Universidad de Valladolid

Facultad de Ciencias

Trabajo de Fin de Grado

Grado en MATEMÁTICAS

El problema de primalidad está en P

Autor: Samuel González Pastor

Tutor: Félix Delgado de la Mata

Curso 2021/2022

Índice general

Introducción

Los números primos han sido, y son en la actualidad, una de las piezas maestras de las matemáticas, se conocen desde hace milenios y siguen suscitando un enorme interés. Este interés va desde el reto matemático que suponen problemas que siguen abiertos, como la conjetura de Riemann, hasta los derivados de sus aplicaciones en desarrollos fundamentales para nuestra vida cotidiana, como es la criptografía. Uno de los problemas más sencillos de enunciar, pero que resistió durante bastante tiempo los intentos de la comunidad científica, es el problema de saber si un número dado es o no un número primo de manera eficiente, es decir, mediante un algoritmo de complejidad polinomial. Evidentemente la propia definición de número primo proporciona un algoritmo que nos da una respuesta segura al problema, pero el método es impracticable cuando el número es muy grande ya que el tiempo de ejecución hace que sea completamente inservible en la práctica. Este problema se popularizó bajo el nombre de **PRIMES is in P?**. El objetivo de este trabajo es exponer, de manera lo más autocontenida posible, el algoritmo de primalidad de complejidad polinomial desarrollado por Agrawal, Kayak y Saxena en 2002.

El ejemplo del problema de primalidad es, de hecho, un paradigma del desarrollo renovado de la Teoría de Números a partir de la necesidad de incorporar a la misma de forma decisiva el desarrollo de algoritmos eficientes que permitan su uso práctico. Sin ellos, los resultados no son susceptibles de uso. Incorporar la componente algorítmica/eficiente a resultados clásicos de la teoría, haciendo estos resultados útiles computacionalmente, no es una moda, es una necesidad en numerosos ámbitos, especialmente en la transmisión digital de la información. Por un lado se requieren métodos que tienen que ser eficientes computacionalmente para que las operaciones "autorizadas" sean rápidas. Por otro, la propia seguridad en la transmisión reposa en la existencia de problemas de "una vía", es decir, problemas que en una dirección son sencillos computacionalmente, pero en su vía inversa son intratables desde el punto de vista computacional como medio de garantizar que las

operaciones "no autorizadas" sean imposibles en la práctica. Como ejemplo sirve la multiplicación de enteros: sin duda se trata de una operación rápida y sencilla computacionalmente, pero su problema inverso, dado un entero n factorizarlo, es un problema intratable para enteros muy grandes. Éste es el problema en el que se basa el primer sistema criptográfico desarrollado: el RSA (iniciales de sus artífices: Rivest, Shamir y Adleman).

Numerosos métodos criptográficos precisan del uso de números primos de gran tamaño y, en consecuencia, de algoritmos que permitan decidir de forma eficiente si un número es o no primo. Desde el punto de vista práctico el problema se resolvió a partir del uso de algoritmos probabilísticos muy eficientes, los más conocidos son el algoritmo de Rabin-Miller y el algoritmo de Solovay-Strassen, que permiten decidir con rapidez si un número es primo con una probabilidad de error tan pequeña como queramos. No obstante el problema de saber si hay algoritmos deterministas, seguros y eficientes, es decir, **PRIMES is in P?**, permaneció como uno de los problemas abiertos más conocidos durante varias décadas. En él trabajaron numerosos especialistas, tanto del ámbito de las matemáticas como del de la computación, hasta que en 2002, y de forma en cierto modo sorprendente, los matemáticos indios Manindra Agrawal, Neeraj Kayal y Nitin Saxena publicaron el algoritmo que se conoce como AKS resolviendo el problema de forma positiva.

Este Trabajo de Fin de Grado se centra en el estudio de dicho algoritmo AKS. El desarrollo del mismo se lleva a cabo de la forma más autocontenida posible en el capítulo 2 de la memoria. En el mismo, el énfasis se ha puesto en la comprensión del método y de las matemáticas que intervienen en él más que en exponer la versión más rápida del mismo. El algoritmo AKS inicialmente tenía un orden de complejidad de grado 12, posteriormente dicho grado se rebajó considerablemente, de hecho, bajo ciertas hipótesis consideradas aceptables, hay versiones del mismo que reducen el orden de complejidad a 6. No obstante, sigue siendo un algoritmo "lento" comparado con los métodos probabilistas (como Rabin-Miller) que siguen siendo los métodos usados en la práctica. Es por ello que el foco en el trabajo no se ha puesto en ofrecer la mejor versión (que requiere desarrollos matemático/computacionales más complicados), si no en el hecho de que el método es eficiente (polinómico) y por tanto resuelve el problema.

El Capítulo primero de la memoria de dedica a exponer el problema y los algoritmos probablistas más importantes y utilizados. También se incluye el método algorítmico de resolución del problema de búsqueda de números primos de un tamaño prefijado, incluso con algunas condiciones adicionales

necesarias para que los primos sean adecuados a los sistemas criptográficos. Estos resultados hacen uso del Teorema de densidad de los números primos y del Teorema de Dirichlet de la densidad de los primos en progresiones aritméticas.

La memoria incluye un apéndice en el que, de forma muy resumida, se exponen nociones básicas de la teoría de la complejidad de los algoritmos y de las notaciones habituales en la misma. Dada la naturaleza del problema central de la memoria nos ha parecido imprescindible incluir las ideas principales, aunque sin profundizar en lo que sería un tratamiento sistemático de la teoría de la complejidad.

El desarrollo del trabajo se ha basado sobre todo en el artículo original de Agrawal, Kayal y Saxena ([1]) y en el artículo de Granville ([6]), pero el desarrollo concreto de la la presentación se basa en el libro de Rempe-Gillen y Waldecker; "Primality Testing for Beginners" ([8]).

Capítulo 1

Test de primalidad

En este capítulo vamos a tratar los métodos que se pueden usar para abordar el problema planteado como base del trabajo, es decir, los test de primalidad. Pero antes de eso, vamos a dar algunos resultados previos y generales que consideramos importantes tanto en este capítulo como en lo siguiente.

Además, se presentan los diferentes tipos de test y se estudia con detalle el test de Rabin-Miller, debido a que es uno de los más usados hoy en día en la práctica aun siendo un test probabilístico. También se hablará del test de Solovay-Strassen, pero de manera más somera, debido también a su mayor complejidad.

Para finalizar, se tocará un poco otro problema relacionado con los números primos.

1.1. Nociones y resultados previos

Definición 1. Sea $n \geq 2$ y a un número entero tal que $mcd(a, n) = 1$. Se llama orden de a módulo n y se denota por $ord_n(a)$ al menor número natural k tal que $a^k \equiv 1 \bmod(n)$.

El pequeño teorema de Fermat es uno de los resultados básicos y más importantes sobre el que se apoya gran parte de los razonamientos empleados a lo largo del trabajo, pero ya no sólo aquí. Es un resultado clave en Teoría de Números. Lo presentamos a continuación.

Teorema 1 (Pequeño Teorema de Fermat)**.** *Sea p un número primo y a un número entero, entonces se da la congruencia siguiente:*

$$a^p \equiv a \ mod(p).$$

En particular, si $mcd(a,p) = 1$, entonces

$$a^{p-1} \equiv 1 \ mod(p).$$

Otro resultado importante relacionado con las congruencias es la llamada congruencia de Euler, aunque ahí aparece la función φ de Euler, la cuál procedemos a definir antes del teorema.

Definición 2. Sea $n \geq 2$, se denota por $\varphi(n)$ al número de elementos a de $\{1, 2, \ldots, n-1\}$ tales que $mcd(a,n) = 1$.

Teorema 2. *Sea $n \geq 2$, entonces para todo a tal que $mcd(a,n) = 1$, se tiene*

$$a^{\varphi(n)} \equiv 1 \ mod(n).$$

Veamos ahora el teorema chino de los restos.

Teorema 3. *Sean dos números naturales $p, q \geq 2$ tales que $mcd(p,q) = 1$ y sea $n := p \cdot q$. Entonces dos enteros cualesquiera a, b son congruentes módulo n si, y sólo si, son congruentes módulo p y módulo q. Además, para dos enteros cualesquiera c, d, existe x entero tal que $x \equiv c \ mod(p)$ y $x \equiv d \ mod(q)$.*

Observación 1. A lo largo del trabajo, salvo que se diga lo contrario, la notación *log* implicará logaritmo en base 2. Aunque en realidad, se podría trabajar con cualquier base, ya que las relaciones entre ellos vienen determinadas por constantes, y cuando se habla de la complejidad, ya hay diferentes constantes implícitas.

1.2. Test de primalidad

Definición 3. Un test de primalidad es un algoritmo que se usa para determinar si un número es primo o compuesto.

Una primera opción, y que es la más natural, es la de usar la definición de número primo. Es decir, podemos ir dividiendo al número con el que estamos trabajando por todos los números enteros hasta cierta cota, viendo simplemente si alguno de ellos es divisor.

Teorema 4. *Sea n un número natural tal que $n \geq 2$. Entonces, n es primo si, y sólo si, no existe ningún número natural m, con $m \leq \sqrt{n}$, que divida a n.*

Como primera opción puede ser interesante, pero a medida que se trabajan con números más y más grandes, no tiene sentido, en cuánto a coste computacional se refiere, el utilizarlo. De hecho, es un ejemplo elemental de un algoritmo de complejidad exponencial.

Es por ello, que se han ido obteniendo distintos resultados basados en propiedades de los números primos, para así facilitar el trabajo. Ya que el coste computacional se reduce y así se obtienen algoritmos eficientes.

Definición 4. Sea n un número natural mayor que 1, y a un número tal que $mcd(a, n) = 1$ entonces se dice que n es un pseudoprimo respecto de la base a si $a^{n-1} \equiv 1 \bmod(n)$. Dicho número n es un pseudoprimo si lo es respecto de toda base a.

Como podemos observar, la base de la definición es el pequeño teorema de Fermat, sólo que, no todos los pseudoprimos van a ser primos, que es lo que nos gustaría (es decir, que el recíproco del pequeño teorema de Fermat fuera cierto).

Habiendo visto estas definición, vamos a presentar un test de primalidad probabilístico (debido a la elección aleatoria que hay que hacer para la base a), usando el pequeño teorema de Fermat.

TEST DE FERMAT:
Input: un número entero impar $n > 1$.

1. Se escoge un número a entre 1 y $n - 1$ de manera aleatoria.
2. Si $mcd(a, n) \neq 1$, se devuelve 0.
3. Si no, se calcula $b = a^{n-1}$ módulo n.
4. Si $b = 1$, la respuesta del algoritmo es 1.
5. Si no, se devuelve 0.

Cabe remarcar que el algoritmo es eficiente. El paso 2 consiste en calcular el máximo común divisor de dos números, lo cuál se puede hacer de manera

eficiente usando el algoritmo de Euclides. El paso 3 es hacer lo que se llama potenciación modular. Para llevar a cabo esto, existe un método de eficiente, el llamado algoritmo de cuadrados repetidos, que consiste en trabajar con el exponente en base 2, e ir calculando cuadrados módulo n en función de dicha descomposición del exponente.

Si se obtiene como output un 0, entonces se sabe con seguridad que el número es compuesto, es consecuencia directa del pequeño teorema de Fermat. En cambio, si se da la congruencia, tampoco vamos a poder asegurar nada, ya que hay que tener en cuenta la existencia de los llamados números de Carmichael. Es por eso por tanto que el test de Fermat no va a ser válido si no se tienen hipótesis adicionales.

Definición 5. Un número n se dice que es de Carmichael si no es primo, y si $a^{n-1} \equiv 1 \bmod(n)$, para toda base $a \in \mathbb{Z}$.

Ejemplo 1. El número de Carmichael más pequeño que existe es el 561, el cuál podemos escribir como producto, $561 = 3 \cdot 11 \cdot 17$.

Entre otras propiedades, se puede ver que hay infinitos números de Carmichael, y que cada uno de ellos es divisible por al menos tres primos distintos, como es el caso del ejemplo anterior.

Observación 2. El test de Fermat sería en consecuencia la base de un algoritmo eficiente de primalidad salvo por el hecho de que n puede ser de Carmichael. Por tanto, el algoritmo sólo es correcto bajo la hipótesis de que n no es de Carmichael.

Observación 3. Si se supone que n no es de Carmichael y que no es primo, entonces se puede ver que n es pseudoprimo respecto de menos de la mitad de las bases a.
Por tanto, si n no es de Carmichael, cuando el test de Fermat diga que un número es primo, será con probabilidad de error menor de $\frac{1}{2}$.

Dada la invalidez de lo visto hasta ahora, en este capítulo vamos a tratar dos test de primalidad probabilísticos que sí que van a funcionar.

1.3. Test de Rabin-Miller

El test de primalidad de Rabin-Miller es un test probabilístico de los más usados actualmente, ya que, aunque sea probabilístico, se puede acotar

la probabilidad de error tanto como se quiera.

Fue en 1975 cuando Miller creó un test de primalidad determinista eficiente, pero suponiendo que la Hipótesis de Riemann era cierta. Tras unas modificaciones, Rabin posteriormente logró llegar a un test de primalidad eficiente, pero en este caso probabilístico (de tipo Montecarlo), siendo éste el test de Rabin-Miller.

Vamos por tanto, a presentar las ideas generales del mismo, viendo los resultados pertinentes.

Los dos resultados principales sobre los cuáles se va a sostener el test de Rabin-Miller son los dos siguientes.

Lema 1. *Sea p un número primo y a un entero tal que $a^2 \equiv 1$ mod(p), entonces $a \equiv 1$ mod(p), ó, $a \equiv -1$ mod(p).*

Demostración. La primera condición implica que p divide a $a^2 - 1$, pero hay que remarcar que $a^2 - 1 = (a-1)(a+1)$. Luego, ó bien, necesariamente p divide a $a-1$ ó a $a+1$. Dicho de otra forma, $a \equiv 1 \bmod(p)$, ó $a \equiv -1 \bmod(p)$, por lo que queda demostrado.

□

Lema 2. *Sea $n = q \cdot r$, con $q, r > 2$ impares y tales que $mcd(q, r) = 1$. Entonces existe un número natural a tal que $mcd(a, n) = 1$ con $a^2 \equiv 1$ mod(n) pero con $a \not\equiv 1$ mod(n) y $a \not\equiv -1$ mod(n).*

Demostración. Sabemos que por el teorema chino de los restos, existe un número a (con $mcd(a, n) = 1$) tal que se puede escribir $a \equiv 1 \bmod(q)$ y $a \equiv -1 \bmod(r)$. Pero, si se da esto, entonces también por el teorema chino de los restos, no se da ni $a \equiv 1 \bmod(n)$, ni $a \equiv -1 \bmod(n)$. Ahora bien, podemos elevar al cuadrado en las congruencias y obtendríamos que $a^2 \equiv 1 \bmod(q)$, y $a^2 \equiv 1 \bmod(r)$, y por lo que también se tiene que $a^2 \equiv 1 \bmod(n)$.

□

Teorema 5. *Sea p un primo impar, escribimos $p = 2^s \cdot d + 1$, con d impar, $s \geq 1$. Sea a un número natural mayor que 1 cumpliendo que $mcd(a, p) = 1$. Entonces, ó bien $a^d \equiv 1$ mod(p), ó bien, $a^{d \cdot 2^r} \equiv -1$ mod(p) para algún r perteneciente a $\{0, 1, 2, \ldots, s-1\}$.*

Demostración. Lo primero de todo, vamos a hacer un uso directo del pequeño teorema de Fermat, pudiendo así escribir que $a^{n-1} \equiv 1 \bmod(n)$. Podemos sustituir $n-1$ por $2^s \cdot d$ y tendríamos por tanto que $a^{2^s \cdot d} \equiv 1 \bmod(n)$.
Nos remitimos para la continuación de la demostración al lema previo. La congruencia anterior se puede escribir como $(a^{2^{s-1} \cdot d})^2 \equiv 1 \bmod(n)$. Luego, tenemos que $a^{2^{s-1} \cdot d} \equiv 1 \bmod(n)$, ó $a^{2^{s-1} \cdot d} \equiv -1 \bmod(n)$. Si estamos en el segundo caso, hemos terminado, pero si nos encontramos en el otro, entonces podemos repetir el mismo razonamiento, y tendríamos que, $a^{2^{s-2} \cdot d} \equiv 1 \bmod(n)$, ó $a^{2^{s-2} \cdot d} \equiv -1 \bmod(n)$.
Aplicando esto, el peor de los casos en el que nos podríamos encontrar es siempre estar obteniendo 1 en las congruencias y entonces, en este caso, nos encontraríamos con que $a^d \equiv 1 \bmod(n)$ ó $a^d \equiv -1 \bmod(n)$. Cualquiera de los dos casos está en lo posible que enuncia el teorema.

□

A continuación, vamos a ver el resultado fundamental que es el que nos va a dar la probabilidad que vamos a tener de error al aplicar el algoritmo, aunque necesitamos antes de una definición.

Definición 6. Sea n un número impar tal que $n = 2^s \cdot d+1$, donde d es impar. Sea a tal que $mcd(a, n) = 1$, se dice que n es un pseudoprimo fuerte respecto de la base a, si la sucesión $(a^d, a^{2 \cdot d}, \ldots, a^{2^s \cdot d})$ está formada por todos los términos iguales a 1 ó uno de ellos (anterior al último) es -1 (en este último caso, todos los términos posteriores de la sucesión son 1).
Se dice que n es un pseudoprimo fuerte si lo es respecto de toda base a tal que $mcd(a, n) = 1$.

Teorema 6. *Un número n es pseudoprimo fuerte si, y sólo si, es primo.*

Demostración. Es consecuencia directa de los lemas 1 y 2.

□

Teorema 7. *Sea n un número impar que no es primo. Entonces, en el conjunto $\{1, \ldots n-1\}$ hay lo sumo $\frac{\varphi(n)}{4}$ bases (que llamaremos a) tales que $mcd(a, n) = 1$, y que n es pseudoprimo fuerte respecto de a.*

Demostración. Queremos saber el número de bases a para las cuáles n es un pseudoprimo fuerte respecto de dicha base (dicho conjunto lo denotaremos por P).
Nótese que si $a^d \equiv 1 \bmod(n)$, entonces $(-a)^d \equiv -1 \bmod(n)$, dado que d es impar.

Definamos k como el máximo de $\{0, \ldots, s-1\}$ tal que existe una base a con $mcd(a, n) = 1$ y $a^{2^k d} \equiv -1 \bmod(n)$. Y sea $m = 2^k d$. Podemos suponer que dicha base existe ya que si no P tiene cardinal 0 y no hay nada que hacer.
Ahora, podemos escribir también n factorizado, es decir, $n = p_1^{e_1} \cdots p_l^{e_l}$, donde los p_i son primos y distintos dos a dos.
Y definimos los siguientes subgrupos de $\mathbb{Z}_n^*$ (grupo de unidades de $\mathbb{Z}_n$):

$$F := \{a \in \mathbb{Z}_n^* : a^{n-1} \equiv 1 \bmod(n)\}.$$

$$L := \{a < n : a^m \equiv \pm 1 \bmod(p_i^{e_i}), \forall i \in \{1, \ldots, l\}\}.$$

$$K := \{a \in \mathbb{Z}_n^* : a^m \equiv \pm 1 \bmod(n)\}.$$

$$M := \{a \in \mathbb{Z}_n^* : a^m \equiv 1 \bmod(n)\}.$$

Está claro que $P \subseteq K$, que $M \subseteq K$ y, además, $K \subseteq L$ por el teorema chino de los restos. Esta última contención es porque si se tiene un elemento de K, entonces podemos suponer que $a^m \equiv 1 \bmod(n)$ (se hace análogo si es congruente con -1), entonces por el teorema chino de los restos se tiene que $a^m \equiv 1 \bmod(p_i^{e_i})$ para todo i. Por último, $L \subseteq F$ por definición de m.
Justifiquemos que $[L : M]$ es una potencia de 2. Se tiene que, si un elemento a está en L, entonces a^2 está en M, de nuevo por el teorema chino de los restos. Y también se tiene que la aplicación $h : L \longrightarrow M$ definida por $h(x) = x^2$, que hemos justificado ya que está bien definida, es un homomorfismo de grupos (ya que cumple que $h(x \cdot y) = h(x) \cdot h(y)$ para todo x, y de L). Ahora, tenemos que, si cogemos un elemento b de $L/Im(h)$ (sabemos que $Im(h)$ es un subgrupo de M), entonces $b^2 = 1$, luego todo elemento de ahí tiene orden 2 si no es el neutro. Con esto, nos podemos ir al teorema de Cauchy. Supongamos que tenemos un primo impar p que es divisor del orden de $L/Im(h)$, entonces debería existir un elemento de orden p, pero todo elemento de ahí tiene orden 2, luego es absurdo. Por tanto de aquí se concluye que $L/Im(h)$ es potencia de 2. Al tener la siguiente contención, $Im(h) \subset M \subset L$, $[L : K]$ divide a $[L : Im(h)]$, luego, necesariamente, $[L : K]$ es también potencia de 2.
Y por tanto se deduce que $[L : K] = 2^j$ para algún $j \geq 0$ entero, ya que se tiene que $[L : M]$ es una potencia de 2, y que $M \subset K \subset L$.
Si $j \geq 2$, entonces $[L : K] = \frac{|L|}{|K|} \geq 4$, y por tanto despejando, $|K| \leq \frac{|L|}{4}$. Luego, como $P \subseteq K$, usando esto y las desigualdades previas,

$$|P| \leq |K| \leq \frac{|L|}{4} \leq \frac{|\mathbb{Z}_n^*|}{4} = \frac{\varphi(n)}{4}.$$

Si $j = 1$, entonces $n = p \cdot q$, con p, q primos distintos, ya que si se tienen tres factores ó más, entonces hay más del doble de elementos en L que en K, por el teorema chino de los restos, y necesariamente se tendría que tener

$j > 1$. Pero entonces sabemos que n no es un número de Carmichael (hemos dicho que dichos números son producto de al menos tres primos distintos), y por tanto, F es distinto de $\mathbb{Z}_n^*$. Luego, $[\mathbb{Z}_n^* : F] \geq 2$, luego, como estamos suponiendo que $[L : K] = 2$, $[\mathbb{Z}_n^* : K] \geq 4$. Por tanto,

$$|P| \leq |K| \leq \frac{|\mathbb{Z}_n^*|}{4} = \frac{\varphi(n)}{4}.$$

Si $j = 0$, entonces, los conjuntos L y K son iguales, y en consecuencia n es potencia de un primo p, $n = p^t$. Vamos a ver que si $n \neq 9$, $|F| \leq p - 1 \leq \frac{\varphi(n)}{4}$. Aplicando las propiedades de la función φ de Euler, se tiene que $\varphi(n) = p^{t-1} \cdot (p-1)$. Además, usamos que el grupo $\mathbb{Z}_n^*$ es cíclico de orden $n - 1$. Entonces, si se tiene un elemento a de F, se deduce que $ord(a)$ divide a $n - 1$, y también a $\varphi(n)$, por la congruencia de Euler. Ahora bien, $n - 1 = p^t - 1 = (p-1) \cdot (p^{t-1} + \cdots + p + 1)$, luego como tiene que dividir a ambas cosas, necesariamente $ord(a)$ divide a $p - 1$. Por tanto, las únicas bases posibles son las del subgrupo cíclico de $\mathbb{Z}_n^*$ de orden $p - 1$, luego hay $p - 1$ como mucho. En consecuencia, $|P| \leq |F| \leq p - 1 \leq \frac{\varphi(n)}{4}$, salvo que $n = 9$. Ya que se tiene lo siguiente:

$$p - 1 > \frac{\varphi(n)}{4} \iff p - 1 > \frac{p^{t-1} \cdot (p-1)}{4} \iff 4 > p^{t-1}.$$

Pero, necesariamente $t > 1$ ya que n no es primo, y entonces, la única posibilidad es que $p = 3$ y $t = 2$, es decir, $n = 9$.
Vamos a ver por tanto ese caso, $n = 9$. Ahí, $\varphi(9) = 6$, y por tanto tiene que ocurrir que $P \leq \frac{3}{2}$. Pero sólo para $a = 1$ cumple el que 9 sea pseudoprimo respecto de la base a, luego hemos terminado.

□

Ahora ya estamos en condiciones de presentar el algoritmo, después haremos unos comentarios sobre la probabilidad de error.

ALGORITMO DE RABIN-MILLER:
Input: un número entero impar $n > 1$.

1. Se escribe $n - 1 = 2^s \cdot d$ y se coge un número al azar a entre 1 y $n - 1$.
2. Si $mcd(a, n) \neq 1$, entonces n es compuesto.
3. Si no, se hace $b = a^d \bmod(n)$.
4. Si $b = 1$, entonces n es probablemente primo.

5. Si no, se hace $b, b^2, \ldots, b^{2^{s-1}} \mod(n)$.

6. Si ninguno de esos valores es -1, entonces n es compuesto, y si no, n es probablemente primo.

Dado el último teorema que hemos demostrado antes de la presentación del algoritmo, sabemos que la probabilidad de que n sea compuesto si el algoritmo nos da que es primo es menor que $\frac{1}{4}$, luego, para aumentar la probabilidad de que dicho número sea primo lo único que hay que hacer es repetir el algoritmo tantas veces como sea necesario. Ya que si repetimos el algoritmo k veces, si siempre nos da que n es primo, la probabilidad de error en ese caso sería de $\frac{1}{4^k}$.
Además, según lo que se ve en el apéndice de complejidad, se puede asegurar que estamos ante un test probabilístico de tipo Montecarlo.

1.4. Test de Solovay-Strassen

Al igual que el test de Rabin-Miller, el de Solovay-Strassen va a ser un test probabilístico.

Vamos a necesitar introducir varias nociones que no han aparecido antes a lo largo del trabajo, las cuales van a ser la base de dicho test.

Como el objetivo principal del trabajo no es presentar este tipo de test, y ya nos hemos centrado más en el de Rabin-Miller como test de primalidad probabilístico, los resultados que se presentan en esta sección no se demuestran. Se exponen de manera somera, pero suficiente para hacerse una idea de cómo funciona este test de Solovay-Strassen.

Definición 7. Sean a, m dos números enteros tales que $m \geq 2$ y $mcd(a, m) = 1$. Entonces se dice que a es un residuo cuadrático módulo m si $a \equiv x^2 \mod(m)$ para algún número entero x.

Nótese que, aunque en la definición anterior se habla de un entero a, nos vamos a fijar en la clase de a módulo m.

Ejemplo 2. Si cogemos $m = 13$, nos basta con calcular los cuadrados de los números enteros del 1 al 12 y hacer módulo 13 para así determinar los

residuos cuadráticos módulo 13. Ya que si tenemos un x entero que es mayor que 13, entonces al hacer su cuadrado para luego mirar si un a es residuo cuadrático módulo 13, es lo mismo que hacer x módulo 13 y luego hacer el cuadrado.
Haciendo dichas cuentas, se obtiene que los residuos cuadráticos módulo 13 son el $1, 3, 4, 9, 10, 12$. Nótese que hay 6 números ahí, justo la mitad de $12 = 13 - 1$. Esto no es casualidad, ya que 13 es primo.

Observación 4. Se puede ver que si p es primo, entonces hay como mucho $\frac{p-1}{2}$ residuos cuadráticos módulo p.

Definición 8. sea p un primo impar y a un entero. Entonces, el símbolo de Legendre de a respecto de p es:

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{si } a \text{ es un residuo cuadrático módulo } p. \\ -1 & \text{si } a \text{ no es un residuo cuadrático módulo } p. \\ 0 & \text{si } a \text{ es un múltiplo de } p. \end{cases}$$

A continuación se van a exponer algunas de las propiedades del símbolo de Legendre, y el teorema de reciprocidad cuadrática de Gauss para el cálculo del símbolo de Legendre.

Propiedades 1. Sean p un número primo impar, y a, b dos números enteros.

- $\left(\frac{a \cdot b}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$.
- Si $a \equiv b \bmod(p)$, $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.
- $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$.

Teorema 8 (Ley de reciprocidad cuadrática de Gauss 1)**.** *Sean p, q primos, entonces se tiene:*

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1 & si\ p \equiv \pm 1\ mod(8). \\ -1 & si\ n \equiv \pm 3\ mod(8). \end{cases}$$

$$\left(\frac{p}{q}\right) = (-1)^{\frac{(p-1)\cdot(q-1)}{4}} \cdot \left(\frac{q}{p}\right) = \begin{cases} -\left(\frac{q}{p}\right) & si\ p \equiv q \equiv 3\ mod(4). \\ \left(\frac{q}{p}\right) & en\ otro\ caso. \end{cases}$$

Lo que ahora uno se pregunta automáticamente, es si esto se podría extender a cualquier número que no fuera primo. La respuesta es afirmativa, para los números impares positivos; es dónde aparece el símbolo de Jacobi.

Definición 9. Sea n un número impar positivo del cuál además se conoce la factorización en producto de primos, $n = p_1 \cdot p_2 \cdots p_k$, y a un entero. Entonces, el símbolo de Jacobi de a respecto de n se define como:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \cdot \left(\frac{a}{p_2}\right) \cdots \left(\frac{a}{p_k}\right).$$

Al igual que antes, vamos a enumerar algunas de las propiedades más importantes.

Propiedades 2. sean n, m dos números impares positivos, y a, b dos números enteros.

- $\left(\frac{a \cdot b}{n}\right) = \left(\frac{a}{n}\right) \cdot \left(\frac{b}{n}\right)$.
- $\left(\frac{a}{n \cdot m}\right) = \left(\frac{a}{n}\right) \cdot \left(\frac{a}{m}\right)$.
- Si $a \equiv b \bmod(n)$, $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.
- $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$.

Por último, antes de entrar en el algoritmo de Solovay-Strassen, vamos a ver uno de los resultados más importantes de esta teoría, cuya demostración omitiremos.

Teorema 9 (Ley de reciprocidad cuadrática de Gauss 2)**.** *Sean m, n enteros impares positivos tales que $m, n \geq 3$. Entonces, se tienen dos cosas:*

$$\left(\frac{m}{n}\right) = \begin{cases} \left(\frac{n}{m}\right) & \text{si } n \equiv 1 \ mod(4) \ \text{ó } m \equiv 1 \ mod(4). \\ -\left(\frac{n}{m}\right) & \text{si } n \equiv 3 \ mod(4) \ y \ m \equiv 3 \ mod(4). \end{cases}$$

$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{si } n \equiv 1 \ mod(8) \ \text{ó } n \equiv 7 \ mod(8). \\ -1 & \text{si } n \equiv 3 \ mod(8) \ \text{ó } n \equiv 5 \ mod(8). \end{cases}$$

Este resultado que se acaba de enunciar, junto con las propiedades anteriores, son realmente importantes a la hora de realizar los cálculos de símbolos de Jacobi.

Observación 5. Supongamos que se tienen dos números enteros n, m y se ha de calcular $\left(\frac{n}{m}\right)$. Si $n \geq m$, entonces se hace la división euclídea y por las propiedades anteriores, si r es el resto de dicha división, $\left(\frac{n}{m}\right) = \left(\frac{r}{m}\right)$. Ahora bien, para calcular $\left(\frac{r}{m}\right)$, ó $\left(\frac{n}{m}\right)$ en el caso de que $n < m$, entonces se puede acudir al teorema de reciprocidad cuadrática, lo cuál luego nos lleva otra vez

a repetir el proceso.
Así hasta llegar a tener un -1 ó un 2, ya que en esos casos se sabe ya el resultado también por lo que se acaba de ver.
Por tanto, hacer estos cálculos no es complicado, y se pueden implementar sin mayores complicaciones.

Para desarrollar el test de Solovay-Strassen en sí, conviene ver algunos resultados y definiciones más.

Definición 10. Un número n es un pseudoprimo de Euler respecto de una base a, con $mcd(a,n) = 1$, si $\left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} \bmod(n)$. Se dirá que n es un pseudoprimo de Euler si lo es respecto de cualquier base a tal que $mcd(a,n) = 1$.

Observación 6. Se puede ver que todo pseudoprimo respecto de una base a es también un pseudoprimo de Euler respecto de la misma base a.

Teorema 10. *Un número n es primo si, y sólo si, para todo a tal que $mcd(a,n) = 1$, se tiene que $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right)$ mod(n). Es decir, n es primo si, y sólo si, n es un pseudoprimo de Euler.*

Observación 7. Se puede obtener de todo lo anterior un test determinista de primalidad (aunque no eficiente, ya que habría que comprobar lo siguiente para todas las bases a tales que $mcd(a,n) = 1$).

Sin embargo, con el lema siguiente, se va a poder ya determinar el test probabilístico de Solovay-Strassen.

Lema 3. *Sea n un número compuesto impar. Entonces, al menos la mitad de las bases a no cumplen que $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right)$ mod(n).*

Por tanto, el test se va a basar simplemente en ver qué ocurre con la congruencia $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \bmod(n)$. Si no se da, entonces n no será un pseudoprimo de Euler, y por tanto, sabemos que n es compuesto. En cambio, si se da la congruencia, por el lema previo sabemos que, la probabilidad de que sea compuesto es menor que $\frac{1}{2}$, por lo tanto, la probabilidad de que sea primo es mayor ó igual a $\frac{1}{2}$.

ALGORITMO DE SOLOVAY-STRASSEN:
Input: un número entero impar $n \geq 3$.

1. Se coge un número entero $2 \leq a \leq n-1$ aleatorio.

2. Si $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \bmod(n)$, entonces devuelve 1, y si no se da la congruencia, 0.

La devolución de 0 como output implica que n es compuesto por lo que se ha dicho anteriormente, y si es 1, n es primo con probabilidad de error menor que $\frac{1}{2}$.

Observación 8. Para ver que un número es primo con una probabilidad de error baja, bastará con aplicarle el algoritmo al número varias veces, tal y como se hace con Rabin-Miller. Supongamos que lo hacemos t veces, entonces si todas las veces nos da ese resultado el algoritmo, el número será primo con probabilidad de error menor que $\frac{1}{2^t}$.

Observación 9. Se puede ver que la complejidad computacional de este algoritmo es $\mathcal{O}(t \cdot k^3)$, donde t hace referencia al número de veces que ejecutamos el algoritmo (para así adecuarnos a la probabilidad querida), y k a la longitud del input; esta complejidad es la misma para el algoritmo de Rabin-Miller. Por tanto, vemos que es un algoritmo eficiente, aunque eso sí, probabilístico.

1.5. El problema de la búsqueda de primos

Todo sobre lo que gira el trabajo es el determinar si, dado un número, es primo o compuesto. Hemos visto que para ello, podemos usar algoritmos deterministas o probabilísticos, con posibilidad incluso de elegir dentro del mismo tipo de algoritmo, ya que algunos son mejores en cuanto a coste computacional que otros.
Sin embargo, otro problema muy importante es el de buscar números primos. Cuando se habla de él, se hace referencia a números con una cantidad considerable de cifras, lo cuál dificulta las cosas, y es por ello que es un tema relevante. Aquí se incluyen entre otras muchas cosas, por ejemplo, el estudio de saber cuánto se espera tardar en encontrar un número primo si vamos cogiendo números al azar.

Definición 11. Sea $n \geq 2$ un número natural. Se define $\pi(n)$ como el número de primos p que hay tales que $p \leq n$.

Lema 4. *Sea $n \geq 2$ un número natural. Si se coge aleatoriamente un número $k \leq n$, el número medio de veces que hay que repetir este proceso hasta encontrar un número k que sea primo es $\frac{n}{\pi(n)}$.*

Demostración. La probabilidad de encontrar un número primo escogido al azar que sea menor ó igual que n, es de $\frac{\pi(n)}{n}$. Ahora bien, si pensamos, estamos ante una distribución geométrica de parámetro $\frac{\pi(n)}{n}$, ya que queremos ver cuándo ocurre el suceso por primera vez (en nuestro caso el suceso es encontrar un número primo $k \leq n$ habiéndolo escogido al azar). Luego, simplemente, lo que nos indica el enunciado del lema es la esperanza de dicha distribución, la cuál sabemos que es el inverso del parámetro, o sea, $\frac{n}{\pi(n)}$.

□

Observación 10. Lo que aparece en el lema previo, si lo traducimos a lo que a algoritmos se refiere, es el tiempo esperado del algortimo, ya que en este caso, no tendría sentido hablar del tiempo de ejecución hablando del número máximo de operaciones bit.

En lo que concierne a $\pi(n)$, se tienen además los siguientes resultados, que nos dan acotaciones. Las demostraciones no las presentaremos debido a su complejidad.

Este primer resultado es el más débil.

Teorema 11. *Para todo $n \geq 2$, existe una constante $C > 0$ tal que $\pi(n) \geq C \cdot \frac{n}{log(n)}$.*

Teorema 12. *Asintóticamente, $\pi(n)$ se comporta como $\frac{n}{ln(n)}$. Es decir, si tomamos $\epsilon > 0$ arbitrariamente pequeño, entonces existe un n_0 natural tal que:*

$$(1-\epsilon) \cdot \frac{n}{ln(n)} \leq \pi(n) \leq (1+\epsilon) \cdot \frac{n}{ln(n)} \qquad \forall n \geq n_0.$$

Con estos resultados nos podemos hacer una idea de cómo se distribuyen los números primos de manera general. Dado un número z elegido al azar del conjunto $\{3, \ldots, x-1\}$ (con x fijo), la probabilidad de que z sea primo es aproximadamente $\frac{1}{ln(x)}$.

Otra manera de buscar primos que es interesante es verlo en progresiones aritméticas, debido al siguiente resultado.

Teorema 13 (Teorema de Dirichlet)**.** *Sean a y d dos números primos entre sí. Entonces se tiene que en la progresión aritmética $A = a, a+d, a+2 \cdot d, \ldots, a+k \cdot d, \ldots,$ hay infinitos números primos.*

No sólo se tiene el resultado anterior, sino que también se tiene que se puede determinar la densidad de los números primos que hay en una progresión del estilo del cual acabamos de tratar. Si denotamos por $\pi(x, d, a)$ al número de primos de la progresión A menores que x, entonces se puede ver que dicha cantidad es aproximadamente $\frac{\pi(x)}{\varphi(d)}$, ó bien, $\frac{x}{\varphi(d)\cdot ln(x)}$, según lo que se ha visto anteriormente para la función $\pi(x)$.
Y se puede ver que, por tanto, la probabilidad de que un número cogido al azar de la progresión sea primo es de $\frac{1}{ln(x)}$.

Ejemplo 3. La importancia del Teorema de Dirichlet se ve reflejada por ejemplo en la criptografía, y tiene que ver con la búsqueda de un número primo grande p tal que $p-1$ no tenga factores primos pequeños.
Supongamos que encontramos un primo grande q (lo que cuesta viene reflejado en los teoremas de densidad de los números primos), entonces podemos aplicar el Teorema de Dirichlet a la progresión $A = \{1 + k \cdot q : k \geq 1\}$, y así encontrar un nuevo primo $p = 1 + k \cdot q$, con k un número natural. De esta manera, tenemos que $p - 1 = k \cdot q$, dónde q es un factor primo grande.

Capítulo 2

Teorema AKS

En este capítulo veremos el teorema de Agrawal, Kayal y Saxena (AKS), resultado principal del trabajo. Cabe destacar la importancia de este teorema, que ha dado lugar al primer algoritmo determinista y eficiente que se puede usar para resolver el problema de determinar la primalidad de un número. Aunque en la práctica se usen los algoritmos probabilísticos, debido a su mayor rapidez computacional, es realmente importante este resultado. Con él, se ha sido posible determinar que este problema está en la clase de complejidad P (ver apéndice A).

Debido a la complejidad de la demostración del teorema, ya que se necesitan una cantidad extensa de resultados para llegar a la misma, vamos a dividir el capítulo en distintas secciones, atendiendo a la naturaleza de los resultados posteriormente a utilizar.

En primer lugar vamos a dar una idea que va a ser en la que se va a basar el algoritmo AKS, viendo los inconvenientes de la misma, para a continuación pasar a enunciar y demostrar el teorema AKS en sí, viendo los resultados necesarios para ello. Y, por último, vamos a presentar el algoritmo, hablando también de su complejidad.

2.1. Criterio básico

El AKS se basa en un criterio de primalidad simple, que vamos a enunciar a continuación. Sin embargo, éste no será un algoritmo útil, debido a su gran coste computacional.

Teorema 14. *Sea $n \geq 2$ un número natural, y a un número natural, con $mcd(a, n) = 1$.*
Entonces,

$$n \text{ es primo, si, y sólo si, } x^n + a \equiv (x + a)^n \text{ mod}(n).$$

Para llevar a cabo su demostración, se necesitan los resultados que siguen.

El primero de todos es un teorema que recibe el nombre de Fermat para polinomios, debido a su parecido al pequeño teorema de Fermat.

Teorema 15 (Fermat para polinomios)**.** *Sea p un número primo. Entonces se tiene que:*

$$(P(x))^p \equiv P(x^p) \text{ mod}(p)$$

para todo polinomio P de $\mathbb{Z}[x]$.

Demostración. Razonemos por inducción sobre el grado d del polinomio P.
Si $d = 0$, entonces el polinomio P es una constante, $P = n$, con n un número entero distinto de 0. Por un lado se tiene que $P(x)^p = n^p$, y por otro lado, $P(x^p) = n$. Ambas cosas son congruentes módulo p, por el pequeño teorema de Fermat.
Supongamos ahora que se cumple el teorema para grado d, y vamos a ver qué ocurre con $d + 1$. En este caso, el polinomio P se podrá escribir de la siguiente forma: $P(x) = Q(x) + cx^{d+1}$, con el grado de Q menor o igual que d, y c distinto de cero, ya que si no estaríamos de nuevo en el caso anterior. En primer lugar, $P(x^p) = Q(x^p) + cx^{p(d+1)}$. Por otro lado, $P(x)^p = \sum_{k=0}^{p} \binom{p}{k} Q(x)^k (cx^{d+1})^{p-k}$. Puesto que $\binom{p}{k} \equiv 0 \bmod(p)$ cuando k está en $\{1, \ldots, p-1\}$, se tiene que, $P(x)^p \equiv Q(x)^p + c^p x^{(d+1)p} \bmod(p)$. Además sabemos que $c^p \equiv c \bmod(p)$, de nuevo por el pequeño teorema de Fermat. Pero no sólo eso, haciendo uso de la hipótesis de inducción, se tiene que $Q(x)^p \equiv Q(x^p) \bmod(p)$. Por tanto, $P(x)^p \equiv Q(x^p) + cx^{p(d+1)} \bmod(p)$, lo cual nos permite concluir.

□

Lema 5. *Sea n un número natural mayor ó igual que 2, p un divisor primo de n, y j el mayor entero tal que p^j divide a n. Entonces p^j no divide a $\binom{n}{p}$.*

Demostración. Razonemos por reducción al absurdo.
Suponemos que p^j divide a $\binom{n}{p}$. A la vista de la igualdad

$$\binom{n}{p} \cdot p! = n \cdot (n-1) \cdots (n-p+1) = n \cdot (n-p+(p-1)) \cdots (n-p+1),$$

tendríamos que p^{j+1} divide al término de la izquierda de la igualdad. Ahora bien, al tener una igualdad, deberíamos tener que p^{j+1} divide al término de la derecha. Por hipótesis, j es el mayor índice tal que p^j divide a n. Luego, necesariamente hay que tener que p divide a $(n-p+(p-1)) \cdots (n-p+1)$. Pero si p divide al producto, entonces tiene que dividir a uno de los factores. Sin embargo, todos esos factores son de la forma $n-p+i$ con $1 \leq i \leq p-1$. Pero p divide a n y a p, sin embargo, no dividirá a ningún i, ya que son todos menores que p (nótese que esto mismo se ha usado al principio para decir que p divide a $p!$ pero p^2 no lo divide). Y por tanto, se ha llegado a una contradicción.

□

Lema 6. *Sea n un número natural tal que $n \geq 2$, y p un número primo tal que p divide a n. Entonces, $\binom{n}{p}$ no es divisible por n.*

Demostración. Razonemos por reducción al absurdo.
Supongamos que n divide a $\binom{n}{p}$. Entonces, si j es el mayor índice tal que p^j divide a n, necesariamente p^j debería dividir a $\binom{n}{p}$. Pero esto último no puede ocurrir por el lema previo.
Por tanto, hemos llegado a una contradicción y el resultado queda demostrado.

□

Estamos ya en condiciones de ir a la demostración del criterio de primalidad.

Demostración (del Teorema 14)**.** Comenzamos probando la condición necesaria, suponiendo que n es primo. Entonces podemos concluir haciendo uso del teorema anterior, aplicado en este caso a $P(x) = x + a$.
Vamos a hacer una reducción al absurdo para ver la condición suficiente. Supondremos que n no es primo, y entonces veremos que no se dará la congruencia del enunciado. Tenemos que ver que en $(x+a)^n$ hay algún término más además de x^n y a^n que sea no nulo módulo n, ya que será 0 en $x^n + a$.
Como n no es primo, entonces tendrá un divisor primo p. Nos remitimos

ahora al lema anterior, y entonces sabemos que n no divide a $\binom{n}{p}$. Ahora nos vamos pues a la expresión $(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$. Ahí, nos fijamos ahora en el término de grado p, que es $\binom{n}{p} x^p a^{n-p}$. Pero, n no divide a $\binom{n}{p}$, y $mcd(a,n) = 1$, luego n no divide a a^{n-p}. Por tanto, $\binom{n}{p} x^p a^{n-p} \not\equiv 0 \bmod(n)$. Luego, la congruencia no se da, y hemos terminado.

□

Eficiencia del método

En lo que sigue, hasta que se diga lo contrario, n denotará a un número entero, del cuál se quiere estudiar si es ó no primo.

El teorema anterior no sólo es válido para polinomios de la forma en la que se enuncia en el teorema, también lo es para polinomios P que tengan coeficientes coprimos con n. Por tanto, dado un número n, para ver si es primo, bastaría con coger un polinomio adecuado (según lo que acabamos de ver) y comprobar si se da la congruencia.

Ahora bien, llegados a este punto, tenemos un resultado que nos dice exactamente si un número n es primo o compuesto, haciendo una única congruencia, a priori sencilla. El problema radica en la eficiencia del mismo, por tanto no es válido.

Las operaciones aritméticas en $\mathbb{Z}_n$ son eficientes, además, en el apéndice A se ve que la multiplicación es eficiente (con varios métodos incluso). El cociente también lo será haciendo uso del algoritmo de Euclides. Y también se puede hacer de forma eficiente la reducción módulo n.

Lo que no va a ser eficiente en esa congruencia es el cálculo de $(x+a)^n$. Para hacer esta operación, se puede usar el algoritmo de los cuadrados repetidos, y en cada paso, reducir módulo n. Los coeficientes de los polinomios resultantes van a estar siempre acotados (por n), pero en cambio, el grado del polinomio crece, y lo hace con el tamaño de n, es decir, nos va a llevar a una complejidad exponencial.

Sin embargo, a partir de esta congruencia, si además de hacer módulo n en cada paso, reducimos también módulo otro polinomio Q de grado r (pequeño con respecto a n), vamos a llegar a algo eficiente. Vamos a tener el

grado siempre acotado por r, y por tanto, no va a aparecer ese crecimiento exponencial del que se hablaba antes. En la sección siguiente se tratan en profundidad todos los resultados que conciernen a este comentario.

2.2. El teorema AKS

En esta sección se presentan todos los resultados necesarios para poder llevar a cabo la demostración del teorema AKS, el cuál se enunciará y demostrará al final de la misma, cuando se tengan los resultados pertinentes. Además, como se ha anunciado al final de la sección anterior vamos a ahondar en los resultados que hacen que esa modificación funcione.

Lo primero que hacemos por tanto, es esclarecer la idea que se va a seguir en este algoritmo AKS, y para lo cuál luego necesitaremos uso de todos los resultados que se presentan en esta sección.

En primer lugar, como hemos dicho, usar el Teorema 14 directamente no se puede hacer de manera eficiente. Para arreglar este problema, también hemos comentado que se necesita elegir adecuadamente (más adelante veremos cómo) otro polinomio Q sobre el cuál se hará el módulo.
Esto va a implicar que, en vez de tener que realizar una única congruencia, se van a tener que llevar a cabo varias, sobre un polinomio P elegido convenientemente. Lógicamente al estar haciendo módulo n y Q, el teorema 9 ya no se puede aplicar directamente, pero lo podemos usar en parte. Cuando se tenga que $P(x)^n \not\equiv P(x^n) \bmod(n, Q)$, se puede asegurar que n es compuesto. El hecho de pasar a tener que realizar varias congruencias, es porque al reducir módulo Q, va a haber algunos números compuestos que van a cumplir este tipo de congruencias. Veremos que hay resultados que nos aseguran que si miramos un determinado números de ellas, llega un momento donde se puede asegurar la primalidad del número.

Observación 11. Dado un polinomio con coeficientes enteros $P(x)$, hacer $P(x) \bmod(n, Q)$ es la reducción de los coeficientes de P módulo n, seguido de la reducción módulo $Q(x)$. Es decir, es la clase de $P(x)$ en $\mathbb{Z}_n[x]/(Q)$.

Notación. A partir de ahora, n denotará un entero positivo (en general el entero que querremos saber si es ó no primo), p un número primo, r un entero positivo y $Q(x)$ el polinomio $Q(x) := x^r - 1$ de $\mathbb{Z}[x]$.

Más adelante se determinará el r adecuado para el algoritmo AKS.

Observación 12. Observemos que, dado un número natural $m \geq 2$, se tiene por un lado que $Q(x^m) = x^{rm} - 1 \equiv 0 \bmod(Q)$, ya que $x^{rm} - 1 = (x^r - 1) \cdot (x^{r(m-1)} + x^{r((m-2)} + \cdots + 1)$.
Por otro lado, veamos que si P es un polinomio tal que $P \equiv 0 \bmod(n, Q)$, entonces $P(x^m) \equiv 0 \bmod(n, Q)$. Por hipótesis, se tiene que existe un polinomio R tal que $P \equiv R \cdot Q \bmod(n)$, luego $P(x^m) \equiv R(x^m) \cdot Q(x^m) \bmod(n)$, y esto, junto con la primera parte de la observación, permite concluir que si $P \equiv 0 \bmod(n, Q)$, entonces $P(x^m) \equiv 0 \bmod(n, Q)$.

Lema 7. *Sean P un polinomio, y, m_1 y m_2 dos números naturales tales que $P(x)^{m_1} \equiv P(x^{m_1})$ mod(p, Q), y $P(x)^{m_2} \equiv P(x^{m_2})$ mod(p, Q). Entonces, si $m = m_1 \cdot m_2$, se tiene que $P(x)^m \equiv P(x^m)$ mod(p, Q).*

Demostración. La primera condición que nos da el enunciado se puede escribir como $P(x)^{m_1} - P(x^{m_1}) \equiv 0 \bmod(p, Q)$. Esto nos remite al lema anterior, y por tanto, podemos sustituir x por x^{m_2}. Tenemos entonces que $P(x^{m_2})^{m_1} - P(x^{m_1 \cdot m_2}) \equiv 0 \bmod(p, Q)$. De aquí ya obtenemos que $P(x^{m_2})^{m_1} \equiv P(x^m) \bmod(p, Q)$.
También tenemos que $P(x^{m_2}) \equiv P(x)^{m_2} \bmod(p, Q)$, luego, $P(x^{m_2})^{m_1} \equiv (P(x)^{m_2})^{m_1} \bmod(p, Q)$. Pero, $(P(x)^{m_2})^{m_1} = P(x)^m$.
Por tanto, $P(x)^m \equiv P(x^m) \bmod(p, Q)$.

□

En lo que sigue, n denotará un número compuesto y p será un divisor primo de n. Además, llamaremos $\mathcal{P}$ al conjunto de polinomios de $\mathbb{Z}[x]$ tales que cumplen que $P(x)^n \equiv P(x^n) \bmod(p, Q)$.

Corolario 1. *Sea P un polinomio de $\mathcal{P}$. Entonces, $P(x)^m \equiv P(x^m)$ mod(p, Q) para todo m de la forma $m = n^i \cdot p^j$, donde $i, j \geq 0$.*

Demostración. Es directo, ya que es aplicar el lema anterior primero a $m_1 = n, m_2 = n$, luego a $m_1 = n^2, m_2 = n$, y así hasta llegar a que se cumple para los $m = n^i$. De manera análoga se hace para p^j, y luego simplemente es volver a aplicar el lema previo esta vez a $m_1 = n^i, m_2 = p^j$.

□

Lema 8. *Sean P_1 y P_2 dos polinomios de $\mathcal{P}$, entonces el producto $P_1 \cdot P_2$ está también en $\mathcal{P}$.*

Demostración. Vamos a acudir directamente a la definicíon del conjunto $\mathcal{P}$. Para empezar, es claro que si dos polinomios tienen coeficientes enteros, el producto será otro polinomio con coeficientes enteros. Y tenemos que $(P_1 \cdot P_2)(x^n) = P_1(x^n) \cdot P_2(x^n)$. Pero sabemos que $P_1(x)^n \equiv P_1(x^n) \bmod(p, Q)$, y que $P_2(x)^n \equiv P_2(x^n) \bmod(p, Q)$ por hipótesis, luego, $P_1(x^n) \cdot P_2(x^n) \equiv P_1(x)^n \cdot P_2(x)^n \bmod(p, Q)$. Y esta última expresión es igual a $((P_1 \cdot P_2)(x))^n$, luego se tiene el resultado.

□

Notación. En lo que sigue, vamos a necesitar de diversas notaciones:

- H va a denotar un polinomio irreducible módulo p, que es también un divisor de Q módulo p.
- A será el número de polinomios de $\mathcal{P}$ distintos módulo p y H.
- t será el número de monomios de la forma $x^{n^i p^j}$, con $i, j \geq 0$, que son distintos módulo p y H .
- ℓ será el número de enteros a tales que $0 \leq a \leq p-1$ y que cumplen que el polinomio $x + a$ es un elemento de $\mathcal{P}$.

Estas notaciones previas de A, t y ℓ son importantes ya que más adelante veremos que dependiendo de las relaciones entre ellas, vamos a poder decir que, según qué casos, vamos a estar ante un número compuesto que va a ser potencia de un primo, lo cuál va a ser muy útil a la hora de descartar algunas posibilidades.

Llegados a este punto, vamos a hacer un pequeño inciso que va a ser necesario para los resultados que vienen a continuación.

Observación 13. Sea p un número primo y $P(x), H(x)$ polinomios en $\mathbb{Z}[x]$. Vamos a suponer que H es irreducible módulo p, es decir, que su clase módulo p, la cuál denotamos por $\bar{H}(x)$, es irreducible en el anillo $\mathbb{Z}_p[x]$. Escribimos $F = \mathbb{Z}_p[x]/(\bar{H})$. Nótese que, dada la irreducibilidad de $\bar{H}$ en $\mathbb{Z}_p[x]$ hace que F sea un cuerpo finito, cuyo cardinal será p^r, donde p es el grado de $\bar{H}$, luego, necesariamente, A será un número finito.

Una vez tenemos esto, diremos que un polinomio $R(x)$ con coeficientes enteros es **raíz polinómica** de $P(x)$ si $P(R(x)) \equiv 0 \bmod(p, H)$. Lo que estamos diciendo con esto es que la clase en F es nula.

Vamos a enunciar a continuación un teorema que necesitaremos para la demostración del lema siguiente.

Teorema 16. *Sea $P(x)$ un polinomio de grado d con coeficientes enteros. Entonces, existen a lo sumo $s \leq d$ raíces polinómicas de $P(x)$ (que denotaremos por $R_1(x), \ldots, R_s(x)$) tales que sus clases en F son distintas, es decir, tales que $R_i(x) \not\equiv R_j(x)$ mod(p, H) si $i \neq j$.*

Demostración. Empezamos tomando una nueva indeterminada y. Ahora, en vez de ver $P(x)$ como un polinomio de $\mathbb{Z}[x]$, podemos verlo como $P(y)$, donde ahora los coeficientes en lugar de verlos en $\mathbb{Z}$, los pasamos a ver en $\mathbb{Z}[x]$ (dado que son constantes, se pueden ver en cualquiera de los dos lados). Sea ahora $\bar{P}(y)$ la reducción de $P(y)$ módulo p y H. Nótese que, ahora los coeficientes de $\bar{P}(y)$ están en F, según la definición dada previamente para F. Por tanto, $\bar{P}(y) \in F[y]$, y además, el grado de $\bar{P}(y)$ es menor ó igual que el grado de P que es d.
Sea $R(x)$ un polinomio de $\mathbb{Z}[x]$, y $\bar{R}(x)$ su correspondiente clase en F, es decir, la reducción módulo p y H. Tener que $P(R(x)) \equiv 0 \bmod(p, H)$ equivale a decir que $\bar{R}(x) \in F$ sea raíz de $\bar{P}(y)$. Pero, tenemos que $\bar{P}(y)$ es un polinomio de grado menor ó igual que d, y los coeficientes de dicho polinomio están en el cuerpo F. Por tanto, se puede asegurar que el número de raíces de $\bar{P}(y)$ en F es menor ó igual que el grado d.

□

Lema 9. *Sean P_1 y P_2 dos polinomios de $\mathcal{P}$ cuyos grados son menores que t. Se tiene que, si $P_1 \equiv P_2$ mod(p, H), entonces también se da que $P_1 \equiv P_2$ mod(p).*

Demostración. Supongamos que tenemos un número $m = n^i \cdot p^j$. Entonces, por el Corolario 1, podemos escribir $P_1(x^m) \equiv (P_1(x))^m \bmod(p, Q)$, y $(P_2(x))^m \equiv P_2(x^m) \bmod(p, Q)$, pero como H es un divisor de Q, también es verdad que $P_1(x^m) \equiv (P_1(x))^m \bmod(p, H)$ y $P_2(x^m) \equiv (P_2(x))^m \bmod(p, H)$. Por las hipótesis del enunciado, se tiene que $(P_1(x))^m \equiv (P_2(x))^m \bmod(p, H)$, luego juntando todo, se llega a que $P_1(x^m) \equiv P_2(x^m) \bmod(p, H)$.
Ahora, vamos a hacer uso del inciso para coger el polinomio $T(y) := P_1(y) - P_2(y)$, donde P_1 y P_2 son polinomios en y con coeficientes en $\mathbb{Z}[x]$. Pero tenemos que $P_1(x^m) \equiv P_2(x^m) \bmod(p, H)$, es decir, que los elementos $\bar{x}^m$ son raíces de $\bar{T}(y)$, y hay t distintos. Puesto que el grado de $\bar{T}(y)$ es menor que t por hipótesis $\bar{T}(y)$ tiene a lo sumo $t - 1$ raíces. Por tanto, necesariamente, $\bar{T}(y) = 0$, y de aquí se deduce que $T \equiv 0 \bmod(p)$. Por ende, $P_1 \equiv P_2 \bmod(p)$.

□

Vamos a ver ahora una cota inferior para A, de manera que, ahora ya tendremos A acotado inferior y superiormente (aunque después se mejorará la cota superior).

Teorema 17. $A \geq \binom{t+l-1}{t-1}$.

Demostración. Sea $k < t$, y sean $a_1, a_2, \ldots, a_k \in \{0, \ldots, p-1\}$ distintos y tales que los correspondientes polinomios $x + a_1, \ldots, x + a_k$ están en $\mathcal{P}$. Gracias a la elección de k, podemos multiplicarlos todos, y tendríamos un polinomio $T = \prod_{i=1}^{k}(x + a_i)$ de grado menor que t, luego, por el Lema 8 dado que $k < t$, se tiene que dicho producto está en $\mathcal{P}$. Veamos que los a_i con $i \in \{1, \ldots, k\}$ se determinan de manera única por T salvo reordenación cuando se hace módulo un primo p (denotamos por $\bar{T}$ a dicha reducción de T). Razonemos por inducción sobre k. Si $k = 1$, entonces suponemos dos escrituras para $\bar{T}$; $\bar{T} = x + a_1$ y $\bar{T} = x + b_1$. Entonces $\bar{T}$ tiene una raíz a, y dado que en $\mathbb{Z}_p$ no hay divisores de cero, $a_1 \equiv a \equiv b_1 \bmod(p)$ (por ejemplo en $\mathbb{Z}_6$, donde sí que hay divisores de cero, el polinomio $x^2 + 5x$, que tiene como raíces a 0 y a 1, se puede escribir como $(x + 2) \cdot (x + 3)$). Ahora bien, veamos que ocurre para k suponiéndolo cierto para $k - 1$. Supongamos que tenemos que $\bar{T} = (x+a_1) \cdots (x+a_k)$, y por otro lado $\bar{T} = (x+b_1) \cdots (x+b_r)$. Con estas escrituras, se tiene que hay una raíz a en $\bar{T}$, igual que en el caso de $k = 1$, y dado que de nuevo estamos en $\mathbb{Z}_p$ y ahí no hay divisores de cero, entonces hay un índice j para el cuál se tiene que tener que $a_j \equiv b_j \bmod(p)$. Posteriormente, basta aplicar la hipótesis de inducción para concluir.
Si se tienen dos polinomios de esta forma $\bar{T}$ y $\bar{T}'$, y no se dan las congruencias de los coeficientes uno a uno, entonces no son congruentes módulo p, y por el lema 9, se tiene que tampoco lo serán módulo (p, H), luego estamos ante dos elementos distintos de A.
Luego, una vez visto esto, se tienen ℓ polinomios distintos de grado 1, y nos interesa saber cuántos productos de estos se pueden obtener exigiendo que el grado sea menor que t (donde el orden de los factores del producto es irrelevante por lo que hemos visto). Tenemos $\binom{\ell}{1}$ polinomios de grado 1, $\binom{\ell}{2}$ polinomios de grado 2, y así hasta $\binom{\ell}{t-1}$, que es el grado máximo que nos interesa. Luego podríamos hacer la suma de todos esos números combinatorios, ó usar que, por inducción, se puede probar que el número de maneras de coger hasta $t - 1$ números de 1 hasta ℓ es $\binom{t+\ell-1}{t-1}$.

□

Ahora veamos una cota superior mejor que la que se tenía para A.

Teorema 18. *Si n no es potencia de p, entonces $A \leq \frac{n^{2\cdot\sqrt{t}}}{2}$.*

Demostración. Para empezar, nos fijamos en los números $m = n^i \cdot p^j$. Al suponer que n no es potencia de p, entonces se obtienen dos números distintos de la forma de m para distintas elecciones de i y j. Nótese que si, por ejemplo, $n = p^2$, entonces se tendría que $m = p^{2\cdot i+j}$, y en particular, las elecciones $i = 1$, $j = 3$, e $i = 2$, $j = 1$ daría el mismo resultado.
Si además imponemos $0 \leq i, j \leq \lfloor\sqrt{t}\rfloor$, podemos escoger i de $\lfloor\sqrt{t}\rfloor + 1$ formas distintas, e igualmente para j. Luego, hay $(\lfloor\sqrt{t}\rfloor + 1)^2$ posibles elecciones.
Puesto que $(\lfloor\sqrt{t}\rfloor + 1)^2 > t$, existen $m_1 > m_2$ (de la forma $n^i \cdot p^j$), y cumpliendo que $x^{m_1} \equiv x^{m_2} \bmod(p, H)$.
Suponiendo que $m_1 = n^{i_1} \cdot p^{j_1}$, sabemos que, $i_1, j_1 \leq \lfloor\sqrt{t}\rfloor$, y por ello, podemos escribir $m_1 \leq n^{\lfloor\sqrt{t}\rfloor} \cdot p^{\lfloor\sqrt{t}\rfloor} = (n \cdot p)^{\sqrt{t}}$. Pero sabemos que p es un divisor primo de n, luego, necesariamente $p \leq \frac{n}{2}$.
Por tanto, podemos escribir $m_1 \leq (\frac{n^2}{2})^{\sqrt{t}}$, y dado que $2^{\sqrt{t}} \geq 2$, $m_1 \leq \frac{n^{2\cdot\sqrt{t}}}{2}$.
A continuación, pr el Corolario 1 podemos asegurar dos cosas, que $(P(x))^{m_1} \equiv P(x^{m_1}) \bmod(p, H)$ y que $(P(x))^{m_2} \equiv P(x^{m_2}) \bmod(p, H)$. Por la elección de m_1, m_2, tenemos que $(P(x))^{m_1} \equiv (P(x))^{m_2} \bmod(p, H)$. Volvemos a usar lo mismo de antes, cogemos una nueva indeterminada y y escribimos $R(y) = y^{m_1} - y^{m_2}$. Sabemos que, para todo polinomio $P(x)$ de $\mathcal{P}$, se tiene que $\bar{P}(x) \in F$ es raíz de $\bar{R}(y)$. Por tanto, R tendrá al menos A raíces polinómicas. Pero el grado de R es m_1 (ya que hemos supuesto que $m_1 > m_2$). Y sabemos que $m_1 \leq \frac{n^{2\cdot\sqrt{t}}}{2}$, luego, dado que el grado tiene que ser mayor ó igual que el número de raíces, tendremos finalmente que $A \leq m_1 \leq \frac{n^{2\cdot\sqrt{t}}}{2}$.

□

El siguiente corolario va a ser clave en lo que respecta al teorema AKS, ya que nos asegura que, bajo ciertas condiciones sobre t y ℓ, vamos a poder asegurar que estamos ante un número que es potencia de un primo.

Corolario 2. *Si $t > 4 \cdot (log(n))^2$ y $\ell \geq t - 1$, entonces n es potencia de p.*

Demostración. Haciendo uso de la cota inferior para A, se tiene que $A \geq \binom{t+\ell-1}{t-1}$. Ahora, sabemos por hipótesis que, como $\ell \geq t - 1$, tenemos que

$$A \geq \binom{t + t - 1 - 1}{t - 1} = \binom{2 \cdot (t - 1)}{t - 1} \geq \frac{2^t}{2}.$$

Para la última parte se ha usado la desigualdad $\binom{2\cdot k}{k} \geq 2^k$ donde k es un número natural. Dicho resultado se puede probar por inducción sobre k, haciendo uso de las desigualdades (n, h, c denotan números naturales ó el 0) $\binom{n+h}{c} \geq \binom{n}{c}$, $\binom{n+h}{c+h} \geq \binom{n}{c}$, además de la conocida igualdad $\binom{n+1}{c} = \binom{n}{c-1} + \binom{n}{c}$.
La desigualdad $t > 4 \cdot (log(n))^2$ equivale a $\sqrt{t} > 2 \cdot log(n)$, luego, podemos obtener ahora una desigualdad para t; $t = \sqrt{t} \cdot \sqrt{t} > 2 \cdot \sqrt{t} \cdot log(n)$.
Aplicamos esto a lo anterior, y tendríamos que $A > \frac{2^{2\sqrt{t}\cdot log(n)}}{2} = \frac{n^{2\sqrt{t}}}{2}$. Pero entonces, por el teorema anterior, esto sólo sería posible si n es potencia de p.

□

Por último, vamos a ver dos resultados más, que están relacionados con la noción de polinomio ciclotómico, y que van a permitirnos después ir al teorema AKS.

Definición 12. Sea r un número primo. Se define el r-ésimo polinomio ciclotómico y se denota por K_r al polinomio $K_r := x^{r-1} + x^{r-2} + \cdots + x + 1$. Dicho polinomio se obtiene al sacar el factor $x - 1$ del polinomio $x^r - 1$.

Entre otras propiedades de dicho polinomio K_r, conviene destacar que es irreducible en $\mathbb{Z}[x]$.

Lema 10. *Sean p y r dos números primos distintos, y H un factor irreducible módulo p de K_r. Entonces, $x^r \equiv 1$ mod(p, H) y $x^k \not\equiv 1$ mod(p, H) para todo k con $1 \leq k \leq r - 1$.*

Demostración. Comenzamos justificando la primera de las congruencias. Tenemos claramente que $x^r \equiv 1 \bmod(x^r - 1)$. Ahora bien, sabemos que H es un divisor de $x^r - 1$ módulo p, luego $x^r \equiv 1 \bmod(p, H)$.
Sea ahora k el menor entero tal que $x^k \equiv 1 \bmod(p, H)$. Veamos que, en estas cicunstancias, necesariamente k divide a r. Podemos hacer la división euclídea, obteniendo $r = k \cdot t + r_0$, donde $r_0 < k$. Si $r_0 = 0$, no hay nada más que hacer, en cambio, si no es 0, podemos escribir lo siguiente: $1 \equiv x^r = x^{k\cdot t + r_0} \equiv x^{r_0} \bmod(p, H)$. Pero, por hipótesis, k era el menor entero que cumplía eso, y $r_0 < k$, luego necesariamente, $r_0 = 0$, y se tiene que lo que se quería.
Luego, dado que r es primo, sólo hay dos opciones, $k = 1$ ó $k = r$. Razonemos por reducción al absurdo, suponiendo que $k = 1$. Entonces, $x - 1 \equiv 0 \bmod(p, H)$, y por tanto, $x - 1$ es divisor de K_r módulo p. Es decir, $K_r(1) \equiv$

$0 \bmod(p)$, pero $K_r(1) = r$. Se llega a una contradicción, ya que suponemos que r y p son primos distintos y entonces, $k = r$ y queda demostrado.

□

Corolario 3. *Sean r y p dos números primos distintos y H un factor irreducible módulo p de K_r. Sea n un múltiplo de p con $mcd(n, r) = 1$. Entonces $ord_r(n) \leq t \leq r$.*

Demostración. Para ver la primera desigualdad, hay que usar que $k = ord_r(n)$ nos indica el número de elementos distintos de la forma n^j módulo r, vamos a justificar esto. Si tenemos $n^h \equiv n^j \bmod(r)$ con $0 < h, j < k$, entonces se tendría que $n^{h-j} \equiv 1 \bmod(r)$, y por tanto, dado que k es el menor que cumple eso, k divide a $h - j$, y, por tanto, existe un número a tal que $h = a \cdot k + j$, luego $h > k$, lo cuál no tiene sentido con lo que hemos supuesto al principio. Luego, si m_1, m_2 son dos potencias de n distintas módulo r, por el lema previo, $x^{m_1} \not\equiv x^{m_2} \bmod(p, H)$, luego se tiene la desigualdad.
La segunda desigualdad es consecuencia inmediata de la definición de t y de que, por el lema anterior, hay como mucho r polinomios de la forma x^m diferentes módulo (p, H).

□

Estamos ya por tanto, en condiciones de enunciar y demostrar el teorema AKS.

Teorema 19 (Teorema AKS)**.** *Sea n un número entero del cuál se quiere saber si es primo. Sean r un número primo tal que $mcd(r, n) = 1$ y $ord_r(n) > 4 \cdot log(n)^2$, y $Q := x^r - 1$. Si n no es potencia de un primo p, entonces hay como mucho $r - 2$ polinomios de la forma $P = x + a$, con a en $\{0, \ldots, p-1\}$ tales que $(P(x))^n \equiv P(x^n)$ $mod(p, Q)$.*

Demostración. Razonamos por reducción al absurdo, viendo que si se cumplen más de esas congruencias, entonces n es primo o potencia de un primo. Supongamos que n no es primo, es decir, que existe un primo p que divide a n. Vamos a ver que, bajo estas hipótesis, n es necesariamente una potencia de p.
Sea r un primo cumpliendo las hipótesis del enunciado. Dado que $mcd(n, r) = 1$ y que p divide a n, entonces $r \neq p$. Además, haciendo uso de las desigualdades del Corolario 3, se tiene que $4 \cdot (log(n))^2 < t \leq r$.
Ahora, si suponemos que $\ell \geq r - 1$, entonces tenemos que $\ell \geq r - 1 \geq t - 1$, y por tanto, por el corolario 2, se concluye que n es potencia de p.

□

Por tanto, lo que hemos visto con el teorema es que, bajo esas hipótesis, se tiene que, si se cumplen $r-1$ congruencias ó más, necesariamente n va a ser potencia de un primo o primo, pero vamos a comprobar antes de calcular dichas congruencias en el algoritmo que no sea potencia de un primo, luego, nos bastará con ver que se cumplen $r-1$ congruencias distintas para determinar si el número n es primo.

2.3. Algoritmo AKS

Para finalizar este capítulo, lo que vamos a hacer es presentar el algoritmo, y después vamos a ver que funciona, y por otro lado, que es eficiente.

Presentamos primero el algoritmo y luego vamos a justificar cada uno de los pasos.

TEST AKS:
Input: un número entero $n \geq 2$.

1. Si n es potencia de un primo $p < n$, entonces n es compuesto.
2. Si no, para los primos $r = 2, 3, 5, \ldots$ hacemos:
   - *a*) Si r divide a n y $r < n$, entonces n es compuesto, y finalizamos.
   - *b*) Si $r \geq n$, n es primo, y finalizamos.
   - *c*) Si no, se calcula $ord_r(n)$.
   - *d*) Si $ord_r(n) > 4 \cdot (log(n))^2$, $Q := x^r - 1$, y se pasa al paso 3, si no, se repite el paso 2 con el r siguiente.
3. Se comprueban las congruencias $(x+a)^n \equiv x^n + a \bmod(n, Q)$ para los enteros a entre 1 y $r-1$.
4. Si una de las congruencias no se satisface, entonces n es compuesto.
5. Si no, n es primo.

Se ve claramente que el AKS es un test determinista, ya que no se involucran elecciones aleatorias, y la respuesta es siempre que, ó bien n es

compuesto, ó bien es primo.

Ahora bien, para ver la eficiencia del mismo, hay que fijarse en todos los pasos del algoritmo y ver si se pueden hacer de manera eficiente. Lo justificamos en el Teorema 21.

Vamos a justificar que el encontrar el r que satisfaga lo que se quiere para luego usarlo en el polinomio Q se puede hacer de manera eficiente.

Definición 13. Sea $n \geq 2$ y k un número natural. Se denota por $r(n,k)$ al primo más pequeño r tal que, ó bien r divide a n y ó bien $ord_r(n) > k$.

Observación 14. Si $ord_p(n) = m$, entonces eso significa que $n^m \equiv 1 \bmod(p)$, ó lo que es lo mismo, que $n^m - 1$ es múltiplo de p.

Sea N el entero $N := \prod_{m \leq k} (n^m - 1)$. N es múltiplo de todos los primos p tales que $ord_p(n) \leq k$, en particular de los primos $p < r(n,k)$. Si tomamos $\Pi := \prod_{\substack{p \ primo \\ p<r(n,k)}} p$, por las definiciones previas, se tiene que $\Pi \leq N$.

Con esto, vamos por tanto a ver la complejidad de calcular $r(n,k)$.

Teorema 20. *Se tiene que* $r(n,k) = \mathcal{O}(k^4 \cdot (log(n))^2)$.

Demostración. Denotaremos $r = r(n,k)$ por comodidad en la demostración. Vamos a usar ciertos resultados y notaciones de los números primos que aparecen en el Capítulo 1.
Para empezar, sabemos que 2 es el menor de todos los primos y que $\pi(r-1)$ el número de primos que son menores o iguales que $r-1$ (ver Definición 11). Entonces claramente tenemos que, $\Pi \geq 2^{\pi(r-1)}$. Ahora, por el Teorema 11, se tiene que $\pi(r-1) \geq C \cdot \frac{r-1}{log(r-1)}$, donde C es una constante positiva. Además, sabemos que $log(r) > log(r-1)$ y que $r - 1 > \frac{r}{2}$. Por tanto, juntando todo esto se tiene la siguiente cadena de desigualdades:

$$\Pi \geq 2^{\pi(r-1)} \geq 2^{\frac{C \cdot (r-1)}{log(r-1)}} > 2^{\frac{C \cdot (r-1)}{log(r)}} > 2^{\frac{C \cdot r}{2 \cdot log(r)}}.$$

Por otro lado, por definición de N:

$$N < \prod_{m \leq k} n^m = n^{\frac{k \cdot (k+1)}{2}} < n^{\frac{(k+1)^2}{2}} = 2^{\frac{(k+1)^2 \cdot log(n)}{2}}.$$

Ahora, sabíamos que $\Pi \leq N$, luego, haciendo uso de las desigualdades que acabamos de ver para ambas cantidades, se llega a que $2^{\frac{C \cdot r}{2 \cdot log(r)}} < 2^{\frac{(k+1)^2 \cdot log(n)}{2}}$, es decir, $\frac{r}{log(r)} < \frac{(k+1)^2 \cdot log(n)}{C}$, ya que $C > 0$.
Por último, usamos que $(log(r))^2 = \mathcal{O}(r)$, y de aquí se obtiene que $r \cdot (log(r))^2 = \mathcal{O}(r^2)$. Por lo tanto, $r = \mathcal{O}(\frac{r^2}{(log(r))^2})$, y haciendo uso de la desigualdad obtenida previamente para $\frac{r}{log(r)}$ elevando al cuadrado, se obtiene que $r = \mathcal{O}((k+1)^4 \cdot (log(n))^2) = \mathcal{O}(k^4 \cdot (log(n))^2)$, lo que se quería probar.

□

Vamos a enunciar y demostrar a continuación un lema que será importante de cara a la demostración del posterior teorema, y que tiene que ver con el cálculo eficiente de las congruencias que aparecen en el algoritmo AKS.

Lema 11. *Sea $n \geq 2$ un número natural y P, Q polinomios con coeficientes en $\{0, \ldots, n-1\}$ de grados t y r respectivamente. Entonces se da la congruencia siguiente:*

$$(P(x))^n \equiv P(x^n) \ mod(n, Q),$$

y su tiempo de ejecución es polinomial en r, t y $log(n)$.
Además, si suponemos que t y r no llevan a una complejidad exponencial con respecto a n, entonces lo que acabamos probar es que la comprobación de la congruencia es eficiente.

Demostración. Vamos a ver que la división de dos polinomios P, Q como los del enunciado, al hacerla módulo n se puede hacer de manera eficiente.
Escribimos $P = c_0 + c_1 \cdot x + \cdots + c_t \cdot x^t$ y $Q = d_0 + d_1 \cdot x + \cdots + d_r \cdot x^r$ y suponemos que c_t es coprimo con n. Se quieren encontrar T y R polinomios tales que $Q \equiv T \cdot P + R \bmod(n)$.
Estamos ante varios casos; si $r < t$, entonces $T = 0, R = Q$; si no, se definen $k := r - t$ y $a = \frac{d_r \bmod(n)}{c_t}$, y es claro que entonces T es de grado k tiene como coeficiente dominante a a. Para los términos que faltan por determinar de T y R se hace $Q' = Q - a \cdot P \cdot x^k$ y se vuelve a empezar, así hasta estar en el primer caso. Dadas las operaciones que aquí aparecen, y que se repiten como mucho $r - t$ veces, estamos ante un procedimiento con tiempo polinomial con respecto a $log(n)$, r y t.
Ahora bien, si se suponen r y t que no llevan a una complejidad exponencial respecto a n, entonces vamos a ver que es eficiente.
Se tiene que las sumas de estos polinomios módulo n se pueden hacer de manera eficiente , y, dado que la potenciación modular lo es también, el hacer potencias de P y reducir módulo Q y n se puede hacer de manera eficiente también, ya que el grado siempre lo tenemos acotado también, no sólo los coeficientes. Por tanto, sabemos calcular de manera eficiente $(P(x))^n \bmod(n, Q)$.

A partir de aquí, podemos buscar el único polinomio que es congruente con este módulo (n, Q). Para ello, hacemos $P(x^n)$ (sabemos que se puede hacer de manera eficiente ya que calcular las potencias y reducir módulo (n, Q) cada monomio es eficiente según lo que acabamos de ver). Y luego hay que sumar $t + 1$ polinomios y reducir módulo (n, Q), lo cuál es eficiente.
Ahora bien, lo último que hay que hacer es comparar los coeficientes de $P(x^n)$ y $((P(x))^n$, que sabemos que, por la elección de r, que es polinomial con $log(n)$, nos lleva a un número de pasos que es posible hacer de manera eficiente.

□

Teorema 21. *El algoritmo AKS es determinista y eficiente. Además, el algoritmo devuelve como respuesta que n es primo si, y sólo si, n es primo.*

Demostración. Veamos la primera parte del teorema.
El paso 1 se puede llevar a cabo de manera eficiente. Vamos a necesitar calcular el número $m_k = \lfloor \sqrt[k]{n} \rfloor$, para diferentes valores de k. Lo que realmente buscamos es tener que $n = m^k$ para algún m natural y un $k > 1$ natural también. Por ello, basta con calcular m_k y elevarlo a k (sabemos que esto se puede hacer de manera eficiente), obteniendo así que si $m_k^k = n$ se ha encontrado que n es potencia de m y si no es que $m_k^k > n$ y se pasa al siguiente k. Esto para, como mucho, $log(n)$ valores de k (ya que se hace para los m_k con $m_k^k < n$). Por tanto, este paso es eficiente.
Para el paso 2, según la notación anterior, si nos fijamos en el r que se usa en el algoritmo, sería $r(n, 4 \cdot (log(n))^2)$, y por tanto, haciendo uso del teorema anterior, $r(n, 4 \cdot (log(n))^2) = \mathcal{O}(log(n)^{10})$, con lo cual, llegar al r necesario es eficiente (además nótese que al ir cogiendo r primo de manera ascendente, eso no causa ningún problema en cuanto a la eficiencia). Para el apartado a) es suficiente con aplicar el algoritmo de Euclides para calcular el $mcd(r, n)$ de manera eficiente. Sabemos que para el cálculo de $ord_r(n)$ hay que hacer como mucho $r - 2$ multiplicaciones por n módulo r (por cómo es r, hacerlo de esta forma no causa problemas de eficiencia), ya que el pequeño teorema de Fermat nos asegura que $n^{r-1} \equiv 1 \bmod(r)$, por lo que también se realiza de manera eficiente el apartado c). Luego, este paso 2 es eficiente.
En cuanto al tercer y último paso que hay que justificar, se tienen que hacer $r - 1$ congruencias, luego como hemos visto antes, dicho número de pasos no nos causa problemas. Por tanto, sólo hace falta justificar que las congruencias se pueden hacer de forma eficiente. Pero eso es justamente lo que hemos visto en el Lema 11.
Por tanto, se tiene que el algoritmo es eficiente.
Veamos ahora que el algoritmo da siempre la respuesta correcta.

Supongamos en primer lugar que el algoritmo nos da como respuesta que n es compuesto, vamos a ver que entonces n es compuesto. El algoritmo puede habernos dado esa respuesta en diferentes pasos. Puede que haya visto que es potencia de un primo (con exponente mayor que 1), luego n no sería primo. Otra opción es que se haya encontrado un r no trivial que divida a n, lo cual nos indica que n no sería primo tampoco en este caso. Y por último, sería porque no se satisface una de las congruencias del paso 3, luego, por el Teorema 14, se concluye que n sería compuesto. Por tanto, en los tres casos se llegaría a que n es compuesto.
Finalmente supongamos que el algoritmo nos ha dado como respuesta que n es primo. Esta respuesta sólo se da en dos casos. El primero de ellos sería porque se han recorrido todos los $r < n$ y no se ha encontrado ningún divisor no trivial de n, por tanto, se tendría que n es primo. El otro caso sería porque se cumplen las r congruencias del paso 3 para un r adecuado al teorema AKS, y entonces como se ha comprobado al principio del algoritmo que n no es potencia de un primo, se tiene que, necesariamente, n es primo.

□

Observación 15. A continuación, va a aparecer la notación $\mathcal{O}^{\sim}$. Se tiene que $\mathcal{O}^{\sim}(log^k(n)) = \mathcal{O}(log^k(n) \cdot P(log(log(n))))$, donde P es un polinomio.

Aquí lo que hemos visto por tanto es que el algoritmo es eficiente, pero no hemos hablado nada del número de pasos que se realizan en la ejecución del mismo. En el artículo original de Agrawal, Kayal y Saxena ([1]), demuestran que la complejidad del algoritmo es $\mathcal{O}^{\sim}(log^{\frac{21}{2}}(n))$, aunque también son capaces de reducirlo a $\mathcal{O}^{\sim}(log^{\frac{15}{2}}(n))$. Además, comentan que, suponiendo cierta una conjetura que hace referencia a la densidad de los primos de Sophie Germain (números primos p tales que $2 \cdot p + 1$ son primos también), se puede reducir la complejidad del algoritmo hasta $\mathcal{O}^{\sim}(log^6(n))$. Cabe destacar que esta complejidad fue alcanzada por Hendrik Lenstra y Carl Pomerance para este algoritmo AKS llevando a cabo ciertas modificaciones sobre el original. Aunque se puede intentar reducir aún más la complejidad. Para ello, se trabaja sobre la siguiente conjetura: *si r es un número primo que no divide a n y $(x-1)^n \equiv x^n - 1$ mod$(n, x^r - 1)$, entonces n es primo ó $n^2 \equiv 1$ mod(r).*
Si fuera cierta entonces se podría bajar la complejidad hasta $\mathcal{O}^{\sim}(log^3(n))$. Sin embargo, se cree que no es cierta debido a un argumento presentado por Lenstra y Pomerance, aunque se podría trabajar con esta idea de la conjetura haciendo alguna modificación.

Apéndice A

Complejidad

A la hora de afrontar un problema y tener que recurrir a un ordenador para su resolución, es inevitable hablar de la Teoría de la Complejidad. Es por ello que en este apéndice, hablaremos sobre algunas cosas básicas e importantes, que nos servirán de guía para la mejor comprensión del trabajo. Para comenzar, necesitamos definir una de las estructuras más básicas dentro de este ámbito, la noción de algoritmo.

Definición A 1. Un algoritmo es una secuencia finita de pasos elementales, dónde cada uno de ellos viene únicamente determinado por el input y los pasos previos.

Observación A 1. Los algoritmos sobre los que hablaremos y trabajaremos tendrán un input numérico.

Los algoritmos se usan para resolver un problema (en nuestro caso, de índole matemática, pero la noción de algoritmo no sólo es usada en este contexto). Hay distintos tipos de clasificaciones de dichos problemas. Veremos algunas de ellas.

Se dirá que un problema es *decidible* cuando éste pueda ser resuelto por un algoritmo. En caso contrario, se dirá que el problema es *indecidible*. Lógicamente, lo que nosotros trataremos aquí serán todo problemas decidibles. Otra posible disociación de los problemas viene dada por la naturaleza de la respuesta que se espera al mismo. Es decir, por un lado, tenemos los llamados *problemas de decisión*, los cuáles tienen como respuesta SÍ ó NO (en algunos casos las respuestas afirmativa ó negativa se pueden ver identificadas con 1 y 0 respectivamente). Por otro lado, los que no son de decisión, se llaman *problemas de búsqueda*. Es posible ver que todo problema de búsqueda se puede reducir a uno de decisión, pero no entramos aquí en esos detalles.

Ejemplo A 1. Dado un número n, supongamos que queremos determinar si es primo o no. Un algoritmo A resuelve el problema si para una instancia n (input de A) la respuesta es $A(n) = SÍ$, si, y sólo si, n es primo, y la respuesta es $A(n) = NO$, si, y sólo si, n es compuesto.
Por tanto, se tiene un ejemplo de un problema de decisión.

Ejemplo A 2. En cuanto a un problema de búsqueda, se tiene el buscar un factor de un número dado n, ya que la respuesta esperada es un divisor de dicho número.

Definición A 2. La complejidad computacional es el estudio y tratamiento del tiempo de ejecución de un algoritmo.

Observación A 2. En la definición previa se habla de tiempo de ejecución; cabe destacar que, ese tiempo se refiere al número de operaciones básicas que el ordenador necesita realizar entre dos bits (operaciones bit) mientras se ejecuta el algoritmo.

Para medir dicho tiempo, se recurre a la notación $\mathcal{O}$, la cuál definimos, aunque antes damos otra noción importante, sobre cómo representar esos tiempos de ejecución. Si no se dice lo contrario, n hará referencia a la longitud del número en cuanto a dígitos, no al número en sí.

Definición A 3. La complejidad de un algoritmo es una función $f : \mathbb{N} \longrightarrow \mathbb{R}$ siendo $f(n)$ el número máximo de operaciones bit requeridas por el algoritmo cuando el input tiene como longitud n bits.

Definición A 4. Sean $f, g : \mathbb{N} \longrightarrow \mathbb{R}$ dos funciones, se denota $f(n) = \mathcal{O}(g(n))$, y se dice que f es una O grande de g (ó que f es el orden de g) si existen un número natural n_0 y una constante C mayor que 0, tales que $f(n) \leq C \cdot g(n)$ para todo n mayor o igual que n_0.

Ejemplo A 3. Tomamos $f(n) = n^2$, y $g(n) = n^3$. En este caso, tenemos que $f(n) = \mathcal{O}(g(n))$ ya que podemos coger $n_0 = 1$ y $C = 1$, y entonces es claro que se tiene $n^2 \leq n^3$ si $n \geq 1$.

Una vez vistas estas definiciones, vamos a hablar de lo que es en sí la eficiencia de un algoritmo, característica que va a ser muy importante a la hora de que un algoritmo sea útil o no. Es decir, sólo nos interesan aquellos que son eficientes, porque, como veremos, son aquellos que tienen un tiempo de ejecución razonable. Antes de eso, vamos a decir qué es un algoritmo de complejidad polinomial.

Definición A 5. Se dice que un algoritmo es de complejidad polinomial (o polinómica) si $f(n) = \mathcal{O}(P(n))$, donde $P(n)$ es un polinomio. Un algoritmo es eficiente si es de complejidad polinomial.

Observación A 3. Atendiendo a la definición precedente y a lo visto hasta ahora, se tiene que eso es equivalente a que $f(n) = \mathcal{O}(n^k)$ para k el grado del polinomio $P(n)$ mencionado en la definición anterior.

Por contraposición a esto, vamos a definir la complejidad exponencial de un algoritmo.

Definición A 6. Se dice que un algoritmo es de complejidad exponencial si $f(n) = \mathcal{O}(a^n)$, donde a es una constante.

Ejemplo A 4. Para ver que un número es primo, una de las opciones que hemos visto que era ir diviendo al número n (de longitud k) hasta $\sqrt{n}$, y ver si alguno de esos números es divisor. Se tendría que dicho algoritmo tendría complejidad $\mathcal{O}(n^{\frac{1}{2}})$, lo cuál es equivalente a decir que es $\mathcal{O}(2^{0,5 \cdot log(k)})$. Por tanto, la complejidad es exponencial.

Ejemplo A 5 (Método 1 de multiplicación de enteros)**.** Uno de los algoritmos más simples sobre el cuál podemos trabajar a la hora de ver su complejidad es el de la multiplicación de dos números enteros. Veamos primero la complejidad del algoritmo más conocido.
Dados $n = n_k n_{k-1} \dots n_1$ y $m = m_l m_{l-1} \dots m_1$ (suponiendo que $k \geq l$, si no se intercambian los papeles de n y m), tendríamos la siguiente situación:

				n_k	$\cdots$	$\cdots$	n_2	n_1
$\times$					m_l	$\cdots$	m_2	m_1
				$m_1 \cdot n_k$	$\cdots$	$\cdots$	$m_1 \cdot n_2$	$m_1 \cdot n_1$
			$m_2 \cdot n_k$	$\cdots$	$\cdots$	$m_2 \cdot n_2$	$m_2 \cdot n_1$	
		$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$		
$+$		$m_l \cdot n_k$	$\cdots$	$\cdots$	m_2	$m_l \cdot n_1$		
	d_{l+k+1}	d_{l+k}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	d_2	d_1

Donde hemos puesto como d_i el resultado de la suma de la columna i.
Tendríamos a lo sumo ℓ filas, ya que hay ℓ cifras en m. En cada fila hay k multipliaciones bit. Luego, $k \cdot l$ multiplicaciones bit en total en este primer paso. Una vez hecho esto, ahora tenemos que llevar a cabo la suma. Tenemos que sumar ℓ dígitos por columna, y hay $k + \ell$ columnas, luego si juntamos todas las operaciones a realizar, nos salen como mucho $k \cdot \ell + \ell \cdot (k + \ell) = \ell \cdot (2k + \ell)$ operaciones bit. Ahora bien, estábamos suponiendo que $\ell \leq k$,

luego $\ell \cdot (2k + \ell) \leq 3k^2$.
Con esto podemos por tanto decir que la complejidad computacional de este algoritmo es $\mathcal{O}(3k^2) = \mathcal{O}(k^2)$.
Y podemos concluir que este algoritmo de multiplicación tiene complejidad polinomial.

Ejemplo A 6 (Método 2 de multiplicación de enteros)**.** Acabamos de ver la multiplicación de enteros tal cual lo hacemos a mano. Sin embargo, hay otros métodos hoy en día que tienen complejidad computacional más baja y realmente son los usados por varios programas. Vamos a ver uno de ellos; el algoritmo de Karatsuba.
En realidad vamos a ver que, aparentemente, la complejidad es casi idéntica a la anterior, pero un pequeño cambio con respecto al anterior se vuelve importante a tener en cuenta cuando hay que repetir el algoritmo muchas veces.
Supongamos que n y m son los números enteros que queremos multiplicar, con k la longitud de n un número par y siendo mayor o igual que ℓ, la longitud de m. Entonces, se pueden escribir ambos números de la siguiente forma:

$$n = a \cdot 2^{\frac{k}{2}} + b \qquad m = c \cdot 2^{\frac{k}{2}} + d.$$

Donde a, b, c y d son números enteros de longitud menor o igual que $\frac{k}{2}$. Entonces, podemos realizar la multiplicación con esta representación, teniendo que:

$$n \cdot m = a \cdot c \cdot 2^k + (a \cdot d + b \cdot c) \cdot 2^{\frac{k}{2}} + b \cdot d.$$

Ahora, podríamos pensar que ya tenemos lo necesario y ponernos a ver la complejidad, pero si nos fijamos, ahí tendríamos que hacer cuatro multiplicaciones distintas, entre los números a, b, c y d, y luego hacer las multiplicaciones por 2^k y $2^{\frac{k}{2}}$. Lo que ocurre es que hay otra manera de escribir el término que acompaña a $2^{\frac{k}{2}}$:

$$a \cdot d + b \cdot c = a \cdot c + b \cdot d - (a - b) \cdot (c - d).$$

De esta manera, sólo tenemos que calcular $a \cdot c$ y $b \cdot d$, y luego hay que hacer otra multiplicación de números enteros que vienen de restas. Pero eso es mejor que tener cuatro multiplicaciones de enteros. Y dichas multiplicaciones son entre enteros de longitud $\frac{k}{2}$. Lo que es multiplicar por potencias de 2 es simplemente añadir tantos ceros como sea el exponente.
Veamos ahora la complejidad de dicho algoritmo. Definimos $T(k)$ como el coste computacional de realizar la multiplicación de dos enteros de longitud k. Entonces, teniendo en cuenta que tenemos que hacer 3 multiplicaciones de

enteros de longitud $\frac{k}{2}$ y sumas (o restas), entonces podemos escribir:

$$T(k) = 3 \cdot T(\frac{k}{2}) + \mathcal{O}(k).$$

donde $\mathcal{O}(k)$ hace referencia al coste de las sumas y restas. Ahora bien, podemos aplicar lo mismo ahora a $T(\frac{k}{2})$ y se tendría ahora que:

$$T(k) = 3 \cdot (3 \cdot T(\frac{k}{4}) + \mathcal{O}(\frac{k}{2})) + \mathcal{O}(k) = 3^2 \cdot T(\frac{k}{4}) + 3 \cdot \mathcal{O}(\frac{k}{2}) + \mathcal{O}(k).$$

Aplicando esto sucesivamente, en el paso i nos encontraríamos con

$$T(k) = 3^i \cdot T(\frac{k}{2^i}) + 3^{i-1} \cdot \mathcal{O}(\frac{k}{2^{i-1}}) + \cdots + \mathcal{O}(k).$$

Llegados a este punto, hay que pensar cuándo se va a parar, y eso va a ser cuando ya no se pueda dividir a k por 2 elevado a algo, es decir, cuando $i = log_2(k)$.
Por tanto, llegamos a que:

$$T(k) = 3^{log_2(k)} \cdot T(\frac{k}{2^{log_2(k)}}) + \sum_{j=0}^{log_2(k)-1} (3^j \cdot \mathcal{O}(\frac{k}{2^j})) =$$

$$= 3^{log_2(k)} \cdot T(1) + \mathcal{O}(k) \cdot \sum_{j=0}^{log_2(k)-1} (\frac{3}{2})^j =$$

$$= 3^{log_2(k)} \cdot T(1) + \mathcal{O}(k) \cdot \frac{(\frac{3}{2})^{log_2(k)} - 1}{\frac{3}{2} - 1} =$$

$$= 3^{log_2(k)} \cdot T(1) + \mathcal{O}(3^{log_2(k)}) =$$

$$= \mathcal{O}(3^{log_2(k)}) = \mathcal{O}(k^{log_2(3)}) = \mathcal{O}(k^{1,58}).$$

En la penúltima igualdad hemos usado la relación $3^{log_2(k)} = k^{log_2(3)}$.

Observación A 4. Si nos fijamos en los ejemplos anteriores, hemos trabajado con el número en base 2 (ya que trabajamos con operaciones bit), dejando de lado la base 10. Esto será algo a tener en cuenta constantemente cuando se vea la complejidad de cualquier algoritmo, para así determinar el coste computacional.
Por ello también, los logaritmos que aparecen a lo largo del trabajo como log hacen referencia a log_2, como ya se ha mencionado anteriormente.

Observación A 5. Hemos visto dos algoritmos para la multiplicación de dos números enteros, ambos con complejidad polinomial. Sin embargo, existe otro método que tiene mejor complejidad computacional, el de Schönage-Strassen. Este es el mejor en cuanto a que tiene la mejor complejidad computacional, sin embargo, en la práctica no será de gran utilidad para números con menos de 10.000 bits.
Esto se debe al uso de la notación $\mathcal{O}$, ya que estamos dependiendo de una constante C. No estamos haciendo otra cosa que ver cómo se comporta el número de pasos a realizar de un algoritmo asintóticamente, es decir, cuando se tiende a infinito.

Observación A 6. Acabamos de ver que el algoritmo de multiplicación (cualquiera de ellos) tiene complejidad polinómica, y además implícitamente se ha visto que para la suma el algoritmo tiene complejidad polinómica también. Luego, con esto podemos concluir que las cuatro operaciones aritméticas básicas: suma, resta, multiplicación y división; tienen todas complejidad polinómica (la resta y la división tienen el mismo tiempo que la suma y la multiplicación respectivamente).

Además de las operaciones aritméticas elementales, otros algoritmos que tienen complejidad computacional polinómica son el algoritmo de Euclides, el cálculo de inverso modular, o la potenciación modular. Para el cálculo del algoritmo de Euclides entre dos enteros a y b, llegamos a que la complejidad es $\mathcal{O}(log^3(a))$. En cuanto al cálculo del inverso modular, si tenemos un entero n y otro entero a tal que $mcd(a, n) = 1$ y $a < p$, entonces, realizar el cálculo de $a^{-1} \bmod(n)$ tiene como complejidad $\mathcal{O}(log^3(n))$. Por último, si se quiere calcular $a^n \bmod(p)$, con a, n, p enteros, haciendo uso del algoritmo de cuadrados repetidos se llega a una complejidad de $\mathcal{O}(log(n)) \cdot \mathcal{O}(log^2(p))$.

Observación A 7. Es importante recalcar que nosotros estamos continuamente hablando de la complejidad temporal de los algoritmos, pero no es la única. Se podría tratar también la complejidad espacial, es decir, en vez de medir las operaciones bit, medir el espacio de almacenamiento (en bits) que requiere el algoritmo a lo largo de su ejecución. Se tendría en cuenta que hay momentos donde se liberaría memoria por la no necesidad futura del uso de lo que se tiene almacenado en ese momento, pero podría haber otras veces que hubiera que almacenar muchos datos sin posibilidad de limpiarlos así como así.
El problema por tanto que se puede presentar aquí es el hecho de superar la capacidad de almacenamiento de las máquinas con las que se esté trabajando, además de que el tener muchos datos almacenados, luego no significa que la búsqueda de uno de ellos fuera a ser fácil y rápida.

A.1. Clases de complejidad

Cuando hemos hablado anteriormente de posibles clasificaciones de los problemas, no hemos mencionado la que tiene más importancia para nuestro trabajo, que es la división entre las clases de complejidad P y NP.
La clase de complejidad P es el conjunto de problemas decidibles, que pueden ser resueltos por un algoritmo con complejidad polinomial.
La clase de complejidad NP es el conjunto de problemas de decisión que, dado un candidato a solución, puede ser comprobado si lo es o no por un algoritmo de complejidad polinomial.
A partir de aquí, se puede deducir que, en particular, todo problema de la clase P se encuentra también en la clase NP ($P \subset NP$). Sin embargo, acerca de la otra contención no se sabe si es estricta o no. Es uno de los problemas más importantes de los últimos tiempos (de hecho, es uno de los siete problemas del milenio propuestos por la Fundación Clay, y cuya solución tiene como premio un millón de dólares) y para el cuál todavía no se tiene una respuesta.
Por tanto, a día de hoy es importante saber si un problema se encuentra o no en la clase de complejidad P.

Ejemplo A 7. Uno de los problemas más típicos que se sabe que está en la clase NP pero no si está en la clase P es el siguiente.
Supongamos que tenemos un mapa con diferentes países, y queremos colorear el mapa con tres colores, de manera que los países colindantes nunca tengan el mismo color. Este problema presenta la incoveniencia de que, si se empieza a hacer lo pretendido, puede que lleguemos a dos países que comparten frontera y tienen el mismo color, pero eso puede haber dependido de nuestra elección. Por tanto, nada implica que el problema no se pueda resolver de otra manera. Por ello, no se sabe que el problema esté en la clase P.
En cambio, si se tiene un candidato a solución, es fácil ver si es correcta o no, atendiendo al mapa presentado y que no haya países vecinos que estén coloreados con el mismo color. Luego, eso implica que el problema sí pertenece a la clase NP porque es verificable de manera eficiente y sin embargo, no se sabe si está en la clase P.

A.2. Algoritmos probabilísticos y deterministas

Muchas veces es difícil encontrar algoritmos eficientes (o útiles en la práctica) que nos den una respuesta al problema planteado de manera cer-

tera, sin error posible. Cuando esto ocurre, nos tenemos que remitir a la probabilidad, es decir, si no podemos asegurar algo pero podemos dar una respuesta con una cierta probabilidad de error, pues es lo que se hará.
Entran por tanto en juego los algoritmos probabilísticos, que se unen a los llamados algoritmos deterministas.

Definición A 7. Se dice que un algoritmo es determinista si no trabaja con elecciones aleatorias de ningún tipo. Por tanto, la respuesta que proporcionarán estos algoritmos será segura, sin ninguna probabilidad de error.

Definición A 8. Se dice que un algoritmo es probabilístico si trabaja con elecciones aleatorias. En este caso, la respuesta afirmativa al problema puede estar sujeta a una probabilidad de error estrictamente positiva.

Observación A 8. Al hablar de elección aleatoria nos referimos a un experimento aleatorio con dos opciones que tienen la misma probabilidad.

Cuando a lo largo del trabajo se habla de los test de primalidad, simplemente hay que tener en cuenta que un test no deja de ser otra cosa que un algoritmo, y por tanto, podemos aplicar todo lo que se ha visto en este apéndice (diferenciar entre deterministas y probabilísticos, ver su complejidad...).

A continuación, vamos a hablar de los distintos tipos de algoritmos probabilísticos que hay. Hay dos casos a tener en cuenta cuando se habla de este tipo de algoritmos, y en función de las características, se llamarán de tipo Montecarlo o tipo Las Vegas.

Definición A 9. Un algoritmo probabilístico es de tipo Montecarlo si su complejidad computacional es polinómica y proporciona una solución correcta al problema propuesto con una cierta probabilidad $p > \frac{1}{2}$.

Ejemplo A 8. Como ejemplo de un algoritmo probabilístico tipo Montecarlo, vamos a presentar brevemente el algoritmo de factorización de un entero grande n, llamado ρ de Pollard.
Este algoritmo se usa para la obtención de un divisor d, pequeño en comparación con el entero n con el que se está trabajando. La idea es coger una semilla x_0 que sea un número aleatorio entre 0 y $n-1$. Y luego, $x_i \equiv f(x_{i-1}) \bmod(n)$ para i natural, donde $f(x) = x^2 \pm a$, con a un número entero distinto de 0 y -2.
Una vez se tiene esto, el método se basa en que, existirán probablemente

x_i, x_j para índices distintos tales que $x_i \equiv x_j \bmod(d)$, y $x_i \not\equiv x_j \bmod(n)$, sabiendo que d es el divisor pequeño de n. Y en ese caso el $mcd(x_i - x_j, n)$ será un divisor no trivial de n.
Como en teoría no se conoce d, lo que hay que hacer es ir calculando $mcd(x_i - x_j, n)$ para todo $j \leq i$ y cuando sea distinto de 1, eso será un divisor no trivial de n.

Definición A 10. Un algoritmo probabilístico es de tipo Las Vegas si proporciona siempre una solución correcta al problema propuesto y el tiempo de ejecución esperado está acotado por un polinomio Q. En lo que respecta al tiempo de ejecución máximo, no se tiene ninguna información sobre él.

Ejemplo A 9. Uno de los casos donde puede aparecer un algoritmo de tipo Las Vegas es cuando se tiene el problema de ordenar una lista numérica por tamaño. Si uno se plantea este problema, una opción es comparar todos los números e ir poniendo al más pequeño el primero, luego repetir el proceso y ponerlo el segundo, etc.
Pero hay otra opción que se basa en lo que se llama "divide y vencerás", es decir, se coge un número al azar x y por un lado se trata una lista con los números más pequeños que x, y por otro lado, una lista con los más grandes. Haciendo esto sucesivamente, se mejora la complejidad del algoritmo, aunque ya se empieza a hablar de tiempo de ejecución promedio (debido a la elección aleatoria de x), en vez de lo que se ha estado hablando hasta ahora. Ya que, de media, este algoritmo tendrá una complejidad de $\mathcal{O}(k \cdot log(k))$, teniendo en cuenta cómo se elige x.
Como vemos, la respuesta será siempre cierta, y por tanto es claramente un algoritmo de tipo Las Vegas.

Bibliografía

[1] M. Agrawal, N. Kayal, N. Saxena. PRIMES is in P. Annals of Mathematics, 160 (2004), 781-793.

[2] J.A. Buchmann. Introduction to cryptography. Second Edition. Springer. 2004.

[3] R. Crandall, C. Pomerance. A computational prespective. Second edition. Springer Verlag. 2005.

[4] Notas de Criptografía de Félix Delgado.

[5] M.Dietzfelbinger. Primality Testing in Polynomial Time. Lecture Notes in Computer Sciences. Springer Verlag 2005.

[6] A. Granville. It is easy to determine whether a given integer is prime. Bulletin of the American Mathematical Society (New series), Vol 42, n 1, 3–38. 2004.

[7] Notas de Criptografía de José Enrique Marcos.

[8] L. Rempe-Gillen, R. Waldecker. Primality Testing for Beginners. Student Mathematical Library, Volume 70. American Mathematical Society. 2014.

[9] Notas de Criptografía de Daniel Sadornil.

[10] J. von zur Gathen, J. Gerhard. Modern computer algebra 3 edition. Cambridge University Press. 2013.

[11] S.Y. Yan. Primality Testing and Integer Factorization in Public-Key Cryptography. Second Edition. Advances in Information Security. Springer. 2009.