

Primalidad en Tiempo Polinomial

Francisco Gallego Salido

Universidad de Granada

fgallego@correo.ugr.es

2 de diciembre de 2021

1 Tests de Primalidad

- Introducción
- Pequeño Teorema de Fermat

2 Algoritmo AKS

- Introducción
- El Algoritmo
- Complejidad

3 Comparaciones

- Test AKS
- Tests Probabilísticos
 - Test de Miller-Rabin
 - Test de Solovay-Strassen
- Comparaciones

4 Conclusiones

Tests de Primalidad

¿Qué es un test de primalidad?

Un test de primalidad es un algoritmo que nos permite determinar si un número es primo o no.

El test más básico es el que se deriva de la definición de primalidad, cuya complejidad es $O(\sqrt{n})$:

¿Qué es un test de primalidad?

Un test de primalidad es un algoritmo que nos permite determinar si un número es primo o no.

El test más básico es el que se deriva de la definición de primalidad, cuya complejidad es $O(\sqrt{n})$:

- Comprobar todos los números menores que $\lfloor \sqrt{n} \rfloor$ y ver si alguno divide a n .

¿Qué es un test de primalidad?

Un test de primalidad es un algoritmo que nos permite determinar si un número es primo o no.

El test más básico es el que se deriva de la definición de primalidad, cuya complejidad es $O(\sqrt{n})$:

- Comprobar todos los números menores que $\lfloor \sqrt{n} \rfloor$ y ver si alguno divide a n .
- Si ninguno lo divide, n es primo.

¿Qué es un test de primalidad?

Un test de primalidad es un algoritmo que nos permite determinar si un número es primo o no.

El test más básico es el que se deriva de la definición de primalidad, cuya complejidad es $O(\sqrt{n})$:

- Comprobar todos los números menores que $\lfloor \sqrt{n} \rfloor$ y ver si alguno divide a n .
- Si ninguno lo divide, n es primo.
- Si alguno lo divide, n es compuesto.

Pequeño Teorema de Fermat

Un test mucho más eficiente es el que se deriva del *Pequeño Teorema de Fermat*.

Pequeño Teorema de Fermat

Un test mucho más eficiente es el que se deriva del *Pequeño Teorema de Fermat*.

Theorem (Pequeño Teorema de Fermat)

Si n es primo, entonces $a^n \equiv a \pmod{n}$ para todo $a \in \mathbb{Z}$.

Pequeño Teorema de Fermat

Un test mucho más eficiente es el que se deriva del *Pequeño Teorema de Fermat*.

Theorem (Pequeño Teorema de Fermat)

Si n es primo, entonces $a^n \equiv a \pmod{n}$ para todo $a \in \mathbb{Z}$.

El test consiste en comprobar varios valores de a y vemos si se cumple la congruencia.

Pequeño Teorema de Fermat

Un test mucho más eficiente es el que se deriva del *Pequeño Teorema de Fermat*.

Theorem (Pequeño Teorema de Fermat)

Si n es primo, entonces $a^n \equiv a \pmod{n}$ para todo $a \in \mathbb{Z}$.

El test consiste en comprobar varios valores de a y vemos si se cumple la congruencia.

Si falla para algún a , entonces n es compuesto. En caso contrario, n probablemente sea primo.

Problema

No es cierto en general que si $a^n \equiv a \pmod{n}$ para todo $a \in \mathbb{Z}$, entonces n sea primo.

No es cierto en general que si $a^n \equiv a \pmod{n}$ para todo $a \in \mathbb{Z}$, entonces n sea primo.

Los *Números de Carmichael* son contraejemplos de ello, donde $561 = 3 \cdot 11 \cdot 17$ es el primer elemento de dicho conjunto.

Algoritmo AKS

Se trata del primer test de primalidad que cumple todas las propiedades deseadas:

Se trata del primer test de primalidad que cumple todas las propiedades deseadas:

- **General.** Válido para todos los enteros.

Se trata del primer test de primalidad que cumple todas las propiedades deseadas:

- **General.** Válido para todos los enteros.
- **Determinista.** Determina con probabilidad 100 % si un número es primo o compuesto.

Se trata del primer test de primalidad que cumple todas las propiedades deseadas:

- **General.** Válido para todos los enteros.
- **Determinista.** Determina con probabilidad 100 % si un número es primo o compuesto.
- **Polinómico.** La complejidad asintótica es polinómica en la cantidad de dígitos.

Se trata del primer test de primalidad que cumple todas las propiedades deseadas:

- **General.** Válido para todos los enteros.
- **Determinista.** Determina con probabilidad 100 % si un número es primo o compuesto.
- **Polinómico.** La complejidad asintótica es polinómica en la cantidad de dígitos.
- **Incondicional.** No depende de hipótesis no probadas aún.

Idea Principal

El *Pequeño Teorema de Fermat* no proporciona un test válido, pero una versión general suya sí.

El *Pequeño Teorema de Fermat* no proporciona un test válido, pero una versión general suya sí.

Theorem

Sea $n > 1$ y $a \in \mathbb{Z}$. Entonces n es primo si, y solo si, se cumple

$$(X + a)^n \equiv X^n + a \pmod{n}$$

El *Pequeño Teorema de Fermat* no proporciona un test válido, pero una versión general suya sí.

Theorem

Sea $n > 1$ y $a \in \mathbb{Z}$. Entonces n es primo si, y solo si, se cumple

$$(X + a)^n \equiv X^n + a \pmod{n}$$

Un test que se deriva de esta propiedad es simplemente comprobar si la congruencia se cumple para algún a .

¿Es este test polinómico?

Este test implica evaluar n coeficientes, luego la complejidad sería $\Omega(n)$.

¿Es este test polinómico?

Este test implica evaluar n coeficientes, luego la complejidad sería $\Omega(n)$.

¿Podemos reducir el número de coeficientes a evaluar?

Si la congruencia anterior la evaluamos módulo $(X^r - 1, n)$ para algún r escogido apropiadamente, podremos reducir el número de coeficientes. Sea pues la siguiente congruencia:

Si la congruencia anterior la evaluamos módulo $(X^r - 1, n)$ para algún r escogido apropiadamente, podremos reducir el número de coeficientes. Sea pues la siguiente congruencia:

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

Nuevo Problema

Esta nueva congruencia sigue siendo cierta cuando n es primo.

Nuevo Problema

Esta nueva congruencia sigue siendo cierta cuando n es primo.

No obstante, algunos números compuestos también la cumplen para algunos valores de r y a .

Esta propiedad se puede reestablecer casi por completo.

Esta propiedad se puede reestablecer casi por completo.

Eligiendo r adecuadamente, si la congruencia se cumple para varios valores de a , entonces podemos asegurar que n es una potencia de un primo.

El Algoritmo

Algorithm 1 AKS

procedure ISPRIME(n) ▷ Comprobar si $n > 1$ es un número primo
 if $n = a^b$ con $a, b > 1$ **return** COMPUESTO ▷ Paso 1
 Encontrar el menor r tal que $\text{ord}_r(n) > \log^2(n)$. ▷ Paso 2
 if $1 < (a, n) < n$ para algún $a \leq r$ **return** COMPUESTO ▷ Paso 3
 if $n \leq r$ **return** PRIMO ▷ Paso 4
 for $a = 1$ hasta $\lfloor \sqrt{\phi(r)} \log(n) \rfloor$ **do** ▷ Paso 5
 if $(X + a)^n \not\equiv X^n + a \pmod{(n, X^r - 1)}$ **return** COMPUESTO
 end for
 return PRIMO ▷ Paso 6
end procedure

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- 1 31 no es una potencia perfecta. Pasamos al siguiente paso.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- 1 31 no es una potencia perfecta. Pasamos al siguiente paso.
- 2 El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- 1 31 no es una potencia perfecta. Pasamos al siguiente paso.
- 2 El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- 3 $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- 1 31 no es una potencia perfecta. Pasamos al siguiente paso.
- 2 El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- 3 $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- 4 $31 \not\leq 29$. Pasamos al siguiente paso.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- 1 31 no es una potencia perfecta. Pasamos al siguiente paso.
- 2 El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- 3 $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- 4 $31 \not\leq 29$. Pasamos al siguiente paso.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- ① 31 no es una potencia perfecta. Pasamos al siguiente paso.
- ② El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- ③ $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- ④ $31 \not\leq 29$. Pasamos al siguiente paso.
- ⑤
 - $\lfloor \sqrt{\phi(r)} \log(n) \rfloor = 26$.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- ❶ 31 no es una potencia perfecta. Pasamos al siguiente paso.
- ❷ El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- ❸ $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- ❹ $31 \not\leq 29$. Pasamos al siguiente paso.
- ❺
 - $\lfloor \sqrt{\phi(r)} \log(n) \rfloor = 26$.
 - $(X + a)^{31} = x^2 + a^{31}$ módulo $(X^{29} - 1, 31)$.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- ① 31 no es una potencia perfecta. Pasamos al siguiente paso.
- ② El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- ③ $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- ④ $31 \not\leq 29$. Pasamos al siguiente paso.
- ⑤
 - $\lfloor \sqrt{\phi(r)} \log(n) \rfloor = 26$.
 - $(X + a)^{31} = x^2 + a^{31}$ módulo $(X^{29} - 1, 31)$.
 - $X^{31} + a = x^2 + a$ módulo $(X^{29} - 1, 31)$.

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- ① 31 no es una potencia perfecta. Pasamos al siguiente paso.
- ② El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- ③ $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- ④ $31 \not\leq 29$. Pasamos al siguiente paso.
- ⑤
 - $\lfloor \sqrt{\phi(r)} \log(n) \rfloor = 26$.
 - $(X + a)^{31} = x^2 + a^{31}$ módulo $(X^{29} - 1, 31)$.
 - $X^{31} + a = x^2 + a$ módulo $(X^{29} - 1, 31)$.
 - $a^{31} \equiv a \pmod{(X^{29} - 1, 31)}$ para todo $1 \leq a \leq 26$. Pasamos al siguiente paso

Ejemplo

Sea el número 31, el cual ya sabemos que es primo.

- ① 31 no es una potencia perfecta. Pasamos al siguiente paso.
- ② El menor r es 29, pues $\text{ord}_{29}(31) = 28 > \log^2(31) \simeq 24,54$.
- ③ $(a, 31) = 1$ para todo $a \leq 29$. Pasamos al siguiente paso.
- ④ $31 \not\leq 29$. Pasamos al siguiente paso.
- ⑤
 - $\lfloor \sqrt{\phi(r)} \log(n) \rfloor = 26$.
 - $(X + a)^{31} = x^{31} + a^{31}$ módulo $(X^{29} - 1, 31)$.
 - $X^{31} + a = x^{31} + a$ módulo $(X^{29} - 1, 31)$.
 - $a^{31} \equiv a \pmod{(X^{29} - 1, 31)}$ para todo $1 \leq a \leq 26$. Pasamos al siguiente paso
- ⑥ Hemos llegado al último paso, por lo que 31 es primo.

Complejidad

El valor encontrado en el segundo paso es crucial para que el test sea polinómico.

El valor encontrado en el segundo paso es crucial para que el test sea polinómico.

De dicho valor depende la complejidad del resto del algoritmo.

El valor encontrado en el segundo paso es crucial para que el test sea polinómico.

De dicho valor depende la complejidad del resto del algoritmo.

Dicho valor se puede probar que está acotado superiormente por $\max\{3, \lceil \log^5(n) \rceil\}$.

Complejidad Paso a Paso

Cada paso tiene un complejidad determinada:

Cada paso tiene un complejidad determinada:

- $O^{\sim}(\log^3(n))$.
- $O^{\sim}(\log^7(n))$.
- $O^{\sim}(\log^6(n))$.
- $O(\log(n))$.
- $O^{\sim}(\log^{21/2}(n))$.
- $O(1)$.

Hay que comprobar $O(r^{1/2} \log(n))$ congruencias polinómicas.

Hay que comprobar $O(r^{1/2} \log(n))$ congruencias polinómicas.

Cada congruencia consiste de $O(\log(n))$ multiplicaciones de polinomios de grado $O(r)$ con coeficientes de tamaño $O(\log(n))$.

Hay que comprobar $O(r^{1/2} \log(n))$ congruencias polinómicas.

Cada congruencia consiste de $O(\log(n))$ multiplicaciones de polinomios de grado $O(r)$ con coeficientes de tamaño $O(\log(n))$.

Esto produce una complejidad asintótica de $O^\sim(r^{3/2} \log^3(n))$, que teniendo que $r = O(\log^5(n))$, nos queda finalmente $O^\sim(\log^{21/2}(n))$.

La complejidad del quinto paso es la más alta, luego el algoritmo **AKS** tiene una complejidad algorítmica de $O^{\sim}(\log^{21/2}(n))$.

La complejidad del quinto paso es la más alta, luego el algoritmo **AKS** tiene una complejidad algorítmica de $O^{\sim}(\log^{21/2}(n))$.

Usando *Teoría de Cribas* se puede probar que $r = O(\log^3(n))$, reduciendo la complejidad hasta $O^{\sim}(\log^{15/2}(n))$.

La complejidad del quinto paso es la más alta, luego el algoritmo **AKS** tiene una complejidad algorítmica de $O^{\sim}(\log^{21/2}(n))$.

Usando *Teoría de Cribas* se puede probar que $r = O(\log^3(n))$, reduciendo la complejidad hasta $O^{\sim}(\log^{15/2}(n))$.

Bajo ciertas hipótesis no probadas (como puede ser la *Hipótesis Generalizada de Riemann*), se puede probar que $r = O(\log^2(n))$, reduciendo una vez más la complejidad hasta $O^{\sim}(\log^6(n))$.

Implementación y Análisis

Detalles de la Implementación del Test AKS

Hay varias consideraciones para el test AKS:

Hay varias consideraciones para el test AKS:

- Es importante usar buena multiplicación de polinomios. De lo contrario el test es inservible en la práctica.
- Calcular cotas lo más fieles posibles.
- Utilizar enteros de precisión arbitraria.

Tests Probabilísticos

Test de Miller-Rabin

Test probabilístico basado en las siguientes congruencias aplicables a enteros mayores que 2 impares y para varias bases a elegidas aleatoriamente:

Test de Miller-Rabin

Test probabilístico basado en las siguientes congruencias aplicables a enteros mayores que 2 impares y para varias bases a elegidas aleatoriamente:

$$a^d \equiv 1 \pmod{n} \tag{1}$$

$$a^{2^r d} \equiv -1 \pmod{n} \text{ con } 0 \leq r < s, \tag{2}$$

donde $n = 2^r d + 1$ con $d \geq 1$ impar y $r \geq 1$.

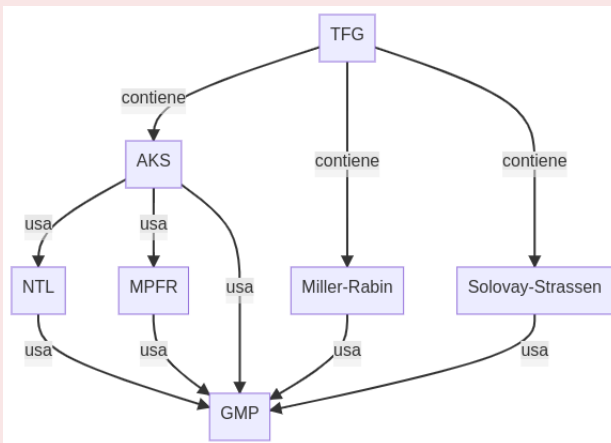
Test de Solovay-Strassen

Test probabilístico basado en la siguiente congruencia relacionada con el *Símbolo de Jacobi*, la cual se comprueba para varias bases a elegidas aleatoriamente:

Test de Solovay-Strassen

Test probabilístico basado en la siguiente congruencia relacionada con el *Símbolo de Jacobi*, la cual se comprueba para varias bases a elegidas aleatoriamente:

$$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}, \quad (3)$$



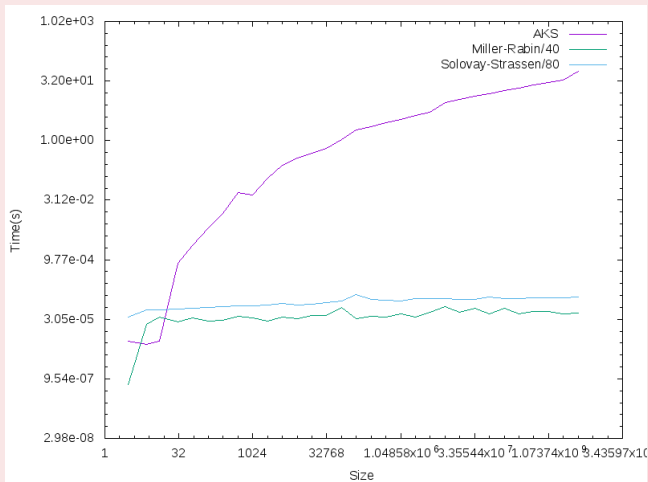
Comparaciones

Comparación Números Primos

Mayores primos que ocupan n bits. Por ejemplo, el 7 es el mayor primo con 3 bits, 31 el mayor primo con 5 bits, etc.

Comparación Números Primos

Mayores primos que ocupan n bits. Por ejemplo, el 7 es el mayor primo con 3 bits, 31 el mayor primo con 5 bits, etc.

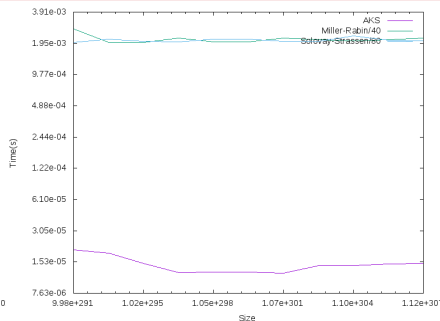
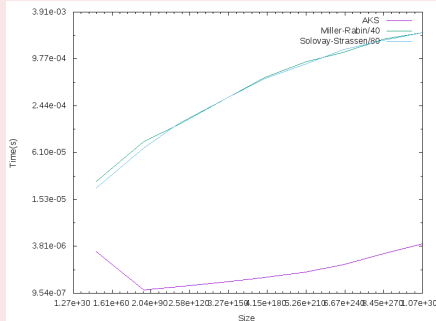


Comparación Potencias Perfectas

- **Izquierda.** Primos de hasta 16 bits elevados a 100.
- **Derecha.** Primos de entre 192 y 256 bits elevados a 5.

Comparación Potencias Perfectas

- **Izquierda.** Primos de hasta 16 bits elevados a 100.
- **Derecha.** Primos de entre 192 y 256 bits elevados a 5.

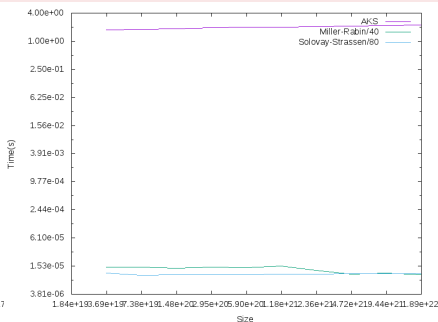
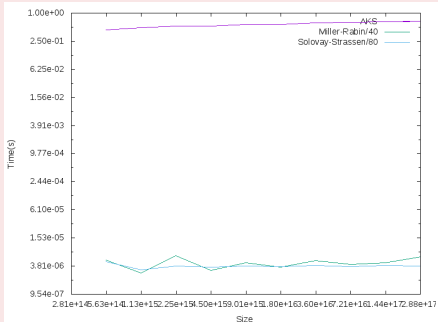


Comparación Compuestos No Potencias Perfectas

- **Izquierda.** Primos de entre 32 y 42 bits multiplicados por un primo de 16 bits.
- **Derecha.** Primos de entre 32 y 42 bits multiplicados por un primo de 32 bits.

Comparación Compuestos No Potencias Perfectas

- **Izquierda.** Primos de entre 32 y 42 bits multiplicados por un primo de 16 bits.
- **Derecha.** Primos de entre 32 y 42 bits multiplicados por un primo de 32 bits.



Conclusiones


El test es brillante desde el punto de vista matemático, ya que la prueba de la validez del test usa herramientas elementales.

El test es brillante desde el punto de vista matemático, ya que la prueba de la validez del test usa herramientas elementales.

Sin embargo, a pesar de ser un test polinómico, en la práctica queda muy por detrás de otros test usados en la actualidad.

 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena.
PRIMES is in P.

Ann. of Math. (2), 160(2):781–793, 2004.

 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena.
Errata: PRIMES is in P.

Ann. of Math. (2), 189(1):317–318, 2019.

 Francisco Gallego Salido.
Tfg.

<https://github.com/fgallegosalido/TFG>.

Fin

Muchas gracias