}$ la descomposición en factores primos de r. Obsérvese que todos los factores primos de r no pueden dividir a la vez a n. Si esto ocurriera, entonces $r \mid n^{\lfloor \log(B) \rfloor}$. Esto se debe a que, para que $r \leq B$, se debe cumplir que todos los c_i con $1 \leq i \leq z$ sean menores o iguales que $\lfloor \log(B) \rfloor$, ya que, si no, $r_i^{c_i} > B$ por la observación 4.1.

Como r no divide a n, $\frac{r}{\text{mcd}(r,n)} \nmid n$ por la definición de máximo común divisor. Además,

$$\frac{r}{\text{mcd}(r,n)} \nmid \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i - 1),$$

ya que si esto ocurriera tanto $\frac{r}{\text{mcd}(r,n)}$ como $\text{mcd}(r,n)$ dividirían a (4.3), y por tanto su producto que es r también dividiría a (4.3), pero eso no puede ser por la definición de r.

En conclusión, $\frac{r}{\text{mcd}(r,n)}$ no divide a (4.3). Por tanto, como r es el menor entero positivo que no divide a (4.3), se tiene que $\text{mcd}(r,n) = 1$.

Además, como para cada $i \in \{1, 2, \ldots, \lfloor \log^2(n) \rfloor\}$ r no divide a $n^i - 1$, necesariamente $\text{ord}_r(n) > \log^2(n)$. Finalmente,

$$\begin{aligned} n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i - 1) &< n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} n^i \\ &\leq n^{\lfloor \log(B) \rfloor} \cdot n^{\left(\sum_{i=1}^{\lfloor \log^2(n) \rfloor} i \right)} = n^{\lfloor \log(B) \rfloor} \cdot n^{\lfloor \log^2(n) \rfloor (1 + \lfloor \log^2(n) \rfloor)/2} \\ &\leq n^{\log^4(n)} \leq 2^{\log^5(n)} \leq 2^B. \end{aligned}$$

Observación 4.2. Para la penúltima desigualdad, obsérvese que, como $b^{\log_b(a)} = a$, se tiene que $n^{\log^4(n)} = (2^{\log(n)})^{\log^4(n)}$.

Ahora, como

$$\text{mcm}(B) \geq 2^B > n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i - 1)$$

puesto que $B = \lceil \log^5(n) \rceil > 10$ y podemos aplicar el lema 4.3, tenemos que $r \leq B$. Veamos por qué. Si $r > B$, para todo $i \leq B < r$ debería cumplirse que i dividiera a (4.3) por la minimalidad de r. Si esto fuera cierto, entonces $\text{mcm}(B)$ tendría que dividir también a (4.3), pero hemos demostrado que $\text{mcm}(B)$ es estrictamente mayor que (4.3), por tanto alcanzamos una contradicción y r debe ser menor o igual que B. □

Como estamos tomando $n > 1$ y hemos probado que $\text{ord}_r(n) > \log^2(n)$, entonces $\text{ord}_r(n) > 1$. Al ocurrir esto, debe existir un factor primo p de n tal que $\text{ord}_r(p) > 1$. Es claro que $p > r$, ya que si no en los pasos 3 o 4 del algoritmo habríamos obtenido un resultado. (Recordemos que estamos evaluando qué ocurrirá en los pasos 5-7 del algoritmo.) Como el $\text{mcd}(r, n) = 1$ y $\text{mcd}(r, p) = 1$, se tiene que $p, n \in \mathbb{Z}_r^*$. Sea $l = \lfloor \sqrt{\varphi(r)} \log(n) \rfloor$; los pasos 5 y 6 del algoritmo comprueban si se cumplen l ecuaciones. Como nos encontramos comprobando el caso en que el algoritmo devuelve PRIMO en el paso 7, se cumplen dichas ecuaciones, es decir, se cumple que

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}$$

para todo a con $0 \leq a \leq l$ (en $a = 0$ la ecuación también se verifica de forma trivial). Como $p \mid n$,

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, p}$$

para todo a con $0 \leq a \leq l$. Además, por el lema 4.2, también tenemos que

$$(x + a)^p \equiv x^p + a \pmod{x^r - 1, p}$$

para todo a con $0 \leq a \leq l$. Así, juntando las dos ecuaciones anteriores vemos que

$$(x + a)^{\frac{n}{p}} \equiv x^{\frac{n}{p}} + a \pmod{x^r - 1, p}$$

para todo a con $0 \leq a \leq l$. Observamos que n y $\frac{n}{p}$ cumplen la misma "propiedad". Vamos a dar nombre a dicha propiedad.

Definición 4.1 (Número introspectivo). Dado un polinomio $f(x)$ y un entero positivo m, decimos que m es un número introspectivo o, simplemente, es introspectivo para $f(x)$, si

$$[f(x)]^m \equiv f(x^m) \pmod{x^r - 1, p}.$$

Así, n y $\frac{n}{p}$ son introspectivos para $f(x) = x + a$ en nuetro caso. A continuación, vamos a ver que este conjunto de números es cerrado respecto a la multiplicación.

Lema 4.6. Sean m y m' dos números introspectivos para un cierto polinomio $f(x)$. Entonces $m \cdot m'$ también es introspectivo para $f(x)$.

Demostración. Como m es introspectivo para $f(x)$, tenemos que

$$[f(x)]^{m \cdot m'} \equiv [f(x^m)]^{m'} \pmod{x^r - 1, p}.$$

Es más, como m' también es introspectivo para $f(x)$, tomando x^m en lugar de x resulta que

$$\begin{aligned}[f(x^m)]^{m'} &\equiv f(x^{m \cdot m'}) \pmod{x^{m \cdot r} - 1, p} \\ &\equiv f(x^{m \cdot m'}) \pmod{x^r - 1, p} \text{ (ya que } (x^r - 1) \text{ divide a } (x^{m \cdot r} - 1)).\end{aligned}$$

Juntando ambas ecuaciones tenemos que

$$[f(x)]^{m \cdot m'} \equiv f(x^{m \cdot m'}) \pmod{x^r - 1, p}. \qquad \square$$

Otra propiedad importante que también se cumple es que, dado un entero positivo m, el conjunto de los polinomios para los cuales m es introspectivo es también cerrrado respecto a la multiplicación.

Lema 4.7. Sea m un entero positivo y sean $f(x)$ y $g(x)$ dos polinomios para los que m es introspectivo. Entonces, m es también introspectivo para el polinomio $f(x) \cdot g(x)$.

Demostración. Es inmediato sin más que observar que

$$[f(x) \cdot g(x)]^m \equiv [f(x)]^m \cdot [g(x)]^m \equiv f(x^m) \cdot g(x^m) \pmod{x^r - 1, p}. \qquad \square$$

Los lemas anteriores justifican que todos los números del conjunto

$$I = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j : i, j \geq 0 \right\}$$

son introspectivos para todos los polinomios del conjunto

$$P = \left\{ \prod_{a=0}^{l} (x+a)^{e_a} : e_a \geq 0 \right\}.$$

Ahora vamos a definir dos grupos relacionados con estos dos conjuntos que van a tener un papel crucial en la demostración.

El primero de esos grupos va a ser el formado por los elementos de I módulo r. Este grupo es un subgrupo del grupo $\mathbb{Z}_r^*$, ya que $\text{mcd}(r, n) = 1 = \text{mcd}(r, p)$. A este primer grupo lo denotaremos como G, y tomaremos t como su orden, es decir, $|G| = t$. Nótese que en G vamos a tener como mínimo $\text{ord}_r(n)$ elementos. Esto se debe a que si en $\left(\frac{n}{p}\right)^i \cdot p^j$ tomamos $j = i$ sabemos que, para todo $i \in \{1, 2, \ldots, \lfloor \log^2(n) \rfloor\}$, se tiene que $n^i \not\equiv 1 \pmod{r}$, ya que $\text{ord}_r(n) > \log^2(n)$, y además esas potencias de n van a ser todas distintas módulo r. Como $t \geq \text{ord}_r(n)$, entonces $t > \log^2(n)$.

Para definir el segundo grupo utilizaremos polinomios ciclotómicos sobre cuerpos finitos. Tomamos ahora $\Phi_r(x)$, que es el r-ésimo polinomio ciclotómico, sobre F_p. El polinomio $\Phi_r(x)$ divide a $x^r - 1$ por su propia definición y se descompone como factores irreducibles de grado $\text{ord}_r(p)$ en $F_p[x]$ (ver la proposición 2.4). Sea $h(x)$ uno de esos factores irreducibles; como $\text{ord}_r(p) > 1$, el grado de $h(x)$ también es mayor que 1. Nuestro segundo grupo va a sar el conjunto de todos los restos de los polinomios de P módulo $h(x)$ y p, al que llamaremos H, y que está generado por los elementos $\{x, x+1, \ldots, x+l\}$ del cuerpo $F = F_p[x]/(h(x))$. Además, H es un subgrupo del grupo multiplicativo F^*.

Una vez definidos estos dos grupos, vamos a enunciar (pero no a demostrar) un lema que permite acotar el orden del grupo H. La idea del lema es obra de Hendrik Lenstra Jr.

Lema 4.8. Sea H el grupo descrito anteriormente, se cumple que $|H| \geq \binom{t+l}{t-1}$.

El siguiente lema (que tampoco demostraremos) nos va a permitir acotar superiormente el orden de H, en el caso de que n no sea una potencia de p.

Lema 4.9. Si n no es una potencia de p, entonces $|H| \leq n^{\sqrt{t}}$.

(El lector puede encontrar la demostración de estos dos últimos lemas en [1].) Con estas cotas para el orden de H, ya podemos terminar de demostrar que el algoritmo funciona.

Teorema 4.4. Si el algoritmo 2 devuelve PRIMO, entonces n es primo. (Recordemos de nuevo que nos estamos centrando en el caso en el que el algoritmo devuelve PRIMO en el paso 7, puesto que ya hemos visto que funcionaría correctamente si devolviera PRIMO en el paso 4).

Demostración. Supongamos que el algoritmo devuelve PRIMO. Por el lema 4.8, tenemos que $|H| \geq \binom{t+l}{t-1}$. Ahora, como $t > \log^2(n)$, se tiene la siguiente cadena de implicaciones:

$$\begin{aligned} t > \log^2(n) &\Leftrightarrow \sqrt{t} > \log(n) \Leftrightarrow t > \sqrt{t}\log(n) \\ &\Leftrightarrow t > \lfloor \sqrt{t}\log(n) \rfloor \Leftrightarrow t \geq \lfloor \sqrt{t}\log(n) \rfloor + 1, \end{aligned}$$

de donde deducimos que $t \geq \lfloor \sqrt{t}\log(n) \rfloor + 1$. Con esto tenemos que

$$\binom{t+l}{t-1} \geq \binom{\lfloor \sqrt{t}\log(n) \rfloor + 1 + l}{\lfloor \sqrt{t}\log(n) \rfloor}.$$

También sabemos que $l = \lfloor \sqrt{\varphi(r)}\log(n) \rfloor \geq \lfloor \sqrt{t}\log(n) \rfloor$. Esto se debe a que $\varphi(r) \geq t$, ya que G es un subgrupo del grupo $\mathbb{Z}_r^*$, que como hemos visto anteriormente tiene $\varphi(r)$ elementos. Luego

$$\binom{\lfloor \sqrt{t}\log(n) \rfloor + 1 + l}{\lfloor \sqrt{t}\log(n) \rfloor} \geq \binom{2\lfloor \sqrt{t}\log(n) \rfloor + 1}{\lfloor \sqrt{t}\log(n) \rfloor}.$$

Ahora vamos a probar que

$$\binom{2\lfloor \sqrt{t}\log(n) \rfloor + 1}{\lfloor \sqrt{t}\log(n) \rfloor} > 2^{\lfloor \sqrt{t}\log(n) \rfloor + 1}. \tag{4.4}$$

Sea $M = \lfloor \sqrt{t}\log(n) \rfloor$. Por lo visto anteriormente, sabemos que si $n = 2$, $r = 3$ cumpliría las condiciones del algoritmo y el paso 4 de este devolvería PRIMO, luego, como estamos tratando el caso en el que el algoritmo devuelve PRIMO en 7, supondremos que $n > 2$, o, lo que es lo mismo, que $n \geq 3$. Así, $\log^2(n) \geq \log^2(3) = 2.5121\ldots$, por tanto, debido a que $t > \log^2(n)$, se tiene que

$$M = \lfloor \sqrt{t}\log(n) \rfloor \geq \lfloor \log^2(n) \rfloor \geq 2,$$

y finalmente esto nos permite concluir que $M > 1$. Nos centramos ya en probar la desigualdad (4.4). Desarrollando el número combinatorio resulta

$$\begin{aligned} \binom{2M+1}{M} &= \frac{(2M+1)!}{(M+1)! \cdot M!} = \frac{(2M+1)(2M)(2M-1)\cdots(M+1)M!}{(M+1)M(M-1)\cdots 2 \cdot 1 \cdot M!} \\ &= (2M+1)\frac{(2M)(2M-1)\cdots(M+2)}{M(M-1)\cdots 2}. \end{aligned}$$

En el último paso hemos obtenido una fracción con $M-1$ factores en el numerador y otros tantos en el denominador. Agrupándolos convenientemente dos a dos tenemos que $\frac{2M-i}{M-i} \geq 2$, para todo $i \in \{1, 2, \ldots, M-2\}$. Además, como $M > 1$ y M es entero, $M \geq 2$, y de ahí $2M+1 \geq 5$. Así, llegamos a la conclusión de que la fracción obtenida es mayor que 2^{M+1}.

Finalmente hacemos un paso más y, utilizando que $\sqrt{t}\log(n) \leq \lfloor \sqrt{t}\log(n) \rfloor + 1$ y la idea de la observación 4.2, tenemos que

$$2^{\lfloor \sqrt{t}\log(n) \rfloor + 1} \geq 2^{\sqrt{t}\log(n)} = n^{\sqrt{t}}.$$

Por tanto, acabamos de probar que $|H| > n^{\sqrt{t}}$. Por el lema 4.9, $|H| \leq n^{\sqrt{t}}$ si n no es una potencia de p, luego $n = p^k$ para un cierto $k > 0$. Si k fuese mayor que 1 el algoritmo habría devuelto COMPUESTO en el paso 1, luego necesariamente se debe cumplir que $n = p$.

Con esto hemos terminado la demostración de la corrección del algoritmo. □

Existen versiones posteriores de este algoritmo en las que se mejora la precisión de la r elegida y con ello la complejidad del mismo.

Debido a la dificultad práctica que entraña este algoritmo no vamos a mostrar ejemplos detallados del mismo. Si que daremos una implementación en *Mathematica* del test y la probaremos con algunos ejemplos. Se puede hallar en las últimas páginas del apéndice A.

4.3. Otros algoritmos

Como en el caso de los números con formas especiales, nosotros solo hemos expuesto algunos de los diversos algoritmos para todo tipo de números que existen. Hay algunos más que son muy conocidos y que no queremos dejar de mencionar como el APRCL. Este algoritmo fue propuesto inicialmente por Adleman, Pomerance y Rumely, y fue mejorado posteriormente por Cohen y Lenstra, de ahí su nombre. Se puede encontrar en [4].

Otro de los algoritmos más importantes, y que más se utiliza hoy en día para esta labor, es el de las curvas elípticas, denominado ECCP. Ya lo habíamos mencionado anteriormente, cuando comenzamos la explicación del AKS. El lector puede encontrarlo en la sección 6 del capítulo 7 de [9].

Conclusiones

El objetivo de esta memoria era mostrar al lector algunos de los algoritmos deterministas de primalidad más importantes y más utilizados actualmente. Además, aparte de dar explicaciones, implementaciones y ejemplos de algunos tests, también pretendía aportar una bibliografía interesante en la que algún interesado/a pueda obtener más información sobre este tema.

Los números primos tienen una importancia enorme en las matemáticas y a medida que pasa el tiempo se descubren nuevas utilidades de estos números. La criptografía es una materia que tiene un papel muy importante en el mundo hoy en día, y en la que los números primos tienen una función clave, nunca mejor dicho.

En cuanto a mis impresiones tras haber realizado este trabajo, una de las cosas que más me ha sorprendido es la potencia de ciertos tests como el de Lucas-Lehmer. Con un ordenador normal de uso doméstico he podido ver, en segundos, que un número de 13395 cifras, que sería prácticamente imposible memorizar, era primo.

Para terminar solo quiero decir que he disfrutado haciendo este trabajo de fin de grado sobre este tema, y animo a todo el/la que lea estas líneas a investigar más acerca de ello. Aún queda mucho trabajo por hacer en este campo.

Bibliografía

[1] Agrawal, M., Kayal, N., y Saxena, N.: PRIMES is in P, *Ann. of Math. (2)* **160** (2004), no. 2, 781–793.

[2] Apostol, T. M.: *Introduction to analytic number theory*, Springer-Verlag, New York-Heidelberg, 1976.

[3] Berrizbeitia, P.: Pruebas determinísticas de primalidad, *La Gaceta de la RSME* **4** (2001), no. 2, 447–456.

[4] Berrizbeitia, P.: *Algoritmos deterministas de primalidad*, XVII Escuela Venezolana de Matemáticas, Asociación Matemática Venezolana, Centro de Estudios Avanzados, IVIC, Caracas, 2004.

[5] Berrizbeitia, P., y Berry, T. G.: Cubic reciprocity and generalised Lucas-Lehmer tests for primality of $A \cdot 3 \pm 1$, *Proc. Amer. Math. Soc.* **127** (1999), no. 7, 1923–1925.

[6] Berrizbeitia, P., y Berry, T. G.: Biquadratic reciprocity and a Lucasian primality test, *Math. Comp.* **73** (2004), no. 247, 1559–1564.

[7] Bressoud, D. M.: *Factorization and primality testing*, Springer-Verlag, New York, 1989.

[8] Chamizo, F.: *¡Qué bonita es la teoría de Galois!*, Madrid, 2004.

[9] Crandall, R., y Pomerance, C.: *Prime numbers. A computational perspective*, segunda edición, Springer, New York, 2005.

[10] Grau, J. M., Oller-Marcén, A. M., Rodríguez, M., y Sadornil, D.: Fermat test with Gaussian base and Gaussian pseudoprimes, *Czechoslovak Math. J.* **65** (2015), no. 4, 969–982.

[11] Grau, J. M., Oller-Marcén, A. M., y Sadornil, D.: A primality test for $Kp^n + 1$ numbers, *Math. Comp.* **84** (2015), no. 291, 505–512.

[12] Grau, J. M., Oller-Marcén, A. M., y Sadornil, D.: A primality test for $4Kp^n - 1$ numbers, *Monatsh. Math.* **191** (2020), no. 1, 93–101.

[13] Lidl, R., y Niederreiter, H.: *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, 1986.

[14] Riesel, H.: Lucasian criteria for the primality of $N = h \cdot 2^n - 1$, *Math. Comp.* **23** (1969), no. 108, 869–875.

[15] Riesel, H.: *Prime numbers and computer methods for factorization*, segunda edición, Birkhäuser, New York, 2012.

[16] Rödseth, Ö. J.: A note on primality tests for $N = h \cdot 2^n - 1$, *BIT* **34** (1994), no. 3, 451–454.

[17] Sadornil, D., y Varona, J. L.: Existen infinitos primos (desde Euclides hasta el siglo XXI), *La Gaceta de la RSME* **24** (2021), no. 2, 301–324.

[18] Varona, J. L.: *Recorridos por la teoría de números*, segunda edición, Ediciones Electolibris y Real Sociedad Matemática Española, Murcia, 2019.

Apéndice A

Programación de algunos tests y ejemplos

A lo largo del trabajo hemos ido mostrando algunos ejemplos en los que hemos aplicado los tests, pero en esos ejemplos hemos utilizado números bastante pequeños.

En este apéndice vamos a hacer uso del programa *Mathematica* para dar una implementación de algunos algoritmos, verificar las cuentas mostradas en algunos de los ejemplos previos y probar los tests programados con números un poco más grandes. Veremos que algunos tests son bastante más potentes que otros simplemente con el tamaño de los números que cogeremos. Hay algoritmos que con números de 4 cifras ya son bastante lentos, como el test para números de la forma $4K \cdot p^n - 1$, y otros que con números de incluso más de 10000 dígitos aún son muy rápidos y eficientes, como el test de Lucas-Lehmer.

Sin más preámbulos, mostramos el documento a continuación.

Programación de algunos tests y ejemplos

Test de Pépin

Implementación del test de Pépin

Reduciremos módulo F_n cada potencia obtenida para simplificar los cálculos y que el algoritmo sea más rápido.

In[214]:=
```
(*----------------------Test de Pépin--------------------------*)
TestPepin[n_] := Module[{},
   Fn = 2^(2^n) + 1; (* Este es el número de Pépin estudiado. *)
   Print["El número estudiado es: F_", n, "=", Fn, "."];
   potaux = Mod[3^2, Fn];
   limit = 2^n - 1;
   For[i = 2, i ≤ limit, i++,
    potaux = Mod[potaux^2, Fn];
   ];
   If[Mod[potaux + 1, Fn] == 0,
    Print["El número es primo."], Print["El número NO es primo."]];
  ];
```

Ejemplo 1: Tomamos n=5 y estudiamos el número de Fermat F_5.

In[215]:=
```
n = 5;
TestPepin[n];
```

```
El número estudiado es: F_5=4 294 967 297.

El número NO es primo.
```

Ejemplo 2: Tomamos n=6 y estudiamos el número de Fermat F_6.

In[217]:=
```
n = 6;
TestPepin[n];
```

```
El número estudiado es: F_6=18 446 744 073 709 551 617.

El número NO es primo.
```

Ejemplo 3: Tomamos n=7 y estudiamos el número de Fermat F_7.

In[219]:=
```
n = 7;
TestPepin[n];
```

```
El número estudiado es: F_7=340 282 366 920 938 463 463 374 607 431 768 211 457.

El número NO es primo.
```

Ejemplo 4: Tomamos n=8 y estudiamos el número de Fermat F_8.

In[221]:=
```
n = 8;
TestPepin[n];
```

```
El número estudiado es: F_8=
 115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 129 639 937.

El número NO es primo.
```

Ejemplo 5: Tomamos n=9 y estudiamos el número de Fermat F_9.

In[223]:=
```
n = 9;
TestPepin[n];
```

```
El número estudiado es: F_9=
 13 407 807 929 942 597 099 574 024 998 205 846 127 479 365 820 592 393 377 723 561 443 721 764 030 073 546 -
  976 801 874 298 166 903 427 690 031 858 186 486 050 853 753 882 811 946 569 946 433 649 006 084 097.

El número NO es primo.
```

El test de Pépin es bastante eficiente hasta n=15. Para n=16 ya se ralentiza bastante más.

Cálculos del ejemplo mostrado en la sección del test de Pépin.

Vamos a mostrar como hemos obtenido los resultados del ejemplo mostrado en la sección de la explicación del test de Pépin. En primer lugar, calculamos el número de Fermat con n=6.

In[225]:=
```
F6 = 2^(2^6) + 1;
Print[F6];
```

```
18 446 744 073 709 551 617
```

Ahora calculamos las potencias. En todos los casos reducimos módulo F_6. No vamos a mostrar todas las potencias para no alargarnos demasiado. Si alguien está interesado en ver los cálculos al detalle puede utilizar este fragmento de código y descomentar la función Print dentro del For para verlas.

In[227]:=
```
pot0 = Mod[3^(2), F6];
potaux = pot0;
For[i = 2, i ≤ 63, i++,
  potaux = Mod[potaux^2, F6];
  (* Print["3^(2^",i,") ≡ ",potaux]; *)
 ];
Print["3^(2^", i - 1, ") ≡ ", potaux];
```

```
3^(2^63) ≡ 11 860 219 800 640 380 469
```

Test de Proth

Implementación del test de Proth

In[231]:=
```
(*----------------------Test de Proth--------------------------*)
TestProth[k_, n_] := Module[{},
   m = k 2^n + 1; (* Este es el número de Proth estudiado. *)
   Print["El número estudiado es: ", m, "."];
   For[a = 1, a ≤ m - 1, a++, (* Probamos con todos los a posibles en Z_m. *)
    If[Mod[a^((m - 1) / 2) + 1, m] == 0, Return[Print["El número es primo."]],
      (* Comprobamos la condición. Si se cumple, m es primo. *)
      If[JacobiSymbol[a, m] == -1, Return[Print["El número NO es primo."]],];
      (* Si no se cumple la condición, comprobamos si se cumple la condición
       necesaria para el si y solo si. Si se cumple, podremos asegurar que
       el número es compuesto, pero si no tendremos que seguir probando. *)
     ];
   ];
   Return[Print["No se ha podido determinar si el número es primo o no."]];
   (* Si no hemos obtenido un resultado antes,
   entonces no podremos asegurar si el número es primo o no. *)
  ];
```

Ejemplo 1: Tomamos k=5 y n=2, y probamos si el número 21 es primo o no.

In[232]:=
```
k = 5;
n = 2;
TestProth[k, n];
```

```
El número estudiado es: 21.
```

```
El número NO es primo.
```

Ejemplo 2: Tomamos k=3 y n=4, y probamos si el número 49 es primo o no.

In[235]:=
```
k = 3;
n = 4;
TestProth[k, n];
```

```
El número estudiado es: 49.

No se ha podido determinar si el número es primo o no.
```

Ejemplo 3: Tomamos k=27 y n=16, y probamos si el número 1769473 es primo o no.

In[238]:=
```
k = 27;
n = 16;
TestProth[k, n];
```

```
El número estudiado es: 1769473.

El número es primo.
```

Ejemplo 4: Tomamos k=13 y n=20, y probamos si el número 13631489 es primo o no.

In[241]:=
```
k = 13;
n = 20;
TestProth[k, n];
```

```
El número estudiado es: 13631489.

El número es primo.
```

Ejemplo 5: Tomamos k=9111 y n=14, y probamos si el número 149274625 es primo o no.

In[244]:=
```
k = 9111;
n = 14;
TestProth[k, n];
```

```
El número estudiado es: 149274625.

El número NO es primo.
```

Test Kp^n+1

Funciones previas

In[247]:=
```
(* Polinomio ciclotómico de grado p (p primo). *)
Op[x_, p_] := Cyclotomic[p, x];
(* Función para ver que elementos de Z_m no son restos de una potencia p-
 ésima módulo m. *)
NoRestosPotenciaPesima[p_, m_] := Module[{Res},
   Res = List[]; Lrestos = List[];
   (* Construimos la lista Lrestos con
     todos los restos de alguna potencia p-ésima módulo m. *)
   For[i = 0, i ≤ m - 1, i++,
    resto = Mod[i^p, m];
    If[MemberQ[Lrestos, resto], , AppendTo[Lrestos, resto];];
    (* Si el resto ya está en la lista no lo añadimos. *)
   ];
   (* Construimos Res con todos los elementos de Z_m que no estén en Lrestos. *)
   Clear[i];
   For[i = 0, i ≤ m - 1, i++,
    If[MemberQ[Lrestos, i], , AppendTo[Res, i]];
   ];
   Return[Res];
  ];
```

Implementación del test Kp^n+1

In[249]:=
```
(*-------------------------Test Kp^n+1-----------------------------*)
TestKpnMas1[K_, p_, n_] := Module[{},
   m = K p^n + 1; (* Este es el número de la forma Kp^n+1 estudiado. *)
   Print["El número estudiado es: ", m, "."];
   (* Obtenemos la lista de a posibles con la función anterior. *)
   Lposibles = NoRestosPotenciaPesima[p, m]; limit = Length[Lposibles];
   If[limit == 0, Return[
      Print["No se puede aplicar el test a este número debido a que no hay elementos
         de Z_m que no sean restos de una potencia p-ésima módulo m."]];,
    Clear[i];
    For[i = 1, i ≤ limit, i++,
     resto = Mod[Op[Lposibles[[i]]^((m - 1) / p), p], m];
     If[resto == 0,
      Return[Print["El número es primo. Basta tomar a=", Lposibles[[i]], "."]],];
    ];
    Return[Print["El número NO es primo."]];
   ];
  ];
```

Ejemplo 1: Tomamos K=6, p=5 y n=1, luego estudiamos el número m=31.

In[250]:=
```
K = 6;
p = 5;
n = 1;
TestKpnMas1[K, p, n];
```

```
El número estudiado es: 31.

El número es primo. Basta tomar a=2.
```

Ejemplo 2: Tomamos K=2, p=5 y n=3, luego estudiamos el número m=251.

In[254]:=
```
K = 2;
p = 5;
n = 3;
TestKpnMas1[K, p, n];
```

```
El número estudiado es: 251.

El número es primo. Basta tomar a=3.
```

Ejemplo 3: Tomamos K=2, p=5 y n=5, luego estudiamos el número m=6251.

In[258]:=
```
K = 2;
p = 5;
n = 5;
TestKpnMas1[K, p, n];
```

```
El número estudiado es: 6251.

No se puede aplicar el test a este número debido a que no hay
  elementos de Z_m que no sean restos de una potencia p-ésima módulo m.
```

Ejemplo 4: Tomamos K=6, p=11 y n=3, luego estudiamos el número m=7987.

In[262]:=
```
K = 6;
p = 11;
n = 3;
TestKpnMas1[K, p, n];
```

```
El número estudiado es: 7987.

El número NO es primo.
```

Ejemplo 5: Tomamos K=6, p=13 y n=3, luego estudiamos el número m=13183.

In[266]:=
```
K = 6;
p = 13;
n = 3;
TestKpnMas1[K, p, n];
```

```
El número estudiado es: 13183.

El número es primo. Basta tomar a=2.
```

Test 4Kp^n-1

Funciones previas

In[270]:=
```
(* Polinomio ciclotómico de grado p (p primo). *)
Op[x_, p_] := Cyclotomic[p, x];
(* Función que devuelve una lista con los pares a
 y b del grupo G_N descrito en la explicación del test. *)
Gm[m_] := Module[{Res},
   Res = List[]; Clear[a, b];
   (* Hacemos 2 bucles para recorrer todos los valores de Z_m dos veces. *)
   For[a = 0, a ≤ m - 1, a++,
    For[b = 0, b ≤ m - 1, b++,
      If[Mod[a^2 + b^2, m] == 1, AppendTo[Res, {a, b}],];
      (* Si se cumple la condición, añadimos la pareja {a,b} a Res. *)
     ];
   ];
   Return[Res];
  ];
```

Implementación del test 4Kp^n-1

In[272]:=
```
(*-------------------------Test 4Kp^n-1-----------------------------*)
Test4KpnMenos1[K_, p_, n_] := Module[{},
   m = 4 K p^n - 1; (* Este es el número de la forma 4Kp^n-1 estudiado. *)
   Print["El número estudiado es: ", m, "."];
   For[j = 1, j ≤ n, j++,
    Print["Prueba j=", j, ":"];
    If[2 j ≥ Log[p, 4 K] + n, (* Comprobamos si el j cumple
      la segunda condición. Si no la cumple pasamos al siguiente. *)
     G = Gm[m];
     For[it = 1, it ≤ Length[G], it++, (* Para los j que pasen la cota,
      comprobamos si alguno de los valores de Gm sirven para el test. *)
      a = G[[it]][[1]]; (* Recuperamos el valor a del elemento de Gm *)
      b = G[[it]][[2]]; (* Recuperamos el valor b del elemento de Gm *)
      prueba = Op[(a + b I)^(4 K p^(j - 1)), p];
      If[Mod[prueba, m] == 0, (* Comprobamos la otra condición del teorema. *)
       Return[Print["El número es primo. Basta tomar w=", a, "+", b, "i."]];,
      ];
     ];,
     Print["No se cumple la cota del logaritmo."];
    ];
   ];
   Return[Print["No se ha podido determinar si el número es primo o no."]];
  ];
```

Ejemplo 1: Tomamos k=2, p=3 y n=2, luego estudiamos el número 71.

In[273]:=
```
K = 2;
p = 3;
n = 2;
Test4KpnMenos1[K, p, n];
```

```
El número estudiado es: 71.

Prueba j=1:

No se cumple la cota del logaritmo.

Prueba j=2:

El número es primo. Basta tomar w=8+24i.
```

Ejemplo 2: Tomamos K=1, p=3 y n=3, luego estudiamos el número 107.

In[277]:=
```
K = 1;
p = 3;
n = 3;
Test4KpnMenos1[K, p, n];
```

```
El número estudiado es: 107.

Prueba j=1:

No se cumple la cota del logaritmo.

Prueba j=2:

No se cumple la cota del logaritmo.

Prueba j=3:

El número es primo. Basta tomar w=3+45i.
```

Ejemplo 3: Tomamos K=2, p=17 y n=1, luego estudiamos el número 135.

In[281]:=
```
K = 2;
p = 17;
n = 1;
Test4KpnMenos1[K, p, n];
```

```
El número estudiado es: 135.

Prueba j=1:

No se ha podido determinar si el número es primo o no.
```

Ejemplo 4: Tomamos K=3, p=7 y n=2, luego estudiamos el número 587.

In[285]:=
```
K = 3;
p = 7;
n = 2;
Test4KpnMenos1[K, p, n];
```

```
El número estudiado es: 587.

Prueba j=1:

No se cumple la cota del logaritmo.

Prueba j=2:

El número es primo. Basta tomar w=3+173i.
```

Ejemplo 5: Tomamos K=2, p=13 y n=2, luego estudiamos el número 1351 (no es primo, es 7x193).

In[289]:=
```
K = 2;
p = 13;
n = 2;
Test4KpnMenos1[K, p, n];
```

```
El número estudiado es: 1351.

Prueba j=1:

No se cumple la cota del logaritmo.

Prueba j=2:

No se ha podido determinar si el número es primo o no.
```

Ejemplo 6: Tomamos K=7, p=3 y n=4, luego estudiamos el número 2267.

In[293]:=
```
K = 7;
p = 3;
n = 4;
Test4KpnMenos1[K, p, n];
```

```
El número estudiado es: 2267.

Prueba j=1:

No se cumple la cota del logaritmo.

Prueba j=2:

No se cumple la cota del logaritmo.

Prueba j=3:

No se cumple la cota del logaritmo.

Prueba j=4:

El número es primo. Basta tomar w=3+404i.
```

Test de Lucas-Lehmer

Implementación del test de Lucas-Lehmer

In[297]:=
```
(*------------------------Test de Lucas-Lehmer-----------------------------*)
TestLucasLehmer[p_] := Module[{},
   Mp = 2^p - 1; (* Este es el número de Mersenne estudiado. *)
   Print["El número de Mersenne estudiado es: M_", p, "."];
   (* Podríamos mostrar el número, pero en principio no lo vamos
    a hacer porque vamos a probar números bastante grandes. *)
   If[p == 2, Return[Print["El número es primo."]],];
   S0 = 4;
   S = Mod[S0, Mp];
   (* Print["k=0"," S=",S]; *) (* Si queremos podemos mostrar cada S_k. *)
   For[k = 1, k ≤ p - 2, k++,
    S = Mod[S^2 - 2, Mp];
    (* Damos a S_k el valor de S_{k-1}^2-2 reducido módulo Mp. *)
    (* Print["k=",k," S=",S]; *) (* Si queremos podemos mostrar cada S_k. *)
   ];
   If[Mod[S, Mp] == 0, Return[Print["El número es primo."]],
    Return[Print["El número NO es primo."]]];
  ];
```

Ejemplo 1: Tomamos p=7993, de modo que el número de Mersenne obtenido es M_7993. Este número posee 2407 dígitos.

In[298]:=
```
p = 7993;
TestLucasLehmer[p];
```

```
El número de Mersenne estudiado es: M_7993.

El número NO es primo.
```

Ejemplo 2: Tomamos p=11213, de modo que el número de Mersenne obtenido es M_11213. Este número posee 3376 dígitos.

In[300]:=
```
p = 11 213;
TestLucasLehmer[p];
```

```
El número de Mersenne estudiado es: M_11 213.

El número es primo.
```

Ejemplo 3: Tomamos p=21701, de modo que el número de Mersenne obtenido es M_21701. Este número posee 6533 dígitos.

In[302]:=
```
p = 21701;
TestLucasLehmer[p];
```

```
El número de Mersenne estudiado es: M_21701.

El número es primo.
```

Ejemplo 4: Tomamos p=23209, de modo que el número de Mersenne obtenido es M_23209. Este número posee 6987 dígitos.

In[304]:=
```
p = 23209;
TestLucasLehmer[p];
```

```
El número de Mersenne estudiado es: M_23209.

El número es primo.
```

Estos cálculos los ha hecho un portátil de uso doméstico de manera casi inmediata, y aún podríamos meterle números más altos y haría los cálculos (aunque ya tardando más tiempo). Esto nos da una idea de la potencia de este algoritmo. (Para los interesados/as, hasta p=44497, que es el próximo primo para el que su número de Mersenne asociado es primo, lo hace bien pero ya de forma bastante más lenta. El problema es que el siguiente primo de Mersenne es el M_86243, y conforme aumentamos p desde 44497 el tiempo de espera va muy en aumento. No en vano, el M_44497 tiene 13395 cifras.)

Test n-1

Funciones previas

Función que devuelve una lista con todos los factores primos de un número.

In[306]:=
```
PrimosDivisores[p_] :=
  Module[{L}, L = List[]; Laux = FactorInteger[p];
   For[i = 1, i <= Length[Laux], i++,
    AppendTo[L, Laux[[i]][[1]]];
   ];
   Return[L];
  ];
```

Función que devuelve el convergente de la fracción continua de c1/F que verifica que el denominador (que denotamos por v) es el máximo valor que cumple que v<F^2/(n^(1/2)). Esta implementación hace uso de las sucesiones recurrentes que permiten calcular los convergentes k-ésimos de una fracción continua, en lugar de las funciones predefinidas por Mathematica.

In[307]:=
```
ConvergenteFraccionContinua[F_, R_, c1_] := Module[{Res},
  Clear[Res]; Res = List[];
  r0 = c1; r1 = F; rD = r0; rd = r1;
  n = F R + 1;
  Clear[L]; L = List[];
  While[rd ≠ 0,
   AppendTo[L, Quotient[rD, rd]];
   raux = rd; (* Para no perder el valor. *)
   rd = Mod[rD, rd]; (* El próximo divisor es el resto de la división actual. *)
   rD = raux; (* El próximo dividendo es el cociente de la división actual. *)
  ];
  (* L es la lista de los q_k de las fracciones continuas. *)
  q[k_] := L[[k + 1]];
  PMenos1 = 1; P0 = q[0]; P = P0; PkMenos1 = P0; PkMenos2 = PMenos1;
  QMenos1 = 0; Q0 = 1; Q = Q0; QkMenos1 = Q0; QkMenos2 = QMenos1;
  Clear[k];
  k = 1;
  (* Iniciamos el valor de k a 1 porque la recurrencia esta definida para k≥1. *)
  bool = True;
  (* Hacemos el bucle hasta Length[L]-1, que es donde está el últimto q_k. *)
  While[k <= Length[L] - 1 && bool == True,
   Paux = P;
   P = q[k] PkMenos1 + PkMenos2;
   PkMenos2 = PkMenos1;
   PkMenos1 = Paux;
   Qaux = Q;
   Q = q[k] QkMenos1 + QkMenos2;
   QkMenos2 = QkMenos1;
   QkMenos1 = Qaux;
   If[Q < F^2 / (n^(1 / 2)), ,
    bool = False;
    P = PkMenos1;
    Q = QkMenos1;
   ];
   k++;
  ];
  AppendTo[Res, P];
  AppendTo[Res, Q];
  Return[Res];
 ];
```

Función que devuelve el convergente de la fracción continua de c1/F que verifica que el denominador (que denotamos por v) es el máximo valor que cumple que v<F^2/(n^(1/2)). En esta versión de la función hemos utilizado las funciones predefinidas por Mathematica. Ambas funciones producen el mismo resultado y, aunque esta es más sencilla, no queríamos dejar de ofrecer otra variante de la misma al lector.

In[308]:=
```
ConvergenteFraccionContinua[F_, R_, c1_] := Module[{Res},
  Clear[Res]; Res = List[];
  n = F R + 1;
  k = 0; (* Iniciamos el valor de k a 0 porque los convergentes empiezan en 0. *)
  bool = True;
  While[bool == True, (* Los convergentes empiezan en 0,
   pero la función de Mathematica empieza en 1. Por eso usamos k+1. *)
   vaux = Denominator[Convergents[c1 / F, k + 1][[k + 1]]];
   (* Obtenemos el denominador del siguiente convergente. *)
   If[vaux < F^2 / (n^(1 / 2)),
    (* Comprobamos si cumple la restricción. Si la cumple, actualizamos u y v. *)
    u = Numerator[Convergents[c1 / F, k + 1][[k + 1]]];
    v = Denominator[Convergents[c1 / F, k + 1][[k + 1]]];
    If[u / v == c1 / F, bool = False,];,
    (* Si el convergente obtenido es la propia fracción, paramos. *)
    bool = False; (* Si no se cumple la condición del denominador,
    no actualizamos u y v y paramos. *)
   ];
   k++;
  ];
  AppendTo[Res, u];
  AppendTo[Res, v];
  Return[Res]; (* Finalmente, devolvemos la lista con u y v. *)
 ];
```

Implementación del test n-1:

In[309]:=
```
(*--------------------------------Test n-1---------------------------------*)
TestnMenos1[F_, R_] :=
  Module[{},
   n = F R + 1; (* Este es el número estudiado. *)
   Print["El número estudiado es n=", n, "."];
   (* 1. Test de Pocklington------------------------------------------------ *)
   goto = False; (* Booleano para ver si hay que buscar otro valor de a. *)
   For[a = 2, a ≤ n - 2, a++,
    If[Mod[a^(n - 1) - 1, n] ≠ 0, Return[Print["NO"]],];
    L = PrimosDivisores[F];
    i = 1;
    While[i <= Length[L] && goto == False,
     g = GCD[Mod[a^((n - 1) / L[[i]]), n] - 1, n];
     (* Reducimos módulo n para simplificar los cálculos. *)
     (* g=GCD[a^((n-1)/L[[i]])-1,n]; *)
     If[1 < g && g < n, Return[Print["NO"]],];
     If[g == n, goto = True,];
     i++;
    ];
   ];
```

```
(* 2. Test con la primera cota------------------------------------------- *)
If[F ≥ n^(1 / 2), Return[Print["YES"]],];
(* 3. Test con la segunda cota------------------------------------------- *)
If[n^(1 / 3) <= F < n^(1 / 2),
 Clear[c2, c1];
 (* Vamos a buscar la representación en la base F de n. *)
 L = Solve[{n == c2 F^2 + c1 F + 1 && c2 ≤ F - 1 && c2 ≥ 0 && c1 ≤ F - 1 && c1 ≥ 0},
   {c2, c1}, Integers];
 L = Flatten[L];
 c2 = c2 /. L[[1]];
 c1 = c1 /. L[[2]];
 If[IntegerQ[Sqrt[(c1^2 - 4 c2)]] == False,
  Return[Print["YES"]], Return[Print["NO"]]];
];
(* 3. Test con la tercera cota------------------------------------------- *)
If[n^(3 / 10) <= F < n^(1 / 3),
 Clear[c4, c3, c2, c1];
 L = Solve[{n == c3 F^3 + c2 F^2 + c1 F + 1 && c3 ≤ F - 1 && c3 ≥ 0 &&
     c2 ≤ F - 1 && c2 ≥ 0 && c1 ≤ F - 1 && c1 ≥ 0}, {c3, c2, c1}, Integers];
 L = Flatten[L];
 c3 = c3 /. L[[1]];
 c2 = c2 /. L[[2]];
 c1 = c1 /. L[[3]];
 c4 = c3 F + c2;
 (* Comprobamos si se cumple I. *)
 boolI = True; t = 1;
 While[t ≤ 5 && boolI,
  If[IntegerQ[Sqrt[(c1 + t F)^2 + 4 t - 4 c4]] == True, boolI = False,];
  t++;
 ];
 (* Comprobamos si se cumple II. *)
 boolII = True;
 Frac = ConvergenteFraccionContinua[F, R, c1];
 u = Frac[[1]]; v = Frac[[2]];
 d = Floor[c4 v / F + 1 / 2];
 Clear[x];
 Solu = Solve[{v x^3 + (u F - c1 v) x^2 + (c4 v - d F + u) x - d == 0}, x, Integers];
 (* Analizamos las raíces enteras del polinomio a estudiar. *)
 i = 1;
 While[i ≤ Length[Solu] && boolII == True,
  xi = x /. Solu[[i]];
  If[xi F + 1 ≠ 1, boolII = False,];
  i++;
 ];
 If[boolI == True && boolII == True, Return[Print["YES"]], Return[Print["NO"]]];
];
Return[Print["Tamaño de F insuficiente. "]];
];
```

Ejemplo 1: Tomamos F=9 y R=26, y estudiamos el número 235.

In[310]:=
```
F = 9;
R = 26;
TestnMenos1[F, R];
```
```
El número estudiado es n=235.

NO
```

Ejemplo 2: Tomamos F=42 y R=35, y estudiamos el número 1471.

In[313]:=
```
F = 42;
R = 35;
TestnMenos1[F, R];
```
```
El número estudiado es n=1471.

YES
```

Ejemplo 3: Tomamos F=23 y R=434, y estudiamos el número 9983 (no es primo, es 67x149).

In[316]:=
```
F = 23;
R = 434;
TestnMenos1[F, R];
```
```
El número estudiado es n=9983.

NO
```

Ejemplo 4: Tomamos F=20 y R=539, y estudiamos el número 10781.

In[319]:=
```
F = 20;
R = 539;
TestnMenos1[F, R];
```
```
El número estudiado es n=10781.

YES
```

Ejemplo 5: Tomamos F=11 y R=980, y estudiamos el número 10781 de nuevo. En este caso, F es menor que n^(3/10), luego el algoritmo no puede devolver un resultado concreto.

In[322]:=
```
F = 11;
R = 980;
TestnMenos1[F, R];
```
```
El número estudiado es n=10781.

Tamaño de F insuficiente.
```

Test de Wilson

Implementación del test de Wilson:

In[325]:=
```
(*--------------------------Test de Wilson-----------------------------*)
TestWilson[p_] := Module[{},
   Print["El número estudiado es: p=", p, "."];
   Print["(p-1)!=", (p - 1)!];
   If[Mod[(p - 1)! + 1, p] == 0,
    Print["El número ", p, " es primo."], Print["El número ", p, " NO es primo."]];
  ];
```

Ejemplo 1: Tomamos p=12 y aplicamos el test.

In[326]:=
```
p = 17; (* Este es el número estudiado *)
TestWilson[p];
```

```
El número estudiado es: p=17.

(p-1)!=20 922 789 888 000

El número 17 es primo.
```

Ejemplo 2: Tomamos p=17 y aplicamos el test.

In[328]:=
```
p = 22; (* Este es el número estudiado *)
TestWilson[p];
```

```
El número estudiado es: p=22.

(p-1)!=51 090 942 171 709 440 000

El número 22 NO es primo.
```

Ejemplo 3: Tomamos p=83 y aplicamos el test.

In[330]:=
```
p = 83; (* Este es el número estudiado *)
TestWilson[p];
```

```
El número estudiado es: p=83.

(p-1)!=
 475 364 333 701 284 174 842 138 206 989 404 946 643 813 294 067 993 328 617 160 934 076 743 994 734 899 148 -
   613 007 131 808 479 167 119 360 000 000 000 000 000 000

El número 83 es primo.
```

Algoritmo AKS

Funciones previas

Función que nos devuelve la r necesaria para aplicar el algoritmo. Sabemos que existe una r que verifica que r>=2 y que r es menor o igual que el máximo entre 3 y la función techo del logaritmo de n a la quinta; así que vamos a probar para esos r posibles cual es el menor que verifica que el ord_r(n)>log^2(n). (Recordemos que aquí el log es en base 2.)

In[332]:=
```
rAKS[n_] := Module[{r},
   r = 2;
   For[r = 2, r ≤ Ceiling[Log2[n]^5], r++,
    For[k = Ceiling[Log2[n]^2], k ≤ r - 1, k++, (* Como k debe ser
       mayor que log^2(n) probamos con los k mayores que ese valor. *)
      If[Mod[n^k - 1, r] == 0 , Return[r],];
     ];
   ];
  ];
```

Implementación del algoritmo AKS:

In[333]:=
```
(*------------------------Algoritmo AKS-----------------------------*)
AlgoritmoAKS[n_] := Module[{},
   (* El número n es el número estudiado. *)
   Print["El número estudiado es: ", n, "."];
   a = 2; boola = True; boolb = True;
   If[n == 2, Return[Print["PRIMO"]],];
   (*Paso 1.-----------------------------------------------------------*)
   While[boola == True,
    b = 2;
    While[boolb == True,
     If[n == a^b, Return[Print["COMPUESTO"]],];
     If[n < a^b, boolb = False,];
     b++;
    ];
    a++;
    If[a == n, boola = False,];
   ];
   (*Paso 2.-----------------------------------------------------------*)
   r = rAKS[n];
   (*Print["r=",r];*)
   (*Paso 3.-----------------------------------------------------------*)
   For[a = 2, a ≤ r, a++,
    If[GCD[a, n] > 1 && GCD[a, n] < n, Return[Print["COMPUESTO"]],];
   ];
   (*Paso 4.-----------------------------------------------------------*)
   If[n ≤ r, Return[Print["PRIMO"]],];
   (*Pasos 5 y 6.-------------------------------------------------------------*)
   limit = Floor[Sqrt[EulerPhi[n]] Log2[n]];
   For[a = 1, a ≤ limit, a++,
    If[PolynomialMod[PolynomialMod[(x + a)^n - x^n - a, x^r - 1], n] != 0,
      Return[Print["COMPUESTO"]],];
   ];
   (*Paso 7.-----------------------------------------------------------*)
   Return[Print["PRIMO"]];
  ];
```

Ejemplo 1: Tomamos n=7 y aplicamos el algoritmo.

In[334]:=
```
n = 7;
AlgoritmoAKS[n];
```

```
El número estudiado es: 7.

PRIMO
```

Ejemplo 2: Tomamos n=53 y aplicamos el algoritmo.

In[336]:=
```
n = 53;
AlgoritmoAKS[n];
```
```
El número estudiado es: 53.

PRIMO
```

Ejemplo 3: Tomamos n=91 y aplicamos el algoritmo.

In[338]:=
```
n = 91;
AlgoritmoAKS[n];
```
```
El número estudiado es: 91.

COMPUESTO
```

Ejemplo 4: Tomamos n=107 y aplicamos el algoritmo.

In[340]:=
```
n = 107;
AlgoritmoAKS[n];
```
```
El número estudiado es: 107.

PRIMO
```

Ejemplo 5: Tomamos n=233 y aplicamos el algoritmo.

In[342]:=
```
n = 233;
AlgoritmoAKS[n];
```
```
El número estudiado es: 233.

PRIMO
```