Complejidad Computacional CC71D

1 PRIMES is in P

2 La Idea Básica

El test esta basado en la siguiente identidad para numeros primos

Identidad 1: Supongamos que a es coprimo con p. Luego, p es primo si y solo si

$$(x-a)^p \equiv (x^p - a) (\mathrm{mod}\ p) \quad (1)$$

Luego, dado p como entrada, podríamos tomar un polinomio $P(x) = x - a$ y calcular cuando la identidad (1) se satisface. No obstante esto tomaria tiempo $\Omega(p)$ porque tendriamos que calcular p coeficientes en el lado izquierdo de la identidad en el peor caso. Por lo tanto, para hacer posible el test anterior, evaluaremos ambos lados de (1) pero tomando además el módulo de un polinomio de la forma $x^r - 1$. Luego, Una Iteración de nuestro algoritmo consistirá en evaluar cuando se verifica la siguiente identidad.

$$(x-a)^p \equiv (x^p - a) (\mathrm{mod}\ x^r - 1, p) \quad (2)$$

Inmediatamente notamos que todos los primos satisfacen esta nueva identidad para todos los valores de a,r; No obsante, algunos números compuestos también podrían satisfacer (2) para algunos valores de (a,r). La identidad anterior toma tiempo $O(r^2 \log^3 p)$ de verificación (el lado izquierdo es evaluado usando la técnica de "repeated squaring"), o aun mejor $O(r \log^2 p)$ usando "Fast Fourier Multiplication [KNU98]".

Nuestro Algoritmo primero escoge un r "conveniente".

Diremos que un r es "conveniente" si:

- r es un primo $O(\log^6 p)$
- r-1 tiene un factor primo "q" tal que $q \geq r^{\left(\frac{1}{2}+\delta\right)}$ para alguna constante $\delta > 0$.

[FOU85, BH96] Aseguran la exitencia de tal r "conveniente"

Una vez hallado r, el algoritmo verifica la identidad (2) para una cantidad "pequeña" ($O(\sqrt{r}\log p)$) de a's. Probaremos que esta idea funciona y que por lo tanto el algoritmo determina correctamente cuando p es primo o no.

3 Preliminares

Lema 3.1

ii) Sea $f(x)$ un polinomio de coeficientes enteros. Luego $f(x)^p \equiv f(x^p) \ (mod\ p)$.

Lema 3.2

Sea $P(n)$ una función que retorna el máximo factor primo de n. Existen las constantes $c > 0$ y n_0 tales que, para todo $x \geq n_0$ se tiene que:

$$|\{p|p \text{ is prime, } p \leq x \text{ and } P(p-1) > x^{\frac{2}{3}}\}| \geq c\frac{x}{\log x}.$$

Ejercicio: Reemplazar x por $c_2(\log n)^6$

Lema 3.3

Sea $\pi(n)$ la funcion que retorna la cantidad de primos $\leq n$. Luego, para $n \geq 1$:

$$\frac{n}{6\log n} \leq \pi(n) \leq \frac{8n}{\log n}.$$

Ejercicio: Reemplazar n por $c_1(\log n)^6$

4 El Algoritmo

Input: Integer n > 1

1	`if (n has the form a^b with b > 1) then output COMPOSITE`
2	r := 2

primos esta en p

3	while (r < n) {
4	if (gcd(n,r) is not 1) then output COMPOSITE
5	if (r is prime greater than 2) then
6	let q be the largest factor of r-1
7	if (q > 4sqrt(r)log n) and ($n^{(r-1)/q}$ is not 1 (mod r)) then
8	break
9	r := r+1
10	}
11	for a = 1 to 2sqrt(r)log n
12	if ($(x-a)^n$ is not (x^n-a) (mod x^r-1, n)) then output COMPOSITE
13	output PRIME;

Teorema 4.1 El algoritmo retorna PRIME si y solo si n es primo.
Esquema de la demostración:
1 - Demostrar que "si n es primo, el algoritmo retorna PRIME".
Hasta ahora, podría ocurrir que el algoritmo retorne PRIME si n es compuesto. Luego, la demostración se completa al
2- Demostrar que "si n es compuesto, el algoritmo retorna COMPOSITE".
Esta demostración se logra por contradicción. Suponemos que la entrada n es un número compuesto y que el algoritmo retorno PRIME; Si el supuesto anterior ocurre, implica que necesariamente n es de la forma a^b (produciendoce una contradicción al considerar la primera línea del algoritmo).

LOOP 1: el while
Notar que el algoritmo tiene dos loops. El primer loop intenta encontrar un r tal que r-1 tenga un factor primo que cumpla con

1. $q \geq 4\sqrt{r}\log n$,
2. $q|o_r(n)$, donde $o_r(n)$ es el orden de n módulo r.

A continuación examinaremos la cantidad de iteraciones del loop while requeridas para encontrar r.

Lema 4.2 Para un n suficientemente grande, existen constantes positivas c1 y c2 para las cuales existe un primo aceptable r en el intervalo $[c_1(\log n)^6, c_2(\log n)^6]$ tal que r-1 tiene un factor primo $q \geq 4\sqrt{r}\log n$, y $q|o_r(n)$. O mas simple, un r de acuerdo a las condiciones de break del loop while.

primos esta en p

La propiedad 1

La siguiente desigualdad, a partir de las densidades de los primos admisibles r (Lema 3.2) y de los números primos en general (Lema 3.3), nos da una cota inferior para la cantidad de r's que podemos encontrar en el intervalo propuesto por este Lema.

La cantidad de primos r t.q.....

$$\begin{aligned} &\geq \text{No of special primes in } [1 \cdots c_2(\log n)^6] - \text{No of primes in } [1 \cdots c_1(\log n)^6] \\ &\geq \frac{cc_2(\log n)^6}{7 \log\log n} - \frac{8c_1(\log n)^6}{6 \log\log n} \text{ (using Lemma 3.3)} \\ &= \frac{(\log n)^6}{\log\log n}\left(\frac{cc_2}{7} - \frac{8c_1}{6}\right). \end{aligned}$$

Escogiendo $c_1 \geq 4^6$ y c_2 de manera tal que la cantidad entre parentesis sea una constante positiva, en adelante c_3 ., tenemos que siempre podemos hallar un r cuyo q cumpla la propiedad 1.

La propiedad 2

Sea $x = c_2(\log n)^6$ y consideremos el producto

$$\Pi = (n-1)(n^2-1)\cdots(n^{x^{\frac{1}{2}}}-1).$$

Este producto tiene al menos $x^{\frac{2}{3}} \log n$ factores primos. Notemos que

$$x^{\frac{2}{3}} \log n < \frac{c_3(\log n)^6}{\log\log n}.$$

Por lo tanto existe almenos un primo conveniente r, tal que no divide al producto Π.

Lema 4.3 Si n es primo, el algoritmo retorna PRIME.
Demostracion: El loop while no puede retornar COMPOSITE pues gcd(n,r)=1 para todo $r \leq c_2(\log n)^6$, donde c_2 es el del lema anterior. Por el lema 3.1ii), el loop for tampoco puede retornar COMPOSITE. Luego , el algoritmo identificara n como PRIME.

---

5 Análisis de complejidad temporal

Nota: Se esta usando la notacion $\tilde{O}(t(n))$ para $O(t(n)poly(\log t(n)))$.

Teorema 5.1 La complejidad asintótica en tiempo del algoritmo es $\tilde{O}(\log^{12} n)$.

primos esta en p

Linea(s)	Invocacion	Complejidad asintotica
1	form a^b	$O(\log^3 n)$
3,4,5,6,7,8,9,10	while	$O(\log^6 n)$ iteraciones; $\tilde{O}(r^{\frac{3}{2}}\log^3 n) = \tilde{O}(\log^{12} n)$ tiempo
4	gcd(n,r)	$\text{poly}(\log\log r)$
5	r is prime?	$r^{\frac{1}{2}}\text{poly}(\log\log n)$ (Brute Force)
6	largest factor of r-1	$r^{\frac{1}{2}}\text{poly}(\log\log n)$ (Brute Force)
7	4sqrt(r)log n); $n^{(r-1)/q}$;$n^{(r-1)/q} \pmod r$	$\text{poly}(\log\log n)$
11,12,13	for	O($\sqrt{r}\log n$) iteraciones; $\tilde{O}(r^{\frac{3}{2}}\log^3 n) = \tilde{O}(\log^{12} n)$ tiempo
12		$\tilde{O}(\log n \cdot r \log n)$ "Fast Fourier Multiplication"

Comentarios

- r = $O(\log^6 n)$
- form a^b. Biseccion $f(n)=a^b$-n

Atte
Eduardo Maureira
emaureir@dcc.uchile.cl