Algoritmos en teoría de números

Parte VI

Segundo semestre 2022

IIC2283

Prof. Nicolás Van Sint Jan

Recordatorio: Tercera versión del test de primalidad

Vamos a diseñar un test de primalidad considerando los conjuntos:

$$\begin{aligned} S_n^+ &= \{a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv 1 \pmod{n}\} \\ S_n^- &= \{a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv -1 \pmod{n}\} \end{aligned}$$

Así, podemos definir S_n a partir de estos conjuntos:

$$S_n = S_n^+ \cup S_n^-$$

Para hacer esto necesitamos estudiar algunas propiedades de los conjuntos S_n^+, S_n^- y S_n.

- Consideramos primero el caso en que n es primo, y luego el caso en que n es compuesto

Recordatorio: caracterizando S_n, S_n^+ y S_n^-

Proposición 1

Si $n \geq 3$ es primo, entonces $S_n = \mathbb{Z}_n^*$.

Proposición 2

Si $n \geq 3$ es primo: $|S_n^+| \; = \; |S_n^-| \; = \; \dfrac{n-1}{2}$

Outline

Una propiedad fundamental de S_n para n compuesto

Teorema

Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$. Si existe $a \in \mathbb{Z}_n^*$ tal que $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, entonces:

$$|S_n| \quad \leq \quad \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Para demostrar el teorema necesitamos el **Teorema Chino del resto**.

Para recordar: un teorema muy útil

Teorema Chino del Resto

Suponga que $\gcd(m, n) = 1$. Para todo a y b, existe c tal que:

$$\begin{aligned} c &\equiv a \pmod{m} \\ c &\equiv b \pmod{n} \end{aligned}$$

Demostración

Dado que $\gcd(m, n) = 1$, existen d y e tales que:

$$n \cdot d \equiv 1 \pmod{m}$$

$$m \cdot e \equiv 1 \pmod{n}$$

Sea $c = a \cdot n \cdot d + b \cdot m \cdot e$. Se tiene que:

$$c \equiv a \pmod{m}$$

$$c \equiv b \pmod{n}$$

$\square$

La demostración del teorema inicial

Teorema

Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$. Si existe $a \in \mathbb{Z}_n^*$ tal que $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, entonces:

$$|S_n| \quad \leq \quad \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Demostración

Suponga que $a \in \mathbb{Z}_n^*$ y $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$

Por Teorema Chino del Resto, existe b tal que:

$$\begin{aligned} b &\equiv a \pmod{n_1} \\ b &\equiv 1 \pmod{n_2} \end{aligned}$$

Entonces: $a = \alpha \cdot n_1 + b$ y $1 = \beta \cdot n_2 + b$

- Por lo tanto $\gcd(b, n) = 1$, ya que $n = n_1 \cdot n_2$ y $a \in \mathbb{Z}_n^*$

La demostración del teorema inicial

Teorema

Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$. Si existe $a \in \mathbb{Z}_n^*$ tal que $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, entonces:

$$|S_n| \quad \leq \quad \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Demostración

Además, tenemos que:

$$\begin{aligned} b^{\frac{n-1}{2}} \quad \equiv \quad a^{\frac{n-1}{2}} \quad &\equiv \quad -1 \pmod{n_1} \\ b^{\frac{n-1}{2}} \quad &\equiv \quad 1 \pmod{n_2} \end{aligned}$$

Dado que $n = n_1 \cdot n_2$ concluimos que:

$$\begin{aligned} b^{\frac{n-1}{2}} \quad &\not\equiv \quad 1 \pmod{n} \\ b^{\frac{n-1}{2}} \quad &\not\equiv \quad -1 \pmod{n} \end{aligned}$$

La demostración del teorema inicial

Teorema

Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$. Si existe $a \in \mathbb{Z}_n^*$ tal que $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, entonces:

$$|S_n| \quad \leq \quad \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Demostración

Sea $c = (b \mod n)$. Concluimos que $c \notin S_n$ y $c \in \mathbb{Z}_n^*$. Es decir:

$$S_n \subsetneq \mathbb{Z}_n^*$$

Pero se tiene que $(S_n, \cdot)$ es un subgrupo de $(\mathbb{Z}_n^*, \cdot)$ **(¿Por qué?)**

Entonces por Teorema de Lagrange:

$$|S_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

□

Un test de primalidad aleatorizado

Ya tenemos los ingredientes esenciales para el test de primalidad

- Sólo nos falta implementar algunas funciones auxiliares

Necesitamos desarrollar un algoritmo eficiente para determinar si un número n es la potencia (no trivial) de otro número.

Verificando si un número es la potencia de otro

Primero necesitamos una función para calcular n^k

- Usamos el algoritmo de exponenciación rápida pero sin considerar el módulo

EXP(n, k)
    if $k = 1$ **then return** n
    else if k es par **then**
        $val :=$ **EXP**$(n, \frac{k}{2})$
        return $val \cdot val$
    else
        $val :=$ **EXP**$(n, \frac{k-1}{2})$
        return $val \cdot val \cdot n$

Verificando si un número es la potencia de otro

Dado un número natural $n \geq 2$, la siguiente función verifica si existen $m, k \in \mathbb{N}$ tales que $k \geq 2$ y $n = m^k$

EsPotencia(n)
    if $n \leq 3$ **then return** no
    else
        for $k := 2$ **to** $\lfloor \log_2(n) \rfloor$ **do**
            if TieneRaízEntera(n, k, 1, n) **then return** sí
        return no

Verificando si un número es la potencia de otro

La siguiente función verifica si existe $m \in \{i, \ldots, j\}$ tal que $n = m^k$

- Vale decir, la llamada **TieneRaízEntera**(n, k, 1, n) verifica si n tiene raíz k-ésima entera

TieneRaízEntera(n, k, i, j)
    if $i = j$ **then**
        if EXP$(i, k) = n$ **then return** sí
        else return no
    else if $i < j$ **then**
        $p := \left\lfloor \frac{i+j}{2} \right\rfloor$
        $val :=$ **EXP**(p, k)
        if $val = n$ **then return** sí
        else if $val < n$ **then return TieneRaízEntera**(n, k, $p + 1$, j)
        else return TieneRaízEntera(n, k, i, $p - 1$)
    else return no

La complejidad de **EsPotencia**

Consideramos la multiplicación de números enteros como la operación básica a contar

Tenemos que:

- En el peor caso **EsPotencia**(n) realiza ($\lfloor \log_2(n) \rfloor$ - 1) llamadas a la función **TieneRaízEntera**
- Existe $c \in \mathbb{N}$ tal que la llamada **TieneRaízEntera**(n, k, 1, n) realiza en el peor caso a lo más $c \cdot \log_2(n)$ llamadas a la función **EXP**
- **EXP**(n, k) en el peor caso es $O(\log_2(k))$

La complejidad de **EsPotencia**

Concluimos que **EsPotencia**(n) en el peor caso es $O([\log_2(n)]^3)$

Vale decir, **EsPotencia** en el peor caso es de orden polinomial en el tamaño de la entrada

- Se puede llegar a la misma conclusión si consideramos todas las operaciones realizadas por **EsPotencia**

Un test de primalidad aleatorizado

El siguiente algoritmo aleatorizado determina si un número entero $n \geq 2$ es primo.

El algoritmo recibe como entrada un valor entero $k \geq 1$ que es usado para controlar la probabilidad de error.

Un test de primalidad aleatorizado

TestPrimalidad(n, k)
    if $n = 2$ **then return** PRIMO
    else if n es par **then return** COMPUESTO
    else if EsPotencia(n) **then return** COMPUESTO
    else
        sea $a_1, \ldots, a_k$ una secuencia de números elegidos de
                manera uniforme e independiente desde $\{1, \ldots, n-1\}$
        for $i := 1$ **to** k **do**
            if MCD(a_i, n) > 1 **then return** COMPUESTO
            else $b_i :=$ **EXP**($a_i, \frac{n-1}{2}, n$)
        $neg := 0$
        for $i := 1$ **to** k **do**
            if $b_i \equiv -1 \bmod n$ **then** $neg := neg + 1$
            else if $b_i \not\equiv 1 \bmod n$ **then return** COMPUESTO
        if $neg = 0$ **then return** COMPUESTO
        else return PRIMO

Test de primalidad: probabilidad de error

TestPrimalidad se puede equivocar de dos formas:

- Suponga que $n \geq 3$ es primo. En este caso **TestPrimalidad** da una respuesta incorrecta si $b_i \equiv 1 \bmod n$ para todo $i \in \{1, \ldots, k\}$

  Dado que $|S_n^+| = |S_n^-| = \frac{n-1}{2}$:
  - La probabilidad de que para un número a elegido con distribución uniforme desde $\{1, \ldots, n-1\}$ se tenga que $a^{\frac{n-1}{2}} \equiv 1 \bmod n$ es $\frac{1}{2}$

  Por lo tanto, la probabilidad de que **TestPrimalidad** diga COMPUESTO para $n \geq 3$ primo es $\left(\frac{1}{2}\right)^k$

Test de primalidad: probabilidad de error

- Suponga que n es compuesto, n es impar y n no es de la forma m^ℓ con $\ell \geq 2$
  - Si n es par o n es de la forma m^ℓ con $\ell \geq 2$, entonces **TestPrimalidad** da la respuesta correcta COMPUESTO

Tenemos entonces que $n = n_1 \cdot n_2$ con $n_1 \geq 3$, $n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$

Además debe existir $a \in \{1, \ldots, n-1\}$ tal que $\gcd(a, n) = 1$ y $a^{\frac{n-1}{2}} \equiv -1 \bmod n$

- Si esto no es cierto **TestPrimalidad** retorna COMPUESTO, dado que si **TestPrimalidad** logra llegar a la última instrucción **if** entonces *neg* necesariamente es igual a 0

Test de primalidad: probabilidad de error

Concluimos que $|S_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$

- Por la caracterización que dimos de S_n para n compuesto

Vamos a utilizar este resultado para acotar la probabilidad de error:

$$\Pr\Bigg(\Bigg(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\Bigg) \wedge \Bigg(\bigvee_{j=1}^{k} b_j \equiv -1 \bmod n\Bigg)\Bigg)$$

Test de primalidad: probabilidad de error

Tenemos que:

$$\Pr\Bigg(\bigg(\bigwedge_{i=1}^{k} \mathrm{gcd}(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\bigg) \wedge \bigg(\bigvee_{j=1}^{k} b_j \equiv -1 \bmod n\bigg)\Bigg) \;\leq$$

$$\Pr\bigg(\bigwedge_{i=1}^{k} \mathrm{gcd}(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\bigg)$$

Por lo tanto sólo necesitamos una cota superior para la última expresión.

Test de primalidad: probabilidad de error

Tenemos que:

$$\Pr\Big(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \land (b_i \equiv 1 \bmod n \lor b_i \equiv -1 \bmod n)\Big)$$

$$= \prod_{i=1}^{k} \Pr(\gcd(a_i, n) = 1 \land (b_i \equiv 1 \bmod n \lor b_i \equiv -1 \bmod n))$$

$$= \prod_{i=1}^{k} \Pr((b_i \equiv 1 \bmod n \lor b_i \equiv -1 \bmod n) \mid \gcd(a_i, n) = 1) \cdot$$

$$\Pr(\gcd(a_i, n) = 1)$$

$$\leq \prod_{i=1}^{k} \Pr((b_i \equiv 1 \bmod n \lor b_i \equiv -1 \bmod n) \mid \gcd(a_i, n) = 1)$$

$$= \prod_{i=1}^{k} \Pr(a_i \in S_n \mid a_i \in \mathbb{Z}_n^*) \leq \prod_{i=1}^{k} \frac{1}{2} = \frac{1}{2^k}$$

Test de primalidad: probabilidad de error

Concluimos que la probabilidad de que el test diga PRIMO para el valor compuesto n está acotada por $\left(\frac{1}{2}\right)^k$

En ambos casos (si n es primo o compuesto) la probabilidad de error del algoritmo está acotada por $\left(\frac{1}{2}\right)^k$

- ¡Si $k = 100$, está probabilidad está acotada por $\left(\frac{1}{2}\right)^{100} \approx 7.9 \times 10^{-31}$!